Maze

Έκδοση: 4.1

Εγχειρίδιο Χρήσης

Μια προσπάθεια οπτικοποίησης
πέντε αλγόριθμων αναζήτησης

Νικόλαος Κανάργιας
Φοιτητής ΕΑΠ
Αθήνα 2013

Περιεχόμενα

Λίγα λόγια για το πρόβλημα

Το ερέθισμα για την εκπόνηση της παρούσας εργασίας ήταν η 1η ΓΕ της ΘΕ ΠΛΗ31 του Ακαδ. Έτους 2012-13, δύο ερωτήματα της οποίας είχαν σχέση με τον σχεδιασμό κίνησης (motion planning) για την αναζήτηση της διαδρομής προς τον στόχο από ένα ρομπότ το οποίο μπορεί να κινηθεί σε ένα πλέγμα το οποίο περιλαμβάνει εμπόδια. Την ιδέα για την παρούσα υλοποίηση μου την έδωσε το animation που μπορεί κάποιος να δει στο άρθρο της Wikipedia το σχετικό με τον αλγόριθμο του Dijkstra (http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm).

Το περιγραφόμενο λογισμικό λύνει και οπτικοποιεί το παραπάνω πρόβλημα υλοποιώντας μια παραλλαγή των αλγόριθμων DFS, BFS και A*, όπως αυτοί περιγράφονται στο βιβλίο "Τεχνητή Νοημοσύνη και Έμπειρα Συστήματα" της Ε. Κεραυνού, ΠΑΤΡΑ 2000, καθώς και τον αλγόριθμο της άπληστης αναζήτησης, σαν ειδική περίπτωση του A*.

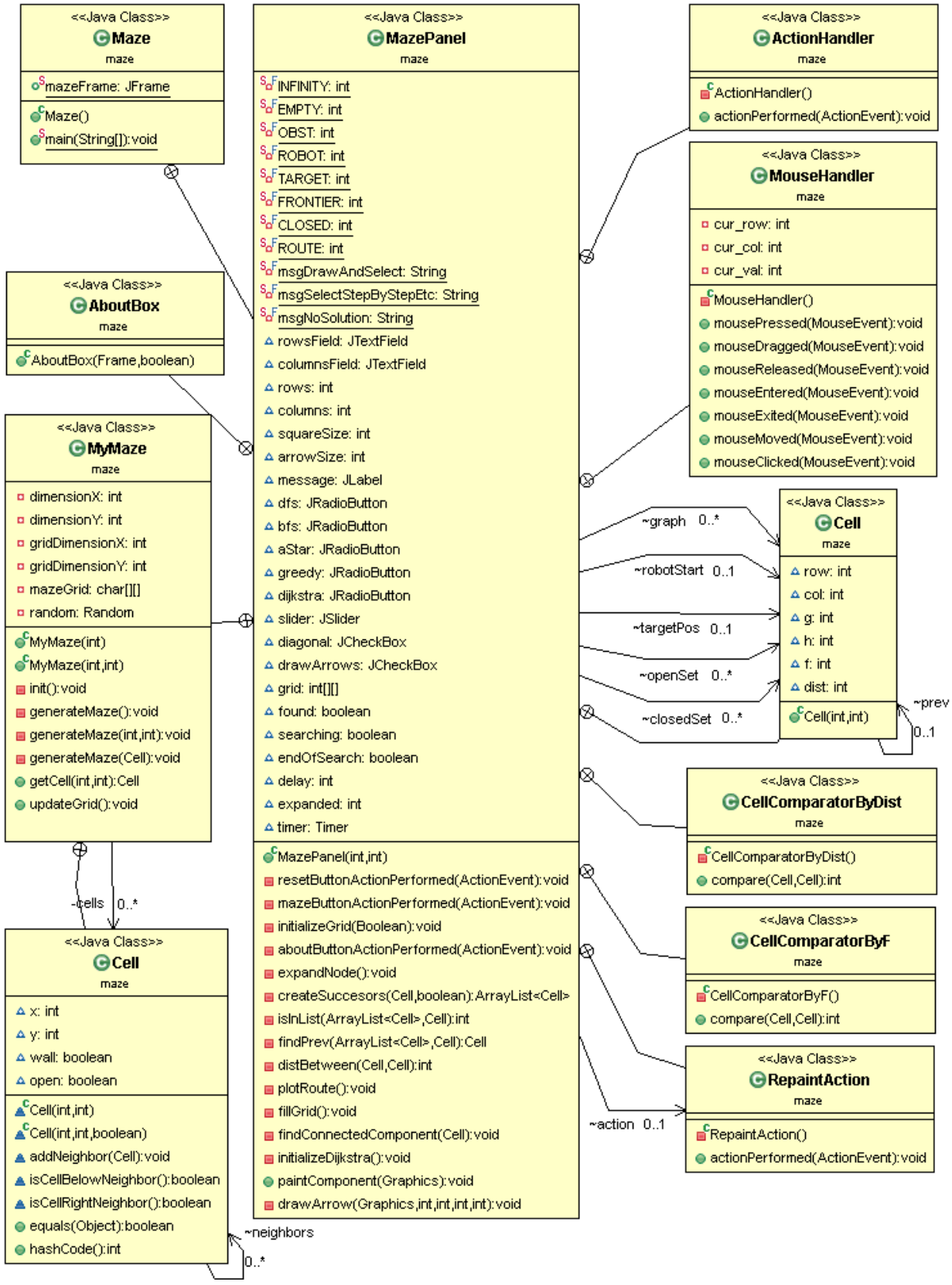
Το λογισμικό επίσης υλοποιεί τον αλγόριθμο του Dijkstra όπως ακριβώς αυτός περιγράφεται στο σχετικό άρθρο της Wikipedia.

Τις αρχικές εκδόσεις (ver. 1.0 μέχρι 3.1) του λογισμικού τις ανάρτησα στο Forum της ΠΛΗ31 στο Moodle από 17-1-2013 μέχρι 29-1-2013.

Εκφράζω τις θερμές ευχαριστίες μου στον Καθηγητή κ. Δημήτριο Καλλέ για την ενθάρρυνση και την υποστήριξή του, στον Καθηγητή κ. Παναγιώτη Σταματόπουλο για τις πολύτιμες παρατηρήσεις του και στον δάσκαλό μου, Καθηγητή κ. Σπύρο Λυκοθανάση για την εξαίρετη διδασκαλία του.

Ν. Κ.
Αθήνα, Νοέμβριος 2013

Το διάγραμμα κλάσεων



Το παραπάνω διάγραμμα δημιουργήθηκε με τη βοήθεια του προσθέτου ObjectAid (http://www.objectaid.com) του Eclipse IDE.

Σχολιασμός της δομής των κλάσεων

Πέρα από τον εκτενή σχολιασμό που υπάρχει στον κώδικα Java του λογισμικού, μπορούν να προστεθούν τα παρακάτω:

Η κύρια κλάση είναι η **Maze** η οποία έχει σαν πεδίο την κύρια φόρμα (mazeFrame) της εφαρμογής.

Ένθετη στην κλάση Maze είναι η κλάση MazePanel.

Η κλάση **MazePanel** καθορίζει το περιεχόμενο της κύριας φόρμας και περιέχει όλη την λειτουργικότητα του προγράμματος.

Ένθετες στην κλάση MazePanel είναι οι κλάσεις:

ActionHandler : Καθορίζει την λειτουργικότητα που θα εκτελεστεί όταν ο χρήστης πιέσει κάποιο από τα τρία κουμπιά της κύριας φόρμας.

MouseHandler : Χειρίζεται τις κινήσεις του ποντικιού καθώς ο χρήστης "ζωγραφίζει" εμπόδια ή μετακινεί το ρομπότ ή/και τον στόχο.

Cell : Αναπαριστά το κελί του πλέγματος.

CellComparatorByF : Καθορίζει ότι τα κελιά θα ταξινομούνται με βάση το πεδίο τους f (χρησιμοποιείται από τους αλγόριθμους A* και Greedy).

CellComparatorByDist : Καθορίζει ότι τα κελιά θα ταξινομούνται με βάση το πεδίο τους dist (χρησιμοποιείται από το αλγόριθμο του Dijkstra).

RepaintAction : Η κλάση που είναι υπεύθυνη για το animation.

AboutBox : Η κλάση η οποία δημιουργεί το About Box της εφαρμογής.

MyMaze : Δημιουργεί έναν τυχαίο, τέλειο (χωρίς κύκλους) λαβύρινθο. Ο κώδικας της κλάσης είναι μια προσαρμογή, με τον αυθεντικό σχολιασμό, της απάντησης που έδωσε ο χρήστης DoubleMx2 στις 25 Αυγούστου σε σχετικό ερώτημα του χρήστη nazar_art στο stackoverflow.com:

http://stackoverflow.com/questions/18396364/maze-generation-arrayindexoutofboundsexception

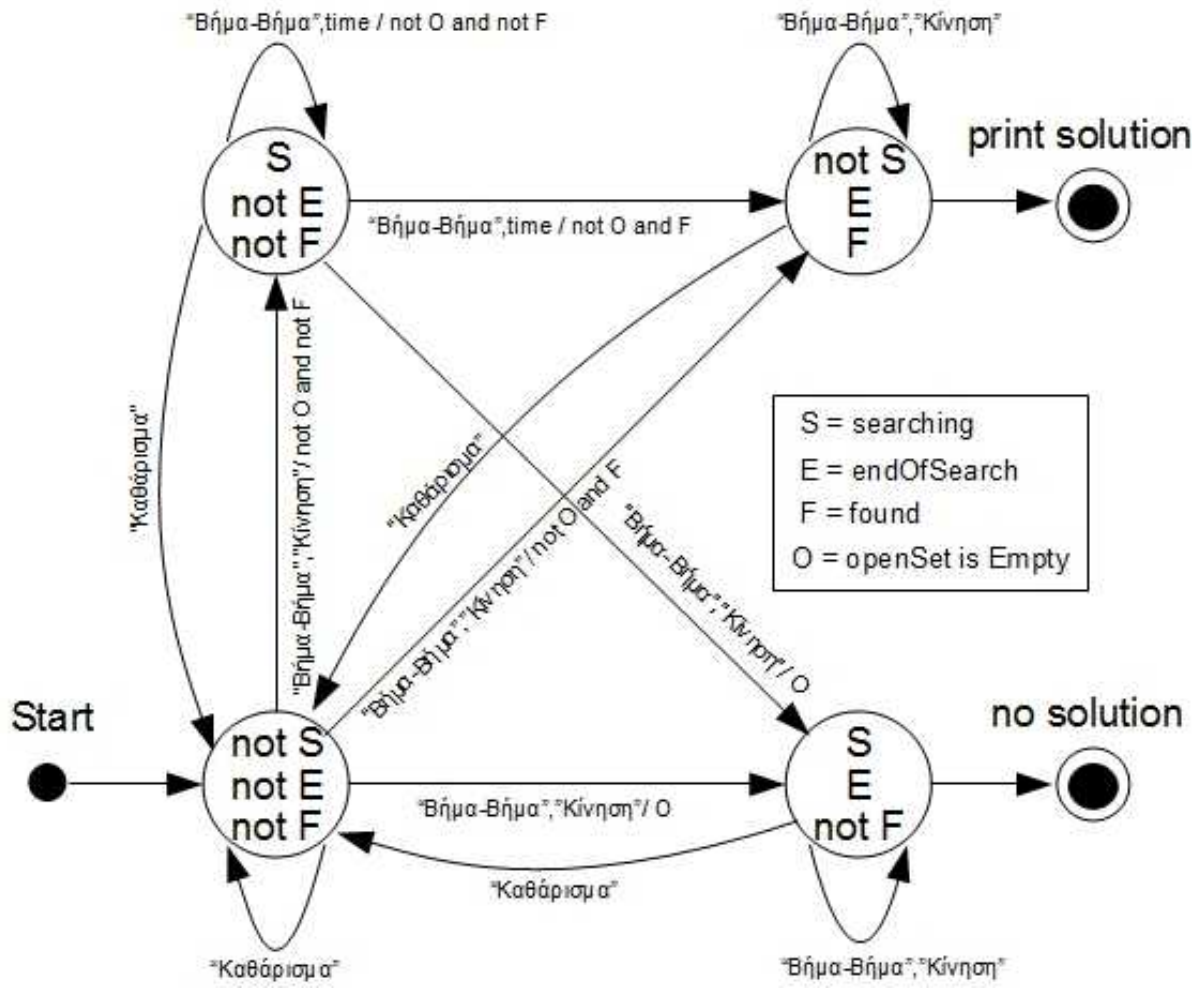
Ένθετη στην κλάση MyMaze είναι η κλάση **Cell** η οποία αναπαριστά το κελί για τις ανάγκες δημιουργίας του τυχαίου λαβύρινθου.

Το διάγραμμα μετάβασης καταστάσεων

Η κατάσταση της αναζήτησης είναι συνδυασμός τριών παραμέτρων

- Του κατά πόσον η αναζήτηση είναι σε εξέλιξη (searching).
- Του κατά πόσον η αναζήτηση έχει φθάσει στο τέρμα της (endOfSearch).
- Του κατά πόσον έχει βρεθεί ο στόχος (found).

Υπάρχουν οκτώ συνδυασμοί τιμών που μπορεί να λάβουν οι παραπάνω λογικές μεταβλητές. Από αυτούς μόνον τέσσαρες είναι οι συνδυασμοί που αντιστοιχούν στις πιθανές καταστάσεις αναζήτησης και απεικονίζονται στο παρακάτω διάγραμμα μετάβασης καταστάσεων:

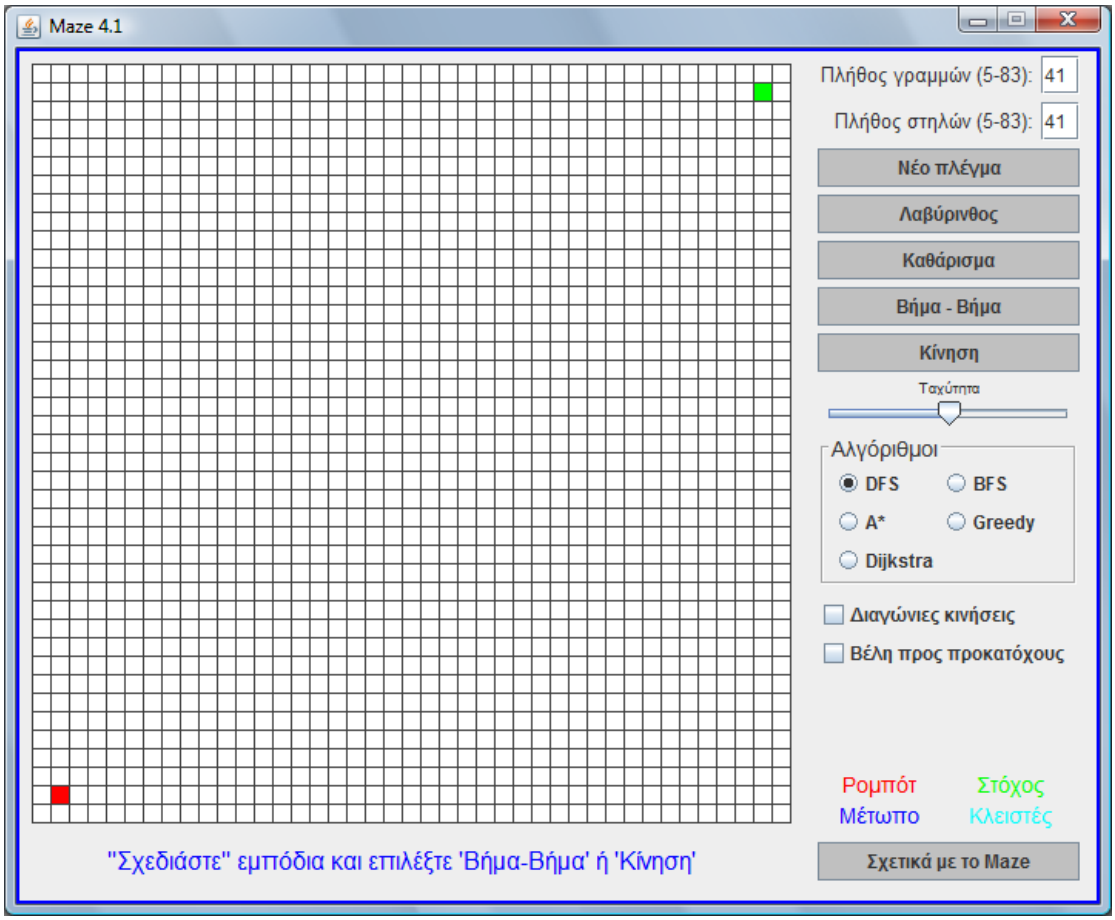


Οι ενδείξεις "Βήμα-Βήμα", "Κίνηση" και "Καθάρισμα" σημαίνουν: μετά το πάτημα του αντίστοιχου κουμπιού.

Η ένδειξη time σημαίνει: μετά την πάροδο του χρόνου καθυστέρησης (στην περίπτωση που η αναζήτηση ξεκίνησε με το πάτημα του κουμπιού "Κίνηση").

Η λειτουργία του προγράμματος

Με την εκκίνηση το πρόγραμμα εμφανίζει την παρακάτω φόρμα:



Η εφαρμογή χρησιμοποιεί μια επιφάνεια σταθερού εμβαδού, διαστάσεων 500×500 pixels, για να εμφανίσει το πλέγμα.

Ο χρήστης μπορεί να μεταβάλει τα πλήθη των γραμμών και των στηλών του πλέγματος, εισάγοντας στα αντίστοιχα πεδία τις επιθυμητές τιμές και πιέζοντας στη συνέχεια το κουμπί "Νέο πλέγμα".

Το πρόγραμμα θα προσαρμόσει το μέγεθος του κελιού για την καλύτερη αξιοποίηση της διαθέσιμης επιφάνειας.

Ο χρήστης μπορεί να προσθέσει όσα εμπόδια θέλει, όπως θα σχεδίαζε ελεύθερες καμπύλες με ένα σχεδιαστικό πρόγραμμα.

Αφαίρεση μεμονωμένων εμποδίων γίνεται κάνοντας κλικ επάνω τους.

Η θέση του ρομπότ ή/και του στόχου μπορεί να αλλάξει με σύρσιμο με το ποντίκι.

Η επιλογή του αλγόριθμου αναζήτησης γίνεται πιέζοντας το αντίστοιχο κουμπί.

Αν επιθυμούμε και διαγώνιες κινήσεις του ρομπότ, τσεκάρουμε το αντίστοιχο κουτί.

Για κάθε κόμβο που αναπτύσσεται η εφαρμογή μπορεί να σχεδιάζει βέλη από τους διαδόχους προς την προκάτοχο κατάσταση. Ο υπολογισμός και η σχεδίαση αυτών των βελών είναι μια χρονοβόρα διαδικασία. Μέχρι και την παρούσα έκδοση της εφαρμογής, για κάθε κόμβο που αναπτύσσεται πρέπει να υπολογιστούν και

σχεδιαστούν ξανά και τα βέλη όλων των κόμβων που έχουν ήδη αναπτυχθεί. Ο απαιτούμενος χρόνος για αυτόν τον υπολογισμό είναι δυστυχώς εκθετική συνάρτηση του πλήθους των κόμβων που αναπτύχθηκαν και στην χειρότερη περίπτωση μια αναζήτηση μπορεί να διαρκέσει περίπου 6 λεπτά της ώρας (με επεξεργαστή Intel Core2 Duo @ 2.66GHz). Το πρόβλημα ξεπερνιέται με την απενεργοποίηση του σχεδιασμού των βελών για μεγάλα πλέγματα, με την κατάλληλη χρήση του κουτιού "Βέλη προς προκατόχους", οπότε ο μέγιστος χρόνος αναζήτησης γίνεται μικρότερος από 45 δευτερόλεπτα. Σημειωτέον ότι τα βέλη δεν είναι επαρκώς ορατά, όταν το μέγεθος του κελιού είναι πολύ μικρό.

Η εφαρμογή θεωρεί ότι το ίδιο το ρομπότ έχει κάποιον όγκο. Συνεπώς δεν μπορεί να κινηθεί διαγώνια προς ελεύθερο κελί ανάμεσα από δύο εμπόδια που εφάπτονται σε μία κορυφή τους. Με άλλα λόγια, διαγώνια κίνηση επιτρέπεται μόνον όταν ένα τουλάχιστον από τα δύο κελιά που εφάπτονται με την αρχική και την τελική θέση του ρομπότ είναι ελεύθερα.

Η αναζήτηση μπορεί να γίνει με δύο τρόπους:

1. Με επαναληπτική πίεση στο κουμπί "Βήμα-Βήμα", οπότε βλέπουμε να επεκτείνεται ένας κόμβος κάθε φορά.
2. Με εφάπαξ πίεση στο κουμπί "Κίνηση", οπότε η επέκταση των κόμβων γίνεται αυτόματα, παρεμβάλλοντας ανάμεσα σε δύο διαδοχικές επεκτάσεις μια χρονική παύση από 0 μέχρι 1 δευτερόλεπτο. Το μέγεθος της χρονικής παύσης μπορεί να ρυθμιστεί ανάλογα με την θέση στην οποία θα τοποθετηθεί ο slider "Ταχύτητα".

Μεταπήδηση από την αναζήτηση "Βήμα-Βήμα" στην αναζήτηση με "Κίνηση" και αντίστροφα γίνεται πιέζοντας το αντίστοιχο κουμπί, ακόμη και όταν η αναζήτηση είναι σε εξέλιξη.

Η ταχύτητα μιας αναζήτησης με κίνηση μπορεί να μεταβληθεί, ακόμη και αν η αναζήτηση είναι σε εξέλιξη, αρκεί να τοποθετηθεί ο slider "Ταχύτητα" στην νέα επιθυμητή θέση και στη συνέχεια πιεστεί το κουμπί "Κίνηση".

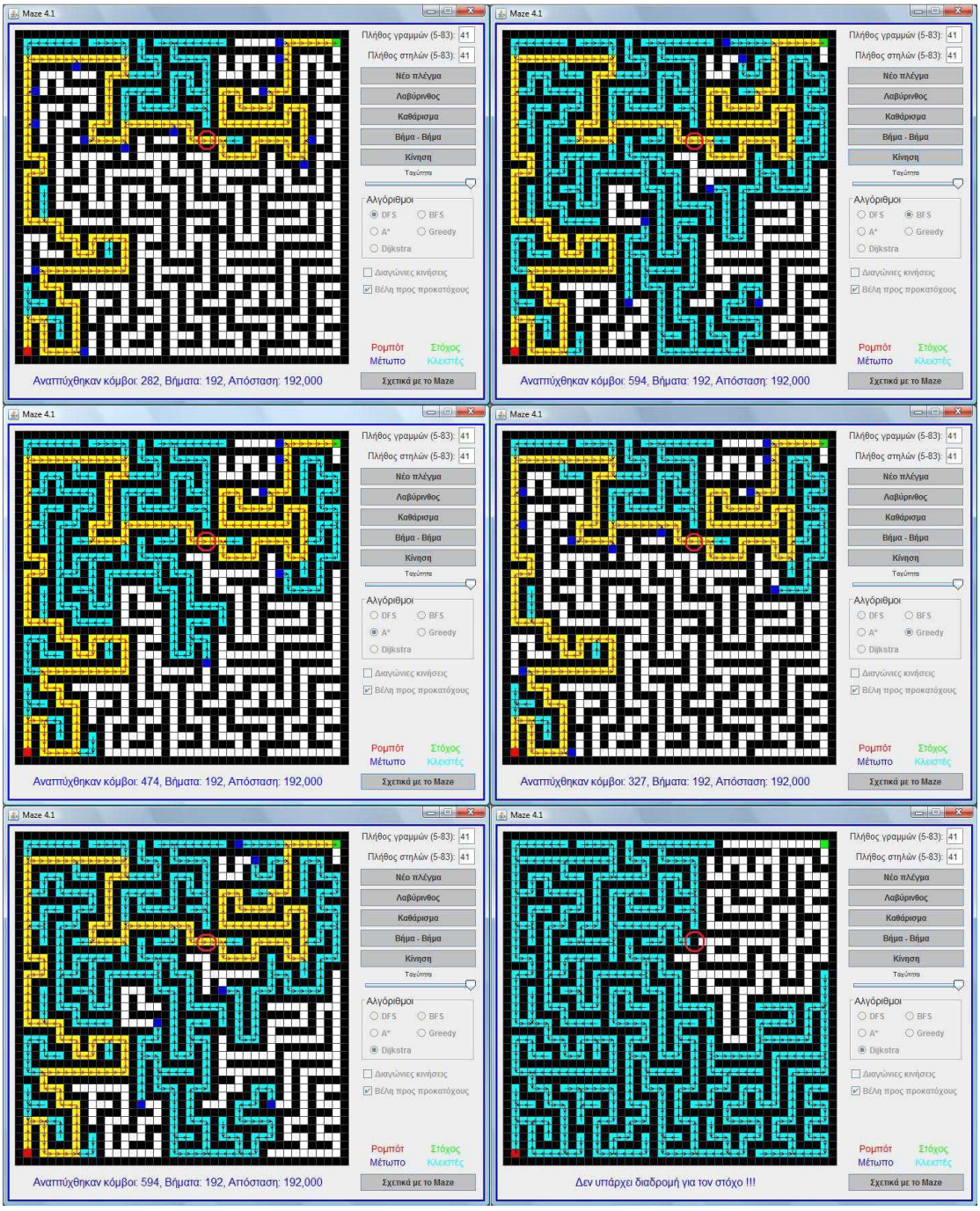
Το πάτημα του κουμπιού "Καθάρισμα" μόλις ολοκληρωθεί μια αναζήτηση, ή όταν αυτή είναι ακόμη σε εξέλιξη, έχει σαν συνέπεια να καθαρίσει η τρέχουσα αναζήτηση, αλλά τα εμπόδια, το ρομπότ και ο στόχος παραμένουν στην θέση τους, προκειμένου να είναι δυνατός ο πειραματισμός με κάποιον άλλον αλγόριθμο με τα ίδια δεδομένα.

Αν πατηθεί το κουμπί "Καθάρισμα" για δεύτερη συνεχόμενη φορά, τότε καθαρίζουν και τα εμπόδια, το δε ρομπότ και ο στόχος επανέρχονται στην αρχική τους θέση.

Δεν είναι δυνατή η αλλαγή των θέσεων εμποδίων, ρομπότ και στόχου όπως και του είδους του αλγόριθμου, ενόσω η αναζήτηση είναι σε εξέλιξη.

Σχετικά με την επίλυση του λαβυρίνθου

Αν αντί για το κουμπί “Νέο πλέγμα” πιεστεί το κουμπί “Λαβύρινθος”, η εφαρμογή θα δημιουργήσει και θα τοποθετήσει πάνω στο πλέγμα έναν τυχαίο λαβύρινθο. Αν θεωρηθούν τα ελεύθερα κελιά του λαβυρίνθου σαν ένας γράφος, τότε τα ελεύθερα κελιά που δημιουργούνται είναι ένα συνεκτικό δένδρο και σαν τέτοιο δεν περιέχει κύκλους.



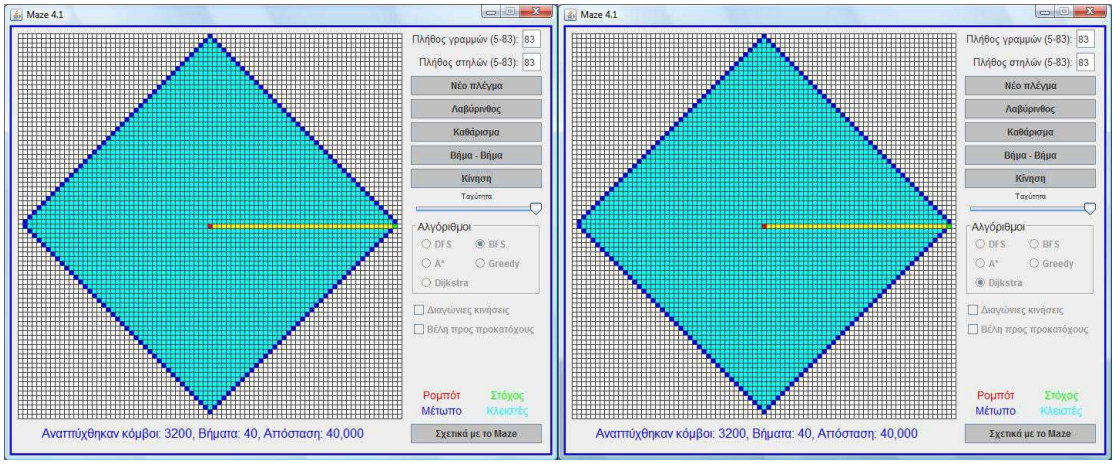
Όπως είναι γνωστό, σε κάθε συνεκτικό δένδρο υπάρχει μία μόνο διαδρομή που συνδέει δύο οποιουσδήποτε κόμβους του. Όπως φαίνεται και στις παραπάνω εικόνες και οι πέντε αλγόριθμοι αναζήτησης υπολόγισαν την μοναδική διαδρομή προς τον στόχο, έχοντας φυσικά αναπτύξει διαφορετικούς κόμβους ο καθένας.

Όπως επίσης είναι γνωστό, αν σε ένα συνεκτικό δένδρο αφαιρέσουμε μια οποιαδήποτε ακμή, το δένδρο παύει να είναι πλέον συνεκτικό και αν η ακμή που αφαιρέθηκε ήταν τμήμα της διαδρομής μεταξύ δύο συγκεκριμένων κόμβων, οι κόμβοι αυτοί δεν θα είναι πλέον αμοιβαία προσπελάσιμοι.

Στην τελευταία από τις προηγούμενες εικόνες φαίνεται ότι τοποθετώντας ένα εμπόδιο (στον κόκκινο κύκλο) πάνω στην μοναδική διαδρομή μεταξύ ρομπότ και στόχου κάνει πλέον τον στόχο απροσπέλαστο από το ρομπότ. Στην εικόνα φαίνεται η αποτυχία του Dijkstra να εντοπίσει τον στόχο. Ανάλογο αποτέλεσμα έχουν και οι υπόλοιποι αλγόριθμοι.

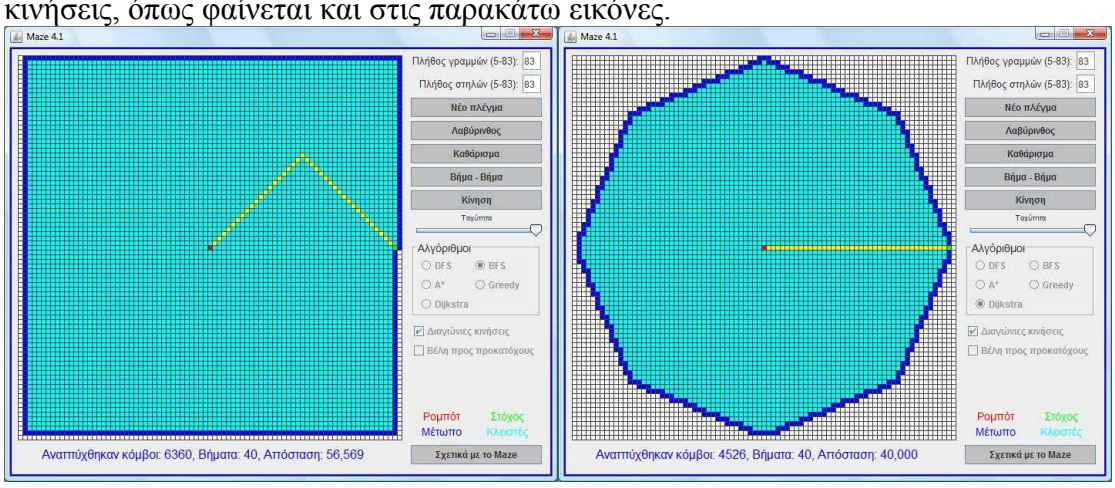
Από τους πέντε αλγόριθμους που υλοποιεί η εφαρμογή, μόνον οι BFS και Dijkstra εξερευνούν προς όλες τις κατευθύνσεις, ενώ το σύνολο των ανοικτών καταστάσεων έχει την μορφή ενός κύματος που διαστέλλεται με κέντρο την αρχική θέση του ρομπότ.

Στην περίπτωση που δεν επιτρέπονται διαγώνιες κινήσεις, το κύμα αυτό για τους δύο παραπάνω αλγόριθμους έχει την μορφή ρόμβου όπως φαίνεται και στις παρακάτω εικόνες, ενώ οι δύο αυτοί αλγόριθμοι έχουν ακριβώς την ίδια συμπεριφορά. Δηλαδή για ένα δεδομένο πρόβλημα βρίσκουν την ίδια λύση, έχοντας αναπτύξει το ίδιο σύνολο κόμβων.



Αυτός είναι και ο λόγος για τον οποίο οι λύσεις του λαβυρίνθου που εικονίζονται στην προηγούμενη σελίδα τις οποίες έδωσαν οι δύο αλγόριθμοι είναι πανομοιότυπες.

Οι δύο αλγόριθμοι συμπεριφέρονται διαφορετικά, όταν επιτρέπονται και διαγώνιες κινήσεις, όπως φαίνεται και στις παρακάτω εικόνες.



Οι αλγόριθμοι αναζήτησης

Η εφαρμογή υλοποιεί τους παρακάτω αλγόριθμους:

Αλγόριθμος για Αναζήτηση σε Βάθος

1. ΑΝΟΙΚΤΕΣ := [s_o], ΚΛΕΙΣΤΕΣ := [ ]
2. Εάν ΑΝΟΙΚΤΕΣ = [ ], τότε τερμάτισε. Δεν υπάρχει λύση.
3. Αφαίρεσε την πρώτη κατάσταση, s_i, από τις ΑΝΟΙΚΤΕΣ και πρόσθεσέ την στις ΚΛΕΙΣΤΕΣ. Εάν η s_i είναι η s_g , τότε τερμάτισε και επέστρεψε τη διαδρομή από την s_g στην s_o.
4. Δημιούργησε τις διαδόχους της s_i, που δεν ανήκουν ήδη στις ΑΝΟΙΚΤΕΣ ή ΚΛΕΙΣΤΕΣ, με βάση τις ενέργειες που μπορούν να εφαρμοστούν στην s_i. Η κάθε διάδοχος έχει ένα δείκτη προς την s_i, ως την προκάτοχό της.
5. Πρόσθεσε τις διαδόχους στην **αρχή** της λίστας ΑΝΟΙΚΤΕΣ και επανάλαβε από την εντολή 2.

Ο αλγόριθμος της αναζήτησης σε πλάτος (BFS) είναι ακριβώς ίδιος με τον αλγόριθμο αναζήτησης σε βάθος (DFS), με την μόνη διαφορά ότι στο βήμα 5 οι νέες ανοικτές καταστάσεις προστίθενται στο τέλος και όχι στην αρχή της λίστας.

Ευρετική Αναζήτηση: Αλγόριθμος A*

1. ΑΝΟΙΚΤΕΣ := [s_o], ΚΛΕΙΣΤΕΣ := [ ]
2. Εάν ΑΝΟΙΚΤΕΣ = [ ], τότε τερμάτισε. Δεν υπάρχει λύση.
3. Αφαίρεσε την κατάσταση, s_i, από την λίστα ΑΝΟΙΚΤΕΣ, για την οποία $f(s_i) \leq f(s_j)$ για όλες τις άλλες ανοικτές καταστάσεις s_j και πρόσθεσέ την στις ΚΛΕΙΣΤΕΣ. Εάν η s_i είναι η s_g , τότε τερμάτισε και επέστρεψε τη διαδρομή από την s_g στην s_o.
4. Δημιούργησε τις διαδόχους της s_i και δώσε στην κάθε διάδοχο έναν δείκτη προς την s_i, ως την προκάτοχό της.
5. Επανάλαβε τα ακόλουθα για κάθε διάδοχο, s_j, της s_i :
   - Υπολόγισε την τιμή $f(s_j)$.
   - Εάν η s_j δεν ανήκει ούτε στις ΑΝΟΙΚΤΕΣ, ούτε στις ΚΛΕΙΣΤΕΣ καταστάσεις, τότε πρόσθεσε την s_j στις ΑΝΟΙΚΤΕΣ με τιμή αξιολόγησης, $f(s_j)$.
   - Εάν η s_j ήδη ανήκει στις ΑΝΟΙΚΤΕΣ ή ΚΛΕΙΣΤΕΣ, τότε σύγκρινε τη νέα τιμή αξιολόγησής της, έστω *νέα*, με την παλαιότερή της, έστω *παλαιά*. Εάν *παλαιά* ≤ *νέα*, τότε απόβαλε το νέο κόμβο με την κατάσταση s_j. Διαφορετικά, αφαίρεσε το στοιχείο (s_j, *παλαιά*) από τη λίστα στην οποία ανήκει (ΑΝΟΙΚΤΕΣ ή ΚΛΕΙΣΤΕΣ) και πρόσθεσε το στοιχείο (s_j, *νέα*) στις ΑΝΟΙΚΤΕΣ.
6. Επανέλαβε από την εντολή 2.

Ο αλγόριθμος της άπληστης αναζήτησης (Greedy) είναι ακριβώς ίδιος με τον αλγόριθμο A*, με την μόνη διαφορά ότι για τον υπολογισμό της τιμής της συνάρτησης $f(s_j)$ στο βήμα 5 χρησιμοποιείται μόνο η συνάρτηση h, δηλαδή $g(s) = 0$ για όλες τις καταστάσεις.

Υπάρχουν λεπτομερειακά σχόλια στον κώδικα που αντιστοιχούν κάθε ένα από τα παραπάνω βήματα με την εντολή που τα υλοποιεί.

Για τον αλγόριθμο του Dijkstra χρησιμοποιήθηκε ο ψευδοκώδικας από το σχετικό άρθρο της Wikipedia:

```
 1  function Dijkstra(Graph, source):
 2      for each vertex v in Graph:                                  // Initializations
 3          dist[v] := infinity ;                                    // Unknown distance function from
 4                                                                   // source to v
 5          previous[v] := undefined ;                               // Previous node in optimal path
 6      end for                                                      // from source
 7
 8      dist[source] := 0 ;                                          // Distance from source to source
 9      Q := the set of all nodes in Graph ;                         // All nodes in the graph are
10                                                                   // unoptimized - thus are in Q
11      while Q is not empty:                                        // The main loop
12          u := vertex in Q with smallest distance in dist[] ;      // Source node in first case
13          remove u from Q ;
14          if dist[u] = infinity:
15              break ;                                              // all remaining vertices are
16          end if                                                   // inaccessible from source
17
18          for each neighbor v of u:                                // where v has not yet been
19                                                                   // removed from Q.
20              alt := dist[u] + dist_between(u, v) ;
21              if alt < dist[v]:                                    // Relax (u,v,a)
22                  dist[v] := alt ;
23                  previous[v] := u ;
24                  decrease-key v in Q;                             // Reorder v in the Queue
25              end if
26          end for
27      end while
28      return dist;
29  endfunction
```

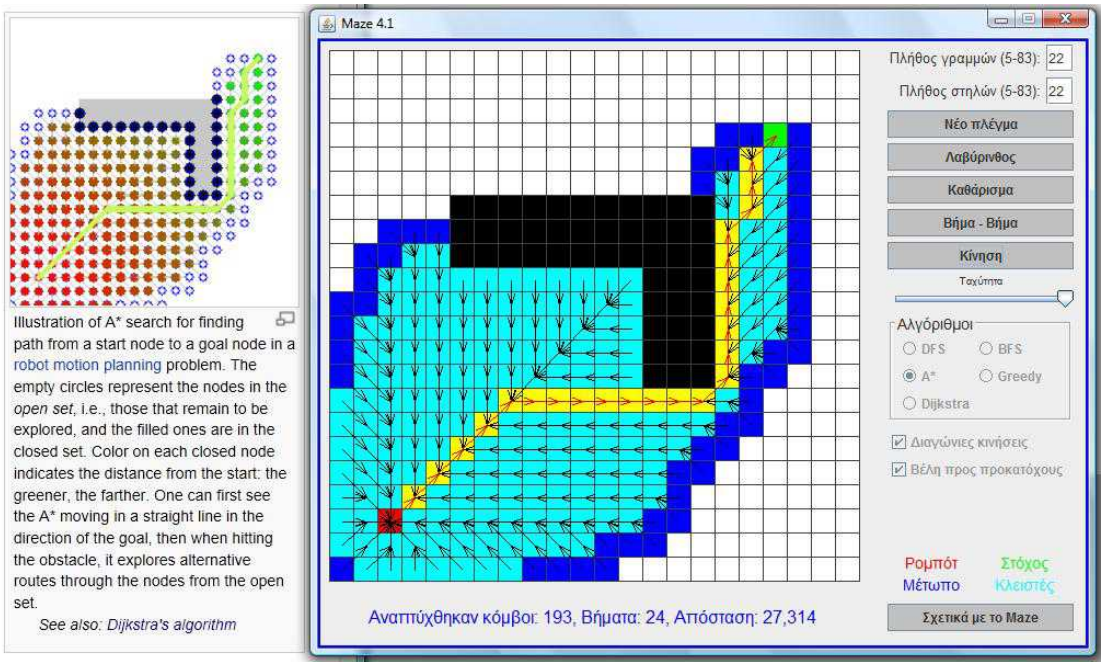
Και πάλι λεπτομερειακά σχόλια στον κώδικα αντιστοιχούν τα βήματα του παραπάνω ψευδοκώδικα με τις εντολές του προγράμματος.

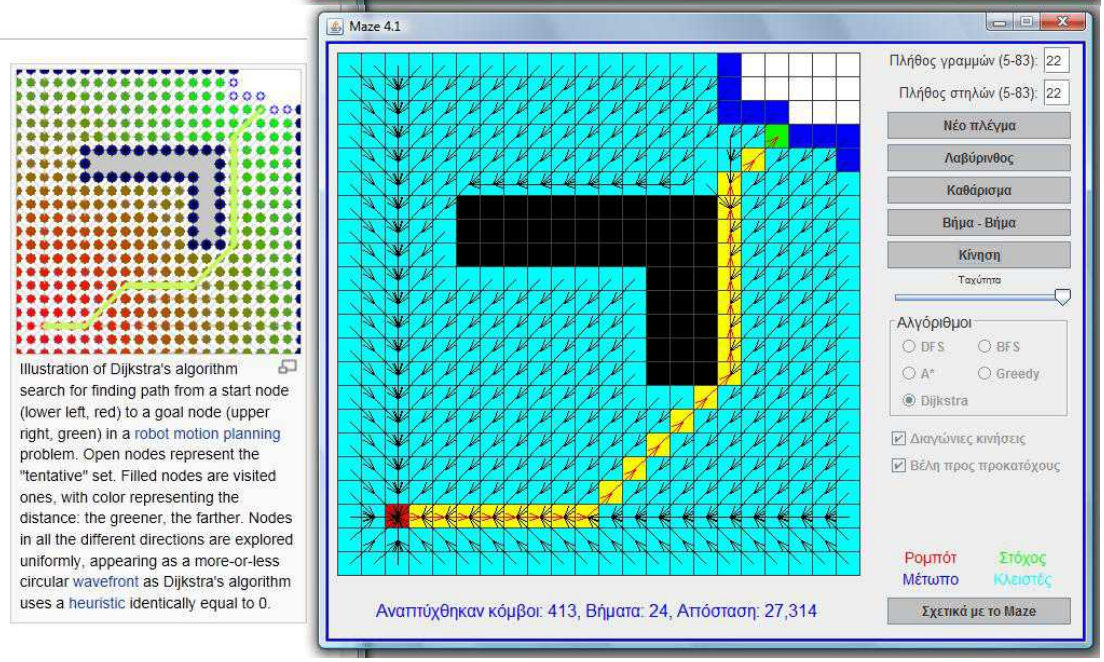
Ένα μέρος της αρχικοποίησης του αλγόριθμου που δεν περιλαμβάνεται στον παραπάνω ψευδοκώδικα και το οποίο σχολιάζεται επαρκώς στον κώδικα της εφαρμογής μπορεί να απαιτήσει αρκετό χρόνο (δευτερόλεπτα), ιδίως σε πλέγματα μεγάλων διαστάσεων.

Αμέσως μετά από αυτήν την αρχικοποίηση είναι ευτελές για την εφαρμογή να δώσει την ενδεχόμενη αρνητική απάντηση στο εκάστοτε πρόβλημα (δεν έχει παρά να ελέγξει αν ο στόχος βρίσκεται μέσα στο σύνολο των κόμβων που αποτελούν την συνεκτική συνιστώσα στην οποία ανήκει η αρχική θέση του ρομπότ). Για εποπτικούς και μόνο λόγους επιχειρεί και την αναζήτηση, μέχρι να καταλήξει στην προδιαγεγραμμένη αποτυχία.

Στιγμιότυπα του προγράμματος

Ακολουθούν δύο στιγμιότυπα αναζητήσεων με A* και Dijkstra σε σύγκριση με τις λύσεις από τα αντίστοιχα άρθρα της Wikipedia:
http://en.wikipedia.org/wiki/A*_search_algorithm
http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm





Ιστορικό εκδόσεων

1.0 (17-01-2013), Γραμμές: 845
Σταθερές διαστάσεις πλέγματος και κελιού, Αλγόριθμοι: DFS, BFS, A*.

1.1 (18-01-2013), Γραμμές: 809
Πιο σύντομη κωδικοποίηση της μεθόδου createSuccesors.

1.2 (18-01-2013), Γραμμές: 812
Διόρθωση λάθους.

1.3 (20-01-2013), Γραμμές: 812
Καλύτερη κωδικοποίηση της διόρθωσης του λάθους της έκδοσης 1.1.

1.4 (20-01-2013), Γραμμές: 864
Σχεδίαση βέλους προς την προκάτοχο κατάσταση.

2.0 (22-01-2013), Γραμμές: 968
Διαγώνιες κινήσεις.

2.1 (25-01-2013), Γραμμές: 981
Σωστή προτεραιότητα κινήσεων.

2.2 (26-01-2013), Γραμμές: 1008
Άπληστος αλγόριθμος.

2.3 (26-01-2013), Γραμμές: 1021
Υπολογισμός απόστασης διαδρομής προς τον στόχο.

2.4 (26-01-2013), Γραμμές: 1024
Διόρθωση λάθους.

3.0 (29-01-2013), Γραμμές: 1344
Αλγόριθμος Dijkstra.

3.1 (29-01-2013), Γραμμές: 1340
Διόρθωση λάθους.

3.2 (14-08-2013), Γραμμές: 1519
Ρυθμιζόμενες διαστάσεις πλέγματος και κελιού.

3.3 (07-10-2013), Γραμμές: 1546
Εναλλακτικός τερματισμός αλγορίθμων DFS, BFS, A* και Greedy.

3.3α (09-10-2013), Γραμμές: 1546
Δυνατότητα για πλέγμα διάστασης 5×5.

3.4 (10-10-2013), Γραμμές: 1554
Σωστή προτεραιότητα κινήσεων αλγόριθμου BFS.

3.4α (12-10-2013), Γραμμές: 1559
Μικρές βελτιώσεις στην εμφάνιση που αφορούν τον αλγόριθμο Dijkstra.

3.4β (14-10-2013), Γραμμές: 1559
Καλύτερη διαχείριση των στοιχείων της διεπαφής.

4.0 (21-11-2013), Γραμμές: 1767
Σταθερό διαθέσιμο εμβαδόν πλέγματος, προαιρετική σχεδίαση βελών, λαβύρινθος.

4.1 (23-11-2013), Γραμμές: 1739
Αφαίρεση υποστήριξης αλγόριθμων κ. Κεραυνού