ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΠΜΣ ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΒΙΝΤΕΟΠΑΙΧΝΙΔΙΟΥ ΠΟΛΛΑΠΛΩΝ ΠΑΙΚΤΩΝ ΣΕ ΜΕΤΑΒΑΛΛΟΜΕΝΟ ΠΕΡΙΒΑΛΛΟΝ ΜΕ ΤΟ UNITY

Επιβλέπων Καθηγητής: Δημητρουλάκος Γρηγόριος
Συνεπιβλέπων Καθηγητής: Βασιλάκης Κωνσταντίνος

Μεταπτυχιακός Φοιτητής: Νικολακάκος Δημήτριος
| Α.Μ.: 2022202202012

Τρίπολη | Μάρτιος 2024

Περιεχόμενα

Ευρετήριο εικόνων

Περίληψη

Στο χώρο των σύγχρονων βιντεοπαιχνιδιών, η δυνατότητα ταυτόχρονης σύνδεσης πολλαπλών παικτών, οι οποίοι είτε συναγωνίζονται είτε ανταγωνίζονται μεταξύ τους, αποτελεί ένα χαρακτηριστικό το οποίο συχνά καθορίζει την πορεία και την δημοτικότητα ενός παιχνιδιού. Οι σύγχρονες μηχανές ανάπτυξης παιχνιδιών είναι αρκετές και πανίσχυρες, προσφέροντας στους προγραμματιστές μια πληθώρα εργαλείων, ικανών να συμβάλλουν σε όλη την διαδικασία ανάπτυξης του παιχνιδιού. Εξίσου πανίσχυρες είναι και οι τεχνολογίες υλοποίησης της ταυτόχρονης σύνδεσης πολλών παικτών (multiplayer), με την επιλογή της καταλληλότερης να αποτελεί ιδιαίτερα δύσκολη απόφαση.

Στα πλαίσια της παρούσας διπλωματικής εργασίας εξετάζεται εις βάθος η μηχανή παιχνιδιών Unity σε συνδυασμό με τα βασικά χαρακτηριστικά που τη διέπουν. Το Unity, αποτελεί μια εκ των δημοφιλέστερων και πληρέστερων μηχανών ανάπτυξης παιχνιδιών, με μερικά από τα πιο επιτυχημένα παιχνίδια των τελευταίων ετών, να έχουν βασιστεί σε αυτή. Το Unity παρέχει επίσης τα εργαλεία και τη λογική για την ανάπτυξη παιχνιδιών πολλαπλών παικτών μέσω του Netcode for GameObjects, το οποίο συμβάλλει στην ανάπτυξη συναρπαστικών παιχνιδιών. Για την περαιτέρω ανάλυση και παρουσίαση των παραπάνω, υλοποιήθηκε ένα Multiplayer Third Person Shooter παιχνίδι, δηλαδή ένα παιχνίδι πολλαπλών παικτών τρίτου προσώπου για υπολογιστή, όπου οι παίκτες θα μπορούν να αλληλεπιδρούν με το ευρύτερο περιβάλλον τους, έχοντας την ικανότητα καταστροφής των αντικειμένων της πίστας. Για την υλοποίηση και δημιουργία καταστρέψιμων αντικειμένων, αξιοποιήθηκε το πακέτο RayFire, οι υπηρεσίες του οποίου ήταν πολύτιμες για την επίτευξη του τελικού αποτελέσματος. Επιπρόσθετα, τα περισσότερα εκ των αντικειμένων που απαρτίζουν την πίστα, σχεδιάστηκαν εξ ’ολοκλήρου στα πλαίσια της παρούσας διπλωματικής εργασίας, με στόχο να προσδώσουν το στοιχείο της ρεαλιστικότητας στο τελικό αποτέλεσμα.

Στα πρώτα κεφάλαια της εργασίας γίνεται αναφορά στα βασικά χαρακτηριστικά και εργαλεία του Unity, εστιάζοντας ιδιαίτερα σε αυτά που χρησιμοποιήθηκαν περισσότερο για την ανάπτυξη του παιχνιδιού. Ύστερα, αναλύεται σε βάθος η φιλοσοφία και τα επιπρόσθετα χαρακτηριστικά του παιχνιδιού, εστιάζοντας κυρίως στον τρόπο ανάπτυξης της πίστας, των αντικειμένων, του χαρακτήρα και των διεπαφών που θα έχουν στη διάθεσή τους οι παίκτες. Παράλληλα, γίνεται εκτενής αναφορά στις μεθόδους υλοποίησης του multiplayer καθώς και στον τρόπο που αυτά αναπτύχθηκαν στα πλαίσια του παιχνιδιού. Εντός του παιχνιδιού, παρέχεται η δυνατότητα στους παίκτες να συνδέονται μεταξύ τους είτε μέσω dedicated server είτε να δημιουργούν οι ίδιοι δωμάτια (host), τα οποία φιλοξενούνται στους υπολογιστές τους και είναι διαθέσιμα προς σύνδεση για όλους τους υπόλοιπους παίκτες. Για όλα τα παραπάνω, παρέχονται οι αλγόριθμοι

που υλοποιήθηκαν σε συνδυασμό με την επεξήγηση της λειτουργίας του καθενός. Οι τελικές ενότητες, αφορούν τη διαδικασία δημιουργίας των εκτελέσιμων αρχείων των παικτών και των servers.

Απώτερος στόχος της παρούσας εργασίας είναι η παρουσίαση και ανάλυση ολόκληρης της πορείας ανάπτυξης ενός βιντεοπαιχνιδιού για ηλεκτρονικό υπολογιστή μέσω της ενδελεχούς περιγραφής αρκετών πτυχών αυτής. Η ανάπτυξη παιχνιδιών αποτελεί ένα αρκετά πολυσύνθετο αντικείμενο και η δημιουργία ενός ολοκληρωμένου βιντεοπαιχνιδιού αποτελεί απόρροια της συνεργασίας διαφόρων ειδικών. Πιο συγκεκριμένα, ένα πλήρες παιχνίδι, κατά το ταξίδι ανάπτυξης του, περνά από στάδια όπως ο σχεδιασμός επιπέδων (level design), ο σχεδιασμός γραφικών και τέχνης (art design), η ανάπτυξη της λειτουργικότητας (development, coding), η δυνατότητα ύπαρξης πολλαπλών παικτών (multiplayer), η ηχητική σχεδίαση (sound design), οι δοκιμές και αποσφαλμάτωση (testing, debugging), η βελτιστοποίηση (optimization) και αρκετά άλλα. Στόχος λοιπόν της παρούσας διπλωματικής εργασίας, ήταν η ανάδειξη και η όσο το δυνατόν, βαθύτερη εντρύφηση με τα προαναφερθέντα αντικείμενα. Μεγάλης σημασίας επίσης, αποτέλεσε η παρουσίαση και ανάδειξη των τεχνικών υλοποίησης της δυνατότητας σύνδεσης πολλαπλών παικτών (multiplayer), αναλύοντας παράλληλα τα βασικά τους χαρακτηριστικά. Το multiplayer, αποτελεί ένα αντικείμενο που χρήζει ιδιαίτερου χειρισμού, καθώς σε περίπτωση λανθασμένης μεταχείρισής του, το αποτέλεσμα δεν θα είναι ελκυστικό προς τους παίκτες, κάνοντας το παιχνίδι να αποκλίνει από τον αρχικό του στόχο, αυτόν δηλαδή της διασκέδασης των χρηστών. Τελικά, καθώς τα αρχεία κώδικα (scripts) που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας είναι αρκετά, ο συνολικός κώδικας στον οποίο βασίστηκε το παιχνίδι που θα περιγράφει στη συνέχεια, βρίσκεται στην παρακάτω διεύθυνση[1].

Λέξεις – κλειδιά: Unity, 3D, Multiplayer, Netcode For GameObjects, Βιντεοπαιχνίδια

[1] https://github.com/DimNik9/Scripts

Abstract

In the realm of modern video games, the ability for simultaneous connection of multiple players, who either cooperate or compete, is a characteristic that often determines the course and popularity of a game. The modern game development engines are numerous and powerful, providing developers with a plethora of tools capable of contributing to the entire game development process. Equally powerful are the technologies for implementing multiplayer connections, with choosing the most appropriate one being a particularly challenging decision.

Within the framework of this thesis, the Unity game engine is thoroughly examined in combination with its underlying features. Unity is one of the most popular and comprehensive game development engines, with some of the most successful games in recent years being based on it. Unity also provides the tools and logic for developing multiplayer games through Netcode for GameObjects, which contributes to the development of exciting games. For further analysis and presentation of the above, a Multiplayer Third Person Shooter game was implemented. This game allows multiple players to interact within its environment, with the ability to destroy objects on the map. The RayFire package was utilized for implementing destructible objects, which was instrumental in achieving the result. Additionally, most of the objects comprising the map were designed entirely within the scope of this thesis, aiming to add an element of realism to the outcome.

The initial chapters of the thesis refer to the basic characteristics and tools of Unity, focusing particularly on those used most for the development of the game. Then, the philosophy and additional features of the game are analyzed in depth, focusing on the development of the map, objects, character, and interfaces available to the players. Extensive reference is also made to the methods of implementing multiplayer and how they were developed within the game. Within the game, players have the option to connect either through dedicated servers or by creating their own rooms (host), which are hosted on their computers and are available for connection to all other players. Algorithms implemented in conjunction with the explanation of their operation are provided for all the above. The final sections concern the process of creating executable files for players and servers.

The goal of this thesis is to present and analyze the entire development process of a computer video game through a detailed description of various aspects. Game development is a complex subject, and creating a complete video game is the result of collaboration among various specialists. Specifically, a complete game, during its development journey, goes through stages such as level design, art design, development, coding, multiplayer capability, sound design, testing, debugging,

optimization, and many others. Therefore, the aim of this thesis was to highlight and delve as deeply as possible into these aspects. Also of significant importance was the presentation and highlighting of the technical implementation of the multiplayer capability, while simultaneously analyzing its basic characteristics. Multiplayer is a subject that requires careful handling, as mishandling it can lead to unattractive results for players, deviating the game from its original goal, which is the enjoyment of users. Eventually, since the code files (scripts) developed within the scope of this thesis are quite numerous, the total code upon ,which the game described below, is based can be found at the following address[2].

Keywords: Unity, 3D, Multiplayer, Netcode for GameObjects, Video games

[2] https://github.com/DimNik9/Scripts

Πρόλογος

Η παρούσα διπλωματική εργασία αποτέλεσε απόρροια της επαγγελματικής και προσωπικής μου ενασχόλησής με τον προγραμματισμό σε συνδυασμό με την αγάπη μου για τα βιντεοπαιχνίδια, τα οποία και με συντρόφευαν από τα πρώτα χρόνια αλληλεπίδρασής μου με τους ηλεκτρονικούς υπολογιστές. Βασικός αρωγός σε αυτό, ο οποίος συντέλεσε καθοριστικά, τόσο στην επιλογή του αντικειμένου της διπλωματικής εργασίας, όσο και στην ολοκλήρωσή της, ήταν ο επιβλέπων καθηγητής κ. Δημητρουλάκος Γρηγόριος, του οποίου οι τομείς ενασχόλησης και τα ενδιαφέροντα, αποτέλεσαν το έναυσμα για όλα όσα περιγράφονται παρακάτω. Η στήριξη του σε συνδυασμό με τις ιδέες του, λειτούργησαν ως πυλώνας γύρω από τον οποίο οικοδομήθηκε ολόκληρο το περιεχόμενο της εργασίας.

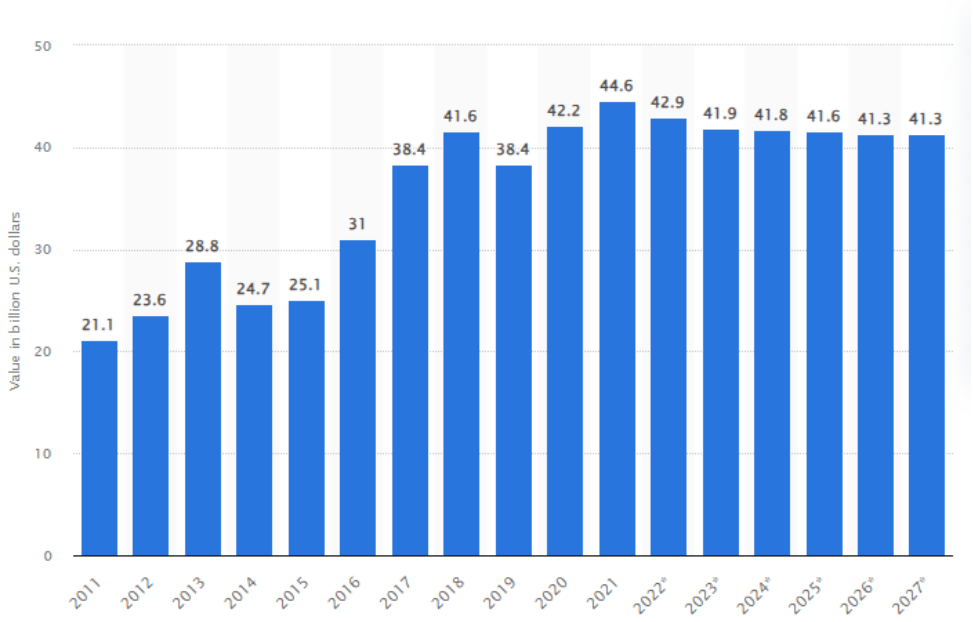
Κύριο αντικείμενο της εργασίας είναι η ανάπτυξη παιχνιδιού πολλαπλών παικτών σε καταστρέψιμο περιβάλλον, όπου οι χρήστες θα μπορούν να συνδέονται μεταξύ τους αξιοποιώντας διάφορα μοντέλα και αρχιτεκτονικές συνδεσιμότητας. Στο πλαίσιο αυτό, περιγράφεται η μηχανή παιχνιδιών Unity, το Unity for GameObjects για την ανάπτυξη όλων των διαδικτυακών λειτουργιών του παιχνιδιού και το πακέτο RayFire για την δημιουργία αντικειμένων τα οποία θα καταστρέφονται βάσει των επιλογών των παικτών. Όλα τα παραπάνω, βασίστηκαν στην εμπεριστατωμένη μελέτη βιβλίων, ηλεκτρονικών βιβλιοθηκών, των επίσημων εγχειριδίων της εκάστοτε τεχνολογίας, όπως επίσης και στο άφθονο υλικό που παρέχεται μέσω των διαφόρων προγραμματιστικών κοινοτήτων.

Το αποτέλεσμα αποδείχθηκε αντάξιο των προσδοκιών μου και ήταν απόρροια της συνεχούς προσπάθειας, αποτελώντας διαρκώς κίνητρο για περαιτέρω μελέτη και βελτίωση του παιχνιδιού. Μείζονος σημασίας ήταν η συνεισφορά της οικογένειας μου καθόλη τη διάρκεια των μεταπτυχιακών μου σπουδών προσφέροντας μου την απαραίτητη στήριξη και καθοδήγηση. Ευελπιστώ τελικά, η παρούσα διπλωματική εργασία, να αποτελέσει τη βάση για περαιτέρω μελέτη και έρευνα και να συνεισφέρει ουσιαστικά σε αντίστοιχες εργασίες από ερευνητές εξίσου γοητευμένους από τον κόσμο των βιντεοπαιχνιδιών.

1. Εισαγωγή

Στον σύγχρονο κόσμο της ψυχαγωγίας, τα παιχνίδια σε προσωπικό υπολογιστή αντιπροσωπεύουν έναν συναρπαστικό και συνεχώς αναπτυσσόμενο κλάδο της βιομηχανίας του ηλεκτρονικού αθλητισμού. Η εξέλιξη της τεχνολογίας έχει διαμορφώσει μια νέα εποχή όπου οι χρήστες μπορούν να επικοινωνούν με εικονικούς κόσμους, να ανταγωνίζονται μεταξύ τους και να απολαμβάνουν συναρπαστικές εμπειρίες. Η ανάπτυξη του gaming σε υπολογιστή έχει επηρεάσει όχι μόνο την ψυχαγωγία, αλλά και τον τρόπο με τον οποίο οι χρήστες αλληλοεπιδρούν με την τεχνολογία γενικότερα.

Από τα πρώτα στάδια των βιντεοπαιχνιδιών, όπου οι χρήστες είχαν στη διάθεσή τους περιορισμένα γραφικά και λειτουργίες, έχουμε φτάσει σε μια εποχή όπου η ρεαλιστικότητα στην αναπαράσταση των κόσμων σε έναν υπολογιστή, έχει φτάσει σε εντυπωσιακά επίπεδα. Με εκατομμύρια παίκτες παγκοσμίως που συνδέονται διαδικτυακά σε όλων των ειδών τα παιχνίδια, το gaming σε υπολογιστή αναδεικνύεται ως μια κοινότητα όπου η φαντασία συναντά την τεχνολογία, με την αξία της να είναι πλέον στα ύψη[3] και να διατηρεί μεγάλο μέρος της ευρύτερης αγοράς των βιντεοπαιχνιδιών[4].



ΕΙΚΟΝΑ 1 - ΤΟ MARKET VALUE ΤΟΥ ONLINE PC GAMING

Η τεχνολογία των βιντεοπαιχνιδιών έχει αναπτυχθεί σε τεράστιο βαθμό με αποτέλεσμα οι διαθέσιμες μηχανές παιχνιδιών να είναι πολλές. Πλέον, πολλές εταιρείες σχεδιάζουν και

[3] https://www.statista.com/statistics/292516/pc-online-game-market-value-worldwide/
[4] https://www.statista.com/statistics/292460/video-game-consumer-market-value-worldwide-platform/

αναπτύσσουν τις δικές τους μηχανές, επάνω στις οποίες δημιουργούν τα παιχνίδια. Οι βασικές μηχανές παιχνιδιών, που είναι διαθέσιμες και χρησιμοποιούνται από τους περισσότερους χρήστες είναι:

- Unity3D
- Unreal Engine

Για την παρούσα διπλωματική εργασία χρησιμοποιήθηκε το Unity, καθώς παρέχει δωρεάν πρόσβαση, ενώ η κοινότητα υποστήριξης όπως και το διαθέσιμο υλικό στο Asset Store, είναι αρκετά για την ανάπτυξη πλήρως λειτουργικών και μοντέρνων παιχνιδιών.

1.1 Περιγραφή του παιχνιδιού

Παρακάτω θα γίνει αναφορά στα βασικά χαρακτηριστικά του παιχνιδιού που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας και στις δυνατότητες που προσφέρει στους χρήστες.

Το συγκεκριμένο παιχνίδι ανήκει στην κατηγορία των Third Person Shooter, δηλαδή αποτελεί ένα παιχνίδι βολών τρίτου προσώπου, όπου ο παίκτης παρακολουθεί και χειρίζεται τον χαρακτήρα του από μια τρίτη γραμμική προοπτική, έχοντας την δυνατότητα να βλέπει και να στοχεύει τους αντίπαλούς του από μια γωνία θέασης που βρίσκεται πίσω και πάνω από τον χαρακτήρα του, προσφέροντας ταυτόχρονα ολόκληρη άποψη του περιβάλλοντος. Πρόκειται για ένα multiplayer παιχνίδι, ένα παιχνίδι δηλαδή πολλών παικτών, του οποίου οι διαδικτυακές λειτουργίες έχουν υλοποιηθεί βάσει των υπηρεσιών Netcode for GameObjects, που παρέχει η μηχανή παιχνιδιών Unity3D.

Ακόμη ένα βασικό χαρακτηριστικό στοιχείο του παιχνιδιού, είναι η δυνατότητα τροποποίησης της πίστας και του ευρύτερου χώρου, μέσω των καταστρέψιμων αντικειμένων που απαρτίζουν το χάρτη του παιχνιδιού. Οι παίκτες δηλαδή, μπορούν να καταστρέφουν πολλά από τα μέρη των κτηρίων ή αντικειμένων, αλληλοεπιδρώντας έτσι ενεργά με το περιβάλλον τους. Η αλληλεπίδραση με το περιβάλλον όχι μόνο προσφέρει στρατηγικές ευκαιρίες αλλά επίσης εμπλουτίζει την εμπειρία του παίκτη, καθιστώντας το παιχνίδι ακόμη πιο εθιστικό και επιβραβευτικό. Στόχος του παίκτη θα είναι η επικράτηση εις βάρος των αντιπάλων του, καθώς και η επιβίωση από τους κινδύνους που θα κληθεί να αντιμετωπίσει.

Εικονα 2 - Το τελικο παιχνιδι

1.2 Τεχνολογίες / πακέτα που έχουν χρησιμοποιηθεί

Για την υλοποίηση του παιχνιδιού, έχουν χρησιμοποιηθεί αρκετά πακέτα – βιβλιοθήκες. Βασικότερη όλων, είναι το Netcode For GameObjects, το οποίο αποτέλεσε την βάση όλων των απαραίτητων διαδικτυακών λειτουργιών, για την μετατροπή του παιχνιδιού σε multiplayer.

Για τον σχεδιασμό, την γραφική απεικόνιση και των φωτισμό των κτηρίων, αξιοποιήθηκαν οι δυνατότητες που παρέχονται από το Unity. Πιο συγκεκριμένα, χρησιμοποιήθηκε το πακέτο ProBuilder, αποτελεί ένα εργαλείο υψηλού επιπέδου για την σχεδίαση και μοντελοποίηση πιστών, αντικειμένων, κτηρίων.

Προκειμένου η πίστα να μεταβάλλεται, οι παίκτες θα μπορούν μέσω των κινήσεων τους να καταστρέφουν τα αντικείμενα τριγύρω τους, καθώς και διάφορα μέρη κτηρίων, αλλάζοντας κατά αυτό τον τρόπο την πορεία του παιχνιδιού. Αυτό επιτυγχάνεται μέσω του πακέτου RayFire, το οποίο παρέχει την δυνατότητα πραγματοποίησης δυναμικών αλλαγών στην μορφή των αντικειμένων και της πίστας.

Τέλος, για την ολοκλήρωση και πλήρωση του παιχνιδιού, έχουν χρησιμοποιηθεί ορισμένα assets που παρέχονται από το Asset Store του Unity, τα οποία και θα αναφέρονται στην πορεία της εργασίας.

2. Εισαγωγή στο Unity

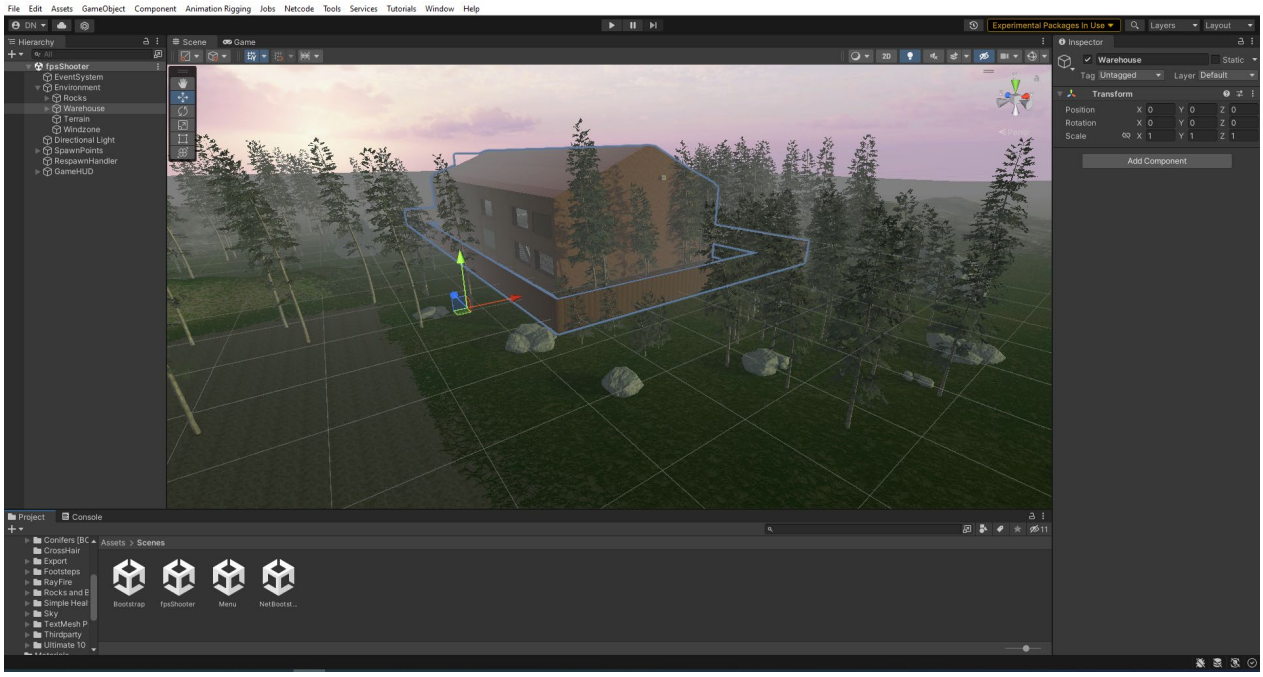
Η παρακάτω ενότητα αποτελεί εισαγωγή στο διαπλατφορμικό (cross platform) προγραμματιστικό περιβάλλον Unity. Τα θέματα τα οποία πραγματεύεται, είναι οι δυνατότητες που προσφέρει στο χρήστη, τα πλεονεκτήματα, τα μειονεκτήματά , ενώ γίνεται αναφορά και στο σύστημα συστατικών (components) το οποίο αξιοποιεί το Unity για την δημιουργία των αντικειμένων (objects). Σκοπός της παρακάτω ενότητας είναι η εξοικείωση του αναγνώστη με τα βασικά εργαλεία που παρέχονται καθώς και με τη διεπαφή (interface) του Unity, δημιουργώντας τελικά και τα πρώτα του αρχεία κώδικα (scripts) μέσω κάποιου ολοκληρωμένου περιβάλλοντος ανάπτυξης (IDE).

2.1 Επισκόπηση διεπαφής Unity

Η συγκεκριμένη ενότητα εστιάζει στην παρουσίαση της διεπαφής του Unity, αναλύοντας τα βασικότερα εργαλεία που παρέχονται στο χρήστη, καθένα από τα οποία συντελεί στην ανάπτυξη και ολοκλήρωση ενός παιχνιδιού.

Καθένα από τα παρακάτω εργαλεία, εντοπίζεται συνήθως σε μια ξεχωριστή καρτέλα ή παράθυρο, στο οποίο μπορεί ο χρήστης να ανατρέξει ώστε να έχει πρόσβαση σε περισσότερα εργαλεία της

αντίστοιχης κατηγορίας. Η διεπαφή του Unity είναι πλήρως παραμετροποιήσιμη, παρέχοντας την δυνατότητα μετακίνησης των καρτελών σε διαφορετικές θέσεις ή ξεχωριστά παράθυρα, ευνοώντας κατά αυτόν τον τρόπο την εκτέλεση πολλών ενεργειών ταυτοχρόνως.



ΕΙΚΟΝΑ 3 - Η ΔΙΕΠΑΦΗ ΤΟΥ UNITY

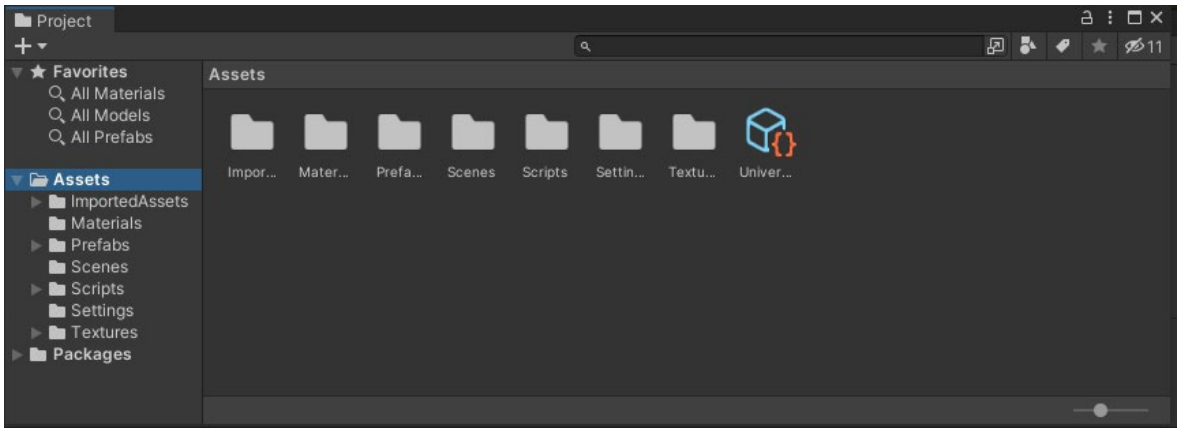
Παρακάτω θα γίνει αναφορά στις βασικότερες και πιο συχνά χρησιμοποιούμενες καρτέλες, παρουσιάζοντας ταυτόχρονα τα εργαλεία που η καθεμία παρέχει.

2.1.1 Project View

Η συγκεκριμένη καρτέλα εντοπίζεται συνήθως στο κάτω μέρος του Unity Editor και αποτελεί μια συλλογή όλων των αντικειμένων και γενικότερα των assets, τα οποία είτε έχουν δημιουργηθεί από το χρήστη είτε έχουν εισαχθεί στο project από κάποια εξωτερική πηγή, όπως το Unity Asset Store.

Ως asset ενός παιχνιδιού, ορίζεται κάθε αρχείο ή πόρος που χρησιμοποιείται στο παιχνίδι, και μπορεί να είναι εικόνες, ήχοι, μοντέλα 3D, κώδικας (scripts), animations (κινούμενα σχέδια) και άλλα. Είναι δηλαδή όλα εκείνα τα στοιχεία που συνθέτουν ένα ολοκληρωμένο παιχνίδι.

Όπως φαίνεται στην Εικόνα 4, παρέχεται η δυνατότητα ύπαρξης εμφωλευμένων φακέλων εντός των assets, τα οποία υποδηλώνονται με τα χαρακτηριστικά βέλη δίπλα από το όνομα του φακέλου. Ακόμη, η καρτέλα Project διαθέτει μπάρα αναζήτησης, μέσω της οποίας ο χρήστης εντοπίζει το επιθυμητό asset γρήγορα, χωρίς να ξοδεύει χρόνο αναζητώντας το, κάτι που μπορεί να γίνει αρκετά χρονοβόρο εξαιτίας του μεγάλου αριθμού αρχείων που απαιτούνται.



ΕΙΚΟΝΑ 4 - Η ΚΑΡΤΕΛΑ PROJECT

2.1.2 Hierarchy

Μια βασική καρτέλα του Editor είναι η καρτέλα Hierarchy. Όπως και στην καρτέλα Project, η συγκεκριμένη καρτέλα απεικονίζει αρχεία και αντικείμενα, με τη διαφορά ότι στην Hierarchy εμφανίζονται μονάχα τα assets που χρησιμοποιούνται στην τρέχουσα σκηνή.

Στο παράδειγμα της Εικόνα 5, η τρέχουσα σκηνή είναι η NetBoostrap και τα αντικείμενα που φαίνονται παρακάτω, εντοπίζονται μονάχα στη συγκεκριμένη σκηνή.

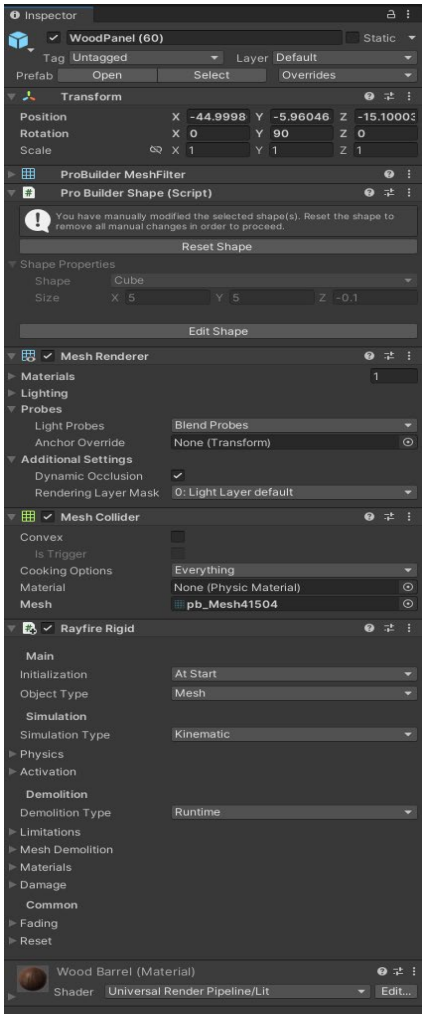
ΕΙΚΟΝΑ 5 - Η ΚΑΡΤΕΛΑ HIERARCHY

Μια πολύ σημαντική δυνατότητα του Hierarchy View, είναι η δυνατότητα ομαδοποίησης των αντικειμένων, μέσω δημιουργίας σχέσεων πατέρα – παιδιού, κάτι το οποίο διευκολύνει ιδιαίτερα την αναζήτηση ενός συγκεκριμένου αντικειμένου. Τα αντικείμενα – παιδιά, μπορούν να κληρονομούν τα χαρακτηριστικά και τις ιδιότητες από το αντικείμενο – πατέρα, καθιστώντας έτσι

πολύ ευκολότερη την τροποποίηση και επεξεργασία των αντικειμένων που μοιράζονται τα ίδια χαρακτηριστικά.

2.1.3 Inspector

Κύρια λειτουργία του παράθυρου Inspector είναι η δυνατότητα απεικόνισης μιας σειράς πληροφοριών για το επιλεχθέν αντικείμενο. Οι πληροφορίες αυτές, ομαδοποιούνται σε ξεχωριστά τμήματα, όπου κάθε τέτοιο τμήμα αποτελεί ένα component, όπως θα αναλυθεί στη συνέχεια. Συνήθως, κάθε αντικείμενο θα περιέχει πολλαπλά components για την υλοποίηση των λειτουργιών του, συμπεριλαμβανομένων και των αρχείων κώδικα (***scripts***). Κάθε αντικείμενο, θα περιέχει κατά τη δημιουργία του το component ***Transform***, το οποίο είναι υπεύθυνο για τη θέση, την περιστροφή και το μέγεθος του εκάστοτε αντικειμένου.



ΕΙΚΟΝΑ 6 - Η ΚΑΡΤΕΛΑ INSPECTOR

Στο παράδειγμα της Εικόνα 6, απεικονίζεται η καρτέλα Inspector για το αντικείμενο **WoodPanel (60)**, το οποίο πρόκειται για ένα φράκτη της πίστας στην οποία λαμβάνει χώρα το παιχνίδι. Τα αντικείμενα ProBuilderMeshFilter, Mesh Renderer, Mesh Collider, Rayfire Rigid αποτελούν

components τα οποία διαθέτει ο φράκτης, ενώ το Wood Barrel, αφορά το material του αντικειμένου, δηλαδή το αρχείο το οποίο είναι υπεύθυνο για το γραφιστικό μέρος του φράκτη.

Τα συνηθέστερα components που συναντά κανείς σε ένα αντικείμενο είναι το Transform, το Collider το οποίο δίνει φυσική υπόσταση στο αντικείμενο παρέχοντας ταυτόχρονα δυνατότητα ανίχνευσης συγκρούσεων και αλληλεπιδράσεων με αυτό, και το Renderer το οποίο είναι υπεύθυνο για την απεικόνιση του αντικειμένου στην οθόνη.

2.1.4 Toolbar

Η γραμμή εργαλείων αποτελείται από ορισμένα menu και από πέντε κατηγορίες εργαλείων που συμβάλλουν στον έλεγχο του παιχνιδιού. Όπως φαίνεται στην Εικόνα 7, τα menu τοποθετούνται στο επάνω μέρος του editor και το καθένα παρέχει μια πληθώρα εργαλείων, τα οποία ομαδοποιούνται βάσει των λειτουργιών τους.

ΕΙΚΟΝΑ 7 - Η ΓΡΑΜΜΗ ΕΡΓΑΛΕΙΩΝ ΤΟΥ UNITY

- **File:** Περιέχει όλα τα εργαλεία που είναι απαραίτητα για το άνοιγμα και την αποθήκευση σκηνών και projects, καθώς και οτιδήποτε απαιτείται για το build του παιχνιδιού.

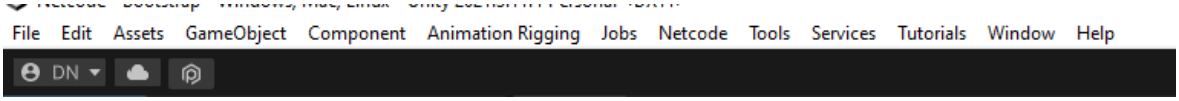
  Ως build, ορίζεται η διαδικασία δημιουργίας εκτελέσιμων αρχείων του παιχνιδιού, τα οποία μπορούν να εκτελεστούν εκτός του Unity Editor. Το αποτέλεσμα συνήθως του build είναι κάποιο αρχείο εφαρμογής (.exe), το οποίο διανέμεται στη συνέχεια στους παίκτες, προκειμένου να μπορούν να παίξουν το παιχνίδι. Η διαδικασία του build θα αναλυθεί περαιτέρω στις επόμενες ενότητες, καθώς το build που θα πραγματοποιηθεί είναι διαφορετικό στην περίπτωση των dedicated servers και των παικτών (clients).
- **Edit:** Τα βασικότερα εργαλεία του menu αυτού αφορούν την αντιγραφή, επικόλληση, διαγραφή αντικειμένων, καθώς και ορισμένες επιλογές που αφορούν τη δημιουργία σχέσεων πατέρα – παιδιού μεταξύ των αντικειμένων.
- **Assets:** Στο συγκεκριμένο menu βρίσκονται τα εργαλεία εκείνα που εστιάζουν στην δημιουργία, εισαγωγή και εξαγωγή νέων assets.
- **GameObject:** Στο menu αυτό βρίσκονται τα εργαλεία που αφορούν την δημιουργία, προβολή νέων αντικειμένων στην τρέχουσα σκηνή (game objects), όπως επίσης και

μερικές ακόμη επιλογές για τη δημιουργία σχέσεων πατέρα – παιδιού μεταξύ των αντικειμένων στη σκηνή.

- **Component:** Το menu αυτό εστιάζει στη δημιουργία νέων συστατικών στοιχείων (components) των διαφόρων αντικειμένων. Οι επιλογές που παρέχονται βασίζονται στα πακέτα που έχουν εισαχθεί στο παιχνίδι.
- **View:** Μέσω αυτού του menu, μπορεί ο χρήστης να ανοίξει, μεγιστοποιήσει ή ελαχιστοποιήσει τα διάφορα παράθυρα και καρτέλες που έχει στη διάθεση του, καθώς και να αναδιατάξει τη μορφή του editor. Μπορεί επίσης, να δημιουργήσει τα δικά του layouts, τα οποία και να εναλλάσσει άμεσα προς δική του διευκόλυνση.
- **Help:** Περιέχει όλα τα εργαλεία που αφορούν τα εγχειρίδια χρήσης, τα διαθέσιμα forums, μια πληθώρα από διαθέσιμα tutorials, καθώς και γενικές πληροφορίες για το Unity.

Τα παραπάνω αποτελούν τα βασικά menus τα οποία συναντά κανείς όταν χρησιμοποιεί το Unity Editor για πρώτη φορά. Σε όλα αυτά, μπορούν να προστεθούν και καινούρια, βάσει των πακέτων που έχουν εισαχθεί στο εκάστοτε παιχνίδι.

Στο παράδειγμα της Εικόνα 8, φαίνονται τα διάφορα menus που προστέθηκαν κατά την ανάπτυξη του παιχνιδιού που πραγματεύεται η παρούσα εργασία.



ΕΙΚΟΝΑ 8 - ΤΑ ΜΕΝΟ ΤΟΥ UNITY

Εκτός των βασικών menus, ο χρήστης συναντά και ορισμένες ακόμη ομάδες εργαλείων, όπως φαίνεται στην Εικόνα 7.

• **Transform tools**: Τα συγκεκριμένα εργαλεία χρησιμοποιούνται στην σκηνή του παιχνιδιού και είναι υπεύθυνα για το χειρισμό των αντικειμένων. Τα διαθέσιμα εργαλεία είναι το Hand tool, Translate tool, Rotate tool και Scale tool, για επιλογή, μετακίνηση, περιστροφή και κλιμάκωση των αντικειμένων αντίστοιχα.

• **Transform Gizmo toggles:** Οι συγκεκριμένοι διακόπτες αλλάζουν τον τρόπο με τον οποίο τα Transform εργαλεία λειτουργούν στην σκηνή του παιχνιδιού,

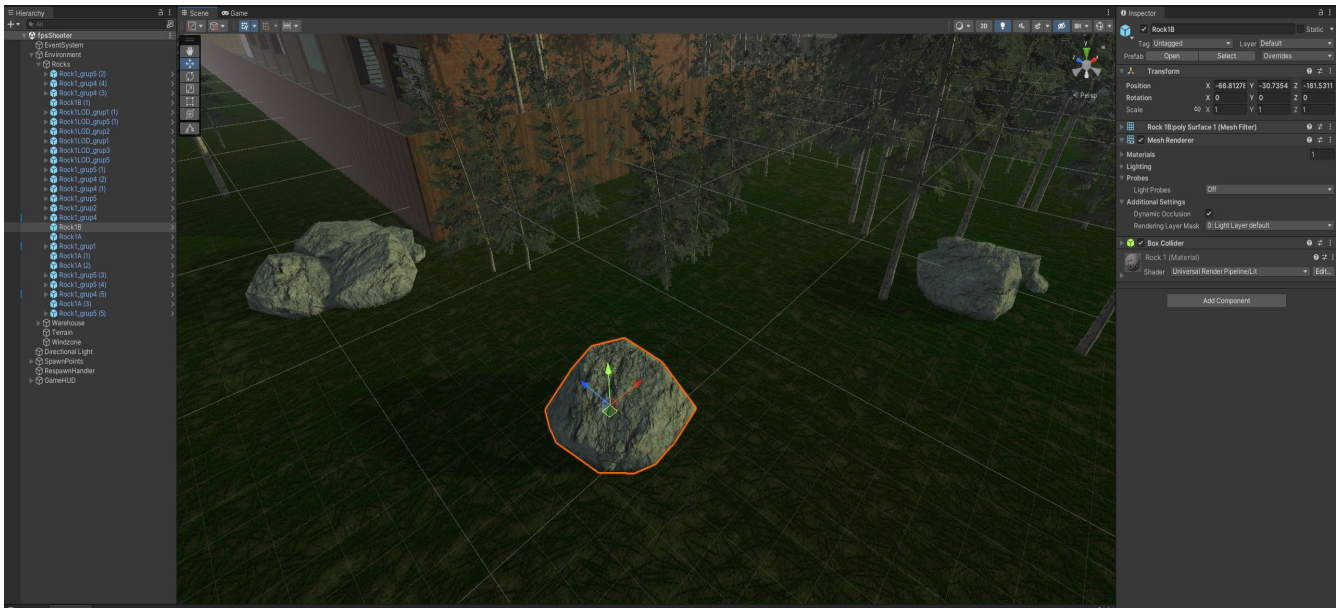
• **Play Controls group:** Με αυτά τα κουμπιά πραγματοποιείται αναπαραγωγή/παύση και ολική αναστολή ενός παιχνιδιού εντός του editor για δοκιμές,

•**Layers drop-down list:** Μέσω αυτής της λίστας παρέχεται χειρισμός για το ποια αντικείμενα θα εμφανίζονται στην σκηνή οποιαδήποτε χρονική στιγμή,

• **Layout drop-down list:** Μέσω αυτής της λίστας τροποποιείται η διαρρύθμιση των παραθύρων και των views. Επιπλέον, μπορεί και να αποθηκευτεί όποια προσαρμοσμένη διάταξη έχει δημιουργηθεί.

2.1.5 Scene View

Αποτελεί την καρτέλα με την οποία αλληλεπιδρά περισσότερο ο χρήστης κατά τη διάρκεια ανάπτυξης του παιχνιδιού. Μέσω της καρτέλας Scene, ο χρήστης έχει πρόσβαση στην οπτική αναπαράσταση του παιχνιδιού και μπορεί να χειριστεί, να επιλέξει, να επεξεργαστεί και να μετακινήσει όλα τα αντικείμενα της τρέχουσας σκηνής.



ΕΙΚΟΝΑ 9 - Η ΚΑΡΤΕΛΑ SCENE VIEW

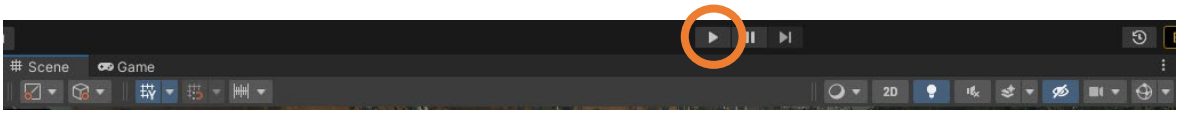
Όπως φαίνεται στην Εικόνα 9, τα αντικείμενα του Hierarchy απεικονίζονται οπτικά εντός του Scene. Ο χρήστης μπορεί να επιλέξει τα αντικείμενα είτε μέσω του Hierarchy είτε μέσω της τρέχουσας σκηνής, πατώντας απευθείας επάνω στο αντικείμενο που επιθυμεί. Έχοντας επιλέξει κάποιο αντικείμενο της σκηνής, ανανεώνεται αυτόματα και η καρτέλα Inspector, μέσω της οποίας μπορεί ο χρήστης να τροποποιήσει τα χαρακτηριστικά του αντικειμένου. Το Unity Editor παρέχει μια πληθώρα εργαλείων και συντομεύσεων για γρήγορο πλοήγηση εντός των σκηνών, κάτι που μπορεί να διευκολύνει τον προγραμματιστή σε μεγάλο βαθμό.

2.1.6 Game View

Δίπλα συνήθως από το Scene View βρίσκεται το Game View. Η καρτέλα Game επιτρέπει στο χρήστη να τρέχει και να δοκιμάζει το παιχνίδι του, χωρίς να χρειάζεται κάθε φορά να ακολουθεί τη διαδικασία του build. Η παραπάνω λειτουργία μπορεί να φαίνεται λιγότερο σημαντική, ωστόσο στην πραγματικότητα η δυνατότητα αυτή είναι ιδιαίτερα σημαντική και διευκολύνει αρκετά τον προγραμματιστή, καθώς πλέον υπάρχει η δυνατότητα εναλλαγών και δοκιμών χωρίς την ανάγκη περιττών διακοπών.

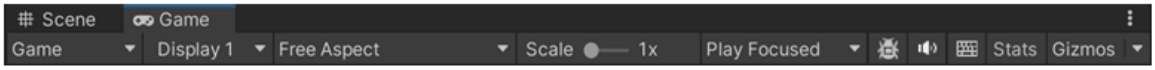
Ο χρήστης μπορεί παράλληλα, μέσω του Scene View, να τροποποιεί στοιχεία του παιχνιδιού και αμέσως να ελέγχει την επίδραση τους στο παιχνίδι, επιστρέφοντας πίσω στην καρτέλα Game. Για παράδειγμα, αν κατά τη διάρκεια των δοκιμών παρατηρηθεί πως κάποιο χαρακτηριστικό του παίκτη χρήζει τροποποίησης (π.χ. ταχύτητα κίνησης παίκτη), τότε αυτό μπορεί άμεσα να μεταβληθεί και να δοκιμαστεί εκ νέου, χωρίς να απαιτείται η έξοδος και η επαναλαμβανόμενη εκκίνηση του παιχνιδιού.

Για να δοκιμαστεί το παιχνίδι, αρκεί ο χρήστης να πατήσει το κουμπί εκκίνησης, όπως φαίνεται στην Εικόνα 10. Μόλις ολοκληρωθεί η φόρτωση των απαιτούμενων στοιχείων, αυτομάτως θα εναλλαχθεί η σκηνή, και ο χρήστης πλέον θα βλέπει την καρτέλα Game.



Εικονα 10 - Το κουμπι εκκινησης του παιχνιδιου

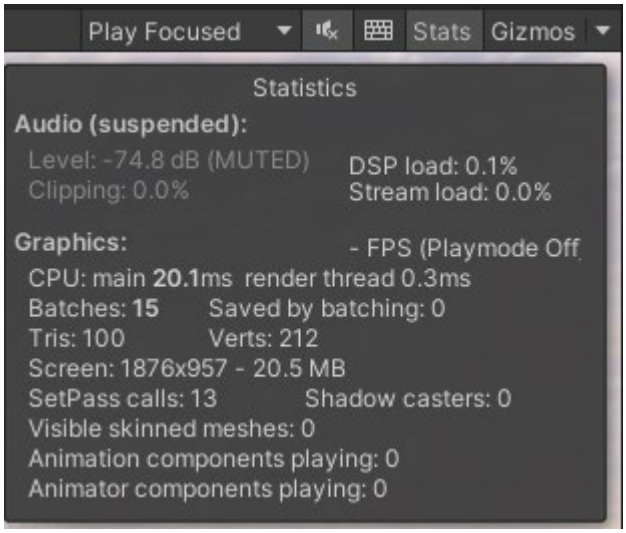
Όπως και οι άλλες καρτέλες, έτσι και η Game καρτέλα έχει τη δική της γραμμή εργαλείων, όπως φαίνεται στην Εικόνα 11.



Εικονα 11 - Η γραμμη εργαλειων της καρτελας Game view

- **Game/Simulator:** Η πρώτη λίστα έχει ως διαθέσιμες τιμές τις Game και Simulator. Η Game είναι η προκαθορισμένη προβολή του παιχνιδιού για υπολογιστή, ενώ η επιλογή Simulator εστιάζει στα παιχνίδια για κινητά, οπότε και παρέχει τη δυνατότητα προβολής του παιχνιδιού σε έναν προσομοιωτή (device simulator).
- **Display:** Στην περίπτωση που υπάρχουν αρκετές κάμερες σε μια σκηνή, ο χρήστης μπορεί από αυτή τη λίστα να πλοηγηθεί μεταξύ τους. Στο συγκεκριμένο παιχνίδι, υπάρχει μονάχα μια διαθέσιμη επιλογή, καθώς η μοναδική κάμερα που υπάρχει είναι αυτή που βρίσκεται πάνω από το χαρακτήρα.

- **Aspect drop-down list:** Μέσω αυτής της λίστας, μπορεί να δοκιμαστεί το παιχνίδι σε διαφορετικές αναλογίες διαστάσεων προβολής, επιτρέποντας έτσι την προσομοίωση του τρόπου με τον οποίο θα φαινόταν το παιχνίδι σε διαφορετικές οθόνες.
- **Scale slider:** Επιτρέπει τη δυνατότητα μεγέθυνσης (zoom), ώστε να είναι εφικτή η σε βάθος εξέταση ορισμένων περιοχών του παιχνιδιού.
- **Play Mode Behavior:** Παρέχει ορισμένες ρυθμίσεις που αφορούν τον τρόπο με τον οποίο θα φαίνεται το παιχνίδι στο Unity Editor και ειδικότερα όταν υπάρχουν περισσότερες από μία ενεργές Game καρτέλες.
- **Mute audio:** Ενεργοποιεί ή απενεργοποιεί τον ήχο του παιχνιδιού κατά τη διάρκεια δοκιμών.
- **Stats toggle:** Εμφανίζει μια πληθώρα στατιστικών, που αφορούν κυρίως τα γραφικά του παιχνιδιού. Είναι ιδιαίτερα χρήσιμα κατά τη διάρκεια βελτιστοποιήσης της απόδοσης του παιχνιδιού.



ΕΙΚΟΝΑ 12 - ΤΟ ΠΑΡΑΘΥΡΟ ΤΩΝ ΣΤΑΤΙΣΤΙΚΩΝ

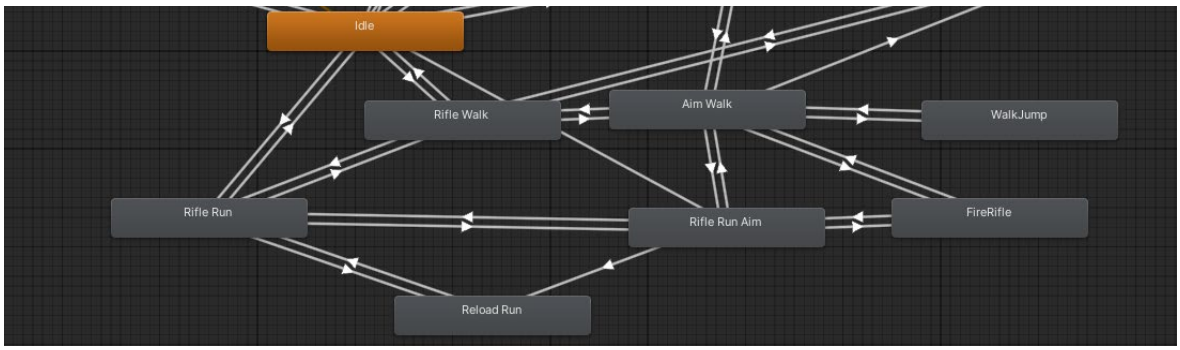
- **Gizmos:** Παρέχει μια σειρά ρυθμίσεων που αφορούν την εμφάνιση των Gizmos, δηλαδή των γραφικών στοιχείων που χρησιμοποιούνται κατά τη διάρκεια ανάπτυξης του παιχνιδιού, όπως είναι επίδειξη της θέσης των αντικειμένων, η εμφάνιση τροχιών κίνησης, η επίδειξη όγκων σύγκρουσης, και άλλες χρήσιμες πληροφορίες. Τα Gizmos είναι συνήθως χρήσιμα κατά τη διάρκεια της ανάπτυξης, καθώς προσφέρουν επιπρόσθετες οπτικές καθοδηγήσεις κατά την επεξεργασία της σκηνής.

2.1.7 Animation Views

Ένα από τα σημαντικότερα χαρακτηριστικά του Unity, είναι η δυνατότητα προβολής animations, κινούμενων σχεδίων δηλαδή, καθώς και η δυνατότητα σύνδεσης και εναλλαγής τους, για την δημιουργία πολύπλοκων αλληλουχιών κινήσεων. Το στοιχείο αυτό είναι απαραίτητο στα περισσότερα παιχνίδια, όπου οι χαρακτήρες αλληλεπιδρούν με το περιβάλλον τους ενώ ταυτόχρονα προβαίνουν σε διαφορετικές κινήσεις.

Στο παιχνίδι που αναπτύχθηκε στα πλαίσια της τρέχουσας διπλωματική εργασίας, ο χαρακτήρας διαθέτει μια πληθώρα από animations, που αφορούν το περπάτημα, το τρέξιμο, την χρήση του όπλου, την αλλαγή γεμιστήρα στο όπλο κ.ά.

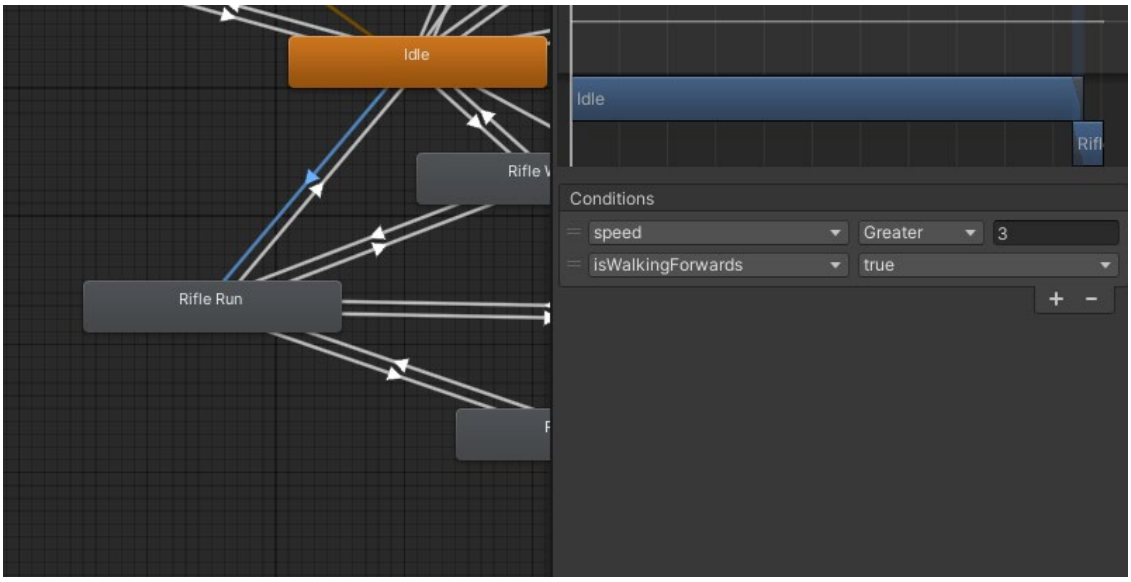
Στο παράδειγμα της Εικόνα 13 φαίνεται ένα μέρος των animations του χαρακτήρα που θα έχουν στη διάθεση τους οι χρήστες του συγκεκριμένου παιχνιδιού.



ΕΙΚΟΝΑ 13 - ΤΑ ANIMATIONS ΤΟΥ ΠΑΙΧΝΙΔΙΟΥ

Κάθε ορθογώνιο αποτελεί μια ξεχωριστή κίνηση του χαρακτήρα, ενώ τα βέλη μεταξύ των animations αποτελούν τις εναλλαγές μεταξύ τους. Κάθε τέτοιο βέλος, αποτελεί ουσιαστικά μια συνθήκη που ελέγχεται διαρκώς, ώστε αν οι απαραίτητες προϋποθέσεις πληρούνται, ο χαρακτήρας να προβεί σε κάποια διαφορετική κίνηση ή να συνεχίσει να εκτελεί την τρέχουσα.

Στην Εικόνα 14 φαίνεται η παραπάνω διαδικασία. Για παράδειγμα, για να εφαρμοστεί το animation με την ονομασία **Rifle Run** την στιγμή που ο χαρακτήρας βρίσκεται στην κατάσταση **Idle**, χρειάζεται να πληρούνται και οι 2 προϋποθέσεις που φαίνονται, δηλαδή η μεταβλητή **speed** να είναι μεγαλύτερη της τιμής 3 και η Boolean μεταβλητή **isWalkingForwards** να είναι αληθής.



ΕΙΚΟΝΑ 14 - ΣΥΝΘΗΚΗ ΕΝΑΛΛΑΓΗΣ ΜΕΤΑΞΥ ΤΩΝ ANIMATIONS

Οι μεταβλητές που προαναφέρθηκαν, ορίζονται και μεταβάλλονται μέσω του κώδικα του παιχνιδιού, όπως φαίνεται στο παρακάτω τμήμα κώδικα (Κώδικας 1).

```
public class ThirdPersonController : NetworkBehaviour
{
...
private bool IsSprinting =>  Input.GetKey(sprintKey);
private bool IsWalkingForwards => Input.GetKey(KeyCode.W);
...
[SerializeField] private float walkSpeed = 1.5f;
[SerializeField] private float sprintSpeed = 6.0f;
...
[SerializeField] private KeyCode sprintKey = KeyCode.LeftShift;
...
private void HandleMovementInput() {
        ...
        if (IsWalkingForwards && !IsSprinting) {
            animator.SetFloat("speed", walkSpeed);
            animator.SetBool("isWalkingForwards", true);
        } else if (IsSprinting && IsWalkingForwards) {
            animator.SetFloat("speed", sprintSpeed);
            animator.SetBool("isWalkingForwards", true);
        }
        ...
    }
}
```

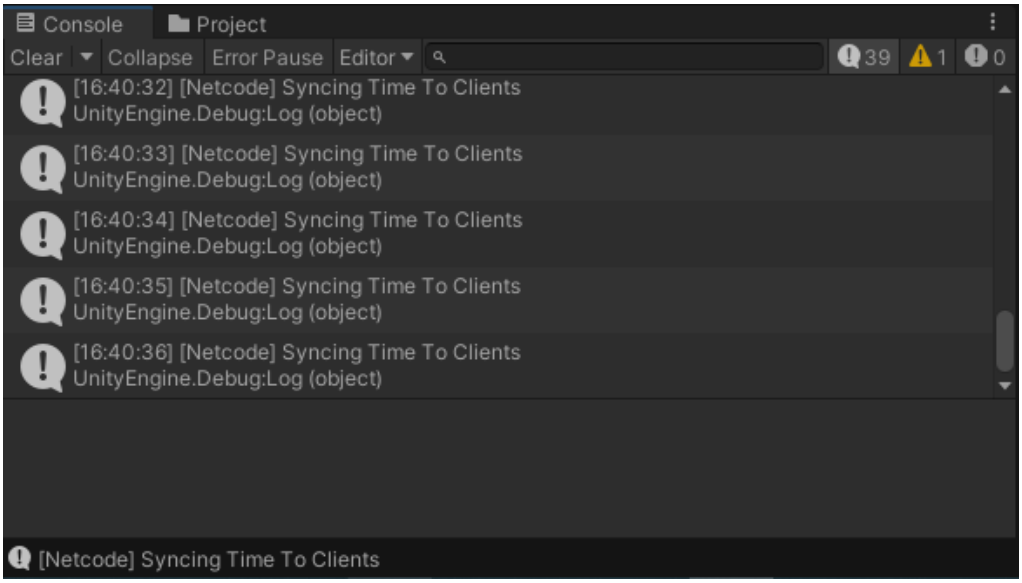
ΚΩΔΙΚΑΣ 1 - Η ΕΝΑΛΛΑΓΗ ΤΩΝ ANIMATIONS ΤΟΥ ΠΑΙΚΤΗ

Εδώ ανάλογα την κατάσταση στην οποία βρίσκεται ο χαρακτήρας, αν δηλαδή στο συγκεκριμένο παράδειγμα τρέχει ή περπατά, τίθεται και η αντίστοιχη τιμή στις απαιτούμενες μεταβλητές, προκειμένου στη συνέχεια ο παίκτης να εκτελέσει την κατάλληλη κίνηση εντός του παιχνιδιού. Με

όμοιο τρόπο έχουν σχεδιαστεί όλες οι μεταβάσεις μεταξύ των διαφόρων animations του χαρακτήρα.

2.1.8 Console / Status Bar

Στο κάτω μέρος συνήθως του Editor, συναντά κανείς μια ακόμη καρτέλα, με ονομασία Console Tab, και μία μπάρα με όνομα Status Bar.



ΕΙΚΟΝΑ 15 - Η ΚΑΡΤΕΛΑ CONSOLE

Πρόκειται για δύο πολύ χρήσιμα εργαλεία τα οποία εστιάζουν στον εντοπισμό και αντιμετώπιση σφαλμάτων κατά τη διάρκεια ανάπτυξης της εφαρμογής.

Εδώ εμφανίζονται όλα τα μηνύματα που αφορούν τον κώδικα του παιχνιδιού και μπορούν να αποσκοπούν στην αποσφαλμάτωση του παιχνιδιού, έχοντας προσαρμοστεί από τον ίδιο τον προγραμματιστή, είτε να είναι μηνύματα λάθους (errors) και να αφορούν λάθη στα αρχεία κώδικα (scripts).

Παρέχει επίσης πρόσβαση σε στοιχεία που αφορούν πληροφορίες εκτέλεσης της εφαρμογής και αρχεία καταγραφών (logs) του παιχνιδιού.

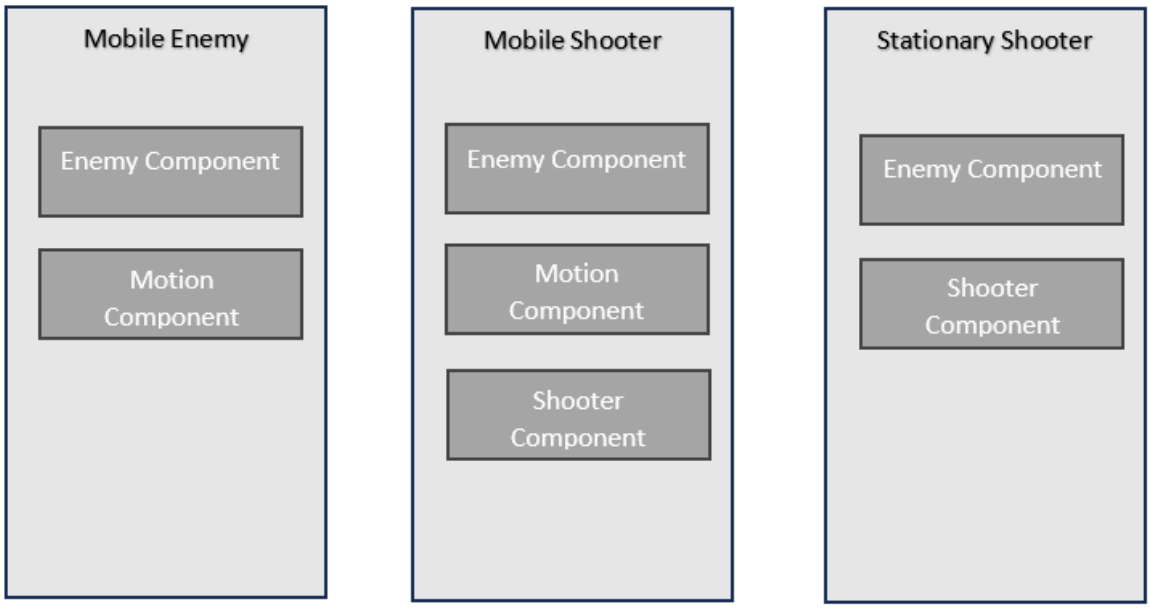
2.2 Πλεονεκτήματα Unity

Η διεπαφή του Unity επιτρέπει την διασύνδεση μεταξύ του εικαστικού και του προγραμματιστικού μέρους, έχοντας ως αποτέλεσμα τη δημιουργία διαδραστικών αντικειμένων. Αυτό συμβάλλει στην δημιουργία υψηλής ποιότητας παιχνιδιών, γρήγορα και αποδοτικά, κάνοντας τον προγραμματιστή ιδιαίτερα παραγωγικό, αξιοποιώντας ταυτόχρονα τις πιο

πρόσφατες τεχνολογίες παιχνιδιών. Ακόμα ένα μεγάλο πλεονέκτημα του Unity είναι η δυνατότητα των πολλαπλών και ταχύτατων επαναλήψεων, επιτρέποντας έτσι στον προγραμματιστή να τρέχει το παιχνίδι του μέσα από πολλούς και συνεχόμενους κύκλους επαναλήψεων, παρατηρώντας τις αλλαγές που επιφέρουν οι τροποποιήσεις του κώδικα ή των σκηνών. Παρέχει ακόμη τη δυνατότητα επεξεργασίας του παιχνιδιού καθώς το ίδιο το παιχνίδι τρέχει. Για παράδειγμα, μπορεί ο χρήστης να μετακινήσει τα αντικείμενα της σκηνής, και να παρατηρήσει σε πραγματικό χρόνο τις αλλαγές που αυτό θα επιφέρει στην πορεία του παιχνιδιού. Όπως αναφέρθηκε και στην εισαγωγή, το Unity είναι ένα διαπλατφορμικό περιβάλλον. Αυτό ωστόσο ισχύει τόσο στο κομμάτι του κοινού, όπου ένα παιχνίδι μπορεί για παράδειγμα να αναπτυχθεί για υπολογιστή, για κινητά και για κονσόλες, όσο και στο κομμάτι των εργαλείων ανάπτυξης, όπου ο προγραμματιστής μπορεί να αναπτύξει το παιχνίδι του σε Windows ή MacOS, ενώ πλέον παρέχει τη δυνατότητα ανάπτυξης εφαρμογών WebGL, VR, AR, Oculus και VIVE.

2.2.1 Component System

Ακόμα ένα σημαντικό εργαλείο του Unity είναι το σύστημα συστατικών που αναφέρθηκε στην εισαγωγή. Πρόκειται ουσιαστικά για ένα μοντέλο το οποίο αξιοποιεί μια από τις βασικότερες αρχές του αντικειμενοστραφούς προγραμματισμού, την σύνθεση. Εδώ, κάθε συστατικό (component), αποτελεί ένα πακέτο ξεχωριστών λειτουργιών. Η σύνθεση διαφόρων τέτοιων συστατικών, δημιουργεί ένα αντικείμενο. Πολλά αντικείμενα τα οποία στην συνέχεια θα αλληλοεπιδρούν, οδηγούν στο επιθυμητό αποτέλεσμα, το παιχνίδι.



Εικονα 16 - Το Component System του Unity

2.3 Μειονεκτήματα Unity

Σε περιπτώσεις παιχνιδιών όπου υπάρχουν σκηνές με πολλαπλά αντικείμενα που αλληλοεπιδρούν, είναι πιθανό ο προγραμματιστής να μην γνωρίζει ποια αντικείμενα (objects) έχουν συγκεκριμένα συστατικά, και έτσι να χρειαστεί να τα εντοπίσει, κάτι το οποίο μπορεί να γίνει αρκετά περίπλοκο σε ορισμένες περιπτώσεις. Η πληθώρα επιλογών που παρέχεται καθώς και η δυνατότητα υλοποίησης μιας λειτουργίας με διάφορους τρόπους, μπορεί ορισμένες φορές φορές να προκαλέσει σύγχυση τον προγραμματιστή, μέχρις ότου να ξεκαθαρίσει τις επιλογές του και να αποφασίσει ποιο μονοπάτι θα ακολουθήσει. Για παράδειγμα, το Unity παρέχει 3 ξεχωριστούς τρόπους ανάπτυξης διεπαφής (User Interface), κάτι που σημαίνει πως ο χρήστης θα πρέπει να τους εξετάσει όλους και να επιλέξει τον ιδανικότερο για αυτόν.

3. Υλοποίηση του παιχνιδιού

Στη συγκεκριμένη ενότητα θα γίνει εκτενής αναφορά στα σημαντικότερα στοιχεία που απαρτίζουν και ολοκληρώνουν το παιχνίδι το οποίο αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας. Ειδικότερα, θα περιγραφεί ο τρόπος δημιουργίας των παρακάτω:

- Πίστα,
- Καταστρέψιμα κτήρια και αντικείμενα,
- Βολές,
- Χαρακτήρας,
- Διεπαφή του χρήστη (UI),
- Δυνατότητα σύνδεσης πολλαπλών παικτών (multiplayer).

3.1 Δημιουργία της πίστας

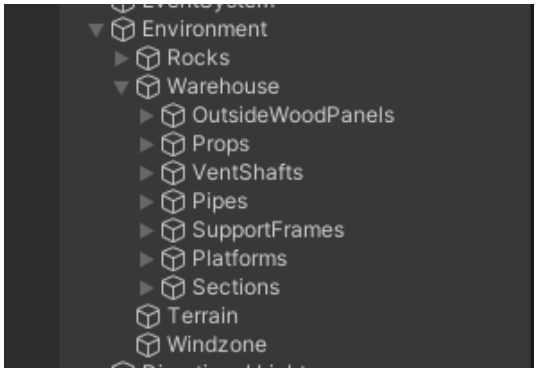
Ένα από τα βασικότερα κομμάτια της ανάπτυξης ενός παιχνιδιού, είναι η σχεδίαση της πίστας, του κόσμου δηλαδή τον οποίο θα βλέπουν και θα αλληλεπιδρούν οι χρήστες του παιχνιδιού. Η σχεδίαση της πίστας είναι ένα σημαντικό στοιχείο που επηρεάζει πολλές πτυχές του παιχνιδιού και συνεισφέρει στην καλύτερη απόλαυση του παίκτη.

Για το συγκεκριμένο παιχνίδι, η σχεδίαση της πίστας έγινε με μέριμνα την εμπειρία του χρήστη, την ανάγκη εφαρμογής στρατηγικής αλλά και την ύπαρξη ρεαλισμού στα στοιχεία του παιχνιδιού. Για τους παραπάνω λόγους, το μεγαλύτερο μέρος της πίστας αποτελείται από λόφους, δέντρα, βράχους και κτήρια, όπως φαίνεται στην Εικόνα *3*.

Ένα σημαντικό κομμάτι της ανάπτυξης ενός παιχνιδιού, είναι η επιλογή των assets, των αρχείων και των αντικειμένων δηλαδή που θα απαρτίζουν το παιχνίδι. Φυσικά, ο προγραμματιστής έχει τη

δυνατότητα να δημιουργήσει – σχεδιάσει τα δικά του αντικείμενα, όπως και έγινε στην παρούσα εργασία για τη σχεδίαση των κτηρίων. Για τα υπόλοιπα αντικείμενα του παιχνιδιού, χρησιμοποιήθηκαν αντικείμενα που παρέχονται μέσω του Unity Asset Store είτε δωρεάν είτε επί πληρωμή, χωρίς να απαιτείται η χρήση επιπρόσθετων σχεδιαστικών εργαλείων.

Μόλις ο χρήστης επιλέξει τα επιθυμητά assets, χρειάζεται απλώς να τα εισάγει (import) στο project του και στη συνέχεια να τα σύρει εντός του Hierarchy. Εκεί, όπως περιεγράφηκε νωρίτερα, μπορεί να δημιουργεί σχέσεις πατέρα παιδιού μεταξύ των αντικειμένων που μοιράζονται συγκεκριμένα χαρακτηριστικά, προκειμένου να τα ομαδοποιεί προς διευκόλυνση του στη συνέχεια, όπως φαίνεται στο παράδειγμα της Εικόνα 17.



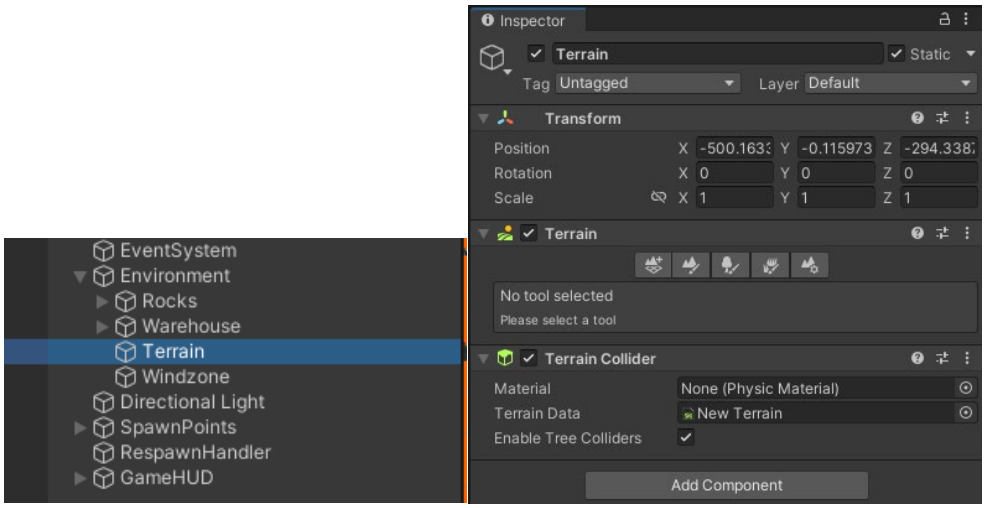
ΕΙΚΟΝΑ 17 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ ΠΟΥ ΣΥΝΘΕΤΟΥΝ ΤΗΝ ΠΙΣΤΑ

3.1.1 Σχεδίαση περιβάλλοντος

Ειδικότερα, τα assets που χρησιμοποιήθηκαν για τη σχεδίαση της πίστας, αφορούν τους βράχους και τα δέντρα και αποκτήθηκαν μέσω του Unity Asset Store[5] [6].

Για τη σχεδίαση του εδάφους (terrain), το Unity παρέχει ένα εργαλείο δημιουργίας και επεξεργασίας terrains. Συγκεκριμένα, επιλέγοντας κανείς GameObject -> 3D Object -> Terrain, δημιουργείται ένα αντικείμενο τύπου Terrain, το οποίο έχει τα χαρακτηριστικά της Εικόνα 18.

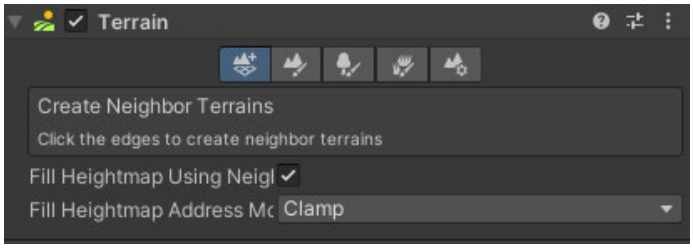
[5] https://assetstore.unity.com/packages/3d/props/exterior/rock-and-boulders-2-6947
[6] https://assetstore.unity.com/packages/3d/vegetation/trees/conifers-botd-142076



Εικονα 18 - Το αντικειμενο Terrain

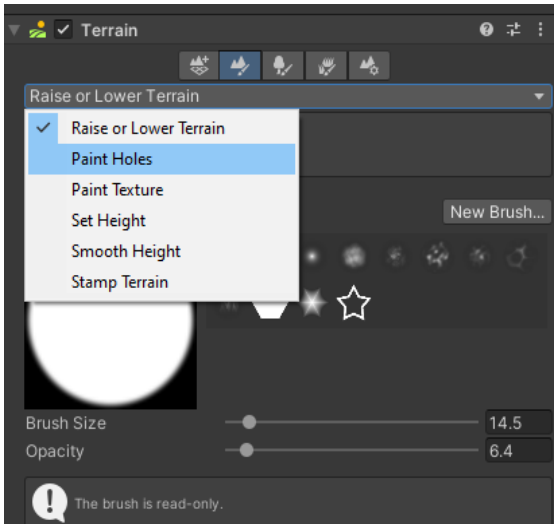
Όπως φαίνεται στην Εικόνα 18, στο component Terrain, παρέχονται ορισμένα εργαλεία για την σχεδίαση και μορφοποίηση του εδάφους. Συγκεκριμένα:

- **Δημιουργία γειτονικών Εδαφών**



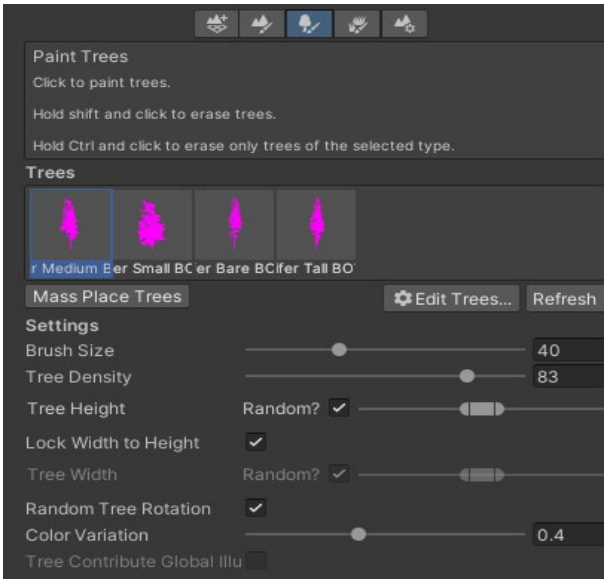
Εικονα 19 - Εργαλειο σχεδιασης γειτονικων terrain

- **Χάραξη και βαφή εδάφους (λόφοι, κοιλάδες κτλ.)**



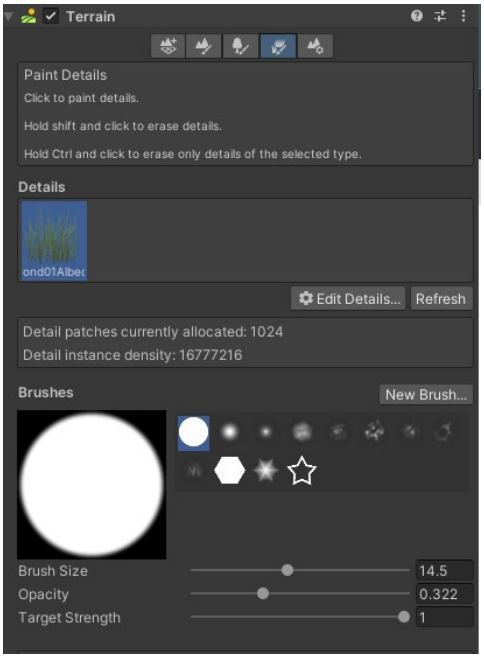
Εικονα 20 - Εργαλειο μορφοποιησης terrain

- **Προσθήκη δέντρων**



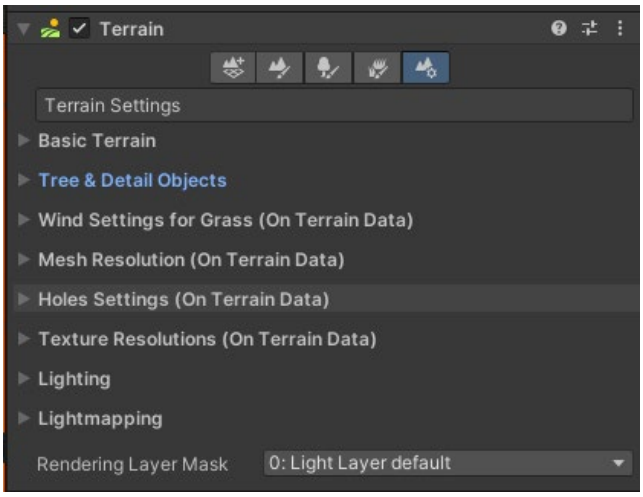
ΕΙΚΟΝΑ 21 - ΕΡΓΑΛΕΙΟ ΠΡΟΣΘΗΚΗΣ ΔΕΝΤΡΩΝ

- **Προσθήκη αντικειμένων (βλάστηση, γρασίδι κτλ.)**



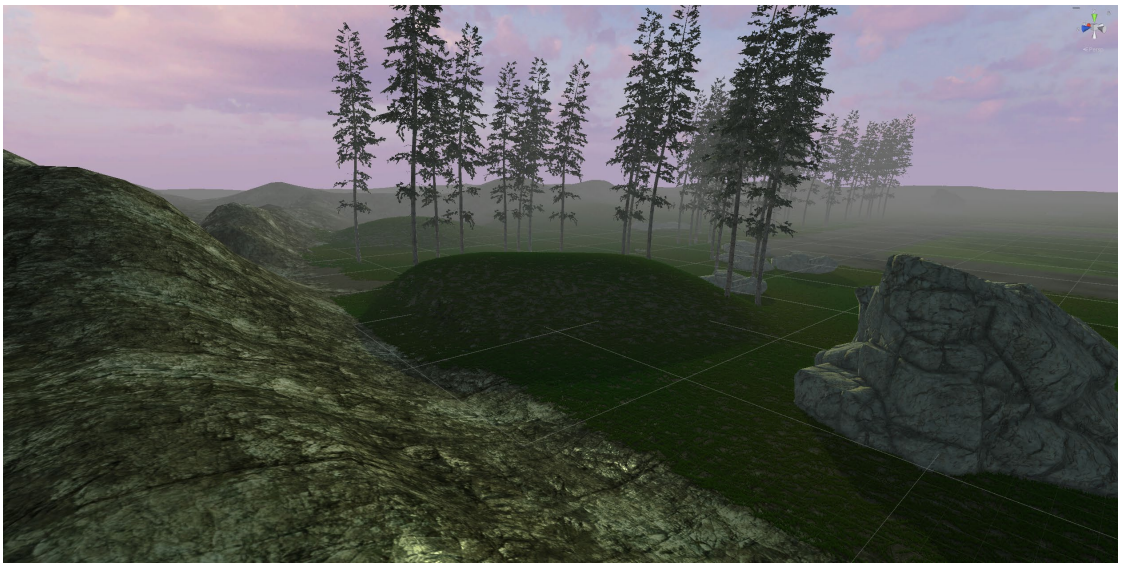
ΕΙΚΟΝΑ 22 - ΕΡΓΑΛΕΙΟ ΠΡΟΣΘΗΚΗΣ ΑΝΤΙΚΕΙΜΕΝΩΝ ΣΕ TERRAIN

- **Γενικές ρυθμίσεις για το τρέχον επιλεγμένο Έδαφος**



Εικονα 23 - Γενικεσ ρυθμισεισ των terrain

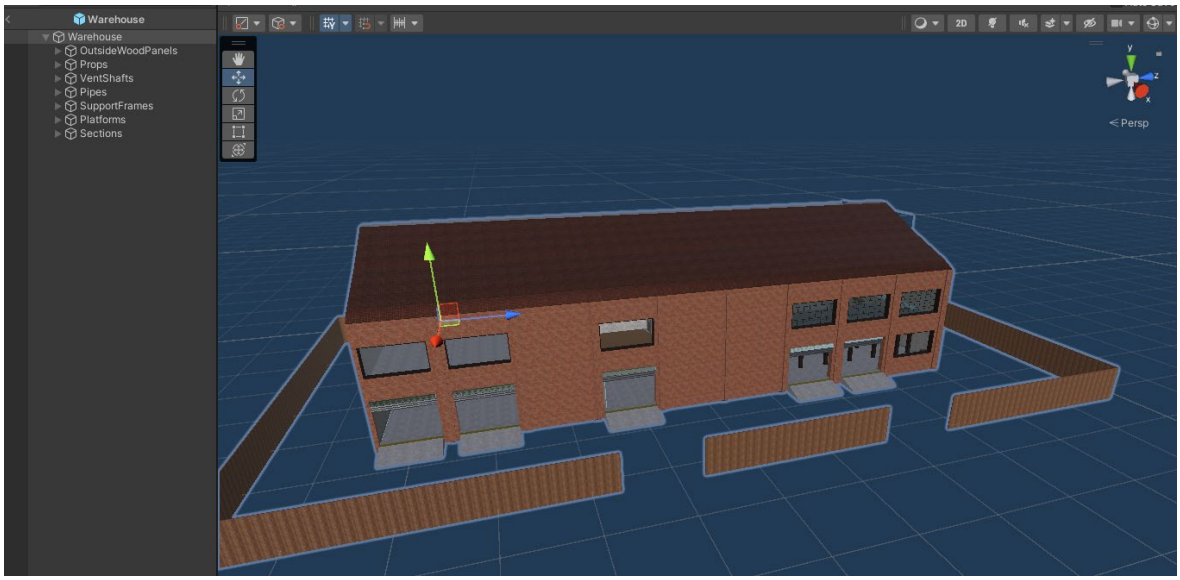
Στην Εικόνα 24, φαίνεται ένα μέρος της πίστας προτού τοποθετηθούν τα κτήρια και αποτελεί ουσιαστικά το αποτέλεσμα της αξιοποίησης όλων των παραπάνω εργαλείων, περιέχοντας όλα όσα προαναφέρθηκαν.



Εικονα 24 - Το περιβαλλον της πιστασ

3.1.2 Δημιουργία κτηρίων και αντικειμένων

Το παιχνίδι που αναπτύχθηκε, εκτός του φυσικού περιβάλλοντος από το οποίο αποτελείται και περιεγράφηκε στην παραπάνω ενότητα, θα περιλαμβάνει μια σειρά από κτήρια και αντικείμενα, με τα οποία οι παίκτες θα μπορούν να αλληλεπιδρούν. Συγκεκριμένα, στα πλαίσια του παιχνιδιού, σχεδιάστηκε και δημιουργήθηκε ένα κτήριο ως αποθήκη εργοστασίου, μαζί με επιπρόσθετους χώρους – δωμάτια, τα οποία προστέθηκαν στο τελικό αποτέλεσμα. Παράλληλα, για λόγους εμπλουτισμού του παιχνιδιού, σχεδιάστηκε μια σειρά αντικειμένων τα οποία συνάδουν με το ευρύτερο περιβάλλον και τα οποία κάνουν την εμπειρία του χρήστη ακόμα πιο συναρπαστική και ρεαλιστική.

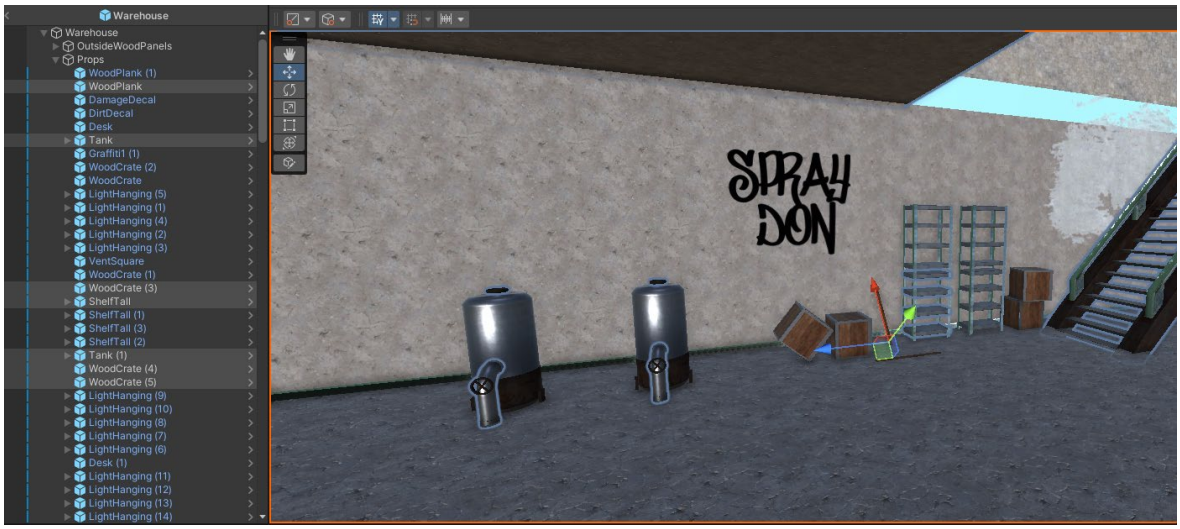


ΕΙΚΟΝΑ 25 - ΤΟ ΚΤΗΡΙΟ ΠΟΥ ΔΗΜΙΟΥΡΓΗΘΗΚΕ

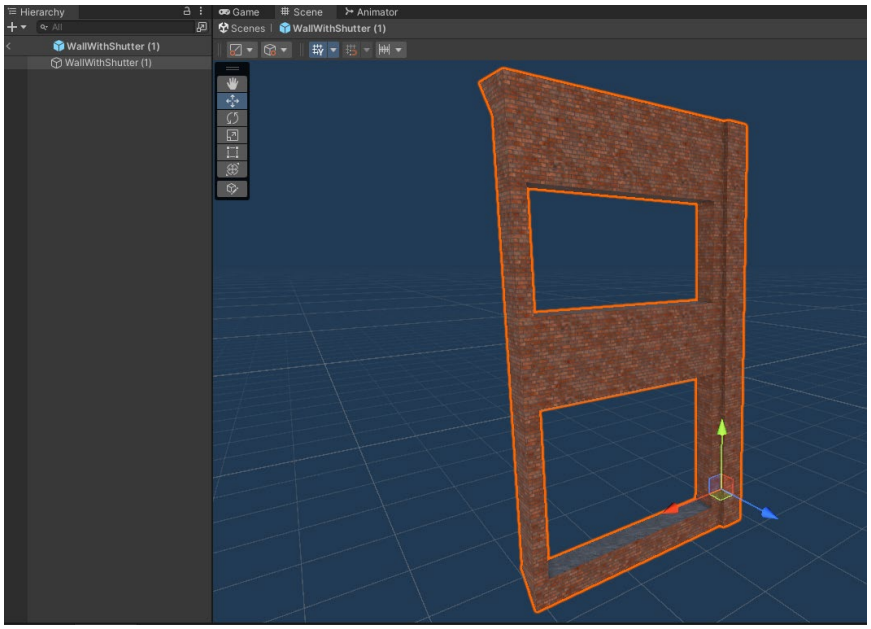
Στο σημείο αυτό, χρειάζεται να τονιστεί, πως όλα τα μέρη των κτηρίων και τα αντικείμενα, σχεδιάστηκαν και υλοποιήθηκαν με γνώμονα την δυνατότητα επαναχρησιμοποίησής τους. Μπορεί δηλαδή κανείς να αξιοποιήσει τα παραπάνω μέρη, και να δημιουργήσει πολύπλοκους συνδυασμούς κτηρίων και αντικειμένων. Όπως φαίνεται στην Εικόνα 26, αναπτύχθηκαν περισσότερα από 60 αντικείμενα και μέρη κτηρίων, τα οποία μπορούν να τοποθετηθούν σε όποιο σημείο επιθυμεί ο προγραμματιστής.

ΕΙΚΟΝΑ 26 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ ΠΟΥ ΣΧΕΔΙΑΣΤΗΚΑΝ

Στην Εικόνα 27, φαίνεται ένα μέρος των αντικειμένων τοποθετημένων εντός της αποθήκης, ενώ η Εικόνα 28 περιέχει ένα μέρος τοίχου, το οποίο όπως περιεγράφηκε νωρίτερα, μπορεί να επαναχρησιμοποιηθεί και να συνδεθεί με όλα τα υπόλοιπα μέρη του κτηρίου.



Εικονα 27 - Παραδειγμα τοποθετησης αντικειμενων εντος της αποθηκης



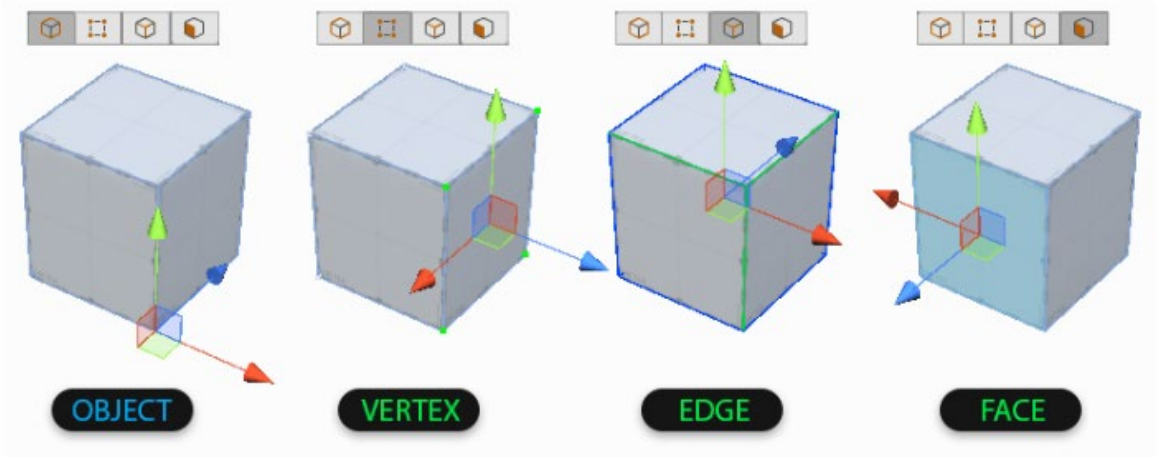
Εικονα 28 - Επαναχρησιμοποιουμενο μερος αποθηκης

Η σχεδίαση και η δημιουργία όλων των κτηρίων και των επιμέρους τμημάτων τους, καθώς και των αντικειμένων που απαρτίζουν την πίστα, βασίστηκε στο εργαλείο ProBuilder. Το ProBuilder αποτελεί ένα ενσωματωμένο εργαλείο του Unity, που επιτρέπει στον προγραμματιστή να δημιουργεί και να επεξεργάζεται 3D μοντέλα εντός του Unity, χωρίς την ανάγκη χρησιμοποίησης εξωτερικών λογισμικών. Το ProBuilder προσφέρει ενσωματωμένα εργαλεία μοντελοποίησης όπως για παράδειγμα κατασκευή πολυγώνων και άλλων σχημάτων.

Με την εγκατάσταση του ProBuilder, ο χρήστης βλέπει πλέον μια νέα γραμμή εργαλείων στο επάνω μέρος του Editor. Η σχεδίαση και μοντελοποίηση των αντικειμένων βασίζεται στη δυνατότητα που παρέχεται από το ProBuilder, να τροποποιεί τα χαρακτηριστικά, όχι μόνο του

αντικειμένου, κάτι που γινόταν και μονάχα μέσω του Editor, αλλά και των κορυφών, των ακμών και των πλευρών ενός αντικειμένου.

Η λειτουργία αυτή αποτελεί τη βάση για δημιουργία και επεξεργασία οποιουδήποτε αντικειμένου επιθυμεί ο χρήστης. Στην Εικόνα 29, απεικονίζεται η γραμμή εργαλείων, μέσω της οποίας ο χρήστης μπορεί να επιλέγει το αντικείμενο, τις κορυφές, τις ακμές ή τις όψεις του.



ΕΙΚΟΝΑ 29 - ΤΑ EDIT MODES ΤΟΥ PROBUILDER

Ειδικότερα, το πρώτο mode που συναντά ο χρήστης είναι το Object Mode και αποτελεί το προκαθορισμένο mode του Unity Editor, όπου η επιλογή ενός αντικειμένου οδηγεί στην επιλογή ολόκληρου του πλέγματός του.

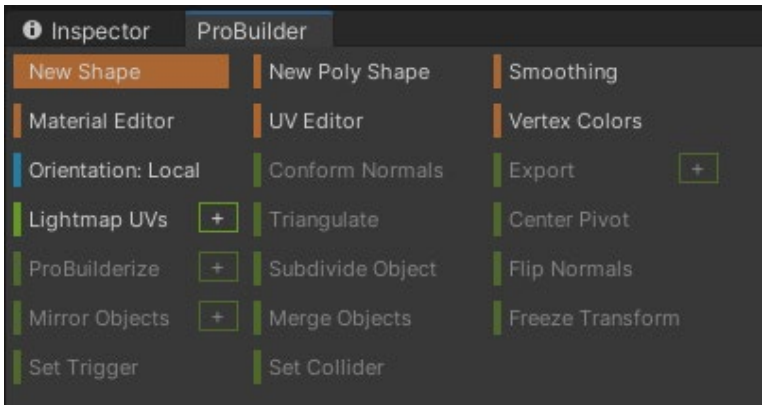
Τα υπόλοιπα τρία modes της Εικόνα 29, ονομάζονται Element modes. Μέσω αυτών, μπορούν να επιλεχθούν και να τροποποιηθούν μεμονωμένα στοιχεία της γεωμετρίας ενός αντικειμένου, δηλαδή κορυφές, ακμές και όψεις. Συγκεκριμένα:

- Ως **κορυφή** (vertex) ορίζεται το σημείο στο οποίο συναντιούνται δύο ή περισσότερες καμπύλες, γραμμές ή ακμές.
- Ως **ακμή** (edge) ορίζεται το τμήμα που συνδέει δύο κορυφές (vertices).
- Ως **όψη** (face) ορίζεται το τμήμα μεταξύ τριών ή περισσοτέρων ακμών.

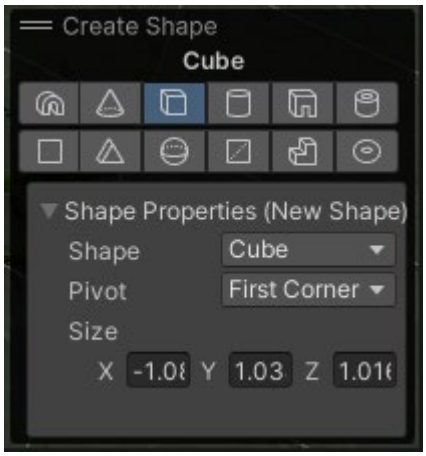
Τελικά, μέσω των παραπάνω αρχών και της γραμμής εργαλείων, σε συνδυασμό με μερικά ακόμη εργαλεία που θα παρουσιαστούν στη συνέχεια, ο χρήστης μπορεί να δημιουργεί, να επιλέγει και να τροποποιεί όλα τα αντικείμενα στην επιθυμητή μορφή.

Μαζί με την παραπάνω γραμμή εργαλείων, υπάρχει πλέον στον Editor και ακόμη μια νέα καρτέλα, η ProBuilder, η οποία παρέχει πρόσβαση σε ορισμένα πολύ χρήσιμα εργαλεία, ανάλογα και το Edit Mode που έχει επιλεχθεί. Κοινό σε όλα τα modes είναι το New Shape, η επιλογή του οποίου ανοίγει

ένα νέο παράθυρο για δημιουργία των πιο συνηθισμένων σχημάτων, όπως φαίνεται στην Εικόνα 30.



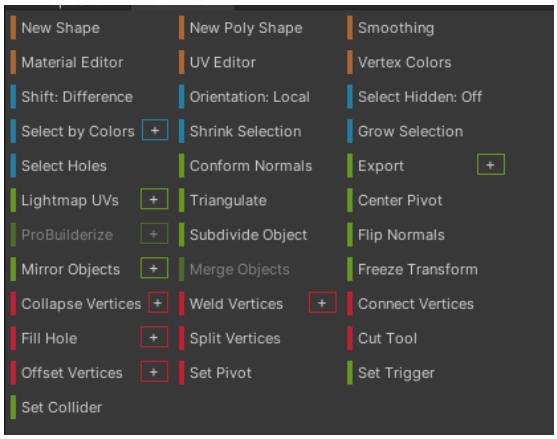
ΕΙΚΟΝΑ 30 - Η ΚΑΡΤΕΛΑ PROBUILDER



ΕΙΚΟΝΑ 31 - ΔΗΜΙΟΥΡΓΙΑ ΝΕΩΝ ΣΧΗΜΑΤΩΝ ΜΕ ΤΟ PROBUILDER

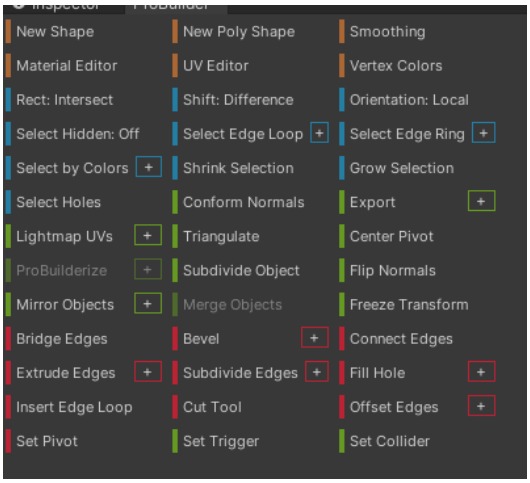
Συνοπτικά, όπως φαίνεται στην Εικόνα 31, τα βασικά σχήματα που παρέχονται είναι αψίδα, κώνος, κύβος, κύλινδρος, πόρτα, σωλήνας, επίπεδη επιφάνεια, πρίσμα, σφαίρα, σκάλα και δακτύλιος. Τα σχήματα αυτά, αποτελούν τη βάση για τη σχεδίαση πολύπλοκων και ολοκληρωμένων αντικειμένων.

- **Vertices** (κορυφές): Μόλις επιλεχθούν μία ή περισσότερες κορυφές ενός αντικειμένου, τότε οι διαθέσιμες επιλογές θα είναι:



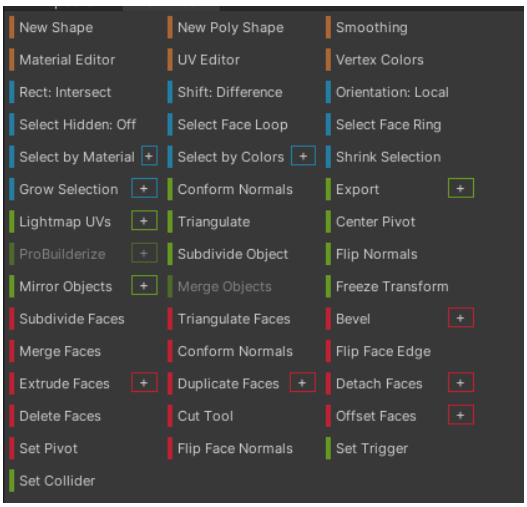
Εικονα 32 - Οι διαθεσιμες επιλογες του ProBuilder για τις κορυφες

- **Edges** (ακμές): Μόλις επιλεχθούν μία ή περισσότερες ακμές, τότε οι διαθέσιμες επιλογές θα είναι:



Εικονα 33 - Οι διαθεσιμες επιλογες του ProBuilder για τις ακμες

- **Faces** (όψεις): Μόλις επιλεχθεί μία ή περισσότερες όψεις, τότε οι διαθέσιμες επιλογές θα είναι:



Εικονα 34 - Οι διαθεσιμες επιλογες του ProBuilder για τις οψεις

Όπως είναι εμφανές, οι διαθέσιμες επιλογές – εργαλεία για καθένα από τα edit modes είναι αρκετές και η ανάλυση της καθεμιάς θα ήταν περιττή για τα πλαίσια της παρούσας εργασίας. Ωστόσο, θα γίνει παρακάτω μια συνοπτική περιγραφή των επιλογών που χρησιμοποιήθηκαν περισσότερο για την δημιουργία των κτηρίων του συγκεκριμένου παιχνιδιού.

Flip Normals: Αντιστρέφει τα κανονικά διανύσματα της επιλεγμένης επιφάνειας.

Bridge Edges: Συνδέει δύο ακμές δημιουργώντας μια νέα όψη μεταξύ τους.

Bevel Edges: Διαχωρίζει την επιλεγμένη ακμή σε δύο νέες ακμές, με μια όψη ενδιάμεσά τους.

Bevel Faces: Πραγματοποιεί την λειτουργία του Bevel Edges για όλες τις ακμές της επιλεγμένες επιφάνειας.

Connect Vertices: Δημιουργεί ακμή μεταξύ των επιλεγμένων κορυφών.

Connect Edges: Δημιουργεί μια νέα ακμή μεταξύ των κέντρων των επιλεγμένων ακμών.

Extrude Edges: Εξωθεί μια νέα ακμή για κάθε επιλεγμένη ακμή, συνδεδεμένη μέσω μιας όψης.

Flip Face Edge: Αντιστρέφει τον προσανατολισμό της επιλεγμένης όψης.

Collapse Vertices: Συμπτύσσει τις επιλεγμένες κορυφές σε ένα κοινό σημείο.

Connect Vertices: Δημιουργεί μια νέα ακμή που συνδέει τις επιλεγμένες κορυφές.

Merge Objects: Συγχωνεύει δύο ή περισσότερα επιλεγμένα ProBuilder αντικείμενα σε ένα κοινό.

Merge Faces: Συγχωνεύει τις επιλεγμένες όψεις σε μία κοινή όψη, αφαιρώντας οποιεσδήποτε διαχωριστικές ακμές.

3.2 Καταστροφή κτηρίων – αντικειμένων

Ένα βασικό χαρακτηριστικό του συγκεκριμένου παιχνιδιού είναι η δυνατότητα καταστροφής των αντικειμένων και των κτηρίων της πίστας. Η λειτουργία αυτή, αποσκοπεί στην μεταβολή της πορείας του παιχνιδιού, δίνοντας του έτσι ένα γρηγορότερο και πιο συναρπαστικό ρυθμό, καθώς θα αναγκάζει τους παίκτες να αλλάξουν την στρατηγική τους.

Οι παίκτες θα μπορούν να αλληλεπιδρούν με ορισμένα αντικείμενα του χάρτη, τα οποία μόλις δεχθούν κάποια χτυπήματα θα καταστρέφονται και θα υποχωρούν. Θα μπορούν έτσι να πλησιάσουν και να πλήξουν τους αντίπαλους τους από διαφορετικά σημεία, καταστρέφοντας παράλληλα τα σημεία κάλυψής τους.

Εφόσον τα περισσότερα σημεία, κυρίως των κτηρίων, θα καταστρέφονται, οι παίκτες θα χρειάζεται συνεχώς να μετακινούνται, προκειμένου να εντοπίσουν και να προχωρήσουν σε νέο ασφαλές σημείο. Με αυτόν τον τρόπο, η συνεχόμενη κίνηση των παικτών θα οδηγεί σε νέες συγκρούσεις μεταξύ τους, οπότε και θα διατηρείται ο γρήγορος ρυθμός του παιχνιδιού.

Για την επίτευξη όλων των παραπάνω χρησιμοποιήθηκε το πακέτο RayFire. Πρόκειται ουσιαστικά για ένα προηγμένο πακέτο καταστροφής, το οποίο επιτρέπει στο χρήστη να σπάει, να κατεδαφίζει και να τεμαχίζει 3D αντικείμενα. Το πακέτο RayFire παρέχει μια πληθώρα από scripts – components τα οποία μπορούν να αξιοποιηθούν για τον περαιτέρω εμπλουτισμό του παιχνιδιού. Το βασικότερο ωστόσο όλων και απαραίτητο, είναι το Rayfire Rigid, το οποίο και φαίνεται στην Εικόνα 35.

Η κύρια λειτουργία του αφορά τον έλεγχο της φυσικής γύρω από τα αντικείμενα που θα καταστρέφονται, όπως για παράδειγμα την συνθήκη καταστροφής, τον αριθμό θραυσμάτων, την δυνατότητα περαιτέρω καταστροφής των ήδη καταστραμμένων θραυσμάτων, τον τρόπο εξαφάνισης των θραυσμάτων κ.ά.

Παρακάτω θα αναλυθούν σε βάθος ορισμένες από τις βασικές ρυθμίσεις του Rigid, οι οποίες και τροποποιήθηκαν για τις ανάγκες του παιχνιδιού.

ΕΙΚΟΝΑ 35 - ΟΙ ΒΑΣΙΚΕΣ ΡΥΘΜΙΣΕΙΣ ΤΟΥ RIGID COMPONENT

- **Initialization**

  Ορίζει την χρονική στιγμή κατά την οποία θα αρχικοποιείται το αντικείμενο, δηλαδή τη στιγμή που θα προστίθενται στο αντικείμενο όλα τα απαραίτητα components, η ρύθμισή τους και η εκκίνηση των απαραίτητων λειτουργιών. Οι διαθέσιμες επιλογές είναι:

  - **By Method:** Τα αντικείμενα αρχικοποιούνται μέσω της συνάρτησης Initialize(). Με αυτό τον τρόπο, μπορεί ο χρήστης να προσθέσει το component Rigid στον

αντικείμενο μέσα από τον κώδικα του, και το αντικείμενο δεν θα αρχικοποιηθεί μέχρις ότου κληθεί η **Initialize()** μέθοδος.

- **At Start:** Το αντικείμενο θα αρχικοποιηθεί κατά την εκκίνηση του παιχνιδιού.

- **Object Type**

  Το component Rigid μπορεί να προσομοιώσει ένα μεμονωμένο αντικείμενο χρησιμοποιώντας το πλέγμα του (mesh), αλλά μπορεί επίσης να προσομοιώσει πολλαπλά αντικείμενα ως ένα σύνθετο αντικείμενο, χρησιμοποιώντας το πλέγμα του κάθε αντικειμένου – παιδιού. Η συγκεκριμένη λειτουργία είναι χρήσιμη όταν η καταστροφή ενός αντικειμένου επιφέρει αλλαγές και σε γειτονικά αντικείμενα. Στην προκειμένη περίπτωση ωστόσο, η συγκεκριμένη λειτουργία δεν έχει αξιοποιηθεί, οπότε και η επιλογή Object Type έχει τεθεί στην τιμή Mesh για όλα τα αντικείμενα της πίστας, οπότε η προσομοίωση κάθε αντικειμένου θα βασίζεται αποκλειστικά στη δομή του ίδιου και μόνο αντικειμένου.

  Mesh: Στον τομέα της γραφικής και της υπολογιστικής γεωμετρίας, το "mesh" αναφέρεται σε ένα 3D μοντέλο που αποτελείται από σύνολο συνδεδεμένων πολυγώνων. Τα πολύγωνα αυτά μπορεί να είναι τρίγωνα, τετράγωνα ή άλλες γεωμετρικές μορφές.

  Τα mesh χρησιμοποιούνται ευρέως στον τομέα των γραφικών υπολογιστών και της 3D γραφικής, καθώς αποτελούν τη βασική δομική μονάδα για την απεικόνιση αντικειμένων στον τρισδιάστατο χώρο. Τα mesh αποτελούνται από κόμβους (vertices), άκρα (edges) και όψεις (faces), που σχηματίζουν το τελικό 3D μοντέλο.

  Κατά την απεικόνιση, το mesh καθορίζει το σχήμα και την εμφάνιση του αντικειμένου, περιλαμβάνοντας πληροφορίες όπως το χρώμα, τις υφές και τα υλικά. Η επεξεργασία και η διαχείριση mesh είναι σημαντική στην ανάπτυξη παιχνιδιών, εφαρμογών εικονικής πραγματικότητας, κινηματογραφίας CGI και άλλων σχετικών πεδίων.

- **Simulation Type**

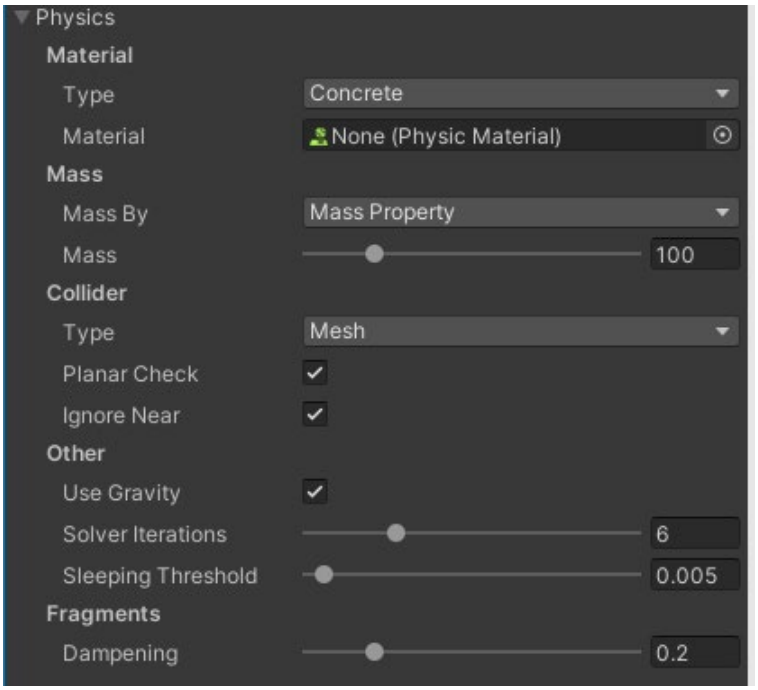
  Ορίζει τη συμπεριφορά του αντικειμένου κατά την προσομοίωση. Οι διαθέσιμες επιλογές είναι οι εξής:

  - **Dynamic:** Το αντικείμενο θα επηρεάζεται από τη βαρύτητα, θα αρχίσει να πέφτει και είναι πιθανό να επηρεαστεί από την κίνηση άλλων αντικειμένων.
  - **Sleeping:** Το αντικείμενο θα παραμείνει ακίνητο στον αέρα μέχρις ότου συγκρουστεί για πρώτη φορά με άλλο αντικείμενο, και τότε θα αρχίσει να συμπεριφέρεται σαν Dynamic αντικείμενο.
  - **Inactive:** Το αντικείμενο θα παραμείνει ακίνητο στον αέρα, θα μπορεί να επηρεαστεί από όλα αντικείμενα αλλά δεν θα ξεκινήσει να πέφτει κάτω μέχρις

ότου ενεργοποιηθεί. Μετά την ενεργοποίηση, το αντικείμενο θα συμπεριφέρεται σαν Dynamic αντικείμενο.

- **Kinematic:** Το αντικείμενο θα χρησιμοποιεί τις κινήσεις του για να επηρεάζει άλλα αντικείμενα αλλά το ίδιο δεν θα επηρεάζεται με κανένα τρόπο, από κανένα άλλο αντικείμενο.
- **Static:** Το αντικείμενο δεν θα μετακινείται από τη θέση του, θα αλληλεπιδρά με άλλα Dynamic αντικείμενα, αλλά δεν θα μπορεί να επηρεαστεί από κανένα άλλο αντικείμενο.

- **Physics:** Επιτρέπει τον έλεγχο της δυναμικής προσομοίωσης των ρυθμίσεων που αφορούν τη φυσική των αντικειμένων.



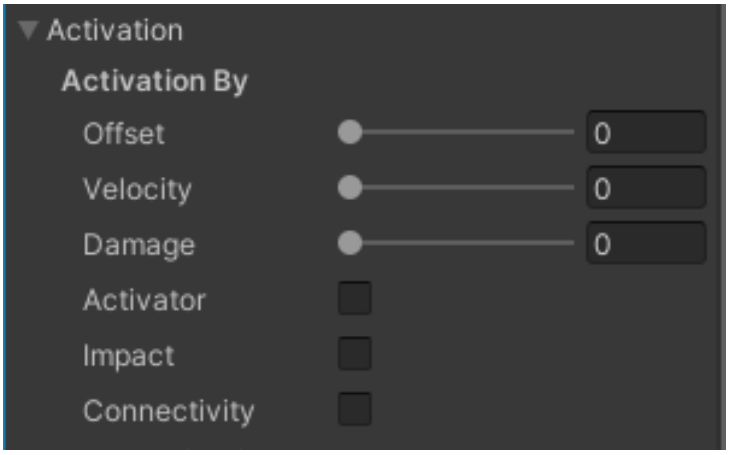
ΕΙΚΟΝΑ 36 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΓΙΑ ΤΑ PHYSICS ΣΤΟ RAYFIRE RIGID

- **Material Type**: Παρέχει μια σειρά επιλογών όπως Concrete, Heavy Metal, Light Metal, Dense Rock, Porous Rock, Brick, Glass, Rubber, Ice και Wood, τα οποία είναι προκαθορισμένες τιμές για την πυκνότητα, την τριβή, την ελαστικότητα και την στερεότητα. Στο συγκεκριμένο παιχνίδι, έχουν χρησιμοποιηθεί κυρίως οι τιμές Concrete και Wood.
- **Material:** Επιτρέπει τη δημιουργία ενός νέου Material Type, διαφορετικού από τα προαναφερθέντα.
- **Mass By**: Επιτρέπει την επιλογή του τρόπου με τον οποίο η δύναμη του Βάρους θα εφαρμόζεται στο Rigid component. Εδώ, έχει τεθεί η τιμή Mass Property, όπου η τιμή του βάρους του αντικειμένου θα ορίζεται από το slider Mass που ακολουθεί.

- **Collider Type**: Επιτρέπει την επιλογή της μορφής του collider που θα έχει το αντικείμενο, δηλαδή της περιοχής εκείνης που θα αναγνωρίζει και θα αντιδρά σε φυσικά γεγονότα, όπως οι συγκρούσεις. Οι επιλογές που παρέχονται είναι Mesh, Sphere, Box, None, οπότε για την συγκεκριμένη περίπτωση, έχει προτιμηθεί η επιλογή Mesh Collider, ώστε η προαναφερθείσα περιοχή να βασίζεται στη δομή του εκάστοτε αντικειμένου.
- **Planar Check**: Αφορά τη συμπεριφορά που έχουν τα αντικείμενα με Mesh Collider, τα οποία τοποθετούνται γειτονικά με άλλα τέτοια αντικείμενα. Τίθεται ως true, καθώς διαφορετικά, η καταστροφή ενός αντικειμένου μπορεί να έχει αναπάντεχα αποτελέσματα στα γειτονικά αντικείμενα.
- **Ignore Near**: Επιτρέπει στο αντικείμενο να αγνοεί όλους τους colliders γύρω από το αντικείμενο στην περίπτωση που αυτοί συνορεύουν με άλλους. Η τιμή του τέθηκε ως αληθής, καθώς στο συγκεκριμένο παιχνίδι υπάρχουν αρκετά αντικείμενα τοποθετημένα γειτονικά, προκειμένου να αποφευχθούν αστάθειες στην προσομοίωση.

- **Activation:**

  Ορίζει τη συνθήκη με την οποία θα ενεργοποιηθούν τα Inactive αντικείμενα. Στην τρέχουσα περίπτωση, τα αντικείμενα έχουν τεθεί ως Kinematic, οπότε δεν χρήζουν ενεργοποίησης.



Εικονα 37 - Οι ρυθμισεις για το Activation στο Rayfire Rigid

  - **By Velocity**: Τα Inactive αντικείμενα θα ενεργοποιούνται μόλις η ταχύτητά τους γίνει μεγαλύτερη από την ορισμένη τιμή, μόλις δηλαδή κάποιο dynamic αντικείμενο τα μετακινήσει.
  - **By Offset**: Τα Inactive αντικείμενα θα ενεργοποιούνται μόλις μετακινηθούν από την αρχική τους θέση κατά την τιμή που έχει οριστεί.

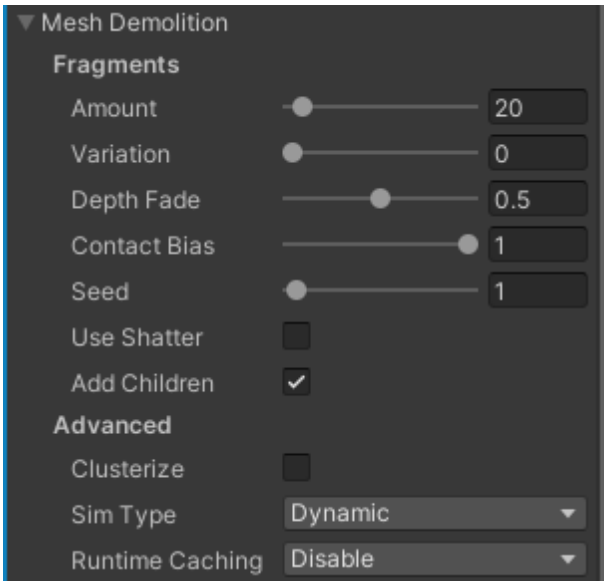
- **By Damage**: Το Inactive αντικείμενο θα ενεργοποιηθεί μόλις η τιμή του Current Damage ξεπεράσει την τιμή που έχει οριστεί.
- **By Activator**: Το Inactive αντικείμενο θα ενεργοποιηθεί μόλις αλληλεπιδράσει με κάποιο άλλο αντικείμενο με το component RayFire Activator
- **By Impact**: Το Inactive αντικείμενο θα ενεργοποιηθεί μόλις αλληλεπιδράσει με αντικείμενο που έχει το RayFire Gun αντικείμενο.
- **By Connectivity**: Το Inactive αντικείμενο θα ενεργοποιηθεί μόλις χαθεί η σύνδεση του αντικειμένου με άλλα συνδεδεμένα inactive αντικείμενα, τα οποία έχουν το component RayFire Connectivity. Η επιλογή αυτή είναι χρήσιμη όταν χρειάζεται να υλοποιηθεί αλυσιδωτή αλληλουχία καταστροφών μεταξύ των αντικειμένων.

- **Demolition Type:** Ορίζει το πότε και πως θα καταστρέφεται ένα αντικείμενο κατά τη διάρκεια του παιχνιδιού. Οι διαθέσιμες επιλογές είναι οι παρακάτω:
  - **None**: Το αντικείμενο δεν θα καταστρέφεται.
  - **Runtime**: Το αντικείμενο θα καταστραφεί κατά τη διάρκεια του παιχνιδιού και τα θραύσματα θα δημιουργηθούν τη στιγμή της καταστροφής. Προτείνεται να χρησιμοποιείται για αντικείμενα που δεν δημιουργούν αρκετά θραύσματα για λόγους βελτιστοποίησης της απόδοσης του παιχνιδιού.
  - **Awake Precache**: Το αντικείμενο θα έχει προφορτώσει τα θραύσματα, και θα τα εμφανίσει κατά τη στιγμή της επαφής, οπότε θα έχουν ήδη πραγματοποιηθεί οι περισσότερες χρονοβόρες διαδικασίες. Προτείνεται για αντικείμενα που δημιουργούν πολλά θραύσματα.
  - **Awake Prefragment**: Το αντικείμενο θα έχει ήδη διασπαστεί σε μέρη και τα θραύσματα θα έχουν ήδη σχηματιστεί σαν ξεχωριστά αντικείμενα αλλά θα παραμένουν απενεργοποιημένα, αναμένοντας την καταστροφή. Προτείνεται σε περιπτώσεις όπου τα αντικείμενα θρυμματίζονται σε πάρα πολλά μικρά μέρη.
  - **Reference Demolition**: Επιτρέπει στο χρήστη να ορίζει ο ίδιος τον τρόπο με τον οποίο θα καταστρέφεται κάποιο αντικείμενο. Προτείνεται σε περιπτώσεις όπου οι χρήστες επιθυμούν να δημιουργήσουν πολύπλοκες σκηνές καταστροφών.
- **Limitations:** Περιέχει ρυθμίσεις σχετικές με την καταστροφή των αντικειμένων.
  - **By Collision:** Ενεργοποιεί την καταστροφή μέσω σύγκρουσης.
  - **Solidity:** Ορίζει την τιμή της στερεότητας του αντικειμένου. Η χαμηλή τιμή του solidity κάνει τα αντικείμενα περισσότερο εύθραυστα. Αν τεθεί σε 0, τότε το αντικείμενο θα καταστραφεί κατά την πρώτη επαφή.
  - **Tag:** Τα αντικείμενα θα καταστρέφονται μέσω σύγκρουσης μόνο όταν έρχονται σε επαφή με αντικείμενα των οποίων τα tags βρίσκονται σε αυτή τη λίστα.

- **Depth:** Ορίζει το βάθος στο οποίο θα καταστρέφονται τα αντικείμενα και στη συνέχεια τα θραύσματα τους εκ νέου. Αν για παράδειγμα τεθεί σε 2, τότε το αρχικό αντικείμενο θα καταστραφεί μία φορά, τα θραύσματα του (θραύσματα 1ου επιπέδου) θα μπορούν να καταστραφούν εκ νέου και να δημιουργήσουν νέα θραύσματα (θραύσματα 2ου επιπέδου), τα οποία όμως δεν θα μπορούν να καταστραφούν ξανά.
- **Time:** Είναι ουσιαστικά μια μεταβλητή ασφαλείας, η οποία αποτρέπει τα νέα θραύσματα από το να καταστραφούν ξανά, αν δεν έχει περάσει η τιμή που ορίζεται εδώ.
- **Size:** Αποτρέπει αντικείμενα με εμβαδόν μικρότερο από την τιμή αυτή να καταστραφούν.
- **Visible:** Αν τεθεί ως αληθής, τότε τα αντικείμενα θα καταστρέφονται μονάχα αν είναι ορατά σε κάποια κάμερα του παιχνιδιού. Αν όχι, τότε δεν θα καταστραφούν ακόμα και αν έρθουν σε σύγκρουση με κάποιο άλλο αντικείμενο.
- **Slice by Blade:** Η συγκεκριμένη επιλογή χρησιμοποιείται όπου χρειάζεται τα αντικείμενα να κόβονται ή να σκίζονται.

ΕΙΚΟΝΑ 38 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΤΟΥ LIMITATIONS ΣΤΟ RAYFIRE RIGID

- **Mesh Demolition:** Παρέχει ρυθμίσεις σχετικές με τα θραύσματα που δημιουργεί το κάθε αντικείμενο.

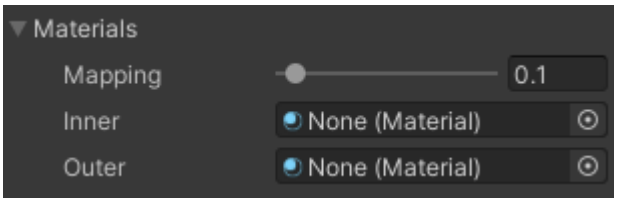


ΕΙΚΟΝΑ 39 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΤΟΥ MESH DEMOLITION ΣΤΟ RAYFIRE RIGID

- **Amount**: Ορίζει το πλήθος των θραυσμάτων μετά την καταστροφή του αντικειμένου.
- **Variation**: Αποτελεί ένα επιπρόσθετο ποσοστό διάσπασης των αντικειμένων σε συνάρτηση με την τιμή του Amount. Αν για παράδειγμα η τιμή του amount είναι 50 και του variation 10%, τότε τα τελικά θραύσματα θα είναι τυχαία, μεταξύ 50 και 55.
- **Depth Fade**: Σε περίπτωση που το Depth είναι μεγαλύτερο του 1, οπότε και θα υπάρχουν διαφορετικά επίπεδα θραυσμάτων, η τιμή του Depth Fade επιτρέπει την μείωση των θραυσμάτων για τα επόμενα επίπεδα. Αν για παράδειγμα το Amount είναι 100 και το Depth Fade είναι 0.5, τότε το αρχικό αντικείμενο θα διασπαστεί σε 100 κομμάτια. Κάθε τέτοιο κομμάτι, θα μπορεί να διασπαστεί σε 100 * 0.5 = 50 νέα κομμάτια. Στη συνέχεια, κάθε τέτοιο θραύσμα, θα μπορεί να διασπαστεί σε 50 * 0.5 = 25 νέα κομμάτια.
- **Contact Bias**: Επιτρέπει τη δημιουργία μικρότερων θραυσμάτων κοντά στο σημείο της σύγκρουσης και μεγαλύτερα μακριά από αυτήν. Με αυτήν την ρύθμιση, μπορεί ο παίκτης να καταλάβει πως έχει εφαρμοστεί η μέθοδος Runtime Demolition και όχι κάποια από τις μεθόδους που προφορτώνουν τα θραύσματα.
- **Seed**: Δίνεται στον όρισμα στον αλγόριθμο διάσπασης των αντικειμένων. Το ίδιο seed θα έχει ως αποτέλεσμα την παραγωγή των ίδιων ακριβώς θραυσμάτων για το αντικείμενο κάθε φορά.
- **Use Shatter**: Χρησιμοποιείται μαζί με το component RayFire Shatter όταν ο χρήστης επιθυμεί να δημιουργήσει ή να εισάγει πιο πολύπλοκες μεθόδους καταστροφής και διάσπασης.

- **Add Children**: Ορίζει το αν τα θραύσματα θα αποτελούν αντικείμενα – παιδιά του αρχικού αντικειμένου μετά την καταστροφή.
- **Clusterize**: Μετατρέπει τα θραύσματα σε συνδεδεμένα clusters και χρειάζεται όταν δημιουργούνται αλυσιδωτές καταστροφές, όπου η μία καταστροφή επιφέρει κάποια άλλη.
- **Sim Type**: Ορίζει το Simulation Type, που περιεγράφηκε νωρίτερα, για τα δημιουργηθέντα θραύσματα.
- **Runtime Caching**: Απαιτείται στην περίπτωση που χρειάζεται να αποθηκεύονται τα θραύσματα μετά την καταστροφή τους.

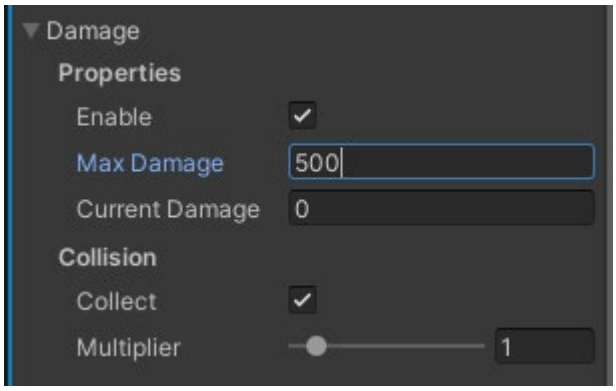
- **Materials:**



Εικονα 40 - Οι ρυθμισεις του Rayfire Rigid για τα materials

- **Inner Material:** Ορίζει το material, δηλαδή το στοιχείο που είναι υπεύθυνο για την εμφάνιση, το χρώμα, την λάμψη και τη διαφάνεια του αντικειμένου, για την **εσωτερική** επιφάνεια των θραυσμάτων. Αν δεν οριστεί διαφορετικά, τότε το RayFire χρησιμοποιεί το material του αρχικού αντικειμένου.
- **Mapping:** Μέσω του mapping ορίζεται το μέγεθος του material της εσωτερικής επιφάνειας. Μικρότερες τιμές του mapping έχουν ως αποτέλεσμα περισσότερη λεπτομέρεια στην εμφάνιση των θραυσμάτων.
- **Outer Material:** Ορίζει το material των εξωτερικών επιφανειών των θραυσμάτων.

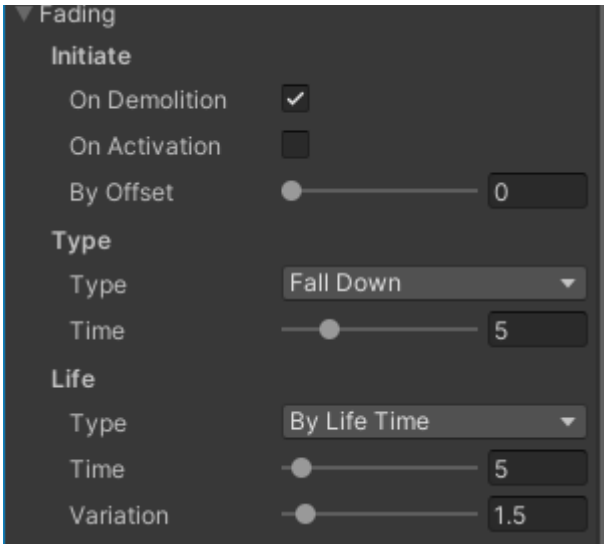
- **Damage:**



Εικονα 41 - Οι ρυθμισεις του Rayfire Rigid για το Damage

- **Enable:** Επιτρέπει την καταστροφή ενός αντικειμένου μέσω μιας μεταβλητής Damage value, διαφορετικής για το κάθε αντικείμενο. Το αντικείμενο θα καταστραφεί μόλις η τιμή της τρέχουσας ζημιάς είναι μεγαλύτερη ή ίσα της μέγιστης ζημιάς. Η τιμή της τρέχουσας ζημιάς μπορεί να αυξηθεί μέσω των RayFireBomb και RayFireGun components, ωστόσο στο παρόν παιχνίδι, η τιμή της ζημιάς αυξάνεται μέσω των συγκρούσεων.
- **Max Damage:** Η τιμή της μέγιστης ζημιάς προτού το αντικείμενο καταστραφεί.
- **Current Damage:** Η τιμή της τρέχουσας ζημιάς.
- **Collisions collect:** Επιτρέπει την συσσώρευση της ζημιάς όταν προέρχεται από συγκρούσεις.
- **Multiplier:** Πολλαπλασιάζει τη ζημιά από συγκρούσεις.

- **Fading:** Ορίζει τον τρόπο και την συνθήκη με την οποία θα εξαφανίζονται τα θραύσματα μετά την καταστροφή των αντικειμένων. Η εξαφάνιση των θραυσμάτων είναι απαραίτητη καθώς κρατά την σκηνή καθαρή και βελτιώνει την απόδοση του παιχνιδιού.



ΕΙΚΟΝΑ 42 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΤΟΥ RAYFIRE RIGID ΓΙΑ ΤΟ FADING

- **On Demolition:** Η διαδικασία εξαφάνισης των θραυσμάτων θα εκκινήσει αμέσως μετά την καταστροφή του αρχικού αντικειμένου.
- **On Activation:** Η διαδικασία εξαφάνισης των θραυσμάτων θα εκκινήσει αμέσως μετά την ενεργοποίηση των αντικειμένων.
- **By Offset:** Η διαδικασία εξαφάνισης των θραυσμάτων θα εκκινήσει μόλις τα αντικείμενα μετακινηθούν κατά την τιμή που ορίζεται από την αρχική τους θέση.
- **Type:** Ορίζει τον τρόπο με τον οποίο θα εξαφανιστούν τα θραύσματα και οι διαθέσιμες επιλογές είναι None, Sim Exclude, Fall Down, Scale Down, Move Down, Destroy, Set Static και Set Kinematic.

- **Life Type**: Ορίζει τον χρόνο για τον οποίο τα θραύσματα θα φαίνονται προτού αρχίσουν να εξαφανίζονται. Μπορεί να τεθεί σε **By Life Time**, όπου ο μετρητής της διάρκειας ζωής των θραυσμάτων ξεκινά αμέσως μόλις εκκινηθεί η διαδικασία εξαφάνισης του αντικειμένου, και είναι ίσο με την τιμή της μεταβλητής Time. Μπορεί επίσης να τεθεί σε **By Life Time and Simulation**, όπου ο μετρητής της διάρκειας ζωής ξεκινά μόλις το αντικείμενο σταματήσει να κινείται για κάποιο χρονικό διάστημα.
- **Life Variation**: Προσθέτει τυχαίο χρονικό διάστημα στο τελικό **Life Time**.

3.3 Δημιουργία του χαρακτήρα

Έχοντας πλέον δημιουργήσει την πίστα του παιχνιδιού, χρειάζεται τώρα να δημιουργηθεί και ο χαρακτήρας που θα έχουν στη διάθεσή τους οι παίκτες. Για το σκοπό αυτό, χρησιμοποιήθηκε ένα έτοιμο asset χαρακτήρα – στρατιώτη, προκειμένου να ταιριάζει στο ύφος του παιχνιδιού. Ο χαρακτήρας, μαζί με το σύνολο των animations – κινήσεων του, παρέχεται από την ιστοσελίδα Mixamo[7]. Στο Mixamo μπορεί κανείς να βρει μια πληθώρα κινουμένων σχεδίων που αφορούν την κίνηση του χαρακτήρα με ή χωρίς όπλο, τον τρόπο που πυροβολεί, την δυνατότητα να σκύβει και να σύρεται καθώς και πολλά άλλα, μαζί φυσικά με τους όλους τους δυνατούς συνδυασμούς αυτών. Στα πλαίσια της τρέχουσας εργασίας, χρησιμοποιήθηκαν ορισμένα animations που εστιάζουν στις βασικές κινήσεις του παίκτη όπως:

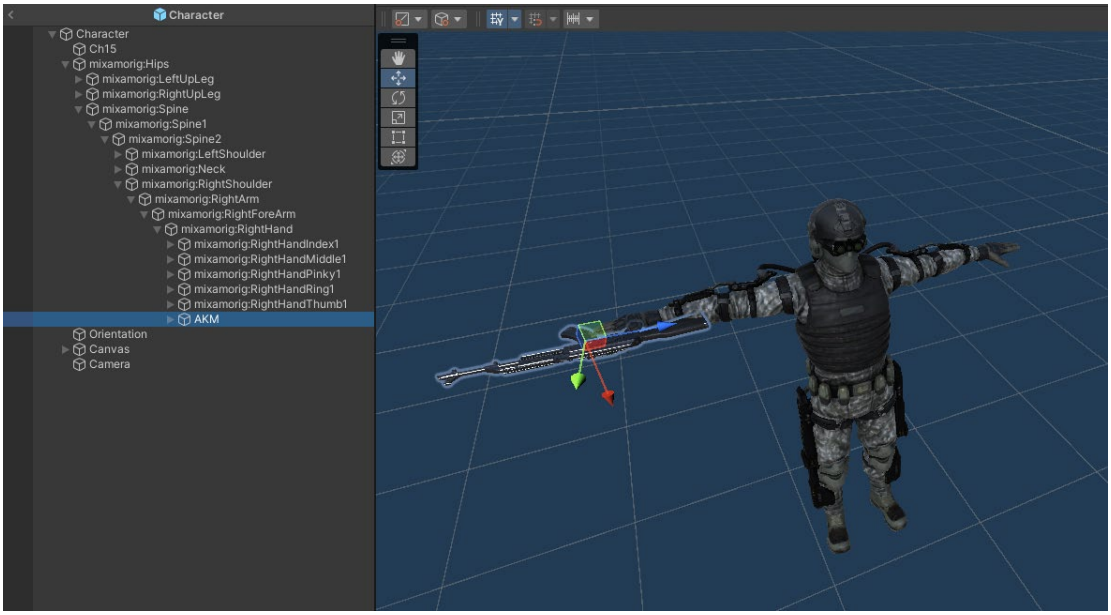
- Βάδισμα με και χωρίς στόχευση του όπλου προς τα εμπρός και προς τα πίσω,
- Τρέξιμο με και χωρίς στόχευση του όπλου προς τα εμπρός και προς τα πίσω,
- Πυροβολισμός και ταυτόχρονη στόχευση και κίνηση του παίκτη,
- Αλλαγή γεμιστήρα (Reload).

Όλα τα παραπάνω μπορούν να εμπλουτιστούν περαιτέρω και να δημιουργήσουν έναν πλήρως ολοκληρωμένο χαρακτήρα. Φυσικά, οι παραπάνω κινήσεις διαφοροποιούνται βάσει της φύσης του όπλου. Για παράδειγμα, στο παιχνίδι που υλοποιήθηκε, ο χαρακτήρας έχει στη διάθεση του ένα πυροβόλο τυφέκιο. Αν επίσης, είχε στη διάθεση του ένα ακόμη όπλο τύπου πιστολιού, τότε θα χρειαζόταν να υλοποιηθεί ένα τελείως διαφορετικό σύνολο από animations, μιας και οι κινήσεις πλέον του χαρακτήρα θα ήταν τελείως διαφορετικές.

Το όπλο που χρησιμοποιήθηκε παρέχεται δωρεάν από το Unity Asset Store[8] και το μόνο που χρειάζεται αφού εισαχθεί το asset στο project, είναι να συρθεί και να γίνει αντικείμενο - παιδί του

[7] https://www.mixamo.com/#/
[8] https://assetstore.unity.com/packages/3d/props/guns/free-fps-weapon-akm-180663

δεξιού καρπού του στρατιώτη, όπως φαίνεται στην Εικόνα 43. Πλέον, με λίγη περιστροφή και μετακίνηση, το όπλο βρίσκεται ακριβώς στη θέση που πρέπει και κάθε κίνηση πλέον του στρατιώτη, θα μετακινεί αντίστοιχα και το όπλο. (Εικόνα 44)



ΕΙΚΟΝΑ 43 - ΤΟΠΟΘΕΤΗΣΗ ΟΠΛΟΥ ΣΤΟΝ ΔΕΞΙ ΚΑΡΠΟ ΤΟΥ ΧΑΡΑΚΤΗΡΑ

ΕΙΚΟΝΑ 44 - ΜΕΤΑΚΙΝΗΣΗ ΟΠΛΟΥ ΒΑΣΕΙ ΤΩΝ ΚΙΝΗΣΕΩΝ ΤΟΥ ΧΑΡΑΚΤΗΡΑ

3.3.1 Κίνηση Χαρακτήρα

Για την κίνηση και περιστροφή του χαρακτήρα, υπεύθυνο κατά κύριο λόγο είναι το script **ThirdPersonController.cs**. Παρακάτω θα γίνει αναφορά και επεξήγηση στα κυριότερα μέρη του κώδικά του.

```
// Update is called once per frame
void Update() {
    …
        HandleMovementInput();
        HandleMouseLook();
        ApplyFinalMovements();
    …
}
```

ΚΩΔΙΚΑΣ 2 - ΟΙ ΣΥΝΑΡΤΗΣΕΙΣ ΚΙΝΗΣΗΣ ΤΟΥ ΠΑΙΚΤΗ

Η **Update()** είναι η μέθοδος που καλείται αυτόματα σε κάθε καρέ (frame) του παιχνιδιού, οπότε σε κάθε επανάληψή της θα εκτελούνται όλες οι ενέργειες που αφορούν την κίνηση του χαρακτήρα και οι οποίες παρουσιάζονται στον Κώδικας 2.

Για τον χειρισμό και επεξεργασία της εισόδου του χρήστη για την κίνηση του παίκτη, δηλαδή της εισόδου που δίνει ο χρήστης μέσω του πληκτρολογίου του, έχει υλοποιηθεί η συνάρτηση **HandleMovementInput()**, η οποία αξιοποιεί την **Input.GetAxis(),** όπως φαίνεται στο παρακάτω τμήμα κώδικα (Κώδικας 3).

```
private void HandleMovementInput() {

    currentInput = new Vector2((IsSprinting ? sprintSpeed :  walkSpeed) *
Input.GetAxis("Vertical"), (IsSprinting ? sprintSpeed : walkSpeed) *
Input.GetAxis("Horizontal"));

    if (IsWalkingForwards && !IsSprinting) {
       animator.SetFloat("speed", walkSpeed);
       animator.SetBool("isWalkingForwards", true);
    } else if (IsSprinting && IsWalkingForwards) {
         animator.SetFloat("speed", sprintSpeed);
         animator.SetBool("isWalkingForwards", true);
    }
    …
    else {
        animator.SetFloat("speed", 0f);
        animator.SetBool("isWalkingForwards", false);
        animator.SetBool("isWalkingBackwards", false);
    }
    float   moveDirectionY = moveDirection.y;
            moveDirection =
            (transform.TransformDirection(Vector3.forward)*
            currentInput.x) +
            (transform.TransformDirection(Vector3.right) *
            currentInput.y);
        moveDirection.y = moveDirectionY;
    }
```

ΚΩΔΙΚΑΣ 3 - ΧΕΙΡΙΣΜΟΣ ΚΙΝΗΣΗΣ ΤΟΥ ΠΑΙΚΤΗ ΜΕΣΩ ΠΛΗΚΤΡΟΛΟΓΙΟΥ

Αντίστοιχα, για τον χειρισμό της εισόδου του χρήστη μέσω του ποντικιού, που είναι υπεύθυνη για την περιστροφή του χαρακτήρα και της κάμερας, υλοποιήθηκε η συνάρτηση **HandleMouseLook()** όπως φαίνεται στον παρακάτω κώδικα (Κώδικας 4).

```
private void HandleMouseLook() {
  rotationX -= Input.GetAxis("Mouse Y") * lookSpeedY;
  rotationX = Mathf.Clamp(rotationX, -upperLookLimit,
  lowerLookLimit);
  //Rotating camera
  playerCamera.transform.localRotation = Quaternion.Euler(rotationX,
  0f, 0f);
  //Rotating player
  transform.rotation *= Quaternion.Euler(0,Input.GetAxis("Mouse X") *
  lookSpeedX, 0f);}
```

ΚΩΔΙΚΑΣ 4 - ΧΕΙΡΙΣΜΟΣ ΚΙΝΗΣΗΣ ΤΟΥ ΠΑΙΚΤΗ ΜΕΣΩ ΠΟΝΤΙΚΙΟΥ

Το μόνο που απομένει είναι η εφαρμογή των παραπάνω, στο σώμα του χαρακτήρα, προκειμένου να μετακινηθεί και να περιστραφεί. Αυτό επιτυγχάνεται μέσω της συνάρτησης **Move()** που ανήκει στο component Character Controller, το οποίο επισυνάπτεται στο αντικείμενο του χαρακτήρα και παρουσιάζεται στον Κώδικας 5.

```
private void ApplyFinalMovements() {
  characterController.Move(moveDirection* Time.deltaTime);
}
```

ΚΩΔΙΚΑΣ 5 - ΕΦΑΡΜΟΓΗ ΚΙΝΗΣΕΩΝ ΣΤΟ ΑΝΤΙΚΕΙΜΕΝΟ ΤΟΥ ΧΑΡΑΚΤΗΡΑ

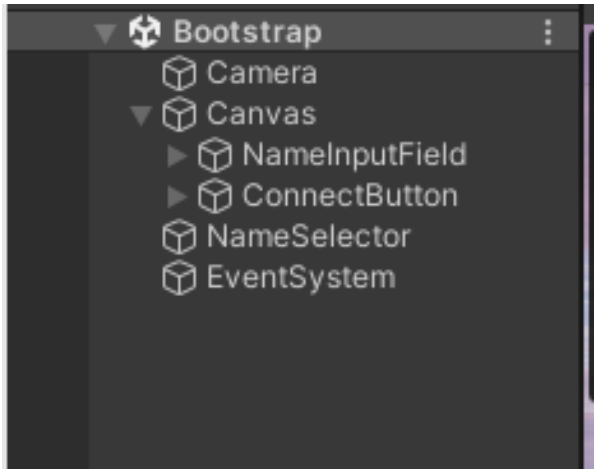
3.4 Διεπαφή χρήστη (User Interface)

Πέρα των βασικών λειτουργιών που περιλαμβάνει ένα παιχνίδι, χρειάζεται και μια διεπαφή με την οποία θα μπορεί να αλληλεπιδρά ο χρήστης. Ένα ολοκληρωμένο παιχνίδι αποτελείται από αρκετές σκηνές όπου η καθεμία υλοποιεί διαφορετική λειτουργία. Στην προκειμένη περίπτωση, θα υπάρχει μια σκηνή στην οποία ο χρήστης θα μπορεί να εισαγάγει το όνομά του ώστε να φαίνεται στους υπόλοιπους. Στην συνέχεια θα υπάρχει η σκηνή με τις διαθέσιμες επιλογές του χρήστη σε ότι αφορά τη σύνδεση σε υπάρχον ή τη δημιουργία ενός νέου δωματίου. Τέλος, θα εμφανίζεται η κύρια σκηνή του παιχνιδιού, η οποία επίσης θα περιλαμβάνει μερικά ακόμη στοιχεία διεπαφής, όπως η μπάρα ζωής του χαρακτήρα (health bar), και ο πίνακας κατάταξης. Παρακάτω θα παρουσιαστούν αναλυτικά οι σκηνές που προαναφέρθηκαν.

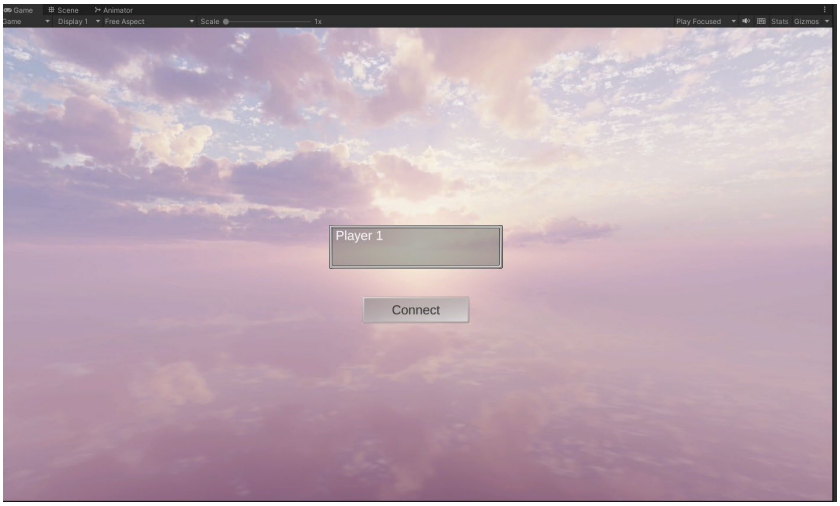
3.4.1 Σκηνή Bootstrap

Αποτελεί τη σκηνή που συναντά πρώτη κατά σειρά ο παίκτης και στην οποία εισάγει το όνομά του και προχωρά στη σύνδεση με το παιχνίδι. Αποτελείται από τα αντικείμενα της Εικόνα 45. Εντός

του αντικειμένου Canvas, έχουν δημιουργηθεί, ένα πεδίο εισόδου (NameInputField) στο οποίο ο χρήστης πληκτρολογεί το όνομα του, και ένα κουμπί (ConnectButton), το πάτημα του οποίου οδηγεί στην επόμενη σκηνή.

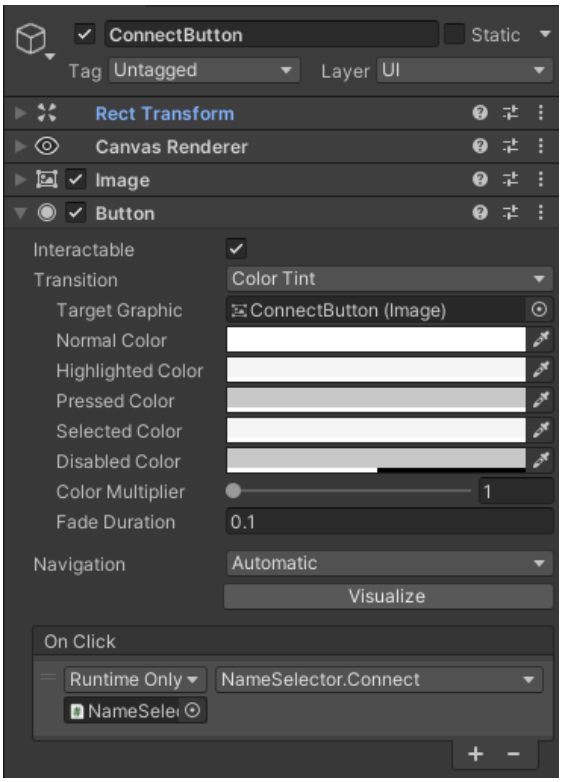


ΕΙΚΟΝΑ 45 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ ΤΗΣ ΣΚΗΝΗΣ BOOTSTRAP



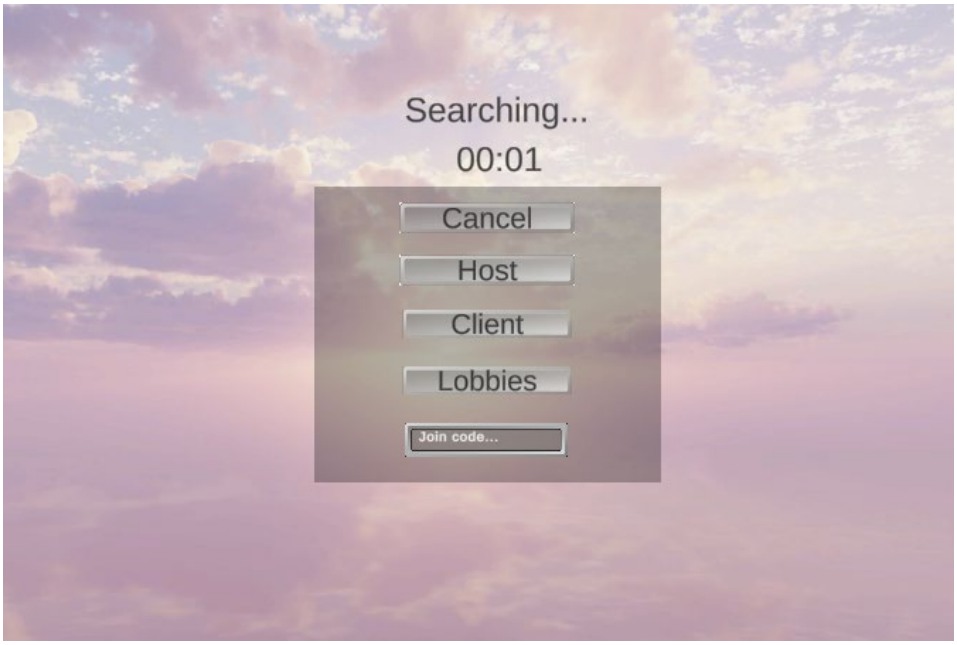
ΕΙΚΟΝΑ 46 - Η ΣΚΗΝΗ BOOTSTRAP

Εκτός του Canvas, στη σκηνή βρίσκεται και ένα κενό αντικείμενο με όνομα NameSelector. Η ύπαρξη του αντικειμένου αυτού είναι απαραίτητη προκειμένου να είναι δυνατή η επισύναψη σε αυτό του script **NameSelector.cs**, το οποίο ελέγχει την καταλληλότητα του ονόματος χρήστη, και αν πληρούνται οι προϋποθέσεις, δίνει τη δυνατότητα στο κουμπί Connect να προχωρήσει στη φόρτωση της επόμενης σκηνής.



Εικονα 47 - Το περιεχομενο του κουμπιου Connect

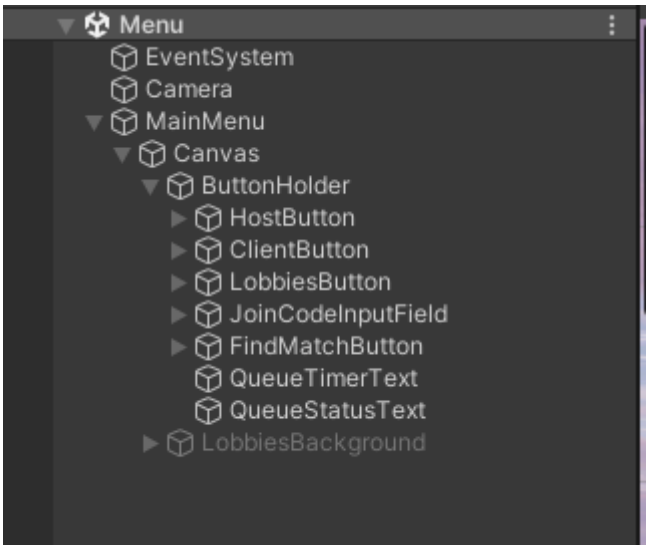
3.4.2 Σκηνή Menu



Εικονα 48 - Η σκηνη Menu

Η σκηνή Menu αποτελεί τη σκηνή στην οποία ο παίκτης επιλέγει τον τρόπο με τον οποίο θα συνδεθεί στο παιχνίδι, και ειδικότερα αν θα συνδεθεί σε κάποιο server, αν θα δημιουργήσει ο ίδιος ένα δωμάτιο ως host, αν θα επιλέξει να συνδεθεί σε κάποιο υπάρχον δωμάτιο του οποίο

γνωρίζει τον κωδικό ή τέλος αν θα επιλέξει να δει όλα τα διαθέσιμα δωμάτια άλλων παικτών. Πιο συγκεκριμένα, η σκηνή Menu αποτελείται από τα αντικείμενα της Εικόνα 49.



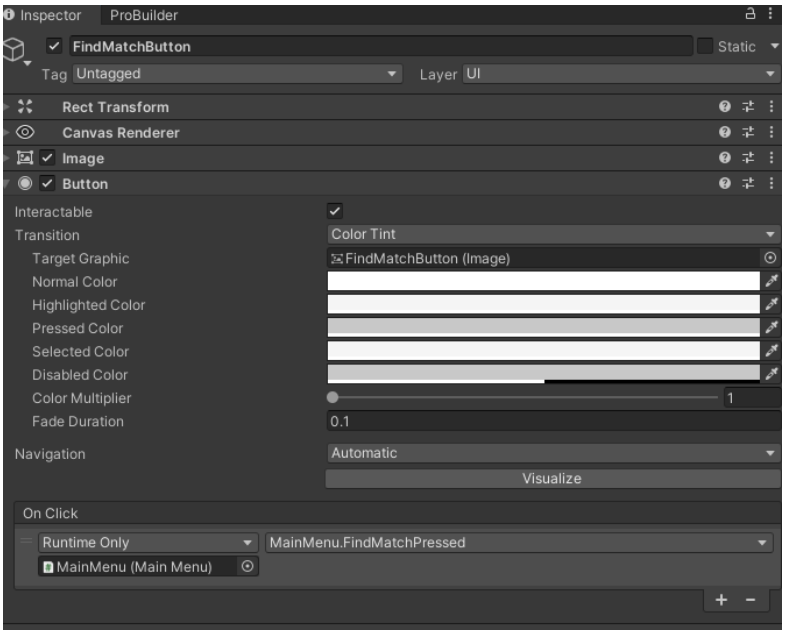
ΕΙΚΟΝΑ 49 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ ΤΗΣ ΣΚΗΝΗΣ MENU

Η σκηνή περιλαμβάνει το αντικείμενο MainMenu στο οποίο επισυνάπτεται το αρχείο κώδικα **MainMenu.cs** και αφορά κυρίως την αρχικοποίηση των απαραίτητων κλάσεων για την εκάστοτε λειτουργία που επιλέχθηκε. Στη συνέχεια συναντάται το αντικείμενο Canvas το οποίο περιλαμβάνει όλα τα επιμέρους αντικείμενα που απαρτίζουν το γραφικό περιβάλλον του menu. Το αντικείμενο **ButtonHolder:** Αποτελεί το γκρι υπόβαθρο, εντός του οποίου τοποθετούνται τα λοιπά αντικείμενα. Ύστερα ακολουθούν τα παρακάτω αντικείμενα:

- **FindMatchButton**: Το πάτημα του κουμπιού αυτού οδηγεί στην κλήση της συνάρτησης **MainMenu.FindMatchPressed()** η οποία προκαλεί με τη σειρά της την κλήση όλων των απαραίτητων διαδικασιών που χρειάζεται ένας παίκτης προκειμένου να συνδεθεί σε κάποιο διαθέσιμο server του παιχνιδιού και παρουσιάζεται στον Κώδικας 6.

```
public async void FindMatchPressed() {
    ClientSingleton.Instance.GameManager.MatchmakeAsync(OnMatchMade);
    findMatchButtonText.text = "Cancel";
    queueStatusText.text = "Searching...";
    timeInQueue = 0f;
    isMatchmaking = true;
    isBusy = true;
}
```

ΚΩΔΙΚΑΣ 6 - Η ΣΥΝΑΡΤΗΣΗ ΓΙΑ ΕΥΡΕΣΗ ΔΙΑΘΕΣΙΜΟΥ ΔΩΜΑΤΙΟΥ ΣΕ SERVER

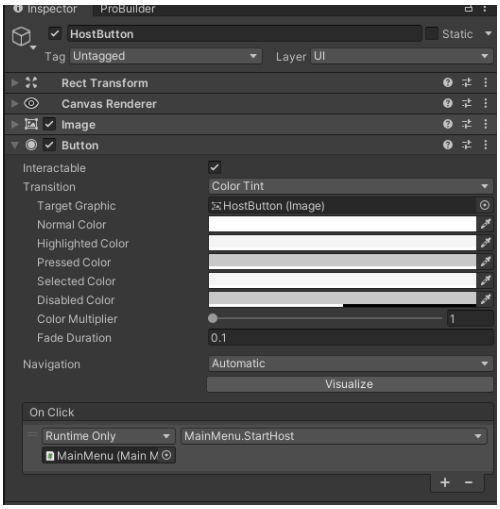


ΕΙΚΟΝΑ 50 - ΤΑ COMPONENTS ΤΟΥ ΚΟΥΜΠΙΟΥ FIND MATCH

- **HostButton**: Το πάτημα του κουμπιού αυτού οδηγεί στην κλήση της συνάρτησης MainMenu.StartHost() η οποία προκαλεί με τη σειρά της την κλήση όλων των απαραίτητων διαδικασιών που χρειάζεται ένας παίκτης προκειμένου να μετατραπεί σε οικοδεσπότη – παίκτη, και να μπορεί να δεχθεί συνδέσεις νέων παικτών, όπως φαίνεται στον Κώδικας 7.

```
public async void StartHost() {
    if (isBusy) { return; }
    isBusy = true;
    await HostSingleton.Instance.GameManager.StartHostAsync();
    isBusy = false;
```

ΚΩΔΙΚΑΣ 7 - Η ΣΥΝΑΡΤΗΣΗ STARTHOST

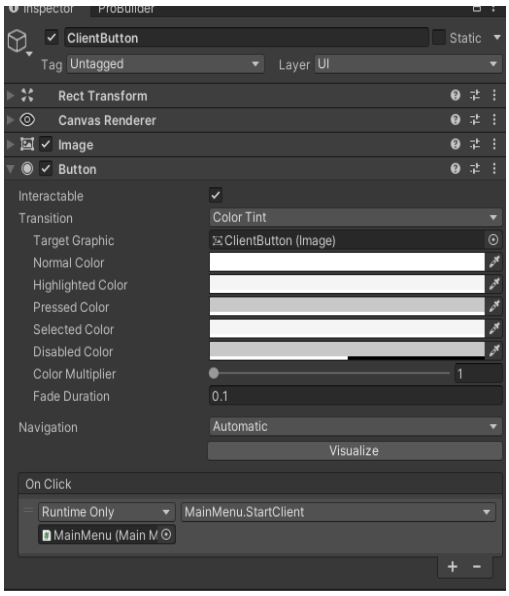


ΕΙΚΟΝΑ 51 - ΤΑ COMPONENTS ΤΟΥ ΚΟΥΜΠΙΟΥ HOSTBUTTON

- **ClientButton**: Το πάτημα του κουμπιού αυτού οδηγεί στην κλήση της συνάρτησης MainMenu.StartClient(),όπως φαίνεται παρακάτω (Κώδικας 8), η οποία ομοίως με παραπάνω, κινεί όλες τις απαραίτητες διαδικασίες προκειμένου ο χρήστης να μπορεί να συνδεθεί σε κάποιο δωμάτιο. Το δωμάτιο ωστόσο που θα συνδεθεί, ορίζεται από την τιμή που έχει εισαγάγει ο χρήστης στο πεδίο JoinCodeInputField, το οποίο φαίνεται στην οθόνη ως Join Code.

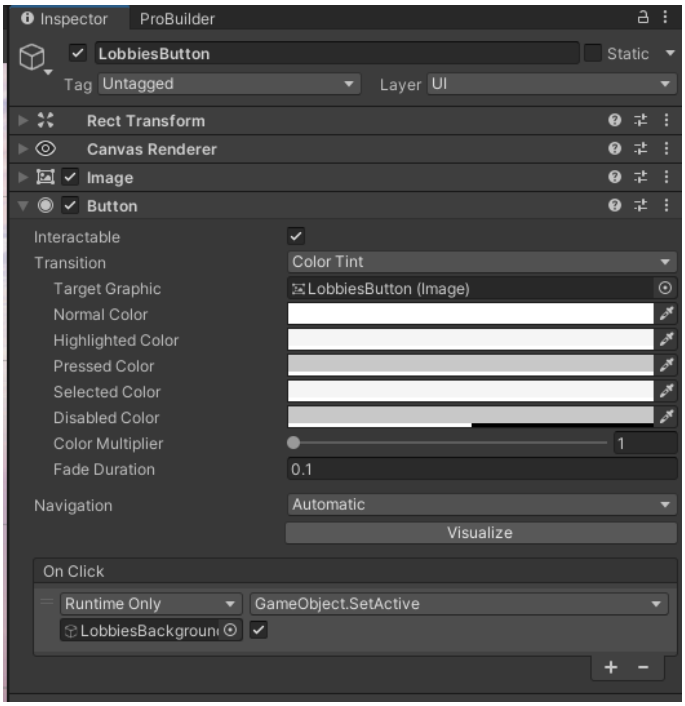
```
public async void StartClient() {
  if (isBusy) { return; }
  isBusy = true;
  await
  ClientSingleton.Instance.GameManager.StartClientAsync(joinCodeField
  .text);
  isBusy = false;
}
```

ΚΩΔΙΚΑΣ 8 - Η ΣΥΝΑΡΤΗΣΗ STARTCLIENT



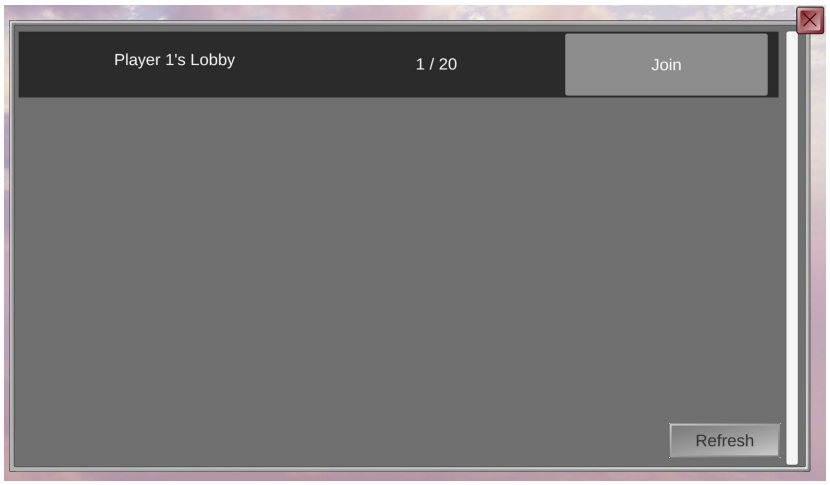
ΕΙΚΟΝΑ 52 - ΤΑ COMPONENTS ΤΟΥ CLIENTBUTTON

- **LobbiesButton**: Το πάτημα του κουμπιού Lobbies οδηγεί στην εμφάνιση ενός νέου παραθύρου, το οποίο θα προβάλλει όλα τα διαθέσιμα δωμάτια στα οποία θα μπορεί να συνδεθεί ο χρήστης. Όπως φαίνεται στην Εικόνα 48, υπάρχει το αντικείμενο LobbiesBackground, το οποίο αρχικά δεν εμφανίζεται στην οθόνη. Το πάτημα του LobbiesButton λοιπόν, θα προκαλέσει την εμφάνιση του LobbiesBackground μέσω της **GameObject.SetActive()**, όπως φαίνεται στην Εικόνα 53.



ΕΙΚΟΝΑ 53 - ΤΑ COMPONENTS ΤΟΥ LOBBIESBUTTON

Αν για παράδειγμα υπάρχει διαθέσιμο μόνο ένα δωμάτιο, τότε ο χρήστης θα αντικρύσει την Εικόνα 54.

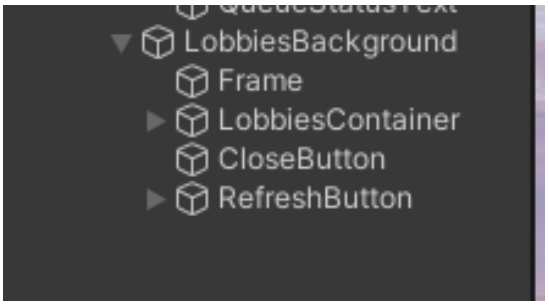


ΕΙΚΟΝΑ 54 - ΤΟ ΠΑΡΑΘΥΡΟ ΣΥΝΔΕΣΗΣ ΣΤΑ ΔΙΑΘΕΣΙΜΑ LOBBIES

- **JoinCodeInputField**: Όπως προαναφέρθηκε, αποτελεί το πεδίο στο οποίο ο χρήστης εισάγει τον μοναδικό αριθμό που χαρακτηρίζει ένα δωμάτιο προκειμένου να συνδεθεί σε αυτό. Ο χρήστης πρώτα πληκτρολογεί τον κωδικό στο πεδίο και ύστερα πατά το κουμπί Client ώστε να προσπαθήσει να συνδεθεί σε αυτό.
- **QueueTimerText**: Προβάλλει τον χρόνο που πέρασε από τη στιγμή που ο χρήστης πάτησε το κουμπί Find Match έως ότου βρέθηκε διαθέσιμη θέση σε κάποιον server.

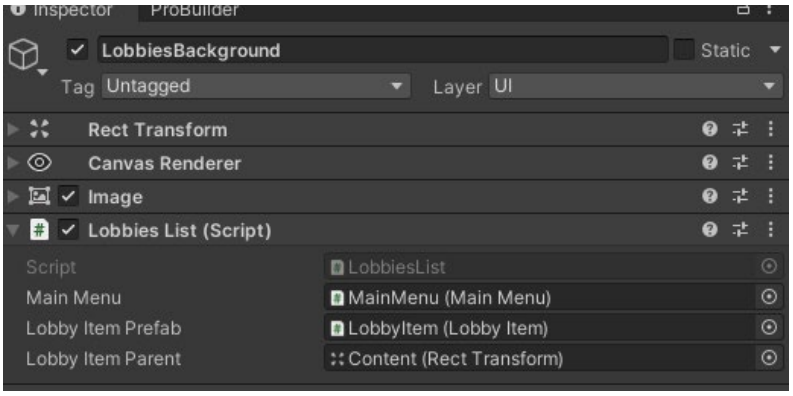
- **QueueStatusText**: Προβάλλει τα μηνύματα ενημέρωσης για την πορεία του αιτήματος του χρήστη κατά την προσπάθεια σύνδεσης του σε κάποιο server. Το συνηθέστερο είναι το μήνυμα ***Searching*** κατά τη διάρκεια αναζήτησης, ωστόσο στο πεδίο αυτό μπορεί να εμφανιστούν και μηνύματα λάθους σε περίπτωση που η διαδικασία αποτύχει ή ακυρωθεί.

Όπως προαναφέρθηκε, το πάτημα του κουμπιού Lobbies οδηγεί στην εμφάνιση ενός νέου αντικειμένου, του LobbiesBackground. Παρακάτω θα περιεγραφεί η δομή και η λειτουργία του.



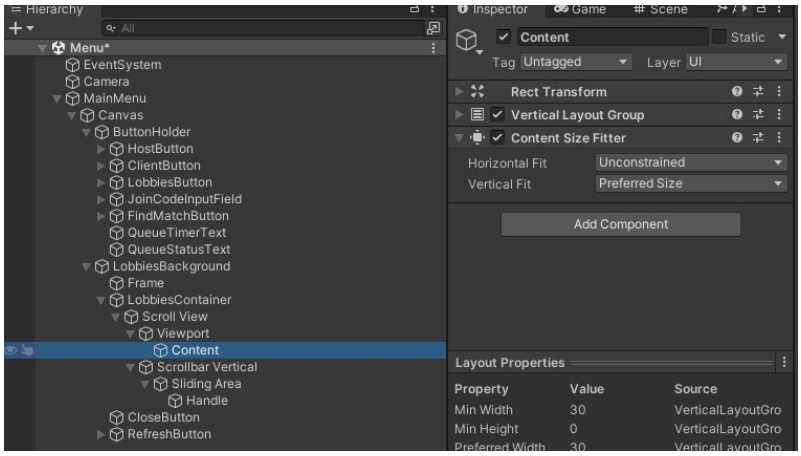
ΕΙΚΟΝΑ 55 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ ΤΟΥ LOBBIESBACKGROUND

Επιλέγοντας αρχικά το LobbiesBackground βλέπει κανείς πως διαθέτει το component – αρχείο κώδικα **LobbiesList.cs**. Το συγκεκριμένο script αναλαμβάνει τις διαδικασίες που απαιτούνται για την εμφάνιση, ανανέωση, διαγραφή δωματίων καθώς και τη δυνατότητα σύνδεσης σε αυτά.



ΕΙΚΟΝΑ 56 - ΤΑ COMPONENTS ΤΟΥ LOBBIESBACKGROUND

Αποτελείται επίσης από τα αντικείμενα Frame και LobbiesContainer, τα οποία αποτελούν τη βάση για την εμφάνιση νέων δωματίων καθώς και την μπάρα ολίσθησης στο δεξί μέρος του παραθύρου και η δομή του είναι αυτή της Εικόνα 57.



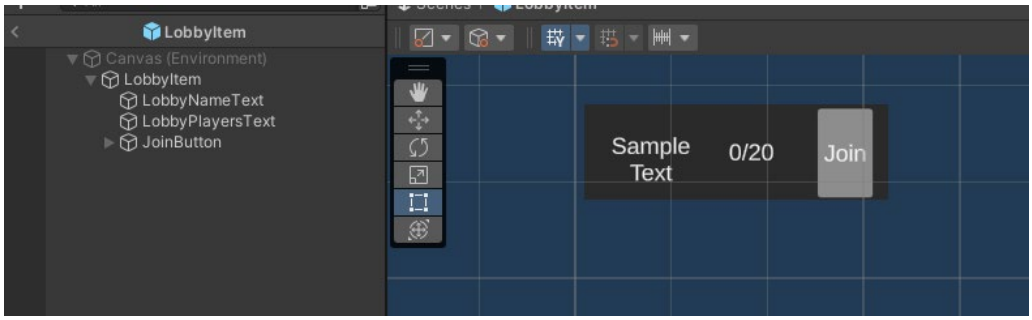
ΕΙΚΟΝΑ 57 - ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ CONTENT

Περιέχει επίσης 2 κουμπιά, τα Close και Refresh, τα οποία καλούν τις συναρτήσεις **GameObject.SetActive()** ώστε να μην είναι ορατό το αντικείμενο LobbiesBackground, και την **LobbiesList.RefreshList()**, για την ανανέωση των διαθέσιμων δωματίων, η οποία και παρατίθεται παρακάτω (Κώδικας 9). Στο συγκεκριμένο παράδειγμα, η **RefreshList**() δημιουργεί ένα ερώτημα (query), το οποίο αποστέλλει μέσω της **QueryLobbiesAsync**(). Η απάντηση που θα λάβει, θα περιλαμβάνει όλα τα διαθέσιμα δωμάτια στα οποία μπορεί ο χρήστης να συνδεθεί και τα οποία πληρούν τις προϋποθέσεις που έχουν οριστεί εντός της λίστας μεταβλητής options. Στην προκειμένη περίπτωση, οι συνθήκες που ελέγχονται είναι οι διαθέσιμες θέσεις να είναι περισσότερες από 0 και το δωμάτιο να μην είναι κλειδωμένο.

```
public async void RefreshList() {
        QueryLobbiesOptions options = new QueryLobbiesOptions();
        options.Count = 25;
        //Filters which lobbies will be displayed
        options.Filters = new List<QueryFilter>() {
        new QueryFilter(
            field: QueryFilter.FieldOptions.AvailableSlots,
            op: QueryFilter.OpOptions.GT, value: "0"),
        new QueryFilter(
            field: QueryFilter.FieldOptions.IsLocked,
            op: QueryFilter.OpOptions.EQ, value: "0")};
      QueryResponse lobbies = await
      Lobbies.Instance.QueryLobbiesAsync(options);
      foreach(Transform child in lobbyItemParent) {
            Destroy(child.gameObject);
      }
      foreach(Lobby lobby in lobbies.Results) {
            LobbyItem lobbyItem = Instantiate(lobbyItemPrefab,
            lobbyItemParent);
            lobbyItem.Initialise(this, lobby);
      }… }
```

ΚΩΔΙΚΑΣ 9 - Η ΣΥΝΑΡΤΗΣΗ REFRESHLIST()

Στον κώδικα της **RefreshList(),** για κάθε δωμάτιο της λίστας lobbies.Results, αρχικοποιείται και το αντίστοιχο αντικείμενο το οποίο θα απεικονίζεται στα διαθέσιμα προς σύνδεση δωμάτια. Το αντικείμενο αυτό ορίζεται μέσω της μεταβλητής lobbyItemPrefab και στην προκειμένη περίπτωση είναι το LobbyItem το οποίο βρίσκεται εντός του φακέλου Prefabs.



ΕΙΚΟΝΑ 58 - ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ LOBBYITEM

Το αντικείμενο αυτό, περιλαμβάνει δύο πεδία text, στα οποία θα αναγράφεται το όνομα του δωματίου, δηλαδή το όνομα του host, και ο αριθμός συνδεδεμένων παικτών, όπως επίσης και ένα κουμπί Join.

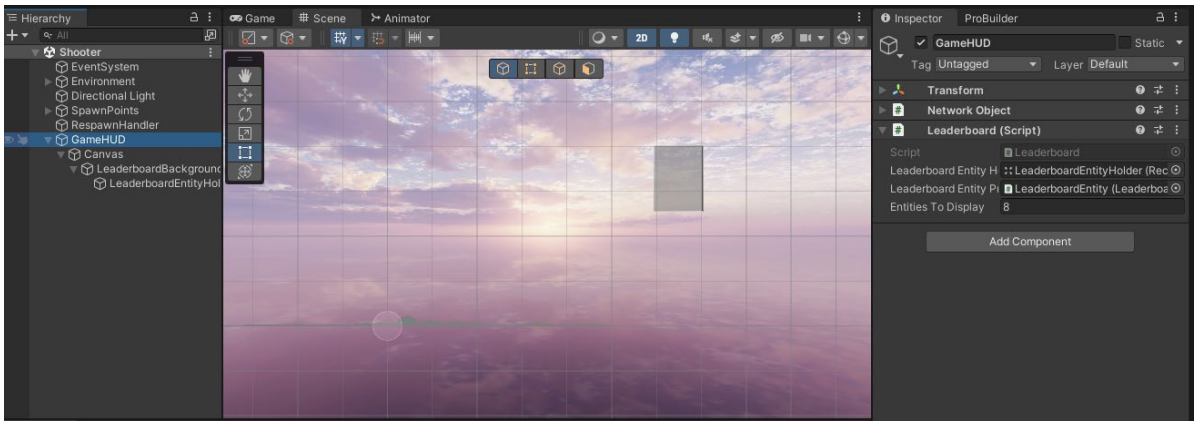
Το πάτημα του κουμπιού, καλεί τη συνάρτηση Join() του LobbyItem.cs δίνοντας σαν ορίσματα τα στοιχεία του συγκεκριμένου lobby. Στη συνέχεια, καλείται η **LobbiesList.JoinAsync(lobby)**, η οποία με τη σειρά της καλεί την **MainMenu.JoinAsync(lobby)**, η οποία και εκκινεί όλες τις απαραίτητες διαδικασίες προκειμένου ο χρήστης να συνδεθεί στο δωμάτιο που ζήτησε, όπως φαίνεται παρακάτω (Κώδικας 10).

```
public async void JoinAsync(Lobby lobby) {
      Lobby joiningLobby = await
      Lobbies.Instance.JoinLobbyByIdAsync(lobby.Id);
      string joinCode = joiningLobby.Data["JoinCode"].Value;
      await
      ClientSingleton.Instance.GameManager.StartClientAsync(joinCode);}
}
```

ΚΩΔΙΚΑΣ 10 - ΣΥΝΔΕΣΗ ΣΕ LOBBY ΜΕΣΩ ΤΗΣ JOINASYNC

3.4.3 Σκηνή Shooter

Η τελευταία σκηνή την οποία συναντά ο χρήστης και πρόκειται ουσιαστικά για το κυρίως παιχνίδι είναι η σκηνή Shooter. Στην Εικόνα 59, φαίνεται πως η σκηνή περιλαμβάνει το αντικείμενο GameHUD, το οποίο έχει ως component το αρχείο κώδικα Leaderboard.cs. Το GameHUD διαθέτει ως αντικείμενο – παιδί ένα αντικείμενο Canvas, το οποίο με τη σειρά του αποτελείται από τα αντικείμενα LeaderboardBackground και LeaderboardEntityHolder.

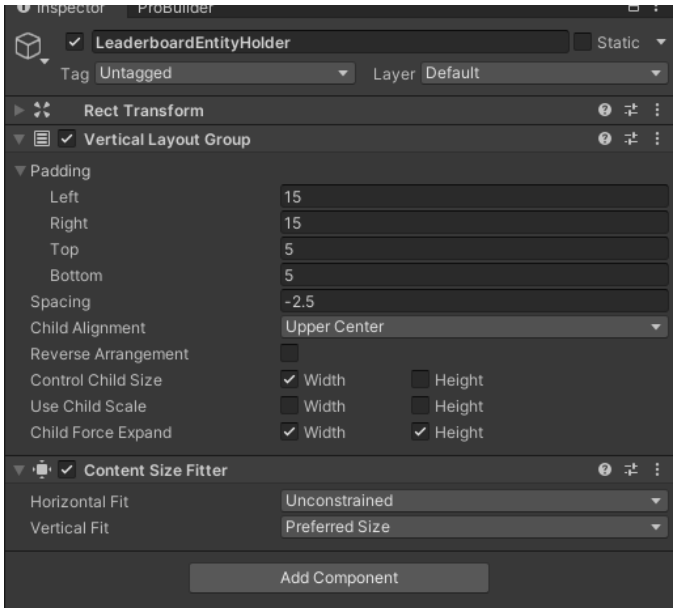


ΕΙΚΟΝΑ 59 - ΑΝΤΙΚΕΙΜΕΝΑ ΣΚΗΝΗΣ SHOOTER

Ο συνδυασμός των δύο, δημιουργεί τον πίνακα κατάταξης του παιχνιδιού, στον οποίο θα φαίνονται τα ονόματα και οι πόντοι όλων των παικτών, αντίστοιχο της Εικόνα 60.

ΕΙΚΟΝΑ 60 - ΠΙΝΑΚΑΣ ΚΑΤΑΤΑΞΗΣ ΠΑΙΧΝΙΔΙΟΥ

Το LeaderboardBackground αποτελεί το φόντο του πίνακα μαζί με το πλαίσιο του. Το LeaderboardEntityHolder αποτελεί τη δομή εντός της οποίας θα τοποθετούνται τα στοιχεία των παικτών. Αυτό επιτυγχάνεται επισυνάπτοντάς του τα components Vertical Layout Group και Contect Size Fitter με τις ρυθμίσεις της Εικόνα 61. Τα δύο αυτά components είναι απαραίτητα προκειμένου ο πίνακας κατάταξης να μπορεί να αποτελείται από τα επιμέρους LeaderboardEntities και να μπορεί να εναλλάσσει τη σειρά με την οποία αυτά εμφανίζονται στους παίκτες.



ΕΙΚΟΝΑ 61 - ΤΑ COMPONENTS ΤΟΥ LEADERBOARDENTITYHOLDER

Το component Leaderboard.cs είναι υπεύθυνο για όλες τις ενέργειες που απαιτούνται για την ομαλή λειτουργία του πίνακα κατάταξης, όπως είναι η δημιουργία νέων θέσεων όταν συνδέεται ένας παίκτης ή η ανανέωση των πόντων των παικτών, και οι οποίες θα περιγραφούν παρακάτω. Αρχικά κατά τη δημιουργία του πίνακα κατάταξης, καλούνται οι συναρτήσεις Awake() και OnNetworkSpawn() που ακολουθούν. Αρχικά δημιουργείται μια NetworkList από αντικείμενα LeaderboardEntities, καθεμιά από τις οποίες αποτελεί μια ξεχωριστή εγγραφή του πίνακα κατάταξης. Κάθε τέτοια εγγραφή, θα περιλαμβάνει το όνομα και τους πόντους του κάθε παίκτη. Εντός της συνάρτησης **OnNetworkSpawn()**, ο κώδικας που θα εκτελεστεί χρειάζεται να είναι διαφορετικός στην περίπτωση εκτέλεσής του σε server από ότι σε κάποιον client. Σε client, θα πρέπει να προστεθούν αρχικά όλες οι υπάρχουσες εγγραφές, καθώς και ο client να ρυθμιστεί ώστε να ακούει στα αντίστοιχα event ώστε να ενημερώνει τον πίνακα του κατάλληλα. Σε περίπτωση εκτέλεσης σε server, χρειάζεται ο server να ελέγξει πόσοι χρήστες έχουν ήδη συνδεθεί και να εμφανίσει τα αντίστοιχα LeaderboardEntities. Πρέπει επίσης, μέσω των **FirstPersonController.OnPlayerSpawned += HandlePlayerSpawned** και **FirstPersonController.OnPlayerDespawned += HandlePlayerDespawned**, ο server να «ακούει» στα αντίστοιχα events, και κατά την εκπομπή αυτών, να καλεί τις κατάλληλες συναρτήσεις, όπως φαίνεται στον Κώδικας 11.

```
private NetworkList<LeaderboardEntityState> leaderboardEntities;
private List<LeaderboardEntityDisplay> entityDisplays = new
List<LeaderboardEntityDisplay>();

private void Awake() {
  leaderboardEntities = new NetworkList<LeaderboardEntityState>(); }

public override void OnNetworkSpawn() {
 if (IsClient) {
  leaderboardEntities.OnListChanged += HandleLeaderboardEntitiesChanged;
   foreach (LeaderboardEntityState entity in leaderboardEntities) {
      HandleLeaderboardEntitiesChanged(new
      NetworkListEvent<LeaderboardEntityState>{
        Type = NetworkListEvent<LeaderboardEntityState>.EventType.Add,
        Value = entity });
 }}
if (IsServer) {
    FirstPersonController[] players =
    FindObjectsOfType<FirstPersonController>();
    foreach (FirstPersonController player in players) {
                HandlePlayerSpawned(player);}
    FirstPersonController.OnPlayerSpawned += HandlePlayerSpawned;
    FirstPersonController.OnPlayerDespawned += HandlePlayerDespawned;
 }}
```

ΚΩΔΙΚΑΣ 11 - Η ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΟΥ ΠΙΝΑΚΑ ΚΑΤΑΤΑΞΗΣ

Στη συνέχεια ακολουθεί η συνάρτηση **HandlePlayerSpawned** (Κώδικας 12) η οποία δημιουργεί ένα νέο αντικείμενο τύπου leaderboardEntities όταν ο παίκτης συνδέεται στο παιχνίδι, ενώ όμοια λειτουργεί και η **HandlePlayerDespawned()**.

```
private void HandlePlayerSpawned(FirstPersonController player) {
    leaderboardEntities.Add(new LeaderboardEntityState {
    ClientId = player.OwnerClientId,
    PlayerName = player.PlayerName.Value,
    Points = 0});;
    player.CurrentPoints.OnValueChanged += (oldPoints, newPoints) =>
    HandlePointsChanged(player.OwnerClientId, newPoints);
 }
```

ΚΩΔΙΚΑΣ 12 - ΔΗΜΙΟΥΡΓΙΑ ΘΕΣΕΩΝ ΣΤΟΝ ΠΙΝΑΚΑ ΚΑΤΑΤΑΞΗΣ ΓΙΑ ΚΑΘΕ ΝΕΟ ΠΑΙΚΤΗ

Παρόμοια λογική ακολουθεί και η συνάρτηση HandlePointsChanged (Κώδικας 13), η οποία όταν καλείται εντοπίζει το ζητούμενο leaderboardEntity και προχωρά στην ενημέρωση των πόντων του. Η **HandlePointsChanged** καλείται αυτόματα, όταν μεταβάλλονται οι πόντοι ενός παίκτη.

```
private void HandlePointsChanged(ulong clientId, int newPoints) {
    for (int i = 0; i < leaderboardEntities.Count; i++) {
        if (leaderboardEntities[i].ClientId != clientId) {
           continue;}
        leaderboardEntities[i] = new LeaderboardEntityState {
           ClientId = leaderboardEntities[i].ClientId,
           PlayerName = leaderboardEntities[i].PlayerName,
           Points = newPoints};
        return;}}
```

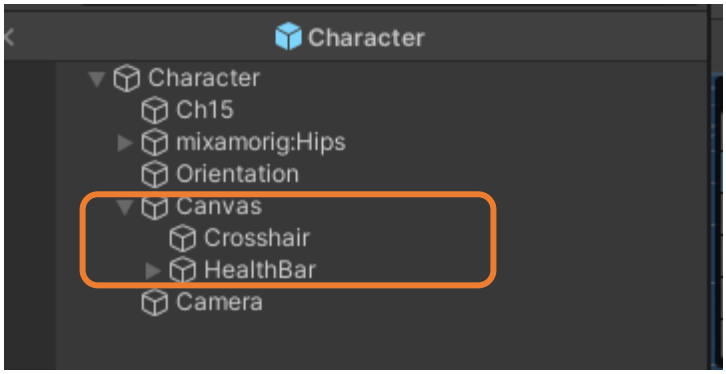
ΚΩΔΙΚΑΣ 13 - ΕΝΗΜΕΡΩΣΗ ΠΟΝΤΩΝ ΠΑΙΚΤΩΝ

Στην περίπτωση των clients ωστόσο, προκειμένου κάθε παίκτης να ενημερώνεται διαρκώς για τους πόντους τόσο του ίδιου όσο και των αντιπάλων του, τα πράγματα είναι πιο πολύπλοκα. Θα πρέπει ο client ανάλογα τον τύπου του event που λαμβάνει, να το εντοπίζει και να προχωρά στις αντίστοιχες ενέργειες. Στην προκειμένη περίπτωση, αυτό επιτυγχάνεται μέσω της συνάρτησης **HandleLeaderboardEntitiesChanged(changeEvent),** η οποία αρχικά εντοπίζει και ενημερώνει το αντίστοιχο leaderboardEntity, και στη συνέχεια προχωρά σε ενημέρωση του πίνακα, ώστε κάθε φορά να εμφανίζεται ο κατάλληλος αριθμός εγγραφών στον πίνακα (Κώδικας 14).

```
private void
HandleLeaderboardEntitiesChanged(NetworkListEvent<LeaderboardEntityState>
changeEvent) {
  switch (changeEvent.Type) {
    case NetworkListEvent<LeaderboardEntityState>.EventType.Add:
       if (!entityDisplays.Any(x => x.ClientId ==
changeEvent.Value.ClientId)) {
            LeaderboardEntityDisplay leaderboardEntity=
            Instantiate(leaderboardEntityPrefab,
            leaderboardEntityHolder);
            leaderboardEntity.Initialise(changeEvent.Value.ClientId,
            changeEvent.Value.PlayerName, changeEvent.Value.Points);
              entityDisplays.Add(leaderboardEntity); }
        break;
     case NetworkListEvent<LeaderboardEntityState>.EventType.Remove:
            LeaderboardEntityDisplay displayToRemove =
            entityDisplays.FirstOrDefault(x => x.ClientId ==
            changeEvent.Value.ClientId);
              if (displayToRemove != null) {
                 displayToRemove.transform.SetParent(null);
                 Destroy(displayToRemove.gameObject);
                 entityDisplays.Remove(displayToRemove);
              }
              break;
     case NetworkListEvent<LeaderboardEntityState>.EventType.Value:
            LeaderboardEntityDisplay displayToUpdate =
            entityDisplays.FirstOrDefault(x => x.ClientId ==
            changeEvent.Value.ClientId);
              if (displayToUpdate != null) {
                displayToUpdate.UpdatePoints(changeEvent.Value.Points);
              }
              break;
       }
       entityDisplays.Sort((x, y) => y.Points.CompareTo(x.Points));
       for (int i = 0; i < entityDisplays.Count; i++) {
           entityDisplays[i].transform.SetSiblingIndex(i);
           entityDisplays[i].UpdateText();
           bool shouldShow = i <= entitiesToDisplay - 1;
           entityDisplays[i].gameObject.SetActive(shouldShow);
       }
     …}
```

ΚΩΔΙΚΑΣ 14 - ΕΝΗΜΕΡΩΣΗ ΠΙΝΑΚΑ ΓΙΑ ΝΕΑ ΓΕΓΟΝΟΤΑ ΓΙΑ ΤΟΥΣ CLIENTS

Εκτός της παραπάνω ο διεπαφής, ο χρήστης θα βλέπει μερικά ακόμη στοιχεία στην οθόνη του. Τα στοιχεία αυτά, έχουν προστεθεί στο prefab του αντικειμένου το οποίο χειρίζεται ο κάθε παίκτης και συγκεκριμένα είναι το Crosshair, το στόχαστρο δηλαδή μέσω του οποίου ο παίκτης επιλέγει την κατεύθυνση των σφαιρών του, και το HealthBar, μια μπάρα δηλαδή που απεικονίζει την τρέχουσα ζωή του παίκτη και η οποία ανανεώνεται μέσω των scripts των σφαιρών.



ΕΙΚΟΝΑ 62 - ΤΑ ΑΝΤΙΚΕΙΜΕΝΑ CROSSHAIR ΚΑΙ HEALTHBAR

3.5 Multiplayer

Το Netcode for GameObjects είναι ένα σύνολο τεχνικών και βιβλιοθηκών προγραμματισμού που σχεδιάστηκαν για τη δημιουργία παιχνιδιών που υποστηρίζουν δικτυακή λειτουργία. Το Netcode for GameObjects είναι μια προσπάθεια για την ευκολότερη διαχείριση και συγχρονισμό των game objects (αντικειμένων του παιχνιδιού) σε ένα περιβάλλον δικτύου.

Κατά τη δημιουργία παιχνιδιών που λειτουργούν διαδικτυακά, υπάρχουν προκλήσεις που πρέπει να αντιμετωπιστούν, όπως η μετάδοση δεδομένων μεταξύ παικτών, η εξομοίωση της καθυστέρησης δικτύου (network latency), η αντιμετώπιση των προβλημάτων που προκύπτουν από την ασυμμετρία του δικτύου κ.ά.

Το Netcode for GameObjects προσφέρει λύσεις για αυτά τα προβλήματα μέσω της βελτιστοποίησης της διαχείρισης των game objects κατά τη διάρκεια του δικτυακού συγχρονισμού. Κάποια από τα χαρακτηριστικά που περιλαμβάνει το Netcode for GameObjects είναι:

1. **Πρόβλεψη Καθυστέρησης (Lag Compensation):** Τεχνικές για την εξομοίωση της καθυστέρησης δικτύου, προκειμένου να διατηρείται η ομαλή εμπειρία του παίκτη.
2. **Αντιμετώπιση Ασυμμετρίας Δικτύου:** Διαχείριση των διαφορών στην ασυμμετρία του δικτύου μεταξύ των παικτών.

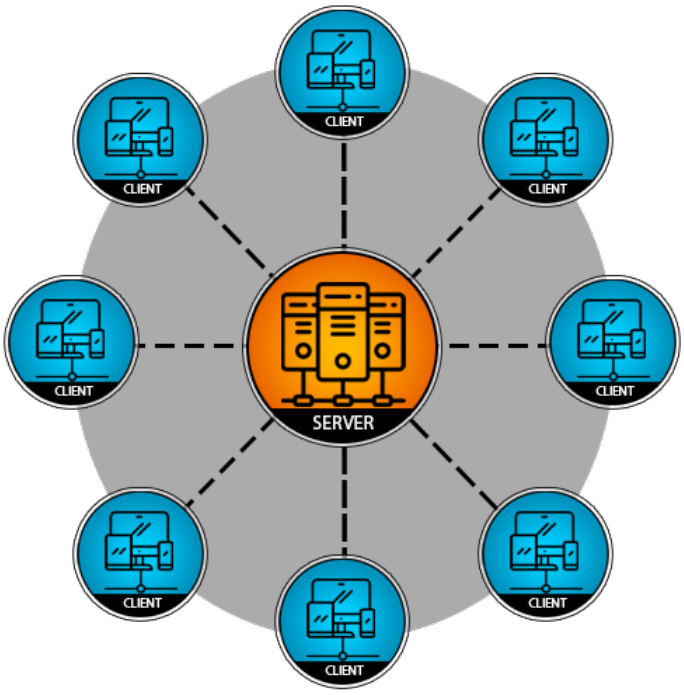
3. **Συγχρονισμός Δεδομένων:** Μηχανισμοί για τον αποτελεσματικό συγχρονισμό των δεδομένων των αντικειμένων μεταξύ των παικτών.

4. **Βελτιστοποίηση Απόδοσης:** Βελτιστοποίηση της απόδοσης του δικτύου για ελαχιστοποίηση της καθυστέρησης.

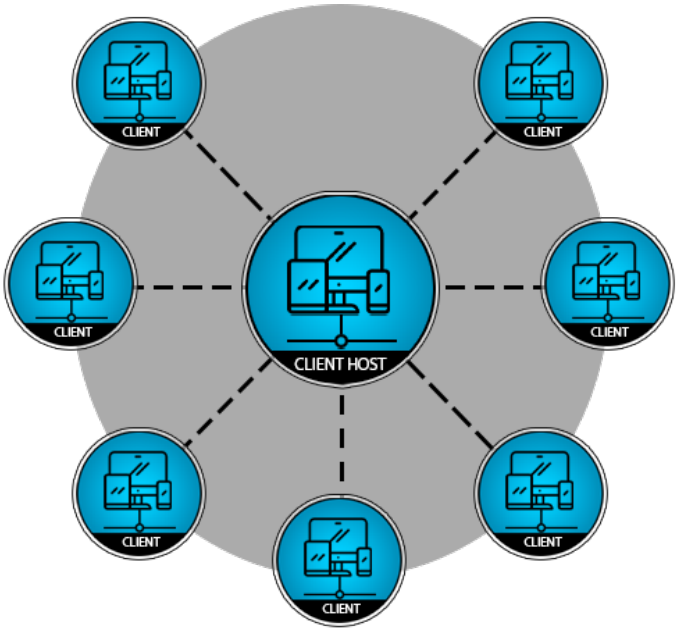
Το Netcode for GameObjects βοηθάει τους προγραμματιστές να υλοποιήσουν δικτυακή λειτουργία σε παιχνίδια με σχετική ευκολία, ενεργώντας ως εργαλείο για την αντιμετώπιση των δυσκολιών που προκύπτουν κατά τη διάρκεια του δικτυακού παιχνιδιού.

3.5.1 Dedicated Server και Host based παιχνίδια

Τα παιχνίδια μπορούν να λειτουργούν με διάφορους τρόπους όσον αφορά τον τρόπο που διαχειρίζονται τις συνδέσεις των παικτών και την επεξεργασία των γεγονότων στο παιχνίδι. Οι δύο κυριότεροι τύποι αρχιτεκτονικής για την διασύνδεση παικτών είναι οι Host-based (βασισμένοι σε κεντρικό υπολογιστή) και οι Dedicated Server-based (βασισμένοι σε αφιερωμένο διακομιστή). Ας δούμε τις βασικές διαφορές μεταξύ τους:



Εικονα 63 - Τοπολογια με dedicated server



Εικονα 64 - Τοπολογια με Host

1. **Υλοποίηση Host based:**
    - **Υπολογιστής του παίκτη ως Host:** Σε ένα Host-based σύστημα, ο ένας από τους παίκτες επιλέγεται ως "κεντρικός" υπολογιστής, ο οποίος είναι υπεύθυνος για τη διαχείριση των συνδέσεων και των γεγονότων του παιχνιδιού.
    - **Περιορισμένη Διαθεσιμότητα Παιχνιδιού:** Εάν ο host αποσυνδεθεί, το παιχνίδι μπορεί να διακοπεί έως ότου επιλεχθεί νέος host.
2. **Υλοποίηση Dedicated Server based:**
    - **Αφιερωμένος Διακομιστής:** Σε ένα σύστημα με Dedicated Server, χρησιμοποιείται ένας αφιερωμένος διακομιστής που διαχειρίζεται τις συνδέσεις και τα γεγονότα για όλους τους συνδεδεμένους παίκτες.
    - **Αυξημένη Σταθερότητα:** Το παιχνίδι συνεχίζεται ακόμη και αν αποσυνδεθεί κάποιος παίκτης.
3. **Απόδοση και Καθυστέρηση:**
    - **Host-based:** Η απόδοση μπορεί να επηρεαστεί από την ασυμμετρία της σύνδεσης του host και την επεξεργαστική δικτυακή ισχύ του.
    - **Dedicated Server-based:** Η απόδοση είναι συνήθως καλύτερη, καθώς ο διακομιστής είναι σχεδιασμένος για να διαχειρίζεται πολλαπλούς παίκτες.

4. **Ασφάλεια:**

    - **Host-based:** Υπάρχει μεγαλύτερος κίνδυνος για παραβιάσεις ασφάλειας, καθώς ένας παίκτης μπορεί να έχει πρόσβαση σε δεδομένα άλλων παικτών.
    - **Dedicated Server-based:** Ο αφιερωμένος διακομιστής παρέχει περισσότερα επίπεδα ασφάλειας, καθώς οι λειτουργίες είναι συγκεντρωμένες σε έναν ελεγχόμενο χώρο.

5. **Δυνατότητες Ελέγχου και Επέκταση:**

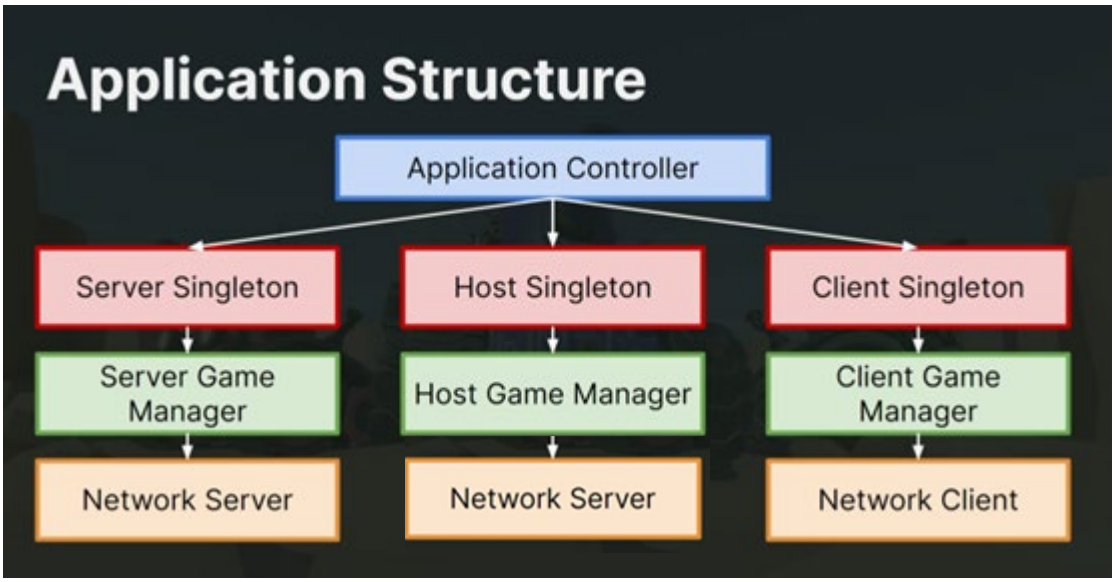
    - **Host-based:** Ο παίκτης που επιλέγεται ως host έχει τον έλεγχο και οι δυνατότητες επέκτασης είναι περιορισμένες.
    - **Dedicated Server-based:** Υπάρχουν περισσότερες δυνατότητες για ελέγχους, παραμετροποίηση και επέκταση, καθώς ο server μπορεί να προσαρμοστεί σύμφωνα με τις ανάγκες του παιχνιδιού.

Η επιλογή μεταξύ Host-based και Dedicated Server-based εξαρτάται συχνά από τις απαιτήσεις του συγκεκριμένου παιχνιδιού, τον τύπο του και τις προτιμήσεις των δημιουργών του.

Γενικά, σε παιχνίδια γρήγορου ρυθμού και εναλλαγών, όπως τα παιχνίδια shooter, η dedicated server υλοποίηση προσφέρει καλύτερη εμπειρία παιχνιδιού για τους παίκτες εξαλείφοντας πολλά από τα ζητήματα που μπορεί να προκύψουν με το Host-based σύστημα.

1. **Χαμηλότερη Καθυστέρηση (Lag):** Σε ένα dedicated server, ο διακομιστής χειρίζεται την κατανομή των δεδομένων και τον συγχρονισμό των ενεργειών των παικτών. Αυτό μειώνει την καθυστέρηση (lag) μεταξύ της ενέργειας του παίκτη και της αντίδρασης του παιχνιδιού.
2. **Αποφυγή του Host Advantage:** Σε Host-based συστήματα, ο παίκτης που είναι ο host έχει συνήθως πλεονέκτημα λόγω χαμηλότερης καθυστέρησης. Σε dedicated server, όλοι οι παίκτες έχουν ισότιμη εμπειρία.
3. **Δυνατότητα Επέκτασης και Ενημερώσεων:** Ο dedicated server επιτρέπει ευκολότερη ενσωμάτωση ενημερώσεων και επεκτάσεων στο παιχνίδι, καθώς οι αλλαγές μπορούν να εφαρμοστούν στον διακομιστή χωρίς την ανάγκη για αναβάθμιση σε κάθε παίκτη ξεχωριστά.

3.5.2 Δομή multiplayer εφαρμογής



ΕΙΚΟΝΑ 65 - ΔΙΚΤΥΑΚΗ ΔΟΜΗ ΕΦΑΡΜΟΓΗΣ

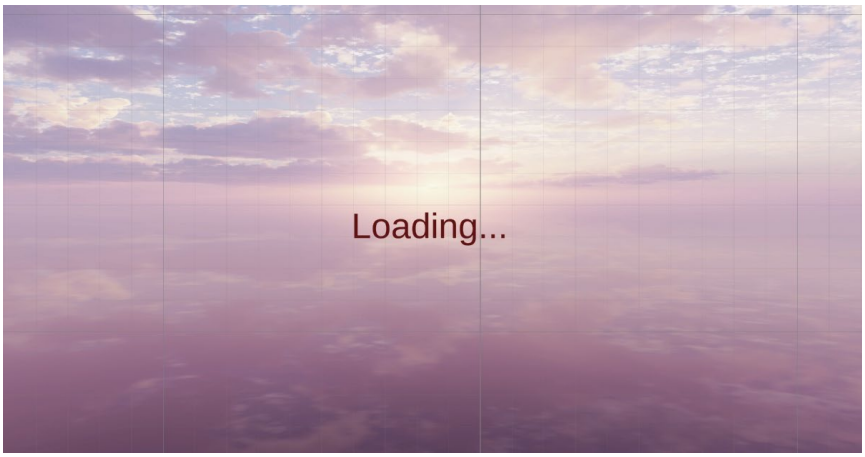
Στην Εικόνα 65, απεικονίζεται η δομή των δικτυακών λειτουργιών που προσφέρει το παιχνίδι. Όπως έχει προαναφερθεί, το συγκεκριμένο παιχνίδι παρέχει τη δυνατότητα επιλογής της διαδικτυακής αρχιτεκτονικής στην οποία θα βασιστούν οι παίκτες.

Βάση όλων αυτών, είναι η κλάση ApplicationController, η οποία είναι υπεύθυνη ουσιαστικά για τον διαχωρισμό των αρχιτεκτονικών, αναγνωρίζοντας κατά την εκκίνηση του παιχνιδιού, αν αυτό εκτελείται σε κάποιον dedicated server ή στον υπολογιστή κάποιου παίκτη.

Στην πρώτη περίπτωση, η κλάση Application Controller θα δημιουργήσει τις κλάσεις που θα εκτελέσει ο server και συγκεκριμένα τις Server Singleton, Server Game Manager, Network Server.

Στην δεύτερη περίπτωση, η κλάση Application Controller θα δημιουργήσει τις κλάσεις που αφορούν τους Host και Client, προκειμένου οι παίκτες να έχουν τη δυνατότητα και να γίνουν οι ίδιοι host ενός παιχνιδιού στο οποίο θα μπορούν να συνδεθούν άλλοι παίκτες αλλά και να συνδέονται οι ίδιοι σε άλλα διαθέσιμα δωμάτια.

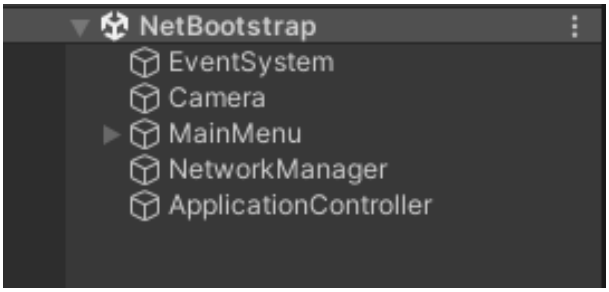
Μόλις ο χρήστης εισάγει το όνομα του, θα συναντήσει την Εικόνα 66.



Εικονα 66 - Η σκηνη NetBootstrap

Στα περισσότερα παιχνίδια πολλαπλών παικτών, οι χρήστες συναντούν αρκετές φορές και σε διάφορα μέρη του παιχνιδιού είτε με τη μορφή ξεχωριστών σκηνών, όπως εδώ, είτε με τη μορφή ειδοποιήσεων – μηνυμάτων, ενδείξεις φόρτωσης (loading), όπως η παραπάνω. Οι ενδείξεις αυτές, υποδεικνύουν την εκτέλεση κάποιων λειτουργιών στο παρασκήνιο του παιχνιδιού, οι οποίες συνήθως αφορούν την ρύθμιση κάποιων δικτυακών παραμέτρων, απαραίτητων για την ομαλή λειτουργία του παιχνιδιού. Οι λειτουργίες αυτές μπορεί να γίνουν χρονοβόρες, για αυτό και γίνονται στο παρασκήνιο, ενώ οι χρήστες βλέπουν το αντίστοιχο μήνυμα.

Στην παρούσα εργασία, η ανωτέρω λειτουργία πραγματοποιείται μέσω μιας νέας σκηνής, της NetBootstrap, η οποία περιλαμβάνει τα παρακάτω αντικείμενα (Εικόνα 67).



Εικονα 67 - Τα αντικειμενα της σκηνης NetBootstrap

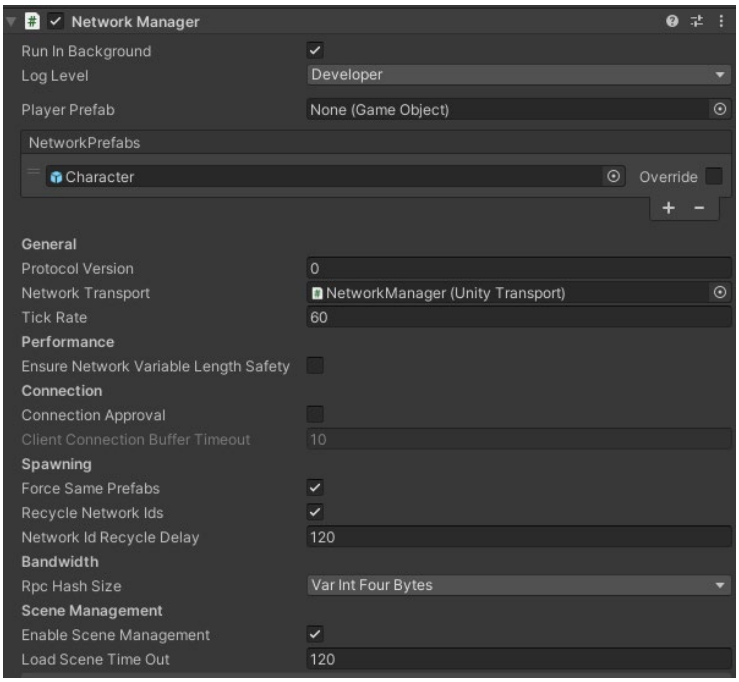
3.5.3 Networking Components

NetworkManager

Το αντικείμενο NetworkManager αποτελεί τη βάση οποιασδήποτε διαδικτυακής λειτουργίας με το Unity Netcode. Η NetworkManager, αποτελεί μια κλάση βασισμένη στο μοτίβο Singleton, επιτρέποντας την ύπαρξη μιας μοναδικής εκδοχής της κλάσης η οποία παρέχει ένα κεντρικό σημείο πρόσβασης σε αυτήν. Η κλάση αυτή μπορεί να θεωρηθεί ο κεντρικός πίνακας ελέγχου

των διαδικτυακών λειτουργιών του παιχνιδιού καθώς περιέχει όλες τις απαραίτητες διαδικτυακές ρυθμίσεις ενός παιχνιδιού.

Παρακάτω παρουσιάζονται οι διαθέσιμες ρυθμίσεις του Network Manager.



ΕΙΚΟΝΑ 68 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΤΟΥ NETWORK MANAGER

- **LogLevel**: Ορίζει το είδος των δικτυακών logs.
- **PlayerPrefab**: Η βασικότερη όλων, και η μοναδική που έχει τροποποιηθεί στην συγκεκριμένη περίπτωση, είναι η NetworkPrefabs ή η PlayerPrefab. Στις θέσεις αυτές, πρέπει να τοποθετηθεί το αντικείμενο το οποίο θα αποτελεί το χαρακτήρα τον οποίο θα χειρίζονται οι παίκτες του διαδικτυακού μας παιχνιδιού. Για αυτό το λόγο, έχει συρθεί στη θέση του NetworkPrefabs το αντικείμενο Character που εισαγάγαμε νωρίτερα. Αν το παιχνίδι παρείχε τη δυνατότητα εναλλαγής μεταξύ διαφόρων χαρακτήρων, τότε θα έπρεπε να προστεθούν αντιστοίχως.
- **Protocol Version**: Ο ορισμός αυτής της τιμής βοηθά στη διάκριση μεταξύ των διαφόρων builds προκειμένου να μην έρχονται σε σύγκρουση τα νέα με τα παλαιά.
- **Network Transport**: Ορίζει το που θα βρίσκονται ορισμένες δικτυακές ρυθμίσεις.
- **Tick Rate**: Ελέγχει τον ρυθμό ανανέωσης, δηλαδή πόσο συχνά το Netcode στέλνει και δέχεται δεδομένα. Η τιμή του μεταφράζεται σε ticks per second, οπότε για παράδειγμα η τιμή 60 συνεπάγεται 60 ticks (ανανεώσεις) ανά δευτερόλεπτο.

- **Ensure Network Variable Length Safety**: Αν επιλεχθεί, το Netcode θα διαφυλάξει ότι ο κώδικας του χρήστη δεν θα μπορεί να γράψει πέρα από τα όρια μιας NetworkVariable, μιας διαδικτυακής μεταβλητής την οποία μοιράζονται οι χρήστες.
- **Connection Approval**: Επιτρέπει την έγκριση συνδέσεων (connection approval) όταν επιλέγεται σε συνδυασμό με την χρήση της NetworkManager.ConnectionApprovalCallback εντός του κώδικα.
- **Client Connection Buffer Timeout**: Ορίζει το χρονικό διάστημα που χρειάζεται να περάσει προκειμένου ένας client να ολοκληρώσει τη διαδικασία έγκρισης σύνδεσης. Αν υπερβεί το χρόνο αυτό, τότε ο χρήστης θα αποσυνδεθεί.
- **Force Same Prefabs**: Αν επιλεχθεί, τότε ελέγχει ότι όλοι οι clients που συνδέονται στο παιχνίδι έχουν τα ίδια ακριβώς δικτυακά prefabs όπως και ο server. Αν δεν επιλεχθεί, τότε θα αγνοηθούν οποιεσδήποτε διαφορές.
- **Recycle Network Ids**: Αν επιλεχθεί, επιτρέπει την επαναχρησιμοποίηση των NetworkObject.NetworkObjectIds που έχουν ανατεθεί ήδη, μετά τον ορισμένο χρόνο, δηλαδή της Network Id Recycle Delay που ακολουθεί.
- **Enable Scene Management**: Αν επιλεχθεί, τότε το Netcode θα χειριστεί τη διαχείριση των σκηνών και τον συγχρονισμό των clients. Αν δεν επιλεχθεί, τότε ο προγραμματιστής θα πρέπει να χτίσει τη δική του λογική συγχρονισμού.
- **Load Scene Time Out**: Αν έχει επιλεχθεί το Enable Scene Management, η τιμή αυτή ορίζει το χρονικό διάστημα στο οποίο ο NetworkSceneManager θα περιμένει καθώς η σκηνή φορτώνεται ασύγχρονα, προτού θεωρήσει ότι απέτυχε η φόρτωσή της.

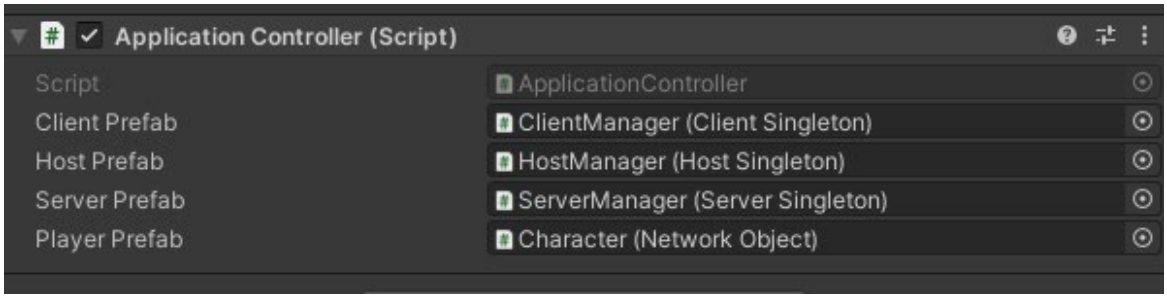
NetworkBehaviour: Αποτελεί μια abstract κλάση που κληρονομεί από την MonoBehaviour και χρησιμοποιείται κυρίως για δημιουργία διαδικτυακής λογικής. Χρησιμοποιεί NetworkVariables και RPC για να συγχρονίσει την κατάσταση των παικτών και να στέλνει μηνύματα μέσω του διαδικτύου. Λειτουργεί σε συνδυασμό με το NetworkObject.

NetworkObject: Αποτελεί απαραίτητο component για οποιοδήποτε αντικείμενο θα βρίσκεται στη σκηνή και χρειάζεται να συγχρονίζεται και να ενημερώνεται η κατάσταση του μεταξύ των παικτών.

NetworkAnimator: Το component αποτελεί τη βάση για τον συγχρονισμό των animations διαμέσω μιας διαδικτυακής σύνδεσης. Τα animations συγχρονίζονται μεταξύ των παικτών που συμμετέχουν στην ίδια συνεδρία.

ApplicationController

Στην Εικόνα 67, παρατηρείται η ύπαρξη ενός ακόμη αντικειμένου, του ApplicationController. Το αντικείμενο αυτό, αποτελεί ακόμη ένα βασικό συστατικό της διαδικτυακής λειτουργίας του παιχνιδιού, καθώς είναι υπεύθυνο για την δημιουργία και εκκίνηση του server που θα εξυπηρετεί τους χρήστες.



ΕΙΚΟΝΑ 69 - ΤΟ COMPONENT APPLICATION CONTROLLER

Από την Εικόνα 69 παρατηρείται πως η κλάση του ApplicationController δέχεται σαν ορίσματα κάποιες κλάσεις, οι οποίες θα αναλυθούν περαιτέρω στη συνέχεια. Είναι σημαντικό να αναφερθεί εδώ, πως η λειτουργία της ApplicationController διαφοροποιείται σημαντικά από το αποτέλεσμα της συνθήκης:

```
SystemInfo.graphicsDeviceType ==
UnityEngine.Rendering.GraphicsDeviceType.Null
```

ΚΩΔΙΚΑΣ 15 - ΣΥΝΘΗΚΗ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ ΓΙΑ SERVER ΚΑΙ CLIENT

Το αποτέλεσμα της συνθήκης αυτής (Κώδικας 15) θα ορίσει ποιο τμήμα του κώδικα θα εκτελεστεί και ουσιαστικά αναγνωρίζει αν η εφαρμογή εκτελείται σε κάποιον server ή σε κάποιο υπολογιστή χρήστη. Στην πρώτη περίπτωση, αρχικοποιούνται οι κλάσεις του Server και αρχικά η Server Singleton. Καλούνται επίσης οι συναρτήσεις **`serverSingleton.CreateServer()`** και **`serverSingleton.GameManager.StartGameServerAsync()`** οι οποίες είναι υπεύθυνες για τη δημιουργία και έναρξη του server. Στη δεύτερη περίπτωση, αρχικοποιούνται οι HostSingleton και ClientSingleton. Στη συνέχεια ακολουθεί μέρος του κώδικα της ApplicationController κλάσης με τις σημαντικότερες λειτουργίες της (Κώδικας 16).

```
public class ApplicationController : MonoBehaviour {
    [SerializeField] private ClientSingleton clientPrefab;
    [SerializeField] private HostSingleton hostPrefab;
    [SerializeField] private ServerSingleton serverPrefab;
    [SerializeField] private NetworkObject playerPrefab;
    private const string GameSceneName = "Shooter";
    private ApplicationData appData;

    private async void Start() {
      DontDestroyOnLoad(gameObject);
      await LaunchInMode(SystemInfo.graphicsDeviceType ==
      UnityEngine.Rendering.GraphicsDeviceType.Null);
    }

    private async Task LaunchInMode(bool isDedicatedServer) {
        if (isDedicatedServer) {
            Application.targetFrameRate = 60;
            appData = new ApplicationData();
            ServerSingleton serverSingleton = Instantiate(serverPrefab);
            StartCoroutine(LoadGameSceneAsync(serverSingleton));
        } else {
            HostSingleton hostSingleton = Instantiate(hostPrefab);
            hostSingleton.CreateHost(playerPrefab);
            ClientSingleton clientSingleton = Instantiate(clientPrefab);
            bool authenticated =  await clientSingleton.CreateClient();
            if (authenticated) {
               clientSingleton.GameManager.GoToMenu();
            }
        }
    }

private IEnumerator LoadGameSceneAsync(ServerSingleton serverSingleton) {
     AsyncOperation asyncOperation =
     SceneManager.LoadSceneAsync(GameSceneName);
     while (!asyncOperation.isDone) {
         yield return null;}

     Task createServerTask =  serverSingleton.CreateServer(playerPrefab);
     yield return new WaitUntil(() => createServerTask.IsCompleted);

     Task startServerTask =
     serverSingleton.GameManager.StartGameServerAsync();
     yield return new WaitUntil(() => startServerTask.IsCompleted);}
}
```

ΚΩΔΙΚΑΣ 16 – ΚΩΔΙΚΑΣ ΤΟΥ APPLICATION CONTROLLER SCRIPT

3.5.4 Συγχρονισμός καταστάσεων και RPCs

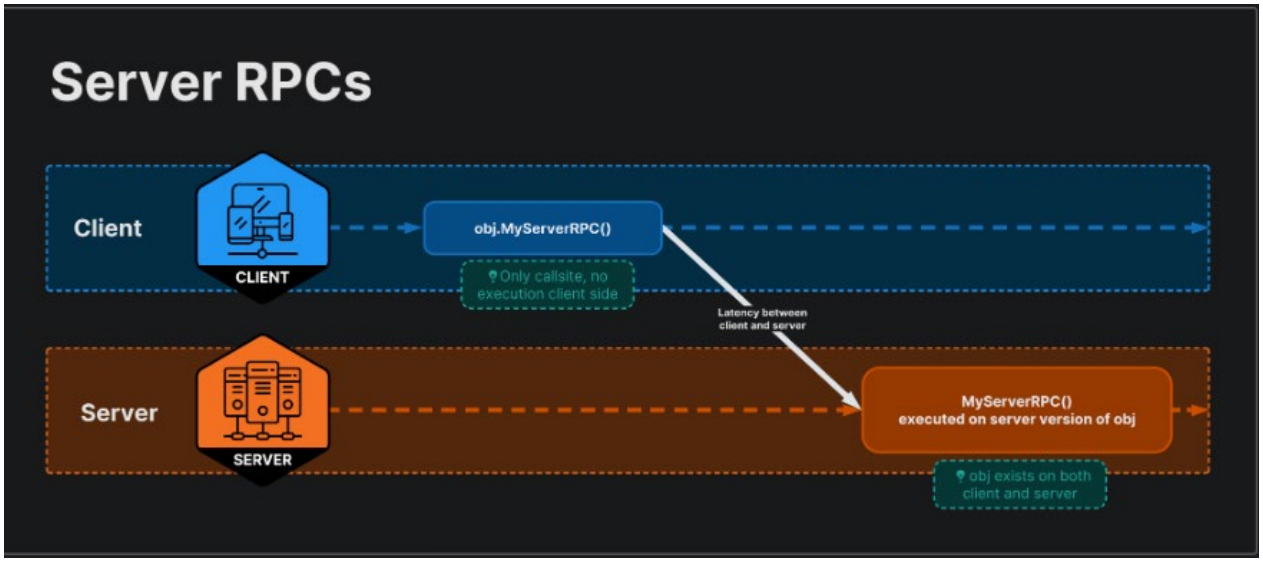
Το Netcode περιλαμβάνει τρεις επιλογές για συγχρονισμό καταστάσεων και γεγονότων.

- **Σύστημα Μηνυμάτων (Messaging System)**

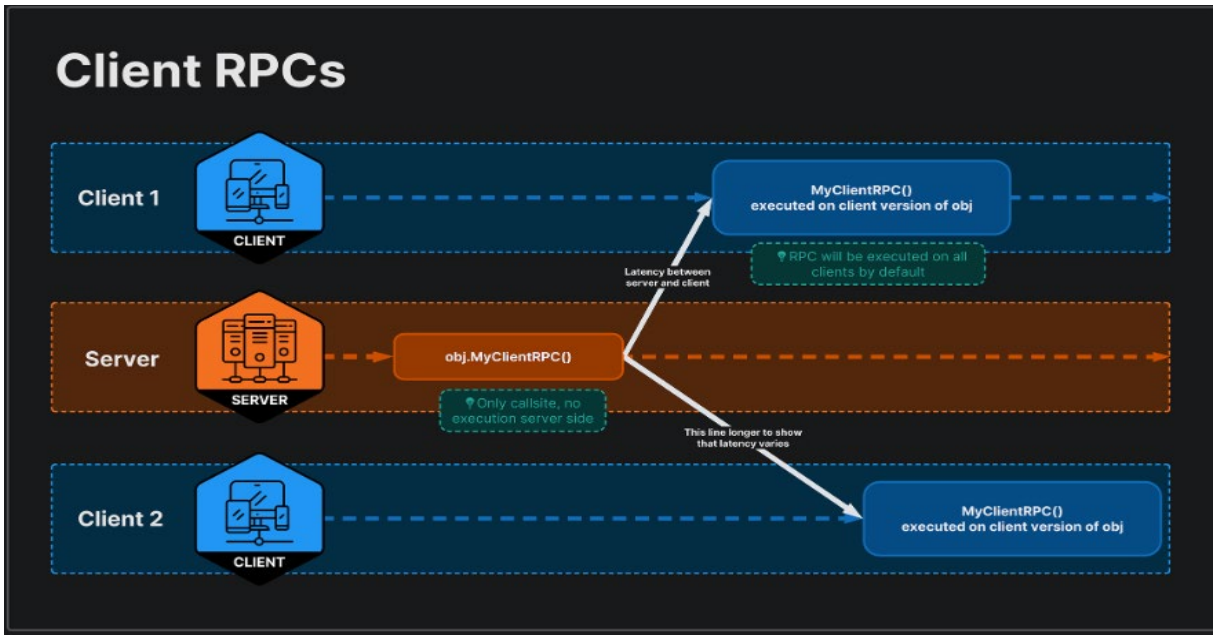
  Το σύστημα μηνυμάτων του Netcode επιτρέπει την αποστολή και λήψη μηνυμάτων και γεγονότων. Ολόκληρο το σύστημα υποστηρίζει την σειριοποίηση των βασικών τύπων δεδομένων καθώς και κλάσεων ή/και δομών που υλοποιούν την διεπαφή INetworkSerializable.

  - **Remote Procedure Calls (RPCs)**

    Τα RPCs αποτελούν έναν τρόπο αποστολής ειδοποιήσεων καθώς και ως ένας τρόπος χειρισμού της απευθείας επικοινωνίας μεταξύ ενός server και ενός client ή αντίστροφα. Είναι ιδιαίτερα χρήσιμα στις περιπτώσεις όπου η ιδιοκτησία της κλάσης NetworkBehavior, εντός της οποίας χρησιμοποιείται το RPC, ανήκει στο server, αλλά χρειάζεται ένας ή περισσότεροι clients να επικοινωνούν με το συσχετισμένο NetworkObject. Στο παιχνίδι που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας, τα RPCs χρησιμοποιήθηκαν κυρίως στο κομμάτι των πυροβολισμών μεταξύ των παικτών.



Εικονα 70 - Αρχιτεκτονικη Server RPCs



Εικονα 71 - Αρχιτεκτονική Client RPCs

- **Custom Messages**

  Επιτρέπει στον προγραμματιστή τη δημιουργία προσαρμοσμένων τύπων μηνυμάτων σε περιπτώσεις όπου αυτά απαιτούνται.

- **NetworkVariables**

  Μια NetworkVariable χρησιμοποιείται συνήθως για να συγχρονίσει μια κατάσταση ή μεταβλητή μεταξύ των ήδη συνδεδεμένων παικτών και όσων συνδέθηκαν αργότερα. Στην προκειμένη περίπτωση, χρησιμοποιείται μια NetworkVariable για να αποθηκεύει και να συγχρονίζει μεταξύ όλων των παικτών το score που έχει συγκεντρώσει ο καθένας.

Παρακάτω ακολουθεί η περιγραφή των βασικών λειτουργιών των κλάσεων της Εικόνα 65.

3.5.5 Κλάσεις Server

3.5.5.1 ServerSingleton.cs

Η κλάση ServerSingleton.cs αποτελεί το script που καλείται πρώτα από την ApplicationController.cs στην περίπτωση της εκτέλεσης του κώδικα από κάποιον dedicated server. Πρόκειται για μια κλάση βασισμένη στο προγραμματιστικό μοτίβο Singleton, με κύρια λειτουργία της την δημιουργία και αρχικοποίηση του ServerGameManager, καθώς και τον ορισμό των απαραίτητων για τον server μεταβλητών, όπως των ip,port και query port. Να σημειωθεί εδώ πως, ως game port ή απλώς port ορίζεται η θύρα του server στην οποία συνδέονται οι παίκτες, ενώ ως query port ορίζεται η θύρα η οποία χρησιμοποιείται για τη συλλογή πληροφοριών όπως το όνομα του server, ο χάρτης, ο αριθμός παικτών κ.ά. Τέλος, καλεί την συνάρτηση

UnityServices.InitializeAsync(), η οποία είναι απαραίτητη για την αρχικοποίηση όλων των υπηρεσιών του Unity Game Services, η οποία και παρουσιάζεται παρακάτω (Κώδικας 17).

```
public class ServerSingleton : MonoBehaviour {
…
    public async Task CreateServer(NetworkObject playerPrefab) {
       await UnityServices.InitializeAsync();
       GameManager = new ServerGameManager(
          ApplicationData.IP(),
          ApplicationData.Port(),
          ApplicationData.QPort(),
          NetworkManager.Singleton,
          playerPrefab);
    }
…}
```

ΚΩΔΙΚΑΣ 17 - Η ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΩΝ ΑΠΑΡΑΙΤΗΤΩΝ ΥΠΗΡΕΣΙΩΝ ΓΙΑ ΤΗ ΔΗΜΙΟΥΡΓΙΑ SERVER

3.5.5.2 ServerGameManager.cs

Πρόκειται ουσιαστικά για τον εγκέφαλο των υπηρεσιών του server, καθώς διαχειρίζεται όλα τα νέα εισερχόμενα αιτήματα, όπως είναι η σύνδεση ή η αποσύνδεση ενός παίκτη από τον server. Αποτελεί επίσης την βάση δύο απαραίτητων λειτουργιών κάθε multiplayer παιχνιδιού, του matchmaking και του backfilling.

Matchmaking

Το matchmaking είναι μια διαδικασία που χρησιμοποιείται σε όλα τα multiplayer παιχνίδια για να συνδέσει αυτόματα παίκτες που έχουν παρόμοιο επίπεδο δεξιοτήτων ή εμπειρίας, προκειμένου να μπορούν να παίξουν μαζί. Στόχος είναι να δημιουργηθούν ισόρροπες και απολαυστικές αναμετρήσεις, όπου κανένας παίκτης δεν έχει πλεονέκτημα εις βάρος των υπολοίπων.

Οι αλγόριθμοι matchmaking λαμβάνουν υπόψη διάφορους παράγοντες, όπως η δεξιότητα των παικτών, οι βαθμολογίες, οι νίκες/ήττες και άλλα στατιστικά, προκειμένου να προσπαθήσουν να βρουν ταιριαστούς αντιπάλους. Αυτό διασφαλίζει ότι οι αγώνες είναι πιο αμφίρροποι και πιο δίκαιοι, προσφέροντας ένα καλύτερο και πιο απολαυστικό περιβάλλον.

Στην προκειμένη περίπτωση, το matchmaking του παιχνιδιού δεν στηρίζεται σε κάποιον αλγόριθμο βασισμένο στην δεξιότητα ή την εμπειρία των παικτών, αλλά κατανέμει τους παίκτες με μοναδικό κριτήριο την ύπαρξη διαθέσιμων θέσεων ή όχι, σε κάποιο από τα ενεργά δωμάτια.

Backfilling

Το backfilling είναι μια διαδικασία που συνδέεται με τα multiplayer παιχνίδια και σχετίζεται με όσα αναφέρθηκαν νωρίτερα. Πρόκειται ουσιαστικά για τον τρόπο με τον οποίο συμπληρώνονται

κενά στον αριθμό των παικτών σε ένα παιχνίδι, είτε λόγω αποχώρησης παικτών από ένα δωμάτιο, είτε επειδή δεν υπάρχουν αρκετοί παίκτες για να ξεκινήσει ένα νέο παιχνίδι. Όταν κάποιος παίκτης αποχωρεί από ένα παιχνίδι, το σύστημα backfilling προσπαθεί να συμπληρώσει αυτήν τη θέση με έναν άλλο παίκτη, είτε προσκαλώντας άλλον παίκτη από την ουρά αναμονής είτε ενσωματώνοντας έναν νέο παίκτη που είναι διαθέσιμος για παιχνίδι. Αυτό συμβάλλει στη διατήρηση του αριθμού των παικτών και στη διασφάλιση ότι οι παρτίδες ξεκινούν με ένα πλήρες σύνολο παικτών.

Το backfilling είναι σημαντικό για τη συνοχή της εμπειρίας παιχνιδιού, αποτρέποντας τη δημιουργία κενών δωματίων και τη διακοπή του ρυθμού του παιχνιδιού. Παρακάτω γίνεται η περιγραφή των σημαντικότερων σημείων του κώδικα της κλάσης ServerGameManager.cs.

Εντός του constructor της ServerGameManager (Κώδικας 18) γίνεται η αρχικοποίηση των τιμών για τις ip και port μέσω της κλάσης ServerSingleton.cs, ενώ παράλληλα δημιουργείται και ένα instance της MultiplayAllocationService, η οποία μέσω των callback συναρτήσεων χειρίζεται διάφορα γεγονότα που αφορούν το matchmaking, το backfilling, τη σύνδεση ή αποσύνδεση των παικτών.

```
public ServerGameManager(string serverIP, int serverPort, int
serverQPort, NetworkManager manager, NetworkObject playerPrefab) {
      this.serverIP = serverIP;
      this.serverPort = serverPort;
      this.queryPort = serverQPort;
      NetworkServer = new NetworkServer(manager, playerPrefab);
      multiplayAllocationService = new MultiplayAllocationService();
}
```

ΚΩΔΙΚΑΣ 18 - Ο CONSTRUCTOR ΤΗΣ ΚΛΑΣΗΣ SERVERGAMEMANAGER

Η **StartGameServerAsync()** αποτελεί τη συνάρτηση που είναι υπεύθυνη για την εκκίνηση του server καθώς και όλων των υπηρεσιών που είναι απαραίτητες για το matchmaking και το backfilling. Για να επιτευχθούν όλα αυτά, υλοποιήθηκαν οι συναρτήσεις **GetMatchmakerPayload(), StartBackfill()**, οι οποίες και φαίνονται παρακάτω. Τέλος, εντός της **StartGameServerAsync()**, καλείται η συνάρτηση **OpenConnection()** της κλάσης NetworkServer, η οποία και καθιστά τον server έτοιμο να δεχθεί συνδέσεις παικτών. Η ServerGameManager περιλαμβάνει επίσης τις συναρτήσεις **UserJoined()** και **UserLeft()**, οι οποίες καλούνται αυτόματα μόλις ο NetworkServer ενημερώσει για το αντίστοιχο event, δηλαδή **OnUserJoined** και **OnUserLeft** αντίστοιχα. Όλα τα παραπάνω, παρουσιάζονται στη συνέχεια (Κώδικας 19).

```
public async Task StartGameServerAsync() {
   //constantly updates UGS with the state of the server
   await multiplayAllocationService.BeginServerCheck();
   MatchmakingResults matchmakerPayload = await GetMatchmakerPayload();
   if (matchmakerPayload != null) {
          await StartBackfill(matchmakerPayload);
          NetworkServer.OnUserJoined += UserJoined;
          NetworkServer.OnUserLeft += UserLeft;
   }else{
          Debug.LogWarning("Matchmaker payload timed out");}
   if (!NetworkServer.OpenConnection(serverIP, serverPort)) {
       Debug.LogWarning("Network Server did not start as expected");
       return;}}

  private async Task<MatchmakingResults> GetMatchmakerPayload() {
     Task<MatchmakingResults> matchmakerPayloadTask =
     multiplayAllocationService.SubscribeAndAwaitMatchmakerAllocation();
     if (await Task.WhenAny(matchmakerPayloadTask, Task.Delay(20000)) ==
     matchmakerPayloadTask) {
       return matchmakerPayloadTask.Result;
     }
     return null;
  }

  private async Task StartBackfill(MatchmakingResults payload) {
     backfiller = new MatchplayBackfiller($"{serverIP}:{serverPort}",
     payload.QueueName, payload.MatchProperties, 20);
     if (backfiller.NeedsPlayers()) {
           await backfiller.BeginBackfilling();}
  }

  private void UserJoined(UserData user) {
     backfiller.AddPlayerToMatch(user);
     multiplayAllocationService.AddPlayer();
     if (!backfiller.NeedsPlayers() && backfiller.IsBackfilling) {
     _ = backfiller.StopBackfill();}
  }

  private void UserLeft(UserData user) {
     int playerCount =
     backfiller.RemovePlayerFromMatch(user.userAuthId);
     multiplayAllocationService.RemovePlayer();
     if (playerCount <= 0) {
         CloseServer();
         return;}
     if (backfiller.NeedsPlayers() && !backfiller.IsBackfilling) {
        _ = backfiller.BeginBackfilling(); }
     }
```

ΚΩΔΙΚΑΣ 19 - ΟΙ ΣΥΝΑΡΤΗΣΕΙΣ ΓΙΑ MATCHMAKING, BACKFILLING ΚΑΙ ΣΥΝΔΕΣΗ/ΑΠΟΣΥΝΔΕΣΗ ΠΑΙΚΤΩΝ

3.5.5.3 NetworkServer.cs

Είναι η κλάση η οποία αρχικοποιείται κατά σειρά τελευταία, ωστόσο η λειτουργία της είναι εξίσου σημαντική με τις προηγούμενες. Η κυριότερη λειτουργία της είναι η έναρξη και διακοπή των συνδέσεων από τον server προς τους παίκτες, όταν αυτοί επιθυμούν να συνδεθούν σε κάποιο διαθέσιμο server.

Μόλις η σύνδεση ενός νέου παίκτη ολοκληρωθεί, τότε η NetworkServer.cs προχωρά στην αρχικοποίηση (Instantiate) του χαρακτήρα, τον οποίο θα χειρίζεται ο χρήστης εντός του παιχνιδιού.

Ειδικότερα, κατά την κλήση του constructor της NetworkServer, αρχικοποιούνται ορισμένες μεταβλητές και ορίζεται ποιες συναρτήσεις θα καλούνται αυτόματα μόλις στέλνονται τα αντίστοιχα events από τον networkManager. Ορίζεται επίσης σε true η ρύθμιση **`networkManager.NetworkConfig.ConnectionApproval`**, η οποία και ελέγχει το αν θα εγκριθεί μια νέα σύνδεση ή όχι. Αυτό είναι απαραίτητο για τον έλεγχο των νέων συνδέσεων και αν τα αντίστοιχα requests πληρούν ορισμένα κριτήρια.

Μόλις σταλθεί το event **networkManager.ConnectionApprovalCallback**, τότε θα καλείται αυτομάτως η συνάρτηση ApprovalCheck, η οποία θα επεξεργάζεται και θα ελέγχει τα στοιχεία της νέας σύνδεσης. Συγκεκριμένα, κάθε τέτοιο request, περιλαμβάνει ένα μοναδικό αναγνωριστικό σύνδεσης, το **request.ClientNetworkId**, καθώς και οτιδήποτε άλλο αφορά τη νέα σύνδεση, το οποίο βρίσκεται εντός της **request.Payload** μεταβλητής, όπως για παράδειγμα το όνομα του παίκτη κ.ά. Μόλις αυτά ολοκληρωθούν, τότε καλείται η **SpawnPlayerDelayed**() η οποία και τοποθετεί τον χαρακτήρα του παίκτη εντός της πίστας.

Όλα όσα προαναφέρθηκαν και αφορούν την δυνατότητα σύνδεσης των παικτών με τον server, παρουσιάζονται με τη μορφή κώδικα στη συνέχεια ().

```
public class NetworkServer  : IDisposable {
   public Action<UserData> OnUserJoined;
   public Action<UserData> OnUserLeft;
   private Dictionary<ulong,string> clientIdToAuth = new
Dictionary<ulong,string>();
   private Dictionary<string,UserData> authIdToUserData = new
Dictionary<string,UserData>();

public NetworkServer(NetworkManager networkManager, NetworkObject
playerPrefab){
       this.networkManager = networkManager;
       this.playerPrefab = playerPrefab;
       networkManager.NetworkConfig.ConnectionApproval = true;
       networkManager.ConnectionApprovalCallback += ApprovalCheck;
       networkManager.OnServerStarted += OnNetworkReady; }

private void ApprovalCheck(NetworkManager.ConnectionApprovalRequest
request, NetworkManager.ConnectionApprovalResponse response) {
      string payLoad =
      System.Text.Encoding.UTF8.GetString(request.Payload);
      UserData userData = JsonUtility.FromJson<UserData>(payLoad);
      clientIdToAuth[request.ClientNetworkId] = userData.userAuthId;
      authIdToUserData[userData.userAuthId] = userData;
      OnUserJoined?.Invoke(userData);
      _ = SpawnPlayerDelayed(request.ClientNetworkId);
      response.Approved = true;
      response.CreatePlayerObject = false;
}

 private async Task SpawnPlayerDelayed(ulong clientId) {
      await Task.Delay(1000);
      NetworkObject playerInstance = GameObject.Instantiate(playerPrefab,
      SpawnPoint.GetRandomSpawnPos(), Quaternion.identity);
      playerInstance.SpawnAsPlayerObject(clientId);
 }

 public bool OpenConnection(string ip, int port) {
      UnityTransport transport =
      networkManager.gameObject.GetComponent<UnityTransport>();
      transport.SetConnectionData(ip, (ushort)port);
      return networkManager.StartServer();
 }

public UserData GetUserDataByClientId(ulong clientId) {
     if (clientIdToAuth.TryGetValue(clientId, out string authId)) {
          if (authIdToUserData.TryGetValue(authId, out UserData data)) {
                return data; }
            return null; }
return null; }}
```

ΚΩΔΙΚΑΣ 20 – ΟΙ ΣΥΝΑΡΤΗΣΕΙΣ ΤΗΣ ΚΛΑΣΗΣ NETWORKSERVER

3.5.6 Κλάσεις Host

Όπως περιεγράφηκε νωρίτερα για τις λειτουργίες της ApplicationController.cs, στην περίπτωση όπου ο κώδικας της κλάσης εκτελείται σε κάποιον υπολογιστή και όχι σε dedicated server, τότε θα ακολουθηθεί διαφορετική πορεία και θα αρχικοποιηθούν οι κλάσεις που αφορούν τον Host και τους Clients.

3.5.6.1 HostSingleton.cs

Η HostSingleton.cs είναι η πρώτη κλάση που αρχικοποιείται μόλις η εφαρμογή εκτελείται στον υπολογιστή κάποιου χρήστη. Είναι ουσιαστικά μια κλάση βασισμένη στο πρότυπο Singleton, επιτρέποντας την υπάρξη μιας μοναδικής εκδοχής της κλάσης η οποία παρέχει ένα κεντρικό σημείο πρόσβασης σε αυτήν. Κατά την δημιουργία της HostSingleton, δημιουργείται και ένα instance της HostGameManager (Κώδικας 21), όπως θα αναλυθεί στη συνέχεια. Η HostSingleton δηλαδή, θα παρέχει ουσιαστικά πρόσβαση και στην HostGameManager και τις υπηρεσίες που παρέχει.

```
public class HostSingleton : MonoBehaviour {
    public void CreateHost(NetworkObject playerPrefab) {
       GameManager = new HostGameManager(playerPrefab);
    }}
```

Κωδικας 21 - Η δημιουργια ενος instance της HostGameManager

3.5.6.2 HostGameManager.cs

Η HostGameManager.cs εστιάζει στην δημιουργία των lobbies στα οποία οι χρήστες μπορούν να συνδεθούν και στην αρχικοποίηση ενός μοναδικού κωδικού τον οποίο οι παίκτες μπορούν να χρησιμοποιήσουν προκειμένου να συνδεθούν στο δωμάτιο που επιθυμούν.

Η HostGameManager είναι ιδιαίτερα σημαντική καθώς πραγματοποιεί όλες τις ενέργειες που είναι απαραίτητες για το **Relay**, τα **Lobbies** και το **HeartBeat**, τα οποία αναλύονται παρακάτω (Κώδικας 22, Κώδικας 23).

Συγκεκριμένα, μέσω της **Relay**.**Instance**.**CreateAllocationAsync**(**maxConnections**), στέλνει αίτημα προς το Unity Game Services για την κράτηση ενός relay server, ενημερώνοντας το παράλληλα για το μέγιστο αναμενόμενο αριθμό παικτών. Αν αυτό επιτύχει, τότε μέσω της **Relay**.**Instance**.**GetJoinCodeAsync**() ανακτά τον μοναδικό κωδικό σύνδεσης στο lobby, τον οποίο μπορεί να διαμοιράσει μεταξύ των υπολοίπων παικτών, προκειμένου να συνδεθούν στο δωμάτιο που δημιούργησε. Έχοντας πλέον πραγματοποιήσει την κράτηση του Relay Server, χρειάζεται

πλέον, όλη η κίνηση που προέρχεται από το Netcode, να περνά μέσα από τον Relay Server. Αυτό επιτυγχάνεται μέσω των παρακάτω συναρτήσεων:

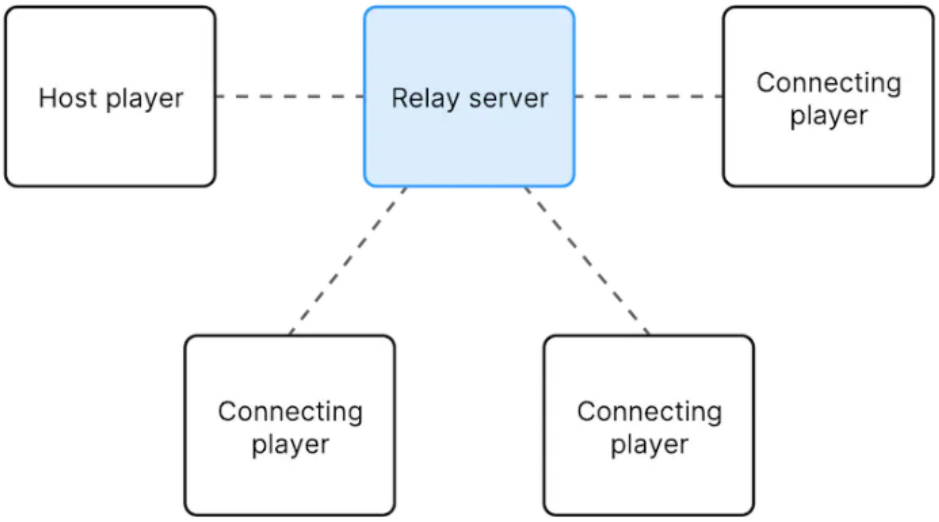
- `UnityTransport transport = NetworkManager.Singleton.GetComponent<UnityTransport>()` για την ανάκτηση του UnityTransport component του NetworkManager,
- `transport.SetRelayServerData(new RelayServerData(allocation, connectionType:"udp"));` για τον ορισμό των στοιχείων του relay server και του πρωτοκόλλου σύνδεσης σε αυτόν.

Στη συνέχεια χρειάζεται να γίνει η δημιουργία του δωματίου το οποίο θα φιλοξενεί τους παίκτες μέσω της **Lobbies.Instance.CreateLobbyAsync(),** η οποία δέχεται σαν ορίσματα το όνομα του host και τον μέγιστο αριθμό συνδέσεων, ενώ τέλος καλείται η NetworkManager.Singleton.StartHost() για την έναρξη λειτουργίας του Host.

Επιπλέον, όπως φαίνεται στην Εικόνα 65, η HostGameManager, δημιουργεί την κλάση NetworkServer, η οποία αναλύθηκε νωρίτερα. Στην πραγματικότητα, είναι ακριβώς η ίδια κλάση που χρησιμοποιήθηκε στην περίπτωση των dedicated servers, με μοναδική διαφορά ότι ο κώδικας της εκτελείται πλέον στον υπολογιστή κάποιου παίκτη, μετατρέποντάς τον σε οικοδεσπότη (host). Ο τρόπος με τον οποίο θα διαχειρίζεται ο host τις συνδέσεις νέων παικτών, είναι ακριβώς ίδιος με τον τρόπο με τον οποίο θα τις διαχειριζόταν ένας dedicated server.

Unity Relay

Ο τρόπος με τον οποίο οι παίκτες συνδέονται σε κάποιο host, βασίζεται στην υπηρεσία Relay που παρέχει το Unity, η οποία εκτελεί όλες τις απαραίτητες ενέργειες προκειμένου να επιτευχθεί η σύνδεση. Χωρίς αυτό, θα έπρεπε να υπάρξει μέριμνα από τον ίδιο τον προγραμματιστή για την αντιμετώπιση ορισμένων συνηθισμένων προβλημάτων προκειμένου να επιτευχθεί η δυνατότητα απρόσκοπτης επικοινωνίας μεταξύ των συσκευών του τοπικού δικτύου και του ευρύτερου διαδικτύου. Το Relay λειτουργεί σαν μεσάζων μεταξύ των παικτών κάνοντας ουσιαστικά αναμετάδοση των δεδομένων που λαμβάνει.



ΕΙΚΟΝΑ 72 - ΑΡΧΙΤΕΚΤΟΝΙΚΗ RELAY

Οι εξυπηρετητές Relay διανέμουν τα μηνύματα μεταξύ των παικτών με χαμηλή καθυστέρηση με τρόπο τέτοιο ώστε να αποφεύγεται η απευθείας σύνδεση μεταξύ δύο παικτών μεταξύ τους. Η υπηρεσία Relay είναι κατάλληλη για παιχνίδια που βασίζονται στη λογική όπου ένας παίκτης – client (παίκτης host) φέρεται ως server και οι υπόλοιποι παίκτες ως clients. Όπως φαίνεται στην Εικόνα 72, οι εξυπηρετητές relay αποτελούν το τελικό σημείο (endpoint) και είναι προσβάσιμοι από όλους τους παίκτες. Το μοτίβο αυτό επιλύει συνηθισμένα προβλήματα όπως η εναλλαγή δικτύων και διευθύνσεων IP, η μετάφραση διευθύνσεων δικτύου (NAT) και τα firewalls μεταξύ των παικτών. Κάθε παίκτης μπορεί να συνδεθεί στην ίδια διεύθυνση IP και στο ίδιο port, δηλαδή τη διεύθυνση και το port του επιλεγμένου Relay Server, και μπορούν να είναι σίγουροι ότι οι πληροφορίες σύνδεσης θα παραμείνουν ίδιες καθόλη τη διάρκεια του παιχνιδιού. Με τον τρόπο αυτό, οι παίκτες μια διαδικτυακής σύνδεσης δεν χρειάζεται να γνωρίζουν τις διευθύνσεις IP των υπολοίπων, αυξάνοντας έτσι την ασφάλεια και την ιδιωτικότητα.

Unity Lobby

Εκτός της υπηρεσίας Relay, μια ακόμη απαραίτητη υπηρεσία είναι η Lobby για την επίτευξη της λειτουργίας host – client. Η Lobby σε συνδυασμό με την Relay, επιτρέπει στους παίκτες να συνδέονται στα υπάρχοντα δωμάτια, να δημιουργούν δικά τους και να ενημερώνονται διαρκώς για τις διαθέσιμες επιλογές δωματίων. Συγκεκριμένα, οι δυνατότητες που παρέχει είναι:

- Περιήγηση στη λίστα των διαθέσιμων δωματίων και δυνατότητα επιλογής και σύνδεσης σε αυτά,
- Διαμοιρασμός ενός μοναδικού κωδικού (join code) με άλλους παίκτες, προκειμένου να μπορούν να συνδεθούν απευθείας,

- Δυνατότητα Γρήγορης Σύνδεσης (Quick Join) για εύρεση και σύνδεση σε τυχαίο δωμάτιο,
- Δημιουργία ιδιωτικού δωματίου και πρόσκληση σε άλλους παίκτες,
- Αναζήτηση και εύρεση δωματίων που πληρούν κάποια κριτήρια (χάρτης, είδος παιχνιδιού, αριθμός παικτών κ.ά.).

Heartbeat

Τέλος, μια ακόμη βασική λειτουργία της HostGameManager, είναι το Heartbeat του lobby. Ως Heartbeat ορίζεται η αποστολή ενός request από τον host, προκειμένου να μην θεωρεί το lobby ως αδρανές. Ο κύριος λόγος που αυτό συμβαίνει είναι για να μην εμφανίζονται στους άλλους παίκτες τα δωμάτια που είναι αδρανή. Όσα δηλαδή δωμάτια εμφανίζονται στη λίστα, θα είναι τα δωμάτια που έχουν ενεργούς συνδεδεμένους παίκτες.

```
public class HostGameManager : IDisposable {
  private Allocation allocation;
  private string joinCode;
  private string lobbyId;
  private const int maxConnections = 20;
  private const string GameSceneName = "fpsShooter";
  private NetworkObject playerPrefab;

  public HostGameManager(NetworkObject playerPrefab) {
        this.playerPrefab = playerPrefab;
    }

  public async Task StartHostAsync() {
      allocation = await
      Relay.Instance.CreateAllocationAsync(maxConnections);
      joinCode = await
      Relay.Instance.GetJoinCodeAsync(allocation.AllocationId);
      UnityTransport transport =
      NetworkManager.Singleton.GetComponent<UnityTransport>();
      RelayServerData relayServerData = new RelayServerData(allocation,
      "udp");
      transport.SetRelayServerData(relayServerData);
      //Creating Lobby
      CreateLobbyOptions lobbyOptions = new CreateLobbyOptions();
      lobbyOptions.IsPrivate = false;
      lobbyOptions.Data = new Dictionary<string, DataObject>() {
      {
        "JoinCode", new DataObject(
        visibility: DataObject.VisibilityOptions.Member,
        value: joinCode)}
       };
      string playerName =
      PlayerPrefs.GetString(NameSelector.PlayerNameKey, "Unknown Lobby");
      Lobby lobby = await
      Lobbies.Instance.CreateLobbyAsync($"{playerName}'s Lobby",
      maxConnections, lobbyOptions);
```

```
        lobbyId = lobby.Id;
        HostSingleton.Instance.StartCoroutine(HeartbeatLobby(15));
        UserData userData = new UserData {
              userName =
              PlayerPrefs.GetString(NameSelector.PlayerNameKey,"Missing
              Name"),
              userAuthId = AuthenticationService.Instance.PlayerId };
        string payload = JsonUtility.ToJson(userData);
        byte[] payloadBytes = Encoding.UTF8.GetBytes(payload);
        NetworkManager.Singleton.NetworkConfig.ConnectionData =
        payloadBytes;
        NetworkServer = new NetworkServer(NetworkManager.Singleton,
        playerPrefab);
        NetworkManager.Singleton.StartHost();
        NetworkManager.Singleton.SceneManager.LoadScene(GameSceneName,
LoadSceneMode.Single);}

private IEnumerator HeartbeatLobby(float waitTimeSeconds) {
        WaitForSecondsRealtime delay = new
        WaitForSecondsRealtime(waitTimeSeconds);
        while (true) {
            Lobbies.Instance.SendHeartbeatPingAsync(lobbyId);
            yield return delay;
        }
}
```

ΚΩΔΙΚΑΣ 23 - Ο ΚΩΔΙΚΑΣ ΤΗΣ HOSTGAMEMANAGER ΚΛΑΣΗΣ

3.5.7 Κλάσεις Client

3.5.7.1 ClientSingleton.cs

Πρόκειται για άλλη μια κλάση βασισμένη στο μοτίβο Singleton, η οποία θα είναι προσβάσιμη από διάφορα μέρη του κώδικα. Η βασική της λειτουργία, είναι η δημιουργία και αρχικοποίηση της κλάσης ClientGameManager (Κώδικας 24), η οποία θα περιεγραφεί παρακάτω, και εν συνεχεία η κλήση της **GameManager.InitAsync().**

```
public class ClientSingleton : MonoBehaviour {

public async Task<bool> CreateClient() {
        GameManager = new ClientGameManager();
        return await GameManager.InitAsync();}}
```

ΚΩΔΙΚΑΣ 24 - ΔΗΜΙΟΥΡΓΙΑ ΕΝΟΣ INSTANCE ΤΗΣ CLIENTGAMEMANAGER ΜΕΣΩ ΤΗΣ ΣΥΝΑΡΤΗΣΗΣ CREATECLIENT

3.5.7.2 ClientGameManager.cs

Αποτελεί την κλάση που χειρίζεται τη λογική πίσω από τη δυνατότητα σύνδεσης των παικτών τόσο σε dedicated servers όσο και σε δωμάτια (lobbies) άλλων παικτών. Οι παίκτες μέσω των συναρτήσεων της κλάσης ClientGameManager, μπορούν να συνδέονται σε servers είτε μέσω της

επιλογής Find Match (matchmaking), είτε μέσω ενός μοναδικού κωδικού, που αν τον γνωρίζουν θα τους επιτρέψει τη σύνδεση σε ένα συγκεκριμένο δωμάτιο, είτε μέσω της επιλογής Lobbies, όπου θα μπορούν να δουν όλα τα διαθέσιμα δωμάτια και να επιλέξουν. Οι διαδικασίες αυτές, παρουσιάζονται παρακάτω (Κώδικας 25,Κώδικας 26).

```
public class ClientGameManager : IDisposable {
    public async Task<bool> InitAsync() {
       await UnityServices.InitializeAsync();
       networkClient = new NetworkClient(NetworkManager.Singleton);
       matchmaker = new MatchplayMatchmaker();
       AuthState authState = await AuthenticationWrapper.DoAuth();

    if (authState == AuthState.Authenticated) {
            userData = new UserData {
            userName = PlayerPrefs.GetString(NameSelector.PlayerNameKey,
            "Missing Name"),
            userAuthId = AuthenticationService.Instance.PlayerId };
            return true;}
            return false;}

    public async Task CancelMatchmaking() {
        await matchmaker.CancelMatchmaking();
    }

    public void GoToMenu() {
        SceneManager.LoadScene(MenuSceneName);}

    public void StartClient(string ip, int port) {
      UnityTransport transport =
      NetworkManager.Singleton.GetComponent<UnityTransport>();
      transport.SetConnectionData(ip, (ushort)port);
      ConnectClient();
    }

    public async Task StartClientAsync(string joinCode) {
       allocation = await Relay.Instance.JoinAllocationAsync(joinCode);
      UnityTransport transport =
      NetworkManager.Singleton.GetComponent<UnityTransport>();
      RelayServerData relayServerData = new RelayServerData(allocation,
      "udp");
      transport.SetRelayServerData(relayServerData);
      ConnectClient();
    }

    private void ConnectClient() {
      string payload = JsonUtility.ToJson(userData);
      byte[] payloadBytes = Encoding.UTF8.GetBytes(payload);
      NetworkManager.Singleton.NetworkConfig.ConnectionData =
      payloadBytes;
      NetworkManager.Singleton.StartClient();
    }
```

ΚΩΔΙΚΑΣ 25 - ΣΥΝΔΕΣΗ ΤΩΝ CLIENTS ΣΕ SERVERS/LOBBIES

```
public async void MatchmakeAsync(Action<MatchmakerPollingResult>
onMatchmakeResponse) {
    if (matchmaker.IsMatchmaking) {
        return;}
    MatchmakerPollingResult matchResult = await GetMatchAsync();
    onMatchmakeResponse?.Invoke(matchResult);
}

private async Task<MatchmakerPollingResult> GetMatchAsync() {
  MatchmakingResult matchmakingResult = await
  matchmaker.Matchmake(userData);
  if (matchmakingResult.result == MatchmakerPollingResult.Success) {
        StartClient(matchmakingResult.ip, matchmakingResult.port); }
  return matchmakingResult.result;}
}
```

ΚΩΔΙΚΑΣ 26 - ΣΥΝΔΕΣΗ ΤΩΝ CLIENTS ΣΕ SERVERS/LOBBIES (ΣΥΝΕΧΕΙΑ)

3.5.7.3 NetworkClient.cs

Η κύρια λειτουργία της κλάσης αυτής είναι η εναλλαγή μεταξύ των σκηνών, όταν ο παίκτης αποσυνδέεται από το παιχνίδι, καθώς και η εκπομπή του αντίστοιχου event, όπως παρουσιάζεται παρακάτω (Κώδικας 27).

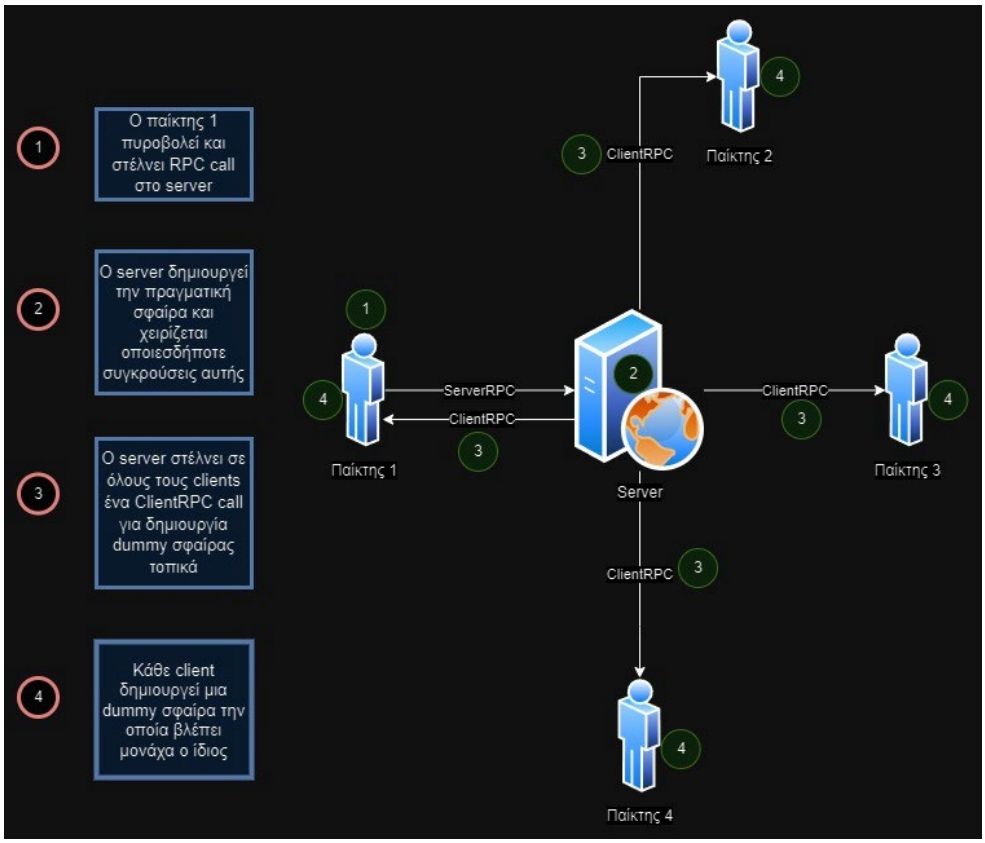
```
public class NetworkClient : IDisposable {
    private NetworkManager networkManager;
    private const string MenuSceneName = "Menu";

    public NetworkClient(NetworkManager networkManager) {
        this.networkManager = networkManager;
        networkManager.NetworkConfig.ConnectionApproval = true;
        networkManager.OnClientDisconnectCallback += OnClientDisconnect;
    }

    private void OnClientDisconnect(ulong clientId) {
        if (clientId != 0 && clientId != networkManager.LocalClientId) {
            return;
        }
        if (SceneManager.GetActiveScene().name != MenuSceneName) {
            SceneManager.LoadScene(MenuSceneName);
        }
        if (networkManager.IsConnectedClient) {
            networkManager.Shutdown();
        }
    }
```

ΚΩΔΙΚΑΣ 27 - ΑΠΟΣΥΝΔΕΣΗ ΠΑΙΚΤΗ ΚΑΙ ΕΝΑΛΛΑΓΗ ΣΚΗΝΩΝ

3.6 Βολές



ΕΙΚΟΝΑ 73 - ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΣΥΓΧΡΟΝΙΣΜΟΣ ΠΥΡΟΒΟΛΙΣΜΩΝ

Έχοντας πλέον δημιουργήσει το χαρακτήρα και την πίστα, χρειάζεται να υλοποιηθεί η λογική που αφορά τη δυνατότητα του παίκτη να πυροβολεί, κάτι που είναι απαραίτητο φυσικά για κάθε shooter παιχνίδι. Η λογική αυτή είναι φαινομενικά απλή, εφόσον όμως το παιχνίδι θα απευθύνεται σε πολλαπλούς παίκτες, χρειάζεται να υπάρξει μέριμνα για συγκεκριμένα θέματα.

Είναι πολύ σημαντικό, όπως θα επεξηγηθεί και στη συνέχεια, να υπάρχει όσο το δυνατόν μικρότερη καθυστέρηση στο συγχρονισμό των σφαιρών, ώστε η εμπειρία των χρηστών να είναι η καλύτερη δυνατή. Ένα multiplayer shooter παιχνίδι με αυξημένες καθυστερήσεις (lag), θα δυσανασχετούσε τους παίκτες και θα δημιουργούσε σημαντικές διαφορές στην εμπειρία του καθενός.

Είναι επίσης μείζονος σημασίας, η ανίχνευση των συγκρούσεων και η εφαρμογή ζημιάς (damage), μεταξύ των παικτών, να γίνεται μέσω ενός κεντρικού σημείου. Αυτό θα βελτιώσει σημαντικά την εμπειρία των παικτών, και επίσης θα ελαχιστοποιήσει τις πιθανότητες κάποιος παίκτης να «κλέβει» τους υπόλοιπους. Το κεντρικό αυτό σημείο (server ή host), θα είναι αρμόδιο για όλες

αυτές τις λειτουργίες. Παρακάτω θα περιεγραφεί ο τρόπος με τον οποίο υλοποιήθηκε η λογική των πυροβολισμών και της ζημιάς μεταξύ των παικτών.

Οι παίκτες θα μπορούν, μόλις πληρούνται κάποιες προϋποθέσεις, όπως η στόχευση με το πάτημα του δεξιού κλικ και η ύπαρξη επαρκών σφαιρών στο γεμιστήρα, να πυροβολούν προκειμένου να πλήξουν τους αντίπαλούς τους ή να καταστρέψουν κάποιο αντικείμενο. Μόλις λοιπόν ο παίκτης πυροβολεί, θα γίνεται μια κλήση RPC προς τον server ή τον host, προκειμένου ο server ή ο host να δημιουργήσει τη σφαίρα και να την ωθήσει προς το σημείο όπου ο παίκτης στοχεύει. Η σφαίρα αυτή, θα είναι η σφαίρα που μέσω της ανίχνευσης συγκρούσεων του Unity, θα αναγνωρίζει αν συγκρούστηκε με κάποιον άλλο παίκτη και τότε θα του κάνει ζημιά. Πρόκειται δηλαδή για την πραγματική σφαίρα. Με τον τρόπο αυτό, αποτρέπεται οποιαδήποτε πιθανότητα κάποιος παίκτης να αδικεί τους υπόλοιπους, καθώς ο χειρισμός των συγκρούσεων γίνεται αποκλειστικά από το server.

Στη συνέχεια, ο εξυπηρετητής θα ειδοποιεί όλους τους παίκτες (clients), προκειμένου να δημιουργούν μια εικονική σφαίρα, ακριβώς ίδια με την προηγούμενη, αυτή τη φορά όμως τοπικά. Έτσι λοιπόν, οι υπόλοιποι clients θα δημιουργούν μια σφαίρα την οποία θα βλέπουν μονάχα οι ίδιοι, η οποία όμως δεν θα κάνει ζημιά όταν συναντά άλλους παίκτες. Ο λόγος που αυτό γίνεται, είναι για να επιτυγχάνεται το κομμάτι της καταστρεψιμότητας των κτηρίων αλλά και της αποφυγής αποκλίσεων μεταξύ των παικτών, κάτι που είναι πολύ πιθανό για αντικείμενα που κινούνται γρήγορα, όπως στην προκειμένη περίπτωση οι σφαίρες.

Με την παραπάνω διαδικασία, ο server είναι εξ ’ολοκλήρου υπεύθυνος για την ζημιά που ασκείται μεταξύ των παικτών, ενώ παράλληλα, μέσω της δημιουργίας τοπικών σφαιρών, αποτρέπεται η ύπαρξη αποκλίσεων στο πως οι παίκτες βλέπουν την πορεία των σφαιρών των υπολοίπων. Δημιουργείται έτσι, μια ισάξια εμπειρία για όλους.

Έχοντας εξηγήσει τη λογική των πυροβολισμών καθώς και την αρχιτεκτονική των RPC κλήσεων παραπάνω, είναι χρήσιμο εδώ να γίνει αναφορά στον τρόπο με τον οποίο ο server χειρίζεται τη ζημιά που ασκείται στους παίκτες, όταν αυτοί πλήττονται από κάποια αντίπαλη σφαίρα. Όπως προαναφέρθηκε, ο server είναι αυτός που δημιουργεί και χειρίζεται την «πραγματική» σφαίρα. Στη σφαίρα αυτή, επισυνάπτεται το script **DealDamageOnContact.cs**. Εντός του αρχείου και συγκεκριμένα εντός της συνάρτησης **OnCollisionEnter**(), η οποία καλείται αυτόματα μόλις το αντικείμενο συγκρουστεί με οποιοδήποτε άλλο αντικείμενο, ο server θα ελέγχει τη φύση του αντικειμένου με το οποίο η σφαίρα συγκρούστηκε. Αν πρόκειται για αντικείμενο – παίκτη, τότε ο server θα μειώνει τη ζωή του παίκτη και ταυτόχρονα θα ελέγχει αν η ζωή του βρίσκεται κάτω από το 0, οπότε και θα εκκινεί όλες τις διαδικασίες για την «αναγέννηση» του παίκτη, δηλαδή την εκ

νέου αρχικοποίηση του χαρακτήρα του σε κάποιο τυχαίο σημείο του χάρτη. Υπεύθυνη για όλα αυτά, είναι η κλάση ProjectileLauncher, τα σημαντικότερα μέρη της οποίας παρατίθενται στη συνέχεια, όπως αναλύθηκαν παραπάνω (Κώδικας 28).

```
    void Update() {
        if (!IsOwner) return;
        UserInput();
    }

    private void UserInput() {
        …
      if (readyToShoot && shooting && !reloading && aiming &&
      bulletsLeft > 0) {
          PrimaryFireServerRpc();
      }
    }

   [ServerRpc]
    private void PrimaryFireServerRpc() {
      …
      GameObject currentBullet = Instantiate(serverProjectilePrefab,
      projectileSpawnPoint.position, Quaternion.identity);
      …
      SpawnDummyProjectileClientRpc();
    }

  [ClientRpc]
   private void SpawnDummyProjectileClientRpc() {
     SpawnDummyProjectile();
   }

   private void SpawnDummyProjectile() {
      …
      GameObject currentBullet = Instantiate(clientProjectilePrefab,
      projectileSpawnPoint.position, Quaternion.identity);
      …
   }
```

ΚΩΔΙΚΑΣ 28 – ΔΙΑΔΙΚΑΣΙΑ ΔΗΜΙΟΥΡΓΙΑΣ ΤΩΝ ΒΟΛΩΝ ΓΙΑ SERVER ΚΑΙ CLIENT

Στα παραπάνω τμήματα κώδικα παρουσιάζεται ολόκληρη η διαδικασία που περιεγράφηκε παραπάνω. Φαίνεται δηλαδή, πως ο server ελέγχει την αλληλουχία των πυροβολισμών, ενώ είναι σημαντικό να σημειωθεί πως η PrimaryFireServerRpc() και η SpawnDummyProjectile αρχικοποιούν διαφορετικά αντικείμενα, το serverProjectilePrefab και το clientProjectilePrefab. Ενώ φαινομενικά τα δύο αυτά αντικείμενα – σφαίρες, είναι οπτικά ίδια, στην πραγματικότητα έχουν μια σημαντική διαφορά. Προκειμένου η serverProjectilePrefab να μπορεί να κάνει ζημιά στους παίκτες με τους οποίους συγκρούεται, χρειάζεται να διαθέτει το script **DealDamageOnContact**, τα σημαντικότερα μέρη του οποίου βρίσκονται παρακάτω (Κώδικας 29).

```
public class DealDamageOnContact : MonoBehaviour {
    …
    private void OnCollisionEnter(Collision col) {
    …
        if (col.gameObject.TryGetComponent<Health>(out Health health)) {
            health.TakeDamage(damage);
            …
       }}}
```

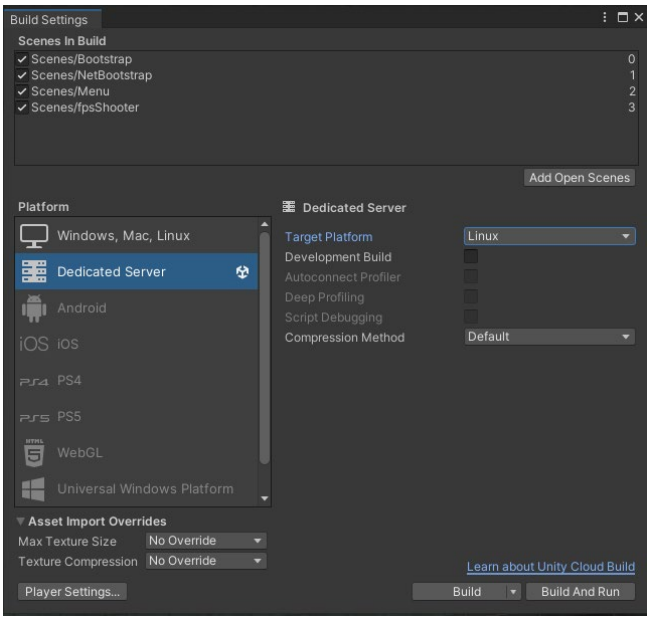
ΚΩΔΙΚΑΣ 29 - ΧΕΙΡΙΣΜΟΣ ΤΩΝ ΣΥΓΚΡΟΥΣΕΩΝ ΤΩΝ ΒΟΛΩΝ ΜΕ ΑΛΛΟΥΣ ΠΑΙΚΤΕΣ

4. Build Παιχνιδιού & Χειρισμός

Στην παρακάτω ενότητα θα γίνει αναφορά στον τρόπο που γίνεται το build του παιχνιδιού, δηλαδή στη διαδικασίας κατασκευής του τελικού εκτελέσιμου αρχείου. Πρόκειται για το τελικό πακέτο του παιχνιδιού το οποίο στη συνέχεια μπορεί να εκτελεστεί και να διανεμηθεί σε διάφορους χρήστες και πλατφόρμες. Στα πλαίσια της παρούσας εργασίας χρειάζεται να εκτελεστεί η διαδικασία του build δύο φορές. Η πρώτη αφορά το build που θα εκτελείται από τους dedicated servers και το άλλο από τους ίδιους τους παίκτες στον υπολογιστή τους. Στις παρακάτω ενότητες περιγράφονται οι επιμέρους διαδικασίες.

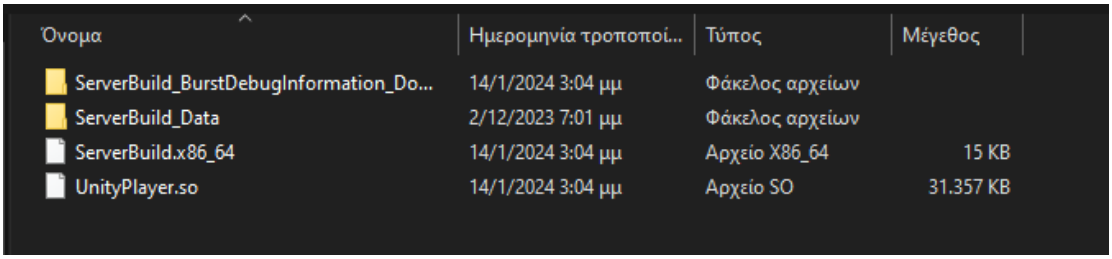
4.1 Server Build

Η πρώτη περίπτωση αφορά τους dedicated servers. Όπως θα αναλυθεί και στη συνέχεια, οι servers που χρησιμοποιήθηκαν για την επίτευξη της διαδικτυακής λειτουργίας του παιχνιδιού βασίζονται στην υπηρεσία Game Server Hosting (Multiplayer) που παρέχεται μέσω του Unity Gaming Services. Συγκεκριμένα, πρόκειται για Linux Servers, οπότε θα πρέπει και το build να έχει τις κατάλληλες ρυθμίσεις. Επιλέγοντας File -> Build Settings, εμφανίζεται το παράθυρο της Εικόνα 74. Στο επάνω μέρος του παραθύρου, ο χρήστης χρειάζεται να εισαγάγει με τη σωστή σειρά τις σκηνές που απαρτίζουν το παιχνίδι και στη συνέχεια να προχωρήσει στη δημιουργία των εκτελέσιμων πατώντας Build ή Clean Build.



ΕΙΚΟΝΑ 74 - ΟΙ ΡΥΘΜΙΣΕΙΣ ΤΟΥ SERVER BUILD

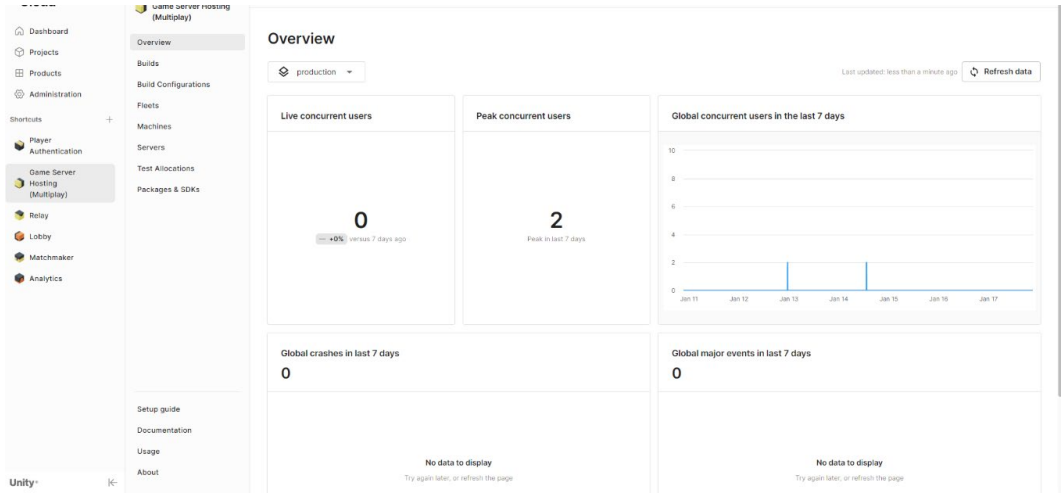
Η διαδικασία αυτή μπορεί να διαρκέσει αρκετά καθώς εξαρτάται από τον όγκο των αρχείων που περιλαμβάνει το παιχνίδι. Το τελικό αποτέλεσμα θα μοιάζει με αυτό της Εικόνα 75.



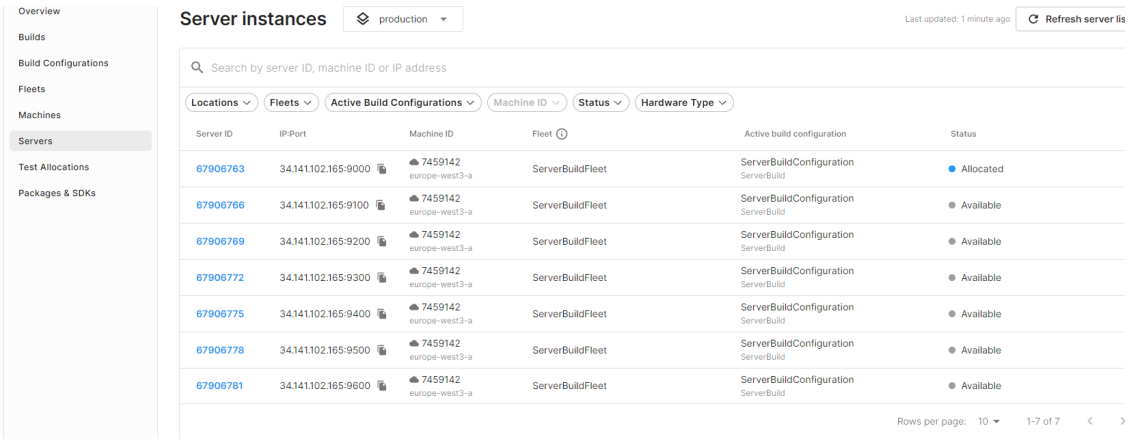
ΕΙΚΟΝΑ 75 - ΤΑ ΑΡΧΕΙΑ ΤΟΥ SERVER BUILD

4.1.1 Εκκίνηση server και ανέβασμα αρχείων

Το Unity μέσω του Game Server Hosting (Multiplay) προσφέρει στο χρήστη τη δυνατότητα να χρησιμοποιεί έναν server που βρίσκεται στο Unity Cloud, ο οποίος θα αποτελεί τον οικοδεσπότη του παιχνιδιού του. Μέσω της διεπαφής που φαίνεται στην Εικόνα 76, ο δημιουργός του παιχνιδιού έχει πρόσβαση σε μια πληθώρα στατιστικών στοιχείων σχετικά με τον αριθμό των παικτών, των όγκο των δεδομένων που ανταλλάχθηκε καθώς και στοιχεία που αφορούν τους διαθέσιμες servers όπως η IP, το port, η τοποθεσία τους κ.ά., όπως φαίνεται στην Εικόνα 77.



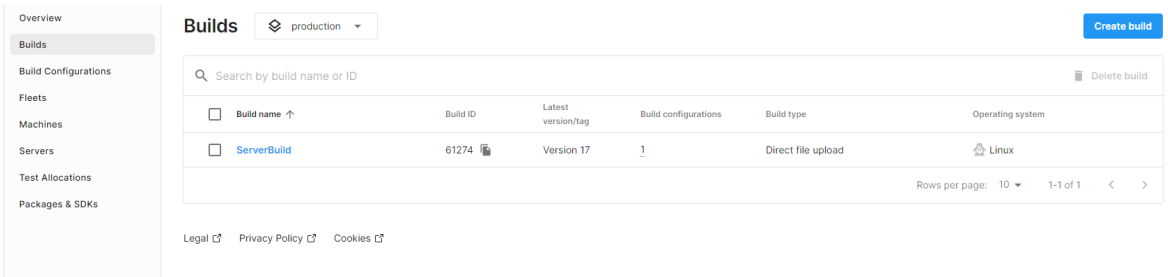
Εικονα 76 - Το dashboard του Multiplay



Εικονα 77 - Τα στοιχεια των Servers

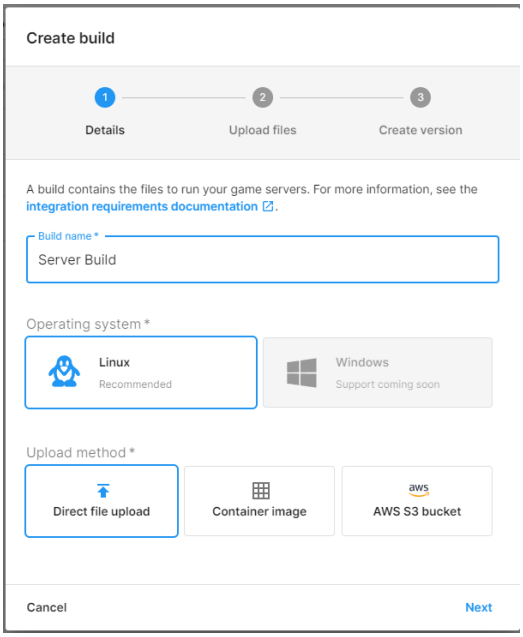
Έχοντας πλέον δημιουργήσει το build του server όπως περιεγράφηκε νωρίτερα, το μόνο που απομένει είναι το ανέβασμα των αρχείων που αποτελούν το build στους διαθέσιμους servers.

Συγκεκριμένα ο χρήστης χρειάζεται να πλοηγηθεί στην καρτέλα Builds και να δημιουργήσει ένα νέο Build επιλέγοντας Create Build. (Εικόνα 78)



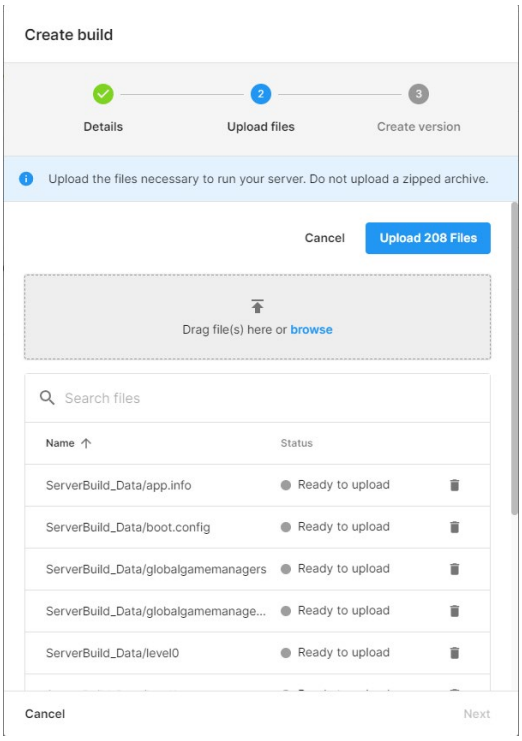
Εικονα 78 - Τα διαθεσιμα builds

Το μόνο που απομένει είναι να ονομάσει το νέο Build και να επιλέξει τον τρόπο με τον οποίο θα ανεβάσει τα αρχεία, όπως φαίνεται στην Εικόνα 79.



ΕΙΚΟΝΑ 79 - Η ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ SERVER BUILD

Τέλος, όπως φαίνεται και στην Εικόνα 80, χρειάζεται απλώς να συρθούν τα αρχεία και οι φάκελοι του build στο αντίστοιχο πεδίο. Μόλις αυτό ολοκληρωθεί, το παιχνίδι θα βρίσκεται στους servers που έχει επιλέξει ο χρήστης κατά τη διαδικασία ρύθμισης του Unity Gaming Services.

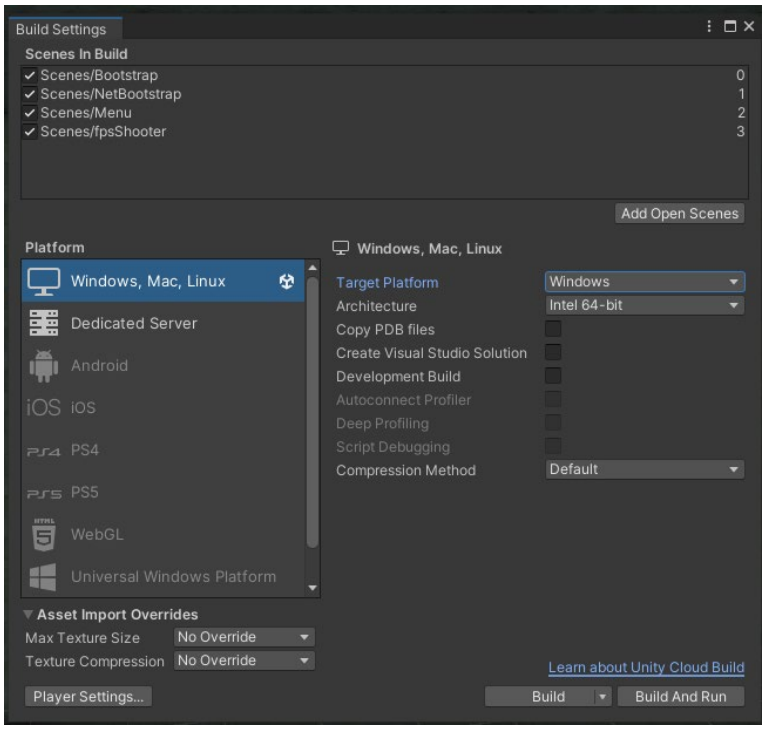


ΕΙΚΟΝΑ 80 - ΑΝΕΒΑΣΜΑ ΤΩΝ ΑΡΧΕΙΩΝ ΤΟΥ BUILD

4.2 Client Build

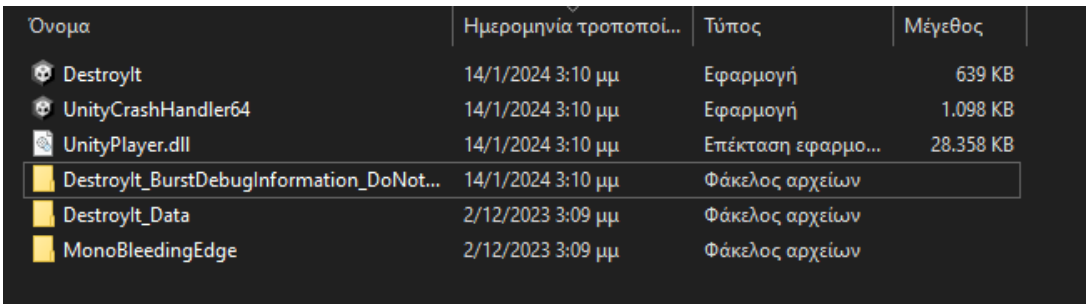
Η περίπτωση του build που θα χρησιμοποιείται από τους παίκτες προκειμένου να παίζουν το παιχνίδι είναι πιο απλή. Είναι το build το οποίο θα διανέμεται είτε απευθείας είτε μέσω κάποιας πλατφόρμας για κατέβασμα μεταξύ των παικτών, ώστε να έχουν πρόσβαση σε αυτό.

Συγκεκριμένα, όπως φαίνεται στην Εικόνα 81, οι ρυθμίσεις του build διαφέρουν από αυτές των servers παραπάνω. Ειδικότερα, εφόσον το παιχνίδι απευθύνεται σε υπολογιστές με λειτουργικό σύστημα Windows, θα πρέπει το Target Platform να είναι το αντίστοιχο.



Εικονα 81 - Οι ρυθμισεις του Client Build

Τα αρχεία που παράγονται από το client build θα μοιάζουν με αυτά της Εικόνα 82, μεταξύ των οποίων θα βρίσκεται και ένα εκτελέσιμο (.**exe**) αρχείο, μέσω του οποίου θα γίνεται η εκκίνηση του παιχνιδιού.



Εικονα 82 - Τα αρχεια του Client Build

4.3 Χειρισμός παιχνιδιού

Έχοντας στην κατοχή του ο χρήστης το παραπάνω εκτελέσιμο θα μπορεί πλέον να παίξει κανονικά το παιχνίδι. Συγκεκριμένα, στη πρώτη οθόνη που θα αντικρύσει θα εισάγει το όνομα με το οποίο θα φαίνεται στους υπόλοιπους παίκτες. Στη συνέχεια, θα επιλέξει τον τρόπο με τον οποίο θα συνδεθεί στο παιχνίδι, δηλαδή είτε μέσω κάποιου dedicated server, είτε μέσω του δωματίου κάποιου άλλου παίκτη, είτε τέλος, να δημιουργήσει ο ίδιος ένα δωμάτιο στο οποίο θα συνδέονται άλλοι παίκτες. Οποιαδήποτε επιλογή και αν κάνει, η επόμενη σκηνή μόλις προχωρήσει θα είναι η κύρια σκηνή του παιχνιδιού. Εκεί ο χειρισμός είναι αρκετά απλός και ακολουθεί τα πρότυπα όλων των σύγχρονων παιχνιδιών που χρησιμοποιούν το ποντίκι και το πληκτρολόγιο. Ειδικότερα, ο παίκτης θα κινείται εμπρός, πίσω, δεξιά, αριστερά μέσω των πλήκτρων W,S,D,A, θα περιστρέφεται τόσο ο ίδιος όσο και η κάμερα μέσω της κίνησης του ποντικιού, θα τρέχει πατώντας μαζί με τα πλήκτρα κίνησης, και το πλήκτρο LShift, και τέλος θα κάνει άλμα πατώντας το πλήκτρο Space. Σε ότι αφορά το κομμάτι των πυροβολισμών, απαιτείται το πάτημα του δεξιού κλικ του ποντικιού προκειμένου ο παίκτης να στοχεύσει, και στη συνέχεια το πάτημα του αριστερού κλικ ώστε να δημιουργηθεί και το αντικείμενο – σφαίρα.

Συμπεράσματα

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτέλεσε η ανάπτυξη ενός βιντεοπαιχνιδιού το οποίο απευθύνεται σε πολλαπλούς παίκτες. Κύριος στόχος της εργασίας, ήταν η παρουσίαση και η εκτενής μελέτη όλων των απαραίτητων εργαλείων και βιβλιοθηκών, για την επίτευξη της παραπάνω λειτουργίας. Επιδιώκοντας παράλληλα τη δημιουργία μιας συναρπαστικής, για τους παίκτες, εμπειρίας, προστέθηκε στο παιχνίδι η δυνατότητα καταστροφής αντικειμένων, και ως εκ τούτου, η δυνατότητα μεταβολής του ευρύτερου περιβάλλοντος στο οποίο διεξάγεται το παιχνίδι. Βάση όλων αυτών, αποτέλεσε η μηχανή παιχνιδιών Unity, της οποίας τα εργαλεία και οι βιβλιοθήκες περιγράφονται στην παρούσα διπλωματική εργασία, αποδεικνύοντας περίτρανα τους λόγους για τους οποίους το Unity βρίσκεται στα κορυφαία και δημοφιλέστερα λογισμικά ανάπτυξης παιχνιδιών.

Τόσο η μηχανή παιχνιδιών Unity, όσο και το περιβάλλον του Unity Editor, παρέχουν ένα πλήθος εργαλείων και δυνατοτήτων, ικανών να συμβάλλουν στη δημιουργία οποιουδήποτε είδους παιχνιδιού. Μέσω της αξιοποίησης επιπρόσθετων πακέτων, όπως στην προκειμένη περίπτωση, το ProBuilder, το Unity μπορεί να μετατραπεί σε ένα πανίσχυρο λογισμικό, ένα πολυεργαλείο που συνδυάζει την ανάπτυξη παιχνιδιών με τη σχεδίαση αντικειμένων, πιστών και χαρακτήρων, με απώτερο σκοπό την δημιουργία ολοκληρωμένων παιχνιδιών, χωρίς την ανάγκη χρησιμοποίησης εξωτερικών λογισμικών. Μείζονος σημασίας επίσης, αποτέλεσε η μελέτη και αξιοποίηση της βιβλιοθήκης Netcode for GameObjects, μέσω της οποίας σχεδιάστηκε και υλοποιήθηκε ολόκληρη η διαδικτυακή υποδομή του παιχνιδιού. Στόχος της εργασίας είναι ο αναγνώστης να μπορεί να κατανοήσει τα διάφορα μοντέλα διασύνδεσης πολλαπλών παικτών (multiplayer) και στη συνέχεια να έχει τη δυνατότητα αξιοποίησης αυτών. Συμπερασματικά, είναι ιδιαίτερα σημαντική η κατανόηση των αρχιτεκτονικών διασύνδεσης μεταξύ των παικτών στα σύγχρονα παιχνίδια σε συνδυασμό με τα πλεονεκτήματα και τα μειονεκτήματα της καθεμιάς. Τέλος, απόρροια της παρούσας διπλωματικής εργασίας, ήταν η αναζήτηση και έρευνα του διαθέσιμου υλικού που παρέχεται μέσω των διαφόρων προγραμματιστικών κοινοτήτων. Οι διαθέσιμες πηγές πλέον είναι άφθονες και οι πόροι που μπορεί κανείς να αξιοποιήσει, αγγίζουν όλες τις πτυχές ενός ολοκληρωμένου παιχνιδιού. Οι διαθέσιμες βιβλιοθήκες, όπως για παράδειγμα το RayFire που χρησιμοποιήθηκε στην παρούσα εργασία, είναι ικανές να αποτελέσουν βασικούς πυλώνες οικοδόμησης ενός παιχνιδιού.

Συμπερασματικά, η ανάπτυξη, συγγραφή και οργάνωση του παρόντος πονήματος αποτέλεσε μια συναρπαστική για εμένα εμπειρία, ξεπερνώντας συνεχώς τις αρχικές μου προσδοκίες. Η συνεχής μελέτη και αναζήτηση πηγών, αποτέλεσε έναν διαρκή άθλο, εντρυφώντας παράλληλα σε

νέους, αθέατους μέχρι πρότινος τομείς, προσφέροντας με αυτόν τον τρόπο περισσότερη αξία στην παρούσα εργασία. Ο χώρος της ανάπτυξης παιχνιδιών αποτελεί έναν συναρπαστικό κόσμο, ο οποίος καθημερινά προοδεύει και εξελίσσεται, κάνοντας έτσι την εναρμόνιση με τα παγκόσμια δεδομένα πραγματικά δοκιμασία. Οι τεχνολογίες του χώρου συνεχώς εξελίσσονται, προσφέροντας συναρπαστικές δυνατότητες, ικανές να ανταποκριθούν σε ό,τι φαντάζεται ο ανθρώπινος νους. Απώτερος στόχος και ελπίδα του συγγραφέα της παρούσας εργασίας είναι, η τρέχουσα μορφή του παιχνιδιού να αποτελέσει βάση για περαιτέρω μελέτη και αξιοποίηση, κάνοντας κάθε στοιχείο του παιχνιδιού αντάξιο ενδελεχούς ανάλυσης και βελτίωσης, για την δημιουργία τελικά, ενός παιχνιδιού ικανού να εκδοθεί και να ανταγωνιστεί άλλα βιντεοπαιχνίδια του αντίστοιχου βελησεκούς.

Πηγές - Βιβλιογραφία

[1] Hocking, J., 2022. Unity in Action Third Edition.

[2] Menard, M. and Wagstaff, B., 2014 Game development with Unity.

[3] RayFire [Internet]. Rayfire Studios. [cited 2024Feb] Available from https://rayfirestudios.com/

[4] ProBuilder [Internet]. Unity. [cited 2024Feb] Available from https://docs.unity3d.com/Packages/com.unity.probuilder@4.0/manual/index.html

[5] Netcode for GameObjects [Internet]. Unity. [cited 2024Feb] Available from: https://docs-multiplayer.unity3d.com/netcode/current/about/

[6] Make Online Games Using Unity's NEW Multiplayer Framework [Internet]. Udemy. [cited 2024Feb] Available from: https://www.udemy.com/course/unity-multiplayer-netcode/learn/lecture/38410702#overview

[7] Third-person shooter [Internet]. Wikipedia. Wikimedia Foundation; 2024 [cited 2024Feb]. Available from: https://en.wikipedia.org/wiki/Third-person_shooter

[8] Online PC gaming market value worldwide from 2011 to 2027 [Internet]. Statista. [cited 2024Feb] Available from: https://www.statista.com/statistics/292516/pc-online-game-market-value-worldwide/