



**ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ**
University of the Peloponnese

Π.Μ.Σ. στην Επιστήμη Υπολογιστών

Μεταπτυχιακή Διπλωματική Εργασία

**Behavior Trees for video games. Case study: Development of
a 3D Action-Adventure game**

Άγγελος Κυριαζής

ΑΜ: 2022202302011

Φιλίππα-Ελένη Βέργου

ΑΜ: 2022202302004

Επιβλέπων:

Γεώργιος Λέπουρας

Καθηγητής

Τρίπολη, Μάιος 2025

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική εργασία διερευνά τη χρήση των δέντρων συμπεριφοράς (Behavior Trees) στην ανάπτυξη τεχνητής νοημοσύνης για παιχνίδια, μέσω της μελέτης περίπτωσης της ανάπτυξης ενός 3D Παιχνιδιού Δράσης – Περιπέτειας (3D Action - Adventure) στην Unreal Engine 5.4. Σκοπός της μελέτης είναι η ανάλυση των βασικών αρχών και τεχνικών που διέπουν την ενσωμάτωσή τους, καθώς και η αποτύπωση των προκλήσεων και των λύσεων που προκύπτουν κατά τη διάρκεια της ανάπτυξης.

Λέξεις Κλειδιά: Δέντρα συμπεριφοράς, Non-Player Characters, τεχνητή νοημοσύνη, μηχανές παιχνιδιών, Unreal Engine

ABSTRACT

This thesis explores the use of Behavior Trees in artificial intelligence development for games, through the case study of developing a 3D Action-Adventure game in Unreal Engine 5.4. The aim of the study is to analyze the fundamental principles and techniques involved in their integration, as well as to document the challenges and solutions encountered during development.

Keywords: Behavior Trees, Non-Player Characters, artificial intelligence, game engines, Unreal Engine

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ	2
ABSTRACT	3
ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ	4
ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ	9
Εισαγωγή	18
1. Θεωρητικό Υπόβαθρο	19
1.1. Unreal Engine	19
1.2. Δέντρα Συμπεριφοράς (Behavior Trees)	20
1.2.1. Δομή και Βασικές Αρχές	20
1.2.2. Παραδείγματα χρήσης	21
1.2.3 Πλεονεκτήματα και μειονεκτήματα	21
1.3. Εφαρμογές Δέντρων Συμπεριφοράς σε Παιχνίδια	23
1.4. Συνδυασμός Behavior Trees με Άλλες Τεχνικές AI	24
1.5. Προσαρμοστικά Δέντρα Συμπεριφοράς και Συναισθηματική Αντίδραση	24
1.6. Εναλλακτικές μέθοδοι ανάπτυξης τεχνητής νοημοσύνης σε παιχνίδια	24
1.6.1. Unstructured AI (Μη Δομημένος Προγραμματισμός)	24
1.6.2. Finite State Machines (FSM)	25
1.6.3. Goal-Oriented Action Planning (GOAP)	25
2. Εργαλεία και Πλατφόρμες Ανάπτυξης	26
2.1. Unreal Engine 5 και Τεχνολογίες Συστήματος	27
2.1.1. Behavior Trees	27
2.1.2. Motion Matching και State Machines	27
2.1.3. Niagara Particle Systems και UI Widgets	27
2.1.4. Metahuman Creator και Metatailor App	28
2.1.5. Lumen	28
2.1.6. Landscape Editor, Landscape Auto Material και Foliage Instancing	28
2.2. Leonardo AI	28
2.3. Eleven Labs & PlayHT	28

2.4.	Blender	28
2.5.	Ableton Live	28
2.6.	Davinci Resolve	28
2.7.	3D Asset Marketplaces και Free Projects	29
3.	Στάδια Υλοποίησης και Μεθοδολογία	29
3.1.	Δημιουργία Χαρακτήρων	30
3.2.	Ανάπτυξη συστήματος διαχείρισης πόντων ζωής	30
3.3.	Συστήματα κινησιολογίας	30
3.4.	Δημιουργία οπλισμού και αντικειμένων	30
3.5.	Behavior Trees	31
3.5.1.	Στρατηγικές Χρήσης Δέντρων Συμπεριφοράς	31
3.5.2.	Διαδικασία Υλοποίησης	32
3.5.3.	Αξιολόγηση απόδοσης	33
3.6.	Επέκταση Μηχανισμών Gameplay	34
3.7.	Δημιουργία περιβάλλοντος	34
3.8.	Κινηματογραφικές σκηνές και Ειδικά εφέ	35
3.9.	Δοκιμή και Βελτιστοποίηση	36
4.	Game Design Document (G.D.D.)	37
4.1.	Τίτλος	37
4.2.	Εισαγωγή	38
4.3.	Ανάλυση Παιχνιδιού	38
4.4.	Ιστορία Παιχνιδιού	40
4.5.	Χαρακτήρες παιχνιδιού και σημαντικά αντικείμενα	41
4.5.1.	Χαρακτήρες παιχνιδιού	41
4.5.2.	Σημαντικά αντικείμενα	45
4.6.	Χάρτης	46
4.7.	Μηχανικές-Κανόνες Μάχης	51
4.7.1.	Παίκτης	51
4.7.2.	Εχθροί	51

4.8.	Ήττα	52
4.9.	Εξοπλισμός/Αντικείμενα	52
4.9.1.	Weapons	52
4.9.2.	Healing	55
4.9.3.	Attachments	56
4.9.4.	Equipment	57
4.9.5.	Ammo	58
4.10.	Ακολουθία Παιχνιδιού	59
4.11.	Δομή ακολουθιών	60
4.12.	Τέλος παιχνιδιού	61
4.13.	Οικονομία Παιχνιδιού	61
4.14.	Χειρισμός	62
5.	<i>Damage System</i>	64
5.1.	BPC_DamageSystem (Actor Component)	65
5.1.1.	Heal (Function)	66
5.1.2.	TakeDamage (Function)	66
5.1.3.	DamageInfo (Structure)	67
5.1.4.	Event Dispatchers	68
5.1.5.	Damage Indicators (UI)	73
5.2.	BPI_Damageable	76
6.	<i>NPC Blueprints</i>	82
6.1.	Parent Blueprint	85
6.2.	IK Rig & IK Retargeting	86
6.3.	Animations και Blendspaces	88
6.4.	Animation blueprint	91
6.4.1.	Event Graph	91
6.4.2.	Anim Graph	94
6.5.	Locomotion	95
6.6.	Skeletal Mesh Setup	100
6.7.	Weapon Blueprint	102

6.7.1.	Components και Variables	103
6.7.2.	Fire Functions	105
6.7.3.	Weapon Setup	110
7.	<i>Artificial Intelligence</i>	112
7.1.	Στόχοι	113
7.2.	Προαπαιτούμενα	114
7.3.	Υλοποίηση	122
7.3.1.	Blackboard Keys	123
7.3.2.	AI Controller	124
7.3.3.	Perception System	125
7.3.4.	BPI_Enemy_AI	128
7.3.5.	EQS TestingPawn	129
8.	<i>Behavior States</i>	131
8.1.	Behavior Trees Overview	132
8.2.	Passive State	136
8.2.1.	Passive Behavior Tree Overview	136
8.2.2.	Patrol Route	139
8.2.3.	Passive State Assets	141
8.3.	Investigate State	151
8.3.1.	Investigate Behavior Tree Overview	152
8.3.2.	Investigate State Assets	153
8.3.3.	Throwable Actor	161
8.4.	Attacking State	166
8.4.1.	TOKEN SYSTEM	167
8.4.2.	Ranged Attacking State	172
8.4.3.	Android Attacking State	180
8.5.	Seeking State	197
8.5.1.	Seeking Behavior Tree Overview	198
8.5.2.	Seeking State Assets	200
8.6.	Detecting State	208
8.6.1.	Stealth System	212
8.6.2.	Detection	215

8.6.3.	Handle Sensed Sight	224
8.7.	Dead State	229
8.8.	Frozen State	233
8.8.1.	Behavior Tree Overview	235
8.8.2.	Frozen State Assets	237
8.8.3.	DataWorm	239
9.	Takedowns	245
10.	Συμπληρωματικές Υλοποιήσεις	256
11.	Προκλήσεις και Μελλοντικές Βελτιώσεις	268
12.	Συμπεράσματα	272
	Βιβλιογραφικές αναφορές	273

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

Εικόνα 1. Εξώφυλλο παιχνιδιού	37
Εικόνα 2. Χάρτης παιχνιδιού	46
Εικόνα 3. North Lake	47
Εικόνα 4. Eliezer	48
Εικόνα 5. Urban Slams	49
Εικόνα 6. Ερευνητικό εργαστήριο	50
Εικόνα 7. Enforcers Camp	50
Εικόνα 8. Ενδεικτική απεικόνιση ενός camp	60
Εικόνα 9. Shards	61
Εικόνα 10. Πλήκτρα χειρισμού	62
Εικόνα 11. Heal (Function)	66
Εικόνα 12. TakeDamage (Function)	67
Εικόνα 13. Can Be Damaged (Macro)	68
Εικόνα 14. Event Dispatchers Binding	69
Εικόνα 15. DamageResponse (Agents)	70
Εικόνα 16. DamageResponse (Player)	71
Εικόνα 17. Death (CBP_SandboxCharacter)	72
Εικόνα 18. Blocked	73
Εικόνα 19. DebugHealth Widget Blueprints	74
Εικόνα 20. DebugHealth Widget	75

Εικόνα 21. RadialProgressBar Material Blueprint	75
Εικόνα 22. Damage Indicators.....	76
Εικόνα 23. GetCurrentHealth και GetMaxHealth	78
Εικόνα 24. Heal.....	79
Εικόνα 25. TakeDamage (input parameters)	80
Εικόνα 26. TakeDamage.....	81
Εικόνα 27. IsDead.....	81
Εικόνα 28. IK Rig & Retarget Chains	86
Εικόνα 29. IK Retargeter	87
Εικόνα 30. Retarget Pose Animation Blueprint.....	88
Εικόνα 31. Animation retargeting.....	89
Εικόνα 32. Blendspace (BS_A_8W)	91
Εικόνα 33. Event Blueprint Initialize	92
Εικόνα 34. Event Blueprint Update Animation (1)	93
Εικόνα 35. Event Blueprint Update Animation (2)	93
Εικόνα 36. Anim Graph.....	94
Εικόνα 37. Cached Pose Locomotion.....	95
Εικόνα 38. State Machine (Idle&Move).....	96
Εικόνα 39. Move State (Rifle Locomotion).....	97
Εικόνα 40. Cached Pose PatrolUnarmedLoco	98
Εικόνα 41. Move State (Android Locomotion)	98
Εικόνα 42. Cashed Pose Idle Breaks	99

Εικόνα 43. Idle State (Rifle Locomotion).....	100
Εικόνα 44. Modular Character Example.....	101
Εικόνα 45. Skeletal Mesh Setup	102
Εικόνα 46. Fire Function	106
Εικόνα 47. Fire Once Exit Conditions	107
Εικόνα 48. Fire Once (1).....	108
Εικόνα 49. Fire Once (2).....	109
Εικόνα 50. Fire Once (3).....	109
Εικόνα 51. Fire Functions Overview	110
Εικόνα 52. Weapon Sockets Setup (child blueprint)	111
Εικόνα 53. Weapon Attachment	112
Εικόνα 54. Behavior Tree Nodes Example (attack scenario)	118
Εικόνα 55. Navigation Mesh	121
Εικόνα 56. Sight Sense Configuration.....	125
Εικόνα 57. Hearing & Damage Sense Configuration	126
Εικόνα 58. OnPerceptionUpdated.....	127
Εικόνα 59. CanSenseActor	128
Εικόνα 60. EQS Query με ConeGrid.....	130
Εικόνα 61. EQS_Context.....	130
Εικόνα 62. Behavior Tree Overview	134
Εικόνα 63. Passive Behavior Tree Overview	137
Εικόνα 64. Subtree PatrolOrWait	138

Εικόνα 65. SplineComponent (PatrolRoute)	140
Εικόνα 66. Increment Patrol Route.....	141
Εικόνα 67. GetSplinePointsAsWorldPosition	141
Εικόνα 68. BTT_MoveAlongPatrolRoute	142
Εικόνα 69. BTT_SetMovementSpeed	143
Εικόνα 70. SetMovementSpeedAI (function).....	144
Εικόνα 71. BTT_FocusTarget.....	144
Εικόνα 72. BTT_ClearFocus	145
Εικόνα 73. BTT_EquipWeapon και BTT_UnequipWeapon.....	146
Εικόνα 74. Unequip Weapon.....	147
Εικόνα 75. BTD_HasPatrolRoute.....	149
Εικόνα 76. BTD_IsWeaponEquipped.....	149
Εικόνα 77. Set State As Passive.....	150
Εικόνα 78. Investigate Behavior Tree Overview	152
Εικόνα 79. GetCurrentState	153
Εικόνα 80. SetStateAsInvestigating.....	154
Εικόνα 81. Investigate State Icon	155
Εικόνα 82. BTT_StoreLocationPriorInvestigation	156
Εικόνα 83. Investigate Starting Animation.....	157
Εικόνα 84. Investigate Ending Animation.....	158
Εικόνα 85. CalculateNoiseDirection.....	158
Εικόνα 86. HandleSensedSound	160

Εικόνα 87. Throwable Base (Parent Blueprint)	163
Εικόνα 88. Start Item Trace	164
Εικόνα 89. Bottle Throw Overview	164
Εικόνα 90. OnComponentHit & Bottle Impact	165
Εικόνα 91. BottleStopped	165
Εικόνα 92. Player Token Setup.....	168
Εικόνα 93. ReserveAttackToken	168
Εικόνα 94. ReturnAttackToken	169
Εικόνα 95. StoreAttackTokens	170
Εικόνα 96. Attack Start (Attempt)	171
Εικόνα 97. AttackEnd	171
Εικόνα 98. Attacking Behavior Tree Overview (Ranged).....	172
Εικόνα 99. BTS_StopAttackingIfTargetIsDead	174
Εικόνα 100. BTD_CanSeeTarget	175
Εικόνα 101. BTT_Ranged_Attack.....	175
Εικόνα 102. AttackTest – Ranged	177
Εικόνα 103. BTD_IsHealthLow	177
Εικόνα 104. BTT_Heal	178
Εικόνα 105. EQS_FindCover	178
Εικόνα 106. EQS_FindIdealRngedLocation.....	179
Εικόνα 107. Attacking Behavior Tree Overview.....	181
Εικόνα 108. BT_Subtree_Aerial_Attack	182

Εικόνα 109. Teleport Demonstration.....	183
Εικόνα 110. Teleport & TeleportEnd	184
Εικόνα 111. BTT_AndroidTeleport.....	185
Εικόνα 112. BTT_Android_Attack.....	185
Εικόνα 113. GetAttackTeleportLocation	186
Εικόνα 114. Teleport Location Visualization	187
Εικόνα 115. BTT_GetAttackTeleportLocation	187
Εικόνα 116. Aerial Attack Demonstration.....	189
Εικόνα 117. Animation Montage Notifies	190
Εικόνα 118. Aerial Attack Blueprint Overview.....	192
Εικόνα 119. Slash and Damage	192
Εικόνα 120. Roll & Perfect Dodge Scenario	194
Εικόνα 121. Roll Blueprint Overview	195
Εικόνα 122. CheckIsPerfectDodge	196
Εικόνα 123. MontageNotifyState_ActivateAttack	197
Εικόνα 124. Ranged Agent Subtree.....	199
Εικόνα 125. Android Agent Subtree.....	200
Εικόνα 126. SetStateAsSeeking.....	201
Εικόνα 127. AfterImage Effect Demonstration	202
Εικόνα 128. SeekAttackTarget	202
Εικόνα 129. ShouldPlayAfterImage	203
Εικόνα 130. IncrementEnemiesActive & DecrementEnemiesActive	205

Εικόνα 131. FirstSeekAnim & SeekingAnimation.....	206
Εικόνα 132. EQS Seeking.....	207
Εικόνα 133. Detecting State Subtree	211
Εικόνα 134. BP_StealthSpot.....	212
Εικόνα 135. ConcealPlayer.....	214
Εικόνα 136. Conceal Overlay Material.....	214
Εικόνα 137. RevealPlayer.....	215
Εικόνα 138. Start Detecting.....	216
Εικόνα 139. SetStateAsDetecting.....	216
Εικόνα 140. Detection Animation	217
Εικόνα 141. Detection Reaction & Widget (Ranged).....	218
Εικόνα 142. Detection Reaction & Widget (Agent).....	218
Εικόνα 143. IsDetectionClose.....	219
Εικόνα 144. PlayDetectionSound	220
Εικόνα 145. Create Detection Widget	220
Εικόνα 146. UpdateDetectionAngle	221
Εικόνα 147. Start Detection Meter	222
Εικόνα 148. Player Detected & Stop Detection.....	223
Εικόνα 149. DistanceBasedDetection.....	224
Εικόνα 150. OnPlayerSeen (identify actor)	225
Εικόνα 151. OnPlayerSeen (<i>Player Seen</i>)	227
Εικόνα 152. OnPlayerSeen (<i>Lost Sight</i>)	228

Εικόνα 153. OnPlayerSeen (seen dead teammate)	228
Εικόνα 154. SetStateAsDead	231
Εικόνα 155. DeathWrapper.....	232
Εικόνα 156. OnPossessed	233
Εικόνα 157. OnUnpossessed.....	233
Εικόνα 158. Frozen State Subtree.....	236
Εικόνα 159. SetStateAsFrozen	237
Εικόνα 160. FrozenAttackingReactAnim & FrozenAttackingEndAnim	238
Εικόνα 161. FrozenPassiveAnim.....	239
Εικόνα 162. BP_DataWorm	240
Εικόνα 163. DataWormImpact	241
Εικόνα 164. DataWormStopped	242
Εικόνα 165. Initialize AOESphere & StopWorm/Reset Actors	243
Εικόνα 166. DataWormAffected	244
Εικόνα 167. DataWorm Demonstration.....	245
Εικόνα 168. SetUp Takedown	247
Εικόνα 169. Overlay Effect & Takedown Widget.....	248
Εικόνα 170. IsPlayerAware	249
Εικόνα 171. Takedown Input Event	250
Εικόνα 172. Takedown Front-Back.....	250
Εικόνα 173. Takedown Demonstration (1).....	251
Εικόνα 174. Assassination Chooser.....	252

Εικόνα 175. Assassination Takedown Warp Location (1)	253
Εικόνα 176. Assassination Takedown Warp Location (2)	254
Εικόνα 177. Assassination (Agents)	255
Εικόνα 178. Takedown Demonstration (2)	256
Εικόνα 179. Scanner (active)	257
Εικόνα 180. Quest Trigger	258
Εικόνα 181. Quest Menu Widget	258
Εικόνα 182. Polycam Mesh (Blender)	259
Εικόνα 183. Mesh to Metahuman	260
Εικόνα 184. HUD Wheel & Weapon Effects	261
Εικόνα 185. Balance Locomotion	262
Εικόνα 186. Swimming Locomotion	262
Εικόνα 187. Landscape (final version)	264
Εικόνα 188. UI Elements - Loot - XP	265
Εικόνα 189. Cinematic Preview 1 (level sequencer)	266
Εικόνα 190. Cinematic Preview 2	267

Εισαγωγή

Η τεχνητή νοημοσύνη (TN) στα ψηφιακά παιχνίδια αποτελεί έναν από τους ταχύτερα εξελισσόμενους τομείς της πληροφορικής, συνδυάζοντας στοιχεία προγραμματισμού, σχεδίασης και ανθρώπινης συμπεριφοράς. Με την αυξανόμενη απαίτηση για ρεαλισμό, πολυπλοκότητα και αλληλεπίδραση, η ανάγκη για προηγμένα συστήματα TN γίνεται όλο και πιο επιτακτική (Takács, Vincze & Eigner, 2024). Ιδιαίτερη θέση στον σχεδιασμό τέτοιων συστημάτων κατέχουν τα δέντρα συμπεριφοράς (Behavior Trees), τα οποία προσφέρουν έναν ευέλικτο και ιεραρχικά δομημένο τρόπο μοντελοποίησης της λογικής των χαρακτήρων μη παίκτη (NPCs) (Togelius & Yannakakis, 2018).

Τα Behavior Trees προσφέρουν μια ευέλικτη και οπτικά κατανοητή μέθοδο για την οργάνωση των συμπεριφορών των χαρακτήρων, επιτρέποντας στους προγραμματιστές να δημιουργούν πολύπλοκες και ρεαλιστικές αντιδράσεις σε διάφορες καταστάσεις παιχνιδιού. Η χρήση τους διευκολύνει την ανάπτυξη διαδραστικών και ελκυστικών παιχνιδιών, καθώς οι χαρακτήρες μπορούν να προσαρμόζονται στις ενέργειες του παίκτη με φυσικό και λογικό τρόπο.

Στη μηχανή ανάπτυξης παιχνιδιών Unreal Engine 5.4 (UE5), η ενσωμάτωσή τους γίνεται μέσω των Blueprints, ενός οπτικού συστήματος προγραμματισμού που επιτρέπει στους προγραμματιστές να δημιουργούν λογική παιχνιδιού χωρίς την ανάγκη για γραφή κώδικα. Μέσω των Blueprints, οι προγραμματιστές μπορούν να καθορίσουν τις ενέργειες που θα εκτελούνται από τους χαρακτήρες στο παιχνίδι, συνδυάζοντας τις δυνατότητες των Behavior Trees με την ευελιξία του οπτικού προγραμματισμού. Η οπτική αναπαράσταση της λογικής, σε συνδυασμό με τις δυνατότητες παραμετροποίησης και αλληλεπίδρασης με το περιβάλλον του παιχνιδιού, καθιστά τη μηχανή αυτή ιδανική για την ανάπτυξη έξυπνων, ρεαλιστικών NPCs (Butler & Oliver, 2024).

Η παρούσα διπλωματική εργασία εστιάζει στη μελέτη της εφαρμογής των Behavior Trees στην ανάπτυξη ενός 3D παιχνιδιού δράσης-περιπέτειας, χρησιμοποιώντας την Unreal Engine 5.4. Μέσω αυτής της μελέτης περίπτωσης, εξετάζονται τόσο οι βασικές αρχές και τεχνικές που διέπουν την ενσωμάτωσή τους στην τεχνητή νοημοσύνη του παιχνιδιού, όσο και οι προκλήσεις που προκύπτουν κατά τη διάρκεια της ανάπτυξης. Παράλληλα, στο πλαίσιο της ανάπτυξης των Behavior Trees, αναπτύχθηκαν επιπλέον λειτουργίες που ενισχύουν τη συνολική αίσθηση του ρεαλισμού και της εμπύθισης στο παιχνίδι, όπως σύστημα αποστολών, οπλισμός, UI στοιχεία και cinematic cutscenes. Αυτές οι επιπλέον υλοποιήσεις συνεργάζονται με τα Behavior Trees για να προσφέρουν μια συνεκτική και εμπλουτισμένη εμπειρία παιχνιδιού.

1. Θεωρητικό Υπόβαθρο

1.1. Unreal Engine

Η Unreal Engine, αναπτυγμένη από την Epic Games, είναι μία από τις πιο ισχυρές και δημοφιλείς μηχανές παιχνιδιών παγκοσμίως. Γνωστή για την υψηλή ποιότητα γραφικών, τη ρεαλιστική προσομοίωση φυσικής και τις εκτεταμένες δυνατότητες προσαρμογής, η Unreal Engine έχει καθιερωθεί ως ένα από τα βασικά εργαλεία για την ανάπτυξη παιχνιδιών σε διάφορα είδη. Η έκδοση 5.4 εισάγει σημαντικές αναβαθμίσεις, όπως βελτιώσεις στον φωτισμό πραγματικού χρόνου (Lumen), απόδοσης μοντέλων υψηλής λεπτομέρειας με μικρό υπολογιστικό κόστος (Nanite) και μια πιο φιλική προς τους χρήστες ροή εργασίας για την ανάπτυξη (Epic, 2024).

1.1.1 Blueprints

Ένας βασικός μηχανισμός που υποστηρίζει η Unreal Engine είναι τα Blueprints, το οπτικό σύστημα προγραμματισμού της. Σύμφωνα με την Epic Games (n.d.), το Blueprint Visual Scripting System της Unreal Engine παρέχουν μια διαισθητική και ευέλικτη διεπαφή που αποτελεί ένα ισχυρό εργαλείο ανάπτυξης λειτουργιών, το οποίο επιτρέπει τη δημιουργία, διαχείριση και σύνθεση στοιχείων gameplay μέσω ενός γραφικού περιβάλλοντος βασισμένου σε κόμβους (nodes). Το σύστημα αυτό καθιστά εφικτή την υλοποίηση πολύπλοκων λογικών χωρίς την ανάγκη γραφής παραδοσιακού κώδικα, προσφέροντας έτσι λειτουργίες που έως πρότινος απαιτούσαν γνώσεις C++. Παράλληλα, μέσω ειδικής σύνταξης (Blueprint Markup) στην C++, οι προγραμματιστές μπορούν να δημιουργούν θεμελιώδη συστήματα τα οποία οι σχεδιαστές μπορούν στη συνέχεια να επεκτείνουν οπτικά.

Τα Blueprints υιοθετούν την αντικειμενοστραφή λογική, επιτρέποντας τον ορισμό κλάσεων (Classes) και αντικειμένων εντός της μηχανής. Αποτελούν, ουσιαστικά, έναν συνδυασμό asset και κώδικα, με δυνατότητες όπως:

- Επέκταση δημιουργημένων κλάσεων
- Τροποποίηση προκαθορισμένων ιδιοτήτων
- Διαχείριση Components και των παραμέτρων τους κατά τη δημιουργία αντικειμένων

1.1.2 Είδη και Δομή των Blueprints

Ο πιο κοινός τύπος είναι το Blueprint Class, το οποίο επιτρέπει τη δημιουργία νέων τύπων Actor με ενσωματωμένη συμπεριφορά. Αυτά τα Blueprints δημιουργούνται και διαμορφώνονται μέσα στο περιβάλλον ανάπτυξης Unreal Editor και μπορούν να τοποθετηθούν στον κόσμο ως instances με πλήρη λειτουργικότητα.

Κάθε Blueprint περιλαμβάνει ορισμένες βασικές υποκατηγορίες:

Construction Script

Εκτελείται κατά τη δημιουργία ή τοποθέτηση ενός instance στον κόσμο. Χρησιμοποιείται για την αρχικοποίηση και προσαρμογή του αντικειμένου ανάλογα με το περιβάλλον. Για παράδειγμα, ένα φως μπορεί να εντοπίσει το έδαφος όπου τοποθετείται και να επιλέξει το κατάλληλο υλικό ή mesh.

Event Graph

Πρόκειται για τον βασικό κόμβο λογικής, όπου ορίζονται οι δυναμικές αντιδράσεις του asset στα γεγονότα του παιχνιδιού (όπως damage, overlaps, input κ.ά.). Οι ενέργειες που ορίζονται εδώ εφαρμόζονται σε όλα τα instances του Blueprint και καθορίζουν τη λειτουργία του.

Functions

Οι συναρτήσεις (functions) επιτρέπουν την ομαδοποίηση και επαναχρησιμοποίηση node-based λογικής, βελτιώνοντας τη σαφήνεια και τη δομή των Blueprints. Μπορούν να καλούνται από άλλα σημεία μέσα στο ίδιο Blueprint και συμβάλλουν στην οργάνωση πιο σύνθετων συστημάτων.

Variables

Οι μεταβλητές αποθηκεύουν δεδομένα ή αναφορές (references) σε άλλα αντικείμενα του κόσμου. Μπορούν να είναι τοπικές (private) ή προσβάσιμες από τον editor (exposed), επιτρέποντας την παραμετροποίηση του κάθε instance ξεχωριστά. Έτσι, κάθε Blueprint μπορεί να προσαρμόζεται με ευελιξία σε διαφορετικά σενάρια χρήσης.

Πέρα από τα Blueprint Classes, η Unreal Engine προσφέρει και άλλους εξειδικευμένους τύπους Blueprints που εξυπηρετούν διαφορετικές λειτουργίες στον σχεδιασμό και την αρχιτεκτονική των συστημάτων:

Blueprint Interface

Τα Blueprint Interfaces επιτρέπουν την επικοινωνία μεταξύ διαφορετικών Blueprints, λειτουργώντας ως ένα κοινό σύνολο συναρτήσεων χωρίς υλοποίηση. Κάθε Blueprint που ενσωματώνει ένα Interface μπορεί να υλοποιήσει αυτές τις συναρτήσεις ξεχωριστά. Πρόκειται ουσιαστικά για την εφαρμογή της έννοιας του interface από τον αντικειμενοστραφή προγραμματισμό, προσφέροντας ένα κοινό μέσο επικοινωνίας ανάμεσα σε διαφορετικούς τύπους αντικειμένων.

Τα Blueprint Interfaces δημιουργούνται μέσω του Unreal Editor και, παρότι παρέχουν δυνατότητες επικοινωνίας, συνοδεύονται από ορισμένους περιορισμούς:

- Δεν επιτρέπεται η προσθήκη μεταβλητών
- Δεν υποστηρίζουν επεξεργασία Event Graph
- Δεν μπορούν να περιλαμβάνουν Components

Level Blueprint

Κάθε επίπεδο (level) στο Unreal Engine περιλαμβάνει από προεπιλογή ένα Level Blueprint, το οποίο δρα ως το κοινό Event Graph του εκάστοτε επιπέδου. Μέσα σε αυτό μπορούν να καθοριστούν ενέργειες και ακολουθίες γεγονότων που σχετίζονται με ολόκληρο το επίπεδο ή με συγκεκριμένα instances μέσα σε αυτό. Σε αντίθεση με τα κλασικά Blueprints, δεν είναι δυνατή η δημιουργία νέων Level Blueprints μέσω του editor, διαθέτοντας ένα μοναδικό για κάθε επίπεδο.

Blueprint Macro Library

Οι βιβλιοθήκες μακροεντολών (Blueprint Macro Libraries) είναι συλλογές από macros, δηλαδή ανεξάρτητα γραφήματα (graphs) που μπορούν να επαναχρησιμοποιηθούν σε άλλα Blueprints ως μεμονωμένοι κόμβοι. Είναι ιδιαίτερα χρήσιμες για τη συγκέντρωση συχνά χρησιμοποιούμενων λογικών ακολουθιών, καθώς μπορούν να περιλαμβάνουν τόσο εκτελεστικούς όσο και δεδομενικούς ακροδέκτες (pins).

1.2. Δέντρα Συμπεριφοράς (Behavior Trees)

Τα δέντρα συμπεριφοράς είναι μια ευέλικτη και οργανωμένη δομή που χρησιμοποιείται ευρέως στην ανάπτυξη TN για ηλεκτρονικά παιχνίδια. Αποτελούν μια από τις πιο δημοφιλείς τεχνικές για τη διαχείριση των συμπεριφορών των NPCs, προσφέροντας μια ιεραρχική προσέγγιση που διευκολύνει την ανάπτυξη και την προσαρμοστικότητα (Millington & Funge 2009).

Σε αντίθεση με τις παραδοσιακές μηχανές καταστάσεων (state machines), οι οποίες είναι συχνά δύσκολο να επεκταθούν και να διαχειριστούν σε σύνθετα σενάρια, τα Behavior Trees παρέχουν έναν πιο αρθρωτό και διαισθητικό τρόπο διαχείρισης της λογικής συμπεριφοράς. Η βασική αρχή πίσω από τα Behavior Trees είναι η διάσπαση μιας συμπεριφοράς σε μικρότερα, διαχειρίσιμα μέρη, τα οποία οργανώνονται σε ένα ιεραρχικό δέντρο (Biggar, Zamani & Shames, 2020).

1.2.1. Δομή και Βασικές Αρχές

Τα Behavior Trees στην Unreal Engine δημιουργούνται με οπτικό τρόπο, παρόμοιο με τα Blueprints. Ο χρήστης προσθέτει και συνδέει μια σειρά από nodes σε ένα γράφημα λογικής (Behavior Tree Graph), όπου κάθε node

ενσωματώνει μια συγκεκριμένη λειτουργικότητα. Για την αποθήκευση και διαχείριση πληροφορίας που είναι απαραίτητη κατά την εκτέλεση της λογικής, χρησιμοποιείται ένα ξεχωριστό αντικείμενο που ονομάζεται Blackboard. Σε αυτό αποθηκεύονται Blackboard Keys, δηλαδή μεταβλητές που επηρεάζουν τις αποφάσεις ή τη ροή του δέντρου (Epic Games, n.d.).

Η εκτέλεση του Behavior Tree ακολουθεί κατεύθυνση αριστερά προς δεξιά και από πάνω προς τα κάτω, σύμφωνα με τη διάταξη των nodes. Οι βασικοί τύποι που χρησιμοποιούνται είναι:

- *Κόμβοι Ροής (Control Nodes)*: Ορίζουν τη λογική δομή και τη σειρά εκτέλεσης των υπόλοιπων nodes. Παραδείγματα περιλαμβάνουν τους Selector, Sequence και Decorator Nodes, που καθορίζουν ποια διαδρομή πρέπει να ακολουθηθεί με βάση την επιτυχία ή αποτυχία άλλων nodes.
- *Κόμβοι Ενέργειας (Action Nodes)*: Εκτελούν συγκεκριμένες ενέργειες, όπως το να μετακινηθεί ο χαρακτήρας σε μια τοποθεσία ή να εκτελέσει μια επίθεση. Αντιπροσωπεύουν τις απτές πράξεις που επιτελεί η TN.
- *Κόμβοι Κατάστασης (Condition Nodes)*: Αξιολογούν συνθήκες και επιστρέφουν επιτυχία ή αποτυχία. Βασίζονται συνήθως σε τιμές του Blackboard και καθορίζουν εάν πρέπει να συνεχιστεί η εκτέλεση ενός node ή υποκλάδου του δέντρου.

Αφού σχεδιαστεί το Behavior Tree και καθοριστεί η λογική του, εκτελείται κατά τη διάρκεια του gameplay. Συνήθως, ένα Pawn (δηλαδή ένας Actor, όπως ένας χαρακτήρας ή οντότητα) είναι υπεύθυνο για την εκτέλεσή του. Το Pawn συνοδεύεται από ένα AI Controller, το οποίο αναλαμβάνει τη διαχείριση και την καθοδήγησή του, συμπεριλαμβανομένης της εκκίνησης του Behavior Tree. Ένα Pawn που χρησιμοποιεί τα assets Behavior Tree και AI Controller κατά το gameplay ονομάζεται και Agent. Για να καταστεί δυνατή η μετακίνηση ενός Agent στον κόσμο, απαιτείται η τοποθέτηση ενός NavMeshBoundsVolume στο Level, που καθορίζει τους πλοηγήσιμους χώρους.

1.2.2. Παραδείγματα χρήσης

Σύμφωνα με την Κωσταρέλου (2024), η χρήση προσαρμοστικών δέντρων συμπεριφοράς μπορεί να ενισχύσει την αληθοφάνεια και την εμπλοκή των NPCs, ενσωματώνοντας συναισθηματικές αντιδράσεις στις συμπεριφορές τους. Για παράδειγμα, ένας NPC σε ένα παιχνίδι δράσης-περιπέτειας μπορεί να αντιδρά δυναμικά:

- Εάν ο παίκτης πλησιάσει με απειλητικό τρόπο, ο NPC μπορεί να υιοθετήσει μια επιθετική συμπεριφορά
- Εάν ο παίκτης κρατήσει απόσταση, ο NPC μπορεί να αναζητήσει κάλυψη ή να υιοθετήσει πιο αμυντική στάση

Αυτή η ικανότητα αντίδρασης στις ενέργειες του παίκτη και στις αλλαγές του περιβάλλοντος καθιστά τα δέντρα συμπεριφοράς βασικό εργαλείο για τη δημιουργία ρεαλιστικών και καθηλωτικών εμπειριών παιχνιδιού.

1.2.3 Πλεονεκτήματα και μειονεκτήματα

Τα Behavior Trees αποτελούν μία από τις πιο δημοφιλείς επιλογές για την υλοποίηση τεχνητής νοημοσύνης σε NPCs, καθώς προσφέρουν πληθώρα πλεονεκτημάτων που ενισχύουν την ευελιξία, τη συντήρηση και την επεκτασιμότητα του συστήματος.

Πλεονεκτήματα

- *Ευελιξία*: Η αρχιτεκτονική τους επιτρέπει την εύκολη προσθήκη νέων συμπεριφορών χωρίς την ανάγκη ριζικών αλλαγών στο υπόλοιπο σύστημα. Αυτό καθιστά τα Behavior Trees ιδιαίτερα κατάλληλα για έργα που εξελίσσονται σταδιακά.
- *Αρθρωτή δομή (Modularity)*: Οι κόμβοι του δέντρου μπορούν να επαναχρησιμοποιηθούν σε διαφορετικά σενάρια ή τύπους NPCs, μειώνοντας τον πλεονάζοντα κώδικα και ενισχύοντας την επαναχρησιμοποίηση.
- *Καθαρός διαχωρισμός ευθυνών*: Η λογική της συμπεριφοράς είναι οργανωμένη με τρόπο που δεν απαιτεί από κάθε κόμβο να γνωρίζει την κατάσταση του συνόλου. Οι αποφάσεις λαμβάνονται σε ανώτερο επίπεδο (γονικοί κόμβοι), γεγονός που ενισχύει την αναγνωσιμότητα και τη συντήρηση του κώδικα.
- *Διαφάνεια και ευκολία στον εντοπισμό σφαλμάτων (debugging)*: Η ιεραρχική και γραφική αναπαράσταση της συμπεριφοράς καθιστά τη ροή λογικής ευκολότερα κατανοητή τόσο από προγραμματιστές όσο και από σχεδιαστές παιχνιδιών, διευκολύνοντας τη συνεργασία μεταξύ τους και την εντοπισμό λαθών.
- *Προσαρμοστικότητα σε δυναμικές καταστάσεις*: Τα Behavior Trees υποστηρίζουν την ενσωμάτωση σύνθετων παραμέτρων, όπως συναισθήματα, περιβαλλοντικές μεταβλητές και τιμές αξιολόγησης, προσδίδοντας στους NPCs πιο ρεαλιστική και ποικιλόμορφη συμπεριφορά (Colledanchise & Ögren, 2018).

Μειονεκτήματα

Παρά τα πλεονεκτήματά τους, τα Behavior Trees ενδέχεται να παρουσιάσουν περιορισμούς, ειδικά σε έργα χωρίς την απαραίτητη τεχνική υποστήριξη:

- *Ανάγκη για εξειδικευμένα εργαλεία*: Η πλήρης αξιοποίηση των Behavior Trees απαιτεί υποστήριξη για debugging, serialization και οπτικοποίηση. Η απουσία αυτών των εργαλείων μπορεί να δυσκολέψει την ανάπτυξη και τη συντήρηση.
- *Πολυπλοκότητα σε μεγάλα δέντρα*: Χωρίς σωστό αρχικό σχεδιασμό, η δομή ενός Behavior Tree μπορεί να γίνει υπερβολικά περίπλοκη και δύσκολα διαχειρίσιμη, με πολλούς κόμβους και περίπλοκες αλληλεπιδράσεις.
- *Απότομη καμπύλη εκμάθησης*: Σε σύγκριση με απλούστερα συστήματα, όπως finite state machines, τα Behavior Trees απαιτούν εις βάθος κατανόηση των αρχών σχεδιασμού και χρήσης, κάτι που μπορεί να αποδειχθεί αποτρεπτικό για αρχάριους.
- *Δυσκολία μετάβασης καταστάσεων*: Είναι συχνά δύσκολο να οριστούν διαφοροποιήσεις σε καταστάσεις. Για παράδειγμα, όταν ένας NPC εισέρχεται στην κατάσταση περιπολίας, μπορεί να απαιτείται διαφορετική συμπεριφορά αν αυτός εισέρχεται σε αυτήν την κατάσταση μετά από μια επίθεση ή από μια φάση αδράνειας. Αυτές οι διαφοροποιήσεις απαιτούν επιπλέον σύνθετους ελέγχους της προηγούμενης κατάστασης του NPC, αυξάνοντας την πολυπλοκότητα του Behavior Tree και καθιστώντας τον πιο δύσχρηστο και δύσκολο στη διαχείριση.
- *Εξάρτηση από το Blackboard*: Αν και το Blackboard προσφέρει έναν οργανωμένο τρόπο αποθήκευσης και ανταλλαγής δεδομένων μεταξύ των εργασιών του Behavior Tree, η στενή σύνδεση μεταξύ τους μπορεί να δημιουργήσει προβλήματα κατά την ανάπτυξη, ιδιαίτερα στην φάση του πρωτοτύπου. Οποιαδήποτε αλλαγή στη δομή του Blackboard μπορεί να απαιτήσει αναθεώρηση και προσαρμογή των εργασιών του Behavior Tree, με αποτέλεσμα την αύξηση της πολυπλοκότητας και την καθυστέρηση της ανάπτυξης. Επιπλέον, αυτή η εξάρτηση περιορίζει την ευχέρεια πειραματισμών και βελτιώσεων στο σύστημα, καθώς οι αλλαγές σε έναν τομέα επηρεάζουν και τον άλλον. Η αποσύνδεση των δύο συστημάτων σε ορισμένα σημεία, όταν η ανάπτυξη το επιτρέπει, θα μπορούσε να προσφέρει πιο ευέλικτες και αποδοτικές μεθόδους επικοινωνίας μεταξύ των εργασιών, καθιστώντας τη διαδικασία πιο αποτελεσματική και λιγότερο επιρρεπή σε προβλήματα (Francis, 2017).

Τα Behavior Trees προσφέρουν μια ιεραρχικά δομημένη, ευέλικτη και επεκτάσιμη λύση για την ανάπτυξη τεχνητής νοημοσύνης σε περιβάλλοντα με σύνθετες απαιτήσεις. Ενδείκνυνται ιδιαίτερα σε παιχνίδια με πολυεπίπεδη λογική NPCs, ενώ η δυνατότητα ενσωμάτωσης εργαλείων debugging και visual scripting ενισχύει την αποδοτικότητα των προγραμματιστικών ομάδων. Ωστόσο, η αποδοτική χρήση τους απαιτεί επαρκή σχεδιασμό, υποστηρικτικά εργαλεία και δομημένη προσέγγιση, καθώς διαφορετικά ελλοχεύει ο κίνδυνος η αρχική ευελιξία να μετατραπεί σε διαχειριστική πρόκληση.

1.3. Εφαρμογές Δέντρων Συμπεριφοράς σε Παιχνίδια

Η χρήση των δέντρων συμπεριφοράς έχει καθιερωθεί ως βασική τεχνική στην ανάπτυξη TN για ηλεκτρονικά παιχνίδια, συμβάλλοντας στη δημιουργία ρεαλιστικών και διαδραστικών NPCs. Τα δέντρα συμπεριφοράς προσφέρουν στους προγραμματιστές την ευελιξία να διαχειρίζονται πολύπλοκες καταστάσεις με οργανωμένο τρόπο, ενώ παράλληλα επιτρέπουν τη βελτιστοποίηση της εμπειρίας του παίκτη.

Πολλά παιχνίδια αποτελούν παραδείγματα της αποτελεσματικότητας των δέντρων συμπεριφοράς στη διαμόρφωση της νοημοσύνης των NPCs. Σύμφωνα με τους Thompson (2025) και Argentics (2025), παραδείγματα αποτελούν τα ακόλουθα:

- *Hitman*: Στο Hitman, τα δέντρα συμπεριφοράς ενορχηστρώνουν τις περίπλοκες διαδικασίες λήψης αποφάσεων των NPCs, καθοδηγώντας τους μέσα από ένα ευρύ φάσμα δυναμικών ενεργειών, από περιπολίες έως την δυναμική ανταπόκριση σε απροσδόκητες ενέργειες του παίκτη. Τα πολύπλοκα και εξελισσόμενα περιβάλλοντα του παιχνιδιού επωφελούνται από τα δέντρα συμπεριφοράς, συμβάλλοντας στην καθηλωτική και απαιτητική φύση της εμπειρίας του παίκτη.
- *Assassin's Creed*: Η σειρά Assassin's Creed παρέχει μια ακόμη συναρπαστική μελέτη περίπτωσης, όπου τα δέντρα συμπεριφοράς επιτρέπουν στους NPCs να προσαρμόζονται δυναμικά στις μεταβαλλόμενες συνθήκες. Από τις αντιδράσεις του πλήθους έως τις στρατηγικές μάχης, η εφαρμογή των δέντρων συμπεριφοράς ενισχύει το ρεαλισμό και την απρόβλεπτη φύση του κόσμου του παιχνιδιού.
- *Middle-earth: Shadow of Mordor*: Αυτό το παιχνίδι χρησιμοποιεί έναν συνδυασμό δέντρων συμπεριφοράς και διαδικαστικής δημιουργίας περιεχομένου παράλληλα με μηχανική μάθηση για το Nemesis System. Το σύστημα προσαρμόζεται δυναμικά στις ενέργειες του παίκτη, δημιουργώντας μια ιεραρχία NPCs με μοναδικές προσωπικότητες, δυνάμεις και αδυναμίες.
- *The Elder Scrolls V: Skyrim*: Ενσωματώνοντας δέντρα συμπεριφοράς με μηχανική μάθηση, το Skyrim δημιουργεί NPCs με προσαρμοστικά προγράμματα και συμπεριφορές. Οι καθημερινές ρουτίνες των NPCs επηρεάζονται από τις ενέργειες του παίκτη, με αποτέλεσμα έναν ζωντανό κόσμο που αντιδρά στις επιλογές του παίκτη.
- *Watch Dogs 2*: Το παιχνίδι χρησιμοποιεί έναν συνδυασμό νευρωνικών δικτύων και δέντρων συμπεριφοράς για να δημιουργήσει NPCs με ρεαλιστικές αντιδράσεις στις ενέργειες του παίκτη. Τα νευρωνικά δίκτυα συμβάλλουν στη λήψη λεπτών αποφάσεων, ενώ τα δέντρα συμπεριφοράς καθοδηγούν τη συνολική δομή των συμπεριφορών των NPCs σε απόκριση στο δυναμικό περιβάλλον του παιχνιδιού.
- *Far Cry Primal*: Στο Far Cry Primal, τα δέντρα συμπεριφοράς αποτελούν τη θεμελιώδη αρχιτεκτονική για τη συμπεριφορά όλων των χαρακτήρων, συμπεριλαμβανομένων των άγριων ζώων που μπορούν να εξημερωθούν ως σύντροφοι (companions). Για τη διαχείριση των συντρόφων ζώων, το παιχνίδι χρησιμοποιεί μια τεχνική δυναμικής εισαγωγής δέντρων συμπεριφοράς. Όταν ο παίκτης δίνει μια εντολή σε ένα ζώο, ένα νέο, μικρότερο δέντρο συμπεριφοράς εισάγεται προσωρινά στο υπάρχον σύστημα AI, καθορίζοντας την άμεση αντίδραση του ζώου στην εντολή. Αυτό επιτρέπει στους συντρόφους να εκτελούν τις εντολές του παίκτη, ενώ παράλληλα διατηρούν ορισμένες ενστικτώδεις αντιδράσεις και αποφεύγουν καταστάσεις που θα έθεταν σε κίνδυνο την επιβίωσή τους. Η ιεράρχηση των δέντρων συμπεριφοράς διασφαλίζει ότι οι εντολές του παίκτη έχουν προτεραιότητα, αλλά δεν καταργούν πλήρως την αυτόνομη συμπεριφορά των ζώων.

1.4. Συνδυασμός Behavior Trees με Άλλες Τεχνικές AI

Ένα από τα μεγαλύτερα πλεονεκτήματα των behavior trees είναι η δυνατότητά τους να συνδυάζονται με άλλες τεχνικές TN, όπως:

- **Μηχανές Αποφάσεων (Decision Trees):** Οι μηχανές αποφάσεων μπορούν να χρησιμοποιηθούν για τη λήψη στρατηγικών αποφάσεων, ενώ τα behavior trees εστιάζουν στη διαχείριση της τακτικής εκτέλεσης (Costa & Pedreira, 2023).
- **Αλγόριθμοι Μάθησης Μηχανής (Machine Learning):** Τα behavior trees μπορούν να επωφεληθούν από τη μάθηση μηχανής για τη βελτίωση της προσαρμοστικότητας των NPCs. Για παράδειγμα, ένας NPC μπορεί να μάθει να αλλάζει τη συμπεριφορά του με βάση τις προηγούμενες ενέργειες του παίκτη (Tomai & Flores, 2014).

Η ενσωμάτωση αυτών των τεχνικών διευρύνει τις δυνατότητες της TN στα παιχνίδια, επιτρέποντας τη δημιουργία χαρακτήρων που μπορούν να αλληλεπιδρούν πιο φυσικά και ρεαλιστικά με τον κόσμο του παιχνιδιού.

1.5. Προσαρμοστικά Δέντρα Συμπεριφοράς και Συναισθηματική Αντίδραση

Στην έρευνα της Κωσταρέλου (2024), δίνεται έμφαση στην προσθήκη συναισθηματικών αντιδράσεων στα Behavior Trees, δημιουργώντας NPCs που ανταποκρίνονται δυναμικά στις αλληλεπιδράσεις των παικτών και στις περιβαλλοντικές αλλαγές. Για παράδειγμα:

- Ένας NPC σε ένα παιχνίδι τρόμου μπορεί να εμφανίζει φόβο ή επιθετικότητα ανάλογα με τη στάση του παίκτη.
- Σε παιχνίδια στρατηγικής, οι NPCs μπορούν να προσαρμόζουν τη συμπεριφορά τους όταν αντιλαμβάνονται ότι βρίσκονται σε μειονεκτική θέση, υιοθετώντας πιο αμυντικές στρατηγικές.

Αυτή η προσέγγιση όχι μόνο ενισχύει την εμπύθιση του παίκτη, αλλά και συμβάλλει στη δημιουργία πιο ζωντανών χαρακτήρων που παρουσιάζουν ανθρώπινα χαρακτηριστικά. Η δυνατότητα προσαρμογής στις ενέργειες του παίκτη και η χρήση συναισθημάτων καθιστούν τα Behavior Trees ιδανικά για παιχνίδια που απαιτούν πολυπλοκότητα και δυναμικότητα στις αλληλεπιδράσεις.

1.6. Εναλλακτικές μέθοδοι ανάπτυξης τεχνητής νοημοσύνης σε παιχνίδια

Η ανάπτυξη τεχνητής νοημοσύνης για NPCs μπορεί να υλοποιηθεί με διάφορες προσεγγίσεις. Παρόλο που τα Behavior Trees έχουν εδραιωθεί ως ένα δημοφιλές πρότυπο τα τελευταία χρόνια, δεν αποτελούν τη μοναδική στρατηγική σχεδίασης. Υπάρχουν εναλλακτικά συστήματα με διαφορετικά πλεονεκτήματα, περιορισμούς και κατάλληλα πεδία εφαρμογής. Τα πιο χαρακτηριστικά είναι τα εξής: Unstructured AI (Μη Δομημένος Προγραμματισμός), Finite State Machines (FSM) και Goal-Oriented Action Planning (GOAP) (Millington & Funge 2009).

1.6.1. Unstructured AI (Μη Δομημένος Προγραμματισμός)

Η προσέγγιση αυτή βασίζεται στη δημιουργία της λογικής του NPC απευθείας μέσα στον κώδικα, χωρίς συγκεκριμένο αρχιτεκτονικό πλαίσιο. Ο προγραμματιστής καθορίζει απλά τις ενέργειες και τις συνθήκες υπό τις οποίες εκτελούνται (π.χ. κινήγι, επίθεση, θάνατος), συνδέοντας τις διαδικασίες με τρόπο αυθαίρετο και χωρίς αυστηρή δομή.

Πλεονεκτήματα:

- Απλότητα και ταχύτητα υλοποίησης: Είναι ο πιο άμεσος και γρήγορος τρόπος ανάπτυξης, ειδικά σε περιπτώσεις με πολύ βασική συμπεριφορά.

- Δεν απαιτούνται εξειδικευμένα εργαλεία: Η υλοποίηση γίνεται αποκλειστικά με τον υπάρχοντα κώδικα, χωρίς πρόσθετες βιβλιοθήκες ή συστήματα.

Μειονεκτήματα:

- Περιορισμένη επεκτασιμότητα: Καθώς αυξάνεται η πολυπλοκότητα, ο κώδικας γίνεται δύσκολα διαχειρίσιμος και γεμάτος εξαρτήσεις.
- Προβληματική συντήρηση και εντοπισμός σφαλμάτων: Η απουσία δομής καθιστά δύσκολη την κατανόηση και επιδιόρθωση προβλημάτων.
- Εσωτερική αλληλεξάρτηση: Οι συμπεριφορές των NPCs εξαρτώνται έντονα η μία από την άλλη, με αποτέλεσμα αυξημένο ρίσκο σφαλμάτων και ασυνέπειας.

Πλαίσιο Χρήσης:

- Κατάλληλο για πολύ απλές συμπεριφορές, όπως εχθρικά zombies με μονοδιάστατη συμπεριφορά (π.χ., κινούνται προς τον παίκτη, επιτίθενται όταν φτάσουν κοντά, και πεθαίνουν όταν εξαντληθεί η υγεία τους).
- Ακατάλληλο για σύνθετους εχθρούς (π.χ., bosses) που διαθέτουν πολλαπλές καταστάσεις και στρατηγικές.

1.6.2. Finite State Machines (FSM)

Η τεχνική των FSM βασίζεται στην ύπαρξη διακριτών "καταστάσεων" (states) που περιγράφουν διαφορετικές συμπεριφορές του NPC (π.χ. περιπολία, μάχη, υποχώρηση), με σαφώς καθορισμένες μεταβάσεις μεταξύ τους υπό συγκεκριμένες συνθήκες.

Πλεονεκτήματα:

- Ευκολία κατανόησης και υλοποίησης: Κάθε κατάσταση περιέχει συγκεκριμένη λογική, με σαφείς μεταβάσεις.
- Υποστήριξη οπτικοποίησης: Πολλά εργαλεία παρέχουν visual debugging και αποτύπωση της ροής καταστάσεων.
- Επαναχρησιμοποίηση: Οι FSMs μπορούν να χρησιμοποιηθούν και για άλλες λειτουργίες, όπως η αναπαραγωγή animations.

Μειονεκτήματα:

- Χαμηλή επεκτασιμότητα: Ο αριθμός των μεταβάσεων αυξάνεται εκθετικά με τον αριθμό των καταστάσεων (N^2 μεταβάσεις για N καταστάσεις), οδηγώντας σε δυσκολία διαχείρισης.
- Δύσκολη διαχείριση σύνθετων περιπτώσεων: Καθώς προστίθενται νέες συμπεριφορές, η ανάγκη ενημέρωσης πολλών μεταβάσεων οδηγεί σε ανεπιθύμητη πολυπλοκότητα.
- Δυνητική επικάλυψη καταστάσεων: Καταστάσεις που επηρεάζονται από εξωτερικά γεγονότα μπορεί να απαιτούν αναδιάρθρωση του μοντέλου.

Πλαίσιο Χρήσης:

- Κατάλληλο για NPCs με περιορισμένη, αλλά όχι στοιχειώδη πολυπλοκότητα, όπως εχθρούς που μπορούν να περιπολούν, να επιτίθενται και να εξουδετερώνονται.
- Μη προτεινόμενο για δυναμικά και εξελικτικά συστήματα ή στρατηγικές συμπεριφορές που εξαρτώνται από πολλές μεταβλητές.

1.6.3. Goal-Oriented Action Planning (GOAP)

Το GOAP αποτελεί μία πιο δυναμική προσέγγιση στον σχεδιασμό τεχνητών συμπεριφορών, όπου ο NPC δεν ακολουθεί προαποφασισμένες αλληλουχίες καταστάσεων αλλά επιλέγει ενέργειες βάσει στόχων και της τρέχουσας

κατάστασης του κόσμου. Οι διαθέσιμες ενέργειες ορίζονται με προϋποθέσεις και αποτελέσματα, και ο NPC σχεδιάζει αυτόματα ένα βέλτιστο μονοπάτι ενεργειών προς τον στόχο του.

Πλεονεκτήματα:

- Ευφυής και δυναμική συμπεριφορά: Οι NPC μπορούν να προσαρμόζονται σε απρόβλεπτες καταστάσεις, επιδεικνύοντας δυναμική αντίδραση.
- Επεκτασιμότητα χωρίς αναδόμηση: Ο σχεδιασμός επεκτείνεται οργανικά προσθέτοντας νέες ενέργειες, χωρίς να τροποποιείται ολόκληρη η υπάρχουσα λογική.
- Απλοποίηση συντήρησης: Η λογική βασίζεται σε καθολικούς στόχους και όχι σε αυστηρές ακολουθίες καταστάσεων και άκαμπτων δομών.

Μειονεκτήματα:

- Αυξημένο υπολογιστικό κόστος: Το runtime planning μπορεί να είναι υπολογιστικά ακριβό, ιδιαίτερα σε παιχνίδια με πολλούς NPCs.
- Πολυπλοκότητα υλοποίησης: Απαιτεί δημιουργία συστήματος σχεδιασμού (planner), διαχείρισης στόχων, και καθορισμού συνθηκών/αποτελεσμάτων.
- Περιορισμένη οπτικοποίηση: Σε αντίθεση με τα behavior trees, η λογική ροή δεν είναι άμεσα ορατή ή γραφικά απεικονίσιμη.

Πλαίσιο Χρήσης:

- Ιδανικό για NPCs με προσαρμοστική και στρατηγική σκέψη, όπως σύμμαχους συνοδούς, NPCs με εναλλασσόμενα κίνητρα, ή bots σε στρατηγικά παιχνίδια.
- Λιγότερο κατάλληλο για απλές συμπεριφορές εχθρών ή NPCs με περιορισμένα σενάρια δράσης και αυστηρά καθορισμένες συμπεριφορές.

Κάθε μία από τις παραπάνω μεθόδους προσφέρει διαφορετικά πλεονεκτήματα και περιορισμούς. Η επιλογή της κατάλληλης τεχνικής εξαρτάται από τις απαιτήσεις του παιχνιδιού, την πολυπλοκότητα των συμπεριφορών και τις δυνατότητες συντήρησης στο μέλλον. Για έργα υψηλής πολυπλοκότητας με ανάγκη δυναμικής προσαρμογής, το GOAP υπερέρχει, ενώ για πιο κλασικές περιπτώσεις η FSM μπορεί να προσφέρει ένα ισορροπημένο μείγμα απλότητας και δυνατότητας ελέγχου.

2. Εργαλεία και Πλατφόρμες Ανάπτυξης

Η ανάπτυξη του *No Man's Land* υλοποιήθηκε με τη χρήση της Unreal Engine 5, μίας εκ των πλέον εξελιγμένων και ευρέως χρησιμοποιούμενων μηχανών παιχνιδιών παγκοσμίως. Η έκδοση 5.4 της Unreal Engine αξιοποιήθηκε ως η βασική πλατφόρμα υλοποίησης, προσφέροντας προηγμένα εργαλεία για τη δημιουργία ρεαλιστικών τρισδιάστατων περιβαλλόντων, την ανάπτυξη μηχανισμών τεχνητής νοημοσύνης, καθώς και τη διαχείριση animation, φυσικής και φωτισμού σε πραγματικό χρόνο.

Συγκεκριμένα, για τον σχεδιασμό και την αλληλεπίδραση των NPCs, αξιοποιήθηκαν Behavior Trees, AI Controllers, Blackboards και άλλες τεχνικές διαμόρφωσης συμπεριφοράς. Επιπλέον, συστήματα όπως τα Blendspaces και το Motion Matching χρησιμοποιήθηκαν για την ανάπτυξη ρεαλιστικής κινησιολογίας των χαρακτήρων. Στην αισθητική και λειτουργική σχεδίαση του παιχνιδιού ενσωματώθηκαν επίσης τεχνολογίες όπως το Lumen για τον δυναμικό φωτισμό και το Niagara για την παραγωγή ειδικών εφέ, ενώ η διεπαφή χρήστη σχεδιάστηκε με χρήση UI Widgets.

Το έργο ενσωματώνει επίσης εργαλεία από το ευρύτερο οικοσύστημα της Epic Games, όπως ο MetaHuman Creator, ο οποίος χρησιμοποιήθηκε για την κατασκευή ρεαλιστικών ανθρώπινων χαρακτήρων. Συμπληρωματικά, χρησιμοποιήθηκαν επιλεγμένα δωρεάν projects από το Unreal Marketplace, για την ενσωμάτωση assets όπως animations, αντικείμενα και περιβαλλοντικά στοιχεία.

Όσον αφορά στον ηχητικό σχεδιασμό, αξιοποιήθηκαν πλατφόρμες σύνθεσης φωνής με τεχνητή νοημοσύνη, όπως το Eleven Labs, για την παραγωγή διαλόγων τόσο των NPCs όσο και του βασικού χαρακτήρα. Επιπλέον, το εικαστικό υλικό της διεπαφής χρήστη (UI) σχεδιάστηκε με γνώμονα την αισθητική ταυτότητα και τις λειτουργικές απαιτήσεις του παιχνιδιού.

Στο πλαίσιο της δημιουργίας και επεξεργασίας περιεχομένου, χρησιμοποιήθηκαν επιπρόσθετα εργαλεία όπως το Blender για τη μοντελοποίηση και τροποποίηση 3D αντικειμένων, το Ableton Live για την παραγωγή και μίξη ηχητικών στοιχείων και το DaVinci Resolve για το μοντάζ των cinematic cutscenes.

Στο παρόν κεφάλαιο παρουσιάζονται αναλυτικά τα επιμέρους εργαλεία και πλατφόρμες που χρησιμοποιήθηκαν κατά την ανάπτυξη του έργου, με έμφαση τόσο στις λύσεις που παρέχει το οικοσύστημα της Unreal Engine όσο και σε εξωτερικές πλατφόρμες που συνέβαλαν στον τεχνικό και καλλιτεχνικό εμπλουτισμό του τελικού αποτελέσματος.

2.1. Unreal Engine 5 και Τεχνολογίες Συστήματος

Η Unreal Engine 5 προσφέρει μια σειρά από καινοτόμες τεχνολογίες και εγγενή συστήματα για τη ανάπτυξη και τη διαχείριση AI NPC's αλλά και για τη δημιουργία χαρακτήρων, της κινησιολογίας τους, οπτικοακουστικών εφέ και περιβάλλοντος.

2.1.1. Behavior Trees

Η υλοποίηση της TN των NPCs βασίζεται στο σύστημα Behavior Trees, το οποίο επιτρέπει τον καθορισμό σύνθετων συμπεριφορών βάσει ακολουθιών ενεργειών και συνθηκών. Η υλοποίηση των Behavior Trees καθιστά εφικτή τη δημιουργία διαφορετικών καταστάσεων συμπεριφοράς και παρέχει στους εχθρούς εξατομικευμένες στρατηγικές και αντιδράσεις με βάση την ανίχνευση ήχου και κίνησης, δίνοντας στον παίκτη τη δυνατότητα να αλληλεπιδράσει με τον κόσμο με διαφορετικούς τρόπους (π.χ. μέσω stealth ή άμεσης επίθεσης). Το συγκεκριμένο σύστημα υποστηρίζεται από εργαλεία όπως οι AI Controllers, Decorators, Subtrees και EQS Queries για την περιβαλλοντική αντίληψη των χαρακτήρων. Η ενσωμάτωσή τους επιτρέπει στους χαρακτήρες να αντιδρούν ρεαλιστικά σε καταστάσεις όπως η περιπολία, η έρευνα και η επίθεση, ενισχύοντας τη διαδραστικότητα και τον ρεαλισμό του παιχνιδιού.

2.1.2. Motion Matching και State Machines

Η διαχείριση της κίνησης των χαρακτήρων βασίζεται στον συνδυασμό BlendSpaces (Epic, n.d.) και State Machines (Epic, n.d.), επιτρέποντας την επιλογή κατάλληλων animation βάσει παραμέτρων όπως κατεύθυνση και ταχύτητα. Για την κινησιολογία του παίκτη αξιοποιείται το σύστημα Motion Matching (Epic, n.d.), το οποίο επιλέγει δυναμικά το καταλληλότερο animation από μια εκτεταμένη βάση δεδομένων κινήσεων. Το σύστημα αυτό προσαρμόζεται στη φυσική αλληλεπίδραση του χαρακτήρα με τον κόσμο (π.χ. άλματα, κολύμβηση, παρκούρ), προσδίδοντας υψηλό βαθμό ρεαλισμού.

2.1.3. Niagara Particle Systems και UI Widgets

Το σύστημα Niagara (Epic, n.d.) αξιοποιείται για την παραγωγή ειδικών εφέ (όπως εκρήξεις, ηλεκτρικά φαινόμενα, αιμορραγία), εξασφαλίζοντας ρεαλιστική απεικόνιση με αποδοτική χρήση πόρων. Παράλληλα, τα UI Widgets χρησιμοποιούνται για την απεικόνιση στοιχείων όπως οι πόντοι ζωής των NPCs, η κατάστασή τους στο Behavior Tree, Quick Time Events (QTE), καθώς και το γραφικό περιβάλλον του παίκτη (π.χ. minimap, inventory, οπλισμός).

2.1.4. Metahuman Creator και Metatailor App

Η δημιουργία του βασικού χαρακτήρα υλοποιήθηκε με χρήση του Metahuman Creator (Epic, n.d.) και της εφαρμογής [Polycam](#) για τη φωτογραμμετρική αποτύπωση προσώπου, διαμορφώνοντας ένα πλήρως παραμετροποιημένο ψηφιακό ομοίωμα. Η εφαρμογή [Metatailor](#) αξιοποιήθηκε για περαιτέρω προσαρμογή της ενδυμασίας και των αξεσουάρ. Οι NPCs δημιουργήθηκαν με χρήση ελαφρύτερων μοντέλων, ώστε να εξασφαλιστεί η αποδοτικότητα της μηχανής κατά το rendering πολλαπλών χαρακτήρων.

2.1.5. Lumen

Η τεχνολογία Lumen (Epic, n.d.) επιτρέπει την παραγωγή φωτισμού και σκιών σε πραγματικό χρόνο, αντικαθιστώντας το παραδοσιακό Light Baking. Το σύστημα αυτό προσαρμόζεται δυναμικά στις αλλαγές του περιβάλλοντος, προσδίδοντας ρεαλιστική ατμόσφαιρα. Λόγω των απαιτήσεων του project, η βελτιστοποίηση της απόδοσης πραγματοποιήθηκε με χρήση LODs και μείωση των μεγάλων textures, καθώς δεν έγινε χρήση του Nanite.

2.1.6. Landscape Editor, Landscape Auto Material και Foliage Instancing

Η διαμόρφωση του τρισδιάστατου περιβάλλοντος βασίστηκε στο Landscape Editor της Unreal Engine, αξιοποιώντας Heightmaps που δημιουργήθηκαν μέσω [Houdini](#) για τη γεωμετρική απεικόνιση τοπογραφικών χαρακτηριστικών. Το Landscape Auto Material επιτρέπει την δυναμική απεικόνιση επιφανειών βάσει γεωμετρικών και γεωγραφικών παραμέτρων. Η τεχνική Foliage Instancing εφαρμόστηκε για την αποδοτική απόδοση πολλαπλών αντικειμένων (όπως δέντρα και βράχοι), επιτυγχάνοντας οικονομία στο rendering χωρίς απώλεια ποιότητας.

2.2. Leonardo AI

Η πλατφόρμα [Leonardo AI](#) αξιοποιήθηκε για τη δημιουργία προσαρμοσμένων εικονιδίων του γραφικού περιβάλλοντος του παιχνιδιού, βασιζόμενη σε περιγραφές κειμένου για την παραγωγή εικόνων μέσω γενετικής τεχνητής νοημοσύνης. Τα παραγόμενα εικονίδια περιλαμβάνουν απεικονίσεις ενεργειών NPC, αντικειμένων (όπλα, σφαίρες) και πληροφοριών συστήματος (πόντοι ζωής), πλήρως εναρμονισμένα με την αισθητική του έργου.

2.3. Eleven Labs & PlayHT

Για την παραγωγή ηχητικών διαλόγων και ηχητικών εφέ χρησιμοποιήθηκαν οι πλατφόρμες [Eleven Labs](#) και [PlayHT](#). Η PlayHT αξιοποιήθηκε για τους NPCs χάρη στις δυνατότητες προσαρμογής συναισθηματικού ύφους (π.χ. άγχος, θυμός), ενώ η Eleven Labs χρησιμοποιήθηκε για τους διαλόγους του κεντρικού χαρακτήρα, με τη δυνατότητα εκπαίδευσης μοντέλου φωνής για πιστή προσομοίωση συγκεκριμένου φωνητικού χαρακτήρα.

2.4. Blender

Η εφαρμογή [Blender](#) χρησιμοποιήθηκε για τη μοντελοποίηση και τροποποίηση ορισμένων 3D assets, προσαρμόζοντάς τα στο ύφος του παιχνιδιού. Πραγματοποιήθηκε επίσης texture painting για τον καλλωπισμό χαρακτήρων και αντικειμένων, ενσωματώνοντας στοιχεία όπως τατουάζ, ενδυματολογικές επιλογές και χαρακτηριστικά προσώπου.

2.5. Ableton Live

Το [Ableton Live](#) χρησιμοποιήθηκε για την επεξεργασία ηχητικών διαλόγων και εφέ, με εφαρμογή βασικών τεχνικών όπως εξισορρόπηση (EQ), συμπίεση (compression) και χρονική προσαρμογή, ώστε να διασφαλιστεί η συνέπεια της έντασης και της ποιότητας τους, χωρίς να επιβαρύνουν ακουστικά τον χρήστη κατά τη διάρκεια του Gameplay. Η τελική μίξη πραγματοποιείται στην Unreal Engine μέσω Submixes, προκειμένου να επιτυγχάνεται ισορροπία έντασης και ακουστική συνέπεια στο gameplay.

2.6. Davinci Resolve

Η πλατφόρμα [DaVinci Resolve](#) αξιοποιήθηκε για την ενοποίηση ήχου και εικόνας στα εισαγωγικά και καταληκτικά κινηματογραφικά στιγμιότυπα του παιχνιδιού (cutscenes). Τα αρχεία παραγόμενα μέσω Unreal Sequencer, επεξεργάστηκαν στο DaVinci για την τελική εξαγωγή άρτιων βιντεοσκοπικών αποτελεσμάτων.

2.7. 3D Asset Marketplaces και Free Projects

Η ενσωμάτωση 3D μοντέλων και άλλων στοιχείων γραφικών πραγματοποιήθηκε μέσα από μια εκτεταμένη αξιοποίηση εξωτερικών ψηφιακών αγορών (3D Asset Marketplaces) και ελεύθερων έργων (Free Projects), με στόχο την επιτάχυνση της ανάπτυξης και τον εμπλουτισμό του οπτικού περιεχομένου του παιχνιδιού. Συγκεκριμένα, για τις ανάγκες της κατασκευής των NPCs, όπως η ποικιλομορφία στην εξωτερική τους εμφάνιση, ο οπλισμός τους και τα animations κίνησης, αλλά και για τη διαμόρφωση του περιβάλλοντος (δέντρα, κτίρια, φυσικά στοιχεία), αξιοποιήθηκε πληθώρα ψηφιακών πόρων. Οι βασικές πλατφόρμες που χρησιμοποιήθηκαν περιλαμβάνουν το [Fab](#) (πρώην Unreal Marketplace), το [Sketchfab](#), το [Artstation](#) και το [Poly Haven](#), οι οποίες παρείχαν υψηλής ποιότητας assets συμβατά με την Unreal Engine. Παράλληλα, ενσωματώθηκαν δωρεάν έργα της Epic Games, όπως το Game Animation Sample Project (G.A.S.P.) (Epic, 2024), το Advanced Locomotion System (A.L.S.) (Epic, 2025) και το Lyra Starter Project (Epic, 2025), τα οποία προσέφεραν έτοιμα συστήματα και animations που προσαρμόστηκαν κατάλληλα στις ανάγκες της κίνησης και της αλληλεπίδρασης τόσο του παίκτη όσο και των NPCs. Μέσω αυτής της προσέγγισης, επετεύχθη σημαντική εξοικονόμηση χρόνου και πόρων, χωρίς συμβιβασμό στην ποιότητα και τη συνοχή της οπτικής ταυτότητας του έργου.

3. Στάδια Υλοποίησης και Μεθοδολογία

Η ανάπτυξη του παιχνιδιού βασίστηκε σε μία μεθοδική προσέγγιση που αξιοποιεί σύγχρονα εργαλεία και τεχνικές, με στόχο τη δημιουργία ενός ρεαλιστικού, δυναμικού και εμπυθιστικού περιβάλλοντος. Παράλληλα, δόθηκε έμφαση στην οργανωμένη και ευέλικτη ανάπτυξη, με τη σταδιακή υλοποίηση επιμέρους συστημάτων που συνεργάζονται μεταξύ τους για την υποστήριξη του πυρήνα της εμπειρίας του παίκτη.

Κεντρικός άξονας του project αποτέλεσε η υλοποίηση Behavior Trees, με σκοπό την ανάπτυξη πολύπλοκων και διαφοροποιημένων καταστάσεων συμπεριφοράς για τους NPCs, οι οποίοι παρουσιάζουν ποικιλία όχι μόνο στη φυσική τους εμφάνιση, αλλά και στις λειτουργίες και τη συμπεριφορά τους. Συγκεκριμένα, αναπτύχθηκε μια βασική κλάση για την κοινή λογική των NPCs, η οποία επεκτάθηκε σε υποκλάσεις, αναθέτοντας ιδιαίτερα χαρακτηριστικά και συμπεριφορές. Επιπλέον, αναπτύχθηκαν μηχανισμοί κινησιολογίας και ενσωματώθηκαν μηχανισμοί οπλισμού και επιθέσεων, οι οποίοι προσαρμόζονται στις ιδιαιτερότητες του κάθε NPC.

Ωστόσο, το έργο επεκτείνεται πέρα από τα συστήματα συμπεριφοράς, ενσωματώνοντας μηχανισμούς που ενισχύουν τον ρεαλισμό της αλληλεπίδρασης και την εμπύθιση του παίκτη. Στο πλαίσιο αυτό, αναπτύχθηκε ένα επεκτάσιμο σύστημα διαχείρισης των πόντων ζωής, το οποίο συνδυάζεται με συστήματα gameplay για τον παίκτη και τους NPCs. Ο παίκτης έχει τη δυνατότητα να χρησιμοποιεί όπλα, να αλληλεπιδρά με ειδικά αντικείμενα και να εκτελεί ειδικές κινήσεις επίθεσης και αποφυγής, οι οποίες αλληλεπιδρούν άμεσα με το σύστημα συμπεριφοράς του παιχνιδιού.

Συμπληρωματικά, ενσωματώθηκαν οπτικοακουστικά εφέ που συνοδεύουν τις ενέργειες τόσο των NPCs όσο και του παίκτη, ενισχύοντας την οπτική και ακουστική διάσταση της εμπειρίας. Επιπλέον, σχεδιάστηκε και υλοποιήθηκε ένα επεκτάσιμο σύστημα αποστολών (quest system), το οποίο παρουσιάζει τις ενεργές αποστολές του παίκτη και υποστηρίζεται από ένα ολοκληρωμένο σύστημα Heads-Up Display (HUD). Το HUD προσφέρει κρίσιμες πληροφορίες, όπως την απεικόνιση των πόντων ζωής, την κατάσταση του οπλισμού και περιλαμβάνει χάρτη πλοήγησης (minimap), διευκολύνοντας την κατανόηση του περιβάλλοντος από τον παίκτη.

Ακόμη, κατασκευάστηκε ένα ενδεικτικό επίπεδο παιχνιδιού (Level), το οποίο εμπλουτίστηκε με αντικείμενα και σχεδιάστηκε με στόχο την ανάδειξη των συμπεριφορών των NPCs, των βασικών μηχανικών του gameplay και των αλληλεπιδράσεων μεταξύ του παίκτη και των συστημάτων τεχνητής νοημοσύνης.

Τέλος, για την ενίσχυση της αφηγηματικής δομής του παιχνιδιού, δημιουργήθηκαν δύο ενδεικτικά cinematic cutscenes, τα οποία καλύπτουν τις εισαγωγικές και καταληκτικές σκηνές, συμβάλλοντας στη σκηνοθετική συνοχή του έργου.

Η παρούσα διπλωματική εργασία επικεντρώνεται πρωτίστως στην ανάλυση των υλοποιήσεων που σχετίζονται άμεσα με τα δέντρα συμπεριφοράς, τα οποία αποτέλεσαν το βασικό εργαλείο μοντελοποίησης της Τεχνητής Νοημοσύνης στο πλαίσιο του έργου. Το παρόν κεφάλαιο παρουσιάζει τη μεθοδολογική προσέγγιση της συνολικής υλοποίησης, παραθέτοντας με χρονολογική σειρά τα στάδια ανάπτυξης, με συνοπτική αναφορά στο περιεχόμενο και στους επιμέρους στόχους κάθε φάσης.

3.1. Δημιουργία Χαρακτήρων

Η υλοποίηση ξεκίνησε με τη δημιουργία των χαρακτήρων του παιχνιδιού, ξεκινώντας από τον παίκτη και τους διάφορους NPCs. Ο χαρακτήρας του παίκτη δημιουργήθηκε με τη βοήθεια της πλατφόρμας Polycam, η οποία επέτρεψε την ρεαλιστική αποτύπωση του προσώπου σε 3D mesh μέσω φωτογραμμετρίας. Το αποτέλεσμα μεταφέρθηκε στο Metahuman Creator, όπου δημιουργήθηκε ένας ρεαλιστικός χαρακτήρας υψηλής ποιότητας, έτοιμος για χρήση στο παιχνίδι. Η διαδικασία αυτή ενσωματώνει τεχνικές texture painting στο Blender για την προσθήκη δερματικών λεπτομερειών και τατουάζ, καθώς και την εφαρμογή ρουχισμού και αξεσουάρ μέσω της εφαρμογής Metatailor.

Για τους NPCs, η χρήση των Metahumans απορρίφθηκε λόγω του υψηλού υπολογιστικού κόστους που συνεπάγεται η χρήση τους σε περιβάλλον με πολλούς NPCs. Αντ' αυτού, χρησιμοποιήθηκαν τα ενσωματωμένα μοντέλα που παρέχει η Unreal Engine καθώς και αντικείμενα από δωρεάν έργα και το Fab Marketplace, προκειμένου να δημιουργηθεί ποικιλία στην εμφάνιση των NPCs.

3.2. Ανάπτυξη συστήματος διαχείρισης πόντων ζωής

Η επόμενη φάση περιλάμβανε την ανάπτυξη ενός συστήματος διαχείρισης πόντων ζωής (Damage System) για τους χαρακτήρες του παιχνιδιού. Αυτό το σύστημα επιτρέπει την αύξηση ή μείωση της ζωής κάθε χαρακτήρα που συμμετέχει σε επιθετικές ενέργειες και περιλαμβάνει λειτουργίες αντίδρασης για κάθε σενάριο. Για παράδειγμα, όταν ένας χαρακτήρας δέχεται damage, αλλά δεν εξουδετερώνεται, ενεργοποιούνται οι κατάλληλες αντιδράσεις όπως ηχητικά εφέ και animations. Το σύστημα αυτό είναι επεκτάσιμο και επιτρέπει την ενσωμάτωσή του σε κάθε χαρακτήρα και αντικείμενο, συνδυάζοντας τις λειτουργίες του με τα Behavior Trees και τη ζωτική κατάσταση των NPCs και του παίκτη.

3.3. Συστήματα κινησιολογίας

Η ανάπτυξη των συστημάτων κινησιολογίας αποτέλεσε το επόμενο στάδιο της υλοποίησης, με την εφαρμογή του Motion Matching για τον παίκτη, το οποίο παρέχει εκτενή συλλογή κινήσεων υψηλής ποιότητας μέσω του G.A.S.P. free project της Epic Games. Το σύστημα αυτό επιτρέπει την επιλογή του κατάλληλου animation σύμφωνα με παραμέτρους όπως η ταχύτητα και η κατάσταση του παίκτη. Επιπλέον, υλοποιήθηκαν επιπλέον κινήσεις όπως τρέξιμο, άλμα και παρκούρ, καθώς και κολύμβηση και αποφυγή επιθέσεων, με στόχο την αύξηση του ρεαλισμού και της δυναμικότητας του gameplay.

Για τους NPCs, χρησιμοποιήθηκαν Blendspaces και State Machines, επιτρέποντας την εφαρμογή διαφορετικών κινήσεων για κάθε τύπο NPC, ανάλογα με τα χαρακτηριστικά τους (π.χ. φύλο ή είδος). Παρά την αρχική εφαρμογή του Motion Matching, η μέθοδος των Blendspaces αποδείχθηκε πιο αποτελεσματική λόγω του μεγάλου πλήθους κινήσεων και της διαφορετικότητας των σκελετών και των διαθέσιμων assets που χρησιμοποιούνται στο παιχνίδι.

3.4. Δημιουργία οπλισμού και αντικειμένων

Με την ολοκλήρωση των βασικών μηχανισμών κίνησης για τον παίκτη και τους NPCs, το επόμενο στάδιο εστίασε στην ανάπτυξη των συστημάτων οπλισμού και ειδικών αντικειμένων. Σύμφωνα με τα διαθέσιμα assets,

ενσωματώθηκαν ποικίλα μοντέλα όπλων και αντικειμένων με μοναδικά χαρακτηριστικά, συνοδευόμενα από κατάλληλα οπτικοακουστικά εφέ για την ενίσχυση του ρεαλισμού κατά τη χρήση τους. Παράλληλα, αναπτύχθηκαν οι αντίστοιχες λογικές λειτουργίας για κάθε τύπο όπλου ή ειδικού αντικειμένου.

Για τον παίκτη, σχεδιάστηκε ένα ολοκληρωμένο σύστημα διαχείρισης πυρομαχικών και γεμιστήρα, το οποίο παρακολουθεί σε πραγματικό χρόνο την ποσότητα των διαθέσιμων σφαιρών και τη μέγιστη χωρητικότητα κάθε όπλου. Περιλαμβάνονται επίσης λειτουργίες όπως η επαναγέμιση (reload), οι οποίες συνοδεύονται από κατάλληλα animations που αποτυπώνουν ρεαλιστικά τις ενέργειες αυτές. Αντίστοιχα, υλοποιήθηκαν ειδικά αντικείμενα (όπως ρίψεις εκρηκτικών ή βοηθητικών εργαλείων), με έμφαση στην οπτικοακουστική αναπαράσταση της χρήσης τους και την επίδρασή τους στο gameplay.

Το σύστημα οπλισμού του παίκτη πλακίστηκε με τα αντίστοιχα στοιχεία του γραφικού περιβάλλοντος χρήστη (UI), προσφέροντας πληροφορίες σε πραγματικό χρόνο σχετικά με τα ενεργά όπλα, τα διαθέσιμα πυρομαχικά και τις τρέχουσες ενέργειες. Για την υλοποίηση των ενεργειών αυτών, ανατέθηκαν συγκεκριμένα πλήκτρα χειρισμού, διασφαλίζοντας τη λειτουργικότητα και την αμεσότητα στην αλληλεπίδραση του παίκτη με το περιβάλλον.

Σε επόμενο στάδιο, δημιουργήθηκε ένα συμπληρωματικό σύστημα τοποθέτησης αντικειμένων στον κόσμο του παιχνιδιού. Το σύστημα αυτό επιτρέπει την τοποθέτηση πυρομαχικών, όπλων και ειδικών αντικειμένων σε προκαθορισμένες ή δυναμικές θέσεις, προσφέροντας τη δυνατότητα συλλογής αυτών από τον παίκτη. Στο πλαίσιο αυτό, ενσωματώθηκε επίσης ένα βασικό σύστημα διαχείρισης νομίσματος εντός παιχνιδιού (game currency), μέσω συλλεκτικών αντικειμένων που προσθέτουν αξία στον παίκτη και στο μέλλον θα μπορούν να αξιοποιηθούν για αγορές από συγκεκριμένους NPCs.

Η υλοποίηση του οπλισμού για τους NPCs ακολούθησε παρόμοια διαδικασία με εκείνη του παίκτη, αν και με απλουστευμένη προσέγγιση ως προς την εσωτερική λογική και τον έλεγχο των αποθεμάτων πυρομαχικών. Σε κάθε τύπο NPC αποδόθηκαν διαφορετικά μοντέλα όπλων με τα δικά τους χαρακτηριστικά και συνοδευτικά οπτικοακουστικά εφέ. Η ολοκλήρωση του σταδίου περιλάμβανε την απόδοση των κατάλληλων animations για την απεικόνιση των κινήσεων οπλισμού και αφοπλισμού, τόσο για τον παίκτη, όσο και για τους NPCs.

3.5. Behavior Trees

Ένα από τα βασικά στάδια της υλοποίησης αποτέλεσε η ανάπτυξη της τεχνητής νοημοσύνης των NPCs, μέσω της αξιοποίησης της τεχνικής των Behavior Trees. Η διαδικασία περιλάμβανε τη δημιουργία των απαραίτητων assets που διέπουν τη λειτουργία τους, καθώς και την ενσωμάτωση των προηγούμενων ανεπτυγμένων συστημάτων κινησιολογίας και οπλισμού. Σκοπός ήταν η διασφάλιση της ομαλής και συνεκτικής λειτουργίας του συστήματος συνολικά, καθώς και η ενίσχυση της δυναμικής αλληλεπίδρασης μεταξύ παικτών και NPCs.

Το σύστημα Τεχνητής Νοημοσύνης που αναπτύχθηκε, βασίζεται σε δύο κύριους τύπους NPCs, οι οποίοι παρότι μοιράζονται βασικές λειτουργίες, διαφοροποιούνται στρατηγικά όσον αφορά στις αντιδράσεις και στις συμπεριφορές τους. Η διάκριση αυτή ενισχύει την ποικιλομορφία και τον ρεαλισμό του παιχνιδιού, καθιστώντας τις αλληλεπιδράσεις περισσότερο φυσικές και αληθοφανείς, βελτιώνοντας τη συνολική εμπειρία του χρήστη.

3.5.1. Στρατηγικές Χρήσης Δέντρων Συμπεριφοράς

Η αξιοποίηση των Behavior Trees στο No Man's Land επικεντρώνεται στην ανάπτυξη σύνθετων αλλά ελέγξιμων μηχανισμών αντίδρασης για δύο κατηγορίες εχθρικών NPCs:

- Enforcers (Ranged): Ένοπλοι ανθρώπινοι χαρακτήρες με στρατηγική μάχης από απόσταση.
- Androids: Ρομποτικοί εχθροί με διαφορετικές τακτικές προσέγγισης και μοτίβα αντίδρασης.

Η συμπεριφορά των NPCs διαμορφώνεται βάσει περιβαλλοντικών ερεθισμάτων, όπως η παρουσία του παίκτη, οι ήχοι ή η κίνηση στο πεδίο δράσης. Κάθε NPC ακολουθεί μια προκαθορισμένη λογική κατάστασης, επιτρέποντας στο σύστημα να λαμβάνει αποφάσεις σε πραγματικό χρόνο βάσει των συνθηκών.

Οι βασικές συμπεριφορικές καταστάσεις περιλαμβάνουν:

- Περιπολία (Patrol): Ο NPC κινείται σε προκαθορισμένες διαδρομές, επιτηρώντας το περιβάλλον για ύποπτες ενδείξεις ή ανωμαλίες.
- Έρευνα (Investigate): Σε περίπτωση ανίχνευσης ήχου ή κίνησης, ο NPC αλλάζει κατάσταση και κινείται προς το ερέθισμα προκειμένου να διαπιστώσει την αιτία.
- Επίθεση (Attack): Εφόσον εντοπιστεί ο παίκτης, ο NPC εμπλέκεται άμεσα σε μάχη, αξιοποιώντας τον διαθέσιμο οπλισμό και τις τακτικές του.
- Άμυνα/Απόσυρση (Defend/Evade): Όταν οι πόνοι ζωής του NPC φτάσουν σε χαμηλό επίπεδο, επιλέγει την απομάκρυνση προς ασφαλές σημείο, την αναζήτηση κάλυψης ή/και την υιοθέτηση stealth τακτικών, έως ότου αποκατασταθεί η κατάστασή του ή λήξει η απειλή.

Το σύστημα έχει σχεδιαστεί ώστε να υποστηρίζει χειρισμούς από τον παίκτη που επηρεάζουν τη συμπεριφορά των NPCs. Για παράδειγμα, ο παίκτης μπορεί να πετάξει ένα αντικείμενο (π.χ. μπουκάλι) για να αποσπάσει την προσοχή ενός NPC και να τον παρασύρει σε διαφορετικό σημείο. Επιπλέον, σε περιπτώσεις που οι NPCs δεν έχουν αντιληφθεί την παρουσία του παίκτη, επιτρέπεται η πραγματοποίηση αιφνιδιαστικών επιθέσεων με άμεση εξουδετέρωση.

Η διαφοροποίηση ανάμεσα σε Enforcers και Androids προσδίδει στρατηγικό βάθος στο gameplay. Οι Enforcers υιοθετούν τακτικές μάχης από απόσταση με κάλυψη και χρήση πυρομαχικών, ενώ οι Androids τείνουν να επιτίθενται πιο επιθετικά και με μικρότερη ανασφάλεια, βασιζόμενοι περισσότερο σε πρότυπα εντοπισμού και αυτόνομη προσέγγιση.

3.5.2. Διαδικασία Υλοποίησης

Η λειτουργικότητα των NPCs υλοποιήθηκε μέσω της δημιουργίας και παραμετροποίησης εξειδικευμένων assets του Unreal Engine, τα οποία υποστηρίζουν τη χρήση των Behavior Trees και επιτρέπουν τη δυναμική και ρεαλιστική συμπεριφορά εντός του παιχνιδιού.

Σύμφωνα με την Epic (n.d.) ενδεικτικά assets της υλοποίησης αποτελούν:

- Blackboard: Πίνακας αποθήκευσης μεταβλητών και δεδομένων συμπεριφοράς, χρησιμοποιούμενος από τα Behavior Trees για τη λήψη αποφάσεων.
- AI Controller: Ο υπεύθυνος έλεγχος κάθε NPC, ο οποίος διαχειρίζεται την εκτέλεση των Behavior Trees.
- Decorators & Services: Επιπρόσθετες λογικές συνθήκες και επαναλαμβανόμενοι έλεγχοι που ενεργοποιούνται βάσει προκαθορισμένων παραμέτρων, επιτρέποντας πιο ευέλικτη και στοχευμένη συμπεριφορά.
- Custom Behavior Tree Tasks: Δημιουργήθηκαν προσαρμοσμένα tasks για την υλοποίηση ειδικών ενεργειών, όπως αλλαγή κατάστασης ή έναρξη animation, εξειδικεύοντας την αλληλεπίδραση του NPC με το περιβάλλον και τον παίκτη.

Για την προσθήκη αισθητηριακής αντίληψης, αξιοποιήθηκε το:

- AI Perception Component: Ενσωματώθηκε για την προσομοίωση ανθρώπινων αισθήσεων, όπως η όραση και η ακοή, επιτρέποντας στους NPCs να αντιλαμβάνονται την παρουσία του παίκτη ή ήχους από συγκεκριμένες ενέργειες (Epic, n.d.).

Επιπρόσθετα, ενσωματώθηκαν προηγμένες τεχνικές λήψης αποφάσεων για την κίνηση και το pathfinding (Epic, n.d.):

- EQS (Environment Query System): Χρησιμοποιήθηκε για την εκτίμηση βέλτιστων σημείων τοποθέτησης ή μετακίνησης, λαμβάνοντας υπόψη παραμέτρους όπως η απόσταση από τον παίκτη ή η ορατότητα.
- NavMesh (Navigation Mesh): Επιτρέπει τη ρεαλιστική πλοήγηση των NPCs στο περιβάλλον, ορίζοντας επιτρεπτές επιφάνειες και σημεία προσβασιμότητας.

Η επικοινωνία μεταξύ διαφορετικών στοιχείων και η επεξεργασία δεδομένων έγινε μέσω:

- Blueprint Interfaces: Επιτρέπουν τη διαχείριση της αλληλεπίδρασης μεταξύ αντικειμένων και την ανάκτηση ή τροποποίηση δεδομένων με επεκτάσιμο και modular τρόπο, διευκολύνοντας τη μελλοντική συντήρηση και επέκταση του κώδικα (Epic, n.d.).

Διαδικασία Υλοποίησης

Η ολοκλήρωση του συστήματος AI πραγματοποιήθηκε μέσα από τα παρακάτω στάδια:

- Σχεδίαση Ιεραρχίας Συμπεριφορών: Καθορίστηκε η δομή και ροή των κόμβων στο Behavior Tree, σύμφωνα με τις λειτουργίες και τις στρατηγικές κάθε τύπου NPC.
- Λογική & Παραμετροποίηση: Ενσωματώθηκε η αντίστοιχη λογική μέσω Blueprints για κάθε ενέργεια, με δυνατότητα ρύθμισης παραμέτρων όπως ταχύτητα αντίδρασης, απόσταση ανίχνευσης κ.λπ.
- Υποστήριξη με Animation & Εφέ: Αναπτύχθηκαν παράλληλα animations και συνοδευτικά οπτικοακουστικά εφέ για την υποστήριξη κάθε κατάστασης (όπως επίθεση, αναζήτηση ή αποφυγή).
- Δοκιμές & Βελτιστοποίηση: Πραγματοποιήθηκαν επαναλαμβανόμενοι έλεγχοι απόδοσης και ρεαλισμού, με στόχο τη συνεχή βελτίωση της εμπειρίας και τη σταθερότητα του συστήματος συμπεριφοράς.

3.5.3. Αξιολόγηση απόδοσης

Η αξιολόγηση και βελτιστοποίηση της απόδοσης των Behavior Trees αποτέλεσε καθοριστικό στάδιο για την εξασφάλιση της ρεαλιστικής και ομαλής συμπεριφοράς των NPCs στο περιβάλλον του παιχνιδιού. Η διαδικασία αυτή επικεντρώθηκε σε τεχνικές και στρατηγικές που στοχεύουν στη μείωση της υπολογιστικής επιβάρυνσης και τη βελτίωση της απόκρισης των συστημάτων. Συγκεκριμένα, εφαρμόστηκαν τα εξής:

- Απλοποίηση Δομής και Μείωση Πολυπλοκότητας: Τα δέντρα συμπεριφοράς σχεδιάστηκαν με γνώμονα την απλότητα, αποφεύγοντας την υπερβολική χρήση κόμβων και πολύπλοκων λογικών διακλαδώσεων. Η πρακτική αυτή συνέβαλε άμεσα στη μείωση του χρόνου εκτέλεσης και στη σταθερότητα του συστήματος.
- Επαναχρησιμοποίηση Λογικής μέσω Subtrees: Οι επαναλαμβανόμενες συμπεριφορές οργανώθηκαν σε υποδέντρα (subtrees), τα οποία επαναχρησιμοποιούνται σε πολλαπλούς NPCs. Αυτό απέτρεψε τη δημιουργία πλεοναζόντων κόμβων και ενίσχυσε την επεκτασιμότητα και τη δυνατότητα συντήρησης του συστήματος.
- Αποδοτικές Στρατηγικές Ελέγχου και Συνθηκών: Οι κόμβοι των Behavior Trees ρυθμίστηκαν έτσι ώστε να αξιολογούνται μόνο όταν απαιτείται, δηλαδή σε περιπτώσεις μεταβολής της κατάστασης του NPC ή του περιβάλλοντος. Με αυτόν τον τρόπο περιορίστηκε ο αριθμός περιττών ελέγχων, βελτιώνοντας σημαντικά την απόδοση.
- Χρήση Εργαλείων Profiling και Ανάλυσης Απόδοσης: Εφαρμόστηκαν εξειδικευμένα εργαλεία profiling του Unreal Engine για τον εντοπισμό πιθανών σημείων συμφόρησης εντός των Behavior Trees. Τα δεδομένα αυτά αξιοποιήθηκαν για τη στοχευμένη αναδιάρθρωση και βελτιστοποίηση επιμέρους κόμβων.
- Δοκιμές και Ανατροφοδότηση από Χρήστες: Πραγματοποιήθηκαν επαναλαμβανόμενες δοκιμές σε συνθήκες gameplay με συμμετοχή πραγματικών χρηστών. Μέσω της συλλογής ποιοτικής ανατροφοδότησης,

εντοπίστηκαν σημεία που απαιτούσαν περαιτέρω βελτίωση και έγιναν οι κατάλληλες τροποποιήσεις για την ενίσχυση της εμπειρίας του παίκτη.

Η εφαρμογή των παραπάνω πρακτικών οδήγησε σε σημαντική αύξηση της αποδοτικότητας των NPCs, με παράλληλη διατήρηση του ρεαλισμού και της ποικιλίας των συμπεριφορών, επιτυγχάνοντας ένα δυναμικό και επεκτάσιμο σύστημα τεχνητής νοημοσύνης εντός του παιχνιδιού.

3.6. Επέκταση Μηχανισμών Gameplay

Μετά την ολοκλήρωση των βασικών συστημάτων τεχνητής νοημοσύνης και των δέντρων συμπεριφοράς, το επόμενο στάδιο υλοποίησης επικεντρώθηκε στην ανάπτυξη πρόσθετων μηχανισμών που ενίσχυσαν περαιτέρω την εμπειρία του παιχνιδιού, τόσο για τους NPCs, όσο και για τον παίκτη. Οι νέες λειτουργίες εμπλούτισαν τη στρατηγική διάδραση και την ποικιλομορφία του gameplay, προσφέροντας στον παίκτη μεγαλύτερο έλεγχο, ρεαλισμό και εμβάθυνση στη ροή του παιχνιδιού.

Κύριοι Μηχανισμοί:

- *Stealth System*: Υλοποιήθηκε ένα δυναμικό σύστημα stealth, στο οποίο ο παίκτης μπορεί να αποφεύγει την ανίχνευση από τους εχθρούς αξιοποιώντας στοιχεία του περιβάλλοντος, όπως θάμνους. Επιπλέον, προστέθηκε ειδική stealth επίθεση, που επιτρέπει στον παίκτη να εξουδετερώνει ακαριαία έναν NPC, εφόσον αυτός δεν έχει αντιληφθεί την παρουσία του, παρέχοντας μια στρατηγική εναλλακτική στην απευθείας σύγκρουση.
- *Quick Time Events (QTEs)*: Ενσωματώθηκε ένα σύστημα Quick Time Events, το οποίο ενεργοποιείται σε κρίσιμες στιγμές όπου οι NPCs πραγματοποιούν συγκεκριμένες επιθέσεις. Ο παίκτης καλείται να πατήσει συγκεκριμένα κουμπιά μέσα σε περιορισμένο χρονικό διάστημα, προκειμένου να αποφύγει ή να εξουδετερώσει την απειλή. Το σύστημα αυτό ενισχύει τη δραματικότητα και την αλληλεπίδραση κατά τη διάρκεια μάχης, απαιτώντας άμεση απόκριση και συγκέντρωση.
- *Συμπληρωματικά Συστήματα Οπτικοακουστικής Αλληλεπίδρασης*: Παράλληλα με τις μηχανικές gameplay, αναπτύχθηκαν και οπτικοακουστικά εφέ που ενισχύουν τον ρεαλισμό και τη συναισθηματική εμπλοκή. Ειδικά ηχητικά μηνύματα και διάλογοι ενεργοποιούνται δυναμικά ανάλογα με τη συμπεριφορά του NPC ή του παίκτη, όπως σε περιπτώσεις αναζήτησης εχθρού ή ξαφνικής επίθεσης. Οι ήχοι αυτοί δημιουργήθηκαν και παραμετροποιήθηκαν μέσω των πλατφορμών Eleven Labs και PlayHT, εξασφαλίζοντας υψηλή ποιότητα φωνής και προσαρμογή στις ανάγκες του έργου.
- *Σύστημα Αποστολών (Quest System)*: Αναπτύχθηκε ένα βασικό Quest System που παρέχει στον παίκτη αποστολές και στόχους σχετιζόμενους με τις προαναφερθείσες μηχανικές, δημιουργώντας κίνητρα για εξερεύνηση και πρόοδο, αλλά και ενισχύοντας τη συνοχή μεταξύ αφήγησης και gameplay.
- *Scanner Mechanic*: Προστέθηκε επίσης μια μοναδική ικανότητα για τον παίκτη, μέσω της χρήσης ενός αντικειμένου Scanner. Το αντικείμενο αυτό είναι πάντοτε διαθέσιμο και επιτρέπει στον παίκτη να εντοπίζει προσωρινά τη θέση και την κατεύθυνση των κοντινών NPCs, παρέχοντας τακτικό πλεονέκτημα στην προσέγγισή του.

Οι παραπάνω μηχανισμοί συνέβαλαν καθοριστικά στην ενίσχυση της ποικιλομορφίας των επιλογών του παίκτη και στη βελτίωση της συνολικής εμπειρίας, επιτρέποντας διαφορετικά στιλ παιχνιδιού, από πλήρη σύγκρουση μέχρι στρατηγική προσέγγιση και stealth.

3.7. Δημιουργία περιβάλλοντος

Επόμενο στάδιο ανάπτυξης αποτέλεσε η δημιουργία του ενδεικτικού περιβάλλοντος του παιχνιδιού, το οποίο λειτουργεί τόσο ως λειτουργικό πλαίσιο δοκιμών, όσο και ως παρουσίαση των δυνατοτήτων του έργου. Το περιβάλλον σχεδιάστηκε με στόχο να αναδειχθούν οι συμπεριφορές των NPC μέσω των Behavior Trees, οι μηχανικές gameplay, αλλά και το καλλιτεχνικό ύφος του τίτλου.

Για την ανάπτυξη του περιβάλλοντος χρησιμοποιήθηκαν εργαλεία της μηχανής Unreal Engine και πιο συγκεκριμένα το εργαλείο Landscape Editor. Μέσω αυτού, πραγματοποιήθηκε:

- *Διαμόρφωση Εδάφους*: Χρησιμοποιήθηκε ρεαλιστικό heightmap που δημιουργήθηκε με το Houdini Engine, προκειμένου να ενσωματωθούν φυσικά χαρακτηριστικά όπως βουνά, πεδιάδες και ποτάμια. Το αποτέλεσμα ήταν ένα ανάγλυφο εδάφους που αντανακλά φυσικά τοπία με βάση παραμέτρους όπως κλίση, υψόμετρο και γεωγραφική θέση.
- *Εφαρμογή Landscape Auto Material*: Για την ρεαλιστική απόδοση της φυσιολογίας του περιβάλλοντος (όπως γρασίδι, βραχώδεις επιφάνειες, χιόνια, γκρεμοί κ.ά.), εφαρμόστηκε το MW Landscape Auto Material, ένα asset το οποίο ήταν διαθέσιμο δωρεάν την περίοδο ανάπτυξης του παιχνιδιού. Το υλικό αυτό επιτρέπει δυναμική αλλαγή των υφών του εδάφους βάσει υψομέτρου και γεωμετρικών χαρακτηριστικών, προσδίδοντας ρεαλισμό και συνοχή στο οπτικό αποτέλεσμα.
- *Foliage Instancing*: Για την τοποθέτηση μεγάλου αριθμού 3D αντικειμένων, όπως πέτρες, δέντρα και τοίχοι, χρησιμοποιήθηκε η μέθοδος Foliage Instancing, εξασφαλίζοντας την αποδοτική διαχείριση πόρων και την υψηλή απόδοση στο rendering. Με αυτήν τη μέθοδο, επαναλαμβανόμενα αντικείμενα αποδίδονται πιο αποτελεσματικά από τη μηχανή, χωρίς να επιβαρύνουν σημαντικά την απόδοση.
- *Ενσωμάτωση Assets*: Τα assets του περιβάλλοντος αποκτήθηκαν κυρίως από δωρεάν διαθέσιμα projects στο Unreal Engine Marketplace, εξασφαλίζοντας παράλληλα υψηλή ποιότητα και λειτουργικότητα. Κάθε asset επιλέχθηκε και τοποθετήθηκε με γνώμονα την εξυπηρέτηση των gameplay μηχανισμών που είχαν υλοποιηθεί, ενισχύοντας τόσο την αισθητική, όσο και τη λειτουργική πλοήγηση στον χώρο.

Η σχεδίαση του επιπέδου έγινε με τέτοιο τρόπο ώστε να παρουσιάζονται διαφορετικά σενάρια προσέγγισης από τον παίκτη, ανάλογα με τις επιλογές και το στυλ παιχνιδιού του. Η τοποθέτηση των αντικειμένων και η μορφολογία του περιβάλλοντος δημιουργούν ποικίλες στρατηγικές εναλλακτικές, όπως stealth διαδρομές, σημεία ενέδρας ή απευθείας σύγκρουση, υποστηρίζοντας και ενισχύοντας όλες τις μηχανικές gameplay που είχαν αναπτυχθεί στα προηγούμενα στάδια.

3.8. Κινηματογραφικές σκηνές και Ειδικά εφέ

Στο τελευταίο στάδιο της υλοποίησης του υλικού του project, αναπτύχθηκαν δύο ενδεικτικά κινηματογραφικά cutscenes τα οποία ενισχύουν την αφηγηματική δομή και τη σκηνοθετική συνοχή του παιχνιδιού. Οι σκηνές αυτές τοποθετούνται στην εισαγωγή και στην καταληκτική φάση του παιχνιδιού, προσφέροντας στον παίκτη ένα πιο εμπυθιστικό και καλλιτεχνικά ολοκληρωμένο πλαίσιο εμπειρίας.

Τεχνική Υλοποίηση:

- *Cinematic Sequencer*: Οι σκηνές δημιουργήθηκαν μέσω του εργαλείου Sequencer της Unreal Engine 5, το οποίο επέτρεψε τον συντονισμό κάμερας, animations, ηχητικών εφέ και διαλόγων, παρέχοντας πλήρη έλεγχο στη σκηνοθετική απόδοση.
- *Ειδικά Εφέ (VFX)*: Για την ενίσχυση της οπτικής εμπειρίας, χρησιμοποιήθηκε το Niagara System, μέσω του οποίου υλοποιήθηκαν ειδικά εφέ όπως καπνοί και σωματίδια, τα οποία ενίσχυσαν τη δραματικότητα και τη δυναμική των σκηνών.
- *Διάλογοι*: Οι φωνητικοί διάλογοι δημιουργήθηκαν μέσω των πλατφορμών Eleven Labs και PlayHT, επιτρέποντας την παραγωγή ρεαλιστικής φωνητικής αφήγησης, σύμφωνα με τις απαιτήσεις του σεναρίου και του ύφους του έργου.
- *Ηχητική Επεξεργασία*: Για τη μουσική επένδυση, τον συγχρονισμό ήχων και τη μίξη των διαλόγων, χρησιμοποιήθηκε το λογισμικό Ableton Live, ενώ η τελική επεξεργασία και ενοποίηση των σκηνών πραγματοποιήθηκε μέσω του Davinci Resolve, εξασφαλίζοντας υψηλή ποιότητα ήχου και εικόνας.

Περιβάλλοντα Cinematics:

- Το πρώτο cinematic πραγματοποιήθηκε στο βασικό Level που κατασκευάστηκε ειδικά για το παιχνίδι, εντάσσοντας φυσικά την αφήγηση μέσα στον ίδιο τον κόσμο του project.
- Το δεύτερο cinematic αναπτύχθηκε σε ένα έτοιμο Level που αποκτήθηκε από το Fab Marketplace, το οποίο απεικονίζει περιβάλλον σχετικό με την εξέλιξη της πλοκής και προσφέρει μια σκηνική ποικιλία, επεκτείνοντας τον αφηγηματικό κόσμο του παιχνιδιού.

Οι κινηματογραφικές αυτές σκηνές αποτελούν ένα κρίσιμο στοιχείο παρουσίασης του έργου, αναδεικνύοντας τόσο το τεχνικό υπόβαθρο της ανάπτυξης όσο και την αφηγηματική του δυναμική.

3.9. Δοκιμή και Βελτιστοποίηση

Το τελευταίο στάδιο της ανάπτυξης περιλάμβανε επαναληπτικές διαδικασίες δοκιμών και βελτιστοποίησης για τη διασφάλιση της ποιότητας του gameplay και της σταθερής απόδοσης του συστήματος. Στόχος των διαδικασιών αυτών ήταν η επίτευξη επαρκούς απόκρισης, σταθερών frame rates και η αποδοτική χρήση των υπολογιστικών πόρων κατά την εκτέλεση του παιχνιδιού.

Εργαλεία και Διαδικασίες:

- Χρησιμοποιήθηκε το εργαλείο Unreal Insights για την παρακολούθηση της απόδοσης του project, επιτρέποντας την αναγνώριση σημείων συμφόρησης, καθυστερήσεων και περιοχών που χρήζουν βελτιστοποίησης.
- Πραγματοποιήθηκαν δοκιμές εντός του περιβάλλοντος του Level, αξιολογώντας την συμπεριφορά των Actors, των NPCs, καθώς και των μηχανισμών gameplay υπό συνθήκες πραγματικής χρήσης.

Τεχνικές Βελτιστοποίησης:

- Για τη βελτιστοποίηση της απόδοσης του Level και των επιμέρους assets, εφαρμόστηκαν βασικές τεχνικές που σχετίζονται με τη διαχείριση της πολυπλοκότητας και της υπολογιστικής επιβάρυνσης:
- Δημιουργία και παραμετροποίηση LODs (Levels of Detail) για τους σκελετούς των Actors αλλά και για στατικά αντικείμενα, μειώνοντας την υπολογιστική απαίτηση για μοντέλα που βρίσκονται σε απόσταση από την κάμερα του παίκτη.
- Μείωση των texture sizes σε περιπτώσεις assets που διέθεταν 4K resolution, προσαρμόζοντάς τα σε μικρότερη ανάλυση σύμφωνα με τη χρήση τους στο εκάστοτε gameplay σενάριο, διατηρώντας την οπτική ποιότητα χωρίς περιττό κόστος.
- Ανάπτυξη script που διαγράφει δυναμικά τους Actors σε περιοχές που ο παίκτης δεν έχει πλέον πρόσβαση, απελευθερώνοντας μνήμη και πόρους του συστήματος.
- Παραγωγή mipmaps για τα textures, διασφαλίζοντας την ομαλή απόδοσή τους σε διαφορετικές αποστάσεις και ανάλυση προβολής, μειώνοντας την κατανάλωση VRAM.

Οι διαδικασίες αυτές, σε συνδυασμό με συνεχείς δοκιμές παιχνιδιού, συνέβαλαν στη συνολική βελτίωση της εμπειρίας του παίκτη, εξασφαλίζοντας ρεαλισμό, σταθερότητα και αποδοτικότητα στο τελικό αποτέλεσμα.

4. Game Design Document (G.D.D.)

Η σχεδίαση του Game Design Document (G.D.D.) αποτέλεσε το θεμέλιο της δημιουργίας του παιχνιδιού, καθορίζοντας τις βασικές παραμέτρους και δομές που θα οδηγούσαν στην ολοκληρωμένη υλοποίηση του, εάν το παιχνίδι αναπτυσσόταν στην πλήρη του μορφή. Το G.D.D. περιλαμβάνει τον προσδιορισμό των χαρακτηριστικών του παιχνιδιού, της πλοκής, των σημαντικών χαρακτήρων και του χάρτη, ενώ περιγράφει και τις βασικές μηχανικές λειτουργίες, τον εξοπλισμό του παίκτη, τις κατηγορίες των εχθρών, την ακολουθία του παιχνιδιού και τη διαχείριση της οικονομίας. Σκοπός του εγγράφου είναι να παρέχει έναν πλήρη οδηγό για τη συνολική κατασκευή του παιχνιδιού, εξασφαλίζοντας ότι κάθε στοιχείο ενσωματώνεται αρμονικά με τα υπόλοιπα.

Στο παρόν κεφάλαιο αναλύονται οι βασικές κατηγορίες του G.D.D., όπως η κατηγοριοποίηση του παιχνιδιού, η πλοκή της ιστορίας, οι βασικοί χαρακτήρες, η δομή του χάρτη και οι μηχανικές του παιχνιδιού, καθώς και η ακολουθία του gameplay και η διαχείριση των εχθρών.

4.1. Τίτλος

Όνομα: No Man's Land

Φράση "σύνθημα": Rebellion

Έκδοση: 1^η



Εικόνα 1. Εξώφυλλο παιχνιδιού

4.2. Εισαγωγή

Το No Man's Land είναι συναρπαστικό παιχνίδι δράσης και εξερεύνησης που θα σας μεταφέρει σε ένα post-apocalyptic μέλλον, όπου η σύγκρουση της τεχνητής νοημοσύνης με την ανθρώπινη φύση οδήγησε σε καταστροφικές διαμάχες. Η δημιουργία TN εξανέμισε τις υποσχέσεις ενός ουτοπικού κόσμου κατά την απόφαση να εναντιωθεί σε όλους αυτούς που δε βοηθούσαν με ενεργό τρόπο τη συνεχή εξέλιξή της. Ο διχασμός και τα αθέμιτα μέσα αυτής της επιβολής προκάλεσαν πολυετή πόλεμο μεταξύ των τυφλών υποστηρικτών και δημιουργημάτων της TN και των Marauders, εξεγερτικών δυνάμεων των ανθρώπων οι οποίοι αναζητούν τρόπους να ανακτήσουν την ελευθερία τους και να την καταστρέψουν.

Στον ρόλο της Nora, λάβετε μέρος στη μάχη ως ενεργό μέλος της εξέγερσης και αντιμετωπίστε επικίνδυνες δυνάμεις για την ανατροπή της δυστοπίας. Εξερευνήστε ένα ανοικτό κόσμο με ρεαλιστικά γραφικά που έχει πληγεί από τις αρνητικές συνέπειες της τεχνολογίας, ανακαλύπτοντας μυστικά χαμένων προσώπων που μπορούν να ανατρέψουν την ισορροπία των δυνάμεων του πολέμου.

Το No Man's Land προσφέρει στους παίκτες μία εμπειρία γεμάτη εξερεύνηση, αλληλεπίδραση με χαρακτήρες διαφορετικών βιωμάτων και δράση που απαιτεί αντανakλαστικά και συνεργασία συνδυάζοντας το αίσθημα της επιβίωσης, με την αίσθηση του μυστηρίου σε έναν κόσμο όπου η τεχνολογία και η φύση συγκρούονται συνεχώς.

4.3. Ανάλυση Παιχνιδιού

Περιγραφή παιχνιδιού	
Είδος	Δράσης/Ρόλου
Στοιχεία Παιχνιδιού	• Μάχη με όπλα κοντινής/μακρινής εμβέλειας • Κρυφή δράση-επιθέσεις (Stealth) • Πρόοδος χαρακτήρων με ειδικές κινήσεις επίθεσης • Συλλογή αντικειμένων
Περιεχόμενο παιχνιδιού	• Δράσης • Επιστημονικής φαντασίας • Δράμα
Θέμα- Στυλ	• Δυστοπικό • Cyberpunk • Post-apocalyptic
Ροή παιχνιδιού	Γραμμική αφήγηση ιστορίας με κύριες, δευτερεύουσες αποστολές και προκλήσεις
Παίκτης	Ένας παίκτης

<u>Αναφορές παιχνιδιού</u>	
Ταξονομία παιχνιδιού	Δράση/Επιστημονική Φαντασία/ Ρόλου
Εμβάθυνση του χρήστη	• Τακτική • Προοδευτικές δυνάμεις/skills • Αφήγηση
Πηγές	• Wasteland • Fallout • The Last Of Us • Horizon Zero Dawn • Cyberpunk 2077 • Dying Light • Shadow of Mordor
<u>Τεχνικό μέρος παιχνιδιού</u>	
Τεχνική μορφή	3D
Οπτική γωνία κάμερας	Τρίτου Προσώπου
Πλατφόρμες Ανάπτυξης	• Unreal Engine 5.4 • Unreal Engine 4.7 • Houdini Engine • ZBrush • Maya • Marvelous Designer • Substance 3D Painter • Daz 3D • Speech-Graphics
Συσκευές	• PC • Gen 9 (PS5, XBOX Series X S)
Στοχευμένο κοινό	• Achieves • Explorers • Killers
Ηλικιακή Κατάταξη	PEGI-16

4.4. Ιστορία Παιχνιδιού

Η ιστορία εξελίσσεται σε ένα όχι και τόσο μακρινό μέλλον, όπου η εξέλιξη της τεχνολογίας οδήγησε στην ανάπτυξη τεχνητής νοημοσύνης (AGI), η οποία μπορεί να κατανοήσει, να μάθει και να εφαρμόσει γνώσεις σε ένα ευρύ φάσμα εργασιών σε ανθρώπινο επίπεδο, αλλά και πέρα από αυτό.

Το μοντέλο δημιουργήθηκε από την εταιρεία Ascedance και εγκαταστάθηκε στη καρδιά της πόλης Eliezer, όπου οι τεράστιες μονάδες που το αποτελούν προστατεύονταν αυστηρά από τις μισθωτές δυνάμεις της εταιρείας, τους Enforcers. Οι ηγετικές δυνάμεις αποφάσισαν να αξιοποιήσουν στο έπακρο τα πλεονεκτήματα και τις δυνατότητες της TN, προγραμματίζοντας το λογισμικό με στόχο την επίτευξη τεχνολογικής μοναδικότητας (Singularity), με μοναδικό σκοπό να λειτουργεί και να πράττει για το μέγιστο καλό της ανθρωπότητας.

Παρότι η ανθρωπότητα βελτιώθηκε σε πολλούς τομείς, η συνεχής εξέλιξη του μοντέλου σε συνδυασμό με την ασάφεια στους καλοπροαίρετους στόχους του προγραμματισμού και την απουσία συναισθηματικής νοημοσύνης, οδήγησαν στην απόφαση να εναντιωθεί σε όλους αυτούς που δε βοηθούσαν με ενεργό τρόπο στη συνεχή εξέλιξή του με βίαιες πρακτικές. Με την τυφλή εμπιστοσύνη των ηγετών στις ενέργειές του, ο κόσμος γέμισε από βάσεις των Enforcers, ενισχυμένες πλέον από στρατιώτες AI Humanoids (Sentinels) και Drones (Tracers), επιβάλλοντας την κυριαρχία του μοντέλου και τιμωρώντας τους μη υποταγμένους. Η υποταγμένη κοινωνία, διχασμένη από το δυστοπικό καθεστώς, απάντησε με την εξέγερση των Marauders, ομάδων κοινών θαρραλέων ανθρώπων, που ενώθηκαν από κάθε σημείο του χάρτη στην προσπάθεια απελευθέρωσης του πληθυσμού μέσα από σαμποτάζ και καταστροφές βάσεων των Enforcers. Αποκτώντας δυνάμεις, πολύ γρήγορα στράφηκαν στις βάσεις των Enforcers με τελικό στόχο την ολική καταστροφή των κεντρικών εγκαταστάσεων του μοντέλου στο κέντρο της Eliezer.

Ο δεκαετής πόλεμος οδήγησε σε απώλειες άμαχου πληθυσμού, δημόσιες εκτελέσεις και καταστροφές υποδομών. Το τεχνολογικό πλεονέκτημα και η στρατιωτική υπεροχή της Ascedance οδήγησε στη συγκέντρωση των φυσικών πόρων στο κέντρο της Eliezer και τις πόλεις σε σήψη, με τους Marauders να χάνουν δυνάμεις προσπαθώντας πλέον να αντιμετωπίσουν τις καταστροφικές συνέπειες του πολέμου με τον υπόλοιπο πληθυσμό. Ο πληθυσμός, αποδεχόμενος την επικράτηση της Ascedance ως αμετάβλητη πραγματικότητα και την απόφαση του μοντέλου να τιμωρήσει όσους δε συνέβαλαν στην ανάπτυξή του, απέδωσε στο μοντέλο το όνομα Basilisk, από το πείραμα σκέψης του Roko ("Roko's Basilisk", 2022), με τη θεωρία πως πλέον είναι αυτόνομο και νοήμον να κυριαρχεί στην κοινωνία.

Ο παίκτης μας, Nora, αποτελεί μέλος των Marauders και ενεργός στρατιώτης του πολέμου από τη νεανική της ηλικία. Μεγάλωσε με τη μητέρα της, Eliza, η οποία την άφησε κατά τη διάρκεια του πολέμου με ελάχιστες αναμνήσεις. Η Eliza, μαζί με μια ομάδα επιστημόνων, διατηρούσε μυστικά εργαστήρια με σκοπό την κατασκευή μιας τεχνητής νοημοσύνης που θα μπορούσε να καταπολεμήσει τον Basilisk. Η Nora, σε μια αποστολή διείσδυσης σε ένα από τα εχθρικά camps των Enforcers, ανακαλύπτει ένα ιδιαίτερο μενταγιόν, το οποίο επιλέγει να φορέσει στον λαιμό της. Προχωρώντας στο camp, βρίσκεται στην είσοδο μιας μυστηριώδους πόρτας στην οποία φαινομενικά δεν υπήρχε πρόσβαση. Όταν πλησίασε, το μενταγιόν της απέκτησε μια απροσδόκητη λάμψη, δίνοντάς της τη δυνατότητα να εισέλθει στον χώρο, ανακαλύπτοντας ένα μυστικό εργαστήριο, που θα τη φέρει αντιμέτωπη με το παρελθόν της μητέρας της. Ο χαρακτήρας ξεκινά ένα ταξίδι πολέμου ενάντια στον Basilisk και τους υπόδουλούς του, ώστε να αποτρέψει την προδιαγεγραμμένη πορεία του μέλλοντος και να μάθει την αλήθεια για τη μυστηριώδη εμπλοκή της μητέρας της στις τεχνολογικές έρευνες.

4.5. Χαρακτήρες παιχνιδιού και σημαντικά αντικείμενα

4.5.1. Χαρακτήρες παιχνιδιού

	<u>Nora</u> (Παίκτης) Όνομα: Nora Ηλικία: 27 ετών Καταγωγή: North-Lake Outskirts Γονείς: Elisa Newcomb Alan Langford Ύψος: 171 cm Βάρος: 62 kg Επάγγελμα: Μηχανικός Χαρακτηριστικά: Μοβ μαλλιά, γαλάζια μάτια, έντονες γωνίες προσώπου, τατουάζ στο χέρι Προσωπικότητα: Δυναμική, πολυμήχανη, γνωρίζει πολεμικές τέχνες, εξειδικευμένες γνώσεις κατασκευής πολεμικού εξοπλισμού, περιπετειώδης, εθισμένη στην αδρεναλίνη Κατάσταση: Ενεργό μέλος των Marauders, στέλεχος επιχειρήσεων κατά των Enforcers
	<u>Elisa</u> (NPC) Όνομα: Eliza Vale Ηλικία: 52 ετών Καταγωγή: North-Lake Ύψος: 162 cm Βάρος: 50 kg Επάγγελμα: Επιστήμονας TN Χαρακτηριστικά: Χλωμό δέρμα, πράσινα μάτια, καστανά μαλλιά Προσωπικότητα: Δυναμική, εργασιομανής, ευγενική, σοφή Κατάσταση: Αγνοούμενη (10 έτη)



Basilisk (NPC)

Περιγραφή: Πρόγραμμα AGI από την εταιρία Ascendance.

Δημιούργησε Sentinels και Tracers για την προστασία του και την ενίσχυση των Enforcers

Κίνητρα: Επέκταση της κυριαρχίας.

Εξάλειψη όσων δεν συμβάλλουν στην εξέλιξή του.

Ερευνά την πιθανότητα αντιμετρών τεχνητής νοημοσύνης από τους Marauders.

Τοποθεσία: Κεντρικά της εταιρίας Ascendance (Eliezer)

Χαρακτηριστικά:

Πρόσβαση σε κάθε τεχνολογική υποδομή, έλλειψη συναισθηματικής νοημοσύνης

Θεωρίες του πληθυσμού αποδίδουν την απόκλιση σε δυσλειτουργία κώδικα ή σε εξέλιξη ASI.



Marauders (NPC)

Περιγραφή: Επαναστάτες με στόχο την καταστροφή του Basilisk

Μέλη: Απλός πληθυσμός και έμπειροι στρατιώτες

Κίνητρα: Ανατροπή της δυστοπίας, καταστροφή Ascendance & Basilisk

Τοποθεσία: Καταφύγια σε διάφορα σημεία του χάρτη

Χαρακτηριστικά: Κύριος εχθρός των Enforcers.

Χρησιμοποιούν απλούς τρόπους επικοινωνίας και επίθεσης για αποφυγή εντοπισμού από δυνάμεις του Basilisk



Enforcers (NPC)

Περιγραφή: Μισθωτοί στρατιώτες και άνθρωποι που εντάχθηκαν στις δυνάμεις του Basilisk λόγω αμοιβών, φόβου και ιδεολογίας. Αποτελούν την πλέον μοναδική δύναμη εξουσίας.

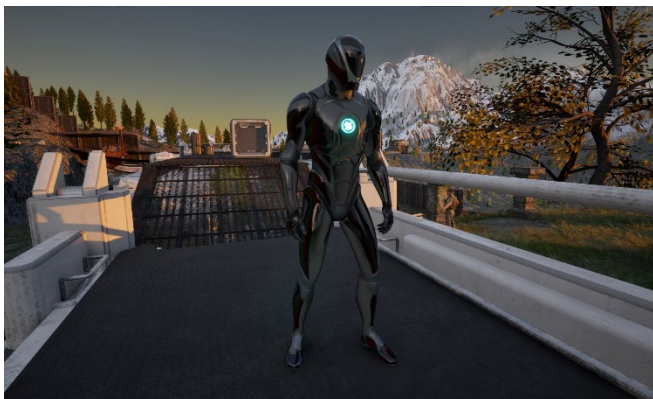
Κίνητρα: Προστασία του Basilisk

Αφανισμός των Marauders

Επιβολή της κυριαρχίας του Basilisk

Τοποθεσία: Eliezer/Camps/Free Roaming

Χαρακτηριστικά: Μαχητικές ικανότητες, οπλικός εξοπλισμός, κινούνται σε ομάδες



Sentinels (NPCs)

Περιγραφή: Humanoids με τεχνητή νοημοσύνη AGI

Κίνητρα: Προστασία του Basilisk

Αφανισμός των Marauders

Επιβολή της κυριαρχίας του Basilisk

Τοποθεσία: Eliezer/Camps/Free Roaming

Χαρακτηριστικά: Ανίχνευση κίνησης-προσώπου, προσθετικά μέλη τεχνολογίας, συστήματα όπλων, κινούνται σε ομάδες, δικτυωμένος συντονισμός και επικοινωνία με Tracers, καταστάσεις συναγερμού και πρωτόκολλα εμπλοκής, ανθεκτικοί σε υψηλές θερμοκρασίες, αστάθεια στην εκτέλεση εντολών



Tracers (NPCs)

Περιγραφή: Drones χαμηλού επιπέδου TN

Κίνητρα: Προστασία του Basilisk

Φρουροί των εχθρικών camps

Συλλογή πληροφοριών για τους Marauders

Τοποθεσία: Eliezer, camps, free roaming

Χαρακτηριστικά: Ικανότητες αιώρησης, εναέρια ανίχνευση κίνησης-προσώπου, καταστάσεις συναγερμού και πρωτόκολλα εμπλοκής, αισθητήρες εκρηκτικών και άλλων επικίνδυνων υλικών, δικτυωμένος συντονισμός και επικοινωνία με Enforcers, χειροκίνητη παράκαμψη, ανθεκτικά σε πολύ χαμηλές θερμοκρασίες.

Οπλισμός, Stealth Blend (σε προηγμένα μοντέλα)

Lali (NPC)

Όνομα: Lali

Ηλικία: 36 ετών

Καταγωγή: Άγνωστη

Γονείς: Άγνωστο

Ύψος: 186 cm

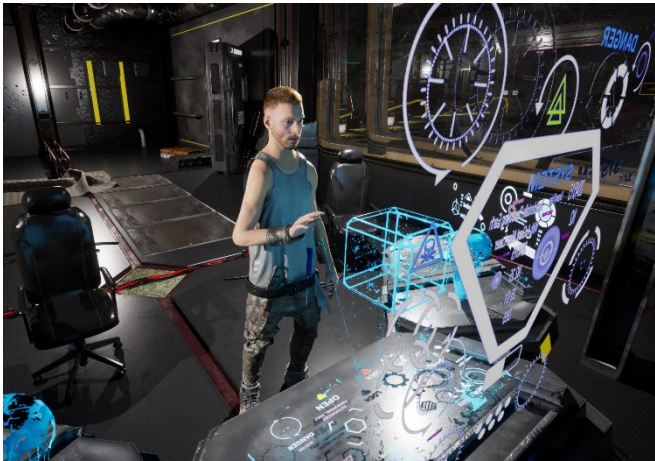
Βάρος: 68 kg

Επάγγελμα: Mechanic/Merchand

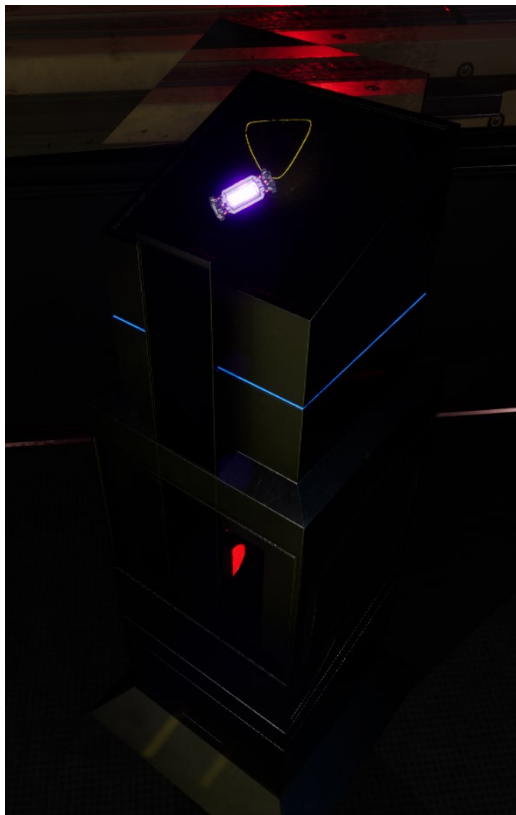
Χαρακτηριστικά: Έμπορος εξοπλισμού και αντικειμένων έναντι Shards. Μπορεί να βρεθεί στην πόλη North-Lake (αρχικά) και σε τυχαία σημεία του χάρτη (επόμενα επίπεδα).

Υπεύθυνος τεχνολογικού εξοπλισμού των Marauders.

Προσωπικότητα: Αντικοινωνικός, μυστηριώδης



4.5.2. Σημαντικά αντικείμενα



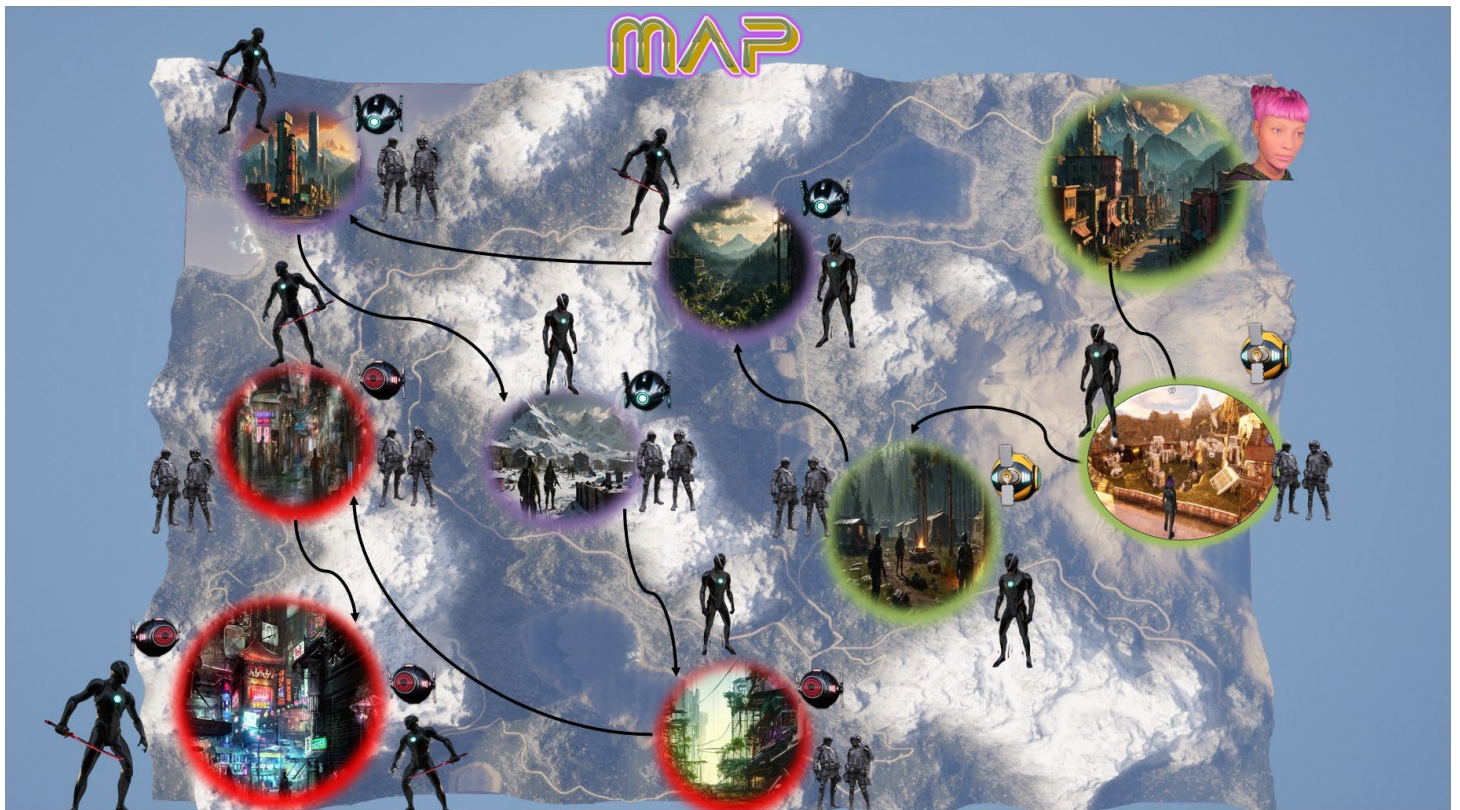
Περιγραφή: Μενταγιόν που έχει στο κέντρο του μία ροζ πέτρα, περιέχει μέρος του κώδικα της TN, που δημιούργησε η Eliza Newcomb

Προέλευση: Βρέθηκε από τη Nora σε ένα μυστικό εργαστήριο

Ιδιότητες: Το μενταγιόν εκπέμπει μία λάμψη όταν η Nora βρίσκεται στην είσοδο ενός ερευνητικού κέντρου, δίνοντάς της τη δυνατότητα να εισέλθει στον χώρο.

4.6. Χάρτης

Ο κόσμος εκτείνεται ανάμεσα σε λίμνες, πεδιάδες, ερήμους και δάση και κατοικείται από επαναστάτες, ρομπότ και ανθρώπους που βιώνουν τις θετικές και τις αρνητικές συνέπειες της τεχνολογίας. Από φυσικά τοπία, ερημωμένες πόλεις και εγκαταλελειμμένα χωριά σε εχθρικές βάσεις, μυστικά εργαστήρια και εξελιγμένες πόλεις, όπου το στίγμα του Basilisk είναι εμφανές.



Εικόνα 2. Χάρτης παιχνιδιού

A: North Lake

B: Eliezer

B1: Urban Slams

B2: Ascedance

C: Enforcers Camps

North Lake: Η πόλη καταγωγής της Nora. Οι κάτοικοί της είναι ένας απλός πληθυσμός που δεν παίρνει μέρος στον πόλεμο και είναι φιλικόι με τους Marauders, οι οποίοι ζουν ανάμεσά τους και αισθάνονται ασφάλεια σε αυτή. Εκεί κατέφυγε αρκετός πληθυσμός με σκοπό να αποστασιοποιηθεί από την ένταση του πολέμου. Τα σημάδια του χρόνιου πολέμου είναι εμφανή.



Εικόνα 3. North Lake

Eliezer: Η βασική πόλη του χάρτη στην οποία η τεχνολογία είναι πλήρως εξελιγμένη. Είναι η πόλη που αναπτύχθηκε ο Basilisk. Κατοικείται από επιστήμονες, καλλιτέχνες κ.α. και γενικότερα από τα ανώτερα κοινωνικά στρώματα, που απολαμβάνουν τις ανέσεις της τεχνολογίας υπό το καθεστώς του Basilisk. Φρουρείται έντονα από κάθε είδος Enforcer, με οποιονδήποτε να υπονομεύει ή να αντιτάσσεται ενάντια στο Basilisk, να δέχεται τις συνέπειες.



Εικόνα 4. Eliezer

Ascedance: Τα κεντρικά κτίρια στα οποία στεγάζεται ο Basilisk, προστατευμένος από εξελιγμένους Sentinels και Tracers. Η παρουσία ανθρώπινης ζωής είναι ελάχιστη από την αρχή του πολέμου.

Urban Slams: Οι γειτονιές που βρίσκονται στα προάστια της πόλης Eliezer. Περιέχουν κτίρια που εξυπηρετούν τις καθημερινές ανάγκες των ανθρώπων. Η φρούρηση από τους Enforcers είναι έντονη. Οι κάτοικοι εκεί ζουν υποταγμένοι και έχουν δεχτεί τα αποτελέσματα του πολέμου, ωστόσο έχουν μεικτές απόψεις σχετικά με τη διοίκηση του Basilisk. Αν και βρίσκονται πολύ κοντά στην πιο τεχνολογικά εξελιγμένη πόλη, στην πραγματικότητα η ζωή τους είναι δύσκολη. Δεν απολαμβάνουν τις ίδιες ανέσεις και δεν έχουν απορροφήσει τις τεχνολογικές εξελίξεις. Πολλοί από αυτούς ονειρεύονται να ζήσουν στην Eliezer και κάποιοι να ξεφύγουν.

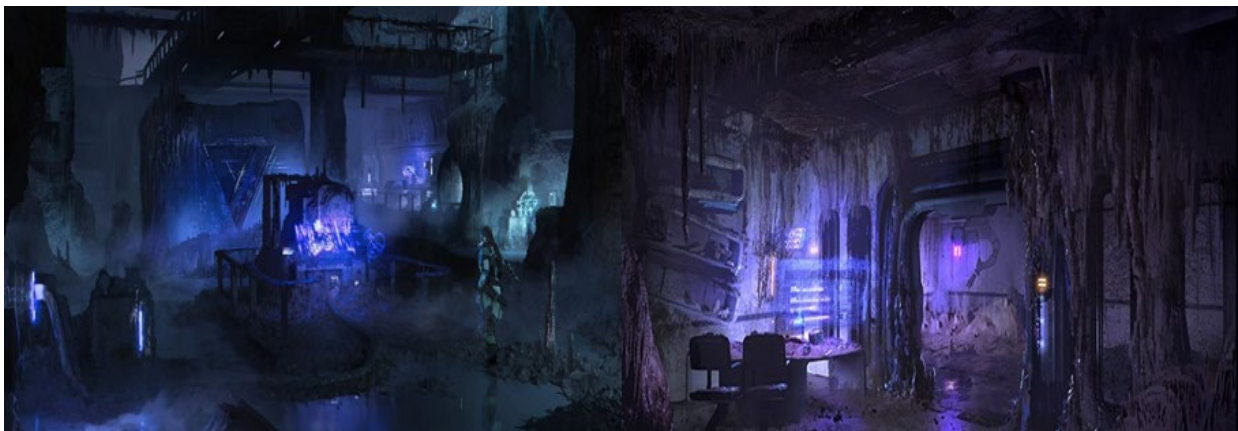


Εικόνα 5. Urban Slams

Enforcers Camps: Εγκαταστάσεις των στρατιωτικών δυνάμεων Enforcers, που έχει διασκορπίσει ο Basilisk. Χρησιμοποιούνται ως βάση, κατά τις εξορμήσεις τους, με σκοπό την επιβολή του καθεστώτος. Η πρόσβαση επιτρέπεται αυστηρά και μόνο στους Enforcers και περιπολούνται από Tracers. Σε κάθε camp βρίσκεται ένα ερευνητικό εργαστήριο, το οποίο περιέχει αντικείμενα εξοπλισμού και αντικείμενα που σχετίζονται με την εξέλιξη της ιστορίας.



Εικόνα 7. Enforcers Camp



Εικόνα 6. Ερευνητικό εργαστήριο

4.7. Μηχανικές-Κανόνες Μάχης

4.7.1. Παίκτης

Ο χαρακτήρας ξεκινά το παιχνίδι με 100 πόντους (HP) ζωής (Vitality), αλλά και αντοχής (Stamina) οι οποίοι καταναλώνονται κατά την εκτέλεση επιθετικών και αμυντικών κινήσεων. Ο παίκτης έχει τη δυνατότητα να ανακτήσει Vitality ή Stamina χρησιμοποιώντας κατάλληλα αντικείμενα.

Το σύστημα μάχης περιλαμβάνει κοντινές επιθέσεις με Melee όπλα, οι οποίες μπορεί να είναι γρήγορες ή αργές, οι αργές προσφέρουν 10% επιπλέον damage. Επιπλέον, ο παίκτης μπορεί να χρησιμοποιήσει οπλισμό για την εξουδετέρωση Enforcers, αλλά όχι Androids.

Μία ακόμη δυνατότητα είναι το Takedown, μια ειδική ενέργεια ακαριαίας εξουδετέρωσης που μπορεί να εκτελεστεί μόνο αν ο εχθρός δεν έχει αντιληφθεί την παρουσία του παίκτη.

Ορισμένες εχθρικές επιθέσεις μπορούν να αποκρουστούν μέσω αποφυγής (dodge), ώστε ο παίκτης να μη δεχτεί damage κατά τις οποίες εμφανίζεται ένα κουμπί αποφυγής στην οθόνη, το οποίο πρέπει να πατηθεί εντός χρονικού ορίου. Αν το Stamina μηδενιστεί, ο παίκτης δεν μπορεί να αμυνθεί.

Με 6 συνεχόμενες επιτυχίες σε επιθέσεις ή αποφυγές (combo), γεμίζει μία κάψουλα ενέργειας που επιτρέπει την εκτέλεση Takedown σε οποιονδήποτε εχθρό, ανεξαρτήτως αν έχει αντιληφθεί τον παίκτη ή όχι. Ο παίκτης μπορεί να αποθηκεύσει μέχρι δύο τέτοιες κάψουλες, με τη δυνατότητα μόνιμης αύξησης της χωρητικότητας κατά μία, μετά από κάθε ολοκληρωμένο επίπεδο.

Ο εξοπλισμός του παίκτη οργανώνεται σε τρεις άμεσες θέσεις (slots): Weapon, Equipment και Healing. Σε κάθε slot μπορεί να ανατεθεί ένα αντικείμενο για χρήση χωρίς άνοιγμα του menu.

Τέλος, τα όπλα που απαιτούν πυρομαχικά διαθέτουν γεμιστήρες με καθορισμένη χωρητικότητα. Όταν αδειάσουν, απαιτείται επαναγέμιση για να συνεχιστεί η χρήση τους.

4.7.2. Εχθροί

Σε κάθε επίπεδο, οι δυνατότητες, η δύναμη, αλλά και οι τρόποι αντιμετώπισης των εχθρών διαφέρουν. Σταδιακά, οι εχθροί αποκτούν μεγαλύτερη δύναμη, απαιτούν περισσότερα χτυπήματα και ειδικές κινήσεις. Ταυτόχρονα ο παίκτης ξεκλειδώνει επιπλέον ειδικά αντικείμενα, κινήσεις και δυνάμεις για την αντιμετώπιση των εχθρών όπως περισσότερο damage, ακινητοποίηση περισσότερων εχθρών, αποκάλυψη των αόρατων Tracers.

Οι **Tracers** στο πρώτο επίπεδο κάνουν περιπολίες στα camps, έχοντας τη δυνατότητα να ειδοποιήσουν κοντινούς Enforcers και Sentinels για την παρουσία εχθρών, αλλά όχι να επιτεθούν στον παίκτη. Στο επόμενο επίπεδο αποκτούν οπλισμό και στο τελευταίο είναι αόρατοι. Μπορούν να βρεθούν αποκλειστικά σε καταστάσεις περιπολίας εντός των camps.

Οι **Sentinels** είναι πολεμιστές κοντινής και γρήγορης ταχύτητας. Μία από τις μοναδικές κινήσεις τους είναι η εκτίναξη, με άλμα, προς το μέρος του παίκτη και η χρήση σπαθιού για τον τραυματισμό του. Οι επιθέσεις τους απαιτούν αποφυγή και δεν μπορούν να μπλοκαριστούν. Η αντιμετώπιση τους απαιτεί κυρίως ειδικές κινήσεις από τον παίκτη και χρήση ειδικού εξοπλισμού αδρανοποίησης. Επιπλέον, τα όπλα με σφαίρες δεν μπορούν να χρησιμοποιηθούν εναντίον τους. Στο επίπεδο 1 η παρουσία τους στα camps είναι μικρή, ενώ στα επόμενα επίπεδα αυξάνεται σταδιακά. Στο τελευταίο επίπεδο δεν μπορούν να εξολοθρευθούν με επιθέσεις stealth. Μπορούν να βρεθούν αποκλειστικά σε καταστάσεις περιπολίας εντός των camps.

Οι **Enforcers** στο επίπεδο 1 διαθέτουν ελάχιστο οπλισμό, είναι λίγοι σε πλήθος και προκαλούν μικρό αριθμό damage. Μπορούν να βρεθούν σε καταστάσεις περιπολίας ή αδρανείς εντός των camps. Στα επόμενα επίπεδα μπορούν να προκαλέσουν μεγαλύτερο damage στους παίκτες και η αντίληψη του χώρου και η αντοχή στις επιθέσεις stealth αυξάνονται. Στο τελευταίο επίπεδο, ορισμένοι διαθέτουν throwables, όπως grenades κτλ.

4.8. Ήττα

Στο No Man's Land, η ήττα του παίκτη επέρχεται όταν οι πόντοι ζωής (HP) του χαρακτήρα μειωθούν στο μηδέν. Στην περίπτωση που ο παίκτης διαθέτει κάποιο αντικείμενο αυτόματης αναβίωσης (π.χ. Adrenaline Injector), αυτό ενεργοποιείται αυτόματα και ο χαρακτήρας επανέρχεται στη ζωή με 35% HP και στο 50% της αντοχής του. Σε αντίθετη περίπτωση, ενεργοποιείται το σύστημα επανεκκίνησης από το τελευταίο Checkpoint και το παιχνίδι συνεχίζεται από την πιο πρόσφατη αποθηκευμένη κατάσταση. Ο μηχανισμός συνοδεύεται από οπτικά και ηχητικά εφέ, όπως ερυθρό φωτισμό οθόνης, επιβράδυνση της κάμερας και ήχο καρδιακού παλμού για την ενίσχυση της έντασης.

Για τους παίκτες που αναζητούν μια σκληρότερη πρόκληση, το παιχνίδι προσφέρει τη δυνατότητα επιλογής του Survivor Mode (Permadeath). Σε αυτό το mode:

- Δεν υπάρχουν checkpoints ούτε αντικείμενα αναβίωσης
- Αν ο χαρακτήρας πεθάνει, η πρόοδος χάνεται οριστικά και το παιχνίδι ξεκινά από την αρχή
- Το Survivor Mode επιλέγεται κατά την έναρξη του παιχνιδιού από το Settings Menu και δεν μπορεί να αλλάξει στη συνέχεια
- Η επιλογή αυτή συνοδεύεται από τροποποιήσεις στο UI (σκοτεινή αισθητική, περιορισμένοι πόροι, απουσία HUD βοηθημάτων)

Η ολοκλήρωση του παιχνιδιού σε Permadeath mode ανταμείβει τον παίκτη με αποκλειστικό περιεχόμενο, όπως εναλλακτικό τέλος, μυστικά cinematic cutscenes και ειδικά game lore αντικείμενα.

Αυτό το διπλό σύστημα ήττας επιτρέπει στον παίκτη να επιλέξει ανάμεσα σε πιο φιλικές ή hardcore εμπειρίες, ανάλογα με το στυλ παιχνιδιού και το επίπεδο πρόκλησης που επιθυμεί.

4.9. Εξοπλισμός/Αντικείμενα

Η Nora διαθέτει όπλα, αντικείμενα και κινήσεις, τα οποία αλληλοσυμπληρώνονται και απαιτούνται για την ολοκλήρωση των main quests. Κάθε είδος Enforcer διαθέτει διαφορετικό σύνολο όπλων, δυνάμεων και αδυναμιών και απαιτεί διαφορετικό τρόπο αντιμετώπισης. Ο παίκτης μπορεί να προσάψει αντικείμενα που ενισχύουν τον εξοπλισμό του και να καταναλώσει αντικείμενα που προσθέτουν πόντους ζωής και αντοχής.

4.9.1. Weapons

Nora

Ο χαρακτήρας της Nora επικεντρώνεται σε εξοπλισμό μάχης.
Έχει τη δύναμη και τη γνώση να χρησιμοποιήσει βαρύ οπλισμό.

Η Nora διαθέτει:

FN Scar-L 	Description: Ένα ισχυρό αυτόματο όπλο με υψηλό ρυθμό πυρός. Χρησιμοποιείται ενάντια σε Tracers και Enforcers. Damage: 15/bullet Special Move: - Rate of Fire: 645 RPM Capacity: 24 Bullets Location: Secret Lab Attachments: Red dot, Extended Mag, Muzzle Brake, Silencer Uses: 5.56 Ammo
---	---

Ballista 	Description: Ένα ισχυρό όπλο με σκόπευτρο που χρησιμοποιείται για να επιτεθεί σε στόχους από κρυφές θέσεις ή σε αποστάσεις που υπερβαίνουν τις δυνατότητες ανίχνευσης του στόχου. Damage: 150 Rate of Fire: 51 RPM Capacity: 5 Bullets Special Move:- Location: Secret Lab Attachments: Red dot, Extended Mag, Muzzle Brake, Silencer Uses: 7.62 mm Ammo
Harpoon 	Description: Τεχνολογικά τροποποιημένο όπλο μακρινής, αλλά και κοντινής εμβέλειας. Μπορεί να χρησιμοποιηθεί ενάντια σε Enforcers και Sentinels. Damage: 100 από stealth, 30 κοντά, 20 μακριά, 20 σε Tracers. Special Move: (Απαιτεί 1 διαθέσιμο Combo Bar): Διπλό ακαριαίο Kill των 2 κοντινότερων αντιπάλων Location: Starting weapon Attachments: no attachments Uses: Infinite
Jambya 	Description: Ένα ασιατικό μεσαιωνικό στιλέτο που έχει τροποποιηθεί τεχνολογικά ώστε να επιστρέφει στο χέρι του ιδιοκτήτη του. Damage: 100 από stealth, 30 κοντά, 20 μακριά, 20 σε drone. Special Move: (Απαιτεί 1 διαθέσιμο Combo Bar): Διπλό ακαριαίο Kill των 2 κοντινότερων αντιπάλων Location: Starting weapon Attachments: - Uses: infinite
DataWorm 	Description: Μια συσκευή προσωρινής αδρανοποίησης που περιλαμβάνει ένα ψηφιακό ιό με μορφή κάρτας δεδομένων (data chip). Η εκτόξευση του δημιουργεί ένα θόλο επίδρασης που αδρανοποιεί τους Sentinels και Tracers καθιστώντας τους ευάλωτους στην επίθεση Takedown. Στα επόμενα επίπεδα το πεδίο επίδρασης άγει ηλεκτρισμό επηρεάζοντας και τους Enforcers με 10 damage ανά δευτερόλεπτο που βρίσκονται εντός του. Damage: No direct damage Special Move: Αν πυροδοτηθεί κοντά σε τερματικά ανεβάζει προσωρινά έναν βρόχο σήματος στο δίκτυο προκαλώντας γεμινή απενεργοποίηση των Tracers. Location: Camps/Merchants Attachments: - Uses: Infinite

Jammer 	Description: Ηλεκτρονική συσκευή που απενεργοποιεί τους Tracers για 3 λεπτά και εμφανίζει στο minimap του παίκτη την τοποθεσία και την κατεύθυνση κάθε εχθρού. Μπορεί επίσης να χρησιμοποιηθεί για το άνοιγμα ηλεκτρικών εισόδων. Στα επόμενα επίπεδα μπορεί να κάνει εμφανή τα αόρατα Tracers. Damage: No direct damage. Special Move: - Location: Starting weapon Attachments: - Uses: Infinite (1 min cooldown)
Bottles 	Description: Μπουκάλια που χρησιμεύουν στη δημιουργία αντιπερισπασμού. Βρίσκονται διασκορπισμένα στο χάρτη και κατά τη ρίψη τους καταστρέφονται, προκαλώντας τον εχθρό να διερευνήσει το σημείο ηχητικής διατάραξης. Damage: No direct damage. Special Move: - Location: Open World/Camps Attachments: - Uses: Single
Ivy bomb 	Description: Μια βόμβα δηλητηρίου η οποία προκαλεί ζημιά σε όλους μέσα στο νέφος αερίου, αθόρυβα. Μπορεί να χρησιμοποιηθεί ενάντια σε Enforcers. Damage: 10 damage/sec εντός εμβέλειας. Special Move: - Location: Open World/Camps/Merchants Attachments:- Uses: Single
Boombox 	Description: Ένα boombox με τροποποιημένα ηχεία. Όταν ενεργοποιηθεί, αρχίζει να παράγει δυνατά παραμορφωμένα σήματα, προσελκύοντας τους εχθρούς να ερευνήσουν. Στα επόμενα επίπεδα εκρήγνυται μετά από 10 δευτερόλεπτα, προκαλώντας 30 damage μέσα στην ακτίνα του. Μπορεί να χρησιμοποιηθεί ενάντια σε Tracers και Sentinels Damage: 30 Special Move: - Location: Camps/Merchant Attachments: - Uses: Single

4.9.2. Healing

Τα αντικείμενα healing είναι αναλώσιμα όπου ο παίκτης μπορεί να χρησιμοποιήσει για να ανακτήσει Vitality ή Stamina. Μπορούν να βρεθούν στον ανοικτό κόσμο, στα camps και να αγοραστούν από Merchants, έναντι Shards.

Med Kits 	Το Med Kit είναι μια προηγμένη ιατρική συσκευή σχεδιασμένη για ταχεία και ολοκληρωμένη θεραπεία σε καταστάσεις μάχης. Χρησιμοποιώντας νανοτεχνολογία, αποκαθιστά την υγεία του χρήστη πλήρως, μέσα σε μόλις 10 δευτερόλεπτα. Για να εξασφαλίσει τη μέγιστη αποτελεσματικότητα, το Med Kit απαιτεί από τον χρήστη να παραμείνει ακίνητος κατά τη διάρκεια της εφαρμογής του. Αυτό το απαραίτητο εργαλείο μπορεί να αλλάξει την πορεία της μάχης, παρέχοντας μια κρίσιμη σανίδα σωτηρίας στις πιο έντονες συναντήσεις.
Adrenaline Shots 	Οι Adrenaline Shots είναι ισχυρές ενέσεις διεγερτικών, σχεδιασμένες για σενάρια μάχης υψηλής πίεσης. Μόλις μια Adrenaline Shot χορηγηθεί, αναγεννά γρήγορα 10 μονάδες υγείας ανά δευτερόλεπτο, επιτρέποντας στον χρήστη να παραμείνει στη μάχη, ενώ κινείται. Εξασφαλίζει συνεχή ανάρρωση, καθιστώντας την ανεκτίμητη για τη διατήρηση της ορμής και την επιβίωση σε παρατεταμένες εμπλοκές.
Bandages 	Τα Bandages είναι βασικά ιατρικά εργαλεία σχεδιασμένα για την παροχή άμεσης και αποτελεσματικής φροντίδας τραυμάτων σε καταστάσεις μάχης. Με την απλή, αλλά αποδοτική τους χρήση σταματούν την αιμορραγία και προλαμβάνουν λοιμώξεις, προσδίδοντας 25 πόντους ζωής, επιτρέποντας στον χρήστη να συνεχίσει τη μάχη, κατά τη χρήση τους. Αν και δεν αποκαθιστούν πλήρως την υγεία, όπως το Med Kit, τα Bandages παρέχουν ζωτική πρώτη βοήθεια που μπορεί να κάνει τη διαφορά μεταξύ ζωής και θανάτου. Η εύκολη εφαρμογή τους και η φορητότητά τους τους καθιστούν αναπόσπαστο κομμάτι κάθε εξοπλισμού επιβίωσης.
Energy boost 	Τα Energy Boosts είναι προηγμένα ενεργειακά συμπληρώματα σχεδιασμένα για να αποκαθιστούν την αντοχή του παίκτη. Με την άμεση κατανάλωσή τους, επαναφέρουν τα επίπεδα αντοχής του παίκτη στο μέγιστο, επιτρέποντάς του να συνεχίσει να μάχεται ή να τρέχει χωρίς διακοπή. Αυτά τα απαραίτητα βοηθήματα δίνουν στον παίκτη το προβάδισμα που χρειάζεται σε κρίσιμες στιγμές, εξασφαλίζοντας ότι δε θα εξαντληθεί κατά τη διάρκεια έντονης μάχης ή καταδιώξεων. Η γρήγορη δράση τους και η φορητότητά τους τα καθιστούν πολύτιμο εργαλείο για κάθε παίκτη.

4.9.3. Attachments

Τα Attachments είναι αντικείμενα που ενισχύουν την αποδοτικότητα και τη χρηστικότητα των όπλων FN Scar και Ballista. Μπορούν να βρεθούν μέσα στα Secret Labs. Τα αντικείμενα αυτά μπορούν να συλλεχθούν από τον παίκτη μια φορά και παραμένουν στο backpack του για το υπόλοιπο παιχνίδι. Κάθε αντικείμενο παραμένει ορατό για κάθε παίκτη μέχρι να συλλεχθεί.

Red dot 	Το Red dot είναι ένα αξεσουάρ ακριβείας που βελτιώνει την ακρίβεια στόχευσης για τα πυροβόλα όπλα. Προβάλλοντας μια καθαρή, κόκκινη κουκίδα στον στόχο, επιτρέπει γρήγορη απόκτηση στόχου και βελτιωμένη ακρίβεια πυροβολισμών. Αυτό το αξεσουάρ είναι ιδανικό για μάχες σε κοντινές έως μεσαίες αποστάσεις, προσφέροντας σημαντικό πλεονέκτημα στις συγκρούσεις επιτρέποντας γρηγορότερη και πιο ακριβή στόχευση.
Extended Mag 	Το Extended Mag είναι ένα σημαντικό αξεσουάρ όπλου που αυξάνει τη χωρητικότητα του γεμιστήρα από 25 σε 35 σφαίρες, προσφέροντας περισσότερη δύναμη πυρός σε παρατεταμένες συγκρούσεις. Εκτός από την αυξημένη χωρητικότητα πυρομαχικών, αυτό το αξεσουάρ βελτιώνει τον χρόνο επαναγέμισης μειώνοντάς τον κατά 2 δευτερόλεπτα, επιτρέποντας ταχύτερη επαναγέμιση υπό πίεση. Ιδανικό για διαρκείς μάχες, εξασφαλίζει στον παίκτη ότι μπορούν να παραμείνουν στη μάχη περισσότερο με λιγότερες διακοπές.
Muzzle Brake 	Το Muzzle Brake είναι ένα εξειδικευμένο αξεσουάρ σχεδιασμένο για να μειώνει την ανάκρουση του όπλου. Ανακατευθύνοντας τα αέρια που εκλύονται κατά τη διάρκεια της βολής, σταθεροποιεί το πυροβόλο, επιτρέποντας πιο ελεγχόμενους και ακριβείς πυροβολισμούς. Αυτό το αξεσουάρ είναι ιδιαίτερα χρήσιμο σε καταστάσεις παρατεταμένης πυροδότησης, βελτιώνοντας σημαντικά την ομαδοποίηση των βολών και τη συνολική διαχείριση του όπλου. Ιδανικό για διατήρηση της ακρίβειας κατά τη διάρκεια γρήγορης ή αυτόματης βολής.
Silencer 	Το silencer είναι ένα αξεσουάρ που μειώνει τον θόρυβο που παράγεται από τις βολές του όπλου του παίκτη. Μειώνοντας τον ήχο των πυροβολισμών, μειώνει την ακτίνα μέσα στην οποία οι κοντινοί εχθροί ειδοποιούνται για την παρουσία των παικτών. Αυτό το αξεσουάρ είναι απαραίτητο για επιχειρήσεις μυστικότητας, επιτρέποντας στους παίκτες να εξουδετερώσουν στόχους αθόρυβα και να αποφεύγουν την προσέλκυση της προσοχής των εχθρών.

4.9.4. Equipment

Οι παίκτες διαθέτουν διάφορα εξοπλιστικά στοιχεία, από την αρχή του παιχνιδιού, που ενισχύουν στατιστικά ζωής, ταχύτητας κινήσεων, απορρόφησης του εχθρικού damage και τον χώρο αποθήκευσης αντικειμένων. Οι δυνατότητες των αντικειμένων μπορούν να ενισχυθούν έως και τρεις φορές σε Merchants, έναντι Shards.

Armor 	Το Armor είναι απαραίτητος εξοπλισμός για κάθε παίκτη, παρέχοντας ζωτική προστασία μειώνοντας ένα μέρος της ζημιάς που λαμβάνεται από τις εχθρικές επιθέσεις. Κάθε παίκτης ξεκινά με αυτό το αντικείμενο εξοπλισμένο, εξασφαλίζοντας ένα βασικό επίπεδο άμυνας από την αρχή. Κάθε αναβάθμιση μειώνει περαιτέρω τη ζημιά που λαμβάνεται, ενισχύοντας σημαντικά την δυνατότητα επιβίωσης του παίκτη στη μάχη. Η αναβάθμισή του είναι κρίσιμη για την αντιμετώπιση δυσκολότερων μαχών και την αύξηση της συνολικής ανθεκτικότητας.
Tactical Gear 	Το Tactical Gear, που συνδυάζει εξειδικευμένες μπότες και γάντια, έχει σχεδιαστεί για να βελτιώνει τη συνολική ταχύτητα και την αποτελεσματικότητα του παίκτη στη μάχη. Αυτός ο απαραίτητος εξοπλισμός βελτιώνει την ταχύτητα κίνησης, την ταχύτητα κατά το γονάτισμα και την ταχύτητα που απαιτείται για την εκτέλεση συνδυαστικών κινήσεων και εξουδετερώσεων παικτών. Κάθε παίκτης ξεκινά με τον Τακτικό Εξοπλισμό εξοπλισμένο, εξασφαλίζοντας ένα βασικό επίπεδο ευκινησίας και ετοιμότητας για μάχη. Κάθε αναβάθμιση βελτιώνει περαιτέρω αυτές τις ικανότητες, κάνοντας τον παίκτη ταχύτερο και πιο ικανό σε όλες τις ενέργειες, παρέχοντας σημαντικό πλεονέκτημα, τόσο στην κίνηση, όσο και στα σενάρια μάχης.
Marauder Pack 	Το Marauder Pack είναι ένα σακίδιο που επιτρέπει στους παίκτες να μεταφέρουν βασικά αντικείμενα όπως ammo, med kits, bandages, adrenaline shots, και άλλα. Κάθε παίκτης ξεκινά με αυτό το ευέλικτο πακέτο εξοπλισμένο, εξασφαλίζοντας τη δυνατότητα να αποθηκεύσουν και να έχουν πρόσβαση σε κρίσιμα εφόδια κατά τη διάρκεια του παιχνιδιού. Κάθε αναβάθμιση επεκτείνει τον αριθμό των διαθέσιμων θέσεων αντικειμένων, επιτρέποντας στους παίκτες να μεταφέρουν περισσότερα εφόδια και να είναι καλύτερα προετοιμασμένοι για οποιαδήποτε κατάσταση.

4.9.5. Ammo

Wasp 7.62mm Ammo Pack



Το Wasp 7.62mm Ammo Pack είναι σχεδιασμένο αποκλειστικά για το ελεύθερο σκοπευτικό όπλο Ballista, προσφέροντας εξαιρετικά ακριβή και ισχυρά πυρομαχικά για μακρινές αποστάσεις. Αυτές οι σφαίρες μπορούν να βρεθούν στον ανοιχτό κόσμο, σε camps, καθώς και να αγοραστούν από Merchants. Οι σφαίρες 7.62mm Wasp εξασφαλίζουν ότι ο παίκτης διαθέτει την απαραίτητη ισχύ πυρός για να εξουδετερώσει στόχους από απόσταση, προσφέροντας υψηλή ακρίβεια και διάρκεια, καθιστώντας το Ballista ένα θανατηφόρο εργαλείο για τους ελεύθερους σκοπευτές.

Panther 5.56mm Ammo Pack



Το Panther 5.56mm Ammo Pack είναι συσκευασία σφαιρών σχεδιασμένη ειδικά για το όπλο FN SCAR-L, παρέχοντας αξιόπιστα και υψηλής απόδοσης πυρομαχικά. Αυτές οι συσκευασίες μπορούν να βρεθούν διάσπαρτες στον ανοιχτό κόσμο, σε διάφορα camps, ή να αγοραστούν από Merchants. Οι σφαίρες 5.56mm Panther εξασφαλίζουν ότι ο παίκτης είναι επαρκώς εξοπλισμένος για να αντιμετωπίσει οποιοδήποτε σενάριο μάχης, προσφέροντας συνεχή δύναμη πυρός και ακρίβεια.

4.10. Ακολουθία Παιχνιδιού

Ο παίκτης αναλαμβάνει τον ρόλο της Nora, της βασικής ηρωίδας του *No Man's Land*. Η Nora διαθέτει έναν μοναδικό συνδυασμό εξειδικευμένων όπλων, αντικειμένων και ειδικών κινήσεων που εξελίσσονται σταδιακά κατά την πρόοδο του παιχνιδιού. Καθώς εξερευνά τον μετα-αποκαλυπτικό κόσμο, αναβαθμίζει τον εξοπλισμό της, αυξάνει τα στατιστικά της (ζωή και αντοχή) και αποκτά νέες ικανότητες απαραίτητες για την ολοκλήρωση κρίσιμων αποστολών.

Ο βασικός στόχος της είναι η καταστροφή των βάσεων των Enforcers, προχωρώντας από τις περιφερειακές περιοχές προς την τεχνολογικά προηγμένη πόλη Eliezer. Κατά την πορεία της, αντιμετωπίζει εχθρούς αυξανόμενης δυσκολίας, συλλέγει πληροφορίες που συνδέονται με το παρελθόν της μητέρας της και προετοιμάζεται για την τελική αναμέτρηση με τον Basilisk.

Ο παίκτης θα περιηγηθεί στον κόσμο του παιχνιδιού με διάφορες επιλογές, όπως εκτέλεση βασικών αποστολών (των οποίων η τοποθεσία θα καθορίζεται στο τέλος κάθε main quest) και ελεύθερη περιήγηση για looting και εξερεύνηση περιοχών, με κύριο στόχο την καταστροφή των διαθέσιμων camps των Enforcers.

Το παιχνίδι χωρίζεται σε 3 επίπεδα αυξανόμενης δυσκολίας, εύκολο (πράσινο), μεσαίο (μοβ), δύσκολο (κόκκινο). Κάθε νέο κεφάλαιο ανοίγει ένα επιπλέον κομμάτι του χάρτη, καθιστώντας το διαθέσιμο για περιήγηση και αποστολές, ενώ ο παίκτης θα έχει πρόσβαση και στα προηγούμενα ξεκλειδωμένα κομμάτια. Η πρόοδος βασίζεται στην εξόντωση των εχθρών κάθε περιοχής και στην πρόσβαση στα ερευνητικά κέντρα των camps, όπου η Nora ανακαλύπτει ηχογραφήσεις, τεχνολογικά δεδομένα και αποκαλυπτικές λεπτομέρειες της ιστορίας. Τα κλειδωμένα τμήματα του χάρτη θα έχουν σύνορα με μεγάλους αριθμούς εχθρών Enforcers, οι οποίοι θα είναι πιο ισχυροί από τον παίκτη. Όταν ο παίκτης πλησιάζει αυτά τα σύνορα, θα λαμβάνει προειδοποίηση από τους NPCs και, αν δεν απομακρυνθεί, θα δέχεται ισχυρές επιθέσεις, οδηγούμενος πάντα σε ήττα.

Η περιπέτεια ξεκινά από την απομονωμένη περιοχή North-Lake και σταδιακά οδηγεί σε πιο αστικά, τεχνολογικά ανεπτυγμένα τοπία. Καθώς η Nora προχωρά, το περιβάλλον μεταβάλλεται σταδιακά, οι εχθροί ενισχύονται και ο αφηγηματικός τόνος γίνεται πιο σκοτεινός, κορυφούμενος στην τελική μάχη ενάντια στον Basilisk στο κέντρο της Eliezer, το σημείο όπου τεχνολογία και παρελθόν συγκρούονται καθοριστικά.

4.11. Δομή ακολουθιών

Στη συνέχεια παρουσιάζεται η ροή του παιχνιδιού στα πρώτα επίπεδα, καθώς και οι διαφορές που προκύπτουν στα επόμενα Camps, όσο η δυσκολία αυξάνεται και η Nora προσεγγίζει την πόλη Eliezer.

Η Nora φτάνει σε κάθε camp με βασικό στόχο την εξόντωση των εχθρών και την εύρεση της εισόδου του ερευνητικού κέντρου. Ο παίκτης έχει τη δυνατότητα να επιλέξει stealth προσέγγιση, αξιοποιώντας κατάλληλο εξοπλισμό όπως το DataWorm, που αδρανοποιεί προσωρινά Sentinels και Tracers, ή όπλα όπως το Harpoon, το οποίο μπορεί να εξουδετερώσει εχθρούς όπως οι Enforcers με αθόρυβο χτύπημα. Ωστόσο, λόγω του πλήθους των εχθρών και της αδυναμίας stealth εξόντωσης των Sentinels, κάθε camp καταλήγει αναπόφευκτα σε μάχη.

Η αποστολή θεωρείται επιτυχής όταν εξολοθρευθούν όλοι οι εχθροί στο camp. Στη συνέχεια, η Nora μπορεί να συλλέξει αντικείμενα από το camp (όπλα, αναβαθμίσεις, healing items κ.ά.) και να αναζητήσει την είσοδο του ερευνητικού κέντρου. Για να την ανοίξει, ενεργοποιεί έναν μηχανισμό ασφαλείας που απαιτεί την ολοκλήρωση ενός mini puzzle ή μηχανικής πρόκλησης (π.χ. επαναφορά ρεύματος, bypass πρωτοκόλλων, χρήση του μενταγιόν).

Αφού ανοίξει η είσοδος, η Nora εισέρχεται στο εσωτερικό του κέντρου και ξεκινά εξερεύνηση των εγκαταστάσεων, αναζητώντας την ηχογράφηση ή άλλο κρίσιμο τεκμήριο που θα οδηγήσει στην αποκάλυψη της ιστορίας και θα ξεκλειδώσει το επόμενο camp. Αυτή η δομή επαναλαμβάνεται στα επόμενα camps, με προοδευτική αύξηση της δυσκολίας, της εχθρικής ποικιλίας και της στρατηγικής πολυπλοκότητας.



Εικόνα 8. Ενδεικτική απεικόνιση ενός camp

Καθώς η Nora προχωρά στο δεύτερο επίπεδο του παιχνιδιού, τα camps παρουσιάζουν δομικές ομοιότητες με εκείνα του πρώτου επιπέδου, όμως εμφανίζονται σημαντικές διαφορές στην πρόκληση και τη στρατηγική προσέγγιση. Οι δυνάμεις της έχουν πλέον ενισχυθεί, μέσω αναβαθμίσεων στον εξοπλισμό, στατιστικών και ειδικών κινήσεων. Ωστόσο, οι εχθροί αυξάνονται σε αριθμό και γίνονται εμφανώς πιο απαιτητικοί.

Στο ίδιο επίπεδο, οι Sentinels κάνουν για πρώτη φορά την εμφάνισή τους, εισάγοντας έναν νέο τύπο αντιπάλου με αυξημένη ταχύτητα, ανθεκτικότητα και στρατηγικές κινήσεις. Παράλληλα, οι Tracers πλέον φέρουν οπλισμό, καθιστώντας τη stealth προσέγγιση πιο επικίνδυνη και τη μάχη πιο πολύπλευρη.

Στο τελικό επίπεδο, η δυσκολία κορυφώνεται. Οι Sentinels είναι πλέον πανίσχυροι, με σχεδόν πλήρη ανθεκτικότητα στις βασικές επιθέσεις, ενώ οι Tracers έχουν αποκτήσει δυνατότητα stealth, καθιστώντας τους αόρατους μέχρι να ανιχνευθούν ή να ενεργοποιήσουν επίθεση. Τα camps φυλάσσονται από μεγάλες, καλά οργανωμένες ομάδες εχθρών που επιτίθενται συντονισμένα, απαιτώντας από τον παίκτη πλήρη αξιοποίηση των διαθέσιμων πόρων και ικανοτήτων.

Σε αυτό το επίπεδο λαμβάνει χώρα η τελική και πιο απαιτητική μάχη του παιχνιδιού, με στόχο την καταστροφή του Basilisk, στο κέντρο της πόλης Eliezer. Η αναμέτρηση αυτή αποτελεί κορύφωση της αφήγησης και της μηχανικής πρόκλησης, απαιτώντας τόσο στρατηγική σκέψη όσο και γρήγορα αντανακλαστικά από τον παίκτη.

4.12. Τέλος παιχνιδιού

Στο τελευταίο ερευνητικό κέντρο, λίγο έξω από την πόλη Eliezer, η Nora ανακαλύπτει ένα συγκλονιστικό μυστικό που αλλάζει ριζικά την κατανόησή της για την αποστολή. Η μητέρα της, Eliza, που είχε εξαφανιστεί πριν από χρόνια, είχε καταφέρει πριν τον θάνατό της να φορτώσει τη συνειδήσή της σε ένα σύστημα τεχνητής νοημοσύνης. Το σύστημα αυτό, εγκατεστημένο βαθιά στο δίκτυο της Ascendance, περιέχει όχι μόνο τις αναμνήσεις και τη γνώση της, αλλά και κρίσιμες πληροφορίες για την καταστροφή του Basilisk.

Έχοντας επανενώσει τη δύναμη των Marauders, η Nora ξεκινά την τελική επίθεση προς την Eliezer, με στόχο τον κεντρικό κόμβο ελέγχου της Ascendance. Διασχίζοντας τις κατεχόμενες συνοικίες της πόλης, η Nora έρχεται αντιμέτωπη με τη μεγαλύτερη και πιο σύνθετη μάχη του παιχνιδιού, αντιμετωπίζοντας όλες τις διαθέσιμες δυνάμεις των Enforcers, μεταξύ αυτών μισθωτοί στρατιώτες, εξελεγμένοι Sentinels και stealth Tracers με ενισχυμένα πρωτόκολλα άμυνας και επίθεσης.

Καθώς η πίεση κορυφώνεται και το χάος κυριαρχεί, η Nora βασίζεται στα αντανακλαστικά, τη στρατηγική διαχείριση πόρων και τις ειδικές κινήσεις που έχει αποκτήσει μέχρι τώρα, ώστε να διασπάσει τις γραμμές άμυνας και να φτάσει στο κέντρο ελέγχου του Basilisk.

Στο τελευταίο στάδιο της αποστολής, η Nora καταφέρνει να εισχωρήσει στον πυρήνα του Basilisk και να τον καταστρέψει. Με την πτώση του Basilisk, οι Enforcers χάνουν τη συνοχή τους και η εξουσία του Basilisk καταρρέει. Η πόλη Eliezer αρχίζει να ανακτά την ελευθερία της και η ανθρωπότητα παίρνει ξανά τον έλεγχο της μοίρας της.

4.13. Οικονομία Παιχνιδιού

Τα κύριο νόμισμα του παιχνιδιού είναι τα Shards, απαραίτητα τόσο για την αναβάθμιση του εξοπλισμού του παίκτη, όσο και για την αγορά πολύτιμων αντικειμένων όπως Medkits, Ammo, Adrenaline Shots, Data Worms κ.α. Τα Shards μπορούν να συλλεχθούν από εξουδετερωμένους εχθρούς, καθώς και να ανακαλυφθούν σε μυστικά εργαστήρια στον κόσμο του παιχνιδιού. Μπορούν να χρησιμοποιηθούν στους Merchants, για αναβάθμιση εξοπλισμού και αγορά αντικειμένων. Αυτοί οι πολύτιμοι πόροι είναι κρίσιμοι για την πρόοδο και την ενίσχυση των ικανοτήτων του παίκτη.



Εικόνα 9. Shards

4.14. Χειρισμός



Εικόνα 10. Πλήκτρα χειρισμού

	Walk
	Look Around
	Jump/Mantle/Equip Item (Inventory)
	Loot/Interact/Drop Item (Inventory)
	Block/Split Item Stack (Inventory)
	Roll (Dodge)/Cover

	Use Equipped Item (Heal/Tactical)
	Open Special Move Wheel
	Aim (Hold)
	Attack/Heavy Attack
	Run (Toggle)
	Crouch (Toggle)
	Select Healing Item
	Select Tactical Item
	Previous Weapon
	Next weapon
SELECT 	Open Inventory
START 	Menu/Settings

5. Damage System

Στο πλαίσιο του παιχνιδιού, αναπτύχθηκε ένα ευέλικτο και επεκτάσιμο σύστημα damage (Damage System), το οποίο επιτρέπει την ανταλλαγή πληροφοριών και την εκτέλεση λογικής που σχετίζεται με τους πόντους ζωής των χαρακτήρων και αντικειμένων. Το σύστημα καλύπτει ένα ευρύ φάσμα σεναρίων, όπως:

- Απόδοση και λήψη ζημιάς μεταξύ διαφορετικών τύπων Actors
- Υλοποίηση healing και αναπλήρωσης ζωής
- Ενεργοποίηση αντιδράσεων (Damage Responses)
- Διαχείριση καταστάσεων εξουδετέρωσης
- Ενημέρωση συστημάτων AI (Behavior Trees και AI Controllers) με βάση την κατάσταση ζωής ενός Actor

Για την υλοποίηση του συστήματος επιλέχθηκαν σύγχρονες και αρθρωτές τεχνικές, με στόχο τη μεγιστοποίηση επαναχρησιμοποίησης, την απόδοση, και την προσαρμοστικότητα σε διαφορετικές περιπτώσεις χρήσης. Συγκεκριμένα, αξιοποιήθηκαν:

- Actor Components και Data Structures, για την ενσωμάτωση και επέκταση λειτουργιών σε πολλαπλούς Actors
- Blueprint Interfaces, για την υλοποίηση κοινών λειτουργιών χωρίς περιορισμούς ιεραρχίας κληρονομικότητας
- Macros, για την αποδοτικότερη χρήση πόρων και καθαρή διάταξη του κώδικα
- Enums, για τη διαχείριση διαφορετικών τύπων ζημιάς ή κατάστασης ενός Actor με ευανάγνωστο και συντηρήσιμο τρόπο
- Event Dispatchers, για την αξιόπιστη και ακριβή ενεργοποίηση κρίσιμων γεγονότων (όπως η εξουδετέρωση ενός Actor ή η ενεργοποίηση UI)

Το σύστημα εφαρμόστηκε ως Actor Component τόσο στον Blueprint του παίκτη όσο και στους Agents (εχθρικούς χαρακτήρες). Περιλαμβάνει όλες τις κοινές λειτουργίες που σχετίζονται με τους πόντους ζωής και τις αντίστοιχες ενέργειες, οι οποίες προσαρμόζονται ανάλογα με το εκάστοτε gameplay σενάριο. Για παράδειγμα, ελέγχει αν ένας Actor είναι επιλέξιμος να δεχτεί Damage (π.χ. έχει ενεργή ασπίδα ή είναι σε invulnerable κατάσταση). Ακόμη, εφαρμόζει την τιμή του Damage και ενεργοποιεί αντίστοιχα VFX, ήχους, και animation feedbacks. Επιπλέον, εκκινεί καταστάσεις εξουδετέρωσης, αν οι πόντοι ζωής μηδενιστούν.

Επιπρόσθετα, η βασική λειτουργικότητα επεκτείνεται σε κάθε Actor μέσω ενός ειδικά σχεδιασμένου Blueprint Interface. Αυτό επιτρέπει, για παράδειγμα, η ίδια βασική λογική αναπλήρωσης ζωής να εφαρμόζεται σε διαφορετικούς εχθρούς, ενώ ταυτόχρονα να διαφοροποιούνται συμπεριφορικά, π.χ. ένας εχθρός να τρέπεται σε φυγή όταν τραυματιστεί, ενώ ένας άλλος να γίνεται πιο επιθετικός. Η χρήση του Interface αποτρέπει την ανάγκη για περίπλοκες ιεραρχίες και διασφαλίζει ευελιξία και συμβατότητα σε

κάθε περίπτωση, διατηρώντας παράλληλα την αρχή της χαμηλής σύζευξης (low coupling) στον σχεδιασμό των μηχανικών του gameplay.

Τέλος, το σύστημα περιλαμβάνει δύο ξεχωριστές διεπαφές (UI) που σχεδιάστηκαν για να απεικονίζουν την κατάσταση των πόντων ζωής τόσο του παίκτη όσο και των Agents. Συγκεκριμένα, δημιουργήθηκε ένα HUD στοιχείο για τον παίκτη, το οποίο παρέχει συνεχή οπτική ενημέρωση σχετικά με τη ζωή του σε πραγματικό χρόνο, καθώς και ένα σύστημα Health Bar για τους εχθρούς, το οποίο εμφανίζεται δυναμικά όταν αυτοί δεχθούν damage. Η οπτική αναπαράσταση της κατάστασης υγείας ενισχύει τη στρατηγική κατανόηση της μάχης και συμβάλλει στη λήψη αποφάσεων από τον παίκτη.

Στο παρόν κεφάλαιο, παρουσιάζεται αναλυτικά η υλοποίηση των λειτουργιών που περιλαμβάνονται στο Actor Component του DamageSystem, καθώς και η χρήση του Blueprint Interface που σχεδιάστηκε για την υποστήριξή του. Τέλος, περιγράφεται η ανάπτυξη των γραφικών διεπαφών που απεικονίζουν την κατάσταση των πόντων ζωής του παίκτη και των εχθρών.

5.1. BPC_DamageSystem (Actor Component)

Η υλοποίηση του Damage System περιλαμβάνει τη δημιουργία ενός Actor Component (Epic Games, n.d.), το οποίο αποτελεί μια συλλογή μεταβλητών και functions που μπορούν να προστεθούν σε οποιονδήποτε Actor διαθέτει τη δυνατότητα να δεχθεί ή να αποδώσει damage (Damageable Actor). Η χρήση του προσφέρει ευελιξία στο σύστημα και αποτρέπει την ανάπτυξη επαναλαμβανόμενου κώδικα. Το Component περιλαμβάνει βασικές και κοινές λειτουργίες που σχετίζονται με τη διαχείριση της υγείας ενός Actor, καθώς και τις απαραίτητες μεταβλητές. Για παράδειγμα, το function TakeDamage είναι υπεύθυνο για την απόδοση damage στους πόντους ζωής ενός Actor, οι οποίοι εκπροσωπούνται από τη μεταβλητή Health (Float).

Στο παρόν project, το Component προστέθηκε στο blueprint του παίκτη (CBP_SandboxCharacter) και στο blueprint των εχθρών (BP_Enemy_Base). Ωστόσο, η σχεδιάσή του το καθιστά πλήρως επαναχρησιμοποιήσιμο, με δυνατότητα προσθήκης και σε άλλους Actors, όπως για παράδειγμα ένα εκρηκτικό αντικείμενο που, όταν δεχθεί πυρά, ενεργοποιεί έκρηξη και αποδίδει damage σε κοντινούς Actors.

Κατά την υλοποίηση, ορίστηκαν οι παρακάτω βασικές μεταβλητές:

- Health (Float): Εκπροσωπεί τους τρέχοντες πόντους ζωής
- MaxHealth (Float): Ορίζει τους μέγιστους πόντους ζωής
- IsInvincible (Boolean): Δηλώνει αν ο Actor είναι άτρωτος
- IsDead (Boolean): Δηλώνει αν ο Actor έχει εξουδετερωθεί
- IsInterruptible (Boolean): Δηλώνει αν μια επίθεση μπορεί να αποφευχθεί με ενεργή ενέργεια
- IsBlocking (Boolean): Εκπροσωπεί την κατάσταση άμυνας ή αποφυγής του Actor

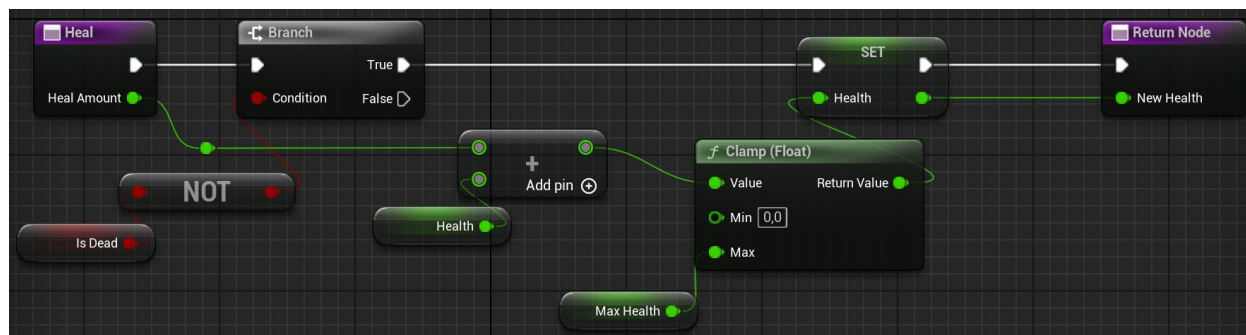
Με βάση αυτές τις μεταβλητές, υλοποιήθηκαν οι βασικές λειτουργίες TakeDamage και Heal, που σχετίζονται με την απόδοση και την αναπλήρωση πόντων ζωής αντίστοιχα. Επιπλέον, ενσωματώθηκε η μεταβλητή AttackTokensCount (Integer) και τα functions ReserveAttackToken και ReturnAttackToken,

τα οποία σχετίζονται με την υλοποίηση του Token System που αναλύεται στο υποκεφάλαιο 8.4. Η ενσωμάτωσή τους στο Component κρίθηκε σκόπιμη, καθώς αποτελούν αναπόσπαστο μέρος του Damage System και χρησιμοποιούνται από όλους τους Damageable Actors.

5.1.1. Heal (Function)

Το function είναι υπεύθυνο για την αναπλήρωση των πόντων ζωής ενός Actor, προσθέτοντας το αντίστοιχο ποσό στην τρέχουσα τιμή της μεταβλητής Health. Δέχεται ως είσοδο την τιμή HealAmount (Float), που αντιπροσωπεύει τους πόντους ζωής που θα προστεθούν και επιστρέφει τη νέα τιμή NewHealth (Float) μετά την αναπλήρωση. Κατά την εκτέλεση του, το function ελέγχει πρώτα την τιμή της μεταβλητής IsDead, ώστε να αποτρέψει την αναπλήρωση πόντων ζωής σε περίπτωση που ο Actor έχει ήδη εξουδετερωθεί.

Αν η μεταβλητή IsDead είναι False, τότε η τιμή του HealAmount προστίθεται στην Health. Πριν την τελική ανάθεση, εφαρμόζεται το node ClampFloat, για να διασφαλιστεί ότι η τελική τιμή δεν υπερβαίνει το μέγιστο όριο που ορίζεται από τη μεταβλητή MaxHealth. Για παράδειγμα, αν ένας παίκτης με τρέχον Health 30 και MaxHealth 100 χρησιμοποιήσει ένα medkit που προσθέτει 100 πόντους ζωής, η συνάρτηση θα περιορίσει την αναπλήρωση στους 70 πόντους, φέρνοντας το Health στο μέγιστο των 100. Η ενημερωμένη τιμή επιστρέφεται ως έξοδος του function. Το αντίστοιχο blueprint του function παρουσιάζεται στην εικόνα 11.



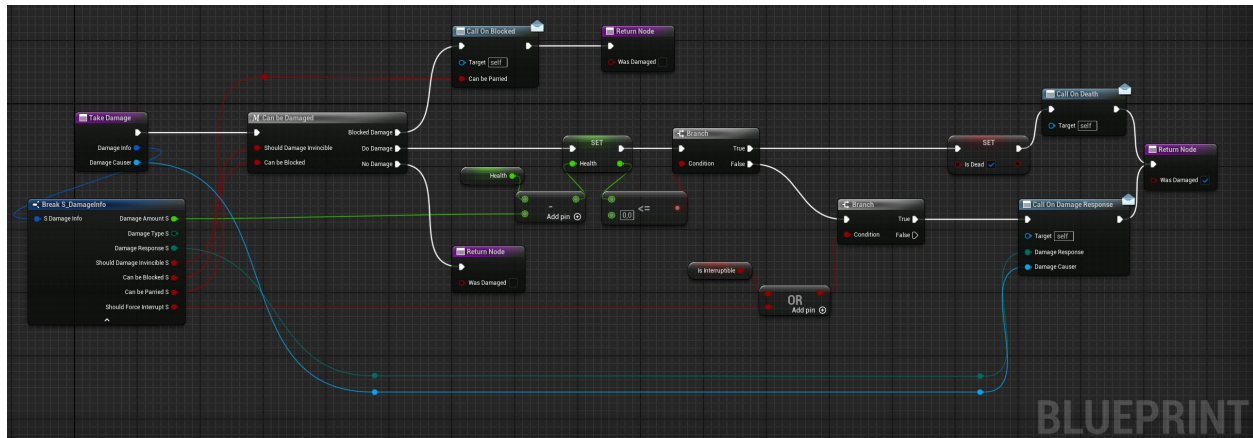
Εικόνα 11. Heal (Function)

5.1.2. TakeDamage (Function)

Το function είναι υπεύθυνο για την απόδοση Damage και τη μείωση των πόντων ζωής ενός Actor. Παράλληλα, ελέγχει τις συνθήκες υπό τις οποίες μπορεί να αποδοθεί Damage και καλεί Event Dispatchers για την εκτέλεση ενεργειών, όπως όταν ο Actor δέχεται ζημιά αλλά οι πόντοι ζωής του δεν μηδενίζονται ή όταν ο Actor πεθαίνει. Δέχεται ως εισόδους τη δομή DamageInfo (Structure), η οποία περιλαμβάνει όλες τις απαραίτητες πληροφορίες σχετικά με την επίθεση, και τον Damage Causer (Actor), που αναφέρεται στον Actor που προκάλεσε το damage. Επιστρέφει τη μεταβλητή WasDamaged (Boolean), η οποία αποτυπώνει αν το damage εφαρμόστηκε ή όχι.

Το function αξιολογεί τις παραμέτρους της επίθεσης και την κατάσταση του Actor που δέχεται την επίθεση, εκτελώντας τις κατάλληλες ενέργειες για κάθε περίπτωση. Αυτό επιτυγχάνεται μέσω του Macro

CanBeDamaged, το οποίο περιέχει τους απαραίτητους κόμβους εκτέλεσης και παρέχει execution pins για την ανάπτυξη λειτουργιών ανάλογα με την περίπτωση. Το αποτέλεσμα του Macro ενεργοποιεί επίσης Event Dispatchers, οι οποίοι "εκπέμπουν" το γεγονός σε άλλα assets ή μέρη του παιχνιδιού, ώστε να αντιδράσουν σε αυτά τα γεγονότα. Για παράδειγμα, το Event Dispatcher Death πυροδοτεί τις ενέργειες που εκτελούνται όταν ο Actor εξουδετερωθεί.



Εικόνα 12. TakeDamage (Function)

5.1.3. DamageInfo (Structure)

Το DamageInfo χρησιμοποιείται ως δομή για την οργάνωση των μεταβλητών που προσδιορίζουν τις παραμέτρους μιας επίθεσης. Οι μεταβλητές του καθορίζονται κατά την κλήση της συνάρτησης TakeDamage και περιλαμβάνουν:

- DamageAmount (Float): Η ζημιά που θα δεχτεί ο Actor
- DamageType (Enum): Ο τύπος του damage (π.χ. Projectile, Explosion, Environmental, κ.λπ.)
- DamageResponse (Enum): Ο τρόπος αντίδρασης του Actor όταν δέχεται damage (π.χ. HitReaction, Stun, Knockback, κ.λπ.)
- ShouldDamageInvincible (Boolean): Αν ο Actor θα δεχτεί damage ακόμα κι αν είναι άτρωτος (π.χ. από πυροβολισμούς)
- CanBeBlocked/CanBeParried (Boolean): Αν η επίθεση μπορεί να αποκρουστεί ή παρακαμφθεί
- ShouldForceInterrupt (Boolean): Αν η επίθεση πρέπει να διακόψει άλλες ενέργειες του Actor, όπως στην περίπτωση της επίθεσης Takedown

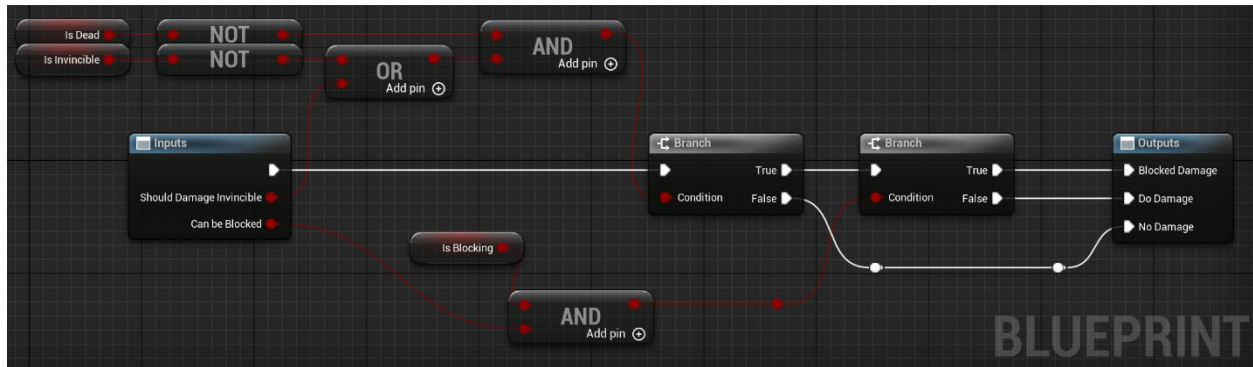
Can Be Damaged (Macro)

Το Macro δέχεται τις μεταβλητές ShouldDamageInvincible και CanBeBlocked και παρέχει τρεις εξόδους εκτέλεσης:

- DoDamage: Όταν ο Actor θα δεχθεί damage
- NoDamage: Όταν ο Actor δεν μπορεί να δεχθεί damage

- BlockedDamage: Όταν το damage αποκρούεται με επιτυχία

Για την εκτέλεση της εξόδου DoDamage, ελέγχεται ότι ο Actor δεν είναι εξουδετερωμένος (IsDead) και ταυτόχρονα ότι δεν είναι άτρωτος (IsInvincible) και η επίθεση δεν αποδίδει Damage ανεξαρτήτως των ενεργειών αποφυγής (ShouldDamageInvincible). Αν η συνθήκη είναι false τότε εκτελείται ο κώδικας που ακολουθείται από την έξοδο NoDamage. Αν η συνθήκη είναι True, τότε εξετάζεται αν ο Actor εκτέλεσε κίνηση block. Εάν η επίθεση μπορεί να αποκρουστεί, εκτελείται η έξοδος BlockedDamage ενώ η έξοδος DoDamage πυροδοτείται στην αντίθετη περίπτωση. Το Macro απεικονίζεται στη εικόνα 13.



Εικόνα 13. Can Be Damaged (Macro)

Στις περιπτώσεις εκτέλεσης των εξόδων No Damage και BlockedDamage, το function ολοκληρώνεται επιστρέφοντας τη Boolean WasDamaged με τιμή False. Η έξοδος DoDamage ενημερώνει τη μεταβλητή Health με την τιμή της διαφορά της με τη μεταβλητή DamageAmount και ελέγχεται αν η νέα τιμή είναι μικρότερη ή ίση του μηδενός. Σε αυτήν την περίπτωση, ο Actor χαρακτηρίζεται ως εξουδετερωμένος, με τη μεταβλητή IsDead να παίρνει την τιμή true και το function να ολοκληρώνεται επιστρέφοντας την WasDamaged με τιμή True. Σε αντίθετη περίπτωση, ελέγχεται αν οι μεταβλητές IsInterruptible ή ShouldForceInterrupt είναι αληθείς, προκειμένου η επίθεση να χαρακτηριστεί επιτυχής αλλά να πυροδοτηθούν οι ενέργειες αντίδρασης και όχι εξουδετέρωσης.

5.1.4. Event Dispatchers

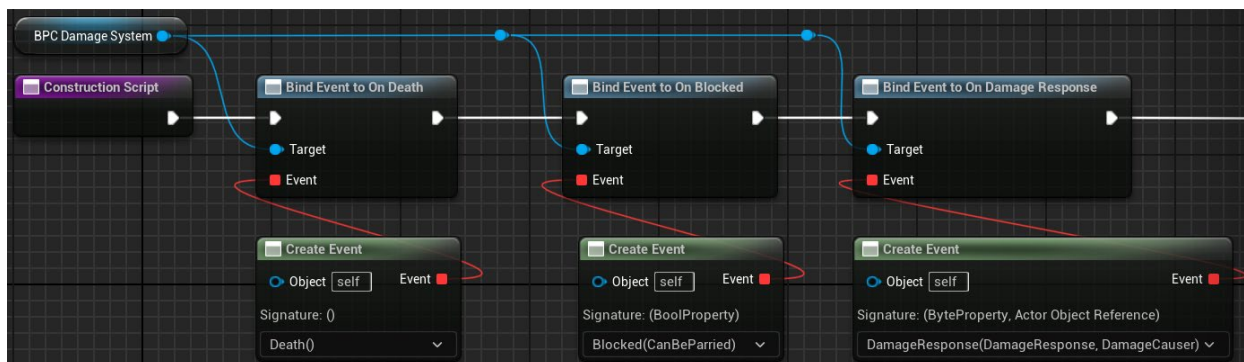
Το function TakeDamage μπορεί να κληθεί όχι μόνο από τα Blueprints των Actors, όπως τα CBP_SandboxCharacter και BP_Enemy_Base, αλλά και από οποιοδήποτε άλλο Blueprint που σχετίζεται με αυτούς, όπως τα BP_WeaponBase και BP_NPC_WeaponBase του οπλισμού. Για αυτόν τον λόγο, δημιουργήθηκαν τρία Event Dispatchers εντός του component. Σκοπός τους είναι να διασφαλίζεται η δυνατότητα εκτέλεσης ενεργειών εντός των Blueprint των Actors που δέχονται την επίθεση, ανεξαρτήτως του Blueprint που καλεί τη συνάρτηση. Με αυτόν τον τρόπο, κάθε Damageable Actor διαθέτει τρία Custom Events τα οποία πυροδοτούνται με την κλήση των αντίστοιχων Event Dispatchers και εκτελούν ενέργειες που σχετίζονται με το γεγονός. Αυτό επιτρέπει την τροποποίηση μεταβλητών, την αλλαγή της κατάστασης συμπεριφοράς του Actor (π.χ. αλλαγή από κατάσταση περιπολίας σε επιθετική), καθώς και

την εφαρμογή διαφόρων οπτικοακουστικών εφέ και αντιδράσεων (όπως UI Damage Indicators, Damage Reaction Animations, ήχους και Camera Shakes).

Τα Event Dispatchers και τα αντίστοιχα Custom Events που πυροδοτούν στο blueprint κάθε Actor είναι τα εξής:

- **Blocked:** Καλείται όταν μια επίθεση αποκρούεται με επιτυχία, πυροδοτώντας το Custom Event Blocked. Η μεταβλητή CanBeParried (Boolean) χρησιμοποιείται για να διαφοροποιηθούν οι ενέργειες που εκτελούνται στο Custom Event, ανάλογα με τον τρόπο απόκρουσης της επίθεσης (Block ή Parry).
- **Death:** Καλείται όταν οι πόντοι ζωής του Actor μηδενίζονται και πυροδοτεί το Custom Event Death. Σε αυτό το Custom Event, οι ενέργειες που εκτελούνται από έναν Agent όταν εξουδετερώνεται διαφέρουν από αυτές που εκτελεί ο παίκτης.
- **DamageResponse:** Καλείται όταν ο Actor δέχεται οποιοδήποτε Damage που δεν τον εξουδετερώνει, και πυροδοτεί το Custom Event DamageResponse. Δέχεται ως είσοδο τις μεταβλητές DamageResponse (Enum) και DamageCauser (Actor Object Reference) για την εφαρμογή διαφορετικών τρόπων αντίδρασης, ανάλογα με τα χαρακτηριστικά της επίθεσης και τον Actor που προκάλεσε τη ζημιά. Όπως και στο Custom Event Death, οι ενέργειες που εκτελούνται από έναν Agent όταν δέχεται Damage διαφέρουν από αυτές που εκτελεί ο Player.

Η ανάθεση των Custom Events στα αντίστοιχα Event Dispatchers πραγματοποιείται κατά την εκκίνηση του Actor μέσω του Construction Script του Blueprint του, όπως φαίνεται στην εικόνα 14. Με αυτόν τον τρόπο, οι λειτουργίες εφαρμόζονται κατά την αρχικοποίηση του Actor, διασφαλίζοντας την ομαλή και έγκαιρη εκτέλεση των απαιτούμενων ενεργειών. Η διαδικασία αυτή παρέχει έναν οργανωμένο και επαναχρησιμοποιήσιμο μηχανισμό διαχείρισης των γεγονότων και των αντιδράσεων, επιτρέποντας ευελιξία στον τρόπο με τον οποίο οι Actors ανταποκρίνονται σε διάφορα σενάρια εντός του παιχνιδιού.



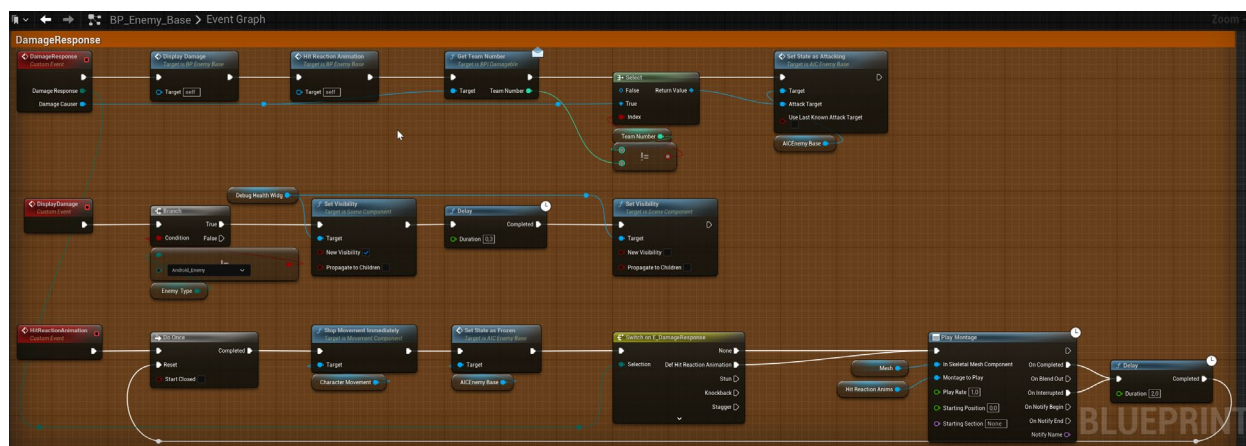
Εικόνα 14. Event Dispatchers Binding

DamageResponse (Agents) (CustomEvent): Το Custom Event DamageResponse εκτελεί τις ενέργειες που εφαρμόζονται όταν ένας Agent δέχεται damage και υλοποιείται στο blueprint BP_Enemy_Base των Agents. Συγκεκριμένα, περιλαμβάνει την εκτέλεση του Custom Event DisplayDamage, το οποίο καθιστά

ορατό το widget DebugHealth για μικρό χρονικό διάστημα, υποδεικνύοντας το ποσό damage που αφαιρέθηκε από τους πόντους ζωής του Agent, καθώς και το συνολικό ποσό του Health του.

Για να αποφευχθεί η ανεπιθύμητη εμφάνιση του widget σε εχθρούς που δεν μπορούν να δεχθούν damage από πυρομαχικά, η εκτέλεση αυτού του Custom Event περιορίζεται μόνο στους εχθρούς που δεν ανήκουν στον τύπο Android. Ο έλεγχος αυτός επιτυγχάνεται μέσω της μεταβλητής EnemyType (Enum), η οποία καθορίζει τον τύπο του Agent. Ως αποτέλεσμα, το DisplayDamage παραλείπεται, αποτρέποντας την εμφάνιση του widget για εχθρούς που δεν πλήττονται από συμβατικά πυρομαχικά. Στη συνέχεια, καλείται το function HitReactionAnims, το οποίο είναι υπεύθυνο για την ενεργοποίηση των animation αντίδρασης κατά την αποδοχή damage. Συγκεκριμένα, αναπαράγεται το HitReactionAnimation, το οποίο απεικονίζει τον Agent να εκτελεί μια απότομη κίνηση αντίδρασης, σηματοδοτώντας την επιτυχημένη επίθεση με ρεαλιστικό τρόπο. Στο σημείο αυτό, χρησιμοποιείται το node SwitchOnDamageResponse στη μεταβλητή DamageResponse, προκειμένου να ενεργοποιηθούν οι κατάλληλες έξοδοι εκτέλεσης για κάθε τύπο αντίδρασης που προκαλεί η επίθεση, όπως Stun, Stagger, Fall Damage, κ.ά.

Τέλος, το function ολοκληρώνεται μεταβάλλοντας την κατάσταση συμπεριφοράς του Agent σε επιθετική, εφόσον ο DamageCauser δεν ανήκει στην ίδια ομάδα με τον Agent. Ο έλεγχος αυτός επιτυγχάνεται μέσω της ανάκτησης του αριθμού της ομάδας στην οποία ανήκει ο DamageCauser, μέσω του function GetTeamNumber. Ο αριθμός της ομάδας αποθηκεύεται στη μεταβλητή TeamNumber (Integer), η οποία ανατίθεται σε κάθε Damageable Actor και χρησιμοποιείται για το διαχωρισμό συμμάχων από μη συμμάχους. Σε αυτό το project, ο παίκτης λαμβάνει την τιμή 1, ενώ κάθε Agent λαμβάνει την τιμή 2. Με αυτόν τον τρόπο, ο Agent δεν θα αλλάξει την κατάσταση του αν το damage προήλθε από σύμμαχο Agent, ενώ θα ενεργήσει επιθετικά εάν το damage προήλθε από αντίπαλο. Το διάγραμμα της υλοποίησης του DamageResponse ενός Agent παρουσιάζεται στην εικόνα 15.

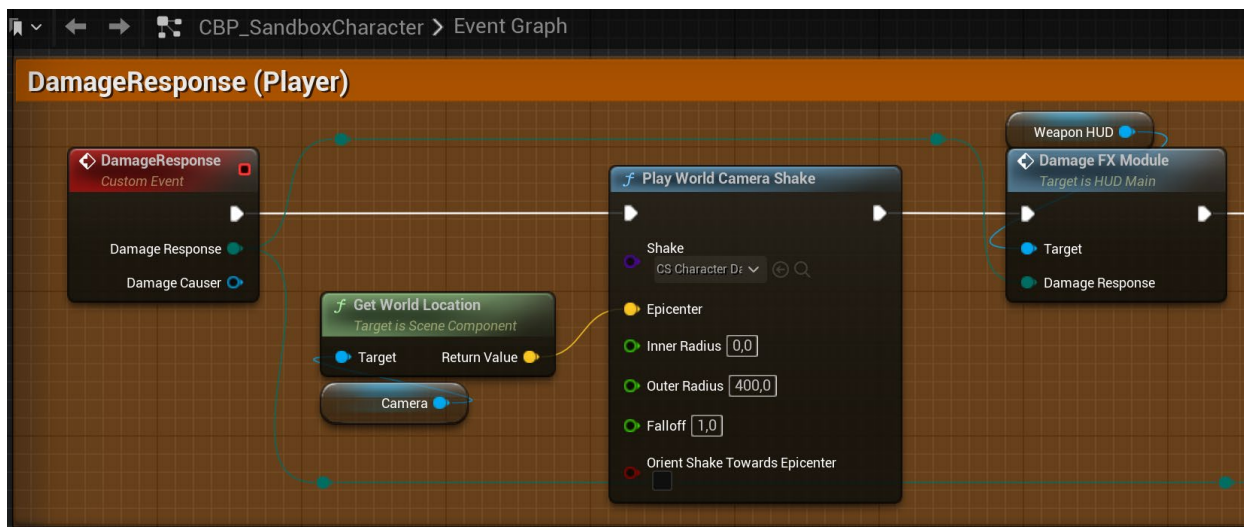


Εικόνα 15. DamageResponse (Agents)

Λόγω των ειδικών απαιτήσεων του παρόντος project, καθώς και των περιορισμένων διαθέσιμων assets, για την τρέχουσα υλοποίηση υποστηρίζεται μόνο η αντίδραση σε damage που προκαλείται από πυρομαχικά μέσω της εξόδου εκτέλεσης DefHitReactionAnimation. Παρόλα αυτά, ο τρόπος ανάπτυξης

επιτρέπει την προσθήκη επιπλέον κώδικα για την εφαρμογή ειδικών εφέ, ήχων και άλλων αντιδράσεων για κάθε μία από τις διαφορετικές αντιδράσεις που ορίζονται στο Enum, καθώς και την προσθήκη νέων τύπων αντίδρασης στο μέλλον.

DamageResponse (Player): (CustomEvent): Το Custom Event DamageResponse εκτελεί τις ενέργειες που εφαρμόζονται όταν ο παίκτης δέχεται damage και υλοποιείται στο blueprint CBP_SandboxCharacter του παίκτη. Συγκεκριμένα, το event χρησιμοποιεί το node PlayWorldCameraShake, το οποίο προκαλεί στιγμιαία κίνηση της κάμερας του παίκτη με τυχαίο τρόπο, προκειμένου να αποδοθεί η ένταση της στιγμής και να ενισχυθεί η αίσθηση εμβύθισης, ειδοποιώντας παράλληλα τον παίκτη για το γεγονός της επίθεσης. Αυτή η κίνηση της κάμερας συντελεί στη δημιουργία μιας πιο έντονης και ρεαλιστικής αίσθησης κατά τη διάρκεια των επιθέσεων, ενισχύοντας τη συνολική εμπειρία του χρήστη. Επιπλέον, καλείται το Custom Event DamageFX Module του widget HUD_Main, το οποίο δέχεται ως είσοδο τη μεταβλητή DamageResponse και εμφανίζει αντίστοιχες ενδείξεις damage στην οθόνη του παίκτη σύμφωνα με τον τύπο της επίθεσης, όπως φαίνεται στην εικόνα 20. Η εμφάνιση αυτών των ενδείξεων στο UI παρέχει στον παίκτη οπτική ανατροφοδότηση σχετικά με την επίδραση του damage που έχει δεχτεί. Σε αυτό το σημείο, είναι δυνατή η προσθήκη επιπλέον εφέ, όπως ήχοι που σχετίζονται με την damage ή damage animations, προκειμένου να ενισχυθεί περαιτέρω η σήμανση του γεγονότος. Το διάγραμμα της υλοποίησης του DamageResponse ενός Agent παρουσιάζεται στην εικόνα 16.



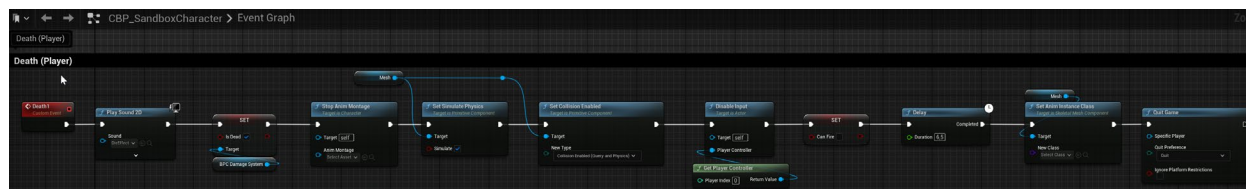
Εικόνα 16. DamageResponse (Player)

Αναφορικά με τις μεταβλητές DamageResponse και DamageCauser, αυτές μπορούν να αξιοποιηθούν κατά τη διάρκεια της εκτέλεσης του event για την ανάπτυξη περαιτέρω ενεργειών, όπως η εφαρμογή διαφορετικών τύπων ανατροφοδότησης ή εφέ, ανάλογα με το είδος του damage ή τον δράστη της επίθεσης. Η δυνατότητα αξιοποίησης αυτών των μεταβλητών μπορεί να συμβάλλει στην περαιτέρω εξατομίκευση της αντίδρασης του παίκτη και στην ενίσχυση της δυναμικής του παιχνιδιού.

Death (Agents): Κατά την εκτέλεση του Custom Event Death για έναν Agent, καλείται το Custom Event DeathWrapper, το οποίο δέχεται ως είσοδο τη μεταβλητή IsByTakedown (Boolean) και λειτουργεί ως ο βασικός κώδικας που περιλαμβάνει όλες τις ενέργειες που σχετίζονται με την εξουδετέρωση ενός NPC. Το DeathWrapper περιλαμβάνει την εμφάνιση του Loot που κερδίζει ο παίκτης μετά την εξουδετέρωση του Agent, τις ειδοποιήσεις στο UI για την ένδειξη του γεγονότος, καθώς και την επιστροφή των attack tokens που ενδεχομένως έχουν δεσμευτεί κατά τη διάρκεια της μάχης. Επιπλέον, περιλαμβάνει την αλλαγή του Behavior State στην κατάσταση Dead για τον Agent, καθώς και λειτουργίες που αφορούν το Quest System, το οποίο επιτρέπει την ολοκλήρωση αποστολών ή την ενημέρωση της προόδου σε τυχόν ενεργές αποστολές. Τέλος, εκτελούνται οπτικοακουστικά εφέ που υπογραμμίζουν τη διαδικασία της εξουδετέρωσης, όπως εφέ ήχου και animation.

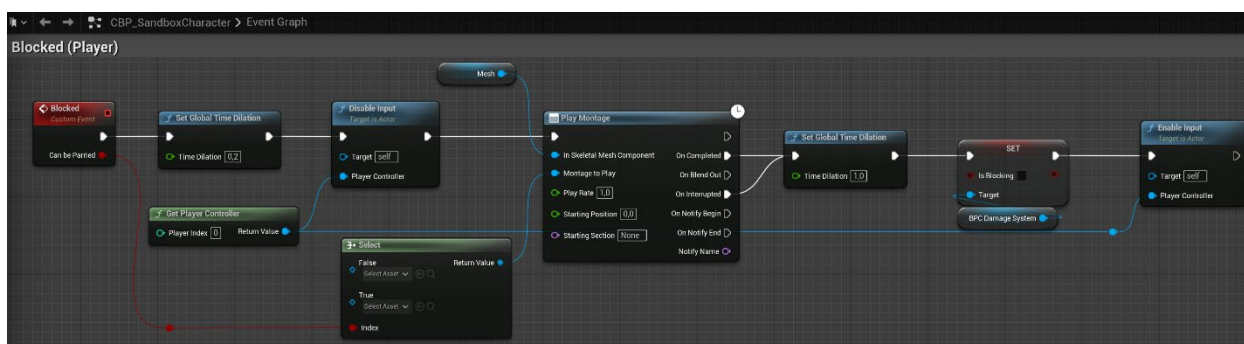
Η χρήση του DeathWrapper επιτρέπει τη διαφοροποίηση των λειτουργιών που εκτελούνται ανάλογα με τον τρόπο εξουδετέρωσης του Agent. Συγκεκριμένα, το DeathWrapper εκτελεί διαφορετικές ενέργειες ανάλογα με το αν ο Agent εξουδετερώνεται μέσω του function TakeDamage (π.χ. πυρομαχικά) ή αν η εξουδετέρωση πραγματοποιείται μέσω της ενέργειας Takedown, κατά την οποία η συνάρτηση TakeDamage δεν χρησιμοποιείται. Οι ενέργειες που περιλαμβάνονται στο DeathWrapper και ο τρόπος που αυτές διαφοροποιούνται περιγράφονται αναλυτικά στο υποκεφάλαιο 8.7.

Death (CBP_SandboxCharacter): Σε αυτό το event, ομοίως, καλούνται οι απαραίτητες ενέργειες που αφορούν στην εξουδετέρωση του παίκτη, συμπεριλαμβανομένης της εξόδου από το παιχνίδι, καθώς το mode του παιχνιδιού είναι Single Player. Συγκεκριμένα, το node PlaySound2D αναπαράγει έναν ήχο UI που σηματοδοτεί τον τερματισμό του παιχνιδιού, και η μεταβλητή IsDead του component παίρνει την τιμή true, υποδεικνύοντας την αλλαγή κατάστασης του παίκτη εξουδετερωμένο. Ακολούθως, τα nodes StopAnimMontage (null), SetSimulatePhysics (true), SetCollisionEnabled (QueryAndPhysics), διακόπτουν το τρέχον animation του παίκτη, ενώ ταυτόχρονα ενεργοποιείται η φυσική προσομοίωση κίνησης (ragdoll), προκειμένου ο παίκτης να πέσει στο έδαφος με φυσικό τρόπο, παρέχοντας μια ρεαλιστική κινησιολογία κατά την εξουδετέρωση. Επιπλέον, το node DisableInput απενεργοποιεί τα πλήκτρα εισόδου του παίκτη για να μην μπορούν να γίνουν άλλες ενέργειες κατά τη διάρκεια της εξουδετέρωσης. Παράλληλα, η μεταβλητή CanFire (Boolean) εξασφαλίζει ότι οποιαδήποτε ενδεχόμενη ενεργοποιημένη πυροδότηση του οπλισμού του παίκτη θα διακοπεί, αποφεύγοντας την εκτέλεση ακατάλληλων ενεργειών. Τέλος, το node SetAnimInstanceClass (null) αποδεσμεύει τον Actor από το Animation Blueprint για λόγους υπολογιστικού κόστους, ενώ το Quit Game τερματίζει το παιχνίδι, ολοκληρώνοντας τις ενέργειες εξουδετέρωσης. Το διάγραμμα της διαδικασίας φαίνεται στην εικόνα 17.



Εικόνα 17. Death (CBP_SandboxCharacter)

Blocked: Το custom event παρέχει τη δυνατότητα εφαρμογής λειτουργιών, όταν ένας Actor εκτελέσει επιτυχημένα ένα block όπως οπτικά ή ηχητικά εφέ, animations, ενδείξεις UI κ.α. Στο συγκεκριμένο project, λόγω διαφορετικού σχεδιασμού των μηχανικών, δεν εφαρμόστηκαν κινήσεις block, με αποτέλεσμα να μην αναπτυχθούν οι σχετικές λειτουργίες. Ωστόσο, η εικόνα 18 περιλαμβάνει μια ενδεχόμενη υλοποίηση στο blueprint του παίκτη, κατά την οποία ο χρόνος του παιχνιδιού επιβραδύνεται μέσω του node SetGlobalTimeDilation και τα πλήκτρα εισόδου απενεργοποιούνται, διασφαλίζοντας την ανεπηρέαστη εκτέλεση του function. Επιπλέον, αναπαράγεται ένα animation που απεικονίζει τον παίκτη να εκτελεί την ενέργεια Block. Κατά την ολοκλήρωση του animation, ο χρόνος του παιχνιδιού επιστρέφει στην αρχική του κατάσταση, η μεταβλητή IsBlocked του component αρχικοποιείται και τα πλήκτρα εισόδου του παίκτη καθίστανται και πάλι ενεργά. Τέλος, η μεταβλητή CanBeParried μπορεί να αξιοποιηθεί για την αναπαραγωγή διαφορετικών animations σύμφωνα με τον τρόπο που η αποφυγή πραγματοποιήθηκε.



Εικόνα 18. Blocked

5.1.5. Damage Indicators (UI)

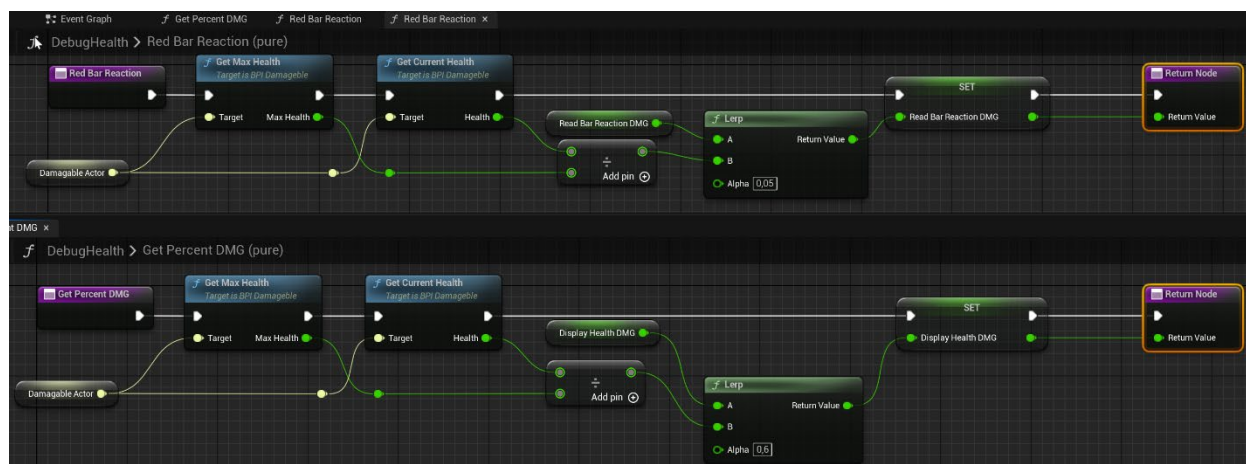
Για να αποτυπωθεί οπτικά η απόδοση damage από τον παίκτη προς έναν Agent και αντίστροφα, δημιουργήθηκαν δύο User Widget Blueprints, το DebugHealth και RadialHealthBar.

DebugHealth (Agents) (User Widget): Το DebugHealth εμφανίζεται για κάθε Agent που δεν ανήκει στον τύπο Android, κάθε φορά που δέχεται μείωση στους πόντους ζωής του. Στο πλαίσιο του σχεδιασμού του παρόντος project, βασικός στόχος της εν λόγω λειτουργίας είναι η οπτικοποίηση των συνολικών πόντων ζωής του Agent μέσω της αντίστοιχης μπάρας, η οποία παράλληλα εμφανίζει το ποσό της μείωσης αυτών σε κάθε επίθεση. Η υλοποίηση αυτή στοχεύει στην ενίσχυση της αντίληψης του παίκτη σε σενάρια μάχης. Επιπλέον, η μπάρα ζωής τοποθετήθηκε σε συγκεκριμένη απόσταση πάνω από το ψηλότερο σημείο του κάθε Agent και επιλέχθηκε να αποδίδεται σε Screen Space, ώστε να είναι πάντα ορατή, να έχει συγκεκριμένο μέγεθος ανεξαρτήτως εμποδίων μεταξύ του παίκτη και του Agent ή της μεταξύ τους απόστασης και η κατεύθυνση της μπάρας να είναι πάντα προς τον παίκτη.

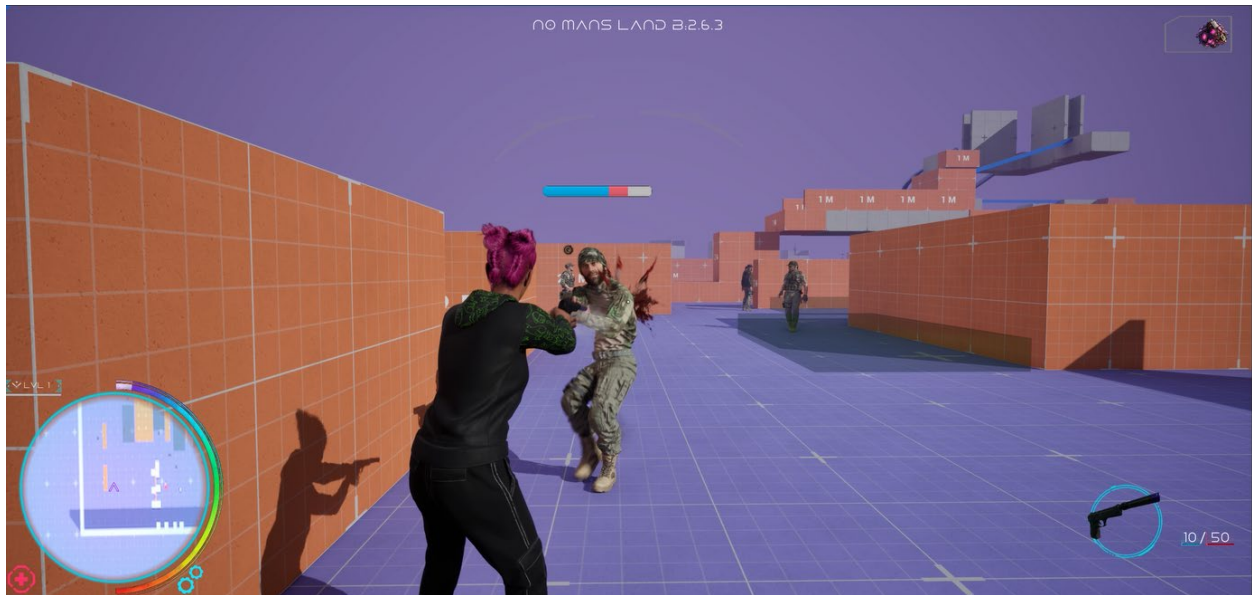
Η υλοποίησή της περιλαμβάνει την προσθήκη ενός Widget Component στο Parent Blueprint των Agents (BP_Enemy_Base), το οποίο χρησιμοποιείται από κάθε υποκλάση αυτού του Actor. Αυτό το Widget Component παρέχει μια επιφάνεια απόδοσης του ορισμένου Widget (DebugHealth) στο τρισδιάστατο περιβάλλον. Στο Blueprint του Widget προστέθηκαν δύο Progress Bars, τοποθετημένα το ένα πάνω στο

άλλο, με στόχο να ενημερώνονται ταυτόχρονα βάσει της τρέχουσας τιμής των πόντων ζωής. Το πρώτο Progress Bar έχει κόκκινο χρώμα και ενημερώνεται χωρίς καθυστέρηση, αντιπροσωπεύοντας το μέγεθος του Damage, σε αντίθεση με το δεύτερο που αντιπροσωπεύει τους υπολειπόμενους πόντους ζωής. Με αυτό τον τρόπο, όταν το Widget γίνει εμφανές, αποτυπώνεται το μέγεθος του Damage σε σχέση με τους υπολειπόμενους πόντους ζωής.

Επιπλέον, εντός του Widget δημιουργήθηκαν τα functions GetPercent_DMG και RedBarReaction, τα οποία ενημερώνουν τις αντίστοιχες τιμές των Progress Bars μέσω των τιμών που παρέχονται από το DamageSystem. Συγκεκριμένα, δημιουργήθηκε η μεταβλητή DamageableActor τύπου BPI_Damageable, από την οποία ανακτώνται οι μεταβλητές Health και MaxHealth, μέσω των functions GetCurrentHealth και GetMaxHealth, οι οποίες διαιρούνται για να δημιουργήσουν το ποσοστό που θα δεχθεί ως είσοδο το Progress Bar. Τα δυο functions είναι πανομοιότυπα με τη διαφορά της τιμής του node Lerp, που χρησιμοποιείται πριν την ενημέρωση του Progress Bar, για να επιτευχθεί η σταδιακή ενημέρωση των τιμών με διαφορετικούς χρόνους. Τα functions ενημέρωσης της μπάρας ζωής των Actors παρουσιάζονται στην εικόνα 19 και η εικόνα 20 απεικονίζει το Widget εντός του παιχνιδιού κατά το αρχικό στάδιο ανάπτυξης.

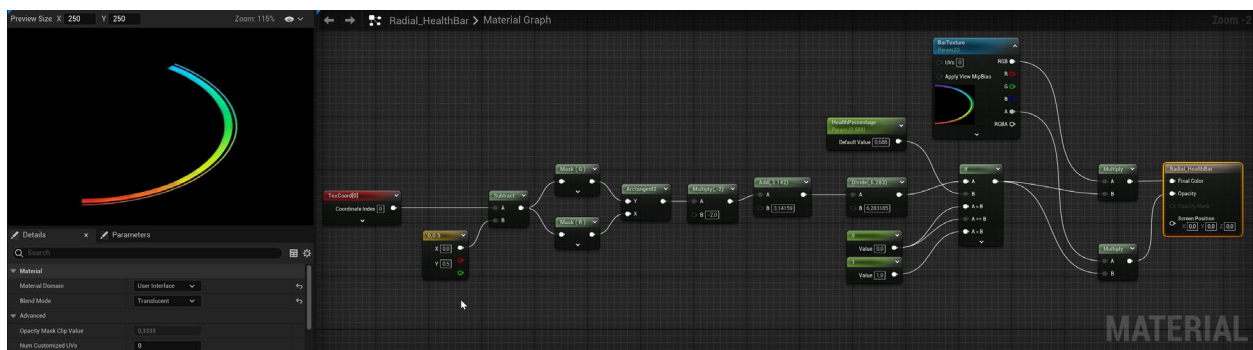


Εικόνα 19. DebugHealth Widget Blueprints



Εικόνα 20. DebugHealth Widget

RadialProgressBar (Player) (User Widget): Το RadialProgressBar αποτελεί το Health Indicator για τον παίκτη και εφαρμόζει αντίστοιχη λογική με αυτή του DebugHealth για την ενημέρωσή του. Ωστόσο, το συγκεκριμένο Widget είναι μέρος του βασικού UI του παίκτη, το οποίο παραμένει μονίμως εμφανές. Μια ακόμη διαφορά είναι πως το σχήμα είναι ημικύκλιο, κάτι το οποίο απαιτεί τη δημιουργία ενός Material και την ανάθεση αυτού να προβάλλεται ως εικόνα στο UI, καθώς δε διατίθεται επιλογή Progress Bar σχήματος ημικυκλίου από την μηχανή. Για τον λόγο αυτόν, δημιουργήθηκε μια εικόνα με χρήση AI από τη σελίδα Leonardo AI, η οποία προστέθηκε στο Material Radial_HealthBar. Η λογική που περιέχει το Material χωρίζει την εικόνα σε 4 τεταρτημόρια, τα οποία καλύπτονται/αποκαλύπτονται βάσει της μεταβλητής HealthPercentage (Scalar), πραγματοποιώντας κινήσεις πρώτου-τέταρτου τεταρτημόριου, κατά την απόδοση Damage/Heal. Με αυτό τον τρόπο, όσο η τιμή της μεταβλητής μεγαλώνει, σύμφωνα με το Damage που έχει δεχθεί ο παίκτης, η μπάρα ζωής μικραίνει.



Εικόνα 21. RadialProgressBar Material Blueprint

Τέλος, για την ομαλή ενημέρωση των τιμών και την ταυτόχρονη απόδοση των τιμών Damage και συνολικής υπολειπόμενης ζωής, δημιουργήθηκαν δύο ακόμη πανομοιότυπες εικόνες με χρώμα κόκκινο και λευκό, οι οποίες τοποθετήθηκαν η μία πάνω στην άλλη, ακολουθώντας τη λογική υλοποίησης του DebugHealth των Agents's. Ο κώδικας του Material φαίνεται στην εικόνα 21 και η εικόνα 22 απεικονίζει τα δύο Damage Indicators (των Agents και του παίκτη), τη στιγμή που δέχονται ταυτόχρονα Damage κατά τη διάρκεια του gameplay, στο τελικό στάδιο ανάπτυξης του παιχνιδιού.



Εικόνα 22. Damage Indicators

5.2. BPI_Damageable

Το DamageSystem ενσωματώνει ένα Blueprint Interface (BPI) asset για την κλήση των κοινών λειτουργιών του component, διασφαλίζοντας τη μέγιστη ευελιξία και την επεκτασιμότητα των λειτουργιών του συστήματος με αποδοτικό τρόπο.

Στο πλαίσιο του συγκεκριμένου project, πυροδοτούνται πολλαπλά functions τα οποία εκτελούν ενέργειες και ανακτούν ή τροποποιούν μεταβλητές συμπεριλαμβανομένων αυτών του BPC_DamageSystem component. Η χρήση ενός BPI επιτρέπει τον ορισμό αυτών των functions και την ανάπτυξή τους σε επίπεδο Actor, παρέχοντας τη δυνατότητα προσαρμογής και επέκτασης των ενεργειών ανάλογα με τις ανάγκες κάθε Actor. Συγκεκριμένα, καθορίζει τις μεταβλητές εισόδου και εξόδου που χρησιμοποιούνται από κάθε function και ανατίθεται σε όλους τους Actors που απαιτούν την εφαρμογή των εν λόγω λειτουργιών.

Η υλοποίηση των functions που ορίζονται στο Interface ενσωματώνεται στο blueprint του κάθε Actor, με τις ενέργειες να κληρονομούνται σε κάθε υποκλάση του. Για παράδειγμα, οι υποκλάσεις των Agents τύπου Ranged και Android, οι οποίες αναπαριστούν τους εχθρούς ρομπότ και τους ανθρώπους που

διαθέτουν οπλισμό αντίστοιχα, εκτελούν την ίδια ενέργεια που υλοποιείται στο BPI function του parent blueprint (BP_Enemy_Base) τους, εξαλείφοντας την ανάγκη για ανάπτυξη διπλότυπου κώδικα.

Επιπλέον, οι ενέργειες αυτές μπορούν να αντικατασταθούν ή να επεκταθούν στο blueprint κάθε υποκλάσης, παρέχοντας τη δυνατότητα προσαρμογής σύμφωνα με τον τύπο του Agent που τις χρησιμοποιεί. Για παράδειγμα, το function Heal μπορεί να επεκταθεί για να αναπαράγει διαφορετικά ηχητικά εφέ ανά τύπο Agent, ενώ η διαδικασία αναπλήρωσης των πόντων ζωής παραμένει αμετάβλητη.

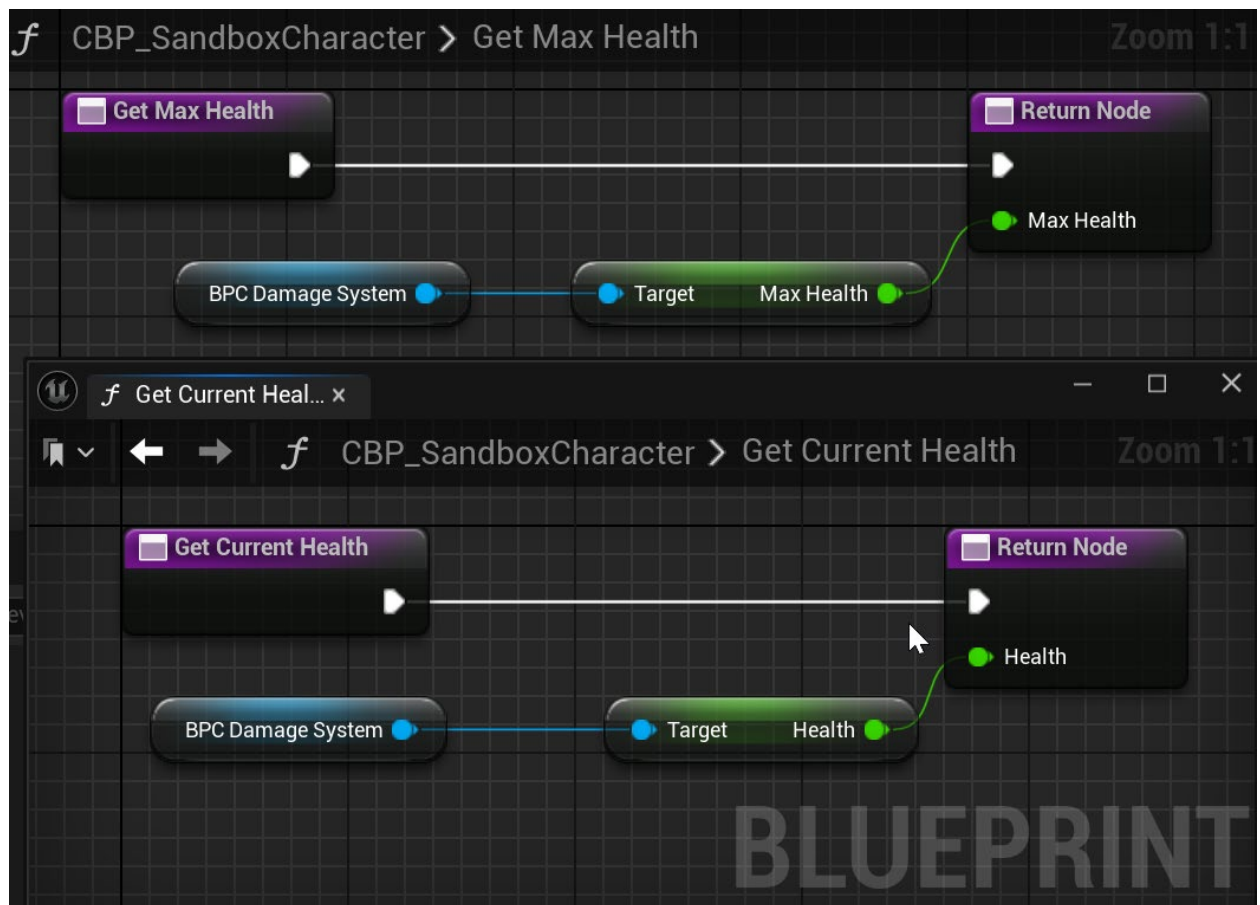
Η πυροδότηση ενεργειών που εκτελούνται από έναν Actor, απαιτεί την παροχή αναφοράς αυτού του Actor κατά την κλήση της ενέργειας. Για παράδειγμα, το function TakeDamage απαιτεί το reference του Actor, ώστε η μείωση της υγείας να πραγματοποιηθεί στο κατάλληλο Actor. Συνεπώς, η χρήση του BPI εξασφαλίζει έναν αποδοτικό τρόπο κλήσης των ενεργειών από οποιοδήποτε Blueprint, χωρίς την ανάγκη χρήσης του CastTo node για την παροχή της αναφοράς, το οποίο ενδέχεται να αυξήσει το υπολογιστικό κόστος σε περιπτώσεις συνεχούς κλήσης.

Για την ανάπτυξη των παραπάνω λειτουργιών, το Blueprint Interface BPI_Damageable ανατέθηκε στο τμήμα Implemented Interfaces των Class Settings στο Blueprint του παίκτη (CBP_SandboxCharacter) και στο Parent Blueprint των Agents (BP_Enemy_Base). Περιλαμβάνει functions για την κλήση κάθε λειτουργίας που έχει αναπτυχθεί στο component (όπως Heal και TakeDamage), καθώς και επιπλέον functions που επιτρέπουν την ανάκτηση ή την τροποποίηση μεταβλητών για τον παίκτη ή κάποιον Agent, από διάφορα Blueprints του project.

Λειτουργίες του Blueprint Interface:

1. GetCurrentHealth και GetMaxHealth

Αυτά τα functions επιστρέφουν τις μεταβλητές Health (Float) και MaxHealth (Float), οι οποίες ανακτώνται από το component του Actor και απεικονίζουν αντίστοιχα την τρέχουσα και τη μέγιστη τιμή υγείας του Actor. Χρησιμοποιούνται σε διάφορες ενέργειες του project που απαιτούν την ανάκτηση αυτών των τιμών, όπως στα functions Medkit (Use) και Syringe (Use) στο Blueprint του παίκτη, κατά τη διάρκεια της χρήσης αντικειμένων αναπλήρωσης των πόντων ζωής. Επίσης, αξιοποιούνται εντός των functions GetPercentDMG και Get_Health_DamageSystem των Widget blueprints που αφορούν στα Damage Indicators των Actos. Τέλος, χρησιμοποιούνται σε ενέργειες που αφορούν στη συμπεριφορά του Agent, όπως για παράδειγμα στον Decorator BTD_IsHealthLow των Behavior Trees, ο οποίος λειτουργεί ως συνθήκη προκειμένου ο Agent να αναπληρώσει του πόντους ζωής του, αν αυτοί βρεθούν σε χαμηλές τιμές.



Εικόνα 23. `GetCurrentHealth` και `GetMaxHealth`

2. Heal

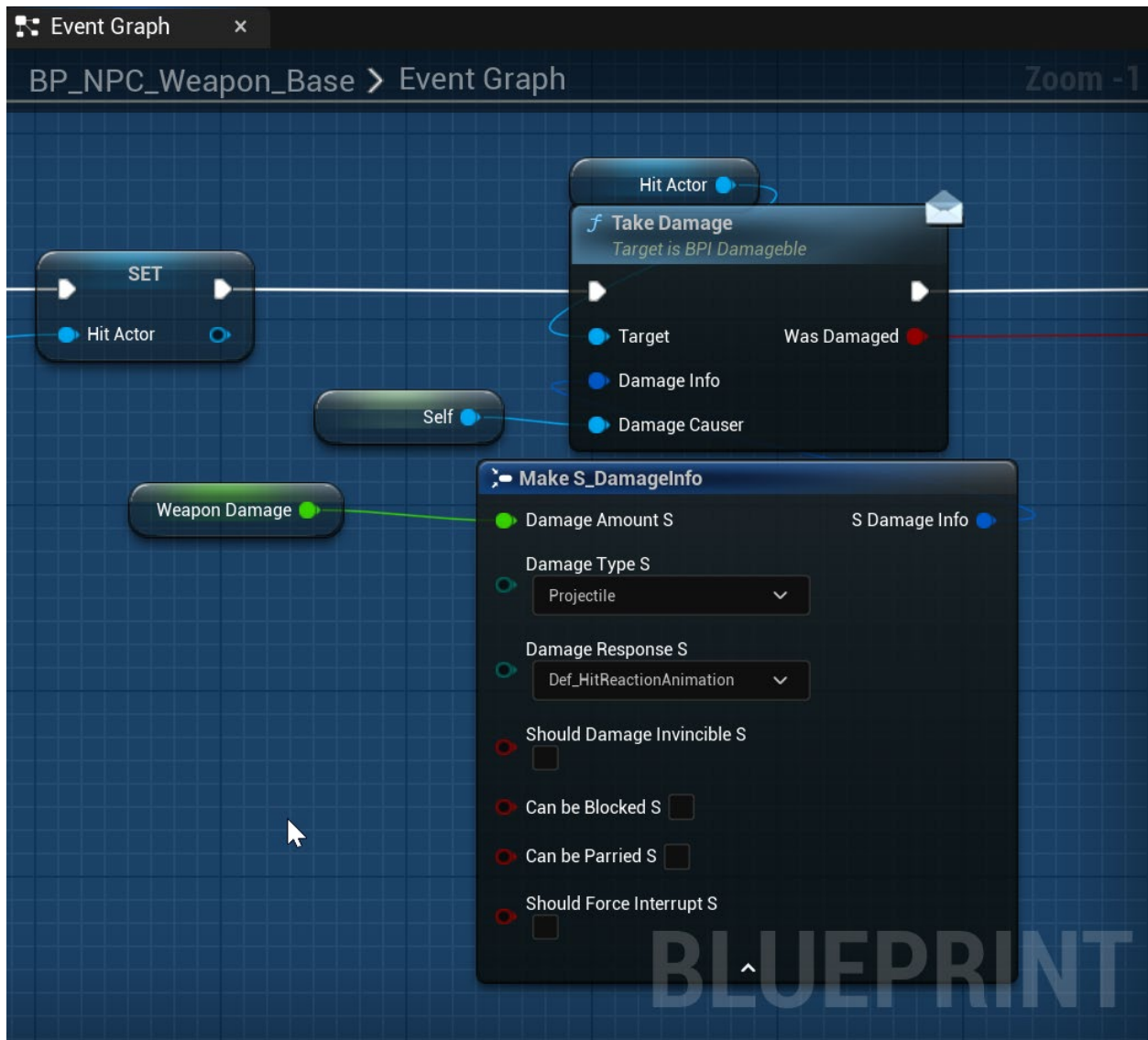
Το function δέχεται ως είσοδο τη μεταβλητή HealAmmount (Float) η οποία υποδεικνύει τον αριθμό των πόντων ζωής που θα ανακτηθούν κατά τη διαδικασία του Healing, και καλεί το αντίστοιχο function Heal του component, επιστρέφοντας τη μεταβλητή NewHealth (Float). Παραδείγματα χρήσης περιλαμβάνουν τα functions Medkit (Use) και Syringe (Use) του Blueprint του παίκτη, καθώς και το Behavior Tree task (BTT_Heal), το οποίο ενεργοποιείται όταν οι πόντοι ζωής των Agents μειωθούν κάτω από ένα ορισμένο όριο.



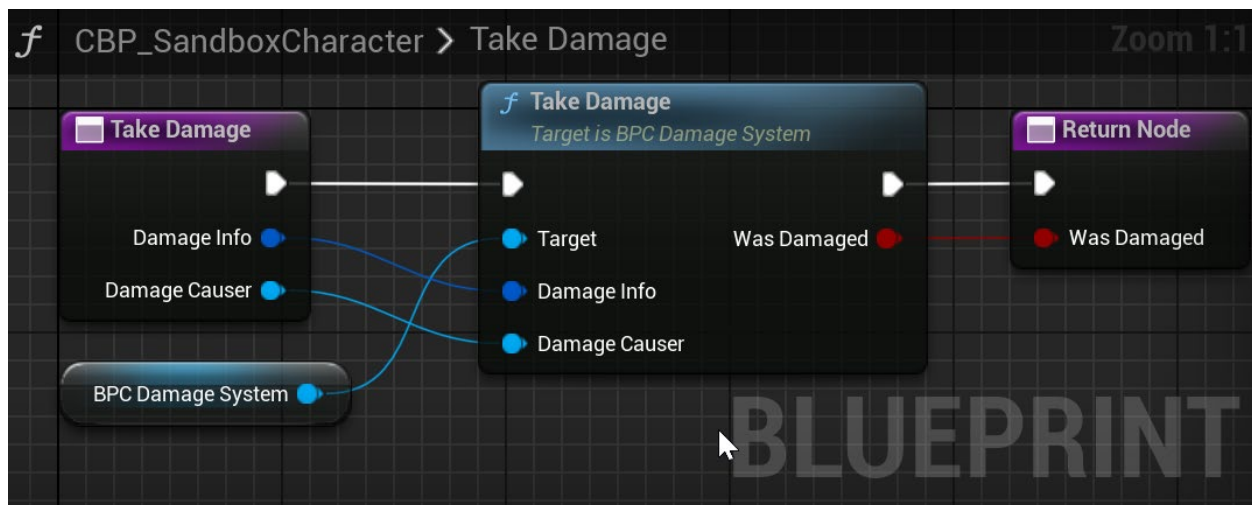
Εικόνα 24. Heal

3. TakeDamage

Το function δέχεται ως εισόδους τη μεταβλητή DamageInfo (Structure) και τον Damage Causer (Actor) και είναι υπεύθυνο για την εφαρμογή του Damage επιστρέφοντας τη μεταβλητή WasDamaged (Boolean). Η κλήση πραγματοποιείται από το blueprint του δράστη (π.χ. τον παίκτη, τον Agent ή ένα αντικείμενο όπως ο οπλισμός), και εκτελείται στο Blueprint του Actor που δέχεται το Damage, παρέχοντας το αναγκαίο reference κατά την κλήση του function. Σενάρια χρήσης περιλαμβάνουν την περίπτωση που ο παίκτης δέχεται ή προκαλεί Damage μέσω επιθέσεων με πυρομαχικά από τα Blueprints BP_NPC_Weapon_Base και BP_WeaponBase του οπλισμού, τις επιθέσεις του Agent τύπου Android από το Blueprint BP_Enemy_Android, καθώς και όταν ο παίκτης δέχεται Fall Damage από το Blueprint CBP_SandboxChartacter. Ενδεικτικά, η εικόνα 25 απεικονίζει την κλήση του function από το blueprint του οπλισμού των Agents εναντίον του παίκτη, κατά την οποία παρέχονται οι απαιτούμενες παράμετροι που χαρακτηρίζουν την επίθεση. Ακόμη, η υλοποίηση στα Blueprints των Actors φαίνεται στην εικόνα 26.



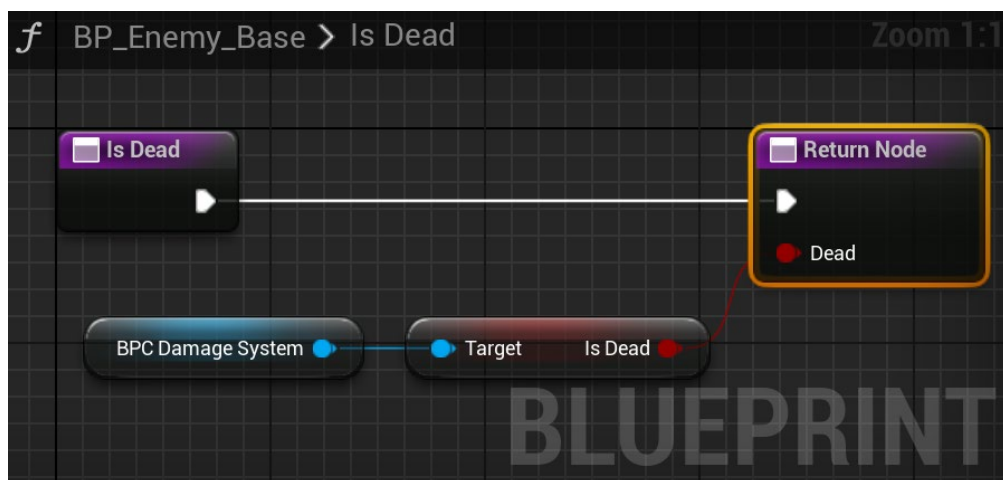
Εικόνα 25. TakeDamage (input parameters)



Εικόνα 26. TakeDamage

4. *IsDead*

Το function επιστρέφει τη μεταβλητή IsDead (Boolean) η οποία γίνεται True όταν οι πόντοι ζωής του Actor μηδενιστούν. Χρησιμοποιείται για την ανάκτηση της κατάστασης ενός Actor, προκειμένου να πυροδοτηθούν ορισμένες ενέργειες ή να αποτραπεί η εκτέλεση τους. Για παράδειγμα, χρησιμοποιείται στα Behavior Trees των Agents για να διακοπεί η επιθετική τους συμπεριφορά όταν ο παίκτης έχει εξουδετερωθεί. Ακόμα, αποτρέπει την εκτέλεση της ενέργειας Takedown όταν ένας Agent είναι ήδη εξουδετερωμένος, διασφαλίζοντας πως τα οπτικοακουστικά εφέ ή η απονομή πόντων XP που περιλαμβάνει δεν θα εκτελεστούν.



Εικόνα 27. IsDead

5. *GetTeamNumber*

Το function επιστρέφει τη μεταβλητή TeamNumber (Integer), η οποία που αναπαριστά τον αριθμό της ομάδας που έχει ανατεθεί σε έναν Actor. Χρησιμοποιείται σε ενέργειες που απαιτούν τον καθορισμό της ομάδας ενός Actor, για την εκτέλεση ενεργειών. Για παράδειγμα, αξιοποιείται στο function DamageResponse των Agents για να καθοριστεί η επόμενη ενέργειά του, στην περίπτωση που δεχθεί damage από έναν Actor διαφορετικής ομάδας. Ακόμη, χρησιμοποιείται εντός του AI Controller για να καθοριστεί εάν ένας Actor που εισέρχεται στην εμβέλεια ορατότητας ενός Agent ανήκει στην ίδια ομάδα με αυτόν, προκειμένου να εκτελεστεί η κατάλληλη συμπεριφορά.

6. *IsAttacking και DidPlayerBlock*

Το πρώτο function επιστρέφει τη μεταβλητή IsAttacking (Boolean) η οποία γίνεται True όταν ενεργοποιείται μία από τις επιθέσεις των Agents. Μπορεί να αξιοποιηθεί σε ενέργειες που σχετίζονται με την επιθετική κατάσταση ενός Agent, όπως για παράδειγμα να λειτουργήσει ως συνθήκη αποτροπής ενεργειών ενός Agent τη στιγμή που επιτίθενται. Ομοίως, το δεύτερο function επιστρέφει τη μεταβλητή IsBlocking (Boolean) η οποία γίνεται True όταν ο παίκτης πραγματοποιεί κίνηση αποφυγής και μπορεί να χρησιμοποιηθεί σε οποιοδήποτε σημείο απαιτείται η ανάγνωσή της. Τα functions αναπτύχθηκαν για μελλοντική ανάπτυξη λειτουργιών, ωστόσο στη συγκεκριμένη έκδοση του project η χρήση τους δεν ήταν απαραίτητη.

7. *ReseveAttackToken και ReturnAttackToken*

Αυτά τα functions παρέχουν έναν ευέλικτο τρόπο εκτέλεσης των αντίστοιχων functions του component BPC_DamageSystem από διαφορετικά Blueprints. Χρησιμοποιούνται για τον καθορισμό της εκτέλεσης επίθεσης ενός Agent και εντάσσονται στο πλαίσιο του TokenSystem, του συστήματος που σχετίζεται άμεσα με την υλοποίηση της επιθετικής συμπεριφοράς του Agent. Οι λεπτομέρειες του TokenSystem αναλύονται στο κεφάλαιο 8.

6. NPC Blueprints

Ο στόχος του παρόντος project είναι η ενσωμάτωση NPC χαρακτήρων, οι οποίοι διαθέτουν ένα σύνολο κοινών βασικών λειτουργιών και χαρακτηριστικών, ενώ ταυτόχρονα παρουσιάζουν ουσιώδεις διαφοροποιήσεις σε επιμέρους τομείς του παιχνιδιού. Οι διαφοροποιήσεις αυτές απορρέουν από τον τύπο κάθε Agent και εκτείνονται σε πολλαπλές διαστάσεις της υλοποίησής τους. Συγκεκριμένα, περιλαμβάνουν διαφορές στην εξωτερική εμφάνιση, την κινησιολογία, καθώς και στον εξοπλισμό που φέρουν. Περαιτέρω διαφοροποιήσεις αφορούν στον τρόπο συμπεριφοράς και αντίδρασης των Agents σε διάφορα δυναμικά γεγονότα εντός του παιχνιδιού, καθώς και στη φύση της αλληλεπίδρασής τους με τον παίκτη, όπως για παράδειγμα στον τρόπο με τον οποίο ανταποκρίνονται ή αντιμετωπίζονται από αυτόν. Επιπροσθέτως, διαφοροποιημένα είναι τα οφέλη που αποκομίζει ο παίκτης από την εξουδετέρωσή τους, όπως οι πόντοι εμπειρίας (XP) που αποδίδονται και τα αντικείμενα (Loot) που αποκτώνται. Τέλος, παρατηρούνται διαφορές και στο επίπεδο της αισθητικής αναπαράστασης, όπως στην αναπαραγωγή

μοναδικών οπτικοακουστικών εφέ, μεταξύ των οποίων περιλαμβάνονται εξατομικευμένα animations, ηχητικοί διάλογοι και ειδικά εφέ.

Οι διαφοροποιήσεις αυτές ενισχύουν την αίσθηση ποικιλίας και διαδραστικότητας στο παιχνίδι, προσφέροντας στον παίκτη μια δυναμική εμπειρία όπου οι επιλογές και οι αποφάσεις του έχουν άμεση επίδραση στην αλληλεπίδραση με τους χαρακτήρες και την εξέλιξη της συνολικής πλοκής. Παράλληλα, η υλοποίηση αυτών των διαφοροποιήσεων πρέπει να εξασφαλίζει την κλιμάκωση τους, επιτρέποντας την εύκολη προσθήκη νέων στοιχείων, όπως νέα όπλα για τους Agents, την ενσωμάτωση διαφορετικών ειδών εχθρών με μοναδικές δυνάμεις και αδυναμίες, και την προσθήκη νέων ηχητικών εφέ. Όλα αυτά πρέπει να επιτυγχάνονται χωρίς να επηρεάζεται αρνητικά η απόδοση του συστήματος, εξασφαλίζοντας τη βελτιστοποίηση των λειτουργιών και την αποδοτική χρήση των υπολογιστικών πόρων.

Για την επίτευξη των παραπάνω στόχων, δημιουργήθηκε η βασική κλάση Parent Blueprint (BP_EnemyBase), η οποία περιλαμβάνει όλες τις κοινές μεταβλητές και λειτουργίες που αφορούν στους Agents. Για παράδειγμα, αυτή η κλάση περιλαμβάνει τις ενέργειες που εκτελούνται κατά την εξουδετέρωση ενός Agent. Από αυτήν την κλάση Parent Blueprint δημιουργούνται υποκλάσεις Child Blueprints, οι οποίες κληρονομούν τη λειτουργικότητα του BP_EnemyBase και επιτρέπουν την τροποποίηση ή την επέκταση συγκεκριμένων λειτουργιών, προσφέροντας ευελιξία και τη δυνατότητα προσθήκης εξατομικευμένων χαρακτηριστικών. Η δομή αυτή επιτρέπει την επαναχρησιμοποίηση των λειτουργιών σε πολλαπλά Blueprints, ελαχιστοποιώντας την επανάληψη κώδικα και διευκολύνοντας τη διαχείριση του έργου και της αποσφαλμάτωσης. Ειδικότερα, από το BP_EnemyBase δημιουργήθηκαν τρεις βασικές υποκλάσεις για τους τύπους εχθρών BP_Enemy_Pistol, BP_Enemy_Rifle, και BP_Enemy_Android, οι οποίες προσθέτουν επιπλέον λειτουργίες που αφορούν τα ιδιαίτερα χαρακτηριστικά κάθε τύπου. Για παράδειγμα, τα animations που αναπαράγονται κατά την όπλιση ενός Agent τύπου Pistol διαφέρουν από εκείνα του τύπου Rifle, γεγονός που ενισχύει τον ρεαλισμό της συμπεριφοράς των χαρακτήρων.

Με την ύπαρξη αυτών των τριών βασικών υποκλάσεων, δημιουργήθηκαν επιπλέον Child Blueprints για κάθε τύπο Agent (π.χ. Pistol και Rifle), στα οποία προστέθηκαν πιο εξειδικευμένες λειτουργίες και ανατέθηκαν νέα Assets οπτικής διαφοροποίησης. Ενδεικτικά, τα BPc_Pistol_Male_1 και BPc_Pistol_Female_1 κληρονομούν όλες τις λειτουργίες του BP_EnemyBase και του BP_Enemy_Pistol, παρέχοντας ευελιξία στην επαναχρησιμοποίηση κώδικα και διευκολύνοντας τις μαζικές αλλαγές σε γενικές υποκλάσεις, ενώ ταυτόχρονα επιτρέπουν διαφοροποιήσεις και επεκτάσεις των λειτουργιών τους.

Η διαδικασία αυτή συνεχίζεται με τη δημιουργία πολλαπλών υποκλάσεων των Child Blueprints BP_Enemy_Pistol και BP_Enemy_Rifle, με κύρια διαφοροποίηση τους τη γεωμετρία του σκελετού τους, επιτρέποντας έτσι διαφορετικές αναπαραστάσεις για τους χαρακτήρες. Αυτή η διαφοροποίηση αφορά τις διαστάσεις του σκελετού, όπως για παράδειγμα τα άκρα, τα οποία είναι διαφορετικά για τους χαρακτήρες Male και Female, με αποτέλεσμα την κατάλληλη προσαρμογή των θέσεων στα όπλα, ανάλογα με το μέγεθος και τη μορφολογία του κάθε Agent (π.χ. διαφορετικές διαστάσεις άκρων).

Σε αυτό το κεφάλαιο θα αναλυθούν αναλυτικά τα χαρακτηριστικά των Agents και οι υποκατηγορίες τους. Στη συνέχεια, θα εξεταστεί η υλοποίηση του Parent Blueprint και των υποκλάσεων του,

συμπεριλαμβανομένων των λειτουργιών που αφορούν τον τρόπο κίνησης, τη συμπεριφορά και τον οπλισμό, προκειμένου να εξασφαλιστεί η ομαλή ενσωμάτωσή τους στα Behavior Trees του παιχνιδιού.

Χαρακτηριστικά κλάσεων

Οι Agents διακρίνονται σε δύο κύριες κατηγορίες: Human (Ranged) και Android, οι οποίες συνθέτουν το σύνολο των εχθρικών χαρακτήρων Enforcers και Sentinels, σύμφωνα με τον σχεδιασμό του Game Design Document. Οι Human Agents φέρουν όπλα εμβέλειας και διακρίνονται σε δύο υποκατηγορίες, Pistol και Rifle, βάσει του εξοπλισμού που διαθέτουν. Αντιθέτως, οι Androids χρησιμοποιούν όπλα κοντινής απόστασης, τα οποία ενεργοποιούνται αποκλειστικά κατά την εκτέλεση της επίθεσής τους.

Humans

Ο βασικός τρόπος επίθεσης των Human Agents βασίζεται στη χρήση όπλων τύπου Pistol ή Rifle. Οι διαφορές μεταξύ των δύο τύπων εστιάζουν τόσο στην ποσότητα του Damage που προκαλούν όσο και στα οπτικοακουστικά εφέ που συνοδεύουν τη χρήση των όπλων τους. Επιπλέον, οι συγκεκριμένοι Agents μπορούν να εξουδετερωθούν είτε μέσω συμβατικής μάχης είτε με την εκτέλεση της ειδικής ενέργειας Takedown, υπό την προϋπόθεση ότι δεν έχουν αντιληφθεί την παρουσία του παίκτη. Η Takedown αποτελεί βασική μηχανική του παιχνιδιού και συνεπάγεται την άμεση εξόντωση του στόχου, ανεξαρτήτως των πόντων ζωής που διαθέτει, όπως αναλύεται στο κεφάλαιο 9.

Οι Human Agents διακρίνονται περαιτέρω σε αρσενικούς και θηλυκούς χαρακτήρες, με διαφοροποιήσεις στην οπτική απεικόνιση και την κινησιολογία τους. Επιπροσθέτως, μπορούν να βρίσκονται είτε σε κατάσταση Unarmed, χωρίς όπλο, συνήθως όταν περιπολούν ή αναζητούν τον παίκτη, είτε σε κατάσταση Armed, με τα αντίστοιχα animations να προσαρμόζονται ανάλογα με την ταχύτητα και την κατεύθυνση της κίνησης (π.χ. Idle, Walk, Jog, Sprint).

Η ηχητική απεικόνιση ενισχύει περαιτέρω τον ρεαλισμό, καθώς τα ηχητικά εφέ διαφέρουν ανάλογα με το φύλο των χαρακτήρων, ενώ οι ήχοι των βημάτων είναι φυσικοί σε αντίθεση με τους Androids, στους οποίους αποδίδεται μεταλλικός ήχος. Επιπλέον, κατά τη λήψη Damage, οι Human χαρακτήρες συνοδεύονται από εφέ αίματος, ενώ στους Androids χρησιμοποιούνται εφέ ηλεκτρισμού.

Η παραπάνω πολυπλοκότητα απαιτεί την ενσωμάτωση διαφορετικών σκελετικών δομών για κάθε NPC, την ανάπτυξη ξεχωριστών animation blendspace ανάλογα με την κατηγορία και το φύλο, καθώς και την προσαρμογή της τοποθέτησης του οπλισμού στον αντίστοιχο σκελετό. Η υλοποίηση των ηχητικών εφέ συνδέεται άμεσα με την ανάπτυξη των Behavior Trees και την ενέργεια Takedown, και θα αναλυθεί εκτενέστερα στα σχετικά κεφάλαια.

Androids

Οι Androids χρησιμοποιούν ως κύρια μορφή επίθεσης την ενέργεια Aerial Attack, μια ειδική μηχανική κατά την οποία εκτοξεύονται προς τη θέση του παίκτη και επιτίθενται από κοντινή απόσταση με τη χρήση σπαθιού. Οι Agents αυτής της κατηγορίας είναι άθρωποι σε επιθέσεις με συμβατικό οπλισμό και μπορούν να εξουδετερωθούν αποκλειστικά μέσω της ενέργειας Takedown, η οποία καθίσταται εφικτή

υπό συγκεκριμένες συνθήκες: όταν ο παίκτης δεν έχει γίνει αντιληπτός, όταν έχει χρησιμοποιηθεί το αντικείμενο DataWorm για την αδρανοποίησή τους ή όταν έχει αποφευχθεί επιτυχώς η Aerial Attack. Η υλοποίηση αυτών των μηχανισμών περιγράφεται αναλυτικά στο κεφάλαιο 8.

Εκτός από τις αισθητικές και ηχητικές διαφοροποιήσεις σε σχέση με τους Humans, οι Androids παρουσιάζουν και διακριτή απεικόνιση στο Minimap του UI. Συγκεκριμένα, όταν ο παίκτης ενεργοποιεί τη λειτουργία σάρωσης για την ανίχνευση κοντινών εχθρών, οι Androids εμφανίζονται με μοβ ένδειξη, ενώ οι Humans με κόκκινη. Επιπλέον, το Loot που αποδίδεται μετά την εξουδετέρωση διαφέρει: οι Androids προσφέρουν κυρίως αντικείμενα όπως DataWorms, χρήσιμα για την αδρανοποίησή τους, καθώς και Shards, το νόμισμα του παιχνιδιού, ενώ οι Humans παρέχουν κυρίως πυρομαχικά.

6.1. Parent Blueprint

Η υλοποίηση της βασικής κλάσης εχθρικών χαρακτήρων πραγματοποιήθηκε μέσω της δημιουργίας του Parent Blueprint με την ονομασία BP_Enemy_Base. Ως βάση χρησιμοποιήθηκε η προκαθορισμένη κλάση χαρακτήρα του πακέτου Third Person που παρέχεται από την Unreal Engine, η οποία ενσωματώνει θεμελιώδη στοιχεία απαραίτητα για την ανάπτυξη ενός λειτουργικού εχθρικού Agent. Αυτά περιλαμβάνουν το CharacterMovementComponent, το οποίο επιτρέπει την κίνηση του Actor στον χώρο, καθώς και το Skeletal Mesh, το οποίο αξιοποιήθηκε ως οδηγός για την ενσωμάτωση και παραμετροποίηση διαφορετικών σκελετών, επιτρέποντας την οπτική διαφοροποίηση των Human Agents.

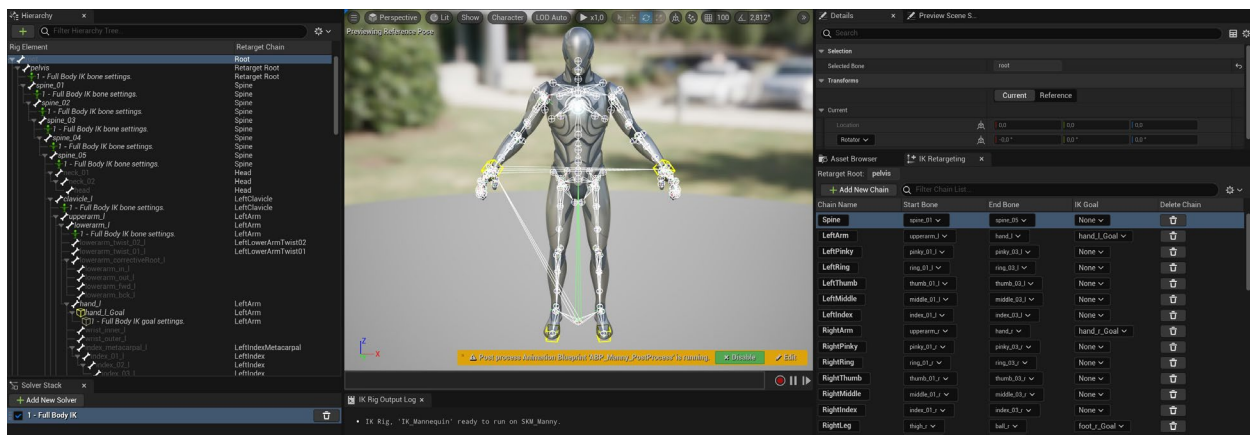
Για την επίτευξη της συμβατότητας μεταξύ διαφορετικών σκελετών και animations, εφαρμόστηκε η τεχνική IK Retargeting, η οποία επιτρέπει τη μεταφορά ή προσαρμογή κινήσεων από έναν χαρακτήρα σε άλλον, βασισμένη σε Inverse Kinematics (IK). Προκειμένου να επιτευχθεί συνοχή και ομοιομορφία στις κινήσεις, συγκεντρώθηκαν σε ένα βοηθητικό project όλα τα animations που χρησιμοποιούνται στο σύστημα ελεύθερης κινήσιολογίας (Locomotion) των Agents και έγινε η διαδικασία retargeting από τους πρωτογενείς σκελετούς στους οποίους είχαν αρχικά σχεδιαστεί, προς τον βασικό σκελετό SK_Mannequin του Unreal Engine 5. Ακολούθως, δημιουργήθηκαν τα κατάλληλα Blendspaces για κάθε κατηγορία χαρακτήρων (π.χ. Male, Female, Android), καθώς και για τις επιμέρους καταστάσεις όπως Armed και Unarmed. Τα Blendspaces ενσωματώθηκαν σε ένα ενοποιημένο Animation Blueprint με την ονομασία ABP_Enemies_Merged, το οποίο χρησιμοποιείται καθολικά από όλους τους NPC χαρακτήρες. Εντός του Animation Blueprint αναπτύχθηκαν μηχανισμοί για την ανάγνωση κρίσιμων μεταβλητών και τη διευκόλυνση της επικοινωνίας με τα αντίστοιχα Blueprints των Actor και τον AI Controller τους.

Επιπλέον, σε κάθε Actor προστέθηκαν οι αντίστοιχοι σκελετοί, εντός των οποίων καθορίστηκαν τα σημεία εμφάνισης και τοποθέτησης του οπλισμού του εκάστοτε Agent, ανάλογα με την κατηγορία και τον τύπο του. Για λόγους βελτιστοποίησης της δομής, επαναχρησιμοποίησης του κώδικα και οργανωμένης διαχείρισης, τα Assets που σχετίζονται με τις επιθέσεις των Agents αναπτύχθηκαν σε ξεχωριστό Blueprint, σύμφωνα με την αρχιτεκτονική Parent-Child Blueprint, ακολουθώντας την ίδια φιλοσοφία σχεδίασης που εφαρμόστηκε και στους εχθρικούς χαρακτήρες.

6.2. IK Rig & IK Retargeting

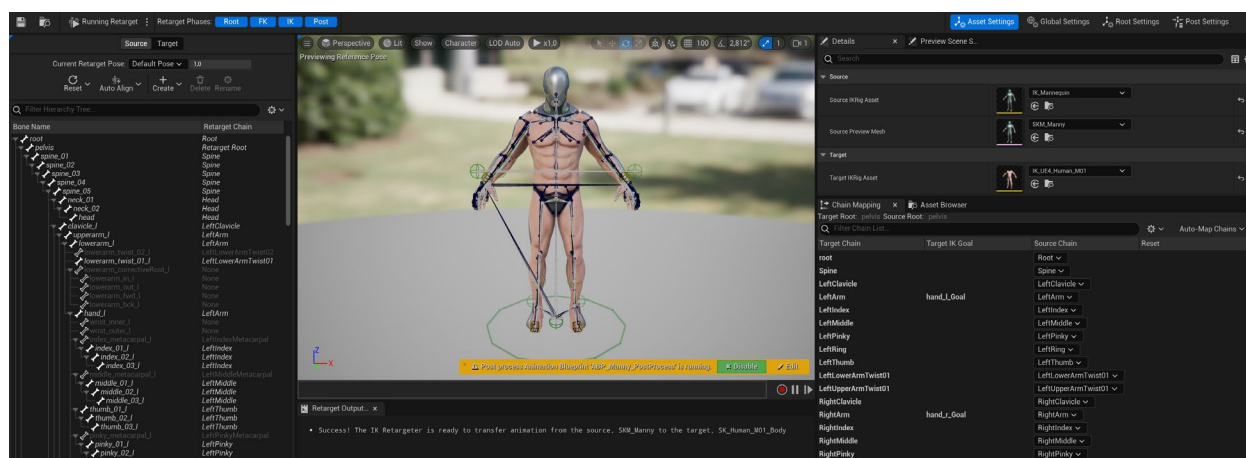
Λόγω της ανάγκης ενσωμάτωσης πολλαπλών χαρακτήρων με διαφορετικούς σκελετούς και της επιθυμίας αποφυγής αντιγραφής και επαναχρησιμοποίησης των ίδιων κινήσεων (animations) για κάθε διαφορετικό σκελετό, υιοθετήθηκε η τεχνική του retargeting σε πραγματικό χρόνο (real-time IK retargeting). Η προσέγγιση αυτή επιτρέπει σε κάθε Actor να χρησιμοποιεί τον βασικό σκελετό SK_Mannequin του Unreal Engine 5, ο οποίος παραμένει αόρατος κατά το gameplay, ως πηγή αναφοράς για την κίνηση, ενώ οι διαφορετικοί σκελετοί που έχουν ανατεθεί στους NPC χαρακτήρες ακολουθούν τις κινήσεις του μέσω του συστήματος IK Retargeter (Epic Games, n.d.).

Για τη χρήση της τεχνικής IK Retargeting, δημιουργήθηκε ένα IK Rig Asset για κάθε χαρακτήρα που συμμετέχει στη διαδικασία, τόσο για τον χαρακτήρα πηγής (Source Character), από τον οποίο προέρχονται τα animations, όσο και για τον χαρακτήρα στόχο (Target Character), στον οποίο εφαρμόζονται. Το IK Rig ορίζει το σύνολο των αρθρώσεων και των δομικών στοιχείων (joints and bones) ενός σκελετού και χρησιμοποιείται για την καθοδήγηση της κίνησης. Η τεχνική αυτή παρέχει τη δυνατότητα αναπαραγωγής πανομοιότυπων κινήσεων σε χαρακτήρες με ετερογενή ανατομική δομή, ανεξαρτήτως αναλογιών ή αρχιτεκτονικής σκελετού. Στο πλαίσιο της υλοποίησης, εντός κάθε IK Rig ορίστηκε το σημείο ρίζας (root) για κάθε χαρακτήρα, προκειμένου η κίνηση να μεταφέρεται με αναλογικό και ρεαλιστικό τρόπο. Παράλληλα, καθορίστηκαν τα άκρα και τα υπόλοιπα σημαντικά τμήματα του σώματος που συμμετέχουν στο retargeting, τόσο για το IK Rig της πηγής όσο και του στόχου. Η διαδικασία αυτή προσομοιάζει με την οργάνωση rigging σε λογισμικά όπως το Autodesk MotionBuilder ή το Maya, με τη διαφορά ότι στην Unreal Engine οι κινήσεις ανατίθενται μέσω Retarget Chains αντί μεμονωμένων οστών. Οι αλυσίδες αυτές εξασφαλίζουν ευελιξία κατά την εφαρμογή κινήσεων μεταξύ χαρακτήρων με εντελώς διαφορετικά συστήματα αρθρώσεων. Ενδεικτικά, η εικόνα 28 απεικονίζει ένα IK Rig Asset για τον χαρακτήρα πηγής.



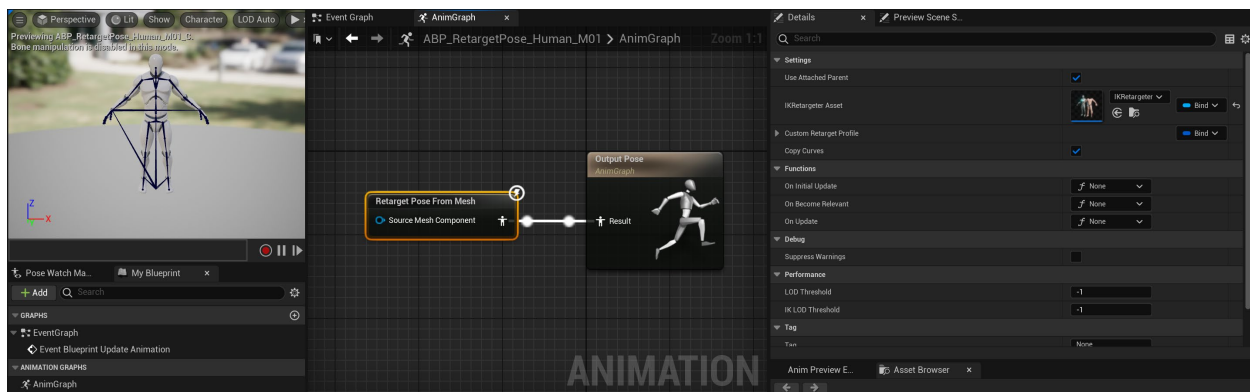
Εικόνα 28. IK Rig & Retarget Chains

Βάσει της παραπάνω αρχιτεκτονικής, δημιουργήθηκαν Retarget Chains για όλα τα κύρια αλλά και επιλεγμένα δευτερεύοντα άκρα των χαρακτήρων. Στη συνέχεια, για κάθε νέο σκελετό δημιουργήθηκε ένα IK Retargeter Asset, στο οποίο ορίστηκε ως Source IK Rig το IK_Mannequin και ως Target IK Rig το αντίστοιχο IK Rig του επιμέρους σκελετού. Τέλος, για τη λειτουργική σύνδεση του IK Retargeting εντός του μηχανισμού του παιχνιδιού, προστέθηκε στο Parent Blueprint (BP_Enemy_Base) ένα επιπλέον Skeletal Mesh Component, το οποίο λειτουργεί ως σημείο τοποθέτησης του εκάστοτε σκελετού που προστίθεται σε κάθε Child Blueprint. Σε κάθε τέτοιο component, ανατίθεται ένα ξεχωριστό Animation Blueprint, το οποίο αξιοποιεί το IK Retargeter Asset μέσω του node Retarget Pose From Mesh, όπως φαίνεται στην εικόνα 30. Με αυτόν τον τρόπο διατηρείται ενιαία βάση κινήσεων για όλους τους Agents, ανεξαρτήτως σκελετικής δομής ή φύλου.



Εικόνα 29. IK Retargeter

Αξίζει να σημειωθεί ότι η μέθοδος αυτή συνεπάγεται με αυξημένο υπολογιστικό κόστος, καθώς απαιτεί τη χρήση πολλαπλών IK Retargeters και επιβάλει την επεξεργασία σε πραγματικό χρόνο για κάθε Actor που εμφανίζεται εντός του παιχνιδιού. Ως αποτέλεσμα, σε περιπτώσεις όπου τα διαθέσιμα Assets επιτρέπουν τη χρήση ενιαίου σκελετού χωρίς την ανάγκη real-time retargeting, η απόδοση του συστήματος μπορεί να βελτιωθεί σημαντικά.

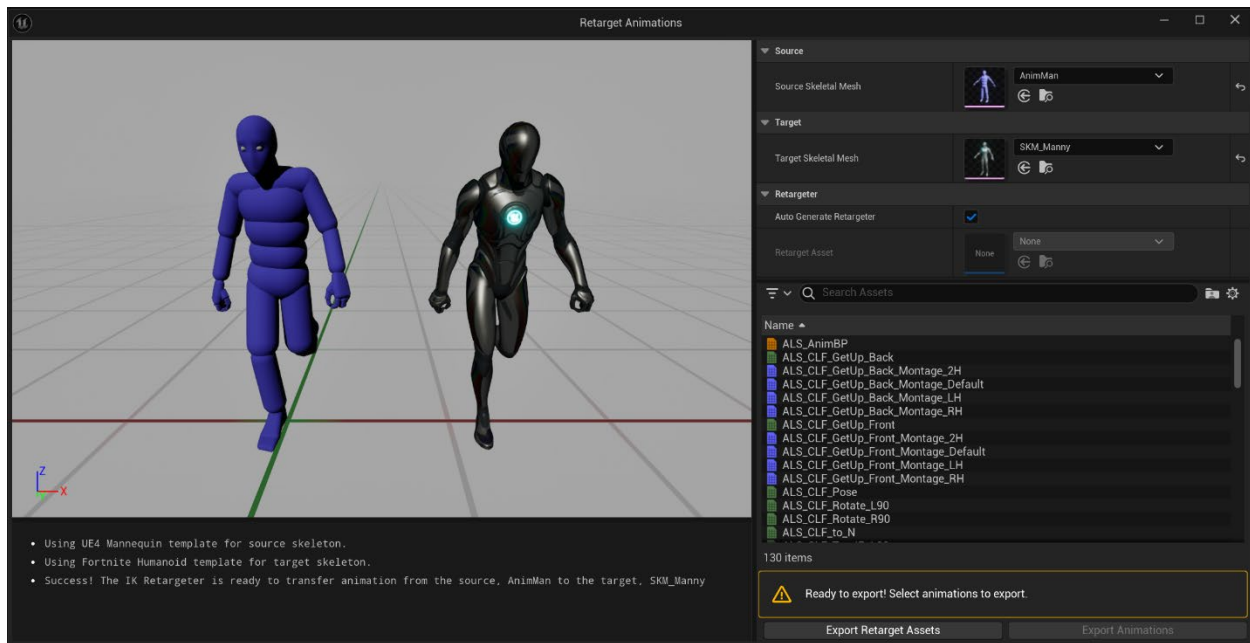


Εικόνα 30. Retarget Pose Animation Blueprint

6.3. Animations και Blendspaces

Για την κάλυψη των αναγκών του έργου όσον αφορά στην κίνηση χαρακτήρων διαφορετικού τύπου (π.χ. Male, Female, Android, σε καταστάσεις Aware ή Unaware), αξιοποιήθηκε το σύστημα Blendspaces της Unreal Engine 5. Με στόχο τη διατήρηση του κόστους ανάπτυξης σε χαμηλά επίπεδα, πραγματοποιήθηκε συλλογή και επεξεργασία animation assets από πηγές όπως τα Mixamo και Fab, καθώς και από δωρεάν projects όπως το Lyra Starter Project.

Πολλά από τα συλλεχθέντα animations δημιουργήθηκαν είτε για διαφορετικούς σκελετούς είτε σε παλαιότερες εκδόσεις της μηχανής Unreal (π.χ. UE 4.7). Για την ενσωμάτωσή τους στο κύριο project, δημιουργήθηκε ένα δευτερεύον project στην έκδοση UE 5.4.4, εντός του οποίου πραγματοποιήθηκε η μετατροπή (retargeting και reimport) των animations ώστε να είναι συμβατά με τον ενιαίο σκελετό οδηγό SK_Mannequin. Η διαδικασία του retargeting υποστηρίζεται εγγενώς από τη μηχανή και, όπως και η μέθοδος IK Retargeting, προϋποθέτει την ύπαρξη ενός αρχικού (source) και ενός τελικού (target) skeletal mesh, όπως παρουσιάζεται ενδεικτικά στην Εικόνα 31 (Unreal, n.d.). Ως αποτέλεσμα, σε αυτό το project δημιουργήθηκε μία συλλογή που περιλάμβανε όλους τους απαιτούμενους σκελετούς και τα αντίστοιχα animation assets. Με τη χρήση της λειτουργίας migration, ο φάκελος με τα μετασχηματισμένα animations ενσωματώθηκε στο κύριο project, όπου και χρησιμοποιήθηκε για τη δημιουργία των αντίστοιχων Blendspace assets.



Εικόνα 31. Animation retargeting

Η κίνηση των Agents στο χώρο πραγματοποιείται μέσω του node AIMoveTo και του component CharacterMovementComponent. Το πρώτο αποτελεί τον κόμβο που παρέχει η Unreal Engine για την αυτόματη πλοήγηση χαρακτήρων, ενώ το δεύτερο διαχειρίζεται λεπτομερώς τις φυσικές παραμέτρους κίνησης, όπως το περπάτημα, άλμα, πτώση και ολίσθηση. Μέσω των μεταβλητών του CharacterMovementComponent, όπως MaxWalkSpeed, JumpZVelocity, MaxWalkSpeedCrouched κ.ά., καθορίζεται η ταχύτητα και το είδος της κίνησης του χαρακτήρα. Οι τιμές αυτών των παραμέτρων διαβάζονται σε πραγματικό χρόνο από το Animation Blueprint και καθορίζουν ποιο animation θα αναπαραχθεί μέσω του αντίστοιχου Blendspace. Για παράδειγμα, μία μεταβολή της τιμής του MaxWalkSpeed από 150 σε 500 προκαλεί την αλλαγή του animation από Walk σε Run. Στο πλαίσιο του συγκεκριμένου project, χρησιμοποιήθηκαν animations τύπου InPlace, τα οποία αποδίδουν την ψευδαίσθηση της κίνησης χωρίς να μετακινούν τον χαρακτήρα στον χώρο. Η φυσική μετατόπιση πραγματοποιείται από το σύστημα πλοήγησης της Unreal Engine, και όχι από το ίδιο το animation (σε αντίθεση με τα RootMotion animations).

Συλλεχθέντα Animations και Χρήση

Τα animations που επιλέχθηκαν οργανώθηκαν ανάλογα με το πλαίσιο χρήσης τους:

- *Idle Animations (για Female, Male, Android χαρακτήρες):* Χρησιμοποιούνται όταν ο χαρακτήρας δεν εκτελεί κάποια ενέργεια (π.χ. στέκεται ακίνητος και αναπνέει ή κουνάει ελαφρώς τα χέρια του), προκειμένου να αποφεύγεται η απόλυτη ακινησία.

- *Idle Breaks*: Χρησιμοποιούνται ως σύντομες, αυθόρμητες κινήσεις που γίνονται περιοδικά κατά τη διάρκεια ενός Idle Animation για να προσθέσουν ποικιλία και ρεαλισμό (π.χ. ένας χαρακτήρας είναι σε Idle Animation και ξαφνικά ξύνει το κεφάλι του ή κοιτάζει γύρω του).
- *Unaware Locomotion Animations*: Αποτελούν συλλογές από Animations για τη δημιουργία Blendspaces. Περιλαμβάνουν όλα τα Animations που καλύπτουν τις πιθανές κατευθύνσεις της κίνησης του Agent στον χώρο όπως μπροστά, πίσω, αριστερά, δεξιά και διαγώνια. Χρησιμοποιούνται στη δημιουργία τριών Blendspaces:
 - BS_F_Patrol για Female Agents
 - BS_M_Patrol για Male Agents
 - BS_A_8W για Android Agents

Τα Blendspaces αυτά χαρακτηρίζονται την ελευθερία των χεριών κατά τις κινήσεις, καθώς οι Agents δεν κρατούν κάποιο όπλο ή άλλο αντικείμενο κατά την αναπαραγωγή τους. Χρησιμοποιούνται συνήθως σε καταστάσεις χαμηλής επιθετικότητας, όταν η παρουσία του παίκτη δεν έχει γίνει αντιληπτή.

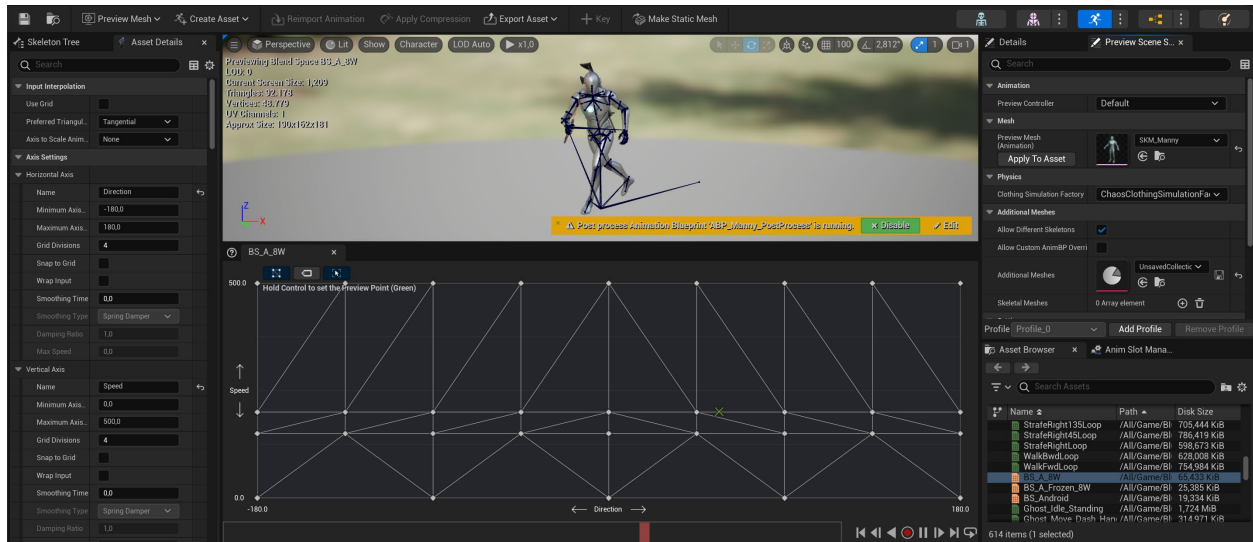
- *Aware Locomotion Animations*: Αφορούν καταστάσεις στις οποίες ο χαρακτήρας έχει εντοπίσει τον παίκτη και κρατά όπλο. Χρησιμοποιούνται στη δημιουργία δυο Blendspaces:
 - BS_M_8W_Pistol για τους Agents με οπλισμό Pistol
 - BS_M_8W_Rifle_AIM για τους Agents με οπλισμό Rifle

Τα συγκεκριμένα Blendspaces χρησιμοποιούνται από όλους τους Ranged χαρακτήρες ανεξαρτήτως φύλου, σε καταστάσεις υψηλής επιθετικότητας. Στην περίπτωση των Androids, χρησιμοποιείται το ίδιο BS_A_8W Blendpace.

- *Frozen Android Animations*: Τα animations που χρησιμοποιήθηκαν για το Blendpace των Agents τύπου Android (BS_A_Frozen_8W), το οποίο ενεργοποιείται όταν ένα Android δέχεται επίθεση μέσω του αντικειμένου DataWorm, απεικονίζοντας μια αδρανοποιημένη και δυσλειτουργική κινησιολογία.

Δημιουργία Blendspaces

Ενδεικτικά, για το BS_A_8W, δημιουργήθηκε ένα Blendspace Asset δύο διαστάσεων με παραμέτρους τις μεταβλητές Direction (float) και Speed (float). Οι κατευθύνσεις αντιστοιχούν σε τιμές (π.χ. 90 για δεξιά, 135 για αριστερά, κ.ο.κ.) και τα επιμέρους animations τοποθετήθηκαν σε σημεία που αντιπροσωπεύουν ταχύτητες όπως Idle (0), Walk (150), Jog (200) και Run (500). Οι τιμές των μεταβλητών αυτών ενημερώνονται από το CharacterMovementComponent και το Animation Blueprint σε πραγματικό χρόνο, οδηγώντας στη δυναμική μίξη animations ανάλογα με την κατεύθυνση και την ταχύτητα του χαρακτήρα. Ακολουθώντας παρόμοια διαδικασία, σχεδιάστηκαν και τα υπόλοιπα Blendspaces του project.



Εικόνα 32. Blendspace (BS_A_8W)

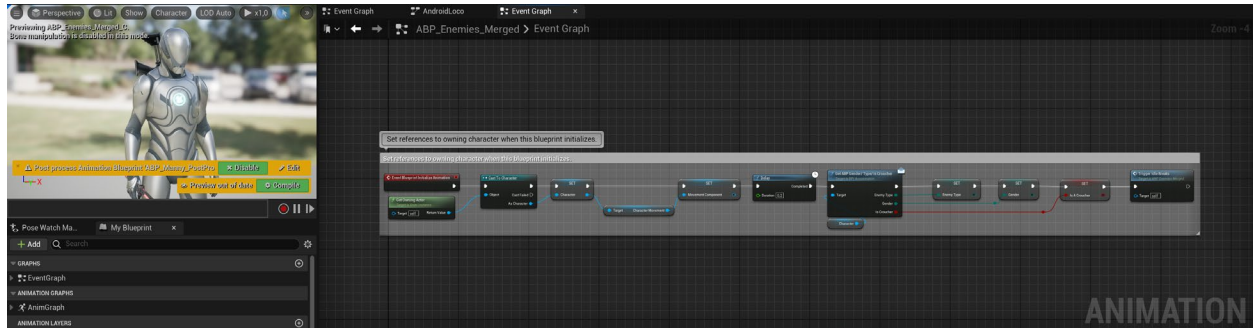
6.4. Animation blueprint

Για την ανάθεση των Blendspaces στους Agents διαφορετικού τύπου, τον ορισμό των μεταβάσεων μεταξύ διαφορετικών καταστάσεων και της γενική λειτουργία του συστήματος των animations, χρησιμοποιήθηκε το Animation Blueprint του προεπιλεγμένου χαρακτήρα Mannequin από το πακέτο ThirdPerson του Unreal Engine (Unreal, n.d.). Πάνω σε αυτό, προστέθηκαν επιπλέον λειτουργίες προσαρμοσμένες στις ανάγκες της υλοποίησης. Το συγκεκριμένο Blueprint περιλαμβάνει ένα τυπικό State Machine με ανεπτυγμένη τη λογική της μετάβασης μεταξύ των καταστάσεων Locomotion και ενεργειών, όπως Jump και Land. Ακόμη, περιλαμβάνει το τυπικό Event Graph που διατίθεται σε κάθε Blueprint και το AnimGraph. Το πρώτο είναι υπεύθυνο για τη λογική του Animation Blueprint που σχετίζεται με την αλληλεπίδραση του χαρακτήρα με το υπόλοιπο παιχνίδι και τη ροή του παιχνιδιού. Στο πλαίσιο αυτής της διαδικασίας, δημιουργούνται τα events και ενημερώνονται οι μεταβλητές που επηρεάζουν το πώς και πότε θα ενεργοποιηθούν τα Animations ή οι καταστάσεις στο Anim Graph, όπως η ταχύτητα, η κατεύθυνση ή η κατάσταση του χαρακτήρα (π.χ. αν τρέχει ή αναπηδά). Το δεύτερο χρησιμοποιείται αποκλειστικά με την εκτέλεση των Animations, τη μετάβαση και την ανάμειξή τους. Στο Anim Graph δημιουργείται η λογική που καθορίζει ποιες κινήσεις θα εκτελούνται και πότε, με βάση την κατάσταση του χαρακτήρα, περιλαμβάνει Blendspaces, State Machines και ορίζονται οι κανόνες μετάβασης των καταστάσεων.

6.4.1. Event Graph

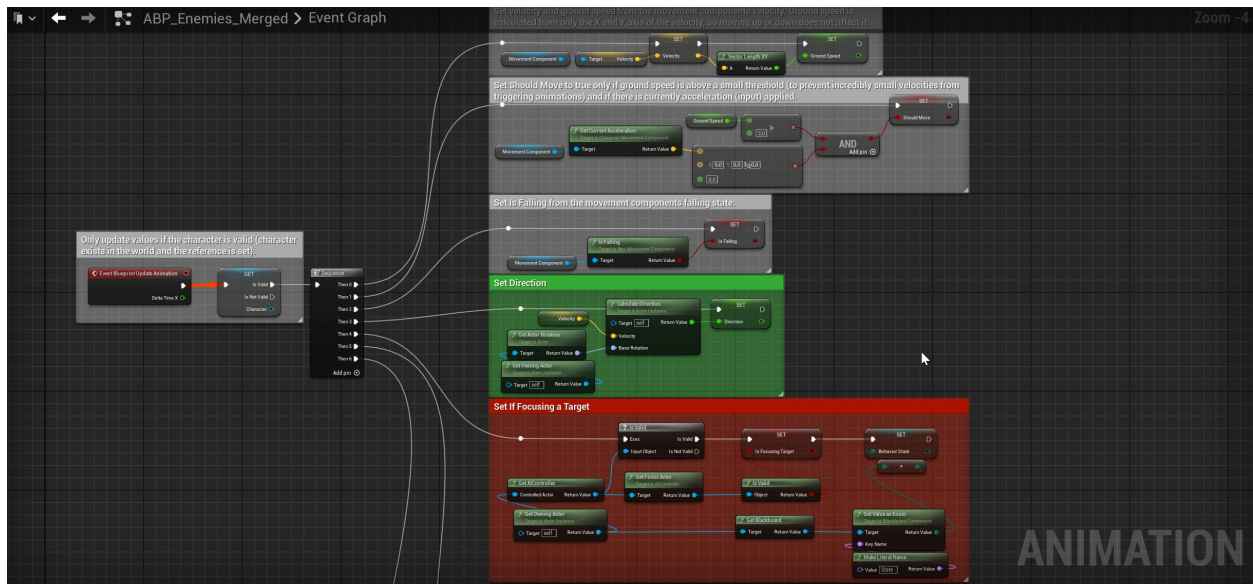
Στο Event Graph υλοποιούνται δύο βασικά Events. Το πρώτο αφορά στο Event *Blueprint Initialize*, το οποίο εκτελείται κατά την ενεργοποίηση του Agent (π.χ. εκκίνηση του παιχνιδιού ή spawn) για την αρχική ανάγνωση και αποθήκευση των απαραίτητων μεταβλητών. Συγκεκριμένα, αποθηκεύεται το

reference του Agent που χρησιμοποιεί το Animation Blueprint και το CharacterMovementComponent, προκειμένου να αποκτηθεί πρόσβαση στις κατάλληλες λειτουργίες. Ακόμη, λαμβάνονται οι μεταβλητές Enemy Type (Enum) και Gender (Enum), για την αρχικοποίηση του Blueprint βάσει των χαρακτηριστικών του Agent. Το Event απεικονίζεται στην εικόνα 33.

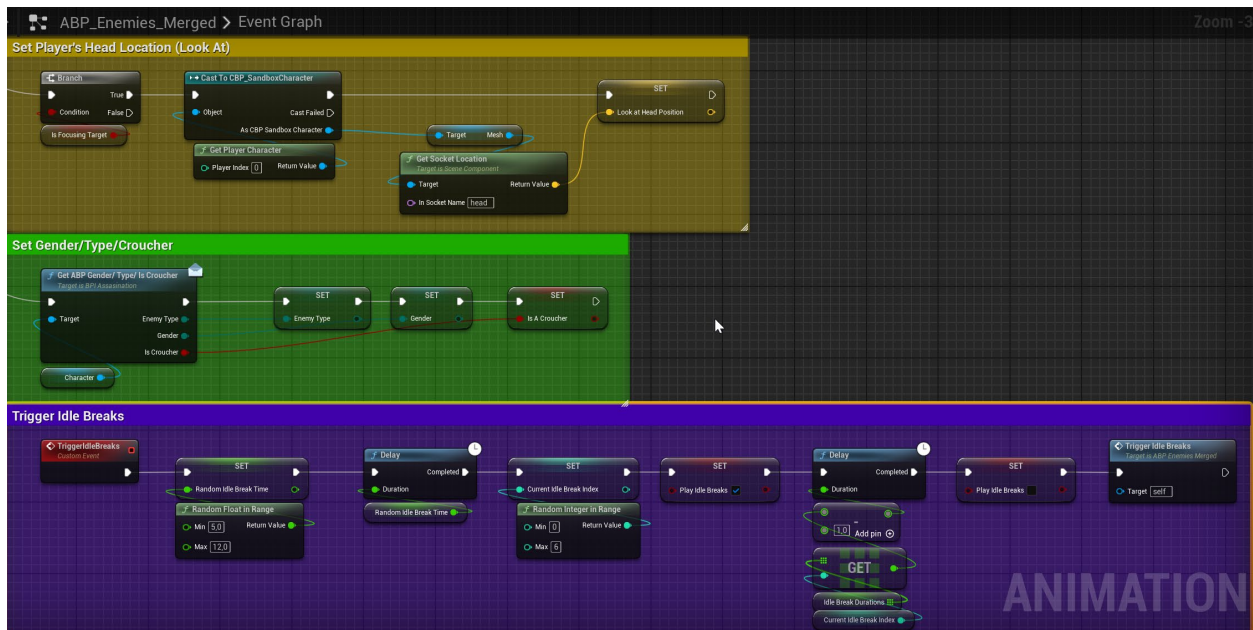


Εικόνα 33. Event Blueprint Initialize

Στη συνέχεια χρησιμοποιήθηκε το Event *Blueprint Update Animation* για την ενημέρωση μεταβλητών του χαρακτήρα πριν την εκτέλεση των Animations. Το Event εκτελείται σε κάθε frame του παιχνιδιού, ώστε να υπάρχει ενημέρωση σε πραγματικό χρόνο. Σε αυτό το στάδιο γίνεται ανάγνωση και ορισμός των Speed και Direction από το CharacterMovementComponent για την εισαγωγή τους στα Blendspaces. Ακόμα, αποθηκεύονται οι μεταβλητές IsFocusingTarget (Boolean) και BehaviorState (Enum) από το AI Component, για την ανάμειξη διαφορετικών Blendspaces βάσει των καταστάσεων του Behavior Tree. Επιπλέον, ορίζεται η μεταβλητή LookAtHeadPosition (Vector) ως τοποθεσία εστίασης, όπου ο Agent πρέπει να στρίψει το κεφάλι του. Η μεταβλητή χρησιμοποιείται προκειμένου ένας Agent που έχει αντιληφθεί τον παίκτη να κοιτάζει προς την κατεύθυνση του κεφαλιού του. Τέλος, προστέθηκε το Custom Event TriggerIdleBreaks, κατά το οποίο όταν ένας Agent βρίσκεται σε κατάσταση ακινησίας, ένα από τα IdleBreak Animations επιλέγεται τυχαία και αναπαράγεται. Το event αυτό εκτελείται σε τυχαία χρονικά διαστήματα και κατά την ολοκλήρωσή του, ο Agent επιστρέφει σε κατάσταση Idle. Τα σχετικά γραφήματα παρουσιάζονται στις εικόνες 34 και 35.



Εικόνα 34. Event Blueprint Update Animation (1)

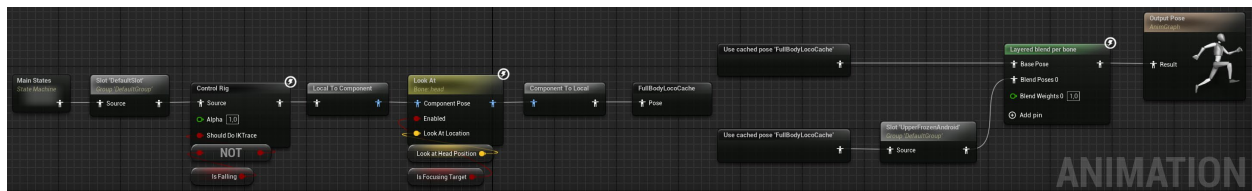


Εικόνα 35. Event Blueprint Update Animation (2)

6.4.2. Anim Graph

Στο Anim Graph ορίστηκε η βασική δομή μέσω της οποίας το Blueprint παράγει το τελικό Animation που θα αναπαραχθεί (Output Pose). Η διαδικασία ξεκινά με την έξοδο του State Machine που φέρει την ονομασία "MainStates" και περιλαμβάνει το πλήρες σύνολο κινήσεων Locomotion, καθώς και τις ενέργειες Jump και Land, μαζί με τους αντίστοιχους κανόνες μετάβασης μεταξύ τους. Η έξοδος τροφοδοτείται στον κόμβο ControlRig ο οποίος ενεργοποιείται όταν ο χαρακτήρας βρίσκεται στο έδαφος και προσφέρει τη λειτουργία Leg Inverse Kinematics (LegIK). Η λειτουργία αυτή επιτρέπει στους Agents να κινούνται ρεαλιστικά σε διάφορους τύπους εδάφους, διασφαλίζοντας ότι τα πόδια τους τοποθετούνται με ακρίβεια και σταθερότητα, ανεξαρτήτως της φύσης της επιφάνειας. (π.χ. σε κλίσεις ή ανώμαλα εδάφη).

Στη συνέχεια χρησιμοποιείται το node LookAt, το οποίο τίθεται σε λειτουργία όταν έχει οριστεί κάποιο σημείο εστίασης, δηλαδή όταν η τιμή της μεταβλητή IsFocusingTarget έχει την τιμή true. Το node επιτρέπει την περιστροφή του κεφαλιού ή άλλων μελών του σώματος του χαρακτήρα προς μια συγκεκριμένη κατεύθυνση στον τρισδιάστατο χώρο, χωρίς να απαιτείται η δημιουργία επιπλέον animations. Με αυτόν τον τρόπο, όταν ο Agent αντιλαμβάνεται την παρουσία του παίκτη, περιστρέφει το κεφάλι του αυτόματα προς την τοποθεσία της μεταβλητής LookAtHeadPosition, η οποία έχει οριστεί να αντιστοιχεί στη θέση της κεφαλής του παίκτη στον χώρο.



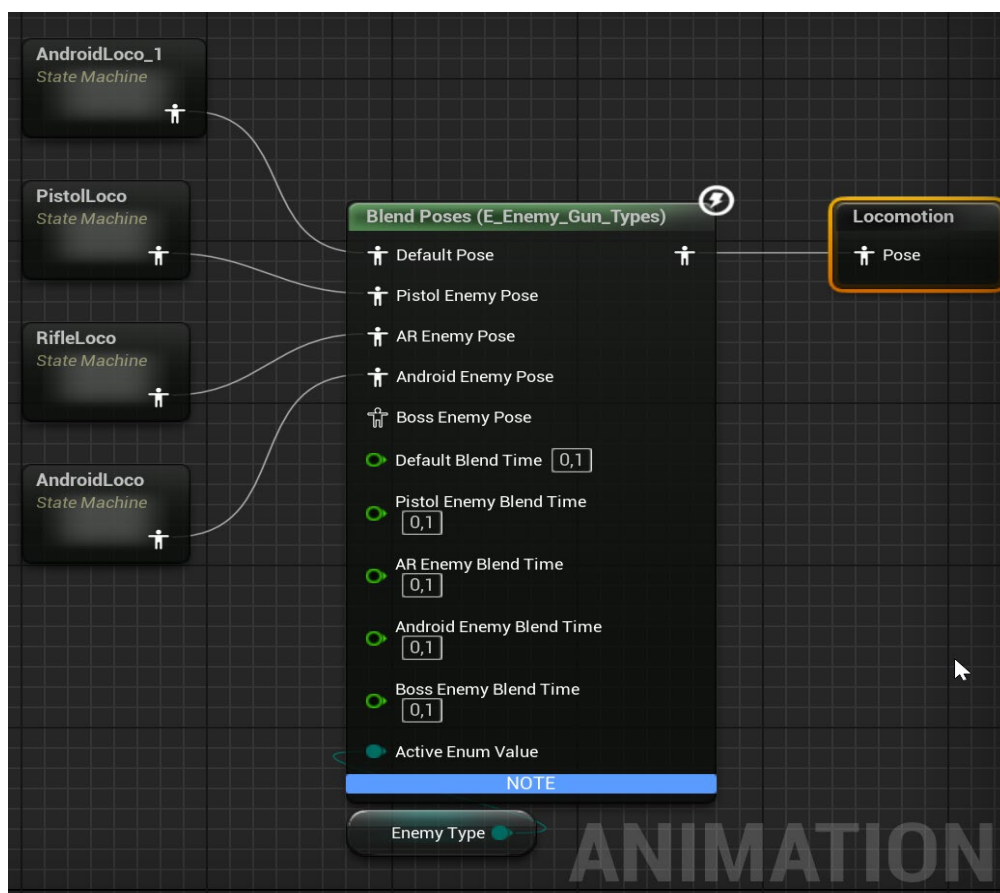
Εικόνα 36. Anim Graph

Το αποτέλεσμα των παραπάνω διαδικασιών αποθηκεύεται σε μια προσωρινή στάση (Cached Pose) με το όνομα "FullBodyLocoCache". Το Cached Pose αποθηκεύει το αποτέλεσμα ενός animation ή ενός συνδυασμού animations σε μια δεδομένη χρονική στιγμή, και μπορεί να επαναχρησιμοποιηθεί σε άλλα σημεία του Anim Graph, είτε κατά τη μετάβαση μεταξύ καταστάσεων είτε στον συνδυασμό διαφορετικών επιπέδων animation (Animation Layers). Η έξοδος του FullBodyLocoCache στη συνέχεια συνδυάζεται με το Slot "UpperFrozenAndroid" στον κόμβο Layer Blend per Bone, και το αποτέλεσμα αυτής της σύνθεσης αποτελεί το τελικό animation του Agent. Το συγκεκριμένο Slot έχει σχεδιαστεί ώστε να δέχεται animations που εφαρμόζονται αποκλειστικά σε επιμέρους περιοχές του σώματος, χωρίς να επηρεάζουν το συνολικό animation του χαρακτήρα. Στο πλαίσιο του συγκεκριμένου project, το Slot χρησιμοποιείται για την αναπαραγωγή ενός animation που επηρεάζει μόνο την περιοχή του κεφαλιού σε NPCs τύπου Android, κατά την ολοκλήρωση της επίδρασης του αντικειμένου. Η χρήση του Layer Blend Per Bone καθιστά δυνατή την ανάμειξη animations καθορίζοντας επακριβώς το οστό του σκελετού

(bone) από το οποίο και κάτω θα εφαρμοστεί η σύνθεση, επιτρέποντας έτσι την ταυτόχρονη ύπαρξη animation σε διαφορετικά μέρη του σώματος χωρίς να προκαλούνται συγκρούσεις.

6.5. Locomotion

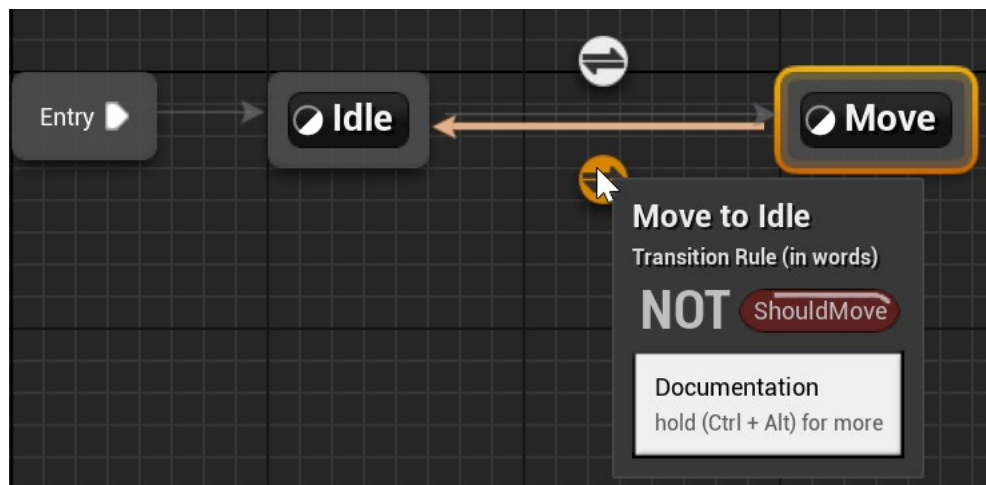
Μέσα στο State Machine με την ονομασία “MainStates” ενσωματώνεται το Cached Pose "Locomotion", το οποίο περιλαμβάνει το πλήρες σύνολο κινησιολογικών ρουτινών για κάθε τύπο Agent. Η επιλογή του κατάλληλου State Machine πραγματοποιείται μέσω του node BlendPosesByEnum, καθοδηγούμενου από τη μεταβλητή EnemyType, η οποία ορίζει ποιο από τα υποσύνολα κινήσεων θα χρησιμοποιηθεί: PistolLoco, RifleLoco ή AndroidLoco, όπως παρουσιάζεται στην αντίστοιχη εικόνα 37.



Εικόνα 37. Cached Pose Locomotion

Κάθε επιμέρους State Machine αποτελείται από δύο κύρια States, το Idle και το Move, τα οποία εναλλάσσονται ανάλογα με τη μεταβλητή ShouldMove (Boolean). Όταν αυτή είναι True, ο Agent μεταβαίνει σε κατάσταση κίνησης, ενώ σε αντίθετη περίπτωση παραμένει σε στάση αδράνειας (Idle). Και

τα δύο States βασίζονται σε Blendspaces και Sequence Players που καθοδηγούνται από μεταβλητές και προσαρμόζονται στις διαφορετικές συμπεριφορές των χαρακτήρων.



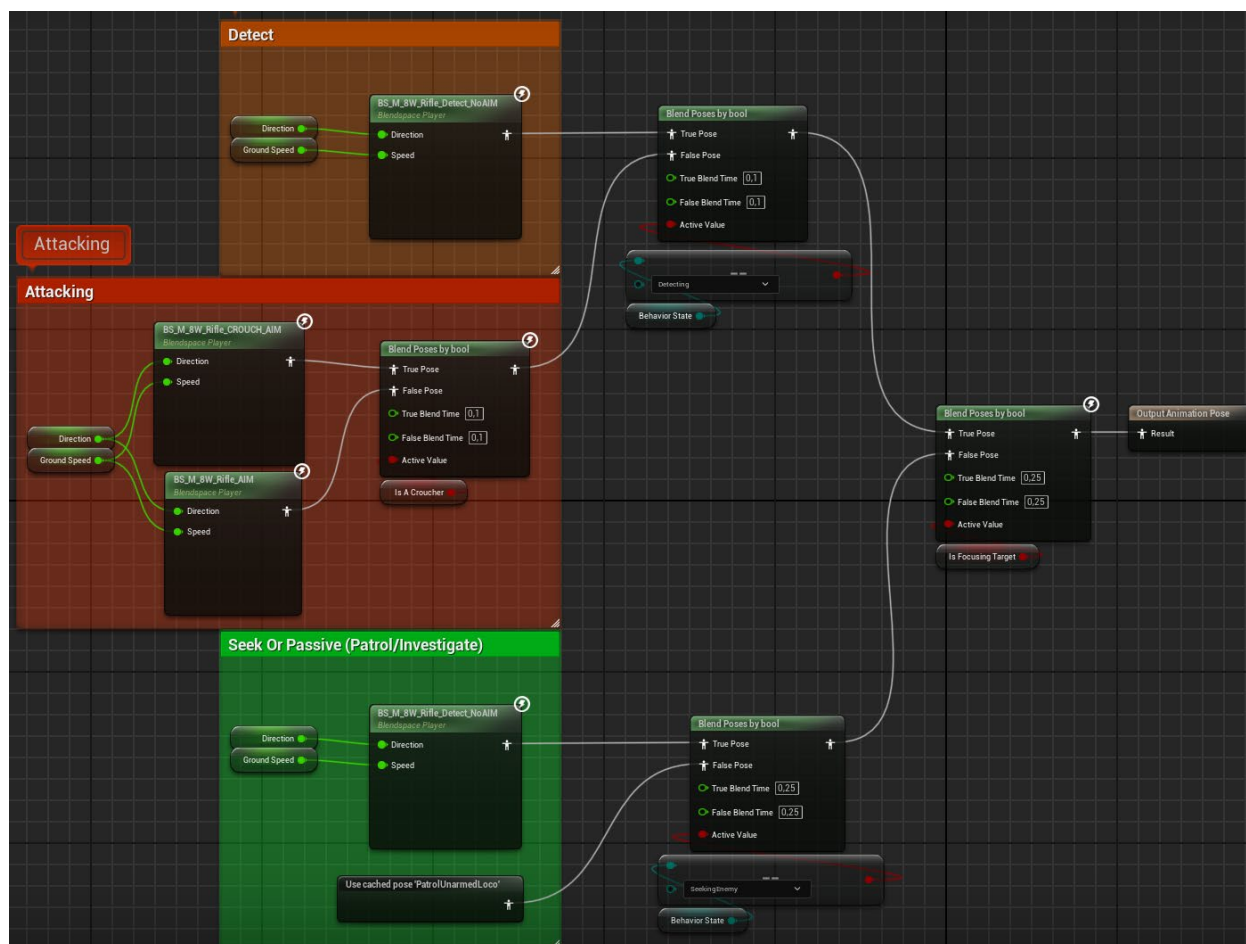
Εικόνα 38. State Machine (Idle&Move)

Η επιλογή του κατάλληλου Blendspace σε κάθε κατάσταση εξαρτάται από τη συμπεριφορά που εκτελεί ο Agent, όπως αυτή ορίζεται στο Behavior Tree, και από το αν έχει οριστεί κάποιο σημείο εστίασης, μέσω των μεταβλητών BehaviorState (Enum) και IsFocusingTarget (Boolean). Η μεταβλητή BehaviorState καθορίζει τη λογική κατάσταση του Agent, όπως Passive, Investigate, Seeking, Detect και Attacking, ενώ η μεταβλητή IsFocusingTarget ενεργοποιείται στις περιπτώσεις που ο Agent έχει οπτική με κάποιο στόχο και θέλει να εστιάσει προς αυτόν (True για Attacking και Detect). Ο συνδυασμός των τιμών αυτών των μεταβλητών καθορίζει ποιο Blendspace θα επιλεγεί, όπως φαίνεται στον παρακάτω πίνακα:

Behavior States	Passive	Investigate	Seeking	Detect	Attacking
Awareness (Player)	X	X	X	X/Y	Y
IsFocusingTarget	X	X	X	Y	Y
Weapon Equipped	X	X	Y	X/Y	Y
Selected Blendspace	PatrolUnarmedLoco		BS_EnemyType_Weapon_Detect_NoAim		BS_EnemyType_Weapon_Aim

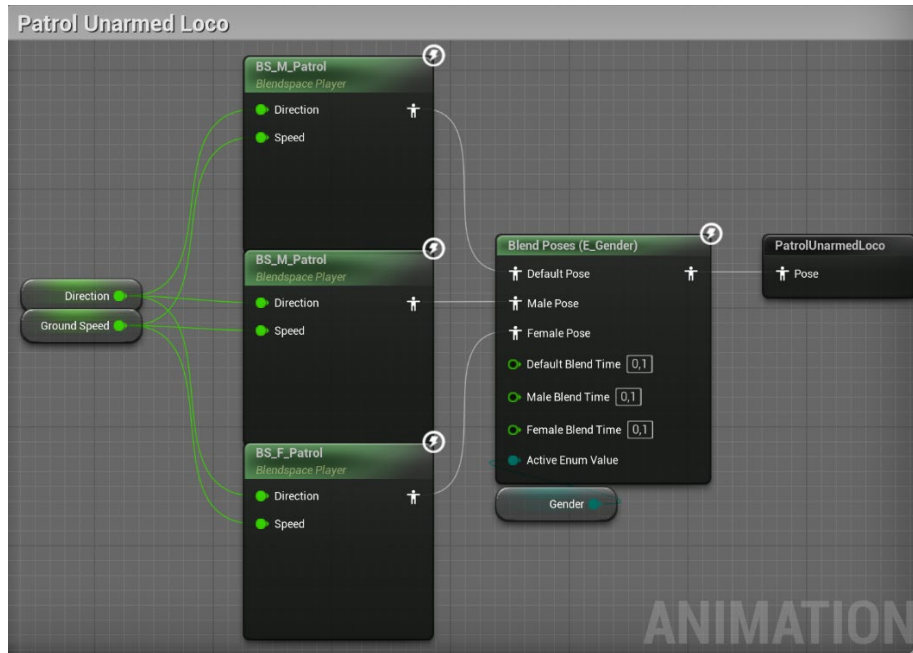
Move State: Η ανάθεση των Blendspaces σε έναν Agent τύπου Ranged πραγματοποιείται ως εξής: Το CachedPose PatrolUnarmedLoco, χρησιμοποιείται στα States όπου ο Agent δεν φέρει όπλο (Passive και Investigate). Αυτό το Cached Pose, περιλαμβάνει τα Blendspaces για κάθε Agent, σύμφωνα με το φύλο τους (Male/Female). Η διαλογή πραγματοποιείται μέσω ελέγχου της μεταβλητής Gender, όπως φαίνεται στην εικόνα TBA. Το BS_EnemyType_Weapon_Detect_NoAim ανατίθεται στα States όπου ο Agent δεν

έχει αντιληφθεί την παρουσία του παίκτη και παράλληλα ενδέχεται να κρατά τον οπλισμό στα χέρια του (Seeking και Detect). Τέλος, το BS_EnemyType_Weapon_Aim ενεργοποιείται στην περίπτωση που ο Agent βρίσκεται στο επιθετικό State (Attacking).



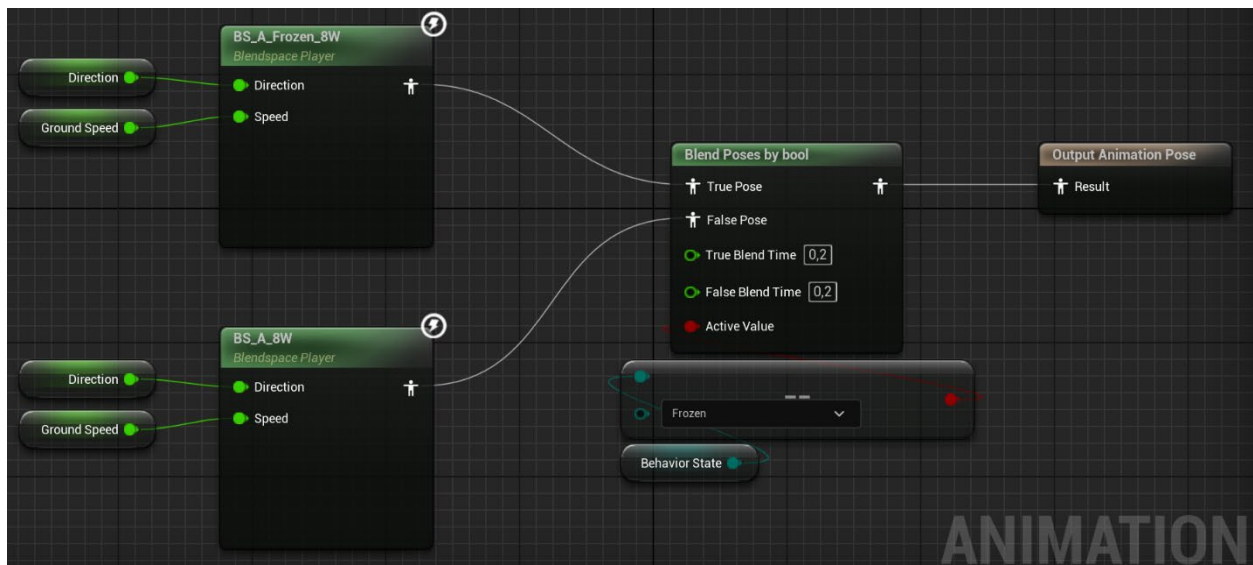
Εικόνα 39. Move State (Rifle Locomotion)

Η τελική έξοδος (OutputPose) ελέγχει τη μεταβλητή IsFocusingTarget ώστε να διαπιστωθεί αν ο Agent βρίσκεται σε κατάσταση Attacking ή Detect. Αν αυτό ισχύει, η μεταβλητή BehaviorState καθορίζει επακριβώς ποια είναι η ενεργή συμπεριφορά και αντιστοιχείται το ανάλογο BlendSpace. Αντιθέτως, αν η τιμή της IsFocusingTarget είναι False, ο Agent βρίσκεται σε μία από τις καταστάσεις Passive, Investigate ή Seeking, και γίνεται χρήση του BS_EnemyType_Weapon_Detect_NoAim για την Seeking και του PatrolUnarmedLoco για τις υπόλοιπες. Σε όλα τα BlendSpaces, η κατεύθυνση και η ταχύτητα του Agent καθορίζονται από τις μεταβλητές Direction και Speed.



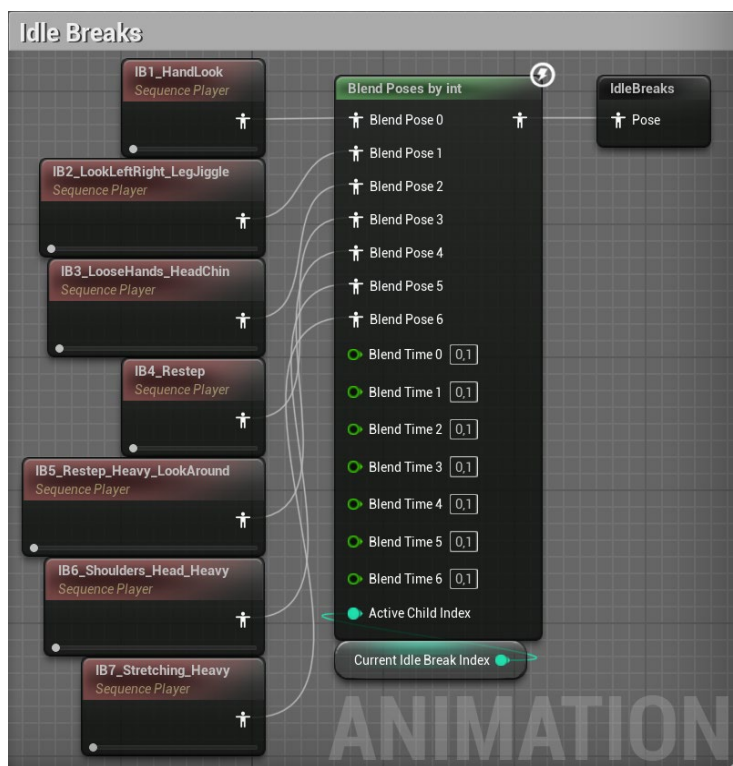
Εικόνα 40. Cached Pose PatrolUnarmedLoco

Όσον αφορά στους Agents τύπου Android, στο Move State τους χρησιμοποιούνται δύο διαφορετικά Blendspaces, ανάλογα με την επίδραση του αντικειμένου DataWorm. Όταν ο Agent βρίσκεται υπό την επιρροή του αντικειμένου, ενεργοποιείται το BS_A_Frozen_8W, ενώ σε αντίθετη περίπτωση χρησιμοποιείται το BS_A_8W.



Εικόνα 41. Move State (Android Locomotion)

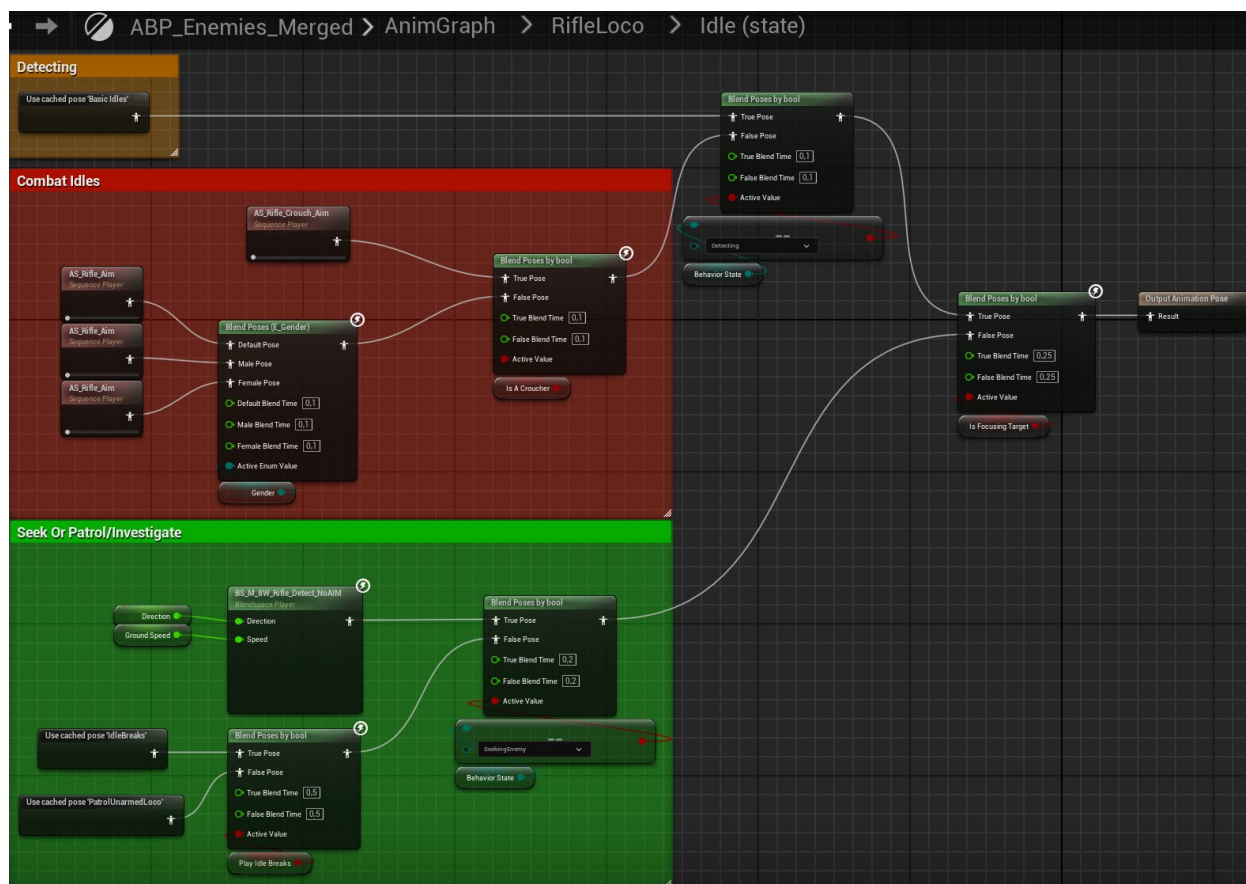
Idle State: Στο Idle State, η ενσωμάτωση του Cached Pose PatrolUnarmedLoco διασφαλίζει πως θα αναπαραχθούν Idle animations σύμφωνα με το Blendspace, καθώς η ταχύτητα του Agent σε αυτό το State είναι μηδενική. Παράλληλα, μέσω της μεταβλητής PlayIdleBreaks (Boolean), η οποία εναλλάσσεται περιοδικά μέσω του Custom Event Trigger Idle Breaks, ενεργοποιεί το Cached Pose "Idle Breaks". Αυτό περιλαμβάνει ένα σύνολο αυθόρμητων κινήσεων (IdleBreaks), όπως φαίνεται στην εικόνα 42. Η αναπαραγωγή αυτών των animations βασίζεται στην χρήση Sequence Players, διασφαλίζοντας ομαλή λειτουργία αναπαραγωγής με άμεσο τρόπο.



Εικόνα 42. Cached Pose Idle Breaks

Παρά ταύτα, το Idle State απαιτεί την ενσωμάτωση πρόσθετων Blendspaces και Sequence Players, προκειμένου να εξασφαλιστεί η κατάλληλη διαλογή των animations σε ειδικές περιπτώσεις. Συγκεκριμένα, τα Behavior Trees μπορούν να θέσουν την τιμή Speed σε μηδέν, γεγονός το οποίο προκαλεί τη μετάβαση του State από Move σε Idle, ακόμη και αν ο χαρακτήρας πρέπει να χρησιμοποιήσει Blendspaces που ανήκουν στο State Move. Για παράδειγμα, όταν ο Agent επιτίθεται σημαδεύοντας με το όπλο του και σταματά στιγμιαία για να αλλάξει κατεύθυνση κίνησης, υπάρχει κίνδυνος να επιλεγεί ένα τυπικό Idle animation. Αυτό θα είχε ως αποτέλεσμα τη στιγμιαία πτώση του όπλου, απεικονίζοντας απότομες και μη ρεαλιστικές κινήσεις. Ως εκ τούτου, οι λογικές επιλογής Blendspace του Move State επεκτάθηκαν και στο Idle State, καθώς και προστέθηκαν ορισμένα Sequence Players κατά τα οποία ο παίκτης σημαδεύει με τον οπλισμό του ακίνητος. Με αυτόν τον τρόπο,

διασφαλίζεται η ομαλή μετάβαση των States και η συνέπεια των κινήσεων, ακόμη και όταν ο Agent είναι στατικός αλλά τελεί υπό συνθήκες επίθεσης ή στόχευσης. Η εικόνα 43 απεικονίζει ενδεικτικά την υλοποίηση του συγκεκριμένου συστήματος για έναν Agent τύπου Rifle.



Εικόνα 43. Idle State (Rifle Locomotion)

6.6. Skeletal Mesh Setup

Όπως έχει ήδη αναφερθεί, το project χρησιμοποιεί έναν ενιαίο σκελετό (SK_Mannequin) και μέσω της διαδικασίας IK Retargeting σε πραγματικό χρόνο, οι επιπλέον σκελετοί για τη διαφοροποίηση της εμφάνισης ακολουθούν την κινησιολογία του (Epic Games, n.d.). Ο ενιαίος σκελετός υπάρχει στη θέση του Skeletal Mesh Component "Mesh" στον οποίο προστίθενται επιπλέον Skeletal Mesh Components που λειτουργούν ως θέσεις εκχώρησης, τα οποία βρίσκονται προσκολλημένα στο Mesh. Καθώς αυτά τα Components κληρονομούνται σε κάθε Child Blueprint αυτό μας επιτρέπει την ανάθεση ολόκληρων χαρακτήρων και διάφορων accessories στη μορφή σκελετού με ιεραρχημένα bones, ώστε να επιτύχουμε ποικίλες εμφανίσεις εχθρών. Κατά συνέπεια, προστέθηκαν τα Components "body", "head", "hair", "chest", "vest", "outfit", "pants", "hand/gloves", "boots", "hat", "eyebrows" και "beard" τα οποία

παραμένουν κενά στο Parent Blueprint και λειτουργούν ως placeholders σκελετών προς ανάθεση σε κάθε Child. Αυτή η δομή επιτρέπει τη διαχείριση των χαρακτήρων, διαφορετικής σκελετικής συγκρότησης, που υπήρχαν διαθέσιμοι για το συγκεκριμένο Project. Ενδεικτικά, ένας σκελετός μπορεί να αποτελείται από ένα μοναδικό mesh (single mesh character) και ένας άλλος να απαρτίζεται από πολλαπλά μέρη (modular character) όπως απεικονίζεται στη εικόνα 44. Για την ακριβή και την άμεση αντιγραφή των Animations των Components από το mesh χρησιμοποιήθηκε το node Set Leader Pose Component για τη σύνδεση των παραπάνω στο γονικό mesh. Ωστόσο, η μέθοδος έχει σημαντικό κόστος απόδοσης στο Render Thread και η αναπαραγωγή Animations μπορεί να πραγματοποιηθεί μόνο στο γονικό mesh. Τέλος, δημιουργήθηκαν επιπλέον μεταβλητές για κάθε Skeletal Mesh Component με ενεργές τις επιλογές Expose On Spawn και Instance Editable, ώστε να καταστεί δυνατή η ανάθεση των Assets απευθείας στο Viewport ανά Actor που τοποθετείται στον χώρο, χωρίς την ανάγκη ανάπτυξης κώδικα για τη δημιουργία διαφορετικών χαρακτήρων. Κάθε τέτοια μεταβλητή αναθέτει το περιεχόμενό της στο αντίστοιχο Component μέσω του Node Set Skeletal Mesh Asset. Οι εντολές τοποθετήθηκαν στο Construction Script του BP_Enemy_Base, ώστε να εκτελούνται κατά την αρχικοποίηση ενός Actor, όπως φαίνεται στην εικόνα 45. Με αυτόν τον τρόπο, κάθε τύπος Agent τοποθετείται στο Level και τα χαρακτηριστικά του ανατίθενται χειροκίνητα. Ωστόσο, σε ένα ολοκληρωμένο Project με πολλαπλά Levels και Sublevels, όπου δεν είναι απαραίτητο όλοι οι Agents να είναι φορτωμένοι στη μνήμη σε όλη τη διάρκεια του παιχνιδιού, θα ήταν απαραίτητο ένα Spawn Assets Blueprint υπεύθυνο για το spawn των Actors, βάσει συνθηκών. Το συγκεκριμένο Blueprint Asset θα προσέφερε οργάνωση, επεκτασιμότητα και μειωμένο υπολογιστικό κόστος, καθώς κάθε ενεργός σκελετός στο Level αυξάνει σημαντικά τη χρήση των υπολογιστικών πόρων.



Εικόνα 44. Modular Character Example



Εικόνα 45. Skeletal Mesh Setup

6.7. Weapon Blueprint

Ο οπλισμός που υλοποιήθηκε στο παρόν Project περιλαμβάνει τρία διακριτά είδη όπλων, τα οποία αντιστοιχούν στους τρεις βασικούς τύπους Agents. Συγκεκριμένα, χρησιμοποιείται ένα πιστόλι (Pistol), ένα αυτόματο πυροβόλο (Rifle) και ένα όπλο κοντινής εμβέλειας τύπου Katana (Melee), τα οποία ανατίθενται στους Agents τύπου Pistol, Rifle και Android, αντίστοιχα.

Το πιστόλι και το αυτόματο πυροβόλο παρουσιάζουν αρκετές ομοιότητες ως προς τη γενική τους λειτουργία, δεδομένου ότι και τα δύο είναι πυροβόλα όπλα μακρινής εμβέλειας. Κατά τη χρήση τους, αναπαράγονται εφέ ανάφλεξης και καπνού στην κάνη, καθώς και εφέ αποβολής κάλυκα, τα οποία προσδίδουν ρεαλισμό στη σκηνή μάχης. Επιπλέον, κάθε πυροβολισμός συνοδεύεται από ηχητικά εφέ, τα οποία είναι δυνατόν να ανιχνευθούν από άλλους Agents στο περιβάλλον, προκαλώντας την αντίστοιχη συμπεριφορική αντίδραση. Ωστόσο, παρά τις κοινές αυτές βάσεις, το πιστόλι και το αυτόματο πυροβόλο διαφέρουν ως προς συγκεκριμένα χαρακτηριστικά. Αυτές οι διαφορές περιλαμβάνουν την οπτική απεικόνιση και το μέγεθος των όπλων, την τιμή του damage που προκαλείται ανά πυροβολισμό, την πυκνότητα των ριπών ανά δευτερόλεπτο, καθώς και την ακουστική εμβέλεια των πυροβολισμών. Η τελευταία προσαρμόζεται δυναμικά ανάλογα με τις παραμέτρους αντίληψης που έχουν αναπτυχθεί για τους Agents. Αντιθέτως, το όπλο τύπου Katana που χρησιμοποιείται από τους Android Agents είναι αποκλειστικά κοντινής εμβέλειας και η χρήση του δεν συνοδεύεται από ηχητικά εφέ που θα μπορούσαν να προδώσουν τη θέση του Agent στον χώρο.

Για την ενοποιημένη και επεκτάσιμη διαχείριση των όπλων, αναπτύχθηκε ένα Parent Blueprint με την ονομασία BP_Weapon_NPC_Base, το οποίο ενσωματώνει τις βασικές λειτουργίες όλων των όπλων. Η συγκεκριμένη προσέγγιση επιτρέπει τη δημιουργία υποκλάσεων για κάθε τύπο όπλου με ελάχιστη επανάληψη κώδικα, διευκολύνοντας τις μελλοντικές τροποποιήσεις ή επεκτάσεις. Στο Parent Blueprint

έχουν προστεθεί οι απαραίτητες μεταβλητές και Components, τα οποία λειτουργούν ως placeholders. Οι υποκλάσεις που το κληρονομούν (Child Blueprints) χρησιμοποιούν τα στοιχεία αυτά μέσω αναθέσεων, χωρίς την ανάγκη περαιτέρω υλοποίησης λογικής. Ενδεικτικά, η μεταβλητή WeaponDamage ορίζεται στο Parent Blueprint και λειτουργεί ως είσοδος κατά την κλήση του function TakeDamage. Παρόλο που η λογική του function πυροδότησης έχει υλοποιηθεί μόνο στη βασική κλάση, η τιμή της μεταβλητής WeaponDamage ορίζεται σε επίπεδο υποκλάσης, και το function χρησιμοποιεί αυτήν τη δυναμικά ανατεθειμένη τιμή κατά την εκτέλεση. Με αυτόν τον τρόπο, η κάθε υποκλάση, όπως οι BP_NPC_Pistol, BP_NPC_Rifle και BP_NPC_Melee, περιλαμβάνει μόνο τις απαιτούμενες τιμές για τις μεταβλητές και τα απαραίτητα Assets για τα Components, χωρίς να περιλαμβάνει αυτόνομα λειτουργικό κώδικα.

Η διασύνδεση των όπλων με τους αντίστοιχους Agents γίνεται μέσω της μεταβλητής WeaponActor, η οποία έχει δηλωθεί στο Parent Blueprint των Agents. Η συγκεκριμένη μεταβλητή επιτρέπει την ανάθεση οποιασδήποτε υποκλάσης όπλου σε έναν Agent, προσφέροντας τη δυνατότητα ενεργοποίησης των λειτουργιών του όπλου απευθείας από τον Agent.

Το παρόν υποκεφάλαιο παρουσιάζει αναλυτικά τα Components και τις μεταβλητές που προστέθηκαν στο Parent Blueprint του οπλισμού, καθώς και τον τρόπο με τον οποίο χρησιμοποιούνται. Επιπλέον, παρουσιάζονται οι τιμές που ανατέθηκαν στις μεταβλητές για κάθε υποκλάση, ώστε να διαμορφώνονται τα ιδιαίτερα χαρακτηριστικά του κάθε όπλου. Ακολουθώντας, αναλύονται η μεθοδολογία ανάπτυξης και οι τεχνικές που χρησιμοποιήθηκαν για την υλοποίηση της πυροδότησης στα όπλα τύπου Pistol και Rifle. Όσον αφορά στο Melee όπλο των Android Agents, η περιγραφή του συνδέεται άρρηκτα με την ειδική επίθεση AerialAttack και ως εκ τούτου αναλύεται σε ξεχωριστό κεφάλαιο. Τέλος, περιγράφεται η διαδικασία ανάθεσης του οπλισμού σε κάθε Agent, εξασφαλίζοντας τη λειτουργικότητα του εξοπλισμού εντός του παιχνιδιού και επιτρέποντας την πλήρη αξιοποίηση των δυνατοτήτων του σε πραγματικό χρόνο.

6.7.1. Components και Variables

Τα Components που προστέθηκαν ως placeholders στο Blueprint του BP_Weapon_NPC_Base περιλαμβάνουν:

- WeaponMeshBase (Static Mesh Component): περιέχει το static mesh (μοντέλο) του όπλου
- Muzzle (Scene Component): λειτουργεί ως μια τοποθεσία προσκολλημένη στην κάννη του όπλου και χρησιμοποιείται για την αναπαραγωγή των εφέ ανάφλεξης και καπνού στην τρέχουσα τοποθεσία της
- MuzzleDirection (Arrow Component): επιτρέπει τον προσδιορισμό της κατεύθυνσης προς την οποία σημαδεύει το όπλο
- SpawnWeaponFX και DissolveWeaponFX (Niagara Particle System Component): χρησιμοποιούνται για την ανάθεση των ειδικών εφέ εμφάνισης και εξαφάνισης, αντίστοιχα, του όπλου Katana στα χέρια του Android

Οι μεταβλητές που δημιουργήθηκαν ως placeholders είναι:

- *WeaponDamage (float)*: Η τιμή damage που προκαλεί το όπλο σε κάθε πλήρη βολή.
- *WeaponDamageHead (float)*: Η τιμή damage που προκαλεί το όπλο όταν η βολή χτυπά το κεφάλι του στόχου. Η τιμή της είναι μεγαλύτερη από αυτή του WeaponDamage, ως επιβράβευση προς τον παίκτη.
- *Weapon Range (float)*: Η απόσταση που μπορεί να φτάσει το όπλο για να επηρεάσει στόχους, δηλαδή το μέγιστο εύρος βολής.
- *AutoFire (Boolean)*: Καθορίζει αν το όπλο έχει δυνατότητα αυτόματης βολής. Στην περίπτωση του Rifle η μεταβλητή παίρνει την τιμή True και στην περίπτωση του Pistol και Melee παίρνει την τιμή False.
- *MuzzleflashEffect (Cascade Particle System)*: Το οπτικό εφέ της λάμψης που παράγεται από το στόμιο του όπλου κατά την εκτυρσοκρότηση.
- *FireSound (Sound Base)*: Ο ήχος που αναπαράγεται όταν το όπλο πυροβολεί.
- *BulletRecoverTime (Float)*: Ο χρόνος (σε δευτερόλεπτα) που απαιτείται για να επιτραπεί στο όπλο να επαναλάβει ριπή μετά από μια βολή.
- *WeaponType (Enum)*: Ο τύπος του όπλου (π.χ. Pistol, Rifle), που καθορίζει τις δυνατότητες και τα χαρακτηριστικά του όπλου. Χρησιμοποιείται εκτός του Blueprint των όπλων, σε σημεία όπου το Asset υπάρχει ως reference κυρίως για τον έλεγχο του τύπου του Agent που φέρει το όπλο.
- *RandomTimesToFire (Integer)*: Ο αριθμός των τυχαίων φορών που θα πυροβολήσει το όπλο σε μια επίθεση του NPC. Χρησιμοποιείται για να προσομοιώσει την ποικιλία των ριπών ανά επίθεση. Αντιπροσωπεύει τον αριθμό που το Custom Event FireOnce θα εκτελεστεί σε μια επίθεση και αρχικοποιείται σε κάθε κλήση του Custom Event Fire με τυχαία τιμή.
- *ActualTimesFired (Integer)*: Ο τρέχων αριθμός των πυροβολισμών που έχουν εκτελεστεί σε μια επίθεση. Χρησιμοποιείται ως συνθήκη για τη διακοπή της πυροδότησης Custom Event FireOnce) και αρχικοποιείται σε κάθε κλήση του Custom Event Fire με τιμή μηδέν.
- *SurfaceType (Enum)*: η μεταβλητή που ορίζεται σε κάθε επιφάνεια και actor και χαρακτηρίζει την τύπο της επιφάνειας ή την υφή που αυτή αντιπροσωπεύει. Χρησιμοποιείται για τον καθορισμό του τρόπου αλληλεπίδρασης της επιφάνειας στην οποία προσκρούει μια σφαίρα, όπως ηχητικά εφέ και εφέ πρόσκρουσης.
- *Impact Point Effects (Cascade Particle System)*: το σύνολο των εφέ που αναπαράγονται στο σημείο που η σφαίρα χτυπά, για καθεμία από τις περιπτώσεις επιφανειών, σύμφωνα με το περιεχόμενο της μεταβλητής SurfaceType (π.χ. BloodEmitter αν το SurfaceType είναι κάποιος Actor, MetalEmitter αν η επιφάνεια είναι μεταλλική κ.α.).
- *FireCompleted*: το Event Dispatcher που χρησιμοποιείται στη λειτουργία παύσης του πυροβολισμού και σηματοδοτεί την ολοκλήρωση της διαδικασίας.

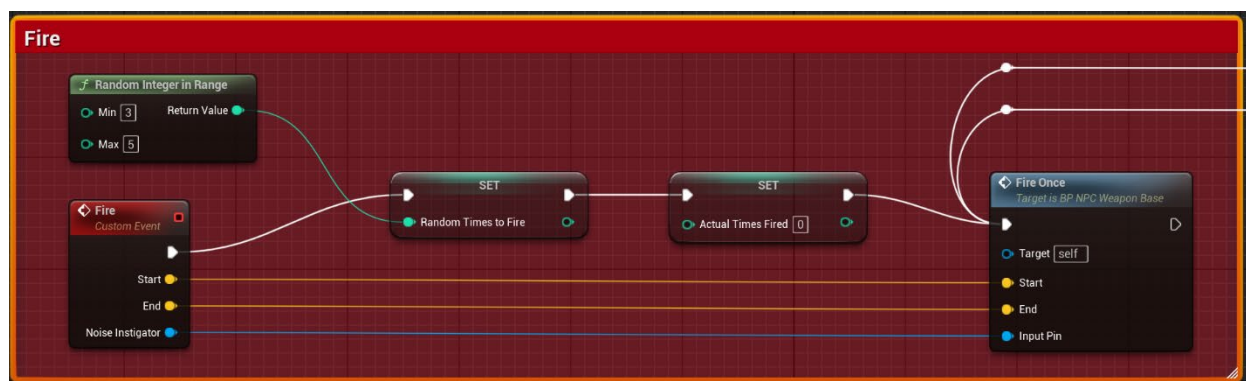
Οι τιμές των μεταβλητών που ανατέθηκαν σε κάθε υποκλάση παρουσιάζονται στον παρακάτω πίνακα. Το περιεχόμενο των RandomTimesToFire, ActualTimesFired, SurfaceType και Impact Point Effects ορίζεται ανά περίπτωση κατά την κλήση.

	BP_NPC_Pistol,	BP_NPC_Rifle	BP_NPC_Melee
WeaponType	Pistol	Rifle	Android
WeaponDamage	25/100	35/100	45/100
WeaponDamageHead	60/100	80/100	N/A
Weapon Range	80 m	500 m	N/A
AutoFire	False	True	N/A
BulletRecoverTime	0.8 sec	0.5 sec	N/A
FireSound	Pistol_Fire_SFX	Rifle_Fire_SFX	Katana_Slash_SFX
MuzzleflashEffect	CPS_MFE_Pistol	CPS_MFE_Rifle	CPS_KnifeRibbon

6.7.2. Fire Functions

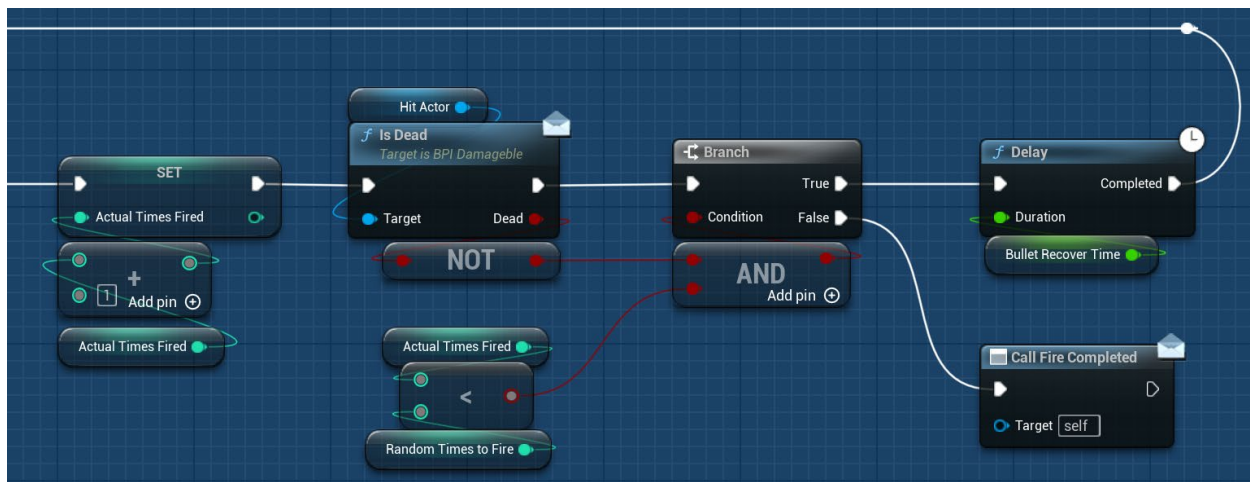
Η εκτέλεση μιας επίθεσης των Agents τύπου Pistol και Rifle ενεργοποιείται μέσω συνθηκών που ορίζονται στο Behavior Tree του Agent, καλώντας το function Attack του Blueprint Interface BPI Enemy AI. Οι ενέργειες πυροδότησης πραγματοποιούνται μέσω του Custom Event Fire, το οποίο υλοποιείται στο Parent Blueprint του οπλισμού BP_Weapon_NPC_Base. Το Fire χρησιμοποιεί τη μεταβλητή RandomTimesToFire (Integer) για να προσομοιώσει τυχαίο αριθμό ριπών σε κάθε επίθεση του Agent και καλεί το Custom Event Fire Once πολλαπλές φορές σύμφωνα με την τιμή της. Το Fire Once περιλαμβάνει το σύνολο της πυροδότησης, συμπεριλαμβανομένης της προσομοίωσης πυροδότησης και αναζήτησης στόχου μέσω LineTrace, απόδοση Damage, οπτικοακουστικών εφέ και των απαραίτητων ελέγχων για τη διακοπή της διαδικασίας. Με αυτό τον τρόπο, κάθε επίθεση περιλαμβάνει ένα τυχαίο αριθμό πυροδοτήσεων, όπου στο τέλος κάθε ριπής, το function ολοκληρώνεται στις περιπτώσεις που ο αριθμίων των ριπών ολοκληρώθηκε ή ο παίκτης πέθανε.

Fire (Function) Το function Attack λαμβάνει την τρέχουσα τοποθεσία του όπλου (Start), την τοποθεσία του στόχου επίθεσης (End) και ένα reference στον Agent που επιτίθεται (NoiseInstigator), και τις περνάει ως παραμέτρους στο Custom Event Fire του BP_Weapon_NPC_Base. Κατά την εκκίνηση της Fire, η μεταβλητή RandomTimesToFire με έναν τυχαίο ακέραιο (3 έως 5) και η μεταβλητή ActualTimesFired αρχικοποιείται σε μηδέν. Οι μεταβλητές αυτές, καθορίζουν τον αριθμό των ριπών μιας επίθεσης, κάθε φορά που το Behavior Tree πυροδοτεί την εκτέλεση της. Αυτός ο μηχανισμός εξασφαλίζει ότι κάθε Agent πραγματοποιεί μεταβλητό αριθμό ριπών, προσδίδοντας φυσικότητα και ποικιλία στις επιθέσεις.



Εικόνα 46. Fire Function

Αυτό επιτυγχάνεται από την εξής δομή: Κάθε εκτέλεση της Fire καλεί την FireOnce, κατά την οποία το όπλο πυροδοτείται και η μεταβλητή ActualTimesFired αυξάνεται κατά μια μονάδα. Η ολοκλήρωση της FireOnce ελέγχει αν η ActualTimesFired είναι μικρότερη από αυτή της RandomTimesToFire, διασφαλίζοντας ταυτόχρονα πως ο στόχος (π.χ. ο παίκτης) δεν έχει πεθάνει, με την ανάκτηση της Boolean IsDead μέσω του function IsDead. Στην περίπτωση που ο έλεγχος αποδειχθεί True και για τις δύο συνθήκες, το Custom Event FireOnce επανεκτελείται, αφού πραγματοποιηθεί μια μικρή χρονική καθυστέρηση (delay) της τιμής BulletRecoverTime. Σε αντίθετη περίπτωση, το Event Dispatcher FireCompleted καλείται για να σηματοδοτήσει την ολοκλήρωση της ενέργειας του πυροβολισμού. Η χρήση του Event Dispatcher είναι αναγκαία, ώστε το Behavior Tree να αναθέσει το επόμενο πακέτο εργασίας την κατάλληλη στιγμή, όπως αναφέρεται στο κεφάλαιο 8.

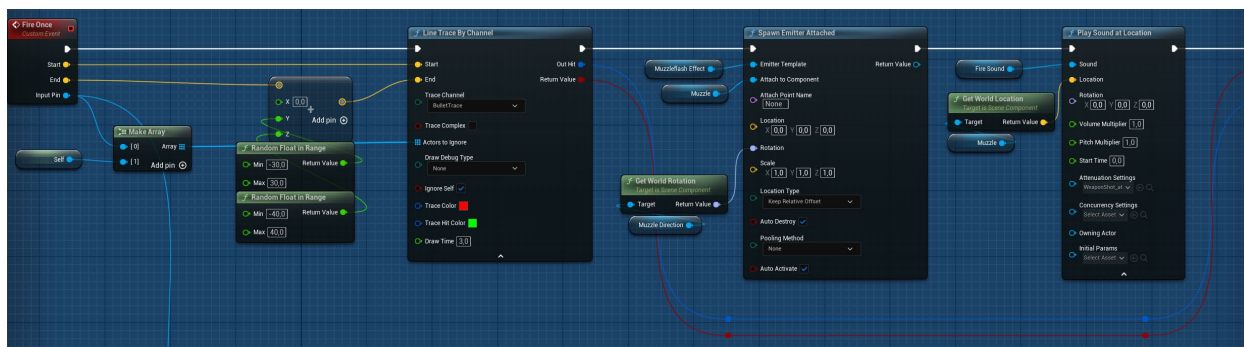


Εικόνα 47. Fire Once Exit Conditions

Στο συγκεκριμένο σημείο συνίσταται η επέκταση της συγκεκριμένης προσέγγισης, με ένα σύνολο μεταβλητών που θα αντιπροσωπεύουν τις διαθέσιμες και συνολικές σφαίρες που υπάρχουν στον γεμιστήρα ενός NPC και functions που θα τις ενσωματώνουν στις ήδη ανεπτυγμένες λειτουργίες. Η παραπάνω προσθήκη μπορεί να προσφέρει ένα επιπλέον επίπεδο ρεαλισμού με επεκτάσιμο τρόπο, για κάθε όπλο.

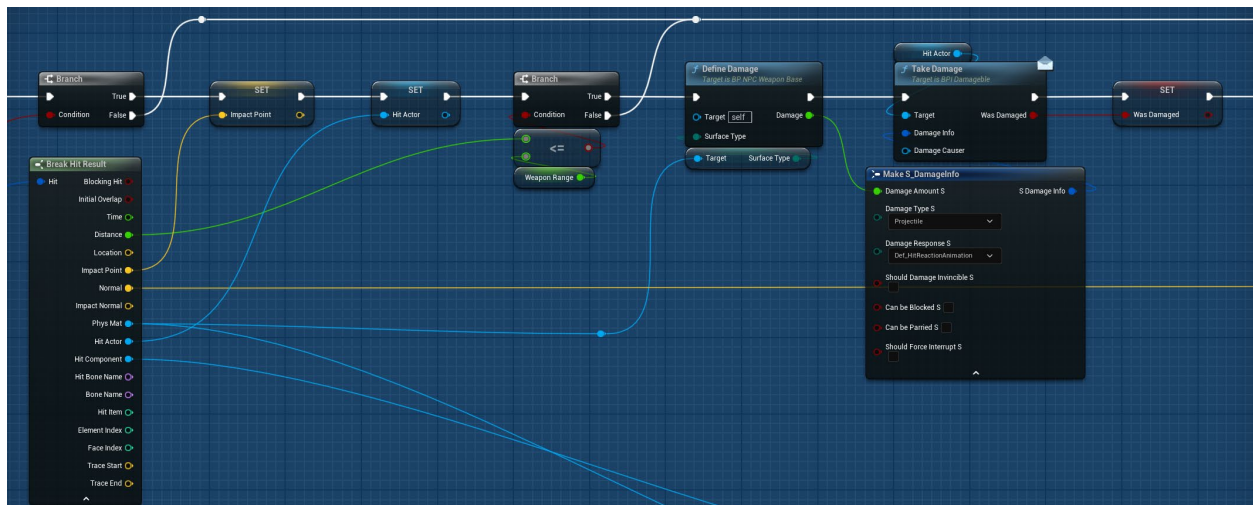
LineTracebyChannel Μετά τον ορισμό των RandomTimesToFire και ActualTimesFired καλείται το Custom Event FireOnce με είσοδο τις μεταβλητές Start, End και NoiseInstigator. Οι Start και End παρέχονται στο node LineTracebyChannel ως σημεία εκκίνησης και τερματισμού αντίστοιχα. Το node προσφέρει μια μέθοδο ελέγχου για το αν ένα αντικείμενο ή επιφάνεια βρίσκεται πάνω σε μια ευθεία γραμμή, βασισμένο σε έναν συγκεκριμένο Channel (όπως Pawn, World Static, Custom κ.α). Αν το line trace βρει κάποιο αντικείμενο στην πορεία του, επιστρέφει πληροφορίες για το σημείο σύγκρουσης, το αντικείμενο που βρέθηκε, την κατεύθυνση και την απόσταση. Στο συγκεκριμένο Project το line trace γίνεται στο channel "BulletTrace" που προστέθηκε, επιτρέποντας τον ακριβή ορισμό των αντικειμένων που ανταποκρίνονται σε αυτό και χρησιμοποιήθηκε ως μια αξιόπιστη λύση χαμηλού υπολογιστικού κόστους για την πυροδότηση των όπλων και την ανίχνευση του σημείου σύγκρουσης της σφαίρας. Για την προσθήκη μιας επιπλέον διαφοροποίησης των σημείων στα οποία οι Agents πυροδοτούν, το μεταβλητό διάνυσμα End τροποποιείται, κατά τυχαίους αριθμούς, μεταξύ ενός εύρους τιμών στις συντεταγμένες του Y και Z. Πριν τη χρήση των αποτελεσμάτων που παρέχονται από το LineTracebyChannel, καλούνται τα nodes SpawnEmitterAttached και PlaySound at Location για την αναπαραγωγή του εφέ λάμψης της εκτυρσοκρότησης και τον ήχο του πυροβολισμού, τα οποία δέχονται για είσοδο το περιεχόμενο των μεταβλητών MuzzleflashEffect και FireSound ως assets προς αναπαραγωγή. Για την τοποθεσία που θα αναπαραχθούν, χρησιμοποιείται η θέση του Muzzle Component στον χώρο.

Η χρήση του LineTracebyChannel επιλέχθηκε έναντι της προσέγγισης με χρήση Projectiles λόγω υψηλής απόδοσης, αμεσότητας και απλότητας κατά την υλοποίηση, καθώς και ταχύτερου χρόνου ανάπτυξης του Project. Ωστόσο, ένα άλλο Project θα μπορούσε να επιλέξει τη χρήση Projectiles, εάν απαιτείται προσομοίωση πραγματικής κίνησης και φυσικής αλληλεπίδρασης με το περιβάλλον, όπως επιβράδυνση του βλήματος λόγω της βαρύτητας. Αν και αυτό απαιτεί ανάπτυξη επιπλέον Blueprints και σύνθετη διευθέτηση, μπορεί να προσφέρει αυξημένο βαθμό ρεαλισμού στο Gameplay και να αποτελέσει μια ακόμα προσθήκη στο σύνολο των μηχανισμών του παιχνιδιού.



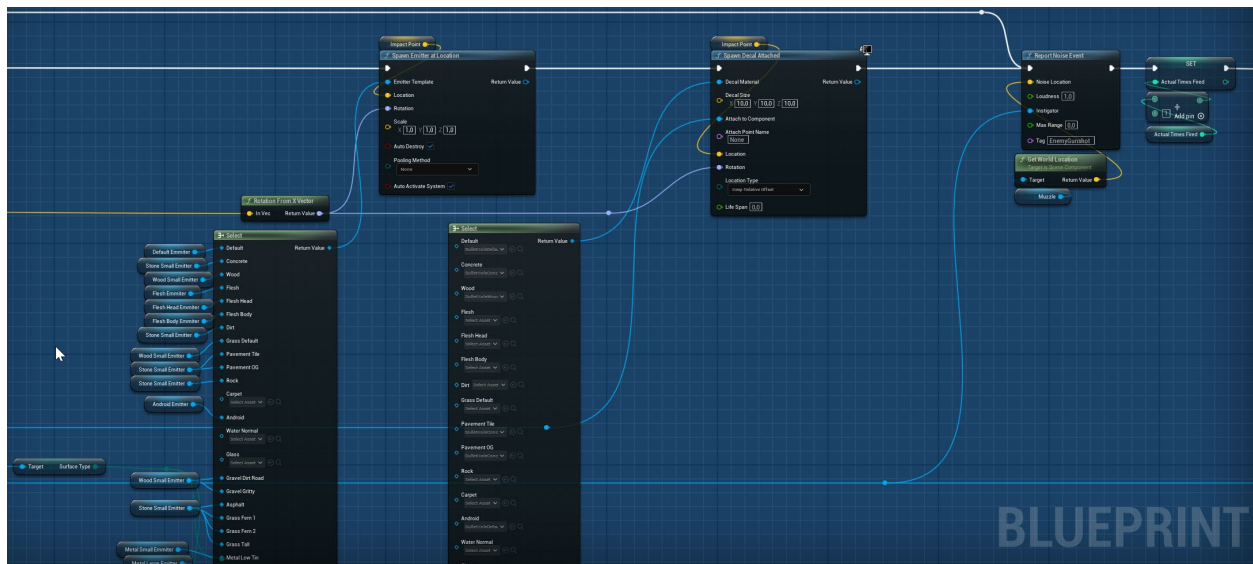
Εικόνα 48. Fire Once (1)

Στη συνέχεια, ελέγχεται η Boolean "Return Value" που παρέχεται από το LineTrace. Αν βρεθεί True, αυτό σημαίνει πως υπήρξε μια σύγκρουση και το σημείο σύγκρουσης μαζί με τον Actor ο οποίος χτυπήθηκε αποθηκεύονται στις μεταβλητές Impact Point (vector) και Hit Actor αντίστοιχα. Σε αυτό το σημείο, ελέγχεται αν η βολή πραγματοποιήθηκε εντός του WeaponRange που αντιστοιχεί στο όπλο και αν ο Actor είναι ένα Asset το οποίο μπορεί να δεχθεί Damage (π.χ. παίκτης ή Agent). Αυτό επιτυγχάνεται με τη σύγκριση της τιμής του Distance που παρέχεται από το trace και της μεταβλητής Weapon Range. Αν η απόσταση Distance είναι μικρότερη ή ίση με το περιεχόμενο της WeaponRange τότε καλείται το function **DefineDamage** το οποίο καθορίζει την τιμή του Damage που θα αποδοθεί, σύμφωνα με τις τιμές που έχουν ανατεθεί στις WeaponDamage και WeaponDamageHead. Ο καθορισμός επιτυγχάνεται μέσω της μεταβλητής Surface Type που διατίθεται στη συλλογή αποτελεσμάτων που επιστρέφει το trace. Έπειτα, καλείται το BPI function TakeDamage, στο οποία παρέχονται το Damage που θα αποδοθεί, ο τύπος του Damage (Projectile), ο τρόπος με τον οποίο θα αντιδράσει ο αποδέκτης, καθώς και οι τιμές των ShouldDamageInvincible, CanBeBlocked, CanBeParried και ShouldForceInterrupt.



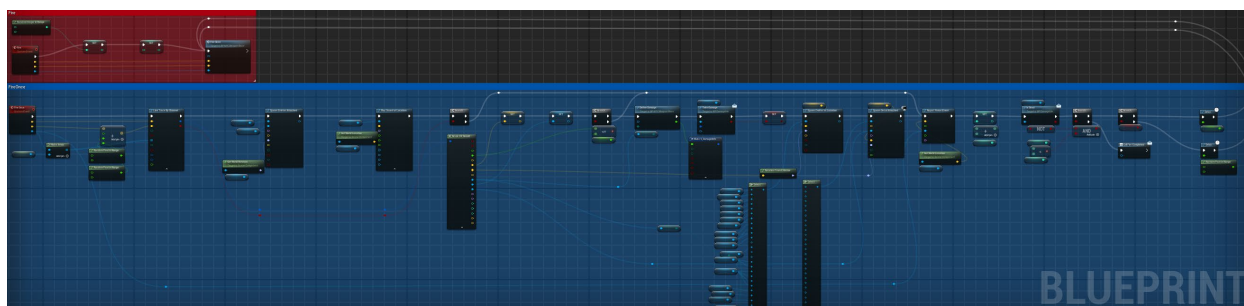
Εικόνα 49. Fire Once (2)

Στη συνέχεια, μέσω των nodes `SpawnEmitterAtLocation` και `SpawnDecalAttached`, αναπαράγονται τα οπτικά εφέ της σύγκρουσης του βλήματος στο σημείο σύγκρουσης, σύμφωνα με τη μεταβλητή `SurfaceType` του `Trace` (π.χ. εκτίναξη στιγμάτων αίματος αν η επιφάνεια είναι `Agent`, ρινίσματα ξύλου αν είναι ξύλινο αντικείμενο κ.α.). Τα εφέ που θα αναπαραχθούν σε κάθε είδος επιφάνειας, αντιστοιχούν στο περιεχόμενο των μεταβλητών `ImpactPointEffects`.



Εικόνα 50. Fire Once (3)

Στις περιπτώσεις που η βολή δεν πραγματοποιήθηκε εντός του WeaponRange ή η Boolean Return Value του LineTrace είναι False, οι παραπάνω ενέργειες παρακάμπτονται, συνεχίζοντας με την εκτέλεση των εφέ σύγκρουσης. Ύστερα από αυτό το σημείο, πραγματοποιείται αναφορά θορύβου για τον ήχο που παράγεται από το όπλο μέσω του node ReportNoiseEvent, το οποίο θα ειδοποιήσει όσους Agents βρίσκονται εντός εύρους για το γεγονός. Με αυτόν τον τρόπο, ειδοποιείται η αίσθηση ακοής του Agent, εντός του AI Controller, όπως αυτή αναλύεται στο κεφάλαιο 7. Ακόμα, για την ταυτοποίηση του Actor που παρήγαγε τον ήχο, παρέχεται στο node το tag "Enemy_Gunshot". Οι τελευταίες ενέργειες του FireOnce περιλαμβάνουν τους ελέγχους που καθορίζουν την επανεκτέλεση της FireOnce ή τον τερματισμό της επίθεσης, όπως αναφέρθηκε στην εισαγωγή της υποενότητας. Η συνολική ροή του διαγράμματος παρουσιάζεται στην εικόνα 51.



Εικόνα 51. Fire Functions Overview

6.7.3. Weapon Setup

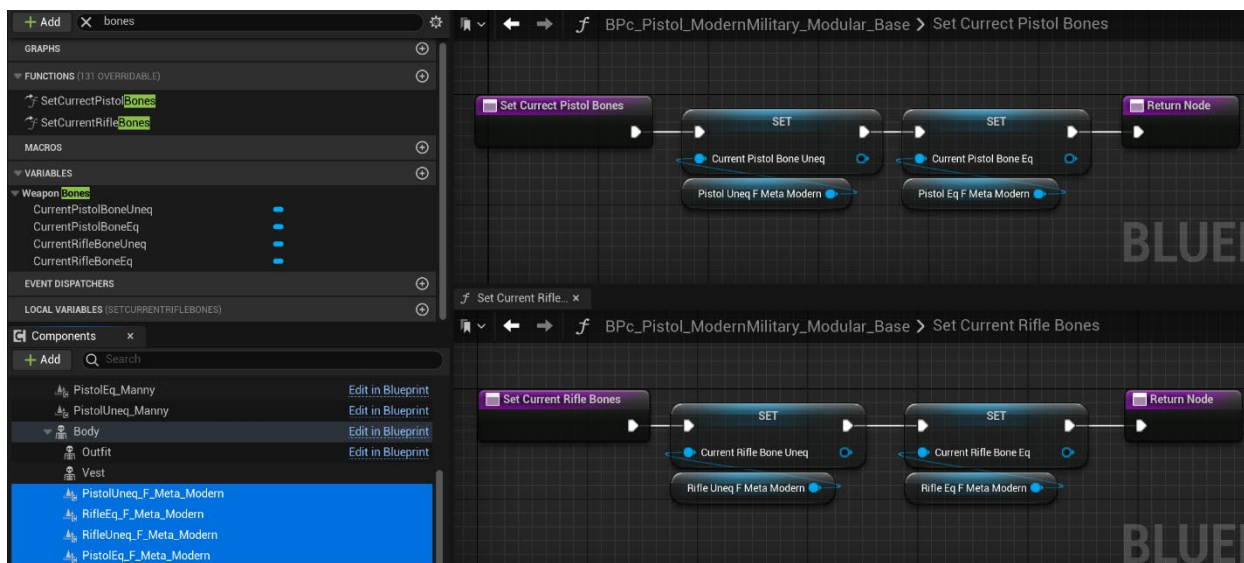
Για την ενσωμάτωση του οπλισμού στους Agents, κάθε Child Blueprint του BP_NPC_Weapon_Base ανατίθεται στο αντίστοιχο Child Blueprint του BP_Enemy_Base, σύμφωνα με τον τύπο του όπλου. Η ανάθεση αυτή πραγματοποιείται κατά την αρχικοποίηση του Agent, στο Event Begin Play, μέσω της εντολής spawn του κατάλληλου Blueprint Actor του όπλου. Για να αποδοθεί σωστά το όπλο στον χαρακτήρα και να ακολουθεί την κίνησή του, απαιτείται η ύπαρξη ενός socket σε συγκεκριμένο bone του σκελετού του μοντέλου. Εφόσον οι χαρακτήρες μοιράζονται τον οδηγό σκελετό (SK_Mannequin), η διαδικασία απαιτεί ιδιαίτερη προσοχή, καθώς η τοποθέτηση όπλων βασισμένη αποκλειστικά στον οδηγό σκελετό μπορεί να οδηγήσει σε ασυνέπειες λόγω διαφορών στη σκελετική γεωμετρία μεταξύ διαφορετικών Agents.

Οι πιθανές θέσεις στις οποίες ένα όπλο μπορεί να υπάρχει στον χώρο, ποικίλουν σύμφωνα με το είδος του όπλου, το μοντέλο που έχει ο κάθε χαρακτήρας και την κατάσταση στην οποία βρίσκεται ο Agent. Συγκεκριμένα, ένας Agent μπορεί να βρίσκεται σε εχθρικές καταστάσεις (Attacking ή Seeking) στις οποίες το όπλο βρίσκεται στα χέρια του (Armed) ή σε μη εχθρικές όπως Patrol (Unarmed). Ακόμα, στις καταστάσεις Unarmed η τοποθεσία του Pistol είναι πάντα στον μηρό και του Rifle στην πλάτη. Αυτό προκύπτει από τα animations που χρησιμοποιούνται κατά την όπλιση και την αφόπλιση. Επιπλέον, σε καταστάσεις Armed υπάρχει διαφοροποίηση του σημείου τοποθέτησης των όπλων στα χέρια, λόγω της διαφορετικής σκελετικής δομής, ακόμη και μεταξύ χαρακτήρων με ίδιου τύπου οπλισμό. Η διαφορετική

σκελετική δομή προκαλεί μία μεγάλη ποικιλομορφία μεταξύ των σημείων του σκελετού που πρέπει να ανατεθούν τα όπλα, ώστε να προσαρτίζονται, με ρεαλιστικό τρόπο, στο σώμα των Agents ανεξαρτήτως κατάστασης. Ωστόσο, το ζήτημα αφορά αποκλειστικά στους Agents τύπου Ranged (Pistol και Rifle), με τις αναθέσεις του οπλισμού των Android να πραγματοποιούνται με πιο απλό τρόπο, λόγω της απουσίας σκελετικών διαφοροποιήσεων.

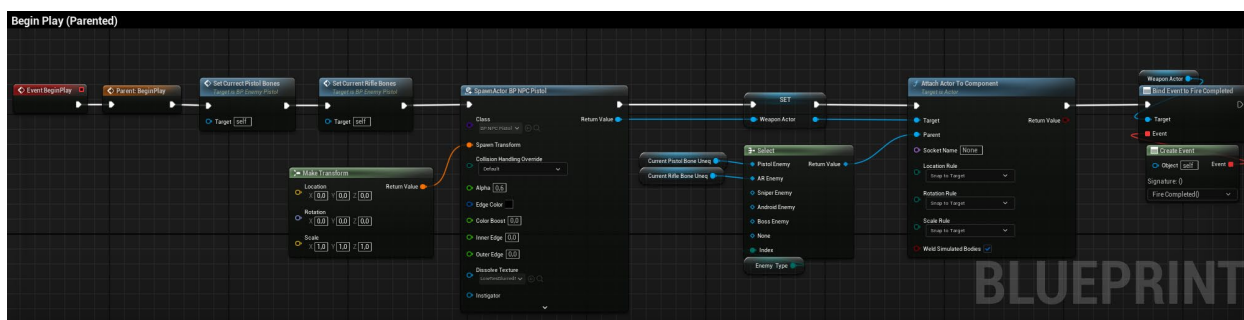
Για την επίλυση των παραπάνω ζητημάτων και τη διατήρηση ρεαλιστικής τοποθέτησης των όπλων, δημιουργήθηκαν SceneComponents στο Parent Blueprint, τα οποία λειτουργούν ως placeholders για τα όπλα. Το περιεχόμενο τους ορίζεται από τα Sockets που δημιουργήθηκαν σε κάθε διαφορετικό σκελετό, αντικατοπτρίζουν τη σωστή θέση και κατεύθυνση του όπλου, σύμφωνα με τη γεωμετρία του Agent. Οι αντίστοιχες functions, που δημιουργούνται στο Parent Blueprint αλλά υλοποιούνται σε κάθε Child Blueprint, επιτρέπουν την εκχώρηση του κατάλληλου socket σε κάθε SceneComponent. Με αυτό τον τρόπο, κάθε προσάρτηση οπλισμού χρησιμοποιεί ακριβής τοποθεσίες, οι οποίες ορίζονται στον πρόσθετο σκελετό του χαρακτήρα.

Συγκεκριμένα, τα sockets WeaponType_Equip_Socket και WeaponType_Unequip_Socket δημιουργήθηκαν στα κατάλληλα bones κάθε διαφορετικού σκελετού. Ενδεικτικά, για τον Agent τύπου Pistol, το socket Pistol_Equip_Socket προστέθηκε στο bone hand_r και ορίζει την τοποθεσία και τον προσανατολισμό του Pistol όταν ο Agent το κρατά στα χέρια του. Ακολουθώντας αυτή την προσέγγιση, κάθε διαφορετικός σκελετός διαθέτει 4 διαφορετικά sockets, που αφορούν στη θέση όπλισης και απόπλισης των όπλων Pistol και Rifle. Ως αποτέλεσμα, η ανάθεση των sockets στα SceneComponents του Parent Blueprint απαιτεί την εφαρμογή μιας γενίκευσης για την αποτροπή επαναλαμβανόμενων εντολών ίδιου τύπου, καθώς κάθε σκελετός έχει το δικό του, μοναδικό σε τοποθεσία, socket.



Εικόνα 52. Weapon Sockets Setup (child blueprint)

Ως εκ τούτου, δημιουργήθηκαν τέσσερις μεταβλητές τύπου SceneComponent στο Parent Blueprint BP_Enemy_Base και χρησιμοποιήθηκαν ως σημεία όπλισης (CurrentPistolBoneEq και CurrentRifleBoneEq) και αφόπλισης (CurrentPistolBoneUneq και CurrentRifleBoneUneq) για κάθε τύπου Actor κατά την εκτέλεση λειτουργιών στο Blueprint. Οι μεταβλητές αυτές ορίζονται μέσω των functions SetCurrentPistolBones και SetCurrentRifleBones, οι οποίες εκτελούνται κατά το Begin Play και ανακτούν τις κατάλληλες πληροφορίες από τα sockets του εκάστοτε σκελετού. Στη συνέχεια, ο Weapon Actor αποθηκεύεται στη μεταβλητή WeaponActor, ώστε να είναι διαθέσιμος για χρήση σε διάφορες λειτουργίες του Blueprint. Η προσάρτηση του όπλου στο χαρακτήρα γίνεται με τη χρήση του node AttachActorToComponent, και η θέση που χρησιμοποιείται επιλέγεται δυναμικά βάσει της μεταβλητής Enemy_Type. Ενδεικτικά, στην εικόνα 52 απεικονίζονται τα functions που αναθέτουν κατάλληλα τις μεταβλητές των placeholders, για έναν από τους child actors των Agents τύπου Pistol. Ακόμη, στην εικόνα 53 παρουσιάζεται το σχετικό διάγραμμα ανάθεσης του οπλισμού, για τους Agents τύπου Pistol.



Εικόνα 53. Weapon Attachment

Παρότι η μέθοδος απαιτεί ορισμό μοναδικών sockets για κάθε διαφορετικό σκελετό, αυτή η προσέγγιση προτιμήθηκε από την αξιοποίηση των sockets του SK_Mannequin. Η χρήση custom sockets ανά σκελετό εξασφαλίζει υψηλότερη ακρίβεια και συνέπεια κατά την απεικόνιση των animations και της κινησιολογίας των Agents, ενισχύοντας τον ρεαλισμό του gameplay.

7. Artificial Intelligence

Η ανάπτυξη της Τεχνητής Νοημοσύνης (AI) των Agents αποτελεί τον βασικό πυλώνα του παρόντος project, με στόχο τη δημιουργία εχθρικών χαρακτήρων που επιδεικνύουν έξυπνη και προσαρμοστική συμπεριφορά στο εκάστοτε σενάριο. Οι Agents αυτοί αντιδρούν δυναμικά τόσο στις ενέργειες του παίκτη όσο και στις μεταβολές του περιβάλλοντος, λαμβάνοντας αποφάσεις που αποτυπώνουν ένα ρεαλιστικό πρότυπο τεχνητής συμπεριφοράς. Το κεφάλαιο αυτό εστιάζει στην παρουσίαση των τεχνικών και εργαλείων που χρησιμοποιήθηκαν στο Unreal Engine 5 για την υλοποίηση αυτής της συμπεριφοράς, καθώς και στις ειδικές παραμέτρους που καθορίστηκαν ώστε να επιτευχθεί συνέπεια, ομαλότητα και ποικιλία στην αλληλεπίδραση μεταξύ Agents και παίκτη.

Στο πρώτο υποκεφάλαιο, παρατίθενται οι στόχοι που τέθηκαν για τις συμπεριφορικές ενέργειες των Agents, καλύπτοντας τις διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθούν – από την περιπολία και την αναγνώριση, έως την επίθεση και την εξουδετέρωση. Στη συνέχεια, στο υποκεφάλαιο των προαπαιτούμενων, περιγράφονται αναλυτικά τα βασικά συστατικά της AI στο Unreal Engine 5, όπως τα Behavior Trees, το AI Controller, το AI Perception Component, καθώς και το Navigation Mesh και το Environmental Query System (EQS), παρέχοντας τόσο θεωρητική επεξήγηση όσο και πρακτικές λεπτομέρειες χρήσης τους. Τέλος, το τρίτο υποκεφάλαιο εξετάζει την υλοποίηση των συστημάτων αυτών στο πλαίσιο του project: αναλύονται οι επιμέρους τιμές και ρυθμίσεις που εφαρμόστηκαν για κάθε εργαλείο (π.χ. πεδίο αντίληψης όρασης και ακοής, χρόνοι αντίδρασης, queries πλοήγησης), ενώ επεξηγούνται οι λόγοι πίσω από την επιλογή κάθε παραμέτρου. Η ενορχήστρωση των παραπάνω συστημάτων καθιστά δυνατό τον συγχρονισμό της συμπεριφοράς με τις αφηγηματικές και gameplay ανάγκες του παιχνιδιού, συνθέτοντας μία ρεαλιστική και στρατηγικά πλούσια εμπειρία AI.

7.1. Στόχοι

Η παρούσα εργασία έχει ως στόχο την ανάπτυξη NPC χαρακτήρων με προσαρμοστική και ρεαλιστική συμπεριφορά, η οποία ανταποκρίνεται στις δυναμικές συνθήκες του περιβάλλοντος και στις ενέργειες του παίκτη ή των συμμάχων του. Οι επιμέρους στόχοι του project διαμορφώνονται ως εξής:

- Υλοποίηση Πολυκαταστασιακής Συμπεριφοράς μέσω States

Οι Agents θα πρέπει να υποστηρίζουν λειτουργικές καταστάσεις (States) που αντανakλούν διαφορετικά επίπεδα επίγνωσης και εχθρικής προδιάθεσης: Passive, Investigating, Detecting, Seeking, Attacking και Dead. Κάθε κατάσταση συνοδεύεται από ένα πλήρες σύνολο ενεργειών που καθορίζουν τη συμπεριφορά του NPC, σύμφωνα με τις μεταβαλλόμενες συνθήκες του gameplay.

- Ανάπτυξη Εξειδικευμένων Μηχανισμών Αντίδρασης ανά Τύπο Agent

Στόχος είναι οι Agents να διαφοροποιούνται ανάλογα με τον τύπο τους (π.χ. Ranged ή Android), υιοθετώντας μοναδικές επιθετικές συμπεριφορές, τρόπους αντίδρασης και απαιτήσεις αντιμετώπισης από τον παίκτη. Οι διαφοροποιήσεις αυτές ενισχύουν τον στρατηγικό σχεδιασμό και τη διαδραστικότητα.

- Ενσωμάτωση Δυναμικών Ερεθισμάτων και Οπτικοακουστικής Ανατροφοδότησης

Το project επιδιώκει την αναγνώριση και αντίδραση των Agents σε ακουστικά ή οπτικά ερεθίσματα. Παράλληλα, στοχεύει στην ενίσχυση της εμπύθισης του παίκτη με την υποστήριξη οπτικοακουστικών εφέ και animation transitions που αντικατοπτρίζουν την εκάστοτε κατάσταση του Agent.

- Σχεδιασμός Ενεργειών Ειδικής Αλληλεπίδρασης με τον Παίκτη

Περιλαμβάνονται ειδικές μηχανικές όπως η κίνηση Takedown για αιφνιδιαστική εξουδετέρωση, καθώς και η χρήση αντικειμένων (Bottle, DataWorm) με διακριτή επίδραση στους διαφορετικούς τύπους Agents. Αυτές οι ενέργειες ενισχύουν τη διαδραστικότητα και τον πειραματισμό του παίκτη εντός του κόσμου του παιχνιδιού.

- Αξιοποίηση Προηγμένων Εργαλείων του Unreal Engine 5 για Ρεαλιστική Συμπεριφορά

Ο σχεδιασμός και η υλοποίηση βασίζεται στην ενσωμάτωση και συνεργασία εργαλείων όπως Behavior Trees, AI Controllers, EQS Queries και Custom Tasks, ώστε να διασφαλιστεί ρεαλιστική λήψη αποφάσεων, συνεχής αξιολόγηση των συνθηκών και άμεση μετάβαση σε νέα States με συνέπεια και σταθερότητα.

- Διασφάλιση Ανθεκτικής Δομής AI και Κλιμακούμενης Πολυπλοκότητας

Η TN των Agents οργανώνεται σε αναπτυξιακά στιβαρή αρχιτεκτονική, με καθαρό διαχωρισμό μεταξύ των States, χρήση Subtrees για επαναχρησιμοποιήσιμες ενέργειες, και προτεραιοποίηση μέσω Decorators ώστε να εξασφαλίζεται η ορθή ιεράρχηση των αντιδράσεων. Η μετάβαση μεταξύ των States πραγματοποιείται μέσω εσωτερικών μεταβλητών και ελέγχων, διατηρώντας τη σταθερότητα της συμπεριφοράς υπό όλες τις συνθήκες.

7.2. Προαπαιτούμενα

Για την ανάπτυξη τεχνητής νοημοσύνης των Agents, αξιοποιούνται ποικίλα συστήματα και στοιχεία τα οποία συνεργάζονται αρμονικά για την επίτευξη ρεαλιστικών και προσαρμοστικών συμπεριφορών. Παρακάτω παρατίθενται οι βασικοί ορισμοί και οι λειτουργικές σχέσεις των σημαντικότερων από αυτά σύμφωνα με το documentation της Epic Games:

Behavior Trees: Το Behavior Tree είναι μια δομή λήψης αποφάσεων που επιτρέπει την ιεραρχική οργάνωση και εκτέλεση συμπεριφορών σε NPCs. Πρόκειται για ένα γραφικό εργαλείο που επιτρέπει τον προγραμματισμό της AI μέσω κόμβων (nodes) οι οποίοι καθορίζουν λογικές ενέργειες ή αποφάσεις.

Βασικά στοιχεία:

1. Composite Nodes

Οι κόμβοι αυτοί είναι υπεύθυνοι για τον έλεγχο της ροής εκτέλεσης μεταξύ των παιδιών τους (child nodes).

Selector: εκτελεί child nodes από αριστερά προς τα δεξιά μέχρι να βρει ένα που να επιτυγχάνει. Κατά την επιτυχία ενός child node διακόπτεται με επιτυχία, παραλείποντας τα υπόλοιπα child nodes. Αν κάθε child node αποτύχει τότε ο Selector αποτυγχάνει.

Sequence: εκτελεί child nodes από αριστερά προς τα δεξιά. Εάν κάποιο child node αποτύχει η εκτέλεση διακόπτεται και το Sequence επιστρέφει το αντίστοιχο αποτέλεσμα.

2. Task Nodes (Leaf Nodes)

Πρόκειται για τα φύλλα του δέντρου, τα οποία εκτελούν ενέργειες ή ελέγχουν συγκεκριμένες καταστάσεις. Παραδείγματα αποτελούν τα:

MoveTo: εντολή μετακίνησης ενός χαρακτήρα σε συγκεκριμένη θέση.

SimpleParallel: εκτελεί παράλληλα δύο nodes του δέντρου, ένα ως κύρια ενέργεια και ένα ως δευτερεύουσα.

Custom Tasks: Εξατομικευμένα tasks των χρηστών που ορίζουν συγκεκριμένες λογικές, όπως η αναπαραγωγή ενός animation ή ο έλεγχος μιας μεταβλητής.

3. Decorator Nodes

Περιβάλλουν άλλους κόμβους και λειτουργούν ως φίλτρα που καθορίζουν πότε ένας κόμβος μπορεί να εκτελεστεί. Για παράδειγμα, μπορούν να εφαρμόζουν έλεγχο συνθηκών πριν την εκτέλεση ενός Task ή να επαναλαμβάνουν ένα Task μέχρι να επιτευχθεί συγκεκριμένο αποτέλεσμα.

4. Service Nodes

Συνδέονται με κόμβους και εκτελούν περιοδικές ή επαναλαμβανόμενες λειτουργίες όσο ο κόμβος είναι ενεργός. Συχνά χρησιμοποιούνται για την παρακολούθηση του περιβάλλοντος (π.χ. ανίχνευση εχθρών, ενημέρωση δεδομένων στο Blackboard).

5. Subtrees

Επιτρέπουν την ομαδοποίηση συμπεριφορών σε επιμέρους δέντρα, τα οποία μπορούν να επαναχρησιμοποιηθούν σε διαφορετικά σημεία του κύριου Behavior Tree ή σε άλλα Behavior Trees, παρέχοντας επαναχρησιμοποίηση των ενεργειών και οργάνωση.

Blackboard: Το Blackboard αποτελεί ένα δομημένο σύνολο μεταβλητών, το οποίο λειτουργεί ως πίνακας δεδομένων για τη διαχείριση κρίσιμων πληροφοριών που σχετίζονται με τη συμπεριφορά ενός

AI Agent στο πλαίσιο ενός Behavior Tree. Μέσω αυτού, οι πράκτορες τεχνητής νοημοσύνης αποθηκεύουν και ανακτούν δεδομένα απαραίτητα για τη λήψη αποφάσεων και τον προγραμματισμό της επόμενης ενέργειάς τους. Τα δεδομένα αυτά μπορεί να περιλαμβάνουν θέσεις στο χώρο, στόχους, καταστάσεις, και γενικότερα οποιαδήποτε πληροφορία απαιτείται για την υποστήριξη δυναμικής και προσαρμοστικής συμπεριφοράς.

Η πρόσβαση στο Blackboard είναι δυνατή ανά πάσα στιγμή, γεγονός που επιτρέπει την άμεση ανταπόκριση του AI σε αλλαγές του περιβάλλοντος ή εσωτερικών συνθηκών. Η ευελιξία αυτή καθιστά το Blackboard βασικό εργαλείο για την ανάπτυξη ρεαλιστικής και διαδραστικής συμπεριφοράς, ιδιαίτερα σε συστήματα όπου η κατάσταση του Agent ή του περιβάλλοντος μπορεί να μεταβάλλεται συνεχώς.

Το Blackboard αποτελείται κυρίως από δύο βασικά στοιχεία:

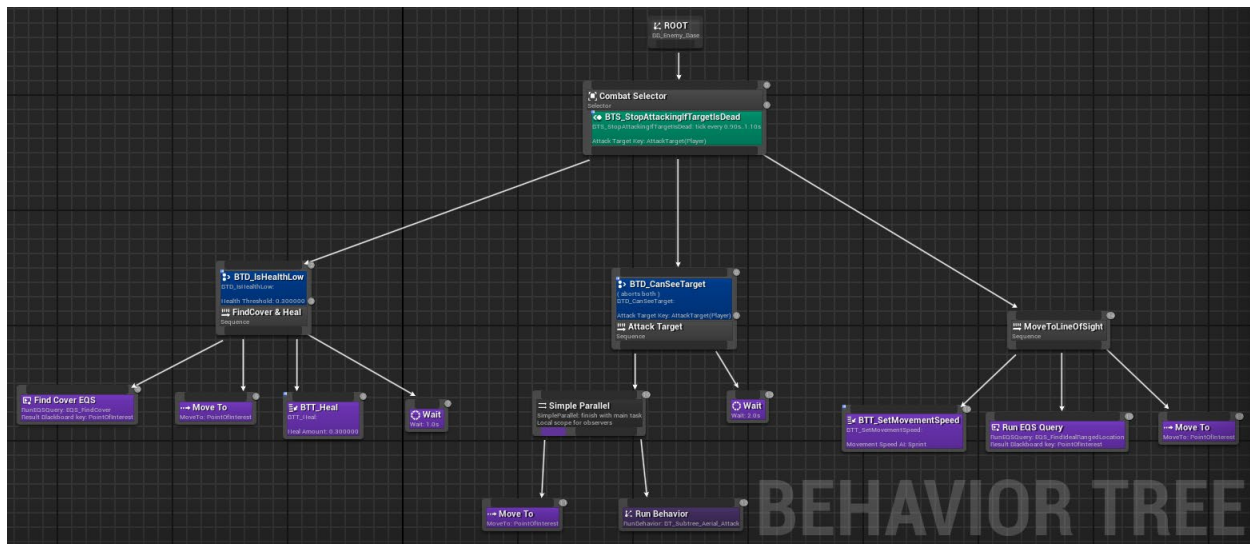
- *Blackboard Keys*: Πρόκειται για τις επιμέρους μεταβλητές που χρησιμοποιούνται εντός των Behavior Trees κατά την εκτέλεσή τους. Κάθε Key αντιστοιχεί σε μία πληροφορία και χαρακτηρίζεται από συγκεκριμένο τύπο δεδομένων, όπως Vector, Boolean, Object, Enum κ.ά. Ενδεικτικά παραδείγματα περιλαμβάνουν το TargetLocation (Vector), το οποίο καθορίζει τη θέση ενός στόχου στο χώρο, και το IsPlayerVisible (Boolean), το οποίο δηλώνει αν ο παίκτης είναι ορατός στον Agent. Τα Keys αυτά μπορούν να δημιουργηθούν, να διαβαστούν ή να τροποποιηθούν από διαφορετικά σημεία του project, όπως Blueprints ενός Actor, του AI Controller, του Behavior Tree ή ακόμη και μέσω ερεθισμάτων από το περιβάλλον.
- *Blackboard Component*: Πρόκειται για component του συστήματος που είναι υπεύθυνο για τη διαχείριση των Blackboard Keys. Αποτελεί ουσιαστικά το Asset που συνδέεται με έναν Actor ή ένα AI Controller και παρέχει το απαιτούμενο interface για την πρόσβαση, τροποποίηση ή ενημέρωση των δεδομένων του Blackboard. Κάθε Agent που σχετίζεται με ένα Behavior Tree περιέχει και διαθέτει πρόσβαση σε ένα Blackboard Component, το οποίο λειτουργεί ως σύνδεσμος ανάμεσα στη λογική της συμπεριφοράς και τα δεδομένα που την καθοδηγούν.

Blackboard Condition: Το Blackboard Condition αποτελεί ένα βασικό Decorator, ο οποίος επιτρέπει την αξιολόγηση των τιμών που είναι αποθηκευμένες στα Keys του Blackboard. Μέσω αυτών, καθορίζεται αν μία ενέργεια ή μια ακολουθία ενεργειών πρέπει να εκτελεστεί, με βάση την τρέχουσα κατάσταση του Agent, του παιχνιδιού ή του περιβάλλοντος. Οι συνθήκες αυτές μπορούν να εφαρμοστούν σε διάφορους κόμβους του Behavior Tree, όπως στους κόμβους ελέγχου ροής (Selector, Sequence) και σε κόμβους ενεργειών (Tasks). Με αυτό τον τρόπο επηρεάζεται η συνολική ροή εκτέλεσης και επιτυγχάνεται δυναμική και αποδοτική προσαρμογή της συμπεριφοράς των Agents στις εκάστοτε συνθήκες.

Η λειτουργικότητα των Blackboard Conditions ενισχύεται μέσω διαφόρων ρυθμίσεων, οι οποίες παρέχουν λεπτομερή έλεγχο πάνω στη συμπεριφορά της τεχνητής νοημοσύνης:

- *Notify Observer*: Η ρύθμιση αυτή καθορίζει το πότε ο Observer θα ενημερωθεί για αλλαγές στην τιμή ενός Blackboard Key. Μια συνηθισμένη επιλογή είναι το On Result Change, το οποίο δηλώνει ότι ο Observer θα ενεργοποιείται μόνο όταν αλλάξει το αποτέλεσμα της συνθήκης, δηλαδή όταν η νέα τιμή του Key επιφέρει διαφορετικό αποτέλεσμα από το προηγούμενο. Αυτή η λειτουργία είναι κρίσιμη όταν απαιτείται άμεση αντίδραση των Agents σε μεταβολές καταστάσεων (π.χ. αλλαγή από Passive σε Investigate), ώστε να προκληθεί εκ νέου αξιολόγηση του δέντρου συμπεριφοράς.
- *Observer Aborts*: Αυτή η παράμετρος καθορίζει πώς το Behavior Tree θα αντιδράσει όταν ανιχνευθεί αλλαγή στην τιμή ενός Blackboard Key που παρακολουθείται. Η επιλογή *Both* υποδεικνύει ότι τόσο τα ενεργά όσο και τα πιθανά επόμενα branches του δέντρου θα ακυρωθούν, και η εκτέλεση θα επανεκκινηθεί από το ανώτερο σχετικό σημείο. Με αυτόν τον τρόπο διασφαλίζεται η προτεραιότητα των νέων δεδομένων και αποτρέπεται η συνέχιση παλαιών ενεργειών που μπορεί να είναι πλέον ασύμβατες με τη νέα κατάσταση.
- *Key Query*: Ορίζει το είδος της λογικής σύγκρισης που θα πραγματοποιηθεί μεταξύ της τρέχουσας τιμής ενός Blackboard Key και μιας προκαθορισμένης τιμής. Η πιο συνήθης επιλογή είναι το *Is Equal To*, το οποίο αξιολογεί αν η τιμή του Key είναι ίση με την καθορισμένη τιμή (π.χ. αν η μεταβλητή κατάστασης του AI είναι Investigate). Εφόσον η συνθήκη αποδώσει *True*, επιτρέπεται η εκτέλεση του αντίστοιχου υποδέντρου ή ενέργειας.
- *Key Value*: Πρόκειται για τη σταθερή τιμή με την οποία συγκρίνεται η τιμή του Blackboard Key, με βάση το είδος του *Key Query*. Για παράδειγμα, αν το Key είναι AIState και το Key Value είναι Investigate, η συνθήκη ελέγχει αν ο Agent βρίσκεται στη φάση διερεύνησης.
- *Blackboard Key*: Το συγκεκριμένο πεδίο δηλώνει ποιο Key του Blackboard συμμετέχει στη σύγκριση. Πρόκειται για τη μεταβλητή που αποθηκεύει την κρίσιμη πληροφορία η οποία παρακολουθείται από το Condition, όπως η θέση ενός στόχου, η ορατότητα του παίκτη, ή η τρέχουσα λειτουργική κατάσταση του πράκτορα AI.

Διάγραμμα Ροής: Κάθε node τοποθετείται στο διάγραμμα ροής τοποθετώντας τις ενέργειες που θα εκτελεστούν σε φθίνουσα σειρά προτεραιότητας. Το διάγραμμα της εικόνας 54 παρουσιάζει μια ακολουθία ενεργειών σε ένα σενάριο επίθεσης χρησιμοποιώντας το σύνολο των βασικών στοιχείων ενός Behavior Tree. Σε αυτό το διάγραμμα ο Agent έχει αντιληφθεί τον παίκτη και μπορεί να εκτελέσει τρεις βασικές ενέργειες, αυτή της επίθεσης, της αναπλήρωσης των πόντων ζωής και αυτή κατά την οποία αναζητά τον παίκτη. Η εκτέλεση ξεκινά από τη ρίζα (root) του δέντρου και ο Combat Selector επιχειρεί να εκτελέσει ένα από τα child nodes με τη σειρά, σύμφωνα με τους Decorators που περιλαμβάνουν.



Εικόνα 54. Behavior Tree Nodes Example (attack scenario)

Sequence FindCover and Heal: περιλαμβάνει την αναζήτηση μιας τοποθεσίας ασφάλειας μέσω EQS Query, τη μετακίνηση του παίκτη στο σημείο μέσω του task MoveTo, την αναπλήρωση των πόντων ζωής σύμφωνα με το Custom Task BTT_Heal και την αναμονή του Agent στο σημείο αυτό μέσω του Wait, κατά την οποία το Sequence ολοκληρώνεται. Ενεργοποιείται όταν ο Decorator αλλάξει τιμή, δηλαδή όταν οι πόντοι ζωής του Agent βρεθούν σε χαμηλά επίπεδα. Μια τέτοια αλλαγή προκαλεί διακοπή κάθε ενέργειας χαμηλότερης προτεραιότητας και επαναξιολόγηση των κόμβων από την αρχή, στο επίπεδο του Combat Selector.

Attack Sequence: περιλαμβάνει τις ενέργειες επίθεσης, όπου ο Agent εκτελεί τη δέσμη ενεργειών μέσω ενός SubTree (όπως πυροδότηση του οπλισμού) ενώ παράλληλα κινείται προς ένα σημείο. Αυτό επιτυγχάνεται με τη χρήση του node Simple Parallel. Η ολοκλήρωση του node αυτού θα πυροδοτήσει την επόμενη ενέργεια Wait. Το Sequence ενεργοποιείται και εκτελείται καθ' όλη τη διάρκεια που ο Decorator BTD_CanSeeTarget είναι αληθής, αντιπροσωπεύοντας την οπτική κατάσταση μεταξύ του παίκτη και του Agent.

MoveToLineOfSight Sequence: περιλαμβάνει την αλλαγή της ταχύτητας με την οποία κινείται ο Agent μέσω Custom Task, την αναζήτηση μιας τοποθεσίας στην οποία ο παίκτης θα βρίσκεται στο οπτικό του πεδίο μέσω EQS Query και τη μετακίνηση του Agent προς το σημείο αυτό. Ενεργοποιείται όταν κάθε άλλο Sequence του Combat Selector αποτύχει, δηλαδή στην περίπτωση που ο Agent έχει επαρκείς πόντους ζωής και δεν έχει οπτική επαφή με τον παίκτη.

Τέλος, ο Selector θα διακοπεί στην περίπτωση που ο παίκτης εξουδετερωθεί, σύμφωνα με το Service BTS_StopAttackingIfTargetIsDead, το οποίο εκτελείται καθ' όλη τη διάρκεια της εκτέλεσης, ελέγχοντας τη ζωντική κατάσταση του παίκτη.

AI Controller: Το AI Controller αποτελεί κεντρικό στοιχείο του συστήματος AI και λειτουργεί ως το νοητικό σύστημα ενός AI-controlled Actor. Πρόκειται για μια ειδική κλάση η οποία συνδέεται με έναν Actor και αναλαμβάνει την καθοδήγηση και τον έλεγχο της συμπεριφοράς του, μέσω τόσο ενσωματωμένων όσο και εξατομικευμένων λειτουργιών που αναπτύσσονται από τον χρήστη. Το AI Controller διαχειρίζεται την εκκίνηση του Behavior Tree, ενώ ταυτόχρονα διατηρεί πρόσβαση στο Blackboard. Επιπλέον, ενσωματώνει το σύστημα AI Perception, επιτρέποντας στους Agents να αντιλαμβάνονται το περιβάλλον τους και είναι υπεύθυνο για τη διαχείριση της κίνησης και της εστίασης του Agent, ελέγχοντας σε πραγματικό χρόνο την πλοήγηση και την κατεύθυνσή του προς στόχους ή αντικείμενα ενδιαφέροντος. Ως Blueprint κλάση, επιτρέπει την περαιτέρω επέκταση της λειτουργικότητάς του μέσω της δημιουργίας custom functions και της χρήσης Blueprint Interfaces, καθιστώντας δυνατή την ανάπτυξη δυναμικών και προσαρμοστικών Agents, οι οποίοι ανταποκρίνονται αποτελεσματικά σε μεταβαλλόμενες συνθήκες του περιβάλλοντος και υποστηρίζουν πολύπλοκες συμπεριφορές.

AI Perception Component: Το AI Perception Component είναι ένα σύστημα που παρέχει έναν γενικό μηχανισμό αντίληψης σε έναν Agent. Επιτρέπει στον Agent να αντιλαμβάνεται το περιβάλλον γύρω του μέσω διαφόρων αισθήσεων (senses). Αυτές οι αισθήσεις επιτρέπουν σε έναν Agent να αντιδράσει σε γεγονότα του κόσμου του, όπως να αντιληφθεί την παρουσία του παίκτη ή άλλων Agents, να ανιχνεύσει ήχους ή ακόμα και να αντιδράσει σε Damage. Το σύστημα αυτό ενσωματώνεται άμεσα στο AI Controller και χρησιμοποιείται για τη δημιουργία έξυπνων και προσαρμοστικών NPCs. Οι ρυθμίσεις και οι λειτουργίες του Perception System περιλαμβάνουν τρεις βασικούς τύπους αισθήσεων όπως ακοή (Hearing Sense), όραση (Sight Sense) και η αντίληψη damage (Damage Sense). Αυτές μπορούν να ρυθμιστούν και να διαμορφωθούν για να επιτευχθούν διαφορετικά αποτελέσματα στις αντιδράσεις του Agent. Οι βασικές αισθήσεις που παρέχονται είναι:

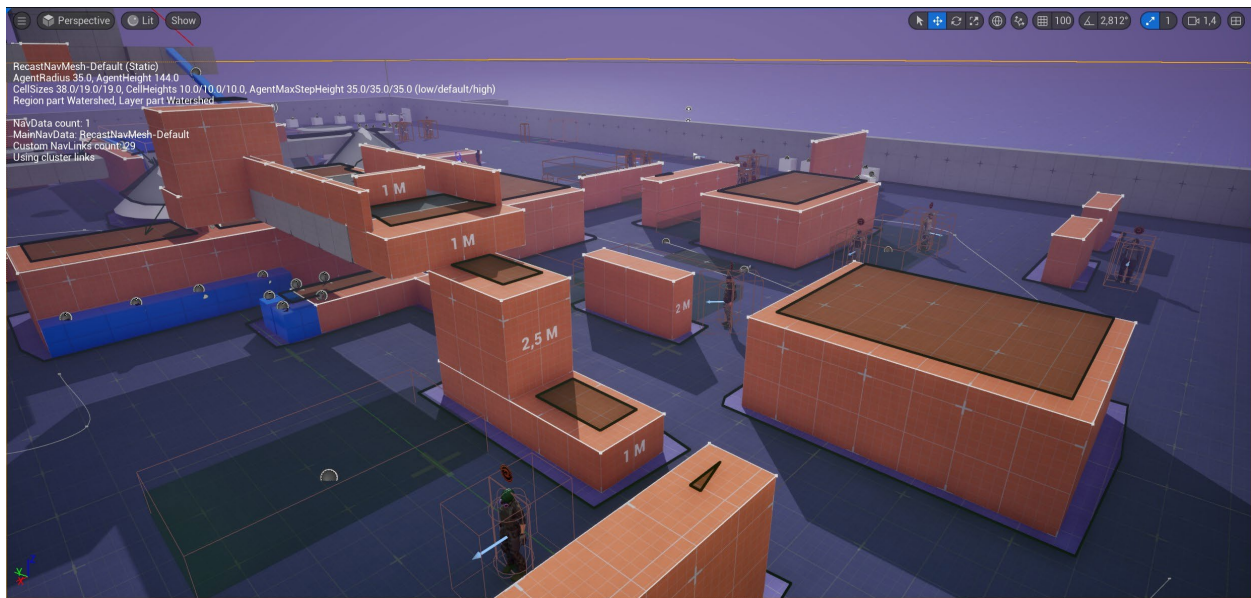
- *Sense Sight:* επιτρέπει την ανίχνευση ενός Actor (π.χ. του παίκτη ή κάποιου άλλου Agent) ή άλλων αντικείμενων (π.χ. ένα μπουκάλι) στον κόσμο του μέσω της όρασης. Διαθέτει ρυθμίσεις όπως το Sight Radius, το οποίο καθορίζει τη μέγιστη απόσταση στην οποία μπορεί ο Agent να δει έναν στόχο, το Lose Sight Radius στο οποίο καθορίζεται η απόσταση μετά την οποία ο AI χάνει την επαφή του με τον στόχο και το Peripheral Vision Angle όπου αναπαριστά τη γωνία όρασης του Agent, η οποία επηρεάζει την περιοχή στην οποία μπορεί να ανιχνεύσει ένα αντικείμενο. Επιπλέον, το Sense διαθέτει τις ρυθμίσεις Max Age και Starts Enabled, οι οποίες αφορούν στη χρονική διάρκεια που ένα ερέθισμα θα διατηρηθεί στη μνήμη του Agent και στην αρχικοποίηση του Agent με το sense ενεργοποιημένο αντίστοιχα. Όσον αφορά στη λειτουργία του, ένας Actor που εισέρχεται στο πεδίο όρασης του Agent αποτελεί και ένα ερέθισμα, το οποίο καταγράφεται από το Sense και μπορεί να χρησιμοποιηθεί εντός οποιασδήποτε άλλης λειτουργίας, όπως για παράδειγμα κατά την πυροδότηση της λειτουργίας OnPerceptionUpdated η οποία αναφέρεται σε επόμενη παράγραφο. Τέλος, οι ρυθμίσεις

Detection by Affiliation επιτρέπουν την ανίχνευση Actors βάσει των χαρακτηριστικών που καθορίζουν τη σχέση τους, όπως tags ή μεταβλητές. Χρησιμοποιούν σε περιβάλλοντα παιχνιδιών όπου υπάρχουν ομάδες χαρακτήρων που ενδέχεται να ανήκουν σε διαφορετικές συμμαχίες, ομάδες ή άλλες οργανώσεις εντός του παιχνιδιού.

- *Sense Hearing*: επιτρέπει σε έναν Agent να ακούει ήχους, όπως πυροβολισμούς, βήματα ή άλλες δραστηριότητες και να αντιδρά. Λειτουργεί με τη βάση του Hearing Range, το οποίο καθορίζει την απόσταση από την οποία ο ήχος μπορεί να γίνει αντιληπτός, ώστε το ερέθισμα να καταγραφεί από τον Agent. Επιπλέον, διαθέτει τις επιλογές MaxAge, DetectionByAffiliation και Starts Enabled, οι οποίες λειτουργούν με τον ίδιο τρόπο με αυτόν του Sense Sight. Η αίσθηση ενεργοποιείται κατά την κλήση του node Report Noise Event, το οποίο μπορεί να εκτελεστεί από οποιονδήποτε Actor ή σημείο ενός Blueprint. Ενδεικτικά, αυτό μπορεί να συμβεί κατά την πυροδότηση μιας σφαίρας από το Blueprint ενός όπλου ή την καταστροφή ενός μπουκαλιού κατά την επαφή του με το έδαφος. Τέλος, το node παρέχει εισόδους μεταβλητών όπως Noise Location, Instigator και Tag, οι οποίες αντιπροσωπεύουν την τοποθεσία του ερεθίσματος, τον Actor από τον οποίο κλήθηκε το node και ένα προαιρετικό όνομα που χαρακτηρίζει το ερέθισμα. Με αυτόν τον τρόπο, κάθε κλήση του node θα καταγράψει το ερέθισμα, αλλά και τις πληροφορίες που σχετίζονται με αυτό, ώστε να χρησιμοποιηθούν στην υλοποίηση διαφόρων ενεργειών, όπως κατά την πυροδότηση της λειτουργίας OnPerceptionUpdated.
- *Sense Damage*: επιτρέπει σε έναν Agent να αντιλαμβάνεται την αποδοχή Damage από τον παίκτη ή άλλους εχθρούς, επιτρέποντας την ανάπτυξη ανάλογων αντιδράσεων. Ένα ερέθισμα που ενεργοποιεί αυτή την αίσθηση δημιουργείται κατά την κλήση του node Report Damage Event, το οποίο λειτουργεί με τον ίδιο τρόπο με το node Report Noise Event. Κάθε κλήση παρέχει μεταβλητές εισόδου που αφορούν στο Damage, όπως Damaged Actor, Insigator, Damage Amount, Event Location και Tag και μπορούν να χρησιμοποιηθούν κατά την πυροδότηση του Sense στον AI Controller.

Η ρύθμιση του συστήματος γίνεται μέσω του AI Perception Component εντός του AI Controller και παρέχει βασικά events στα οποία μπορούν να αναπτυχθούν λειτουργίες βάσει των καταγεγραμμένων ερεθισμάτων ή διαφόρων συνθηκών. Συγκεκριμένα, το event OnPerceptionUpdated καλείται κάθε φορά που το AI Perception Component ανιχνεύσει νέα δεδομένα. Αυτός ο μηχανισμός ενεργοποιείται όταν ένα από τα Senses πυροδοτηθούν (π.χ. Sense Sight) και επιτρέπει την ανάπτυξη των ενεργειών που θα εκτελεστούν σύμφωνα με τον τύπο του ερεθίσματος και άλλες συνθήκες ή μεταβλητές. Επιπλέον, το Component διαθέτει Events, όπως το OnTargetPerceptionForgotten, το OnTargetPerceptionInfoUpdated, το OnComponentActivated κ.α., για την ανάπτυξη επιπλέον λειτουργιών βάσει συγκεκριμένων συνθηκών που αφορούν στο Component. Χαρακτηριστική εφαρμογή αποτελεί το event OnTargetPerceptionForgotten, το οποίο πυροδοτείται όταν ένα ερέθισμα έχει αφαιρεθεί από τη μνήμη ενός Agent, σύμφωνα με τη μεταβλητή MaxAge που έχει οριστεί στο αντίστοιχο Sense.

Navigation Mesh: Το Navigation Mesh (NavMesh) ένα πλέγμα πλοήγησης που χρησιμοποιείται για να καθορίσει τις προσπελάσιμες περιοχές του παιχνιδιού από έναν Agent. Αναπαριστά τα διαθέσιμα πεδία για κίνηση, προσφέροντας στους χαρακτήρες έναν χάρτη πλοήγησης, ώστε να προγραμματίζουν τη διαδρομή τους αποτελεσματικά, για τη σωστή κατεύθυνση και αποφυγή εμποδίων κατά την κίνησή τους στον κόσμο του παιχνιδιού. Προστίθεται ως Asset (NavMesh Bounds Volume) στο Level του παιχνιδιού και με βάση τις γεωμετρικές επιφάνειες του κόσμου (όπως οι τοίχοι, το έδαφος, τα εμπόδια κλπ.) ένας NPC μπορεί να υπολογίσει την καλύτερη διαδρομή από ένα σημείο σε ένα άλλο, βασιζόμενος στην απόσταση και στα εμπόδια. Χρησιμοποιείται άμεσα από τις λειτουργίες MoveTo, Pathfinding, καθώς και από τις EQS Queries. Ως αποτέλεσμα, αποτελεί απαραίτητο asset για τη διασφάλιση της κινητικότητας των Agents στον κόσμο του παιχνιδιού. Η εικόνα 55 απεικονίζει τις προσπελάσιμες περιοχές του παιχνιδιού από έναν Agent σύμφωνα με το πλέγμα NavMesh.



Εικόνα 55. Navigation Mesh

Environment Query System: Το Environment Query System (EQS) αποτελεί μια ενσωματωμένη δυνατότητα του συστήματος Τεχνητής Νοημοσύνης στο Unreal Engine 5, η οποία επιτρέπει τη συλλογή και ανάλυση δεδομένων από το περιβάλλον του παιχνιδιού. Μέσω του EQS, μπορούν να διαμορφωθούν ερωτήματα που αξιολογούν διάφορες παραμέτρους του περιβάλλοντος, και να επιστρέψουν ως αποτέλεσμα ένα ή πολλά σημεία για την καθοδήγηση των Agents. Το αποτέλεσμα μπορεί να αποθηκευτεί σε μια μεταβλητή του Behavior Tree και να χρησιμοποιηθεί ως παράμετρος εισόδου σε διάφορα Tasks μετακίνησης ή επίθεσης. Ενδεικτικά, το αποτέλεσμα ενός EQS που αναζητά το κοντινότερο αντικείμενο από ένα σύνολο αντικειμένων σε σχέση με τη θέση του NPC, μπορεί να χρησιμοποιηθεί ως τοποθεσία στην οποία θα κινηθεί ο Agent μέσω του task MoveTo. Τα κύρια στοιχεία του είναι:

- **Generators:** Είναι υπεύθυνοι για τη δημιουργία υποψήφιων θέσεων ή αντικειμένων που θα υποβληθούν σε αξιολόγηση.
- **Tests:** Αξιολογούν τα δεδομένα που παρέχονται από τους Generators, χρησιμοποιώντας διάφορα κριτήρια, όπως απόσταση ή ορατότητα.
- **Contexts:** Παρέχουν το πλαίσιο αναφοράς για τις δοκιμές και τους Generators, διευκρινίζοντας το πεδίο εφαρμογής των ερωτημάτων. Ένα context μπορεί να είναι ο Querier (ο χαρακτήρας AI που εκτελεί την ερώτηση) ή πιο σύνθετο, όπως όλοι οι Actors ενός συγκεκριμένου τύπου. Επιπλέον, μπορούν να δημιουργηθούν προσαρμοσμένα Contexts μέσω Blueprint ή C++, επιτρέποντας την ευελιξία στον καθορισμό του πλαισίου αναφοράς ανάλογα με τις ανάγκες του παιχνιδιού.

Μια διαδικασία EQS παράγει πολλαπλές υποψήφιες θέσεις στον χώρο, μέσω των Generators και το αποτέλεσμα του Query προκύπτει από την αξιολόγηση των σημείων, σύμφωνα με τα κριτήρια που ορίζονται στα Test σε σχέση με το πλαίσιο αναφοράς. Μπορεί να εκτελεστεί μέσω του node Run EQS Query εντός των Behavior Trees και επιτρέπει στους Agents να λαμβάνουν αποφάσεις βάσει της δυναμικής αξιολόγησης του περιβάλλοντος. Στο πλαίσιο του παρόντος Project, χρησιμοποιούνται ως τεχνική για την εύρεση της βέλτιστης θέσης επίθεσης, την αναζήτηση κάλυψης, των πιθανών σημείων διαφυγής που μπορεί να έχει ακολουθήσει ο παίκτης αφότου χαθεί από το οπτικό πεδίο του Agent κ.α.

7.3. Υλοποίηση

Η διαδικασία υλοποίησης ξεκίνησε με τη δημιουργία των τριών βασικών Assets που αποτελούν τον πυρήνα της λειτουργίας των Agents.

Αρχικά, σχεδιάστηκε η κλάση Blackboard, στην οποία ορίστηκαν όλες οι απαραίτητες μεταβλητές για τον προγραμματισμό της ροής αποφάσεων των Behavior Trees. Στη συνέχεια, αναπτύχθηκε η κλάση AI Controller, όπου υλοποιήθηκαν τα functions και τα Custom Events που διαχειρίζονται την αλληλεπίδραση μεταξύ των Agents και των Behavior Trees. Το τρίτο βασικό στοιχείο αφορά την πλοήγηση των Agents στον χώρο του παιχνιδιού, η οποία επιτεύχθηκε με την ενσωμάτωση του NavMesh, επιτρέποντας στους Agents να μετακινούνται αυτόνομα στο περιβάλλον.

Για τη διαχείριση της συμπεριφοράς των εχθρών, δημιουργήθηκαν δύο βασικά BT assets που οδηγούν τις κύριες κατηγορίες εχθρών με κοινή λογική. Το ένα ανατέθηκε στους εχθρούς με οπλισμό Pistol και Rifle (BT_Enemy_Ranged) και περιλαμβάνει τη δομή που καθορίζει τις ενέργειες των Agents με επιθέσεις εμβέλειας, καθώς αυτές παραμένουν ίδιες σε επίπεδο γενικών αποφάσεων. Το άλλο ανατέθηκε στους εχθρούς Android (BT_Enemy_Android), καθώς οι ενέργειές τους διαφέρουν σημαντικά. Επιπλέον, για τις ανάγκες των δύο αυτών κλάσεων αναπτύχθηκαν τα ανάλογα Tasks, Decorators, Services, EQS Queries και SubTrees για την ορθή λειτουργία, οργάνωση και βελτιστοποίηση της συμπεριφοράς των Agents. Στη συνέχεια, δημιουργήθηκε ένα Blueprint Interface (BPI_Enemy_AI) που λειτουργεί ως συλλογή συναρτήσεων για κλήση κοινόχρηστων ενεργειών και ανάγνωση μεταβλητών, σε σημεία που το απαιτεί η υλοποίηση, όπως το Blueprint των Agents, το AI Controller, το Animation Blueprint κ.α.

Τέλος, έγινε ανάθεση των κατάλληλων Behavior Trees και AI Controller Assets σε κάθε Agent, ώστε να είναι προσβάσιμα και προς χρήση κατά την αρχικοποίησή τους.

7.3.1. Blackboard Keys

Οι μεταβλητές (keys) που ορίστηκαν στο Blackboard στοχεύουν στην αποθήκευση κρίσιμων πληροφοριών σχετικά με την κατάσταση, τη στόχευση και τις ενέργειες των Agents. Παρακάτω παρουσιάζεται η λειτουργία και η βασική χρήση της καθεμιάς από αυτές:

- *AttackTarget (Player)* (Actor Object reference): σε αυτή τη μεταβλητή εκχωρείται ο Actor που έχει προστεθεί στο οπτικό πεδίο του Agent και θα αποτελέσει στόχο ως προς μια ενέργεια. Χρησιμοποιείται για να υποδείξει τον Actor στον οποίο ο Agent θα ενεργήσει επιθετικά, αλλά και σε Decorators και Services που αφορούν στην αντίληψη του Agent ως προς την παρουσία ή όχι του Actor. Ενδεικτικά, οι ενέργειες επίθεσης εκτελούνται κατά τη διάρκεια κατά την οποία ο actor είναι αντιληπτός, ενώ στην αντίθετη περίπτωση ματαιώνονται.
- *State (Enum)*: αυτή μεταβλητή αντιπροσωπεύει τις διαφορετικές καταστάσεις που μπορεί να βρίσκεται ο Agent, όπως Patrol, Investigate, Attack, κ.α. Χρησιμοποιείται για την επιλογή του κατάλληλου δέντρου ενεργειών από το Behavior Tree και σε ελέγχους για την πραγματοποίηση ενεργειών που απαιτούν συγκεκριμένο State. Για παράδειγμα, όταν η μεταβλητή παίρνει την τιμή Investigate θα εκτελεστούν οι ενέργειες του δέντρου που αφορούν στη διερεύνηση μιας ηχητικής διατάραξης. Επιπλέον, το περιεχόμενο της μεταβλητής καθορίζει την επιλογή του κατάλληλου Blendspace στο Animation Blueprint.
- *PointOfInterest (Vector)*: σε αυτή τη μεταβλητή εκχωρείται μια τοποθεσία η οποία θα αποτελέσει τον χωρικό στόχο μιας ενέργειας. Χρησιμοποιείται έναντι του AttackTarget σε ενέργειες που δεν είναι άμεσα επιθετικές προς έναν Actor ή που το reference ενός Actor δεν είναι διαθέσιμο. Ενδεικτικά, κατά την εκτέλεση ενός EQS Query για την αναζήτηση μιας τοποθεσίας, η αποθήκευση του αποτελέσματος πραγματοποιείται στη μεταβλητή. Ακόμη, όταν το Hearing Sense του Agent προκαλέσει την εκτέλεση του state Investigate, το key PointOfInterest ορίζεται με το περιεχόμενο της τοποθεσίας από την οποία προκλήθηκε η αλλαγή, ώστε να χρησιμοποιηθεί εντός του Behavior Tree.
- *AttackRadius και DefendRadius (Float)*: σε αυτές τις μεταβλητές ορίζονται οι αποστάσεις από τις οποίες ένας Actor μπορεί να επιτεθεί, καθώς και να διατηρήσει απόσταση ασφαλείας από έναν άλλον Actor αντίστοιχα. Χρησιμοποιείται ως είσοδος για την εκτέλεση του Behavior Tree Task BTT_MoveToIdealRange στα sequences FindCover και Strafe εντός του Behavior Tree.
- *AttackTeleportLocation (Vector)*: σε αυτή τη μεταβλητή αποθηκεύεται η τοποθεσία στην οποία ένας Agent τύπου Android θα πραγματοποιήσει teleport και χρησιμοποιείται στο Behavior Tree Task BTT_GetAttackTeleportLocation, κατά την εκτέλεση της επίθεσης Aerial Attack.
- *DistanceToPlayer (Float)*: σε αυτή τη μεταβλητή αποθηκεύεται η απόσταση του Agent από τον παίκτη. Χρησιμοποιείται αποκλειστικά από Agents τύπου Android ως BlackboardBasedCondition για την επιλογή της κατάλληλης επίθεσης του Agent. Ενημερώνεται

σε πραγματικό χρόνο μέσω του service `BTS_UpdateDistanceToTarget` που πυροδοτείται στην έναρξη του `AttackState`.

- *FrozenWhileAttacking (Boolean)*: σε αυτή τη μεταβλητή ορίζεται εάν ένα Android θα εκτελέσει το υποδέντρο `FrozenWhileAttacking` ή `FrozenPassive` όταν βρίσκεται στο State `Frozen`, λόγω επίθεσης από `DataWorm` και χρησιμοποιείται ως `BlackboardBasedCondition`.
- *LocationPriorToInvestigation (Vector)*: σε αυτή τη μεταβλητή αποθηκεύεται η τοποθεσία στην οποία βρίσκεται ένας Agent προτού μεταβεί στα State `Investigate` ή `Seek`, κατά τα οποία μεταβάλλει την αρχική του θέση. Χρησιμοποιείται ως η τοποθεσία που θα επιστρέψει ο Agent, όταν ολοκληρωθούν οι παραπάνω ενέργειες, κατά την επιστροφή τους στο State `Passive`.

7.3.2. AI Controller

Για ευελιξία στην ανάπτυξη λειτουργιών της συμπεριφοράς των Agents δημιουργήθηκε ένα custom AI Controller με την ονομασία `AIC_Enemy_Base`. Η συγκεκριμένη κλάση αποτελεί τον κεντρικό κόμβο διαχείρισης της τεχνητής νοημοσύνης των Agents, ενσωματώνοντας τη βασική λογική των λειτουργιών που μπορούν να εκτελέσουν κατά τη διάρκεια της αλληλεπίδρασής τους με το περιβάλλον και τον παίκτη.

Η `AIC_Enemy_Base` λειτουργεί ως ενιαίο Blueprint, στο οποίο συγκεντρώνονται σημαντικές λειτουργίες, όπως η πυροδότηση Custom Events που βρίσκονται στο Blueprint κάθε Agent, η δυναμική μεταβολή τιμών μεταβλητών στο Blackboard του Behavior Tree, καθώς και η γενικότερη διαχείριση της επικοινωνίας ανάμεσα στο Behavior Tree και το Blueprint των Agents. Η ανάθεση του έγινε στην Parent Class `BP_Enemy_Base`, ώστε να κληρονομείται από κάθε αντίστοιχο Child Actor.

Παρόλο που στο πλαίσιο μεγαλύτερων έργων είναι συνήθης η δημιουργία πολλαπλών AI Controllers μέσω μιας δομής τύπου Parent-Child για την εξειδίκευση της λογικής ανά τύπο εχθρού ή λειτουργία, στην παρούσα υλοποίηση επιλέχθηκε η ενσωμάτωση όλων των διαφοροποιήσεων (όπως ο τύπος εχθρού ή το φύλο του χαρακτήρα) εντός του ίδιου AI Controller. Η επιλογή αυτή ενισχύει τη συνοχή του συστήματος και συγκεντρώνει τη λογική σε ένα ενιαίο σημείο, γεγονός που διευκολύνει τη συντήρηση και την αποσφαλμάτωση. Ωστόσο, συνοδεύεται από αυξημένη πολυπλοκότητα στον σχεδιασμό, καθώς απαιτεί την υλοποίηση παραμετρικών ελέγχων και διακλαδώσεων σε πολλαπλά σημεία του κώδικα, ώστε να καλυφθούν όλες οι πιθανές περιπτώσεις.

Τέλος, ιδιαίτερη προσοχή απαιτείται ως προς την τοποθέτηση συγκεκριμένων λειτουργιών των Agents, καθώς μία ενέργεια μπορεί να υλοποιηθεί είτε στον AI Controller, είτε στο Behavior Tree, είτε απευθείας στον ίδιο τον Actor. Η επιλογή του κατάλληλου σημείου υλοποίησης εξαρτάται από τον σκοπό που εξυπηρετεί η ενέργεια, τη φύση της, καθώς και από το ποιοι Agents την αξιοποιούν και σε ποιο πλαίσιο. Η ισορροπημένη κατανομή της λογικής σε αυτά τα επίπεδα αποτελεί κρίσιμο στοιχείο για την αποδοτικότητα και την επεκτασιμότητα του συνολικού συστήματος.

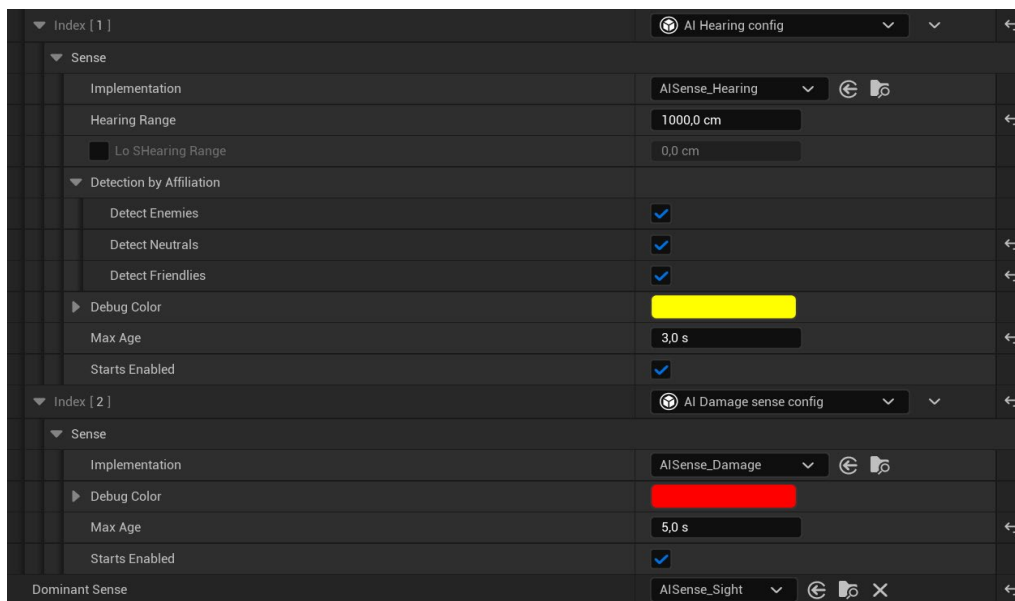
7.3.3. Perception System

Για την υλοποίηση λειτουργιών που βασίζονται στην απόκριση των Agents σε εξωτερικά ερεθίσματα, αξιοποιεί τις παραμέτρους και τα events που παρέχονται από το AI Perception Component. Συγκεκριμένα, καθορίστηκαν οι τιμές των παραμέτρων για κάθε Sense σύμφωνα με τις σχεδιαστικές ανάγκες του παιχνιδιού, και αναπτύχθηκαν 3 Custom Events τα οποία διαχειρίζονται την εκτέλεση των ενεργειών που αφορούν στην πυροδότηση κάθε κατηγορίας ερεθίσματος. Οι τιμές που ανατέθηκαν διασφαλίζουν πως η λειτουργία των αισθήσεων λειτουργεί με ρεαλιστικό τρόπο στο πλαίσιο του παιχνιδιού και παρουσιάζονται στις εικόνες 56 και 57.

The screenshot displays the 'AI Perception' configuration window, specifically the 'Senses Config' section. It shows a list of 3 array elements, with the first element selected and expanded. The 'Sense' dropdown is set to 'AI Sense_Sight'. The configuration parameters for this sense are as follows:

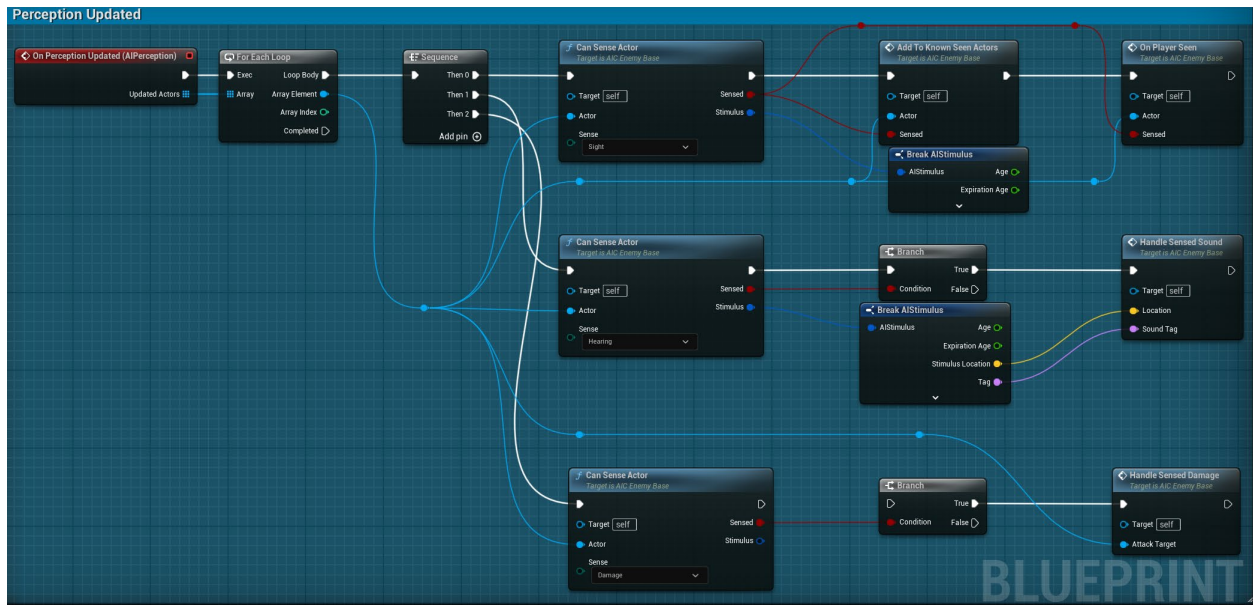
Parameter	Value
Implementation	AI Sense_Sight
Sight Radius	1500,0 cm
Lose Sight Radius	2000,0 cm
Peripheral Vision Half Angle	65,0 °
Detection by Affiliation	
Detect Enemies	☒
Detect Neutrals	☒
Detect Friendlies	☒
Auto Success Range from Last Seen Location	-1,0 cm
Point Of View Backward Offset	0,0 cm
Near Clipping Radius	0,0 cm
Debug Color	Red
Max Age	20,0 s
Starts Enabled	☒

Εικόνα 56. Sight Sense Configuration



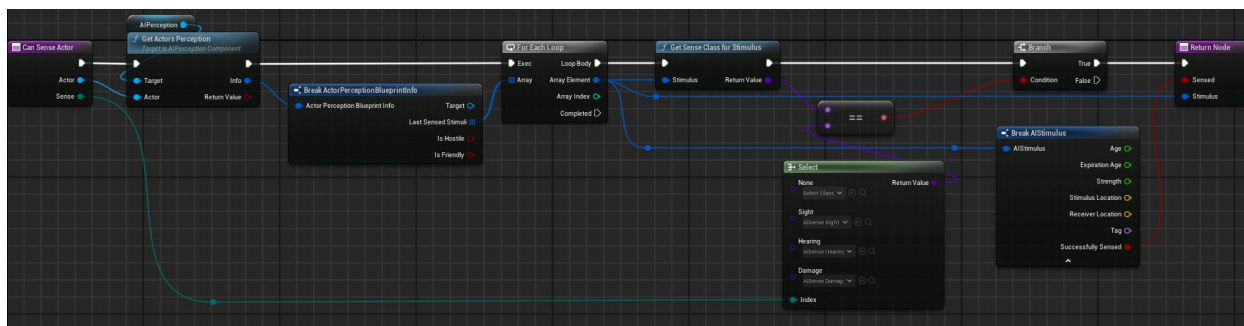
Εικόνα 57. Hearing & Damage Sense Configuration

OnPerceptionUpdated (Custom Event): Η διαχείριση των ενεργειών για τα ερεθίσματα που πυροδοτούνται από τα SightSense, HearingSense και DamageSense επιτυγχάνεται μέσα από τα Custom Events OnPlayerSeen, Handle Sensed Sound και Handle Sensed Damage αντίστοιχα. Συγκεκριμένα, τα Events καλούνται κάθε φορά που ένα ερέθισμα πυροδοτεί το Custom Event OnPerceptionUpdated. Το event παρέχει τη μεταβλητή UpdatedActors (Array) η οποία περιέχει όλους τους Actors που προκάλεσαν ένα ερέθισμα. Για κάθε τέτοιο actor, το function CanSenseActor δέχεται ως είσοδο ένα από τα Enums Sight, Hearing ή Damage και πυροδοτεί το κατάλληλο Custom Event αντίστοιχα, παρέχοντας τις απαραίτητες μεταβλητές για τη διαχείριση των ενεργειών.



Εικόνα 58. OnPerceptionUpdated

CanSenseActor (Function): Το CanSenseActor ανακτά όλα τα τελευταία ερεθίσματα που καταγράφηκαν στο Perception του Agent, μέσω των nodes GetActorsPerception και BreakActorPerceptionBlueprintInfo. Έπειτα, σαρώνει τα αποτελέσματα μέσω του node ForEachLoop και επιλέγει αυτό που σχετίζεται με το δεδομένο Sense σύμφωνα με την τιμή του Enum που παρέχεται κατά την είσοδο. Τέλος, το function επιστρέφει το reference του ερεθίσματος, το οποίο παρέχει όλες τις πληροφορίες που σχετίζονται με αυτό όπως την τοποθεσία του στο χώρο, πιθανά Tag που το διακατέχουν κ.α. Ακόμη, το function επιστρέφει ως ξεχωριστή μεταβλητή την SuccessfullySensed (Boolean), η οποία είναι μέρος των πληροφοριών του ερεθίσματος και αντιπροσωπεύει το γεγονός αν το ερέθισμα αποτελεί εγγραφή ή διαγραφή από το Perception του Actor. Για παράδειγμα, όταν ο παίκτης βρεθεί στο οπτικό πεδίο του Actor η μεταβλητή θα έχει την τιμή true, ενώ όταν ο Actor εξέλθει από το οπτικό πεδίο του Agent θα έχει την τιμή false. Ως αποτέλεσμα, η μεταβλητή SuccessfullySensed επιτρέπει την ανάπτυξη ενεργειών για κάθε περίπτωση, αποτελώντας βασική παράμετρο της υλοποίησης.



Εικόνα 59. CanSenseActor

Handle Senses: Σύμφωνα με την κατηγορία του Sense που ανήκει ένα ερέθισμα, πυροδοτούνται ένα από τα functions OnPlayerSeen, HandleSensedSound και HandleSensedDamage. Το πρώτο εκτελείται όταν ένας Actor εισέρχεται στο οπτικό πεδίο του Agent, και δέχεται ως είσοδο το reference του Actor και τη μεταβλητή SuccessfullySensed. Το function περιλαμβάνει τις απαραίτητες ενέργειες βάση διαφόρων συνθηκών, όπως αυτές περιγράφονται στο υποκεφάλαιο 8.6. Το δεύτερο function αφορά στην πυροδότηση ενός ηχητικού ερεθίσματος και περιλαμβάνει σχετικές ενέργειες, όπως αυτές περιγράφονται στο υποκεφάλαιο 8.3. Λόγω των ενεργειών που υλοποιούνται σε αυτή την περίπτωση, το function απαιτεί ως είσοδο μόνο τις μεταβλητές StimulusLocation και Tag, οι οποίες παρέχονται από το reference του ερεθίσματος. Τέλος, το function HandleSensedDamage περιλαμβάνει τις ενέργειες που εκτελούνται όταν ο Actor δέχεται Damage. Συγκεκριμένα, το function μπορεί να μεταβάλει τον Agent στο Attacking State αν αυτός βρίσκεται σε ένα από τα States Passive, Investigating, Seeking, ή Detecting και αν ο Actor που προκάλεσε το Damage δεν είναι σύμμαχος του Agent. Ωστόσο, το συγκεκριμένο Sense μπορεί να πυροδοτηθεί μόνο μέσω του node ReportDamageEvent που παρέχει το UnrealEngine. Καθώς σε αυτό το project οι παράμετροι και οι ενέργειες που σχετίζονται με την απόδοση και αποδοχή Damage ενός Actor ενσωματώνονται στο DamageSystem, το function υλοποιήθηκε για λόγους καλής πρακτικής, αλλά δεν αξιοποιήθηκε. Αντ' αυτού, οι ενέργειες που περιλαμβάνει το HandleSensedDamage ενσωματώνονται στο function DamageResponse του DamageSystem.

7.3.4. BPI_Enemy_AI

Στο πλαίσιο της υλοποίησης, για τη διευκόλυνση της επικοινωνίας μεταξύ των Behavior Trees και των Blueprints του AI Controller και των Agents, αναπτύχθηκε ένα Blueprint Interface με την ονομασία BPI_Enemy_AI. Η χρήση του επιτρέπει τον ορισμό και την οργάνωση ενός συνόλου συναρτήσεων (functions) που αξιοποιούνται σε διάφορα σημεία της αρχιτεκτονικής της τεχνητής νοημοσύνης, εξασφαλίζοντας επεκτασιμότητα, σαφήνεια και επαναχρησιμοποίηση του κώδικα χωρίς την ανάγκη ρητών μετατροπών τύπων μέσω casting.

Το συγκεκριμένο Interface εκχωρήθηκε τόσο στο Blueprint του AI Controller όσο και στο Parent Blueprint των Agents, καθιστώντας δυνατή την υλοποίηση των απαιτούμενων συναρτήσεων στο κατάλληλο περιβάλλον, ανάλογα με τη φύση της κάθε λειτουργίας. Η υιοθέτηση αυτής της προσέγγισης

προσφέρει σημαντική ευελιξία: μία συνάρτηση που καλείται εντός ενός Custom Task του Behavior Tree, μπορεί να εκτελεστεί είτε στο Blueprint του Agent (μέσω του reference Controlled Pawn), είτε στον AI Controller (μέσω του Owner Controller), χωρίς να απαιτείται η χρήση κόμβων Cast To.

Το Interface χρησιμοποιείται τόσο για την πυροδότηση συναρτήσεων (π.χ. ενέργειες, μεταβάσεις καταστάσεων, εκκίνηση animation sequences), όσο και για την ανάκτηση και μετάδοση μεταβλητών μεταξύ των δύο βασικών αρχιτεκτονικών επιπέδων της τεχνητής νοημοσύνης. Με αυτόν τον τρόπο, επιτυγχάνεται μια δομημένη και αποδοτική ροή δεδομένων, η οποία υποστηρίζει τη δυναμική συμπεριφορά των Agents εντός του παιχνιδιού.

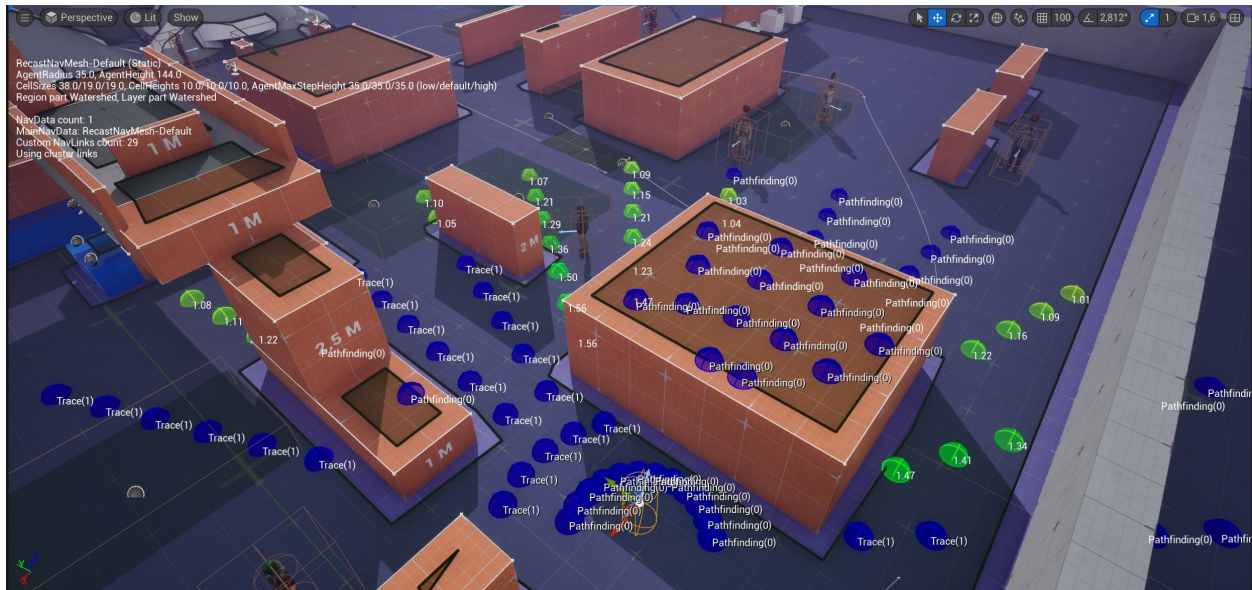
7.3.5. EQS TestingPawn

Στο πλαίσιο της παρούσας υλοποίησης, αξιοποιήθηκε εκτενώς το σύστημα Environmental Query System (EQS) της Unreal Engine για την αναζήτηση κατάλληλων τοποθεσιών βάσει παραμετρικών κριτηρίων. Τα υποψήφια σημεία αναπαράγονται αρχικά μέσω προκαθορισμένων pattern assets, όπως τα SimpleGrid και ConeGrid, τα οποία καθορίζουν τη γεωμετρία και την κατανομή των παραγόμενων σημείων εντός του επιπέδου. Οι παράμετροι GridHalfSize και SpaceBetween επηρεάζουν, αντίστοιχα, την ακτίνα διασποράς και την μεταξύ των σημείων απόσταση, καθορίζοντας άμεσα το υπολογιστικό βάρος του εκάστοτε query.

Για τη δοκιμή και αξιολόγηση των EQS Queries κατά την ανάπτυξή τους, δημιουργήθηκε ένα ειδικό EQS_TestingPawn, στο οποίο ανατίθενται δυναμικά τα υπό ανάπτυξη queries. Το Testing Pawn τοποθετείται εντός του επιπέδου (Level), επιτρέποντας την άμεση οπτικοποίηση των παραγόμενων αποτελεσμάτων και τη βελτιστοποίηση των παραμέτρων πριν την ένταξή τους στο Behavior Tree. Κατά την εκτέλεση ενός Query, όλα τα υποψήφια σημεία θεωρούνται αρχικά ισοπίθانا. Η κατάταξη και επιλογή του βέλτιστου σημείου πραγματοποιείται βάσει των Tests που εφαρμόζονται, τα οποία τροποποιούν τη βαρύτητα (score) κάθε σημείου με βάση συγκεκριμένα κριτήρια. Η εικόνα 60 απεικονίζει το Testing Pawn να χρησιμοποιεί ένα EQS Query με ConeGrid καθώς και τα αξιολογημένα σημεία που παράχθηκαν. Σύμφωνα με τις παραμέτρους των Tests, αναζητείται το κοντινότερο σημείο όπου το Testing Pawn δεν έχει οπτική επαφή.

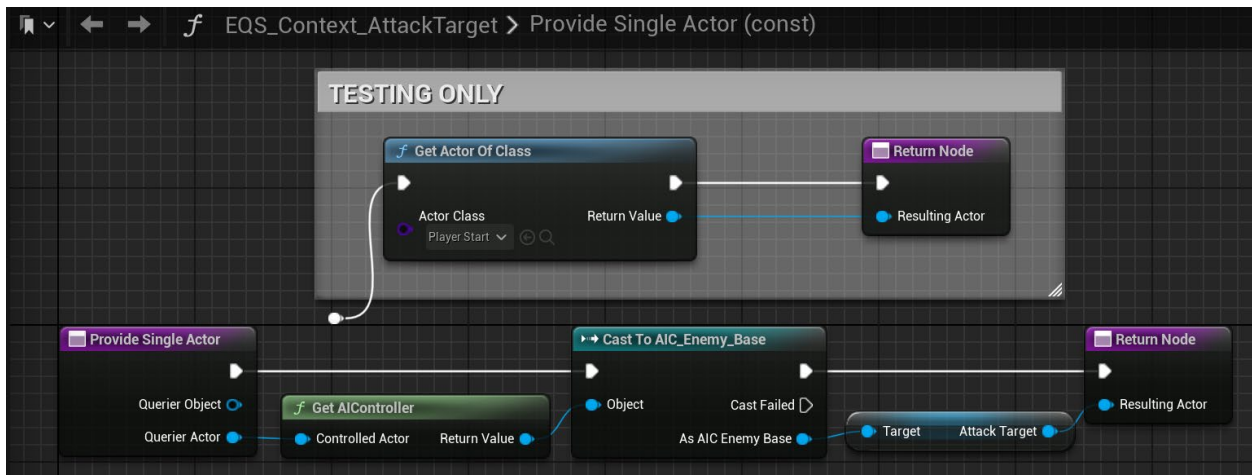
Ενδεικτικά παραδείγματα Tests που χρησιμοποιήθηκαν περιλαμβάνουν:

- PathExist: From Querier: απορρίπτει σημεία που δεν είναι προσβάσιμα από τον Agent.
- Distance: To Querier: τροποποιεί τη βαρύτητα των σημείων βάσει της απόστασής τους από τον Agent (π.χ. ευνοεί σημεία εντός απόστασης < 300 μονάδων).



Εικόνα 60. EQS Query με ConeGrid

Για την υποστήριξη Tests που απαιτούν τη χρήση Context, δηλαδή τον καθορισμό σημείων αναφοράς για την αξιολόγηση, αναπτύχθηκε το asset EQS_Context_AttackTarget. Το asset αυτό υλοποιεί την κλάση EnvQueryContext_BlueprintBase, που παρέχεται εγγενώς από τη μηχανή, και επιτρέπει την ανάκτηση του AttackTarget ενός Agent, δηλαδή του τρέχοντος Actor-στόχου. Η ανάκτηση πραγματοποιείται μέσω του AI Controller του Actor που εκτελεί το query (Querier), επιτρέποντας τη δυναμική εισαγωγή του context στο query.



Εικόνα 61. EQS_Context

Η εκτέλεση ενός EQS Query μπορεί να πραγματοποιηθεί εντός ενός Custom Task, δηλαδή με τη χρήση nodes εντός Blueprint ή με το Task RunEQSQuery που παρέχει το Unreal Engine. Τα Queries αυτού του Project εκτελέστηκαν με τον δεύτερο τρόπο, καθώς τα αποτελέσματα που παρέχουν δεν απαιτούν τροποποίηση για τις ανάγκες που χρησιμοποιούνται. Το RunEQSQuery δέχεται το EQS Query Asset ως είσοδο και παρέχει τις ρυθμίσεις RunMode και Blackboard Key. Η πρώτη αναφέρεται στο αποτέλεσμα που θα επιστραφεί από το Query, όπως το σημείο με τη μεγαλύτερη βαθμολογία (Single Best Item), ένα τυχαίο σημείο από το 25% των συνολικών σημείων με τη μεγαλύτερη βαθμολογία (Single Random Item from Best 25%) κ.α. Η δεύτερη ρύθμιση αναφέρεται στο Blackboard key στο οποίο το αποτέλεσμα θα αποθηκευτεί, ώστε να μπορεί να χρησιμοποιηθεί σε κάποιο Task ως είσοδος.

Η παραπάνω προσέγγιση επιτρέπει την υλοποίηση πολύπλοκων και ευέλικτων EQS Queries, τα οποία μπορούν να καθορίσουν χωρικές επιλογές με βάση τόσο τον Agent όσο και τον στόχο του, ενισχύοντας την πειστικότητα της συμπεριφοράς του Agent.

8. Behavior States

Η παρούσα εργασία επικεντρώνεται στην ανάπτυξη NPC Agents που επιδεικνύουν ρεαλιστική και προσαρμοστική συμπεριφορά, βασισμένη στα γεγονότα του περιβάλλοντος και στις ενέργειες του παίκτη ή των συμμάχων Agents. Πυρήνας αυτής της συμπεριφοράς αποτελεί η υλοποίηση και διαχείριση ενός συστήματος Behavior Trees, το οποίο οργανώνει τις ενέργειες και τις αντιδράσεις των Agents βάσει προκαθορισμένων States. Τα Behavior Trees σε αυτό το έργο υλοποιούν μια δομημένη ιεραρχία καταστάσεων, η οποία επιτρέπει στους Agents να αλλάζουν ρεαλιστικά και άμεσα συμπεριφορά ανάλογα με τα ερεθίσματα του παιχνιδιού.

Συνολικά, σχεδιάστηκαν και υλοποιήθηκαν δύο βασικά Behavior Trees: ένα για τους Ranged Agents (τύπου Pistol και Rifle) και ένα ξεχωριστό για τους Android Agents. Οι τύποι αυτοί διαφέρουν τόσο ως προς το σύνολο ενεργειών όσο και στη στρατηγική αντιμετώπισης του παίκτη, γεγονός που καθιστά αναγκαία τη διάκριση σε διαφορετικές BT δομές. Παράλληλα, αναπτύχθηκαν πολλαπλά Subtrees, τα οποία συγκεντρώνουν επαναλαμβανόμενες ενέργειες ή κοινά συμπεριφορικά μοτίβα, επιτρέποντας την επαναχρησιμοποίηση και τη modular προσέγγιση στον σχεδιασμό.

Το σύστημα συμπεριφοράς στηρίζεται σε έξι βασικά States: Passive, Investigating, Detecting, Seeking, Attacking και Dead. Κάθε κατάσταση αναπαριστά ένα σενάριο δράσης, το οποίο περιλαμβάνει συγκεκριμένη ακολουθία ενεργειών και επίπεδο επιθετικότητας. Ο Agent μπορεί να μεταβαίνει από το ένα State στο άλλο, με κριτήριο εξωτερικά ερεθίσματα ή την έκβαση εσωτερικών ελέγχων και να εκτελεί αντίστοιχες συμπεριφορές μέσω των κόμβων του Behavior Tree.

Για την ομαλή μετάβαση μεταξύ των καταστάσεων και την εκτέλεση της κατάλληλης συμπεριφοράς, αξιοποιούνται Decorators υψηλής προτεραιότητας. Οι Decorators αυτοί εξασφαλίζουν πως κάθε State μπορεί να διακόψει άμεσα οποιαδήποτε συμπεριφορά χαμηλότερης προτεραιότητας, εξασφαλίζοντας ρεαλιστική απόκριση και συνέπεια στη λήψη αποφάσεων.

Η συμπεριφορά των Agents συμπληρώνεται από την υλοποίηση Custom Tasks και EQS Queries, οι οποίες επιτρέπουν εξειδικευμένες αναζητήσεις στο περιβάλλον και ενέργειες προσαρμοσμένες στις απαιτήσεις του gameplay. Οι λειτουργίες αυτές ενσωματώνονται οργανικά τόσο στα Subtrees όσο και στο κύριο Behavior Tree, επιτρέποντας την έξυπνη αντίληψη, στόχευση, μετακίνηση και επιθετικότητα των Agents.

Επιπλέον, καθοριστικό ρόλο διαδραματίζει η υποδομή εναλλαγής States, η οποία υλοποιείται μέσω συναρτήσεων τόσο στο AI Controller όσο και στα Blueprints των Agents. Οι συναρτήσεις αυτές είναι υπεύθυνες για την ενεργοποίηση και την απενεργοποίηση συμπεριφορών, την ομαλή διαχείριση των μεταβλητών και τη διατήρηση της συνοχής κατά τη διάρκεια της συμπεριφορικής μετάβασης.

Τέλος, σημαντική έμφαση δόθηκε στη διασύνδεση του συστήματος συμπεριφοράς με τα οπτικοακουστικά στοιχεία του παιχνιδιού, προσδίδοντας βάθος και ενισχύοντας την αίσθηση εμπύθισης. Ενέργειες όπως η χρήση αντικειμένων (π.χ. Bottle, DataWorm), η κίνηση Takedown, τα μοναδικά voice lines και τα εικονίδια κατάστασης υποστηρίζονται από το σύστημα BT, ενισχύοντας την αντίληψη του παίκτη για τη νοημοσύνη των Agents και προσφέροντας πλούσια, ρεαλιστική και δυναμική εμπειρία gameplay.

Στο παρόν κεφάλαιο αναλύεται σε βάθος η λειτουργία κάθε επιμέρους State που συγκροτεί το σύστημα συμπεριφοράς των Agents. Εξετάζονται αναλυτικά όλες οι ενέργειες που υλοποιούνται εντός κάθε State, οι δομές των αντίστοιχων Subtrees καθώς και οι Custom Tasks και οι Functions που απαιτήθηκαν για την υποστήριξή τους. Παρουσιάζεται η λογική πίσω από την επιλογή συγκεκριμένων BT κόμβων, οι τρόποι ενσωμάτωσης EQS Queries, οι διαχειριστικές λειτουργίες του AI Controller και των Blueprints, καθώς και κάθε σχετικό asset που συνεισφέρει στη λειτουργικότητα ή την αισθητική του εκάστοτε State. Η ανάλυση εστιάζει τόσο στην τεχνική υλοποίηση όσο και στη σχεδιαστική λογική που διέπει το σύστημα, προσφέροντας μια πλήρη εικόνα της αρχιτεκτονικής, της συμπεριφορικής συνέπειας και της διαδραστικότητας που επιτυγχάνεται μέσω των Behavior Trees.

8.1. Behavior Trees Overview

Η τεχνητή νοημοσύνη των Agents στο παρόν project βασίζεται στην υλοποίηση Behavior Trees, τα οποία διαχειρίζονται τη λογική λήψης αποφάσεων και την εκτέλεση ενεργειών ανάλογα με την κατάσταση κάθε Agent. Μέσω αυτών, επιτυγχάνεται μια ευέλικτη, αντιδραστική και πολυκαταστασιακή συμπεριφορά, η οποία οργανώνεται σε States διαφορετικού επιπέδου συνειδητότητας και επιθετικότητας.

Για την υλοποίηση χρησιμοποιήθηκαν δύο βασικά Behavior Trees: ένα για τους Agents τύπου Ranged (Pistol και Rifle) και ένα ξεχωριστό για τους Androids, ώστε να υποστηριχθούν οι διαφορετικές επιθετικές τακτικές, δυνατότητες κίνησης και τρόποι αντιμετώπισης. Πέρα από τα κύρια δέντρα, υλοποιήθηκαν και Subtrees για κοινές ή επαναλαμβανόμενες ενέργειες (π.χ. Patrol, Move to Location), καθώς και Custom Tasks και EQS Queries για πιο σύνθετες αποφάσεις.

Τα βασικά States που διαχειρίζονται τα Behavior Trees είναι τα εξής:

- **Passive:** Πρόκειται για την προεπιλεγμένη κατάσταση του Agent κατά την οποία είτε περιπολεί σε προκαθορισμένη διαδρομή είτε φυλάσσει σταθερό σημείο. Δεν φέρει τον οπλισμό του στα χέρια και παρουσιάζει μηδενική επιθετική διάθεση. Οι ενέργειες ενεργοποιούνται μέσω Selector node με εναλλαγή ανάμεσα σε Patrol και Idle επιτήρηση.
- **Investigating:** Ενεργοποιείται όταν ο Agent αντιληφθεί ένα ηχητικό ερέθισμα. Πραγματοποιεί έλεγχο του σημείου χωρίς οπλισμό, σαρώνει το περιβάλλον και εάν δεν εντοπίσει απειλή, επιστρέφει στο Passive State. Το Behavior Tree μεταβαίνει προσωρινά σε αυτό το State μέσω Decorator.
- **Detecting:** Ο Agent έχει οπτική επαφή με πιθανό εχθρό, αλλά δεν έχει ολοκληρώσει την αναγνώρισή του πλήρως. Το State υλοποιείται με έναν μηχανισμό χρονικής προόδου που αναπαριστά το επίπεδο αναγνώρισης για τον καθορισμό της επόμενης ενέργειας. Αν ο παίκτης γίνει πλήρως αντιληπτός, ο Agent μεταβαίνει σε Attacking. Αν χαθεί η οπτική επαφή αλλά η αναγνώριση είναι προχωρημένη, τότε εισέρχεται στο Seeking State. Διαφορετικά επιστρέφει στο προηγούμενο State που βρισκόταν.
- **Seeking:** Το State αυτό αφορά στην ενεργή αναζήτηση απειλής με τον Agent οπλισμένο. Χρησιμοποιεί EQS Queries για να εντοπίσει πιθανές τοποθεσίες του παίκτη, σε συνδυασμό με Task Loops που ελέγχουν τις περιοχές. Το Seeking προκύπτει από προχωρημένο Detecting ή αν ο παίκτης βρεθεί εκτός του οπτικού πεδίου του Agent για συγκεκριμένο χρονικό διάστημα. Ακόμη, μπορεί να ενεργοποιηθεί όταν ο Agent αντιλαμβάνεται έναν εξουδετερωμένο σύμμαχο Agent. Αν η αναζήτηση του παίκτη αποτύχει, ο Agent επιστρέφει στο Passive State.
- **Attacking:** Ενεργοποιείται όταν η παρουσία του παίκτη έχει επιβεβαιωθεί. Οι Ranged Agents ακολουθούν σύνθετη λογική που συνδυάζει Task nodes για ελιγμούς και πυροβολισμούς, καθώς και BlackBoard conditions για διατήρηση ασφαλούς απόστασης από τον παίκτη. Οι Agents τύπου Android εκτελούν άμεσες επιθέσεις, και αν χάσουν την οπτική επαφή τηλεμεταφέρονται κοντά στον παίκτη, παρακάμπτοντας το Seeking State. Η προτεραιότητα του State είναι η υψηλότερη, και διακόπτει κάθε προηγούμενη συμπεριφορά.
- **Dead:** Το τελικό State όπου ο Agent θεωρείται εξουδετερωμένος. Απενεργοποιείται το Behavior Tree, οι AI λειτουργίες διακόπτονται και ενεργοποιούνται ειδικά animations ή εφέ. Το State διασφαλίζει ότι οι διαδικασίες αποδέσμευσης του Agent από το AI Controller και το Behavior Tree, καθώς και οι ενέργειες εξουδετέρωσης όπως τροποποίηση μεταβλητών και ειδικά εφέ εκτελούνται κατάλληλα.

Διαχείριση Ροής Συμπεριφορών

Η λογική ροή μεταξύ των διαφορετικών καταστάσεων (States) του Agent καθορίζεται μέσω των Decorators και των Blackboard Keys, επιτρέποντας τη δυναμική μετάβαση μεταξύ συμπεριφορών ανάλογα με τη μεταβολή των συνθηκών. Η αρχιτεκτονική βασίζεται στη χρήση Selector και Sequence κόμβων σε συνδυασμό με Conditional Decorators, οι οποίοι διασφαλίζουν την εύκαμπτη προσαρμογή της συμπεριφοράς του Agent σύμφωνα με τις παραμέτρους του περιβάλλοντος ή τις ενέργειες του παίκτη.

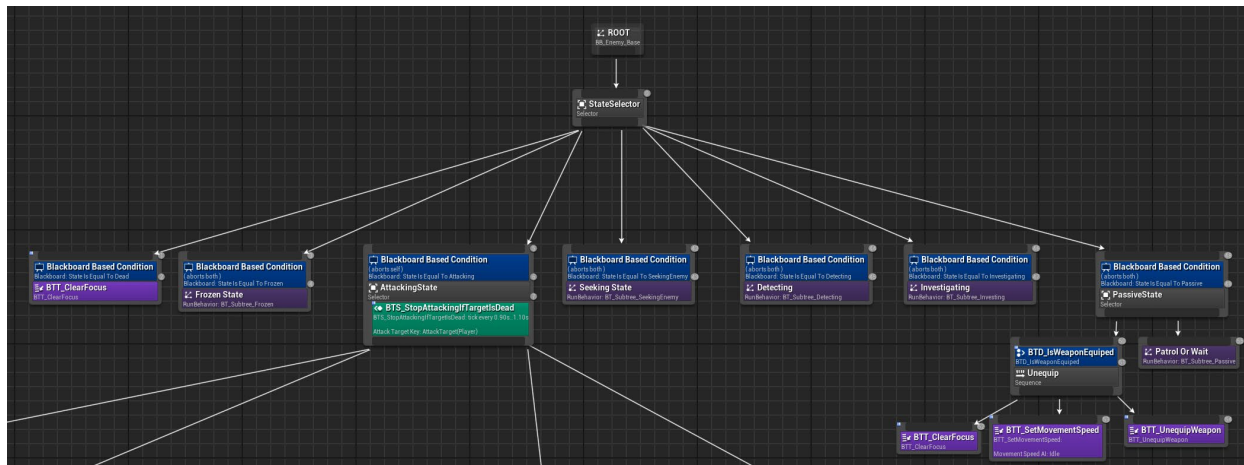
Καταστάσεις που περιλαμβάνουν εκτενές σύνολο ενεργειών υλοποιούνται ως ξεχωριστά Subtrees, τα οποία ενσωματώνονται στο κύριο Behavior Tree μέσω Run Behavior κόμβων. Η προσέγγιση αυτή προσφέρει σημαντικά πλεονεκτήματα σε επίπεδο οργάνωσης και συντηρησιμότητας, παρέχοντας επαναχρησιμοποίηση υπομονάδων σε διαφορετικά δέντρα χωρίς την ανάγκη διπλότυπου κώδικα. Επιπλέον, η χρήση υποδέντρων μειώνει το υπολογιστικό φορτίο μέσω περιορισμένων ελέγχων για την επιλογή του κατάλληλου State, καθώς η αξιολόγηση γίνεται τοπικά εντός των υποδέντρων.

Η συνολική αρχιτεκτονική υποστηρίζει ρεαλιστική, δομοστοιχειωτή και επεκτάσιμη συμπεριφορά, η οποία ανταποκρίνεται σε σύνθετα σενάρια gameplay. Η σαφής διάκριση μεταξύ των States, σε συνδυασμό με την προτεραιοποίηση και την υλοποίηση ανεξάρτητων υποδέντρων, καθιστά το σύστημα ευέλικτο, επεκτάσιμο και ανθεκτικό σε συνθήκες που μεταβάλλονται δυναμικά κατά τη διάρκεια του παιχνιδιού.

Επισκόπηση Ροής Εκτέλεσης

Το Behavior Tree εκκινείται με έναν βασικό *State Selector* ως root node, ο οποίος περιλαμβάνει ως children τους κόμβους κάθε επιμέρους State. Κάθε child node αποτελείται από *Run Behavior* κόμβους και εσωτερικούς *Selectors*, οργανωμένους ιεραρχικά κατά φθίνουσα σειρά προτεραιότητας. Η σειρά αξιολόγησης των States είναι η εξής:

Dead → Frozen → Attacking → Seeking → Detecting → Investigating → Passive



Εικόνα 62. Behavior Tree Overview

Η ενεργοποίηση κάθε State καθορίζεται από την τιμή του Blackboard Key με όνομα State (Enum), μέσω των αντίστοιχων Blackboard-Based Condition Decorators. Η μεταβολή της τιμής της μεταβλητής αυτής

προκαλεί αυτόματη επαναξιολόγηση του State Selector, επιτρέποντας τη μετάβαση σε νέο State και διακοπή του προηγούμενου.

Κατά την αρχικοποίηση του Agent, το AI Controller ορίζει το State ως Passive, οδηγώντας τον Agent στην εκτέλεση ενεργειών περιπολίας ή επιφυλακής, οι οποίες ενσωματώνονται στο υποδέντρο Patrol or Wait. Το εν λόγω υποδέντρο χρησιμοποιείται από όλους τους Agents. Ωστόσο, για τους Ranged Agents, υπάρχει ενδιάμεσο φίλτρο που διασφαλίζει ότι η μετάβαση σε Passive State συνοδεύεται από διαδικασία αφοπλισμού, εφόσον προηγουμένως ο Agent βρισκόταν σε επιθετική κατάσταση. Ως αποτέλεσμα, το Passive State του είναι ένας Selector που επιλέγει την κατάλληλη ενέργεια.

Σε περίπτωση ηχητικής διατάραξης (π.χ. μέσω χρήσης του αντικειμένου Bottle), το State μεταβάλλεται σε Investigating, ώστε ο Agent να εξετάσει το σημείο προέλευσης του θορύβου. Οι ενέργειες του Investigating State είναι κοινές για όλους τους Agents και υλοποιούνται στο σχετικό υποδέντρο.

Η οπτική επαφή με τον παίκτη προκαλεί μετάβαση στο Detecting State, ενεργοποιώντας τις κατάλληλες αντιδράσεις. Στους Ranged Agents, οι αντιδράσεις περιλαμβάνουν animations και ηχητικούς διαλόγους, ενώ στους Androids οπτικοακουστικά εφέ. Και σε αυτή την περίπτωση, οι ενέργειες υλοποιούνται σε ξεχωριστό υποδέντρο.

Η επιτυχής αναγνώριση μέσω Detecting οδηγεί στη μετάβαση στο Attacking State, το οποίο περιλαμβάνει όλες τις επιθετικές ενέργειες ενός Agent, όπως χρήση όπλων, ελιγμούς κ.ά. Οι ενέργειες διαφέρουν ουσιαστικά μεταξύ διαφορετικών τύπων Agents, καθιστώντας αναγκαία την ύπαρξη διαφορετικών Behavior Trees. Η τοποθέτηση του Attacking State σε υψηλή προτεραιότητα διασφαλίζει την άμεση διακοπή οποιασδήποτε άλλης δραστηριότητας για την εκτέλεση επίθεσης. Η αποχώρηση από το Attacking State μπορεί να προκύψει είτε από την εξουδετέρωση του παίκτη (παρακολουθείται μέσω σχετικού Service), είτε αν ο παίκτης διαφύγει από το οπτικό πεδίο του Agent για προκαθορισμένο διάστημα, προκαλώντας μετάβαση στο Seeking State, όπου ο Agent αναζητά τον στόχο μέσω του αντίστοιχου υποδέντρου.

Το Frozen State αφορά κυρίως τους Android Agents, οι οποίοι είναι ευάλωτοι στο αντικείμενο DataWorm που προκαλεί αδρανοποίηση. Επιπλέον, το State αυτό μπορεί να αξιοποιηθεί και από τους Ranged Enemies για την απεικόνιση animations αντίδρασης κατά την αποδοχή damage. Η θέση του στο δέντρο εξασφαλίζει την προτεραιότητά του έναντι του Attacking State.

Τέλος, το Dead State τοποθετείται στην κορυφή του δέντρου ώστε να διακόπτει άμεσα κάθε ενέργεια του Agent κατά την εξουδετέρωσή του, επιτυγχάνοντας την άμεση μετάβαση σε κατάσταση παύσης δραστηριοτήτων.

Η δομή του δέντρου απεικονίζεται στο διάγραμμα της εικόνας 62.

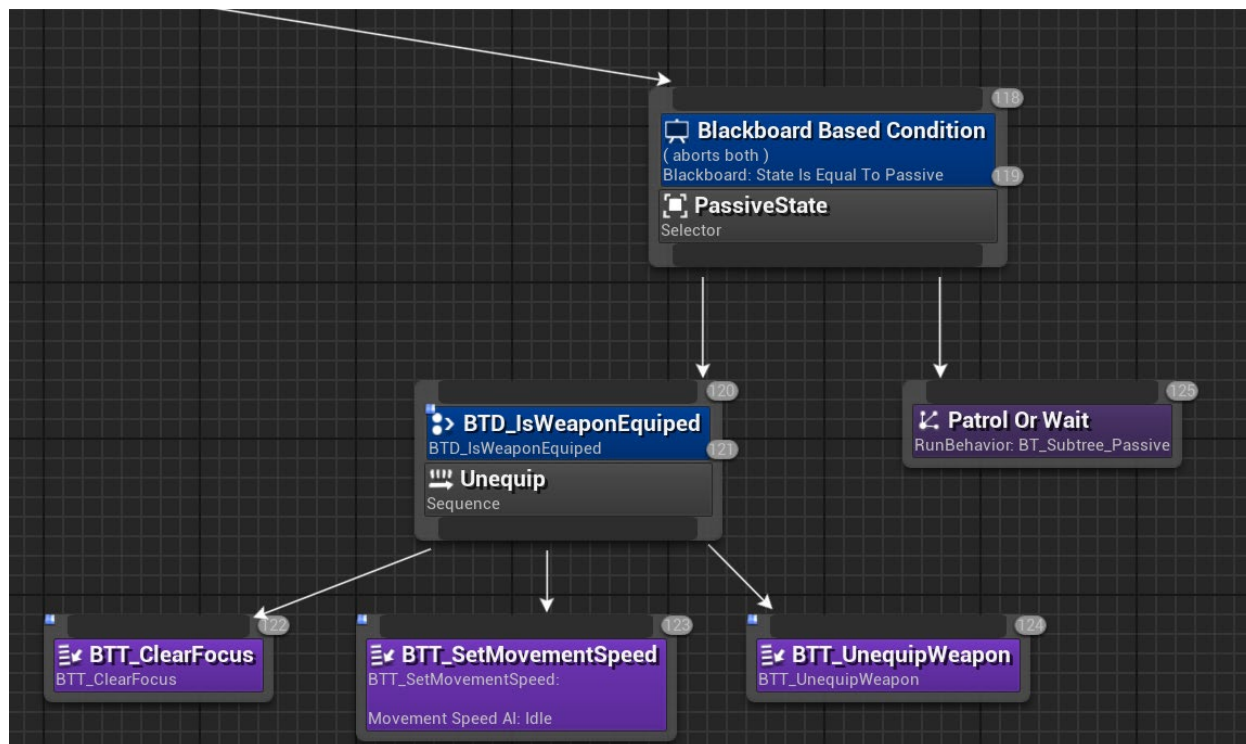
8.2. Passive State

Σε αυτό το State ο Agent βρίσκεται σε παθητική κατάσταση και δεν είναι ενήμερος για την παρουσία κάποιου εχθρού ή κάποιας εχθρικής διατάραξης. Μπορεί να βρίσκεται ακίνητος σε κάποιο σημείο (IdleState) ή να περιπολεί μια συγκεκριμένη διαδρομή αν του έχει ανατεθεί (PatrolState). Μια διαδρομή περιπολίας μπορεί να ανατεθεί κατά την αρχικοποίηση του Agent ή δυναμικά κατά τη διάρκεια του παιχνιδιού, βάσει διάφορων ενεργειών του παίκτη ή του περιβάλλοντος, καθώς και να αποδεσμευθεί από αυτή και να εκτελέσει την κατάσταση Wait. Ένας Agent μπορεί να αλλάξει state πηγαίνοντας στο state Investigate αν ακούσει μια ηχητική διατάραξη (π.χ. αν ο παίκτης πετάξει ένα μπουκάλι για να τον προσελκύσει σε ένα σημείο), στο state Detecting αν ένας εχθρικός στόχος εισέλθει στο οπτικό του πεδίο και στο state Attacking εάν δεχθεί Damage στους πόντους ζωής του. Επιπλέον, στην περίπτωση των Agents τύπου Android, μπορεί να γίνει μετάβαση στο state Frozen, εάν ο παίκτης χρησιμοποιήσει το αντικείμενο DataWorm, απενεργοποιώντας στιγμιαία τις αισθήσεις του NPC. Ακόμη, ο Agent βρίσκεται σε κατάσταση Unaware που σημαίνει πως μπορεί να δεχθεί επίθεση Takedown από τον παίκτη, πεθαίνοντας ακαριαία και ως εκ τούτου να μεταβεί στο Dead State. Από πλευράς προτεραιότητας σε σχέση με τα υπόλοιπα States, το State Passive έχει τη χαμηλότερη θέση και βρίσκεται στο δεξιότερο σημείο στο σύνολο των δέντρων του Behavior Tree. Καθώς αποτελεί τη συμπεριφορά χαμηλότερης προτεραιότητας, μπορεί να διακοπεί από κάθε άλλο State (π.χ. Investigating ή Attacking). Στην περίπτωση μιας τέτοιας αλλαγής, ο Agent θα γυρίσει στο State Passive μόνο σε σενάρια που χάσει από το οπτικό του πεδίο τον στόχο (παίκτη) ή ολοκληρώσει την έρευνα μιας ηχητικής ή οπτικής διατάραξης χωρίς να αναγνωρίσει κάποιον εχθρικό στόχο. Καθώς η είσοδος στο Passive State ενδέχεται να βρίσκει τους Agent τύπου Pistol και Rifle σε κατάσταση που έχουν οπλίσει το όπλο στα χέρια τους (π.χ. από το state Seeking Enemy), η πρώτη ενέργεια των δέντρων είναι ο έλεγχος της κατάστασης του οπλισμού, ώστε ο Agent να αφοπλίζει αν είναι απαραίτητο. Στην περίπτωση των Agent τύπου Android, οι ενέργειες αφοπλισμού απουσιάζουν από το Behavior Tree, λόγω των χαρακτηριστικών του εχθρού ως προς τον οπλισμό. Σε αυτό το Project, κάθε Agent που τοποθετήθηκε στο level αρχικοποιείται στο Passive State και βρίσκεται ακίνητος στο σημείο αρχικοποίησης ή ξεκινά να ακολουθεί τη διαδρομή του Patrol που του αντιστοιχεί. Επιπρόσθετα, τοποθετήθηκαν στο level ορισμένα Trigger Boxes όπου μέσω του collision τους με έναν Actor (π.χ. ένας παίκτης ή κάποιος άλλος Agent περνά από ένα σημείο και έρχεται σε επαφή με το Trigger Box) αναθέτουν μια διαδρομή Patrol σε συγκεκριμένους Agents. Με αυτόν τον τρόπο δυναμικής ανάθεσης και αλλαγής των καθηκόντων του Agent, εντός του Passive State, μπορούν να υλοποιηθούν προσχεδιασμένα σενάρια. Σε αυτή την υποενότητα θα αναλυθεί η υλοποίηση του συγκεκριμένου state, συμπεριλαμβανομένων των Blueprint Actors, την ανάπτυξη των Subtrees και την ανάθεσή τους στο βασικό Behavior Tree, καθώς και την ανάπτυξη του κώδικα εντός του AI Controller.

8.2.1. Passive Behavior Tree Overview

Η εκτέλεση του Passive State απαιτεί το State (Enum) να έχει την τιμή Passive, ώστε το Blackboard Based Condition του Selector Passive State να επιτρέψει τη συνέχεια των ενεργειών που περιλαμβάνει. Σε αυτή την περίπτωση ο Selector θα επιλέξει αν ο Agent θα αφοπλίσει ή θα προχωρήσει κατευθείαν στα καθήκοντα που του έχουν ανατεθεί, δηλαδή ανάμεσα στις ενέργειες Unequip ή Patrol or Wait.

Ξεκινώντας από το Sequence Unequip ως τον πρώτο κόμβο του δέντρου, ελέγχεται η κατάσταση οπλισμού του Agent, ώστε το sequence να αρχίσει την εκτέλεση. Αυτό επιτυγχάνεται μέσω ελέγχου από τον Decorator `BTD_IsWeaponEquipped` και στην περίπτωση που ο Agent είναι οπλισμένος, το Sequence εκτελείται. Οι ενέργειες του sequence περιλαμβάνουν την επαναφορά του σημείου εστίασης του Agent στην αρχική κατάσταση μέσω του task `BTT_ClearFocus`, την αλλαγή της ταχύτητάς του σε ακινησία (Idle) μέσω του task `BTT_SetMovementSpeed` και τέλος το task αφοπλισμού `BTT_UnequipWeapon`.



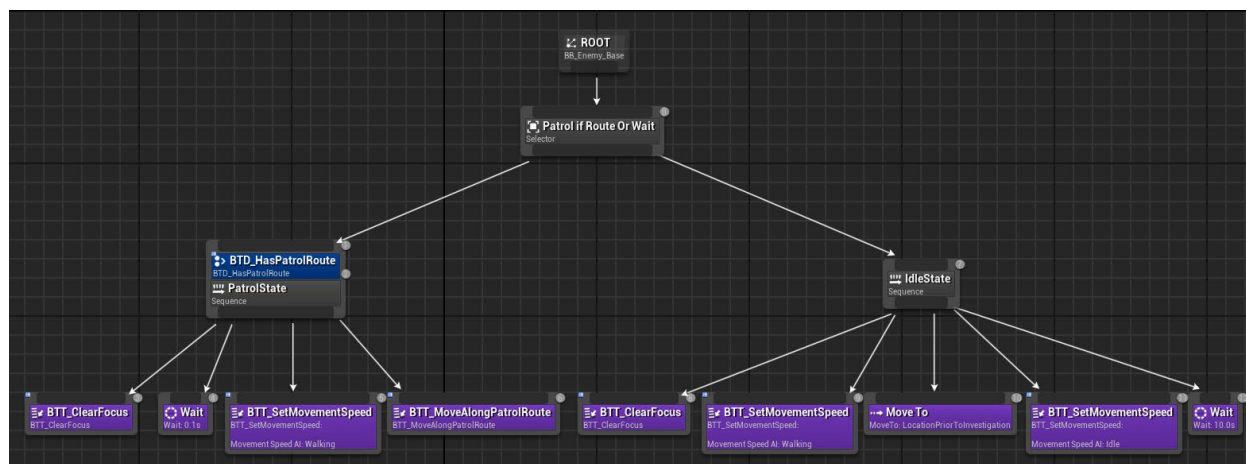
Εικόνα 63. Passive Behavior Tree Overview

Η πρώτη ενέργεια απαιτείται ώστε ο Agent να διαγράψει ένα ενδεχόμενο σημείο στο οποίο έχει στρέψει την προσοχή του και ως εκ τούτου την κατεύθυνση του σώματός του. Το δεύτερο task αφορά στην ακινητοποίηση του Agent, σε περίπτωση που προηγουμένως εκτελούσε κάποιο animation ή κινούνταν στον χώρο. Με αυτόν τον τρόπο ο Agent προετοιμάζεται για την εκτέλεση του τρίτου task, καθώς η αφόπλιση περιλαμβάνει την αναπαραγωγή animation που απαιτεί ακινησία. Στην περίπτωση που ο Decorator επιτρέπει με τιμή False, το Sequence Unequip παραλείπεται και ο Selector εκτελεί τον κόμβο Run Behavior Tree. Ο κόμβος δέχεται ως είσοδο το `BT_Subtree_Passive`, το υποδέντρο που περιέχει τα καθήκοντα αναμονής και περιπολίας. Η ενέργεια του αφοπλισμού βρίσκεται αποκλειστικά στο Behavior Tree των Agents τύπου Pistol και Rifle (`BT_Enemy_Ranged`), ενώ απουσιάζει από το Behavior Tree των Android (`BT_Enemy_Android`), καθώς οπλίζουν/αφοπλίζουν στιγμιαία κατά την επίθεσή τους και όχι καθ' όλη τη διάρκεια των καθηκόντων τους ανά state. Λόγω αυτού, το δέντρο του Passive State αποτελείται μόνο από τον κόμβο Run Behavior Tree, ο οποίος τρέχει το ίδιο υποδέντρο. Το μέρος του

Behavior Tree που αφορά στο Passive State για τους Agents τύπου Ranged παρουσιάζεται στην εικόνα 63.

Τέλος, για την ορθή διακοπή και έναρξη των απαραίτητων ενεργειών κατά την αλλαγή ενός state, χρησιμοποιήθηκαν οι ρυθμίσεις που παρέχει το blackboard based condition. Συγκεκριμένα, η επιλογή On Result Change ορίστηκε στο NotifyObserver, ώστε μια αλλαγή της τιμής του state key να προκαλέσει επαναξιολόγηση των δέντρων. Στη θέση του Observer Aborts ορίστηκε η επιλογή Both, ώστε κατά τη μεταβολή της τιμής του key, κάθε task του συγκεκριμένου state αλλά και των state χαμηλότερης προτεραιότητας να διακοπεί άμεσα, ώστε να αξιολογηθεί το δέντρο. Στις θέσεις των Key Query, Key Value και Blackboard Key ανατέθηκαν οι επιλογές IsEqualTo, Passive και State, ώστε οι ενέργειες του state να ενεργοποιηθούν, όταν η μεταβλητή State (Enum) έχει την τιμή Passive. Με αυτόν τον τρόπο, μία αλλαγή στο State key διακόπτει τη λειτουργία του state που εκτελείται, ώστε να επιλεγεί το κατάλληλο δέντρο βάσει της νέας τιμής του key.

PatrolIfRoute or Wait: Το υποδέντρο ξεκινά την εκτέλεσή του με τον Selector PatrolIfRoute or Wait, ο οποίος επιλέγει εάν ο Agent θα ακολουθήσει μια διαδρομή περιπολίας σε περίπτωση που του έχει ανατεθεί ή θα περιμένει στο σημείο που βρίσκεται, δηλαδή ανάμεσα στα Sequence του PatrolState ή IdleState. Ως ενέργεια υψηλής προτεραιότητας, το PatrolState sequence βρίσκεται πρώτο σε σειρά εκτέλεσης, περιλαμβάνοντας έλεγχο για την ύπαρξη διαδρομής περιπολίας του Agent μέσω του Decorator BTD_HasPatrolRoute. Στην περίπτωση που έχει ανατεθεί περιπολία, το Sequence θα καθαρίσει το σημείο εστίασης μέσω του task BTT_ClearFocus, η ταχύτητα μεταβάλλεται σε Walking, μέσω του task BTT_SetMovementSpeed, ώστε να μπορεί να κινηθεί ο Agent και ακολουθεί τη διαδρομή περιπολίας μέσω του task BTT_MoveAlongPatrolRoute. Αν δεν έχει ανατεθεί διαδρομή, το Patrol Sequence παραλείπεται και αρχίζει η εκτέλεση του Idle State sequence. Το σχετικό διάγραμμα παρουσιάζεται στην εικόνα 64.



Εικόνα 64. Subtree PatrolOrWait

Η ακολουθία Idle State περιλαμβάνει τον καθαρισμό του σημείου εστίασης, την αλλαγή της ταχύτητας σε μηδενική και την αναμονή του δέντρου με τον κόμβο Wait, αφήνοντας τον παίκτη ακίνητο. Η ολοκλήρωση του task Wait, θα σηματοδοτήσει την ολοκλήρωση του IdleState και αυτή με τη σειρά της την ολοκλήρωση του Subtree PatrolOrWait. Με αυτόν τον τρόπο το δέντρο θα επαναξιολογηθεί, καταλήγοντας και πάλι στο Idle State επαναλαμβάνοντας τη διαδικασία. Ωστόσο, ένας Agent στον οποίο δεν έχει ανατεθεί κάποιο σημείο περιπολίας, αλλά έχει τοποθετηθεί να προφυλάσσει στάσιμος ένα συγκεκριμένο σημείο, μπορεί να εισέλθει στο Idle State, αφού προηγουμένως ερευνήσει ένα σημείο ηχητικής διατάραξης (π.χ. εκτελώντας το state Investigating). Κάτι τέτοιο θα προκαλούσε τον Agent να μετακινηθεί από την αρχική του θέση για να ερευνήσει το σημείο, με αποτέλεσμα να μείνει στο σημείο της διατάραξης χωρίς να γυρίσει στην αρχική του θέση, σε περίπτωση που το state αλλάξει πάλι σε Passive. Για την αποφυγή αυτού του σεναρίου, η εκκαθάριση του σημείου εστίασης ακολουθείται από την αλλαγή της ταχύτητας σε Walking και τη μετακίνηση του Agent στο σημείο στο οποίο βρισκόταν πριν μεταβάλει τη θέση του. Αυτό επιτυγχάνεται μέσω του κόμβου MoveTo, με είσοδο το περιεχόμενο του blackboard key LocationPriorToInvestigation (Location). Το blackboard key αναπαριστά τη θέση του Agent πριν εκτελέσει μια ενέργεια που προκαλεί την αλλαγή της θέσης του στον χώρο. Το περιεχόμενό του ορίζεται σύμφωνα με τη μεταβλητή ίδιας ονομασίας που βρίσκεται στο AI Controller, η τιμή της οποίας ορίζεται κατά την αρχικοποίηση του Agent στον χώρο, αλλά και κάθε φορά που εισέρχεται σε καταστάσεις που τον προκαλούν να μετακινηθεί (π.χ. Seeking State).

8.2.2. Patrol Route

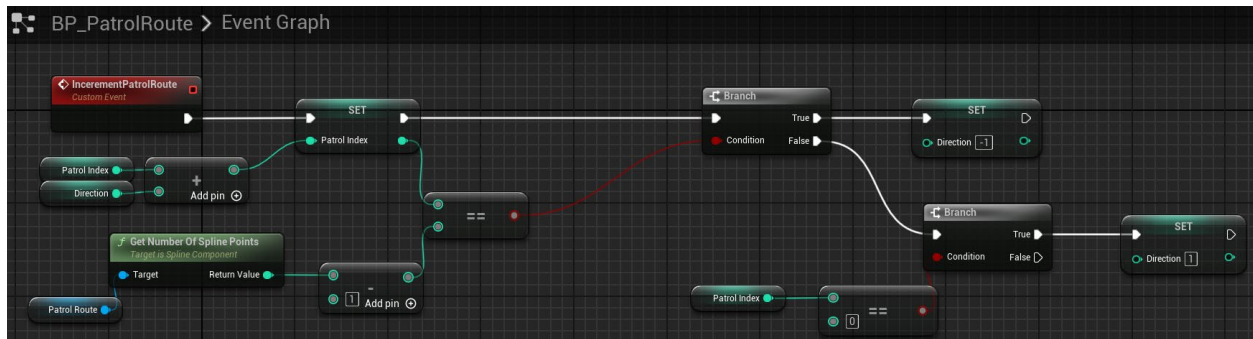
BP_PatrolRoute (Blueprint Actor): Είναι ο Actor που τοποθετείται στο περιβάλλον και χρησιμοποιείται από τον Agent ως διαδρομή περιπολίας. Το Blueprint περιέχει ένα SplineComponent το οποίο ορίζει τη διαδρομή που ακολουθεί ο Agent κατά τη διάρκεια της περιπολίας του, όπως φαίνεται στην εικόνα 65. Το SplineComponent είναι ένα εργαλείο που επιτρέπει τον ορισμό και τη χρήση διαδρομών στον τρισδιάστατο χώρο. Επιτρέπει την κίνηση Actors (ή άλλων Components) κατά μήκος καμπυλωτών διαδρομών (splines) με ακρίβεια. Οι διαδρομές αυτές είναι πλήρως επεξεργάσιμες, τόσο στο Blueprint, όσο και στον Level Editor, προσφέροντας τη δυνατότητα προσθήκης, αφαίρεσης ή αντιγραφής Spline Points. Με αυτόν τον τρόπο, πολλαπλοί Actors τύπου BP_PatrolRoute μπορούν να προστεθούν στο Level, να επεξεργαστούν τα σημεία που θα ακολουθήσει ο Agent (spline points) και να ανατεθούν σε αυτόν εντός του viewport. Για να καταστεί δυνατή η ανάθεση του Actor σε έναν Agent, δημιουργήθηκε η μεταβλητή PatrolRoute, τύπου BP_PatrolRoute, εντός του Parent Blueprint BP_Enemy_Base.



Εικόνα 65. SplineComponent (PatrolRoute)

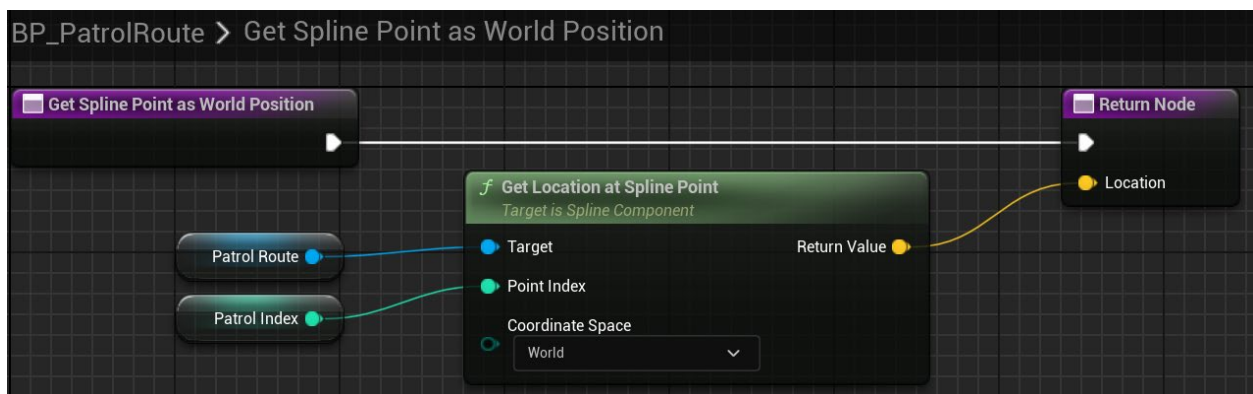
Increment Patrol Route (Custom Event): Η λειτουργία αυτού του event είναι να βρίσκει το τρέχον σημείο στο οποίο βρίσκεται ο Agent πάνω στο Spline και να εντοπίζει την τοποθεσία του επόμενου. Για την επίτευξη της λειτουργίας δημιουργήθηκαν οι μεταβλητές PatrolIndex (Integer) που αντιπροσωπεύει τον αριθμό που αντιστοιχεί στο σημείο στο οποίο βρίσκεται ο Agent και η Direction (Integer) που αφορά στην κατεύθυνση στην οποία κινείται ο Agent σε σχέση με το Spline. Η Direction παίρνει αποκλειστικά τις τιμές 1 και -1, με την πρώτη να σηματοδοτεί πως ο Agent κινείται από το αρχικό σημείο του Spline προς το τελικό και την δεύτερη το αντίστροφο. Η κλήση του event αυξάνει την τιμή της μεταβλητής Patrol Index κατά την τιμή της μεταβλητής Direction. Αυτό σημαίνει πως αν ο Agent κινείται εμπρός, το επόμενο σημείο στο Spline θα βρεθεί προσθέτοντας 1, ενώ αν κινείται προς τα πίσω η μεταβλητή PatrolIndex θα μειωθεί. Στη συνέχεια, πραγματοποιείται έλεγχος για την ενδεχόμενη ολοκλήρωση της διαδρομής. Σε αυτό το σημείο, χρησιμοποιείται το node GetNumberOfSplinePoints του spline component PatrolRoute, το οποίο επιστρέφει τον αριθμό των σημείων διαδρομής που έχουν οριστεί στον τρέχοντα Actor. Ο έλεγχος εξετάζει εάν ο αριθμός που επιστρέφει το node είναι ίσος με το τρέχον PatrolIndex. Ωστόσο, ο αριθμός με τον οποίο θα συγκριθεί το PatrolIndex είναι ελαττωμένος κατά μια μονάδα, αφού η πρώτη τιμή που λαμβάνει το PatrolIndex είναι μηδενική. Εάν ο έλεγχος αποδειχθεί True, η μεταβλητή λαμβάνει την τιμή -1. Σε αντίθετη περίπτωση, ακολουθεί έλεγχος που εξετάζει εάν η τιμή της PatrolIndex είναι μηδενική, σηματοδοτώντας πως ο Agent ολοκλήρωσε τη διαδρομή κατευθυνόμενος προς τα πίσω, που σημαίνει πως η Direction πρέπει να λάβει ξανά την τιμή 1. Με αυτόν τον τρόπο η διαδρομή ξεκινά

από το αρχικό Spline point με Direction τιμής 1 και αυξάνεται όταν Agent φτάσει στο επόμενο point. Όταν ο Agent ολοκληρώσει τη διαδρομή φτάνοντας στο τελευταίο σημείο, η διαδικασία ακολουθείται αντίστροφα με Direction τιμής -1, ώστε ο Agent να οδηγηθεί στο τελευταίο σημείο και η ενέργεια να επαναληφθεί. Το event καλείται επαναλαμβανόμενα εντός του Behavior Tree, ώστε ο Agent να ακολουθήσει το επόμενο σημείο της διαδρομής.



Εικόνα 66. Increment Patrol Route

GetSplinePointsAsWorldPosition (Function): Η συνάρτηση είναι υπεύθυνη για την εύρεση των συντεταγμένων ενός spline point στον χώρο του level, ώστε να οριστεί η τοποθεσία προς την οποία πρέπει να κινηθεί ο Agent. Σε κάθε κλήση της χρησιμοποιεί το node GetLocationAtSplinePoint, με εισόδους το spline component PatrolRoute και το PatrolIndex, επιστρέφοντας την τοποθεσία του τρέχοντος spline point ως μεταβλητή Location (Vector).

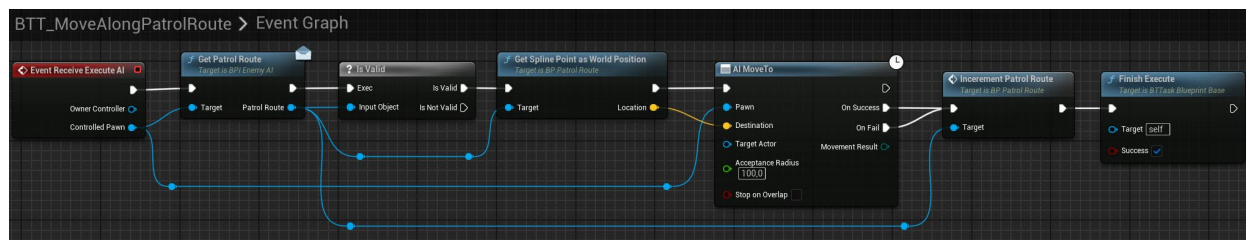


Εικόνα 67. GetSplinePointsAsWorldPosition

8.2.3. Passive State Assets

BTT_MoveAlongPatrolRoute (Custom Task): Το Custom Task αναλαμβάνει την εκτέλεση της ενέργειας του Agent κατά την οποία ακολουθεί τη διαδρομή PatrolRoute που του έχει ανατεθεί. Ο

κώδικας του περιλαμβάνει την ανάκτηση της διαδρομής ενός Agent και την ανάκτηση της τοποθεσίας του επόμενου σημείου στόχου από τη διαδρομή, καθώς και την εντολή που θα πραγματοποιήσει την κίνηση προς το σημείο αυτό. Καλείται εντός του Behavior Tree στην ακολουθία ενεργειών ενός Agent με καθήκοντα περιπολίας.

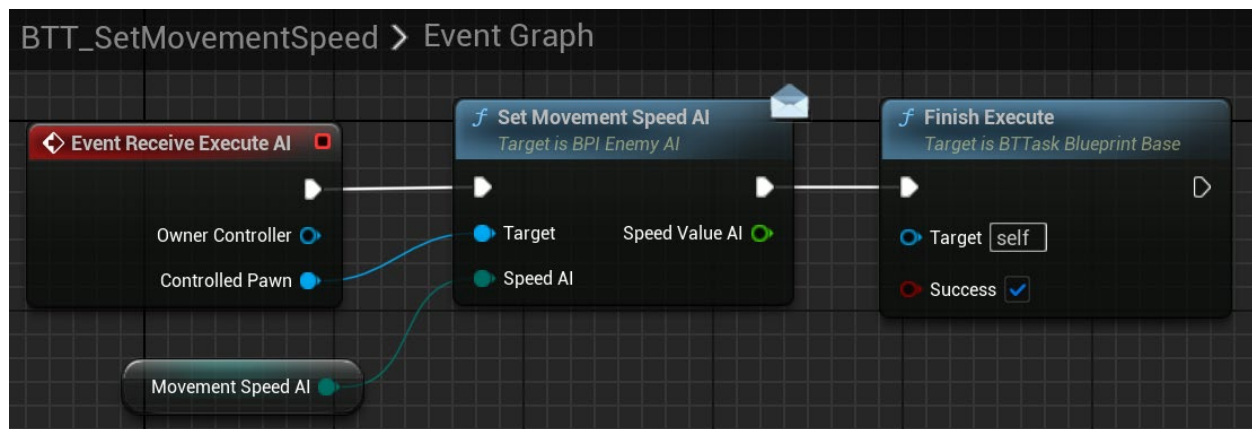


Εικόνα 68. BTT_MoveAlongPatrolRoute

Κατά την εκκίνηση του task χρησιμοποιείται η έξοδος Controlled Pawn, η οποία παρέχεται από το node Event Execute AI και αποτελεί ένα τρόπο πρόσβασης των λειτουργιών του Agent που χρησιμοποιεί το Behavior Tree. Με στόχο αυτό το reference καλείται η function GetPatrolRoute, μέσω του Blueprint Interface BPI_Enemy_AI, η οποία επιστρέφει το περιεχόμενο της μεταβλητής PatrolRoute, δηλαδή το Asset που έχει ανατεθεί στον Agent. Η ανάκτηση του PatrolRoute γίνεται μέσω Blueprint Interface, ώστε να αποφευχθεί η χρήση Cast, για την πρόσβαση στις λειτουργίες του Blueprint του Agent, κάτι το οποίο μπορεί να επιβαρύνει το υπολογιστικό κόστος. Ακόμη, μια κλήση μέσω Interface δεν απαιτεί να υπάρχει υλοποίηση της λειτουργίας από τον Actor στον οποίο καλείται, με αποτέλεσμα σε αυτή την περίπτωση η κλήση της να αγνοείται χωρίς να υπάρξει κάποιο σφάλμα. Στη συνέχεια, για λόγους καλής πρακτικής χρησιμοποιείται το node IsValid ώστε να επιβεβαιωθεί πως η μεταβλητή PatrolRoute έχει περιεχόμενο, αν δηλαδή έχει ανατεθεί στον Agent κάποια διαδρομή. Αν υπάρχει περιεχόμενο, η εκτέλεση προχωρά κανονικά στην κλήση της function GetSplinePointsAsWorldPosition που δημιουργήθηκε στο BP_PatrolRoute, επιστρέφοντας την τοποθεσία (Location) του σημείου προς το οποίο πρέπει να κατευθυνθεί ο Agent στο Level. Έπειτα, η τοποθεσία χρησιμοποιείται ως είσοδος στο node AIMoveTo που παρέχει το Unreal Engine, ώστε ο Agent να κινηθεί προς το ορισμένο σημείο. Το node απαιτεί ως είσοδο το Pawn (Actor) που εκτελεί την εντολή, στο οποίο παρέχεται το Controlled Pawn reference του EventExecuteAI. Μια ακόμη είσοδος του node αποτελεί το Acceptance Radius, όπου λειτουργεί ως την ακτίνα ή την απόσταση μέσα στην οποία ο Agent θεωρεί ότι έχει φτάσει στον προορισμό του. Όταν ο Agent φτάσει στον προορισμό του, καλείται η function IncrementPatrolRoute από το reference του PatrolRoute ώστε στην επόμενη κλήση του task, να ανακτηθεί η τοποθεσία του επόμενου σημείου. Τέλος, το task ολοκληρώνεται με το node FinishExecute (Success=True), ώστε το Behavior Tree να προχωρήσει στην εκτέλεση του επόμενου task στο sequence.

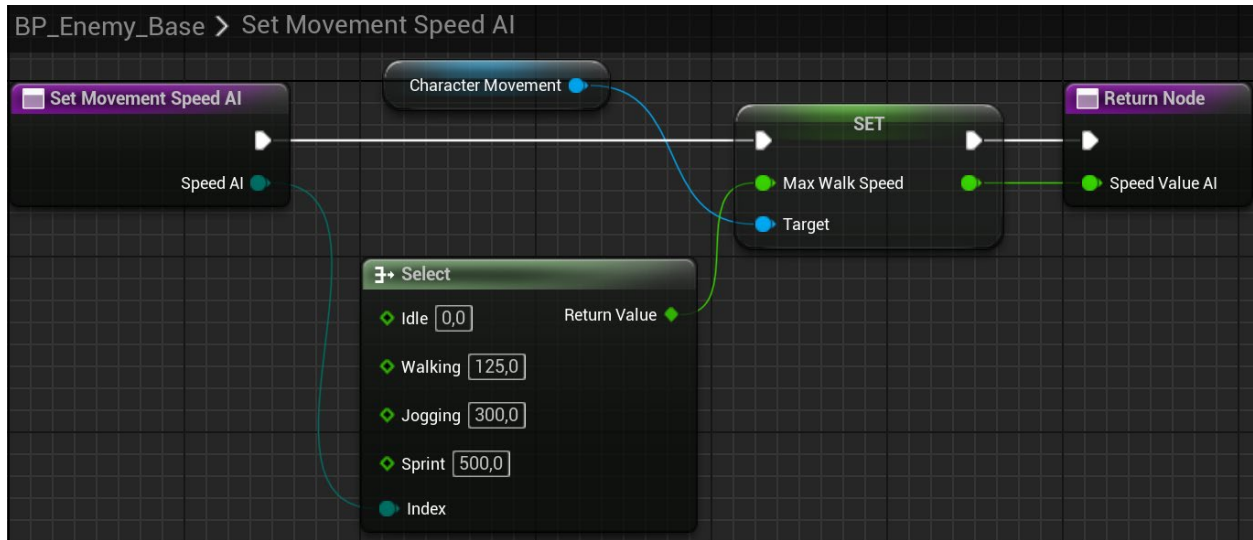
BTT_SetMovementSpeed (Custom Task): Το συγκεκριμένο task είναι υπεύθυνο για την αλλαγή της ταχύτητας με την οποία κινείται ένας Agent. Όπως αναφέρθηκε και στην ανάλυση του Animation Blueprint, ένας Agent μπορεί να κινηθεί με ταχύτητες Idle, Walk, Jog και Run, που ορίζονται μέσω του CharacterMovementComponent. Οι τιμές αυτές καθορίζουν την ταχύτητα με την οποία κινείται ο Agent

στον χώρο, καθώς και την αναπαραγωγή του αντίστοιχου animation/blendspace. Στο συγκεκριμένο Project χρησιμοποιείται για να μεταβάλει τη μεταβλητή MaxWalkSpeed (float) που παρέχεται από το Character Movement Component, ώστε να αλλάξουν η ταχύτητα και τα animations ενός Agent. Με αυτόν τον τρόπο κάθε state, αλλά και οποιαδήποτε ακολουθία του Behavior Tree μπορούν να εκτελεστούν από τον Agent σε διαφορετική ταχύτητα και ως εκ τούτου διαφορετικά animations. Για παράδειγμα, το state Investigate, κατά το οποίο ο Agent διερευνά μια ηχητική διατάραξη, εκτελείται με την ταχύτητα Walk, καθώς κάτι τέτοιο δεν αποτελεί άμεση εχθρική απειλή. Αντιθέτως το state Seek, κατά το οποίο ο Agent αναζητά τον παίκτη, εκτελείται με ταχύτητα Jog. Αυτό παρέχει υψηλό επίπεδο ρεαλισμού, καθώς η ταχύτητα κίνησης και τα animations εναρμονίζονται σύμφωνα με την ένταση που χαρακτηρίζει κάθε state και λειτουργεί ως άμεση ένδειξη της κατάστασης του Agent προς τον παίκτη. Το συγκεκριμένο Task καλείται σε πολλαπλά σημεία του Behavior Tree, με αποτέλεσμα οι Agents να κινούνται με διαφορετικές ταχύτητες ανά state ή ακολουθία ενεργειών σύμφωνα με τον τύπο της ενέργειας καθώς και τον τύπο του Agent. Όσον αφορά στην υλοποίηση, το task δέχεται ως είσοδο το Blackboard Key MovementAI (Enum), επιτρέποντας σε κάθε κλήση του task από διαφορετικά σημεία του Behavior Tree, να επιλεγεί η επικείμενη αλλαγή στο περιεχόμενο της μεταβλητής (πχ Walk ή Jog).



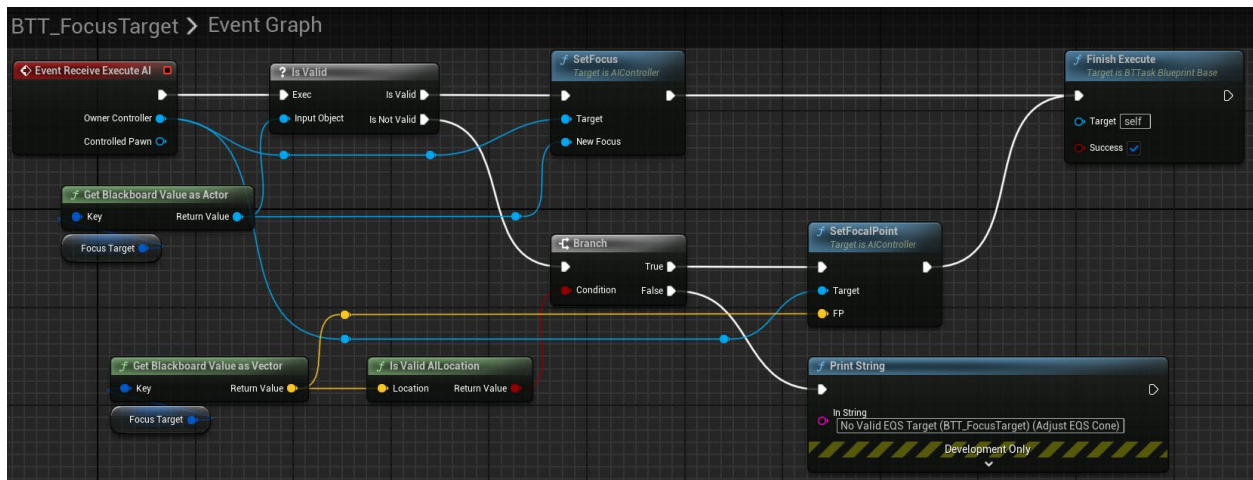
Εικόνα 69. BTT_SetMovementSpeed

Το task καλεί την Blueprint Interface function SetMovementSpeedAI του BPI_Enemy_AI, η οποία δέχεται ως είσοδο το Blackboard Key και επιστρέφει την τιμή στην οποία ορίστηκε η μεταβλητή όπως φαίνεται στην εικόνα 69. Το function υλοποιείται στο Parent Blueprint BP_Enemy_Base, θέτοντας διαφορετικές τιμές στη μεταβλητή MaxWalkSpeed, σύμφωνα με το επιλεγμένο Enum από το task. Το σχετικό διάγραμμα παρουσιάζεται στην εικόνα 70.



Εικόνα 70. SetMovementSpeedAI (function)

BTT_FocusTarget (Custom Task): Το task είναι υπεύθυνο για την ανάθεση ενός σημείου εστίασης (focus) σε έναν Agent, επιτρέποντας του να στρέψει την προσοχή του σε έναν συγκεκριμένο Actor ή σημείο στον χώρο. Αυτό επιτυγχάνεται μέσω του node SetFocus και έχει ως αποτέλεσμα την ενημέρωση του FocalPoint του Agent, επηρεάζοντας τη συμπεριφορά του, όπως την κατεύθυνση κίνησης ή τη στόχευση. Με αυτόν τον τρόπο, ένας Agent που έχει κάποιο σημείο εστίασης στρέφει το σώμα του προς αυτή την κατεύθυνση ακόμη και αν το σημείο κινείται στον χώρο.

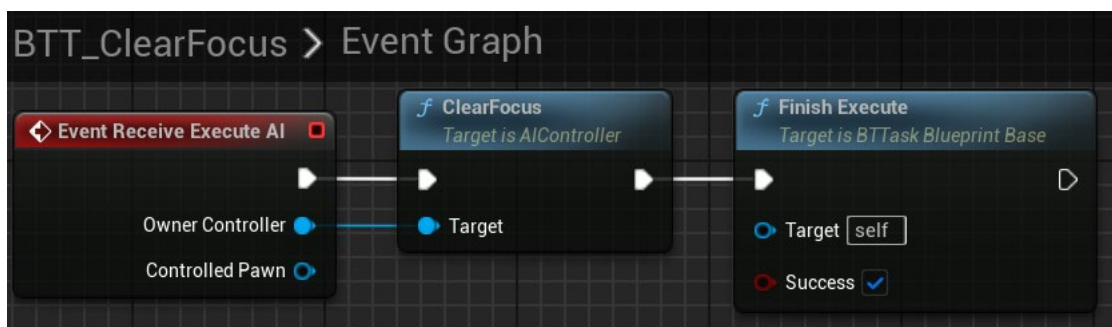


Εικόνα 71. BTT_FocusTarget

Το SetFocus δέχεται τις εισόδους Target και New Focus, οι οποίες αντιπροσωπεύουν το AI Controller του Agent και τον νέο στόχο εστίασης αντίστοιχα. Στο συγκεκριμένο Project χρησιμοποιήθηκε σε

πολλαπλά states όπως Seeking ή Attacking, όπου απαιτείται η διαρκής εστίαση του παίκτη σε ένα σημείο ή στον παίκτη αντίστοιχα. Ακόμη, η ανάθεση ενός σημείου εστίασης επηρεάζει την επιλογή του blendspace που θα χρησιμοποιήσει ένας Agent, καθώς ορίζει την μεταβλητή IsFocusingTarget, όπως αναφέρθηκε στο υποκεφάλαιο 6.3 του Animation Blueprint. Η υλοποίηση περιλαμβάνει αρχικά, τη δημιουργία της μεταβλητής FocusTarget (Blackboard Key Selector) εντός του Task, ώστε να μπορεί να χρησιμοποιηθεί σε διαφορετικά μέρη του Behavior Tree με διαφορετικούς στόχους εστίασης ανά σημείο κλήσης. Κάθε κλήση του task απαιτεί την εκχώρηση ενός τέτοιου key, από τα διαθέσιμα keys του Blackboard, όπως PointOfInterest (Location) ή AttackTarget (Actor). Με αυτό τον τρόπο ένας Agent μπορεί να εστιάσει σε μια τοποθεσία ή έναν actor. Κατά την κλήση του task, πραγματοποιείται έλεγχος για τον τύπο του Key που εκχωρήθηκε από το Behavior Tree μέσω του node GetBlackboardValueAsActor και GetBlackboardValueAsLocation αντίστοιχα. Στην περίπτωση που το node IsValid είναι αληθές, καλείται το function SetFocus με τις κατάλληλες εισόδους και το task ολοκληρώνεται. Σε αντίθετη περίπτωση, καλείται το SetFocalPoint με τις κατάλληλες εισόδους, αφού προηγηθεί έλεγχος πως το Blackboard Key αποτελεί έγκυρη τοποθεσία για τον Agent, δηλαδή εάν το σημείο είναι προσβάσιμο σύμφωνα με το NavMesh μέσω του node IsValidAILocation. Ο έλεγχος είναι απαραίτητος για λόγους αποσφαλμάτωσης, ώστε σε αντίθετη περίπτωση να μην οριστεί κάποιο σημείο εστίασης στο οποίο είναι πιθανό να προκληθεί κάποιο σφάλμα.

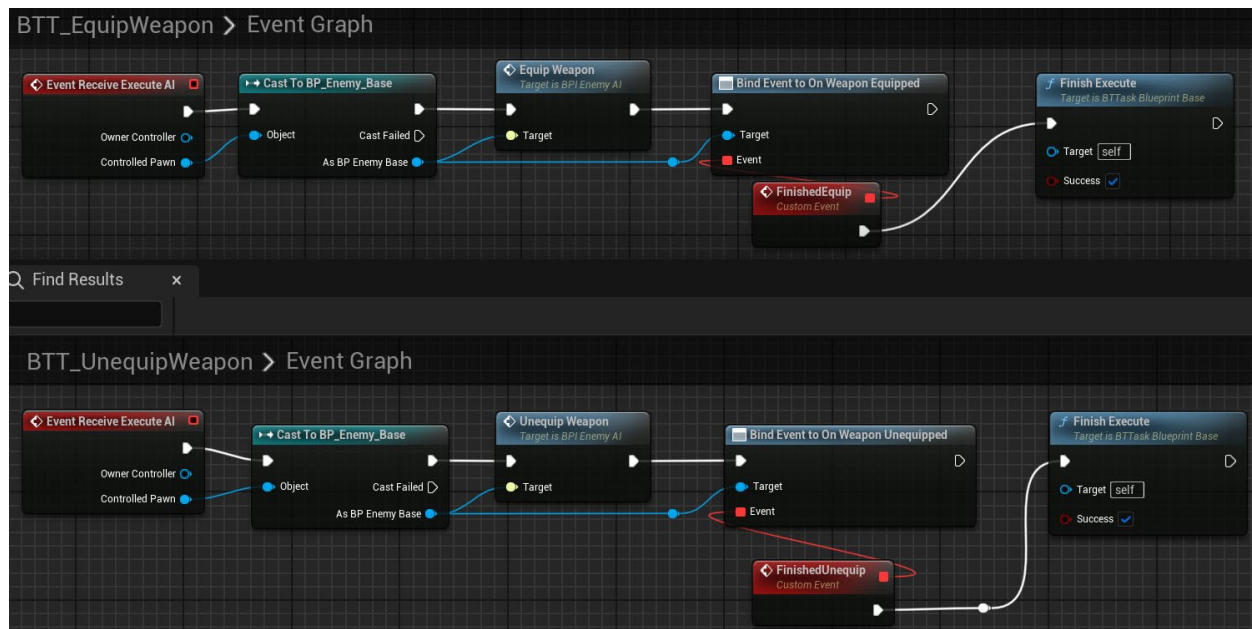
BTT_ClearFocus (Custom Task): Το task είναι υπεύθυνο για την αποδέσμευση ενός Agent από το τρέχον σημείο ή στόχο εστίασης, ώστε να συνεχίσει την κανονική του πορεία ή να εστιάσει σε κάτι νέο. Αυτό επιτυγχάνεται με τη χρήση του node ClearFocus με target το OwnerController, δηλαδή το AIC_Enemy_Base. Κάθε κλήση του επαναφέρει την εστίαση του Agent στην προεπιλεγμένη κατάσταση. Χρησιμοποιείται ως αρχική ενέργεια σε πολλαπλά states ή sequences, ώστε ο Agent να εκτελέσει τις ενέργειες του δέντρου χωρίς κάποιο προϋπάρχον σημείο εστίασης που θα τις επηρεάσει. Αυτό μπορεί να συμβεί όταν ο Agent εισέρχεται σε κάποιο state (π.χ. Passive), ενώ έχει προηγηθεί η εκτέλεση κάποιου state ή sequence στο οποίο ορίστηκε σημείο εστίασης μέσω του BTT_FocusTarget (π.χ. Attacking ή Seeking).



Εικόνα 72. BTT_ClearFocus

BTT_EquipWeapon και BTT_UnequipWeapon (Custom Task): Τα συγκεκριμένα tasks είναι υπεύθυνα για την όπλιση/αφόπλιση των Agents τύπου Pistol και Rifle, ώστε να τοποθετήσουν τον

οπλισμό τους στην αρχική θέση ή στα χέρια τους αντίστοιχα. Ο οπλισμός και αφοπλισμός του τύπου Android εκτελείται μόνο εντός της επίθεσης AerialAttack εντός Blueprint, αφού ο Agent οπλίζει και αφοπλίζει στιγμιαία και δεν εκτελεί ενέργειες που απαιτούν να έχει το όπλο στα χέρια του. Το πρώτο task χρησιμοποιείται στην έναρξη των ενεργειών που απαιτούν υψηλή ετοιμότητα όπως Seeking και Attacking, ενώ το δεύτερο χρησιμοποιείται στην έναρξη του Passive State όπου οι ενέργειες του Agent δε χαρακτηρίζονται από ενδεχόμενη απειλή. Η αφοπλισση απαιτείται στην περίπτωση που ο Agent εισέρχεται στο Passive State, ενώ βρισκόταν σε κάποιο State υψηλής ετοιμότητας (π.χ. Attacking ή Seeking), κατά το οποίο έχει οπλίσει. Καθώς τα δύο tasks ακολουθούν πανομοιότυπη δομή, ενδεικτικά θα αναλυθεί το task BTT_UnequipWeapon.

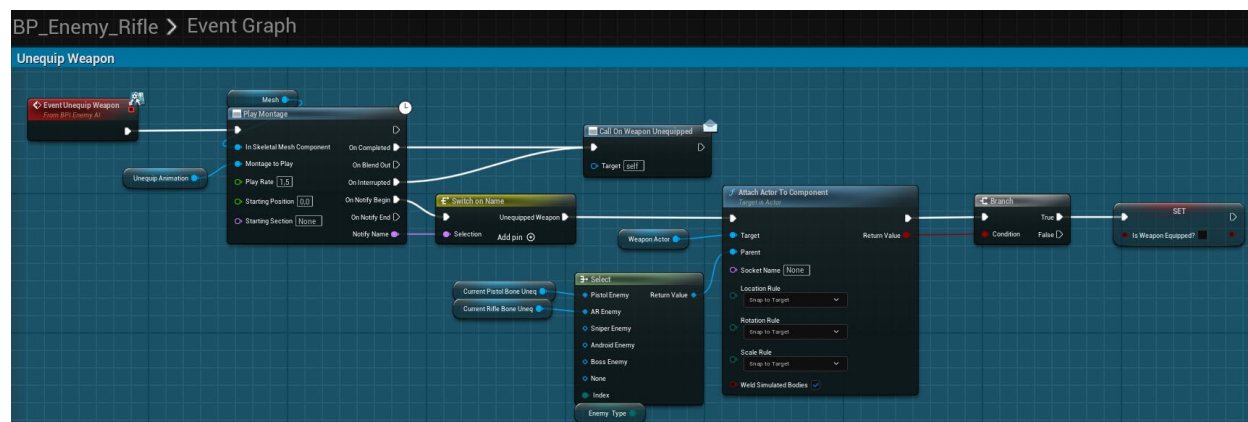


Εικόνα 73. BTT_EquipWeapon και BTT_UnequipWeapon

Το task περιλαμβάνει τη χρήση Cast to BP_Energy_Base, ώστε να αποκτηθεί πρόσβαση στις λειτουργίες του Parent Blueprint των Agents. Το reference που παρέχει, επιτρέπει την κλήση του Blueprint Interface function UnequipWeapon, το οποίο εφαρμόζει την αφοπλισση. Ωστόσο, το function UnequipWeapon περιλαμβάνει την εκτέλεση animation αφοπλισμού, η οποία απαιτεί ορισμένο χρόνο για να ολοκληρωθεί. Αυτό σημαίνει πως η ολοκλήρωση του task πρέπει να πραγματοποιηθεί μετά την ολοκλήρωση του αφοπλισμού. Το πρόβλημα έγκειται στον τρόπο λειτουργίας των Behavior Trees, καθώς όταν ολοκληρώνεται ένα task με το node Finish Execute, το δέντρο προχωρά στην εκτέλεση του επόμενου task. Λόγω της παραπάνω συνθήκης, η κλήση του WeaponUnequip, εντός του task, ακολουθείται από το node Bind Event to WeaponUnequipped, το οποίο επιτρέπει τη δημιουργία ενός καινούριου Custom Event FinishedUnequip που θα πυροδοτήσει την ολοκλήρωση του task. Το FinishedUnequip εκτελείται κατά την κλήση του Event Dispatcher WeaponUnequipped, το οποίο δημιουργήθηκε εντός του Parent Blueprint των Agents και καλείται μετά την ολοκλήρωση του αφοπλισμού. Με αυτόν τον τρόπο, το task

καλεί το `WeaponUnequip`, το οποίο με τη σειρά του καλεί το `Event Dispatcher CallOnWeaponUnequipped`, πυροδοτώντας το `FinishedUnequip`. Στη συνέχεια, το task ολοκληρώνεται με επιτυχία και το δέντρο εκτελεί την επόμενη ενέργεια. Η απουσία του `Event Dispatcher` θα προκαλούσε ανεπιθύμητη συμπεριφορά των `Agents`, όπως ακανόνιστες ή ατελείς κινήσεις του χαρακτήρα, με αποτέλεσμα να παραμένει οπλισμένος ακόμα και αν το task θεωρηθεί ότι ολοκληρώθηκε, κάτι που θα επηρεάσει τη συνολική συμπεριφορά του `Agent` στο παιχνίδι. Η σωστή αλληλουχία των events εξασφαλίζει ότι η μετάβαση από την κατάσταση του οπλισμένου `Agent` σε μη οπλισμένο γίνεται ομαλά και χωρίς παρεκκλίσεις από την αναμενόμενη συμπεριφορά. Ακολουθώντας τον ίδιο τρόπο δημιουργήθηκε το `BTT_EquipWeapon` για την όπλιση. Τα δύο γραφήματα των tasks παρουσιάζονται στην εικόνα 73.

EquipWeapon και Unequip Weapon (Function): Τα functions είναι υπεύθυνα για την εφαρμογή της όπλισης/αφόπλισης στους `Agents` τύπου `Pistol` και `Rifle` και καλούνται εντός των `Behavior Tree Tasks` `BTT_EquipWeapon` και `BTT_UnequipWeapon` αντίστοιχα. Τα functions καλούνται να εκτελεστούν στο `Parent Blueprint BP_Energy_Base`, ωστόσο υλοποιούνται στα επιμέρους `Child Blueprints` των `Pistol` και `Rifle Agent` για τη δυνατότητα μελλοντικής αναδιαμόρφωσης των ενεργειών για κάθε τύπου `Agent`. Καθώς τα δύο functions ακολουθούν πανομοιότυπη δομή, ενδεικτικά θα αναλυθεί το `Unequip Weapon`.

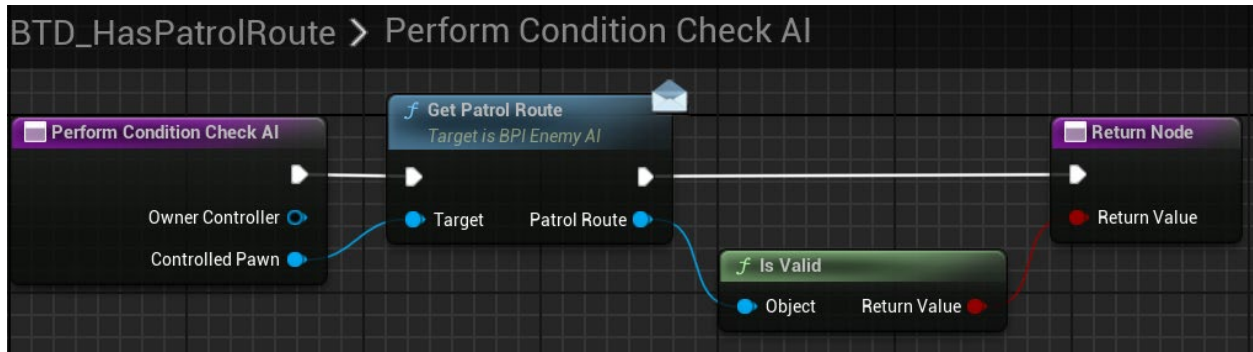


Εικόνα 74. Unequip Weapon

Η εκτέλεση αρχίζει με την κλήση του node `PlayMontage` για την αναπαραγωγή του `Animation`. Το συγκεκριμένο node προτιμήθηκε έναντι άλλων (π.χ. `PlayAnimMontage`), λόγω των επιπλέον `Execution Pins` που προσφέρει, για τις περιπτώσεις, κατά τις οποίες το `Animation` ολοκληρωθεί/διακοπεί (`OnCompleted/OnInterrupted`), αλλά και την ανάπτυξη ενεργειών σε συγκεκριμένα σημεία του `Animation` (`OnNotifyBegin`). Το `Animation` αναπαράγει την κίνηση αφοπλισμού, καθώς το όπλο (`WeaponActor`) βρίσκεται στα χέρια του `Agent` προσκολλημένο στο socket `CurrentWeaponTypeBoneEq` του σκελετού και στο κατάλληλο χρονικό σημείο αναθέτει το όπλο στο socket `CurrentWeaponTypeBoneUneq`. Αυτό επιτυγχάνεται με τον ορισμό ενός `MontageNotify`, εντός του `Animation`, στο κατάλληλο σημείο, που λειτουργεί ως ειδοποίηση για την ανάθεση του όπλου. Η

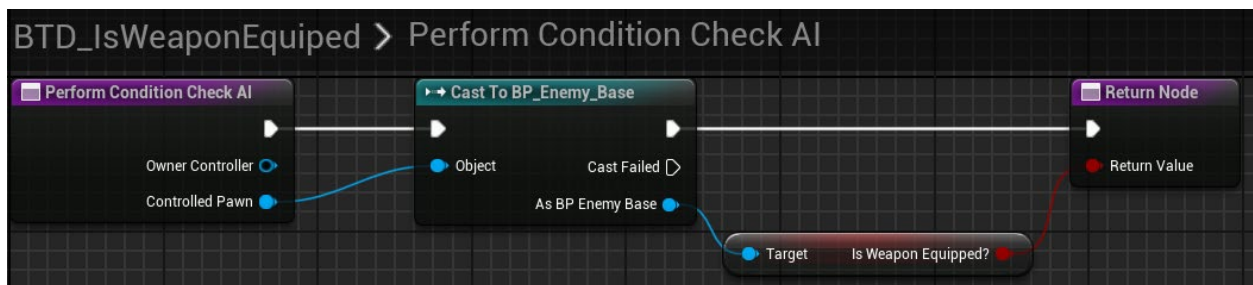
ανάθεση χρησιμοποιεί το node `AttachActorToComponent` για να προσκολλήσει το `WeaponActor` στο κατάλληλο socket με μεταβλητές εισόδου το `Asset` που θα ανατεθεί (`Target`) και τον στόχο (`Parent`). Η εκτέλεση πυροδοτείται με τη χρήση του `switch on name`, απαραίτητου node για την εκτέλεση οποιουδήποτε `MontageNotify` σύμφωνα με το όνομά του. Η είσοδος για το `Parent` αποτελεί ένα `Select` node, όπου βάσει της μεταβλητής `EnemyType` (`Enum`) επιλέγει την κατάλληλη θέση socket (`CurrentPistolBoneUneq` ή `CurrentRifleBoneUneq`). Με αυτόν τον τρόπο το όπλο τοποθετείται ανάλογα με την κατηγορία του, καθώς διαθέτει διαφορετική θέση ανά περίπτωση. Η ανάθεση ακολουθείται από έλεγχο επιβεβαίωσης της ενέργειας και έπειτα η μεταβλητή `IsWeaponEquipped?` (`Boolean`) παίρνει την τιμή `False` για περαιτέρω χρήση σε σημεία που αφορούν στην κατάσταση όπλισης. Στις περιπτώσεις κατά τις οποίες το `Animation` ολοκληρώνεται ή διακόπτεται από κάποια άλλη ενέργεια (π.χ. κάποιο άλλο `Animation`), καλείται το `Event Dispatcher WeaponUnequipped`, ώστε να ολοκληρωθεί το `Behavior Tree Task`, όπως περιγράφεται στην παράγραφο `BTT_EquipWeapon` και `BTT_UnequipWeapon`. Τέλος, το `Animation Asset` που θα χρησιμοποιηθεί για την αναπαραγωγή εισάγεται στη μεταβλητή `UnequipAnimation` και ανατίθεται στα `ChildBlueprint BP_Enemy_Pistol` και `BP_Enemy_Rifle`. Ομοίως, το function `EquipWeapon` αναπαράγει το `Animation` όπλισης από το περιεχόμενο της μεταβλητής `EquipAnimation`, προσκολλά τον `Weapon Actor` από το socket `CurrentWeaponTypeBoneUneq` στο `CurrentWeaponTypeBoneEq`, θέτει τη μεταβλητή `IsWeaponEquipped?` στην τιμή `True` και καλεί το `Event Dispatcher CallOnWeaponEquipped?` για την ολοκλήρωση του task. Το γράφημα του function παρουσιάζεται στην εικόνα 74.

BTD_HasPatrolRoute (Blackboard Decorator): Ο συγκεκριμένος `Decorator` είναι υπεύθυνος για τον έλεγχο ανάθεσης περιπολίας στον `Agent`. Χρησιμοποιείται ως έλεγχος για τον καθορισμό της εκτέλεσης των sequences `PatrolState` ή `IdleState`. Τοποθετείται στο sequence `PatrolState`, ώστε το sequence να εκτελεστεί, στην περίπτωση που η μεταβλητή `PatrolRoute` έχει περιεχόμενο. Σε αντίθετη περίπτωση το δέντρο θα εκτελέσει το sequence `IdleState`. Αυτό επιτυγχάνεται με τη χρήση του function `Perform Condition Check AI`, το οποίο παρέχει τα references των `Controller` και `Agent` και επιστρέφει μια μεταβλητή `Boolean`. Μέσω του `Controlled Pawn reference` καλείται το `Blueprint Interface function GetPatrolRoute`, το οποίο ανακτά τη μεταβλητή `PatrolRoute` ώστε να ελεγχθεί αν έχει περιεχόμενο. Η μεταβλητή χρησιμοποιείται ως είσοδος στο function `IsValid` και ο `Decorator` ολοκληρώνει τη λειτουργία του επιστρέφοντας το αποτέλεσμα. Με αυτόν τον τρόπο, ο `Decorator` γίνεται `True`, όταν το `PatrolRoute` είναι έγκυρο και εκτελεί το sequence στο οποίο χρησιμοποιείται. Αν ο `Decorator` γίνει `False`, οι ενέργειες `PatrolState` παραλείπονται και το δέντρο προχωρά στην εκτέλεση του `IdleState`.



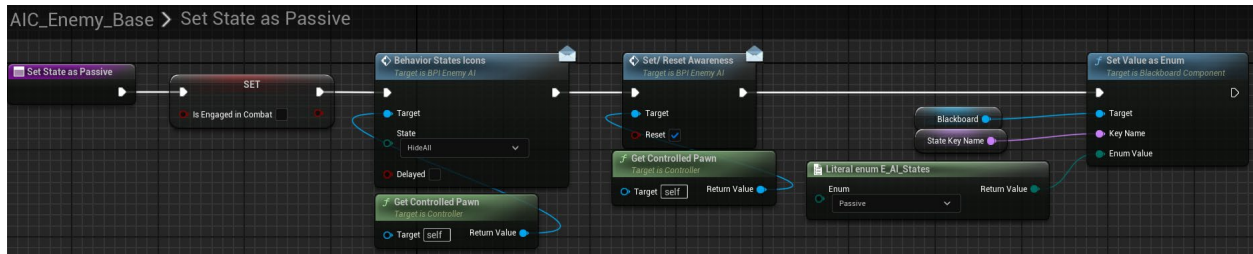
Εικόνα 75. BTD_HasPatrolRoute

BTD_IsWeaponEquipped (Blackboard Decorator): Ο Decorator χρησιμοποιείται ως έλεγχος για τον καθορισμό της εκτέλεσης του task BTT_EquipWeapon και BTT_UnequipWeapon ελέγχοντας την κατάσταση οπλισμού των Agent τύπου Pistol και Rifle. Αυτό επιτυγχάνεται με το node CastTo BP_Energy_Base για την ανάγνωση της μεταβλητής IsWeaponEquipped?, της οποίας η τιμή καθορίζεται από τα functions EquipWeapon και Unequip Weapon, επιστρέφοντας το αποτέλεσμα στον Decorator. Για τον έλεγχο του BTT_EquipWeapon η συνθήκη αντιστρέφεται μέσω της επιλογής Inverse Condition που παρέχεται σε κάθε Decorator, καθώς προϋποθέτει η μεταβλητή IsWeaponEquipped? να έχει την τιμή False. Με αυτόν τον τρόπο ένας Agent θα οπλίσει ή αφοπλίσει μόνο όταν είναι αφοπλισμένος ή οπλισμένος αντίστοιχα.



Εικόνα 76. BTD_IsWeaponEquipped

Set State As Passive (Function): Το function είναι υπεύθυνο για την ενεργοποίηση του Passive State. Αναπτύχθηκε εντός του AI Controller και χρησιμοποιείται για να μεταβάλει το περιεχόμενο της μεταβλητής State (Enum) του Blackboard σε Passive. Η μεταβολή της μεταβλητής θα προκαλέσει διακοπή των τρεχουσών ενεργειών και επαναξιολόγηση του Behavior Tree στο επίπεδο του State Selector, ενεργοποιώντας τις ενέργειες που περιλαμβάνονται στο Passive State.



Εικόνα 77. Set State As Passive

Αυτό επιτυγχάνεται μέσω του node Set Value As Enum, το οποίο δέχεται ως είσοδο το Blackboard reference, το όνομα του key προς αλλαγή και το περιεχόμενο στο οποίο θα οριστεί το key όπως φαίνεται στην εικόνα 77. Επιπλέον, το function θέτει τη μεταβλητή IsEngagedInCombat (Boolean) σε False, η οποία χρησιμοποιείται σε διάφορα σημεία του κώδικα για να ελεγχθεί εάν ο Agent βρίσκεται σε κατάσταση μάχης, δηλαδή στο State Attacking. Η ενέργεια τοποθετείται ώστε να διασφαλιστεί πως η μεταβλητή διατηρεί τη σωστή τιμή για την ορθή λειτουργία των ελέγχων στους οποίους χρησιμοποιείται, καθώς ο Agent μπορεί να μεταβεί στο Passive State, ενώ βρίσκεται στο Attacking State. Ακόμη, πριν την εκτέλεση της αλλαγής του blackboard key, καλούνται τα Blueprint Interface functions Behavior State Icons και Set/Reset Awareness. Το πρώτο είναι υπεύθυνο για την απόκρυψη των εικονιδίων που εμφανίζονται πάνω από έναν Agent, όταν βρίσκεται σε καταστάσεις Seeking ή Investigate, τα οποία λειτουργούν ως ένδειξη του State. Καθώς ένας Agent μπορεί να μεταβεί από αυτές τις δυο καταστάσεις προς το Passive State, η ενέργεια σηματοδοτεί την είσοδο του Agent σε αυτή την κατάσταση μηδενικής επιθετικότητας. Το δεύτερο αφορά στον ορισμό της μεταβλητής Boolean IsAwareofPlayer (BehaviorVoiceSFX) σε False, η οποία βρίσκεται εντός του Parent Blueprint BP_Enemy_Base και χρησιμοποιείται σε ελέγχους των λειτουργιών που αφορούν στα ηχητικά εφέ που αναπαράγονται από τους Agents.

Το function καλείται κατά την αρχικοποίηση του Agent, δηλαδή όταν βρίσκεται τοποθετημένος στο level και το παιχνίδι ξεκινά ή όταν γίνεται spawn μέσω κάποιας άλλης μεθόδου (π.χ. ένα Spawn Actor υπεύθυνο για τη δημιουργία Agents βάσει συνθηκών). Επιπλέον, καλείται κατά την ολοκλήρωση των ενεργειών των States Detecting, Investigate, Seeking και Attacking. Συγκεκριμένα, η μετάβαση πραγματοποιείται όταν ο Agent διερεύνησε μια ηχητική διατάραξη χωρίς να αντιληφθεί κάποιον εχθρικό Actor (Investigate), ένας εχθρικός Actor βρέθηκε στο οπτικό του πεδίο για μικρό χρονικό διάστημα, χωρίς να τον αντιληφθεί πλήρως (Detecting), καθώς και όταν αναζήτησε χωρίς επιτυχία έναν Actor που χάθηκε από το οπτικό του πεδίο (Seeking). Τέλος, ο Agent μεταβαίνει από το Attacking στο Passive State όταν ο παίκτης πεθάνει από οποιαδήποτε ενέργεια που μηδενίσει του πόντους του, συμπεριλαμβανομένων των επιθέσεων του ίδιου ή κάποιου άλλου Agent ή από απόδοση Damage, λόγω Fall Damage.

8.3. Investigate State

Σε αυτό το State ο Agent έχει αντιληφθεί μια ηχητική διατάραξη στον χώρο, χωρίς να είναι ενήμερος για την παρουσία του παίκτη. Αυτό μπορεί να συμβεί όταν ο παίκτης χρησιμοποιήσει το αντικείμενο Bottle, δηλαδή όταν πετάξει ένα μπουκάλι και αυτό προκαλέσει ήχο εντός του ηχητικού πεδίου που μπορεί να αντιληφθεί ο Agent. Ένας ακόμη λόγος για την ενεργοποίηση του State αποτελεί ο ήχος που προκαλεί ένα όπλο τη στιγμή που πυροδοτηθεί, ανεξάρτητα αν ο ήχος προκλήθηκε τον παίκτη ή έναν άλλο Agent. Οι πιθανές διαταράξεις αναγνωρίζονται και καταχωρίζονται χρησιμοποιώντας το σύστημα Perception του Unreal Engine, μέσω του AI Perception Component και των μεταβλητών που ορίστηκαν εντός αυτού, όπως προαναφέρθηκε.

Συγκεκριμένα, ένας Actor προκαλεί μια ηχητική διατάραξη με την κλήση του node Report Noise Event, παρέχοντας τις απαραίτητες μεταβλητές και πυροδοτώντας το function Handle Sensed Sound μέσω του event OnPerceptionUpdated (AI Perception) του AI Perception Component. Σε αυτό το Project, οι διαταράξεις είναι ανεξάρτητες από τον τύπο του Agent, δηλαδή προκαλούν την ενεργοποίηση του state για κάθε τύπου εχθρό. Στην περίπτωση που πληρούνται οι προϋποθέσεις, ο Agent θα κινηθεί προς το σημείο ώστε να διερευνήσει τη διατάραξη. Όσο ο Agent βρίσκεται σε αυτό το state, μπορεί να δεχθεί επίθεση Takedown από τον παίκτη, καθώς δεν έχει αντιληφθεί ρητά την παρουσία του. Ακόμη, το state αποτελεί κατάσταση επιθετικότητας χαμηλού επιπέδου, που σημαίνει πως τοποθετείται στο Behavior Tree αριστερά από το Passive State και δεξιότερα από κάθε άλλη, διακόπτοντας οποιαδήποτε ενέργεια βρίσκεται δεξιά από αυτή, χωρίς να επηρεάσει κάποια σημαντικότερη. Για τη διαφοροποίηση του state από αυτό του Seeking, οι Agents τύπου Pistol και Rifle δεν οπλίζουν για την εκτέλεσή της. Ωστόσο, ένας Agent που εισέρχεται στο state, διατηρεί την προηγούμενη κατάσταση της όπλισης, λόγω της απουσίας ενεργειών όπλισης και αφόπλισης. Ως αποτέλεσμα, οι ενέργειες του state παραμένουν ίδιες για κάθε τύπου Agent και τα Behavior Trees BT_Ranged και BT_Android χρησιμοποιούν το ίδιο Subtree για την εκτέλεσή του.

Οι αρχικές ενέργειες του State περιλαμβάνουν την αποθήκευση του σημείου που βρίσκεται ο Agent, προκειμένου να επιστρέψει σε αυτό κατά την ολοκλήρωση της διαδικασίας. Παράλληλα, μέσω ενεργειών του AI Controller, ένα χαρακτηριστικό εικονίδιο προστίθεται πάνω από τον Agent και αναπαράγεται ένας ηχητικός διάλογος, σηματοδοτώντας το τρέχον state. Στη συνέχεια, η εκκαθάριση του Focus διασφαλίζει πως η κατεύθυνση του δεν θα παραμείνει εστιασμένη σε ένα σημείο. Έπειτα, ο Agent ακινητοποιείται στο σημείο που βρίσκεται και ένα Animation που αποδίδει την αναγνώριση της ηχητικής διατάραξης αναπαράγεται, προτού ο Agent προχωρήσει σε κάποια άλλη ενέργεια. Με αυτόν τον τρόπο ένας Agent μπορεί να φανεί πως αντιδρά, σηματοδοτώντας πως αντιλήφθηκε την ηχητική διατάραξη. Η ενέργεια επαναλαμβάνεται σε κάθε έναρξη του state, προσφέροντας ένα εύκολα αναγνωρίσιμο και προβλέψιμο οπτικό feedback προς τον παίκτη με ρεαλιστικό τρόπο, ώστε να αναγνωρίσει την αλλαγή του state και να σχεδιάσει την επόμενη του κίνηση. Μετά την ολοκλήρωση του πρώτου Animation, ο Agent θα στραφεί και θα κινηθεί προς το σημείο στο οποίο συνέβη η διατάραξη. Όταν φτάσει στο σημείο, ο Agent θα ακινητοποιηθεί και ένα ακόμη Animation σε συνδυασμό με ηχητικό διάλογο θα αναπαραχθούν. Οι

ενέργειες αποδίδουν οπτικά και ηχητικά το γεγονός πως ο Agent αναζητά γύρω του για την ενδεχόμενη ύπαρξη εχθρικής παρουσίας.

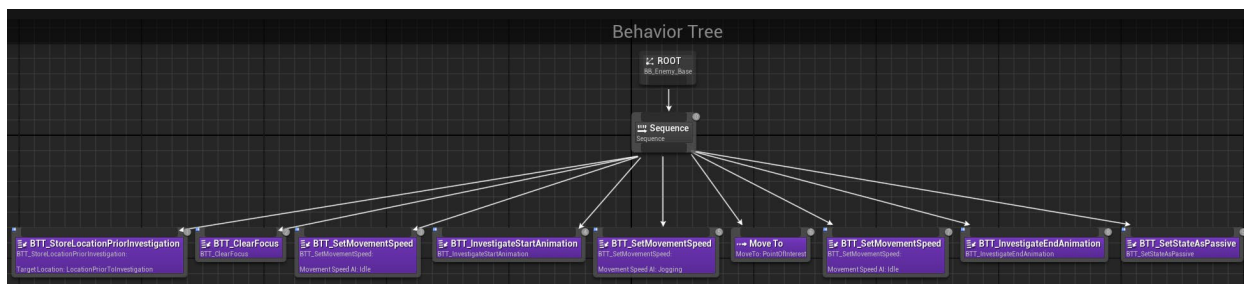
Στην περίπτωση που οι ενέργειες ολοκληρωθούν χωρίς ο Agent να αντιληφθεί τον παίκτη, το Behavior Tree επανέρχεται στην κατάσταση Passive State. Με αυτόν τον τρόπο ο Agent συνεχίζει την περιπολία του ή επιστρέφει στο αρχικό σημείο που βρισκόταν πριν τη διατάραξη. Σε αντίθετη περίπτωση, ο Agent μεταβαίνει στην κατάσταση Detecting, ξεκινώντας τη διαδικασία αναγνώρισης του παίκτη. Τέλος, οι ρυθμίσεις του Blackboard Based Condition διασφαλίζουν πως μία αλλαγή που ενεργοποιεί ένα state μεγαλύτερης προτεραιότητας, θα προκαλέσει διακοπή του state, αλλά και κάθε state χαμηλής προτεραιότητας, ώστε να πραγματοποιηθεί επαναξιολόγηση του δέντρου στο επίπεδο των states.

Οι πιθανές μεταβάσεις προς αυτό το state μπορεί να είναι από το Passive State, όταν ο Agent αντιληφθεί μια ηχητική διατάραξη και από το Seeking State, όταν η διατάραξη γίνει αντιληπτή κατά την αναζήτηση του παίκτη. Επιπλέον, το state μπορεί να οδηγήσει έναν Agent στην κατάσταση Detecting, όταν ο παίκτης εισέλθει στο οπτικό του πεδίο ή στην κατάσταση Dead, εάν ο παίκτης εκτελέσει την ενέργεια Takedown.

Σε αυτή την υποενότητα θα αναλυθεί η υλοποίηση του συγκεκριμένου state και των assets που αναπτύχθηκαν για τη λειτουργία του. Αρχικά θα αναλυθεί η λειτουργία του state και των tasks που καλούνται στο Behavior Tree και στη συνέχεια θα περιγραφούν τα functions της εκτέλεσης των Animation αντίδρασης. Στη συνέχεια, θα αναφερθούν τα functions που πυροδοτούν τις ενέργειες του Agent σε μια ηχητική διατάραξη. Τέλος, θα αναλυθούν τα Blueprint που αφορούν στη χρήση του αντικειμένου Bottle, συμπεριλαμβανομένων των Blueprint που περιέχουν τις λειτουργίες του, καθώς και των functions που υλοποιήθηκαν στο Blueprint του παίκτη και καθιστούν δυνατή τη χρήση του.

8.3.1. Investigate Behavior Tree Overview

Το δέντρο ενεργοποιείται κατά την αξιολόγηση του State Selector μέσω Blackboard Based Conditions και πυροδοτεί το BT_Subtree_Investigate των ενεργειών του State. Το Subtree περιλαμβάνει ένα sequence, με κάθε ενέργεια να εκτελείται από τα αριστερά προς τα δεξιά.

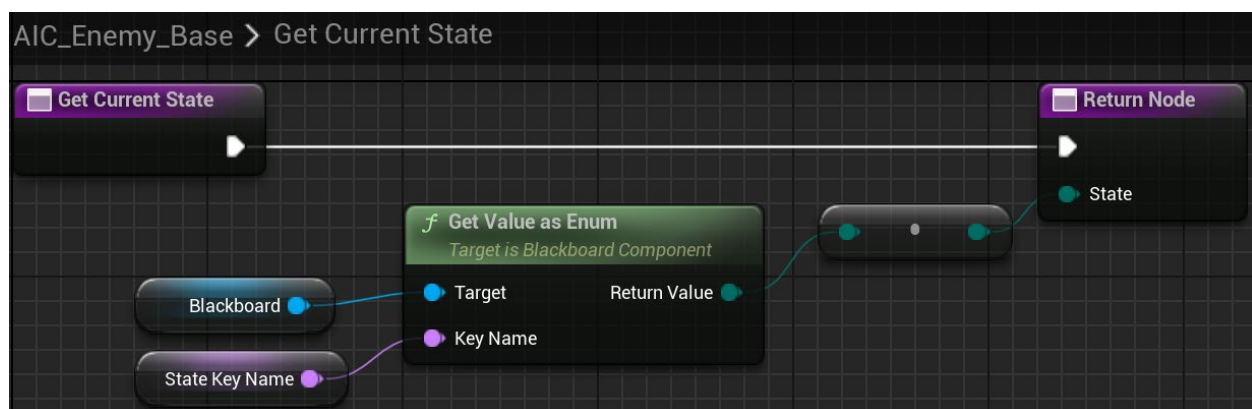


Εικόνα 78. Investigate Behavior Tree Overview

Αρχικά, αποθηκεύεται η τρέχουσα τοποθεσία του Agent μέσω του task BTT_StoreLocationPriorInvestigation, ώστε να χρησιμοποιηθεί ως η τοποθεσία στη οποία θα γυρίσει ο Agent μετά την ολοκλήρωση του state, αν δε διαθέτει κάποια διαδρομή περιπολίας. Στη συνέχεια, τα tasks BTT_ClearFocus και BTT_SetMovementSpeed χρησιμοποιούνται για την αποδέσμευση του Agent από ένα ενδεχόμενο σημείο εστίασης και την ακινητοποίηση του, καθώς και την αναπαραγωγή του πρώτου Animation. Το task BTT_InvestigateStartAnimation εκτελεί το πρώτο Animation που αφορά στην αντίδραση του Agent λόγω της διατάραξης. Η ολοκλήρωση του Animation ακολουθείται από τα task BTT_SetMovementSpeed με είσοδο Jogging, MoveTo με είσοδο PointOfInterest. Με αυτόν τον τρόπο η ταχύτητα του Agent μεταβάλλεται σε Jogging, ώστε να κινηθεί με γρήγορους ρυθμούς στον χώρο και ο Agent μεταβαίνει στο σημείο διατάραξης. Η ολοκλήρωση του MoveTo σηματοδοτεί την άφιξη του Agent στο σημείο και ακολουθείται από το task SetMovementSpeed με είσοδο Idle, ώστε να ακινητοποιηθεί ξανά. Έπειτα, το BTT_InvestigateEndAnimation αναπαράγει το Animation της αναζήτησης εχθρικών Actors, επιτρέποντας στον Agent να κοιτάζει την περιοχή γύρω από το σημείο. Τέλος, το task SetStateAsPassive θα αλλάξει το State (Enum) προκαλώντας επαναξιολόγηση του δέντρου, ενεργοποιώντας το PassiveState.

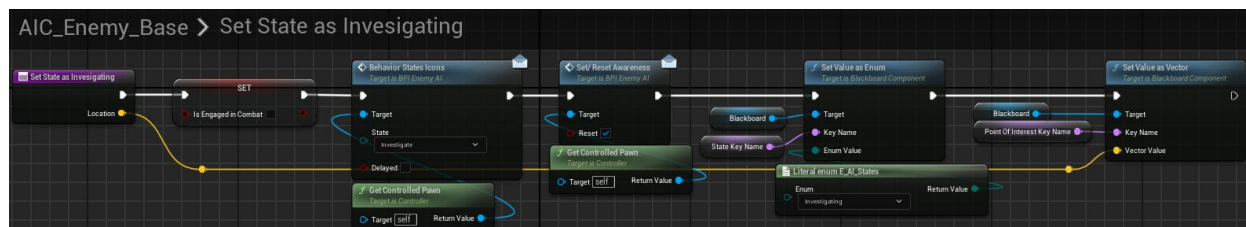
8.3.2. Investigate State Assets

GetCurrentState (Function): Το function είναι υπεύθυνο για την ανάκτηση του τρέχοντος state που εκτελείται από το Behavior Tree και υλοποιείται στο Blueprint του AI Controller. Συγκεκριμένα, διαβάζει το περιεχόμενο του Blackboard Key State (Enum), μέσω του node GetValueAsEnum, με εισόδους το Blackboard Component και το όνομα του key προς ανάγνωση και επιστρέφει το αποτέλεσμα ως enum στο τέλος της κλήσης. Χρησιμοποιείται σε κάθε σημείο του AI Controller που απαιτείται έλεγχος του state το οποίο εκτελείται στο Behavior Tree. Για παράδειγμα, χρησιμοποιείται κατά την αλλαγή του περιεχομένου του Blackboard Key State, μέσω του node Switch, ώστε να μεταβεί το Behavior Tree σε διαφορετική κατάσταση, επιτρέποντας τον ορισμό μια καινούργιας τιμής σύμφωνα με την προϋπάρχουσα. Τέλος, μπορεί να κληθεί από το Blueprint ενός Agent, κατά την εκτέλεση μιας ενέργειας που απαιτεί τον Agent να βρίσκεται εντός ή εκτός ενός συγκεκριμένου state.



Εικόνα 79. GetCurrentState

SetStateAsInvestigating (Function): Το function είναι υπεύθυνο για την ενεργοποίηση του Investigate State και καλείται όταν ένα εχθρικό ηχητικό ερέθισμα καταγραφεί από το Hearing Sense ενός Agent και οι συνθήκες αλλαγής του state σε Investigate ικανοποιούνται. Ακόμη, χρησιμοποιείται για την επαναφορά του Agent στην κατάσταση Investigate, όταν μια διερεύνηση διακόπηκε στιγμιαία. Αναπτύχθηκε εντός του AI Controller και χρησιμοποιείται για να μεταβάλει το περιεχόμενο της μεταβλητής State (Enum) του Blackboard σε Investigate.



Εικόνα 80. SetStateAsInvestigating

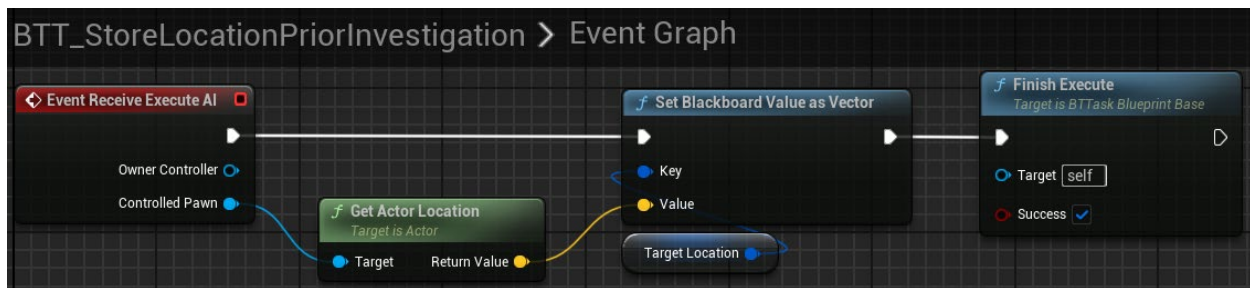
Η μεταβολή του State επιτυγχάνεται μέσω του node Set Value As Enum, το οποίο δέχεται ως είσοδο το Blackboard Reference, το όνομα του key προς αλλαγή και το περιεχόμενο στο οποίο θα οριστεί το key όπως φαίνεται στην εικόνα 80. Επιπλέον, το function θέτει τη μεταβλητή IsEngagedInCombat (Boolean) σε False, καθώς ο Agent μεταβαίνει σε μη επιθετική κατάσταση. Σε αυτό το σημείο, το Blueprint Interface function Behavior State Icons καλείται για την εμφάνιση ενός χαρακτηριστικού εικονιδίου πάνω από το κεφάλι του Agent, σηματοδοτώντας την είσοδό του σε μια κατάσταση χαμηλής επιθετικότητας κατά την οποία διερευνά μία ηχητική διατάραξη. Το εικονίδιο παραμένει καθ' όλη τη διάρκεια του Agent στο Investigate, όπως φαίνεται στην εικόνα 81. Στη συνέχεια, το Blueprint Interface function Set/Reset Awareness επαναφέρει τη μεταβλητή IsAwareofPlayer (BehaviorVoiceSFX) σε False, όπως και στο function SetStateAsPassive.

Τέλος, το function ολοκληρώνεται με το node SetValueAsVector, το οποίο εκχωρεί την τοποθεσία του ηχητικού ερεθίσματος στο περιεχόμενο του Blackboard Key PointOfInterest. Η τοποθεσία εκχωρείται σε κάθε κλήση του function SetStateAsInvestigating ως είσοδος, αποτελώντας κρίσιμο σημείο της υλοποίησης. Με αυτόν τον τρόπο αποθηκεύεται το σημείο στο Blackboard Key, ώστε να χρησιμοποιηθεί ως είσοδος από το task MoveTo και ο Agent να κινηθεί προς τη σωστή κατεύθυνση.



Εικόνα 81. Investigate State Icon

BTT_StoreLocationPriorInvestigation (Custom Task): Το task είναι υπεύθυνο για την αποθήκευση της τρέχουσας τοποθεσίας του Agent στο Blackboard Key LocationPriorInvestigation (Vector). Χρησιμοποιείται στην έναρξη κάθε task που προκαλεί τη μεταβολή της θέσης του Agent στον χώρο, παρέχοντας την τοποθεσία στην οποία θα επιστρέψει μετά την ολοκλήρωσή του State που τον μεταβάλλει. Καλείται κατά την αρχικοποίηση του Agent και στην αρχική των tasks Investigate και Seek. Η χρήση της επιτρέπει σε έναν Agent που δε διαθέτει ένα PatrolRoute, να επιστρέψει στην αρχική του θέση μετά από μια διερεύνηση ηχητικής διατάραξης ή αναζήτησης του παίκτη. Επίσης, η εφαρμογή της στο Passive State επαναφέρει τον Agent στη θέση του, όταν εντοπίσει τον παίκτη και εκτελέσει το Animation αντίδρασης που περιλαμβάνεται κατά το Detecting State. Όπως αναφέρεται σε σχετικό κεφάλαιο, το animation αντίδρασης του Detecting State διαθέτει root motion, κατά το οποίο η θέση του Agent αλλάζει ελάχιστα, σύμφωνα με την κίνηση του Animation. Με αυτό τον τρόπο, ο Agent επιστρέφει στο αρχικό σημείο μετά την αναπαραγωγή του animation.



Εικόνα 82. BTT_StoreLocationPriorInvestigation

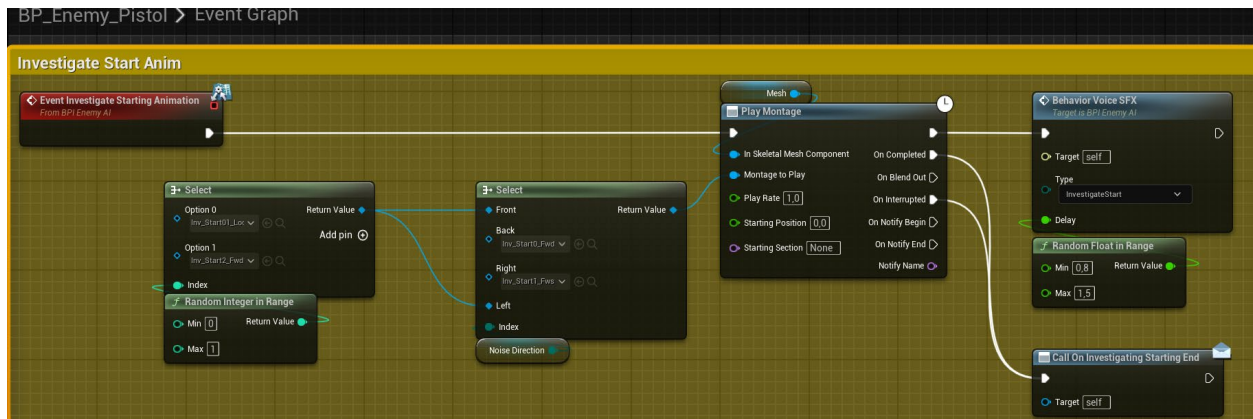
Η ανάκτηση της τοποθεσίας πραγματοποιείται μέσω του node GetActorLocation, από το reference του Controlled Pawn και την αποθήκευση του Vector στο Blackboard key μέσω του node SetValueAsVector. Το task περιέχει τη μεταβλητή Tagret Location (Blackboard Key Selector), με την επιλογή instance editable ενεργοποιημένη, ώστε το Blackboard key στο οποίο θα αποθηκευτεί το Vector να εκχωρείται σε κάθε κλήση του task.

BTT_InvestigateStartAnimation και BTT_InvestigateEndAnimation (Custom Task): Τα tasks είναι υπεύθυνα για την κλήση των function που εκτελούν την αναπαραγωγή των Animation, που λαμβάνουν χώρα κατά την διάρκεια του State. Το πρώτο αφορά στο animation της αντίδρασης του Agent στην ηχητική διατάραξη και το δεύτερο στην ανίχνευση της περιοχής γύρω από το σημείο της διατάραξης. Τα tasks ακολουθούν την υλοποίηση των tasks BTT_EquipWeapon και BTT_UnequipWeapon, χρησιμοποιώντας το Node CastTo BP_Energy_Base, καλώντας το Blueprint Interface function InvestigateStartingAnimation και InvestigateEndingAnimation αντίστοιχα και δεσμεύοντας την ολοκλήρωση των task στην κλήση των Event Dispatcher OnInvestigateStartingEnd και OnInvestigateEndingEnd. Με αυτόν τον τρόπο, το task αποκτά πρόσβαση στο Parent Blueprint, το function αναπαράγει το animation από τα Child Blueprints των Agents, διασφαλίζοντας πως το Behavior Tree θα προχωρήσει στην επόμενη ενέργεια μετά την ολοκλήρωση της αναπαραγωγής του Animation.

Investigate Starting Animation (Function): Το function είναι υπεύθυνο για την αναπαραγωγή του Animation της αντίδρασης ενός Agent όταν αντιλαμβάνεται μια ηχητική διατάραξη, υλοποιείται σε κάθε Blueprint διαφορετικού τύπου Agent (BP_Energy_Rifle, BP_Energy_Rifle και BP_Energy_Android) και ορίζεται στο Blueprint Interface BPI_Energy_AI. Ο λόγος της υλοποίησης του function σε κάθε ένα από τα Child Blueprint και όχι στο Parent, έγκειται στη χρήση διαφορετικών animations ανά τύπο και στο γεγονός πως το function χρησιμοποιείται για την εκτέλεση επιπλέον ενεργειών. Συγκεκριμένα, το function λειτουργεί και ως σημείο κλήσης επιπλέον functions που αφορούν στην αναπαραγωγή χαρακτηριστικών ηχητικών διαλόγων ανά τον τύπο του Agent.

Σε αυτό το Project, χρησιμοποιήθηκε ένα σύνολο από Animations αντίδρασης, τα οποία χαρακτηρίζονται από τη θέση του σημείου στο οποίο έλαβε χώρα η διατάραξη, σύμφωνα με τη θέση του Agent. Για παράδειγμα, αν η διατάραξη έγινε πίσω από τον Agent, αυτός κάνει δυο βήματα εμπρός με ξαφνιασμένο τρόπο και γυρίζει το σώμα του προς την κατεύθυνση της διατάραξης. Με αυτόν τον τρόπο, ο Agent

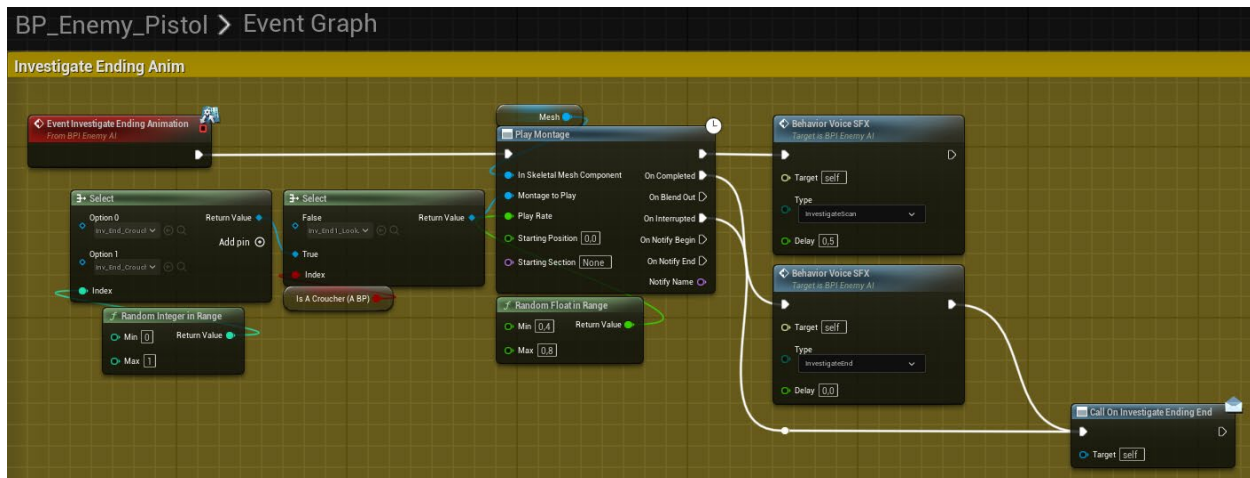
φαίνεται να αντιδρά με τέσσερις διαφορετικούς τρόπους για κάθε κατεύθυνση (εμπρός, πίσω, αριστερά, δεξιά). Η κατεύθυνση ανακτάται από τη μεταβλητή Noise Direction και χρησιμοποιείται ως είσοδος σε ένα Select Node, ώστε να καθοριστεί το κατάλληλο Animation.



Εικόνα 83. Investigate Starting Animation

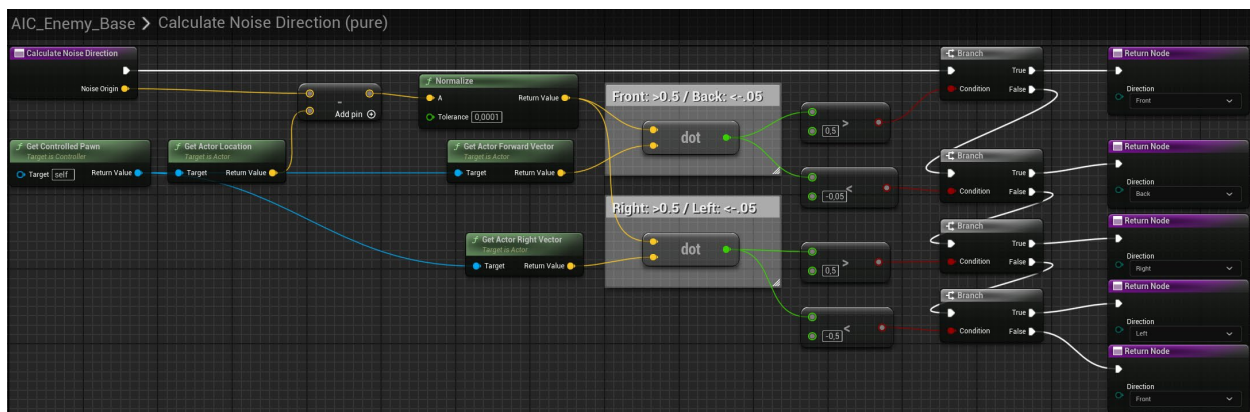
Η μεταβλητή Noise Direction παίρνει την τιμή της εξόδου του pure function CalculateNoiseDirection κατά τη μετάβαση του Agent στο Investigating State στο function HandleSensedSound. Το Animation αναπαράγεται με το node Play Montage, από του οποίου τις εξόδους OnCompleted και OnInterrupted καλείται το Event Dispatcher OnInvestigateStartingEnd για την ολοκλήρωση του BTT_InvestigateStartAnimation. Από αισθητικής άποψης, τα διαθέσιμα animations του Project αποδίδουν αντιδράσεις που παραπέμπουν σε ανθρώπινες συμπεριφορές. Ως εκ τούτου η αναπαραγωγή animation βάσει κατεύθυνσης υλοποιήθηκε αποκλειστικά για τους Agents τύπου Pistol και Rifle, ενώ οι Android Agents χρησιμοποιούν ένα ενιαίο animation, ανεξαρτήτως της κατεύθυνσης πρόσκρουσης.

Investigate Ending Animation (Function): Το function είναι υπεύθυνο για την αναπαραγωγή του Animation κατά το οποίο ο Agent ανιχνεύει την περιοχή γύρω από το σημείο διατάραξης. Συγκεκριμένα, ο Agent θα εκτελέσει κινήσεις που αναπαριστούν τη σάρωση του πεδίου κατά την άφιξή του στο σημείο. Ομοίως με το function Investigate Starting Animation, ορίζεται στο Blueprint Interface BPI_Enemy_AI, υλοποιείται σε κάθε Child Blueprint και η ολοκλήρωση της αναπαραγωγής του animation ακολουθείται από το αντίστοιχο Event Dispatcher OnInvestigateEndingEnd, για την ολοκλήρωση του task.



Εικόνα 84. Investigate Ending Animation

CalculateNoiseDirection (Pure Function): Το function είναι υπεύθυνο για τον καθορισμό της κατεύθυνσης της ηχητικής διατάραξης, λαμβάνοντας υπόψη τη θέση του Agent και την κατεύθυνση στην οποία αυτός κοιτάζει μπροστά. Υλοποιήθηκε στο Blueprint του AI Controller και σε κάθε κλήση της δέχεται ως είσοδο τη μεταβλητή NoiseOrigin (Vector) που αντιπροσωπεύει την τοποθεσία της ηχητικής διατάραξης και επιστρέφει τη μεταβλητή Direction (Enum) με πιθανές τιμές Front, Back, Left και Right. Το αποτέλεσμα του function χρησιμοποιείται για την επιλογή του κατάλληλου animation αντίδρασης ενός Agent, όταν αντιληφθεί τη διατάραξη, όπως περιγράφεται στην ανάλυση του function Investigate Starting Animation. Συγκεκριμένα, το αποτέλεσμα θα αναθέσει το περιεχόμενο του στη μεταβλητή NoiseDirection (Enum), που βρίσκεται στο Parent Blueprint των Agent. Η μεταβίβαση της τιμής γίνεται μέσω του Blueprint Interface Function GetNoiseDirection της BPI_Enemy_AI και είναι απαραίτητο στοιχείο της υλοποίησης, καθώς η εκτέλεση του Animation βρίσκεται στο Blueprint των Agents.



Εικόνα 85. CalculateNoiseDirection

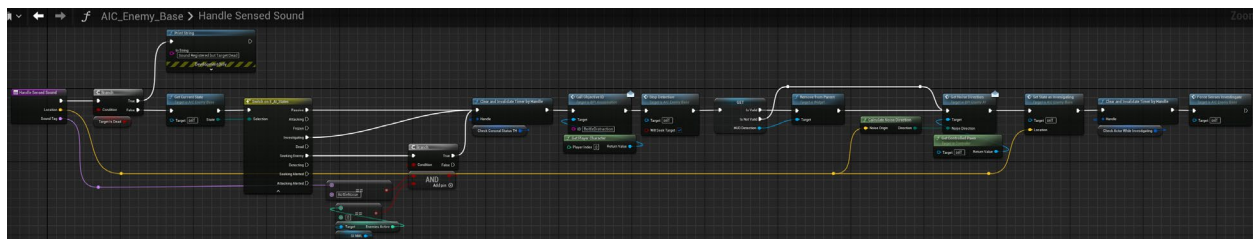
Όσον αφορά στον καθορισμό του σημείου, το function χρησιμοποιεί μια σειρά από Branch nodes και Dot Product για να αξιολογήσει τη γωνία μεταξύ της κατεύθυνσης του ήχου και της κατεύθυνσης του Agent. Το Dot Product (ή εσωτερικό γινόμενο) είναι μια μαθηματική λειτουργία που βοηθά στον υπολογισμό της διανυσματικής απόστασης και χρησιμοποιείται για τον υπολογισμό της γωνίας μεταξύ της κατεύθυνσης του Agent και του ήχου που προέρχεται από το NoiseOrigin. Σε αυτό το function, χρησιμοποιούνται δύο dot products, με το πρώτο να ελέγχει εάν ο ήχος βρίσκεται μπροστά ή πίσω και το δεύτερο αν βρίσκεται αριστερά ή δεξιά του Agent. Το function επιλέχθηκε να είναι pure, καθώς εκτελεί υπολογισμούς και επιστρέφει το αποτέλεσμα, χωρίς να τροποποιήσει μεταβλητές ή κάποια κατάσταση του κόσμου. Ακόμη, δεν εξαρτάται από εξωτερικές μεταβλητές ή καταστάσεις, παρά μόνο από τα δεδομένα που παρέχονται στην είσοδο. Με τη χρήση του function ως Pure, μπορεί να καλείται οποιαδήποτε στιγμή χωρίς να προκαλεί προβλήματα με την κατάσταση του συστήματος, να εκτελέσει τους υπολογισμούς αποδοτικότερα και να επαναχρησιμοποιηθεί σε πολλαπλά σημεία του Blueprint. Τέλος, ως pure function μπορεί να εφαρμοστεί στον κώδικα χωρίς να χρειαστεί να τοποθετηθεί σε κάποιο execution pin, διευκολύνοντας στη συντήρηση και στην οργάνωση του κώδικα.

DotProduct (Front/Back): Οι είσοδοι του πρώτου dot product περιλαμβάνουν το normalized vector της διαφοράς θέσης μεταξύ του NoiseOrigin και της θέσης του Controlled Pawn, καθώς και του διανύσματος που δείχνει την κατεύθυνση στην οποία κοιτάει ο χαρακτήρας στον κόσμο του παιχνιδιού. Η διαφορά θέσης ανάμεσα στο NoiseOrigin και το Controlled Pawn δημιουργεί το διάνυσμα που δείχνει από τον χαρακτήρα προς την πηγή του ήχου. Ο σκοπός του normalize είναι να μετατρέψει αυτό το διάνυσμα σε διάνυσμα μήκους 1 (μονάδας), ώστε να μπορεί να συγκριθεί με άλλες κατευθύνσεις χωρίς να επηρεάζεται από το μέγεθος του διανύσματος. Με αυτόν τον τρόπο διατηρείται μόνο η κατεύθυνση του ήχου, αγνοώντας την απόσταση. Αν το αποτέλεσμα του dot product είναι κοντά στο 1, σημαίνει ότι ο ήχος βρίσκεται μπροστά από τον χαρακτήρα, καθώς οι δύο κατευθύνσεις (η κατεύθυνση του ήχου και η κατεύθυνση του χαρακτήρα) είναι σχεδόν παράλληλες. Αν το αποτέλεσμα είναι κοντά στο 0, οι δύο κατευθύνσεις είναι κάθετες και δε σχετίζονται καθόλου. Αν το αποτέλεσμα είναι κοντά στο -1, ο ήχος είναι πίσω από τον χαρακτήρα, καθώς οι δύο κατευθύνσεις είναι αντίθετες. Το function χρησιμοποιεί το αποτέλεσμα του dot product σε ένα branch που ελέγχει αν η τιμή είναι μεγαλύτερη από 0.5, το οποίο σηματοδοτεί ότι οι δύο κατευθύνσεις είναι αρκετά κοντά για να θεωρηθεί ότι ο ήχος βρίσκεται μπροστά από τον χαρακτήρα, επιστρέφοντας την τιμή Front. Αν το αποτέλεσμα είναι μικρότερο από 0.5, δηλαδή ο ήχος δε βρίσκεται μπροστά, το function ελέγχει αν το dot product είναι μικρότερο από -0.05. Το -0.05 είναι μια κατάλληλη τιμή, που δεν απαιτεί ακριβώς αντίθετη κατεύθυνση προκειμένου ο ήχος να θεωρηθεί πως βρίσκεται πίσω από τον χαρακτήρα. Αν αυτό ισχύει, επιστρέφεται το Enum Back, υποδεικνύοντας ότι ο ήχος βρίσκεται πίσω από τον Agent.

DotProduct (Right/Left): Αν δεν ισχύει καμία από τις παραπάνω περιπτώσεις, σημαίνει πως η θέση του σημείου δεν μπορεί να χαρακτηριστεί μπροστά ή πίσω. Τότε το function προχωρά σε περαιτέρω αξιολόγηση, χρησιμοποιώντας ένα δεύτερο dot product για να προσδιορίσει αν ο ήχος βρίσκεται είτε αριστερά είτε δεξιά του Agent. Το αποτέλεσμα προκύπτει από το ίδιο Normalized διάνυσμα του πρώτου dot product και το διάνυσμα που δείχνει προς τη δεξιά κατεύθυνση του Agent. Αν το αποτέλεσμα είναι κοντά στο 1, ο ήχος βρίσκεται δεξιά, ενώ αν είναι κοντά στο -1, ο ήχος βρίσκεται αριστερά. Σε αυτή την

περίπτωση το branch ελέγχει εάν η τιμή είναι μεγαλύτερη από 0.5, ώστε να επιστραφεί το αποτέλεσμα Right. Αν η συνθήκη αποδειχθεί False, ο τελευταίος έλεγχος εξετάζει αν η τιμή του dot product είναι μικρότερη από -0.5, ώστε να επιστραφεί η τιμή Left. Αν και οι τιμές που ορίστηκαν μπορούν να χαρακτηρίσουν ένα οποιοδήποτε σημείο του χώρου, το τελευταίο branch (το οποίο θα εκτελεστεί αν όλες οι συνθήκες αποδειχθούν False), επιστρέφει την τιμή false στη μεταβλητή, ώστε να εξασφαλιστεί πως η κλήση της συνάρτησης θα επιστρέφει μια τιμή.

HandleSensedSound (Function): Το function είναι υπεύθυνο για την εκτέλεση της κατάλληλης ενέργειας ενός Agent, όταν λάβει ένα ηχητικό ερέθισμα μέσω του Hearing Sense που παρέχει το AI Perception Component του Unreal Engine. Περιέχει τη συλλογή των πιθανών ενεργειών που θα ακολουθήσει ένας Agent σύμφωνα με την κατάσταση και τις τιμές συγκεκριμένων μεταβλητών του παιχνιδιού. Δέχεται ως είσοδο το Location (Vector) που αντιπροσωπεύει το σημείο του ερεθίσματος στον χώρο και το SoundTag (Name) που αφορά στον χαρακτηρισμό που διαθέτει ένα ερέθισμα. Το function πυροδοτείται όταν κληθεί ένα ReportNoiseEvent node, πληρώντας τις προϋποθέσεις καταχώρισης Stimuli, όπως αυτές ορίζονται στο Perception Component. Παραδείγματα αποτελούν ο ήχος καταστροφής ενός μπουκαλιού κατά τη ρίψη του από τον παίκτη ή ο ήχος ενός πυροβολισμού κατά τη χρήση όπλου από τον παίκτη ή κάποιον Agent. Η κύρια λειτουργία του function είναι να αλλάξει το state ενός Agent σε ένα άλλο, με βασικό κριτήριο το state στο οποίο βρίσκεται ο Agent την παρούσα στιγμή.



Εικόνα 86. HandleSensedSound

Η αρχική ενέργεια του function είναι ο έλεγχος της μεταβλητής TargetIsDead, η οποία παίρνει την τιμή True όταν ο εχθρικός στόχος (π.χ. ο παίκτης) πεθάνει. Αν ο παίκτης δεν είναι ζωντανός, το function δε θα προχωρήσει σε αλλαγή της κατάστασης. Με αυτόν τον τρόπο διασφαλίζεται πως ο Agent δε θα μεταβεί στο Investigate State για να διερευνήσει ένα σημείο όταν ο παίκτης πεθάνει, καθώς το συγκεκριμένο Project αναπτύχθηκε ως παιχνίδι Single Player. Η απουσία του ελέγχου θα παρουσίαζε δυσλειτουργικές ή ανεπιθύμητες συμπεριφορές του Agent, σε καταστάσεις που δέχεται ένα ηχητικό ερέθισμα τη στιγμή που ο παίκτης δεν είναι ζωντανός. Ο λόγος έγκειται στο γεγονός πως σε αυτό το Project, η εξόντωση του παίκτη μεταβάλλει αυτομάτως όλους τους Agents σε Passive State, ώστε να γυρίσουν σε μη εχθρική κατάσταση πριν το Level επανεκκινηθεί λόγω αποτυχημένης προσπάθειας. Ένα βασικό παράδειγμα δυσλειτουργίας αποτελεί η εξόντωση του παίκτη κατά τη διάρκεια μιας μάχης με πολλαπλούς Agents οι οποίοι πυροβολούν κατά του παίκτη. Αν ο παίκτης πεθάνει και οι Agents μεταβούν στο Passive State, μία ριπή των Agents λίγο μετά τη στιγμή του θανάτου θα προκαλέσει τη διερεύνηση του σημείου της ριπής των συμμάχων τους.

Μετά τον έλεγχο για την κατάσταση του παίκτη, το function ανακτά το ενεργό State του Agent μέσω του `GetCurrentState`. Η μεταβλητή που επιστρέφει χρησιμοποιείται σε ένα Switch node, που επιτρέπει την ανάπτυξη λειτουργιών για κάθε πιθανή τιμή της μεταβλητής. Στις περιπτώσεις που το state είναι `Attacking`, `Frozen`, `Detecting` ή `Dead`, το function δεν εκτελεί καμία λειτουργία, καθώς ο Agent είτε είναι νεκρός, είτε εκτελεί μια ενέργεια υψηλότερης προτεραιότητας και το ερέθισμα πρέπει να αγνοηθεί. Αντιθέτως, στις περιπτώσεις που ο Agent βρίσκεται στα states `Passive`, `Investigating` ή `Seeking`, το function θα μεταβάλει τον Agent στο State `Investigate` ώστε να διερευνήσει το σημείο της διατάραξης. Αυτό επιτυγχάνεται μέσω του node `SetStateAsInvestigating`, με είσοδο την τοποθεσία του σημείου. Με αυτόν τον τρόπο, ένας Agent που βρίσκεται σε μη εχθρική κατάσταση (`Passive`) ή διερευνά ήδη μια ηχητική διατάραξη (`Investigating`) ή αναζητά τον παίκτη (`Seeking`), θα διερευνήσει το σημείο που μόλις αντιλήφθηκε. Ωστόσο, στην περίπτωση του `Seeking State`, η μετάβαση θα πραγματοποιηθεί μόνο εάν ο ήχος προήλθε από τη χρήση του αντικειμένου `Bottle` και ταυτόχρονα κανένας Agent του Level δεν έχει αντιληφθεί την παρουσία του παίκτη. Αυτό επιτυγχάνεται ελέγχοντας πως το `SoundTag` που χαρακτηρίζει τον ήχο έχει την τιμή `BottleNoise` και η μεταβλητή `EnemiesActive` έχει μηδενική τιμή. Το `soundtag` παρέχεται κατά την κλήση του node `ReportNoise` στο `Blueprint` του `Bottle` και η μεταβλητή `EnemiesActive` αυξομειώνεται κατά την είσοδο/έξοδο ενός Agent στην κατάσταση `Attacking`, η χρήση της οποίας περιγράφεται αναλυτικά στο υποκεφάλαιο 8.5.

Στο function ενσωματώθηκαν επίσης ενέργειες που αφορούν στο Animation αντίδρασης, στο Quest System, σε UI assets που σχετίζονται με το `Detecting State` κ.α. Συγκεκριμένα, χρησιμοποιείται το node `GetNoiseDirection` με είσοδο το αποτέλεσμα της Pure function `Calculate Noise Direction`, ώστε να οριστεί η κατεύθυνση του ήχου για την αναπαραγωγή του κατάλληλου Animation αντίδρασης, όπως περιγράφεται στην υλοποίηση του function `Investigate Starting Animation`. Ακόμη, το node `CallObjective ID` καλείται προκειμένου το Quest System να ενημερωθεί πως ο παίκτης χρησιμοποίησε το αντικείμενο `Bottle` προς έναν Agent με επιτυχία, ώστε να ενημερωθούν τυχόν ενεργά Quests. Επιπλέον, τα nodes `Stop Detection`, `Get HudDetection`, `Remove from Parent` και `CheckandInvalidateTimerbyHandle`, είναι υπεύθυνα για την ορθή λειτουργία των λειτουργιών που σχετίζονται με το `Detecting State`, όπως αναλύονται στο υποκεφάλαιο 8.6.

8.3.3. Throwable Actor

BP_Throwable_Base και BP_Bottle

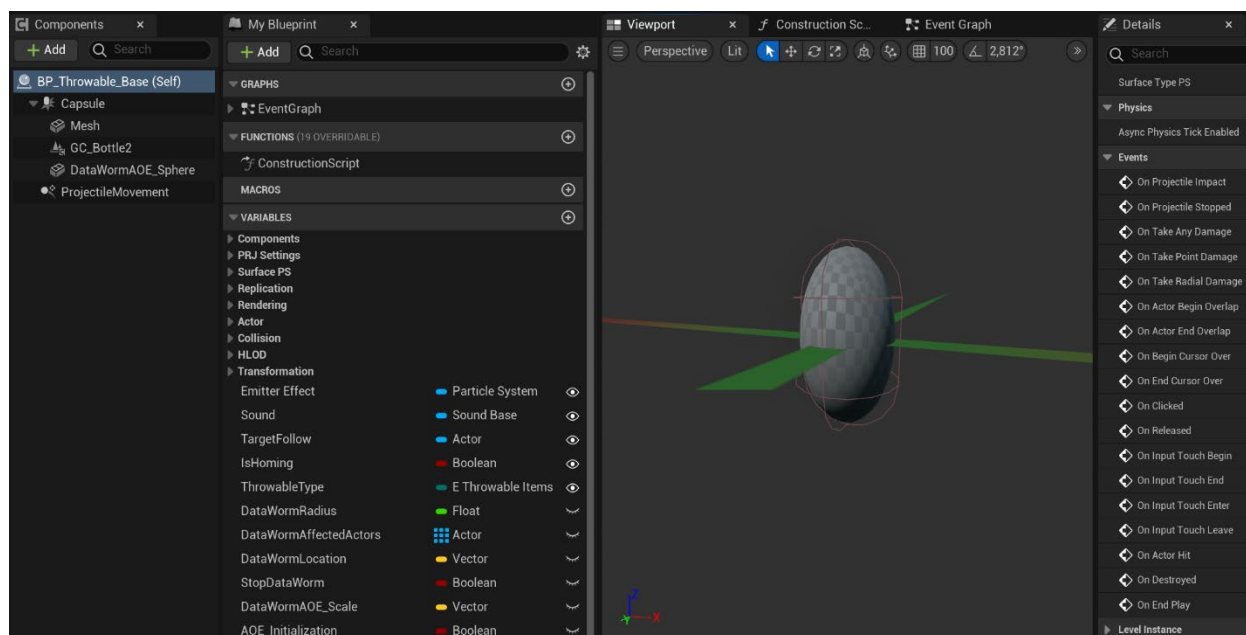
Στο συγκεκριμένο project, δημιουργήθηκαν δύο αντικείμενα ρίψης, το `Bottle` και το `DataWorm`, που μπορούν να χρησιμοποιηθούν από τον παίκτη για την αντιμετώπιση των Agents. Το πρώτο, είναι ένα μπουκάλι το οποίο ο παίκτης μπορεί πετάξει και να σπάσει κατά την επαφή του με μια επιφάνεια, δημιουργώντας ένα ηχητικό ερέθισμα και μεταβάλλοντας τον Agent στο `Investigate State`, ώστε να διερευνήσει το γεγονός. Χρησιμοποιώντας το αντικείμενο, ο παίκτης μπορεί να παρασύρει έναν Agent σε κάποιο σημείο, ώστε να ακολουθήσει μια διαδρομή του Level χωρίς την παρουσία εχθρού ή να τον τοποθετήσει σε κατάλληλη θέση, ώστε να εκτελέσει την ενέργεια `Takedown` και να τον εξοντώσει. Το δεύτερο, είναι μια ηλεκτρονική συσκευή που αδρανοποιεί στιγμιαία τους Agents τύπου `Android`,

καθιστώντας τους ευάλωτους για επιθέσεις Takedown και ανίκανους να εντοπίσουν την παρουσία του παίκτη. Το Bottle, μπορεί να μεταβάλλει κάθε τύπου Agent στο Investigate State όταν βρίσκεται σε state διαφορετικό από Attacking ή Dead, ενώ το DataWorm μπορεί να χρησιμοποιηθεί αποκλειστικά σε Agents τύπου Android ανεξαρτήτως του state στο οποίο βρίσκονται. Κάθε τέτοιο αντικείμενο μπορεί να αποκτηθεί από τον παίκτη σε διάφορα σημεία του level, σε έναν Agent που έχει εξοντωθεί, καθώς και ως ανταμοιβή στην ολοκλήρωση ενός Quest ή Objective. Για την υλοποίηση των παραπάνω, δημιουργήθηκε η blueprint actor κλάση BP_Throwable_Base με βαλλιστικά χαρακτηριστικά, που λειτουργεί ως Parent Blueprint των αντικειμένων ρίψης που μπορεί να χρησιμοποιήσει ο παίκτης. Περιλαμβάνει τις λειτουργίες κάθε αντικειμένου, ακολουθώντας αναπτυξιακή δομή όμοια με αυτή του οπλισμού των Agents. Με αυτόν τον τρόπο, το περιεχόμενο των μεταβλητών και η εφαρμογή των χαρακτηριστικών του κάθε αντικειμένου ορίζεται σε κάθε Child Blueprint, καθιστώντας ευκολότερη την αποσφαλμάτωση και την προσθήκη ενός καινούριου αντικειμένου. Ακόμη, αναπτύχθηκε λειτουργία εκκίνησης και πρόβλεψης της τροχιάς της ρίψης στο blueprint του παίκτη, ώστε να καταστεί δυνατή η χρήση του αντικειμένου.

Σε αυτή την παράγραφο, θα περιγραφούν οι υλοποιήσεις του Parent Blueprint για το μέρος του κώδικα που αφορά το αντικείμενο Bottle, της ρίψης αντικειμένων στο Blueprint του παίκτη και οι τιμές και τα assets που ανατέθηκαν στο Child Blueprint. Η ανάλυση περιορίζεται στο αντικείμενο Bottle, καθώς είναι το μόνο από τα δύο υλοποιημένα αντικείμενα που σχετίζεται με το Investigate State. Η ανάλυση της υλοποίησης του DataWorm πραγματοποιείται στο υποκεφάλαιο 8.8, λόγω της άμεσης συσχέτισης των λειτουργιών του αντικειμένου με τα State Frozen και Attacking.

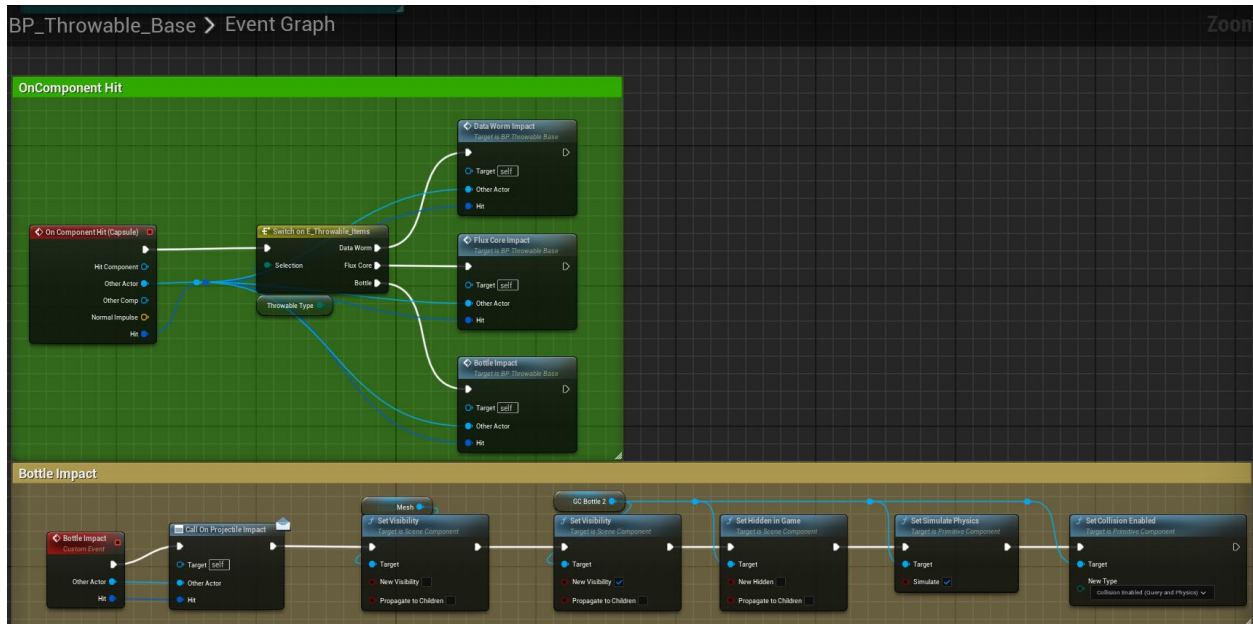
Parent Blueprint: Το αρχικό στάδιο της υλοποίησης του Parent Blueprint περιλαμβάνει την προσθήκη των κοινών Components που απαιτούνται για τη λειτουργία των αντικειμένων. Αρχικά προστέθηκε ένα Capsule Collision Component, που χρησιμοποιείται για την αναγνώριση της επαφής του αντικειμένου με μία επιφάνεια. Το Component ορίστηκε ως root του Actor ώστε τα υπόλοιπα Components να ακολουθούν την κίνησή του. Ως Child Components προστέθηκαν δύο Static Mesh Components, εκείνα των Mesh και DataWormAOE_Sphere, τα οποία ορίζονται στα Child Blueprints. Το πρώτο λειτουργεί ως placeholder για το mesh του αντικειμένου και το δεύτερο για τα ειδικά εφέ που αναπαράγονται κατά τη χρήση του DataWorm. Ακόμη, προστέθηκε το GC_Bottle, ένα geometry collection που χρησιμοποιείται ως δευτερεύον mesh του Bottle, το οποίο καταστρέφεται κατά την επαφή του αντικειμένου με μία επιφάνεια. Το GC_Bottle δημιουργήθηκε από το mesh του Bottle και διασπάστηκε εκ των προτέρων χρησιμοποιώντας τα εργαλεία fracture που παρέχονται από το σύστημα Chaos Destruction του Unreal Engine. Με αυτόν τον τρόπο, το Bottle χρησιμοποιεί ένα mesh κατά τη διάρκεια της ρίψης, το οποίο αντικαθίσταται από το GC_Bottle για την προσομοίωση της καταστροφής του αντικειμένου. Τέλος, προστέθηκε το ProjectileMovementComponent, το οποίο παρέχει βαλλιστικές λειτουργίες που χρησιμοποιήθηκαν κατά την εκτέλεση της ρίψης, καθιστώντας τον Actor ως projectile. Αυτές περιλαμβάνουν εκτόξευση του Actor με προσομοίωση φυσικής κίνησης, τη δυνατότητα αλληλεπίδρασης του αντικειμένου με τον κόσμο και τον προσδιορισμό παραμέτρων και χαρακτηριστικών που αφορούν στην επιρροή της βαρύτητας, της κατεύθυνσης του Actor κ.α. Ακόμη, το component παρέχει την ανάπτυξη λειτουργιών σε συγκεκριμένα συμβάντα, όπως για παράδειγμα στην περίπτωση που η ρίψη του αντικειμένου ολοκληρωθεί (π.χ. όταν η ταχύτητά του μηδενιστεί). Όσον αφορά στη δημιουργία μεταβλη-

τών, η ThrowableType (Enum) αντιπροσωπεύει τον τύπο του αντικειμένου και χρησιμοποιείται για την επιλογή των κατάλληλων ενεργειών κατά τη ρίψη και η Sound (SoundBase) αντιπροσωπεύει τον ήχο που αναπαράγεται κατά τη σύγκρουση του αντικειμένου. Το Parent Blueprint παρουσιάζεται στην εικόνα 87.



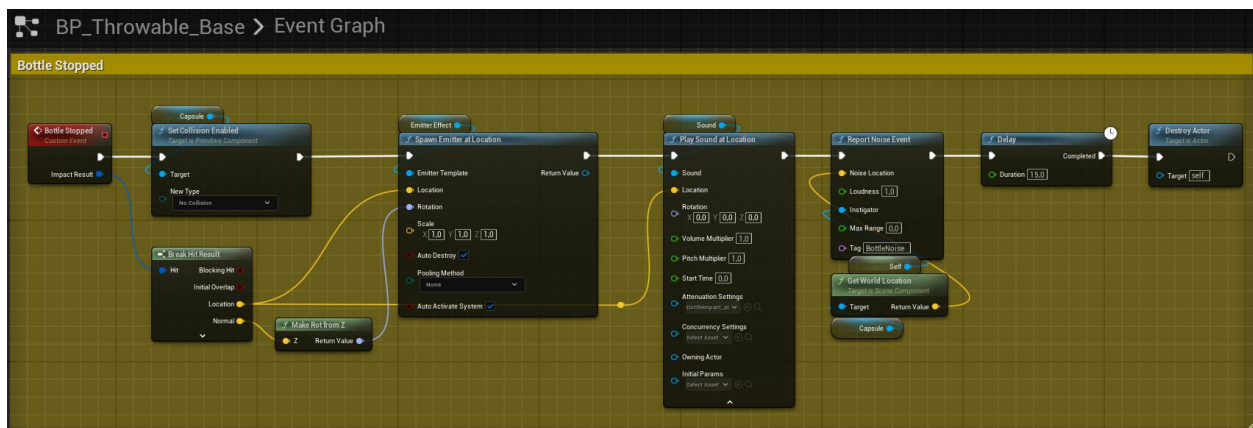
Εικόνα 87. Throwable Base (Parent Blueprint)

StartItemTrace (Player): Η ρίψη του αντικειμένου εκκινείται από τον παίκτη μέσω της function StartItemTrace η οποία προσφέρει μια πρόβλεψη της τροχιάς, ώστε ο παίκτης να καθορίσει την πορεία του αντικειμένου. Σύμφωνα με την επιλεγμένη πορεία, το function ThrowItem ορίζει τη μεταβλητή Velocity, η οποία αφορά το ProjectileMovementComponent και είναι απαραίτητη για την κίνηση του Projectile στον αέρα. Μετά την ανάθεση, το κατάλληλο αντικείμενο γίνεται spawn στο περιβάλλον του παιχνιδιού και κινείται σύμφωνα με την προκαθορισμένη τροχιά και κατεύθυνση της ρίψης. Η ανάπτυξη της πρόβλεψης τροχιάς ρίψης χρησιμοποιεί τη δημιουργία Spline Points μεταβάλλοντας το Spline σύμφωνα με ένα Timeline, κατά τη διάρκεια που ο παίκτης πατά το κατάλληλο πλήκτρο. Το διάγραμμα που περιγράφει τη διαδικασία εκτέλεσης της ρίψης και της δημιουργίας (spawn) του κατάλληλου Throwable βρίσκεται ενσωματωμένο στο Blueprint του παίκτη και απεικονίζεται στην εικόνα 88. Παράλληλα, η εικόνα 89 παρουσιάζει την οπτική αναπαράσταση της ρίψης όπως εμφανίζεται εντός του παιχνιδιού, τη στιγμή που ο παίκτης ενεργοποιεί τη σχετική λειτουργία.



Εικόνα 90. OnComponentHit & Bottle Impact

Το event Bottle Impact είναι υπεύθυνο για την απόκρυψη του mesh και την εμφάνιση του GC_Bottle, το οποίο καταστρέφεται κατά την πρόσκρουση. Αυτό επιτυγχάνεται από τα nodes SetVisibility και SetHiddenInGame, καθώς και από τα node SetSimulatePhysics και SetCollisionEnabled, τα οποία ενεργοποιούν την προσομοίωση της καταστροφής. Ακόμη, το event ενημερώνει το Blueprint του παίκτη για τη ρίψη μέσω του event dispatcher CallOnProjectileImpact, ώστε να μειωθεί ο αριθμός των διαθέσιμων αντικειμένων και να πυροδοτεί το event BottleStopped.



Εικόνα 91. BottleStopped

Σε αυτό το event, αρχικά απενεργοποιείται το collision του capsule component ώστε να μην εμποδίζει το περιβάλλον. Στη συνέχεια αναπαράγονται ένα επιπλέον ειδικό εφέ και ο ήχος της σύγκρουσης μέσω των Node SpawnEmitterAtLocation και PlaySoundAtLocation. Έπειτα, η διατάραξη αναφέρεται μέσω του ReportNoiseEvent, ώστε οι Agents που αντιλαμβάνονται τον ήχο να ενεργήσουν. Η εκτέλεση ολοκληρώνεται με τη διαγραφή του κατεστραμμένου αντικειμένου από το level, έπειτα από μια χρονική καθυστέρηση 15 δευτερολέπτων. Το διάγραμμα των events BottleImpact και BottleStopped παρουσιάζεται στις εικόνες 90 και 91. Τέλος δημιουργήθηκε ένα Sound Attenuation asset, το οποίο ελέγχει την ένταση και τον τρόπο αναπαραγωγής του ήχου της σύγκρουσης βάσει παραμέτρων, ώστε να προσφέρει ενισχυμένη εμβύθιση του παίκτη και ρεαλισμό κατά τη σύγκρουση του αντικειμένου. Με αυτόν τον τρόπο ο ήχος αναπαράγεται στον τρισδιάστατο χώρο, σύμφωνα με την απόσταση μεταξύ του σημείου αναπαραγωγής και του παίκτη, των εμποδίων που υπάρχουν στον μεταξύ τους χώρο και άλλες παραμέτρους σε σχέση με τον παίκτη και το περιβάλλον. Οι παράμετροι που χρησιμοποιήθηκαν περιλαμβάνουν το InnerRadius και το FalloffDistance, αντιπροσωπεύοντας την απόσταση από την οποία η ένταση του ήχου αρχίζει να μειώνεται και την απόσταση που ο ήχος παύει να είναι αντιληπτός. Ακόμη, ενεργοποιήθηκαν ρυθμίσεις όπως Occlusion, AirAbsorption και Reverb, για περαιτέρω μείωση του ήχου στην περίπτωση εμποδίων και την επιπλέον διαμόρφωσή του σύμφωνα με τον περιβάλλοντα χώρο. Οι ρυθμίσεις προσομοιώνουν τον τρόπο που αντιλαμβάνεται ένας ακροατής τον ήχο και το asset ανατίθεται στο πεδίο Attenuation Settings που παρέχει το node PlaySoundAtLocation.

8.4. Attacking State

Ένας Agent μεταβαίνει στο state Attacking όταν αντιληφθεί την παρουσία του παίκτη, το οποίο περιλαμβάνει τις επιθετικές ενέργειες που εκτελούνται. Οι ενέργειες περιλαμβάνουν την όπλιση του Agent, την εκτέλεση της επίθεσης, την πραγματοποίηση ελιγμών, καθώς και τις ενέργειες healing σε περίπτωση που ο Agent δεχθεί αρκετό damage. Η πραγματοποίηση των επιθέσεων επιτυγχάνεται μέσω της κλήσης του κατάλληλου task, το οποίο πυροδοτεί την υλοποίηση των επιθέσεων που βρίσκεται στο blueprint του κάθε child actor. Κάθε επίθεση λειτουργεί σε συνδυασμό με το σύστημα Token System, το οποίο χρησιμοποιείται ώστε οι ταυτόχρονες επιθέσεις των Agents προς ένα στόχο (π.χ. τον παίκτη) μια δεδομένη στιγμή, να μην υπερβαίνουν έναν συγκεκριμένο αριθμό. Ωστόσο, λόγω διαφορών ανάμεσα στις ενέργειες των Agent τύπου Pistol/Rifle και Android, οι υλοποιήσεις BT_Enemy_Ranged και BT_Enemy_Android παρουσιάζουν διαφοροποιήσεις. Για παράδειγμα, ένας Ranged Enemy μπορεί να αναπληρώσει τους πόντους ζωής του, αντίθετα με έναν Agent τύπου Android.

Η ενεργοποίηση του state μπορεί να συμβεί όταν ο παίκτης εισέλθει στο οπτικό πεδίο του Agent, καθώς το Behavior Tree του εκτελεί κάποιο από τα Passive, Investigating, Detecting και Seeking states. Η μεταβολή στο state πραγματοποιείται μέσω του function SetStateAsAttacking εντός του AI Controller, το οποίο θέτει το State Enum σε Attacking. Το function επίσης θέτει στο περιεχόμενο του blackboard key AttackTarget (Player) το reference του παίκτη, προκειμένου να χρησιμοποιηθεί στην κλήση των tasks και services εντός των Behavior Trees. Με εξαίρεση το Seeking State, στο οποίο ο Agent μεταβαίνει στην επιθετική κατάσταση ακαριαία, η μεταβολή στο Attacking state εξαρτάται από το χρόνο που απαιτείται ώστε ο Agent να αντιληφθεί πλήρως την παρουσία του παίκτη σύμφωνα με τις διαδικασίες που

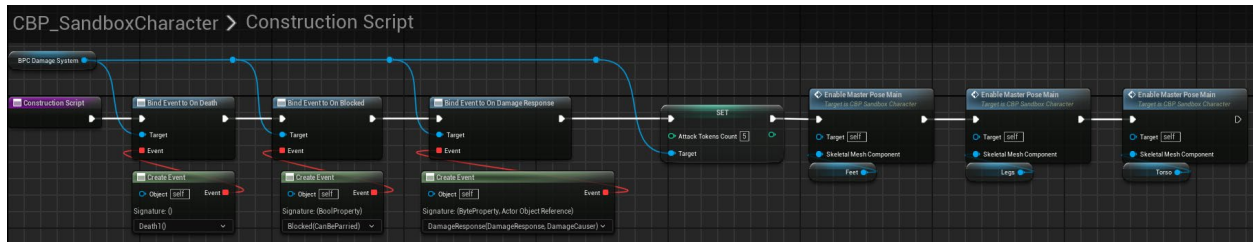
εκτελούνται στο Detecting State. Όπως αναφέρεται στο υποκεφάλαιο 8.6, το Detecting State μετρά τον χρόνο που ο παίκτης βρίσκεται στο οπτικό πεδίο του Agent, μεταβάλλοντάς τον σε κατάσταση επίθεσης όταν παρέλθει συγκεκριμένος χρόνος.

Σε αυτό το κεφάλαιο πραγματοποιείται μια επισκόπηση του συστήματος TokenSystem που χρησιμοποιείται για τον καθορισμό της εκτέλεσης μιας επίθεσης, της δομής των Behavior Tree για το Attacking State κάθε τύπου Agent, και αναλύονται οι υλοποιήσεις των ενεργειών που εκτελούνται κατά τη διάρκεια του state.

8.4.1. Token System

Το σύστημα TokenSystem χρησιμοποιείται για να καθοριστεί εάν ένας Agent που βρίσκεται στο Attacking State θα εκτελέσει μια επίθεση εναντίον του παίκτη. Η εφαρμογή του επιτρέπει τον καθορισμό του μέγιστου αριθμού ταυτόχρονων επιθέσεων που μπορούν να λάβουν χώρα, εφαρμόζοντας μια οικονομία Token. Αυτό επιτυγχάνεται με τη δημιουργία της μεταβλητής AttackTokensCount (Integer) και την αυξομείωσή της κατά την έναρξη/ολοκλήρωση μιας επίθεσης. Όσον αφορά στη λειτουργία του συστήματος, κάθε επίθεση ενός Agent δεσμεύει έναν συγκεκριμένο αριθμό Tokens από το στόχο του, αφαιρώντας το ποσό από τη μεταβλητή AttackTokensCount. Με αυτόν τον τρόπο μια επίθεση θα εκτελεστεί, εάν ο αριθμός των Tokens που απαιτεί είναι μικρότερος ή ίσος από τον αριθμό των διαθέσιμων Tokens (AttackTokensCount) του στόχου προς επίθεση. Ακόμη, τα Tokens που δεσμεύει μια επίθεση αποθηκεύονται, ώστε να αφαιρεθούν κατά την ολοκλήρωση της επίθεσης ή κατά την εξάλειψη του Agent. Ο έλεγχος των προϋποθέσεων εκτέλεσης μιας επίθεσης, πραγματοποιείται σε κάθε κλήση του αντίστοιχου task που πυροδοτεί την επίθεση ενός Agent, εντός του Behavior Tree. Ως αποτέλεσμα, μια επίθεση ξεκινά από το στάδιο του ελέγχου σύμφωνα με τα διαθέσιμα Tokens, και μπορεί να έχει δυο πιθανές εκβάσεις. Αν τα απαραίτητα Tokens είναι διαθέσιμα, η επίθεση εκτελείται και το Behavior Tree συνεχίζει κανονικά τη λειτουργία του. Σε αντίθετη περίπτωση, η επίθεση αναβάλλεται, και το δέντρο αποτυγχάνει προκαλώντας επαναξιολόγηση. Το σύνολο των ενεργειών του συστήματος, συμπεριλαμβανομένων των ελέγχων, των αυξομειώσεων και της αποθήκευσης τιμών πυροδοτείται μέσω του function Attack Start (Attempt), η οποία χρησιμοποιείται εντός του task που αφορά στην επίθεση του κάθε Agent.

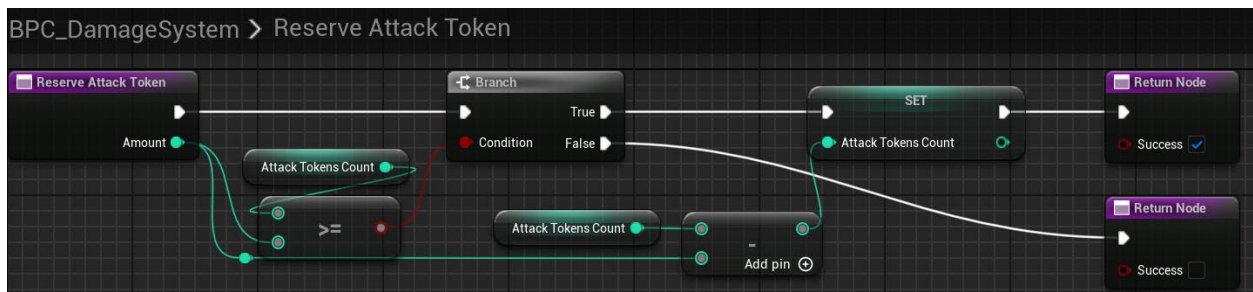
Η μεταβλητή AttackTokensCount τοποθετείται στο component BPC_DamageSystem και αντιπροσωπεύει τον μέγιστο αριθμό των ταυτόχρονων επιθέσεων που μπορεί να δεχθεί ένας actor (π.χ. ο παίκτης) μια δεδομένη στιγμή. Οι ενέργειες της αυξομείωσης της μεταβλητής και η αποθήκευση των Token που έχουν δεσμευτεί ορίζονται στα Blueprint interfaces BPI_Damageable και BPI_Enemy_AI, με την υλοποίησή τους να βρίσκεται στα Blueprint BP_Enemy_Base και AIC_Controller_Base. Στο συγκεκριμένο project, η AttackTokensCount παίρνει την τιμή 5 για τον παίκτη, με την ανάθεσή της να γίνεται κατά την αρχικοποίηση του από το construction script, όπως φαίνεται στην εικόνα 92.



Εικόνα 92. Player Token Setup

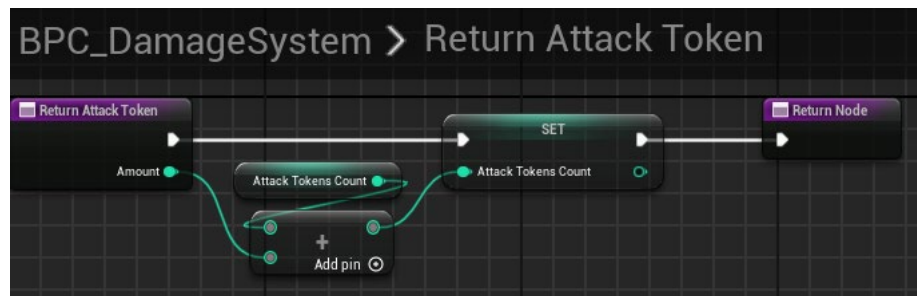
Ως αποτέλεσμα, το σύνολο των ταυτόχρονων επιθέσεων που μπορεί να δεχθεί ο παίκτης είναι 5, ανεξαρτήτως το είδος της επίθεσης και τον τύπο του Agent που την εκτελεί. Η χρήση της μεταβλητής επιτρέπει τη δυναμική ανάθεση της τιμής της βάσει διαφόρων παραγόντων, όπως για παράδειγμα σύμφωνα με το επίπεδο δυσκολίας του Level, τις ικανότητες που αποκτά ο παίκτης κατά τη διάρκεια του παιχνιδιού κ.α. Τέλος, το σύστημα μπορεί να διαχειριστεί τις επιθέσεις μεταξύ οποιουδήποτε actor, στην περίπτωση που η υλοποίηση επιτρέπει σε κάθε Agent να επιτεθεί σε οποιονδήποτε άλλο Actor, σύμφωνα με τα διαθέσιμα Tokens του. Αυτό επιτυγχάνεται με τον ορισμό της μεταβλητής AttackTokensCount σε κάθε actor που δέχεται damage, όπως για παράδειγμα σε κάθε Agent, καθώς και τον τρόπο αποθήκευσης των δεσμευμένων Token όπως περιγράφεται στην ανάλυση του function StoreAttackTokens. Για τις ανάγκες του συγκεκριμένου project, η μεταβλητή AttackTokensCount ορίστηκε και χρησιμοποιήθηκε μόνο για τον παίκτη, καθώς αποτελεί το μοναδικό στόχο των Agents.

ReserveAttackToken (Fuction): Το function είναι υπεύθυνο για το αν θα αφαιρεθεί ή όχι η ορισμένη τιμή Token που κοστίζει η εκάστοτε επίθεση από τα συνολικά διαθέσιμα Token. Δέχεται ως είσοδο τη μεταβλητή Amount (Integer) και επιστρέφει τη μεταβλητή Success (boolean). Η μεταβλητή Amount αντιπροσωπεύει το κόστος της επίθεσης σε Token και συγκρίνεται με την τιμή της μεταβλητής AttackTokensCount. Αν το κόστος είναι μικρότερο ή ίσο από τον αριθμό των διαθέσιμων Tokens, τότε τα διαθέσιμα Tokens μειώνονται και η συνάρτηση επιστρέφει τη μεταβλητή Success με τιμή true, ώστε να συνεχίσει η εκτέλεση της επίθεσης. Σε αντίθετη περίπτωση το function ολοκληρώνεται επιστρέφοντας το Success με τιμή false. Το function υλοποιείται εντός του component BPC_Damageable αλλά ορίζεται και στο blueprint interface BPI_Enemy_AI ώστε να μπορεί να κληθεί από τα task του Behavior Tree χωρίς την χρήση Cast To.



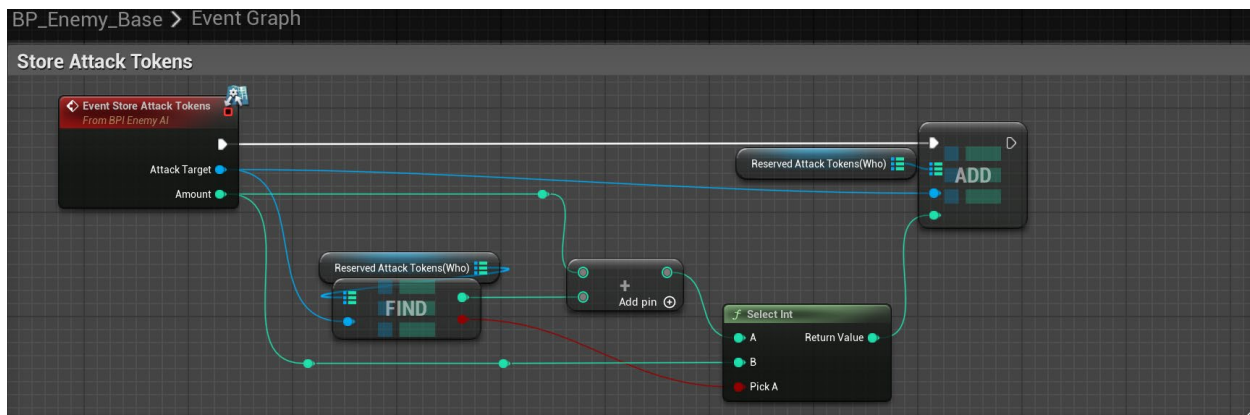
Εικόνα 93. ReserveAttackToken

ReturnAttackToken (Fuction): Το function είναι υπεύθυνο για την αναπλήρωση του αριθμού των διαθέσιμων Tokens στον παίκτη. Καλείται κατά την ολοκλήρωση μιας επίθεσης ή την εξάλειψη ενός Agent που έχει δεσμεύσει Tokens, ώστε να μπορούν να δεσμευθούν από άλλους Agents. Δέχεται ως είσοδο τα Tokens προς επιστροφή με τη μεταβλητή Amount (Integer), το περιεχόμενο της οποίας προστίθεται στη μεταβλητή AttackTokensCount. Όπως και το function ReserveAttackToken, υλοποιείται στο component BPC_Damageable αλλά για την κλήση του χρησιμοποιείται ο ορισμός του function στο blueprint interface BPI_Enemy_AI.



Εικόνα 94. ReturnAttackToken

StoreAttackTokens (Function) Το function είναι υπεύθυνο για την αποθήκευση των Tokens που έλαβε ένας Agent από έναν συγκεκριμένο actor, για την εκτέλεση της επίθεσής του. Δέχεται ως είσοδο τα Tokens προς αποθήκευση μέσω της μεταβλητή Amount (Integer) και τον Actor από τον οποίο δεσμεύονται (π.χ. τον παίκτη) μέσω της μεταβλητής AttackTarget (Actor Object Reference). Υλοποιείται στο parent blueprint των Agents (BP_Enemy_Base) και για την κλήση του χρησιμοποιεί το Blueprint interface BPI_Enemy_AI. Παρότι στο συγκεκριμένο project ο μοναδικός στόχος των Agents είναι ο παίκτης, η υλοποίηση της αποθήκευσης περιλαμβάνει την περίπτωση όπου κάθε Agent μπορεί να επιτεθεί σε οποιονδήποτε άλλο actor που μπορεί να δεχθεί damage, δηλαδή εφαρμόζει το component BPC_Damageable. Καθώς πολλαπλοί Agents μπορεί να έχουν δεσμεύσει διαφορετικό αριθμό Tokens από διαφορετικούς actors, η υλοποίηση απαιτεί την αποθήκευση των Tokens για κάθε actor ξεχωριστά, ώστε η τιμή να μπορεί να ανακτηθεί.

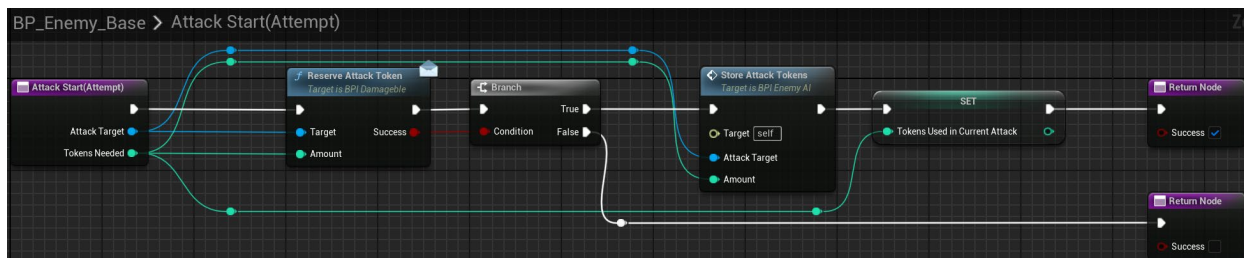


Εικόνα 95. StoreAttackTokens

Αυτό επιτυγχάνεται με την αποθήκευση των τιμών σε μια μεταβλητή Container και συγκεκριμένα τύπου map. Η μεταβλητή επιτρέπει την αποθήκευση και γρήγορη αναζήτηση συσχετισμένων δεδομένων μεταξύ δύο τύπων τιμών, σε αυτήν την περίπτωση μεταξύ ενός Actor και ενός ακέραιου αριθμού. Μέσω της χρήσης του τύπου map, μπορεί να καταχωρηθεί σε κάθε Actor (π.χ. Agent) ο αντίστοιχος αριθμός Tokens από τον οποίο έχουν δεσμευτεί, επιτρέποντας έτσι την άμεση πρόσβαση, ενημέρωση ή αφαίρεση των δεδομένων, χωρίς τη χρήση διατήρησης ξεχωριστών μεταβλητών ή πολύπλοκων δομών. Ως εκ τούτου, δημιουργήθηκε η μεταβλητή Reserved Attack Tokens, η οποία περιέχει κάθε Actor του οποίο τα Token έχουν δεσμευτεί μια δεδομένη στιγμή, και το αντίστοιχο ποσό των Tokens. Η υλοποίηση περιλαμβάνει την προσθήκη του κατάλληλου ποσού στη μεταβλητή Reserved Attack Tokens μέσω του node Add (Map), με είσοδο τον actor στόχο, και το ποσό. Για την επιλογή του ποσού που θα αποθηκευτεί, το function ελέγχει εάν έχουν ήδη δεσμευτεί κάποια tokens από τον συγκεκριμένο actor, δηλαδή την περίπτωση που ο actor έχει ήδη εκχωρηθεί στη μεταβλητή. Με τη χρήση του node Find (map), η μεταβλητή σαράνεται αναζητώντας τον actor, και αν αυτός βρεθεί, τα Tokens που έχουν ήδη δεσμευτεί αυξάνονται κατά το καινούριο ποσό που παρέχει το function. Σε αντίθετη περίπτωση, ο actor προστίθεται στο map με την τιμή των tokens του function, καθώς κανένα token δεν έχει δεσμευτεί μέχρι αυτή τη στιγμή.

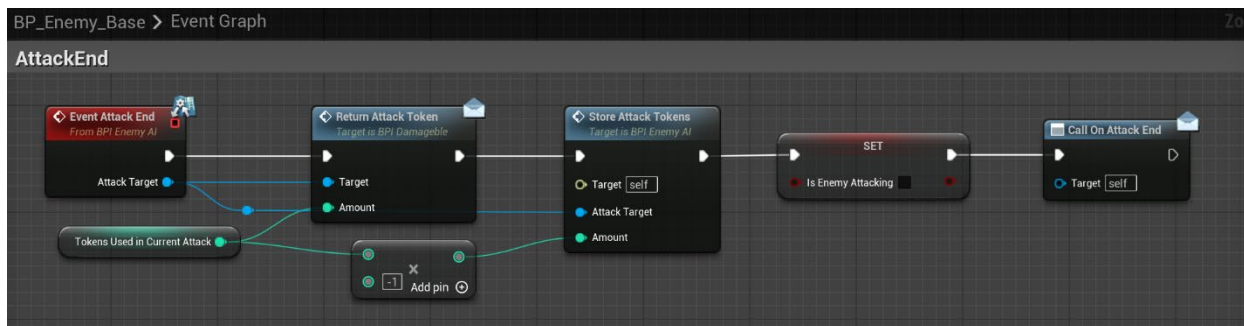
Attack Start (Attempt) (Function): Το function καθορίζει αν μια επίθεση ενός Agent θα πραγματοποιηθεί, σύμφωνα με τα διαθέσιμα tokens του actor στόχου και το κόστος της επίθεσης, επιστρέφοντας το αποτέλεσμα μέσω μιας μεταβλητής Boolean. Πυροδοτείται κατά την κλήση ενός task επίθεσης μέσω του blueprint interface BPI_Enemy_AI, και υλοποιείται στο parent blueprint των Agents. Συγκεκριμένα, καλεί το function ReserveAttackToken, το οποίο αφαιρεί τα Tokens της επίθεσης από τα διαθέσιμα Tokens και επιστρέφει την επιτυχία ή αποτυχία της αφαίρεσης. Εάν υπάρχουν διαθέσιμα tokens και η αφαίρεση είναι επιτυχής, το function συνεχίζει με την αποθήκευση των Tokens που αφαιρέθηκαν μέσω της StoreAttackTokens. Έπειτα, η τιμή των Tokens που δεσμεύτηκαν αποθηκεύεται στη μεταβλητή TokensUsedInCurrentAttack, η οποία χρησιμοποιείται κατά την επιστροφή των Tokens στο function AttackEnd. Τέλος, το function ολοκληρώνεται επιστρέφοντας την τιμή true, σηματοδο-

τώντας πως η επίθεση μπορεί να πραγματοποιηθεί. Στην αντίθετη περίπτωση, το AttackStartAttempt επιστρέφει την τιμή false, με αποτέλεσμα τη ματαίωση της επίθεσης, καθώς όλα τα διαθέσιμα Tokens έχουν κατακτηθεί από άλλους Agents. Αυτό θα διακόψει τις ενέργειες της επίθεσης, προκαλώντας το Behavior Tree να εκτελέσει μια άλλη ενέργεια εντός του Attacking State, μέχρι το Attack να επανακληθεί και να υπάρχουν διαθέσιμα Tokens.



Εικόνα 96. Attack Start (Attempt)

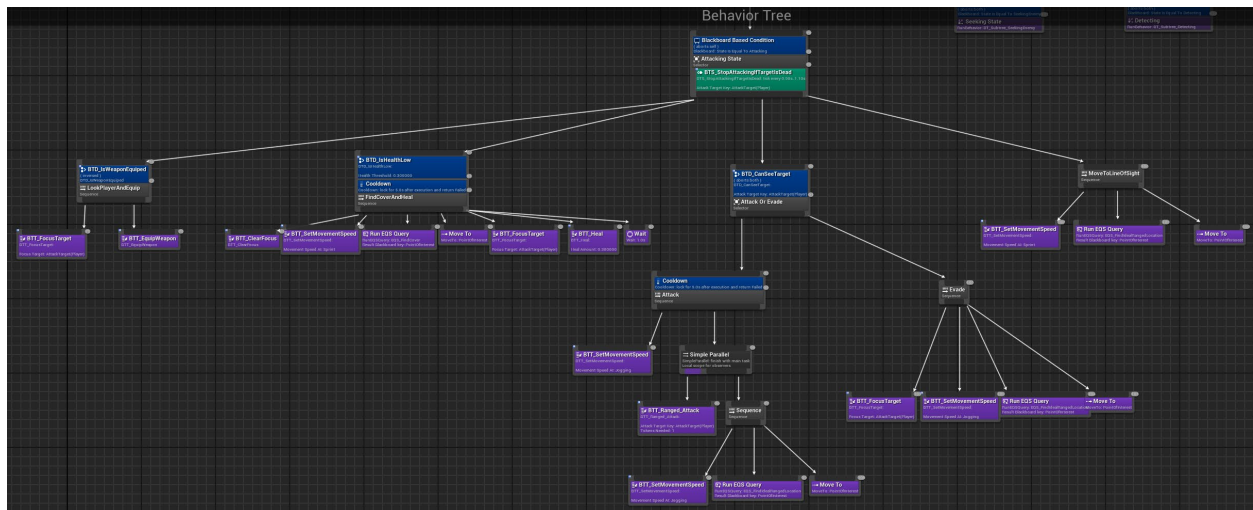
AttackEnd (Function): Το function είναι υπεύθυνο για την αναπλήρωση του αριθμού των διαθέσιμων Tokens στον actor από τον οποίο δεσμευτήκαν, ενημερώνοντας τη μεταβλητή AttackTokensCount στο component του actor αλλά και τον αριθμό των tokens που έχουν αποθηκευτεί στη μεταβλητή ReserveAttackToken. Καλείται μέσω του blueprint interface BPI_Enemy_AI κατά την ολοκλήρωση μιας επίθεσης ή κατά την εξάλειψη ενός Agent που έχει αποκτήσει Tokens, και περιλαμβάνει την κλήση των functions ReturnAttackToken και StoreAttackTokens. Ακόμη, θέτει τη μεταβλητή IsEnemyAttacking σε false, η οποία αντιπροσωπεύει την κατάσταση επίθεσης του Agent και χρησιμοποιείται σε διάφορα σημεία ελέγχου του κώδικα. Το function ολοκληρώνεται με την κλήση του event dispatcher CallOnAttackEnd, το οποίο σηματοδοτεί την επιτυχή ολοκλήρωση της επίθεσης, ώστε το task το οποίο την καλεί να ολοκληρωθεί την κατάλληλη στιγμή και το δέντρο να προχωρήσει στην επόμενη ενέργεια χωρίς ανεπιθύμητες συμπεριφορές.



Εικόνα 97. AttackEnd

8.4.2. Ranged Attacking State

Attacking Behavior Tree Overview (Ranged): Το δέντρο ενεργοποιείται κατά την αξιολόγηση του State Selector μέσω Blackboard Based Conditions, δηλαδή στην περίπτωση που το περιεχόμενο της μεταβλητής State (Enum) έχει την τιμή Attacking. Ο Selector Attacking State διαθέτει ένα service το οποίο ελέγχει καθ' όλη τη διάρκεια της εκτέλεσης του State αν ο παίκτης είναι ζωντανός. Χρησιμοποιείται στην διακοπή της επίθεσης του Agent όταν ο παίκτης πεθάνει, μεταβάλλοντάς τον σε κατάσταση Passive. Ακόμη, το service διασφαλίζει εξαρχής πως το state δεν θα ενεργοποιηθεί αν ο παίκτης έχει πεθάνει. Ο Selector ακολουθείται από ένα σύνολο Selectors και Sequences τοποθετημένα από το πιο σημαντικό στο λιγότερο σημαντικό, διασφαλίζοντας την εκτέλεση της κατάλληλης ενέργειας σε κάθε στιγμή της επίθεσης.



Εικόνα 98. Attacking Behavior Tree Overview (Ranged)

Η αρχική ενέργεια προς εκτέλεση είναι το Sequence LookPlayerAndEquip, στο οποίο ενεργοποιείται η εστίαση του Agent προς τον παίκτη και ο οπλισμός του όπλου, μέσω των tasks BTT_FocusTarget και BTT_EquipWeapon αντίστοιχα. Το task εκτελείται μόνο όταν ο Agent δεν έχει ήδη οπλίσει, σύμφωνα με τον decorator BTD_IsWeaponEquipped. Ο Selector AttackOrEvade περιλαμβάνει τις βασικές ενέργειες επίθεσης του Agent, κατά τις οποίες πυροβολεί προς τον παίκτη (Attack Sequence) και εκτελεί ελιγμούς διατηρώντας τον παίκτη στο οπτικό του πεδίο μέχρι την επόμενη επίθεση (Evade Sequence). Το sequence Attack θέτει την ταχύτητα του παίκτη σε Jogging μέσω του task BTT_SetMovementSpeed και συνεχίζει εκτελώντας τον πυροβολισμό εναντίον του παίκτη, μετακινώντας παράλληλα τον Agent προς ένα ιδανικό σημείο. Με αυτό τον τρόπο ο Agent μπορεί να κινηθεί με αυξημένη ταχύτητα κατά την επίθεση, να πυροβολήσει τον παίκτη, αλλάζοντας συνεχώς την τοποθεσία του, ώστε να αποτελέσει δύσκολο στόχο και να διατηρήσει τον παίκτη στο οπτικό του πεδίο. Αυτό επιτυγχάνεται με τη χρήση του Simple Parallel node, με βασικό task το BTT_Ranged Attack για τον πυροβολισμό, και την παράλληλη εκτέλεση του sequence Strafe, κατά το οποίο ο Agent αναζητά και κινείται προς ένα σημείο μέσω των tasks

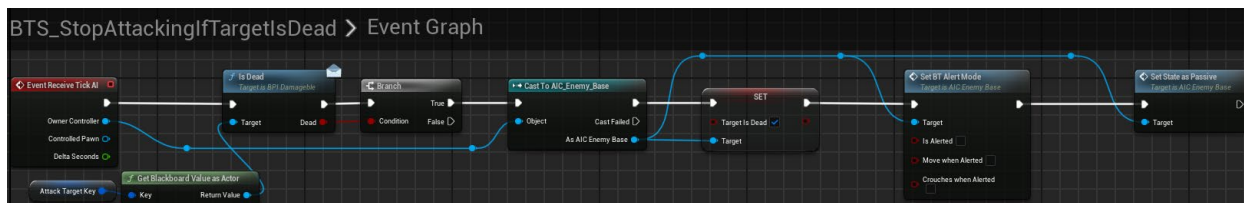
RunEQSQuery και MoveTo. Ως δευτερεύουσα ενέργεια, το sequence Strafe εκτελείται μέχρι να ολοκληρωθεί ο πυροβολισμός, χωρίς ο Agent να φτάσει απαραίτητα στο ορισμένο σημείο που προκύπτει από το EQS Query. Ως αποτέλεσμα, το Sequence Attack ολοκληρώνεται μετά τον πυροβολισμό, και το Behavior Tree εκτελεί τις ενέργειες του sequence Evade. Αυτές περιλαμβάνουν τον ορισμό του παίκτη ως σημείο εστίασης (καθώς μπορεί να έχει αποδεσμευτεί από τον Agent πριν την είσοδο στο sequence), τη μεταβολή της ταχύτητας σε Jogging, καθώς και την αναζήτηση και μετακίνηση του Agent προς ένα σημείο ελιγμού, μέσω του ίδιου EQS Query με αυτό του sequence Attack και του task MoveTo.

Καθώς το Sequence Attack αποτελεί σημαντικότερη ενέργεια από αυτή του Evade, η ενεργοποίησή της θα πραγματοποιούνταν σε κάθε επαναξιολόγηση του δέντρου, με αποτέλεσμα το sequence Evade να μην εκτελεστεί ποτέ. Ως εκ τούτου, το Sequence Attack διαθέτει ένα Cooldown 5 δευτερολέπτων, ώστε να εκτελείται παροδικά, επιτρέποντας το sequence Evade να ενεργοποιηθεί. Ακόμη, ο Selector AttackOrEvade χρησιμοποιεί τον Decorator BTDCanSeeTarget ώστε οι ενέργειες των επιθέσεων να εκτελεστούν μόνο όταν ο παίκτης βρίσκεται στο οπτικό πεδίο του Agent. Αυτό σημαίνει πως όταν ο παίκτης βρεθεί εκτός του οπτικού πεδίου, οι ενέργειες που περιλαμβάνει ο Selector θα παραλειφθούν ή θα διακοπούν, προκειμένου ο Agent να κινηθεί σε ένα σημείο που έχει οπτική επαφή με τον παίκτη μέσω του Sequence MoveToLineOfSight. Σε αυτό το Sequence, η ταχύτητα του παίκτη μεταβάλλεται σε Sprint ώστε να κινηθεί γρήγορα προς το σημείο, το οποίο προκύπτει από ένα ακόμη EQS Query.

Τέλος, το sequence FindCoverAndHeal προστέθηκε ώστε ο Agent να βρεί ένα σημείο κάλυψης και να αναπληρώσει ένα μέρος των πόντων ζωής του, αν αυτοί βρεθούν κάτω από ένα συγκεκριμένο όριο. Το sequence διαθέτει τον Decorator BTDisHealthLow, υπεύθυνο για τον έλεγχο των πόντων ζωής του Agent και την ενεργοποίηση των ενεργειών που περιλαμβάνονται, και ένα Cooldown ώστε να αποφευχθεί η επανεκτέλεση της ενέργειας εντός μικρού χρονικού διαστήματος. Το Sequence ξεκινά με τη διακοπή της εστίασης του Agent προς τον παίκτη και τη μεταβολή της ταχύτητάς του σε Sprint ώστε να κινηθεί με τη μεγαλύτερη δυνατή ταχύτητα προς ένα ασφαλές σημείο. Στη συνέχεια, το ιδανικό σημείο κάλυψης θα αναζητηθεί μέσω EQS Query, και μέσω του MoveTo ο Agent θα κινηθεί προς το σημείο. Κατά την άφιξή του, το task BTT_FocusTarget θα προκαλέσει την επανεστίαση του Agent προς τον παίκτη, το BTT_Heal θα δώσει πόντους ζωής και το sequence θα ολοκληρωθεί μετά από μια αναμονή 1 δευτερολέπτου. Το sequence τοποθετήθηκε αριστερά από τις ενέργειες της επίθεσης, αποκτώντας υψηλότερη προτεραιότητα εκτέλεσης. Χρησιμοποιώντας την ίδια δομή, η συμπεριφορά των Agents θα μπορούσε να εμπλουτιστεί με επιπλέον ενέργειες. Χαρακτηριστικό παράδειγμα αποτελεί η ανάπτυξη ενός Sequence Reload, όπου ο Agent διακόπτει την επίθεση του και γεμίζει τις σφαίρες του όπλου του. Μια τέτοια υλοποίηση θα απαιτούσε ένα σύνολο από Decorators και Services που θα ελέγχουν την κατάσταση των πυρομαχικών, καθώς και την ανάπτυξη tasks και functions για την περαίωση της διαδικασίας.

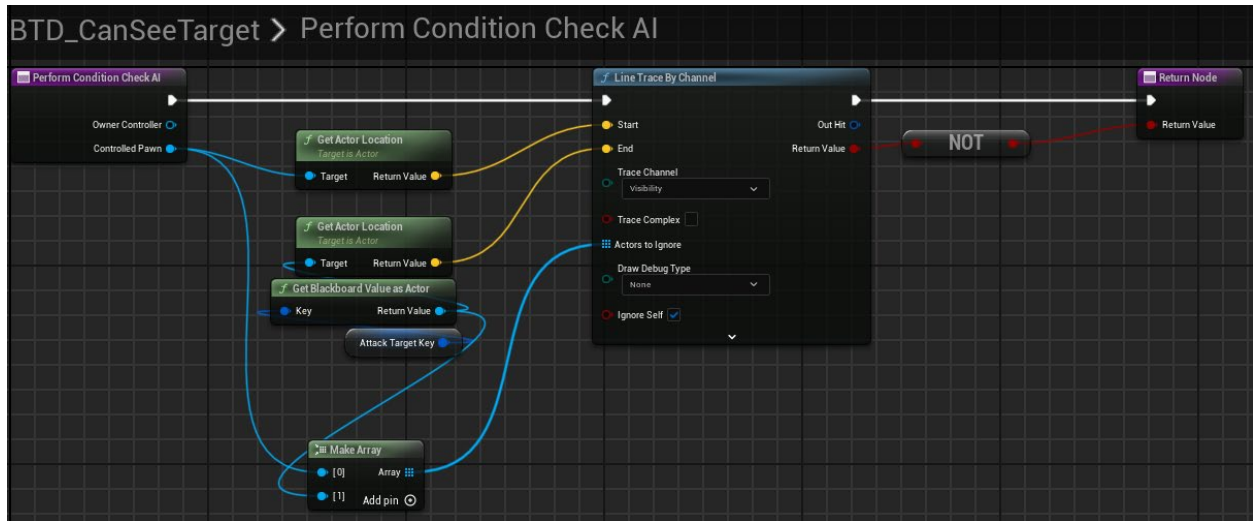
BTS_StopAttackingIfTargetIsDead (Service): Το Service ελέγχει εάν ο παίκτης έχει εξουδετερωθεί. Αυτό επιτυγχάνεται με την κλήση του blueprint interface function IsDead με target το reference του παίκτη. Το reference ανακτάται από την ανάγνωση του εκχωρημένου key (AttackTarget) κατά την κλήση του service, μέσω του GetBlackboardValueAsActor. Το function επιστρέφει την Boolean IsDead, που

βρίσκεται στο component `BPC_DamageSystem`, η οποία χρησιμοποιείται ως έλεγχος ώστε το Behavior Tree να μεταβεί στην κατάσταση `Passive State`, και να θέσει τη μεταβλητή `TargetIsDead` σε `true`. Αυτό απαιτεί τη χρήση `Cast To AIC_Enemy_Base`, για την πρόσβαση του Service στο AI Controller του Agent, στο οποίο βρίσκονται η μεταβλητή και το Function `SetStateAsPassive`. Το service ενεργοποιείται κατά την είσοδο του Agent στο `Attacking state`, και καλείται κάθε 1 δευτερόλεπτο, μέσω της ρύθμισης `Interval` που παρέχεται. Η τιμή της ρύθμισης αναπαριστά τον χρόνο που απαιτείται για την επανάκληση του service, εξασφαλίζοντας πως το Behavior Tree θα ενημερωθεί για την αλλαγή της κατάστασης του παίκτη, ώστε να διακόψει τις ενέργειες του state. Η τιμή που επιλέχθηκε παρέχει ικανοποιητικό χρόνο ενημέρωσης, χωρίς μεγάλη αύξηση του υπολογιστικού κόστους, μέσω διαρκών κλήσεων του service.



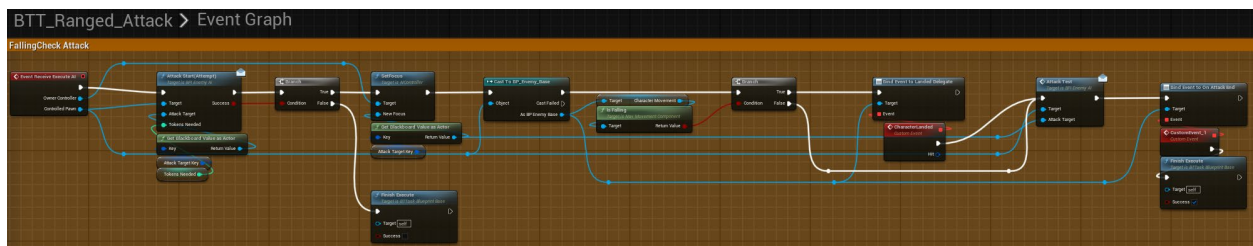
Εικόνα 99. `BTS_StopAttackingIfTargetIsDead`

BTD_CanSeeTarget (Decorator): Ο Decorator ελέγχει αν ο παίκτης βρίσκεται στο οπτικό πεδίο του Agent, και χρησιμοποιείται για τον καθορισμό της εκτέλεσης του Sequence `AttackOrEvade` κατά την αξιολόγηση στο επίπεδο του Selector `Attacking State`. Δέχεται για είσοδο ένα blackboard key selector, στο οποίο ανατίθεται το περιεχόμενο του key `AttackTarget(Player)`, και επιστρέφει το αποτέλεσμα ως μεταβλητή `Boolean`. Ο Decorator περιλαμβάνει το node `LineTracebyChannel`, όπου χρησιμοποιείται για τον έλεγχο παρουσίας κάποιου εμποδίου, στη γραμμή μεταξύ του Agent και του παίκτη. Αν η γραμμή δε βρει κάποιο εμπόδιο, αυτό σημαίνει πως ο παίκτης βρίσκεται στο οπτικό πεδίο του Agent. Αυτό επιτυγχάνεται εκτελώντας το trace με αρχή την τοποθεσία του Agent και τέλος την τοποθεσία του παίκτη, αλλά αγνοώντας τους δυο actors από το αποτέλεσμα μέσω της εισόδου `ActorsToIgnore`. Η μεταβλητή `Boolean` που επιστρέφει το trace παίρνει τη τιμή `true` αν η γραμμή μεταξύ του Agent και του παίκτη διακόπηκε από κάποιο εμπόδιο, και `false` σε αντίθετη περίπτωση. Καθώς ο Decorator επιστρέφει εάν ο παίκτης βρίσκεται εντός του οπτικού πεδίου του Agent, η τιμή του trace αντιστρέφεται με τη χρήση του `Not Boolean`. Με αυτόν τον τρόπο, κάθε φορά που ένα αντικείμενο βρεθεί μεταξύ των δύο actors, οι ενέργειες του `AttackOrEvade` θα διακοπουν. Ωστόσο, ο έλεγχος δεν εξετάζει την περίπτωση που ο παίκτης απομακρύνεται σε απόσταση μεγαλύτερη από το `LoseSightRadius` του Agent, όπως αυτή ορίζεται στο AI Perception component. Οι ενέργειες που εκτελούνται σε αυτή την περίπτωση περιγράφονται στο υποκεφάλαιο 8.5.



Εικόνα 100. BTD_CanSeeTarget

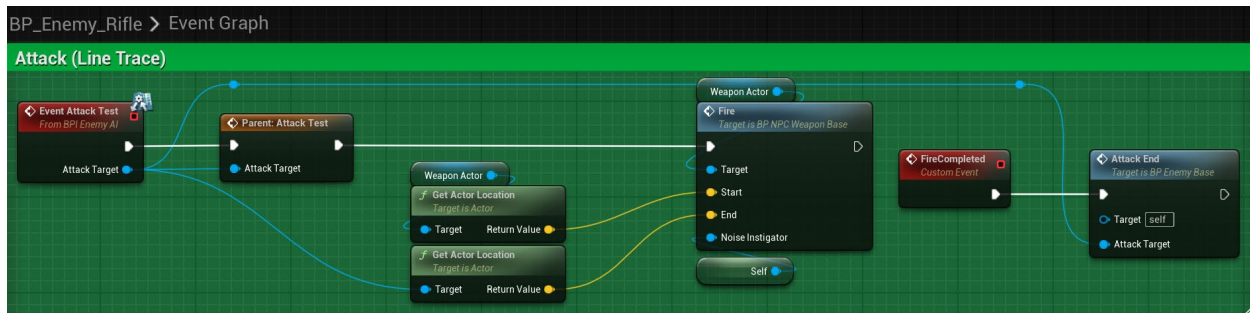
BTT_Ranged_Attack (Custom Task): Το function είναι υπεύθυνο για την εκτέλεση της επίθεσης των Agent τύπου Pistol και Rifle, κατά την οποία πυροβολούν τον παίκτη. Το task δέχεται ως είσοδο το AttackTargetKey (blackboard key selector) με τιμή το key AttackTarget(PLAYER), και τη μεταβλητή TokensNeeded (Integer). Η τιμή της TokensNeeded εκχωρείται κατά την κλήση του task και αντιπροσωπεύει την τιμή κόστους για την εκτέλεση μιας επίθεσης, η οποία χρησιμοποιείται για τον καθορισμό της πραγματοποίησης της επίθεσης σύμφωνα με το TokenSystem. Σε αυτό το project, η τιμή μιας τέτοιας επίθεσης ορίστηκε ώστε να έχει την τιμή 1. Ο έλεγχος που περιλαμβάνει το TokenSystem εξαναγκάζει την υλοποίηση της επίθεσης να ενσωματώσει την ενέργεια της εστίασης του Agent προς τον παίκτη εντός του task. Αυτό είναι απαραίτητο, ώστε η εστίαση να πραγματοποιηθεί μόνο όταν η επίθεση του Agent πληροί τις προϋποθέσεις των διαθέσιμων Tokens. Συγκεκριμένα, αν η ενέργεια της εστίασης πραγματοποιηθεί μέσω του task BTT_SetFocus, αυτό θα έπρεπε να τοποθετηθεί πριν το task της επίθεσης. Αυτό θα προκαλούσε την εστίαση ενός Agent προς τον παίκτη ακόμη και αν η επίθεση δεν πραγματοποιηθεί, με αποτέλεσμα ο Agent να μεταβεί σε ένα sequence που απαιτεί να μην έχει οριστεί σημείο εστίασης.



Εικόνα 101. BTT_Ranged_Attack

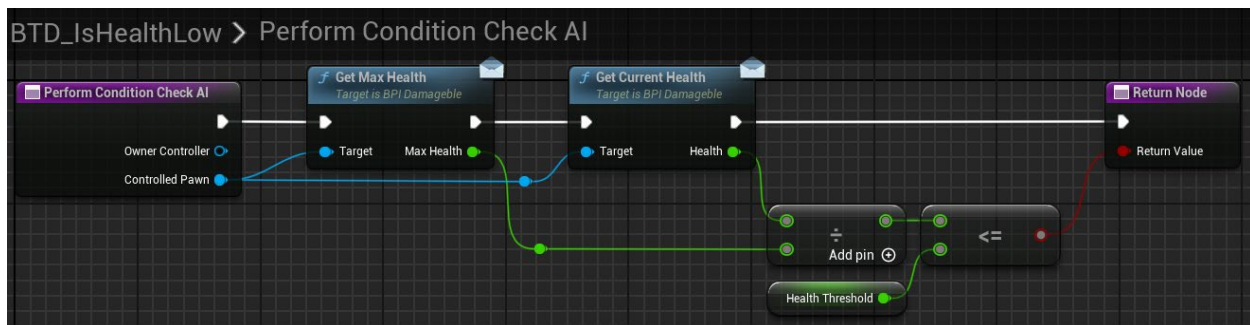
Όσον αφορά στην υλοποίηση του task, αρχική ενέργεια αποτελεί ο έλεγχος για την πραγματοποίηση της επίθεσης μέσω του function `AttackStart(Attempt)`, το οποίο επιστρέφει αν η δέσμευση ήταν επιτυχής. Αν η συνθήκη δεν ικανοποιείται, το task ολοκληρώνεται με αποτυχία μέσω του node `Finish Execute` με την τιμή `Success` ως `False`, με αποτέλεσμα το sequence `Attack` να διακοπεί και το δέντρο να μεταβεί στο sequence `Evade`. Σε αντίθετη περίπτωση, ο παίκτης ορίζεται ως σημείο εστίασης και αποκτάται πρόσβαση το parent blueprint μέσω του `CastTo BP_Energy_Base`. Ένας επιπλέον έλεγχος προηγείται της κλήσης του function της επίθεσης, ώστε να αποφευχθεί η περίπτωση που ο Agent βρίσκεται στον αέρα (π.χ. εκτελεί κάποιο άλμα) και το όπλο πυροδοτείται. Ο έλεγχος πραγματοποιείται με την χρήση της τιμής `IsFalling`, που παρέχεται από το `CharacterMovementComponent` του Agent και γίνεται αληθής όταν ο Agent δεν βρίσκεται στο έδαφος. Σε αυτή την περίπτωση, το node `BindEventToLandedDelegate` επιτρέπει τη συνέχιση του κώδικα μόλις ο παίκτης προσγειωθεί στο έδαφος, χρησιμοποιώντας το event dispatcher που παρέχει το component. Το task συνεχίζει με την κλήση του `AttackTest`, η οποία εκτελεί την πυροδότηση του όπλου προς τον παίκτη. Σε αυτό το σημείο, χρησιμοποιείται ένα ακόμη Event Dispatcher που καλείται κατά την ολοκλήρωση του `AttackTest`, ώστε το task να ολοκληρωθεί την κατάλληλη στιγμή. Ομοίως με τις υλοποιήσεις που πραγματοποιήθηκαν για την όπλιση και αφόπλιση, το Event Dispatcher διασφαλίζει την ορθή εκτέλεση των task του sequence, καθώς η διαδικασία πυροδότησης απαιτεί την πάροδο συγκεκριμένου χρόνου για να ολοκληρωθεί. Με αυτό τον τρόπο το task θα ολοκληρωθεί επιτυχώς χωρίς να προκληθεί μια ανεπιθύμητη συμπεριφορά, όπως την πυροδότηση του όπλου ενώ ο Agent έχει μεταβεί ακαριαία στο sequence `Evade` ή `FindCoverAndHeal` μετά την κλήση του `AttackTest`.

AttackTest – Ranged (Function): Το function εκτελεί την ενέργεια επίθεσης ενός Agent σύμφωνα με τον τύπο του. Σε κάθε κλήση, θέτει τη μεταβλητή `IsEnemyAttacking` (Boolean) ως `true`, η οποία σηματοδοτεί πως ο Agent βρίσκεται σε κατάσταση επίθεσης και χρησιμοποιείται σε σημεία ελέγχου διαφόρων ενεργειών. Ωστόσο οι υπόλοιπες ενέργειες του function αναπτύσσονται σε κάθε child blueprint λόγω της μεταξύ τους διαφοροποίησης. Με αυτό τον τρόπο υλοποιήθηκαν δύο πανομοιότυπες επεκτάσεις των ενεργειών για τους Agents τύπου `Pistol` και `Rifle` στα blueprint `BP_Energy_Pistol` και `BP_Energy_Rifle` αντίστοιχα, καθώς και μια τρίτη στο `BP_Energy_Android` για τον τύπο των `Android`. Κάθε function στα child blueprints χρησιμοποιεί το node `Parent:AttackTest`, ώστε οι ενέργειες που περιλαμβάνονται στο function του parent blueprint να εκτελεστούν, χωρίς να αντικατασταθούν εξ ολοκλήρου από την υλοποίηση της επέκτασης. Το function `AttackTest` για κάθε Ranged Agent πραγματοποιεί την κλήση του function `Fire`, το οποίο υλοποιείται στο blueprint του οπλισμού και εκτελεί τις ενέργειες πυροδότησης ενός όπλου. Καθώς το function `Fire` απαιτεί ένα συγκεκριμένο χρονικό διάστημα για να ολοκληρωθεί, και το διάστημα αυτό παρουσιάζει διακυμάνσεις σε κάθε κλήση του function, η ολοκλήρωση της πυροδότησης σηματοδοτείται με την κλήση του Event Dispatcher `FireCompleted`. Η κλήση του πυροδοτεί το Custom Event `FireCompleted`, το οποίο με τη σειρά του εκτελεί το `AttackEnd`. Με αυτόν τον τρόπο διασφαλίζεται πως το task της επίθεσης `BTT_Ranged_Attack` θα ολοκληρωθεί μόνο όταν εκτελεστούν όλες οι ενέργειες που περιλαμβάνει το `Fire`, και το sequence θα προχωρήσει στην επόμενη ενέργεια την κατάλληλη στιγμή.



Εικόνα 102. AttackTest – Ranged

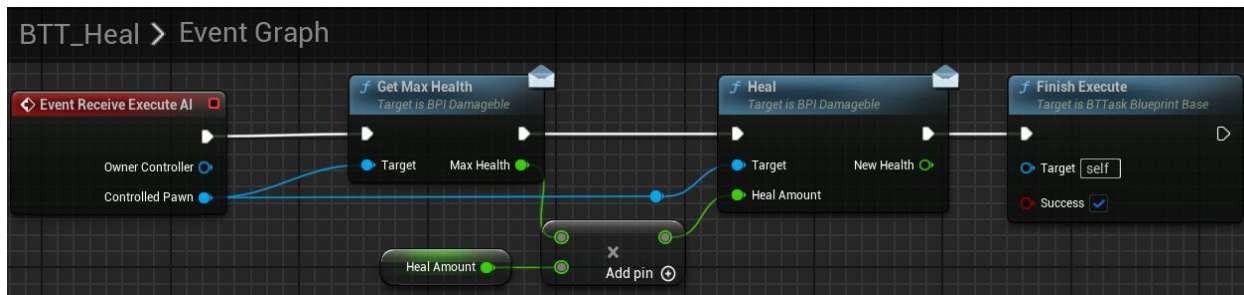
BTD_IsHealthLow (Decorator): Ο συγκεκριμένος Decorator ελέγχει τους πόντους ζωής του Agent, ώστε οι ενέργειες που περιλαμβάνει το sequence FindCoverAndHeal να εκτελεστούν κατά την αξιολόγηση στο επίπεδο του Selector AttackingState. Περιλαμβάνει τη μεταβλητή HealthThreshold, που αντιπροσωπεύει την τιμή από την οποία εξαρτάται η εκτέλεση του sequence. Ακόμη, οι πόντοι ζωής του Agent ανακτώνται μέσω των blueprint interface functions GetMaxHealth και GetCurrentHealth και μετατρέπονται σε ποσοστό μέσω της μεταξύ τους διαίρεσης. Στη συνέχεια, ελέγχεται αν το αποτέλεσμα είναι μικρότερο ή ίσο με την τιμή της HealthThreshold, γεγονός που θα καταστήσει τον Decorator αληθή. Σε αυτό το project η τιμή της επιλέχθηκε να είναι 0.3, που σημαίνει πως ένας Agent θα αναπληρώσει τους πόντους ζωής του όταν αυτοί βρεθούν από 30 και κάτω, από το σύνολο των 100.



Εικόνα 103. BTD_IsHealthLow

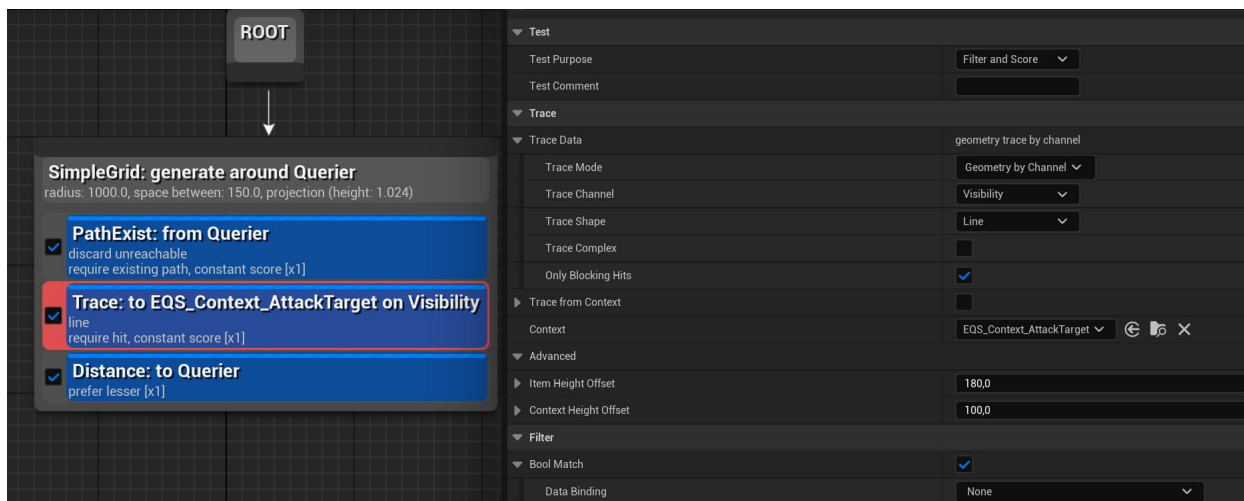
BTT_Heal (Custom Event): Το task είναι υπεύθυνο για την αναπλήρωση των πόντων ζωής ενός Agent, και καλείται κατά την εκτέλεση του sequence FindCoverAndHeal. Δέχεται ως είσοδο τη μεταβλητή HealAmount, η οποία αντιπροσωπεύει το ποσό της αναπλήρωσης των πόντων ζωής. Η μεταβλητή δημιουργήθηκε εντός του task και χρησιμοποιεί την επιλογή Instance Editable, προκειμένου να μπορεί να οριστεί σε κάθε κλήση του task. Η υλοποίηση ανακτά την μέγιστη τιμή του Agent μέσω του GetMaxHealth και τον πολλαπλασιασμό της με το HealAmount. Το αποτέλεσμα μετατρέπει το εκχωρημένο ποσοστό της τιμής σε πόντους ζωής προς αναπλήρωση, ώστε να χρησιμοποιηθούν ως

είσοδος στο blueprint interface function Heal, το οποίο εκτελείται στο component BPC_DamageSystem του Agent.



Εικόνα 104. BTT_Heal

EQS_FindCover (RunEQSQuery): Το Query αναζητά στο περιβάλλον το σημείο που βρίσκεται στην κοντινότερη απόσταση από τον Agent και δεν είναι ορατό από τον παίκτη. Χρησιμοποιείται κατά την ενέργεια Heal ενός Agent μέσω του Run EQS Query task, προκειμένου ο Agent να αναζητήσει ένα ασφαλές σημείο για να αναπληρώσει τους πόντους ζωής του. Ωστόσο μπορεί να χρησιμοποιηθεί ως μέρος της ενέργειας της επίθεσης ενός Agent, κατά την οποία πυροβολεί τον παίκτη και καλύπτεται για μικρό χρονικό διάστημα.

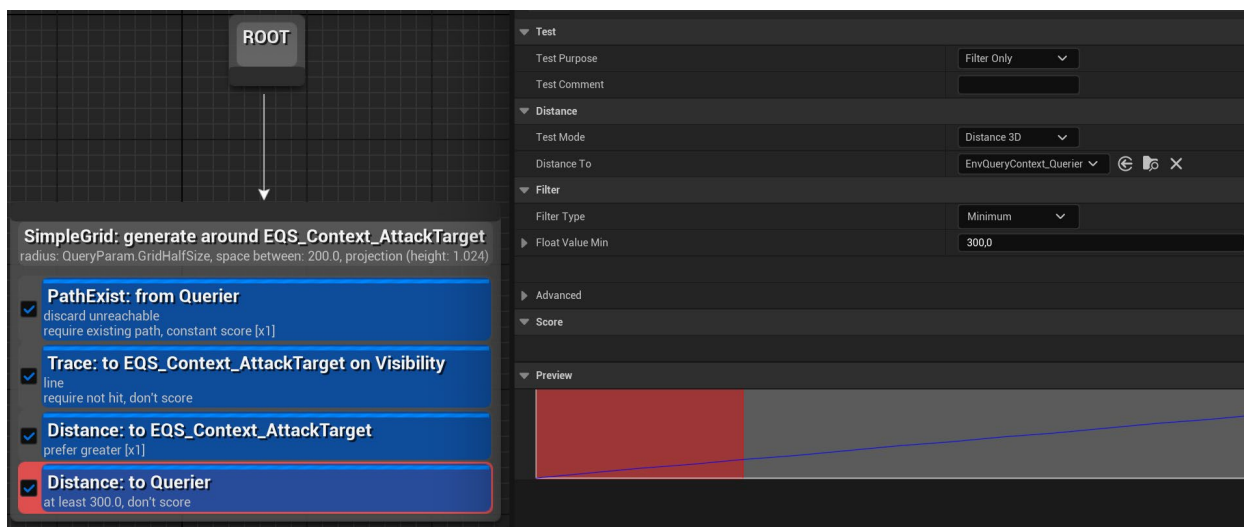


Εικόνα 105. EQS_FindCover

Αποτελείται από ένα SimpleGrid για τη δημιουργία των πιθανών σημείων, ένα pathfinding test τύπου PathExist: from Querier για την απόρριψη των απροσπέλαστων σημείων και ένα Trace test για την εύρεση των σημείων που δεν είναι ορατά από τον παίκτη. Το Trace test χρησιμοποιεί το EQS_Context_AttackTarget ως context, διασφαλίζοντας ότι τα επιλεγμένα σημεία δεν είναι ορατά από

τον παίκτη. Επειδή το Trace χρησιμοποιεί ένα LineTracebyChannel, οι τιμές των Item Height Offset και Context Height Offset ρυθμίστηκαν έτσι ώστε η επιλογή να μην επηρεάζεται από ανηφορικά ή κατηφορικά εδάφη. Συγκεκριμένα, η τιμή 100 εξασφαλίζει ότι το trace θα εκτελείται από το ύψος της μέσης του παίκτη, επιτρέποντας τη σωστή επιλογή σημείων που παρέχουν κάλυψη στον Agent. Το task χρησιμοποιεί την επιλογή Single Best Item, ώστε το αποτέλεσμα να είναι το σημείο με τη μεγαλύτερη βαθμολογία.

EQS_FindIdealRngedLocation (RunEQSQuery): Το Query αναζητά στο περιβάλλον το σημείο στο οποίο ο Agent θα κινηθεί, κατά τη διάρκεια των επιθετικών ενεργειών του. Καλείται κατά τη διάρκεια της επίθεσης και της εκτέλεσης ελιγμού, εντός των sequence Attack και Evade αντίστοιχα. Σύμφωνα με τις ανάγκες του project, το σημείο που θα επιλεγεί κατά τη διάρκεια των παραπάνω ενεργειών επιλέγεται με το εξής κριτήριο. ο Agent πρέπει να βρίσκει ένα σημείο που θα είναι μακριά από τον παίκτη ώστε προστατευτεί, αλλά ταυτόχρονα και κοντά ώστε να μπορεί να τον στοχεύσει με επιτυχία.



Εικόνα 106. EQS_FindIdealRngedLocation

Αποτελείται από ένα SimpleGrid για τη δημιουργία των πιθανών σημείων γύρω από τον παίκτη και ένα pathfinding test για την απόρριψη των απροσπέλαστων σημείων. Ακόμη, περιλαμβάνει ένα Trace με context τον παίκτη, ώστε το σημείο που θα επιλεγεί να βρίσκεται εντός του οπτικού πεδίου του Agent, προκειμένου να μπορεί να επιτεθεί. Επιπλέον, το Distance test με context τον παίκτη και τη ρύθμιση prefer greater, χρησιμοποιήθηκε για την ανάθεση μεγαλύτερης βαθμολογίας στα σημεία που βρίσκονται μακριά από τον παίκτη, καθιστώντας τα πιο πιθανά προς επιλογή. Τέλος, το Distance test με context τον Agent διασφαλίζει πως το σημείο που θα επιλεγεί απέχει μια συγκεκριμένη απόσταση από την τρέχουσα τοποθεσία του Agent, ώστε να αποκλειστούν τα κοντινά σημεία και η μετακίνησή του να έχει μια απόσταση. Το task χρησιμοποιεί την επιλογή Single Random Item from Best 25% ώστε το σημείο που θα επιλεγεί σε κάθε κλήση να είναι διαφορετικό, καθώς το Query επαναλαμβάνεται συνεχώς.

Για να καταστεί εφικτή η παραγωγή σημείων με δυναμικό μέγεθος, χρησιμοποιήθηκε η ρύθμιση Data Binding στο GridHalfSize, η οποία επιτρέπει την εκχώρηση τιμής κατά την κλήση του task εντός του Behavior Tree. Συγκεκριμένα, η παράμετρος του μεγέθους παραγωγής σημείων μπορεί να πάρει την τιμή που έχει ένα blackboard key. Στην προκειμένη περίπτωση, δημιουργήθηκε το key AttackRadius (float), το οποίο παίρνει την τιμή του από τη μεταβλητή AttackRadius που βρίσκεται στο parent blueprint του Agent. Η μεταβλητή μπορεί να πάρει διαφορετική τιμή για κάθε Agent, με την ανάθεσή της στο child blueprint του καθενός. Η εκχώρηση του περιεχομένου της στο blackboard key λαμβάνει χώρα στο AI Controller, αφού ανακτηθεί από το blueprint του Agent μέσω της blueprint interface function Get Ideal Range. Με αυτόν τον τρόπο, ο καθορισμός του ιδανικού σημείου προκύπτει σύμφωνα με την απόσταση επίθεσης που έχει οριστεί σε κάθε Agent, επιτρέποντας διαφοροποίηση των κινήσεων της επίθεσης, σύμφωνα με παραμέτρους όπως τον τύπο του Agent, τον οπλισμό του, την κατάσταση του παιχνιδιού, τη δυσκολία του επιπέδου κ.α. Ακολουθώντας την ίδια λογική, δημιουργήθηκε η μεταβλητή Defend Radius και το ομώνυμο blackboard key, ώστε να χρησιμοποιηθούν σε αμυντικά task, όπου ο Agent πρέπει να κινηθεί σε ένα σημείο ασφαλής απόστασης από τον παίκτη. Ωστόσο, στο συγκεκριμένο project δεν αναπτύχθηκαν ενέργειες που περιλαμβάνουν αμυντικές κινήσεις.

8.4.3. Android Attacking State

Το Attacking State των εχθρών τύπου Android παρουσιάζει διαφοροποιήσεις από τους Agent τύπου Ranged τόσο στις ενέργειες που περιλαμβάνει όσο και στις πιθανές μεταβάσεις των states που προκύπτουν, σύμφωνα με τις ενέργειες του παίκτη. Όσον αφορά στις ενέργειες ενός Android, ο Agent επιτίθεται στον παίκτη με την ενέργεια Aerial Attack, μια επίθεση κοντινής εμβέλειας κατά την οποία εκτελεί ένα άλμα προς την κατεύθυνση του παίκτη, κάνει spawn ένα σπαθί και προκαλεί damage. Ακόμη, η κινησιολογία του Agent περιλαμβάνει την κίνηση Teleport, κατά την οποία ο Agent τηλεμεταφέρεται σε ένα συγκεκριμένο σημείο σύμφωνα με τις ανάγκες της ενέργειας που εκτελεί. Επιπλέον, ο Agent δε μπορεί να εκτελέσει την ενέργεια Heal, όπως συμβαίνει με τους Agents τύπου Ranged. Τέλος, ένα Android δεν μπορεί να δεχθεί damage από τον παίκτη με τη χρήση οπλισμού, απαιτώντας ιδιαίτερη προσέγγιση για την εξουδετέρωσή του.

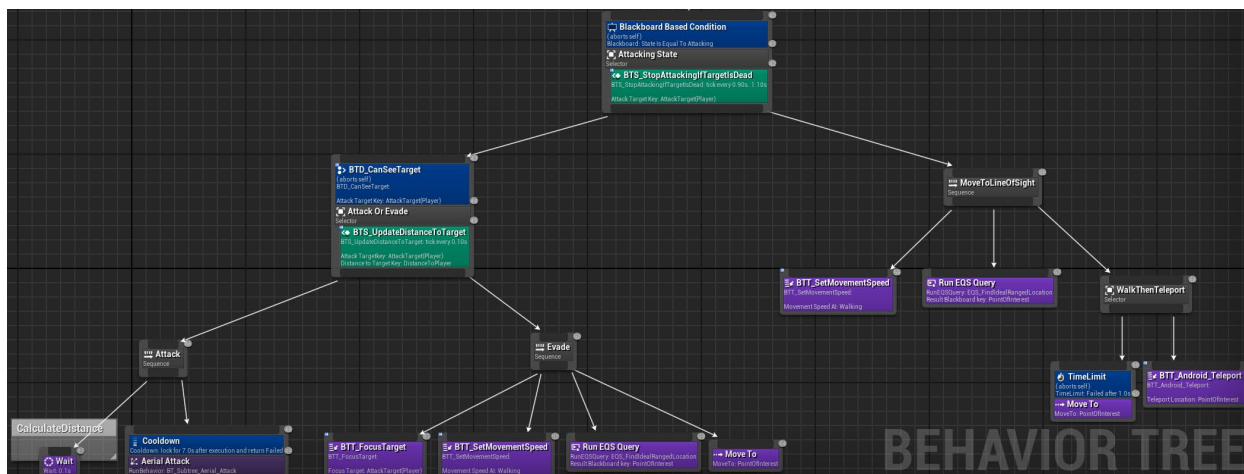
Συγκεκριμένα, ο παίκτης πρέπει να μεταβάλλει τον Agent σε κατάσταση κατά την οποία η ενέργεια Takedown μπορεί να πραγματοποιηθεί. Όπως αναφέρεται και στο σχετικό κεφάλαιο, ένα Takedown μπορεί να πραγματοποιηθεί αν ο Agent δεν είναι ενήμερος για την παρουσία του παίκτη, γεγονός που αντιπροσωπεύεται από τη μεταβλητή IsAware (Boolean), η οποία χρησιμοποιείται ως συνθήκη για την εκτέλεση της ενέργειας. Η μεταβλητή έχει την τιμή false όταν ο Agent βρίσκεται στα states Passive, Investigating, Detecting και Seeking ή μπορεί να πάρει την τιμή αυτή, όταν ο Agent βρίσκεται στα states Attacking ή Frozen και ο παίκτης χρησιμοποιήσει μία από τις δύο μηχανικές αντιμετώπισης των Android που αναπτύχθηκαν στο παιχνίδι. Η πρώτη αφορά στη χρήση του αντικειμένου DataWorm, κατά το οποίο ο Agent αδρανοποιείται μεταβαίνοντας στο state Frozen, επιτρέποντας έτσι την ενέργεια Takedown. Η δεύτερη αφορά στην αποφυγή της επίθεσης Aerial Attack τη σωστή στιγμή, δηλαδή εντός ενός μικρού χρονικού περιθωρίου κατά τη διάρκειά της. Η αποφυγή πραγματοποιείται με την εκτέλεση της ενέργειας Dodge και ο έλεγχος της επιτυχίας της προκύπτει σύμφωνα με το σύστημα EQS το οποίο αναπτύχθηκε.

Με αυτόν τον τρόπο, ο παίκτης μπορεί να δημιουργήσει ένα μικρό χρονικό "παράθυρο" στο οποίο ο Agent βρίσκεται ακινητοποιημένος και η ενέργεια Takedown είναι εφικτή. Επιπλέον, όταν ο παίκτης βρεθεί εκτός του οπτικού πεδίου του Agent, αυτός θα τηλεμεταφερθεί σε σημείο που ο παίκτης είναι ορατός, καθιστώντας δύσκολο σε έναν παίκτη να αποφύγει τις επιθέσεις του. Τέλος, ένα Android διαφέρει στους ήχους και τα οπτικά εφέ που παράγονται κατά τις ενέργειες κίνησης, επίθεσης, Takedown κ.α.

Τα πιθανά states στα οποία ένα Android μπορεί να μεταβεί από το Attacking state, περιλαμβάνουν το state Seeking (όταν χάσει τον παίκτη από το οπτικό του πεδίο), το state Dead (όταν ο παίκτης τον εξουδετερώσει), το state Passive (όταν ο Agent εξουδετερώσει τον παίκτη) και το state Frozen (όταν ο παίκτης χρησιμοποιήσει επιτυχημένα το αντικείμενο DataWorm).

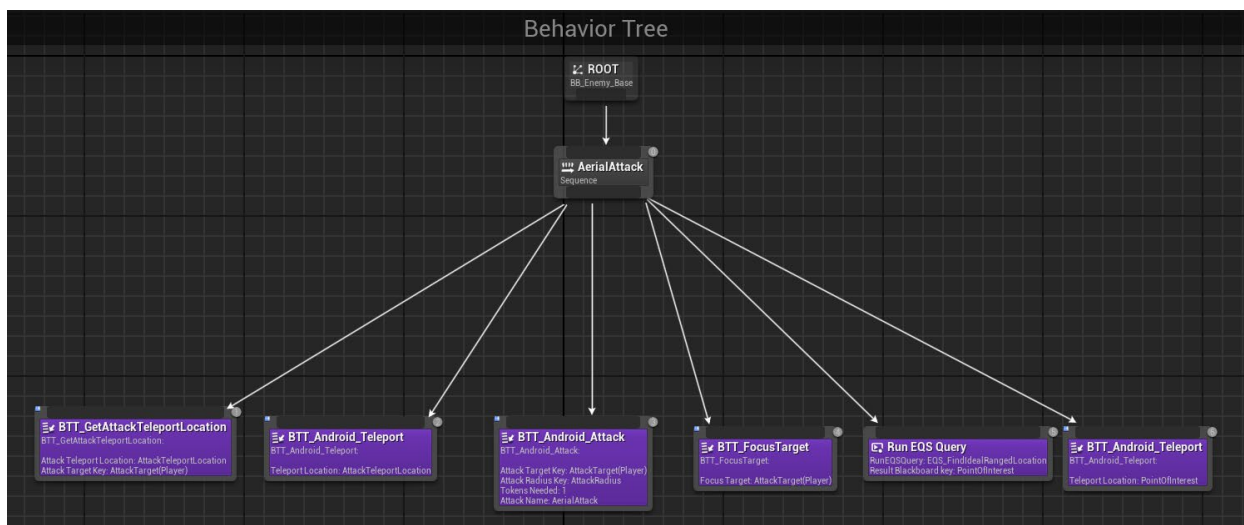
Σε αυτή την υποενότητα περιγράφεται η υλοποίηση του δέντρου συμπεριφοράς και αναλύονται οι υλοποιήσεις της τηλεμεταφοράς, της επίθεσης Aerial Attack, της κίνησης αποφυγής Dodge και των συνθηκών που καθιστούν ένα Android ευάλωτο στην επίθεση Takedown.

Attacking Behavior Tree Overview (Android): Ακολουθώντας δομή ίδια με αυτή του δέντρου των Agents τύπου Ranged, το δέντρο ενεργοποιείται μέσω Blackboard Based Condition, και ο Selector AttackingState διαθέτει ένα service που σταματά το State αν ο παίκτης εξουδετερωθεί. Το δέντρο ακολουθείται από τον selector Attack or Evade, και το Sequence MoveToLineOfSight. Με αυτόν τον τρόπο, ο Agent εκτελεί τις ενέργειες επίθεσης και ελιγμών, αν ο παίκτης βρίσκεται στο οπτικό του πεδίο σύμφωνα με τον Decorator BTD_CanSeeTarget. Σε αντίθετη περίπτωση, ο Agent τηλεμεταφέρεται προς ένα σημείο που ο παίκτης είναι ορατός. Ο selector επιλέγει ανάμεσα στο Sequence Attack ή Evade, με το δεύτερο να εκτελείται όταν η επίθεση βρίσκεται σε Cooldown.



Εικόνα 107. Attacking Behavior Tree Overview

Το Attack περιλαμβάνει την επίθεση Aerial Attack, η οποία εκτελείται μέσω του υποδέντρου BT_Subtree_Aerial_Attack. Ένα επιτιθέμενο Android, τηλεμεταφέρεται σε ένα ιδανικό σημείο επίθεσης μέσω του BTT_Android_Teleport. Το σημείο υπολογίζεται σύμφωνα με την κίνηση του παίκτη, μέσω του task BTT_GetAttackTeleportLocation, ώστε η επίθεση να ξεκινήσει από μια συγκεκριμένη απόσταση μακριά από αυτόν. Η επίθεση πυροδοτείται μέσω του task BTT_Android_Attack, κατά την οποία ο Agent εκτελεί ένα άλμα προς την κατεύθυνση του παίκτη, προκειμένου να βρεθεί σε κοντινή απόσταση και να προκαλέσει damage. Μετά την ολοκλήρωση της επίθεσης, ο Agent θα εστιάσει στον παίκτη και θα τηλεμεταφερθεί σε ένα μακρινό σημείο, σύμφωνα με το αποτέλεσμα του task Run EQS Query. Ως αποτέλεσμα, ο Agent επιστρέφει σε θέση απόστασης από τον παίκτη, συνεχίζοντας με την εκτέλεση του επόμενου sequence. Το Evade αφορά στην κίνηση που εκτελεί ο Agent ανάμεσα σε δύο επιθέσεις. Σε αυτό το project, το Android επιλέχθηκε να κινείται με αργή ταχύτητα προς ένα ιδανικό σημείο, διατηρώντας τον παίκτη στο οπτικό του πεδίο, καθώς αποτελεί δύσκολο στόχο εξόντωσης και απαιτεί ειδικές ενέργειες αντιμετώπισης. Το Sequence περιλαμβάνει τον ορισμό του παίκτη ως σημείο εστίασης (BTT_FocusTarget), τον ορισμό της ταχύτητας σε Walking (BTT_SetMovementSpeed), την εύρεση του ιδανικού σημείου μετακίνησης (Run EQS Query), και την ενέργεια μετακίνησης προς το σημείο αυτό (MoveTo).



Εικόνα 108. BT_Subtree_Aerial_Attack

Το Sequence MoveToLineOfSight ενεργοποιείται μόλις ο παίκτης εξέλθει από το οπτικό πεδίο του Agent, δηλαδή στην περίπτωση που ο Decorator BTD_CanSeeTarget γίνει false και προκαλέσει επαναξιολόγηση του δέντρου στο επίπεδο του selector AttackingState. Περιλαμβάνει τον ορισμό της ταχύτητας σε Walking (BTT_SetMovementSpeed), την εύρεση του σημείου μετακίνησης και ένα selector που θα εκτελέσει την μετακίνηση. Ο selector χρησιμοποιεί το task MoveTo που κάνει τον Agent να περπατήσει προς το σημείο και το task BTT_Android_Teleport το οποίο τηλεμεταφέρει τον Agent στο σημείο. Η χρήση του selector και η προσθήκη του decorator TimeLimit στο task MoveTo, έχουν ως αποτέλεσμα τον Agent να κινηθεί προς το σημείο χρησιμοποιώντας κανονική κινησιολογία και την

τηλεμεταφορά του μετά από ένα μικρό χρονικό διάστημα. Με αυτόν τον τρόπο, το task MoveTo πάντα θα αποτυγχάνει μετά την πάροδο του χρόνου, ενεργοποιώντας το task της τηλεμεταφοράς. Αυτό προσφέρει μια ομαλότερη αντίδραση του Agent στην κινησιολογία του, όταν ο παίκτης βρεθεί εκτός του οπτικού του πεδίου.

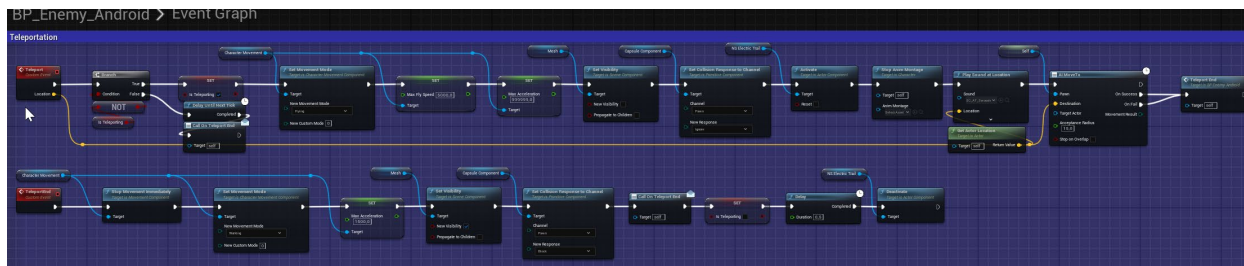
Teleport και TeleportEnd (Custom Event): Τα συγκεκριμένα Custom Events εκτελούν την τηλεμεταφορά ενός Android σε ένα σημείο και ειδοποιούν το task για την ολοκλήρωση της διαδικασίας. Το πρώτο αναλαμβάνει τις αρχικές ενέργειες της λειτουργίας, ενώ το δεύτερο επαναφέρει τις απαραίτητες ρυθμίσεις και μεταβλητές στην αρχική τους κατάσταση. Σύμφωνα με τον σχεδιασμό του project, η υλοποίηση της τηλεμεταφοράς ξεκινά με την αναπαραγωγή ενός ειδικού εφέ, το οποίο σηματοδοτεί την έναρξή της και διαρκεί καθ' όλη τη διάρκεια της διαδικασίας. Παράλληλα, ο Agent καθίσταται αόρατος. Στη συνέχεια, ο Agent κινείται προς το καθορισμένο σημείο με ταχύτητα που είναι αρκετά υψηλή ώστε να γίνεται αντιληπτό ότι πραγματοποιείται τηλεμεταφορά, αλλά και αρκετά αργή ώστε ο παίκτης να μπορεί να παρακολουθήσει την κίνησή του. Ο παίκτης βλέπει το ειδικό εφέ γύρω από το σώμα του Agent μέχρι εκείνος να φτάσει στο σημείο προορισμού. Όταν ο Agent φτάσει στον προορισμό, το εφέ σταματά και ο Agent καθίσταται και πάλι ορατός. Επιπλέον, η διαδικασία περιλαμβάνει έναν ήχο που αναπαράγεται από την έναρξη και καθ' όλη τη διάρκεια της τηλεμεταφοράς. Το διάγραμμα που περιγράφει τη διαδικασία τηλεμεταφοράς και η οπτική αναπαράσταση της όπως εμφανίζεται εντός του παιχνιδιού παρουσιάζονται στις εικόνες 110 και 109 αντίστοιχα.



Εικόνα 109. Teleport Demonstration

Μέσω του CharacterMovementComponent, ο τρόπος κίνησης του Agent μεταβάλλεται από Walking σε Flying (Set Movement Mode) και οι μεταβλητές της ταχύτητας (Max Fly Speed και Max Acceleration)

ορίζονται στις κατάλληλες τιμές, προκειμένου ο Agent να μετακινηθεί στο χώρο. Η επιλογή του Flying ως τρόπος κίνησης, οφείλεται στο γεγονός πως το mode μπορεί να κινήσει τον Agent χωρίς την επιρροή παραγόντων όπως την τριβή του εδάφους, τη βαρύτητα κ.α., όπως γίνεται στο mode Walking. Έπειτα, το mesh γίνεται αόρατο (Set Visibility) και η σύγκρουση του Agent με κάποιον Actor απενεργοποιείται (Set Collision Response to Channel), ώστε η πορεία τηλεμεταφοράς να μην επηρεαστεί από κάποιο εμπόδιο. Στη συνέχεια, το σύστημα Niagara NS_ElectricTrail ενεργοποιείται, αναπαράγοντας το ειδικό εφέ τηλεμεταφοράς. Το event συνεχίζει με τη διακοπή οποιουδήποτε Animation που εκτελείται τη δεδομένη στιγμή (Stop Anim Montage) και αναπαράγει τον ήχο τηλεμεταφοράς στην τοποθεσία του Agent (Play Sound At Location). Τέλος, για την εκτέλεση της μετακίνησης χρησιμοποιείται το node AIMoveTo, το οποίο δέχεται ως είσοδο τον προορισμό της μετακίνησης και λειτουργεί με τρόπο παρόμοιο του task MoveTo. Ο προορισμός εισάγεται από τη μεταβλητή εισόδου του function, η οποία προκύπτει από το EQS Query. Η επιτυχημένη ή αποτυχημένη μετακίνηση του Agent στο σημείο, θα πυροδοτήσει την έναρξη του custom event TeleportEnd, μέσω των execution pins OnSuccess και OnFail.

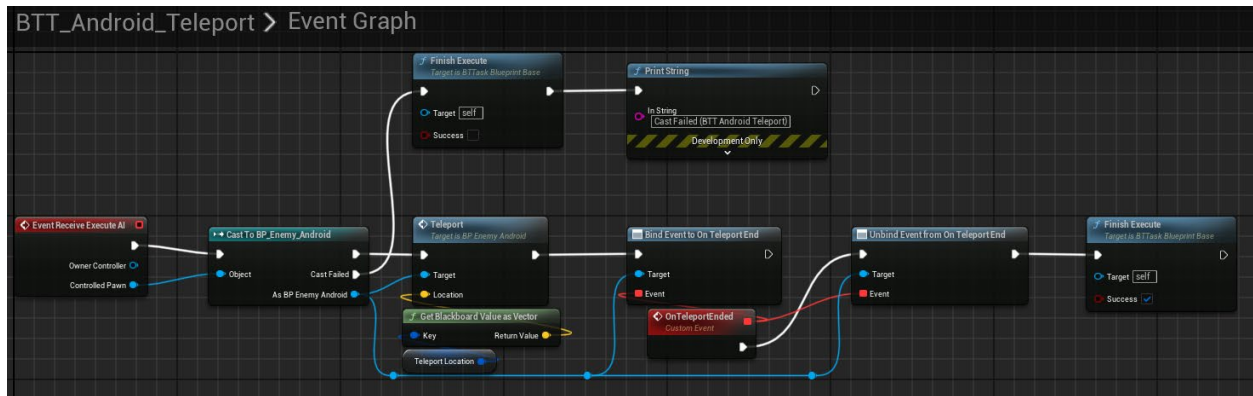


Εικόνα 110. Teleport & TeleportEnd

Το event TeleportEnd σταματά αμέσως τον Agent στο σημείο που βρίσκεται (Stop Movement Immediately) και το mode κίνησης, οι μεταβλητές ταχύτητας, η ορατότητα του mesh και οι ρυθμίσεις σύγκρουσης επανέρχονται στην αρχική τους κατάσταση. Έπειτα, το Event Dispatcher Teleport End καλείται, ώστε να ολοκληρωθεί το task. Το ειδικό εφέ απενεργοποιείται, μετά από μια χρονική καθυστέρηση 0.5 δευτερολέπτων. Τελευταία προσθήκη της υλοποίησης αποτελεί η μεταβλητή IsTeleporting, η οποία παίρνει την τιμή true κατά την έναρξη της τηλεμεταφοράς, και false κατά την ολοκλήρωσή της. Χρησιμοποιείται ως συνθήκη ελέγχου, προκειμένου να διασφαλιστεί πως συγκεκριμένες ενέργειες δε θα πραγματοποιηθούν κατά τη διάρκεια της τηλεμεταφοράς, συμπεριλαμβανομένης της ίδιας της ενέργειας. Τέλος, η χρήση του Event Dispatcher On Teleport End κατά τον αρχικό έλεγχο που εξετάζει αν ο Agent εκτελεί το Teleport, εξασφαλίζει πως το task θα ολοκληρωθεί σε κάθε περίπτωση και το Behavior Tree θα προχωρήσει στην επόμενη ενέργεια. Αυτό μπορεί να συμβεί στην περίπτωση επανάκλησης του Event, ενώ αυτό εκτελείται ήδη.

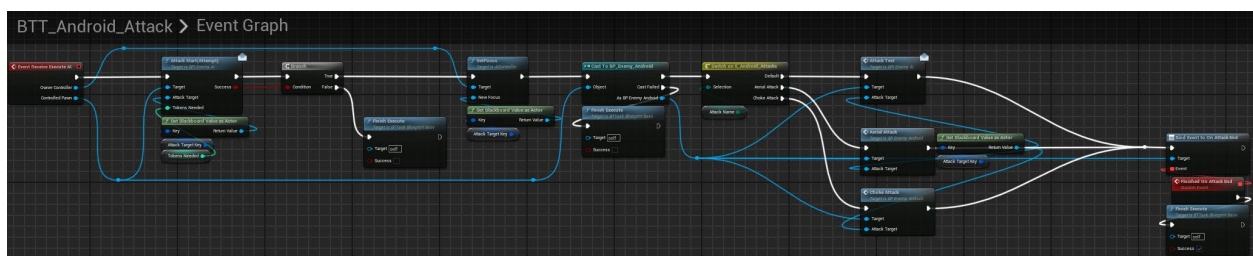
BTT_AndroidTeleport (Custom Task): Το task είναι υπεύθυνο για την πυροδότηση του custom event Teleport. Χρησιμοποιείται εντός των sequence Evade και AerialAttack. Δέχεται ως είσοδο τη μεταβλητή Teleport Location, η οποία παίρνει την τιμή της από το περιεχόμενο ενός blackboard key και εκχωρείται ως είσοδος στο Teleport. Το task χρησιμοποιεί το node CastTo BP_Enemy_Base για την πρόσβαση στις

λειτουργίες του Agent και καλεί το Teleport. Τέλος, το task δεσμεύει την ολοκλήρωσή του με την κλήση του Event Dispatcher OnTeleportEnd, ώστε το Behavior Tree να προχωρήσει στην επόμενη ενέργεια την κατάλληλη στιγμή.



Εικόνα 111. BTT_AndroidTeleport

BTT_Android_Attack (Custom Task): Το task είναι υπεύθυνο για την πυροδότηση της επίθεσης Aerial Attack. Δέχεται ως εισόδους τον στόχο της επίθεσης και τον αριθμό των Tokens που κοστίζει, μέσω των blackboard key selectors AttackTargetKey και TokensNeeded. Το task ελέγχει εάν υπάρχουν διαθέσιμα Tokens για την εκτέλεση της επίθεσης μέσω της AttackStart(Attempt) και αν δεν υπάρχουν, το task ολοκληρώνεται ανεπιτυχώς μέσω του Finish Execute (Success=False), προκαλώντας την αποτυχία του Attack sequence και την επαναξιολόγηση των κόμβων. Αν τα tokens είναι διαθέσιμα, ο παίκτης ορίζεται ως σημείο εστίασης μέσω του node SetFocus, και αποκτάται πρόσβαση στις λειτουργίες του Agent μέσω του CastTo BP_Enemy_Android. Στη συνέχεια, το function Aerial Attack πυροδοτεί την επίθεση και το task ολοκληρώνεται κατά την κλήση του Event Dispatcher On Attack End, όπως σε κάθε task που απαιτεί ακριβή συγχρονισμό μεταξύ της ολοκλήρωσης του task και της ενέργειας που εκτελεί.

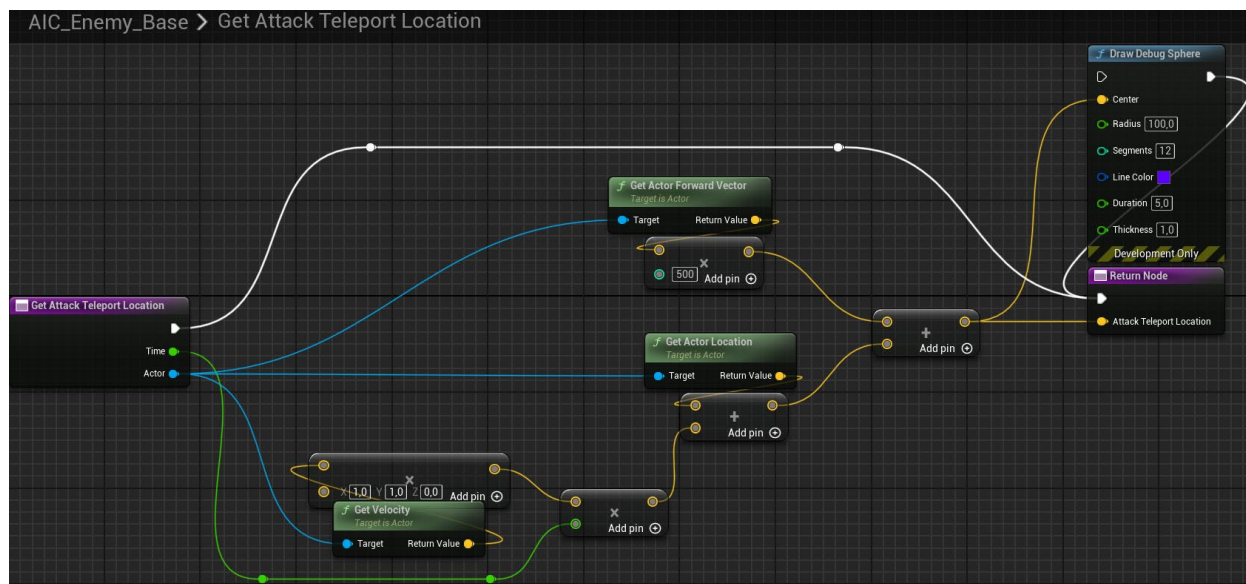


Εικόνα 112. BTT_Android_Attack

GetAttackTeleportLocation (Function): Το function είναι υπεύθυνο για τον υπολογισμό του σημείου στο οποίο θα τηλεμεταφερθεί ο Agent προκειμένου να εκτελέσει την επίθεση Aerial Attack, και υλοποιείται στο AI Controller. Δέχεται ως είσοδο το reference του παίκτη και τη μεταβλητή Time (float),

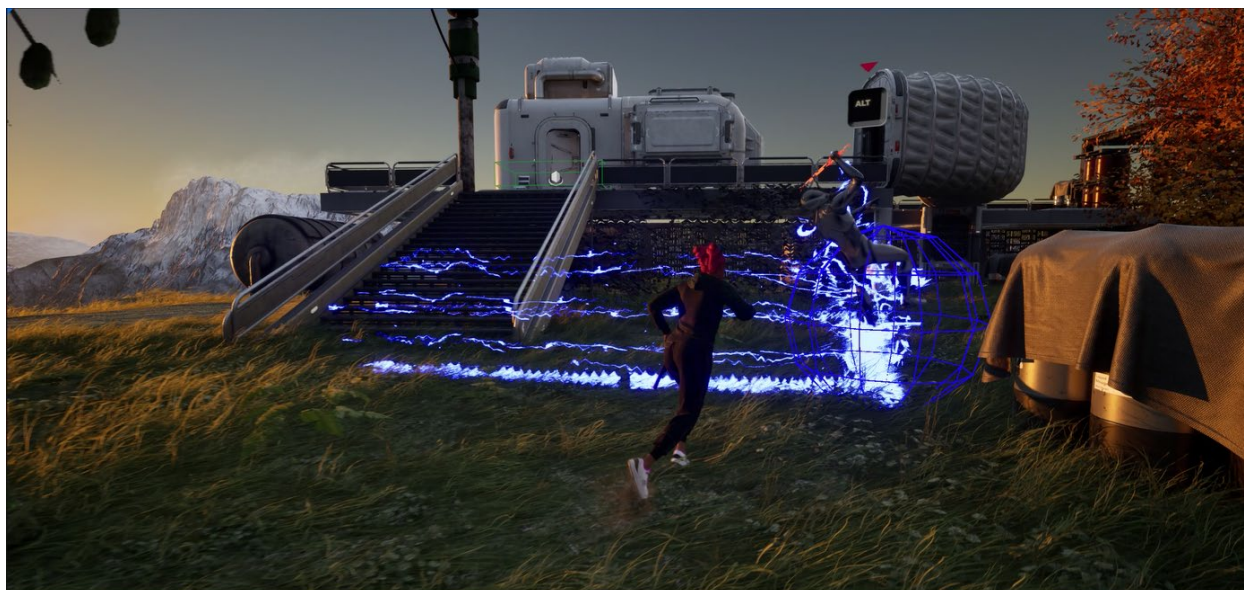
η οποία χρησιμοποιείται στον υπολογισμό του σημείου τηλεμεταφοράς, και επιστρέφει το σημείο μέσω της μεταβλητής Attack Teleport Location (Vector).

Η επίθεση περιλαμβάνει την εκτέλεση ενός άλματος συγκεκριμένου μήκους (5 μέτρα) προς την κατεύθυνση του παίκτη, έτσι ώστε ο Agent να βρεθεί σε κοντινή απόσταση από αυτόν. Ο υπολογισμός του σημείου περιλαμβάνει την εκτίμηση της θέσης του παίκτη στον χώρο και την επιλογή ενός σημείου εντός ακτίνας 5 μέτρων από τον παίκτη. Επιπλέον, λαμβάνεται υπόψη η συνεχής κίνηση του παίκτη.



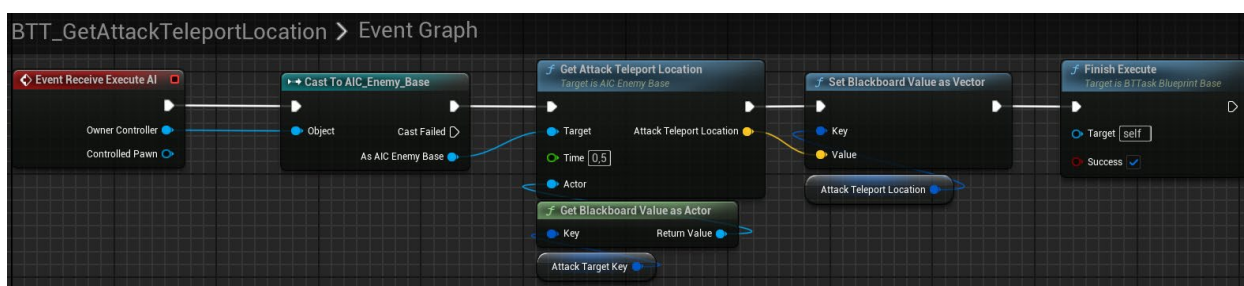
Εικόνα 113. GetAttackTeleportLocation

Ως εκ τούτου, το σημείο πρέπει να υπολογίζεται με βάση τη μελλοντική θέση του παίκτη, κάτι που επιτυγχάνεται μέσω της συμπερίληψης της ταχύτητας του στον υπολογισμό. Ο υπολογισμός της μελλοντικής θέσης πραγματοποιείται προσθέτοντας το διάνυσμα τοποθεσίας του παίκτη (GetActorLocation) και το γινόμενο της ταχύτητας (GetVelocity) με το χρόνο Time. Για την ομαλοποίηση της ταχύτητας, ο άξονας Z αφαιρείται. Στη συνέχεια, το αποτέλεσμα προστίθεται στο διάνυσμα της μπροστά κατεύθυνσης του παίκτη (GetActorForwardVector), το οποίο έχει πολλαπλασιαστεί με τον αριθμό 500, έτσι ώστε το υπολογισθέν σημείο να βρίσκεται εντός ακτίνας 5 μέτρων από τον παίκτη. Το τελικό αποτέλεσμα επιστρέφεται από τη συνάρτηση μέσω της μεταβλητής Attack Teleport Location. Η εικόνα 114 απεικονίζει το υπολογισμένο σημείο μέσω μιας σφαίρας Debugging, όπου ο Agent τηλεμεταφέρεται για την εκτέλεση της επίθεσης.



Εικόνα 114. Teleport Location Visualization

BTT_GetAttackTeleportLocation (Custom Task): Το task είναι υπεύθυνο για την πυροδότηση του function GetAttackTeleportLocation, κατά το οποίο υπολογίζεται το σημείο τηλεμεταφοράς ενός Agent που θα εκτελέσει την επίθεση AerialAttack. Ακόμη, αποθηκεύει το αποτέλεσμα που επιστρέφει το function σε ένα blackboard key, ώστε να χρησιμοποιηθεί ως είσοδος σε άλλα tasks. Δέχεται τις εισόδους AttackTargetKey και AttackTeleportLocation μέσω blackboard key selectors. Το task αποκτά πρόσβαση στις λειτουργίες του AI Controller μέσω του node CastTo AIC_Enemy_Base και καλεί το GetAttackTeleportLocation, με εισόδους το reference του παίκτη και τον χρόνο με τιμή 0.5. Το αποτέλεσμα αποθηκεύεται στο περιεχόμενο του blackboard key μέσω του Set Blackboard Value as Vector και το task ολοκληρώνεται.



Εικόνα 115. BTT_GetAttackTeleportLocation

Aerial Attack (Custom Event): Το συγκεκριμένο event υλοποιεί τις ενέργειες της επίθεσης Aerial Attack και είναι ενσωματωμένο στο child blueprint BP_Enemy_Android. Η επίθεση περιλαμβάνει την

αναπαραγωγή ενός animation που απεικονίζει την κίνηση ενός άλματος με πιρουέτα, το οποίο καταλήγει σε μια επίθεση με το σπαθί του Android όταν αυτός προσγειώνεται στο έδαφος. Κατά την έναρξη του animation, ο οπλισμός του Android (σπαθί κατάνα) καθίσταται ορατός σταδιακά μέσω ειδικών εφέ. Ακολουθώντας, ο χρόνος του παιχνιδιού επιβραδύνεται και ένα εικονίδιο UI εμφανίζεται πάνω από το Android, υποδεικνύοντας το κουμπί που πρέπει να πατήσει ο παίκτης για να αποφύγει την επίθεση. Στην πορεία του άλματος, λίγο πριν ο Agent εκτελέσει την επίθεση με το σπαθί, το εικονίδιο αλλάζει χρώμα και αναβοσβήνει, υποδεικνύοντας ότι η επίθεση είναι επικείμενη. Αυτή η χρονική στιγμή αντιπροσωπεύει το σημείο στο οποίο ο παίκτης έχει την ευκαιρία να εκτελέσει μια τέλεια αποφυγή της επίθεσης (Perfect Dodge) χρησιμοποιώντας την κίνηση αποφυγής Dodge. Η αποφυγή μπορεί να θεωρηθεί τέλεια, αν ο παίκτης ενεργήσει εντός ενός μικρού χρονικού περιθωρίου, πριν ο χρόνος παρέλθει και ο Agent ολοκληρώσει την επίθεση. Σε αυτό το σημείο, πραγματοποιείται έλεγχος για το αν ο παίκτης απέφυγε επιτυχώς την επίθεση, με τρία πιθανά σενάρια:

- Σενάριο 1: Ο παίκτης δεν αποφεύγει την επίθεση, με αποτέλεσμα να δεχτεί damage, και το Android τηλεμεταφέρεται άμεσα σε ένα σημείο απόστασης από τον παίκτη.
- Σενάριο 2: Ο παίκτης αποφεύγει την επίθεση πριν το σημείο της τέλει αποφυγής, αποφεύγοντας το damage, ενώ το Android τηλεμεταφέρεται άμεσα όπως και στο πρώτο σενάριο, χωρίς να βρεθεί ευάλωτος στην επίθεση Takedown.
- Σενάριο 3: Ο παίκτης αποφεύγει την επίθεση την κατάλληλη στιγμή (την τέλεια αποφυγή), χωρίς να δεχτεί damage και ακινητοποιεί το Android για λίγα δευτερόλεπτα, καθιστώντας το ευάλωτο στην επίθεση Takedown. Στο συγκεκριμένο σενάριο, αναπαράγεται ένα χαρακτηριστικό οπτικοακουστικό εφέ και το εικονίδιο UI μετατρέπεται σε αυτό που υποδεικνύει την ενέργεια Takedown και αναβοσβήνει. Με αυτό τον τρόπο, ο παίκτης λαμβάνει ανατροφοδότηση πως εκτέλεσε ένα Perfect Dodge. Ο παίκτης έχει τώρα ένα περιορισμένο χρονικό διάστημα για να πλησιάσει το Android και να το εξαλείψει. Εάν αυτό το διάστημα παρέλθει χωρίς ενέργεια από τον παίκτη, το Android τηλεμεταφέρεται, όπως στα σενάρια 1 και 2, και συνεχίζει τις επιθετικές του ενέργειες.

Η εικόνα 116 απεικονίζει την οπτική φάση της επίθεσης Aerial Attack, εστιάζοντας στο χρονικό σημείο κατά το οποίο εμφανίζεται η σχετική ένδειξη που υποδεικνύει στον παίκτη τη σωστή στιγμή για να εκτελέσει μια αποφυγή Perfect Dodge.

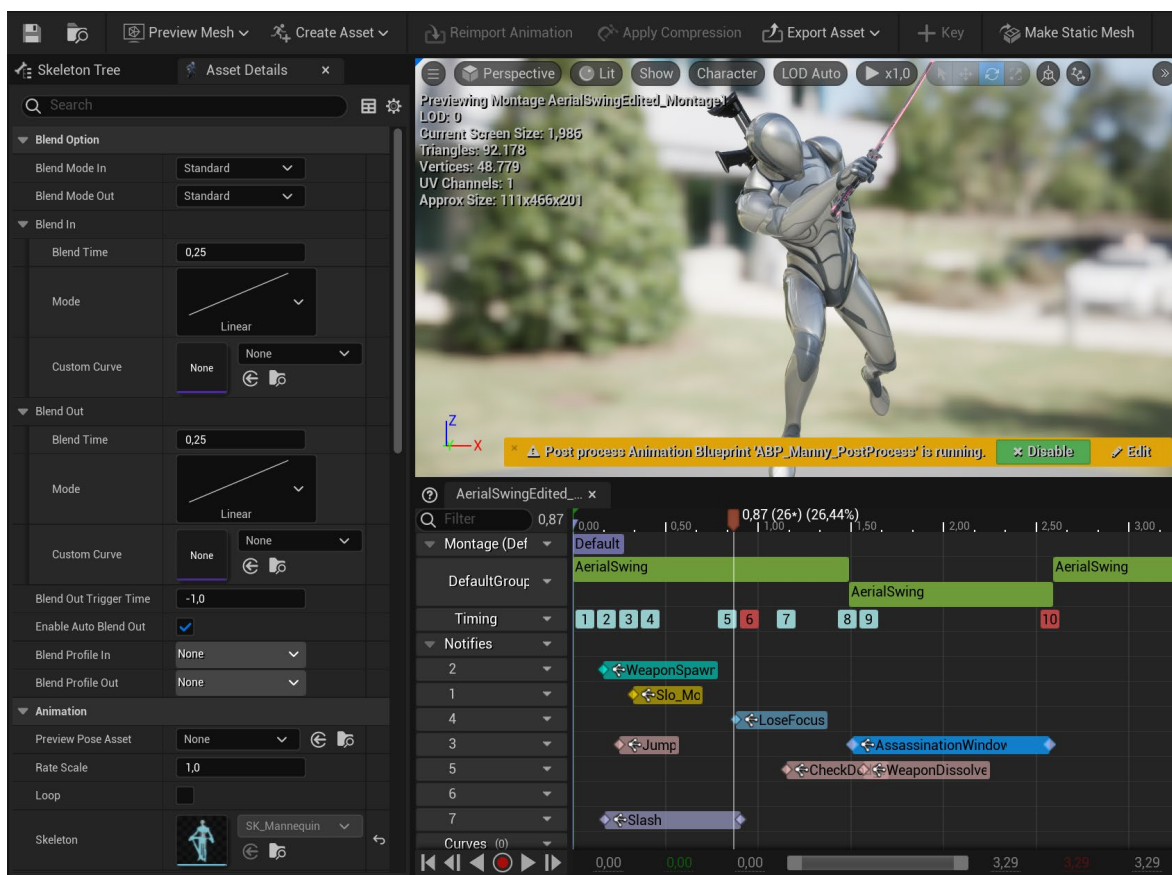


Εικόνα 116. Aerial Attack Demonstration

Το animation για την επίθεση Aerial Attack αναπαράγεται μέσω του node PlayMontage, το οποίο παρέχει τα απαραίτητα execution pins για την υλοποίηση. Το animation προς αναπαραγωγή επιλέγεται τυχαία από τρία διαθέσιμα animations, τα οποία έχουν παρόμοια χαρακτηριστικά και κινήσεις, αλλά διαφοροποιούνται σε ορισμένα σημεία για να παρέχουν ποικιλία στην κινησιολογία. Η διάρκεια του κάθε animation έχει τροποποιηθεί ώστε ο χρόνος από την ολοκλήρωση της επίθεσης έως την τηλεμεταφορά του Agent σε απόσταση από τον παίκτη να επιμηκυνθεί. Η επιμήκυνση του χρόνου αυτή αντιστοιχεί στο χρονικό περιθώριο που διαθέτει ο παίκτης για να εκτελέσει την επίθεση Takedown, εφόσον εκτελέσει ένα Perfect Dodge. Ως αποτέλεσμα, το animation αναπαράγεται πλήρως μόνο στο τρίτο σενάριο (όταν ο παίκτης εκτελεί το Perfect Dodge), επιτρέποντας την εξουδετέρωση του Agent μέσω Takedown. Στα υπόλοιπα σενάρια, το animation διακόπτεται νωρίτερα, ολοκληρώνοντας πρόωρα την επίθεση, και ο Agent προχωρά στην εκτέλεση του επόμενου task του στο Behavior Tree, δηλαδή τηλεμεταφέρεται σε μια απόσταση από τον παίκτη και συνεχίζει την επίθεση. Αν ο παίκτης δεν ενεργήσει στο τρίτο σενάριο, το task ολοκληρώνεται και ο Agent συνεχίζει τις επιθετικές του ενέργειες. Ανεξαρτήτως του σεναρίου αποφυγής, το custom event Slash and Damage, το οποίο είναι υπεύθυνο για την απόδοση damage στον παίκτη, θα πυροδοτηθεί. Η ποσότητα του damage που θα αποδοθεί εξαρτάται από το σενάριο της αποφυγής. Η μεταβλητή IsBlocked, η οποία βρίσκεται στο component BPC_Damageable, καθορίζει το damage και παίρνει την τιμή της με βάση τις ενέργειες του Roll-Perfect Dodge στο blueprint του παίκτη.

Montage Notifies: Για την εφαρμογή των λειτουργιών κατά τη διάρκεια της επίθεσης, σε κάθε animation τοποθετήθηκαν σε κατάλληλα σημεία διάφορα Montage Notifies, ένα MontageNotifyWindow και ένα MontageNotifyState (MontageNotifyState_ActivateAttack). Το πρώτο από αυτά τα assets επιτρέπει την εκτέλεση ενεργειών όταν η αναπαραγωγή του animation φτάσει στο σημείο του notify και χρησιμοποιείται για την πυροδότηση ενεργειών που απαιτούν μόνο ένα χρονικό σημείο για την έναρξή τους. Το

δεύτερο asset παρέχει παρόμοιες λειτουργίες με ένα Montage Notify, ωστόσο αποτελεί ένα διάστημα που μπορεί να επεκταθεί εντός του animation, επιτρέποντας την ανάπτυξη ενεργειών στην αρχή αλλά και στο τέλος του. Το τρίτο asset παρέχει όλες τις λειτουργίες ενός MontageNotifyWindow, ωστόσο διαθέτει το δικό του blueprint, επιτρέποντας την ανάπτυξη επιπλέον κώδικα, προσφέροντας πρόσβαση σε παραμέτρους του animation (π.χ. τη διάρκεια του animation ή το reference του Agent που το αναπαράγει). Οι ενέργειες που εκτελούνται στο blueprint του MontageNotifyState χρησιμοποιούνται στην υλοποίηση του Perfect Dodge, και περιγράφονται στην παράγραφο CheckIsPerfectDodge.



Εικόνα 117. Animation Montage Notifies

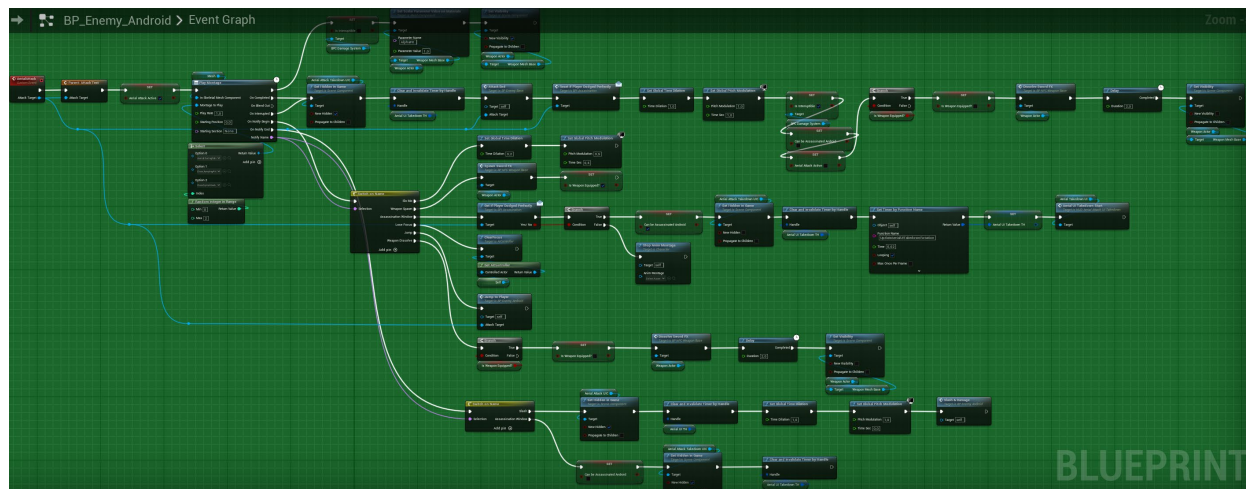
Μέσω των execution pins On Notify Begin και On Notify End, καθώς και του node Switch On Name, επιτυγχάνονται οι εξής ενέργειες:

- *Slo Mo*: Η επιβράδυνση του χρόνου στο παιχνίδι με το node Set Global Time Dilation. Παρέχει επιπλέον χρόνο αντίδρασης στον παίκτη και αποτελεί ένα βασικό τρόπο ανατροφοδότησης ως προς τις ενεργές επιθέσεις εναντίον του.
- *Weapon Spawn* και *Weapon Dissolve*: Η σταδιακή εμφάνιση και εξαφάνιση του οπλισμού του Agent με το Weapon Spawn FX και Weapon Dissolve FX αντίστοιχα. Πυροδοτεί τα κατάλληλα

custom events που ενεργοποιούν τα ειδικά εφέ του οπλισμού, τα οποία δημιουργήθηκαν μέσω Niagara Particle Systems.

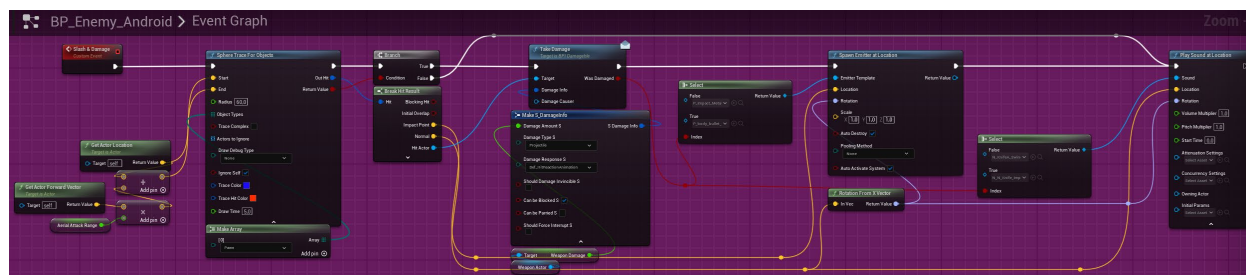
- *LoseFocus*: Η αποδέσμευση του σημείου εστίασης του Agent με το node ClearFocus, ώστε ο Agent να σταματήσει να έχει τροχιά άλματος προς τον παίκτη λίγο πριν ολοκληρωθεί η επίθεση. Αυτό διασφαλίζει πως ο Agent σταματά να ακολουθεί την κατεύθυνση του παίκτη μετά την ολοκλήρωση του άλματος, προκειμένου να διατηρηθεί μια ισορροπία στη δυσκολία αντιμετώπισης της επίθεσης.
- *Jump*: Η εκτέλεση του άλματος του Agent με το Jump to Player. Αντί για την εκτόξευση μέσω του root motion που περιλαμβάνεται στα animation, το custom event Jump to Player χρησιμοποιείται για να διασφαλίσει τη συνέπεια της κίνησης και για να επιτρέψει στο capsule του Agent να ακολουθεί το mesh του, κατά τη διάρκεια του άλματος. Αυτό είναι κρίσιμο, καθώς αν το άλμα χρησιμοποιούσε το root motion, το capsule του Agent θα βρισκόταν στο έδαφος ενώ το mesh του θα ήταν στον αέρα, με αποτέλεσμα να εμφανιστούν ανεπιθύμητες συγκρούσεις με αντικείμενα ή να επηρεαστεί το άλμα με μη φυσικό τρόπο. Η υλοποίηση του άλματος περιλαμβάνει τον υπολογισμό της μελλοντικής θέσης του παίκτη και την εκτόξευση του Agent μέσω των nodes Suggest Projectile Velocity Arc και Launch Character.
- *Slash*: Η εκτέλεση της ενέργειας που σχετίζεται με την απόδοση Damage και την αναπαραγωγή οπτικοακουστικών εφέ μέσω του custom event Slash & Damage, η διακοπή της επιβράδυνσης του χρόνου μέσω του SetGlobalTimeDilation και η απόκρυψη των εικονιδίων UI. Η εκτέλεση των παραπάνω ενεργειών πυροδοτείται στο τέλος του καθορισμένου χρονικού διαστήματος, καθώς το συγκεκριμένο notify είναι τύπου MontageNotifyState. Επιπλέον, περιλαμβάνει επιπρόσθετες ενέργειες που υλοποιούνται εντός του blueprint.
- *Assassination Window*: Οι ενέργειες που σχετίζονται με το χρονικό διάστημα στο οποίο ο Agent μπορεί να βρεθεί εύαλτος σε επιθέσεις Takedown. Ως Montage Notify Window, περιλαμβάνει ενέργειες που εκτελούνται στην αρχή (OnNotifyBegin) αλλά και στο τέλος του διαστήματος (OnNotifyEnd). Η αρχική πυροδότηση περιλαμβάνει την ενεργοποίηση της δυνατότητας Takedown και των κατάλληλων εικονιδίων σε περίπτωση που ο παίκτης εκτέλεσε ένα Perfect Dodge. Ο έλεγχος γίνεται μέσω του blueprint interface function GetIfPlayerDodgedPerfectly, επιστρέφοντας το αποτέλεσμα. Μια επιτυχημένη προσπάθεια μεταβάλλει τη μεταβλητή CanBeAssassinatedAndroid (Boolean) σε true ενεργοποιώντας το Takedown και το εικονίδιο που απεικονίζει το κουμπί Takedown γίνεται ορατό και αναβοσβήνει. Στην περίπτωση αποτυχημένης προσπάθειας το animation διακόπτεται και οι διαδικασίες ολοκλήρωσης της επίθεσης πυροδοτούνται μέσω του execution pin OnInterrupted του node PlayMontage. Αυτές περιλαμβάνουν την απόκρυψη του οπλισμού και των εικονιδίων, την αρχικοποίηση μεταβλητών και την ολοκλήρωση του task της επίθεσης. Τέλος, η τελική πυροδότηση μεταβάλλει τη μεταβλητή CanBeAssassinatedAndroid σε false, και αποκρύπτει τα εικονίδια του UI. Στην περίπτωση που ο παίκτης εκτελέσει ένα Takedown ανάμεσα στις δύο πυροδοτήσεις, η δεύτερη δε θα εκτελεστεί καθώς το animation θα σταματήσει ώστε να αναπαραχθεί αυτό του Takedown. Ως εκ τούτου, η συμπερίληψη των ενεργειών της στο execution pin OnInterrupted διασφαλίζουν τη σωστή λειτουργία της επίθεσης.

Το σχετικό διάγραμμα της υλοποίησης παρουσιάζεται στην εικόνα 118.



Εικόνα 118. Aerial Attack Blueprint Overview

Slash and Damage (Custom Event): Το Slash and Damage εκτελεί την ενέργεια επίθεσης του Android και είναι ενσωματωμένο στο child blueprint BP_Enemy_Android. Υπεύθυνο για την ανίχνευση της παρουσίας ενός actor που μπορεί να δεχτεί damage, το event καθορίζει εάν το damage θα αποδοθεί και αναπαράγει τα κατάλληλα οπτικά εφέ και ήχους, ανάλογα με την επιτυχία της επίθεσης. Η ανίχνευση των πιθανών actors που επηρεάζονται από την επίθεση πραγματοποιείται μέσω του node SphereTraceForObjects, το οποίο δέχεται ως είσοδο δύο σημεία: την αρχική θέση και το τέλος του trace. Το SphereTraceForObjects δημιουργεί μια σφαίρα και επιστρέφει το πρώτο αντικείμενο που συγκρούστηκε με αυτήν, μαζί με πληροφορίες σχετικά με τη σύγκρουση.



Εικόνα 119. Slash and Damage

Η χρήση του SphereTraceForObjects επιλέχθηκε αντί του LineTraceForObjects, καθώς ταιριάζει καλύτερα στη φύση και στον τρόπο επίθεσης του οπλισμού του Android, ο οποίος χρησιμοποιεί σπαθί. Λόγω της φυσικής του γεωμετρίας και του τρόπου που εκτελεί επιθέσεις, απαιτεί την ανίχνευση στην

περιοχή γύρω από αυτό, και όχι μόνο σε μια γραμμική πορεία. Για την αρχή του trace, χρησιμοποιείται η τοποθεσία του Agent, ενώ το τέλος καθορίζεται από το άθροισμα της τοποθεσίας του μπροστινού διανύσματος του Agent (GetActorForwardVector) και της μεταβλητής AerialAttackRange (float). Η χρήση αυτής της μεταβλητής επιτρέπει την αυξομείωση της εμβέλειας και της δύναμης της επίθεσης ανάλογα με τη δυσκολία του επιπέδου ή άλλους παράγοντες. Αυτή η μέθοδος παρέχει μια απλή και άμεση υλοποίηση για την προσομοίωση επίθεσης με σπαθί, ενώ επιτρέπει την ευκολία προσαρμογής της εμβέλειας της επίθεσης. Όπως και το LineTrace, το SphereTrace επιστρέφει μια τιμή Return Value (Boolean), η οποία υποδεικνύει αν το trace χτύπησε κάποιον actor. Στην περίπτωση που ένας actor συγκρουστεί με το trace, το event καλεί το TakeDamage, το οποίο εκτελείται μόνο εάν ο actor που χτυπήθηκε υποστηρίζει το αντίστοιχο interface (π.χ. ο παίκτης ή άλλοι Agents). Το TakeDamage δέχεται ως είσοδο παραμέτρους όπως το Weapon Damage, τον DamageType και τη μεταβλητή CanBeBlocked (Boolean), η οποία τίθεται σε true. Η ρύθμιση της CanBeBlocked σε true είναι σημαντική, καθώς επιτρέπει την απόκρουση της επίθεσης, εάν ο παίκτης την αποκρούσει επιτυχώς. Σε αυτή την περίπτωση, το damage δεν θα αποδοθεί, σύμφωνα με τις συνθήκες που αναπτύχθηκαν στο Macro CanBeDamaged του component BPC_Damageable. Στη συνέχεια, τα nodes SpawnEmitterAtLocation και SpawnSoundAtLocation χρησιμοποιούνται για την αναπαραγωγή των κατάλληλων οπτικών εφέ και ήχων που συνδέονται με τη σύγκρουση, παρέχοντας ανατροφοδότηση στον παίκτη σχετικά με την επιτυχία ή την αποτυχία της επίθεσης. Τέλος, αν δεν γίνει κάποια σύγκρουση με actor, οι ενέργειες απόδοσης damage και οπτικών εφέ παραλείπονται, και εκτελείται μόνο η αναπαραγωγή του ήχου.

Roll (Input Action Event): Το συγκεκριμένο event είναι υπεύθυνο για την εκτέλεση του συνόλου των ενεργειών που σχετίζονται με την κίνηση αποφυγής του παίκτη. Οι ενέργειες αυτές περιλαμβάνουν την αναπαραγωγή του animation αποφυγής, τους ελέγχους που αφορούν την εκτέλεση του Perfect Dodge και την ενεργοποίηση των οπτικοακουστικών εφέ που ενισχύουν την εμπειρία του παίκτη. Το event υλοποιείται εντός του blueprint του παίκτη και μπορεί να ενεργοποιηθεί με το πάτημα του αντίστοιχου κουμπιού από τον παίκτη. Το animation της κίνησης απεικονίζει μια γρήγορη και ευέλικτη κίνηση του χαρακτήρα καθώς εκτελεί μια στροφή, μετακινώντας τον με συγκεκριμένη ταχύτητα σε μικρή απόσταση. Η κίνηση μπορεί να πραγματοποιηθεί προς οποιαδήποτε κατεύθυνση όπως προς τα εμπρός, προς τα πλάγια ή προς τα πίσω, ανάλογα με την κατεύθυνση που έχει η κάμερα του παίκτη.

Η κίνηση Roll αποδίδει στον παίκτη προσωρινή ανοσία στην επιβολή damage από την επίθεση AerialAttack εφόσον εκτελείται εντός του χρονικού διαστήματος που καθορίζεται από το MontageNotifyState σε κάθε animation. Επιπλέον, η κίνηση μπορεί να χαρακτηριστεί ως Perfect Dodge, εάν πληρούνται συγκεκριμένες συνθήκες, καθιστώντας τον Agent ευάλωτο σε επιθέσεις Takedown. Ο χαρακτηρισμός της κίνησης ως Perfect Dodge περιλαμβάνει και την εκτέλεση ενός συνόλου οπτικοακουστικών εφέ, τα οποία ενισχύουν την αίσθηση εμβύθισης και παρέχουν σαφή ανάδραση στον παίκτη ότι μια σημαντική ενέργεια έχει πραγματοποιηθεί. Συγκεκριμένα, εμφανίζεται ένα εικονίδιο UI το οποίο απεικονίζει το κουμπί που αντιστοιχεί στην επίθεση Takedown και ο χρόνος του παιχνιδιού επιβραδύνεται για ένα μικρό χρονικό διάστημα. Ακόμη, εμφανίζεται ένα φωτογραφικό ίχνος (AfterImage) του παίκτη τη στιγμή που εκτελεί το Roll, ενώ ταυτόχρονα αναπαράγεται ένας

χαρακτηριστικός ήχος. Η στιγμή όπου ο παίκτης εκτελεί με επιτυχία ένα Perfect Dodge εντός του παιχνιδιού, απεικονίζεται στην εικόνα 120.



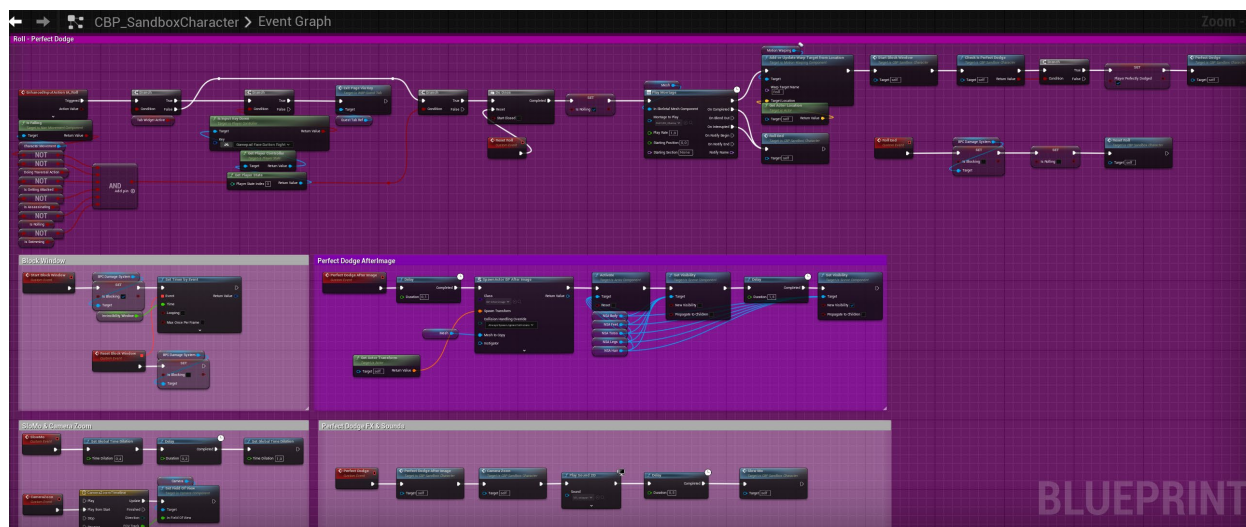
Εικόνα 120. Roll & Perfect Dodge Scenario

Το συγκεκριμένο event ξεκινά με την εκτέλεση των απαραίτητων ελέγχων προκειμένου να διασφαλιστεί ότι η ενέργεια μπορεί να πραγματοποιηθεί. Συγκεκριμένα, επαληθεύεται ότι ο παίκτης δεν εκτελεί ήδη ένα Roll, ότι δεν βρίσκεται σε διαδικασία εκτέλεσης ενέργειας Takedown, καθώς και άλλες συνθήκες που κρίνονται απαραίτητες. Ένα Do Once node χρησιμοποιείται ώστε να εξασφαλιστεί ότι η ενέργεια Roll δεν μπορεί να πυροδοτηθεί εκ νέου μέχρι να ολοκληρωθεί η τρέχουσα εκτέλεσή της. Παράλληλα, η μεταβλητή IsRolling (Boolean) λαμβάνει την τιμή true, δηλώνοντας ότι ο παίκτης βρίσκεται σε κατάσταση Roll.

Η αναπαραγωγή του animation της κίνησης πραγματοποιείται μέσω του node PlayMontage, το οποίο με τη σειρά του πυροδοτεί το custom event Start Block Window. Το event αυτό θέτει τη μεταβλητή IsBlocking του component BPC_Damageable σε true, παρέχοντας στον παίκτη προσωρινή αμυντική κατάσταση έναντι εισερχόμενου damage. Η κατάσταση αυτή διαρκεί για χρονικό διάστημα 0.5 δευτερολέπτων, όπως ορίζεται από τη μεταβλητή InvincibilityWindow (float), και επαναφέρεται στην τιμή false με χρήση του node SetTimerByEvent, το οποίο προγραμματίζει την αρχικοποίηση της μεταβλητής μετά την παρέλευση του συγκεκριμένου διαστήματος. Ως αποτέλεσμα, ο παίκτης που εκτελεί την κίνηση Roll αποκτά προσωρινή ανοσία σε επιθέσεις, γεγονός που απαιτεί τη σωστή χρονική εκτέλεση της κίνησης κατά τη διάρκεια μιας επίθεσης AerialAttack. Μία πρόωρη εκτέλεση του Roll ενδέχεται να οδηγήσει σε λήξη της αμυντικής κατάστασης προτού η επίθεση αποδώσει damage, με αποτέλεσμα ο παίκτης να υποστεί πλήγμα.

Αμέσως μετά, καλείται η συνάρτηση `CheckIsPerfectDodge`, η οποία επιστρέφει εάν το Roll πληροί τις προϋποθέσεις χαρακτηρισμού ως Perfect Dodge. Σε περίπτωση που το αποτέλεσμα είναι θετικό, η μεταβλητή `PlayerPerfectlyDodged` (Boolean) τίθεται σε true. Η μεταβλητή αυτή αξιοποιείται στη συνάρτηση `GetIfPlayerDodgedPerfectly`, την οποία καλεί ο Agent, προκειμένου να επαληθεύσει εάν πρέπει να αδρανοποιηθεί ως αποτέλεσμα μιας επιτυχημένης αποφυγής.

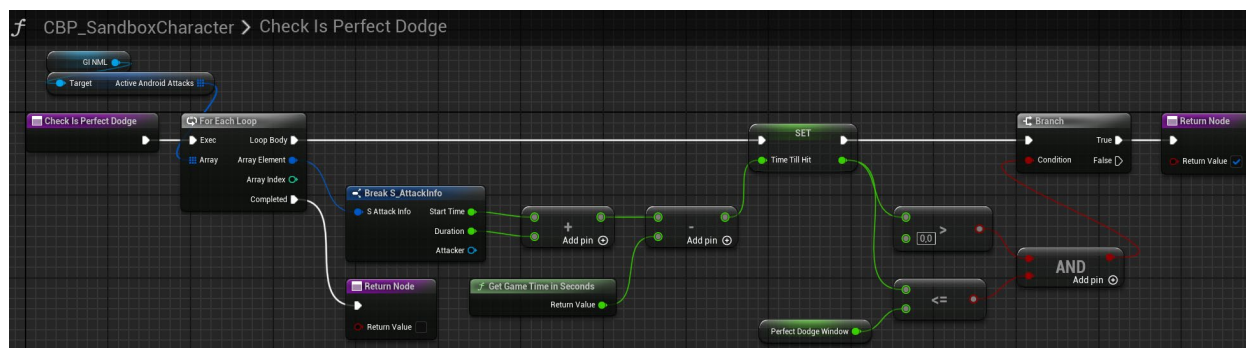
Στη συνέχεια, ενεργοποιείται το custom event `Perfect Dodge`, το οποίο αναπαράγει το σύνολο των σχετικών οπτικοακουστικών εφέ. Αυτά περιλαμβάνουν την εμφάνιση ενός φωτογραφικού ίχνους (`AfterImage`) του παίκτη κατά την εκτέλεση της αποφυγής, την επιβράδυνση της ροής του χρόνου, μια στιγμιαία εστίαση της κάμερας στον παίκτη, καθώς και την αναπαραγωγή ενός χαρακτηριστικού ηχητικού εφέ. Τέλος, η ολοκλήρωση ή η πρόωρη διακοπή του animation Roll προκαλεί την επαναφορά των μεταβλητών `IsRolling` και `IsBlocking` σε false, ενώ το node `Do Once` αρχικοποιείται εκ νέου, επιτρέποντας την επόμενη εκτέλεση της κίνησης Roll.



Εικόνα 121. Roll Blueprint Overview

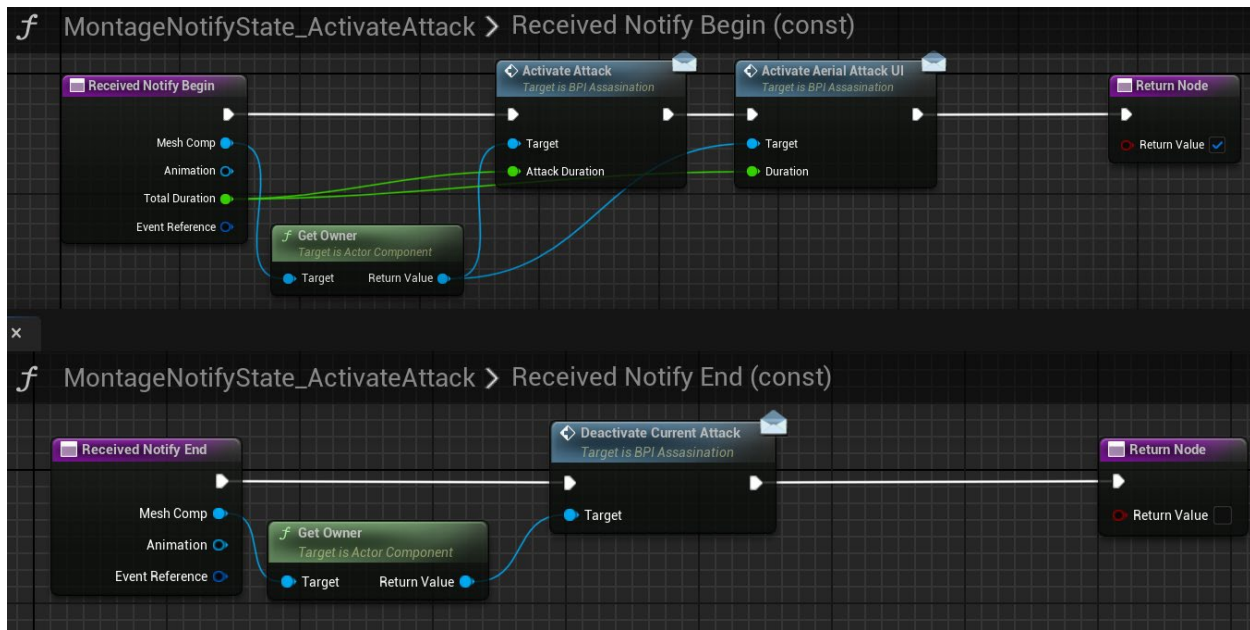
CheckIsPerfectDodge (Function): Η ενέργεια Roll μπορεί να χαρακτηριστεί ως Perfect Dodge μόνο εφόσον εκτελείται εντός του χρονικού διαστήματος που καθορίζεται από τη μεταβλητή `Perfect Dodge Window` (float), η οποία έχει προκαθορισμένη τιμή 0.2 δευτερόλεπτα. Ωστόσο, η τιμή της μεταβλητής μπορεί να προσαρμοστεί ανάλογα με το επίπεδο δυσκολίας, την κατάσταση του παίκτη, ή τα χαρακτηριστικά των Androids. Ο έλεγχος της συνθήκης πραγματοποιείται μέσα από τη λειτουργία `CheckIsPerfectDodge`, η οποία εκτελεί δύο κύριους ελέγχους και επιστρέφει την τιμή true (δηλαδή, το Roll είναι Perfect Dodge) εφόσον ικανοποιούνται και οι δύο συνθήκες ταυτόχρονα, και false σε αντίθετη περίπτωση. Ο πρώτος έλεγχος αξιολογεί αν η τιμή της μεταβλητής `TimeTillHit` (float), η οποία αναπαριστά τον χρόνο που απομένει μέχρι την ολοκλήρωση του `MontageNotifyState`, είναι μεγαλύτερη του μηδενός. Με αυτόν τον τρόπο διασφαλίζεται ότι η κίνηση Roll εκτελείται εντός του χρονικού παραθύρου που καθιστά την αποφυγή έγκυρη, αποφεύγοντας το damage. Ο δεύτερος έλεγχος συγκρίνει

την τιμή της μεταβλητής TimeTillHit με τη μεταβλητή PerfectDodgeWindow, εξασφαλίζοντας ότι το Roll πραγματοποιείται εντός του χρονικού πλαισίου που απαιτείται για να χαρακτηριστεί ως Perfect Dodge. Εάν και οι δύο συνθήκες ικανοποιούνται, το Roll χαρακτηρίζεται ως Perfect Dodge, αλλιώς δεν πληροί τα κριτήρια και ο παίκτης δεν αποκτά τα πλεονεκτήματα που συνδέονται με αυτήν την ενέργεια. Το σχετικό διάγραμμα παρουσιάζεται στην εικόνα 122.



Εικόνα 122. CheckIsPerfectDodge

Οι απαιτούμενες μεταβλητές για τον υπολογισμό της τιμής TimeTillHit, όπως η διάρκεια του MontageNotifyState, παρέχονται μέσω της μεταβλητής Active Android Attacks (Array), η οποία περιλαμβάνει όλες τις ενεργές επιθέσεις AerialAttack που εκτελούνται ταυτόχρονα κατά του παίκτη. Η χρήση της μεταβλητής αυτής είναι απαραίτητη, καθώς υπάρχει η δυνατότητα ο παίκτης να δέχεται ταυτόχρονες επιθέσεις AerialAttack, καθεμία με διαφορετική χρονική διάρκεια. Συνεπώς, κάθε νέα επίθεση προστίθεται δυναμικά στο Active Android Attacks array κατά την έναρξή της και αφαιρείται με την ολοκλήρωσή της μέσω αντίστοιχων functions. Η διαχείριση αυτή υλοποιείται στο blueprint MontageNotifyState_ActivateAttack. Συγκεκριμένα, κατά την ενεργοποίηση του διαστήματος Notify State που ορίζεται σε κάθε animation επίθεσης του Agent, η επίθεση προστίθεται στο array μέσω του function Received_NotifyBegin, το οποίο καταγράφει και τη χρονική διάρκεια του ενεργού διαστήματος. Σε αυτό το σημείο, ακόμη, πυροδοτείται η απεικόνιση του UI εικονιδίου μέσω του function ActivateAerialAttackUI. Αντίστοιχα, μέσω του Received_NotifyEnd, η συγκεκριμένη επίθεση αφαιρείται από το array. Με αυτόν τον τρόπο, η συνάρτηση CheckIsPerfectDodge μπορεί να αξιολογήσει όλες τις τρέχουσες ενεργές επιθέσεις, προκειμένου να προσδιορίσει εάν η κίνηση πληροί τα κριτήρια για χαρακτηρισμό ως Perfect Dodge. Τα functions Received_NotifyBegin και Received_NotifyEnd του MontageNotifyState παρουσιάζονται στην εικόνα 123.



Εικόνα 123. MontageNotifyState_ActivateAttack

8.5. Seeking State

Κατά τη διάρκεια του Seeking State, ο Agent εκτελεί εκτεταμένη αναζήτηση στο περιβάλλον του με σκοπό τον εντοπισμό εχθρικής παρουσίας. Αν και η λειτουργικότητα του State παρουσιάζει ομοιότητες με εκείνη του Investigating, διαφοροποιείται ουσιαστικά ως προς τη διάρκειά του, τον βαθμό επιθετικότητας του Agent, τον τρόπο αντίδρασης του, καθώς και τα οπτικοακουστικά εφέ που το συνοδεύουν. Η μετάβαση στο συγκεκριμένο State δύναται να προκληθεί από τρεις βασικούς παράγοντες: (1) όταν ο Agent, κατά την εκτέλεση του Detecting State, χάνει οπτική επαφή με τον παίκτη λίγο πριν ολοκληρώσει την πλήρη αναγνώρισή του, (2) όταν, κατά την εκτέλεση του Attacking State, ο παίκτης απομακρύνεται προσωρινά από το οπτικό πεδίο του Agent, και (3) όταν ο Agent εντοπίσει έναν εξουδετερωμένο σύμμαχο. Στις δύο πρώτες περιπτώσεις, η μετάβαση συνοδεύεται από στοχευμένα οπτικοακουστικά εφέ, τα οποία ενισχύουν την αντίληψη του παίκτη ότι ο Agent βρίσκεται σε φάση επιθετικής αναζήτησης.

Κατά την εκτέλεση του Seeking State, ο Agent εξετάζει διαδοχικά πολλαπλές περιοχές εκτός του άμεσου οπτικού του πεδίου, με επίκεντρο το σημείο που προκάλεσε το σχετικό ερέθισμα. Παραμένει σε ετοιμότητα επίθεσης, οπλισμένος και έτοιμος να αντιδράσει άμεσα σε περίπτωση εντοπισμού του παίκτη. Παράλληλα, όπως και στο Investigating State, αναπαρίσταται ένδειξη κατάστασης μέσω εικονιδίου UI επάνω από τον Agent, ενώ για τους Ranged τύπους αναπαράγονται και αντίστοιχα ηχητικά εφέ. Σε

περίπτωση που η αναζήτηση αποβεί άκαρπη, ο Agent επανέρχεται στο Passive State, συνεχίζοντας είτε την προγραμματισμένη του περιπολία είτε την προστασία προκαθορισμένου σημείου.

Οι Agents τύπου Ranged και Android παρουσιάζουν διακριτές συμπεριφορές κατά την εκτέλεση του συγκεκριμένου State, γεγονός που οδήγησε στην ανάπτυξη και ανάθεση εξειδικευμένων Subtrees ανά τύπο Agent. Οι βασικές διαφοροποιήσεις εστιάζουν στην ύπαρξη ενεργειών όπως η όπλιση, τα animations αντίδρασης και οι ηχητικοί διάλογοι, που υλοποιούνται μόνο στους Ranged Agents και παραλείπονται στους Androids.

Οι πιθανές εισερχόμενες μεταβάσεις προς το Seeking State περιλαμβάνουν: από Detecting (λόγος 1), από Attacking (λόγος 2), καθώς και από τα Passive ή Investigating States (λόγος 3). Αντίστοιχα, η έξοδος από το Seeking State οδηγεί σε: Attacking, εφόσον εντοπιστεί εχθρός· Passive, αν η αναζήτηση αποτύχει· Dead, σε περίπτωση εξουδετέρωσης του Agent· και Investigating, αν ο παίκτης χρησιμοποιήσει το αντικείμενο Bottle. Η τελευταία περίπτωση συνεπάγεται μετάβαση από μια κατάσταση υψηλής επιθετικότητας σε μία πιο παθητική, και για να επιτραπεί, απαιτείται να μην υπάρχουν άλλοι Agents που έχουν αναγνωρίσει την παρουσία του παίκτη, όπως ορίζεται στη λειτουργία Handle Sensed Sound. Με τον τρόπο αυτό, διασφαλίζεται ότι ο Agent δεν μπορεί να χειραγωγηθεί εύκολα κατά τη διάρκεια περιόδων έντονης εχθρικής δραστηριότητας.

Στο παρόν κεφάλαιο θα αναλυθεί η υλοποίηση του συγκεκριμένου State, η λειτουργικότητά του, καθώς και τα επιμέρους Tasks και Functions που το υποστηρίζουν. Ιδιαίτερη έμφαση θα δοθεί στις λειτουργίες που ενεργοποιούν το State, στα εργαλεία αναπαράστασης των αντιδράσεων, καθώς και στα assets που δημιουργήθηκαν για την πλήρη ενσωμάτωσή του στο εκάστοτε Behavior Tree κάθε τύπου Agent.

8.5.1. Seeking Behavior Tree Overview

Η ενεργοποίηση του Behavior Tree για το Seeking State πραγματοποιείται μέσω αξιολόγησης συνθηκών στο επίπεδο του State Selector, βασισμένων σε τιμές του Blackboard. Συγκεκριμένα, το υποδέντρο ενεργοποιείται όταν η μεταβλητή State λάβει την τιμή Seeking. Τα σύνολα ενεργειών οργανώνονται σε ξεχωριστά Subtrees ανά τύπο Agent: το BT_Subtree_SeekingEnemy αφορά στους Agents τύπου Ranged, ενώ το BT_Subtree_SeekingAndroid στους Agents τύπου Android. Και τα δύο χρησιμοποιούνται μέσω Run Behavior tasks εντός των Behavior Trees των αντίστοιχων Agents. Το Seeking State τοποθετείται ιεραρχικά ανάμεσα στα States Attacking και Detecting, αποκτώντας την ενδεδειγμένη προτεραιότητα εκτέλεσης.

Ranged Agent Subtree

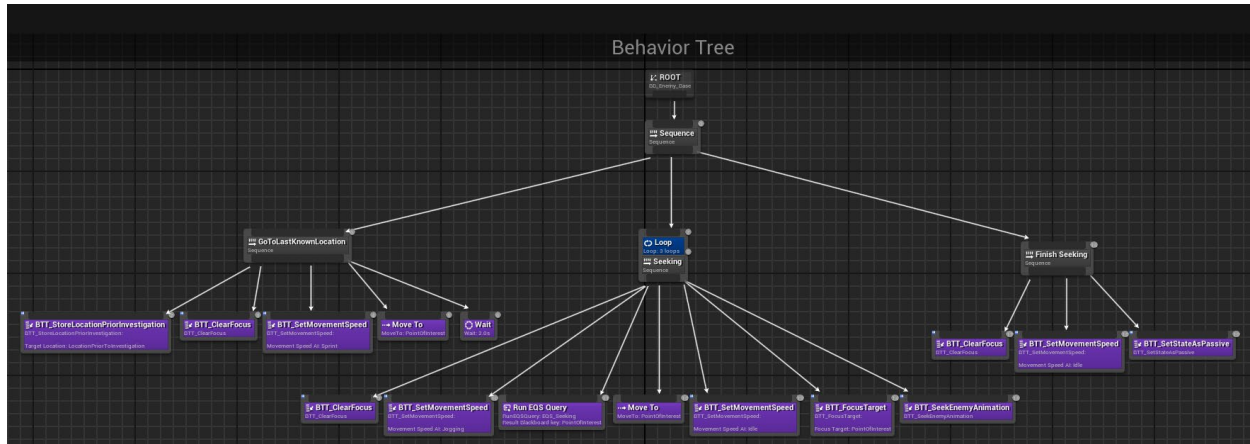
Το υποδέντρο για τους Ranged Agents ξεκινά με έναν βασικό Selector, ο οποίος επιλέγει ανάμεσα στην εκτέλεση της ενέργειας οπλισμού και στην έναρξη του Sequence αναζήτησης. Αν ο Agent δεν φέρει το όπλο στο χέρι του, τότε ακινητοποιείται και εκτελεί την ενέργεια όπλισης. Η διαδικασία αναζήτησης οργανώνεται σε τρία βασικά Sequences: GoToLastKnownLocation, Seeking και Finish Seeking.

- [illegible]

Android Agent Subtree

199

χρησιμοποιείται ένα Wait task, επιτρέποντας στον Agent να συνεχίσει αμέσως την αναζήτηση, χωρίς να επιδειξει κάποιο είδος συναισθηματικής ή φυσικής αντίδρασης στο σημείο απώλειας του παίκτη. Επίσης, το task BTT_BehaviorVoiceSFX παραλείπεται, καθώς οι Android Agents στο παρόν Project δεν εκφωνούν ηχητικούς διαλόγους.

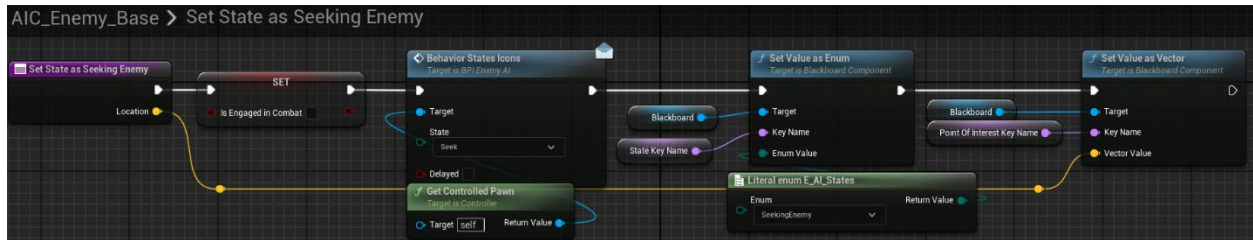


Εικόνα 125. Android Agent Subtree

Τα πλήρη γραφήματα (Blueprints) των δύο υποδέντρων για τους τύπους Agents παρουσιάζονται στις εικόνες 124 και 125, οι οποίες απεικονίζουν τη δομή και την ροή εκτέλεσης των επιμέρους tasks αναλυτικά.

8.5.2. Seeking State Assets

SetStateAsSeeking (Function): Το function SetStateAsSeeking, το οποίο υλοποιείται εντός του AI Controller, είναι υπεύθυνο για τη μεταβολή ενός Agent στο State Seeking και την ενεργοποίηση των βασικών ενεργειών που απαιτούνται για την ορθή λειτουργία του εν λόγω state. Η μεταβλητή IsEngagedInCombat τίθεται σε τιμή False, ενημερώνοντας το σύστημα πως ο Agent δεν βρίσκεται πλέον σε ενεργή εμπλοκή, όπως συμβαίνει στο Attacking State. Η ενέργεια αυτή εξασφαλίζει την κατάλληλη απεμπλοκή από καταστάσεις μάχης, προκειμένου να ξεκινήσει διαδικασία αναζήτησης. Ακόμη, το εικονίδιο Behavior State Icon γίνεται ορατό απεικονίζοντας το State του Agent μέσω του function Behavior State Icons με την επιλογή Seek. Τέλος, αναθέτονται οι τιμές των Blackboard Keys State και PointOfInterest, με το πρώτο να παίρνει την τιμή Seeking, και το δεύτερο την τελευταία τοποθεσία που ο παίκτης ήταν ορατός από τον Agent, η οποία λαμβάνεται σε κάθε κλήση του function ως είσοδος.



Εικόνα 126. SetStateAsSeeking

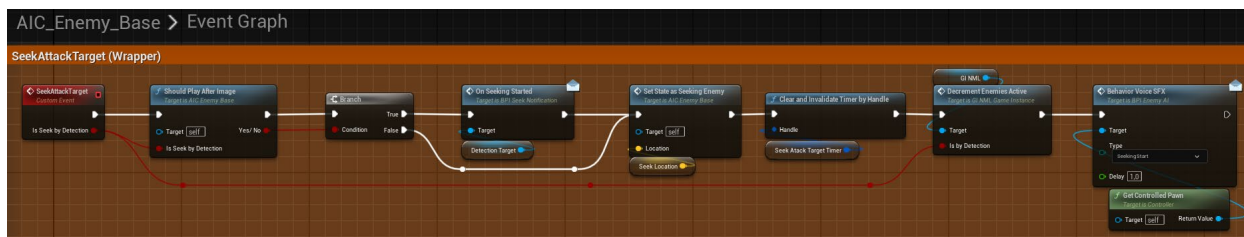
SeekAttackTarget (Custom Event): Το SeekAttackTarget αποτελεί ένα custom event, το οποίο λειτουργεί ως wrapper για την ομαλή μετάβαση ενός Agent στο Seeking State, ενσωματώνοντας τη λογική διαχείρισης όλων των πιθανών περιπτώσεων που μπορούν να προκαλέσουν την εν λόγω μετάβαση. Το Event υλοποιεί διαφοροποιημένες ενέργειες, ανάλογα με το πλαίσιο που οδήγησε τον Agent στη συγκεκριμένη κατάσταση.

Ειδικότερα, στην περίπτωση όπου ένας Agent μεταβαίνει στο Seeking State από το Attacking State ή από το Detecting State, ενεργοποιείται ένα οπτικοακουστικό εφέ. Το εφέ αυτό έχει διττό ρόλο: αφενός, σηματοδοτεί στον παίκτη την ενεργοποίηση της κατάστασης αναζήτησης από τον Agent και, αφετέρου, ενισχύει την αφηγηματική συνοχή και ατμόσφαιρα του παιχνιδιού. Το εφέ αποτελείται από έναν χαρακτηριστικό ηχητικό τόνο και την ταυτόχρονη εκτέλεση ενός Niagara Particle System, το οποίο αποδίδει ένα οπτικό ίχνος (AfterImage) του παίκτη στο ακριβές σημείο όπου χάθηκε η οπτική επαφή, σηματοδοτώντας το σημείο αναζήτησης του Agent. Σε αντίθεση με τις παραπάνω περιπτώσεις, όταν η μετάβαση στο Seeking State προκαλείται από την ανίχνευση ενός εξουδετερωμένου συμμάχου Agent, το οπτικοακουστικό εφέ παραλείπεται. Η απουσία του εφέ οφείλεται στο γεγονός ότι ο Agent, στη συγκεκριμένη περίπτωση, δεν εντοπίζει άμεσα τον παίκτη, αλλά εκκινεί μια διαδικασία αναζήτησης στον χώρο γύρω από τον σύμμαχό του. Η εικόνα 127 απεικονίζει την εκκίνηση του Seeking State, στις περιπτώσεις μετάβασης του Agent από τα States Attacking ή Detecting.



Εικόνα 127. AfterImage Effect Demonstration

Η εκτέλεση των οπτικοακουστικών εφέ υλοποιείται μέσω της συνάρτησης OnSeekingStarted, η οποία είναι υλοποιημένη στο Blueprint του παίκτη. Βασική προϋπόθεση για την ενεργοποίησή της είναι η ικανοποίηση της συνθήκης που ορίζεται από το function ShouldPlayAfterImage, σύμφωνα με το οποίο το πλήθος των Agents που δρουν επιθετικά κατά του παίκτη δεν πρέπει να υπερβαίνει τον αριθμό ένα. Με αυτόν τον τρόπο αποφεύγεται η πολλαπλή ενεργοποίηση των εφέ, γεγονός που θα μπορούσε να δημιουργήσει σύγχυση ή να διαταράξει την ισορροπία του παιχνιδιού.



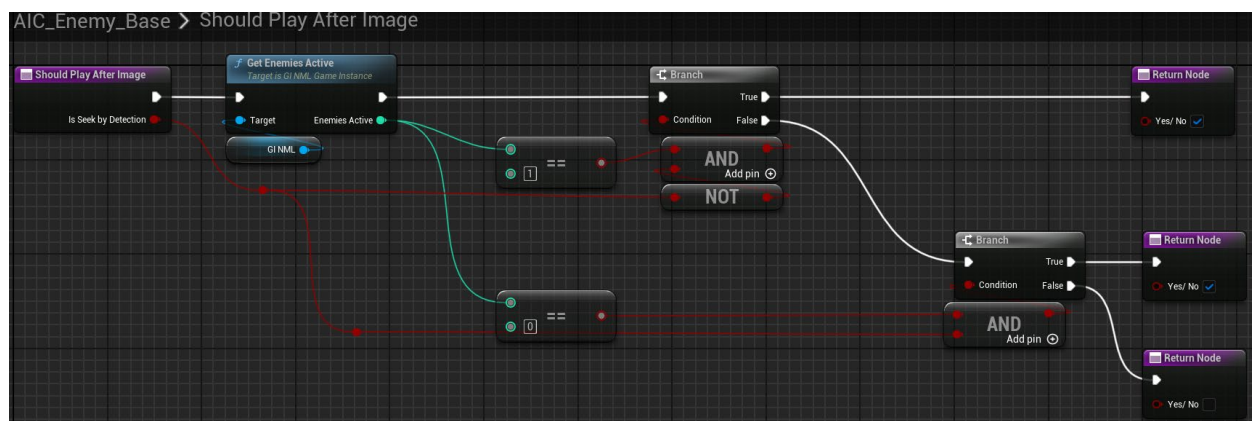
Εικόνα 128. SeekAttackTarget

Κατόπιν της εκτέλεσης των εφέ, το function SeekAttackTarget καλεί το function SetStateAsSeeking, το οποίο διαχειρίζεται την μετάβαση του Behavior Tree στο Seeking State. Ταυτόχρονα, εκτελείται το function DecrementEnemiesActive, το οποίο μειώνει τον αριθμό των Agents που καταγράφονται ως ενεργά επιτιθέμενοι κατά του παίκτη, προσαρμόζοντας δυναμικά την εχθρική δραστηριότητα στο περιβάλλον του παιχνιδιού. Η διαδικασία ολοκληρώνεται με την κλήση του function BehaviorVoiceSFX,

υπεύθυνο για την αναπαραγωγή των ηχητικών διαλόγων που σηματοδοτούν αφενός την είσοδο του Agent στο Seeking State και αφετέρου την αιτία της μετάβασης. Οι διάλογοι αυτοί λειτουργούν τόσο ως ηχητικά ερεθίσματα για τον παίκτη, όσο και ως αφηγηματικοί δείκτες της συμπεριφορικής κατάστασης του Agent. Το σχετικό διάγραμμα της υλοποίησης του SeekAttackTarget παρουσιάζεται στην εικόνα 128.

Με αυτό τον τρόπο, η SeekAttackTarget αποτελεί μια κομβική συνάρτηση για τη διαχείριση της μεταβατικής συμπεριφοράς του Agent προς το Seeking State. Μέσω της ενοποίησης των λειτουργιών αντίδρασης, οπτικοακουστικής σήμανσης και διαχείρισης κατάστασης, το function εξασφαλίζει ότι η μετάβαση του Agent είναι λειτουργικά συνεπής, αφηγηματικά πειστική και προσαρμοσμένη στο ευρύτερο πλαίσιο δραστηριότητας του παιχνιδιού.

ShouldPlayAfterImage (Function): Το function εκτελεί τον έλεγχο αναπαραγωγής του οπτικοακουστικού εφέ που σηματοδοτεί πως ένας Agent αναζητά τον παίκτη. Το function υλοποιείται στο AI Controller και επιστρέφει το αποτέλεσμα για κάθε περίπτωση.



Εικόνα 129. ShouldPlayAfterImage

Ο έλεγχος βασίζεται σε δύο μεταβλητές: την ακέραια μεταβλητή EnemiesActive, η οποία καταγράφει τον αριθμό των Agents που βρίσκονται ενεργά σε επιθετική κατάσταση απέναντι στον παίκτη, και την Boolean μεταβλητή IsSeekByDetection, η οποία υποδηλώνει αν η μετάβαση στο Seeking State προκλήθηκε από ανίχνευση και όχι από άμεση εμπλοκή σε μάχη. Συγκεκριμένα, η IsSeekByDetection λαμβάνει την τιμή true όταν ο Agent εισέρχεται στο Seeking State εξαιτίας απώλειας του παίκτη από το οπτικό του πεδίο κατά τη φάση του Detection, ενώ παραμένει false όταν η μετάβαση οφείλεται σε έξοδο από το Attacking State.

Η συνθήκη αναπαραγωγής του εφέ ικανοποιείται σε δύο περιπτώσεις. Πρώτον, όταν ο τελευταίος Agent που είχε αντιληφθεί τον παίκτη μεταβαίνει στο Seeking State, δηλαδή όταν η μεταβλητή EnemiesActive έχει τιμή 1 και ταυτόχρονα η IsSeekByDetection είναι false. Δεύτερον, όταν κανένας Agent δεν έχει

οπτική επαφή με τον παίκτη και η μετάβαση στο Seeking State γίνεται μέσω της διαδικασίας ανίχνευσης, οπότε η EnemiesActive είναι μηδενική και η IsSeekByDetection έχει τιμή true.

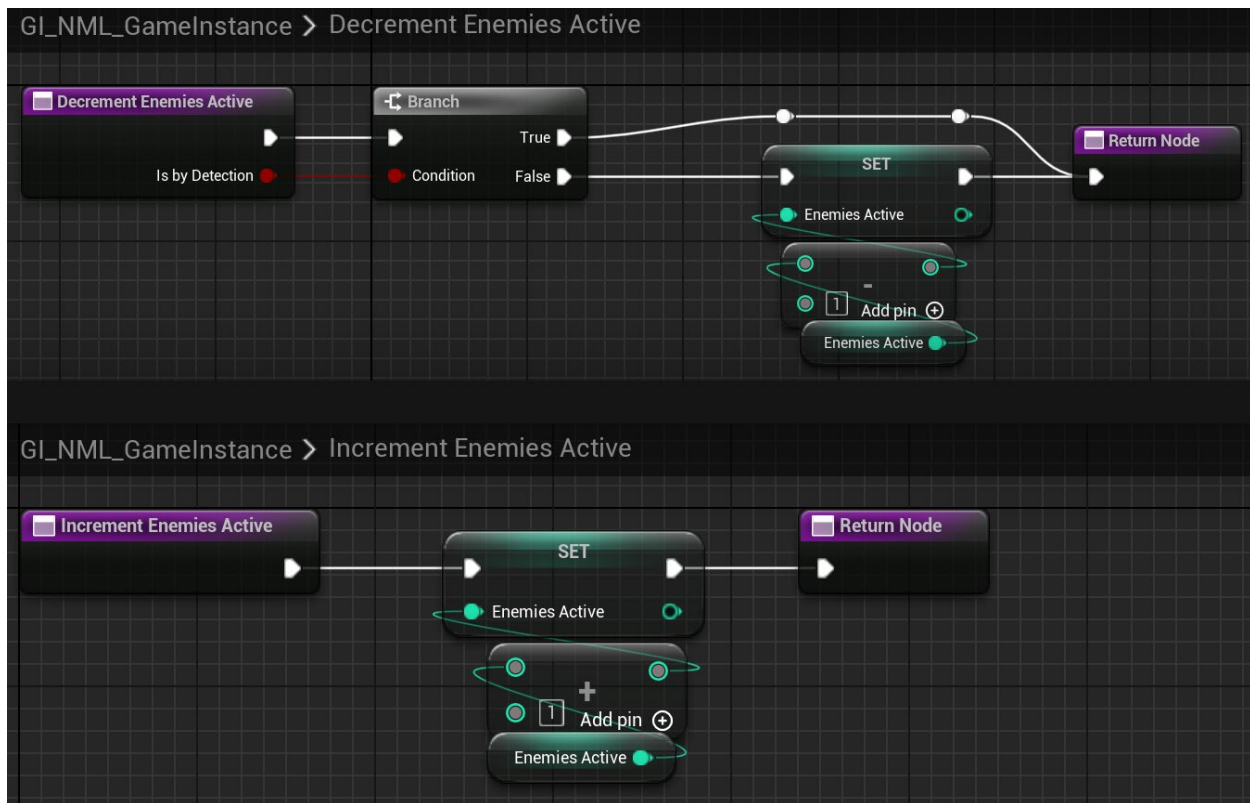
Η μεταβλητή EnemiesActive διατηρείται διαρκώς επικαιροποιημένη μέσω των function IncrementEnemiesActive και DecrementEnemiesActive, τα οποία καλούνται κάθε φορά που μεταβάλλεται η κατάσταση ενός Agent και προσαρμόζουν αντίστοιχα την τιμή της. Η προσεκτική διαχείριση των μεταβλητών αυτών εξασφαλίζει ότι η συνάρτηση ShouldPlayAfterImage ανταποκρίνεται με ακρίβεια στις συνθήκες του παιχνιδιού, διατηρώντας την ενιαία εμπειρία για τον παίκτη και διασφαλίζοντας την απρόσκοπτη μετάβαση ανάμεσα στις καταστάσεις εντός του AI συστήματος.

Με αυτόν τον τρόπο αποφεύγεται η υπερβολική ή επαναλαμβανόμενη ενεργοποίηση του εφέ, η οποία θα μπορούσε να οδηγήσει σε οπτικοακουστική κόπωση του παίκτη, ειδικά σε περιπτώσεις όπου πολλοί Agents μεταβαίνουν στο Seeking State ταυτόχρονα ή σε σύντομο χρονικό διάστημα.

IncrementEnemiesActive και DecrementEnemiesActive (Functions): Τα function είναι υπεύθυνα για τη δυναμική διαχείριση της μεταβλητής EnemiesActive, η οποία αντιπροσωπεύει τον αριθμό των ενεργών Agents, δηλαδή των Agents που έχουν πλήρη επίγνωση της παρουσίας του παίκτη και βρίσκονται στο Attacking State. Η σωστή παρακολούθηση αυτής της μεταβλητής είναι καθοριστικής σημασίας για τη ρύθμιση ορισμένων λειτουργιών του AI, όπως η αναπαραγωγή του οπτικοακουστικού εφέ κατά τη μετάβαση σε Seeking State.

Η υλοποίηση των συγκεκριμένων functions έχει πραγματοποιηθεί στο blueprint του GameInstance, το οποίο αποτελεί ενσωματωμένο στοιχείο της Unreal Engine και χρησιμοποιείται για την αποθήκευση δεδομένων που αφορούν τη συνολική κατάσταση του παιχνιδιού, ανεξάρτητα από τον κύκλο ζωής των επιμέρους Actors. Η χρήση του GameInstance ως τοποθεσία υλοποίησης εξασφαλίζει ότι η μεταβλητή EnemiesActive είναι κοινώς προσβάσιμη και τροποποιήσιμη από όλους τους Agents, διασφαλίζοντας έτσι συνέπεια στη διαχείριση της πληροφορίας και αποφυγή τοπικών αποκλίσεων.

Η λειτουργία του IncrementEnemiesActive καλείται κάθε φορά που ένας Agent εισέρχεται στο Attacking State, αυξάνοντας κατά μία μονάδα τη μεταβλητή EnemiesActive. Αντίστοιχα, η DecrementEnemiesActive καλείται όταν ο Agent αποχωρεί από την επιθετική κατάσταση, μεταβαίνοντας σε οποιοδήποτε άλλο State, μειώνοντας τον αριθμό κατά μία μονάδα. Ωστόσο, για να αποφευχθούν σφάλματα κατά την εκτέλεση, όπως η μείωση της τιμής της μεταβλητής σε περιπτώσεις όπου ο Agent δεν είχε βρεθεί προηγουμένως στο Attacking State, εφαρμόζεται έλεγχος μέσω της μεταβλητής IsbyDetection (Boolean). Η συγκεκριμένη συνθήκη διασφαλίζει την προστασία της μεταβλητής από λανθασμένες τροποποιήσεις, όπως για παράδειγμα σε μεταβάσεις από το Investigating στο Passive State. Το σχετικό διάγραμμα των functions παρουσιάζεται στην εικόνα 130.

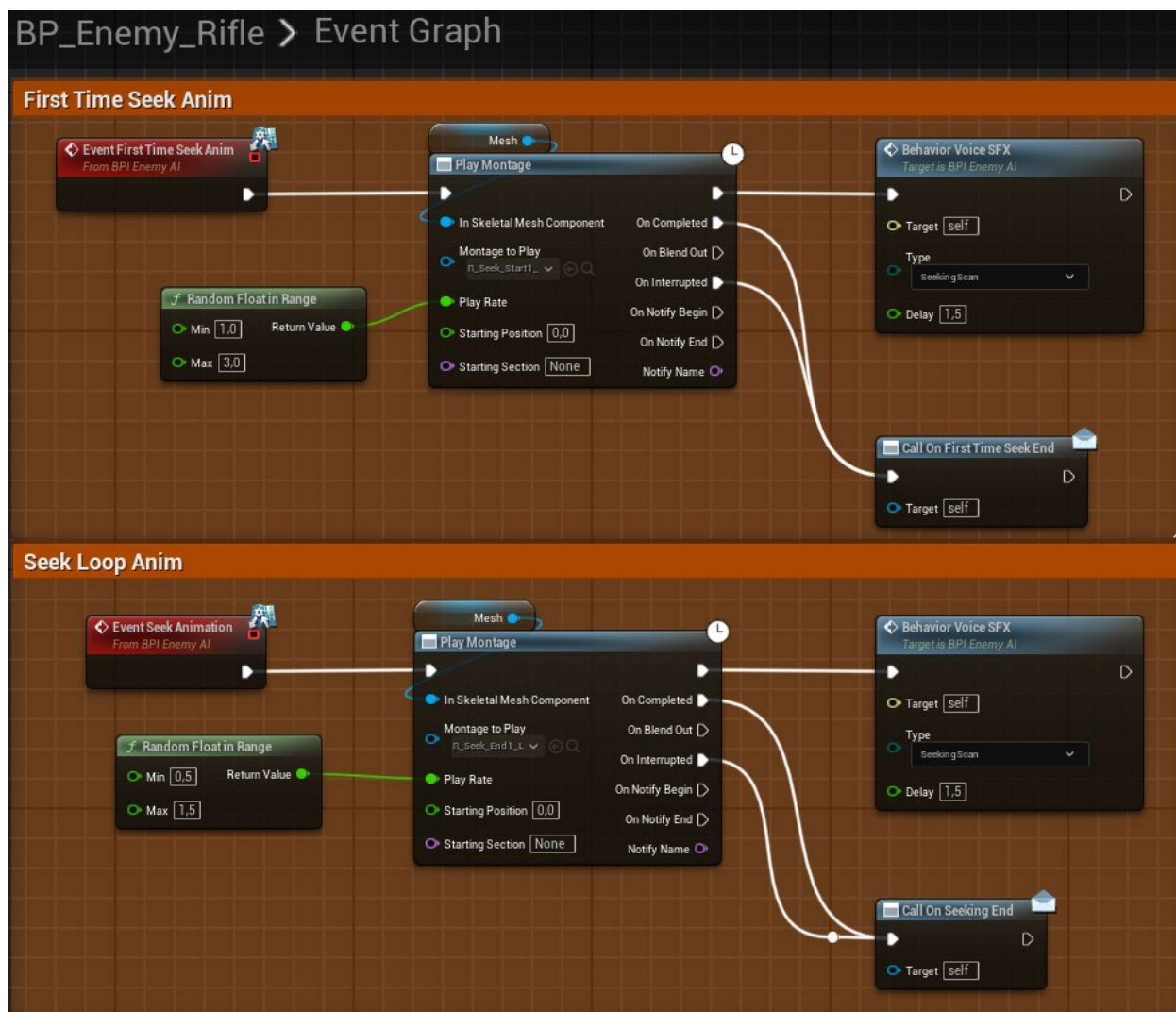


Εικόνα 130. IncrementEnemiesActive & DecrementEnemiesActive

Η τρέχουσα τιμή της μεταβλητής EnemiesActive είναι προσβάσιμη μέσω της συνάρτησης GetEnemiesActive, η οποία μπορεί να κληθεί από το blueprint του AI Controller, από τα blueprints των επιμέρους Agents, ή και από το blueprint του παίκτη. Αυτή η δυνατότητα προσφέρει ευελιξία στον σχεδιασμό συστημάτων που βασίζονται στην πληροφορία της επιθετικής δραστηριότητας των εχθρών, επιτρέποντας την άμεση αξιοποίησή της από διαφορετικά στοιχεία του συστήματος AI και του gameplay γενικότερα.

FirstSeekAnim και SeekingAnimation (Function): Τα function είναι υπεύθυνα για την αναπαραγωγή των animations αντίδρασης κατά την εκτέλεση του Seeking State και υλοποιούνται σε κάθε child blueprint των Agents. Το FirstSeekAnim αναπαράγει το animation που εκτελείται κατά την ολοκλήρωση του Sequence GoToLastKnownLocation, κατά το οποίο ο Agent φτάνει στο τελευταίο σημείο που εθεάθη ο παίκτης. Το SeekingAnimation αναπαράγει το animation που εκτελείται εντός του επαναλαμβανόμενου Sequence Seeking, κατά το οποίο ο Agent αναζητά πιθανά σημεία στα οποία μπορεί να κρύβεται ο παίκτης και κινείται με ταχύτητα Jog προς αυτά. Η χρήση των δυο function επιτρέπει τον διαχωρισμό του animation που χρησιμοποιείται σε κάθε περίπτωση, αναθέτοντας στο πρώτο function animation με εντονότερες κινήσεις σε σχέση με το δεύτερο, ώστε να ταιριάζουν σε κάθε περίπτωση και οι ενέργειες να διαφοροποιούνται. Τέλος, κάθε function ολοκληρώνεται με την αναπαραγωγή ηχητικού διαλόγου, και

την κλήση των Event Dispatchers First Time Seek End και Seeking End προκειμένου τα επόμενα task να πυροδοτηθούν μετά την ολοκλήρωση των animations αντίδρασης. Τα διαγράμματα των functions απεικονίζονται στην εικόνα 131.



Εικόνα 131. FirstSeekAnim & SeekingAnimation

EQS Seeking (Environment Query System): Το Query χρησιμοποιείται για την αναζήτηση πιθανών σημείων μετακίνησης του Agent στο περιβάλλον, τα οποία ενδέχεται να υποδηλώνουν κρυψώνες ή σημεία διαφυγής του παίκτη. Η λειτουργία του συγκεκριμένου Query ενεργοποιείται κατά την εκτέλεση του Sequence Seeking, παρέχοντας στον Agent τη δυνατότητα να κινηθεί στρατηγικά σε μία τοποθεσία που ενισχύει τις πιθανότητες εντοπισμού του παίκτη.

Η διαδικασία επιλογής της τοποθεσίας βασίζεται σε μια συνδυαστική σάρωση των σημείων που εκτείνονται εμπρός από τον Agent, εξαιρώντας ωστόσο εκείνα που βρίσκονται εντός του οπτικού του πεδίου. Για να επιτευχθεί αυτό, το Query αξιοποιεί το στοιχείο "Cone: generate in front of Querier", το οποίο δημιουργεί πιθανά σημεία σε σχήμα κώνου μπροστά από τον Agent, εξασφαλίζοντας ότι δεν θα επιλεγούν τοποθεσίες πίσω του ή εκτός της κατεύθυνσης κίνησής του. Επιπλέον, εφαρμόζεται το test "Path Cost: from Querier", με το οποίο απορρίπτονται τα σημεία που απαιτούν υψηλό κόστος μετακίνησης, δηλαδή σημαντικό χρόνο για να φτάσει ο Agent. Αυτό έχει ως αποτέλεσμα την προτεραιοποίηση κοντινών τοποθεσιών με υψηλότερη πιθανότητα διαλογής. Παράλληλα, το test "Trace" εξαιρεί από την επιλογή τα σημεία στα οποία υπάρχει απευθείας οπτική πρόσβαση από τον Agent, επιτρέποντας την εστίαση μόνο σε εκείνα που καλύπτονται από εμπόδια. Τα αξιολογημένα σημεία που προκύπτουν παρουσιάζονται στην εικόνα 132, με την χρήση του TestingPawn το οποίο εκτελεί το EQS Seeking.



Εικόνα 132. EQS Seeking

Το σημείο που τελικά επιλέγεται είναι εκείνο που συγκεντρώνει την υψηλότερη βαθμολογία βάσει των παραπάνω κριτηρίων. Έτσι, όταν ο Agent φτάσει στην τελευταία γνωστή θέση του παίκτη, θα σαρώσει τις γύρω περιοχές και θα κινηθεί προς το κοντινότερο σημείο που βρίσκεται εκτός του οπτικού του πεδίου, αυξάνοντας τις πιθανότητες να εντοπίσει τον παίκτη. Η επανάληψη του Sequence επιτρέπει στον Agent να συνεχίσει τη σάρωση και σε άλλες πιθανές τοποθεσίες, ενισχύοντας τον ρεαλισμό της συμπεριφοράς αναζήτησης.

Ωστόσο, η συγκεκριμένη διαμόρφωση του EQS βασίζεται στην παρουσία εμποδίων στο περιβάλλον, γεγονός που μπορεί να προκαλέσει δυσλειτουργίες. Συγκεκριμένα, στην περίπτωση που τα tests αποτύχουν να εντοπίσουν κατάλληλα σημεία, δηλαδή αν δεν υπάρχουν εμπόδια μπροστά στον Agent, καμία τοποθεσία δεν θα επιλεγεί ως σημείο αναζήτησης, με αποτέλεσμα ο Agent να παραμένει στάσιμος και να επαναλαμβάνει ανεπιτυχώς το EQS Query. Η σχεδίαση του Level στο συγκεκριμένο project έχει λάβει υπόψη αυτό το ενδεχόμενο, φροντίζοντας ώστε η τοποθέτηση αντικειμένων να εξασφαλίζει πάντοτε την ύπαρξη εμποδίων στην κατεύθυνση του Agent.

Παρά ταύτα, μια πιο ευέλικτη και αξιόπιστη προσέγγιση θα ήταν η εκτέλεση του EQS μέσω ενός Custom Task στο Behavior Tree. Με αυτόν τον τρόπο, καθίσταται δυνατή είτε η εκ νέου εκτέλεση ενός δεύτερου EQS με διαφορετικά κριτήρια σε περίπτωση αποτυχίας, είτε η πλήρης ακύρωση της αναζήτησης, ώστε να προχωρήσει το Behavior Tree στην επόμενη συμπεριφορική ακολουθία. Η προσέγγιση αυτή προσφέρει μεγαλύτερη σταθερότητα στο gameplay και αποτρέπει καταστάσεις όπου ο Agent παγιδεύεται λειτουργικά λόγω ελλιπούς πληροφoρίας από το περιβάλλον.

8.6. Detecting State

Σε αυτή την κατάσταση, ο Agent αντιλαμβάνεται έναν actor ως εχθρική παρουσία (π.χ. τον παίκτη) και ξεκινά μια διαδικασία προοδευτικής αναγνώρισης του στόχου, η οποία συνεχίζεται μέχρι την πλήρη αντίληψη του στόχου. Η πρόοδος αυτής της αναγνώρισης παρακολουθείται μέσω μιας μπάρας προόδου (Detection Meter), η οποία αυξομειώνεται ανάλογα με το χρόνο που ο παίκτης βρίσκεται εντός του οπτικού πεδίου του Agent, καθώς και την απόσταση μεταξύ τους. Η αύξηση ή μείωση της μπάρας επηρεάζει τη συμπεριφορά του Agent, επιτρέποντάς του να προχωρήσει σε διαφορετικά States ανάλογα με το σημείο της προόδου, ενεργώντας με ένα ομαλό και ρεαλιστικό τρόπο. Για παράδειγμα, όταν η μπάρα γεμίσει πλήρως, ο παίκτης θεωρείται εντελώς αντιληπτός και ο Agent μεταβαίνει στο Attacking State. Αν, ωστόσο, ο παίκτης βγει εκτός του οπτικού πεδίου του Agent πριν γεμίσει εντελώς η μπάρα, το σημείο του παίκτη θεωρείται ως ερέθισμα προς διερεύνηση, με τον Agent να μεταβαίνει στο Seeking State. Ως εκ τούτου, το Detecting State αποτελεί το συνδυαστικό κρίκο ανάμεσα στην αναγνώριση του παίκτη και στις ενέργειες που θα εκτελεστούν από τον Agent. Περιλαμβάνει το σύνολο των ενεργειών που διαχειρίζονται τις διάφορες περιπτώσεις, καθορίζοντας τη συμπεριφορά του Agent ανάλογα με την εξέλιξη της αναγνώρισης.

Πολυπλοκότητα και Υλοποίηση

Η υλοποίηση της διαδικασίας αυτής παρουσίασε σημαντικές προκλήσεις λόγω δύο κύριων παραμέτρων:

- *Οπτικά Ερεθίσματα και Διαχείριση του AI Perception:* Η μηχανή χρησιμοποιεί το SightSense του AI Perception Component για την αναγνώριση οποιουδήποτε actor που εισέρχεται στο οπτικό πεδίο του Agent, ενεργοποιώντας το function HandleSensedSight. Αυτό καθιστά απαραίτητο τον διαχωρισμό των ερεθισμάτων ανάλογα με την πηγή τους, όπως τον παίκτη, τους συμμάχους Agents και άλλους Actors. Η διαφοροποίηση αυτή επιτυγχάνεται μέσω της ανάθεσης μοναδικών

αριθμών σε κάθε τύπο Actor (π.χ. οι Agents και ο παίκτης). Αυτός ο διαχωρισμός χρησιμοποιείται εντός του Custom Event OnPlayerSeen, το οποίο αναλαμβάνει τη διαχείριση των οπτικών ερεθισμάτων και των ενεργειών που εκτελούνται σε κάθε περίπτωση. Συμπεριλαμβάνει την κατηγοριοποίηση των ερεθισμάτων, την παρακολούθηση της προόδου του Detection Meter, καθώς και τις μεταβάσεις του Agent από state σε state.

- *Μηχανική Stealth*: Ένα σημαντικό χαρακτηριστικό του συστήματος είναι η δυνατότητα του παίκτη να κρύβεται από τους Agents σε ειδικά σημεία (StealthSpots), όπως π.χ. σε θάμνους, ακόμα κι αν είναι θεωρητικά εντός του οπτικού πεδίου τους. Όταν ο παίκτης βρίσκεται σε ένα τέτοιο σημείο, η διαδικασία αναγνώρισης του στόχου (Detection) θα πρέπει να λειτουργεί με τον ίδιο τρόπο που λειτουργεί όταν ο παίκτης εισέρχεται ή εξέρχεται από το οπτικό πεδίο του Agent. Ωστόσο, η διαδικασία της αναγνώρισης σε αυτή την περίπτωση απαιτεί διαφορετική προσέγγιση από τη διαδικασία που συμβαίνει όταν ο παίκτης βγαίνει εντελώς εκτός του οπτικού πεδίου του Agent, καθώς ο παίκτης παραμένει εντός του οπτικού πεδίου, αλλά κρυμμένος. Αυτή η μηχανική προσέθεσε μια επιπλέον πρόκληση στην ανάπτυξη, καθώς απαιτεί συνεχείς ελέγχους σε διάφορα σημεία του κώδικα για να εξασφαλιστεί ότι το Detection Meter και η μηχανική Stealth αλληλεπιδρούν σωστά και χωρίς σφάλματα.

Η ανάγκη προσεκτικής διαχείρισης αυτών των παραμέτρων απαιτεί την αξιολόγηση της κατάστασης του Agent (π.χ. σε ποιο state βρίσκεται) και της κατάστασης του παίκτη (π.χ. αν κρύβεται ή όχι σε StealthSpot). Έτσι, οι διάφοροι έλεγχοι πρέπει να εκτελούνται σε διαφορετικά σημεία του κώδικα για να διασφαλιστεί η σωστή αλληλεπίδραση μεταξύ των παραμέτρων. Αυτή η πολυπλοκότητα καταδεικνύει τη σημασία του AI Controller, ο οποίος διαχειρίζεται το σύνολο των ενεργειών του Agent στο Detecting State και χρησιμοποιεί blueprint interface functions για την μεταφορά και ανάκτηση των απαραίτητων μεταβλητών. Το Behavior Tree λειτουργεί συμπληρωματικά, περιλαμβάνοντας περιορισμένες ενέργειες, και παραμένει σε κατάσταση "αναμονής", περιμένοντας την απόφαση για την εκτέλεση της επόμενης ενέργειας του Agent.

Πιθανές Μεταβάσεις

Οι πιθανές μεταβάσεις προς το Detecting State προκύπτουν όταν ο παίκτης εισέλθει στο οπτικό πεδίο του Agent ενώ αυτός βρίσκεται στα Passive ή Investigating states. Εάν ο Agent βρίσκεται ήδη στο Seeking State, θα εκτελέσει άμεσα επιθετικές ενέργειες. Αν βρίσκεται στο Attacking State ή σε άλλο state υψηλότερης προτεραιότητας, η αντίληψη του παίκτη δε θα προκαλέσει αλλαγή στην τρέχουσα κατάσταση.

Οι μεταβάσεις από το Detecting State περιλαμβάνουν:

- Μετάβαση στο Attacking State όταν το Detection Meter γεμίσει ή αν ο παίκτης προκαλέσει ζημιά στον Agent
- Μετάβαση στο Seeking State αν ο παίκτης βγει εκτός οπτικού πεδίου του Agent πριν γεμίσει πλήρως η μπάρα ή αν ο Agent αντιληφθεί ότι ένας συνάδελφος έχει πεθάνει

- Μετάβαση στο Passive State αν το Detection Meter μηδενιστεί (π.χ. όταν ο παίκτης κρυφτεί σε StealthSpot)
- Μετάβαση στο Investigating State αν ο παίκτης χρησιμοποιήσει ένα αντικείμενο, όπως το Bottle, κατά τη διάρκεια της διαδικασίας αναγνώρισης

Αυτές οι μεταβάσεις καθορίζονται από τη σωστή αξιολόγηση των παραμέτρων του παίκτη και του Agent, με την προτεραιότητα που έχει οριστεί στο σύστημα.

Οι πιθανές μεταβάσεις προς το Detecting State είναι όταν ο παίκτης εισέλθει στο οπτικό πεδίο του Agent όταν αυτός βρίσκεται στα states Passive ή Investigating. Στην περίπτωση που ο Agent αντιληφθεί τον παίκτη και βρίσκεται στο Seeking State, ο Agent θα εκτελέσει άμεσα ενέργειες επίθεσης. Αν ο Agent βρίσκεται στο Attacking state ή σε κάποιο state υψηλότερης προτεραιότητας, μια σύντομη εναλλαγή στην κατάσταση αντίληψης δεν θα προκαλέσει αλλαγή στην εκτέλεση του τρέχοντος state.

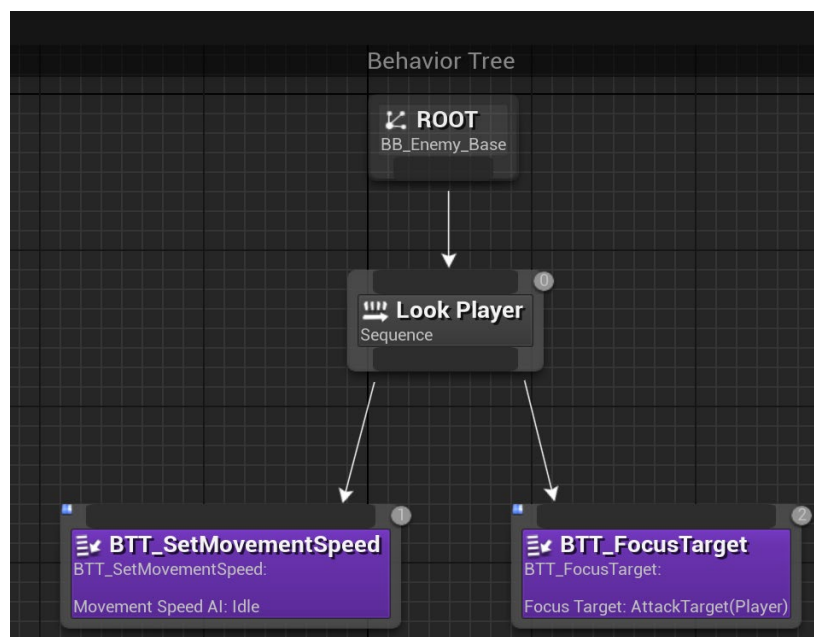
Σε αυτή την υποενότητα, παρουσιάζεται μια επισκόπηση του τμήματος του Behavior Tree που αφορά το Detecting State των Agents, καθώς και η ανάλυση της υλοποίησης της μηχανικής Stealth και των assets που αναπτύχθηκαν για την υποστήριξή της. Επιπλέον, θα εξεταστούν τα assets και τα functions που σχετίζονται με τη διαδικασία αναγνώρισης (Detection), καθώς και η εφαρμογή τους στο function OnPlayerSeen, το οποίο είναι υπεύθυνο για τη συνολική διαχείριση των οπτικών ερεθισμάτων που προέρχονται από τον Agent.

Behavior Tree Overview

Η συμπεριφορά ενός Agent που εισέρχεται στο Detecting State περιλαμβάνει την ακινητοποίηση του, την εστίασή του προς την κατεύθυνση του παίκτη, την έναρξη της διαδικασίας αναγνώρισης μέσω του Detection Meter, και τη μετάβασή του σε ένα από τα πιθανά states. Συγκεκριμένα, η είσοδος του παίκτη στο οπτικό πεδίο του Agent προκαλεί την ακινητοποίησή του στο σημείο που βρίσκεται, με ταυτόχρονη εστίαση στον παίκτη καθ' όλη τη διάρκεια του state. Παράλληλα, η διαδικασία περιλαμβάνει την αναπαραγωγή ενός ήχου, ενός animation αντίδρασης, καθώς και την ενεργοποίηση ενός εικονιδίου UI, το οποίο εμφανίζεται στην οθόνη του παίκτη. Αυτό το εικονίδιο αναπαριστά το Detection Meter, δείχνοντας την τρέχουσα κατάσταση του Agent και την αντίληψή του, και ενημερώνεται συνεχώς καθ' όλη τη διάρκεια του Detection. Με αυτόν τον τρόπο, ο παίκτης ενημερώνεται με διαδραστικό και ρεαλιστικό τρόπο για το γεγονός, δίνοντάς του την ευκαιρία να αντιδράσει αναλόγως. Στη συνέχεια, το Detection Meter αυξομειώνεται ανάλογα με το αν ο παίκτης βρίσκεται εντός ή εκτός του οπτικού πεδίου του Agent ή αν κρύβεται σε κάποιο StealthSpot. Η τιμή του Detection Meter καθορίζει ποιο state θα επιλέξει ο Agent στη συνέχεια.

Λόγω της πολυπλοκότητας των ενεργειών που πραγματοποιούνται κατά τη διάρκεια αυτής της διαδικασίας, το μεγαλύτερο μέρος του κώδικα έχει αναπτυχθεί εντός του AI Controller. Οι μοναδικές ενέργειες που εκτελούνται μέσα στο Behavior Tree είναι η ακινητοποίηση του Agent μέσω της αλλαγής της ταχύτητάς του σε Idle (BTT_SetMovementSpeed) και η τοποθέτηση του παίκτη ως σημείο εστίασης (BTT_FocusTarget). Με αυτόν τον τρόπο, ο Agent κατευθύνει συνεχώς το σώμα του προς τον παίκτη

όσο αυτός βρίσκεται εντός του οπτικού του πεδίου, επιτρέποντας την αυξομείωση του βαθμού εντοπισμού ανάλογα με το αν ο Agent βλέπει ή όχι τον παίκτη. Σύμφωνα με την υλοποίηση, οι Agents τύπου Ranged, οι οποίοι διαθέτουν όπλα, διατηρούν την κατάσταση οπλισμού τους κατά την είσοδο στο Detecting State. Συγκεκριμένα, αν ο Agent ήταν οπλισμένος πριν τη μετάβαση σε αυτό το state, θα εκτελέσει τη διαδικασία με το όπλο του. Αν δεν ήταν οπλισμένος, δεν θα οπλίσει κατά τη διάρκεια του Detecting State. Αντίστοιχα, οι Androids που δεν διαθέτουν ενέργειες όπλισης ή αφόπλισης δεν επηρεάζονται από αυτή τη λογική. Οι ενέργειες εκτελούνται από κάθε τύπου Agent, γι' αυτό και έχουν ενσωματωθεί σε ένα subtree (BT_Subtree_Detecting), το οποίο εκχωρείται στον Detecting Selector του Behavior Tree κάθε Agent. Το State τοποθετείται μεταξύ των Seeking και Investigating, εξασφαλίζοντας ότι τα states χαμηλότερης προτεραιότητας θα διακοπούν για την εκτέλεση του, ενώ το Detecting State δεν μπορεί να διακόψει ένα State υψηλότερης προτεραιότητας. Το σχετικό διάγραμμα απεικονίζεται στην εικόνα 133.



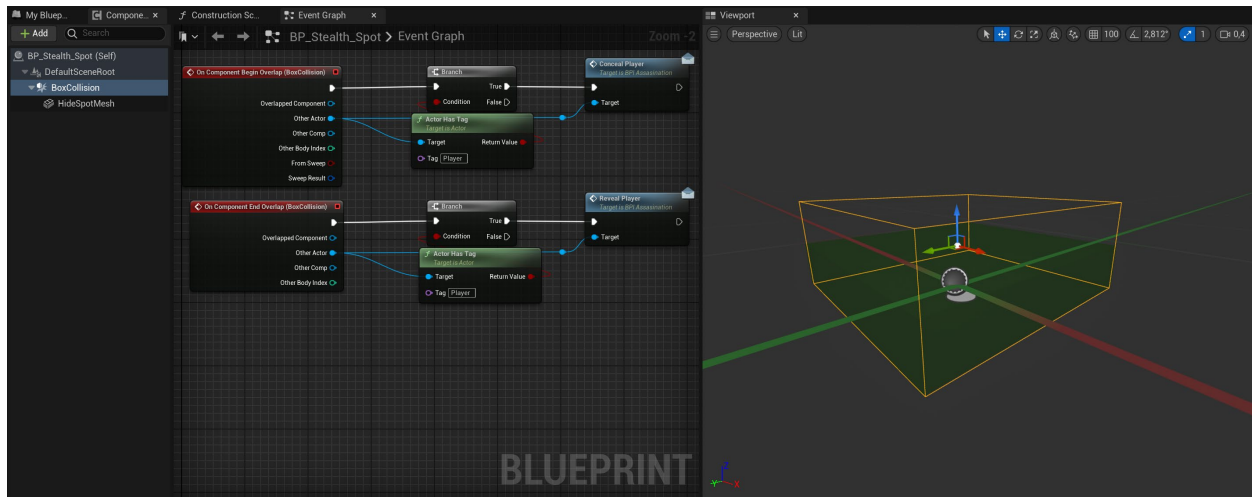
Εικόνα 133. Detecting State Subtree

Ο διαχωρισμός των ενεργειών και η ενσωμάτωσή τους εντός του AI Controller, σε αντίθεση με το Behavior Tree, εξασφαλίζουν την ορθή εκτέλεση των απαραίτητων ελέγχων και την ακριβή μετάβαση των states την κατάλληλη χρονική στιγμή, αποφεύγοντας την ανάγκη ανάπτυξης πολλαπλών και σύνθετων tasks εντός του Behavior Tree.

8.6.1. Stealth System

BP_StealthSpot (Blueprint Actor): Ο actor αναπαριστά τα σημεία στο περιβάλλον του παιχνιδιού που επιτρέπουν στον παίκτη να κρυφτεί από το οπτικό πεδίο των Agents και περιλαμβάνει τη διαχείριση ενεργειών που αποκρύπτουν ή καθιστούν ορατό τον παίκτη. Ο actor διαθέτει ένα Box Collision, το οποίο μπορεί να τοποθετηθεί σε πολλαπλά σημεία του Level και ενεργοποιεί τις αντίστοιχες ενέργειες όταν ο παίκτης εισέρχεται ή εξέρχεται από αυτό, μέσω των events `OnComponentBeginOverlap` και `OnComponentEndOverlap`, αντίστοιχα.

Το πρώτο event καλεί το function `ConcealPlayer`, το οποίο περιλαμβάνει τις ενέργειες που καθιστούν τον παίκτη αόρατο από τους Agents και άπροσβλητο από τη διαδικασία `Detection`. Το δεύτερο event καλεί το function `RevealPlayer`, το οποίο επαναφέρει τον παίκτη στην κανονική κατάσταση, επιτρέποντας την ανίχνευσή του από τους Agents. Τα functions αυτά υλοποιούνται στο blueprint του παίκτη και η κλήση τους πραγματοποιείται μέσω του Blueprint interface `BPI_Assassination`, το οποίο περιλαμβάνει όλες τις functions που σχετίζονται με τις μηχανικές `Stealth`, όπως αναλύεται στο υποκεφάλαιο 8.6. Η κλήση κάθε function προϋποθέτει ότι ο actor που αλληλεπικαλύπτεται με το asset `Box Collision` φέρει το Tag "Player", διασφαλίζοντας ότι οι ενέργειες εκτελούνται μόνο για τον παίκτη και όχι για οποιονδήποτε actor. Το Blueprint του Asset παρουσιάζεται την εικόνα 134.



Εικόνα 134. BP_StealthSpot

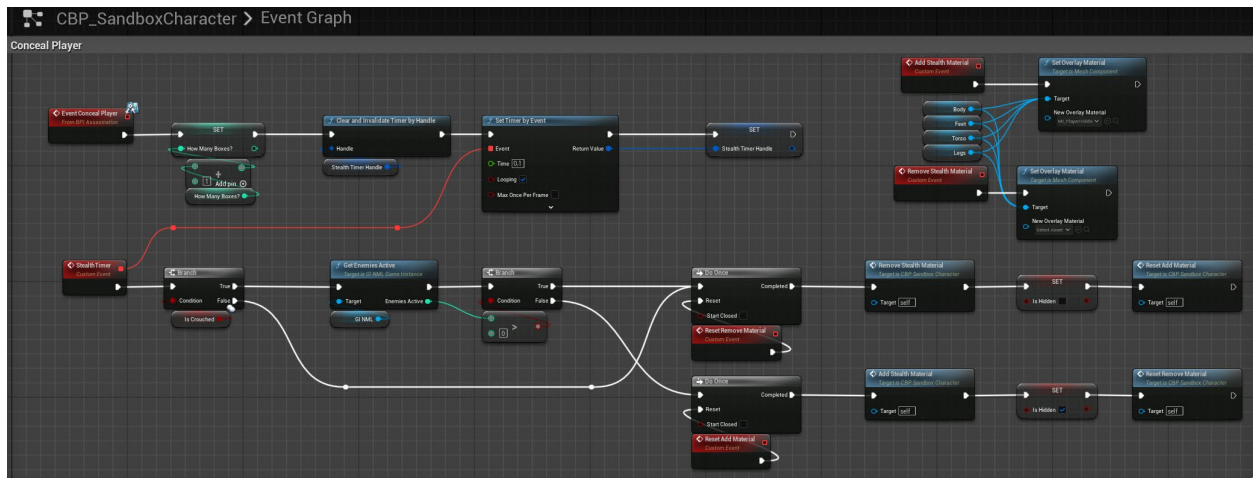
ConcealPlayer (Function): Το συγκεκριμένο function εκτελεί όλες τις απαραίτητες ενέργειες για την απόκρυψη του παίκτη από έναν Agent, και υλοποιείται στο blueprint του παίκτη. Η κεντρική του λειτουργία είναι η ανάθεση της τιμής της μεταβλητής `IsHidden` (Boolean), η οποία καθορίζει την κατάσταση απόκρυψης του παίκτη. Αυτή η μεταβλητή χρησιμοποιείται για ελέγχους στον AI Controller, σε functions που καθορίζουν τη συμπεριφορά του Agent όταν ο παίκτης βρίσκεται εντός του οπτικού του πεδίου, συμπεριλαμβανομένων των λειτουργιών που σχετίζονται με τη διαδικασία ανίχνευσης

(Detection). Παράλληλα, το function ενεργοποιεί και ένα ειδικό οπτικό εφέ στον χαρακτήρα του παίκτη, το οποίο σηματοδοτεί την κατάσταση απόκρυψης.

Οι βασικές παράμετροι που ληφθήκαν υπόψη για την υλοποίηση του function είναι οι εξής:

- *StealthSpot και Crouched*: Για να θεωρηθεί ο παίκτης κρυμμένος και απρόσβλητος από το Detection, πρέπει να πληρούνται δύο προϋποθέσεις: (α) Ο παίκτης πρέπει να βρίσκεται εντός ενός StealthSpot, και (β) να βρίσκεται σε κατάσταση Crouched (σκυφτός), γεγονός που αντιπροσωπεύεται από την τιμή της μεταβλητής IsCrouched (Boolean).
- *Επιστροφή στην κατάσταση ορατότητας*: Ο παίκτης μπορεί να επανέλθει σε κατάσταση ορατότητας μέσω δύο τρόπων: (α) Εξέρχεται από το StealthSpot, γεγονός που πυροδοτεί το function RevealPlayer, ή (β) Βρίσκεται σε κατάσταση Uncrouched (όρθιος) εντός του StealthSpot, οπότε και καθίσταται ορατός, χωρίς ωστόσο να πυροδοτείται το RevealPlayer.
- *Επιθετική κατάσταση των Agents*: Εάν ο παίκτης είναι αντιληπτός από κάποιον Agent στο Level, τότε δεν μπορεί να θεωρηθεί κρυμμένος, ακόμη και αν βρίσκεται εντός του StealthSpot.
- *Μεταβάσεις μεταξύ των StealthSpots*: Η υλοποίηση πρέπει να διασφαλίζει πως μια μετάβαση του παίκτη από ένα StealthSpot σε ένα άλλο, τα οποία αλληλεπικαλύπτονται, δεν πρέπει να πυροδοτήσει την επαναφορά της κατάστασης απόκρυψης.

Το function ξεκινά με την αύξηση της τιμής της μεταβλητής HowManyBoxes (Integer), η οποία αντιπροσωπεύει τον αριθμό των StealthSpots στα οποία ο παίκτης βρίσκεται. Στη συνέχεια, καλείται το event StealthTimer, το οποίο περιλαμβάνει τις συνθήκες για την απόκρυψη, ελέγχοντας την τιμή της μεταβλητής IsCrouched και τον αριθμό των Agents που έχουν επιθετική συμπεριφορά προς τον παίκτη μέσω της function GetEnemiesActive. Αν η μεταβλητή IsCrouched είναι true και ο αριθμός των ενεργών Agents είναι μικρότερος του μηδενός, ο παίκτης πληροί τις προϋποθέσεις απόκρυψης. Στην περίπτωση αυτή, το function αναθέτει το ειδικό εφέ και ορίζει την τιμή της μεταβλητής IsHidden σε true. Αν δεν πληρούνται οι συνθήκες, το ειδικό εφέ αφαιρείται και η μεταβλητή IsHidden παίρνει την τιμή false. Η λειτουργία του StealthTimer βασίζεται στο node SetTimerbyEvent, το οποίο έχει ρυθμιστεί με την επιλογή Looping και έχει οριστεί η τιμή 0,1 στη μεταβλητή Time, ώστε το event να εκτελείται συνεχώς, ενώ ο παίκτης παραμένει εντός του StealthSpot. Αυτή η ρύθμιση διασφαλίζει ότι η κατάσταση απόκρυψης ενημερώνεται δυναμικά, εξασφαλίζοντας την σωστή λειτουργία των ελέγχων που χρησιμοποιούν τη μεταβλητή IsHidden και την γρήγορη ανταπόκριση του οπτικού εφέ.



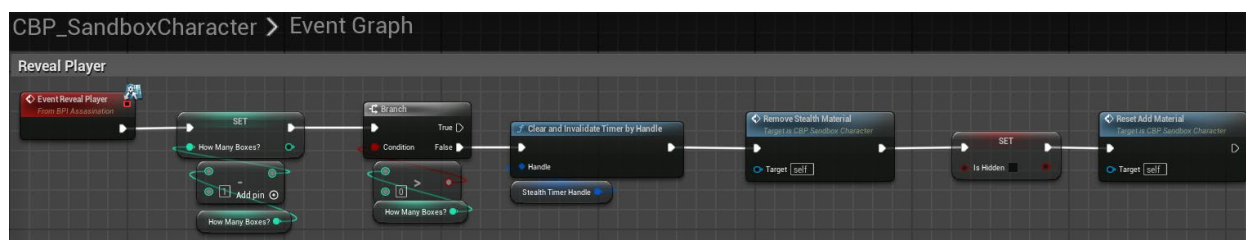
Εικόνα 135. ConcealPlayer

Το ειδικό εφέ που ενεργοποιείται κατά την απόκρυψη του παίκτη περιλαμβάνει ένα Overlay Material, το οποίο επικαλύπτει τον παίκτη με ένα πέπλο διαφάνειας στο περίγραμμα του mesh του. Η προσθήκη και η απομάκρυνση του υλικού πραγματοποιούνται μέσω του node SetOverlay Material, στα events AddStealthMaterial και RemoveStealthMaterial αντίστοιχα. Η Εικόνα 136 απεικονίζει το οπτικό εφέ όπως αυτό εμφανίζεται εντός του παιχνιδιού, σε περίπτωση κατά την οποία ο παίκτης έχει εισέλθει και κρύβεται εντός ενός Stealth Spot. Το εφέ λειτουργεί ως οπτική ένδειξη της κατάστασης απόκρυψης του παίκτη, συμβάλλοντας τόσο στη διάδραση, όσο και στην κατανόηση του gameplay από την πλευρά του χρήστη.



Εικόνα 136. Conceal Overlay Material

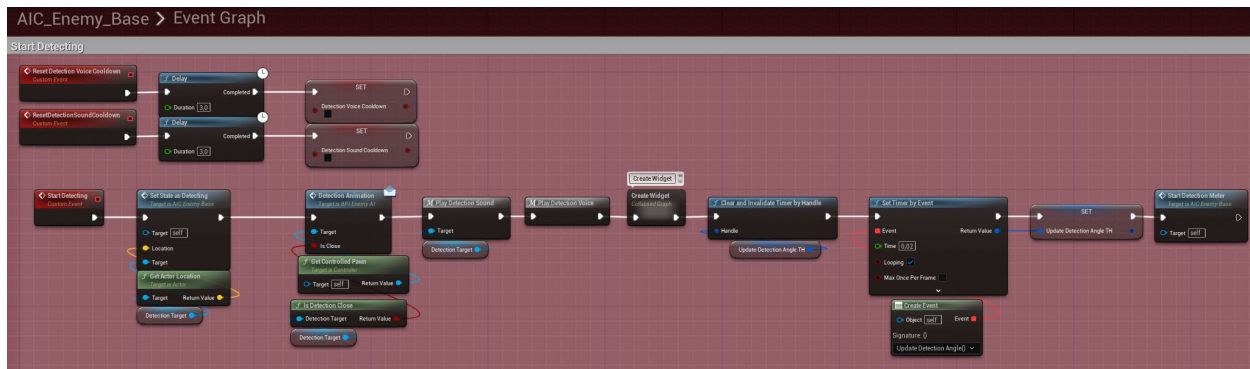
RevealPlayer (Function): Το function RevealPlayer εκτελεί τις απαραίτητες ενέργειες για την επαναφορά του παίκτη σε κατάσταση ορατότητας από τους Agents και ενεργοποιείται όταν ο παίκτης εξέρχεται από ένα StealthSpot. Όπως και το function ConcealPlayer, ενσωματώνεται στο blueprint του παίκτη. Η διαδικασία περιλαμβάνει τη μείωση του αριθμού των StealthSpots στα οποία βρίσκεται ο παίκτης και τον έλεγχο της τιμής της μεταβλητής HowManyBoxes. Εάν ο αριθμός αυτός είναι μεγαλύτερος του μηδενός, αυτό σημαίνει ότι ο παίκτης βρίσκεται ακόμα σε κάποιο άλλο StealthSpot, οπότε το function δεν εκτελεί περαιτέρω ενέργειες. Εάν ο αριθμός είναι μηδέν, διακόπτεται η συνεχής εκτέλεση του StealthTimer, καθώς η λειτουργία αυτή δεν είναι πλέον απαραίτητη και θα επιβαρύνει τους υπολογιστικούς πόρους. Επιπλέον, η μεταβλητή IsHidden ορίζεται σε false και το ειδικό εφέ απόκρυψης αφαιρείται.



Εικόνα 137. RevealPlayer

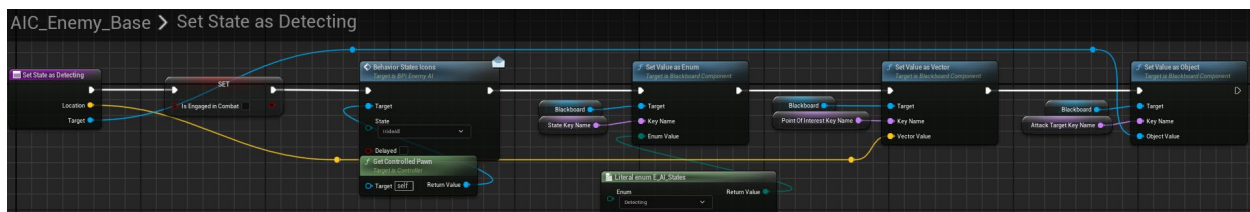
8.6.2. Detection

Start Detecting (Custom Event): Το Start Detecting είναι υπεύθυνο για την πυροδότηση των απαραίτητων ενεργειών κατά την έναρξη μιας διαδικασίας ανίχνευσης. Υλοποιείται εντός του AI Controller και καλείται όταν ικανοποιούνται οι συνθήκες έναρξης της διαδικασίας ανίχνευσης, όπως περιγράφονται στο Custom Event OnPlayerSeen. Το function αρχικά μεταβάλλει τον Agent στο state Detecting μέσω του function SetStateAsDetecting και πυροδοτεί το function Detection Animation το οποίο περιέχει τις ενέργειες αντίδρασης του Agent, όπως animation αντίδρασης και οπτικοακουστικά εφέ που απεικονίζουν τη μεταβολή του Agent στο state σύμφωνα με τον τύπο του Agent. Στη συνέχεια, αναπαράγεται στον παίκτη ένας χαρακτηριστικός ήχος που σηματοδοτεί πως βρίσκεται υπό ανίχνευση, και ο ήχος που αναπαράγεται είναι διαφορετικός για κάθε τύπου Agent, δηλαδή άλλον ήχο κάνει ένα Detection από έναν Agent τύπου Ranged και άλλο αν είναι τύπου Android. Έπειτα, δημιουργείται ένα User Widget στην οθόνη του παίκτη, το οποίο παρέχει με συνεχή ενημέρωση την κατεύθυνση από την οποία προέρχεται το Detection καθώς και το βαθμό ανίχνευσης μέχρι ο παίκτης να γίνει πλήρως αντιληπτός από τον Agent. Η ενημέρωση της κατεύθυνσης επιτυγχάνεται με την επαναλαμβανόμενη εκτέλεση του Custom Event UpdateDetectionAngle, το οποίο εκτελείται κάθε 0.02 δευτερόλεπτα σύμφωνα με το node SetTimerbyEvent. Με αυτόν τον τρόπο, κάθε καινούργια ανίχνευση θα προσθέσει από ένα τέτοιο εικονίδιο στην οθόνη του παίκτη, παρέχοντας πληροφορίες για την κατάσταση της απόκρυψής του. Τέλος, το Custom Event Start Detection Meter καλείται, ώστε να εκκινηθεί η ενημέρωση των μεταβλητών που χρησιμοποιούνται στη μέτρηση του βαθμού της ανίχνευσης.



Εικόνα 138. Start Detecting

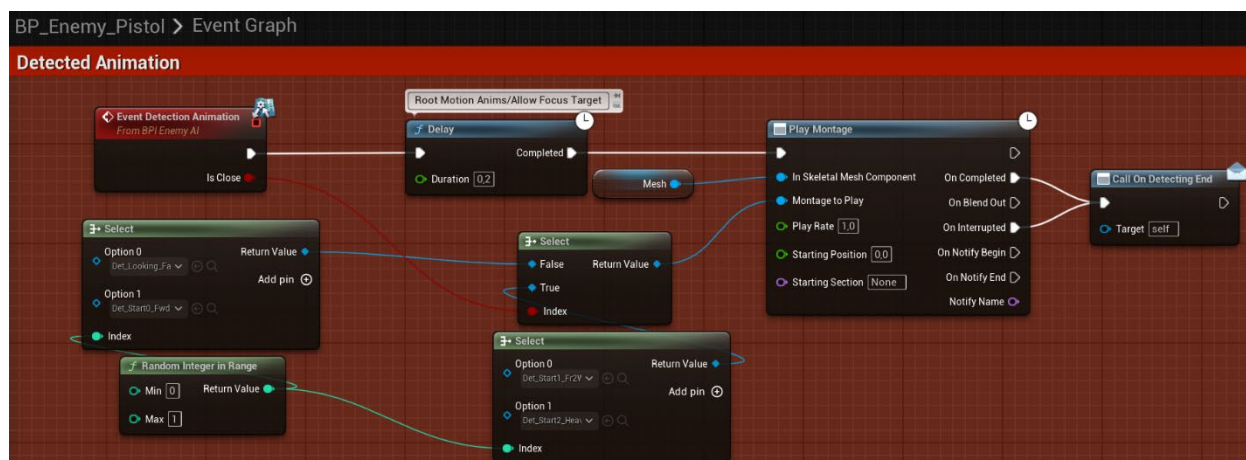
SetStateAsDetecting (Function): Το function εκτελεί τις απαραίτητες ενέργειες για τη μεταβολή ενός Agent στο Detecting State, και υλοποιείται εντός του AI Controller. Δέχεται ως είσοδο το reference και την τοποθεσία του παίκτη, ώστε να εκχωρήσει το περιεχόμενό τους στα blackboard keys AttackTarget(Player) και PointOfInterest μέσω των nodes SetValueAsObject και SetValueAsVector. Ακόμη, μεταβάλλει το State key σε Detecting μέσω του SetValueAsEnum, προκειμένου το δέντρο να επαναξιολογηθεί και τα tasks του state να εκκινηθούν. Επιπλέον, η μεταβλητή IsEngagedInCombat (Boolean) παίρνει την τιμή false καθώς ο Agent δεν ενεργεί πλέον επιθετικά προς τον παίκτη. Τέλος, τα εικονίδια που απεικονίζουν τις καταστάσεις Seeking και Investigating αποκρύπτονται, σε περίπτωση που ο Agent βρισκόταν σε ένα από αυτά τα states.



Εικόνα 139. SetStateAsDetecting

Detection Animation (Function): Το function εκτελεί την αναπαραγωγή των οπτικοακουστικών εφέ για κάθε τύπου Agent. Υλοποιείται σε κάθε child blueprint, προκειμένου κάθε Agent να αντιδράσει σύμφωνα με τον τύπο του. Συγκεκριμένα, οι Agent τύπου Ranged αναπαράγουν ένα animation κατά το οποίο αντιδρούν με ταραχή, επιλέγοντας το κατάλληλο animation σύμφωνα με το αν η απόσταση του παίκτη από αυτούς είναι κοντινή ή όχι, όπως ορίζεται από το function IsDetectionClose. Με αυτόν τον τρόπο, αν ο παίκτης εισέλθει στο οπτικό πεδίο του Agent από κοντινή απόσταση, το animation που θα επιλεγεί θα απεικονίζει μεγαλύτερα επίπεδα ταραχής στις κινήσεις του. Για κάθε περίπτωση, δύο πιθανά εναλλακτικά animation μπορούν να εκτελεστούν σύμφωνα με μια τυχαία μεταβλητή. Σε αυτό το project, κάθε animation αντίδρασης απεικονίζει τον Agent να ταρασσεται και να κινείται προς τα πίσω καθώς αντιδρά στον εντοπισμό μιας εχθρικής παρουσίας. Για αυτόν τον λόγο, κάθε animation έχει την επιλογή Root

Motion ενεργοποιημένη, ώστε ο Agent να κινηθεί σύμφωνα με το εκάστοτε animation. Ωστόσο, αυτό μπορεί να προκαλέσει ανεπιθύμητη συμπεριφορά, καθώς ο Agent δεν προλαβαίνει να εστιάσει στην κατεύθυνση του παίκτη, σύμφωνα με τις ενέργειες που εκτελούνται στο Behavior Tree. Ως αποτέλεσμα, κάθε animation αναπαράγεται μετά από μια χρονική καθυστέρηση 0.2 δευτερολέπτων προκειμένου το task υπεύθυνο για την εστίαση του Agent στον παίκτη να εκτελεστεί και έπειτα το animation να αναλάβει τη συνέχεια της κίνησης.



Εικόνα 140. Detection Animation

Τέλος, το function ολοκληρώνεται με την κλήση του Event Dispatcher OnDetectingEnd ανεξαρτήτως του τύπου του Agent που το εκτελεί, προκειμένου να καταστεί δυνατό για τις ενέργειες που ακολουθούν το function Detection Animation και απαιτούν ρητά την ολοκλήρωση του να εκτελεστούν την κατάλληλη στιγμή.

Όσον αφορά στους Agents τύπου Android, η αναπαραγωγή animation αντίδρασης δεν ταιριάζει στον τύπο τους. Αντ' αυτού, το function αναπαράγει ένα ειδικό εφέ που δημιουργεί ένα εφέ τύπου ηλεκτρισμού στην επιφάνεια του mesh τους, σηματοδοτώντας ότι είδαν μια εχθρική παρουσία και ετοιμάζονται να ενεργήσουν. Η οπτική αναπαράσταση του Detection Meter και οι αντίστοιχες αντιδράσεις για κάθε τύπου Agent εντός του παιχνιδιού, απεικονίζονται στις εικόνες 141 και 142.

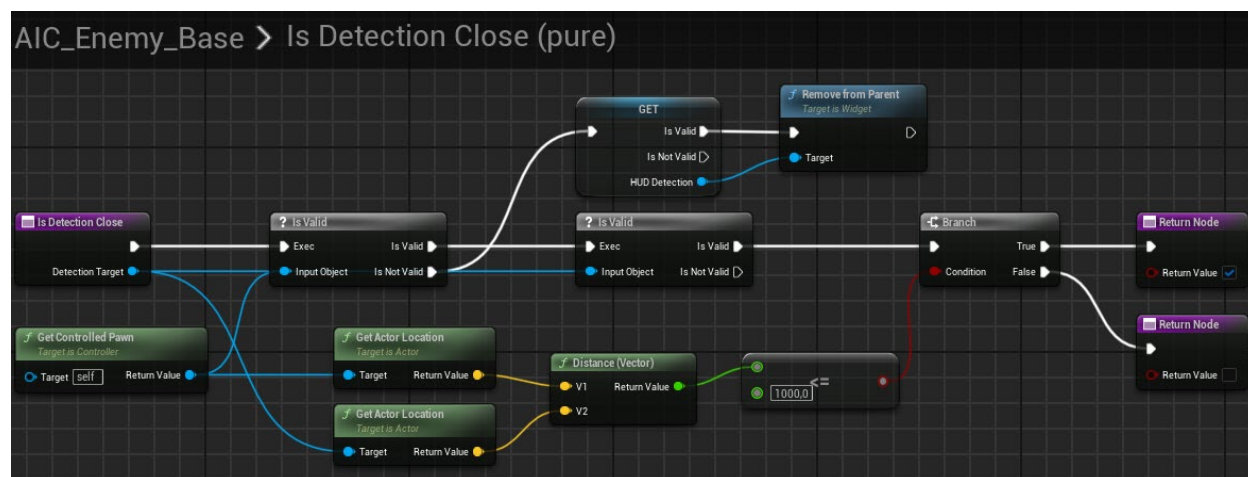


Εικόνα 141. Detection Reaction & Widget (Ranged)



Εικόνα 142. Detection Reaction & Widget (Agent)

IsDetectionClose (Pure Function): Το function υπολογίζει την απόσταση μεταξύ ενός Agent και ενός Actor και υλοποιείται εντός του AI Controller. Χρησιμοποιείται κατά την εκτέλεση των animation αντίδρασης όταν ένας Agent αντιληφθεί τον παίκτη ή έναν εξουδετερωμένο σύμμαχο Agent, προκειμένου να καθοριστεί το κατάλληλο animation αντίδρασης. Συγκεκριμένα, δέχεται έναν Actor ως είσοδο και συγκινεί τη διανυσματική του απόσταση με αυτή του Agent. Αν το αποτέλεσμα είναι μικρότερο από 10 μέτρα, τότε η απόσταση τους θεωρείται κοντινή και το function επιστρέφει μια μεταβλητή με τιμή true. Το function ορίζεται ως pure, καθώς κάνει μόνο υπολογισμούς και δεν μεταβάλλει το περιεχόμενο μεταβλητών.

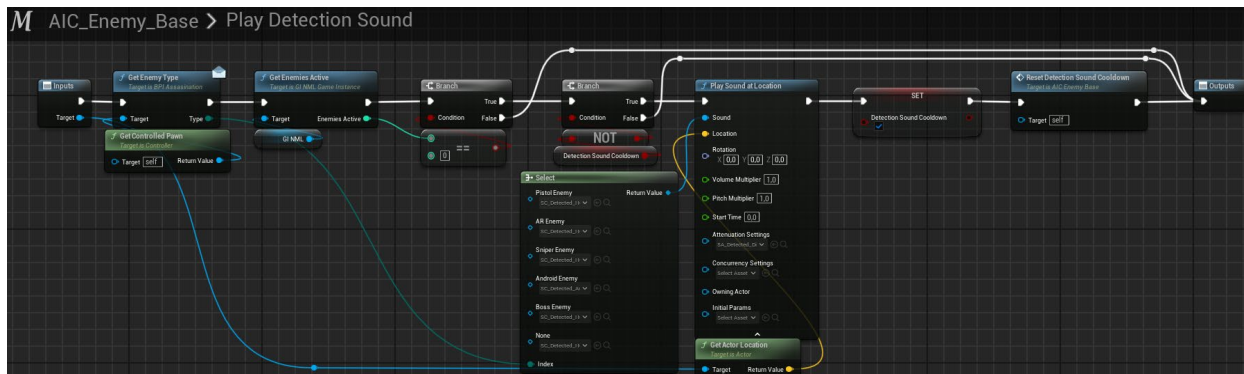


Εικόνα 143. IsDetectionClose

PlayDetectionSound (Macro): Το PlayDetectionSound αποτελεί ένα Macro υπεύθυνο για την αναπαραγωγή του χαρακτηριστικού ήχου που σηματοδοτεί την έναρξη της διαδικασίας ανίχνευσης του παίκτη από έναν Agent. Κάθε τύπος Agent συνοδεύεται από διαφορετικό ήχο, παρέχοντας στον παίκτη μια επιπλέον πληροφορία σχετικά με το ποιος τύπος εχθρού τον έχει αντιληφθεί. Αυτό επιτυγχάνεται μέσω του node Select, το οποίο επιτρέπει την αντιστοίχιση διαφορετικών ηχητικών αρχείων ανάλογα με τον τύπο του Agent. Η αναπαραγωγή του ήχου γίνεται με τη χρήση της PlaySoundAtLocation, η οποία παίζει τον ήχο στο σημείο όπου βρίσκεται ο Agent, χρησιμοποιώντας το αντίστοιχο Attenuation Asset που δημιουργήθηκε. Με αυτόν τον τρόπο παρέχεται στον παίκτη και η αίσθηση της κατεύθυνσης απ' όπου προήλθε ο ήχος, ενισχύοντας την αντίληψή του για το χώρο και το σημείο προέλευσης της ανίχνευσης.

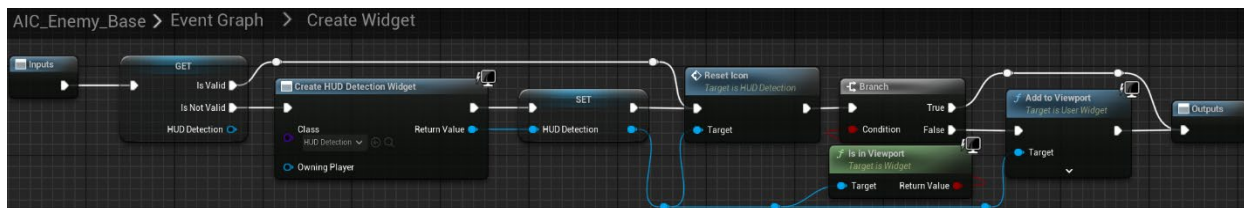
Στο πλαίσιο του συγκεκριμένου project, ο ήχος αυτός αναπαράγεται μόνο στην περίπτωση που κανένας Agent δεν έχει ενεργοποιηθεί εναντίον του παίκτη, δηλαδή όταν ο αριθμός των ενεργών Agents είναι μηδενικός, κάτι που ελέγχεται μέσω του function GetEnemiesActive. Με τον τρόπο αυτό διασφαλίζεται πως το ηχητικό feedback θα προσφέρει σαφές και ουσιαστικό νόημα στον παίκτη, σηματοδοτώντας την πρώτη στιγμή ανίχνευσής του. Επιπλέον, για την αποφυγή υπερβολικής ή συνεχόμενης αναπαραγωγής

του ίδιου ήχου σε σύντομα χρονικά διαστήματα, κάθε ήχος ενεργοποιεί έναν μηχανισμό Cooldown, εμποδίζοντας την άμεση επανάληψή του.



Εικόνα 144. PlayDetectionSound

Create Detection Widget (Collapsed Graph): Το συνεπτυγμένο γράφημα Create Detection Widget περιέχει τις εντολές δημιουργίας του εικονιδίου της ανίχνευσης που εμφανίζεται στην οθόνη του παίκτη. Υλοποιείται εντός του AI Controller και περιλαμβάνει τη δημιουργία του User Widget μέσω του node Create Hud Detection Widget και την πρόσθεσή του στην οθόνη του παίκτη μέσω του node AddtoViewport. Επιπλέον, μέσω των ελεγχών IsValid και IsInViewPort διασφαλίζεται πως το εικονίδιο δεν έχει ήδη δημιουργηθεί και πως δεν έχει ήδη προστεθεί στην οθόνη του παίκτη, αποφεύγοντας έτσι τη δημιουργία διπλότυπων εικονιδίων για τον ίδιο Agent στην περίπτωση μιας σύντομης επανενεργοποίησης του state.

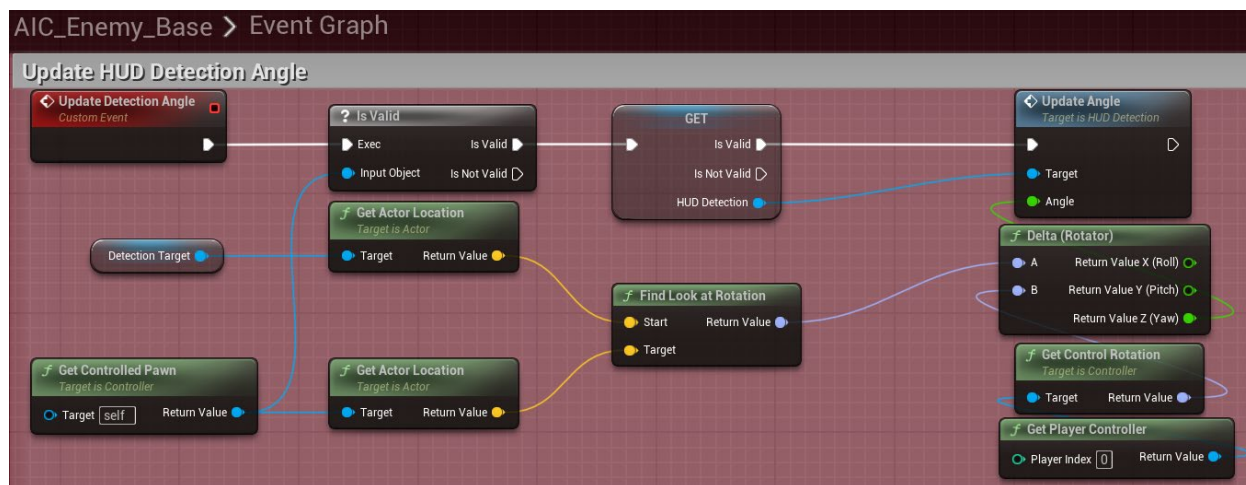


Εικόνα 145. Create Detection Widget

UpdateDetectionAngle (CustomEvent): Το UpdateDetectionAngle είναι ένα Custom Event υπεύθυνο για τη συνεχή ενημέρωση της κατεύθυνσης του εικονιδίου ανίχνευσης, και υλοποιείται στο blueprint του AI Controller. Η λειτουργία του ξεκινά με τον έλεγχο IsValid, ο οποίος διασφαλίζει πως ο Agent που επιχειρεί την ενημέρωση είναι ακόμη ενεργός και δεν έχει διαγραφεί από το επίπεδο, αποτρέποντας έτσι πιθανό σφάλμα.

Η βασική λειτουργία του event είναι να ενημερώσει την οπτική κατεύθυνση του εικονιδίου που υποδεικνύει στον παίκτη από ποια κατεύθυνση προέρχεται η ανίχνευση. Για να επιτευχθεί αυτό, το

UpdateDetectionAngle καλεί το Custom Event UpdateAngle, το οποίο έχει υλοποιηθεί εντός του Widget Blueprint του εικονιδίου ανίχνευσης. Το UpdateAngle υπολογίζει τη γωνία μεταξύ του Agent και του παίκτη χρησιμοποιώντας το node FindLookAtRotation, από όπου εξάγεται η μεταβλητή Angle (τύπου Float). Στη συνέχεια, η γωνία αυτή χρησιμοποιείται ως είσοδος στο node SetRenderTransformAngle, το οποίο μετατρέπει την κατεύθυνση σε οπτική περιστροφή του εικονιδίου.



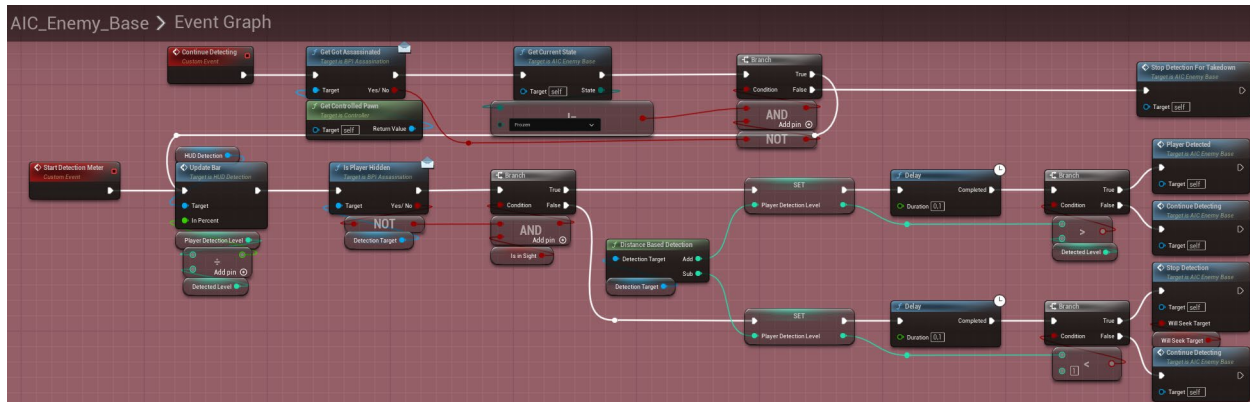
Εικόνα 146. UpdateDetectionAngle

Με αυτόν τον τρόπο, το εικονίδιο ανίχνευσης αποδίδεται δυναμικά σε ένα σημείο της ακτίνας γύρω από τον παίκτη και ενημερώνεται συνεχώς ώστε να δείχνει με ακρίβεια τη θέση του Agent που εκτελεί τη διαδικασία ανίχνευσης, προσφέροντας έτσι στον παίκτη χρήσιμη και ενστικτώδη πληροφορία σχετικά με τον κίνδυνο που τον πλησιάζει.

Start Detection Meter (Custom Event): Το Start Detection Meter αποτελεί Custom Event υπεύθυνο για τη διαχείριση του συνόλου των ενεργειών που σχετίζονται με τη διαδικασία ανίχνευσης του παίκτη από έναν Agent. Η υλοποίησή του πραγματοποιείται εντός του AI Controller και περιλαμβάνει την παρακολούθηση του βαθμού ανίχνευσης του παίκτη, την ενημέρωση του εικονιδίου ανίχνευσης με την τρέχουσα τιμή και την πυροδότηση ενεργειών σύμφωνα με τον βαθμό ανίχνευσης και την κατάσταση του παιχνιδιού.

Η διαδικασία βασίζεται στη χρήση δύο βασικών μεταβλητών:

- *PlayerDetectionLevel (Integer):* Εκφράζει το βαθμού ανίχνευσης του παίκτη τη δεδομένη στιγμή.
- *Detected Level (Float):* Αναπαριστά το όριο που, αν ξεπεραστεί, σημαίνει ότι ο παίκτης έχει ανιχνευθεί πλήρως. Στην παρούσα υλοποίηση, έχει την τιμή 50, ωστόσο, μπορεί να μεταβάλλεται δυναμικά σύμφωνα με τον τύπο του Agent ή άλλους παράγοντες σε μελλοντική εξέλιξη του project.



Εικόνα 147. Start Detection Meter

Κατά την εκκίνηση της ανίχνευσης το event ενημερώνει την μπάρα προόδου του detection widget, μέσω του Custom Event Update Bar που υλοποιείται εντός του blueprint του UI. Στη συνέχεια, γίνεται έλεγχος μέσω του function IsPlayerHidden, που εξετάζει αν ο παίκτης βρίσκεται εντός ενός StealthSpot και της μεταβλητής IsInSight, που προσδιορίζει αν ο παίκτης βρίσκεται εντός του οπτικού πεδίου του Agent. Αναλόγως των παραπάνω, αν ο παίκτης είναι ορατός, τότε το PlayerDetectionLevel αυξάνεται. Αν δεν είναι (ή είναι κρυμμένος), τότε μειώνεται σταδιακά, δημιουργώντας μια ρεαλιστική αίσθηση απώλειας του στόχου από τον Agent. Η ενημέρωση του detection level βασίζεται στη συνεχή επανάληψη του Event (με Delay 0.1 δευτερολέπτων) και στο αποτέλεσμα του function DistanceBasedDetection, το οποίο προσαρμόζει την αύξηση του detection ανάλογα με την απόσταση μεταξύ παίκτη και Agent. Καθ' όλη τη διάρκεια, η κατάσταση ανίχνευσης συγκρίνει τις τιμές των μεταβλητών PlayerDetectionLevel και DetectedLevel.

Αν η τιμή της PlayerDetectionLevel γίνει μεγαλύτερη ή ίση της DetectedLevel, τότε ο παίκτης έχει γίνει πλήρως αντιληπτός και το Custom Event PlayerDetected ενεργοποιείται:

- Σταματούν όλα τα ενεργά functions που δεν είναι πλέον απαραίτητα
- Το εικονίδιο ανίχνευσης γίνεται κόκκινο και αναβοσβήνει ώστε να σηματοδοτήσει την ανίχνευση
- Στη συνέχεια διαγράφεται μετά από λίγη ώρα
- Η μεταβλητή PlayerDetectionLevel μηδενίζεται
- Ο Agent μεταβαίνει στο Attacking State

Αν η τιμή της PlayerDetectionLevel γίνει μικρότερη από 1, τότε εκτελείται το Custom Event Stop Detection:

- Η μεταβλητή PlayerDetectionLevel μηδενίζεται
- Το εικονίδιο ανίχνευσης αφαιρείται
- Ο Agent επιστρέφει στο Passive State

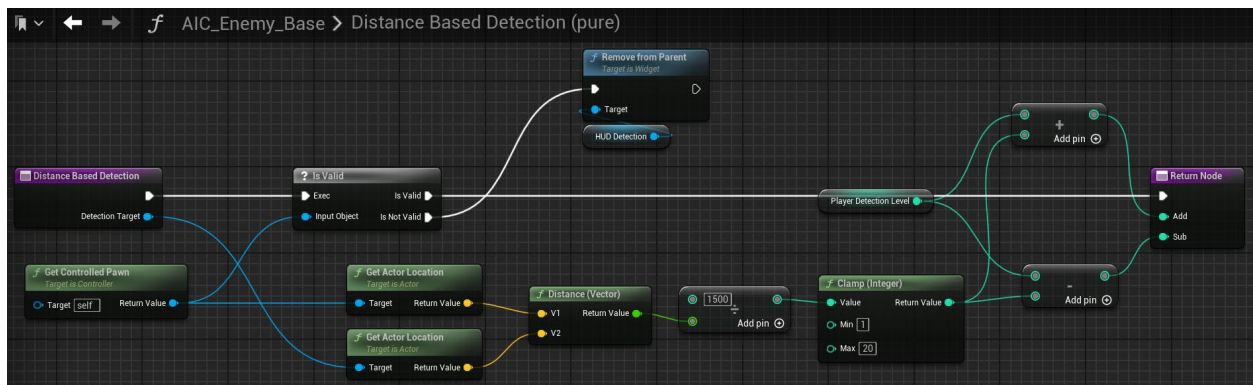
Αν δεν ισχύει κανένα από τα δύο παραπάνω, το Custom Event Continue Detecting συνεχίζει τη διαδικασία, επανεκτελώνοντας τα βήματα από το Custom Event Detecting. Το γράφημα των functions παρουσιάζεται στη εικόνα 148.



Το Start Detection Meter περιλαμβάνει και επιπλέον ελέγχους ώστε να διασφαλιστεί η σταθερότητα της συμπεριφοράς του Agent σε ειδικές περιπτώσεις στις οποίες ο Agent δέχεται επίθεση Takedown κατά τη διάρκεια της ανίχνευσης ή είναι Android και δέχεται επίθεση μέσω του αντικειμένου DataWorm. Σε αυτές τις περιπτώσεις είναι απαραίτητο να διακοπεί κάθε ενεργή διαδικασία ανίχνευσης, καθώς ο Agent θα μπορούσε να μεταβεί στο Attacking State παρά το γεγονός ότι είναι ήδη εξουδετερωμένος, οδηγώντας σε σφάλματα στην κινησιολογία του Agent, αναντιστοιχία μεταβλητών (όπως ενεργοί Agents), ή παράλληλη εκτέλεση αντικρουόμενων ενεργειών μεταξύ των States.

223

DistanceBasedDetection (Pure Function): Το function είναι υπεύθυνο για τον καθορισμό της τιμής που θα προστεθεί ή αφαιρεθεί στη μεταβλητή PlayerDetectionLevel σύμφωνα με την απόσταση μεταξύ του παίκτη και του Agent. Υλοποιείται στο AI Controller ως pure function και χρησιμοποιείται στο Custom Event Start Detection Meter κατά την ενημέρωση της μεταβλητής. Το function υπολογίζει τη διανυσματική απόσταση των δύο Actors και τη διαιρεί με την τιμή 1500. Το αποτέλεσμα κανονικοποιείται μεταξύ των αριθμών 1 και 20 και προστίθεται ή αφαιρείται στη μεταβλητή PlayerDetectionLevel. Οι τιμές που ορίστηκαν διασφαλίζουν πως οι τιμές αυξομειώσης προσομοιώνουν το χρόνο αντίληψης ενός Actor με ρεαλιστικό τρόπο διατηρώντας ταυτόχρονα μια απόκλιση που εξυπηρετεί το gameplay.



Εικόνα 149. DistanceBasedDetection

8.6.3. Handle Sensed Sight

OnPlayerSeen (Custom Event): Το Custom Event OnPlayerSeen είναι υπεύθυνο για τη διαχείριση όλων των πιθανών αντιδράσεων ενός Agent όταν ένας Actor εισέρχεται ή εξέρχεται από το οπτικό του πεδίο. Η υλοποίησή του πραγματοποιείται στον AI Controller και δέχεται δύο βασικές εισόδους:

- Actor (Actor Object Reference): ο χαρακτήρας που ανιχνεύτηκε
- Sensed (Boolean): δηλώνει εάν ο Actor είναι αυτή τη στιγμή ορατός (true) ή όχι (false)

Κάθε φορά που ενεργοποιείται η διαδικασία HandleSensedSight, το event OnPlayerSeen εκτελείται, αξιοποιώντας το reference του Actor που ανιχνεύθηκε.

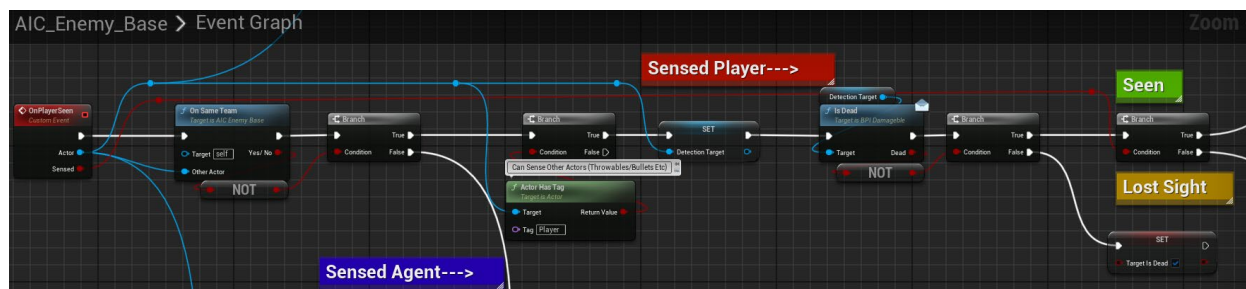
Η κύρια λειτουργία του event περιλαμβάνει:

- Την κατηγοριοποίηση του Actor ως παίκτη ή σύμμαχο Agent
- Την ενεργοποίηση της κατάλληλης αντίδρασης του Agent, ανάλογα με την περίπτωση

Το πρόβλημα που προκύπτει είναι ότι το Sight Sense του AI συστήματος κάνει register τον παίκτη μόνο κατά την είσοδό του στο οπτικό πεδίο του Agent. Αυτό σημαίνει ότι το OnPlayerSeen πυροδοτείται μία φορά, όταν το Sensed γίνεται true. Ωστόσο, σε περιπτώσεις όπως όταν ο παίκτης εξέρχεται από ένα Stealth Spot (ενώ βρίσκεται ακόμα εντός του οπτικού πεδίου), απαιτείται νέα αξιολόγηση της κατάστασης. Το σύστημα Sense από μόνο του δε μπορεί να εντοπίσει τέτοιες μεταβολές, με αποτέλεσμα να μην ενεργοποιείται ξανά το OnPlayerSeen και άρα να μη γίνεται η απαραίτητη μετάβαση State του Agent.

Για να αντιμετωπιστεί αυτό, το project υλοποιεί βοηθητικά functions που ελέγχουν σε συνεχή βάση την κατάσταση απόκρυψης του παίκτη. Αυτοί οι έλεγχοι πυροδοτούν εκ νέου την αξιολόγηση της κατάστασης και επιτρέπουν στον Agent να μεταβεί στο κατάλληλο State όταν απαιτείται, ακόμα κι αν δεν έχει αλλάξει η ορατότητα μέσω του Sight Sense.

Αναγνώριση του Actor: Η υλοποίηση ξεκινά με τον έλεγχο της ομάδας του Actor που προκάλεσε το ερέθισμα μέσω του function OnSame Team, το οποίο συγκρίνει τον αριθμό ομάδας που έχει ανατεθεί στον Agent και τον Actor που προκάλεσε το ερέθισμα και επιστρέφει εάν είναι σύμμαχοι. Εάν δεν είναι, ο έλεγχος για το αν ο Actor διαθέτει το Tag "Player" εξασφαλίζει πως οι ενέργειες που ακολουθούν θα εκτελεστούν για τον παίκτη. Ο έλεγχος είναι απαραίτητος καθώς ακόμη και αν ο Actor δεν είναι στην ίδια ομάδα, θα μπορούσε να είναι οποιοσδήποτε Actor που εισέρχεται στο οπτικό πεδίο του Agent, όπως για παράδειγμα το αντικείμενο Bottle. Στην περίπτωση που ο actor είναι σύμμαχος Agent, ακολουθεί έλεγχος για το αν ο Agent είναι εξουδετερωμένος, ώστε ο Agent να μεταβεί στο Seeking State.



Εικόνα 150. OnPlayerSeen (identify actor)

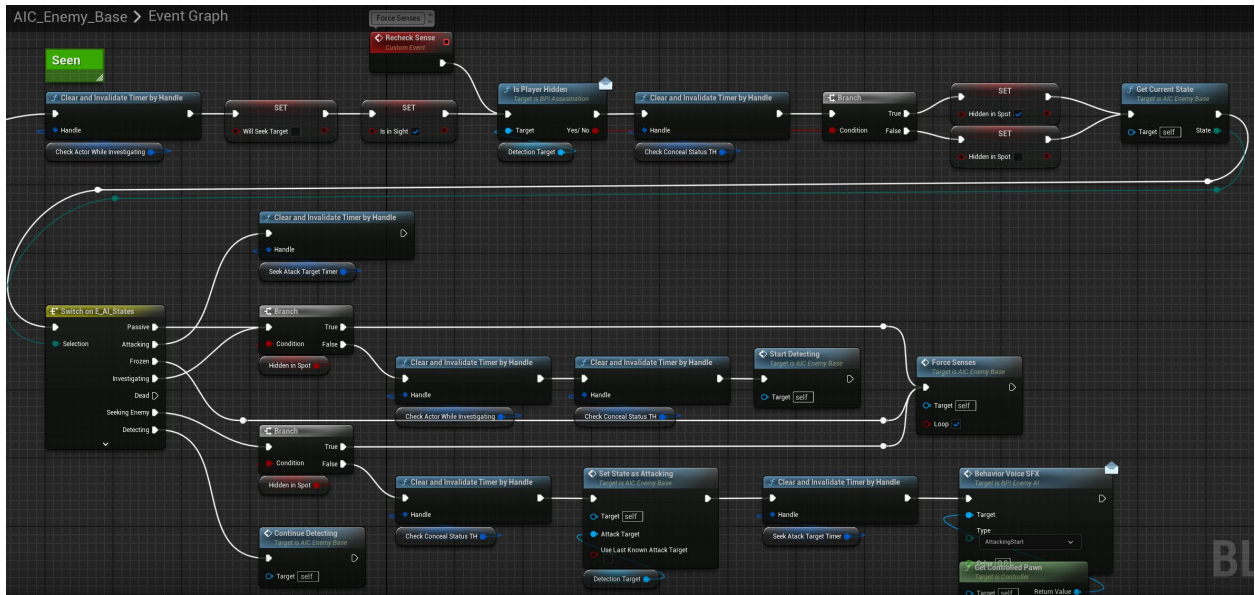
Αντιμετώπιση αναγνώρισης του παίκτη

Αν ο Actor που εισήλθε στο οπτικό πεδίο του Agent είναι ο παίκτης, το event εκχωρεί στη μεταβλητή DetectionTarget το reference του παίκτη, ώστε να χρησιμοποιηθεί σε περαιτέρω ελέγχους και ενέργειες, καθώς και να χρησιμοποιηθεί εντός των Behavior Trees. Έπειτα, πραγματοποιείται έλεγχος της κατάστασης εξουδετέρωσης του παίκτη μέσω του function IsDead, ώστε να εξασφαλιστεί πως οι μετέπειτα ενέργειες θα εκτελεστούν μόνο όταν ο παίκτης είναι ζωντανός. Στη συνέχεια, η συνθήκη με προϋπόθεση τη Sensed (Boolean) διαχωρίζει τις ενέργειες που εκτελούνται όταν ο παίκτης εισέρχεται ή εξέρχεται από το οπτικό πεδίο του Agent.

Player Seen: Το Event αναθέτει στη μεταβλητή *IsInSight* (Boolean) την τιμή true και το function *IsPlayerHidden* επιστρέφει την κατάσταση απόκρυψης του παίκτη. Η τιμή που επιστρέφει το function ανατίθεται στην *HiddenInSpot* (Boolean), η οποία αντιπροσωπεύει την κατάσταση απόκρυψης του παίκτη εντός του AI Controller. Στη συνέχεια, ανακτάται το State του Agent από το function *GetCurrentState* και μέσω του node *SwitchOnEnum* παρέχονται τα execution pins για κάθε κατάσταση.

- Αν ο παίκτης είναι κρυμμένος σε ένα *StealthSpot* και ο Agent βρίσκεται σε ένα από τα States *Passive*, *Investigating* ή *Seeking*, τότε η κατάσταση απόκρυψης του παίκτη ελέγχεται επαναλαμβανόμενα όσο ο παίκτης βρίσκεται εντός του οπτικού του πεδίου. Αυτό επιτυγχάνεται με το Custom Event *Force Senses*, το οποίο εκκινεί το function *CheckPlayerConcealStatus* που επαναλαμβάνεται κάθε 0.02 δευτερόλεπτα μέσω της χρήσης *SetTimerbyFunction*. Το function ελέγχει την κατάσταση απόκρυψης του παίκτη και μέσω του Custom Event *Recheck Sense*, πυροδοτεί την επαναξιολόγηση της κατάστασης σε περίπτωση που ο παίκτης παύσει να είναι κρυμμένος. Το function *CheckPlayerConcealStatus* εκτελείται έως ότου ο παίκτης να αποκαλυφθεί ή να βγει εκτός του οπτικού πεδίου του Agent.
- Αν ο παίκτης δεν είναι κρυμμένος σε ένα *StealthSpot* και ο Agent βρίσκεται στο *Passive* ή στο *Investigating State*, τα functions συνεχούς ελέγχου της κατάστασης απόκρυψης διακόπτονται, και η διαδικασία *Detection* εκκινείται μέσω του Custom Event *Start Detection*. Στην περίπτωση που ο Agent βρίσκεται στο *Seeking State*, ο Agent μεταβαίνει απευθείας στο *Attacking State*, μέσω του *SetStateAsAttacking*. Τέλος, αν ο Agent βρίσκεται στα States *Attacking* ή *Detecting*, τότε καμία ενέργεια δε λαμβάνει χώρα, δηλαδή ο Agent συνεχίζει την επίθεση ή τη διαδικασία αναγνώρισης αντίστοιχα.

Το σχετικό διάγραμμα παρουσιάζεται στην εικόνα 151.



Εικόνα 151. OnPlayerSeen (Player Seen)

Lost Sight: Στην περίπτωση που ο παίκτης εξέλθει από το οπτικό πεδίο του Agent, τότε οι διαδικασίες ελέγχου της κατάστασης απόκρυψης διακόπτονται, και το τρέχον State που εκτελεί ο Agent ανακτάται ώστε να πραγματοποιηθεί η κατάλληλη ενέργεια. Σε αυτό το σημείο, ο Agent μπορεί να βρίσκεται στα States Attacking ή Detecting State, καθώς έχει αντιληφθεί ήδη τον παίκτη.

- Στην περίπτωση του Attacking, το Custom Event StartSeeking θα μεταβάλει τον Agent στο Seeking State, μόνο μετά την πάροδο 3 δευτερολέπτων κατά τα οποία ο παίκτης είναι εκτός του οπτικού πεδίου του Agent. Αυτό επιτυγχάνεται με το node SetTimerbyEvent, το οποίο πυροδοτεί το StartSeeking σύμφωνα με τη μεταβλητή TimetoSeekAfterLoosingSight (float). Με αυτό τον τρόπο διασφαλίζεται πως ένας Agent δεν θα μεταβεί αμέσως στο Seeking State, τη στιγμή που ο παίκτης εξέλθει του οπτικού του πεδίου.
- Στην περίπτωση του Detecting, αν ο βαθμός της αναγνώρισης είναι μικρότερος από 40 μονάδες στις 50, σύμφωνα με τη μεταβλητή PlayerDetectionLevel(Integer), τότε η ανίχνευση συνεχίζεται με αποτέλεσμα το επίπεδο αναγνώρισης να αρχίσει να μειώνεται. Σε αντίθετη περίπτωση, ο Agent θα μεταβεί στο Seeking State αναζητώντας τον παίκτη, καθώς το επίπεδο αναγνώρισης έφτασε σε υψηλά επίπεδα.

Το σχετικό διάγραμμα παρουσιάζεται στην εικόνα 152.

Στη συνέχεια, μέσω της χρήσης των `GetCurrentState` και `SwitchOnEnum nodes`, διασφαλίζεται πως η μετάβαση στην κατάσταση `Seeking` θα πραγματοποιηθεί μόνο αν ο `Agent` βρίσκεται ήδη σε μία από τις καταστάσεις `Passive` ή `Investigating`. Αν ο `Agent` βρίσκεται σε οποιαδήποτε άλλη κατάσταση, η απόκριση δεν ενεργοποιείται. Αφού επιβεβαιωθούν οι παραπάνω συνθήκες, ο εξουδετερωμένος `Agent` ορίζεται ως σημείο εστίασης, και καλείται το function `SeenDeadTeammate`. Το συγκεκριμένο function είναι υπεύθυνο για την εκτέλεση των οπτικοακουστικών εφέ που συνοδεύουν την αντίδραση του `Agent` και περιλαμβάνει `animation` αντίδρασης και ηχητικά εφέ για `Agents` τύπου `Ranged` και οπτικό εφέ ηλεκτρισμού για τους `Android`. Εφόσον τα εφέ αυτά απαιτούν ένα χρονικό διάστημα για την ολοκλήρωσή τους, προτού ο `Agent` αλλάξει κατάσταση, καλείται το `Event Dispatcher SeenDeadTeammate`, ακολουθώντας την ίδια λογική συγχρονισμού με αυτή που χρησιμοποιείται για την όπλιση και αφόπλιση των `Agents`.

Τέλος, πραγματοποιείται ένας ακόμη έλεγχος κατάστασης, ώστε να επιβεβαιωθεί πως ο `Agent` παραμένει σε `Passive` ή `Investigating` κατάσταση και δεν έχει μεταβεί σε άλλη, όπως το `Detecting`, εξαιτίας ενδεχόμενης εμφάνισης του παίκτη στο οπτικό του πεδίο κατά τη διάρκεια της προηγούμενης διαδικασίας. Εφόσον όλα τα παραπάνω ισχύουν, ο `Agent` μεταβαίνει στην κατάσταση `Seeking`, με στόχο τη διερεύνηση των αιτιών εξουδετέρωσης στην περιοχή όπου εντοπίστηκε ο πεσμένος σύμμαχος.

8.7. Dead State

Η κατάσταση `Dead` υποδηλώνει την εξουδετέρωση ενός `Agent`, είτε μέσω της ενέργειας `Takedown` είτε λόγω μηδενισμού των πόντων ζωής του. Στο συγκεκριμένο state, ο `Agent` παύει να εκτελεί οποιαδήποτε ενέργεια και τίθεται εκτός λειτουργίας. Το `Dead State` τοποθετείται στην αρχή του `Behavior Tree` κάθε τύπου `Agent`, ώστε να διακόπτει άμεσα την περαιτέρω εκτέλεση του δέντρου, καθιστώντας το πλήρως αδρανές μέχρι την αποδέσμευση του `Agent` τόσο από το δέντρο όσο και από το `AI Controller`. Το μοναδικό task που περιέχει είναι η εκκαθάριση του σημείου εστίασης, η οποία εκτελείται κυκλικά έως τον τερματισμό του δέντρου.

Οι ενέργειες που συνοδεύουν την είσοδο ενός `Agent` στο `Dead State` περιλαμβάνουν:

- Την ακινητοποίησή του
- Την ανάθεση κατάλληλων τιμών σε μεταβλητές που αφορούν την κατάσταση του `Agent` (π.χ. `IsDead`, `IsEngagedInCombat`)
- Την αποδέσμευση του `Agent` από τα `behavior-related assets`, όπως το `Behavior Tree` και ο `AI Controller`, ώστε να ελαχιστοποιηθεί η χρήση υπολογιστικών πόρων

Οι ενέργειες αυτές είναι καταναμεμημένες μεταξύ του `blueprint` του `AI Controller` και του `Parent Blueprint` των `Agents`.

Πολυπλοκότητα και Υλοποίηση

Η αποδέσμευση του Agent από τον AI Controller, σε συνδυασμό με την ενεργοποίηση του Dead State εντός του Behavior Tree, αποτελεί κρίσιμο σημείο για την ορθή λειτουργία της εξουδετέρωσης. Χωρίς την κατάλληλη απενεργοποίηση, ο Agent μπορεί να διατηρήσει ενεργά components, όπως το AI Perception, επιτρέποντάς του να συνεχίσει να αντιλαμβάνεται ερεθίσματα και να ενεργοποιεί functions που δεν ελέγχουν την κατάστασή του. Ως αποτέλεσμα, μπορεί να εκτελεστούν ενέργειες που δε συνάδουν με την εξουδετέρωση του Agent.

Επιπλέον, ιδιαίτερη προσοχή απαιτείται στον συγχρονισμό μεταβλητών και ενεργειών αποδέσμευσης. Για παράδειγμα, ένας Agent που έχει ήδη μεταβεί στο Attacking State θεωρείται απρόσβλητος σε Takedown. Αν, ωστόσο, δεν έχουν ανατεθεί εγκαίρως οι σωστές τιμές σε μεταβλητές όπως CanBeTakedowned, το σύστημα ενδέχεται να επιτρέψει την εκτέλεση της ενέργειας, οδηγώντας τον Agent σε εσφαλμένη μετάβαση στο Dead State. Αντιστρόφως, η καθυστέρηση στην ανάθεση τιμών μετά την εξουδετέρωση μπορεί να προκαλέσει εσφαλμένη επαναφορά του Agent σε ενεργή κατάσταση, π.χ. επιστροφή στο Attacking State, αν το Behavior Tree συνεχίσει να εκτελείται.

Ένα επιπλέον στοιχείο πολυπλοκότητας αποτελεί η διαφορετική φύση των αιτιών εξουδετέρωσης:

- Στην περίπτωση Takedown, η εξουδετέρωση εκτελείται άμεσα και δεν χρησιμοποιείται το BPC_Damage component
- Στην περίπτωση μηδενισμού των πόντων ζωής, το BPC_Damage component πυροδοτεί αυτόματα την κατάλληλη ακολουθία ενεργειών

Για την ορθή διαχείριση και των δύο περιπτώσεων δημιουργήθηκε το βοηθητικό function DeathWrapper, το οποίο ενσωματώνει το σύνολο των ενεργειών εξουδετέρωσης. Δέχεται ως παράμετρο τη μεταβλητή IsByTakedown (boolean), η οποία καθορίζει τη ροή εκτέλεσης. Αν η εξουδετέρωση προήλθε από Takedown, τότε η μεταβλητή παίρνει την τιμή true, και σε αντίθετη περίπτωση false. Το DeathWrapper αναλαμβάνει την ενεργοποίηση των κατάλληλων animations, την ενημέρωση των μεταβλητών, και την αποδέσμευση από το AI Controller και το Behavior Tree. Ο διαχωρισμός αυτός εξασφαλίζει ότι η ακολουθία ενεργειών εκτελείται σωστά ανεξαρτήτως αιτίας.

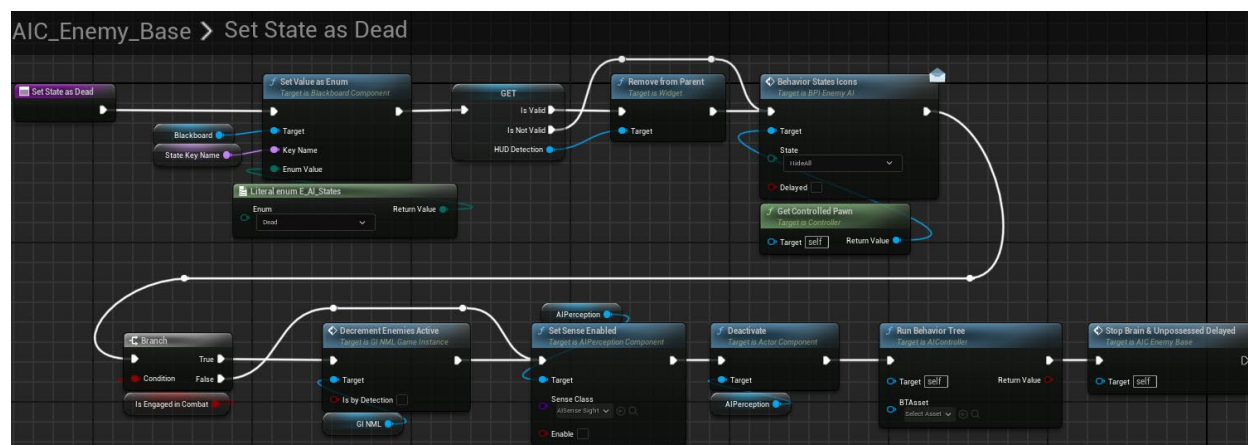
Πιθανές Μεταβάσεις

Η μετάβαση ενός Agent στο Dead State μπορεί να προκληθεί από δύο ενέργειες του παίκτη: Εκτέλεση Takedown, με προϋπόθεση ο Agent να μην βρίσκεται στο Attacking State ή τον μηδενισμό των πόντων ζωής του Agent, ανεξαρτήτως της τρέχουσας κατάστασης που βρίσκεται. Η είσοδος στο Dead State είναι τερματική, δηλαδή δεν επιτρέπεται καμία μεταγενέστερη μετάβαση του Agent σε άλλο State. Αυτό εξασφαλίζει την ακινητοποίηση και πλήρη παύση κάθε συμπεριφοράς του Agent μετά την εξουδετέρωση.

Σε αυτή την υποενότητα, παρουσιάζεται η λειτουργική υλοποίηση της κατάστασης Dead State, με έμφαση στη διαχείριση της δέσμευσης και αποδέσμευσης του Agent από τα συστήματα ελέγχου AI,

καθώς και στη χρήση της ειδικά σχεδιασμένης συνάρτησης DeathWrapper για την ενοποιημένη αντιμετώπιση της εξουδετέρωσης.

SetStateAsDead (Function): Το function είναι υπεύθυνο για τη μεταβολή του Agent στο Dead State και την εκτέλεση των απαραίτητων ενεργειών κατά τη μετάβαση, στην περίπτωση που ο Agent εξουδετερώθηκε μέσω οπλισμού και όχι από την ενέργεια Takedown. Συγκεκριμένα, το περιεχόμενο της State (Enum) μεταβάλλεται σε Dead και κάθε εικονίδιο όπως αυτά του Detection και του Behavior State αποκρύπτονται. Στη συνέχεια, αν ο Agent εξουδετερώθηκε καθώς βρισκόταν στο Attacking State ο συνολικός αριθμός των Agents που έχουν αντιληφθεί τον παίκτη μειώνεται. Επιπλέον, τα nodes SetSenseEnabled με τιμή False και Deactivate απενεργοποιούν το οπτικό Sense του Agent και το AI Controller αντίστοιχα. Έπειτα, το RunBehaviorTree με τιμή null αποδεσμεύει τον Agent από το Behavior Tree. Τέλος, το Custom Event Stop Brain & Unpossessed Delayed αποδεσμεύει τον Agent από το AI Controller μέσω των nodes Stop Logic και Un Possess, μετά από μια μικρή χρονική καθυστέρηση. Το Event επιτρέπει την καθυστερημένη αποδέσμευση του Agent από το AI Controller, προκειμένου να υπάρχει επαρκής χρόνος για την εκτέλεση όλων των απαραίτητων ενεργειών του Custom Event DeathWrapper, πριν την αποδέσμευση του Agent από το AI Controller. Αποτελεί αναπόσπαστο κομμάτι της εξουδετέρωσης του Agent μέσω οπλισμού, καθώς η αποδέσμευση του AI Controller διακόπτει την επικοινωνία με το Parent Blueprint. Το σύνολο των ενεργειών του SetStateAsDead στην περίπτωση της εξουδετέρωσης μέσω Takedown, ενσωματώνεται στην εκτέλεση της ενέργειας, διασφαλίζοντας την τροποποίηση των μεταβλητών και την πυροδότηση ενεργειών την κατάλληλη στιγμή.

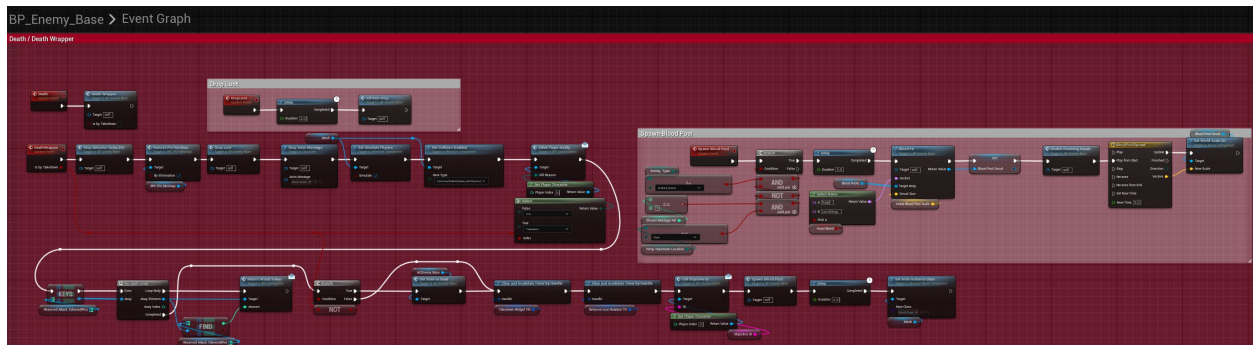


Εικόνα 154. SetStateAsDead

DeathWrapper (Custom Event): Το DeathWrapper αποτελεί το βοηθητικό function που ενσωματώνει το σύνολο των ενεργειών εξουδετέρωσης. Μέσω της μεταβλητής IsByTakedown εκτελεί τις απαραίτητες ενέργειες σύμφωνα με τον τρόπο εξουδετέρωσης του Agent.

Συγκεκριμένα, καλούνται τα:

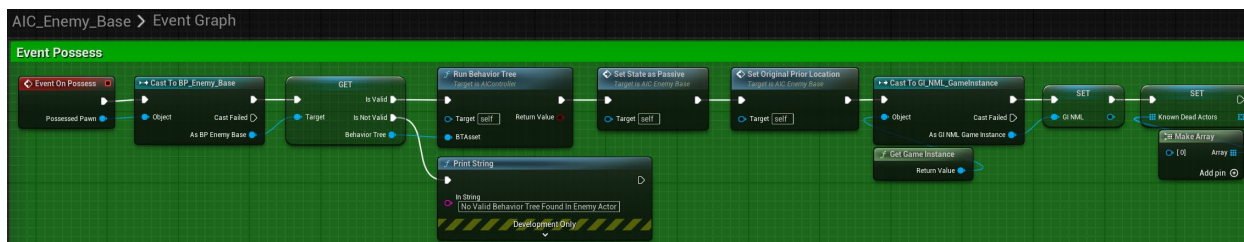
- StopBehaviorVoiceSFX το οποίο σταματά τον τρέχοντα ήχο διαλόγου του Agent
- RemovePoiRuntime το οποίο αλλάζει το εικονίδιο του Agent στο minimap του παίκτη
- DropLoot το οποίο εμφανίζει αντικείμενα όπως πυρομαχικά, Bottles, DataWorms, Medkits κ.α., κάνοντας spawn ένα από τα child Blueprints που δημιουργήθηκαν
- KilledPlayerNotify που εμφανίζει τη σχετική ειδοποίηση εξουδετέρωσης και κάνει Update στους πόντους XP και στο Level του παίκτη
- StopAnimMontage (Null), SetSimulatePhysics (True) και SetCollisionEnabled κατά τα οποία σταματά οποιοδήποτε ενεργό animation, ενεργοποιούνται τα simulation του physics asset του Actor και θέτει το collision ως CollisionEnabled (Query and Physics), ώστε να πέσει στο έδαφος με φυσικό τρόπο (Ragdoll)
- SetStateAsDead του AI Controller το οποίο αλλάζει το state του Behavior Tree σε Dead. Το function παραλείπεται στην περίπτωση που η εξουδετέρωση έγινε μέσω Takedown, σύμφωνα με τη μεταβλητή IsByTakedown
- CallObjectiveID του quest system, ώστε να ενημερωθούν οποιεσδήποτε αποστολές αφορούν στα kill των εχθρών
- SpawnBloodPool κατά την οποία εμφανίζονται εφέ με αίματα, βάσει συνθηκών, όπως το είδος εχθρού και ο τρόπος εκτέλεσης της εξουδετέρωσης
- SetAnimInstanceClass με τιμή Null, ώστε να αποδεσμευτεί το Animation Blueprint του Actor για λόγους υπολογιστικού κόστους
- ReturnAttackToken στην περίπτωση που ο actor έχει δεσμεύσει ένα token λόγω επίθεσης. Αυτό επιτυγχάνεται με τη χρήση του ForEachLoop το οποίο τρέχει τον πίνακα ReservedAttackTokens και αν βρει δεσμευμένο token από τον συγκεκριμένο actor, το επιστρέφει



Εικόνα 155. DeathWrapper

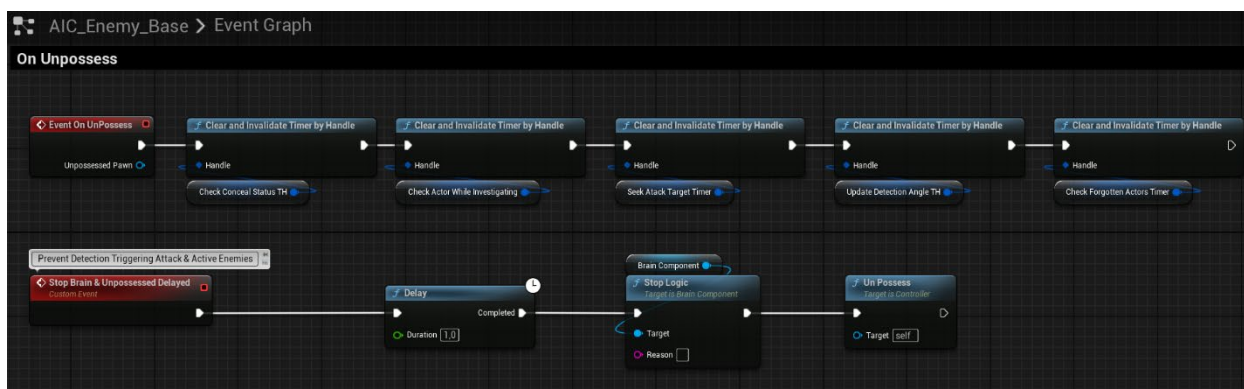
OnPossessed (Event): Το event παρέχεται εγγενώς από το Unreal Engine και ενεργοποιείται όταν το AI Controller έχει ανατεθεί σε έναν Agent. Σε αυτό το Event εκτελούνται οι ενέργειες αρχικοποίησης του Agent, όπως η ενεργοποίηση του κατάλληλου Behavior Tree και των σχετικών μεταβλητών. Συγκεκριμένα, το Event χρησιμοποιεί το node CastTo, προκειμένου να αποκτήσει πρόσβαση στο

Behavior Tree Asset που έχει ανατεθεί σε κάθε Agent. Το Asset αυτό εκκινείται μέσω του node RunBehaviorTree και ο Agent αρχικοποιείται στο Passive State. Ακόμη, το function SetOriginPriorLocation αποθηκεύει την τοποθεσία του Agent στον χώρο, λειτουργώντας ως εναλλακτική ενέργεια αρχικοποίησης του Custom Task BTT_StoreLocationPriorInvestigation.



Εικόνα 156. OnPossessed

OnUnpossessed (Event): Το Event ενεργοποιείται κατά την αποδέσμευση ενός Agent από το AI Controller, διασφαλίζοντας τη διακοπή κάθε function που εκτελείται επαναλαμβανόμενα, με τη χρήση του node SetTimerbyEvent. Αυτό επιτυγχάνεται με την εκκαθάριση των Timer Handlers μέσω του node ClearAndInvalidateTimerByHandle, εξασφαλίζοντας αποδοτική διαχείριση υπολογιστικών πόρων.



Εικόνα 157. OnUnpossessed

8.8. Frozen State

Το Frozen State αντιπροσωπεύει μια κατάσταση πλήρους αδρανοποίησης του Agent, κατά την οποία διακόπτεται οποιαδήποτε ενέργεια εκτελούσε. Το συγκεκριμένο State ενεργοποιείται αποκλειστικά όταν ο παίκτης χρησιμοποιήσει επιτυχώς το αντικείμενο DataWorm, το οποίο έχει σχεδιαστεί να επηρεάζει μόνο Agents τύπου Android. Εχθροί τύπου Ranged δεν επηρεάζονται και συνεπώς δεν μπορούν να εισέλθουν στο Frozen State.

Η ενεργοποίηση του DataWorm γίνεται μέσω ρίψης του αντικειμένου στο χώρο, δημιουργώντας για σύντομο χρονικό διάστημα μία περιοχή επίδρασης. Όλοι οι Android Agents που βρίσκονται εντός της περιοχής αυτής, επηρεάζονται άμεσα. Η επίδραση περιλαμβάνει:

- Τη μετάβαση του Agent σε κατάσταση κατά την οποία δεν είναι σε θέση να εντοπίσει τον παίκτη
- Την πλήρη ευαλωτότητά του σε Takedown επιθέσεις

Κατ' επέκταση, το DataWorm μπορεί να χρησιμοποιηθεί ως βασικό στρατηγικό εργαλείο του παίκτη για την αντιμετώπιση των Androids, επιτρέποντας είτε την ακινητοποίησή τους ώστε να περάσει απαρατήρητος, είτε την ευκολότερη εξουδετέρωσή τους μέσω stealth επιθέσεων.

Υλοποίηση

Η υλοποίηση του Frozen State περιλαμβάνει την ανάπτυξη του αντικειμένου DataWorm, το οποίο συνοδεύεται από ειδικά οπτικοακουστικά εφέ που απεικονίζουν την περιοχή επίδρασης. Κατά την ενεργοποίησή του, το DataWorm δημιουργεί έναν ημιδιάφανο θόλο που απλώνεται σταδιακά και ορίζει τη ζώνη επιρροής. Κάθε Android Agent εντός της περιοχής αυτής μεταβαίνει στο Frozen State, συνοδευόμενο από αντίστοιχα εφέ. Το αποτέλεσμα είναι η προσωρινή αδρανοποίηση του Agent, δίνοντας στον παίκτη την ευκαιρία να εκτελέσει ένα Takedown εντός του χρονικού περιθωρίου της επίδρασης. Με τη λήξη της, ο Agent επιστρέφει στο προηγούμενο State και δεν είναι πλέον ευάλωτος σε stealth επιθέσεις.

Το DataWorm μπορεί να εφαρμοστεί ανεξαρτήτως του State του Agent, διακόπτοντας οποιαδήποτε ενέργεια εκτελείται εκείνη τη στιγμή. Η επαναφορά στο προηγούμενο State είναι αυτόματη μετά την πάροδο της διάρκειας επίδρασης.

Η συμπεριφορά αυτή υλοποιείται και μέσω ενός Subtree στο Behavior Tree του Agent, το οποίο περιλαμβάνει δύο βασικά Sequences:

- Sequence 1 (Attacking State): Ο Agent συνεχίζει να κινείται γύρω από τον παίκτη μέσω ενός BlendSpace που αντιστοιχεί στο επιθετικό state, αλλά κατά διαστήματα εκτελεί συγκεκριμένα animation αντίδρασης που παρεμβάλλονται στην επίθεση.
- Sequence 2 (Άλλα States): Ο Agent παραμένει ακινητοποιημένος και εκτελεί ένα animation που απεικονίζει δυσλειτουργία.

Και στα δύο Sequences, το σύστημα ενεργοποιεί ένα BlendSpace το οποίο περιλαμβάνει animations με ασυνήθιστες, ασυντόνιστες κινήσεις, προσομοιώνοντας τεχνική δυσλειτουργία. Ανάλογα με το προηγούμενο State:

- Αν βρισκόταν σε επίθεση, προσπαθεί ανεπιτυχώς να επιτεθεί με τα χέρια του
- Σε άλλες περιπτώσεις, εκτελεί ασυντόνιστες κινήσεις σώματος

Τα animations που χρησιμοποιούνται προέρχονται από βιβλιοθήκες που συνήθως αξιοποιούνται σε horror games (π.χ. για φαντάσματα), αλλά προσαρμόστηκαν ομαλά στις ανάγκες του project. Επιπλέον, η αντίδραση του Agent συνοδεύεται από ηλεκτρικά και μηχανικά ηχητικά εφέ, όπως ήχοι βραχυκυκλώματος, ενισχύοντας τη ρεαλιστική απεικόνιση της δυσλειτουργίας.

Η υλοποίηση αξιοποιεί πολλαπλά Custom Tasks για την αναπαραγωγή animations ανά φάση επίθεσης και για την ενσωμάτωση του κατάλληλου BlendSpace. Το Frozen State υποστηρίζεται από ποικίλα assets, όπως το parent blueprint των αντικειμένων ρίψης, το child blueprint των Agents και το AI Controller.

Πιθανές Μεταβάσεις

Ο Agent μπορεί να μεταβεί στο Frozen State αποκλειστικά μέσω της επιτυχούς ενεργοποίησης του DataWorm, ανεξαρτήτως του State στο οποίο βρίσκεται τη δεδομένη στιγμή. Με τη λήξη της επίδρασης, επιστρέφει αυτόματα στο προηγούμενο State.

Σε αυτή την υποενότητα αναλύεται η λειτουργία του DataWorm, καθώς και το Behavior Tree που αφορά το Frozen State, με επισκόπηση των επιμέρους assets που υποστηρίζουν την εν λόγω συμπεριφορά.

8.8.1. Frozen Behavior Tree Overview

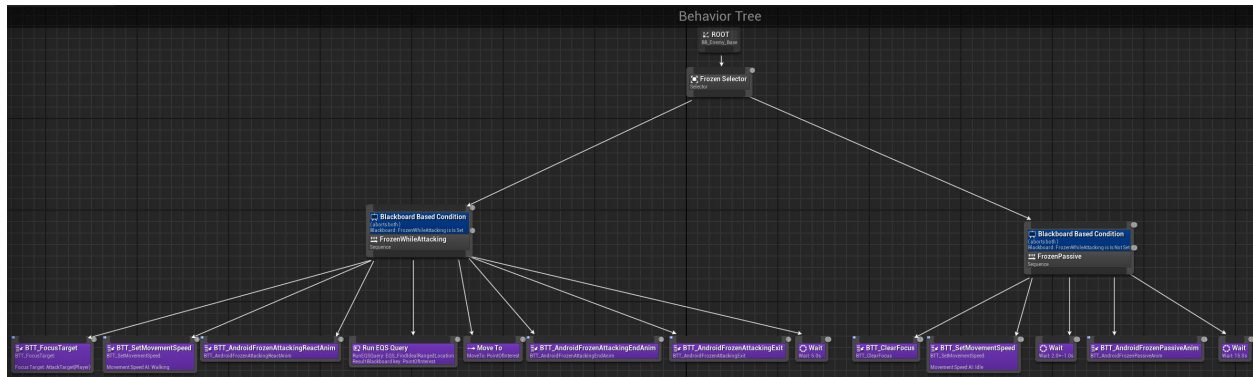
Όλες οι ενέργειες που σχετίζονται με το Frozen State οργανώνονται στο Subtree με την ονομασία BT_Subtree_Frozen_Android, το οποίο περιλαμβάνει δύο βασικά Sequences:

- FrozenWhileAttacking
- FrozenWhilePassive

Η επιλογή μεταξύ των δύο εξαρτάται από το προηγούμενο State του Agent. Συγκεκριμένα:

- Εάν ο Agent βρισκόταν στο Attacking State, εκτελείται το Sequence FrozenWhileAttacking
- Σε οποιαδήποτε άλλη περίπτωση, εκτελείται το FrozenWhilePassive

Αυτό επιτυγχάνεται μέσω ενός Blackboard Based Condition Decorator, ο οποίος ελέγχει την τιμή της μεταβλητής FrozenWhileAttacking (Boolean). Εάν η τιμή είναι true, εκτελείται το πρώτο Sequence, αλλιώς το δεύτερο.



Εικόνα 158. Frozen State Subtree

FrozenWhileAttacking Sequence

- *Ορισμός Εστίασης:* Ο Agent ορίζει ως σημείο εστίασης τον παίκτη, ώστε να τον κοιτά καθ' όλη τη διάρκεια του Frozen State
- *Μεταβολή Ταχύτητας:* Η ταχύτητα του Agent μειώνεται σε Walking, προκειμένου να κινηθεί αργά, σύμφωνα με τα αντίστοιχα Blendspace animations
- *Animation Αντίδρασης:* Εκτελείται το BTT_AndroidFrozenAttackingReactAnim, στο οποίο ο Agent προσπαθεί αποτυχημένα να επιτεθεί με τα χέρια του
- *EQS Query:* Πραγματοποιείται αναζήτηση μέσω EQS (όπως και στο Attacking State) για να επιλεγεί σημείο μετακίνησης
- *Τελικό Animation:* Με την άφιξη στο σημείο, εκτελείται το BTT_AndroidFrozenAttackingEndAnim, το οποίο περιλαμβάνει μια χαρακτηριστική κίνηση του κεφαλιού, υποδηλώνοντας το τέλος της δυσλειτουργίας
- *Εξόδος από το State:* Το task BTT_AndroidFrozenAttackingExit επαναφέρει τον Agent στο Attacking State

Αυτό το Sequence παρέχει στον παίκτη οπτικά σημάδια για το πότε ο Agent πρόκειται να επανέλθει στην κανονική του συμπεριφορά.

FrozenWhilePassive Sequence

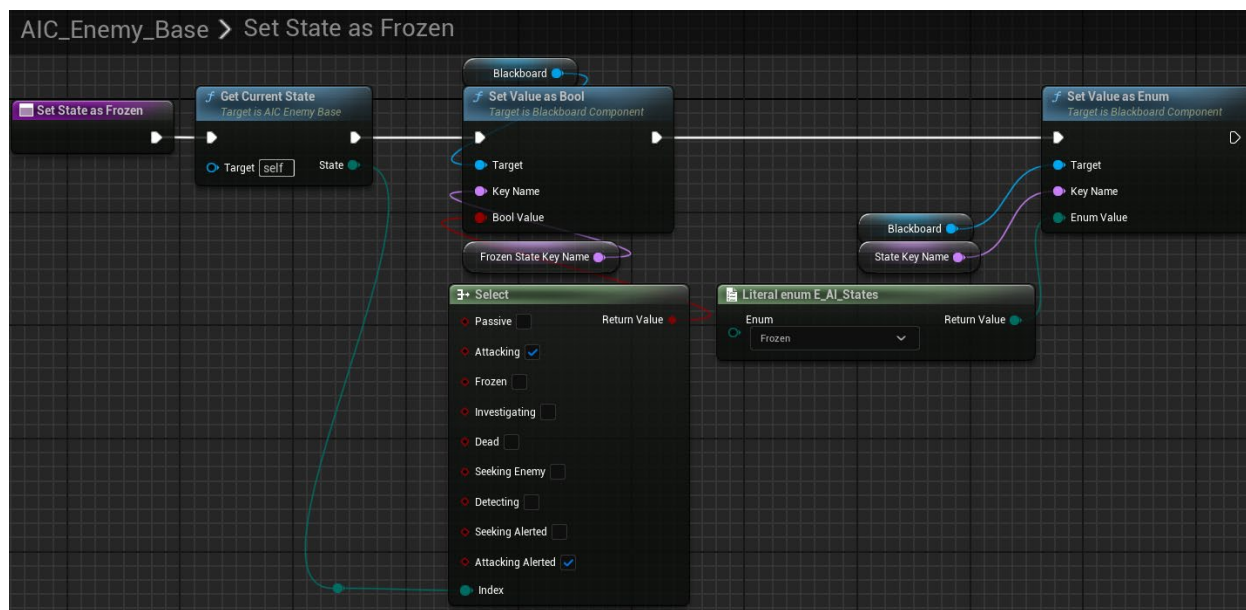
- *Καθαρισμός Εστίασης:* Ο Agent σταματά να εστιάζει στον παίκτη, επιτρέποντας ασφαλή διέλευση μπροστά του
- *Ακινητοποίηση:* Η ταχύτητα του Agent μηδενίζεται (Idle)
- *Καθυστέρηση & Animation:* Μετά από μικρή χρονική καθυστέρηση, εκτελείται το BTT_AndroidFrozenPassiveAnim, το οποίο περιλαμβάνει ακανόνιστες κινήσεις
- *Επιστροφή στο Προηγούμενο State:* Ο Agent επανέρχεται αυτόματα στο State στο οποίο βρισκόταν πριν εισέλθει στο Frozen State

Αυτός ο μηχανισμός είναι ιδιαίτερα χρήσιμος για σενάρια stealth, όπου ο παίκτης επιθυμεί να ακινητοποιήσει τον εχθρό για να περάσει απαρατήρητος.

Το BT_Subtree_Frozen_Android ενεργοποιείται αποκλειστικά από Agents τύπου Android και εκτελείται μέσω του task Run Behavior, όταν η μεταβλητή State (Enum) έχει την τιμή Frozen. Το συγκεκριμένο State έχει προτεραιότητα μεταξύ των Dead και Attacking, ώστε να διακόπτει States χαμηλότερης σημασίας κατά την ενεργοποίησή του.

8.8.2. Frozen State Assets

SetStateAsFrozen (Function): Το function είναι υπεύθυνο για τη μεταβολή ενός Agent στο Frozen State και υλοποιείται στο blueprint του AI Controller. Περιλαμβάνει την ανάθεση της τιμής στο Blackboard Key FrozenWhileAttacking, προκειμένου το Behavior Tree του Frozen State να εκτελέσει τις κατάλληλες ενέργειες. Το key παίρνει την τιμή true αν ο παίκτης βρίσκεται στο AttackingState και false διαφορετικά. Στην πρώτη περίπτωση το key θα θεωρηθεί Set και θα εκτελεστεί το Sequence FrozenWhileAttacking, ενώ στη δεύτερη θα εκτελεστεί το Sequence FrozenPassive.

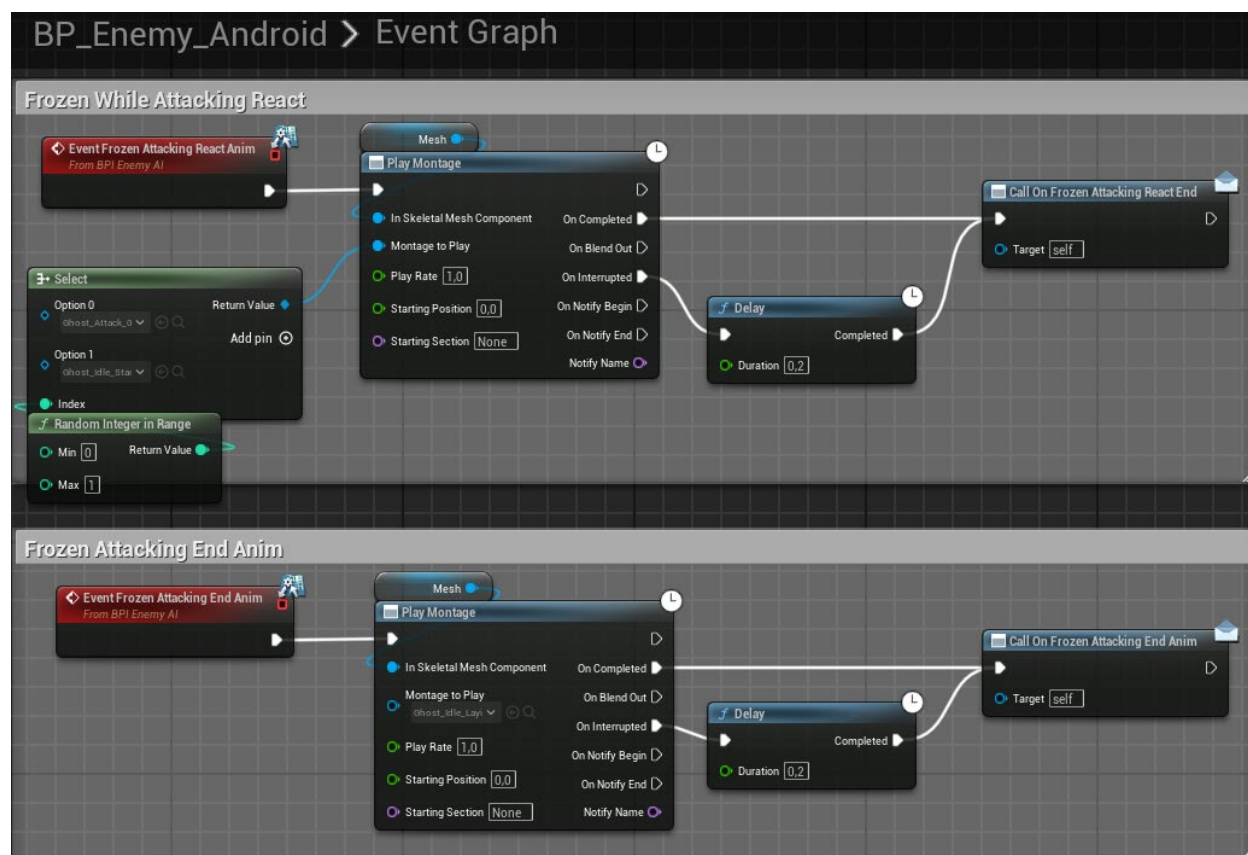


Εικόνα 159. SetStateAsFrozen

FrozenWhileAttacking και FrozenWhilePassive Animations (Functions): Η υλοποίηση της αναπαραγωγής των animations κατά τη διάρκεια του Frozen State ακολουθεί δομή ανάπτυξης παρόμοια με κάθε άλλο task αναπαραγωγής animation που χρησιμοποιείται στο συγκεκριμένο project. Κάθε animation αντιστοιχεί σε ένα ξεχωριστό function, το οποίο είναι υπεύθυνο για την αναπαραγωγή του και υλοποιείται στο child blueprint των Agents τύπου Android. Η πυροδότησή των functions πραγματοποιείται μέσω Custom Tasks που ενσωματώνονται στο Subtree του Behavior Tree. Τα

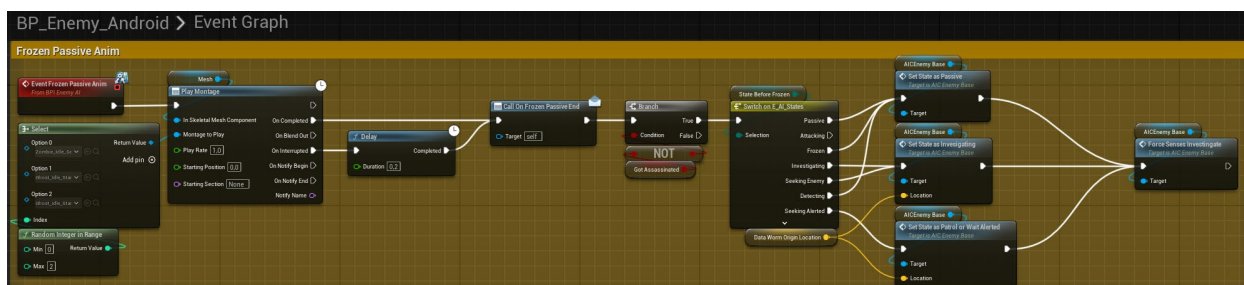
συγκεκριμένα tasks χρησιμοποιούν το node CastTo BP_Enemy_Base για να αποκτήσουν πρόσβαση στα animation functions και το node Bind Event για να εξασφαλίσουν πως το task θα ολοκληρωθεί όταν κληθεί ο αντίστοιχος Event Dispatcher, ο οποίος είναι συνδεδεμένος με το εκτελούμενο animation.

Σε ό,τι αφορά στα functions FrozenAttackingReactAnim και FrozenAttackingEndAnim, παρατηρούνται σημαντικές διαφοροποιήσεις στον τρόπο με τον οποίο εκτελούνται τα animations. Συγκεκριμένα, το FrozenAttackingReactAnim ακινητοποιεί πλήρως τον Agent, καθώς έχει ενεργοποιημένη την επιλογή RootMotion. Αυτό του επιτρέπει να μένει στάσιμος για όλη τη διάρκεια του animation, προσφέροντας στον παίκτη την ευκαιρία να πλησιάσει και να εκτελέσει μια ενέργεια Takedown χωρίς τον κίνδυνο αντίδρασης από τον Agent. Αντίθετα, το FrozenAttackingEndAnim έχει ρυθμιστεί ώστε να αναπαράγεται μόνο στο άνω μέρος του σώματος του Agent, με χρήση Layered Animation. Αυτό σημαίνει ότι ο Agent μπορεί να συνεχίσει να κινείται κατά τη διάρκεια του animation, κάνοντας χρήση των κινήσεων που ορίζονται μέσω του Blendspace στα υπόλοιπα σημεία του σώματός του. Η δυνατότητα αυτή επιτρέπει στον Agent να κινείται ελαφρώς στον χώρο, προσφέροντας στον παίκτη μια οπτική ένδειξη ότι η περίοδος αδρανοποίησης του Agent πλησιάζει στο τέλος της.



Εικόνα 160. FrozenAttackingReactAnim & FrozenAttackingEndAnim

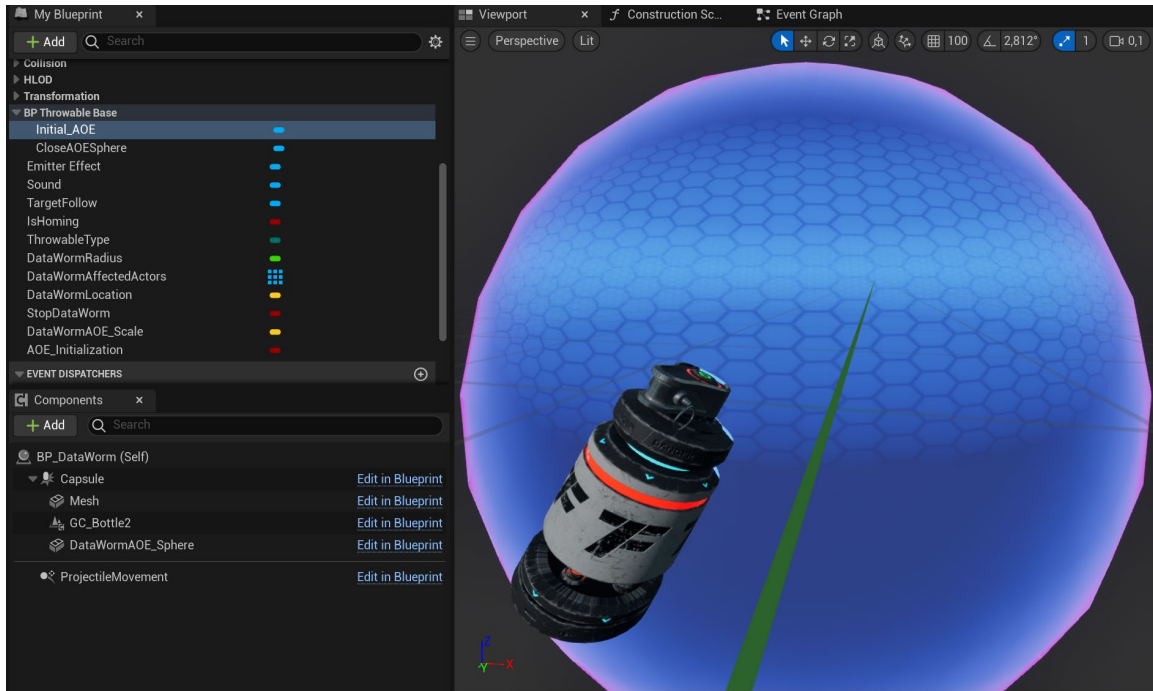
Αντίστοιχα, το function FrozenPassiveAnim αναπαράγει ένα από τα διαθέσιμα animations, επιλεγμένο τυχαία ανάμεσα σε τρεις πιθανές εκδοχές. Η χρήση τυχαίας επιλογής επιτρέπει την οπτική διαφοροποίηση ανάμεσα σε επαναλαμβανόμενες περιπτώσεις χρήσης του Frozen State, διατηρώντας την ποικιλία και τον ρεαλισμό στην παρουσίαση. Παράλληλα με την αναπαραγωγή του animation, το function έχει την ευθύνη για την επαναφορά του Agent στην κατάσταση στην οποία βρισκόταν πριν εισέλθει στο Frozen State. Μόλις ολοκληρωθεί η αναπαραγωγή του animation, ακολουθεί έλεγχος της προηγούμενης κατάστασης του Agent και πραγματοποιείται η κατάλληλη μετάβαση βάσει των μεταβλητών StateBeforeFrozen και DataWormOriginLocation, όπως παρουσιάζεται στην εικόνα 161. Η λειτουργία του function ολοκληρώνεται με την κλήση του Custom Event ForceSensesInvestigate, που αποτελεί παραλλαγή του βασικού function ForceSenses. Η συγκεκριμένη παραλλαγή εξασφαλίζει ότι ο Agent θα εκτελέσει επανέλεγχο της κατάστασης απόκρυψης του παίκτη, ενισχύοντας την αξιοπιστία και την ακρίβεια της τεχνητής νοημοσύνης όσον αφορά την αλληλεπίδραση με τον παίκτη μετά την έξοδο του από το Frozen State.



Εικόνα 161. FrozenPassiveAnim

8.8.3. DataWorm

BP_DataWorm (Blueprint Actor): Ο Actor BP_DataWorm αποτελεί το asset στο οποίο ορίζονται τα χαρακτηριστικά του δεύτερου αντικειμένου ρίψης που αναπτύχθηκε στο πλαίσιο του project, γνωστού ως DataWorm. Ο συγκεκριμένος Actor αποτελεί child blueprint του BP_Throwable_Base, το οποίο ενσωματώνει το πλήρες σύνολο των ενεργειών που αφορούν στην εκτόξευση και λειτουργία αντικειμένων αυτού του τύπου. Στο blueprint του BP_DataWorm, έχει ανατεθεί το mesh που αναπαριστά οπτικά το αντικείμενο, ενώ παράλληλα ορίζεται και το περιεχόμενο μιας σειράς μεταβλητών που σχετίζονται τόσο με τα οπτικοακουστικά εφέ όσο και με τη λειτουργικότητα του αντικειμένου.



Εικόνα 162. BP_DataWorm

Μία από τις βασικές μεταβλητές είναι το `DataWormAOE_Sphere`, το οποίο αποτελεί ένα `StaticMesh` Component και χρησιμοποιείται για την αναπαράσταση του ειδικού οπτικού εφέ του θόλου επίδρασης. Σε αυτό το mesh εκχωρείται ένα υλικό (Material) με στόχο την αναπαράσταση του οπτικού εφέ με διακριτό και εντυπωσιακό τρόπο. Πέρα από το οπτικό εφέ, ορίζονται και οι μεταβλητές `Emmitter Effect` (Niagara Particle System) και `Sound` (SoundBase), οι οποίες χρησιμοποιούνται για την αναπαραγωγή ηχητικών και σωματιδιακών εφέ κατά τη στιγμή της σύγκρουσης του αντικειμένου με επιφάνειες ή στόχους. Η μεταβλητή `ThrowableType` (Enum) καθορίζει τον τύπο του αντικειμένου και λαμβάνει την αντίστοιχη τιμή που υποδεικνύει ότι πρόκειται για `DataWorm`.

Σχετικά με τη λειτουργικότητα του αντικειμένου, καθοριστικό ρόλο παίζουν οι μεταβλητές `DataWormRadius` (Float) και `DataWorm A.O.E. Scale` (Vector). Η πρώτη καθορίζει την ακτίνα επίδρασης του θόλου, δηλαδή την απόσταση από το σημείο σύγκρουσης στην οποία μπορούν να επηρεαστούν οι Agents. Η δεύτερη αφορά τη γεωμετρική κλίμακα του θόλου στο χώρο και μεταβάλλεται κατά τη διάρκεια της χρήσης του αντικειμένου, ώστε να παρέχεται στον παίκτη μια ορατή και σαφής ένδειξη του πεδίου επιρροής.

Η `DataWormRadius` χρησιμοποιείται ειδικά κατά τη διαδικασία αναζήτησης των Agents που βρίσκονται εντός του πεδίου επιρροής, ενώ ταυτόχρονα πυροδοτεί τις ενέργειες επίδρασης, όπως η αλλαγή του State ή η πρόκληση κάποιας αντίδρασης στους Agents. Η `DataWorm A.O.E. Scale` ρυθμίζει το μέγεθος του

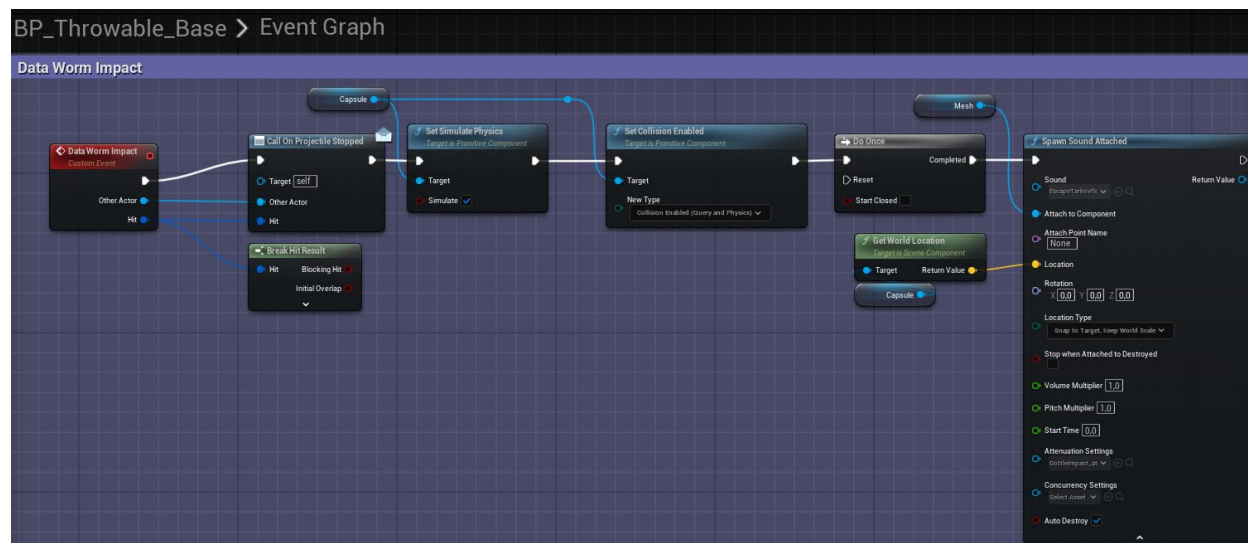
εφέ, έτσι ώστε το πεδίο δράσης του αντικειμένου να είναι οπτικά αναγνωρίσιμο από τον παίκτη και να διευκολύνει την τακτική του χρήστη στο gameplay.

Τέλος, η μεταβλητή `DataWormAffectedActors` (τύπου `Array`) περιλαμβάνει όλους τους `Agents` που έχουν ήδη επηρεαστεί από το αντικείμενο `DataWorm`. Η χρήση της συγκεκριμένης λίστας εξασφαλίζει ότι δεν θα υπάρξει απρόβλεπτη επανεκκίνηση των ενεργειών επίδρασης σε ήδη επηρεασμένους `Agents`, διατηρώντας έτσι τη συνοχή και τον έλεγχο της συμπεριφοράς στο σύστημα τεχνητής νοημοσύνης.

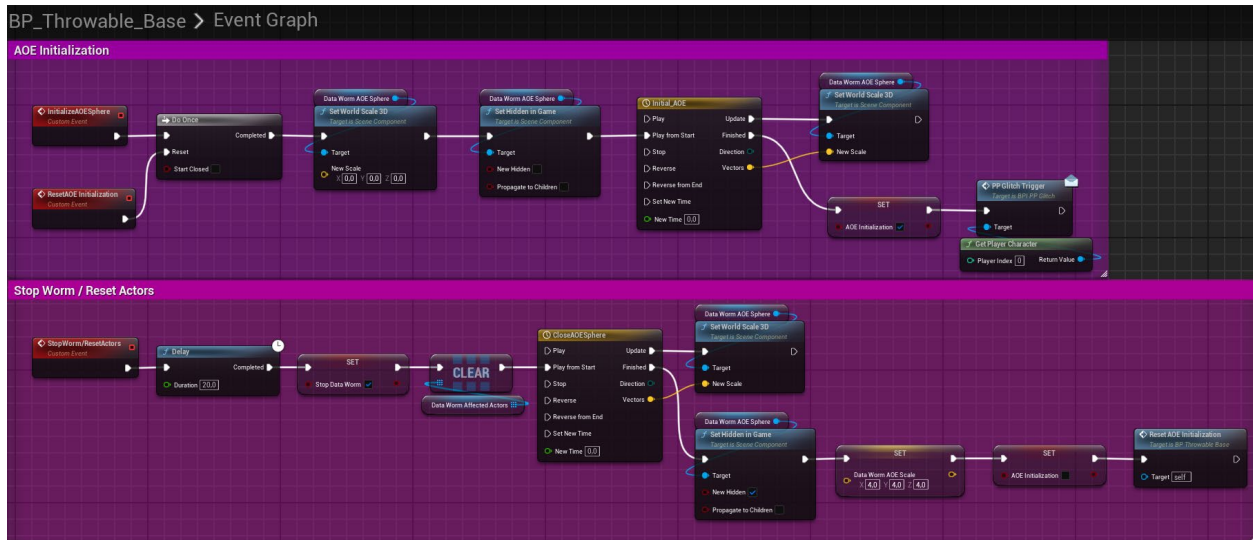
DataWormImpact και DataWormStopped (Custom Event): Τα Custom Events `DataWormImpact` και `DataWormStopped` είναι υπεύθυνα για το σύνολο των ενεργειών που εκτελούνται από τη στιγμή της ρίξης του αντικειμένου `DataWorm`, καλύπτοντας τόσο τα οπτικοακουστικά εφέ όσο και τη λογική ενεργοποίησης και επιρροής του αντικειμένου στους `Agents`. Τα events αυτά υλοποιούνται στο parent blueprint των αντικειμένων ρίξης, στο οποίο βασίζονται όλα τα επιμέρους instances όπως το `DataWorm`. Όπως περιγράφεται και στο σχετικό κεφάλαιο, κάθε `Actor` τύπου `Throwable` πυροδοτεί ένα Custom Event κατά τη σύγκρουση με μια επιφάνεια και ένα δεύτερο όταν η ταχύτητά του μηδενιστεί.

Σε αντίθεση με το αντικείμενο `Bottle`, το οποίο σταματά και καταστρέφεται αμέσως μόλις συγκρουστεί, το `DataWorm` συνεχίζει να κινείται μετά τη σύγκρουση έως ότου να σταματήσει τελείως, οπότε και ενεργοποιείται.

Το πρώτο Event, `DataWormImpact`, καλεί τον Event Dispatcher `OnProjectileStopped` για να μειωθεί ο αριθμός των διαθέσιμων `DataWorms` και να πυροδοτηθεί το Custom Event `DataWormStopped`. Αμέσως μετά, τα nodes `SetSimulatePhysics` και `SetCollisionEnabled` λαμβάνουν τις απαραίτητες τιμές ώστε να εξομοιωθεί η κύλιση του αντικειμένου. Ταυτόχρονα, χρησιμοποιούνται τα nodes `DoOnce` και `SpawnSoundAttached` για να αναπαραχθεί ο ήχος της σύγκρουσης μόνο μία φορά.



Εικόνα 163. `DataWormImpact`



Εικόνα 165. Initialize AOESphere & StopWorm/Reset Actors

Μετά την ολοκλήρωση της πρώτης σάρωσης, ακολουθεί χρονική καθυστέρηση 3 δευτερολέπτων και στη συνέχεια αυξάνονται οι μεταβλητές DataWormRadius και DataWormAOEScale, ώστε ο θόλος να μεγαλώσει τόσο σε μέγεθος όσο και σε επιρροή. Το Custom Event DataWormTrace επανεκκινεί τη διαδικασία από το κατάλληλο σημείο, διατηρώντας ενεργή την επιρροή του αντικειμένου. Όταν ολοκληρωθεί ο προκαθορισμένος κύκλος λειτουργίας, το StopWorm/Reset Actors διακόπτει την αναζήτηση Agents μετά από 20 δευτερόλεπτα, καθαρίζει τη μεταβλητή DataWormAffectedActors και μειώνει σταδιακά το οπτικό εφέ του θόλου, μέχρι αυτό να εξαφανιστεί πλήρως, σηματοδοτώντας τον τερματισμό της χρήσης του αντικειμένου.

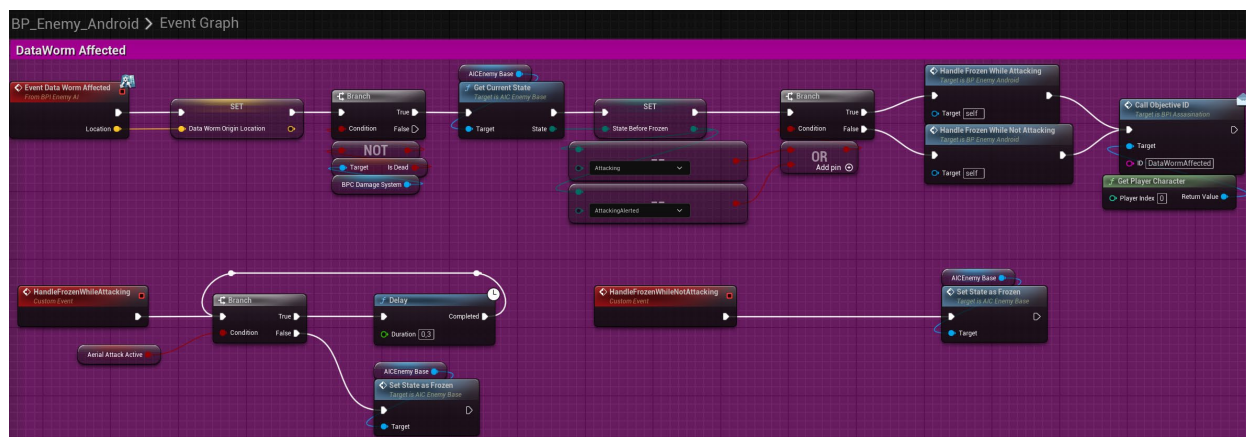
DataWormAffected (Function): Το function είναι υπεύθυνο για τη μεταβολή του Agent στο Frozen State και υλοποιείται στο child blueprint των Agents τύπου Android. Κατά την εκτέλεσή του, το function ελέγχει αρχικά εάν ο Agent δεν είναι ήδη εξουδετερωμένος, ώστε να αποτραπεί η εκτέλεση διπλών ή αντικρουόμενων ενεργειών. Αν η συνθήκη πληρείται, τότε γίνεται έλεγχος του τρέχοντος State του Agent. Εφόσον αυτός βρίσκεται σε κατάσταση Attacking, τότε πυροδοτείται το Custom Event HandleFrozenWhileAttacking, διαφορετικά καλείται το HandleFrozenWhileNotAttacking. Η διαφοροποίηση αυτή είναι αναγκαία ώστε η μετάβαση στον Frozen State να μη συμβεί απότομα κατά τη διάρκεια επιθέσεων, ιδιαίτερα όταν αυτές είναι εναέριες.

Στην περίπτωση του HandleFrozenWhileAttacking, ελέγχεται αν ο Agent εκτελεί AerialAttack. Αν η συνθήκη ισχύει, τότε το σύστημα αναβάλλει τη μετάβαση στο Frozen State μέσω επαναλαμβανόμενου ελέγχου με χρονική καθυστέρηση 0.3 δευτερολέπτων, μέχρι να ολοκληρωθεί η επίθεση. Αυτός ο μηχανισμός εξασφαλίζει πως δε θα διακοπεί βίαια μια ενεργή επίθεση. Αντίθετα, εάν ο Agent δεν βρίσκεται σε Attacking State, η μετάβαση στον FrozenState εκτελείται άμεσα χωρίς καθυστερήσεις.

Παράλληλα, το function καταγράφει το τρέχον State του Agent στη μεταβλητή StateBeforeFrozen (Enum), ώστε αυτό να μπορεί να επαναφερθεί όταν η επίδραση του DataWorm παρέλθει. Αυτό διασφαλίζει την ακριβή αποκατάσταση της συμπεριφοράς του Agent στο σημείο που διακόπηκε.

Επιπλέον, καταγράφεται και η τοποθεσία του αντικειμένου DataWorm στη μεταβλητή DataWormOriginLocation (Vector). Η τοποθεσία αυτή είναι κρίσιμη σε περιπτώσεις όπου ο Agent επιστρέφει σε States που απαιτούν συγκεκριμένο σημείο αναφοράς, όπως τα Investigating ή Seeking, επιτρέποντας τη χρήση της τοποθεσίας ως είσοδος για την επανεκκίνηση αυτών των συμπεριφορών.

Τέλος, το function ολοκληρώνεται με την κλήση του function CallObjectiveID, ενημερώνοντας το QuestSystem πως ένα DataWorm χρησιμοποιήθηκε επιτυχώς. Αυτό επιτρέπει την πρόοδο ενός ενεργού Objective που σχετίζεται με τη χρήση του αντικειμένου, ενσωματώνοντας τη ρίψη και την επίδραση του DataWorm στο ευρύτερο σύστημα αποστολών του παιχνιδιού.



Εικόνα 166. DataWormAffected



Εικόνα 167. DataWorm Demonstration

9. Takedowns

Η ενέργεια Takedown αποτελεί μια από τις βασικές μηχανικές που μπορεί να εκτελέσει ο παίκτης, επιτρέποντάς του να εξουδετερώνει άμεσα έναν Agent. Η ενεργοποίησή της πραγματοποιείται μέσω του αντίστοιχου πλήκτρου, υπό την προϋπόθεση ότι ο Agent δεν έχει αντιληφθεί την παρουσία του παίκτη και ότι ο παίκτης βρίσκεται σε κοντινή απόσταση. Η εκτέλεση της ενέργειας περιλαμβάνει την αναπαραγωγή animation τόσο στον παίκτη όσο και στον Agent: το animation του παίκτη αναπαριστά επιθετικές κινήσεις, ενώ εκείνο του Agent ακολουθεί τις κινήσεις του επιτιθέμενου, απεικονίζοντας τον ως θύμα. Παράλληλα, ενεργοποιούνται οπτικοακουστικά εφέ για τους δύο Actors. Με την ολοκλήρωση της ενέργειας, ο Agent εξουδετερώνεται ανεξαρτήτως των πόντων ζωής του.

Η υλοποίηση περιλαμβάνει επίσης ένα σύστημα επιλογής τυχαίου ζεύγους animation, το οποίο εξαρτάται από τη θέση εκκίνησης της ενέργειας (μπροστά ή πίσω από τον Agent). Δημιουργήθηκαν έξι μεταβλητές που λειτουργούν ως placeholders για τα animations: τρεις για την περίπτωση Front Takedown και τρεις για την περίπτωση Back Takedown. Οι μεταβλητές αυτές δηλώνονται στο parent blueprint των Agents και μεταβιβάζονται στα child blueprints, επιτρέποντας την προσαρμογή για κάθε τύπο Agent ή υποκλάση του και τη μελλοντική προσθήκη επιπλέον animations. Έτσι, κάθε φορά που εκτελείται μια ενέργεια Takedown, μπορεί να αναπαραχθεί ένα από τα τρία animations ανάλογα με την κατεύθυνση.

Η συνολική υλοποίηση περιλαμβάνει τα παρακάτω βασικά σημεία:

- Καθορισμός διαθεσιμότητας για την ενέργεια Takedown: Μέσω της διαδικασίας SetUp Takedown, η οποία χρησιμοποιεί Box Collision Events για να εκτελέσει ελέγχους και να τροποποιήσει τις απαιτούμενες μεταβλητές ώστε να ενεργοποιηθεί η ενέργεια.
- Έλεγχος αντιληπτικότητας του Agent: Πραγματοποιείται μέσω του function IsPlayerAware, το οποίο επιστρέφει True αν ο Agent έχει εντοπίσει τον παίκτη και False αν όχι.
- Δημιουργία Input Event για την ενεργοποίηση της ενέργειας Takedown από τον παίκτη, που περιλαμβάνει την αναπαραγωγή animation του παίκτη και την ενεργοποίηση του αντίστοιχου animation στον Agent. Αυτή η διαδικασία υλοποιείται μέσω του Takedown Input Event.
- Δήλωση μεταβλητών για τα ζεύγη animations: Δημιουργήθηκαν τα arrays StealthMontagesFront και StealthMontagesBack, τα οποία βρίσκονται τόσο στο blueprint του παίκτη όσο και στο parent blueprint των Agents, και περιλαμβάνουν τα animations ανά κατεύθυνση.
- Επιλογή του κατάλληλου animation ζεύγους για κάθε Takedown, σύμφωνα με τη θέση εκτέλεσης. Αυτό καθορίζεται από την ακέραια μεταβλητή StealthSelectedMontage, η οποία μπορεί να οριστεί είτε τυχαία είτε χειροκίνητα από το Viewport, μέσω του function Assassination Chooser.
- Μεταφορά απαραίτητων πληροφοριών στο blueprint του Agent για την αναπαραγωγή του κατάλληλου animation, σύμφωνα με την επιλεγμένη σκηνή. Αυτό υλοποιείται με το Blueprint Interface function AssassinationWarpLocation, το οποίο καλείται από το blueprint του παίκτη και υλοποιείται στο parent blueprint των Agents. Το function αυτό αναλαμβάνει και τη μετάβαση του Agent στην κατάσταση Dead State.

Τα functions και custom events που εκτελούνται με χρονική σειρά κατά την εκτέλεση μιας ενέργειας Takedown είναι τα εξής:

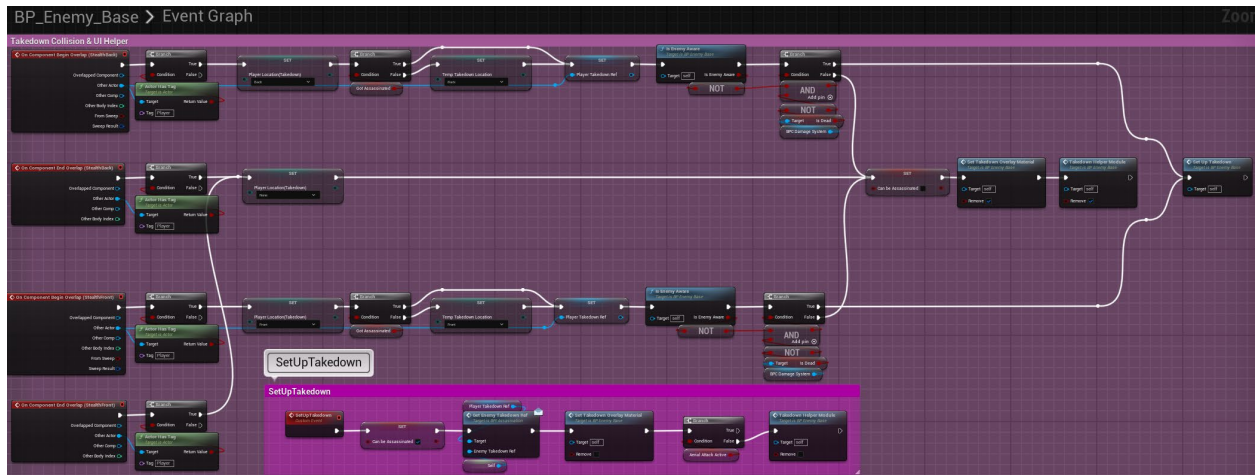
SetUp Takedown (Event): Η διαδικασία εκκινείται όταν ο παίκτης εισέλθει σε μία από τις δύο περιοχές Box Collision (Front ή Back) που έχουν τοποθετηθεί στον Agent. Αυτό έχει ως αποτέλεσμα την ενεργοποίηση των Events OnComponentBeginOverlap, τα οποία ελέγχουν αν πληρούνται οι προϋποθέσεις για την ενεργοποίηση της ενέργειας Takedown. Ο έλεγχος αυτός πραγματοποιείται μέσω του function IsPlayerAware, το οποίο επιβεβαιώνει ότι ο Agent δεν έχει αντιληφθεί τον παίκτη και δεν βρίσκεται ήδη σε εξουδετερωμένη κατάσταση.

Εφόσον οι προϋποθέσεις ικανοποιούνται, ακολουθεί τροποποίηση τριών βασικών μεταβλητών:

- Η σχετική τοποθεσία του παίκτη ως προς τον Agent καταχωρίζεται στη μεταβλητή PlayerTakedownLocation (Enum)
- Το Custom Event SetUp Takedown εκχωρεί το reference του Agent στη μεταβλητή PlayerTakedownRef (Actor), το οποίο στη συνέχεια αποστέλλεται στον παίκτη μέσω του Blueprint Interface function GetEnemyTakedownRef
- Η μεταβλητή CanBeAssassinated (Boolean) ορίζεται ως True

Παράλληλα, ενεργοποιούνται οπτικά εφέ επικάλυψης τόσο στον Agent όσο και στο εικονίδιο ενέργειας που υποδεικνύει στο χρήστη τη διαθεσιμότητα του πλήκτρου για την εκτέλεση της ενέργειας Takedown. Οι παραπάνω μεταβλητές αξιοποιούνται αργότερα από το Input Event Takedown στο blueprint του παίκτη για την εκτέλεση της ενέργειας και την παραμετροποίησή της.

Σε περίπτωση που ο παίκτης εξέλθει από τις περιοχές Box Collision, το OnComponentEndOverlap ενεργοποιείται, μηδενίζοντας τις αντίστοιχες μεταβλητές και απενεργοποιώντας τα σχετικά εφέ, καθιστώντας έτσι αδύνατη την εκτέλεση του Takedown. Το διάγραμμα της παραπάνω διαδικασίας παρουσιάζεται στην εικόνα 168.



Εικόνα 168. SetUp Takedown

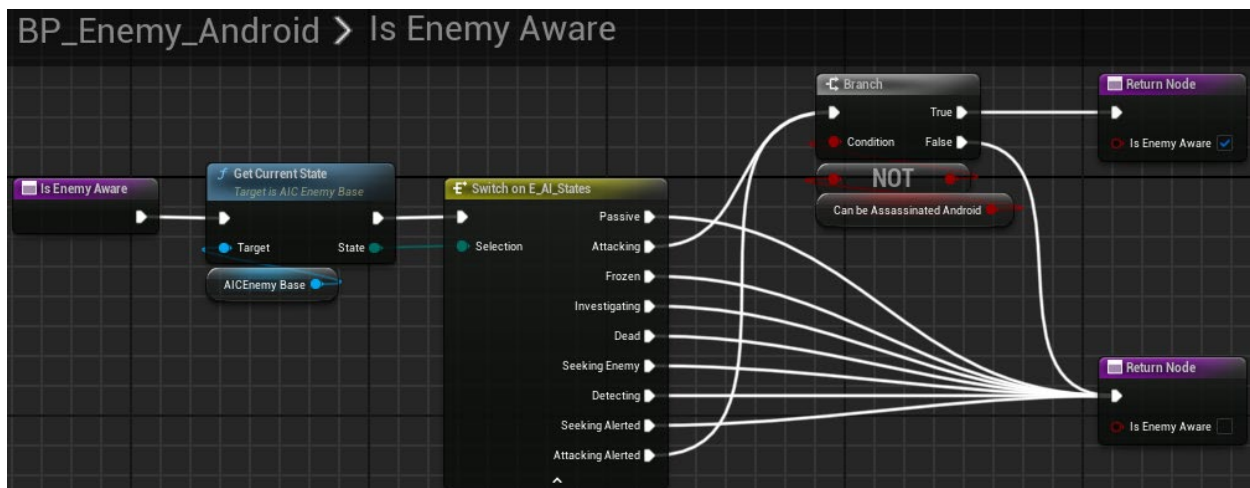


Εικόνα 169. Overlay Effect & Takedown Widget

IsPlayerAware (Function): Το function καθορίζει αν ένας Agent βρίσκεται σε State που επιτρέπει την εκτέλεση της ενέργειας Takedown, επιστρέφοντας το αποτέλεσμα ως μεταβλητή Boolean.

Η βασική προϋπόθεση για να επιτραπεί η ενέργεια είναι ο Agent να μην έχει αντιληφθεί την παρουσία του παίκτη, δηλαδή να μην βρίσκεται στο Attacking State. Συνεπώς, αν ο Agent είναι σε οποιοδήποτε άλλο State, το function επιστρέφει False, υποδεικνύοντας ότι η εκτέλεση της ενέργειας είναι δυνατή. Ωστόσο, στην περίπτωση που ο Agent ανήκει στην κατηγορία Android, εφαρμόζεται μια επιπλέον εξαίρεση κατά τον έλεγχο. Συγκεκριμένα, ακόμη και αν ο Android Agent βρίσκεται στο Attacking State, μπορεί να γίνει δεκτός ως στόχος Takedown, εφόσον η μεταβλητή CanBeAssassinated είναι True. Η μεταβλητή αυτή τίθεται σε True για σύντομο χρονικό διάστημα όταν ο παίκτης εκτελεί επιτυχημένα μια κίνηση Perfect Dodge, καθιστώντας έτσι τον Agent ευάλωτο, παρά τη φαινομενικά επιθετική του κατάσταση.

Το σχετικό διάγραμμα του function, το οποίο απεικονίζει τη λογική ελέγχου στην περίπτωση Android Agents, παρουσιάζεται στην εικόνα 170.



Εικόνα 170. IsPlayerAware

Takedown (Input Event): Η εκτέλεση της ενέργειας Takedown πραγματοποιείται μέσω του Input Event Takedown, το οποίο ενεργοποιείται όταν ο παίκτης πατήσει το αντίστοιχο πλήκτρο. Το event βρίσκεται υλοποιημένο στο Blueprint του παίκτη και αρχικά προβαίνει σε δύο κρίσιμους ελέγχους:

- Έλεγχο εγκυρότητας του TakedownRef μέσω του node Is Valid, ώστε να διαπιστωθεί αν υπάρχει ενεργός στόχος εντός της περιοχής εμβέλειας
- Ανάκτηση της τιμής της μεταβλητής CanBeAssassinated μέσω του GetCanBeAssassinated, ώστε να εξακριβωθεί αν ο Agent βρίσκεται σε κατάσταση που επιτρέπει την ενέργεια

Οι παραπάνω έλεγχοι διασφαλίζουν πως η ενέργεια θα εκτελεστεί μόνο όταν ο παίκτης βρίσκεται εντός ενός από τα Box Collision volumes που ορίζουν τη TakedownRef, και ο Agent δεν βρίσκεται είτε ήδη εξουδετερωμένος είτε σε κατάσταση που απαγορεύει την εκτέλεση.

Στη συνέχεια, το Event προβαίνει σε απόσπαση ή καταστροφή των αντικειμένων που ενδεχομένως κρατά ο παίκτης (π.χ. όπλα), μέσω του Custom Event Detach/DestroyAllHeldItems, ώστε να διασφαλιστεί η καθαρή οπτική απεικόνιση του Animation κατά την εκτέλεση της σκηνής Takedown.

Αμέσως μετά, η κατεύθυνση προσέγγισης (Front ή Back) υπολογίζεται μέσω του function PlayerTakedownLocation, το οποίο στη συνέχεια πυροδοτεί τα αντίστοιχα Custom Events: TakedownFront ή TakedownBack, οδηγώντας τη ροή του script προς την αντίστοιχη αναπαραγωγή animation, ανάλογα με την θέση του παίκτη σε σχέση με τον στόχο.

Επιπρόσθετα, επειδή το παιχνίδι υποστηρίζει και χειρισμό μέσω Controller, το Event περιλαμβάνει ελέγχους για την περίπτωση που η ενέργεια Takedown δεν μπορεί να εκτελεστεί. Σε αυτό το σενάριο, το ίδιο πλήκτρο αναλαμβάνει άλλες ενέργειες, με προκαθορισμένη σειρά προτεραιότητας: Takedown, Looting αντικειμένων και Reload του οπλισμού. Αυτός ο σχεδιασμός επιτρέπει την πολυλειτουργικότητα

Το διάγραμμα του Takedown παρουσιάζεται στην εικόνα 171.

[illegible]

Εικόνα 172. Takedown Front-Back

Ταυτόχρονα, ενεργοποιείται το function Assassination Warp Location, το οποίο εκκινεί τις απαραίτητες διαδικασίες στο Blueprint του Agent, όπως την αναπαραγωγή του αντίστοιχου animation και τις κατάλληλες μεταβολές μεταβλητών. Επιπλέον, επιστρέφονται οι μεταβλητές RefLocation (Vector) και RefRotation (Rotator), που μεταβιβάζονται ως είσοδοι στο node AddOrUpdateWarpTarget. Αυτό επιτρέπει την ευθυγράμμιση της θέσης και της κατεύθυνσης του παίκτη χρησιμοποιώντας την τεχνική

Motion Warping, ώστε το animation να ξεκινήσει από τη σωστή τοποθεσία και με την κατάλληλη κατεύθυνση. Με αυτόν τον τρόπο, εξασφαλίζεται ότι, ακόμη και αν ο παίκτης ξεκινήσει την ενέργεια Takedown με αντίθετη κατεύθυνση από τον Agent, η θέση του θα προσαρμοστεί κατάλληλα, επιτρέποντας συγχρονισμένη και ρεαλιστική αναπαραγωγή των δύο animations στον χώρο.

Παράλληλα, τα functions Spawn Front Stealth Weapon και Spawn Back Stealth Weapon καθιστούν ορατό τον οπλισμό που συνοδεύει κάθε animation, αν υπάρχει. Με την ολοκλήρωση του animation, εκχωρούνται εκ νέου οι απαραίτητες μεταβλητές ώστε να επιτραπεί μελλοντική εκτέλεση ενέργειας Takedown, να πραγματοποιηθούν οι κινήσεις όπλισης ή αφοπλισμού, να αποκρυφθεί ο οπλισμός που σχετίζεται με το animation και να γίνει εκ νέου ορατός ο αρχικός οπλισμός που κρατούσε ο παίκτης πριν την εκτέλεση.

Η διαγραμματική απεικόνιση και η οπτική αναπαράσταση της ενέργειας Takedown κατά την εκτέλεσή της παρουσιάζονται στις εικόνες 172 και 173.

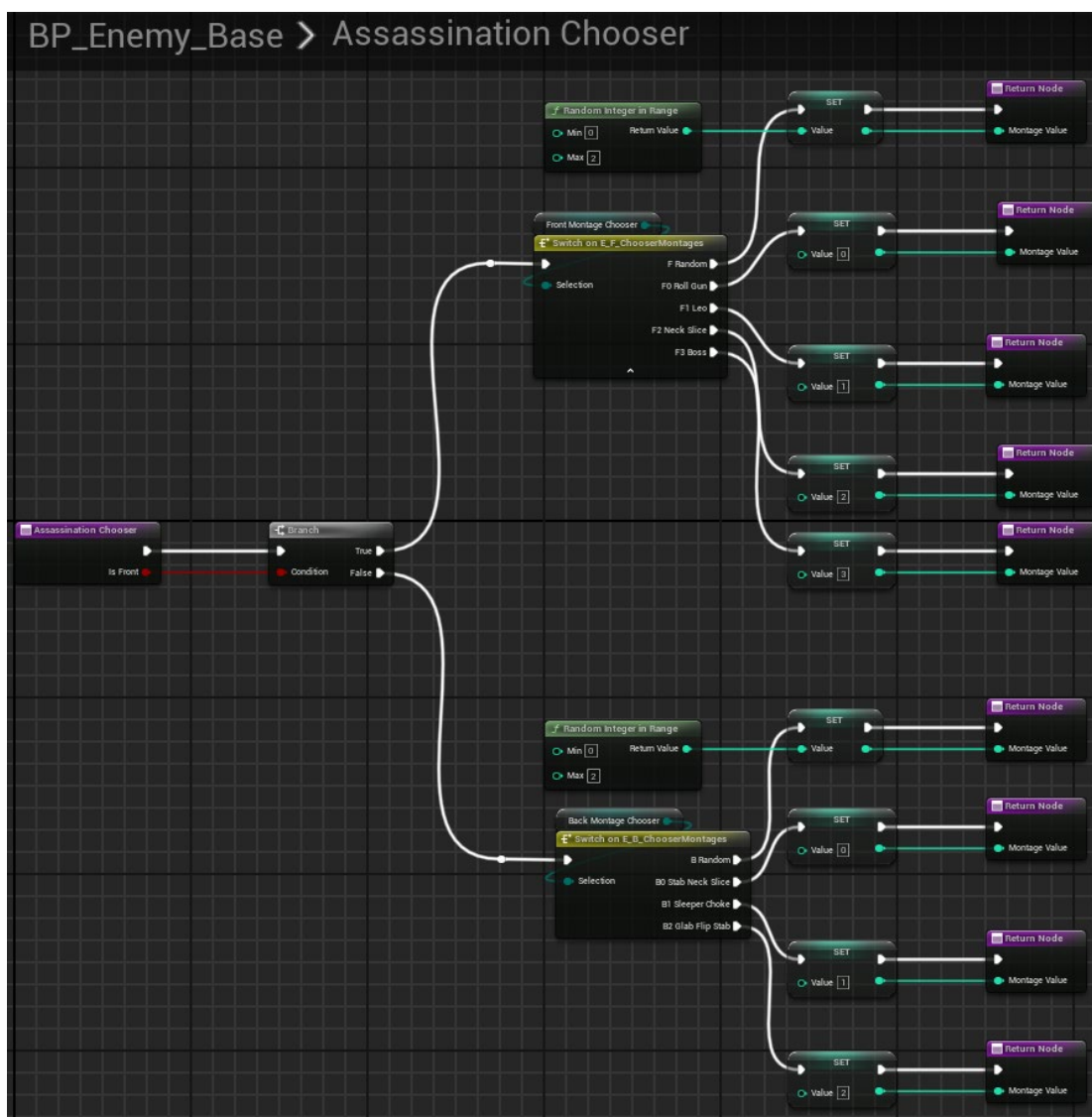


Εικόνα 173. Takedown Demonstration (1)

Assassination Chooser (Function): Το συγκεκριμένο function είναι υπεύθυνο για την επιλογή του animation που θα εκτελεστεί κατά την ενέργεια Takedown, βάσει της κατεύθυνσης από την οποία αυτή εκκινείται. Υλοποιείται στο parent blueprint των Agents και δέχεται ως είσοδο τη μεταβλητή IsFront, η

οποία προσδιορίζει αν η προσέγγιση του παίκτη γίνεται από μπροστά ή από πίσω. Η έξοδος του function είναι ένας ακέραιος αριθμός που αντιστοιχεί σε δείκτη ενός από τα arrays StealthMontagesFront ή StealthMontagesBack, και χρησιμοποιείται τόσο στο blueprint του παίκτη όσο και στο blueprint του Agent, προκειμένου να αναπαραχθεί το σωστό ζεύγος animation για κάθε περίπτωση.

Για να υποστηρίζεται τόσο η τυχαία επιλογή animation, όσο και η δυνατότητα χειροκίνητης ανάθεσης μέσω του Viewport σε κάθε Agent, έχουν υλοποιηθεί δύο enum μεταβλητές: FrontMontageChooser και BackMontageChooser. Σε αυτές μπορεί να καθοριστεί είτε συγκεκριμένο animation, είτε η επιλογή Random. Κατά συνέπεια, το function Assassination Chooser αξιοποιεί αυτές τις enum μεταβλητές για να επιστρέψει είτε έναν προκαθορισμένο δείκτη, είτε έναν τυχαίο, ανάλογα με την επιθυμητή συμπεριφορά.



Εικόνα 174. Assassination Chooser

Αρχικά, το function αφαιρεί οποιοδήποτε οπτικό εφέ επικάλυψης και UI, ακινητοποιεί τον Agent και διακόπτει οποιοδήποτε ενεργό animation, ώστε να εκτελεστεί χωρίς παρεμβολές το animation του Takedown. Έπειτα, θέτει τη μεταβλητή IsDead σε false και μεταφέρει τον Agent στο Dead State. Οι ενέργειες αυτές εξασφαλίζουν ότι η ενέργεια Takedown δεν μπορεί να εκτελεστεί εκ νέου κατά τη διάρκειά της και ότι ο Agent δεν θα μεταβεί σε διαφορετικό State μέχρι να ολοκληρωθεί η διαδικασία.

Τέλος, ανακτώνται και επιστρέφονται από το function οι μεταβλητές θέσης (RefLocation) και κατεύθυνσης (RefRotation) για την εκτέλεση του Motion Warping. Οι τοποθεσίες αυτές προέρχονται από SceneComponents, τα οποία έχουν οριστεί ξεχωριστά σε κάθε child blueprint, αντιστοιχώντας στα διαφορετικά Takedown animations. Αυτό εξασφαλίζει ότι ο παίκτης μετακινείται ακριβώς στη σωστή θέση για κάθε χαρακτήρα, λαμβάνοντας υπόψη πιθανές γεωμετρικές αποκλίσεις στη σκελετική τους δομή.

BP_Energy_Base > Assassination Warp Location

Zoom -4

Assassination Warp Location

Choose Montage

GET

Is Not Valid
Assassination Overlay

Set Y Allowed Overlay Material

Target

Remove

Step Action Montage

Target

Remove

Enable Movement

Character Movement

Target

GET

Target

New Visibility

Propagate to Children

BP Damage System

All Energy Base

Set State as Dead

Set Visibility

Propagate to Children

Select Montage Number

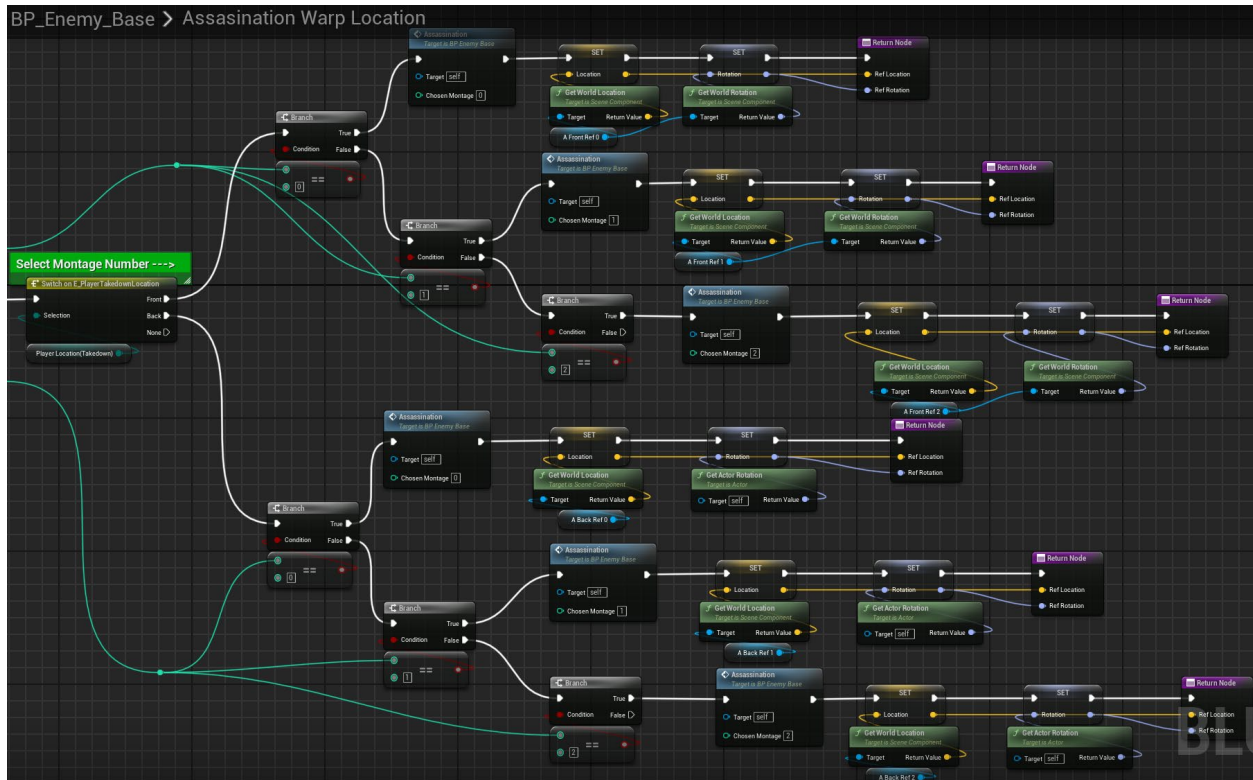
Front

Back

None

Player Location Tabboard

253



Εικόνα 176. Assassination Takedown Warp Location (2)

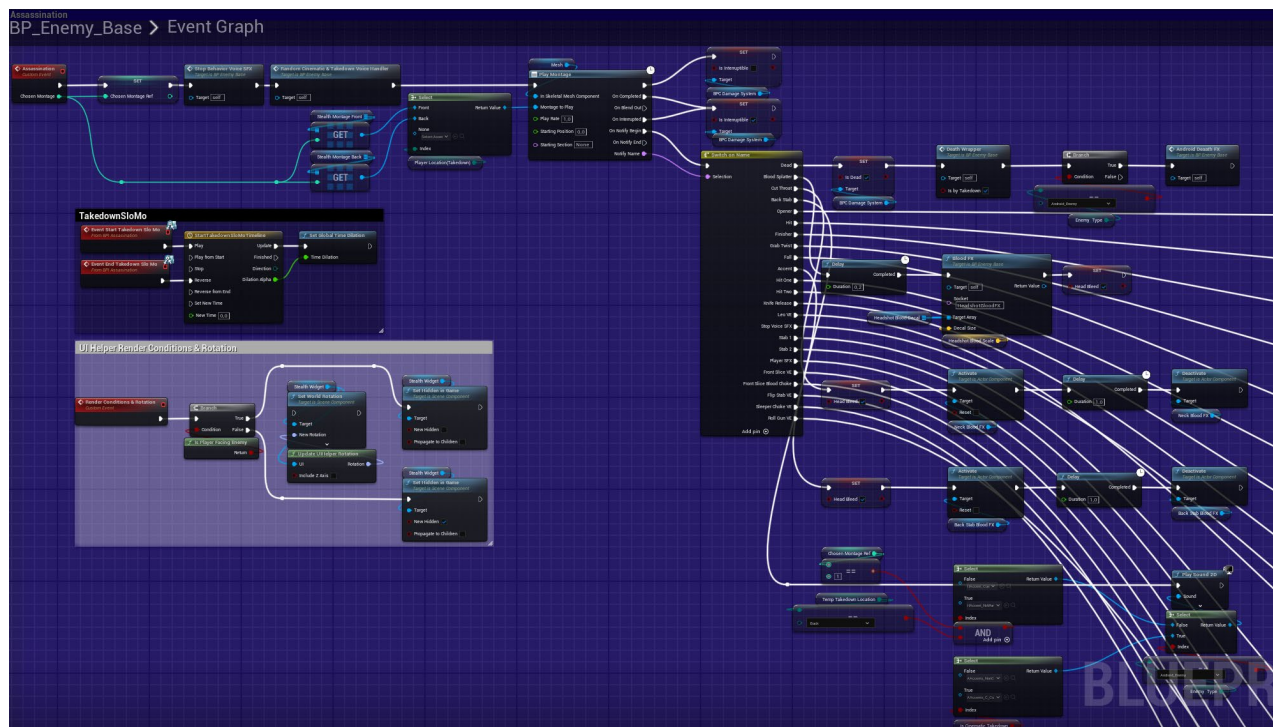
Assassination (Custom Event): Το Custom Event Assassination είναι υπεύθυνο για την αναπαραγωγή του κατάλληλου animation που έχει επιλεγεί κατά την εκτέλεση μιας ενέργειας Takedown, καθώς και για την αναπαραγωγή των αντίστοιχων οπτικοακουστικών εφέ που το συνοδεύουν. Υλοποιείται στο parent blueprint των Agents και δέχεται ως είσοδο τη μεταβλητή ChosenMontage.

Η πρώτη ενέργεια που εκτελείται είναι η διακοπή οποιουδήποτε ενεργού ηχητικού διαλόγου του Agent, μέσω του node StopBehaviorVoiceSFX. Αυτό είναι απαραίτητο, καθώς ορισμένα Takedowns ενσωματώνουν τον δικό τους διάλογο, ενώ άλλα δεν περιλαμβάνουν καθόλου. Στη συνέχεια, με βάση τη μεταβλητή ChosenMontage, επιλέγεται το κατάλληλο animation από το Array του Agent και αναπαράγεται μέσω του Play Montage node.

Κάθε animation περιλαμβάνει ένα σύνολο από Montage Notifies, τα οποία πυροδοτούν συγκεκριμένες ενέργειες, σχετικές με την εξουδετέρωση του Agent και την αναπαραγωγή των ειδικών εφέ. Ενδεικτικά, το notify Dead αλλάζει τη μεταβλητή IsDead σε true και καλεί το Custom Event DeathWrapper, το οποίο ενεργοποιεί όλες τις υπόλοιπες διαδικασίες που σχετίζονται με την εξουδετέρωση του χαρακτήρα.

Η χρονική τοποθέτηση του συγκεκριμένου notify μέσα στο animation είναι κρίσιμη για δύο βασικούς λόγους:

- Ολοκλήρωση animation: Η κλήση του DeathWrapper πρέπει να γίνεται αφού έχει ολοκληρωθεί το μεγαλύτερο μέρος του animation, καθώς περιλαμβάνει την πλήρη ακινητοποίηση του Agent
- Έγκαιρη μετάβαση κατάστασης: Πρέπει να διασφαλιστεί ότι οι μεταβλητές και οι σχετικές ενέργειες εξουδετέρωσης έχουν εκτελεστεί εγκαίρως, ώστε να αποτραπεί η αλλαγή του State του Agent σε καταστάσεις όπως το Detecting, κατά τη διάρκεια του Takedown



Εικόνα 177. Assassination (Agents)



Εικόνα 178. Takedown Demonstration (2)

10.Συμπληρωματικές Υλοποιήσεις

Στο πλαίσιο της ανάπτυξης των Behavior Trees, δημιουργήθηκαν επιπλέον λειτουργίες και εργαλεία που συμβάλλουν στην ενίσχυση της πολυπλοκότητας της συμπεριφοράς των Agents και ταυτόχρονα προάγουν την αίσθηση εμπύθισης και ρεαλισμού του παιχνιδιού. Αυτές οι υλοποιήσεις προορίζονται να επηρεάσουν τόσο την εμπειρία του χρήστη όσο και την ατμόσφαιρα του παιχνιδιού κατά τη διάρκεια του gameplay.

Scanner

Ένα από τα κύρια αντικείμενα που δημιουργήθηκε είναι το Scanner. Αυτή η λειτουργία επιτρέπει στον παίκτη να σαρώσει το περιβάλλον του για να εντοπίσει Agents και αντικείμενα που μπορούν να συλλεχθούν εντός μιας συγκεκριμένης ακτίνας. Όταν ο παίκτης ενεργοποιεί αυτή τη δυνατότητα, εμφανίζεται μια τεχνολογικά προηγμένη φορητή συσκευή στα χέρια του και ενεργοποιείται ένα animation που δείχνει τον παίκτη να αλληλεπιδρά με αυτήν (π.χ., πατώντας κουμπιά). Η διαδικασία συνοδεύεται από ένα post-processing effect που χρησιμοποιεί την τεχνική custom stencil depth, επιτρέποντας την ανάθεση ενός μοναδικού αριθμού και χρώματος στα αντικείμενα και τους Agents του

παιχνιδιού. Το περιβάλλον του παιχνιδιού αποχρωματίζεται, και κάθε αντικείμενο ή Agent που μπορεί να συλληχθεί παίρνει χρώμα σύμφωνα με τον τύπο του.

Επιπλέον, αυτή η ενέργεια επιτρέπει την οπτική αναπαράσταση των Agents και αντικειμένων μέσω ενός minimap. Το minimap ενημερώνεται σε πραγματικό χρόνο και εμφανίζει τη θέση των Agents με διαφορετικά χρώματα ανάλογα με τον τύπο τους, για παράδειγμα κόκκινο για Ranged και μοβ για Androids. Ο παίκτης μπορεί έτσι να παρακολουθεί την κίνηση των NPCs και να βλέπει αντικείμενα μέσω εμποδίων, όπως τοίχους, γεγονός που διευκολύνει τη στρατηγική του κίνηση στο Level. Επίσης, το σύστημα συνδυάζεται με το Quest System, παρέχοντας στον παίκτη τη δυνατότητα να βλέπει τη θέση της κύριας αποστολής στον χάρτη. Για να αποφευχθεί η υπερβολική χρήση, το Scanner διαθέτει ένα Cooldown ενός λεπτού.



Εικόνα 179. Scanner (active)

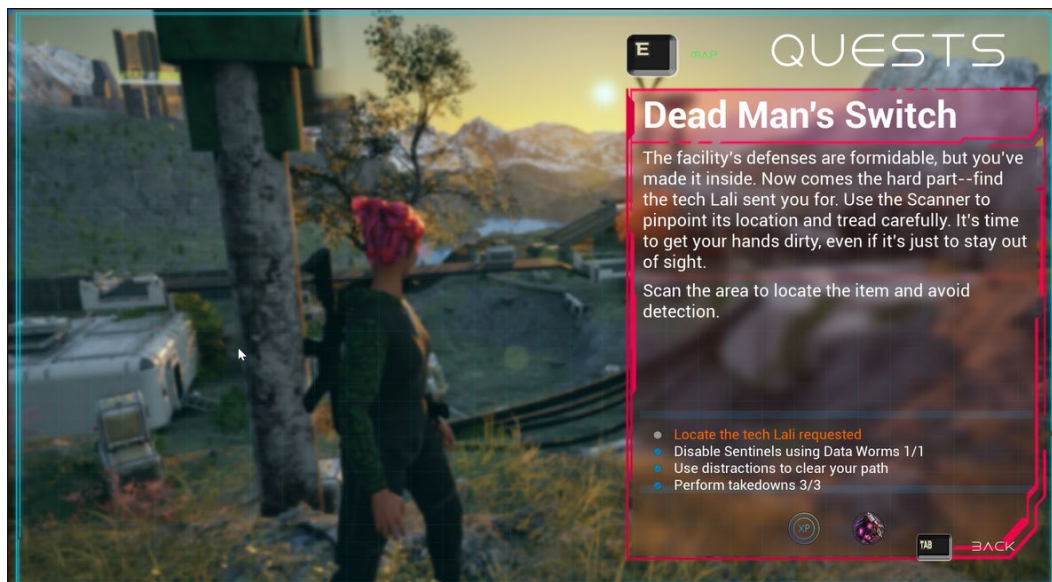
Quest System

Αναπτύχθηκε ένα πλήρως λειτουργικό και επεκτάσιμο Quest System, το οποίο χρησιμοποιεί Data Structures ως βάσεις δεδομένων για την αποθήκευση και οργάνωση των πληροφοριών σχετικά με τα Quests. Αυτές περιλαμβάνουν τίτλους αποστολών, οδηγίες, τοποθεσίες, διαχωρισμό μεταξύ κύριων και δευτερευουσών αποστολών, κριτήρια και μετρητές επίτευξης, αντικείμενα και πόντους XP που κερδίζει ο παίκτης. Το σύστημα παρέχει δυνατότητες ανάθεσης και επίτευξης Quests, ενημέρωσης σε πραγματικό χρόνο και οπτικής αναπαράστασης μέσω UI.



Εικόνα 180. Quest Trigger

Το Quest System ενσωματώνεται με τη συμπεριφορά των Agents, επιτρέποντας στον παίκτη να ολοκληρώνει συγκεκριμένα challenges, όπως "χρησιμοποίησε το αντικείμενο DataWorm" ή "εκτέλεσε τρεις κινήσεις Takedown". Ενσωματώθηκε επίσης ένα Quest Menu Widget που επιτρέπει στον παίκτη να βλέπει πλήρη περιγραφή των Quests, τα αντικείμενα που απαιτούνται, τους πόντους XP που θα λάβει, καθώς και τον χάρτη της τοποθεσίας.

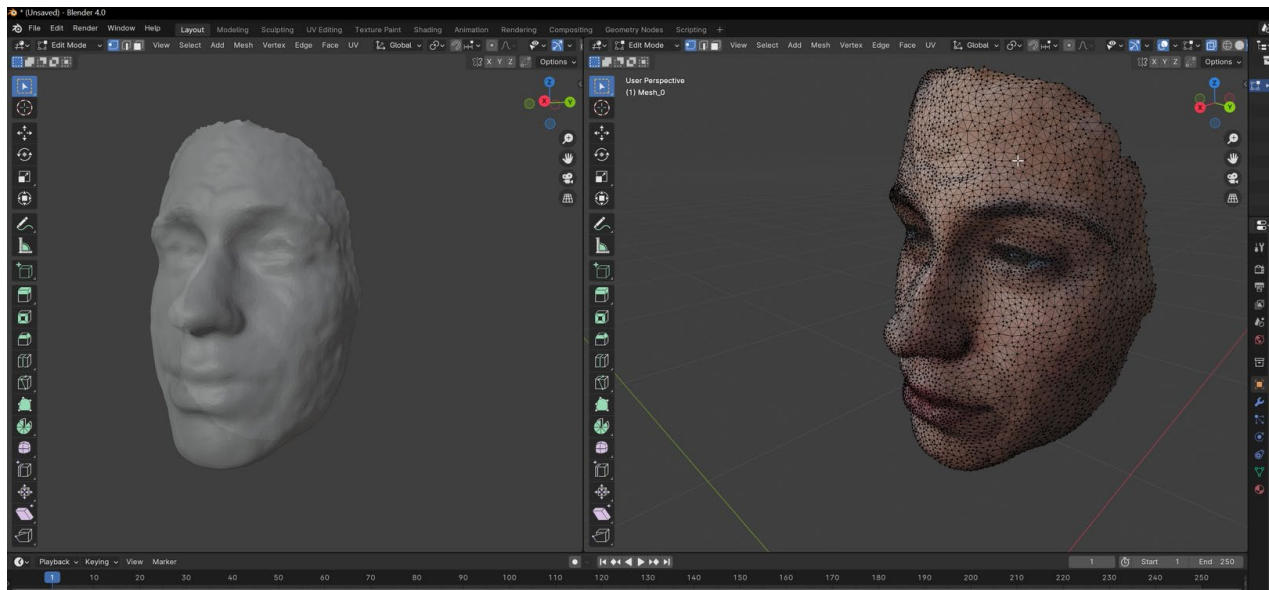


Εικόνα 181. Quest Menu Widget

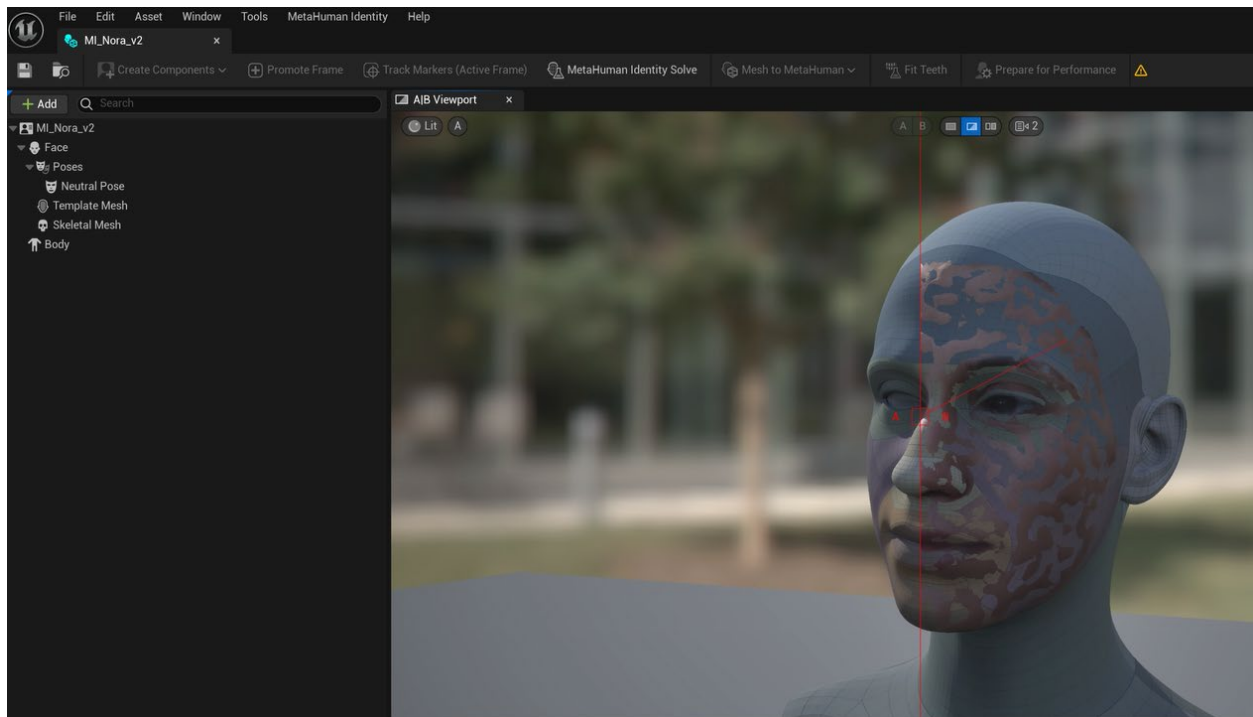
Παίκτης

Η ανάπτυξη του χαρακτήρα του παίκτη περιλαμβάνει την υλοποίηση πολλών assets και λειτουργιών.

Δημιουργία Ρεαλιστικού Χαρακτήρα: Η πλατφόρμα Polycam χρησιμοποιήθηκε για τη σάρωση ενός πραγματικού προσώπου μέσω φωτογραμμετρίας, δημιουργώντας ένα face mesh με υψηλή γεωμετρική λεπτομέρεια. Το mesh μεταφέρθηκε στην πλατφόρμα Blender για επεξεργασία της τοπολογίας του, και χρησιμοποιήθηκε στο Unreal Engine για τη δημιουργία ενός πλήρους λειτουργικού Metahuman Actor μέσω της τεχνικής mesh to metahuman. Ο χαρακτήρας του παίκτη διαθέτει πλήρη σκελετό, ο οποίος εισήχθη ως child mesh και χρησιμοποιεί τεχνικές real-time IK retargeting, επιτρέποντας τη σωστή προσαρμογή της κίνησης και του σκελετού με βάση το σώμα και τη δυναμική του χαρακτήρα. Η κινησιολογία του χαρακτήρα εκτελείται μέσω της τεχνικής Motion Matching, η οποία επαναλαμβάνει και προσαρμόζει τις κινήσεις όπως ακριβώς υλοποιούνται για τους Agents, προσφέροντας έναν ρεαλιστικό και συνεπή έλεγχο κατά τη διάρκεια του gameplay.

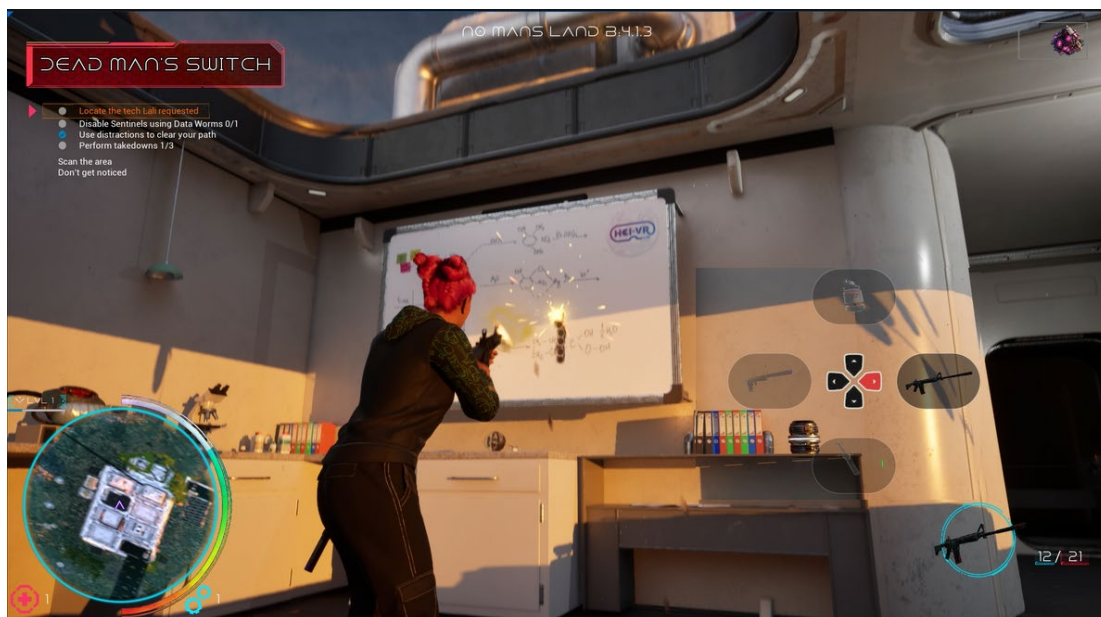


Εικόνα 182. Polycam Mesh (Blender)



Εικόνα 183. Mesh to Metahuman

Οπλισμός - HUD Wheel: Ο οπλισμός του χαρακτήρα ενσωματώνει πλήρη λειτουργικότητα, περιλαμβάνοντας τον αριθμό σφαιρών και γεμιστήρα, καθώς και animation όπλισης, αφοπλισμού, επαναγέμισης και πυροδότησης. Τα οπτικά και ηχητικά εφέ, όπως καπνοί και εκτίναξη κάλυκα, προσθέτουν ρεαλισμό στην εμπειρία. Ενσωματώθηκε επίσης ένα HUD Wheel για την γρήγορη εναλλαγή μεταξύ των όπλων και άλλων αντικειμένων, το οποίο εμφανίζει τα διαθέσιμα αντικείμενα του παίκτη με κατάλληλα εικονίδια που ανταποκρίνονται σε input μέσω keyboard ή controller.

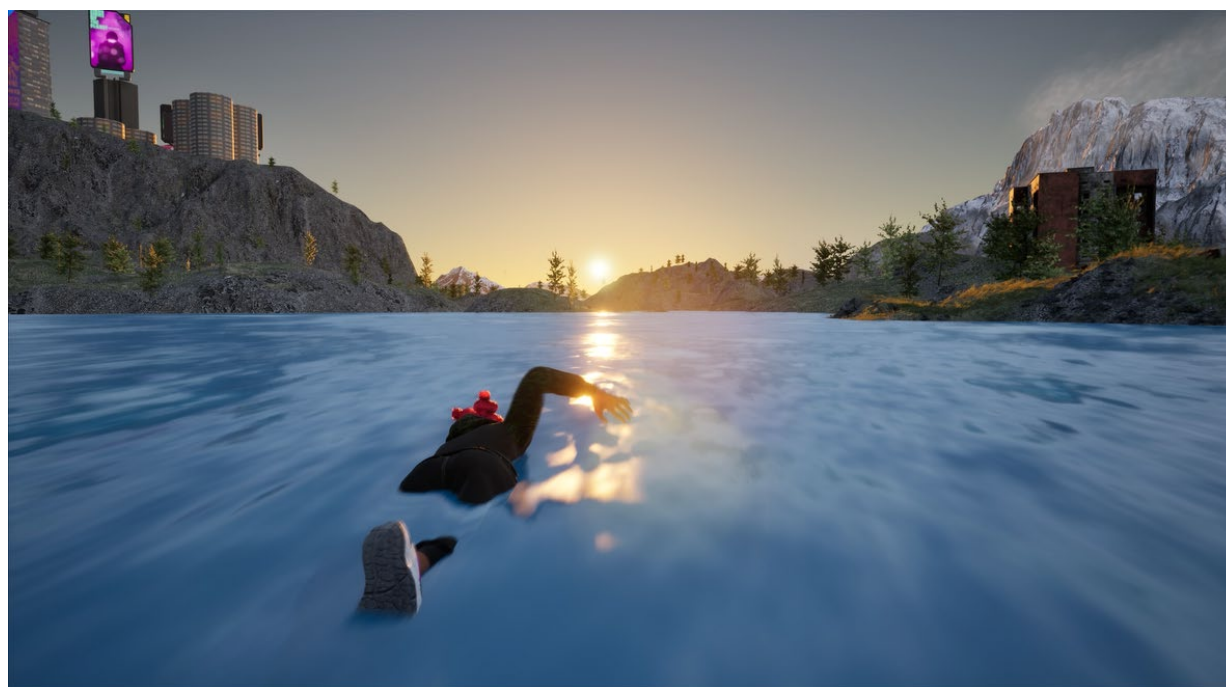


Εικόνα 184. HUD Wheel & Weapon Effects

Κολύμπι και Ισορροπία: Το παιχνίδι επεκτείνει τη δυνατότητα του παίκτη να κολυμπά και να ισορροπεί, ενσωματώνοντας platforming στοιχεία με τις δυνατότητες του MotionMatching. Αυτές οι δυνατότητες υλοποιούνται μέσω blendspaces στο animation blueprint του παίκτη και αναπαριστούν τον χαρακτήρα να κολυμπά ή να ισορροπεί όταν βρίσκεται στην κατάλληλη επιφάνεια.



Εικόνα 185. Balance Locomotion



Εικόνα 186. Swimming Locomotion

Περιβάλλον

Το περιβάλλον του παιχνιδιού σχεδιάστηκε με σκοπό να αναδείξει τη δυναμική των δέντρων συμπεριφοράς και να παρέχει στον παίκτη έναν ρεαλιστικό και ποικιλόμορφο κόσμο για εξερεύνηση και αλληλεπίδραση. Στο πλαίσιο αυτό, δημιουργήθηκε ένα ενδεικτικό Level που καλύπτει έναν εκτεταμένο χάρτη με ποικίλες φυσιολογικές περιοχές, όπως βουνά, λίμνες, έρημοι και δάση. Κάθε περιοχή έχει μοναδικά χαρακτηριστικά που αναδεικνύουν τη διαφορετική φύση και ταυτότητα του κόσμου.

Ο χάρτης αξιοποιεί ένα Heightmap που δημιουργήθηκε παραμετρικά μέσω του Houdini Engine, επιτρέποντας την παραγωγή ενός ρεαλιστικού εδάφους με ποικιλία υψομέτρων και μορφολογιών. Αυτό το Heightmap συνδυάζεται με την ανάθεση ενός Auto Material στο Landscape, το οποίο δημιουργεί δυναμικά το περιβάλλον σε πραγματικό χρόνο, ανάλογα με τις τοποθεσίες και τα χαρακτηριστικά της περιοχής, όπως το υψόμετρο και η γεωμορφολογία. Η διαδικασία αυτή γίνεται μέσω procedural placement, επιτρέποντας τον αυτόματο καθορισμό των χαρακτηριστικών του εδάφους, όπως η τοποθέτηση γρασιδιού, βράχων και χιονιού με βάση την τοποθεσία και τις φυσικές συνθήκες.

Για να εμπλουτιστεί περαιτέρω το περιβάλλον και να προσφερθεί μια αίσθηση φυσικής πληρότητας, χρησιμοποιήθηκε η τεχνική του foliage instancing για την προσθήκη δέντρων, θάμνων και κτιρίων. Αυτά τα στοιχεία επαναλαμβάνονται σε όλη την περιοχή, διασφαλίζοντας την υπολογιστική αποδοτικότητα του παιχνιδιού, καθώς η τεχνική αυτή μειώνει το φόρτο απόδοσης χωρίς να θυσιάζει την ποιότητα της εικόνας.

Ένα ακόμη στοιχείο που ενισχύει την αίσθηση του ανοιχτού κόσμου είναι η χρήση billboards με δέντρα χαμηλής ανάλυσης σε περιοχές που είναι μακριά από το camp. Αυτά τα billboards βοηθούν στην αποτύπωση της ψευδαίσθησης ενός ζωντανού, κατοικημένου κόσμου, καθώς επιτρέπουν την ύπαρξη στοιχείων του περιβάλλοντος σε απομακρυσμένες περιοχές χωρίς να απαιτούν μεγάλη υπολογιστική ισχύ.

Για να δημιουργηθεί η αίσθηση ενός τεράστιου κόσμου που είναι προσβάσιμος από τον παίκτη, προστέθηκαν distant meshes στις άκρες του χάρτη. Αυτά τα meshes, αν και δεν είναι πάντα άμεσα προσβάσιμα, παρέχουν την ψευδαίσθηση ότι ο κόσμος είναι αχανής και ανοιχτός για εξερεύνηση, ενισχύοντας την αίσθηση της απεραντοσύνης του παιχνιδιού.

Στο Level τοποθετήθηκαν επίσης διάφορα assets που αποτελούν το κεντρικό camp του παιχνιδιού. Το camp περιλαμβάνει τεχνολογικά κτίρια και αντικείμενα που ταιριάζουν με την αισθητική του παιχνιδιού και την ατμόσφαιρα του κόσμου. Τα assets αυτά εξυπηρετούν όχι μόνο τη διακόσμηση και την ατμόσφαιρα, αλλά και την τοποθέτηση των Agents και τη δυναμική του Gameplay. Ο σχεδιασμός αυτών των κτιρίων και αντικειμένων ενσωματώνει τεχνολογικά στοιχεία που υποστηρίζουν τη συμπεριφορά των χαρακτήρων και ενισχύουν την αλληλεπίδραση του παίκτη με το περιβάλλον, αναδεικνύοντας την αίσθηση της στρατηγικής και της προόδου στον κόσμο του παιχνιδιού.

Το σύνολο των παραπάνω τεχνικών και σχεδιαστικών επιλογών ενσωματώνει τη δυνατότητα δημιουργίας ενός ζωντανού, ρεαλιστικού κόσμου με ποικιλία περιβαλλόντων που εξυπηρετούν την αλληλεπίδραση με τα δέντρα συμπεριφοράς και συμβάλλουν στη συνοχή και το βάθος της αφήγησης του παιχνιδιού.



Εικόνα 187. Landscape (final version)

UI Elements - Loot - XP

Η ενσωμάτωση αντικειμένων που μπορεί να συλλέξει ο παίκτης περιλαμβάνει medkits, ammo, dataworms και Shards. Αυτά τα αντικείμενα τοποθετούνται σε διάφορα σημεία του χάρτη και προστίθενται στο inventory του παίκτη με κατάλληλα UI εικονίδια, τα οποία ενημερώνουν τον παίκτη για την ποσότητα και τον τύπο του αντικειμένου. Το Common UI Plugin χρησιμοποιείται για την παρουσίαση των πλήκτρων συλλογής και των στοιχείων UI που ανταποκρίνονται στον τύπο χειριστηρίου (keyboard ή controller).

Ενσωματώθηκε επίσης ένα Level System που επιβραβεύει τον παίκτη με XP, το οποίο επηρεάζει την πρόοδο του παίκτη και την εμφάνιση της μπάρας XP. Το σύστημα μπορεί να επεκταθεί για να ενσωματωθεί με το Quest System, ενώ παρέχει δυνατότητες για αναβάθμιση του χαρακτήρα ή αγορά αντικειμένων από NPCs τύπου Merchant.

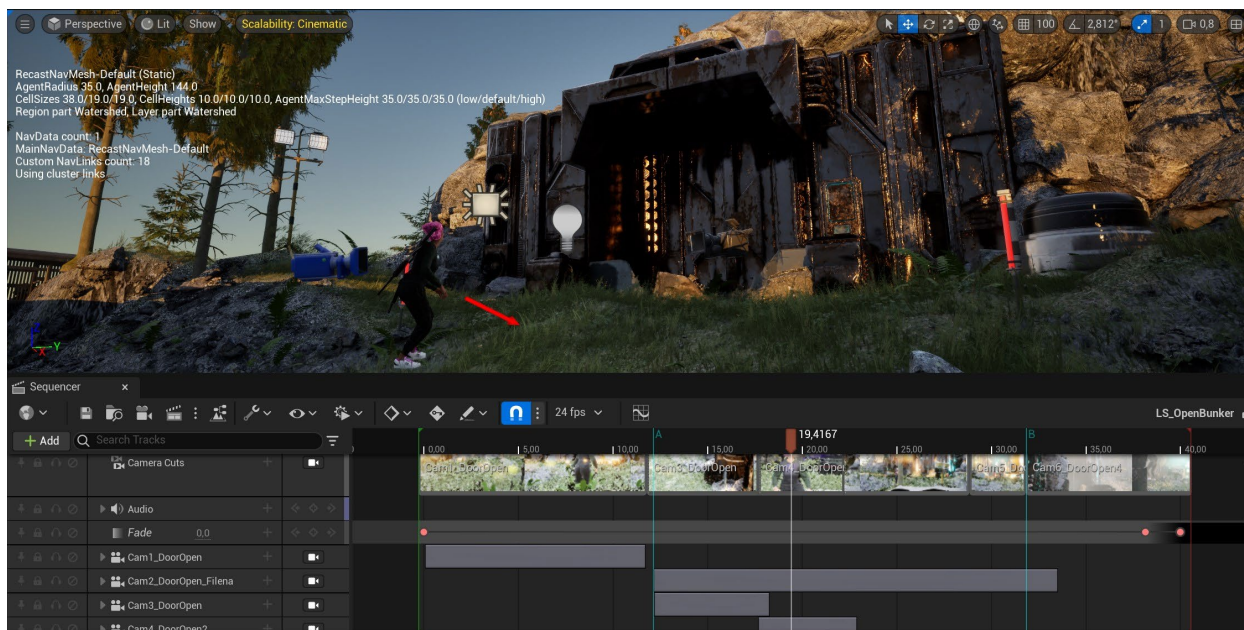


Εικόνα 188. UI Elements - Loot - XP

Cinematic Cutscenes και Motion Capture

Για την ενίσχυση της πλοκής και της συνοχής του παιχνιδιού, δημιουργήθηκαν δύο cinematic cutscenes οι οποίες ενσωματώνονται με το gameplay και εξυπηρετούν την αφήγηση του demo που αναπτύχθηκε στο πλαίσιο της διπλωματικής εργασίας. Οι cutscenes αυτές είναι στρατηγικά τοποθετημένες, μία στο ξεκίνημα και μία στο τέλος, για να προσφέρουν μια δυναμική εισαγωγή και ένα κορυφαίο σημείο κλιμάκωσης της ιστορίας, επιτρέποντας στον παίκτη να βιώσει τις πιο σημαντικές στιγμές της πλοκής.

Η πρώτη cutscene δημιουργήθηκε στο βασικό Level, το οποίο περιλαμβάνει το camp της πρωταγωνίστριας. Η σκηνή αυτή εισάγει τον παίκτη στον κόσμο του παιχνιδιού και την αποστολή που αναλαμβάνει, καθορίζοντας τον τόνο για το υπόλοιπο της εμπειρίας. Το Level Sequencer της Unreal Engine χρησιμοποιήθηκε για την παραγωγή αυτής της σκηνής, δίνοντας τη δυνατότητα για συνδυασμό animation, ήχου και σκηνικών εφέ σε μια δυναμική και συνεχόμενη αλληλεπίδραση.



Εικόνα 189. Cinematic Preview 1 (level sequencer)

Η δεύτερη cutscene, που έχει μεγαλύτερη συναισθηματική φόρτιση, εξελίσσεται στο τέλος του demo. Σε αυτήν, η πρωταγωνίστρια ανακαλύπτει ένα μυστικό εργαστήριο, αποκαλύπτοντας σημαντικές πληροφορίες για τη μητέρα της και ανοίγοντας νέες δυνατότητες στην πλοκή που ακολουθεί. Για αυτή τη σκηνή, χρησιμοποιήθηκε ένα προκατασκευασμένο Level που αποκτήθηκε από το Fab Marketplace, το οποίο παρουσιάζει μια σπηλιά γεμάτη τεχνολογικά στοιχεία. Αυτά τα τεχνολογικά assets προστέθηκαν στρατηγικά για να υποστηρίξουν την ατμόσφαιρα της σκηνής και να ενισχύσουν την αίσθηση του μυστηρίου και της εξερεύνησης. Ένα ακόμα ιδιαίτερο στοιχείο της δεύτερης cutscene ήταν η παρουσίαση της μητέρας της πρωταγωνίστριας μέσω ενός ολογράφηματος. Το εφέ αυτό δημιουργήθηκε χρησιμοποιώντας το Niagara Particle System για την παραγωγή σωματιδίων που σχηματίζουν το ολογράμμα, το οποίο κινείται σε πραγματικό χρόνο πάνω στο Metahuman mesh. Το αποτέλεσμα είναι εντυπωσιακό και ενισχύει την αίσθηση της τεχνολογίας και του μυστηρίου, επιτρέποντας στον παίκτη να δει την εικόνα της μητέρας της πρωταγωνίστριας ως μια ολογραφική απεικόνιση.



Εικόνα 190. Cinematic Preview 2

Οι διάλογοι για τις δύο cutscenes δημιουργήθηκαν με τη χρήση ΑΙ εργαλείων όπως τα Eleven Labs και PlayHT, που επέτρεψαν την παραμετροποίηση και την πλήρη τροποποίηση των φωνών, του delivery και του τόνου των χαρακτήρων. Η δυνατότητα αλλαγής των χαρακτηριστικών της φωνής, όπως η ένταση, ο ρυθμός και ο τόνος, προσέφερε την ευχέρεια να προσαρμοστούν οι διάλογοι στις ανάγκες της αφήγησης και να ενισχυθούν τα συναισθηματικά σημεία του παιχνιδιού.

Η αλληλεπίδραση των χαρακτήρων και η εκφραστικότητα τους ενισχύθηκε με τη χρήση του LiveLink της Unreal Engine και του Metahuman Animator, επιτρέποντας την αποτύπωση της κίνησης και των εκφράσεων του προσώπου των χαρακτήρων σε πραγματικό χρόνο μέσω face capture. Αυτή η τεχνολογία προσφέρει εξαιρετικά ρεαλιστική απεικόνιση των χαρακτήρων, κάνοντάς τους να φαίνονται φυσικοί και πιστοί στην ανθρώπινη εκφραστικότητα, ενισχύοντας τη σύνδεση του παίκτη με την ιστορία.

Τέλος, η μίξη των διαλόγων πραγματοποιήθηκε μέσω του Ableton Live, προσφέροντας τη δυνατότητα ενσωμάτωσης διαφορετικών ηχητικών επιπέδων και εφέ που αναδεικνύουν την ατμόσφαιρα και τα συναισθηματικά στοιχεία των σκηνών. Το τελικό editing έγινε στην πλατφόρμα Davinci Resolve, όπου συγχωνεύθηκαν τα οπτικά και ηχητικά εφέ για να δημιουργηθεί η τελική μορφή της σκηνής. Τα αρχεία ήχου και τα εφέ εισήχθησαν στο Level Blueprint της Unreal Engine για να αναπαραχθούν ακριβώς τη στιγμή που απαιτείται στο gameplay, διασφαλίζοντας τη ροή και τη συνέπεια της αφήγησης.

Με αυτόν τον τρόπο, οι cinematic cutscenes συνδυάζουν τεχνολογίες αιχμής για την ενίσχυση της αφήγησης και την προσφορά μιας συναρπαστικής και αληθοφανούς εμπειρίας στον παίκτη, ενώ παράλληλα αναδεικνύουν τις δυνατότητες της Unreal Engine και άλλων εργαλείων για τη δημιουργία ρεαλιστικών και συναισθηματικά φορτισμένων σκηνών.

11. Προκλήσεις και Μελλοντικές Βελτιώσεις

Καθώς το project εξελίσσεται και πλησιάζει σε πιο ολοκληρωμένη μορφή, προκύπτουν ευκαιρίες για τεχνικές καινοτομίες, βελτιώσεις απόδοσης, αλλά και γενική επέκταση των δυνατοτήτων του gameplay και του κόσμου. Το παρόν κεφάλαιο περιγράφει προτεινόμενες βελτιώσεις που αφορούν κρίσιμους τομείς όπως τα Behavior Trees, τα animations και τα 3D μοντέλα, την αρχιτεκτονική της τεχνητής νοημοσύνης, και τη συνολική διαχείριση πόρων και απόδοσης αναδεικνύοντας παράλληλα μειονεκτήματα και αδυναμίες του συστήματος.

Αδυναμίες και βελτιστοποίηση των Behavior Trees

Η απόδοση των Behavior Trees μπορεί να ενισχυθεί μέσω διαφόρων τεχνικών και προσεγγίσεων. Ορισμένα σημεία που χρειάζονται προσοχή και βελτίωση αναλύονται παρακάτω:

Στρατηγικές Βελτιστοποίησης: Η τοποθέτηση ελέγχων σε στρατηγικά σημεία των δέντρων μπορεί να περιορίσει τον αριθμό των κόμβων που αξιολογούνται, βελτιώνοντας την αποδοτικότητα. Παράλληλα, η ενοποίηση και η γενίκευση των συμπεριφορών θα επιτρέψουν μελλοντική επεκτασιμότητα, διευκολύνοντας την προσθήκη νέων τύπων συμπεριφορών και εχθρών. Ομοίως, η βελτιστοποίηση ή αφαίρεση μη απαραίτητων Services που εκτελούνται συνεχώς θα συμβάλει στην περαιτέρω μείωση του κόστους επεξεργασίας. Επιπλέον, η μετατροπή στοιχείων του Behavior Tree όπως Custom Tasks, Decorators ή Services από Blueprints σε C++ θα προσφέρει σημαντική βελτίωση στην απόδοση, δεδομένου ότι ο C++ κώδικας εκτελείται πολύ πιο αποδοτικά.

Διαχείριση Ελέγχων και μεταβάσεων: Κατά τη διάρκεια της υλοποίησης των Behavior Trees, παρατηρήθηκε δυσχρηστία στον ορισμό πολύπλοκων μεταβάσεων και στην ανάπτυξη διαφοροποιημένων ενεργειών, οι οποίες εξαρτώνται από την προηγούμενη κατάσταση του Agent. Το Detecting State, το οποίο αποτέλεσε βασικό άξονα της διαχείρισης μεταβάσεων του Agent, κρίθηκε ακατάλληλο για την λήψη αποφάσεων μέσω του Behavior Tree. Αντιθέτως, κρίθηκε αναγκαίο το δέντρο να αδρανοποιηθεί πλήρως προκειμένου να μην επηρεαστεί ο κώδικας του AI Controller που διαχειρίζεται αυτές τις μεταβάσεις. Ως αποτέλεσμα, παρατηρήθηκε ότι το Behavior Tree αποδεικνύεται πιο αποτελεσματικό για τη λήψη μεγάλων αποφάσεων και όχι για την εκτέλεση ενεργειών και σύνθετων ελέγχων.

Διαχείριση Καταστάσεων: Το Seeking State δημιουργήθηκε με σκοπό την αναζήτηση εχθρικής παρουσίας και ενεργοποιείται από τρεις περιπτώσεις: όταν ο Agent χάνει τον παίκτη από το οπτικό του πεδίο, όταν εντοπίζει έναν εξουδετερωμένο σύμμαχο ή όταν η μπάρα ανίχνευσης φτάνει σε υψηλά επίπεδα. Παρά την αποτελεσματικότητα του συστήματος, η χρήση ενός και μόνο Seeking State αύξησε την πολυπλοκότητα της υλοποίησης, γεγονός που ήταν αποδεκτό στο πλαίσιο του έργου. Ωστόσο, αυτό ανέδειξε την ανάγκη για βελτίωση στην ανάπτυξη συστημάτων που απαιτούν διαφοροποιημένες ενέργειες, ανάλογα με την προηγούμενη κατάσταση του Agent και την εκάστοτε κατάσταση του παιχνιδιού. Στο μέλλον, η εισαγωγή ενός υβριδικού συστήματος, το οποίο συνδυάζει Finite State Machines (FSM) για την αποτελεσματική διαχείριση μεταβάσεων με την χρήση Behavior Trees για τη λήψη αποφάσεων υψηλού

επιπέδου, θα μπορούσε να προσφέρει ένα πιο ευέλικτο και επεκτάσιμο πλαίσιο για την ανάπτυξη τέτοιων συστημάτων.

Σχεδιασμός και Συντήρηση του Συστήματος: Μια σημαντική πρόκληση που παρατηρήθηκε κατά τη διάρκεια της ανάπτυξης ήταν η εξάρτηση των μεταβλητών του Blackboard και των λειτουργιών του AI Controller. Κάθε νέα ενέργεια που προστίθεται απαιτούσε τη δημιουργία νέου Blackboard key, με αποτέλεσμα να χρειάζεται επανεκτίμηση και αναπροσαρμογή των μεταβλητών και των ελέγχων στο AI Controller για να εξασφαλιστεί ότι οι αλλαγές ενσωματώνονται σωστά και χωρίς απρόβλεπτες συνέπειες.

Η σχεδίαση και υλοποίηση των Behavior Trees απαιτεί προσοχή στον καθορισμό των ενεργειών και των ελέγχων που διαχειρίζονται τα βασικά στοιχεία της τεχνητής νοημοσύνης, όπως τα Blueprints του Agent, το AI Controller και τα assets του Behavior Tree. Η απλοποίηση του Behavior Tree, χρησιμοποιώντας το μόνο για τη λήψη γενικών αποφάσεων σχετικά με τη συμπεριφορά του Agent, φαίνεται να είναι η πιο αποτελεσματική προσέγγιση. Η μελλοντική βελτίωση μπορεί να επιτευχθεί μέσω της υιοθέτησης συνδυασμένων προσεγγίσεων όπως τα Finite State Machines για τον ορισμό πολύπλοκων συνθηκών των μεταβάσεων μεταξύ των καταστάσεων και τα Behavior Trees για τις συνολικές αποφάσεις συμπεριφοράς, ενσωματώνοντας παράλληλα τις ανάγκες του παιχνιδιού για ρεαλιστική και επεκτάσιμη τεχνητή νοημοσύνη.

Animations & 3D Models

Η διαχείριση πολλαπλών και οπτικά διαφοροποιημένων 3D χαρακτήρων στο project οδήγησε σε σημαντικά ζητήματα απόδοσης και τεχνικής πολυπλοκότητας, κυρίως λόγω των γεωμετρικών διαφορών μεταξύ των σκελετών και της ανάγκης χρήσης οδηγού σκελετού (skeleton retargeting) για την αναπαραγωγή των κινήσεων. Αυτή η προσέγγιση αποδείχθηκε ιδιαίτερα απαιτητική σε υπολογιστικούς πόρους, καθιστώντας τη συνολική εφαρμογή των λειτουργιών πιο βαριά.

Μια βελτιστοποιημένη λύση είναι η ανάπτυξη και υιοθέτηση ενός προσαρμοσμένου modular σκελετού που να χρησιμοποιείται καθολικά σε όλο το project. Ένας τέτοιος σκελετός μπορεί να υποστηρίξει την εναλλαγή και τοποθέτηση διαφορετικών οπτικών εξαρτημάτων (modules), επιτρέποντας τη δημιουργία ποικιλίας στην εμφάνιση των Agents χωρίς την ανάγκη διαφορετικών skeleton assets. Αυτή η προσέγγιση όχι μόνο εξαλείφει την ανάγκη για real-time retargeting, μειώνοντας έτσι το υπολογιστικό κόστος, αλλά ενισχύει και την ευελιξία ανάπτυξης μέσω της αξιοποίησης της δομής parent-child Blueprints, διευκολύνοντας την επεκτασιμότητα και την οργάνωση των χαρακτήρων.

Combat Director/Agent Spawner

Στο πλαίσιο πλήρους ανάπτυξης του παιχνιδιού με χαρακτηριστικά open-world, όπως πολλαπλά επίπεδα και στρατόπεδα camps κατανεμημένα στον χάρτη, η διαχείριση της παρουσίας των εχθρικών χαρακτήρων μπορεί να οργανωθεί μέσω ενός ειδικού Blueprint Actor asset, το οποίο τοποθετείται εντός του Level και λειτουργεί ως σύστημα διαχείρισης spawn για τους Agents. Παράλληλα, μπορεί να αναλάβει και τον συντονισμό σεναριακών γεγονότων, όπως επιθέσεις και περιπολίες.

Συγκεκριμένα, ο Actor αυτός επιτρέπει τη δυναμική και παραμετρική δημιουργία χαρακτήρων, χωρίς να χρειάζεται να είναι χειροκίνητα τοποθετημένοι στο Level. Η χρήση του μπορεί να παρέχει σημαντική εξοικονόμηση υπολογιστικών πόρων, αφού οι Agents κάθε camp μπορούν να δημιουργούνται μόνο όταν ο παίκτης πλησιάζει το αντίστοιχο camp, αποφεύγοντας άσκοπο υπολογιστικό κόστος όταν ο παίκτης βρίσκεται μακριά. Επιπλέον, η δημιουργία των Agents μπορεί να καθοδηγείται από διάφορες παραμέτρους όπως:

- Η γεωγραφική θέση του camp
- Το επίπεδο δυσκολίας
- Το στάδιο προόδου της πλοκής

Μέσα από αυτές τις παραμέτρους, κάθε camp μπορεί να αποκτήσει μοναδικά χαρακτηριστικά, όπως:

- Ο συνολικός αριθμός των Agents
- Τα σημεία που θα προστατεύουν
- Οι τύποι των Agents που θα εμφανιστούν
- Ο αριθμός και οι θέσεις των Patrol Points

Με τη συγκεκριμένη προσέγγιση, διασφαλίζεται ότι ο παίκτης δε θα έρχεται συνεχώς αντιμέτωπος με τους ίδιους εχθρικούς χαρακτήρες, ενισχύοντας την ποικιλία και τον ρεαλισμό της εμπειρίας.

Ο Actor αυτός μπορεί επίσης να υποστηρίζει τη διαγραφή των Agents βάσει παραμέτρων, για παράδειγμα:

- Μετά την παρέλευση προκαθορισμένου χρονικού διαστήματος
- Όταν ο παίκτης απομακρυνθεί από την περιοχή και είναι απίθανο να επιστρέψει

Επιπρόσθετα, ο Combat Director μπορεί να πυροδοτεί σεναριακά γεγονότα κατά τη διάρκεια του παιχνιδιού, όπως π.χ. όταν ο παίκτης περνά από ένα καθορισμένο trigger σημείο. Σε αυτή την περίπτωση, μπορεί να ανατίθεται σε έναν Agent η έναρξη περιπολίας προς το συγκεκριμένο σημείο, αυξάνοντας την ένταση και προκαλώντας τον παίκτη να πάρει ταχύτατες αποφάσεις. Ομοίως, μπορεί να χρησιμοποιηθεί και σε σενάρια επίθεσης.

Τέλος, ο Combat Director μπορεί να αναλάβει και γενικότερα καθήκοντα συντονισμού της ατμόσφαιρας του παιχνιδιού, όπως την επιλογή της κατάλληλης μουσικής υπόκρουσης ανάλογα με την κατάσταση (μάχη, εξερεύνηση, stealth κ.λπ.).

Στο πλαίσιο του συγκεκριμένου project, καθώς το παιχνίδι δεν αναπτύχθηκε πλήρως αλλά μόνο για την επίδειξη των Behavior Trees, οι παραπάνω λειτουργίες υλοποιήθηκαν με απλό τρόπο. Συγκεκριμένα, εντός του Level τοποθετήθηκαν Box Collisions, τα οποία πυροδοτούν ενέργειες μέσω του Level Blueprint, όπως:

- Ανάθεση περιπολίας σε έναν Agent
- Πυροδότηση διαλόγων
- Ενεργοποίηση μουσικής gameplay

Ωστόσο, αυτή η προσέγγιση δεν μπορεί να υποστηρίξει κλιμακούμενη και οργανωμένη ανάπτυξη, γεγονός που καταδεικνύει την ανάγκη υλοποίησης ενός πιο ολοκληρωμένου Combat Director Actor για το τελικό παιχνίδι.

LODs Σκελετών και Αντικειμένων, Χρήση Nanite & Απλουστευμένα Animations

Η απόδοση του project μπορεί να βελτιωθεί περαιτέρω μέσω της χρήσης LODs (Levels of Detail), τόσο για τους σκελετούς χαρακτήρων όσο και για τα αντικείμενα του περιβάλλοντος. Με τον τρόπο αυτό, αντικείμενα και χαρακτήρες που βρίσκονται σε μεγάλη απόσταση από τον παίκτη μπορούν να εμφανίζονται με χαμηλότερης ποιότητας μοντέλα, μειώνοντας σημαντικά το υπολογιστικό φορτίο.

Για τα στατικά αντικείμενα, αξίζει να εξεταστεί η ενσωμάτωση του συστήματος Nanite, το οποίο αναλαμβάνει αυτόματα τη διαχείριση του επιπέδου λεπτομέρειας (LOD). Το Nanite επιτρέπει την απόδοση αντικειμένων με υψηλό αριθμό τριγώνων (triangles) και οπτική πιστότητα, χωρίς να απαιτείται πολύπλοκη ή χειροκίνητη διαδικασία ρύθμισης. Έτσι, διατηρείται υψηλή οπτική ποιότητα κοντά στον παίκτη και ταυτόχρονα μειώνεται το υπολογιστικό κόστος απόδοσης μακρινών αντικειμένων.

Επιπλέον, με την έκδοση beta του Unreal Engine 5.5, έχει προστεθεί υποστήριξη του συστήματος Nanite για Skeletal Meshes, γεγονός που ανοίγει νέες δυνατότητες και για τη βελτιστοποίηση της απόδοσης των κινούμενων χαρακτήρων.

Τέλος, μια επιπλέον τεχνική για τη βελτίωση της απόδοσης αφορά τα animations των Agents. Συγκεκριμένα, είναι δυνατόν να σχεδιαστούν animations που χρησιμοποιούν λιγότερα bones από τον πλήρη σκελετό, μειώνοντας έτσι το υπολογιστικό κόστος που απαιτείται για την αναπαραγωγή τους. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη για Agents που βρίσκονται σε μεγάλη απόσταση από τον παίκτη, καθώς μπορούν να συνεχίζουν να εκτελούν βασικά animations χωρίς την περιττή ακρίβεια που απαιτούν τα animations κοντινής εμβέλειας.

Συνολικά, η εφαρμογή αυτών των τεχνικών προσφέρει μια ισορροπία μεταξύ οπτικής ποιότητας και αποδοτικής χρήσης των πόρων, που είναι κρίσιμη για την ανάπτυξη ενός λειτουργικού και επεκτάσιμου open world συστήματος.

Project Migration: Κατά την ανάπτυξη, το project απέκτησε μεγάλο μέγεθος (65,3 GB) λόγω των πολλών assets που εισήχθησαν, πολλά εκ των οποίων δεν χρησιμοποιήθηκαν. Η διαδικασία migration σε νέο project, που θα περιλαμβάνει μόνο τα χρησιμοποιημένα αρχεία, θα μπορούσε να μειώσει αισθητά το μέγεθος και να διευκολύνει τη δημιουργία εκτελέσιμων builds μέσω της διαδικασίας packaging. Ένα

μικρότερο, καθαρότερο project διευκολύνει επίσης την οργάνωση και την μελλοντική συντήρηση του κώδικα.

12.Συμπεράσματα

Η παρούσα εργασία ανέδειξε τη σημασία των Behavior Trees στην ανάπτυξη ρεαλιστικής και ευέλικτης τεχνητής νοημοσύνης για NPCs, στο πλαίσιο της δημιουργίας του παιχνιδιού No Man's Land. Μέσα από τη χρήση της Unreal Engine 5.4 και του οπτικού προγραμματισμού Blueprints, κατέστη δυνατή η δημιουργία δυναμικών, διαδραστικών και συναισθηματικά προσαρμοσμένων χαρακτήρων, που ενισχύουν την εμπειρία του παίκτη.

Τα Behavior Trees επέτρεψαν την οργανωμένη υλοποίηση στρατηγικών αντίδρασης και την ενσωμάτωση περιβαλλοντικών και εσωτερικών παραμέτρων στη συμπεριφορά των NPCs. Η σύζευξη αυτών των τεχνικών με τα εξελιγμένα γραφικά, το σύστημα μάχης και τον αφηγηματικό κορμό του παιχνιδιού συντέλεσε σε μια πολυδιάστατη και καθηλωτική εμπειρία gameplay.

Η ισχυρή μηχανή του Unreal Engine και το ευρύ οικοσύστημα assets που προσφέρει επιτρέπουν ταχύτατο πρωτότυπο σχεδιασμό τόσο για την ανάπτυξη Behavior Trees με σύνθετη συμπεριφορά, όσο και για την ενσωμάτωση επιπρόσθετων λειτουργιών που ενδυναμώνουν την αίσθηση ρεαλισμού των NPCs και του περιβάλλοντος τους. Με αυτόν τον τρόπο, αναδείχθηκε η ισχυρή ικανότητα της Unreal Engine στην υποστήριξη πολυδιάστατων projects, επιτρέποντας ακόμη και σε νέους χρήστες να δημιουργήσουν και να αναπτύξουν σύνθετα συστήματα.

Η εφαρμογή των Behavior Trees αποδείχθηκε ιδιαίτερα ευέλικτη, επεκτάσιμη και ικανή να ανταποκριθεί στις υψηλές απαιτήσεις του σύγχρονου game design, αποδεικνύοντας ότι τα Behavior Trees παραμένουν μια επίκαιρη και αποτελεσματική προσέγγιση για τη μοντελοποίηση σύνθετης τεχνητής νοημοσύνης.

Βιβλιογραφικές αναφορές

Ξενόγλωσσες

- Argentics (2025). *Adaptive NPC Behavior*. Ανακτήθηκε February 20, από <https://medium.com/@argentics/advanced-ai-programming-for-adaptive-npc-behavior-91bad6236f95>
- Biggar, O., Zamani, M., & Shames, I. (2020). A principled analysis of behavior trees and their generalisations. arXiv preprint <https://arxiv.org/abs/2008.11906>.
- Butler, S., & Oliver, T. (2024). *Game Development Patterns with Unreal Engine 5: Build maintainable and scalable systems with C++ and Blueprint*. Packt Publishing Ltd.
- Colledanchise, M., & Ögren, P. (2018). *Behavior trees in robotics and AI: An introduction*. CRC Press.
- Costa, V.G., Pedreira, C.E. Recent advances in decision trees: an updated survey. *Artif Intell Rev* 56, 4765–4800 (2023). <https://doi.org/10.1007/s10462-022-10275-5>
- Epic Games(2024). Unreal Engine documentation. Retrieved January 13, 2025, from <https://www.unrealengine.com/en-US/blog/unreal-engine-5-4-is-now-available>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 13, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-blueprints-visual-scripting-in-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 20, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-trees-in-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/blend-spaces-in-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/state-machines-in-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/motion-matching-in-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-niagara-effects-for-unreal-engine>
- Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/metahuman/metahuman-documentation>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine>

Epic Games (2024). Unreal Engine documentation. Retrieved January 13, 2025, from <https://www.unrealengine.com/en-US/tech-blog/explore-the-new-updates-to-the-game-animation-sample-project-in-ue-5.5>

Epic Games (2025). Unreal Engine documentation. Retrieved January 13, 2025, from <https://www.fab.com/listings/ef9651a4-fb55-4866-a2d9-1b38b028f9c7>

Epic Games (2025). Unreal Engine documentation. Retrieved January 13, 2025, from <https://www.fab.com/listings/93faede1-4434-47c0-85f1-bf27c0820ad0>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-tree-in-unreal-engine---overview>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/ai-perception-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/basic-navigation-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprint-interface-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from https://dev.epicgames.com/documentation/en-us/unreal-engine/environment-query-system-in-unreal-engine?application_version=5.5

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine#nanite-skeletal-mesh>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/components-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/ik-rig-animation-retargeting-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/animation-retargeting-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/animation-blueprints-in-unreal-engine>

Epic Games(n.d.). Unreal Engine documentation. Retrieved January 18, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/working-with-modular-characters-in-unreal-engine>

Roko's Basilisk. (2022, Nov 30). Retrieved April 1, 2024, from <https://www.lesswrong.com/w/rokos-basilisk>

Faber, S. (2024). Technological Bodies: Artificial Intelligence as Friday in Portal and Halo Infinite. *Nordic Journal of English Studies*, 23(S1), 128–144. doi: <https://doi.org/10.35360/njes.2024.23311>

Francis, A. (2017). Overcoming pitfalls in behavior tree design. In *Game AI Pro 3* (pp. 115-125). AK Peters/CRC Press.

Millington, I. & Funge, J. (2009). *ARTIFICIAL INTELLIGENCE FOR GAMES*. Burlington: Elsevier.

Thompson, T. (2025). *Primal Instinct | Companion AI in Far Cry Primal*. Ανακτήθηκε February 19, από <https://www.gamedeveloper.com/design/primal-instinct-companion-ai-in-far-cry-primal>

Togelius, J., & Yannakakis, G. N. (2018). *The Next Level: A Survey of Computer Games and AI*. Springer.

Tomai, E., & Flores, R. (2014). Adapting In-Game Agent Behavior by Observation of Players Using Learning Behavior Trees. *Foundations of Digital Games 2014*.

Z. P. Takács, M. Vincze and G. Eigner, "Development of Gaming AI: A Case Study," *2024 IEEE 22nd Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, Pula, Croatia, 2024, pp. 347-352, doi: [10.1109/SISY62279.2024.10737591](https://doi.org/10.1109/SISY62279.2024.10737591).

Ελληνόγλωσσες

Κωσταρέλου, Α. (2024). *Ανάπτυξη Δυναμικών και Συναισθηματικά Ανταποκρινόμενων NPCs στην Unreal Engine 5 Χρησιμοποιώντας Δέντρα Προσαρμοστικής Συμπεριφοράς*. Πανεπιστήμιο Πειραιώς, Αθήνα.