ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Διπλωματική Εργασία (ΠΜΣ στην Επιστήμη Υπολογιστών)

Καλλίνης Άγγελος 20222018009

Επιβλέπων: Αναπληρωτής Καθηγητής Τσελίκας Νικόλαος

Σχεδιασμός και Υλοποίηση Συστήματος Καταγραφής, Παρακολούθησης και Διαχείρισης Στίγματος Κινητών Χρηστών σε Εσωτερικό Χώρο

Ιανουάριος 2021

Περίληψη

Το θέμα της παρούσας μεταπτυχιακής διατριβής πραγματεύεται τον σχεδιασμό και την υλοποίηση διαδικτυακής (web) εφαρμογής, ικανής να καταγράφει ανθρώπινη κίνηση σε ορισμένο χώρο, να αποθηκεύει και να επεξεργάζεται τα δεδομένα, να την αποτυπώνει στο διδιάστατο επίπεδο και να εξάγει συμπεράσματα. Το παραπάνω σύστημα θεωρητικά βρίσκει μεγάλη ποικιλία εφαρμογών ειδικά σε περιπτώσεις που η παρακολούθηση κίνησης σε έναν περιορισμένο χώρο είναι επιβεβλημένη για λόγους αποδοτικότητας. Παράδειγμα εφαρμογής είναι ένας εκθεσιακός ή μουσειακός χώρος που η παρακολούθηση της κίνησης των επισκεπτών και η εξαγωγή μοτίβων θα βοηθούσε στην καλύτερη κατανόηση της συμπεριφοράς τους και των ενδιαφερόντων τους και στην εκ των πραγμάτων καλύτερη τοποθέτηση των εκθεμάτων. Τα αποτελέσματα για το σκοπό της εργασίας βασίζονται σε τεχνητά δεδομένα.

Λέξεις κλειδιά

Παρακολούθηση κίνησης, Εξομοίωση κίνησης, Θερμογράφημα, Εξαγωγή μοτίβου, heatmap.js, j query, javascript, mysql

Abstract

The following MSc thesis deals with the design and development of a web application capable of capturing human movement in a set environment, storing and processing data, representing it in two-dimensions and finally extracting conclusions. Such an application can be used in a wide range of cases, especially when the need of monitoring human movement in a contained environment is imposed for efficiency purposes. A nice use case could be an exhibition room or museum, where movement monitoring of visitors and the extraction of patterns could help in better understanding their behavior and interests resulting in better placement of the exhibits. For the purpose of this application we rely on generated artificial data.

Key words

Motion tracking, Motion simulation, Heatmap, Pattern extraction, heatmap.js, jquery, javascript, mysql

Περιεχόμενα

Κεφάλαιο 1

1.1 Εισαγωγή

Η εφαρμογή αποτελεί ένα ολοκληρωμένο σύστημα με δυνατότητες που αφορούν στη σχεδίαση χώρου κίνησης και στην τοποθέτηση σημείων πρόσβασης (access points) πάνω σ'αυτόν. Επιτρέπει τη διεξαγωγή προσομοιώσεων και παραγωγή δεδομένων με χωροχρονικό χαρακτήρα. Οι προσομοιώσεις περιλαμβάνουν την αποτύπωση της κίνησης σε πραγματικό χρόνο. Από αυτές τις κινήσεις εξάγονται βασικές πληροφορίες στατιστικού χαρακτήρα, όπως, κατά σειρά σημαντικότητας: α) Θερμογράφημα, β) Συχνότερη διαδρομή, γ) Δημοφιλέστερο σημείο πρόσβασης. Τα δεδομένα αυτά μπορούν να βοηθήσουν στην εξαγωγή συμπερασμάτων που αφορούν τη συμπεριφορά των υποκειμένων στο χώρο.

1.2 Στόχος

Στόχος της εργασίας είναι να σχεδιάσουμε ένα εργαλείο που θα βοηθάει στην αύξηση της αποδοτικότητας ενός οργανισμού ή μιας επιχείρησης που διαθέτει εκθέματα ή προς πώληση προϊόντα σε εσωτερικό χώρο, αναδεικνύοντας ταυτόχρονα και τις δυνατότητες της βιβλιοθήκης `heatmap.js`.

1.3 Εργαλεία ανάπτυξης

Από τεχνικής άποψης θα χρησιμοποιηθούν κυρίως JavaScript, jQuery, AJAX, HTML 5, CSS 3, η βιβλιοθήκη `heatmap.js` και κάποιες άλλες βοηθητικές βιβλιοθήκες όπως η `Chart.js` και η `Fabric.js`. Από την πλευρά του server θα χρησιμοποιηθεί PHP και mySQL.

1.4 Σχεδιασμός

Σε αυτη την ενότητα θα εξηγήσουμε τις βασικές λειτουργίες της εφαρμογής ως προς τη λειτουργικότητά τους.

1.4.1 Το concept

Όταν αναφερόμαστε σε μια εφαρμογή καταγραφής της κίνησης το πρώτο πράγμα που μας απασχολεί στον σχεδιασμό είναι ο τρόπος εκείνος που θα

παρακολουθήσουμε το υποκείμενο. Για να γίνει αυτό όμως πρέπει πρώτα να υπάρχει ο χώρος που εκείνο θα κινηθεί. Για τον σκοπό αυτό θα δημιουργήσουμε ένα εργαλείο εισαγωγής χώρου.

1.4.2 Δημιουργία χάρτη

Εκμεταλευόμενοι τις δυνατότητες της HTML 5 σ' αυτόν τον τομέα, ορίζουμε έναν χώρο σχήματος ορθογωνίου παραλληλογράμου, τα όρια του οποίου θα είναι και ο μέγιστος δυνατός χώρος για τις προσομοιώσεις. Επάνω σ' αυτόν τον χώρο ο χρήστης κάνοντας παρατεταμένο κλικ μπορεί να ορίσει την αρχή ενός εμποδίου. Αφήνοντας το ποντίκι στο σημείο που επιθυμεί, ορίζει το τέλος του εμποδίου. Τα δεδομένα για τα εμπόδια στο διδιάσταστο χώρο αποθηκεύονται ως ζεύγη συντεταγμένων (x1, y1) (x2, y2) και η τιμή τους είναι κάθε φορά το πίξελ της οθόνης πχ. (350, 300). Στη συνέχεια, τα ζεύγη των συντεταγμένων δηλαδή οι τετράδες, αποθηκεύονται στη βάση δεδομένων.

1.4.3 Εισαγωγη Σημείων πρόσβασης

Αφού ο χάρτης με τα εμπόδια είναι αποθηκευμένος, σειρά έχουν τα σημεία πρόσβασης (**access points**). Με ένα κλικ ο χρήστης ορίζει αρχικά το σημείο όπου επιθυμεί να βρίσκεται η είσοδος/έξοδος των επισκεπτών (μόνο μία είσοδος/έξοδος υποστηρίζεται). Η είσοδος έξοδος παίρνει πάντα το αναγνωριστικό 1. Τα υπόλοιπα σημεία έχουν και αυτά τον αύξοντα αριθμό που τα προσδιορίζει. Για κάθε σημείο πρόσβασης, ο χρήστης καλείται επιπλέον να ορίσει και τους γείτονές του. Γείτονες του κάθε σημείου πρόσβασης είναι τα σημεία πρόσβασης εκείνα που ο επισκέπτης μπορεί να μετακινηθεί απ' ευθείας.

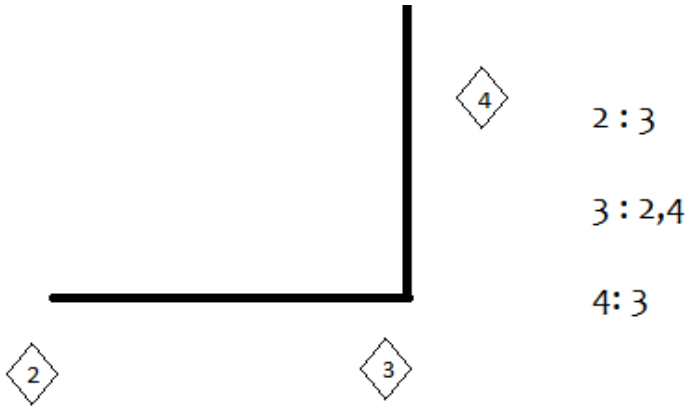


Figure 1 Κινήσεις στον χώρο και γειτνίαση

Όπως φαίνεται και στην παραπάνω εικόνα, ένας επισκέπτης ευρισκόμενος στο σημείο 2 μπορεί να μετακινηθεί προς το σημείο 3. Από το σημείο 3 μπορεί είτε να προχωρήσει προς το σημείο 4 ή να γυρίσει πίσω στο 2 και απ το σημείο 4 μπορεί να γυρίσει πίσω στο 3. Για κάθε σημείο λοιπόν ο χρήστης συμπληρώνει τα απ'ευθείας γειτονικά του. Κάθε σημείο πρόσβασης αποθηκεύεται στη βάση και αναπαρίσταται από τις συντεταγμένες του, όπως ακριβώς περιγράψαμε και με τα εμπόδια. Επίσης για κάθε σημείο πρόσβασης εκτός από τον αύξοντα αριθμό και τις τιμές για τα x και y, αποθηκεύονται: ο συνολικός αριθμός επισκέψεων, ο συνολικός χρόνος επισκέψεων, η ημερομηνία και ένας κωδικός κατάστασης (θα εξηγηθεί παρακάτω η χρησιμότητά του).

1.4.4 Πίνακας Γειτνίασης

Ένα από τα πιο βασικά κομμάτια της τροφοδοσίας του αλγορίθμου που θα προσομοιώνει την κίνηση είναι ο **πίνακας γειτνίασης** που εξάγεται αυτόματα απ' τα δεδομένα των γειτόνων του κάθε σημείου πρόσβασης. Θα εξηγήσουμε τη λειτουργία του με ένα παράδειγμα. Έστω τα εξής σημεία πρόσβασης με τους γείτονές τους.

1:2,3,4

2:3,4

3:2,4,5

4:2,3,5

5:3,4

Παρατηρούμε πως οι ίδιες πληροφορίες συναντώνται πάνω από μία φορά για παράδειγμα οι γείτονες του σημείου πρόσβασης 2 μας δείχνουν μετάβαση προς το 3 και εκείνοι του 3 μετάβαση προς το 2. Επειδή στην παρούσα φάση δεν μας επηρεάζει η έννοια της κατεύθυνσης μπορούμε να παραλείψουμε τη μια πληροφορία. Σύμφωνα με την παραπάνω λογική ο πίνακας γειτνίασης θα είχε την ακόλουθη μορφή:

1:2,3,4

2:3,4

3:4,5

4:5

5:-

Η τελική μορφή βέβαια ενός πίνακα γειτνίασης είναι ένας τετραγωνικός πίνακας από άσσους και μηδενικά. Η παραπάνω μορφή είναι μια αναπαράσταση που θα βοηθήσει στην εξαγωγή του.

1.4.5 Παραγωγή τεχνητών δεδομένων προσομοίωσης

Ο αλγόριθμος που θα παράγει τα τεχνητά δεδομένα κίνησης των επισκεπτών στον χώρο λαμβάνει συγκεκριμένη είσοδο. Όλα τα δεδομένα, εκτός από τον πίνακα γειτνίασης, δίνονται απ' το χρήστη. Αυτά είναι: ο αριθμός των επισκεπτών, ο αριθμός των αλμάτων ανάμεσα στα σημεία πρόσβασης που οι τελευταίοι θα πραγματοποιήσουν, ο ελάχιστος και μέγιστος χρόνος παραμονής σε ένα σημείο πρόσβασης (επιλέγεται τυχαίος αριθμός από το κλειστό σύνολο) και η πιθανότητα επιστροφής σε προηγούμενο σημείο. Πρώτα δημιουργεί τους επισκέπτες και τους αναθέτει μια λίστα με τα σημεία πρόσβασης που έχουν επισκεφθεί. Στη συνέχεια επιλέγει τυχαία έναν επισκέπτη, εξετάζει τα σημεία που έχει περάσει, την πιθανότητα επιστροφής και σύμφωνα με τον πίνακα γειτνίασης αναθέτει το προς επίσκεψη σημείο μαζί με έναν τυχαίο χρόνο παραμονής. Όταν το πλήθος της λίστας των σημείων που έχει επισκεφθεί ένας επισκέπτης ισούται με τον αριθμό των αλμάτων που έχει δώσει ο χρήστης, τότε η κίνησή του ολοκληρώνεται και εκείνος διαγράφεται απ' τη μνήμη.

1.4.6 Ο αλγόριθμος κίνησης στο χώρο

Για να είναι δυνατή η κίνηση στο χώρο προφανώς δεν αρκούν μόνο τα παραπανω τεχνητά δεδομένα καθώς δεν προσδίδουν καμμία απολύτως πληροφορία για το μονοπάτι που θα ακολουθήσει το υποκείμενο από το σημείο έναρξης έως το σημείο προορισμού. Τα μόνα που γνωρίζουμε προς το παρόν είναι οι συντεταγμένες σημείων έναρξης λήξης και ο χρόνος παραμονής. Είναι απαραίτητο να μπορούμε να ανιχνεύσουμε τα εμπόδια σε μια τέτοια κίνηση, ώστε να μοιάζει με τη φυσική. Τα παραδείγματα που φαίνονται στις παρακάτω εικόνες εξηγούν τον τρόπο με τον οποίο ο αλγόριθμος αντιλαμβάνεται τον χώρο και τα εμπόδια, αλλά και πώς επιλέγει το μονοπάτι ανάμεσα στα σημεία 1 και 2.

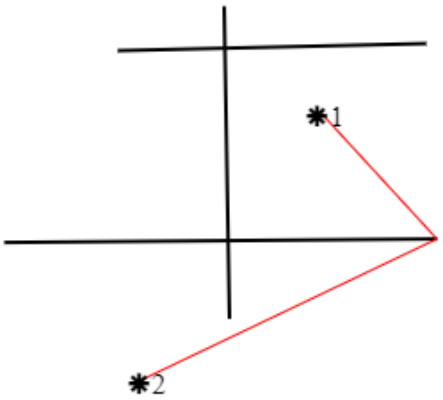


Figure 2 Παράδειγμα διαδρομής 1

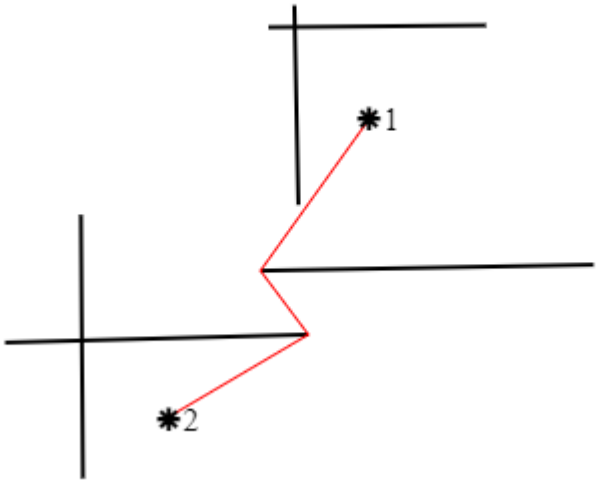


Figure 3 Παράδειγμα διαδρομής 2

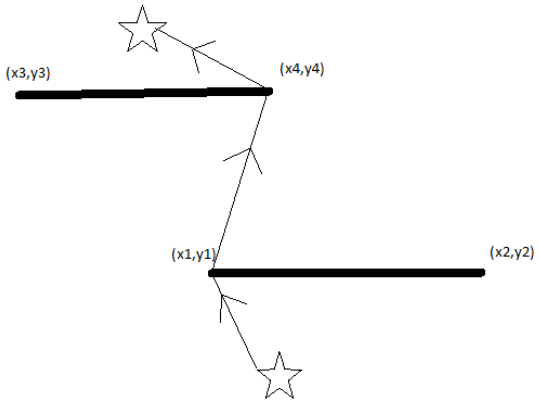


Figure 4 Παράδειγμα διαδρομής 3

Πριν αναλύσουμε τον τρόπο λειτουργίας του αλγορίθμου κίνησης, υπάρχουν δυο βοηθητικοί αλγόριθμοι που βοηθούν στην ανίχνευση σύγκρουσης.

1.4.7 Ο αλγόρθμος τομής δύο ευθειών

Από τα μαθηματικά γνωρίζουμε πως δυο ευθείες y = a x + c και y = b x + d με a,b να είναι η κλίση της ευθείας και c,d οι τιμές του άξονα των y, τέμνονται στο σημείο P $\left(\frac{d-c}{a-b}, a\frac{d-c}{a-b}+c\right)$. Στην πράξη όμως το μόνο που γνωρίζουμε για τις ευθείες είναι τα ζεύγη των συντεταγμένων τους. Όταν για δύο ευθείες L1, L2 γνωρίζουμε τις καρτεσιανές συντεταγμένες τους μπορούμε με τη μέθοδο των οριζουσών να βρούμε το σημείο τομής τους. Προκύπτει ότι για τις ευθείες L1 (x1,y1)(x2,y2) και L2 (x3,y3)(x4,y4) η τομή βρίσκεται απ' τον τύπο :

$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

Ο αλγόριθμος υλοποιεί τις δύο ευθείες και υπολογίζει την τομή τους.

1.4.8 Ο αλγόριθμος του Bresenham

Είναι ένας αλγόριθμος που υπολογίζει τα πίξελ που πρέπει να μεσολαβήσουν ώστε να έχουμε την ευθεία που ορίζεται από δύο σημεία. Ο ψευδοκώδικας για τον αλγόριθμο είναι ο εξής:

```
function line(x0, y0, x1, y1)
    real deltax := x1 - x0
    real deltay := y1 - y0
    real deltaerr := abs(deltay / deltax)    // Assume
deltax != 0 (line is not vertical),
          // note that this division needs to be done in a way
that preserves the fractional part
    real error := 0.0 // No error at start
    int y := y0
    for x from x0 to x1
        plot(x, y)
        error := error + deltaerr
        if error ≥ 0.5 then
            y := y + sign(deltay)
            error := error - 1.0
```

Το αποτέλεσμα του αλγορίθμου μοιάζει με την παρακάτω εικόνα .

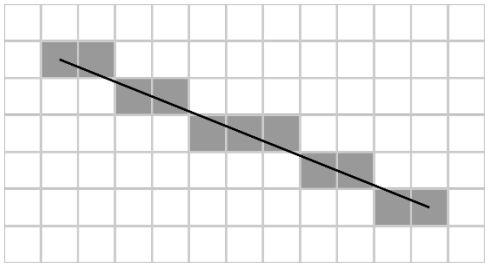


Figure 5 Αποτέλεσμα Bresenham

Στην πράξη οι παραπάνω αλγόριθμοι θα συνδυαστούν για την ανίχνευση εμποδίων και τον υπολογισμό των μονοπατιών.

1.4.9 Ο αλγόριθμος κίνησης με αποφυγή εμποδίων

Η τετριμμένη περίπτωση είναι το σημείο έναρξης και το σημείο προορισμού να έχουν οπτική επαφή, δηλαδή η νοητή ευθεία που τα ενώνει να μην τέμνει καμμία ευθεία-εμπόδιο. Τότε απλά μπορούμε να υποθέσουμε με ασφάλεια πως η διαδρομή είναι τα δυο σημεία.

Η πιο σύνθετη περίπτωση περιλαμβάνει, η νοητή ευθεία ανάμεσα στα σημεία να τέμνει μία ή παραπάνω ευθείες από εμπόδια. Αρχικός μας στόχος είναι να δημιουργήσουμε μια μορφή **γράφου ορατότητας ή visibility graph**. Με τον όρο γράφος ορατότητας εννοούμε όλα εκείνα τα σημεία που ένα σημείο του επιπέδου μπορεί να «δει» , δηλαδή διαγράφοντας μία νοητή ευθεία ανάμεσά τους, δεν παρεμβάλεται κανένα εμπόδιο. Προφανώς και αυτά τα σημεία είναι οι άκρες των ευθειών-εμποδίων καθώς δεν θα είχε νόημα π.χ. το κέντρο. Δουλεύοντας σε γύρους δοκιμάζουμε κάθε φορά το τρέχον σημείο με όλες τις άκρες όλων των ευθειών εμποδίων. Έστω σημείο εκκίνησης το σημείο πρόσβασης 1. Ο πρώτος γύρος συγκρίσεων των νοητών ευθειών απο το 1 με όλες τις άκρες των ευθειών εμποδίων φαίνονται παρακάτω.

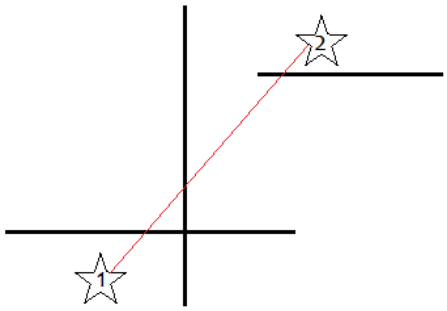


Figure 6 Εύρεση επόμενου σημείου στάδιο 1

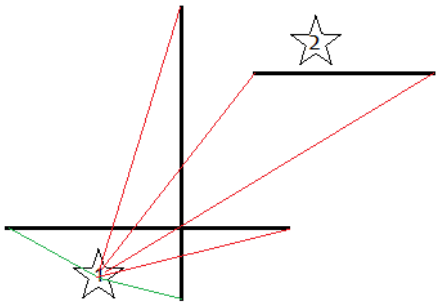


Figure 7 Εύρεση επόμενου σημείου στάδιο 2

Με κόκκινο χρώμα είναι οι νοητές ευθείες που βρέθηκαν τομές σε σημειο διαφορετικό απ'τις άκρες και δε θα συμπεριληφθούν στο γράφο ορατότητας. Αντιθέτως τα σημεία στις άκρες των πράσινων ευθυγράμμων τμημάτων συμπεριλαμβάνονται στον γράφο καθώς δεν τέμνουν ευθείες εμπόδια σε σημεία εκτός των άκρων. Το συμπέρασμα που προκύπτει είναι πως για να συμπεριληφθεί ένα σημείο στο γράφο ορατότητας πρέπει υποχρεωτικά και μόνον να τέμνει μια ακριβώς ευθεία-εμπόδιο σε ένα από τα δύο άκρα της.

Το **κριτήριο επιλογής** του επόμενου σημείου (απ'τα δύο διαθέσιμα του γράφου) είναι η απόστασή του απ' το σημείο 2. Εκ των πραγμάτων το δεξιά σημείο θα είναι το επόμενο που θα επιλεγεί μπαίνοντας στη λίστα της τρέχουσας διαδρομής. Το γεγονός αυτό θα σημάνει τον τερματισμό του γύρου. Στους επόμενους γύρους

επαναλαμβάνονται τα παραπάνω βήματα μέχρις ότου να βρεθεί σημείο που η νοητή ευθεία του με το σημείο προορισμού δηλαδή το 2, να τέμνεται μονάχα στο σημείο εκείνο που είναι και το τρέχον.

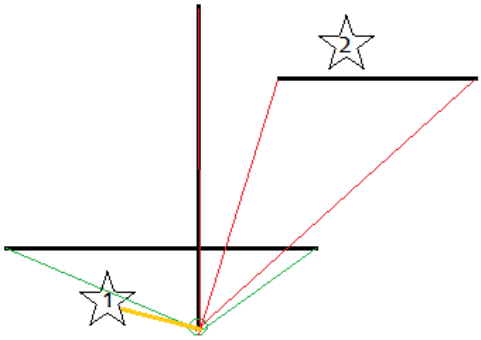


Figure 8 Εύρεση επόμενου σημείου στάδιο 3

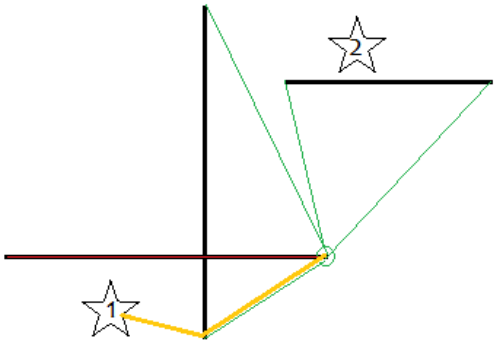


Figure 9 Εύρεση επόμενου σημείου στάδιο 4

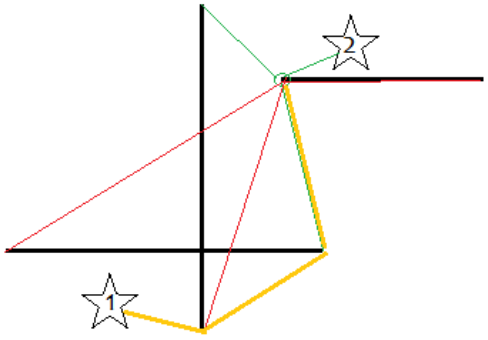


Figure 10 Εύρεση επόμενου σημείου στάδιο 5

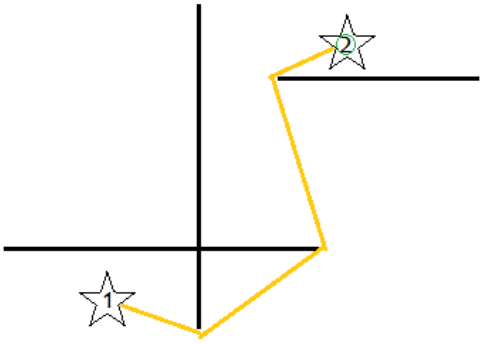


Figure 11 Εύρεση επόμενου σημείου στάδιο 6

Με πράσινο χρώμα – σημεία που ανήκουν στο γράφο ορατότητας. Με κόκκινο χρώμα – σημεία που δεν ανήκουν στο γράφο ορατότητας. Με κίτρινο χρώμα σημεία που έχουν καταχωρηθεί ως επιλεγμένη διαδρομή. Ο αλγόριθμος δεν εξετάζει την εύρεση βέλτιστου μονοπατιού απ' το 1 στο 2. Πριν από κάθε γύρο πρώτα δοκιμάζεται η νοητή ευθεία του τρέχοντος σημείου με το σημείο 2 ανεξαρτήτως θέσης. Σε περίπτωση που βρεθούν τομές διενεργείται η διαδικασία του γράφου ορατότητας. Σε περίπτωση που ένα σημείο υπάρχει στη λίστα διαδρομής τότε δεν επιλέγεται ξανά αλλά το επόμενο στη λίστα με κριτήριο πάλι την απόσταση απ' το 2.

1.4.10 Εύρεση λοιπών μονοπατιών

Η παραπάνω διαδικασία επαναλαμβάνεται όσες φορές χρειαστεί μέχρις ότου έχουμε ένα μονοπάτι για κάθε ζεύγος γειτόνων του πίνακα γειτνίασης. Άρα για έναν τυχαίο πίνακα γειτνίασης όπως ο παρακάτω, ο αλγόριθμος θα ταιριάξει τις συντεταγμένες των 1-2,1-3,2-3,2-4,3-4,4-2 απ' τη βάση δεδομένων και θα προσπαθήσει να βρεί για κάθε ένα ζεύγος, το μονοπάτι χωρίς εμπόδια.

1:2,3

2:3,4

3:2,4

4:2,3

1.4.11 Προσομοίωση σε πραγματικό χρόνο

Ο αλγόριθμος αυτός εκτελεί στην πραγματικότητα μια σειρά κινήσεων σε ορισμένη ευθεία και παύσεων, όσο χρόνο χρειαστεί. Για κάθε επισκέπτη δημιουργείται μια **λίστα κινήσεων** που θα ακολουθήσουν σε όλη τη διάρκεια της επίσκεψης. Η λίστα αυτή είναι της μορφής **[{x:100,y:150},{w:15000},{x:200,y:200},{x:245,y:300},{w:9000}].** Η μετάφραση του πίνακα : ξεκίνημα απ' τη θέση 100,150 και αναμονή 15 δευτερόλεπτα, μετακίνηση στο σημείο 200,200, μετακίνηση στο σημείο 245,300 και αναμονή 9 δευτερόλεπτα. Τα δεδομένα της λίστας είναι συνδυασμός των δεδομένων διαδρομής που εξήγαγε ο αλγόριθμος για τη διαδρομή από το σημείο 1 στο 2 και από τους χρόνους αναμονής που επιλέχθηκαν τυχαία απ' το κλειστό σύνολο. Αυτές οι λίστες διαβάζονται σειριακά και με τη βοήθεια jQuery animations βλέπουμε αλυσιδωτές κινήσεις στις κατάλληλες θέσεις στην οθόνη.

1.4.12 Αποτύπωση πληροφορίας-Θερμογράφημα

Θερμογράφημα αποτελεί μια τεχνική οπτικοποίησης της πληροφορίας που δείχνει το μέγεθος ενός φαινομένου σε δύο διαστάσεις. Συνήθως χρησιμοποιείται σε οπτικοποιήσεις που έχουν να κάνουν με κίνηση στον χώρο, αλλά όχι μόνο. Ιδιαίτερη εφαρμογή βρίσκουμε και στην απεικόνηση της κίνησης του κέρσορα σε μια ιστοσελίδα ώστε να γίνονται αντιληπτά τα ενδιαφέροντα των επισκεπτών. Διάφορα χρώματα απαντώνται στις κλίμακές τους με συνηθέστερα το μπλέ, το κόκκινο, το πράσινο και το κίτρινο. Στις εφαρμογές που οπτικοποιούν κίνηση συνήθως τα πιο ερυθρά χρώματα όπως κίτρινο και πορτοκαλί απεικονίζουν τις μεγαλύτερες τιμές της κλίμακας εξ'ου και το θερμό του χαρακτήρα ενώ αντιστοίχως τα ψυχρά χρώματα που τείνουν στο κυανό απεικονίζουν τις χαμηλές τιμές.

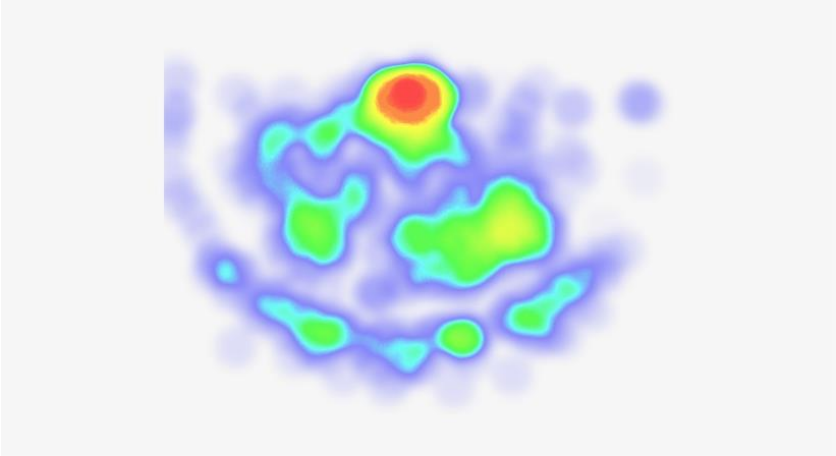


Figure 12 Θερμογράφημα

1.4.13 Διαδικασία της οπτικοποίησης

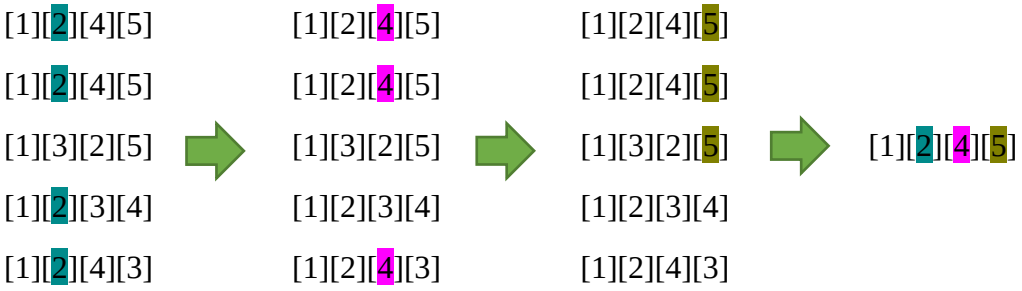
Ύστερα από αρκετές προσομοιώσεις (ή ακόμη και από μία) έχουμε αρκετά δεδομένα για το κάθε σημείο πρόσβασης, ώστε να δημιουργήσουμε μία απεικόνιση. Μετά το τέλος κάθε προσομοίωσης έχουμε ένα συνολικό αριθμό επισκέψεων και τον συνολικό χρόνο που οι επισκέπτες δαπάνησαν κοντά του. Ο **συνολικός χρόνος** επιλέχθηκε ως η κλίμακα για λόγους απλότητας. Ο **αριθμός των επισκέψεων** λαμβάνεται υπόψιν ως ένας τρόπος καθορισμού του πλάτους των σημείων του θερμογραφήματος. Τα σημεία πρόσβασης δίνουν τα κέντρα των σημείων. Πρακτικά κάθε σημείο πρόσβασης αντικαθίσταται από ένα αποτύπωμα του θερμογραφήματος με ανάλογο πλάτος. Τα υπόλοιπα σημεία είναι οι διαδρομές ανάμεσα στα σημεία πρόσβασης. Χρησιμοποιείται ο αλγόριθμος προσομοίωσης για να βρεθούν τα κατάλληλα μονοπάτια. Έπειτα, ο αλγόριθμος του Bresenham βρίσκει τα σημεία που δημιουργούν την ευθεία. Κάθε σημείο που επιστρέφει ο Bresenham είναι και ένα σημείο στο θερμογράφημα με μία συγκεκριμένη τιμή. Στην πραγματικότητα και για λόγους απόδοσης, επειδή για μια ευθεία επιστρέφονται εκατοντάδες σημεία, δεν είναι απαραίτητο να οπτικοποιούνται όλα.

1.4.14 Παλαιότερες καταστάσεις

Η εφαρμογή δίνει τη δυνατότητα προβολής παλαιότερων καταστάσεων αθροιστικά. Κάθε φορά που «τρέχει» μία προσομοίωση, αυτή αποθηκεύεται στη βάση δεδομένων ως ξεχωριστή κατάσταση με βάση την ημερομηνία και σε όλες τις τιμές της τρέχουσας κατάστασης δίνεται ένας κωδικός. Η επιλογή Aggregate δίνει αθροιστικά όλα τα αποτελέσματα απ’ την πρώτη έως την τελευταία.

1.4.15 Δημοφιλέστερη διαδρομή

Από το μενού των ρυθμίσεων επιλέγεται ο αριθμός των σταδίων που θα έχει η δημοφιλέστερη διαδρομή. Ο δυναμικός χαρακτήρας της εφαρμογής φέρνει την ανάγκη μίας τέτοιας επιλογής καθώς δεν μπορούμε να εξάγουμε δημοφιλέστερη διαδρομή πέντε σταδίων σε μία προσομοίωση δέκα σημείων κ.ο.κ Όπως αναφέραμε και παραπάνω, κάθε επισκέπτης διατηρεί μία λίστα με τα σημεία πρόσβασης που έχει επισκεφθεί. Οι λίστες αυτές κρατώνται στη βάση και επεξεργάζονται ώστε να προκύψει η δημοφιλέστερη διαδρομή. Η διαδικασία έχει ως βασικά βήματα τη στοίχιση των λιστών σε πίνακες και τη σύγκριση των ψηφίων σύμφωνα με το εξής παράδειγμα. Κάθε γραμμή αναπαριστά διαδρομή επισκέπτη.



Στη συνέχεια, με τη βοήθεια του αλγορίθμου εύρεσης μονοπατιού σχεδιάζεται η διαδρομή.

1.4.16 Δημοφιλέστερα σημεία πρόσβασης

Η λειτουργία παρέχει τριών ειδών οπτικοποιήσεις. Ένα διαφορετικού χρωματισμού και απλουστευμένο θερμογράφημα, που όμως αναπαριστά το πλήθος των επισκέψεων σε κάθε σημείο πρόσβασης, ένα διάγραμμα μπάρας που απεικονίζει τον αριθμό των συνολικών επισκέψεων ανά σημείο πρόσβασης και ένα διάγραμμα πίτας με τον συνολικό χρόνο που δαπανήθηκε σε κάθε σημείο πρόσβασης.

Κεφάλαιο 2

2.1 Υλοποίηση

2.1.1 Η βάση δεδομένων

Η βάση δεδομένων περιέχει τέσσερις βασικούς πίνακες και έναν βοηθητικό. Τα ονόματά τους `access_pont`, `routes`, `obstacles`, `past_instances` και `settings`. Για να αποθηκεύσουμε ένα εμπόδιο που περιγράφεται από δύο ζεύγη συντεταγμένων χρειαζόμαστε τέσσερα πεδία `integer` και ένα ξεχωριστό `id`.

```
CREATE TABLE `obstacles` (
  `id` int(11) NOT NULL,
  `x1` int(11) NOT NULL,
  `y1` int(11) NOT NULL,
  `x2` int(11) NOT NULL,
  `y2` int(11) NOT NULL
)
```

Για να αποθηκεύσουμε ένα σημείο πρόσβασης χρειαζόμαστε ένα ζεύγος συντεταγμένων, ένα ξεχωριστό `id` με αύξοντα αριθμό, τις τιμές για το πλήθος των επισκέψεων (`times_visited`) και τον συνολικό χρόνο που δαπανήθηκε απ' τους επισκέπτες (`heat`). Αυτή είναι και η τιμή που αποτελεί τον δομικό λίθο του θερμογραφήματος.

```
CREATE TABLE `access_point` (
  `id` int(11) NOT NULL,
  `x` int(11) NOT NULL,
  `y` int(11) NOT NULL,
  `times_visited` int(11) NOT NULL,
  `heat` int(11) NOT NULL
)
```

Για να αποθηκεύσουμε τις καταστάσεις των προσομοιώσεων χρειαζόμαστε πάλι τα πεδία `times_visited` και `heat` μαζί με το `id_` το οποίο είναι απλά ένας αύξων αριθμός και το `id` του συγκεκριμένου σημείου. Ο λόγος που αποθηκεύονται δύο διαφορετικά `id` είναι δίοτι το πρώτο είναι απαραίτητο ως πρωτεύον κλειδί, ενώ το δεύτερο είναι το νούμερο του σημείου πρόσβασης που θα υπάρχει πολλές φορές στον πίνακα για τις πολλές καταστάσεις που θα παρεβρίσκονται. Επιπλέον σε ένα πεδίο

`date` αποθηκεύεται η ημερομηνία και ένα διαχωριστικό κατάστασης `state_id` που μας βοηθάει να γνωρίζουμε πότε ξεκινά και πότε τελειώνει μια προσομοίωση.

```
CREATE TABLE `past_instances` (
  `id_` int(11) NOT NULL,
  `id` int(11) NOT NULL,
  `times_visited` int(11) NOT NULL,
  `heat` int(11) NOT NULL,
  `date` text NOT NULL,
  `state_id` int(11) NOT NULL
)
```

Ο πίνακας των διαδρομών είναι απλός με δύο πεδία, έναν αύξοντα αριθμό και το αλφαριθμητικό `route` που δηλώνει την ακολουθία των σημείων πρόσβασης

```
CREATE TABLE `routes` (
  `id` int(11) NOT NULL,
  `route` varchar(256) NOT NULL
)
```

Τελευταίος ο πίνακας `settings` που είναι επικουρικού χαρακτήρα και κρατά πληροφορίες που άλλες είναι καθαρά προαιρετικές, ενώ άλλες είναι σημαντικές και βοηθούν στην εξέλιξη των διαδικασιών μέσα στην εφαρμογή. Η μεταβλητή `max_hops` έχει να κάνει με το μέγιστο μήκος της δημοφιλέστερης διαδρομής. Η μεταβλητή `max_radius` δηλώνει τη μέγιστη ακτίνα ενός σημείου του θερμογραφήματος και είναι ένας ακέραιος αριθμός. Οι αριθμοί κινητής υποδιαστολής `max_opacity` , `min_opacity` και `blur` είναι οπτικές επιλογές του θερμογραφήματος και αφορούν αντίστοιχα στην ελάχιστη και στη μέγιστη τιμή της αδιαφάνειας και την τιμή της «θολότητας» του κάθε σημείου. Οι τιμές `color1`, `color2`, `color3` καθορίζουν τη χρωματική κλίμακα του θερμογραφήματος. Η τιμή `adj_mat` περιέχει μία ειδική μορφή του πίνακα γειτνίασης για να καταχωρείται αυτόματα πριν την προσομοίωση. Η μορφή του πίνακα στο πεδίο αυτό θα εξηγηθεί σε επόμενη ενότητα. Τέλος, έχουμε την τιμή `current_state` που ξεκινάει απ' το μηδέν και υποδηλώνει ποιο νούμερο κατάστασης θα πάρει η επόμενη προσομοίωση.

```
CREATE TABLE `settings` (
  `id` int(11) NOT NULL,
  `max_hops` int(11) NOT NULL,
  `max_radius` int(11) NOT NULL,
  `max_opacity` float NOT NULL,
  `min_opacity` float NOT NULL,
  `blur` float NOT NULL,
  `color1` varchar(128) NOT NULL,
```

```
  `color2` varchar(128) NOT NULL,
  `color3` varchar(128) NOT NULL,
  `adj_mat` text NOT NULL,
  `current_state` int(11) NOT NULL
)
```

2.1.2 Η οθόνη δημιουργίας χάρτη

Για τη σελίδα `create_map.html` θα χρειαστούμε τη βιβλιοθήκη `Fabric.js`. Στην παρούσα εφαρμογή απλά δημιουργούμε μία αναφορά μέσω του `Content Delivery Network` στο `<head>` της σελίδας ως εξής :

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/fabric.js/4.1.0/fa
bric.min.js"></script>
```

Το κομμάτι της `html` είναι απλό και προϋποθέτει την ύπαρξη ενός στοιχείου `<canvas>`. Εκεί πάνω γίνονται όλες οι εργασίες που αφορούν το γραφικό περιβάλλον.

```
<canvas id="c" width="1000" height="662" style="border:1px
solid #ccc">
```

Σημείωση: Σε όλη τη διάρκεια της υλοποίησης υϊοθετείται η αναλογία 1000x662 για τις διαστάσεις. Για να ξεκινήσουμε να χρησιμοποιούμε τη βιβλιοθήκη `Fabric.js` πρέπει να δημιουργήσουμε μια αναφορά σε ένα αντικείμενό της.

```
var canvas = new fabric.Canvas('c', { selection: false });
```

Επάνω σε αυτό το αντικείμενο θέλουμε να καταλάβουμε πότε ο χρήστης κάνει κλικ ώστε να εισάγουμε τις συντεταγμένες αρχής και τέλους και παράλληλα να εμφανίσουμε μία γραμμή στο σωστό σημείο. Για να πραγματοποιηθεί αυτό καλούμε τη μέθοδο `on()` πάνω στο αντικείμενο που μόλις δημιουργήσαμε .

```
canvas.on(event, function(o){});
```

Το πρώτο όρισμα πρέπει να είναι ένα γεγονός όπως το πάτημα κουμπιού του ποντικιού και στη συνέχεια ένα callback. Στην callback συνάρτηση θα εισάγουμε τον

κώδικα που κρατάει τις τιμές για τις συντεταγμένες. Με τα δεδομένα που λαμβάνουμε κατασκευάζουμε ένα αντικείμενο `Line` της `Fabric` ως εξής:

```
canvas.on('mouse:down', function(o){
  isDown = true;
  var pointer = canvas.getPointer(o.e);
  var points = [ pointer.x, pointer.y, pointer.x, pointer.y ];
  line = new fabric.Line(points, {
    strokeWidth: 5,
    fill: 'red',
    stroke: 'red',
    originX: 'center',
    originY: 'center'
  });
  var num_1 = Math.round(pointer.x)
  var num_2 = Math.round(pointer.y)
  x1.push(num_1);
  y1.push(num_2);
  canvas.add(line);
});
```

Για την αποθήκευση των συντεταγμένων των ευθειών χρησιμοποιούμε τους πίνακες `x1`, `y1`, `x2`, `y2` και η διαδικασία της ενδιάμεσης αποθήκευσης είναι κλήση της `push()`.

```
var num_1 = Math.round(pointer.x);
var num_2 = Math.round(pointer.y);
x1.push(num_1);
y1.push(num_2);
```

Σε κάθε κίνηση του ποντικιού, όσο ο χρήστης κρατά πατημένο το αριστερό κλικ πρέπει να λαμβάνουμε και να αποθηκεύουμε τη νέα θέση, γιατί ενδεχομένως να είναι και η τελευταία. Πάλι με το κατάλληλο event και τη συνάρτηση `on` ως εξής :

```
canvas.on('mouse:move', function(o){
  if (!isDown) return;
  var pointer = canvas.getPointer(o.e);
  var num_1 = Math.round(pointer.x)
  var num_2 = Math.round(pointer.y)
  line.set({ x2: num_1, y2: num_2 });
  canvas.renderAll();
});
```

Κάθε κίνηση του κέρσορα όσο είναι πατημένο το κλικ θα κάνει την τιμή του `isDown` αληθή και έτσι η `getPointer` θα επιστρέψει στον pointer τις νέες τιμές και αυτές με τη σειρά τους θα γίνουν `set` ως τελικές συντεταγμένες της γραμμής που είναι αποθηκευμένη στη μεταβλητή `line`. Η μέθοδος `renderAll` προκαλεί την επαναδιατύπωση του τελευταίου διαθέσιμου καρέ. Μόλις αφήσει ο χρήστης το κλικ θα πρέπει να ανιχνεύεται ένα τρίτο γεγονός που να χειρίζεται το συμβάν. Αυτό σημαίνει πως η γραμμή έληξε και είναι ώρα να εισαχθούν τα δεδομένα στους πίνακες `x2`, `y2`.

```
canvas.on('mouse:up', function(o){
  isDown = false;
  var pointer = canvas.getPointer(o.e);
   x2.push(line.x2);
  y2.push(line.y2);
});
```

Κάτω απ' το ταμπλό σχεδιασμού των γραμμών εμποδίων υπάρχουν δύο απλά κουμπιά. Το δεξία με την ένδειξη UNDO αναιρεί την εισαγωγή του τελευταίου εμποδίου. Η διαδικασία έχει ως εξής:

```
function RemoveLines() {
```

```
  var objects = canvas.getObjects('line');
  canvas.remove(objects[objects.length-1]);
  canvas.renderAll();
  x1.pop();
  y1.pop();
  x2.pop();
  y2.pop();
}
```

Πάνω στη μεταβλητή με όνομα `canvas` η `getObjects` επιστρέφει τον πίνακα αντικειμένων που υπάρχουν πάνω του και είναι τύπου `line`. Η τελευταία γραμμή που εισήχθη θα βρίσκεται στη θέση `objects.length-1`. Το αριστερό κουμπί προκαλεί την κλήση ενός `php` σεναρίου για την αποθήκευση των συντεταγμένων. Ο κώδικας του `save_obs.php` είναι απλός :

```
$x1 = $_POST['x1'];
$y1 = $_POST['y1'];
$x2 = $_POST['x2'];
$y2 = $_POST['y2'];

for($i = 0; $i < count($x1); $i++) {
$query = "insert into obstacles(x1,y1,x2,y2)
values('$x1[$i]','$y1[$i]','$x2[$i]','$y2[$i]')";
      if($conn->query($query) === TRUE){
            echo "<h3>ok.</h3>";
      }else {
    echo "Error: " . $query . "<br>" . $conn->error;
}
}
```

Για κάθε εγγραφή στους πίνακες `x1`, `y1`, `x2`, `y2` πραγματοποιείται ένα ερώτημα εισαγωγής στη βάση δεδομένων.

2.1.3 Η οθόνη εισαγωγής των σημείων πρόσβασης

Αρχικά πρέπει να αντλήσουμε τις πληροφορίες για τα εμπόδια, ώστε να αποτυπωθούν. Αυτό γίνεται στο αρχείο `heatmap.php` ως εξής:

```
$x1 = [];
$y1 = [];
$x2 = [];
$y2 = [];
while($row = mysqli_fetch_array($obs_info)){
  $x1_t = $row['x1'];
  $y1_t = $row['y1'];
  $x2_t = $row['x2'];
  $y2_t = $row['y2'];
array_push($x1,$x1_t);
array_push($y1,$y1_t);
array_push($x2,$x2_t);
array_push($y2,$y2_t);

}
```

Στη συνέχεια έχουμε ξανά το ίδιο `<canvas>` στοιχείο με το ίδιο `id = c`

```
var x1 = <?php echo json_encode($x1); ?>;
var y1 = <?php echo json_encode($y1); ?>;
var x2 = <?php echo json_encode($x2); ?>;
var y2 = <?php echo json_encode($y2); ?>;

var canvas = document.getElementById('c');
```

```
for(i=0;i<=y1.length;i++){


 if (canvas.getContext){
  var context = canvas.getContext('2d');
  context.beginPath();
  context.lineWidth = 5;
  context.moveTo(x1[i],y1[i]);
  context.lineTo(x2[i],y2[i]);
  context.stroke();
   }
}
```

Αρχικά μετατρέπουμε τους πίνακες `x1,y1,x2,y2` από php σε μορφή JSON. Στη συνέχεια δημιουργούμε την αναφορά `canvas` στο αντικείμενο DOM με κωδικό `c`. Η μεταβλητή `context` είναι το σημείο αναφοράς μας στο αντικείμενο `canvas`. Η `beginPath` σηματοδοτεί την έναρξη της γραμμής. Η `moveTo` μετακινεί τον δείκτη στο σημείο έναρξης της ευθείας και η `lineTo` στο σημείο λήξης. Η μέθοδος `stroke` αποτυπώνει την ευθεία.

Κάθε φορά που ο χρήστης κάνει κλικ θα πρέπει να εμφανίζεται σε εκείνη τη θέση ένα σημείο αναφοράς με τον αντίστοιχο αύξοντα αριθμό και οι συντεταγμένες του να αποθηκεύονται στη βάση δεδομένων. Επιπροσθέτως, στα δεξιά της οθόνης θα εμφανίζεται κάθε φορά ένα πεδίο όπου θα συμπληρώνονται οι γειτονικοί κόμβοι. Ως σημείο αναφοράς έχουμε το `wrapper div` με `id = heatmapContainer`

```
document.getElementById("heatmapContainer").addEventListener("
click",function
(e) {
    i++;
    var div = $('<div class="puck">')
      .css({
        "left": e.x + 'px',
        "top": e.y + 'px'
      })
      .append($('<div style="margin-top:-7px;
```

```
margin-left:-3px">&#128959;
      </div>'),i)
      .appendTo(document.body);
var text = $("<div><input type='text' class='child'
onkeyup='addComma(this)'/><br></div>");
$("#textboxDiv").append("Access Point ",i, "
neighbours");$("#textboxDiv").append(text);text.attr('id',
i);});
```

Στη μεταβλητή `div` δίνουμε έναν jQuery selector με θέση `e.x` και `e.y` που είναι οι τιμές του click event και δίνει τις συντεταγμένες του σημείου που έγινε κλικ. Στη μεταβλητή προστίθεται και ένα ακόμα `div` με τον κωδικό για το αστεράκι που θα συμβολίζει τώρα το σημείο πρόσβασης `&#128959`. Η `appendTo` δημιουργεί λοιπόν τον κλάδο `div` και τον προσαρτά στο DOM. Οι υπόλοιπες γραμμές δημιουργούν ένα div κλάσης `child` με τη μέθοδο `addComma` στο γεγονός πληκτρολογίου `onkeyup`. Κάθε φορά που ο χρήστης πατά κάποιον χαρακτήρα ως μέρος των γειτονικών κόμβων, η μέθοδος `addComma` προσθέτει ένα κόμμα ύστερα απ'αυτόν.

```
function addComma(txt){

  if(txt.value.length!=0)
  txt.value=txt.value+',';
}
```

Η αποθήκευση των x,y γίνεται με τη μέθοδο `printMousePos` η οποία είναι το `callback` του event listener στο κλικ πάνω `στο heatmapContainer`.

```
function printMousePos(event) {
   x.push(event.clientX);
   y.push(event.clientY);
}

document.getElementById("heatmapContainer").addEventListener("
click", printMousePos);
```

Τα δυο κουμπιά που υπάρχουν κάτω κάτω στην οθόνη αποθηκεύουν ή καθαρίζουν την κατάσταση των σημείων πρόσβασης. Το κουμπί 'RESET' καλεί τη μέθοδο `reset` που αδειάζει τους πίνακες `x, y` που χρησιμεύουν ως ενδιάμεσος αποθηκευτικός τρόπος των συντεταγμένων και διαγράφει τα `child` nodes που αντιπροσωπεύουν τα `textbox` στα δεξιά.

```
function reset(){
$('.puck').remove();
 x= [];
 y= [];
  $("#textboxDiv").children().remove();
  $("#textboxDiv").empty();
  i=0;
}
```

Το κουμπί 'Save all' είναι συνυφασμένο με μια συνάρτηση που εκτελεί δύο βασικές εργασίες. Η πρώτη είναι να δημιουργεί τον πίνακα γειτνίασης στη μορφή:

AP1#N1,N2…Nm;

AP2#N1,N2…Nm;

Αυτό γίνεται με τον εξής τρόπο :

```
var adj_mat="";
i=0;
$(".child").each(function() {
adj_mat+=(i+1)+'#'+$(this).val().replace(/.$/,";");
 i++;
});
```

Για κάθε selector της κλάσης `child` πρόσθεσε τον αύξοντα αριθμό αρχής γενομένης από το 1. Πρόσθεσε τον χαρακτήρα '#' και στην τιμή του αλφαριθμητικού

του κάθε `child` αντικατέστησε τα κόμματα σε ‘;’. Για να αποθηκεύσουμε τα δεδομένα γίνεται η κλήση το `save_ap.php` με τα κατάλληλα δεδομένα.

```
$.ajax({
  type: "POST",
  url: "save_ap.php",
  data: { x1: y,
        y1: x,
        adj_mat: adj_mat}
});
```

Ο κώδικας των ερωτημάτων :

```
$x = $_POST['x1'];
$y = $_POST['y1'];
$adj_mat = $_POST['adj_mat'];

for($i = 0; $i < count($x); $i++) {
$query = "insert into access_point(x,y,times_visited,heat)
values('$x[$i]','$y[$i]',0,0)";
      if($conn->query($query) === TRUE){
            echo "<h3>ok.</h3>";
      }else {
    echo "Error: " . $query . "<br>" . $conn->error;
}
}
$query2 = "update settings set adj_mat =
'$adj_mat',current_state =0";
$res = mysqli_query($conn,$query2);
```

Οι τιμές `times_visited` και `heat` στο πρώτο ερώτημα αρχικοποιούνται σε μηδέν καθώς κανείς ακόμα δεν τα έχει επισκεφθεί. Επίσης εδώ εισάγεται και ο πίνακας `settings` με το πεδίο `adj_mat` το οποίο κρατά τις τιμές του πίνακα γειτνίασης χωρίς κενά και το πεδίο `current_state` που αρχικοποιείται πάντοτε με μηδέν καθώς βρισκόμαστε στην κατάσταση κανενός τρεξίματος.

2.1.4 Η οθόνη της προσομοίωσης

Σε αυτή την οθόνη στα αριστερά ο χρήστης μπορεί να δει τον χώρο που έχει προηγουμένως ορίσει καθώς και τα σημεία πρόσβασης. Αριστερά υπάρχουν διάφορες επιλογές που αφορούν τα δεδομένα εισαγωγής της προσομοίωσης. Αυτά είναι τα εξής: `max visitors` δηλαδή ο αριθμός των επισκεπτών που ο χρήστης επιθυμεί για την τρέχουσα προσομοίωση. `# of allowed hops per visitor` : είναι ο αριθμός των αλμάτων από σημείο πρόσβασης σε σημείο πρόσβασης που θα εκτελέσει ο κάθε επισκέπτης προκειμένου να τελειώσει η επίσκεψή του. `Min/Max wait time/hop (sec)` είναι ο μέγιστος και ο ελάχιστος χρόνος σε δευτερόλεπτα που ένας επισκέπτης δύναται να παραμείνει σε ένα σημείο πρόσβασης. `return to already visited AP probability` – η πιθανότητα τοις εκατό να επιστρέψει ο επισκέπτης σε σημείο πρόσβασης που έχει ήδη επισκεφθεί. `adjacency matrix` – είναι ο πίνακας γειτνίασης που θα χρησημοποιήσει ο αλγόρθιμος προσομοίωσης για να γνωρίζει τις συζεύξεις των σημείων πρόσβασης. Εισάγεται στο πεδίο αυτόματα μετά το φόρτωμα της σελίδας αλλά μπορεί ο χρήστης να επέμβει με αλλαγές.

2.1.5 Εξαγωγή των δεδομένων προσομοίωσης

Πριν φτάσουμε να βλέπουμε στην οθόνη, σε πραγματικό χρόνο, τους επισκέπτες να κινούνται στον χώρο, προηγούνται ορισμένες διαδικασίες προετοιμασίας. Εν ολοίγοις, αυτές είναι: Αρχικοποίηση πίνακα γειτνίασης, παραγωγή τυχαίων δεδομένων για κάθε επισκέπτη. Η μέθοδος `start` καλείται με το κλικ στο κουμπί ‘Start Simulation’. Αρχικά λαμβάνεται η τιμή του αλφαριθμητικού του πίνακα γειτνίασης. Στόχος είναι να το φέρουμε στην κανονική του μορφή με 0 και 1 όπως παρακάτω:

1#2,3,4,7;

2#3,4,7;

3#2,4,7;

4#2,3,5;

5#3,4,6;

6#5,7;

7#2,6;

(index)	1	2	3	4	5	6	7
1	0	1	1	1	0	0	1
2	0	0	1	1	0	0	1
3	0	1	0	1	0	0	1
4	0	1	1	0	1	0	0
5	0	0	1	1	0	1	0
6	0	0	0	0	1	0	1
7	0	1	0	0	0	1	0

Figure 13 Κανονική μορφή πίνακα γειτνίασης

Οι πληροφορίες εξάγονται αν συνδυάσουμε ένα ζεύγος γραμμής, στήλης. Για παράδειγμα αν θέλουμε να ελέγξουμε αν από το σημείο πρόσβασης 2 μπορούμε να μεταβούμε στο 7, στο σημείο τομής (2,7) βλέπουμε έναν άσσο που σημαίνει αληθής τιμή.

Ας εξετάσουμε βήμα βήμα τη διαδικασία:

```
var resul = document.getElementById("adj").value;
visitors_list=[];
var test_string=resul;
test_string=test_string.replace/(\r\n|\n|\r)/gm,'');
var splited=test_string.split('');
var semicolons=[];
var result={};
for(i=0; i<splited.length-1; i++){

     if(!semicolons.includes(test_string.indexOf(';',i)))
semicolons.push(test_string.indexOf(';',i));
}
```

Αρχικά λαμβάνουμε την τιμή του πεδίου `adj` και την αποθηκεύουμε στη μεταβλητή `result`. Στη συνέχεια μετατρέπουμε το αλφαριθμητικό σε πίνακα χαρακτήρων στη μεταβλητή `splited`. Για το αλφαριθμητικό '1#2,3,4,7;2#3,4,7;3#2,4,7;4#2,3,5;5#3,4,6;6#5,7;7#2,6;' η τιμή του `splited` θα είναι έναν πίνακας

54 θέσεων. Ο βρόχος `for` διατρέχει τον πίνακα `splited` και κρατά στον πίνακα `semicolons` τις θέσεις που βρίσκονται τα ερωτηματικά.

```
var cat_str=[];
var start_pos=0;
for(i=0; i<semicolons.length; i++){
cat_str.push(test_string.substring(start_pos,semicolons
[i]));start_pos=semicolons[i]+1;}
var ap_num=[];
var ap_nom=[];
for(i=0; i<cat_str.length; i++){
  var num=cat_str[i].substring(0,cat_str[i].indexOf('#'))
  ap_num.push(num);
}
for(i=0; i<cat_str.length; i++){
var
nom=cat_str[i].substring(cat_str[i].indexOf('#')+1,cat_str[i].
lastIndexOf(''));ap_nom.push(nom);
}
```

Ο πίνακας `cat_str` περιέχει σε κάθε κελί του την πληροφορία για κάθε σημείο πρόσβασης. Στο παράδειγμά μας είναι :

```
0: "1#2,3,4,7"
1: "2#3,4,7"
2: "3#2,4,7"
3: "4#2,3,5"
4: "5#3,4,6"
5: "6#5,7"
6: "7#2,6"
```

Η μεταβλητή `start_pos` είναι ο κινούμενος δείκτης που δείχνει κάθε φορά στη θέση του επόμενου semicolon ώστε να γνωρίζει η μέθοδος `substring` μέχρι ποια θέση του πίνακα `test_string` να επιστρέψει. Στον πίνακα `ap_num` κρατάμε

έναν αριθμό σε κάθε κελί που αντιπροσωπεύει το `id` του σημείου πρόσβασης. Στον πίνακα `ap_nom` κρατάμε σε κάθε κελί τους γείτονες του κάθε σημείου πρόσβασης.

```
for(i=0; i<ap_num.length; i++){
      result[ap_num[i]]=ap_nom[i].split(',');
}
max_aps=ap_num.length;
```

Οι τιμές των `ap_num` και `ap_nom` μας βοηθούν στην εξαγωγή του `object result` της μορφής

```
1: (4) ["2", "3", "4", "7"]
2: (3) ["3", "4", "7"]
3: (3) ["2", "4", "7"]
4: (3) ["2", "3", "5"]
5: (3) ["3", "4", "6"]
6: (2) ["5", "7"]
7: (2) ["2", "6"]
```

Κάθε property είναι το `id` του σημείου πρόσβασης ενώ η τιμή του είναι ο πίνακας των γειτονικών του σημείων. Αυτός ήταν και στόχος εξ'αρχής, να φτάσει ο πίνακας σ'αυτή τη μορφή.

```
let order =
Object.keys(result).reduce((c,v)=>Object.assign(c,{[v]:0}),{})
let pinakas = Object.keys(result).reduce((c, v) => {
  return Object.assign(c, {[v]: result[v].reduce((p, o)=>{
      p[o] += 1;
      return p;
    }, Object.assign({}, order))});
}, {});
```

Η πρώτη γραμμή δημιουργεί ένα `Object` με τα κλειδιά των σημείων πρόσβασης και μηδενικά ως τιμές. Στη συνέχεια ο πίνακας γεμίζει με μηδενικά και άσσους.

2.1.6 Δημιουργία επισκεπτών

```
for(i = 0; i<max_vis; i++){
  visitors_list.push(create_visitor(i+1))
}
```

Η τιμή `max_vis` προέρχεται απ' το input με το ίδιο όνομα. Η μέθοδος `create_visitor` δέχεται το `id` και δημιουργεί έναν επισκέπτη.

```
function create_visitor(id) {
let visitor = {
id:id,
time_entered:0,
curr_ap:1,
seen:['1']}; return visitor;}
```

Στη λίστα των επισκέψεων που έχει πραγματοποιήσει ο συγκεκριμένος επισκέπτης με όνομα `seen` αρχικοποιείται πάντα η είσοδος που έχει κωδικό 1 . Το ίδιο ισχύει και με την τρέχουσα θέση του , `curr_ap`.

2.1.7 Εκκίνηση διαδικασίας παραγωγής δεδομένων

```
while(visitors_list.length!=0){
var visitor_selector=getRndInteger(0,visitors_list.length-1);
var visitor=visitors_list[visitor_selector];
```

Η διαδικασία μας θα τρέχει όσο υπάρχουν επισκέπτες σύμφωνα με τη συνθήκη στο βρόχο `while`. Επιλέγεται τυχαία ένας επισκέπτης.

```
do{
var select_ap=getRndInteger(2,max_aps);
var curr_vis_pos=visitor.curr_ap;
```

```
var prob_to_visit_again=getRndInteger(1,100);
if(visitor.seen.includes(select_ap) && prob_to_visit_again>=1
&& prob_to_visit_again<=parseInt(ret_prob) &&
pinakas[curr_vis_pos][select_ap]==1){
 console.log("I will visit again "+select_ap);
 break;
}
}while (pinakas[curr_vis_pos][select_ap]!=1 ||
visitor.seen.includes(select_ap));
```

Επιλέγεται τυχαία ένας αριθμός στο `select_ap` που αναπαριστά το επόμενο σημείο πρόσβασης και μία πιθανότητα. Ελέγχεται αν ισχύει η πιθανότητα με βάση εκείνη που έχει δώσει ο χρήστης, αν το σημείο πρόσβασης υπάρχει στη λίστα `seen` του επισκέπτη και κυρίως αν ο αριθμός `select_ap` που επιλέχθηκε ως το επόμενο σημείο πρόσβασης είναι γείτονας με το τρέχον.

```
visitor.curr_ap=select_ap;
var wait=getRndInteger(parseInt(min),parseInt(max));
var exit_time=visitor.time_entered+wait;
visitor.seen.push(select_ap);
visitor.time_entered+=wait;
$.ajax({
  type: "POST",
  url: "update_ap_info.php",
  data: { ap: select_ap,
        wait: wait}
});
```

Αφού έγιναν οι έλεγχοι και το σημείο που επιλέχθηκε είναι σωστό, τότε μπορεί να θεωρείται η επόμενη θέση του επισκέπτη. Επιλέγεται ένας τυχαίος αριθμός αναμονής απ' το κλειστό σύνολο των `min max` που έχει ορίσει ο χρήστης. Το συγκεκριμένο σημείο συμπεριλαμβάνεται και στη λίστα `seen`. Ο χρόνος αναμονής αποθηκεύεται στο `time_entered`. Τελικά, το `select_ap` και ο χρόνος αναμονής στέλνονται στη βάση μέσω του `update_ap_info.php`.

```
$x = $_POST['ap'];
$y = $_POST['wait'];
$query = "update access_point set times_visited =
times_visited + 1,heat = heat+ '$y' where id=$x";
      if($conn->query($query) === TRUE){
            echo "<h3>ok.</h3>";
      }else {
    echo "Error: " . $query . "<br>" . $conn->error;
}
```

Το ερώτημα είναι ένα απλό ερώτημα `update` και ανεβάζει τον μετρητή επισκέψεων `times_visited` κατά ένα. Η τιμή `heat` προσαυξάνεται κατά όσο χρόνο παρέμεινε ο επισκέπτης στο σημείο.

```
var flag = 1;
…
$.ajax({
  type: "POST",
  url: "update_ap_info2.php",
  data: { ap: select_ap,
        wait: wait,
        date: date,
        flag: flag}, success: function(result) {
         },
         error: function(jqxhr, status, exception) {
             alert('Exception:', exception);
         }
});
flag = 2;
```

Τα δεδομένα αποθηκεύονται και στον πίνακα `past_instances` μαζί με μία ημερομηνία. Η τιμή του `flag` αρχικοποιείται με 1 στην αρχή ώστε να γνωρίζουμε ποια είναι η πρώτη φορά που καλείται το αρχείο `update_ap_info2.php`. Έτσι την πρώτη φορά εισάγεται μια εγγραφή για την είσοδο με τις default τιμές.

```
if($flag==1){

$query = "insert into
past_instances(id,times_visited,heat,date,state_id)
values(1,0,0,'$date','$current_state')";

     $res = mysqli_query($conn,$query);

}


$query = "select * from past_instances where id=$x AND
date='$date'";

$res = mysqli_query($conn,$query);

$row = mysqli_fetch_array($res);

$rowcount=mysqli_num_rows($res);

echo $rowcount;




if($rowcount> 0){

$id = $row['id'];

$date1 = $row['date'];

$times_visited = $row['times_visited'];

$heat = $row['heat'];

$query = "update past_instances set times_visited =
times_visited + 1,heat = heat+ '$y' where id=$id AND
date='$date1' AND state_id='$current_state'";

$res = mysqli_query($conn,$query);


}else{
```

```
$query = "insert into
past_instances(id,times_visited,heat,date,state_id)
values('$x',1,'$y','$date','$current_state')";

      $res = mysqli_query($conn,$query);

}
```

Η γενική ιδέα είναι ότι για κάθε instance, δηλαδή για κάθε παλαιότερη προσομοίωση, θα πρέπει να υπάρχει ακριβώς μια εγγραφή για κάθε σημείο πρόσβασης. Για παράδειγμα :

id	times_visited	heat	date	state_id
4	21	379	07/12/2020 20:25 PM	0
6	19	414	07/12/2020 20:25 PM	0
5	17	286	07/12/2020 20:25 PM	0
3	18	374	07/12/2020 20:25 PM	0
1	0	0	07/12/2020 20:25 PM	0
7	24	498	07/12/2020 20:25 PM	0
2	21	423	07/12/2020 20:25 PM	0
1	0	0	07/12/2020 20:27 PM	1
2	10	150	07/12/2020 20:27 PM	1
7	9	166	07/12/2020 20:27 PM	1
4	9	157	07/12/2020 20:27 PM	1
3	10	148	07/12/2020 20:27 PM	1
5	6	121	07/12/2020 20:27 PM	1
6	6	73	07/12/2020 20:27 PM	1

Figure 14 Δομή πίνακα past_instances

Παρατηρούμε πως για κάθε `state_id` και `date` υπάρχει από μια εγγραφή για κάθε `id`. Στο παραπάνω κομμάτι κώδικα, αν το ερώτημα `$query = "select * from past_instances where id=$x AND date='$date'";` επιστρέψει τιμές τότε σημαίνει πως το σημείο ενδιαφέροντος υπάρχει για τη συγκεκριμένη ημερομηνία άρα θα εκτελέσουμε `update`. Σε αντίθετη περίπτωση εκτελούμε `insert`.

```
if(visitor.seen.length==max_hops){
$.ajax({
  type: "POST",
  url: "save_route.php",
```

```
  data: { route: visitor.seen
        }
});
  visitors_list.splice(visitor_selector, 1);
}
```

Το τέλος της επίσκεψης ενός επισκέπτη φτάνει όταν έχει «δει» τόσα εκθέματα όσα τα `max_hops` που ορίστηκαν απ' τον χρήστη. Τότε τον διαγράφουμε και αποθηκεύουμε τη διαδρομή του στέλνοντας τη λίστα seen στο `save_route.php`

```
$route_temp = implode(",", $route);
$route_str = str_replace(",", "",$route_temp);
$query = "insert into routes(route) values('$route_str')";
```

Πρίν αποθηκεύσουμε τη λίστα διαδρομής φροντίζουμε να σβήσουμε τα κόμματα. Αποθηκεύεται τελικά ένα αλφαριθμητικό της μορφής '13457'.

```
$.ajax({
  type: "POST",
  url: "update_ap_info2.php",
  data: { change: "yes"}
});



if($_POST['change']=='yes'){
     $query = "update settings set current_state =
current_state+1";
     $res = mysqli_query($conn,$query);
}
```

Με την έξοδο απ' την επανάληψη έχουμε ουσιαστικά δημιουργήσει τα δεδομένα που χρειάζεται η προσομοίωση. Θα πρέπει η βάση δεδομένων να γνωρίζει λοιπόν ότι αυτή η κατάσταση (`state`) έχει τελειώσει, και η επόμενη θα πρέπει να λάβει διαφορετικό κωδικό.

Πριν προχωρήσουμε στην ανάλυση του αλγορίθμου κίνησης με αποφυγή εμποδίων ας κάνουμε μια αναφορά στον πίνακα `stage1`. Αναφεθήκαμε σε προηγούμενη ενότητα για μια λίστα κίνησης της μορφής **[{x:100,y:150},{w:15000},{x:200,y:200},{x:245,y:300},{w:9000}]** που αποθηκεύει τη συνολική κίνηση ενός επισκέπτη κατά τη συγκεκριμένη προσομοίωση. Για κάθε επισκέπτη υπάρχει και μια τέτοια λίστα. Η πρώτη φάση δημιουργίας της εν λόγω λίστας είναι η δημιουργία του πίνακα `stage1` και η αρχικοποίησή του. Αυτή γίνεται σε κομμάτι κώδικα που ήδη έχουμε αναφέρει αλλά σκόπιμα παραλλείπεται για λόγους ευκρίνειας.

```
for(i = 0; i<max_vis; i++){
  visitors_list.push(create_visitor(i+1));
   stage1.push([i+1]);

}
for(i = 0; i<max_vis; i++){
  stage1[i].push({x:y[0] ,y:x[0]});
  stage1[i].push({w:0});
  }
```

Για κάθε επισκέπτη που δημιουργήσαμε εισάγουμε και ένα κλειδί στον πίνακα, ώστε να έχουμε μια αναφορά σε κάθε επισκέπτη. Εν συνεχεία για κάθε επισκέπτη αρχικοποιούνται οι συντεταγμένες θέσης του σε εκείνες της εισόδου και μια default τιμή αναμονής `w = 0`. Όταν έχουμε τις τιμές για το τρέχον σημείο πρόσβασης του επισκέπτη καθώς και την τιμή αναμονής τότε τις εισάγουμε στη λίστα κίνησης.

```
stage1[visitor.id-1].push({x:y[parseInt(select_ap-1)]
,y:x[parseInt(select_ap-1)]},{w:wait*1000});
```

Η τιμή `wait` του χρόνου αναμονής πολλαπλασιάζεται επί χίλια για να μετατραπεί με ms. Αυτό χρησιμεύει παρακάτω όταν θα πρέπει να εκτελεστεί το `animation`.

2.1.8 Αλγόριθμος κίνησης

Συνοψίζοντας τις εντολές πολύ συνοπτικά έχουμε:

Για κάθε ζευγάρι access point καθενός επισκέπτη στον πίνακα `stage1`:

Επίλεξε συντεταγμένες αρχής

Επίλεξε συντεταγμένες τέλους

Υπολόγισε τη διαδρομή

Επίστρεψε τη διαδρομή σε δύο πίνακες για `x` και `y`

```
for(let b = 0; b<stage1.length; b++){
  for(let v = 3; v<stage1[b].length-1; v=v+2){
     var curr_ap1_x = parseFloat(stage1[b][v-2].y);
     var curr_ap1_y = parseFloat(stage1[b][v-2].x);
     var curr_ap2_x = parseFloat(stage1[b][v].y);
     var curr_ap2_y = parseFloat(stage1[b][v].x);
```

Δουλεύουμε ανά ζευγάρια `x, y` και επιλέγουμε τις τιμές από τον πίνακα `stage1`. Ο λόγος που γίνεται η παραπάνω διάσχιση με το `v` να ξεκινάει από 3 με βήμα 2 και να λαμβάνονται οι τιμές με δείκτες τα `v-2` και `v` είναι δίοτι ο πίνακας είναι της μοφής:

```
Array(4)
  1. 0: Array(13)
       1. 0: 1
       2. 1: {x: "978", y: "362"}
       3. 2: {w: 0}
       4. 3: {x: "418", y: "527"}
       5. 4: {w: 5000}
       6. 5: {x: "405", y: "362"}
       7. 6: {w: 8000}
       8. 7: {x: "413", y: "222"}
       9. 8: {w: 7000}
       10.    9: {x: "559", y: "285"}
       11.    10: {w: 7000}
       12.    11: {x: "502", y: "219"}
       13.    12: {w: 8000}
```

Στην πρώτη θέση υπάρχει ο δείκτης του επισκέπτη όπως αναφέρθηκε και παραπάνω. Εφόσον η δομή είναι πάντα η ίδια , γνωρίζουμε πως για να επεξεργαστούμε για παράδειγμα τις τιμές των συντεταγμένων εισόδου με εκείνες του επόμενου σημείου στη λίστα θα πρέπει να εξάγουμε τα αντικείμενα στις θέσεις 1 και 3 κ.ο.κ.

```
var ap1_x = curr_ap1_y; var ap1_y = curr_ap1_x;

var ap2_x= curr_ap2_x; var ap2_y= curr_ap2_y;

var route_x = []; var route_y = [];

var first_ap= true;

var curr_pos_x; var curr_pos_y;

curr_pos_x = ap1_x; curr_pos_y = ap1_y;

route_x.push(curr_pos_x); route_y.push(curr_pos_y);
```

Επεξηγήσεις τιμών: `ap1_x, ap1_y, ap2_x, ap2_y` – οι συντεταγμένες των σημείων εκκίνησης και τερματισμού. `route_x, route_y` – οι συντεταγμένες των σημείων της διαδρομής ανάμεσα σε `ap1, ap2` (αρχικοποιούνται πάντα με τις τιμές του πρώτου σημείου πρόσβασης). `first_ap` - αληθοτιμή αναγνώρισης πρώτης κίνησης. `curr_pos_x, curr_pos_y` – η τρέχουσα θέση (αρχικοποιείται πάντα με τις τιμές του πρώτου σημείου πρόσβασης). Ολόκληρη η διαδικασία επαναλμβάνεται σε βρόχο `do – while`:

```
do{

…

}while(route_x[route_x.length-1]!=ap2_y &&
route_y[route_y.length-1]!=ap2_x);
```

Η συνθήκη ελέγχει αν η τρέχουσα θέση μας αντιστοιχεί σε εκείνη του σημείου τερματισμού. Αρχικά βρίσκουμε την απόσταση των σημείων εκκίνησης και τερματισμού.

```
d = D(curr_pos_x,curr_pos_y,ap2_y,ap2_x);

d=3*d;

var NumOfblocks = 0;
```

Η απόσταση d είναι η ακτίνα των σημείων του επιπέδου που θα ελεγχθούν. Προφανώς δεν οφελεί να εξετάσουμε όλα τα σημεία του επιπέδου για λόγους πολυπλοκότητας. Αυτή η προσέγγιση δεν είναι ιδιαίτερα ασφαλής σε ορισμένες περιπτώσεις, διότι μπορεί για παράδειγμα τα σημεία εκκίνησης τερματισμού να βρίσκονται εκατέρωθεν ενός εμποδίου. Αν βρίσκονται κοντά σε αυτό τότε η απόστασή τους (αυτή που επιστρέφει το D) είναι πολύ μικρή δεδομένου πως μετράμε την απόσταση σε ευθεία γραμμή.

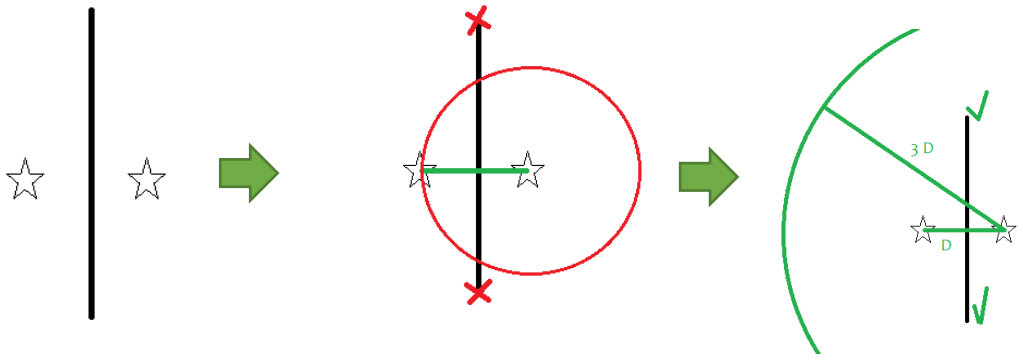


Figure 15 Αιτιολόγηση απόστασης 3D

Για το λόγο αυτό ερευνούμε σε τριπλάσια απόσταση απ' όσο η κανονική απόσταση μεταξύ των δύο σημείων.

```
function D (ap1_x,ap1_y,ap2_x,ap2_y){
var a = ap1_x - ap2_x;
var b = ap1_y - ap2_y;
var d = Math.sqrt( a*a + b*b );
return d;
}
```

Η μέθοδος εύρεσης της απόστασης d έρχεται απ' τα μαθηματικά και ορίζεται ως $\sqrt{(x1-x2)^2+(y1-y2)^2}$. Εξετάζουμε την τετριμμένη περίπτωση. Αν δεν υπάρχει κανένα εμπόδιο τότε μπορούμε με σιγουριά να υποθέσουμε πως τα δύο σημεία μας ενώνονται εξ' επαφής. Μια σημαντική σημείωση είναι πως παρόλο που στη λογική θα λέγαμε κανένα εμπόδιο, στην πράξη βρίσκουμε ένα εμπόδιο. Αν αυτό το εμπόδιο συμπίπτει με την άκρη μίας ευθείας τότε στην πραγματικότητα είναι το σημείο που θα ήταν υποψήφιο για την επόμενη θέση. Η πρώτη επανάληψη έχει την ιδαιτερότητα ότι

(εκτός απ' την τετριμμένη περίπτωση) ο αλγόρθιμος θα επιστρέφει εμπόδιο το οποίο εμείς καλούμαστε να ελέγξουμε αν βρίσκεται στην άκρη της ευθείας ώστε να το δεχτούμε. Ο αλγόριθμος επιστρέφει ως εμπόδιο κάθε τομή των ευθειών άρα και τα ακραία.

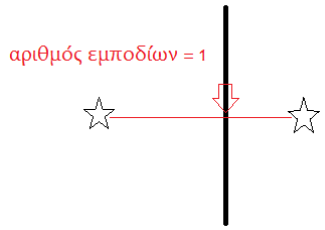


Figure 16 Σωστή εκτίμηση εμποδίου

Στην παράνω περίπτωση η τομή του εμποδίου με τη νοητή ευθεία επιστρέφει το σημείο τομής το οποίο δεν συμπίπτει με κανένα από εκείνα που βρίσκονται στους πίνακες `x, y` άρα η διαδρομή απορρίπτεται ως αδύνατη και η μαύρη γραμμή θεωρείται εμπόδιο.

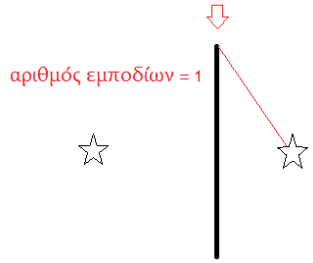


Figure 17 Λανθασμένη εκτίμηση εμποδίου

Στην παραπάνω περίπτωση ο αλγόριθμος θα επιστρέψει το σημείο τομής των δύο ευθειών που όμως δεν πρέπει να απορριφθεί καθώς υπάρχει στη λίστα `x, y`. Συνεπώς θα πρέπει να συμπεριληφθεί στο γράφο ορατότητας. Σημειώνεται πως και στις δύο περιπτώσεις μιλάμε για την πρώτη επανάληψη.

```
for(i=0; i<x1.length; i++){

let a=line(Math.round(curr_pos_x),Math.round(curr_pos_y),

Math.round(ap2_y),Math.round(ap2_x));let
c=line(Math.round(x1[i]),Math.round(y1[i]),

      Math.round(x2[i]),Math.round(y2[i]));

      var commonItemList= getCommonItems(a, c);
```

Οι τιμές των x1,x2,y1,y2 είναι τα ζευγάρια συντεταγμένων των ευθειών. Για παράδειγμα η ευθεία που ορίζεται απ' τα σημεία 10,15 και 50,15 θα βρίσκεται στους πίνακες ως x1[50], x2[15], y1[50], y2[15]. Αρχικά δημιουργούμε τις ευθείες a, c σύμφωνα με τον αλγόρθιμο του Bresenham. Συγκρίνουμε αν έχουν κοινά σημεία. Αν υπάρχουν κοινά σημεία τότε θα πρέπει ο αριθμός NumOfblocks να αυξηθεί όπως παρακάτω :

```
if(commonItemList.length!=0){
  NumOfblocks++;
}
```

Αν δε βρεθούν εμπόδια τότε εισάγουμε και ένα δεύτερο επίπεδο ασφάλειας με τη μέθοδο intersection.

```
if(commonItemList.length==0){
if(intersection(curr_pos_x,curr_pos_y,ap2_y,ap2_x,x1[i],y1[i],
x2[i],y2[i])!=null){
      NumOfblocks++;
}
}
```

Η μέθοδος intersection λαμβάνει ως ορίσματα τέσσερα σημεία που ορίζουν δύο ευθείες, και επιστρέφει το σημείο τομής τους αλλιώς null. Αν αυτή επιστρέψει κάποια τιμή τότε αυξάνουμε την τιμή NumOfblocks. Χρησιμοποιώντας μαζί και τις δύο μεθόδους εξασφαλίζουμε πως θα βρίσκεται πάντα η τομή των ευθειών. Σε περιπτώσεις που ένας απ' τους δύο αλγορίθμους χρησιμοποιήθηκε, προέκυψαν σφάλματα.

```
if(NumOfblocks==1 && commonItemList.length>0 &&
Math.round(curr_pos_x)==commonItemList[0][0]&&
Math.round(curr_pos_y)==commonItemList[0][1]){
      NumOfblocks=0;
}
```

Η μοναδική περίπτωση που θέλουμε να μην υπάρχει εμπόδιο , είναι εκείνη που το επόμενο σημείο είναι του τερματισμού. Γι αυτό το λόγο αν βρεθεί εμπόδιο `NumOfblocks==1 && commonItemList.length>0` ελέχουμε αν είναι η τρέχουσα θέση μας :

```
Math.round(curr_pos_x)==commonItemList[0][0]&&
Math.round(curr_pos_y)==commonItemList[0][1])
```

έστι ώστε να το απορρίψουμε.

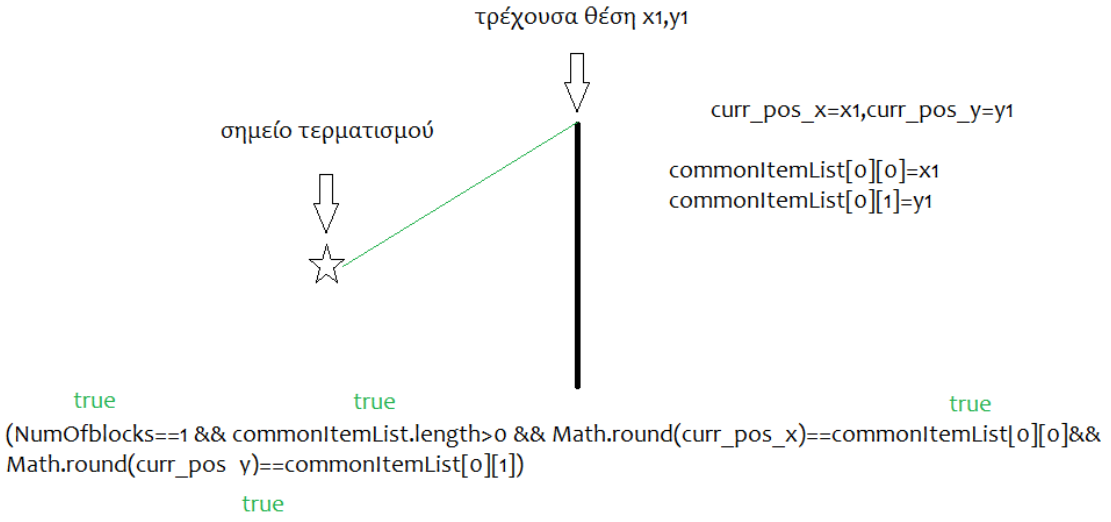


Figure 18 Εκτίμηση συνθήκης τερματισμού

Στο παραπάνω παράδειγμα παρόλο που οι γραμμές έχουν κοινό σημείο την τρέχουσα θέση, ο αλγόριθμος μηδενίζει τον μετρητή καθώς το σημείο τερματισμού είναι ορατό. Στην περίπτωση που έχουμε εμπόδιο αρχίζουμε τη διερεύνηση του γράφου ορατότητατας.

```
if(NumOfblocks>=1){
closest_x = [];
closest_y = [];
visible_x = [];
visible_y = [];
```

`closest_x, closest_y` – υποψήφια σημεία .

`visible_x, visible_y` – τα σημεία που είναι ορατά απ' την τρέχουσα θέση.

```
for(i=0;i<x1.length;i++){
dis1 = D(curr_pos_x,curr_pos_y,x1[i],y1[i]);
dis2 = D(curr_pos_x,curr_pos_y,x2[i],y2[i]); if(d>=dis1){
 if(x1[i]!=curr_pos_x && y1[i]!=curr_pos_y){
  closest_x.push(parseInt(x1[i]));
          closest_y.push(parseInt(y1[i]));
}
     }
if(d>=dis2){
if(x2[i]!=curr_pos_x && y2[i]!=curr_pos_y){
               closest_x.push(parseInt(x2[i]));
               closest_y.push(parseInt(y2[i]));
}}}
```

Στους πίνακες `closest_x, closest_y` εισάγουμε μόνο εκείνα τα στοιχεία από τους `x1,y1,x2,y2` για τα οποία η απόστασή τους είναι μικρότερη ή ίση της απόστασης d.

```
if(first_ap){
     for(i=0;i<closest_x.length;i++){
var CountIntersect = 0;
          for(j=0;j<x1.length;j++){
               var a=line(Math.round(curr_pos_x),
Math.round(curr_pos_y),
Math.round(closest_x[i]),
Math.round(closest_y[i]));
```

```
var c=line(Math.round(x1[j]),    Math.round(y1[j]),
     Math.round(x2[j]),          Math.round(y2[j]));var
commonItemList= getCommonItems(a, c);

if(commonItemList.length==0){

  if(intersection(curr_pos_x,    curr_pos_y,
     closest_x[i],               closest_y[i],
     x1[j],y1[j],    x2[j],y2[j])!=null){

CountIntersect++;

}

}

     if(commonItemList.length>0){

                          CountIntersect++;

                    }

               }

if(CountIntersect==1){

                    visible_x.push(closest_x[i]);

                    visible_y.push(closest_y[i]);

               }


}

first_ap=false;}
```

Στην περίπτωση που βρισκόμαστε στο σημείο εκκίνησης, δηλαδή `first_ap = true` τότε διατρέχουμε τους πίνακες `closest_x, closest_y` και εκτελώντας παρόμοια διαδικασία με προηγουμένως, μετράμε τον αριθμό των συγκρούσεων (`CountIntersect`) κάθε εμποδίου με τη νοητή ευθεία σημείου εκκίνησης – σημείου στους πίνακες `closest`. Αν τελικά η τιμή του `CountIntersect` είναι άσσος, τότε βρέθηκε μόνο το σημείο που πρακτικά δεν είναι εμπόδιο αλλά η άκρη του εμποδίου, άρα εισάγεται στον γράφο ορατότητας `visible_x, visible_y`.

Εισερχόμαστε τώρα στην περίπτωση που η τιμή `first_ap` δεν είναι αληθής. Πιο απλά σε όλες τις ενδιάμεσες περιπτώσεις μετάβασης από σημείο σε σημείο που η προηγούμενη δεν είναι το σημείο εκκίνησης και η επόμενη δεν είναι το σημείο τερματισμού. Ο αλγόριθμος δουλεύει με τον εξής περιορισμό. Απαγορεύεται η μετάβαση σε σημείο το οποίο ανήκει στην ίδια ευθεία με το σημείο τρέχουσας θέσης. Δε θεωρείται ορατό απ' τον αλγόριθμο και γι'αυτο το λόγο αποκλείεται πάντοτε απ' τους πίνακες `closest_x, closest_y` όπως παρακάτω:

```
let result =
getOppositeCoord(curr_pos_x,curr_pos_y,x1,y1,x2,y2);

closest_x.splice(closest_x.indexOf(result.x),1);

closest_y.splice(closest_y.indexOf(result.y),1);
```

Η μέθοδος `getOppositeCoord` λαμβάνει ως είσοδο την τρέχουσα θέση (στην πραγματικότητα είναι ένα σημείο κάποιου εμποδίου) και επιστρέφει την άλλη μεριά του εμποδίου.

```
function getOppositeCoord(curr_pos_x,curr_pos_y,x1,y1,x2,y2){
  let x,y;
  if(x1.includes(curr_pos_x.toString()) &&
     y1.includes(curr_pos_y.toString())){
     x=x2[x1.indexOf(curr_pos_x.toString())];
     y=y2[y1.indexOf(curr_pos_y.toString())];
  }
   if(x2.includes(curr_pos_x.toString()) &&
      y2.includes(curr_pos_y.toString())){
      x=x1[x2.indexOf(curr_pos_x.toString())];
      y=y1[y2.indexOf(curr_pos_y.toString())];
  }
  return {x:parseInt(x),y:parseInt(y)};
}
```

Αν υπάρχουν στους `x1`, `y1` και `x2`, `y2` αντίστοιχα τα `curr_pos_x, curr_pos_y` τότε στις τιμές `x, y` επιστρέφουμε τις αντίστροφες τιμές. Αφού έχουμε έτοιμους τους πίνακες `closest_x, closest_y` με τις υποψήφιες επόμενες θέσεις, ακολουθείται παρόμοια διαδικασία δημιουργίας δύο ευθειών και σύγκρισης.

```
for(i=0;i<closest_x.length;i++){
     var CountIntersect = 0;
```

```
      for(j=0;j<x1.length;j++){

      let a=line(Math.round(curr_pos_x),
      Math.round(curr_pos_y),
      Math.round(closest_x[i]),
      Math.round(closest_y[i]));

      let c=line(Math.round(x1[j]),
      Math.round(y1[j]),
      Math.round(x2[j]),
      Math.round(y2[j]));

      var commonItemList= getCommonItems(a, c);
if(commonItemList.length==0){
if(intersection(curr_pos_x,curr_pos_y,
closest_x[i],closest_y[i],
x1[j],y1[j],
x2[j],y2[j])!=null){
CountIntersect++;

      }
  }
  if(commonItemList.length!=0){
      CountIntersect++;
  }
 }
if(CountIntersect==2){
  visible_x.push(closest_x[i]);
  visible_y.push(closest_y[i]);
 }
```

```
}
```

Σε κάθε επανάληψη, όπως και παραπάνω, η νοητή ευθεία τρέχοντος σημείου `curr_pos_x`, `curr_pos_y` και υποψηφίου σημείου `closest_x[i]`, `closest_y[i]` δοκιμάζεται για συγκρούσεις με όλα τα εμπόδια. Αν βρεθεί εμπόδιο δηλαδή `commonItemList.length!=0` τότε η τιμή του `CountIntersect` αυξάνεται. Η ειδοποιός διαφορά σε αυτή την περίπτωση είναι η συνθήκη για να εισαχθεί ένα σημείο στον γράφο ορατότητας. Θα πρέπει η τιμή του `CountIntersect` να αντιστοιχεί σε ακριβώς δύο σημεία που μάλιστα είναι τα σημεία εκείνα, έτσι ώστε να μην παρεμβάλεται κανένα εμπόδιο.

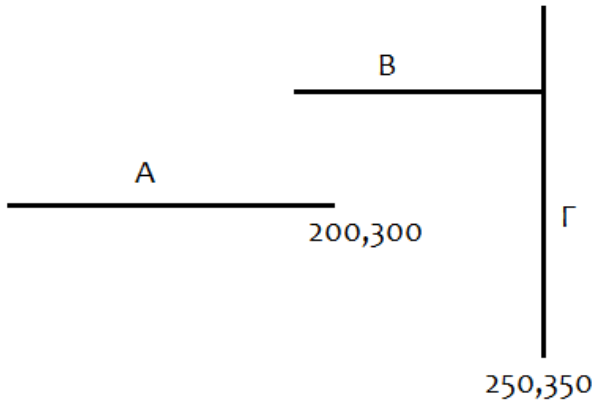


Figure 19 Παράδειγμα εσφαλμένου CounterIntersect = 2

Στο παραπάνω παράδειγμα έστω 200, 300 η τρέχουσα θέση. Έστω πως ο αλγόριθμος δοκιμάζει τη νοητή ευθεία 200, 300 ,250, 350 με τα εμπόδια Α, Β, Γ. Εφόσον τα σημεία 200, 300 και 250, 350 ανήκουν στα εμπόδια Α, Γ αντίστοιχα ο αλγόριθμος θα επιστρέψει `CountIntersect = 2` που όμως πρακτικά είναι οι τιμές 200, 300 και 250, 350. Άρα το σημείο 250,350 εισάγεται στον γράφο ορατότητας `visible_x, visible_y`. Η επόμενη ενέργεια είναι να ταξινομήσουμε τους πίνακες `visible_x, visible_y` ως προς το πόσο απέχουν απ' το σημείο τερματισμού. Αυτό είναι το κριτήριο επιλογής του επόμενου τρέχοντος σημείου άρα και της διαδρομής.

```
var distances=[];

for(i=0;i<visible_x.length;i++){

  distances.push({d:D(visible_x[i],visible_y[i],ap2_y,ap2_x),
x:visible_x[i],y:visible_y[i]})
```

```
}


distances.sort((a, c) => (a.d > c.d) ? 1 : -1);
curr_pos_x = distances[0].x;
curr_pos_y = distances[0].y;
```

Αρχικά δημιουργούμε έναν πίνακα αντικειμένων της μορφής {d: (απόσταση σημείου γράφου ορατότητας απ' το σημείο προορισμού), x: (τρεχον σημείο γράφου x) , y: (τρέχον σημείο γράφου y)}. Χρησιμοποιόντας τη sort ταξινομούμε τα στοιχεία του πίνακα distances και αρχικά αλλάζουμε τις τιμές των curr_pos_x, curr_pos_y ως τις νέες συντεταγμένες για την τρέχουσα θέση. Υπάρχει όμως το εξής πρόβλημα. Σε ορισμένες περιπτώσεις θα πρέπει να «θυμόμαστε» τις θέσεις που υπάρχουν ήδη στους πίνακες route_x, route_y ώστε όταν θα κληθούμε να επιλέξουμε επόμενο σημείο, αν το έχουμε ήδη επισκεφθεί, να μη το ξαναεπιλέξουμε.

```
var n=0;
while(route_x.includes(curr_pos_x) &&
route_y.includes(curr_pos_y)){
  curr_pos_x = distances[n].x;
  curr_pos_y = distances[n].y;
  n++;
}
route_x.push(curr_pos_x);
route_y.push(curr_pos_y);
```

Η παραπάνω διαδικασία ανιχνεύει αν στους πίνακες route_x, route_y υπάρχει η θέση που επιλέξαμε ως νέα τρέχουσα. Αν ναι, τότε στον ήδη ταξινομημένο πίνακα επιλέγει ως τρέχον σημείο το επόμενο. Αν και αυτό υπάρχει, το επόμενο κ.ο.κ. Αν τώρα η αρχική συνθήκη NumOfblocks>=1 δεν ισχυέι, τότε θα είναι 0 λόγω συνθήκης:

```
if(NumOfblocks==1 && commonItemList.length>0 &&
Math.round(curr_pos_x)==commonItemList[0][0]&&
Math.round(curr_pos_y)==commonItemList[0][1]){
  NumOfblocks=0;
```

```
}
```

Επομένως βρισκόμαστε στην τρίτη και τελευταία περίπτωση, το επόμενο σημείο να είναι εκείνο του τερματισμού. Τότε, δεν έχουμε παρά να το τοποθετήσουμε στους πίνακες `route_x, route_y`.

```
route_x.push(ap2_y);

route_y.push(ap2_x);
```

Γνωρίζοντας πλέον τις ενδιάμεσες θέσεις απ' τα σημεία εκκίνησης τερματισμού μπορούμε να αλλάξουμε τον πίνακα `stage1`, δηλαδή να προστεθούν οι ενδιάμεσες.

```
var u1=0;

if(v>3)

  u1= 1;

for(let u = u1; u<route_x.length; u++){
stage2.push({x:parseFloat(route_x[u]),y:parseFloat(route_y[u])
});

}

stage2.push({w:stage1[b][v+1].w});
```

Ο πίνακας `stage2` είναι στην πραγματικότητα ο `stage1` με τις ενδιάμεσες τιμές. Ας θυμηθούμε λίγο τη δομή του `stage1`

```
0: 1
1: {x: "978", y: "362"}
2: {w: 0}
3: {x: "418", y: "527"}
4: {w: 5000}
5: {x: "405", y: "362"}
6: {w: 8000}
7: {x: "413", y: "222"}
8: {w: 7000}
    9: {x: "559", y: "285"}
    10: {w: 7000}
    11: {x: "502", y: "219"}
```

```
13.     12: {w: 8000}
```

Ο έλεγχος πριν τον βρόχο γίνεται ώστε να διατηρηθεί η σωστή σειρά στον πίνακα `stage2` σύμφωνα με το πρότυπο του `stage1`. Για κάθε στοιχείο που εισάγουμε στον καινούριο πίνακα, εισάγεται και η αναμονή `w` που κάθε φορά βρίσκεται στη θέση `stage1[b][v+1].w`. Ο πίνακας `stage2` δεν είναι παρά ένας προσωρινός πίνακας. Ο πραγματικός πίνακας είναι ο `unified` ενώ ο `stage2` απλά αδειάζει στο τέλος κάθε επανάληψης

```
unified.push(stage2);

stage2 = [];
```

Ο πίνακας `unified` έχει τη μορφή όπως παρακάτω. Λαμβάνουμε όλες τις πληροφορίες κίνησης και αναμονής για κάθε επισκέπτη. Ο παρακάτω πίνακας περιέχει πληροφορίες για τέσσερις επισκέπτες. Γνωρίζουμε για παράδειγμα πως ο επισκέπτης 1 με κωδικό 0 ξεκίνησε απ' το σημείο `{x: 978, y: 362}`, συνέχισε στο `{x: 463, y: 589}` και ανέμενε 5 δευτερόλεπτα στο `{w: 5000}`. Έπειτα συνέχισε για το `{x: 274, y: 465}` κ.ο.κ

```
1. Array(4)
    1. 0: Array(17)
        1. 0: {x: 978, y: 362}
        2. 1: {x: 463, y: 589}
        3. 2: {x: 418, y: 527}
        4. 3: {w: 5000}
        5. 4: {x: 274, y: 465}
        6. 5: {x: 405, y: 362}
        7. 6: {w: 8000}
        8. 7: {x: 285, y: 251}
        9. 8: {x: 413, y: 222}
        10.     9: {w: 7000}
        11.     10: {x: 461, y: 142}
        12.     11: {x: 651, y: 249}
        13.     12: {x: 559, y: 285}
        14.     13: {w: 7000}
        15.     14: {x: 651, y: 249}
        16.     15: {x: 502, y: 219}
        17.     16: {w: 8000}
        18.     length: 17
        19.     __proto__: Array(0)
    2. 1: Array(17)
        1. 0: {x: 978, y: 362}
```

```
        2. 1: {x: 461, y: 142}
        3. 2: {x: 502, y: 219}
        4. 3: {w: 5000}
        5. 4: {x: 461, y: 142}
        6. 5: {x: 413, y: 222}
        7. 6: {w: 6000}
        8. 7: {x: 285, y: 251}
        9. 8: {x: 405, y: 362}
        10.     9: {w: 6000}
        11.     10: {x: 274, y: 465}
        12.     11: {x: 418, y: 527}
        13.     12: {w: 5000}
        14.     13: {x: 463, y: 589}
        15.     14: {x: 656, y: 463}
        16.     15: {x: 561, y: 429}
        17.     16: {w: 8000}
        18.     length: 17
        19.     __proto__: Array(0)
    3. 2: Array(15)
        1. 0: {x: 978, y: 362}
        2. 1: {x: 559, y: 285}
        3. 2: {w: 7000}
        4. 3: {x: 651, y: 249}
        5. 4: {x: 502, y: 219}
        6. 5: {w: 5000}
        7. 6: {x: 461, y: 142}
        8. 7: {x: 413, y: 222}
        9. 8: {w: 7000}
        10.     9: {x: 285, y: 251}
        11.     10: {x: 405, y: 362}
        12.     11: {w: 7000}
        13.     12: {x: 274, y: 465}
        14.     13: {x: 418, y: 527}
        15.     14: {w: 8000}
        16.     length: 15
        17.     __proto__: Array(0)
    4. 3: Array(16)
        1. 0: {x: 978, y: 362}
        2. 1: {x: 461, y: 142}
        3. 2: {x: 502, y: 219}
        4. 3: {w: 7000}
        5. 4: {x: 651, y: 249}
        6. 5: {x: 559, y: 285}
        7. 6: {w: 7000}
        8. 7: {x: 561, y: 429}
        9. 8: {w: 8000}
        10.     9: {x: 656, y: 463}
        11.     10: {x: 463, y: 589}
        12.     11: {x: 418, y: 527}
        13.     12: {w: 7000}
        14.     13: {x: 274, y: 465}
```

```
15.     14: {x: 405, y: 362}
16.     15: {w: 8000}
```

2.1.9 Κινήσεις στο 2δ επίπεδο

Δημιουργούμε τον πίνακα αντικειμένων `apinfo` που κρατά τα `id` των σημείων πρόσβασης μαζί με τις συντεταγμένες τους

```
for(i=0; i<id.length; i++){


apinfo.push([{id:id[i]},{x:parseFloat(y[i]),y:parseFloat(x[i])
}]);

}
```

Στη συνέχεια δημιουργούμε τα `id` selectors για jQuery για κάθε επισκέπτη μαζί με τα εικονίδιά τους

```
for(i=0; i<max_vis; i++){

  ids.push('#visitor'+i);

      var div = $('<div style="background-image:
url(./visitor.png);height:20px;width:10px;background-
size:10px;position:absolute;"></div>').css({

        "left": parseFloat(y[0]) + 'px',

        "top": parseFloat(x[0]) + 'px',

      });

    $('#parent').append(div);

    div.attr('id', 'visitor'+(i));

}
```

Ο πίνακας ids περιέχει selectors της μορφής '#visitor1' '#visitor2'. Κάθε εικονίδιο επισκέπτη έχει δημιουργηθεί στη θέση εισόδου δηλαδή x[0], y[0]

```
var delay = 0;
for (i=0; i<unified.length; i++){
$(ids[i]).toggle().delay(delay).fadeIn( 400 );
delay=delay+5000;
}
```

Κάθε στοιχείο του πίνακα unified δηλαδή κάθε επισκέπτης, λαμβάνει μια καθυστέρηση πέντε δευτερολέπτων αθροιστικά. Ο πρώτος πέντε, ο δεύτερος δέκα κ.ο.κ. Δε θέλουμε να εισάγονται μαζεμένοι, έτσι ορίζουμε αυθαίρετα το χρόνο καθυστέρησης χάριν ευκρίνειας. Δυο μεγάλοι εμφωλευμένοι βρόχοι θα διατρέχουν τον πίνακα unified. Ο εξωτερικός διατρέχει κάθε επισκέπτη ενώ ο εσωτερικός διατρέχει την κάθε κίνηση του επισκέπτη.

```
for (i=0; i<unified.length; i++){
      for(j=1; j<unified[i].length; j++){
```

Εδώ διακρίνονται δύο κατηγορίες. Ελέγχουμε το αναγνωριστικό του κάθε αντικειμένου. 1) αν είναι x σημαίνει πως έχουμε κίνηση 2) αν είναι w σημαίνει πως έχουμε αναμονή. Αναλόγως λοιπόν δημιουργούμε και τις κινήσεις.

```
if(Object.getOwnPropertyNames(unified[i][j])[0]=="x"){
```

Αν δηλαδή συναντήσουμε κίνηση :

```
$(ids[i]).animate({left:
unified[i][j].x,top:unified[i][j].y},{
             duration:3000,
start:function(){},complete:function(){}})
```

Δημιουργούμε μια κίνηση διάρκειας τριών δευτερολέπτων προς το σημείο που ορίζεται στα `unified[i][j].x, unified[i][j].y` , επάνω στο στοιχείο του DOM που περιγράφεται στο `ids[i]`. Η μέθοδος `start` καλείται λίγο πριν το ξεκίνημα της κίνησης ενώ η `complete` αμέσως μετά το τέλος. Θα ξεκινήσουμε απ' την `complete`

```
complete:function(){
let p = $(this).position();
for(k=1; k<apinfo.length; k++){
     if(p.left == apinfo[k][1].x &&
p.top == apinfo[k][1].y){        var iD1 = apinfo[k][0].id;
          var finalId = '#ap'+iD1;
          var el =
parseInt($(finalId).children('div').text());
     $(finalId).children('div').text(el+1);    }}}
```

Η μεταβλητή κρατά το `context` του στοιχείου του DOM. Έτσι όταν το `callback complete` αρχίσει να εκτελείται γνωρίζουμε με το `this` τη θέση του στοιχείου στο DOM. Σκοπός μας είναι να αλλάξουμε την τιμή που μετρά τους ταυτόχρονους επισκέπτες. Ας δούμε λίγο τη δομή του κάθε στοιχείου σημείου πρόσβασης όπως αποτυπώνεται στο DOM.

```
for(i=0;i<id.length;i++){
  let vis_counter="<div>0</div>";
   var url = 'url(./ap.png)';
    if(id[i]==1){
      var url = 'url(./exit.png)';
      vis_counter=''
    }
    var div = $('<div class="puck"
style="position:absolute;height:25px;width:25px;background-
size:25px;margin-left:-5px;margin-top:-5px">')
      .css({
        "left": parseInt(y[i]) + 'px',
```

```
        "top": parseInt(x[i])+ 'px',
        "background-image": url
      })
      .append('<span style="margin-left:-
6px">'+id[i]+'</span>',$(vis_counter))
      .appendTo(document.body);
        div.attr('id', 'ap'+(i+1));
   }
```

Η μετάβλητη `vis_counter` αρχικοποιεί τον αριθμό των ταυτόχρονων επισκεπτών ως ένα `div` στοιχείο με τον αριθμό 0 και το `url` για την εικόνα του. Για την είσοδο που έχει πάντα το `id = 1` προφανώς δεν υπάρχει μετρητής ταυτόχρονων επισκεπτών και η εικόνα του στο χάρτη είναι διαφορετική. Τα στοιχεία αυτά εμφανίζονται όπως παρακάτω:

Figure 20 Αριστερά : Σημείο πρόσβασης - Δεξιά σημείο εισόδου

Ο επάνω αριθμός αντιπροσωπεύει το `id` του σημείου πρόσβασης ενώ ο κάτω τον αριθμό των ταυτόχρονων επισκεπτών στο σημείο. Στη μέθοδο `complete` λοιπόν αν ο επισκέπτης μας φτάσει σε ένα απ' τα σημεία που υπάρχουν στον `apinfo` τότε η μεταβλητή `finalId` περιέχει τον `selector` του σημείου πρόσβασης στο οποίο μόλις έφτασε ο επισκέπτης. Στην τιμή `el` αποθηκεύεται η παλιά τιμη του `div` στοιχείου του συγκεκριμένου σημείου πρόσβασης και στη συνέχεια προστίθεται ένα.

```
start:function(){
              let p = $(this).position();
              for(k=1; k<apinfo.length; k++){
                  if(p.left == apinfo[k][1].x &&
p.top == apinfo[k][1].y){
                    var iD1 = apinfo[k][0].id
                    var finalId = '#ap'+iD1;
```

```
                    var el = parseInt($(finalId).
children('div').text());
                    $(finalId).children('div').text(el-1);
                  }
              }
            }
```

Η μέθοδος `start` ακολουθεί την ίδια λογική μόνο που τώρα θα πρέπει να αφαιρέσουμε 1 στο τέλος. καθώς ξεκινώντας η κίνηση ταυτόχρονα φεύγει και απ' το συγκεκριμένο σημείο. Αν η τιμή του αντικειμένου περιέχει `w` τότε προσθέτουμε την ανάλογη καθυστέρηση στο στοιχείο:

```
if(Object.getOwnPropertyNames(unified[i][j])[0]=="w"){
          $(ids[i]).delay(unified[i][j].w);
        }
```

Σε αυτό το σημείο έχουμε εισάγει και έχουμε θέσει σε κίνηση τους επισκέπτες σύμφωνα με τα δεδομένα του πίνακα `unified`. Χάρη στην ασύγχρονη εκτέλεση των `animations` μπορούμε να προσθέτουμε συνεχώς `callbacks` στη στοίβα κλήσεων κάθε DOM element. Σύμφωνα με αυτό, οτιδήποτε ακολουθεί, εκτός των μεγάλων βρόχων `for`, θα πρέπει να εκτελεστεί τελευταίο. Θα προσθέσουμε μια κίνηση `fadeOut` σε κάθε επισκέπτη μόλις τελειώσει την ακολουθία του από κινήσεις και παύσεις. Στο τέλος της κίνησης σύμφωνα με τον `unified` θα υπάρχει πάντα αντικείμενο με (`w`) όποτε μετά και απ' την τελευταία αναμονή θα αρχίσει να εξαφανίζεται σιγά-σιγά απ' το οπτικό πεδίο το εικονίδιο του επισκέπτη.

```
    for (i=0; i<unified.length; i++){
      $(ids[i]).fadeOut({start:function(){
        let p = $(this).position();
        for(k=0; k<apinfo.length; k++){
          if(p.left == apinfo[k][1].x && p.top ==
apinfo[k][1].y){
            var iD1 = apinfo[k][0].id
            var finalId = '#ap'+iD1;
```

```
            var el =
parseInt($(finalId).children('div').text());
            $(finalId).children('div').text(el-1);
          }
       }
      }});
   }
```

Δημιουργούμε λοιπόν και προσάπτουμε σε κάθε αντικείμενο επισκέπτη μια κίνηση `fadeOut` με τον ίδιο τρόπο όπως και παραπάνω. Επίσης, μειώνουμε τον μετρητή επισκεπτών του σημείου πρόσβασης μιας και ο συγκεκριμένος επισκέπτης δεν βρίσκεται πλέον στον χώρο.

```
function stop(){
   var array = new Array();
    $('div','#parent').each(function(){
      array.push($(this).attr('id'));
    });
    for(i=0; i<array.length; i++){
      var finalId = '#'+array[i];
      $(finalId).stop( true, true ).fadeOut();
    }
}
```

Η μέθοδος stop σταματάει όλες τις αλυσίδες `animation` πάνω στα στοιχεία με τη μέθοδο `stop` και τους προσδίδει ένα ωραίο `fadeOut`. Στον πίνακα `array` βρίσκονται όλα τα στοιχεία του DOM που έχουν γονιό το '`parent`'.

2.1.10 Θερμογράφημα

Έφτασε η στιγμή να αποτυπώσουμε στο επίπεδο τα δεδομένα απ' τις κινήσεις των επισκεπτών. Ας εξετάσουμε για λίγο τα βασικά στοιχεία απ' τη βιβλιοθήκη `heatmap.js`.

```
<div id='heatmap'>
<canvas class="heatmap-canvas"></canvas>
</div>
```

Ορίζουμε ένα `div` και ένα στοιχείο `canvas` μέσα του με όνομα `heatmap-canvas`.

```
var heatmapInstance = h337.create({
  container: document.querySelector('.heatmap')
});
```

Δημιουργούμε ένα instance του αντικειμένου `h337` με τη μέθοδο create. Απαραίτητο και υποχρεωτικό στοιχείο είναι το στοιχείο DOM `container` πάνω στο οποίο θα δημιουργηθεί ο χάρτης. Υπάρχουν πολλές προεραιτικές τιμές που μπορούν να αρχικοποιηθούν κατά τη δημιουργία ενός `h337` instance. Κάποιες απ' αυτές είναι οι `maxOpacity`, `minOpacity`, `blur`, `onExtremaChange`.

```
var points = [];


  var point = {
    x: Math.floor(Math.random()*5),
    y: Math.floor(Math.random()*5),
    value: val,
    radius: radius
  };
  points.push(point);
}
var data = {
  max: 50,
  data: points
};
```

```
heatmapInstance.setData(data);
```

Η εισαγωγή των σημείων στο χάρτη του θερμογραφήματος γίνεται συνήθως σε datasets αλλά και μεμονωμένα. Ο πίνακας `points` αντιπροσωπεύει το dataset που θα εισαχθεί. Δημιουργούμε τυχαία δύο τιμές για `x, y` που αναπαριστούν τη θέση του σημείου στο χάρτη καθώς και μια τιμή `value`. Όσο μεγαλύτερο το νούμερο στη `value` τόσο εντονότερο χρώμα κατέχει το σημείο. Η προαιρετική τιμή `radius` δηλώνει την ακτίνα του κύκλου που θα σχηματίζεται με κέντρο το ίδιο το σημείο. Τέλος στο αντικείμενο `data` εισάγουμε και μια τιμή μεγίστου. Η `setData` ορίζει τα σημεία στο αντικείμενο `data` πάνω στο `heatmapInstance` αντικείμενο. Ας εξετάσουμε αναλυτικά πώς χτίζεται το θερμογράφημα για να αναπαραστήσουμε δεδομένα. Ισχύουν κάποιοι κανόνες για τα σημεία του χάρτη. 1) Η τιμή κάθε σημείου στο χάρτη (`value`) αντιστοιχεί σε 1. Για κάθε 1 `heat` στο ίδιο σημείο η τιμή του αυξάνεται αθροιστικά. Για παράδειγμα αν ένα σημείο έχει `heat = 15` , θα του προστεθεί 15 φορές η τιμή 1. Η τιμή της ακτίνας `radius` είναι κυμαινόμενη ανάλογα με τον αριθμό των επισκέψεων, λίγες επισκέψεις μικρή ακτίνα και το αντίθετο. Η μέγιστη τιμή της ακτίνας καθορίζεται απ' την τιμή του πίνακα `settings` με όνομα `max_radius`. Οι διαδρομές ανάμεσα στα σημεία έχουν και αυτές σημεία πάνω σε μια νοητή ευθεία ανά έξι πίξελ με τιμές επιλεγμένες για λόγους ευκρίνειας, `value = 35 , radius = 50`. Οι τιμές αυτές είναι σταθερές.

```
$query1 = "select max_radius from settings";
$settings = mysqli_query($conn,$query1);
$row2 = mysqli_fetch_array($settings);
$max_radius = $row2['max_radius'];
```

Εδώ ζητούμε απ' τη βάση την τιμή `max_radius` της μέγιστης ακτίνας.

```
$dates = [];
$query = "select distinct date from past_instances";
$res = mysqli_query($conn,$query);
while($row = mysqli_fetch_array($res)){
  array_push($dates,$row['date']);
}
```

Ζητούνται απ' τη βάση όλες τις μοναδικές τιμές ημερομηνιών του πίνακα `past_instances`. Ο πίνακας `dates` θα μας βοηθήσει να δημιουργήσουμε ένα δυναμικό dropdown μενού με όλες τις διαθέσιμες παλαιότερες καταστάσεις του θερμογραφήματος.

```
<select name="dropdown" id="dropdown">
  <option value="Aggregate" selected>Aggregate</option>
  <?php for($i = 0; $i < count($dates); $i++) {
    echo "<option value='$dates[$i]'>$dates[$i]</option>";
  }
   ?>
</select>
```

Εδώ δημιουργούμε δυναμικά ένα `dropdown` με πεδία όσα και του πίνακα `dates`. Η τιμή του `label` και του `value` κάθε επιλογής είναι η ίδια η ημερομηνία:

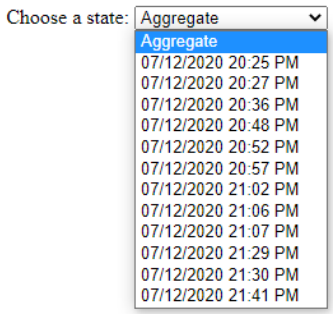


Figure 21 Παλαιότερες καταστάσεις θερμογραφήματος - Dropdown

Ας δημιουργήσουμε ένα instance του `h337` αυτή τη φορά με υπόμνημα κάτω αριστερά.

```
<div id='heatmapContainer' style='height: 662px;width:
1000px;'>
 <div class="puck" style="visibility:
hidden"><span>&#128959;</span></div>


  <div id="heatmapLegend"  style="position: absolute;
                bottom: 0;
```

```
                width: 200px;
                height: 50px;
                border: 1px solid red;">

        <span id="min" style="float: left;"></span>
        <span id="max" style="float: right;"></span>
        <img id="gradient" src="" style="width:100%" />
      </div>

<canvas id="c" width="1000" height="662" style="border:1px
solid #ccc">


</div>
```

Το div με `id = heatmapLegend` εμφανίζεται στην απόλυτη θέση 200 πίξελ απ' το κάτω μέρος της οθόνης και έχει ύψος 50 πίξελ. Περιέχει δυο `spans`, ένα για την ελάχιστη τιμή και ένα για τη μέγιστη και μια εικόνα `gradient` που αναπαριστά τη χρωματική κλίμακα που βλέπουμε στο χάρτη.

Figure 22 Υπόμνημα θερμογραφήματος

```
var config = {
  container: document.getElementById('heatmapContainer'),
  maxOpacity: 1,
  minOpacity: 0,
  blur: .75,
  onExtremaChange: function(data) {
    updateLegend(data);
  }
```

```
};
var heatmapInstance = h337.create(config);
```

Δημιουργούμε ένα instance του `h337` με τα περιεχόμενα του `config`. Το πεδίο `onExtremaChange` περιέχει ένα `callback` που σκανδαλίζεται σε κάθε αλλαγή δεδομένων πάνω στο heatmapInstance. Τότε καλείται η `updateLegend`. Η τιμή του data εισάγεται απ' το `callback`. Ορίζουμε τις απαραίτητες τιμές για το υπόμνημα.

```
var legendCanvas = document.createElement('canvas');
legendCanvas.width = 100;
legendCanvas.height = 10;
var min = document.querySelector('#min');
var max = document.querySelector('#max');
var gradientImg = document.querySelector('#gradient');
var legendCtx = legendCanvas.getContext('2d');
var gradientCfg = {};
```

Δημιουργούμε μέσα στο wrapper στοιχείο, `heatmapLegend` ένα canvas στοιχείο 100x10. Δημιουργούμε αναφορές στα τρία στοιχεία του και ένα στοιχείο αναφοράς `context` όπως ορίζει η χρήση του `canvas`. Επίσης αρχικοποιείται ένα κενό αντικείμενο `gradientCfg`.

```
function updateLegend(data) {
  min.innerHTML = data.min;
  max.innerHTML = data.max;
  if (data.gradient != gradientCfg) {
    gradientCfg = data.gradient;
    var gradient = legendCtx.createLinearGradient(0, 0, 100,
1);
    for (var key in gradientCfg) {
      gradient.addColorStop(key, gradientCfg[key]);
    }
```

```
    legendCtx.fillStyle = gradient;
    legendCtx.fillRect(0, 0, 100, 10);
    gradientImg.src = legendCanvas.toDataURL();
  }
}
```

Στο `data` υπάρχουν τιμές `min`, `max`, `gradientConfig` ώστε να μπορούμε να ανανεώσουμε το χάρτη. Η τιμή `data` επιστρέφεται απ' το `callback` της `onExtermaChange`. Θέτουμε τις τιμές στα `min`, `max`, `gradientCfg` και δημιουργούμε την κλίμακα χρωμάτων με τις μεθόδους `addColorStop` που θέτει τα όρια των χρωμάτων και `fillRect` που γεμίζει με χρώμα.

```
$('#dropdown').on('change', function() {
       var value = $('#dropdown option:selected').text();

         $.ajax({
            type: "POST",
            url: "get_heatmap_data.php",
            data: { value: value},
            dataType : 'json',
             success: function(result) {}});

... }) .trigger('change');
```

Όταν ανιχνεύεται αλλαγή στο dropdown menu εκτελούμε ένα ασύγχρονο `POST` αίτημα στη βάση και ζητούμε τα δεδομένα της αναφοράς `value` που δείχνει κάθε φορά στο επιλεγμένο στοιχείο του dropdown, σε μορφή json. Στο τέλος, αναγκάζουμε την πρώτη αλλαγή να συμβεί αυτόματα, κάνοντας `chain` τη μέθοδο `trigger` με το γεγονός `change` ώστε να φορτωθεί μια αρχική κατάσταση, όταν η σελίδα φορτωθεί.

```
if($value=="Aggregate"){
$query = "select * from access_point";
```

```
$ap_info = mysqli_query($conn,$query);
while($row = mysqli_fetch_array($ap_info)){
  $id_t = $row['id'];
  $x_t = $row['x'];
  $y_t = $row['y'];
  $heat_t = $row['heat'];
  $times_visited_t = $row['times_visited'];
array_push($id,$id_t);
array_push($x,$x_t);
array_push($y,$y_t);
array_push($heat,$heat_t);
array_push($times_visited,$times_visited_t);
}
 $data = array(
        'id' => $id,
        'x' => $x,
        'y' => $y,
        'times_visited' => $times_visited,
        'heat' => $heat
     );
 $data = array_values($data);
 echo json_encode($data);
```

Αν έχει επιλεχθεί η τιμή `Aggregate` τότε θα ζητήσουμε τα δεδομένα απ' τον πίνακα `access_point`. Έπειτα, δημιουργούμε την επιστρεπτέα τιμή `data` και παραμετροποιούμε τις τιμές της. Στη συνέχεια, τυπώνουμε το αποτέλεσμα ώστε να επιστραφεί στον client σαν αποτέλεσμα. Αν δεν έχει επιλεγεί το `Aggregate`:

```
$query4 = "select state_id from past_instances where
date='$value'";
$res4 = mysqli_query($conn,$query4);
```

```
$row4 = mysqli_fetch_array($res4);

$state = $row4['state_id'];
```

Αναζητούμε τον κωδικό κατάστασης μέσω της ημερομηνίας.

```
$query = "select
access_point.id,access_point.x,access_point.y,
SUM(past_instances.times_visited) AS times_visited_sum
,SUM(past_instances.heat) AS heat_sum

from past_instances inner join access_point on access_point.id
= past_instances.id WHERE state_id<='$state' GROUP BY id";
```

Αυτό το ερώτημα είναι μια εσωτερική σύνθεση (inner join) δύο πινάκων με μετονομασία πεδίων. Επιστρέφει `id, x, y, times_visited_sum` ως αθροισμα όλων των τιμών, `heat_sum` ως άθροισμα όλων των εγγραφών.

Κάθε εγγραφή λοιπόν του πίνακα `past_instances` που έχει `state_id` μικρότερο ή ίσο της τιμής `$state` θα αθροίσει τα πεδία `times_visited` και `heat` όπου είναι ίδια τα `id` και θα τα επιστρέψει ως `times_visited_sum, heat_sum`. Ο πίνακας `access_point` θα επιστρέψει τα `id,x,y` όπου το `id` της εγγραφής είναι ίσο με το `id` της εγγραφής στο `past_instances`.

id	x	y	times_visited_sum	heat_sum
1	362	978	0	0
2	429	561	31	573
3	285	559	28	522
4	219	502	30	536
5	222	413	23	407
6	362	405	25	487
7	527	418	33	664

Figure 23 Αποτέλεσμα ερωτήματος σύνθεσης

```
$ap_info = mysqli_query($conn,$query);

while($row = mysqli_fetch_array($ap_info)){

  $id_t = $row['id'];

  $x_t = $row['x'];

  $y_t = $row['y'];

  $heat_t = $row['heat_sum'];
```

```
  $times_visited_t = $row['times_visited_sum'];
array_push($id,$id_t);
array_push($x,$x_t);
array_push($y,$y_t);
array_push($heat,$heat_t);
array_push($times_visited,$times_visited_t);
}
 $data = array(
        'id' => $id,
        'x' => $x,
        'y' => $y,
        'times_visited' => $times_visited,
        'heat' => $heat
     );
 $data = array_values($data);
 echo json_encode($data);
```

Ακολουθείται η ίδια διαδικασία μόλις επιστρέψουν τα αποτελέσματα. Στη συνέχεια, θα εκτελεστεί το περιεχόμενο της `success` όπου στο `result` υπάρχουν ως πεδία πίνακα όλα τα στοιχεία που λάβαμε απ' τη βάση. Στο πεδίο `result[0]` υπάρχει το `id` και τα υπόλοιπα με τη σειρά που εισήχθησαν στην `$data`.

```
success: function(result) {
var id = result[0];
var x = result[1];
var y = result[2];
var times_visited = result[3];
var heat = result[4];
```

Σε αυτό το σημείο θα πρέπει να αρχικόποιήσουμε το θερμογράφημα με ένα κενό σύνολο. Σε περίπτωση που υπάρχουν στοιχεία εκεί από προηγούμενη κλήση (πχ., αν

αλλάξει ο χρήστης επιλογή στο dropdown), διαγράφονται ώστε να μην έχουμε overwrite των τιμών άρα διπλές τιμές στα σημεία του.

```
heatmapInstance.setData({data:[]});


var points = [];
var max_radius = <?php echo $max_radius; ?>;
for(i=0;i<x.length;i++){
 for(j=0;j<heat[i];j++){
if(times_visited[i]>max_radius){
     var radius=max_radius;
 }else{
     var radius = times_visited[i];
 }
 var point = {
   x: y[i],
    y: x[i],
    value: 1,
    radius: radius
  }
  points.push(point);
     }
}
```

Διατρέχουμε τους πίνακες `x, y` που περιέχουν τις τιμές των συντεταγμένων των σημείων ενδιαφέροντος και εισάγουμε τις τιμές σύμφωνα με τους κανόνες. Διευκρινίζεται πως η τιμή του `radius` είναι `max_radius` αν η τιμή `times_visited` την υπερβαίνει.

```
var b=heat.map(Number);
var mx = Math.max.apply(Math, b);
```

```
var data = {
  max:mx,
  data: points
};
heatmapInstance.setData(data);
```

Ως μέγιστη τιμή θέτουμε τη μεγαλύτερη τιμή του `heat` πίνακα. Πρώτα μετατρέπουμε τον πίνακα από `String` σε `Number` και βρίσκουμε το μέγιστο. Τα δεδομένα εισέρχονται μέσω της `setData` στο χάρτη.

2.1.11 Τοποθέτηση διαδρομών στο χάρτη

Εισαγάγαμε τα δεδομένα που αφορούν στα σημεία πρόσβασης. Τι γίνεται όμως με τα υπόλοιπα σημεία του επιπέδου; Πρέπει και αυτά να έχουν πληροφορία και δη τα δεδομένα των διαδρομών. Ο αλγόριθμος είναι ο ίδιος αλγόριθμος εύρεσης μονοπατιού με αποφυγή εμποδίου. Για κάθε ζεύγος σημείων θα υπόλογιζεται `route_x`, `route_y`. Για κάθε ευθεία που ορίζεται απ' τα σημεία του μονοπατιού ο αλγόριθμος Bresenham θα εξάγει τις συντεταγμένες των ενδιάμεσων σημείων mod 6. Αυτό σημαίνει πως η ευθεία είναι πιο «αραιή» και αυτό συμβαίνει για λόγους ταχύτητας. Αυτά τα σημεία, έπειτα, θα εισάγονται στο χάρτη με σταθερή τιμή 35 και ακτίνα 50.

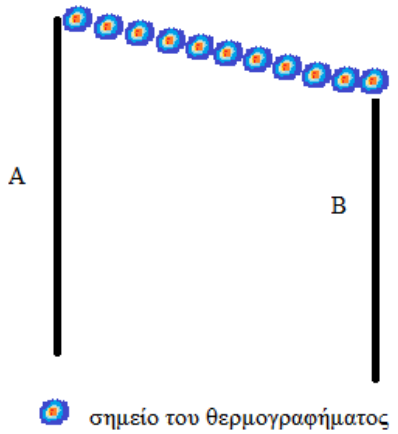


Figure 24 Αναπαράσταση σημείων ευθείας

Κάθε σημείο της παραπάνω εικόνας έχει επιστραφεί απ' τον Bresenham ως κομμάτι του υπολόγισμού των ενδιάμεσων σημείων ανάμεσα στις κορυφές των A, B. Πριν φτάσουμε όμως εκεί, θα πρέπει ο αλγόριθμος κίνησης να γνωρίζει όλες τις πιθανές διαδρομές που μπορεί να προκύψουν ανάμεσα στα σημεία πρόσβασης ώστε να

υπολογίσει τα μονοπάτια και να εξάγει τις συντεταγμένες που εν συνεχεία θα απεικονιστούν στο θερμογράφημα. Το μοναδικό δεδομένο που έχει στη διάθεσή του είναι το πεδίο `adj_mat` που δίνει τις σχέσεις μεταξύ των σημείων. Αυτή όμως η πληροφορία θα πρέπει να συνδυαστεί κατάλληλα ώστε αφενός να είναι αναγνώσιμη, αφετέρου να μην επαλαμβάνονται υπολογισμοί της ίδιας διαδρομής. Για παράδειγμα, αν υπάρχει αμφίδρομη διάσχιση δύο σημείων, αρκεί ένας υπολογισμός προς οποιαδήποτε κατεύθυνση. Το πεδίο `adj_mat` περιέχει, όπως γνωρίζουμε, ένα αλφαριθμητικό της μορφής: 1#2,3,4,7;2#3,4,7;3#2,4,7;4#2,3,5;5#3,4,6;6#5,7;7#2,6. Θα θέλαμε να το μετατρέψουμε σε κάτι της μορφής 12,13,14,17,23,24,27,34,37,45,53,56,67. Η συγκεκριμένη συμβολοσειρά διαβάζεται ως εξής: Απ' το σημείο 1 μπορώ να μετακινηθώ στα 2, 3, 4, 7. Απ' το σημείο 2 μπορώ να μετακινηθώ στα 3, 4, 7 κ.ο.κ.

```
var res = adj_mat.split(";");
if(res[res.length]==null)
res.pop();
```

Αρχικά χωρίζουμε το πρώτο αλφαριθμητικό σε επιμέρους στοιχεία όπου υπάρχει (;). Αν η τελευταία θέση του πίνακα προκύψει κενή τότε δε μας χρειάζεται.

```
for(i=0; i<res.length; i++){
      var res2 = res[i].split("#");
      var res3 = res2[1].replace(/,/g,"");
      for(j=0; j<res3.length; j++){
            neighbours.push(res2[0]+res3[j]);
      }
}
```

Ο πίνακας `res` χωρίζεται σε επιμέρους κομμάτια. Ο πίνακας `res2` περιέχει στο πρώτο του κελί οτιδήποτε βρίσκεται αριστερά της δίεσης (#) και στο δεύτερο κελί το υπόλοιπο αλφαριθμητικό. Για το πρώτο κελί του `res` έχουμε : `res2[0] = 1` `res[1] =2,3,4,7`. Στο `res3` αποθηκεύεται το `res2[1]` χωρίς κόμματα. Στον πίνακα `neighbours` τελικά εισάγεται ο συνδυασμός του `res2[0]` με το επόμενο στοιχείο του res3 κάθε φορά. Στην πρώτη επανάληψη ο `neighbours` θα έχει το '12', στη δεύτερη το '13', στην τρίτη το '14', και στην τελευταία το '17'. Σε αυτό το σύστημα θα υπάρξουν και διπλοεγγραφές. '27' και '72' πρακτικά δίνουν την ίδια ακριβώς πληροφορία.

```
for(i=0; i<neighbours.length; i++){
  if(final.length==0){
    final.push(neighbours[i]);
  }
  else{
    var l = 0;
    do{
      var count1 = count(neighbours[i],final[l]);
      l++;
      if(count1==2)
        break;
      }while(l<final.length);
    if(count1==0 || count1==1){
      final.push(neighbours[i]);
    }
  }
}
```

Ο πίνακας είναι κενός οπότε θα εισαχθεί αναγκαστικά το πρώτο στοιχείο του πίνακα `neighbours`. Στη συνέχεια κάθε στοιχείο που εισάγεται, ελέγχεται με όλα τα προηγούμενα ως προς την ομοιότητα των δύο ψηφίων του με οποιοδήποτε. Η μέθοδος `count` επιστρέφει 0,1 ή 2. Όσο έχουμε κάνενα ή ένα κοινό ψηφίο τότε με ασφάλεια το ζευγάρι τοποθετείται στον τελικό πίνακα `final`. Αν οποιαδήποτε στιγμή βρεθεί ίδιο ζευγάρι, δηλαδή `count1 = 2` τότε τα υπόλοιπα στοιχεία δεν ελέγχονται και φυσικά το ζευγάρι δεν εισάγεται στον `final`.

```
function count (a,b){
  var count = 0;
  if(a[0]==b[0])
    count++;
  if(a[0]==b[1])
```

```
    count++;
  if(a[1]==b[0])
    count++;
  if(a[1]==b[1])
    count++;
   return count;
}
```

Η μέθοδος count σύγκρίνει τους τέσσερις δυνατούς συνδυασμούς των ζευγαριών. Για παράδειγμα η σύγκριση των συμβολοσειρών ‘12’ και ‘34’ θα επιστρέψει 0 καθώς ούτε ένα ψηφίο δεν είναι ίδιο μέσα σε αυτές. Ο πίνακας `final` δίνει τις θέσεις των πινάκων `x, y` τις οποίες μεταφράζουμε στις συντεταγμένες τους.

```
for(t=0; t<final.length; t++){
  var curr_ap1_x = parseFloat(x[id.indexOf(final[t][0])]);
  var curr_ap1_y = parseFloat(y[id.indexOf(final[t][0])]);
  var curr_ap2_x = parseFloat(x[id.indexOf(final[t][1])]);
  var curr_ap2_y = parseFloat(y[id.indexOf(final[t][1])]);


...
```

Ο μεγάλος βρόχος ξεκινάει τον αλγόριθμο εύρεσης μονοπατιού για κάθε ζευγάρι που υπάρχει στον πίνακα `final`. Όπως και προηγουμένως, υπάρχουν οι συντεταγμένες σημείων έναρξης και τερματισμού. Για παράδειγμα, η τιμή του `curr_ap1_x` θα ισούται με την τιμή της έκφρασης δεξιά. Διαβάζεται από μέσα προς τα έξω. Έστω :

```
final                x                    id

0: "12"              0: 150               0: 1

1: "13"              1: 395               1: 2

2: "14"              2: 30                2: 3

3: "17"              3: 45                3: 4

…                    …
```

Έχουμε `final[0][0] = '12' , id.indexOf('12') = 0, x[0]`. Άρα η τελική τιμή της έκφρασης είναι 150. Ο αλγόριθμος λειτουργεί ακριβώς όπως περιγράψαμε σε προηγούμενη ενότητα.

2.1.12 Σχεδιασμός των διαδρομών ως σημεία του χάρτη

Σε αυτό το σημείο ο αλγόριθμος έχει εξάγει τις τιμές του μονοπατιού `route_x, route_y` τις οποίες θέλουμε να αναπαραστήσουμε στο χάρτη. Χρησιμοποιούμε έναν τροποποιημένο αλγόριθμο Bresenham ώστε να επιστρέφει κάθε έξι σημεία της ευθείας και όχι όλα.

```
for(k=0;k<route_x.length-1;k++){

var all = line_2(Math.round(route_x[k]),
Math.round(route_y[k]), Math.round(route_x[k+1]),
Math.round(route_y[k+1]));

var all2=Object.values(all);

for(q=0;q<all2.length;q++){

 var point1 = {

    x: all2[q][0],

    y: all2[q][1],

    value: 35,

  radius: 50
```

```
  };
  points.push(point1);
}
}
var data = {
  max:mx,
  data: points
};

heatmapInstance.setData(data);
```

Η μεταβλητή `all` δέχεται ως αντικείμενο τα ενδιάμεσα σημεία της ευθείας που σχηματίζεται απ' τις συντεταγμένες `route_x[k], route_y[k], route_x[k+1], route_y[k+1]`. Ο πίνακας `all2` περιέχει ως πίνακα τα δεδομένα τα οποία εισάγονται με το γνωστό τρόπο στο `heatmap instance`.Μετά απ' τις παραπάνω διαδικασίες έχουμε συγκεντρωτικά τα σημεία του χάρτη που αφορούν τα πραγματικά δεδομένα μας, δηλαδή τα σημεία πρόσβασης, μαζί με τα στατικά δεδομένα που αφορούν τις διαδρομές.

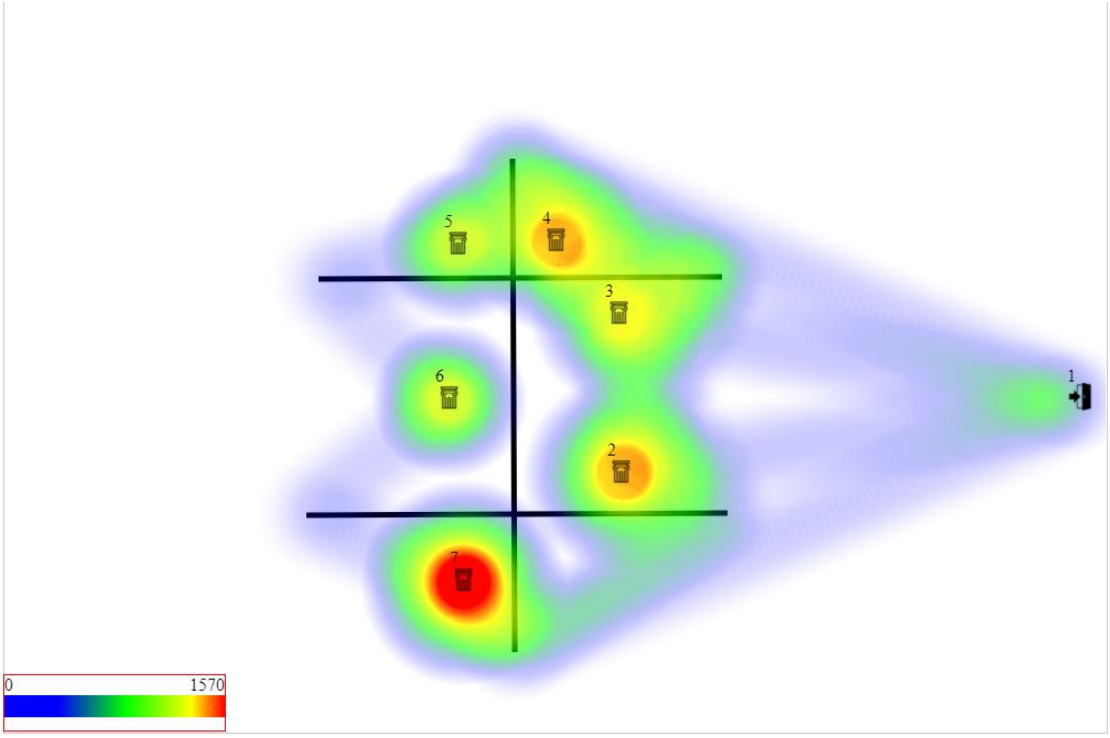


Figure 25 Θερμογράφημα μαζί με υπόμνημα

2.1.13 Δημοφιλέστερες διαδρομές

Σε αυτό το κομμάτι θα ασχοληθούμε με την εξαγωγή μοτίβων. Θα δουλέψουμε σύμφωνα με τη διαδικασία που προαναφέραμε στην ενότητα 'Δημοφιλέστερη Διαδρομή' του πρώτου κεφαλαίου. Κάθε πεδίο του πίνακα `routes` μας παρέχει πληροφορία για την αλληλουχία των κινήσεων ενός επισκέπτη. Σύμφωνα με τη μέθοδο που έχουμε περιγράψει χρειάζεται να μάθουμε πρώτα ποια είναι τα σημεία που επισκέφθηκαν πιο πολύ οι επισκέπτες. Θα μπορούσε κάποιος να σκεφτεί πως μια τέτοια διαδικασία είναι απλή: Ταξινομούμε τα σημεία σε φθίνουσα σειρά `times_visited`. Θα λειτουργούσε, όμως χάνουμε τη σημαντική πληροφορία της σειράς με την οποία γίνονται οι επισκέψεις. Για παράδειγμα μπορεί εύκολα να γνωρίζω πως το σημείο 2 είχε 15 επισκέψεις και το σημειο 3 είχε 20, όμως δε γνωρίζω τίποτα για τη σειρά επίσκεψης. Αν όμως γνωρίζω πως περισσότερες φορές πέρασαν οι επισκέπτες πρώτα απ' το σημείο 3 και έπειτα απ' το 2, δίνω μια διαφορετική διάσταση στην πληροφορία.

```
$conn = new mysqli("localhost","root","","db_heatmap");

$query = "select route from routes";

$r_info = mysqli_query($conn,$query);

$all_routes = array();
```

```
$most_freq = array();

while($r = mysqli_fetch_assoc($r_info)){

 $all_routes[] = str_split($r['route']);

}
```

Ο πίνακας δύο διαστάσεων `all_routes` περιέχει τα περιεχόμενα του πίνακα `routes`.

```
$query1 = "select * from settings";

$settings = mysqli_query($conn,$query1);

$row2 = mysqli_fetch_array($settings);

$max_hops = $row2['max_hops'];
```

Ανασύρεται η τιμή για τον μέγιστο αριθμό αναπηδήσεων του μονοπατιού απ' τον πίνακα `settings`.

```
for($i=0; $i<=$max_hops; $i++){

$col = array_column($all_routes, $i);

$n = count($col) / count($col[0]);

$temp = mostFrequent($col, $n);

array_push($most_freq,$temp);

}
```

Ο βρόχος διατρέχει τον πίνακα `all_routes`. Σύμφωνα με το σχήμα της πρώτης ενότητας, θέλουμε να ελέγχουμε κάθετα τον κάθε πίνακα μέσα στον `all_routes`. Η μέθοδος `array_collumn` παρέχει κάθετη διάσχιση στους επιμέρους πίνακες. Η

μεταβλητή n προσδίδει στην sort που χρησιμοποιείται εντός της mostFrequent το όρισμα για αριθμητική σύγκριση.

```
function mostFrequent( $arr, $n)
{

    sort($arr);
    sort($arr , $n);
```

Ταξινομούμε τον πίνακα πρώτα:

```
    $max_count = 1;
    $res = $arr[0];
    $curr_count = 1;
    for ($i = 1; $i < $n; $i++)
    {
        if ($arr[$i] == $arr[$i - 1])
            $curr_count++;
        else
        {
            if ($curr_count > $max_count)
            {
                $max_count = $curr_count;
                $res = $arr[$i - 1];
            }
            $curr_count = 1;
        }
    }


    if ($curr_count > $max_count)
    {
        $max_count = $curr_count;
        $res = $arr[$n - 1];
    }

    return $res;
}
```

Βρίσκουμε τη μέγιστη συχνότητα χρησιμοποιώντας γραμμική διάσχιση. Έχουμε ένα μέγιστο για κάθε υποπίνακα του all_routes. Η τελική τιμή του most_freq θα είναι ένας αριθμητικός πίνακας μεγέθους max_hops.

```
$conn2 = new mysqli("localhost","root","","db_heatmap");
```

```
for($i=0; $i<$max_hops; $i++){
      $query2 = "select * from access_point where
id='$most_freq[$i]'";
      $r_info2 = mysqli_query($conn2,$query2);
            while($r2 = mysqli_fetch_assoc($r_info2)){
                  array_push($id,$r2['id']);
                  array_push($x,$r2['x']);
                  array_push($y,$r2['y']);
                  array_push($heat,$r2['heat']);
            }
}
```

Σύμφωνα με τον αριθμό που υπάρχει στον `most_freq` κάθε φορά επιλέγουμε την κατάλληλη πληροφορία απ' τον πίνακα `access_point`. Σειρά έχει ο αλγόριθμος εύρεσης μονοπατιού με τον γνωστό τρόπο. Καταλήγουμε με δύο πίνακες `route_x`, `route_y` που περιγράφουν το μονοπάτι. Θα πρέπει να το απεικονίσουμε στην οθόνη.

```
var context = canvas.getContext('2d');
for(k=0;k<route_x.length-1;k++){
      context.beginPath();
      context.lineWidth = 1;
      context.strokeStyle = "#FF0000";

canvas_arrow(context,route_x[k],route_y[k],route_x[k+1],route_
y[k+1]);
      context.stroke();

}
```

Διατρέχοντας τον πίνακα `route_x` και `route_y` ορίζουμε την αρχή του μονοπατιού στην αναφορά context. Η `context` περιέχει τη σύνδεση του DOM στοιχείου `canvas` με τη μεταβλητή. Ορίζουμε πάχος 1 και κόκκινο χρώμα. Η μέθοδος `canvas_arrow` ζωγραφίζει ένα βέλος με αρχή τα `route_x[k], route_y[k]` και τέλος τα `route_x[k+1], route_y[k+1]`. Η μέθοδος `stroke` εμφανίζει στην οθόνη το επόμενο καρέ το οποίο περιέχει το βέλος. Κάθε επανάληψη δημιουργεί ένα βέλος.

```
function canvas_arrow(context, fromx, fromy, tox, toy) {

  var headlen = 10;

  var dx = tox - fromx;

  var dy = toy - fromy;
```

```
  var angle = Math.atan2(dy, dx);

  context.moveTo(fromx, fromy);

  context.lineTo(tox, toy);

  context.lineTo(tox - headlen * Math.cos(angle - Math.PI /
6), toy - headlen * Math.sin(angle - Math.PI / 6));

  context.moveTo(tox, toy);

  context.lineTo(tox - headlen * Math.cos(angle + Math.PI /
6), toy - headlen * Math.sin(angle + Math.PI / 6));

}
```

Η μέθοδος αναλαμβάνει να αποτυπώσει μια ευθεία που ορίζεται απ' τα σημεία `fromx, fromy, tox, toy`. Η μεταβλητή `headlen` προσδίδει μέγεθος στη μύτη του βέλους. Η μέθοδος `moveTo` δηλώνει τη θέση αρχής της ευθείας. Η μέθοδος `lineTo` δηλώνει τη θέση τέλους της ευθείας. Με τον ίδιο τρόπο δημιουργείται και το βέλος.

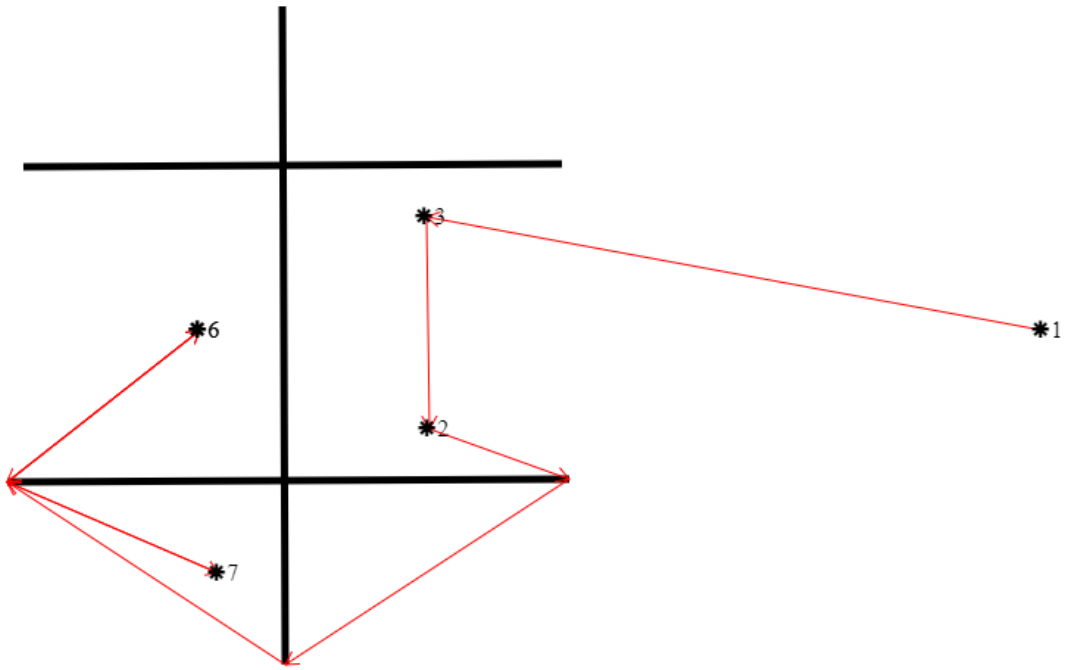


Figure 26 Παράδειγμα διαδρομής με βέλη

Επίσης, για να δείξουμε τη σειρά στα δεξιά :

```
var pop = document.getElementById('popRoute');

pop.innerHTML = 'Order: '+'<div>'+id+'</div>';
```

2.1.14 Δημοφιλέστερα σημεία πρόσβασης

Ας δούμε και διαφορετικές απεικονίσεις. Θα χρησιμοποιήσουμε τριών ειδών οπτικοποιήσεις. Η πρώτη θα είναι βοηθητική και είναι ένας απλός χάρτης με σημεία όπως το θερμογράφημα μονάχα που η κλίμακα θα είναι οι φορές που κάποιος έχει επισκεφθεί ένα σημείο, με τον αριθμό μάλιστα να αναγράφεται δίπλα στο εικονίδιό του εκάστοτε σημείου πρόσβασης. Οι διαδικασίες γίνονται με τους τρόπους που έχουν εξηγηθεί σε προηγούμενες ενότητες.

```
var config = {

  container: document.getElementById('heatmapContainer'),

  radius: 10,

  maxOpacity: max_opacity,

  minOpacity: min_opacity,

  blur: blur,

  gradient: {

    '1': color1,

    '.25': color2,

    '.5': color3

  },

  onExtremaChange: function(data) {

    updateLegend(data);

  }
```

```
};

var heatmapInstance = h337.create(config);
```

Το παραπάνω κομμάτι κώδικα δημιουργεί ένα `heatmap` αλλά υπάρχουν αλλαγές. Μέσω των ρυθμίσεων που θα εξετάσουμε παρακάτω, ο χρήστης μπορεί να επιλέξει τις ιδιότητες των σημείων του χάρτη. Αυτά είναι τρία διαφορετικά βασικά χρώματα της κλίμακας, η θολούρα και οι τιμές της διαφάνειας. Για να δημιουργήσουμε ένα διάγραμμα μπάρας χρησιμοποιούμε τη βιβλιοθήκη `Chart.js`. Αρχικά χρειαζόμαστε έναν τυχαίο επιλογέα χρωμάτων.

```
function getRandomColor() {

  var letters = '0123456789ABCDEF';

  var color = '#';

  for (var i = 0; i < 6; i++) {

    color += letters[Math.floor(Math.random() * 16)];

  }

  return color;

}
```

Η μέθοδος επιλέγει μια τυχαία συμβολοσειρά του δεκαεξαδικού και την επιστρέφει.

```
var ctx = document.getElementById('myChart').getContext('2d');
```

Δημιουργούμε μια αναφορά `ctx` στο αντικείμενο DOM `myChart`

```
<div style="float: right;margin-top: 15px;margin-right:
100px;">

   <canvas id="myChart" width="500" height="400"></canvas>
```

```
 </div>
```

Το στοιχείο `myChart` είναι τύπου canvas και θα φιλοξενήσει την απεικόνισή μας.

```
var clrPallete = [];

for (i=0; i<id.length; i++)

  clrPallete.push(getRandomColor());
```

Επιλέγουμε τα χρώματα.

```
var myChart = new Chart(ctx, {});
```

Δημιουργούμε ένα νέο αντικείμενο της κλάσης `Chart` με όρισμα το `ctx` που δημιουργήσαμε. Το δεύτερο όρισμα είναι τα ίδια τα δεδομένα και ο τρόπος που θα εμφανιστούν.

```
    type: 'horizontalBar',

    data: {

        labels: id,

        datasets: [{

            label: 'times visited',

            data: times_visited,

            backgroundColor: clrPallete,

            borderColor: 'Grey',

            borderWidth: 1

        }]
```

```
    }
```

Επιλέγουμε τον τύπο `horizontalBar`, οι ετικέτες `labels` θα προέρχονται απ' τα `id` των σημείων πρόσβασης. Το πεδίο `datasets` δέχεται έναν πίνακα αντικειμένων με διάφορες πληροφορίες όπως χρώμα που επιλέγεται απ' την `clrPallete` ενώ η πιο σημαντική είναι η `data` που λαμβάνει τους αριθμούς του `times_visited`. Επόμενο αντικείμενο του δεύτερου ορίσματος είναι το `options`.

```
options: {
      legend: {
           display: false },
       scales: {
           yAxes: [{
               ticks: {
                   beginAtZero: true
               },
               scaleLabel: {
                   display: true,
                   labelString: '# of access point'
               }
           }],
           xAxes: [{
               ticks: {
                   beginAtZero: true
               },
               scaleLabel: {
                   display: true,
```

```
                    labelString: '# of times visited'

                }

            }]

        },

         title: {

            display: true,

            text: 'Number of visits/Access point'

        }

    }
```

Σε ένα διάγραμμα μπάρας δεν έχει νόημα ένα υπόμνημα καθώς κάθε χρώμα αντιστοιχεί σε ένα `id` σημείου. Γι' αυτό το λόγο `legend :{display : false}`. Το πεδίο `scales` περιέχει πληροφορίες για το πώς διαμορφώνονται οι άξονες x, y. Διαλέγουμε αν θα φαίνονται και ποια θα είναι η ένδειξή τους. Για το διάγραμμα πίτας ισχύουν τα ίδια με κάποιες αλλαγές στο πεδίο `options`

```
var myChart2 = new Chart(ctx2, {

    type: 'pie',

    data: {

        labels: id,

        datasets: [{

            label: 'time aggregate',

            data: heat,        backgroundColor: clrPallete,
borderColor: 'Grey',           borderWidth: 1        }]   },
options: {       legend: {            display: true,
position: 'left' },        scales: {display: false      },
tooltips: {            mode: 'point'        },        title: {
display: true,            text: 'Aggregate time in (s)/Access
point'        }   }})
```

Δημιουργούμε νέο αντικείμενο με τύπο `pie`. Η τιμή `data` αντιστοιχεί στο `heat` αυτή τη φορά. Το υπόμνημα επίσης είναι κάτι που χρειάζεται καθώς δεν υπάρχουν άξονες.

2.1.15 Ρυθμίσεις

Υπάρχει μια απλή φόρμα με τα εξής πεδία:

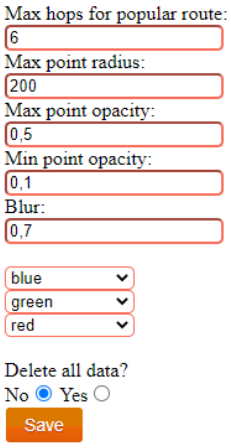


Figure 27 Τα πεδία της SETTINGS

`Max hops for popular route`, `Max point radius`, `Max point opacity`, `Min point opacity`, `Blur`, `Color1`, `Color2`, `Color3`, `Delete all data`. Οι προηγούμενες τιμές είναι προσυμπληρωμένες ώστε ο χρήστης να κάνει τις αλλαγές του χωρίς να χρειάζεται να θυμάται τις υπόλοιπες τιμές.

```
$('form').on('submit', function (e) {

          e.preventDefault();

          $.ajax({

            type: 'post',

            url: 'save_settings.php',

            data: $('form').serializeArray(),

            success: function () {

                let alerttext = "Changes saved.\n"
```

```
if($('form').serializeArray()[8] .value=='delete')
alerttext+="Data deleted.";

                alert(alerttext);

            }

          });

        });
```

Το παραπάνω κομμάτι κώδικα εκτελείται με το που πατηθεί το κουμπί save. Η `preventDefault` σταματάει τη φόρμα απ' το να καλέσει ένα αρχείο και να στείλει τα δεδομένα καθώς η jQuery θα το κάνει αυτό για μας. Αρχικά καλούμε το `save_settings.php`

```
$conn = new mysqli("localhost","root","","db_heatmap");

$query = "update settings set max_hops =
'$max_hops',max_radius = '$max_radius',max_opacity =
'$max_opacity',min_opacity = '$min_opacity',blur
='$blur',color1 = '$color1',color2 = '$color2',color3 =
'$color3'";

$ap_info = mysqli_query($conn,$query);
```

Ενημερώνουμε τις τιμές στον πίνακα `settings`.

```
if(($_POST['Yes'])=='delete'){

     $conn = new mysqli("localhost","root","","db_heatmap");




     $query = "delete  from access_point";

     mysqli_query($conn,$query);

     $query = "alter table access_point AUTO_INCREMENT = 1";
```

```
      mysqli_query($conn,$query);

      $query = "delete from routes";

      mysqli_query($conn,$query);

      $query = "delete from obstacles";

      mysqli_query($conn,$query);

      $query = "delete from past_instances";

      mysqli_query($conn,$query);

      $query = "update settings SET adj_mat = '' WHERE id=1";

      mysqli_query($conn,$query);
```

Αν ο χρήστης έχει πατήσει να διαγραφούν οι παλιές προσομοιώσεις δηλαδή η τιμή value του στοιχείου `<input type="radio" id="Yes" name="Yes" value="delete">` λάβει τιμή, τότε διαγράφονται όλες οι τιμές όλων των πινάκων της βάσης. Επίσης η τιμή `AUTO_INCREMENT` του βασικού πίνακα `access_point` πρέπει να επανέλθει σε 1 ώστε να δεχτεί τα νέα σημεία πρόσβασης. Το πεδίο `data` του AJAX jQuery αιτήματος παραπάνω περιέχει `$('form').serializeArray()`. Αυτό σημαίνει πως τα δεδομένα θα επιστρέψουν ως πίνακας. Στην όγδοη θέση θα είναι πάντα η τιμή `value` του radio `inupt` με `id = 'Yes'`.

```
  success: function () {

              let alerttext = "Changes saved.\n"

              if($('form').

serializeArray()[8].value=='delete')
      alerttext+="Data deleted.";

              alert(alerttext);

   }
```

Το μήνυμα που εμφανίζεται ως απάντηση στον χρήστη διαφέρει ανάλογα με το αν ο χρήστης επιλέξει να διαγράψει τα προηγούμενα δεδομένα. Αν η θέση 8 του πίνακα περιέχει ‘delete’ τότε έγινε διαγραφή άρα στο μήνυμα προσάπτεται ‘Data deleted’.

Κεφάλαιο 3

3.1 Οδηγίες εγκατάστασης/εκτέλεσης

Αρχικά κατεβάζουμε και εγκαθιστούμε το περιβάλλον XAMPP https://www.apachefriends.org/download.html. Το περιβάλλον ανάπτυξης είναι Windows 10. Το XAMPP εγκαθιστά τη διανομή Apache που περιέχει MariaDB, PHP, Perl. Αφού εγκαταστήσουμε το XAMPP ελέγχουμε πως η PHP έχει εγκατασταθεί σωστά με την εντολή command line: > `php -v`. Το αποτέλεσμα θα πρέπει να μοιάζει με την παρακάτω εικόνα:

```
C:\Users\AggelosPC>php -v
PHP 7.3.5 (cli) (built: May  1 2019 13:17:17) ( ZTS MSVC15 (Visual C++ 2017) x64 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.5, Copyright (c) 1998-2018 Zend Technologies
```

Figure 28 Έλεγχος έκδοσης php μέσω Windows terminal

Στη συνέχεια στη διαδρομή `C:\xampp\htdocs` ή όπου έχει εγκατασταθεί το πακέτο XAMPP, δημιουργούμε τον κατάλογο `heatmap_test` που θα φιλοξενήσει την εφαρμογή μας. Απ' τη διεύθυνση https://github.com/pa7/heatmap.js κατεβάζουμε τον πηγαίο κώδικα της βιβλιοθήκης `heatmap.js` και αποσυμπιέζουμε το αρχείο στον κατάλογο που μόλις δημιουργήσαμε. Ο κατάλογος που μας ενδιαφέρει απ' τα αρχεία που εμφανίστηκαν είναι ο `build` ο οποίος περιέχει τον κώδικα της βιβλιοθήκης. Το αρχείο `heatmap.js` είναι αυτό που θα εισάγουμε στην εφαρμογή ως εξής : `<script src="./build/heatmap.js"> </script>`. Οι υπόλοιπες βιβλιοθήκες χρησιμοποιούνται μέσω του CDN τους. Δημιουργούμε τα υπόλοιπα αρχεία της εφαρμογής στον κατάλογο `heatmap_test`. Η εικόνα του project θα πρέπει να είναι κάπως έτσι:

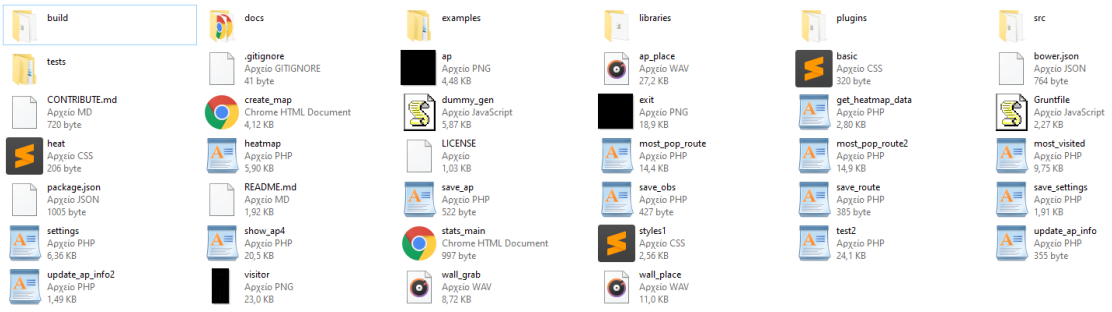


Figure 29 Η δομή του project

Ξεκινούμε τις διαδικασίες του διακομιστή και του SQL server πατώντας ‘START’:

Figure 30 Οι βασικές υπηρεσίες που ξεκινούμε στο περιβάλλον XAMPP

Στη διεύθυνση http://localhost/phpmyadmin/ βρίσκεται ο πίνακας ελέγχου της βάσης δεδομένων. Μπορούμε να δημιουργήσουμε μια νέα βάση δεδομένων πατώντας ‘Νέα’ απ’ το μενού πάνω αριστερά ή να εισάγουμε απ’ το μενού ‘Εισαγωγή’ το `.sql` αρχείο για να δημιουργήσουμε τους πίνακες. Την εφαρμογή ξεκινά το αρχείο http://localhost/heatmap test/heatmap.php

3.2 Διαδικασία τρεξίματος

Ας δούμε απ’ την αρχή τα βήματα που ακολουθούνται απ’ την παραμετροποίηση ως το τρέξιμο μια προσομοίωσης και τα αποτελέσματα.

CREATE MAP

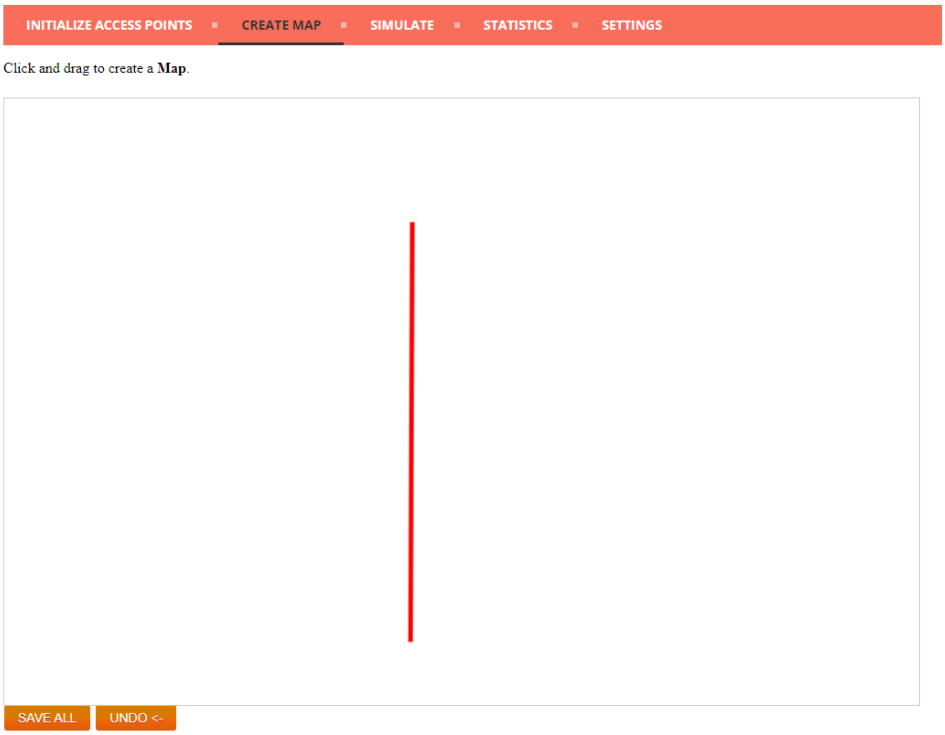


Figure 31 Η αρχική οθόνη της εφαρμογής - Δημιουργία χάρτη

Στην εικόνα βλέπουμε την οθόνη δημιουργίας χάρτη. Πατάμε και κρατάμε το αριστερό κλικ στην τοποθεσία που επιθυμούμε να ξεκινά ένα εμπόδιο. Σύρουμε το ποντίκι στην επιθυμητή θέση τερματισμού του εμποδίου και αφήνουμε το αριστερό κλικ. Σε περίπτωση που κάναμε λάθος ή μετανιώσαμε για μία επιλογή, υπάρχει το κουμπί 'UNDO' που αναιρεί την τελευταία τοποθέτηση.

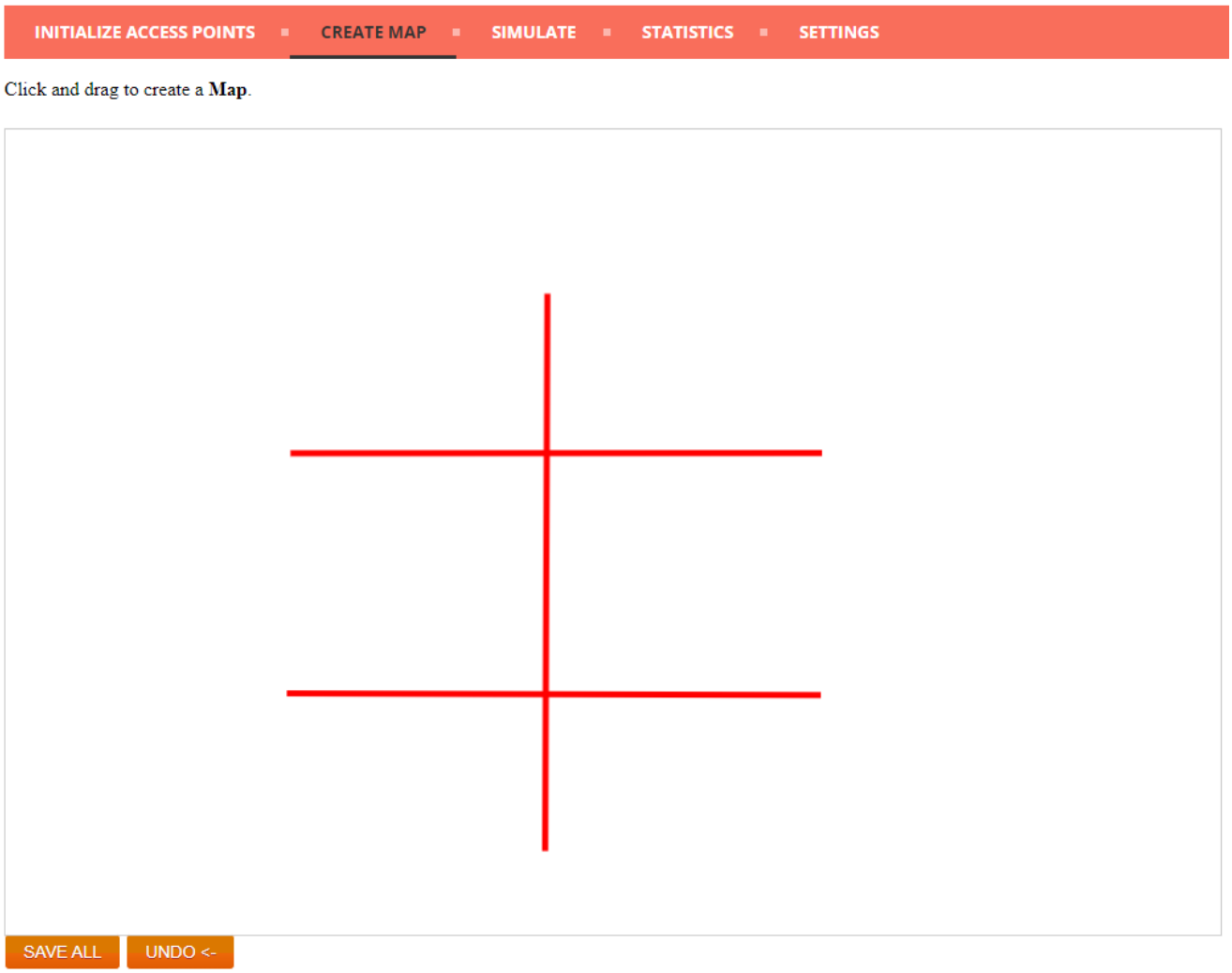


Figure 32 Δημιουργία χάρτη

Μόλις έχουμε επιλέξει τα εμπόδια κάνουμε κλικ στο κουμπί 'SAVE ALL' για να αποθηκευτεί ο χάρτης.

INITIALIZE ACCESS POINTS

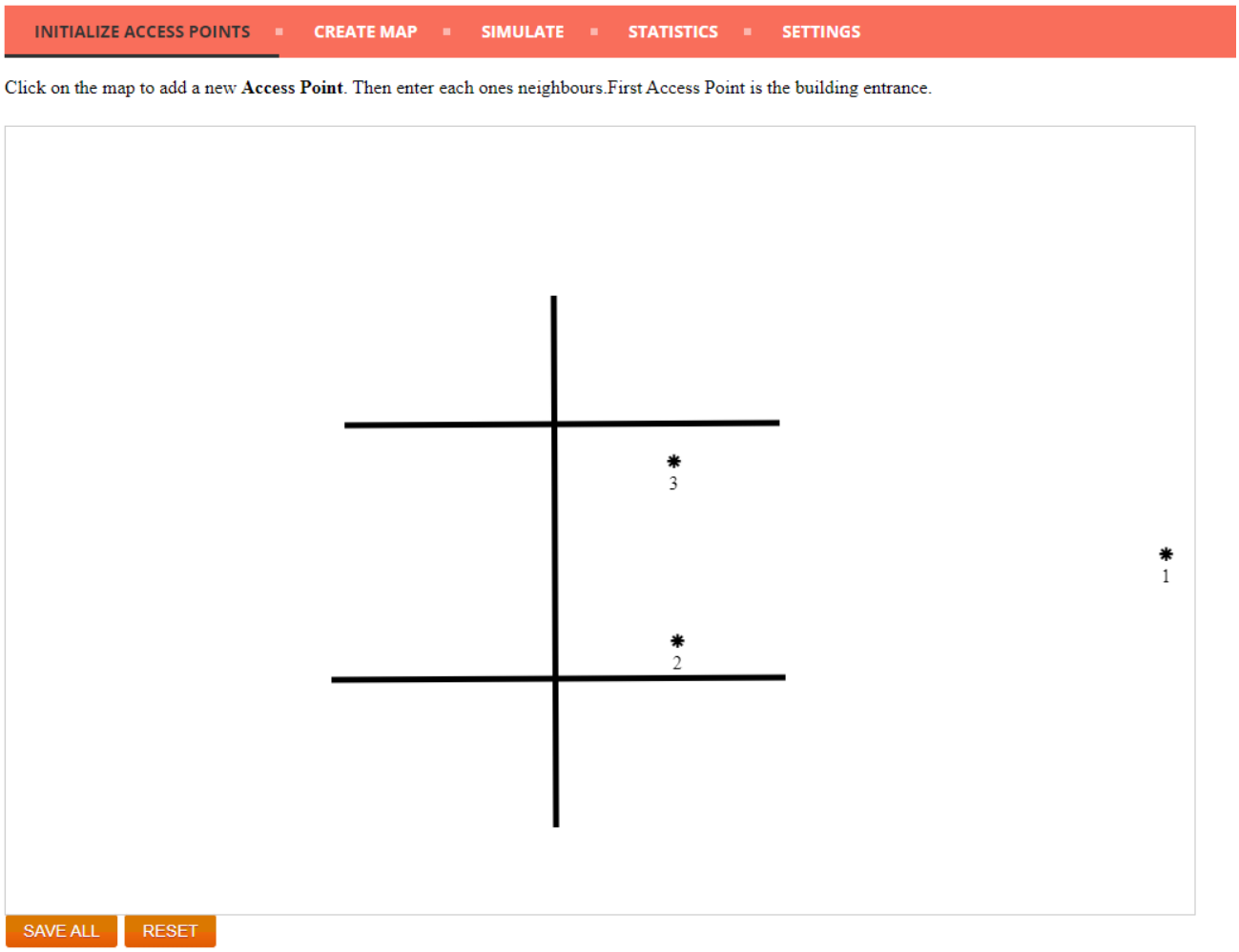


Figure 33 Η τοποθέτηση των σημείων πρόσβασης στο χάρτη

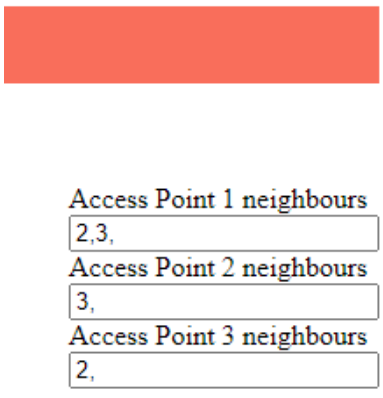


Figure 34 Συμπλήρωση γειτόνων για κάθε σημείο πρόσβασης που προστίθεται στο χάρτη

Στην οθόνη INITIALIZE ACCESS POINTS εισάγουμε με αριστερό κλικ ένα σημείο ενδιαφέροντος στο χάρτη όπως φαίνεται στο Figure 33. Για κάθε σημείο που εισάγουμε εμφανίζεται στα δεξιά της οθόνης ένα πεδίο στο οποίο συμπληρώνουμε τους γείτονές του όπως στο Figure 34. Σε αυτό το πεδίο, το κόμμα εισάγεται αυτόματα ύστερα από κάθε πάτημα αριθμού. Σε περίπτωση που επιθυμούμε να διαγράψουμε τους γείτονες από ένα πεδίο, κρατάμε πατημένο το πλήκτρο BACKSPACE. Το κουμπί 'RESET' επαναφέρει το χάρτη στην κενή κατάσταση διαγράφοντας παράλληλα όλα τα δεδομένα που έχουμε εισάγει. Όταν είμαστε έτοιμοι πατάμε το κουμπί 'SAVE ALL'.

SIMULATE

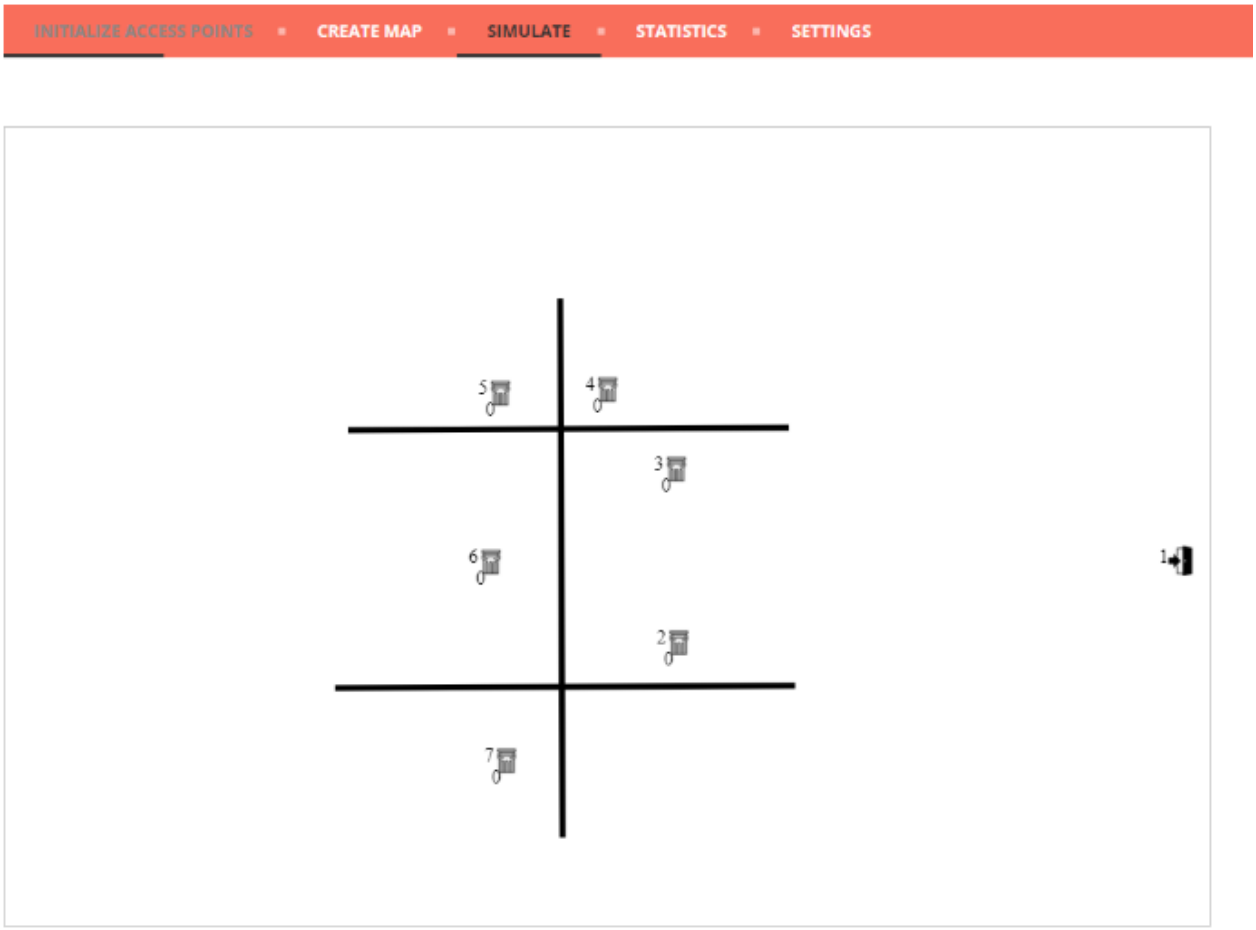


Figure 35 Η οθόνη προσομοίωσης πριν την έναρξη

Στην οθόνη SIMULATE βλέπουμε αριστερά την οθόνη τρεξίματος με τα δεδομένα που έχουμε μέχρι στιγμής εισάγει. Για κάθε σημείο ενδιαφέροντος ο αριθμός που βρίσκεται πάνω αριστερά αντιπροσωπεύει το μοναδικό αναγνωριστικό του (id) ενώ ο αριθμός που βρίσκεται κάτω αριστερά , τον αριθμό των επισκεπτών που βρίσκονται ταυτόχρονα σε αυτό.

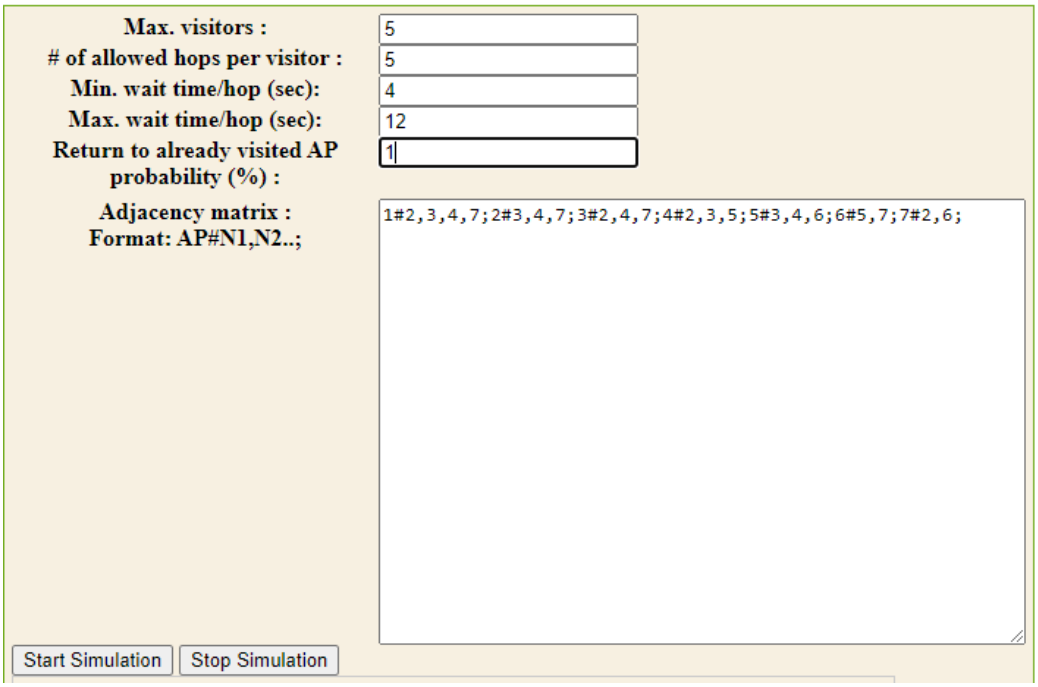


Figure 36 Παραμετροποίηση Προσομοίωσης

Στα δεξιά υπάρχει το μενού παραμετροποίησης της προσομοίωσης όπως φαίνεται στο Figure 36. Τα δεδομένα του πίνακα γειτνίασης εισάγονται αυτόματα. Εμείς επιλέγουμε τον αριθμό των επισκεπτών, τα βήματα που θα εκτελέσουν, το μέγιστο και ελάχιστο χρόνο παραμονής σε ένα σημείο και την πιθανότητα να επιστρέψει κάποιος σε σημείο που έχει ήδη επισκεφθεί. Το κουμπί 'Start Simulation' ξεκινά την προσομοίωση ενώ το 'Stop Simulation' την τερματίζει. Τα δεδομένα ωστόσο αποθηκεύονται κανονικά. Το μόνο που τερματίζεται είναι οι κινήσεις των επισκεπτών. Όταν τελειώσει η πρώτη προσομοίωση μπορούμε να περιηγηθούμε περεταίρω στην εφαρμογή, αποκτώντας πρόσβαση στις υπόλοιπες σελίδες.

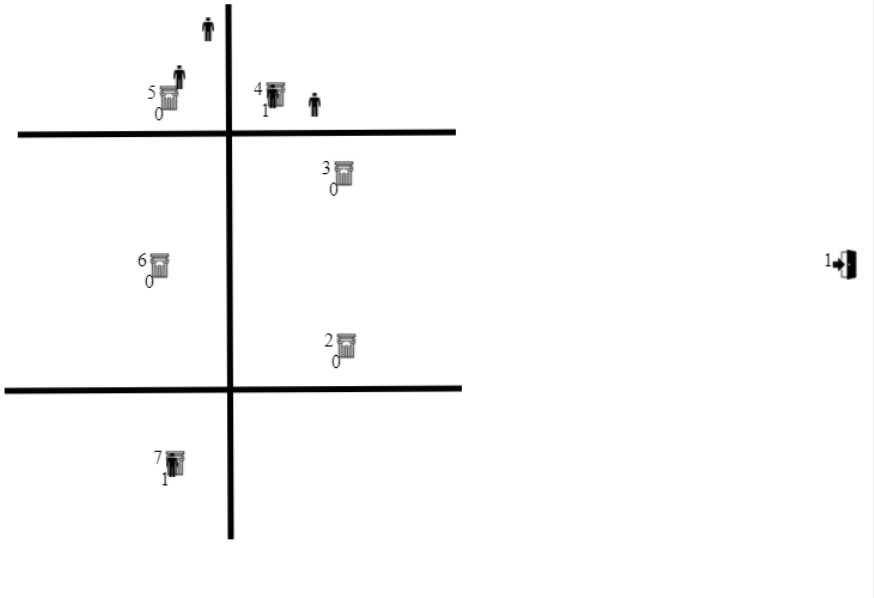


Figure 37 Η προσομοίωση εν δράσει

Στην οθόνη εκτέλεσης εμφανίζονται οι επισκέπτες να κινούνται στο χώρο. Κάθε 5 δευτερόλεπτα εισάγεται και ένας νέος επισκέπτης πάντοτε απ' την είσοδο (σημείο πρόσβασης 1). Ο κάθε επισκέπτης απεικονίζεται με ένα ανθρωπάκι και εμφανίζεται στην είσοδο. Εκτελεί προδιαγεγραμμένη πορεία και στο τέλος της κίνησής του εξαφανίζεται αφού δει και το τελευταίο σημείο πρόσβασης. Τα σημεία πρόσβασης που συμβολίζονται εδώ με τον κίονα, έχουν τον αριθμό τους πάνω αριστερά, ενώ ο κάτω αριστερά αριθμός αντιπροσωπεύει τους ταυτόχρονους επισκέπτες που εντοπίζονται εκεί.

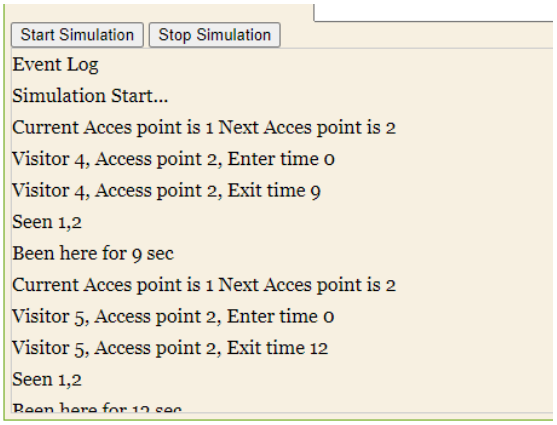


Figure 38 Πληροφορίες για τις κινήσεις της προσομοίωσης

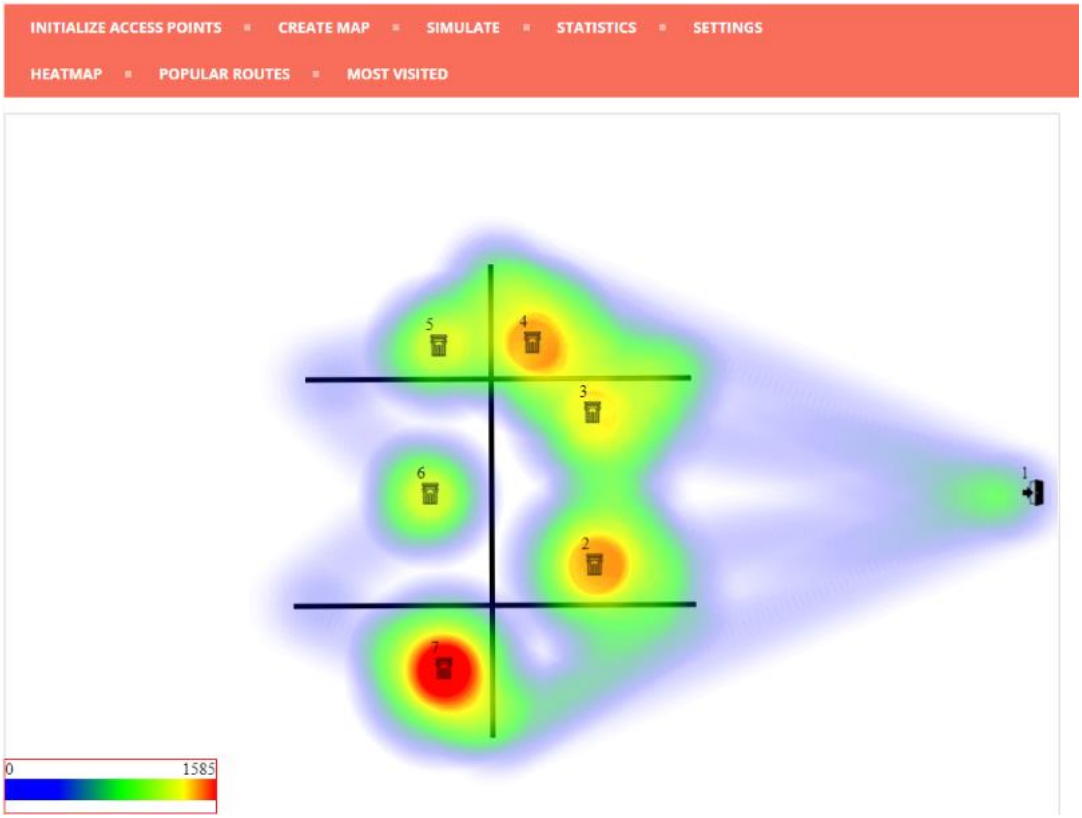


Figure 39 Θερμογράφημα με δεδομένα προσομοιώσεων

Κάτω απ' το μενού STATISTICS->HEATMAP βρίσκεται το θερμογράφημα. Εξ' ορισμού βλέπουμε την κατηγορία Aggregate όμως μπορούμε να αλλάξουμε την προβολή σε παλαιότερη κατάσταση. Κάτω αριστερά βλέπουμε το υπόμνημα που αναπαριστά τη χρωματική κλίμακα και ελάχιστη και μέγιστη τιμή. Οι τιμές που τείνουν προς το κυανό είναι πολύ χαμηλές και άρα αρκετά «κρύες» στη θερμοκρασία ενώ οι τιμές που τείνουν στο ερυθρό είναι «ζεστές». Κάθε φορά που μια νέα προσομοίωση λαμβάνει χώρα, τα δεδομένα που απορρέουν, εντάσσονται στην κατηγορία Aggregate.

Figure 40 Μια παλαιότερη κατάσταση θερμογραφήματος

Figure 41 Επιλογή παλαιότερης κατάστασης

Το μενού στα δεξιά, που φαίνεται στο Figure 41 μας επιτρέπει να προβάλουμε ένα ιστορικό προσομοιώσεων. Παρατηρούμε πως αλλάζοντας τις παλαιότερες καταστάσεις, αλλάζει ο χρωματισμός του χάρτη. Αυτό συμβαίνει γιατί πάντα οι τιμές που προβάλλονται είναι αναλογικές της μέγιστης τιμής. Στο Figure 40 παρατηρούμε πως οι γωνίες των τοίχων ανάμεσα στα σημεία πρόσβασης 5,6,7 έχουν εντονότερο πρασινωπό χρώμα σε σχέση με το αντίστοιχο του Figure 39. Αυτό δε σημαίνει πως μια παλαιότερη κατάσταση είχε μεγαλύτερη τιμή σε εκείνο το σημείο. Αντιθέτως σημαίνει πως τιμή εκείνη, ήταν μεγαλύτερη σε σχέση με το μέγιστο.

STATISTICS->MOST VISITED

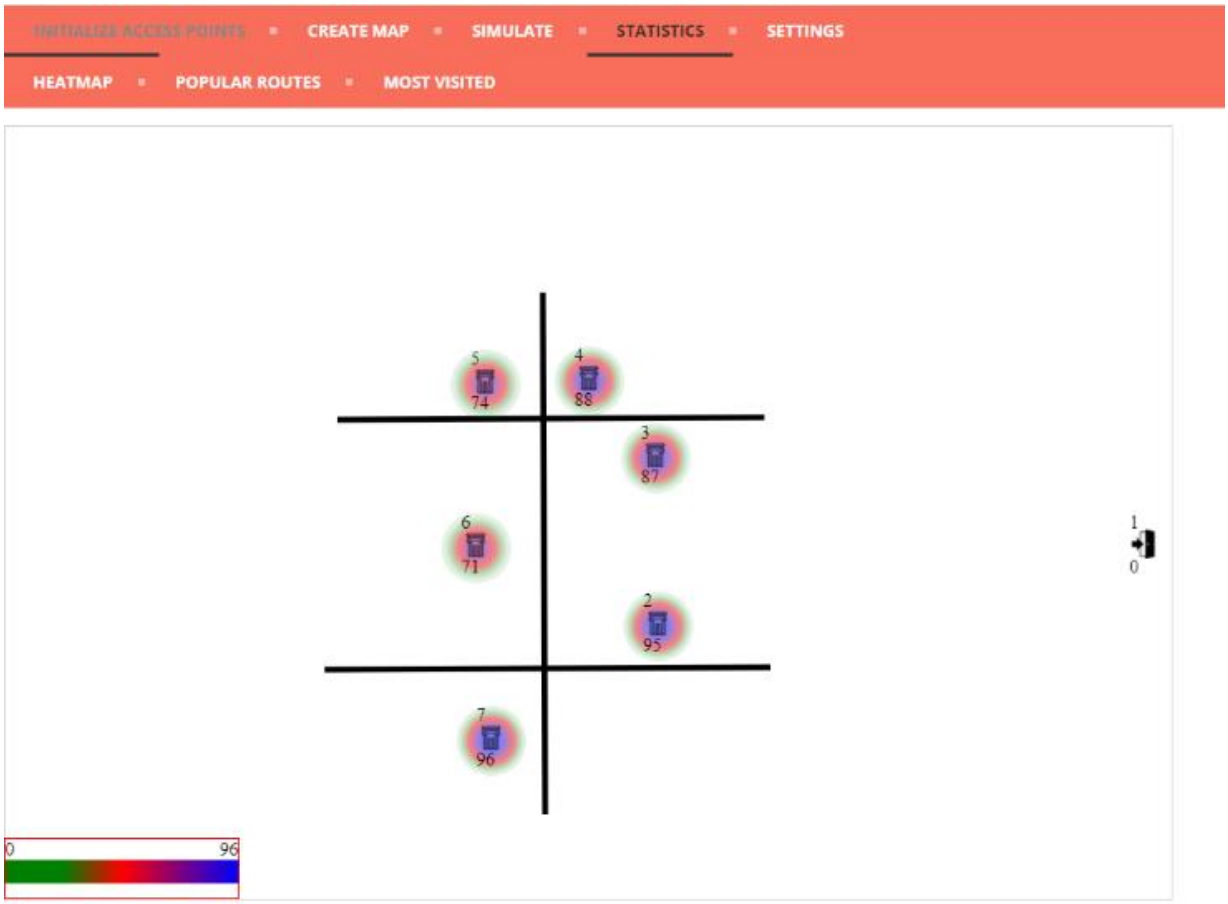


Figure 42 Βοηθητικός χάρτης σημείων πρόσβασης

Κάτω απ' το μενού STATISTICS->MOST VISITED θα βρούμε την καρτέλα εμφάνισης διαγραμμάτων και στατιστικών. Στα αριστερά εμφανίζεται ο χάρτης με τα σημεία πρόσβασης. Η κλίμακα είναι διαφορετική και ρυθμιζόμενη απ' το μενού 'SETTINGS'. Η κλίμακα αντιπροσωπεύει όμως τον αριθμό των επισκέψεων. Ο χάρτης έχει επικουρικό χαρακτήρα στα σχήματα δεξιά. Επίσης τα νούμερα κάτω αριστερά απ' τους κίονες αναπαριστούν τις συνολικές φορές επίσκεψης απ' όλες τις προσομοιώσεις.

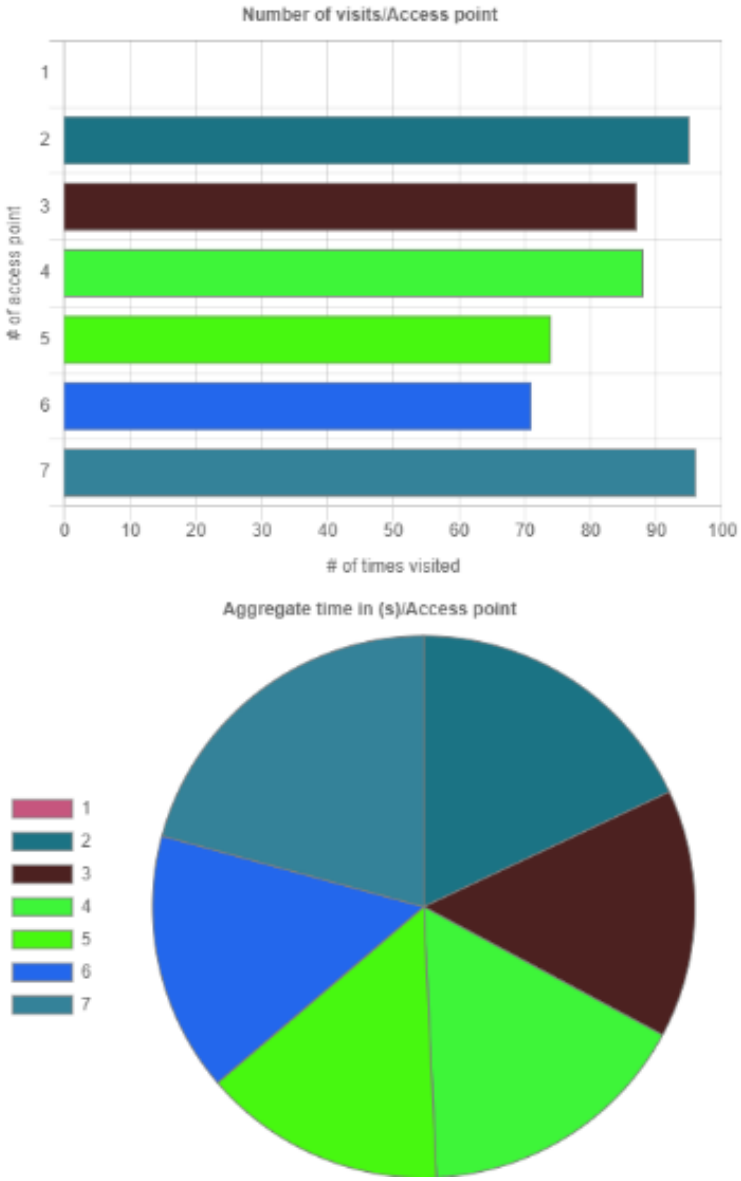


Figure 43 Επάνω- Διάγραμμα μπάρας. Κάτω- Διάγραμμα πίτας

Επάνω δεξιά εμφανίζεται το διάγραμμα μπάρας και ακριβώς από κάτω το διάγραμμα πίτας. Το κάθε σημείο πρόσβασης εμφανίζεται αριστερά με διαφορετικό χρώμα. Η κάθε μπάρα εκτείνεται στην κλίμακα των φορών επίσκεψης. Επίσης τα δεδομένα εμφανίζονται και με hover πάνω απ' τη μπάρα. Το διάγραμμα πίτας εμφανίζει τα δεδομένα για το συνολικό χρόνο επίσκεψης σε κάθε σημείο πρόσβασης που αναπαρίσταται με διαφορετικό χρώμα. Πάλι οι πληροφορίες εμφανίζονται και με hover πάνω απ' το κάθε κομμάτι της πίτας.

STATISTICS->POPULAR ROUTES

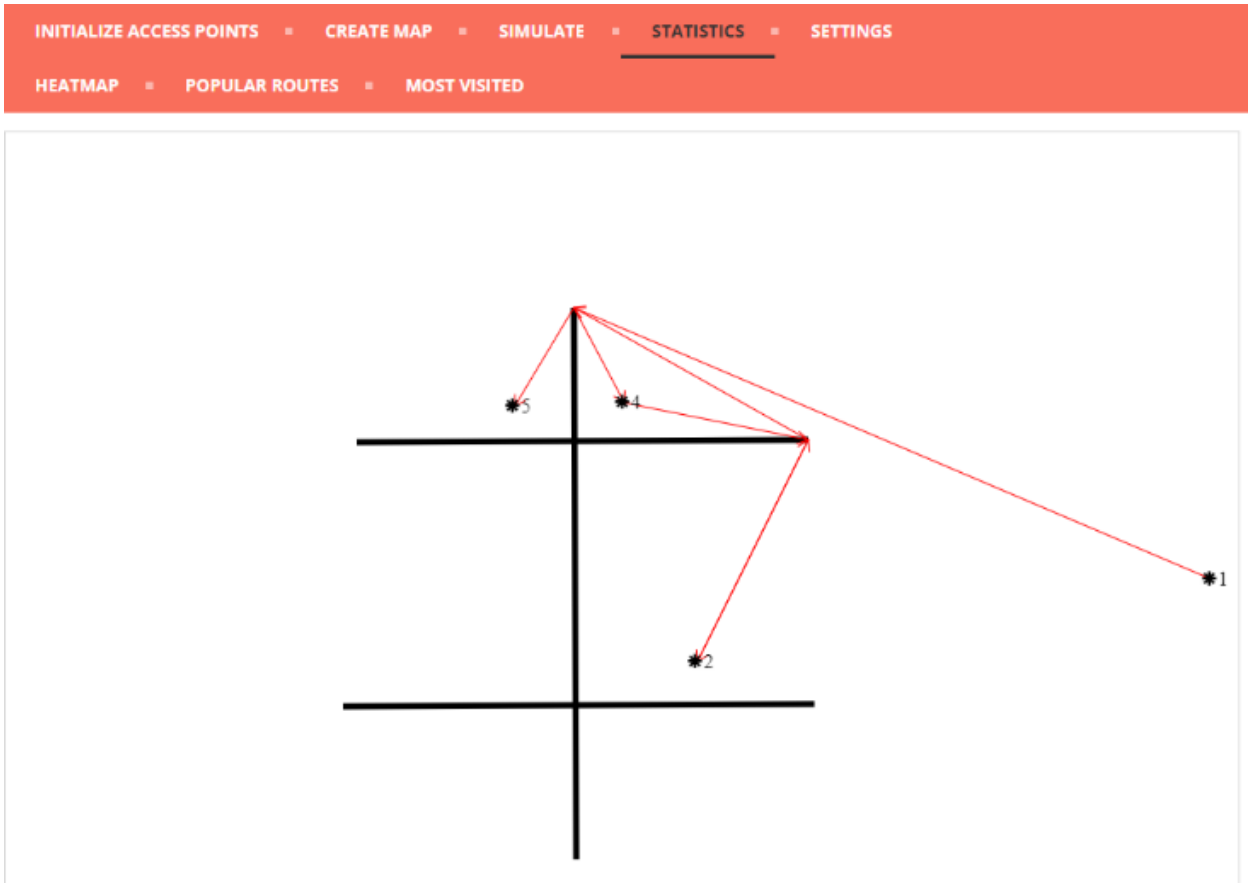


Figure 44 Αποτύπωση συχνότερης διαδρομής

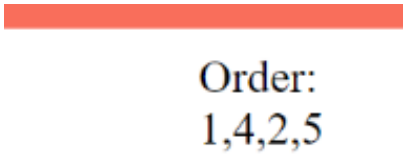


Figure 45 Αποτύπωση συχνότερης διαδρομής με ακολουθία αριθμών

Κάτω απ' το μενού STATISTICS->POPULAR ROUTES βρίσκεται η απεικόνιση της συχνότερης διαδρομής. Απ' το μενού 'SETTINGS' και έπειτα 'Max hops for popular route' ορίζουμε το πόσα άλματα θα βλέπουμε. Στο παράδειγμα βλέπουμε τη συχνότερη διαδρομή που αποτελείται από κόκκινα βέλη που αναπαριστούν το πιο κοινό μονοπάτι για όλες τις προσομοιώσεις μέχρι στιγμής. Η ακολουθία του Figure 45 δείχνει τη σειρά των πιο συχνών σημείων πρόσβασης.

SETTINGS

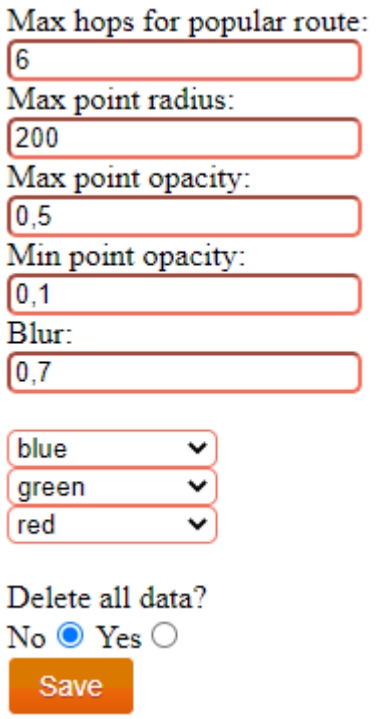


Figure 46 Επιλογές στο μενού SETTINGS

Κάτω απ' το μενού SETTINGS υπάρχουν διάφορες επιλογές όπως το προαναφερθέν Max hops for popular route. Οι υπόλοιπες επιλογές είναι στυλιστικού χαρακτήρα και αλλάζουν μονάχα τον χάρτη της σελίδας MOST POPULAR ROUTES. Τέλος, ο χρήστης έχει τη δυνατότητα να διαγράψει όλα τα δεδομένα (σημεία πρόσβασης ,εμπόδια , δεδομένα προσομοιώσεων) επιλέγοντας 'Yes' στο πεδίο Delete all data. Μετά τη διαγραφή, ο χρήστης είναι ελεύθερος να ξαναδώσει νέα δεδομένα.

3.3 Η βάση δεδομένων

Ακολουθούν οι πίνακες της βάσης δεδομένων καθώς και στιγμιότυπα με δεδομένα αυτών.

	Πίνακας	Ενέργεια	Εγ
☐	access_point	☆ Περιήγηση Δομή Αναζήτηση Προσθήκη Άδειασμα Διαγραφή	
☐	obstacles	☆ Περιήγηση Δομή Αναζήτηση Προσθήκη Άδειασμα Διαγραφή	
☐	past_instances	☆ Περιήγηση Δομή Αναζήτηση Προσθήκη Άδειασμα Διαγραφή	
☐	routes	☆ Περιήγηση Δομή Αναζήτηση Προσθήκη Άδειασμα Διαγραφή	
☐	settings	☆ Περιήγηση Δομή Αναζήτηση Προσθήκη Άδειασμα Διαγραφή	
	5 πίνακες	Σύνολο	

Figure 47 Τα ονόματα των πινάκων

		id ▲ 1	x	y	times_visited	heat
☐	Επεξεργασία Αντιγραφή Διαγραφή	1	362	978	0	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	2	429	561	95	1375
☐	Επεξεργασία Αντιγραφή Διαγραφή	3	285	559	87	1130
☐	Επεξεργασία Αντιγραφή Διαγραφή	4	219	502	88	1251
☐	Επεξεργασία Αντιγραφή Διαγραφή	5	222	413	74	1106
☐	Επεξεργασία Αντιγραφή Διαγραφή	6	362	405	71	1180
☐	Επεξεργασία Αντιγραφή Διαγραφή	7	527	418	96	1585

Figure 48 access_point

		id	x1	y1	x2	y2
☐	Επεξεργασία Αντιγραφή Διαγραφή	118	461	142	463	589
☐	Επεξεργασία Αντιγραφή Διαγραφή	119	285	251	651	249
☐	Επεξεργασία Αντιγραφή Διαγραφή	120	274	465	656	463

Figure 49 obstacles

		id_	id	times_visited	heat	date	state_id
☐	Επεξεργασία Αντιγραφή Διαγραφή	261	4	21	379	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	262	6	19	414	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	263	5	17	286	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	264	3	18	374	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	265	1	0	0	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	266	7	24	498	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	267	2	21	423	07/12/2020 20:25 PM	0
☐	Επεξεργασία Αντιγραφή Διαγραφή	268	1	0	0	07/12/2020 20:27 PM	1
☐	Επεξεργασία Αντιγραφή Διαγραφή	269	2	10	150	07/12/2020 20:27 PM	1
☐	Επεξεργασία Αντιγραφή Διαγραφή	270	7	9	166	07/12/2020 20:27 PM	1
☐	Επεξεργασία Αντιγραφή Διαγραφή	271	4	9	157	07/12/2020 20:27 PM	1

Click the drop-down arrow to toggle column's visibility.

Figure 50 past_instances

		id	route
☐	Επεξεργασία Αντιγραφή Διαγραφή	1108	123765423
☐	Επεξεργασία Αντιγραφή Διαγραφή	1109	176542343
☐	Επεξεργασία Αντιγραφή Διαγραφή	1110	176542342
☐	Επεξεργασία Αντιγραφή Διαγραφή	1111	142376567
☐	Επεξεργασία Αντιγραφή Διαγραφή	1112	124376545
☐	Επεξεργασία Αντιγραφή Διαγραφή	1113	134567272
☐	Επεξεργασία Αντιγραφή Διαγραφή	1114	134276543
☐	Επεξεργασία Αντιγραφή Διαγραφή	1115	145672376
☐	Επεξεργασία Αντιγραφή Διαγραφή	1116	145376723
☐	Επεξεργασία Αντιγραφή Διαγραφή	1117	142765342
☐	Επεξεργασία Αντιγραφή Διαγραφή	1118	127653454
☐	Επεξεργασία Αντιγραφή Διαγραφή	1119	172345672
☐	Επεξεργασία Αντιγραφή Διαγραφή	1120	134567276

Figure 51 routes

id	max_hops	max_radius	max_opacity	min_opacity	blur	color1	color2	color3	adj_mat	current_state
1	4	200	0.5	0.1	0.7	blue	green	red	1#2,3,4,7;2#3,4,7;3#2,4,7;4#2,3,5;5#3,4,6;6#5,7;7#...	14

Figure 52 settings

Κεφάλαιο 4

4.1 Συμπεράσματα

Οι πληροφορίες που λαμβάνουμε απ' τις προσομοιώσεις παρουσιάζουν σημαντικό ενδιαφέρον. Αρχικά κοιτάζοντας ένα θερμογράφημα κατανοούμε εποπτικά γύρω από ποια σημεία κινήθηκαν περισσότερο οι επισκέπτες. Τα αποτελέσματα αυτά, όταν συνδυαστούν και με άλλα χαρακτηριστικά των επισκεπτών όπως είναι η ηλικία, το φύλο ή η εθνικότητα μπορούν να δώσουν χρήσιμα συμπεράσματα τόσο για τα ενδιαφέροντά τους, όσο και για την ίδια τη σημαντικότητα των εκθεμάτων. Μπορούν σύμφωνα με τις παρατηρήσεις να πραγματοποιηθούν αλλαγές στις θέσεις, ώστε να είναι όσο το δυνατόν ομαδοποιημένα ενώ άλλα μπορούν να αντικατασταθούν πλήρως. Ακόμη, η σειρά τους μπορεί να αλλάξει με σκοπό τη διατήρηση αμείωτου ενδιαφέροντος του επισκέπτη κατά τη διάρκεια της επίσκεψης.

Τα δεδομένα της προσομοίωσης είναι εικονικά γεγονός που υπονοεί ότι κάποιες παράμετροι θα λείπουν. Μια τέτοια θα μπορούσε να είναι η τυχαιότητα της κίνησης και σε χώρους εκτός των σημείων ενδιαφέροντος. Μια ενδιαφέρουσα προσθήκη θα ήταν η δημιουργία κατηγοριών σημείων ενδιαφέροντος. Κάποια θα αναπαριστούν εκθέματα ενώ άλλα θα χρησιμεύουν απλά για την καταγραφή της κίνησης σε ένα ορισμένο χώρο. Ως παράδειγμα εναλλακτικής χρήσης σημείου ενδιαφέροντος θα μπορούσε να αναφερθεί ο χώρος εστιατορίου του μουσείου, η ένας χώρος αγοράς αναμνηστικών. Η πολυπλοκότητα βέβαια θα αυξανόταν, αλλά θα υπήρχε ολοκληρωμένη εικόνα. Όπως και να έχει, το ολοκληρωμένο λογισμικό ακόμη και με τις βασικές λειτουργίες, αποτελεί εργαλείο αύξησης της αποδοτικότητας και της αποτελεσματικότητας ενός οργανισμού ή επιχείρησης.

Βιβλιογραφία

[1]Michael Abrash. 1997. Michael Abrash's Graphics Programming Black Book, with CD: The Complete Works of Graphics Master, Michael Abrash (10th. ed.). Coriolis Group Books, USA.

[2]Larry Ullman. 2012 PHP and MySQL for Dynamic Web Sites, Fourth Edition: Visual QuickPro Guide, PeachPit Press

[3]Kyle Simpson ES6 & Beyond Sebastopol, CA : O'Reilly Media, 2016.

Jonathan Chaffer, Karl Swedberg, Learing jQuery – Fourth Edition . Packt Publishing Ltd. 2013

[4]Gehlenborg Nils, Wong, Bang. Heat maps, 2012

https://www.patrick-wied.at/static/heatmapjs/docs.html

https://www.patrick-wied.at/static/heatmapjs/

https://github.com/pa7/heatmap.js/blob/master/examples/heatmap-legend/index.html

https://www.geeksforgeeks.org/frequent-element-array/

https://stackoverflow.com/questions/4672279/bresenham-algorithm-in-javascript

https://www.chartjs.org/docs/latest/

http://fabricjs.com/docs/fabric.Line.html

https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection

https://en.wikipedia.org/wiki/Heat_map