ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδίαση και επίλυση λαβυρίνθων σε Java

Του φοιτητή
Ορμάνη Γεώργιου
Αρ. Μητρώου: 1973

Επιβλέπων καθηγητής
Αλέξανδρος Ζερδελίδης

ΘΕΣΣΑΛΟΝΙΚΗ ΜΑΡΤΙΟΣ 2012

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ **2**

ABSTRACT **3**

ΚΕΦΑΛΑΙΟ 1 ΛΑΒΥΡΙΝΘΟΙ – ΚΑΤΗΓΟΡΙΕΣ **6**

ΚΕΦΑΛΑΙΟ 2 ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΑΣΚΕΥΗΣ ΚΑΙ ΕΠΙΛΥΣΗΣ ΛΑΒΥΡΙΝΘΩΝ **12**

ΚΑΤΑΣΚΕΥΗ ΛΑΒΥΡΙΝΘΩΝ 12

ΕΠΙΛΥΣΗ ΛΑΒΥΡΙΝΘΩΝ 26

ΣΥΓΚΡΙΣΕΙΣ ΔΙΑΦΟΡΩΝ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΑΣΚΕΥΗΣ 33

ΚΕΦΑΛΑΙΟ 3 ΕΠΙΛΟΓΗ ΛΑΒΥΡΙΝΘΟΥ – ΣΧΕΔΙΑΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ **37**

Ο ΛΑΒΥΡΙΝΘΟΣ ΠΟΥ ΔΗΜΙΟΥΡΓΟΥΜΕ 37

ΜΕΘΟΔΟΙ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ 40

ΚΕΦΑΛΑΙΟ 4 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ **53**

ΧΡΟΝΟΣ ΔΗΜΙΟΥΡΓΙΑΣ ΛΑΒΥΡΙΝΘΩΝ 53

ΧΡΟΝΟΣ ΕΠΙΛΥΣΗΣ ΛΑΒΥΡΙΝΘΩΝ 56

ΣΥΜΠΕΡΑΣΜΑΤΑ **59**

ΒΙΒΛΙΟΓΡΑΦΙΑ **60**

ΧΡΗΣΙΜΑ LINKS **60**

ΠΕΡΙΛΗΨΗ

Στόχος της πτυχιακής αυτής ήταν η δημιουργία και η επίλυση ενός λαβυρίνθου με τη γλώσσα Java. Λαβύρινθος είναι ένα παζλ περιήγησης με τη μορφή ενός σύνθετου διακλαδωτού περάσματος, μέσω του οποίου ο λύτης πρέπει να βρει μια διαδρομή. Τέτοια μορφή λαβυρίνθου με πολύπλοκα διακλαδωτά περάσματα αλλά και με επιπλέον παραμέτρους, επιχειρήθηκε να δημιουργηθεί σ' αυτήν την εργασία.

Χρησιμοποιήθηκαν δύο διαφορετικοί αλγόριθμοι κατασκευής κι ένας αλγόριθμος επίλυσης. Οι αλγόριθμοι δημιουργίας λαβυρίνθων ήταν ο «Αλγόριθμος με τοίχους» και ο «Αλγόριθμος με δέντρο», ενώ για την επίλυση χρησιμοποιήθηκε ο αλγόριθμος εύρεσης συντομότερου μονοπατιού του Dijkstra.

Η κύρια διαφορά των δύο αλγορίθμων δημιουργίας, είναι ότι ο αλγόριθμος με τοίχους γεμίζει πρώτα τον λαβύρινθο με τοίχους κι έπειτα βρίσκει τις διαδρομές, ενώ ο αλγόριθμος με δέντρο βρίσκει πρώτα όλες τις πιθανές διαδρομές και μετά τις χαράζει στο λαβύρινθο. Ο λαβύρινθος εκτός από τις διαδρομές περιλαμβάνει και κάποιες επιπλέον παραμέτρους, μία πόρτα κι ένα κλειδί. Για να φτάσει στο τέρμα και να επιλύσει τον λαβύρινθο πρέπει πρώτα να πάρει το κλειδί από το πρώτο μέρος του λαβυρίνθου, να ανοίξει την πόρτα που βρίσκεται στη μέση και τέλος να βρει το τέρμα στο δεύτερο μέρος του λαβυρίνθου. Οι δύο αυτές επιπλέον παράμετροι προσθέτουν αρκετή δυσκολία και πολυπλοκότητα στην επίλυση του λαβυρίνθου γι αυτό και δεν χρησιμοποιήθηκε ένας έτοιμος αλγόριθμος επίλυσης, αλλά ο αλγόριθμος εύρεσης συντομότερου μονοπατιού του Dijkstra. Έτσι, ο αλγόριθμος αυτός χρησιμοποιείται για τρία μονοπάτια, από την αρχή στο κλειδί, από το κλειδί στην πόρτα και από την πόρτα στο τέρμα.

Έπειτα από συγκρίσεις που έγιναν μεταξύ των δύο ειδών λαβυρίνθων, όσον αφορά τον χρόνο δημιουργίας, αλλά και το χρόνο επίλυσης του λαβυρίνθου, ο συγγραφέας προτείνει τη χρήση του αλγορίθμου με δέντρο στις περιπτώσεις που ο χρήστης χρειάζεται έναν πολύπλοκο και γρήγορο στην επίλυση λαβύρινθο, ενώ προτείνει τον αλγόριθμο με τοίχους σε περίπτωση που ο χρήστης χρειάζεται απλό και ταχύ στη δημιουργία λαβύρινθο.

ABSTRACT

The purpose of this study was the creation and solving of a maze with the use of Java language. A maze is a tour puzzle in the form of a complex branching passage through which the solver must find a route. Creation of such a maze with complex branching passages but also with extra parameters was attempted in this thesis.

Two different creation algorithms and a solving algorithm were used. The creation algorithms were the “algorithm with walls” and the “algorithm with tree”, while Dijkstra’s shortest path algorithm was used for solving.

The main difference of the two creation algorithms is in the maze filling process. The algorithm with walls first fills the maze with walls and finds the routes later, while the algorithm with tree first finds all the possible routes then carves them on the maze. The maze, besides the routes, includes some extra parameters, a door and a key. Before it reaches the finish to solve the maze, it has to get the key from the first part of the maze, open the door in the middle of the maze and lastly find the finish at the second part of the maze. These two parameters add some difficulty and complexity in the maze solving process and that’s why a shortest path finder algorithm was used instead of a standard solving algorithm. Thus, this algorithm is used for the three section routes that constitute the main, solving, route of the maze. From start to key, from key to door and from door to finish.

After comparisons between the two kind of mazes, about the creation and solving time of the maze, the author recommends the use of the algorithm with tree when the user needs a complex and fast solving maze, but in case the user requires a fast creation maze with lower complexity he recommends the algorithm with walls.

ΕΙΣΑΓΩΓΗ

Οι περισσότεροι άνθρωποι στο άκουσμα της λέξης λαβύρινθος σκέφτονταν ένα κτίριο με ψηλούς τοίχους που δύσκολα κάποιος μπορεί να βρει το δρόμο του, ενώ στις μέρες μας όλο και περισσότεροι είναι αυτοί που σκέφτονται ένα ηλεκτρονικό παιχνίδι.

Ο πιο γνωστός λαβύρινθος ανά τον κόσμο είναι αυτός που έχτισε ο μηχανικός Δαίδαλος για λογαριασμό του βασιλιά της Κρήτης Μίνωα. Εκεί ο βασιλιάς είχε κλείσει ένα μυθικό τέρας, τον Μινώταυρο, τον οποίο σκότωσε ο Θησέας. Παρόλο που εκείνος ο λαβύρινθος ήταν τόσο περίτεχνα φτιαγμένος, που ακόμα και ο κατασκευαστής του δυσκολεύτηκε να βρει την έξοδο όταν τελείωσε η κατασκευή του, ο Θησέας κατάφερε να σκοτώσει το τέρας και να βγει από τον λαβύρινθο με τη βοήθεια της κόρης του Μίνωα, Αριάδνης. Είναι η γνωστή ιστορία με τον «μίτο της Αριάδνης», το νήμα δηλαδή που ξετύλιξε ο Θησέας και μπόρεσε να βρει το δρόμο για πίσω.

Ο βασικός τύπος, στον οποίο πλέον κάθε λαβύρινθος μπορεί να κατηγοριοποιηθεί, είναι ο «λαβύρινθος βέλος". Αυτό είναι ουσιαστικά, σε μαθηματικούς όρους, ένα κατευθυνόμενο γράφημα (directed graph). Μπορεί να αναπαρασταθεί από ένα σύνολο από σημεία (points) ή κορυφές (vertex), που συνδέονται με βέλη. Στην επίλυση του λαβύρινθου κινούμαστε από μια αρχική κορυφή, σε μια τελική, ακολουθώντας την κατεύθυνση των βελών από κορυφή σε κορυφή. Πιο συγκεκριμένα, ο λαβύρινθος αποτελείται από ένα πεπερασμένο και εύκολα απαριθμήσιμο σύνολο καταστάσεων, όπου σε κάθε κατάσταση υπάρχει ένας πεπερασμένος και εύκολα απαριθμήσιμος αριθμός επιλογών που μπορούμε να κάνουμε για να αλλάξει κατάσταση. Έχουν εφευρεθεί πολλοί τύποι λαβυρίνθων με κανόνες, αλλά είναι όλοι ουσιαστικά κατευθυνόμενοι «λαβύρινθοι βέλη».

Παρόλαυτα, η δημιουργία και επίλυση λαβυρίνθων έχουν αποτελέσει και αποτελούν αντικείμενο ενασχόλησης για τους ανθρώπους διαφόρων εποχών και πολιτισμών. Οι σκανδιναβικές φυλές πίστευαν στις μαγικές ιδιότητες των λαβυρίνθων, οι ιθαγενείς Αμερικάνοι της Αριζόνα απεικονίζουν τον θεό I'itoi να ζει στο κέντρο ενός λαβυρίνθου κάτω από το βουνό Baboquivari, στην Γκόα της Ινδίας έχει βρεθεί σκαλισμένος λαβύρινθος σε πέτρωμα στις όχθες ποταμού, και αυτά είναι μόνο μερικά από τα

παραδείγματα (Labyrinthos). Οποιοσδήποτε κι αν είναι ο λόγος, η έννοια του λαβύρινθου προκαλεί μέχρι και τις μέρες μας το ενδιαφέρον, ακόμα και σαγηνεύει πολλούς ανθρώπους. Η χρήση τους είτε για λόγους διασκέδασης για λόγους διασκέδασης και ψυχαγωγίας (π.χ. παιδικά παιχνίδια, επίλυση λαβυρίνθων σε περιοδικά-κουίζ, λαβύρινθοι ανθρώπινου μεγέθους σε πάρκα, βιντεοπαιχνίδια), είτε για καλλιτεχνικούς και διακοσμητικούς λόγους, είτε για χρήση σε πειράματα με ποντίκια (μελέτη της ικανότητας μάθησης), είναι μέχρι σήμερα σημαντική (Wikipedia). Ένα χαρακτηριστικό παράδειγμα της ψυχαγωγίας που προσφέρουν οι λαβύρινθοι είναι οι λαβύρινθοι του Adrian Fisher, του μεγαλύτερου δημιουργού λαβυρίνθων, πραγματικών διαστάσεων, στον κόσμο. Ο Adrian Fisher (Adrian Fisher Design Ltd) είναι ο πιο παραγωγικός σχεδιαστής λαβυρίνθων στην ιστορία της ανθρωπότητας και έχει κερδίσει έξι φορές τον τίτλο του σχεδιαστή του μεγαλύτερου λαβυρίνθου στον κόσμο. Η εταιρεία του έχει δημιουργήσει τους μισούς λαβύρινθους-καθρέφτες του κόσμου και αυτός σχεδίασε τον πρώτο λαβύρινθο στον κόσμο σε χωράφι από καλαμπόκια το 1993 και εκατοντάδες άλλους από τότε.

Στην εργασία αυτή ασχολούμαστε με τους αλγόριθμους δημιουργίας και επίλυσης λαβυρίνθων σε ηλεκτρονικό υπολογιστή, την παρουσίαση και σύγκριση τους, καθώς και με τη δημιουργία ενός προγράμματος σε java που δημιουργεί και επιλύει έναν λαβύρινθο.

Στο πρώτο κεφάλαιο γίνεται μια παρουσίαση γενικά των λαβυρίνθων, ενώ στο δεύτερο κεφάλαιο γίνεται μια παρουσίαση των αλγορίθμων κατασκευής λαβυρίνθων, του αλγορίθμου επίλυσης και του ψευδοκώδικά τους. Στο τρίτο κεφάλαιο επιλέγουμε τον λαβύρινθο που θα δημιουργήσουμε, παρουσιάζουμε τις λειτουργίες του και ακολουθεί ο κώδικας σε java. Στο τελευταίο κεφάλαιο γίνεται μια σύγκριση των δύο αλγορίθμων κατασκευής λαβυρίνθων που χρησιμοποιήσαμε.

ΚΕΦΑΛΑΙΟ 1

ΛΑΒΥΡΙΝΘΟΙ – ΚΑΤΗΓΟΡΙΕΣ

Ο κλασσικός λαβύρινθος είναι ένα δισδιάστατο σχέδιο από διαδρόμους και διασταυρώσεις. Στις μέρες μας, και κυρίως λόγω της έλευσης των υπολογιστών, υπάρχουν πάρα πολλά είδη λαβυρίνθων οι οποίοι μπορούν να ταξινομηθούν σε πολλές κατηγορίες. Είναι στην κρίση του καθενός, που ασχολείται με αυτόν τον τομέα, σε πόσες κατηγορίες θα τους ταξινομήσει.

Μια αρκετά ολοκληρωμένη ταξινόμηση δίνεται από τον προγραμματιστή Walter Pullen (Pullen, 2005), σύμφωνα με τον οποίο οι λαβύρινθοι μπορούν να ταξινομηθούν σε 7 διαφορετικές κατηγορίες ανάλογα με τα παρακάτω κριτήρια: την διάσταση, την υπερδιάσταση, την τοπολογία, την ψηφίδωση, την επιλογή της διαδρομής, την υφή και την εστίαση.

Η διάσταση (Dimension) έχει να κάνει με το πόσες διαστάσεις στο χώρο καταλαμβάνει ο λαβύρινθος. Τα είδη των λαβυρίνθων ανάλογα με τη διάσταση είναι:

- 2 διαστάσεων (2D): οι περισσότεροι λαβύρινθοι είτε στην πραγματική ζωή, είτε στο χαρτί, είναι αυτής της διάστασης
- 3 διαστάσεων (3D): αυτοί οι λαβύρινθοι έχουν πολλαπλά επίπεδα και πέρα από τα 4 σημεία του ορίζοντα, οι συμμετέχοντες μπορούν να κινηθούν και πάνω κάτω.
- Παραπάνω διαστάσεων (Higher dimensions): είναι δυνατόν να έχουμε λαβυρίνθους 4 ή και παραπάνω διαστάσεων. Συνήθως είναι

τρισδιάστατοι λαβύρινθοι που μπορούν να μας ταξιδέψουν σε κάποια άλλη διάσταση (πχ χρόνος) μέσω κάποιων πυλών.

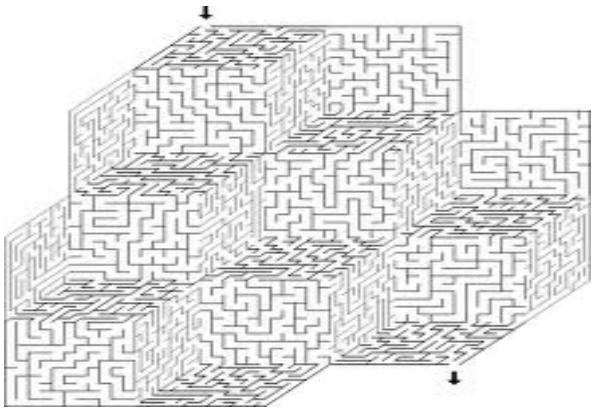
- Ύφανσης (Weave): ένας λαβύρινθος ύφανση είναι ουσιαστικά ένα 2D (ή, ακριβέστερα, ένα 2.5D) λαβύρινθος, με τη διαφορά ότι τα περάσματα μπορούν να επικαλύπτουν το ένα το άλλο. Στο μάτι είναι γενικά προφανές τι είναι το αδιέξοδο και τι είναι το πέρασμα που περνάει κάτω από άλλο πέρασμα. Πραγματικοί λαβύρινθοι της καθημερινής ζωής που έχουν γέφυρες που συνδέουν ένα μέρος του λαβυρίνθου με ένα άλλο είναι εν μέρει ύφανσης.

Υπερδιάσταση (Hyperdimension): αναφέρεται στις διαστάσεις των αντικειμένων που είναι στο λαβύρινθο και όχι στον ίδιο τον λαβύρινθο. Τα είδη αυτά είναι:

- Μη υπερλαβύρινθος: συνήθως όλοι οι λαβύρινθοι, ακόμα και οι αυτοί πολλών διαστάσεων, είναι τέτοιοι λαβύρινθοι. Το αντικείμενο που προσπαθεί να λύσει το λαβύρινθο είναι ένα σημείο και πίσω του αφήνει μια γραμμή.
- Υπερλαβύρινθος: σε αυτούς τους λαβύρινθους το αντικείμενο που λύνει τον λαβύρινθο είναι κάτι παραπάνω από ένα σημείο. Τέτοιος λαβύρινθος μπορεί να υπάρξει σε λαβύρινθο τριών ή και παραπάνω διαστάσεων. Το αντικείμενο που προσπαθεί να λύσει το λαβύρινθο είναι μια γραμμή που αυτό που αφήνει πίσω της διαμορφώνει μια επιφάνεια.

Τοπολογία (Topology): περιγράφει τη γεωμετρία του χώρου του λαβύρινθου και οι τύποι είναι οι εξής:

- Κανονικός: είναι ένας τυποποιημένος λαβύρινθος στην Ευκλείδεια γεωμετρία
- Planair: ο όρος αυτός αναφέρεται σε κάθε λαβύρινθο με περίεργη τοπολογία. Χαρακτηριστικό παράδειγμα είναι ένας λαβύρινθος πάνω σε κύβο.



Ο **όρος tessellation** (ψηφίδωση) αναφέρεται στην ξεχωριστή γεωμετρία των επιμέρους κελιών που συνθέτουν τον λαβύρινθο. Οι διατάξεις είναι:

- Ορθογώνια : Πρόκειται για ένα τυποποιημένο ορθογώνιο πλέγμα, όπου τα κελιά έχουν περάσματα που τέμνονται κάθετα.
- Δέλτα (Delta) : Ένας λαβύρινθος Delta αποτελείται από τρίγωνα που μπλέκονται μεταξύ τους, και το κάθε κελί μπορεί να έχει έως και τρία περάσματα που συνδέονται με αυτό.
- Σίγμα (Sigma) : Ένας τέτοιος λαβύρινθος αποτελείται από εξάγωνα που μπλέκονται μεταξύ τους, και το κάθε κελί μπορεί να έχει έως και έξι περάσματα που συνδέονται με αυτό.
- Θήτα (Theta) : Οι Theta Λαβύρινθοι αποτελούνται από ομόκεντρους κύκλους, το μονοπάτι έναρξης ή λήξης είναι στο κέντρο ενώ το άλλο είναι στην εξωτερική άκρη. Τα κελιά έχουν συνήθως τέσσερα πιθανά περάσματα, αλλά μπορεί να έχουν και περισσότερα λόγω του μεγαλύτερου αριθμού κελιών στα εξωτερικά περάσματα.
- Ύψιλον (Epsilon): Οι Ύψιλον Λαβύρινθοι αποτελούνται από οκτάγωνα και τετράγωνα που μπλέκονται μεταξύ τους, ενώ το κάθε κελί μπορεί να έχει μέχρι οκτώ ή τέσσερα περάσματα που να συνδέονται με αυτό.
- Ζέτα (Zeta): Ο Ζέτα λαβύρινθος είναι σε ένα ορθογώνιο πλέγμα, και εκτός από τα οριζόντια και κάθετα περάσματα επιτρέπονται και τα διαγώνια περάσματα 45 μοιρών.
- Ωμέγα (Omega): Ο όρος "ωμέγα" αναφέρεται σχεδόν σε κάθε λαβύρινθος με μια μη ορθογώνια ψηφίδωση. Οι Delta, Sigma, και Theta Λαβύρινθοι είναι όλοι τέτοιου είδους.

- Σχισματοειδής (Crack): Αυτού του είδους οι λαβύρινθοι είναι άμορφοι, χωρίς καμία συγκεκριμένη ψηφίδωση, αλλά με τοίχους και περάσματα σε τυχαίες γωνίες.
- Μορφοκλασματικός (Fractal) : Πρόκειται για λαβύρινθους που αποτελούνται από μικρότερους λαβύρινθους. Κάθε κελί αυτού του λαβύρινθου είναι ένας άλλος λαβύρινθος και αυτή η διαδικασία μπορεί να επαναληφθεί πολλές φορές. Ένας άπειρος επαναλαμβανόμενος λαβύρινθος είναι ο πραγματικός fractal λαβύρινθος που σε κάθε κελί περιέχει ένα αντίγραφο του εαυτού του.

Δρομολόγηση (Pathfinding): Η δρομολόγηση αναφέρεται στους τύπους των περασμάτων και αφορά οποιαδήποτε από τις παραπάνω κατηγορίες.

- Τέλειος λαβύρινθος : Ένας «τέλειος» λαβύρινθος, είναι χωρίς βρόχους ή κλειστά κυκλώματα, και χωρίς δυσπρόσιτες περιοχές. Επίσης ονομάζεται λαβύρινθος απλής σύνδεσης. Από κάθε σημείο, υπάρχει ακριβώς ένα μονοπάτι προς κάθε άλλο σημείο και ο λαβύρινθος έχει ακριβώς μία λύση. Στην επιστήμη των υπολογιστών μπορεί να περιγραφεί ως ένα επικαλύπτον (spanning) δέντρο πάνω από το σύνολο των κελιών.
- Λαβύρινθος πλεξούδα : Είναι ένας λαβύρινθος που δεν έχει αδιέξοδα. Ονομάζεται επίσης λαβύρινθος πολλαπλής σύνδεσης. Αυτός ο λαβύρινθος χρησιμοποιεί περάσματα που μοιάζουν πλεγμένα και καταλήγουν το ένα στο άλλο. Έτσι μας αναγκάζει να προχωράμε σε κύκλους για αρκετό χρόνο, αντί να πέφτουμε σε αδιέξοδα.
- Μονοκόνδυλος λαβύρινθος : Είναι ο λαβύρινθος που δεν έχει καθόλου διασταυρώσεις. Ο αγγλικός όρος labyrinth αναφέρεται συνήθως σε αυτό το είδος λαβυρίνθου, ενώ ο γενικού τύπου λαβύρινθος ονομάζεται maze.
- Αραιός λαβύρινθος : Ένας αραιός λαβύρινθος δεν έχει πέρασμα από κάθε κελί. Αυτό ισοδυναμεί με το να υπάρχουν δυσπρόσιτες περιοχές, καθιστώντας έτσι αυτό το είδος λαβύρινθου σαν το αντίστροφο του λαβύρινθου πλεξούδας.
- Μερική πλεξούδα: Οι λαβύρινθοι αυτού του είδους είναι απλά μικτοί λαβύρινθοι με βρόχους και αδιέξοδα.

Υφή (Texture): Περιγράφει το στυλ των περασμάτων σε οποιαδήποτε δρομολόγηση ή γεωμετρία. Μερικές παράμετροι που επηρεάζουν την υφή είναι:

- Κλίση (Bias): Όταν ένας λαβύρινθος έχει κλίση τα περάσματά του τείνουν να είναι μεγαλύτερου μήκος ως προς μία διεύθυνση, περισσότερο από τις υπόλοιπες. Αυτός ο λαβύρινθος είναι συνήθως πιο δύσκολος να επιλυθεί ακολουθώντας διεύθυνση διαφορετική από την κλίση του .
- Μήκος (Run): Ο παράγοντας «μήκος» ενός λαβυρίνθου, είναι η μέγιστη απόσταση που μπορεί να διανύσει σε ευθεία ένα μονοπάτι πριν παρουσιαστούν στροφές. Ένας λαβύρινθος με χαμηλό μήκος δε θα χει ευθείες μεγαλύτερες από 3-4 κελιά και θα μοιάζει τελείως τυχαίος. Ένας λαβύρινθος με υψηλό μήκος, θα έχει μακριά ευθεία περάσματα που διανύουν μεγάλο μέρος του λαβυρίνθου, και θα μοιάζει με μικροτσίπ.
- Ελιτισμός (Elite): Ο παράγοντας «ελιτισμός» ενός λαβυρίνθου, υποδηλώνει το μήκος της λύσης σε σχέση με το μέγεθος του λαβυρίνθου. Ένας λαβύρινθος με μεγάλο ελιτισμό έχει γενικά μια σύντομη άμεση λύση, ενώ σ' έναν μη ελιτιστικό λαβύρινθο η λύση περνάει από ένα μεγάλο μέρος του λαβυρίνθου. Ένας καλοσχεδιασμένος ελιτιστικός λαβύρινθος μπορεί να είναι πολύ πιο δύσκολος από έναν μη-ελιτιστικό.
- Συμμετρικός (Symmetric): Ένας συμμετρικός λαβύρινθος έχει συμμετρικά περάσματα, π.χ. συμμετρικά γύρω από ένα κέντρο, ή συμμετρικά σε σχέση με έναν οριζόντιο ή κάθετο άξονα κ.λ.π. Ένας λαβύρινθος μπορεί να είναι μερικώς ή πλήρως συμμετρικός, και μπορεί να επαναλάβει ένα μοτίβο όσες φορές χρειαστεί.
- Ποτάμι (River): Το χαρακτηριστικό «ποτάμι» σημαίνει ότι κατά τη δημιουργία του λαβύρινθου, ο αλγόριθμος θα αναζητήσει και θα καθαρίσει τα κοντινά κελιά (ή τοίχους) γύρω από το τρέχον κελί που δημιουργείται, δηλαδή θα εισρεύσει (εξ ου και ο όρος «ποτάμι») σε μη δημιουργημένα κομμάτια του λαβυρίνθου σαν νερό. Ένας

τέλειος λαβύρινθος με λιγότερα «ποτάμια», θα τείνει να έχει πολλά μικρά αδιέξοδα, ενώ ένας λαβύρινθος με περισσότερα «ποτάμια» θα έχει λιγότερα αλλά πιο μεγάλα αδιέξοδα..

Υπάρχουν και άλλες κατηγοριοποιήσεις λαβυρίνθων. Τρεις επιπλέον κατηγορίες που βρήκαμε είναι οι ακόλουθες (Berg, 2002):

Λαβύρινθος βέλος (arrow maze): Ο βασικός τύπος, στον οποίο κάθε λαβύρινθος μπορεί να κατηγοριοποιηθεί, είναι ο «λαβύρινθος βέλος" (arrow maze). Μπορεί να αναπαρασταθεί από ένα σύνολο από σημεία (points) ή κορυφές (vertices), που συνδέονται με βέλη (arrows). Για την επίλυση του λαβύρινθου κινούμαστε από μια αρχική κορυφή, σε μια τελική, ακολουθώντας την κατεύθυνση των βελών από κορυφή σε κορυφή.

Λαβύρινθος με εμπόδια (Block maze): Ένας λαβύρινθος που δε μπορεί να λυθεί αν δεν καθαριστούν τα μονοπάτια του λαβυρίνθου από κινούμενα τούβλα (εμπόδια) . Όταν σχεδιαστεί σωστά, ακόμα και πολύ μικροί λαβύρινθοι με εμπόδια μπορεί να είναι πολύπλοκο να λυθούν.

Λογικός λαβύρινθος (Logic maze): Ένας λαβύρινθος που πρέπει να οδηγηθεί μέσα από τα μονοπάτια του ακολουθώντας λογικούς κανόνες. Παραδείγματα τέτοιου λαβυρίνθου μπορεί να περιλαμβάνουν έναν λαβύρινθο που περιέχει διαφορετικά χρωματισμένα σύμβολα που πρέπει να «περαστούν» με συγκεκριμένη σειρά, ή ένα λαβύρινθο που έχει μερικά περάσματα που είναι αναγκαστικά μονής κατεύθυνσης (ένας λαβύρινθος βέλος).

Τέλος, στα πλαίσια αυτής της διπλωματικής εργασίας ορίζουμε την παρακάτω υποκατηγορία λογικών λαβυρίνθων :

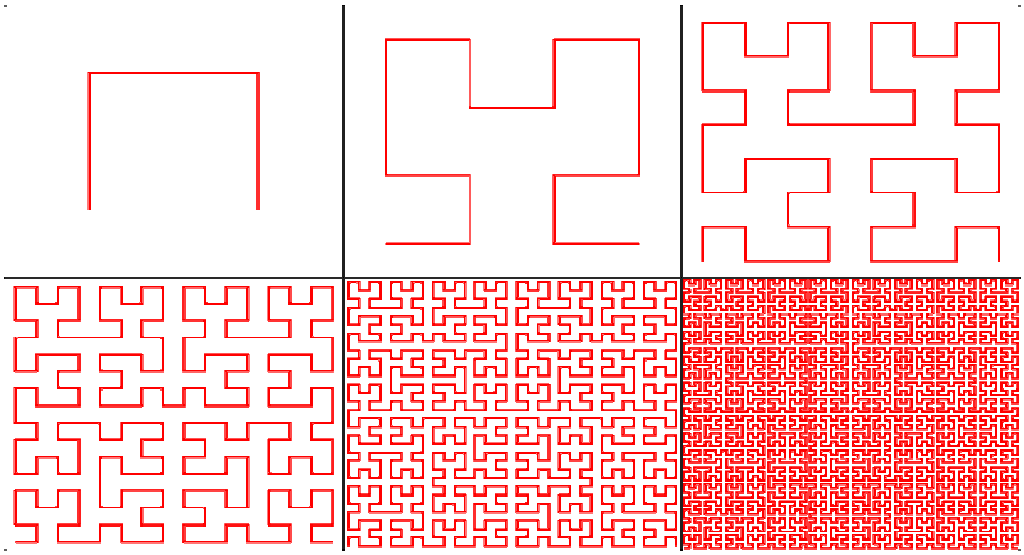
Λαβύρινθος με κλειδιά (Maze with keys): Περιλαμβάνει οποιονδήποτε λαβύρινθο περιέχει ένα ή περισσότερα φραγμένα μονοπάτια, όπου τα φράγματα έχουν την έννοια κλειδωμένων πορτών. Κάθε κλειδωμένη πόρτα ανοίγει με το δικό της κλειδί, το οποίο βρίσκεται σε κάποιο σημείο του λαβυρίνθου. Ο λύτης πρέπει να περάσει πρώτα από το κλειδί, ώστε να μπορεί να ανοίξει την πόρτα για να φτάσει τελικά στην έξοδο.

ΚΕΦΑΛΑΙΟ 2

ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΑΣΚΕΥΗΣ ΚΑΙ ΕΠΙΛΥΣΗΣ ΛΑΒΥΡΙΝΘΩΝ

ΚΑΤΑΣΚΕΥΗ ΛΑΒΥΡΙΝΘΩΝ

Ένας τρόπος σχεδίασης τυχαίων λαβυρίνθων με αλγόριθμους βασίζεται στις καμπύλες γεμίσματος του χώρου (Space-filling curves) (Sagan, 1994) (Pedersen and Singh, 2006). Η έννοια αυτή ξεκίνησε ως μαθηματική από τον Giuseppe Peano το 1890, αλλά η προσέγγιση του D. Hilbert, τον επόμενο χρόνο, ήταν γεωμετρική και δημιούργησε την καμπύλη Hilbert (Hilbert curve). Η τελευταία είναι μια συνεχής μορφοκλασματική (fractal) καμπύλη γεμίσματος του χώρου. Η καμπύλη αυτή είναι μονοδιάστατη και επισκέπτεται όλα τα σημεία ενός δισδιάστατου χώρου. Μπορεί να θεωρηθεί ως το όριο μιας ακολουθίας από καμπύλες που διαγράφονται μέσα στο χώρο. Η έννοια των καμπυλών αυτών είναι πολύπλοκη και περιλαμβάνει πολλούς μαθηματικούς όρους που δεν είναι σχετικοί με το αντικείμενο της εργασίας. Ωστόσο παραθέτω παρακάτω μερικές εικόνες που θα βοηθήσουν στην κατανόησή τους.



Εικ.1 - Στάδια γεμίσματος χώρου της καμπύλης Hilbert

Οι αλγόριθμοι δημιουργίας λαβυρίνθων που βασίζονται στα γραφήματα, δημιουργούν λαβύρινθους χτίζοντας ένα ελάχιστο επικαλύπτον

δέντρο, που είναι ένα είδος γραφήματος. Στο τέλος της διαδικασίας έχει δημιουργηθεί ένα δέντρο που τοπολογικά ανταποκρίνεται σε έναν λαβύρινθο. Υπάρχουν 2 βασικοί τρόποι δημιουργίας λαβυρίνθων, προσθέτοντας τοίχους, ή δημιουργώντας περάσματα.

Υπάρχουν διάφοροι αλγόριθμοι δημιουργίας λαβυρίνθων (Levitin, 2003). Ένας λαβύρινθος είναι βασικά μια γραφική παράσταση που παρουσιάζει μια διάβαση μεταξύ δύο σημείων. Εάν ο γράφος περιέχει βρόχους, τότε μπορεί να υπάρξουν πολλαπλά μονοπάτια μεταξύ των επιλεγμένων διαδρομών. Λόγω αυτού, η δημιουργία ενός λαβυρίνθου προσεγγίζεται συχνά σαν την δημιουργία ενός τυχαίου δέντρου μέσα σε ένα συνδεδεμένο γράφο.

Αλγόριθμος Prim

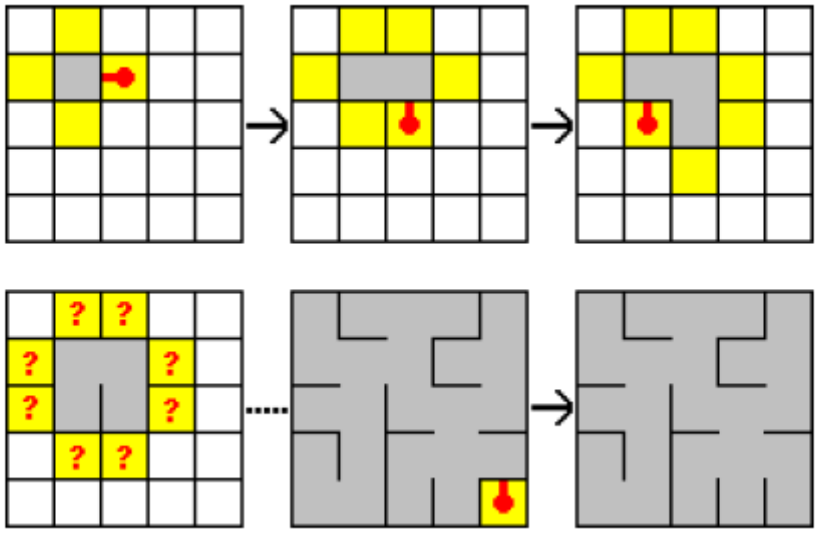
Ο αλγόριθμος Prim (Thomas H. Cormen, 2009) είναι μια μέθοδος εύρεσης ενός ελάχιστα απλωμένου δέντρου (minimum spanning tree - MST) σε μη κατευθυνόμενο γράφημα που βασίζεται στη λογική της απληστίας (greedy). Ο όρος αυτός χαρακτηρίζει την τακτική της επιλογής σε κάθε βήμα της καλύτερης εναλλακτικής, χωρίς προσπάθεια πρόβλεψης των επόμενων βημάτων. Ο αλγόριθμος έχει ως εξής:

1. Επιλέγεται μια κορυφή του γραφήματος ως αρχική, χαρακτηρίζεται ως εξερευνημένη και τοποθετείται στο εξερευνημένο δέντρο ως ρίζα.
2. Εξετάζονται όλες οι ακμές που είναι γείτονες του μερικού δέντρου. Η ακμή με το μικρότερο βάρος, προστίθεται στο δέντρο μαζί με την κορυφή στην οποία καταλήγει.
3. Η διαδικασία αυτή καταλήγει έως ότου προσθέσουμε στο μερικό δέντρο όλες τις κορυφές του γραφήματος. Τότε ο αλγόριθμος τερματίζει έχοντας δημιουργήσει ένα ελάχιστο επικαλύπτον δέντρο.

Για την κατασκευή ενός λαβυρίνθου πιο συγκεκριμένα ο αλγόριθμος λειτουργεί ως εξής:
Κατά τη διάρκεια της δημιουργίας του λαβυρίνθου κάθε κελί μπορεί να ανήκει σε μία από τις παρακάτω κατηγορίες:

-«εσωτερικό»: είναι το κελί που έχει ήδη δημιουργηθεί και αποτελεί μέρος του λαβυρίνθου.
-«συνορεύον» ή «γειτονικό»: είναι το κελί που δεν έχει ακόμα δημιουργηθεί, αλλά γειτονικό σε ένα εσωτερικό κελί που ήδη έχει δημιουργηθεί.
-«εξωτερικό»: είναι το κελί που δεν έχει ακόμα δημιουργηθεί, ούτε είναι γειτονικό σε κάποιο κελί (συνορεύον).

Ο αλγόριθμος ξεκινάει επιλέγοντας ένα κελί που το μετατρέπει σε εσωτερικό και μετατρέποντας όλα τα γειτονικά του σε «συνορεύον». Ο λαβύρινθος έχει δημιουργηθεί όταν δεν υπάρχουν πια άλλα συνορεύοντα κελιά (που σημαίνει ότι δεν υπάρχουν και άλλα «εξωτερικά» κελιά, οπότε όλα είναι εσωτερικά.



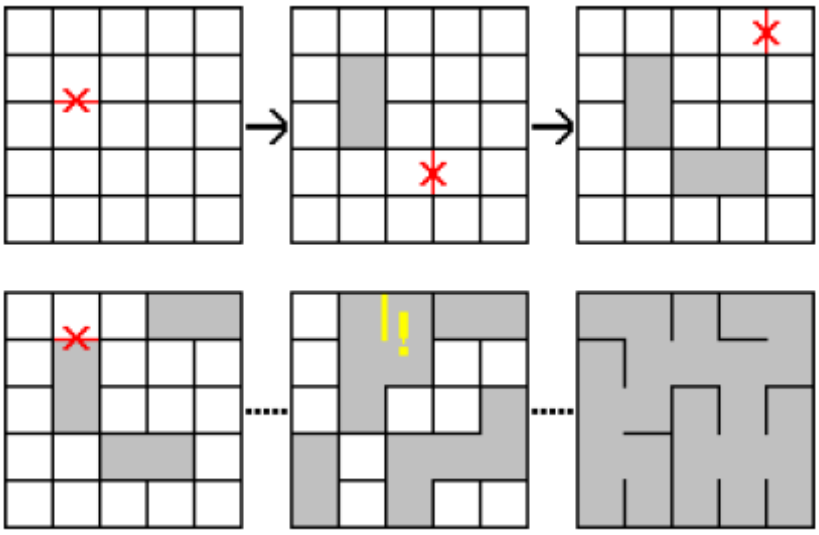
Αλγόριθμος Kruskal

Ο αλγόριθμος Kruskal (Roberts & Tesman, 2009) είναι μια μέθοδος για την εύρεση ενός ελάχιστου επικαλύπτοντος δέντρου σε μη κατευθυνόμενο γράφο. Έχει ως ακολούθως:

1. Εξετάζει τις ακμές μία προς μία κατά αύξουσα τιμή βάρους και τις προσθέτει εφόσον δε δημιουργείται κύκλος.
2. Ξεκινά με V στοιχειώδη δέντρα, τα οποία εν τέλει με τις διαδοχικές προσαρτήσεις ακμών, ενοποιούνται στο τελικό δέντρο.

Ο αλγόριθμος αυτός παρουσιάζει ενδιαφέρον, καθώς δε μεγαλώνει το λαβύρινθο όπως ένα δέντρο, αλλά δημιουργεί τυχαία περάσματα σε όλον τον λαβύρινθο, καταλήγοντας να φτιάξει έναν τέλειο λαβύρινθο.

Ο αλγόριθμος Kruskal είναι όπως και ο αλγόριθμος Prim είναι άπληστος αλγόριθμος που μπορεί να βρει το ελάχιστο επικαλύπτον δέντρο μέσα σε ένα γράφο. Η διαφορά είναι η περιοχή μέσα στην οποία «ψάχνει» ο αλγόριθμος. Ο αλγόριθμος Prim όπως περιγράφηκε παραπάνω φτιάχνει το επικαλύπτον δέντρο από ένα σημείο με έναν επεκτατικό τρόπο. Από την άλλη ο αλγόριθμος Kruskal λειτουργεί σε ολόκληρη την περιοχή του γράφου δημιουργώντας και συνδέοντας κομμάτια δέντρων που δεν είναι κυκλικά μεταξύ τους. Τελικά με την προσθήκη της τελευταίας ακμής δημιουργείται το δέντρο.



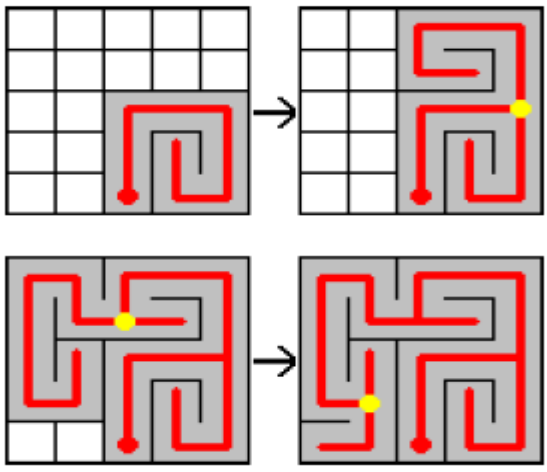
Αλγόριθμος Αναδρομικής Οπισθοδρόμησης (Recursive Backtracking)

Ο αλγόριθμος αναδρομικής οπισθοδρόμησης (Roberts, 2006) είναι στην ουσία ένας αλγόριθμος αναζήτησης πρώτα σε βάθος. Ο αλγόριθμος «περιπλανιέται» μέσα στο γράφημα με προσανατολισμό κατά βάθος. Αυτό σημαίνει ότι αν είναι δυνατόν η επόμενη κορυφή που συναντάει ο αλγόριθμος είναι παιδί της προηγούμενης κορυφής που επισκέφτηκε ο αλγόριθμος. Διαφορετικά αν η κορυφή που συνάντησε τελευταία ο αλγόριθμος δεν έχει διαθέσιμα παιδιά, ο αλγόριθμος επιστρέφει στην κορυφή που επισκέφτηκε προηγούμενα και προσπαθεί ξανά.

Ο αλγόριθμος τελειώνει όταν έχει επισκεφτεί όλες τις κορυφές. Αν ο αλγόριθμος κατά τη διαδρομή που κάνει, δημιουργεί ένα δέντρο, τότε αυτό το δέντρο είναι το ελάχιστο επικαλύπτον δέντρο.

Ο αλγόριθμος έχει ως εξής:

1. Διάλεξε ένα σημείο έναρξης στο πεδίο έρευνας
2. Διάλεξε τυχαία έναν τοίχο και σε αυτό το σημείο φτιάξε ένα πέρασμα προς το γειτονικό κελί, αλλά όνο αν το παρακείμενο κελί δεν έχει ήδη επισκεφθεί ακόμα.
3. Αν όλα τα παρακείμενα κελιά έχουν επισκεφθεί ήδη, οπισθοχώρησε στο τελευταίο κελί που έχει άχτιστους τοίχους και επανέλαβε
4. Ο αλγόριθμος τελειώνει όταν η διαδικασία έχει γυρίσει πίσω μέχρι το σημείο έναρξης.



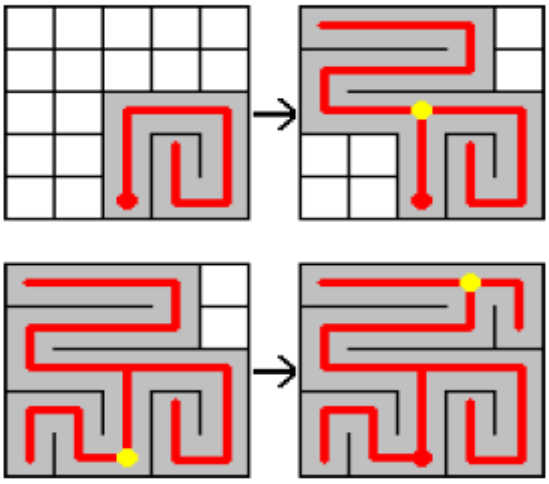
Αλγόριθμος Hunt and Kill

Ο αλγόριθμος αυτός είναι καλός γιατί δεν απαιτεί πρόσθετη μνήμη για αποθήκευση και έτσι μπορεί να χρησιμοποιηθεί για τη δημιουργία μεγάλων λαβυρίνθων σε συστήματα με περιορισμένες δυνατότητες. Πρόκειται για μια παραλλαγή της αναδρομικής οπισθοδρόμησης.

Όταν ο αλγόριθμος φτάνει σε κάποιο κελί που δεν έχει διαθέσιμα γειτονικά κελιά για να επισκεφτεί και να δημιουργήσει πέρασμα, δεν οπισθοδρομεί στο προηγούμενο κελί. Αντί για αυτό, ο αλγόριθμος βρίσκει ένα κελί που δεν έχει επισκεφτεί κι έχει διαθέσιμους γειτονικά κελιά και η δημιουργία περάσματος ξεκινάει από εκείνο το σημείο. Ο λαβύρινθος έχει δημιουργηθεί όταν όλα τα κελιά έχουν «σαρωθεί» πάνω από μια φορά.

Ο αλγόριθμος έχει ως εξής:

1. Διάλεξε μια τοποθεσία έναρξης.
2. Κάνε μια τυχαία βόλτα και δημιούργησε περάσματα προς τα γειτονικά κελιά που δεν έχουν επισκεφθεί ακόμα, μέχρι το τρέχον κελί να μην έχει γειτονικά κελιά που δεν έχουν επισκεφθεί.
3. Βάλε τη λειτουργία του «κυνηγιού», κατά την οποία σαρώνεις το πλέγμα ψάχνοντας για ένα κελί που δεν το έχει επισκεφτεί και είναι γειτονικό σε κάποιο κελί που έχεις επισκεφτεί. Αν βρεθεί, δημιούργησε ένα πέρασμα μεταξύ αυτών των δύο και όρισε το προηγούμενο κελί σαν νέο σημείο έναρξης.
4. Επανέλαβε τα βήματα 2 και 3 μέχρι η λειτουργία του «κυνηγιού» να σαρώσει ολόκληρο το πλέγμα και να μη βρει κελιά που να μην έχουν επισκεφθεί.

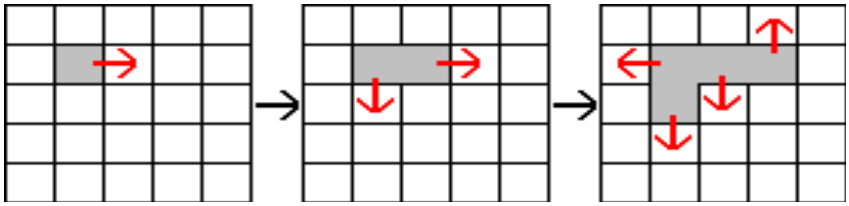


Αλγόριθμος Bacterial Growth

Ο αλγόριθμος αυτός είναι παρόμοιος με τον αλγόριθμο Prim καθώς και οι δύο φτιάχνουν το ελάχιστο επικαλύπτον δέντρο ξεκινώντας από ένα σημείο. Η διαφορά είναι στην πυκνότητα που προστίθενται νέες ακμές στο δέντρο. Ο αλγόριθμος Prim προσθέτει μια ακμή σε κάθε βήμα, ενώ ο αλγόριθμος bacterial growth προσθέτει μία ακμή σε κάθε κελί με διαθέσιμους γείτονες. Με αυτόν τον τρόπο ο αριθμός των επισκεπτόμενων κελιών μπορεί να διπλασιάζεται (τουλάχιστον). Κατά πάσα πιθανότητα γι' αυτό το λόγο αυτός ο αλγόριθμος έχει και το συγκεκριμένο όνομα που εκφράζει την αύξηση σε αυξημένους ρυθμούς (όπως ο πληθυσμός των βακτηρίων).

Ο αλγόριθμος έχει ως εξής:

1. Διάλεξε τυχαία ένα κελί σα σημείο έναρξης και βάλτο στη λίστα με τα χρησιμοποιημένα κελιά.
2. Ανακάτεψε τη λίστα – δηλαδή τοποθέτησε τα στοιχεία της σε τυχαία σειρά (αν η λίστα είναι άδεια, έχουμε τελειώσει).
3. Για κάθε κελί στη λίστα, αν έχει μη χρησιμοποιημένους γείτονες, βγάλτο απ΄ τη λίστα..
4. Διαφορετικά διάλεξε τυχαία ένα μη χρησιμοποιημένο γείτονα αυτού του κελιού, αφαίρεσε τον τοίχο ανάμεσα τους και πρόσθεσε το νέο κελί στη λίστα..
5. Επέστρεψε στο βήμα 2.



Αλγόριθμος του Eller:

Πρόκειται για έναν ενδιαφέρον αλγόριθμο καθώς είναι ένας απ΄ τους πιο γρήγορους στη δημιουργία αλγορίθμων και αποδοτικός με την κατανομή μνήμης. Η βασική αρχή είναι ότι μία σειρά σαρώνεται κάθε φορά.

Κάθε κύτταρο στη σειρά είναι μέρος ενός συνόλου, όπου δύο κελιά βρίσκονται στο ίδιο σύνολο, αν υπάρχει ένα μονοπάτι μεταξύ τους μέσα από το τμήμα του λαβυρίνθου που έχει γίνει μέχρι στιγμής. Η πληροφορία αυτή επιτρέπει να δημιουργούνται περάσματα στην τρέχουσα γραμμή χωρίς τη δημιουργία βρόχων. Είναι παρόμοιος με τον αλγόριθμο του Kruskal, μόνο που αυτός ολοκληρώνει μία γραμμή κάθε φορά, ενώ Kruskal ψάχνει σε όλο το λαβύρινθο. Η δημιουργία μιας σειράς αποτελείται από δύο μέρη: Τυχαία σύνδεση γειτονικών κελιών μέσα σε μια γραμμή, δηλαδή δημιουργία οριζόντιων περασμάτων, στη συνέχεια, τυχαία σύνδεση των κελιών μεταξύ της τρέχουσας σειράς και της επόμενης σειράς, δηλαδή, δημιουργία κάθετων περασμάτων.

Ο αλγόριθμος έχει ως εξής:

1. Αρχικοποίησε τα κελιά της πρώτης γραμμής έτσι ώστε το καθένα να ανήκει σε ένα σύνολο από μόνο του.
2. Τυχαία ένωσε τα γειτονικά κελιά, αλλά μόνο αν δεν ανήκουν στο ίδιο σύνολο. Κατά την ένωση των γειτονικών κελιών, συγχώνευσε τα κελιά των δύο συνόλων σε ένα σύνολο, δηλώνοντας ότι όλα τα κελιά και στα δύο σύνολα είναι συνδεδεμένα (υπάρχει ένα μονοπάτι που συνδέει τα κελιά ανά δύο).
3. για κάθε σύνολο δημιούργησε τυχαία κάθετες συνδέσεις προς τα κάτω στην επόμενη γραμμή. Κάθε εναπομείναν σύνολο πρέπει να έχει τουλάχιστον μία κάθετη σύνδεση. Έτσι τα κελιά στην κάτω γραμμή

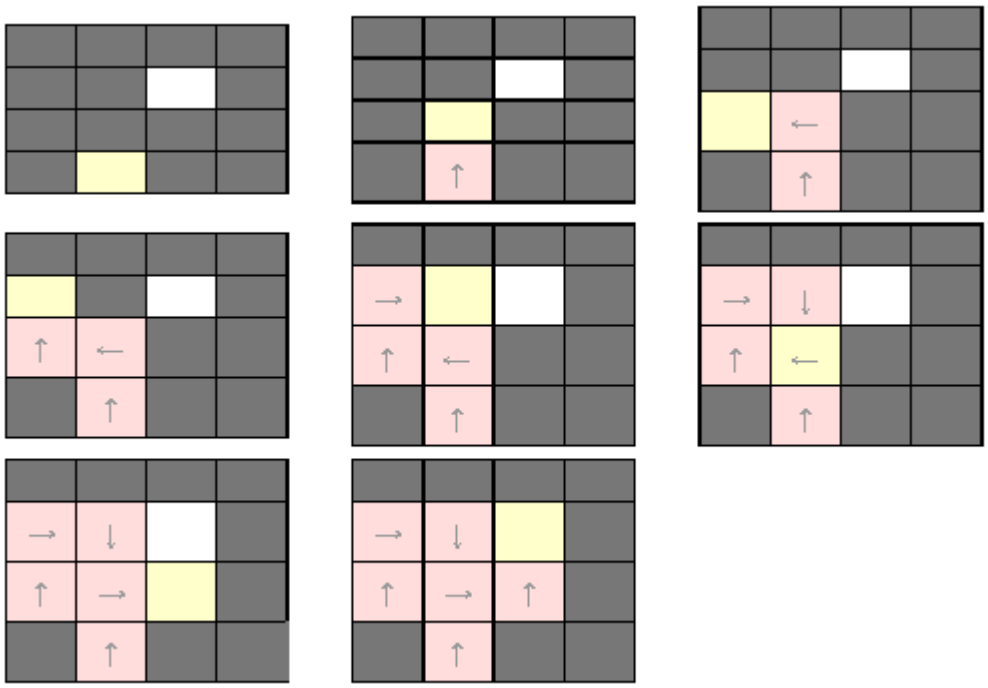
πρέπει να μοιράζονται τα σύνολα των κελιών στην προηγούμενη γραμμή.
4. Ενίσχυσε την επόμενη γραμμή βάζοντας κάθε εναπομείναν κελί στα δικά τους σύνολα.
5. Επανέλαβε μέχρι να φτάσεις στην τελευταία σειρά
6. Για την τελευταία σειρά, ένωσε όλα τα γειτονικά κελιά έτσι ώστε να μην είναι στο ίδιο σύνολο και παρέλειψε τις κάθετες συνδέσεις.

Αλγόριθμος Wilson

Πρόκειται για μια βελτιωμένη έκδοση του αλγόριθμου Aldous-Broder με την έννοια ότι δημιουργεί λαβυρίνθους ακριβώς όπως αυτός ο αλγόριθμος, αλλά είναι πολύ πιο γρήγορος. Με αυτόν τον αλγόριθμο πραγματοποιείται μια τυχαία διαδρομή μεταξύ δύο τυχαία επιλεγμένων κελιών. Στη συνέχεια ένα τρίτο κελί (εκτός του δημιουργημένου μονοπατιού) επιλέγεται σαν σημείο έναρξης για μια ακόμα τυχαία διαδρομή που τελειώνει όταν προσκρούει σε ένα κελί που είναι ήδη μέρος του λαβυρίνθου. Αυτό το κελί γίνεται ρίζα ενός νέου υποδέντρου μέσα στο τελικό επικαλύπτον δέντρο.

Ο αλγόριθμος είναι κάπως έτσι:

1. Διάλεξε οποιαδήποτε κορυφή τυχαία και πρόσθεσε τη στο δέντρο.
2. Διάλεξε οποιαδήποτε κορυφή που δεν είναι ήδη στο δέντρο και κάνε μια τυχαία διαδρομή μέχρι να συναντήσεις μια κορυφή που είναι ήδη στο δέντρο.
3. Πρόσθεσε στο δέντρο τις κορυφές και τις γωνίες που πέρασες στην τυχαία βόλτα.
4. Επανέλαβε το βήμα 2 και 3 μέχρι όλες οι κορυφές να έχουν προστεθεί στο δέντρο.



Αλγόριθμος αναδρομικής διαίρεσης:

Είναι ένας αλγόριθμος που προσθέτει τοίχους.

Ξεκινάει με τη δημιουργία ενός τυχαίου οριζόντιου ή κάθετου τοίχου που διασχίζει τη διαθέσιμη περιοχή σε μια τυχαία σειρά ή μια στήλη, με ένα άνοιγμα που τοποθετείται τυχαία κατά μήκος του. Κατόπιν επαναλαμβάνει τη διαδικασία στις δύο υποπεριοχές που παράγονται από τον τοίχο διαίρεσης.

Ο αλγόριθμος είναι ως εξής:

1. Ξεκίνα με ένα κενό πεδίο.
2. Διαίρεσε το πεδίο με έναν τοίχο είτε οριζόντια, είτε κάθετα. Πρόσθεσε ένα πέρασμα μέσα από τον τοίχο.
3. Επανέλαβε το βήμα 2 με τις περιοχές σε κάθε πλευρά'α του τοίχου.
4. Συνέχισε αναδρομικά μέχρι ο λαβύρινθος να φτάσει στο επιθυμητό σημείο.

Αλγόριθμος Aldous – Broder

Είναι ένας αλγόριθμος παραγωγής ομοιόμορφα απλωμένων δέντρων (uniform spanning trees) (Aldous, 1990) (Broder, 1989). Απλωμένο δέντρο, θεωρείται όποιο δέντρο ενώνει όλες τις κορυφές ενός γράφου. Ένα απλωμένο

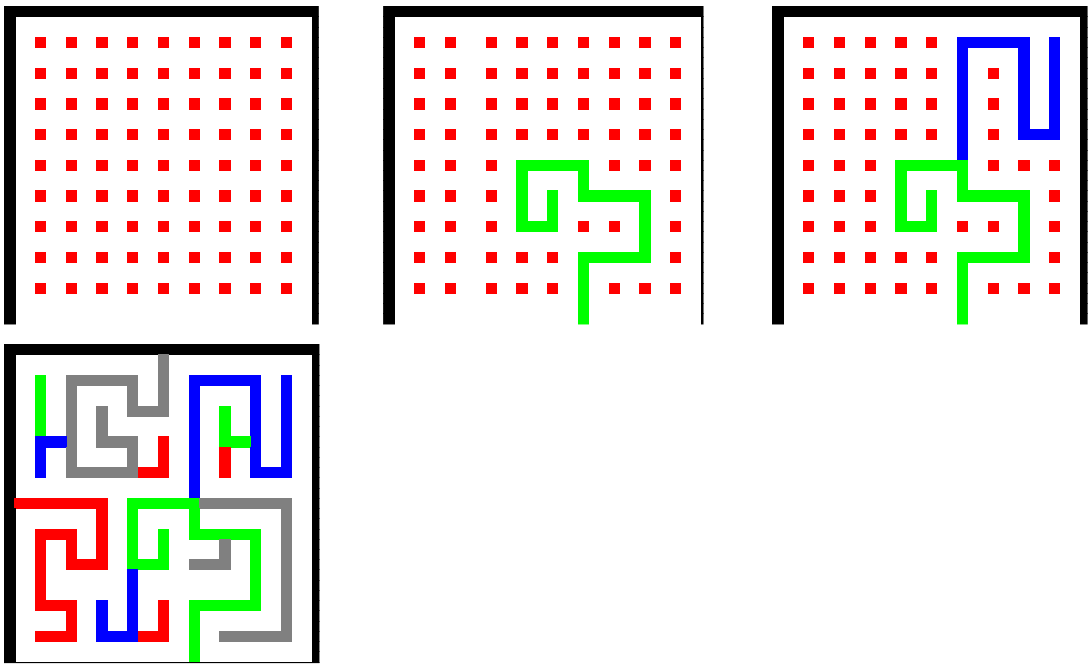
δέντρο που επιλέγεται τυχαία μέσα απ' όλα τα πιθανά απλωμένα δέντρα ενός γράφου, με ίση πιθανότητα, ονομάζεται ομοιόμορφα απλωμένο δέντρο. Το ενδιαφέρον με αυτόν τον αλγόριθμο είναι ότι δημιουργεί όλους τους δυνατούς λαβυρίνθους ενός δεδομένου μεγέθους με ίση πιθανότητα. Επίσης, δεν απαιτεί επιπλέον αποθήκευση ή στοίβα. Διαλέγει ένα σημείο, και προχωράει σε ένα γειτονικό κελί τυχαία. Αν βρεθεί σ' ένα μη χαραγμένο κελί, το χαράζει και προχωράει παρακάτω. Συνεχίζει να κινείται σε γειτονικά κελιά μέχρι να χαραχτούν όλα τα κελιά. Το κακό με αυτόν τον αλγόριθμο είναι ότι είναι πολύ αργός, δεδομένου ότι δεν κάνει έξυπνο κυνήγι για τα τελευταία κελιά, ενώ στην πραγματικότητα δεν είναι καν σίγουρο ότι θα τερματίσει.

Η υλοποίηση του είναι πολύ απλή και γρήγορη, οι λαβύρινθοι που δημιουργεί όμως, έχουν μια καθορισμένη μορφή. Για κάθε κελί χαράζεται ένα πέρασμα είτε προς τα πάνω είτε προς τα αριστερά, αλλά όχι και προς τις δύο κατευθύνσεις. Στην έκδοση που προστίθενται και τοίχοι, για κάθε κορυφή προσθέτει ένα κομμάτι τοίχου προς τα δεξιά ή προς τα κάτω αλλά όχι και προς τις δύο κατευθύνσεις. Κάθε κελί είναι ανεξάρτητο από το άλλο και έτσι δε χρειάζεται να αναφέρεται στην κατάσταση των άλλων κελιών όταν τα δημιουργεί. Οι λαβύρινθοι δυαδικών δέντρων είναι διαφορετικοί από τους τέλειους λαβύρινθους καθώς πολλοί τύποι κελιών δε μπορούν να υπάρχουν σε αυτούς. Οι λαβύρινθοι αυτοί τείνουν να έχουν περάσματα από πάνω αριστερά προς τα κάτω δεξιά.

Οι αλγόριθμοι κατασκευής που χρησιμοποιήσαμε βασίζονται σ' αυτόν τον αλγόριθμο. Είναι παραλλαγές που έγιναν από έλληνες προγραμματιστές και είναι αρκετά βελτιωμένες σε σχέση με τον πρωτότυπο. Τα ελληνικά ονόματα που τους δόθηκαν είναι «Αλγόριθμος με τοίχους» και «Αλγόριθμος με δέντρο» (Γκέσος, 2006). Παρακάτω φαίνονται οι τρόποι λειτουργίας τους.

Αλγόριθμος με τοίχους

1. Σχεδίασε το πλαίσιο του λαβύρινθου.
2. Ξεκίνα από τοίχο και χωρίς να τέμνεις πουθενά αλλού τοίχο, φτιάξε ένα τυχαίο τοίχο μέχρις ότου να μην μπορείς να συνεχίσεις.
3. Επανέλαβε το βήμα #2 μέχρι ο λαβύρινθος να ολοκληρωθεί.

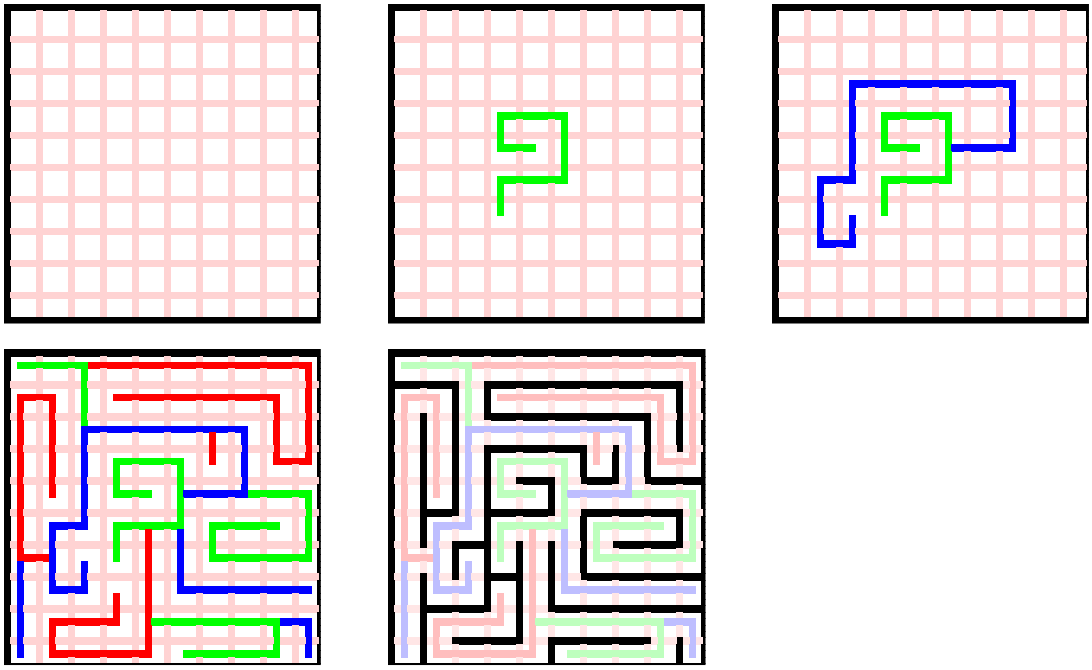


Ο κώδικας του λαβύρινθου χρησιμοποιεί 2 arrays. Ένα array για τους οριζόντιους τοίχους και ένα για τους κατακόρυφους. Κάθε στοιχείο του array αποθηκεύει αν υπάρχει ή όχι τοίχος στη συγκεκριμένη θέση.
Το array των οριζόντιων τοίχων έχει διαστάσεις (κατακόρυφοι διάδρομοι) * (οριζόντιοι διάδρομοι + 1). Το array των κατακόρυφων τοίχων έχει διαστάσεις (κατακόρυφοι διάδρομοι + 1) * (οριζόντιοι διάδρομοι).

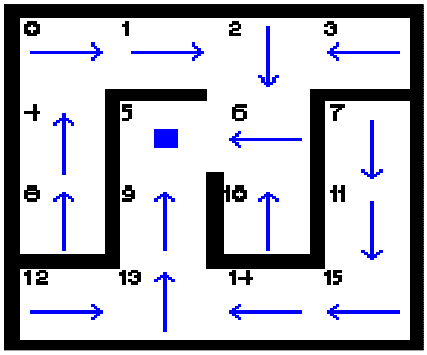
Αλγόριθμος με δέντρο

Τον αλγόριθμο αυτόν ανέπτυξε ο Γκέσος Παύλος. Αυτή τη φορά δε μας ενδιαφέρουν οι τοίχοι αλλά τα κελιά.

1. Την πρώτη φορά ξεκινάμε από ένα οποιοδήποτε κελί. Αν δεν είναι η πρώτη φορά τότε ξεκινάμε από κελί που έχουμε ήδη περάσει.
2. Χαράζουμε ένα τυχαίο διάδρομο από κελί σε κελί, χωρίς να περάσουμε από κελί από το οποίο ξαναπεράσαμε.
3. Αν δεν μπορούμε να συνεχίσουμε άλλο τη χάραξη διαδρόμου επαναλαμβάνουμε τα βήματα #1 και #2 μέχρι να ολοκληρωθεί ο λαβύρινθος.



Ουσιαστικά χρειαζόμαστε ένα array με στοιχεία όλα τα κελιά του λαβύρινθου (διαστάσεις (Οριζόντιοι διάδρομοι) * (Κατακόρυφοι διάδρομοι)). Στην πραγματικότητα φτιάχνουμε ένα δέντρο. Η ρίζα του δέντρου είναι το κελί εκκίνησης. Από εκεί φεύγει ένας διάδρομος (ο κορμός) που αρχίζει και διακλαδίζεται (κλαδιά και φύλλα). Σε κάθε κελί του array αποθηκεύουμε την κατεύθυνση του κελιού από το οποίο ήλθαμε (όπως στην παρακάτω εικόνα).



Εικ.2 - Κατευθύνσεις κελιών

Η ταχύτητα κατασκευής του δέντρου είναι παρόμοια με του προηγούμενου αλγόριθμου με τους τοίχους.

Αλγόριθμος αυξάνοντος δέντρου (growing tree)

Είναι ένας γενικός αλγόριθμος, που μπορεί να δημιουργήσει λαβύρινθους με διαφορετικές υφές. Απαιτεί μνήμη όσο και το μέγεθος του λαβυρίνθου. Κάθε φορά που χαράσσεται ένα κελί, προσθέτει αυτό το κελί σε μια λίστα. Συνεχίζει επιλέγοντας ένα κελί από τη λίστα, και το χαράζει μέσα σε ένα γειτονικό κελί που δεν έχει χαραχτεί. Αν δεν υπάρχουν μη χαραγμένα γειτονικά κελιά δίπλα στο τρέχον κελί, αφαιρεί το τρέχον κελί από τη λίστα. Ο λαβύρινθος δημιουργείται όταν η λίστα είναι κενή.

Ο αλγόριθμος έχει ως εξής:

1. Έστω C μια λίστα κελιών τελείως άδεια. Πρόσθεσε ένα κελί σε αυτή τη λίστα τυχαία.
2. Διάλεξε ένα κελί από τη λίστα και χάραξε ένα πέρασμα σε ένα οποιοδήποτε γειτονικό κελί που δεν έχει επισκεφθεί, προσθέτοντας αυτό το κελί στη λίστα C. Αν δεν υπάρχουν τέτοια γειτονικά κελιά, αφαίρεσε αυτό το κελί από τη λίστα.
3. Επανέλαβε το 2 μέχρι η λίστα C να αδειάσει.

ΕΠΙΛΥΣΗ ΛΑΒΥΡΙΝΘΩΝ

Dead end filler

Είναι ένας απλός αλγόριθμος επίλυσης. Εστιάζει στο λαβύρινθο, είναι πάντα γρήγορος και δε χρησιμοποιεί επιπλέον μνήμη. Απλά σκανάρει τον λαβύρινθο και γεμίζει κάθε αδιέξοδο. Συνεχίζει εκ νέου αυτή τη διαδικασία με τα νέα αδιέξοδα που έχουν δημιουργηθεί και έτσι στο τέλος μένει μόνο η λύση, ή οι λύσεις αν έχει παραπάνω από μία. Με αυτόν τον αλγόριθμο μπορεί να βρεθεί η μοναδική λύση σε έναν τέλειο λαβύρινθο, αλλά σε μεγάλους λαβυρίνθους, ή λαβυρίνθους χωρίς αδιέξοδα δε μπορεί να βοηθήσει.

Wall follower

Πρόκειται επίσης για απλό λαβύρινθο που εστιάζει σε αυτόν που τον χρησιμοποιεί. Είναι επίσης γρήγορος και δε χρησιμοποιεί επιπλέον μνήμη. Ξεκινάει και ακολουθεί τα περάσματα και κάθε φορά που φτάνει σε διασταύρωση στρίβει δεξιά ή αριστερά. Αν θέλει αυτός που χρησιμοποιεί τον αλγόριθμο μπορεί να μαρκάρει τα κελιά που έχει επισκεφτεί και τα κελιά που έχει επισκεφτεί 2 φορές και στο τέλος να βρει τη λύση πισωγυρίζοντας στα κελιά που έχει επισκεφτεί μία φορά. Αυτή η μέθοδος δε βρίσκει απαραίτητα την πιο κοντινή λύση και δε δουλεύει καθόλου αν η λύση βρίσκεται στο κέντρο του λαβυρίνθου και υπάρχει γύρω της ένα κλειστό κύκλωμα, καθώς θα τριγυρνάει γύρω από το κέντρο και στο τέλος θα γυρίζει ξανά στην αρχή.

Recursive backtracker

Ο αλγόριθμος αναδρομικής οπισθοδρόμησης (Roberts E. S., 2006) βρίσκει λύση, αλλά όχι απαραίτητα την πιο κοντινή. Εστιάζει σε αυτόν που τον χρησιμοποιεί, είναι γρήγορος για όλους τους τύπους των λαβυρίνθων και χρησιμοποιεί στοίβα για να αποθηκεύει το μέγεθος του λαβυρίνθου. Πολύ απλά: αν βρίσκεσαι σε ένα τοίχο επέστρεψε αποτυχία, αλλιώς αν είσαι στο τέλος επέστρεψε επιτυχία, αλλιώς προσπάθησε αναδρομικά να κινηθείς προς τις τέσσερις κατευθύνσεις. Σχεδίασε μια γραμμή όταν κατευθύνεσαι προς νέα κατεύθυνση και σβήσε τη γραμμή όταν επιστρέφει αποτυχία. Μια μοναδική λύση θα επιστραφεί όταν είσαι στο σημείο της επιτυχίας. Κατά τη διάρκεια της

οπισθοδρόμησης είναι καλύτερο να μαρκάρεται το διάστημα με μια ειδική τιμή «visited», έτσι ώστε να μην επισκεφτεί ξανά το ίδιο κελί.

Αλγόριθμος Trémaux

Αυτή η μέθοδος επίλυσης είναι σχεδιασμένη για να μπορεί να χρησιμοποιείται από έναν άνθρωπο που βρίσκεται μέσα στο λαβύρινθο. Είναι παρόμοια με την οπισθοδρομική αναζήτηση και βρίσκει λύση για όλους τους λαβυρίνθους. Όπως περπατάς μέσα σε ένα πέρασμα, ζωγράφισε μια γραμμή πίσω σου, σα μονοπάτι. Όταν πέσεις πάνω σε αδιέξοδο γύρνα πίσω και προχώρησε προς τα πίσω από το δρόμο που ήρθες. Όταν βρεθείς σε διασταύρωση που δεν έχεις επισκεφτεί ξανά, φέρσου σα να έπεσες πάνω σε αδιέξοδο και διάλεξε ένα νέο πέρασμα στην τύχη. (Το τελευταίο βήμα είναι το κλειδί που σε εμποδίζει να κάνεις κύκλους). Αν περπατάς μέσα σε ένα πέρασμα που έχεις επισκεφτεί ξανά και συναντήσεις μια διασταύρωση, ακολούθησε ένα μονοπάτι που είναι διαθέσιμο (αν υπάρχει), αλλιώς ακολούθησε ένα νέο μονοπάτι. Όλα τα μονοπάτια θα είναι ή άδεια (που δεν τα έχουμε ακόμα επισκεφτεί), μαρκαρισμένα (που τα έχουμε επισκεφτεί μία φορά), ή μαρκαρισμένα 2 φορές που σημαίνει ότι τα επισκεφτήκαμε ξανά και αναγκαστήκαμε να γυρίσουμε πίσω. Όταν στο τέλος βρεθεί η λύση, τα μονοπάτια που είναι μαρκαρισμένα μία φορά, θα δείχνουν το δρόμο προς το αρχικό σημείο.

Shortest path finder

Όπως φαίνεται και από το όνομα ο αλγόριθμος αυτός βρίσκει το πιο σύντομο μονοπάτι προς τη λύση. Αν υπάρχουν περισσότερες από μία σύντομες λύσεις διαλέγει μία. Είναι γρήγορος για όλους τους τύπους των λαβυρίνθων και απαιτεί λίγο παραπάνω μνήμη αναλογικά με το μέγεθος του λαβυρίνθου. Λειτουργεί σα να γεμίζει τον λαβύρινθο με «νερό» έτσι ώστε όλες οι αποστάσεις να γεμίζουν στον ίδιο χρόνο και κάθε σταγόνα θυμίζει ποιο pixel γέμισε. Μόλις η λύση «βραχεί» από κάποια σταγόνα γυρίζει προς τα πίσω και αφήνει ίχνος, έτσι βρίσκεται το πιο σύντομο μονοπάτι.

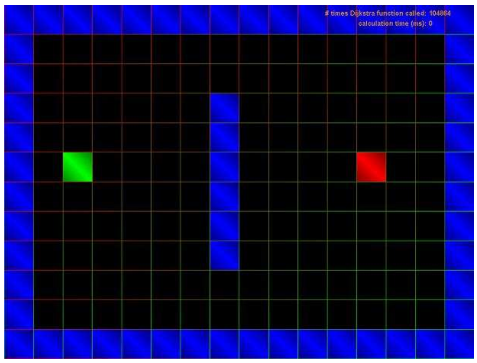
Αλγόριθμος Dijkstra

Ο αλγόριθμος του Dijkstra (Sedgewick, 2003) λύνει το πρόβλημα της συντομότερης διαδρομής, με μία μόνο είσοδο (πηγή) όταν όλα τα άκρα έχουν μη-αρνητικά βάρη. Δεν πρόκειται για έναν αλγόριθμο επίλυσης λαβυρίνθου, αλλά για έναν αλγόριθμο εύρεσης της συντομότερης διαδρομής. Επειδή το πρόγραμμα μας δεν είναι ένας απλός λαβύρινθος αλλά έχει κι άλλες παραμέτρους, όπως το κλειδί και την πόρτα, δεν μπορούμε να χρησιμοποιήσουμε έναν έτοιμο αλγόριθμο επίλυσης. Ο αλγόριθμος ξεκινά από την πηγή κορυφή s, αναπτύσσει ένα δέντρο Τ, που τελικά εκτείνεται σε όλες τις κορυφές προσβάσιμες απ' το S. Προστίθενται κορυφές στο Τ κατά σειρά απόστασης, δηλαδή πρώτα το S, μετά η κορυφή που βρίσκεται πλησιέστερα στο S, έπειτα η επόμενη πιο κοντινή , και ούτω καθεξής. Η επόμενη εφαρμογή υποθέτει ότι ο γράφος G αντιπροσωπεύεται από λίστες γειτνίασης:

```
DIJKSTRA (G, w, s)

ΑΡΧΙΚΟΠΟΙΗΣΕ ΠΗΓΗ (G, s)
S ← { }     //το S θα περιέχει στο τέλος τις κορυφές από τα τελικά βάρη της
συντομότερης διαδρομής από το s
Αρχικοποίησε ουρά προτεραιότητας Q δηλαδή, Q ← V[G]
όσο η ουρά προτεραιότητας Q δεν είναι άδεια κάνε
    u ← EXTRACT_MIN (Q)    / /Βγάλε τη νέα κορυφή
    S ← S È {u}
    / / Εκτέλεσε χαλάρωση για κάθε κορυφή v δίπλα στο u
    για κάθε κορυφή v στο Adj[u] κάνε
       Relax(u, v, w)
```

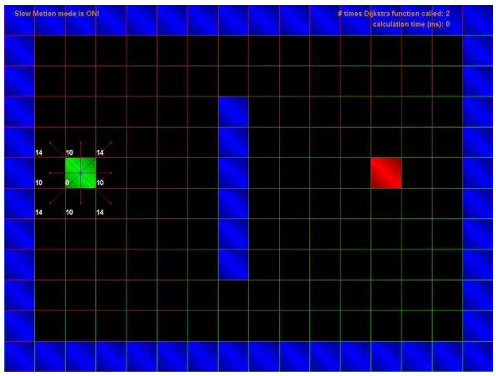
Ο αλγόριθμος του Dijkstra όσον αφορά το πρόβλημα του λαβυρίνθου χρησιμοποιείται για να βρει το κοντινότερο μονοπάτι από ένα σημείο Α σ' ένα σημείο Β μέσα σε τυχαίο λαβύρινθο. Είναι ο αλγόριθμος που επιλέξαμε για το πρόγραμμα μας. Έστω ότι το αρχικό σχέδιο του λαβυρίνθου είναι το παρακάτω:



Εικ. 1 – αρχικό σχέδιο

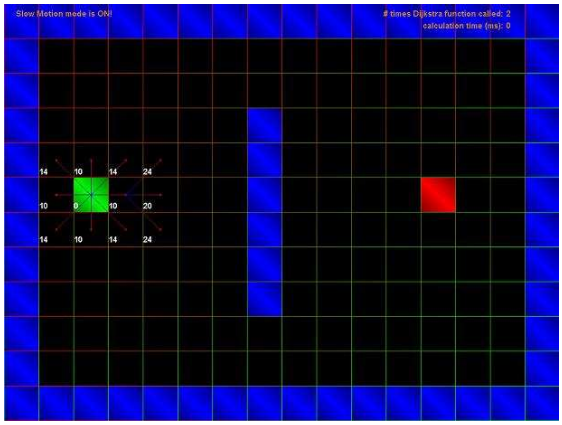
Ο λαβύρινθος αποτελείται από 16 x 12 τετράγωνα. Το πράσινο τετράγωνο είναι το σημείο εκκίνησης και το κόκκινο τετράγωνο είναι το τελικό σημείο. Τα μπλε τετράγωνα αντιπροσωπεύουν τους τοίχους. Τώρα που έχουμε κάνει αυτό το πρόγραμμα μπορούμε να ξεκινήσουμε την εφαρμογή του αλγορίθμου εύρεσης μονοπατιού σ' αυτό.

Το πρόγραμμα πρέπει να δημιουργήσει δύο λίστες μεταβλητών. Αυτές ονομάζονται unvisited και visited. Η unvisited περιέχει τις συντεταγμένες των τετραγώνων που πιθανώς θα ελέγξουμε και η visited περιέχει τις συντεταγμένες των τετραγώνων που πρέπει να βρούμε τη συντομότερη διαδρομή . Η αναζήτηση ξεκινά επιλέγοντας το αρχικό τετράγωνο ως το τρέχον τετράγωνο. Το τρέχον τετράγωνο είναι το τετράγωνο που εξετάζουμε. Το επόμενο βήμα είναι να τοποθετηθούν όλα τα διπλανά τετράγωνα (εκτός από τους τοίχους) του τρέχοντος τετραγώνου στο unvisited. Πρέπει επίσης να υπολογιστεί μια τιμή για κάθε ένα από αυτά τα τετράγωνα. Αυτό το πρώτο βήμα φαίνεται στην εικόνα 2.



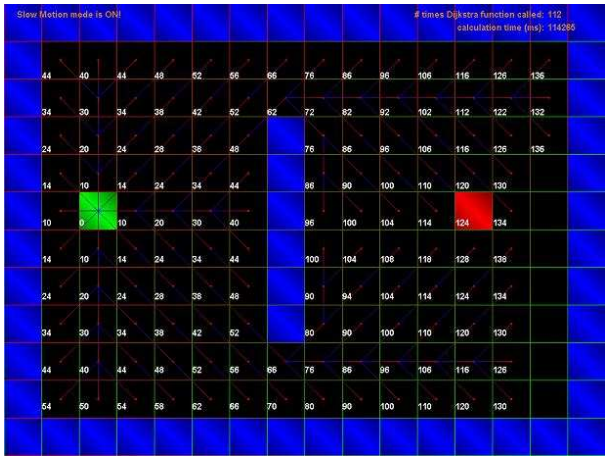
Εικ. 2 – βήμα 1

Εδώ μπορούμε να δούμε ότι τα τετράγωνα γύρω από το σημείο εκκίνησης επισημαίνονται με μια τιμή στην αριστερή κάτω γωνία. Η τιμή αυτή ονομάζεται επίσης G τιμή αυτού του τετραγώνου. Δηλώνει την απόσταση από το σημείο εκκίνησης μέχρι το εν λόγω τετράγωνο. Παρατηρήστε επίσης ότι τα οριζόντια και κάθετα βήματα έχουν βάρος 10 και τα διαγώνια βήματα έχουν βάρος 14. Οι χρωματιστές γραμμές μας δείχνουν επίσης κάτι σημαντικό. Μας δείχνουν για κάθε τετράγωνο ποιος είναι ο γονέας του. Οι γραμμές θα πρέπει να «διαβάζονται» από την κόκκινη στη μπλε πλευρά (είναι ξεθωριασμένες γραμμές). Το τετράγωνο στη μπλε πλευρά της γραμμής είναι το τετράγωνο-γονέας του τετραγώνου στην κόκκινη πλευρά της γραμμής. Το τρέχον τετράγωνο είναι πάντα το τετράγωνο-γονέας για τα τετράγωνα που θέτει σαν visited. Στη συνέχεια, το πρόγραμμα θα πρέπει να πάρει το τρέχον τετράγωνο από το unvisited και να το βάλει στα visited. Μετά από όλα αυτά το πρόγραμμα πρέπει να περάσει από τα unvisited και να ψάξει για το τετράγωνο με τη μικρότερη τιμή G. Το τετράγωνο που θα βρει είναι το νέο τρέχον τετράγωνο και η διαδικασία που συνέβη στο πρώτο τετράγωνο επαναλαμβάνεται. Αυτό φαίνεται στην εικόνα 3.



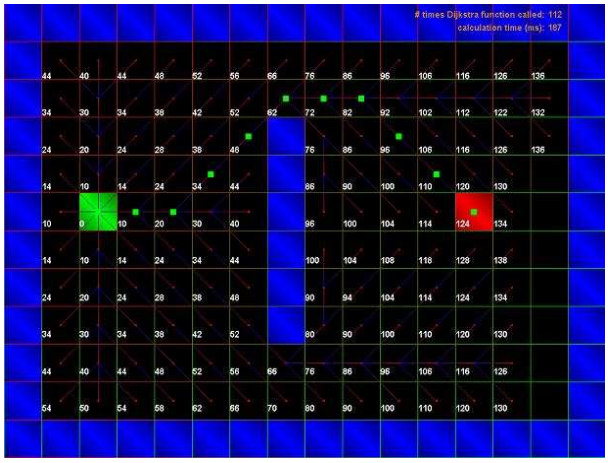
Εικ. 3 – Το επόμενο βήμα

Και πάλι το τρέχον τετράγωνο τοποθετείται στα visited και το πρόγραμμα περνά μέσα από τα unvisited για να βρει το τετράγωνο με τη χαμηλότερη τιμή G. Αυτή η διαδικασία θα επαναλαμβάνεται μέχρι το τελικό σημείο, να μπει στην unvisited. Σε εκείνο το σημείο υπολογίζεται το μονοπάτι (φαίνεται στην εικόνα 4).



Εικ. 4 – Τέλος υπολογισμών

Τώρα θέλουμε να επισημάνουμε το συντομότερο μονοπάτι. Αυτό συμβαίνει ξεκινώντας στο τελικό σημείο και πηδώντας στο τετράγωνο-γονέα του (από την κόκκινη στη μπλε πλευρά της γραμμής), επισημαίνουμε το τετράγωνο και πηδάμε πάλι στο τετράγωνο-γονέα του. Αυτό συνεχίζεται μέχρι να φτάσει στο σημείο εκκίνησης. Σε αυτό το σημείο έχουμε ένα από τα συντομότερα μονοπάτια από το αρχικό μέχρι το τελικό σημείο στην οθόνη μας (φαίνεται στην εικόνα 5).



Εικ. 5 – Το συντομότερο μονοπάτι

Παρακάτω παρέχεται μια περίληψη σε μορφή ψευδοκώδικα για μια καθαρή εικόνα του τρόπου που δουλεύει ο αλγόριθμος:

1. Βάλε το αρχικό τετράγωνο στο Unvisited.
2. Επανέλαβε τα ακόλουθα βήματα:

α) Βρες το τετράγωνο με τη χαμηλότερη τιμή G απ΄ τα unvisited, αυτό το τετράγωνο είναι το τρέχον τετράγωνο.

β) Βάλε το τρέχον τετράγωνο στα visited.

γ) Για κάθε ένα από τα 8 γειτονικά τετράγωνα του τρέχοντος τετραγώνου εκτελούνται τα ακόλουθα βήματα:

Αν το τετράγωνο είναι τοίχος, το αγνοούμε.

Αν το τετράγωνο δεν είναι ακόμη στα unvisited θα τεθεί σ' αυτά. Το τρέχον τετράγωνο είναι το τετράγωνο-γονέας για αυτό το τετράγωνο. Υπολόγισε την G τιμή γι αυτό το τετράγωνο. Αν το τετράγωνο ήταν ήδη στα unvisited τότε πρέπει να γίνει έλεγχος αν το τρέχον μονοπάτι είναι συντομότερο απ' το μονοπάτι που έχει ήδη το τετράγωνο. Χαμηλότερη τιμή G σημαίνει συντομότερο μονοπάτι. Αν αυτό ισχύει, τότε το τετράγωνο αυτό πρέπει να αλλάξει το τετράγωνο-γονέα του και να του δοθεί η νέα τιμή G.

δ) Σταμάτησε τον υπολογισμό, αν:

το τελικό σημείο, είναι στα unvisited. (στην περίπτωση αυτή το συντομότερο μονοπάτι έχει βρεθεί)

Το πρόγραμμα δεν μπορεί να βρει το τελικό σημείο και το unvisited είναι άδειο. (στην περίπτωση αυτή δεν υπάρχει δυνατή διαδρομή από την εκκίνηση ως το τελικό σημείο)

3. Αποθήκευσε στη μνήμη το μονοπάτι που βρέθηκε. Ακολούθησε τα τετράγωνα-γονείς από το τελικό στο αρχικό σημείο. Στην εφαρμογή αυτή απλά ακολούθησε τις γραμμές από την «κόκκινη» πλευρά στην «μπλε» πλευρά μέχρι να φτάσεις στο τελικό σημείο. Τώρα έχει βρεθεί μια από τις συντομότερες διαδρομές από το αρχικό στο τελικό σημείο.

ΣΥΓΚΡΙΣΕΙΣ ΔΙΑΦΟΡΩΝ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΑΣΚΕΥΗΣ

Παρακάτω παραθέτουμε έναν πίνακα σύγκρισης διάφορων αλγορίθμων κατασκευής λαβυρίνθων. Τα χαρακτηριστικά που συγκρίνουμε είναι τα παρακάτω:

Αδιέξοδα: Είναι ένα ποσοστό κατά προσέγγιση κελιών που είναι αδιέξοδα σε ένα λαβύρινθο που δημιουργήθηκε από τον αλγόριθμο. Οι αλγόριθμοι στον πίνακα είναι ταξινομημένοι με βάση αυτό το στοιχείο. Η τιμή του αυξάνοντος δέντρου μπορεί να ποικίλλει από 10% (διαλέγοντας το πιο κοντινό κελί) μέχρι 49%.

Τύπος: υπάρχουν δύο τύποι αλγορίθμων δημιουργίας τέλειων λαβυρίνθων. Ένας αλγόριθμος που βασίζεται στη λογική των δέντρων, αυξάνει τον λαβύρινθο όπως ένα δέντρο, προσθέτοντας οτιδήποτε έχει ήδη παρουσιαστεί, δημιουργώντας έναν τέλειο λαβύρινθο. Ένας αλγόριθμος βασισμένος σε σύνολα, χτίζει όπου «θέλει», αποθηκεύοντας το μονοπάτι για το ποια κομμάτια του λαβυρίνθου είναι συνδεμένα με κάποια άλλα.

Εστίαση: Οι περισσότεροι αλγόριθμοι είτε σκαλίζουν μονοπάτια, είτε προσθέτουν τοίχους. Οι Unicursal λαβύρινθοι δημιουργούν λαβύρινθους με προσθήκη τοίχων, αφού χωρίζουν διάφορα μονοπάτια με τοίχους, παρόλα αυτά ο βασικός λαβύρινθος μπορεί να γίνει και με τους 2 τρόπους.

Αμεροληψία: αυτό το χαρακτηριστικό έχει να κάνει με το αν ο αλγόριθμος απευθύνεται προς όλες τις κατευθύνσεις και τις πλευρές του λαβύρινθου ισότιμα, έτσι ώστε η ανάλυση του αλγορίθμου αργότερα να μην δείχνει μεροληψία.

Μνήμη: το χαρακτηριστικό αυτό έχει να κάνει με το πόση μνήμη επιπλέον απαιτείται για την υλοποίηση του αλγορίθμου. Οι αποδοτικοί αλγόριθμοι απαιτούν και βλέπουν την εικόνα του λαβυρίνθου, ενώ άλλοι απαιτούν μνήμη αντίστοιχη σε μια σειρά (N), ή στον αριθμό των κελιών (N^2). Υπάρχουν λαβύρινθοι που δε χρειάζεται να αποθηκεύουν όλο τον λαβύρινθο (είναι αυτοί με τον αστερίσκο).

Χρόνος: το χαρακτηριστικό αυτό δίνει μια ιδέα για το πόσος χρόνος απαιτείται για τη δημιουργία του λαβυρίνθου, με τη χρήση του αλγόριθμου, τα πιο μικρά νούμερα σημαίνουν ότι ο αλγόριθμος είναι πιο γρήγορος. Τα

νούμερα είναι μόνο σχετικά μεταξύ τους (ο πιο γρήγορος έχει ταχύτητα 10), καθώς η ταχύτητα εξαρτάται και από τον υπολογιστή αλλά και το μέγεθος του λαβυρίνθου. Συνήθως η δημιουργία είτε με την προσθήκη τοίχων, είτε με το σκάλισμα τοίχων χρειάζονται τον ίδιο χρόνο.

Λύση: πρόκειται για το ποσοστό των κελιών στο λαβύρινθο που περνάει το μονοπάτι της λύσης, για έναν λαβύρινθο που έχει δημιουργηθεί με τον συγκεκριμένο αλγόριθμο. Τα δυαδικά δέντρα είναι αρκετά μεροληπτικά, καθώς εύκολα ταξιδεύουν προς μια γωνία και δύσκολα προς την αντίθετη πλευρά.

Αλγόριθμος	**Αδιέξοδο**	**Τύπος**	**Focus**	**Αμεροληψία**	**Μνήμη**	**Χρόνος**	**Λύση**
Unicursal	0	Tree	Wall	Yes	N^2	261	100.0
Recursive Backtracker	10	Tree	Passage	Yes	N^2	24	19.0
Hunt and Kill	11 (21)	Tree	Passage	no	0	55 (105)	9.5 (3.9)
Recursive Division	23	Tree	Wall	Yes	N	8	7.2
Binary Tree	25	Set	Either	no	0*	7	2.0
Eller's Algorithm	28	Set	Either	no	N*	10	4.2 (3.2)
Wilson's Algorithm	29	Tree	Either	Yes	N^2	51 (26)	4.5
Aldous-Border Algorithm	29	Tree	Either	Yes	0	222 (160)	4.5
Kruskal's Algorithm	30	Set	Either	Yes	N^2	32	4.1
Prim's Algorithm	36 (31)	Tree	Either	Yes	N^2	21	2.3
Growing Tree	49 (39)	Tree	Either	Yes	N^2	43	11.0

Πίνακας 1: Αλγόριθμοι δημιουργίας τέλειων λαβυρίνθων

Ο πίνακας αυτός είναι επίσης από τη σελίδα του Walter Pullen (Pullen, 2005) και συγκρίνει κάποια χαρακτηριστικά πολλών αλγορίθμων δημιουργίας τέλειων λαβυρίνθων. Τα νούμερα που βρίσκονται μέσα στην παρένθεση στη στήλη του χρόνου αναφέρονται σε αλγόριθμους που βασίζονται στην προσθήκη τοίχων (στις υπόλοιπες περιπτώσεις ο χρόνος είναι ίδιος).

Όπως φαίνεται και από τον πίνακα παραπάνω:

Ο αλγόριθμος που Eller απαιτεί αποθήκευση για μια σειρά, την τρέχουσα σειρά του λαβυρίνθου. Ό αλγόριθμος δυαδικού δέντρου χρειάζεται να κρατάει το μονοπάτι του τρέχοντος κελιού, ενώ ο αλγόριθμος οπισθοδρομικής διαίρεσης απαιτεί στοίβα για όλες τις σειρές.

Οι Unicursal λαβύρινθοι έχουν τη μέγιστη windiness, καθώς η λύση περνάει από ολόκληρο το λαβύρινθο, ενώ ο αλγόριθμος δυαδικού δέντρου έχει τη μικρότερη δυνατή windiness.

Ο αλγόριθμος Recursive Backtracker δε μπορεί να προσθέτει τοίχους γιατί σε αυτήν την περίπτωση το αποτέλεσμα θα είναι το μονοπάτι της λύσης να ακολουθεί την εξωτερική γωνία. Παρομοίως ο αλγόριθμος της οπισθοδρομικής διαίρεσης μπορεί να προσθέτει τοίχους λόγω της διττής συμπεριφοράς του. Ο αλγόριθμος Hunt and Kill τεχνικά σκαλίζει μονοπάτια για ένα παρόμοιο λόγο, παρόλο που μπορεί να προσθέτει τοίχους.

Ο αλγόριθμος του Eller τείνει να φτιάχνει ένα μονοπάτι παράλληλο στις γωνίες του τέλους ή της αρχής. Ο αλγόριθμος Hunt and Kill είναι σχεδόν αμερόληπτος, παρόλη τη συστηματική μπρος πίσω αναζήτηση.

Από την άποψη της μνήμης βλέπουμε ότι οι αλγόριθμοι Hunt and Kill, Binary Tree και Aldous-Broder χρειάζονται μηδενική μνήμη, ενώ οι αλγόριθμοι Unicursal, Recursive Backtracker, Wilson, Kruskal, Prim, Growing Tree χρειάζονται μνήμη N^2.

Σε κάθε περίπτωση για τη σύγκριση και την επιλογή κάποιου λαβυρίνθου θα πρέπει να λαμβάνονται υπόψη αν όχι όλοι, πολλοί από τους παραπάνω παράγοντες, καθώς κάποιος αλγόριθμος μπορεί να χρειάζεται μικρή μνήμη, ταυτόχρονα όμως να είναι αργός στη λύση.

Άλλος ένας πίνακας σύγκρισης αλγορίθμων βρίσκεται παρακάτω (MU, 2008).

Algorithm	Maze size	SAC	AST (sec.)	TT – SA (min.)	ANM
Bacterial Growth	Large	3	27	1.37	127
	Medium	32	14	7.53	78
	Small	1	11	0.18	40
	Tiny	2	2	0.08	11
Hunt And Kill	Large	6	194	19.47	392
	Medium	46	121	93.45	212
	Small	8	13	1.83	61
	Tiny	40	7	4.77	21
Randomized Kruskal's	Large	14	144	33.70	170
	Medium	18	79	23.83	99
	Small	9	15	2.38	39
	Tiny	35	4	2.45	15
Randomized Prim's	Large	29	67	32.55	129
	Medium	861	30	443.68	82
	Small	9	12	1.80	40
	Tiny	62	4	4.97	13
Recursive Backtracker	Large	3	158	7.90	331
	Medium	10	64	10.73	166
	Small	54	18	16.58	92
	Tiny	42	8	5.65	21

Table D.1: Randomized mazes' global statistics

Πίνακας 2: Σύγκριση επιδόσεων αλγορίθμων δημιουργίας

ΚΕΦΑΛΑΙΟ 3

ΕΠΙΛΟΓΗ ΛΑΒΥΡΙΝΘΟΥ – ΣΧΕΔΙΑΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Ο ΛΑΒΥΡΙΝΘΟΣ ΠΟΥ ΔΗΜΙΟΥΡΓΟΥΜΕ

Ο λαβύρινθος που θα δημιουργήσουμε είναι ένας τέλειος λαβύρινθος. Τέλειος λαβύρινθος αποκαλείται ένας λαβύρινθος ο οποίος διαθέτει ένα και μοναδικό σημείο εισόδου και ένα και μοναδικό σημείο εξόδου. Από κάθε οποιοδήποτε σημείο του λαβυρίνθου, υπάρχει ένα και μόνο μοναδικό μονοπάτι προς ένα οποιοδήποτε άλλο σημείο. Επίσης, σε έναν τέλειο λαβύρινθο δεν υπάρχουν κλειστές διαδρομές (βρόχοι) ούτε και ανοιχτές περιοχές.

Συνέπεια αυτού του ορισμού, είναι ότι όλα τα κελιά σε έναν τέλειο λαβύρινθο είναι προσβάσιμα από την αφετηρία μέσω κάποιας μοναδικής διαδρομής, που σημαίνει ότι οι τέλειοι λαβύρινθοι είναι εγγυημένοι ότι έχουν μια μοναδική λύση.

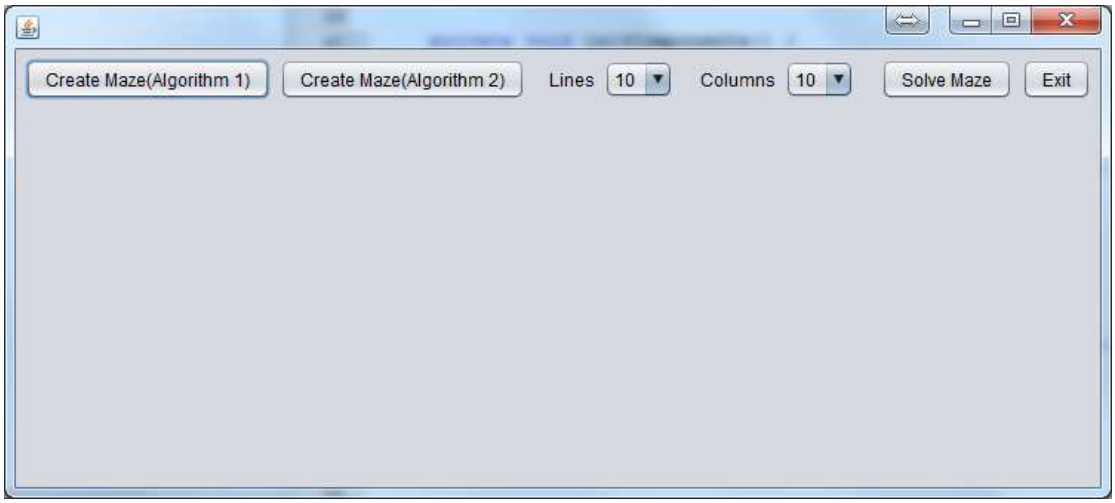
Ο λαβύρινθος που θα δημιουργήσουμε έχει αρχή το πάνω αριστερά κελί και τέλος το κάτω δεξιά κελί. Επιπλέον για να φτάσει στο τέλος κάποιος που λύνει τον λαβύρινθο θα πρέπει να περάσει πρώτα από ένα κλειδί το οποίο βρίσκεται στο αριστερό μισό του λαβυρίνθου. Το κλειδί αυτό χρειάζεται για να ξεκλειδώσει την πόρτα που βρίσκεται στη μέση του λαβυρίνθου, ώστε να περάσει στο δεξί μισό του λαβυρίνθου. Ουσιαστικά ο αλγόριθμος διανύει τρεις διαδρομές: Η πρώτη είναι από την είσοδο μέχρι το κλειδί, η δεύτερη από το κλειδί ως την πόρτα και η τρίτη από την πόρτα μέχρι την έξοδο.

Στις πρώτες δύο διαδρομές ο αλγόριθμος έχει ελευθερία κινήσεων, που σημαίνει ότι μπορεί να περάσει από το ίδιο σημείο όσες φορές χρειάζεται μέχρι να πάρει το κλειδί και να ανοίξει την πόρτα. Στην τρίτη διαδρομή, από την πόρτα ως την έξοδο, ο αλγόριθμος απαγορεύεται να ξαναπεράσει απ’ το ίδιο σημείο.

Ο αλγόριθμος επίλυσης του λαβυρίνθου που υλοποιούμε είναι βασισμένος στον αλγόριθμο του Dijkstra, με μικρές παραλλαγές και βελτιώσεις. Το σκεπτικό όμως είναι ίδιο. Υπολογίζει την τιμή όλων των κελιών

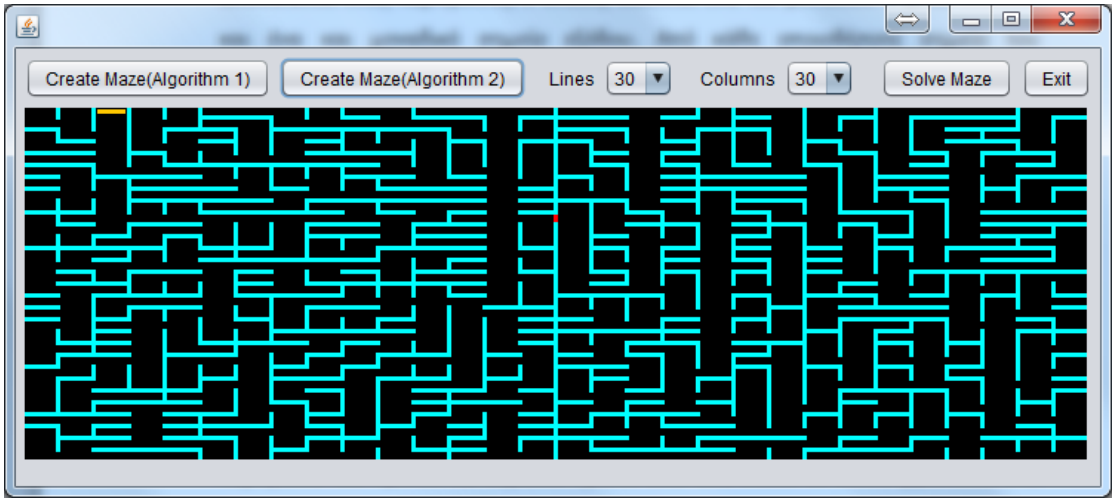
μέχρι να βρει την έξοδο και όταν την βρει οπισθοδρομεί από την συντομότερη διαδρομή, δηλαδή απ' τις μικρότερες τιμές.

Το πρόγραμμα διαθέτει γραφικό περιβάλλον για την ευκολία του χρήστη το οποίο δημιουργήθηκε με το Netbeans. Διαθέτει δύο κουμπιά κατασκευής λαβυρίνθου, δύο combo boxes για την επιλογή των γραμμών και των στηλών του λαβυρίνθου που επιθυμούμε, το κουμπί της επίλυσής του και το κουμπί εξόδου.



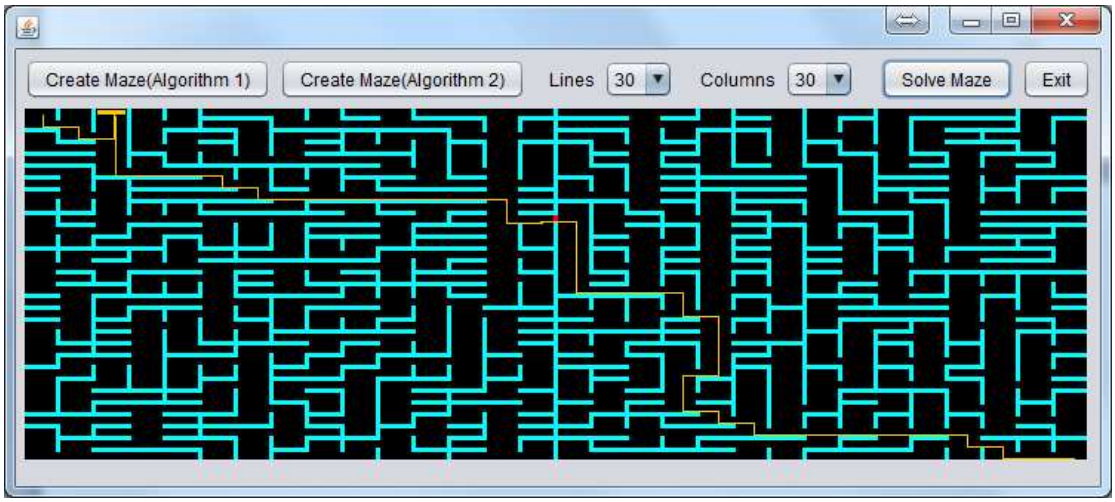
Εικ. 1 – Γραφικό περιβάλλον

Έχουμε δύο μεθόδους κατασκευής του λαβυρίνθου στο πρόγραμμα. Δίνεται η δυνατότητα στον χρήστη να επιλέξει ποια μέθοδο θέλει να χρησιμοποιήσει μέσω του παραπάνω παραθύρου γραφικών. Θα αναλύσουμε τις δύο αυτές μεθόδους παρακάτω. Μόλις ο χρήστης επιλέξει πλήθος γραμμών, πλήθος στηλών και μέθοδο κατασκευής του λαβυρίνθου εμφανίζεται ο τυχαίος λαβύρινθος που παράγεται.



Εικ. 2 – Λαβύρινθος 30x30

Αφού εμφανιστεί ο λαβύρινθος που κατασκευάστηκε απ' το πρόγραμμα ο χρήστης μπορεί να δει τη λύση του με το κουμπί "Solve Maze". Η διαδρομή που ακολούθησε ο αλγόριθμος φαίνεται σαν εφαρμογή πραγματικού χρόνου ώστε να μπορεί ο χρήστης να την παρακολουθήσει σε πραγματικό χρόνο.



Εικ. 3 – Επίλυση λαβυρίνθου

Τα γραφικά σχεδίασης του λαβυρίνθου δεν έγιναν με το Netbeans, αλλά από τον υποφαινόμενο με την μέθοδο PaintMaze με χρήση του πακέτου γραφικών java.awt. Η μέθοδος θα αναλυθεί λεπτομερώς παρακάτω.

ΜΕΘΟΔΟΙ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Αρχικά θα αναλύσουμε τις δύο μεθόδους κατασκευής λαβυρίνθου *CreateMaze* και *CreateMaze_2*. Βασίζονται, όπως ανέφερα στο 2^{ο} κεφάλαιο, στους αλγορίθμους κατασκευής «Αλγόριθμος με τοίχους» και «Αλγόριθμος με δέντρο» που δημιουργήθηκαν από έλληνες προγραμματιστές και είναι παραλλαγές του αλγορίθμου «Aldous - Border». Οι δύο μέθοδοι λειτουργούν με σχεδόν αντίθετο τρόπο. Οι διαφορές στον τρόπο λειτουργίας τους φαίνονται παρακάτω.

Πριν ξεκινήσουμε τις μεθόδους δημιουργίας, δημιουργούμε μία απαρίθμηση, enum MazeCellStatus, η οποία καταγράφει το είδος του κάθε κελιού του λαβυρίνθου. Τα είδη είναι EMPTY, KEY, DOOR, WALL, HELP, START_POSITION, END_POSITION.

CreateMaze: Στην πρώτη συνάρτηση δημιουργίας δεν έχουμε δύο πίνακες όπως ορίζει ο αλγόριθμος με τοίχους, αλλά έναν πίνακα δύο διαστάσεων myMaze[][]. Ο πίνακας αυτός είναι τύπου MazeCellStatus. Ο πίνακας στις ζυγές [γραμμές][στήλες] έχει πιθανά κελιά μονοπατιού, και στις μονές έχει πιθανά κελιά τοίχου. Ο αλγόριθμος αυτός βασίζεται στην κατασκευή τοίχων, που σημαίνει ότι πρώτα χαράσσει τους τοίχους και έπειτα την διαδρομή με βάση αυτούς.
Οι κύριες μεταβλητές μας είναι:
int r: γραμμές,
int c: στήλες
int doorposition: η σειρά που βρίσκεται η πόρτα (τυχαία)
int tt: η στήλη που βρίσκεται η πόρτα
int keyY, keyX: η θέση του κλειδιού.

Δευτερεύουσες μεταβλητές και στοιχεία:
int ar, ac: οι μισές [στήλες][γραμμές] του πίνακα
int ox, oy: μεταβλητές προσωρινής αποθήκευσης τρέχοντος σημείου
int n, n1: μεταβλητές κατεύθυνσης
int x1, y1: μεταβλητές επόμενου βήματος.

Ο αλγόριθμος με τοίχους αρχικά θέτει όλο τον λαβύρινθο άδειο,

```
myMaze[i][j] = MazeCellStatus.EMPTY;
```

Σαν σημείο αναφοράς για την Random έχουμε την ώρα σε ms. Κάτι εντελώς τυχαίο.

```
Random rnd = new Random();
rnd.setSeed(java.util.Calendar.getInstance().getTimeInMillis());
```

Έπειτα τοποθετεί την πόρτα στην κεντρική στήλη και θέτει το υπόλοιπο κέντρο ως τοίχο. Η θέση της πόρτας πολλαπλασιάζεται με το 2 για να είναι πάντα ζυγός αριθμός, δηλαδή πιθανή θέση μονοπατιού,

```
for (int i = 0; i < c; i++) {  myMaze[i][tt] = MazeCellStatus.WALL;    }
myMaze[doorPosition * 2][tt] = MazeCellStatus.DOOR;
```

Τοποθετεί το κλειδί στο αριστερό μισό του λαβυρίνθου, παρομοίως πολλαπλασιάζεται με το 2 για τον ίδιο λόγο,

```
int keyY = rnd.nextInt(ar / 2), keyX = rnd.nextInt(ac / 2);
 myMaze[2 * keyY][2 * keyX] = MazeCellStatus.KEY;
```

Αρχικοποιεί το αρχικό και τελικό σημείο του λαβυρίνθου,

```
myMaze[0][0] = MazeCellStatus.START_POSITION;
myMaze[r - 1][c - 1] = MazeCellStatus.END_POSITION;
```

Ξεκινάει από οποιοδήποτε σημείο τοίχου που υπάρχει και χαράζει τυχαία γραμμή τοίχου σε κάθε επανάληψη η οποία δεν πρέπει να τέμνει κανένα υπάρχον τοίχο. Σε κάθε βήμα ψάχνουμε τυχαία πάνω, κάτω, δεξιά και αριστερά, για το επόμενο βήμα:

```
do {
```

```
if (ox == x && oy == y) {
    counter++;
    n++;
    n %= 4;

} else {
    ox = x;
    oy = y;
    int n1=n;
    n = rnd.nextInt(4);
    if(n==n1)
        n=(n1+1)%4;
        //n=rnd.nextInt(4);
    counter = 0;
}
```

Αν το n είναι 0 κατευθύνεται πάνω, αν είναι 1 κάτω, 2 δεξιά και 3 αριστερά. Στα x1, y1 αποθηκεύονται οι συντεταγμένες του επόμενου βήματος,

```
switch (n) {
        case 0:
            x1 = x + 2;    y1 = y;
            break;
        case 1:
            x1 = x - 2;     y1 = y;
            break;
        case 2:
            y1 = y + 2;     x1 = x;
            break;
        case 3:
            y1 = y - 2;     x1 = x;
            break;
    }
```

Αν το βήμα είναι έγκυρο το αποθηκεύουμε και προχωράμε

```
if (x1 > 0 && y1 > 0 && x1 < r && y1 < c && myMaze[x1][y1] ==
MazeCellStatus.EMPTY) {
        myMaze[x1][y1] = MazeCellStatus.WALL;
        myMaze[(x + x1) / 2][(y + y1) / 2] = MazeCellStatus.WALL;
        x = x1;  y = y1;                }
```

Αυτό γίνεται μέχρι να ολοκληρωθεί η κατασκευή που θέλουμε, πράγμα που ελέγχει η συνάρτηση:

```
checkIfMazeComplete (MazeCellStatus[][] maze)
    for (int i = 1; i < maze.length; i += 2) {
        for (int j = 1; j < maze[i].length; j += 2) {
            if (maze[i][j] == MazeCellStatus.EMPTY)
                    return false;
     }      }
    return true;
}
```

Αν δηλαδή έστω κι ένα κελί του πίνακα είναι EMPTY η κατασκευή δεν έχει ολοκληρωθεί.

CreateMaze_2: Στη δεύτερη συνάρτηση δημιουργίας έχουμε πάλι έναν πίνακα, myMaze[][], όπως ορίζει ο αλγόριθμος με δέντρο. Κι εδώ όπως και στον προηγούμενο αλγόριθμο ο πίνακας στις ζυγές [γραμμές][στήλες] έχει πιθανά κελιά μονοπατιού, και στις μονές έχει πιθανά κελιά τοίχου.
Σε αντίθεση με τον πρώτο αλγόριθμο, εδώ αποφασίζονται πρώτα όλες οι πιθανές διαδρομές και μετά χαράσσονται στον λαβύρινθο. Τοποθετούμε αρχικά παντού τοίχους, αλλά στη συνέχεια, και αφού βρούμε όλες τις πιθανές διαδρομές, σβήνουμε για κάθε διαδρομή τους απαραίτητους τοίχους ώστε να είναι δυνατή η εκτέλεση της διαδρομής αυτής. Έτσι προκύπτει ο λαβύρινθος.

Όπως και στον προηγούμενο αλγόριθμο, ο αλγόριθμος αρχικά τοποθετεί την πόρτα, το κλειδί και το αρχικό και τελικό σημείο. Οι βασικές μεταβλητές είναι οι ίδιες. Οι βοηθητικές μεταβλητές είναι:
int occupiedCells: μετρά τα συνολικά χρησιμοποιημένα κελιά
boolean cond: λογική μεταβλητή συνθήκης τερματισμού της χάραξης διαδρομών
int fx, fy, tx, ty, ix, iy: μεταβλητές προσωρινής αποθήκευσης δεδομένων στοίβας

Εδώ όμως, χρησιμοποιούμε στοίβες για την αποθήκευση των μονοπατιών του λαβυρίνθου.

```
Stack<Integer> fromX, fromY, toX, toY;
```

Στις στοίβες *fromX, fromY* αποθηκεύονται τα σημεία που περνάμε, και στις στοίβες *toX, toY* τα σημεία που θα πάμε. Ουσιαστικά οι στοίβες αυτές περιέχουν τις πιθανές διαδρομές του λαβυρίνθου.

Ο αλγόριθμος αυτός ξεκινάει από ένα τυχαίο σημείο και χαράσσει μια τυχαία διαδρομή χωρίς να τέμνεται πουθενά. Στη συνέχεια, ξεκινάει από οποιοδήποτε υπάρχον σημείο των προηγούμενων διαδρομών και δημιουργεί μια νέα. Ο αλγόριθμος ολοκληρώνεται όταν δεν υπάρχει άλλο μη χρησιμοποιημένο σημείο.

```
do {
  x = rnd.nextInt(ar);          y = rnd.nextInt(ac);
 if (occupiedCells > 0 && myMaze[2 * x][2 * y] ==
MazeCellStatus.EMPTY) {
    continue;
        }
 myMaze[2 * x][2 * y] = MazeCellStatus.HELP;
 int x1 = -1, y1 = -1, n = 0;
 boolean cond = true;
 int prev = 0;
```

Ψάχνουμε πάνω, κάτω, αριστερά κ δεξιά τυχαία, όπως και στον προηγούμενο αλγόριθμο, με τη διαφορά ότι εδώ τα βήματα μας προχωράνε ανά ένα κελί κι όχι ανά δύο. Αυτό γίνεται γιατί, όπως είπαμε, πρώτα χαράσσονται οι πιθανές διαδρομές, οπότε στην αρχή δε μας ενδιαφέρει αν το κελί είναι ζυγός ή μονός αριθμός,

```
do {
   if (prev == 0) {
      int n1 = n;
      n = rnd.nextInt(4);
      if (n1==n)
         n = (n++)%4;
   } else {
      n++;
      n %= 4;
      prev++;
   }
   if (prev == 4) {
      break;
   }
   x1 = x;       y1 = y;
   switch (n) {
      case 0:
         x1++;
         break;
      case 1:
         x1--;
         break;
      case 2:
         y1++;
         break;
      case 3:
         y1--;
         break;  }
```

Αν η κίνηση είναι θεμιτή, την κατοχυρώνει, διαφορετικά συνεχίζει

```
            if (x1 < 0 || y1 < 0 || x1 >= ar || y1 >= ac || myMaze[2 * x1][2 * y1] ==
                 MazeCellStatus.HELP) {
               prev++;
               continue;
            }
            if (myMaze[x + x1][y + y1] == MazeCellStatus.WALL) {
               prev++;
               continue;
            }
            prev = 0;
            fromX.push(x);
            fromY.push(y);
            toX.push(x1);
            toY.push(y1);
            myMaze[2 * x1][2 * y1] = MazeCellStatus.HELP;
            cond = rnd.nextInt() % 20 == 0;
            occupiedCells++;
         } while (cond == true);
} while (checkIfMaze2Complete(myMaze) == false);
```

Γεμίζει όλον τον λαβύρινθο με τοίχους και αδειάζει όλα τα κελιά των πιθανών διαδρομών. Αυτά είναι τα ζυγά κελιά και τα κελιά που έχει αποθηκεύσει στις στοίβες που όπως είπαμε είναι πιθανές διαδρομές. Με βάση τις διαδρομές, σβήνει τους απαραίτητους τοίχους

```
      for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
           myMaze[i][j] = MazeCellStatus.WALL;          }         }

     for (int i = 0; i < r; i += 2) {
        for (int j = 0; j < c; j += 2) {
           myMaze[i][j] = MazeCellStatus.EMPTY;      }       }
```

```
do {
    int fx =  fromX.pop();
    int fy =  fromY.pop();
    int tx =  toX.pop();
    int ty =  toY.pop();
    int ix = fx + tx;
    int iy = fy + ty;
    myMaze[ix][iy] = MazeCellStatus.EMPTY;
} while (fromX.size() > 0);
```

Τέλος, όπως κι ο πρώτος αλγόριθμος, έχει μία μέθοδο που ελέγχει αν ολοκληρώθηκε η κατασκευή μας. Την μέθοδο *checkIfMaze2Complete(MazeCellStatus[][] maze).*

solveMaze: Έπειτα, έχουμε την μέθοδο επίλυσης του λαβυρίνθου. Η συνάρτηση αυτή επιλύει το λαβύρινθο. Η λύση μπορεί να αναλυθεί σε 3 επιμέρους τμήματα

1) Την διαδρομή από την αρχή μέχρι το κλειδί
2) Την διαδρομή από το κλειδί μέχρι την πόρτα
3) Την διαδρομή από την πόρτα μέχρι την έξοδο

int keyX, keyY, doorX, doorY, startX, startY, endX, endY: Συντεταγμένες των καίριων σημείων του αλγορίθμου

Η μέθοδος αρχικά εντοπίζει τα 4 καίρια σημεία του λαβυρίνθου. Πόρτα, κλειδί, αρχή και τέλος.

```
for (int i = 0; i < myMaze.length; i++) {
 for (int j = 0; j < myMaze[i].length; j++) {
switch (myMaze[i][j]) {
        case START_POSITION:
            startX = i;          startY = j;
            break;
        case END_POSITION:
            endX = i;            endY = j;
            break;
```

```
            case DOOR:
               doorX = i;            doorY = j;
               break;
            case KEY:
               keyX = i;             keyY = j;
               break;
         } } }
```

Τέλος, καλεί τη συνάρτηση υπολογισμού της συντομότερης διαδρομής για κάθε μία από τις τρεις περιπτώσεις μ' αυτή τη σειρά: πόρτα-τέλος, κλειδί-πόρτα, αρχή-κλειδί.

```
routeFromDoorToEnd = findPath(doorX / 2, (doorY-1) / 2, endX / 2, endY / 2, myMaze);
routeFromKeyToDoor = findPath(keyX/2, keyY/2, (doorX ) / 2, (doorY-1) / 2, myMaze);
routeFromStartToKey = findPath(startX / 2, startY / 2, keyX / 2, keyY / 2, myMaze);
```

findPath: Η συνάρτηση αυτή υπολογίζει την βέλτιστη διαδρομή μεταξύ δύο σημείων του λαβυρίνθου. Ο τρόπος λειτουργίας του αλγορίθμου στηρίζεται στη δημιουργία και τον υπολογισμού ενός πίνακα αποστάσεων. Η τιμή κάθε κελιού, αναπαριστά πόσα βήματα χρειάζονται για να μεταβούμε σε αυτό από το σημείο x1,y1. Βασίζεται στον αλγόριθμο του Dijkstra που αναφέρθηκε παραπάνω. Ξεκινάμε και τοποθετούμε την τιμή 0 στο σημείο x1,y1 του πίνακα και άπειρο στα υπόλοιπα.

```
    for (int i = 0; i < localMatrix.length; i++) {
     for (int j = 0; j < localMatrix[i].length; j++) {
            localMatrix[i][j] = 99999;
    }    }
  localMatrix[x1][y1] = 0;
```

Στη συνέχεια, ανανεώνουμε επαναληπτικά τις τιμές των αποστάσεων, έως ότου δεν υπάρχει αλλαγή. Για να περιορίσουμε τον αριθμό των επαναλήψεων χωρίζουμε τον λαβύρινθο σε τέσσερα τμήματα με βάση το αρχικό σημείο. Έτσι, σαρώνουμε όλα τα σημεία του λαβυρίνθου, ξεκινώντας από το σημείο x1, y1.

```
for (int i = x1; i < localMatrix.length; i++) {
        for (int j = y1; j < localMatrix[i].length; j++) {
         if (localMatrix[i][j] == 99999) {
            continue;
         }
         for (int k = 0; k < 4; k++) {
            int ii = i, jj = j;
            if (k == 0) {
               ii++;     }
            if (k == 1) {
               ii--;      }
            if (k == 2) {
               jj++;      }
            if (k == 3) {
               jj--;      }
            if (ii < 0 || jj < 0 || ii >= localMatrix.length || jj >= localMatrix[i].length) {                continue;                }
            if (maze[i + ii][j + jj] != MazeCellStatus.WALL) {
               if (localMatrix[ii][jj] > localMatrix[i][j] + 1) {
          localMatrix[ii][jj] = localMatrix[i][j] + 1;  //Βάζει την μικρότερη τιμή στον localMatrix
                  change = true;
            }       }
    ……..
```

Παρομοίως, για τα άλλα τρία τεταρτημόρια του λαβυρίνθου.

Στη συνέχεια, ξεκινώντας από το σημείο x2,y2 και πηγαίνοντας συνεχώς σε σημεία μικρότερης αξίας φτάνουμε στο σημείο x1,y1 κρατώντας την διαδρομή που ακολουθήσαμε.

```
int value = localMatrix[x2][y2];
retVal.setSize(value + 1);
 cval = new Vector<Integer>(2);
cval.setSize(2);
cval.setElementAt(x2, 0);
cval.setElementAt(y2, 1);
retVal.setElementAt(cval, value);
int xi = x2, yi = y2;
 do {
 for (int k = 0; k < 4; k++) {
    int ii = xi, jj = yi;
    if (k == 0) {
       ii++;            }
    if (k == 1) {
       ii--;            }
    if (k == 2) {
       jj++;            }
    if (k == 3) {
       jj--;           }
    if (ii < 0 || jj < 0 || ii >= localMatrix.length || jj >= localMatrix[xi].length)
{
       continue;         }
    if (maze[xi + ii][yi + jj] != MazeCellStatus.WALL) {
       if (localMatrix[ii][jj] == value - 1) {
          cval = new Vector<Integer>(2);
          xi = ii;
          yi = jj;
          cval.setSize(2);
          cval.setElementAt(xi, 0);
          cval.setElementAt(yi, 1);
```

```
                value--;
                retVal.setElementAt(cval, value);
                break;
            }  }  }
    } while (value > 0);
```

PaintMaze, PaintRoute: Οι δύο αυτές συναρτήσεις ζωγραφίζουν το λαβύρινθο και τις διαδρομές στην οθόνη. Χρησιμοποιούν τα πακέτα java.awt.Graphics και java.awt.Color της βιβλιοθήκης της Java. Τα χρώματα που χρησιμοποιούνε είναι μαύρο φόντο του λαβυρίνθου, πορτοκαλί για την αρχή, το τέλος και τις διαδρομές, κόκκινο για το κλειδί και την πόρτα. Για την ακριβή σχεδίαση των στοιχείων χρειαζόμαστε την μεταβλητή offset που ορίζει τα περιθώρια μεταξύ των σημείων του λαβυρίνθου. Έτσι, η διαδρομή σχεδιάζεται ακριβώς στη μέση ανάμεσα από δύο τοίχους και δεν τέμνονται οι γραμμές του λαβυρίνθου. Αυτό που πρέπει να επισημανθεί, είναι ότι ενώ η διαδρομή φαίνεται να γίνεται σε πραγματικό χρόνο, στην πραγματικότητα έχει βρεθεί το τέλος απ' τη στιγμή που πατάμε το κουμπί "Solve Maze" απλά εμφανίζεται σταδιακά. Αυτό γίνεται στη συνάρτηση PaintRoute (Smith, 1990) με τη βοήθεια της κλάσης *Thread* και της μεθόδου της *sleep* που μας επιτρέπει να κάνουμε την εφαρμογή να περιμένει κάποιο διάστημα πριν εμφανίσει κομμάτια της διαδρομής. Έτσι, φαίνεται σαν να λύνεται ο λαβύρινθος σε πραγματικό χρόνο.

MazeCellStatus: Είναι μία κλάση τύπου enum ή απλά απαρίθμηση. Η απαρίθμηση αυτή καταγράφει το είδος του κάθε κελιού του λαβυρίνθου. Οι πιθανές σταθερές τιμές ενός κελιού είναι: *EMPTY, KEY, DOOR, WALL,START_POSITION, END_POSITION, HELP.*

Τέλος, οι μέθοδοι *initComponents, btnExitActionPerformed, btnCreateActionPerformed, btnCreate1ActionPerformed, btnSolveActionPerformed* και η μέθοδος *main*, δημιουργούνται απ' το Netbeans αυτόματα με την κατασκευή του παραθύρου και των κουμπιών του

προγράμματος. Εμείς απλά θέσαμε τις λειτουργίες του κάθε κουμπιού, τι συνάρτηση θα καλεί, τι ενέργεια θα πραγματοποιεί κλπ.

ΚΕΦΑΛΑΙΟ 4

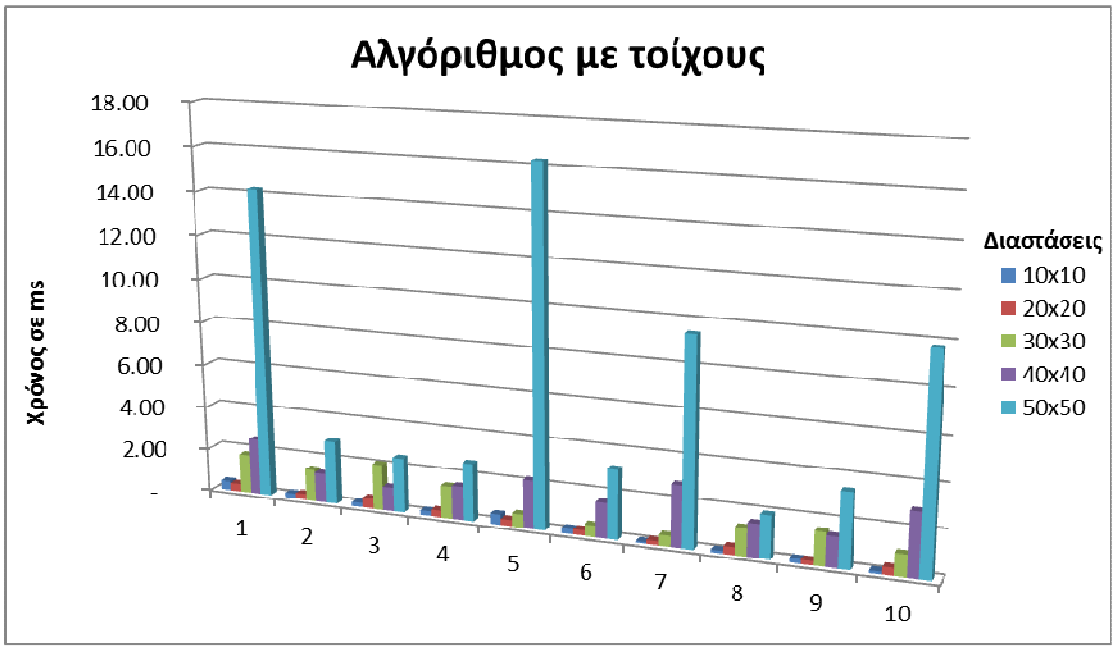
ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ

Σε αυτό το κεφάλαιο θα κάνουμε σύγκριση των δύο αλγορίθμων κατασκευής που χρησιμοποιήσαμε: αλγόριθμος με τοίχους και αλγόριθμος με δέντρο. Κάποιες βασικές τους διαφορές έχουν αναφερθεί σε διάφορα σημεία της εργασίας, εδώ όμως θα τις συγκεντρώσουμε όλες μαζί.

Αρχικά, αξίζει να αναφερθούμε και πάλι στο γεγονός ότι και οι δύο αλγόριθμοι προήλθαν από τον ίδιο αλγόριθμο των Aldous-Broder. Αυτό σημαίνει ότι και οι δύο αλγόριθμοι είναι αλγόριθμοι εύρεσης ομοιόμορφα απλωμένων δέντρων (uniform spanning trees). Η σύγκριση που έκανα αφορά τον χρόνο δημιουργίας και επίλυσης κάθε αλγορίθμου. Παρακάτω παραθέτω στατιστικά για τον κάθε αλγόριθμο ξεχωριστά, αλλά και συγκριτικά για τους δύο αλγορίθμους κατασκευής.

ΧΡΟΝΟΣ ΔΗΜΙΟΥΡΓΙΑΣ ΛΑΒΥΡΙΝΘΩΝ

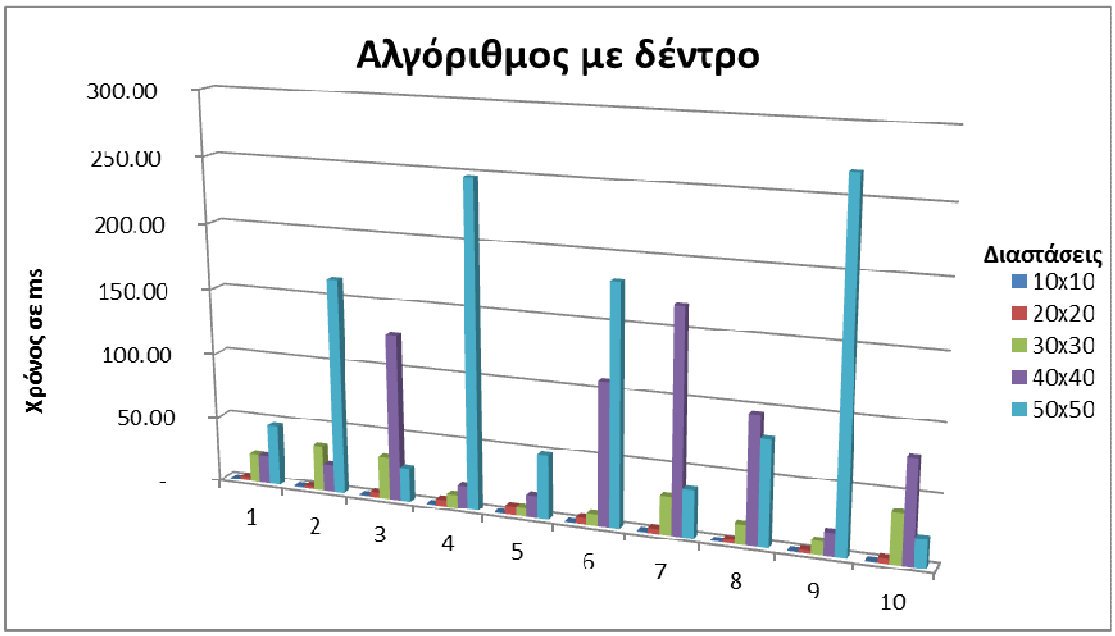
Για τον αλγόριθμο με τοίχους:



Γράφημα 1

Όπως βλέπουμε σε 10 δοκιμές ο μέγιστος χρόνος δημιουργίας λαβυρίνθου για τον αλγόριθμο με τοίχους είναι 16 ms σε διαστάσεις 50x50. Στις υπόλοιπες διαστάσεις ο χρόνος είναι πολύ μικρός, κάτω από 4 ms. Από το σχήμα διαπιστώνεται ότι όσο μεγαλύτερες είναι οι διαστάσεις του λαβυρίνθου ο χρόνος δημιουργίας πολλαπλασιάζεται.

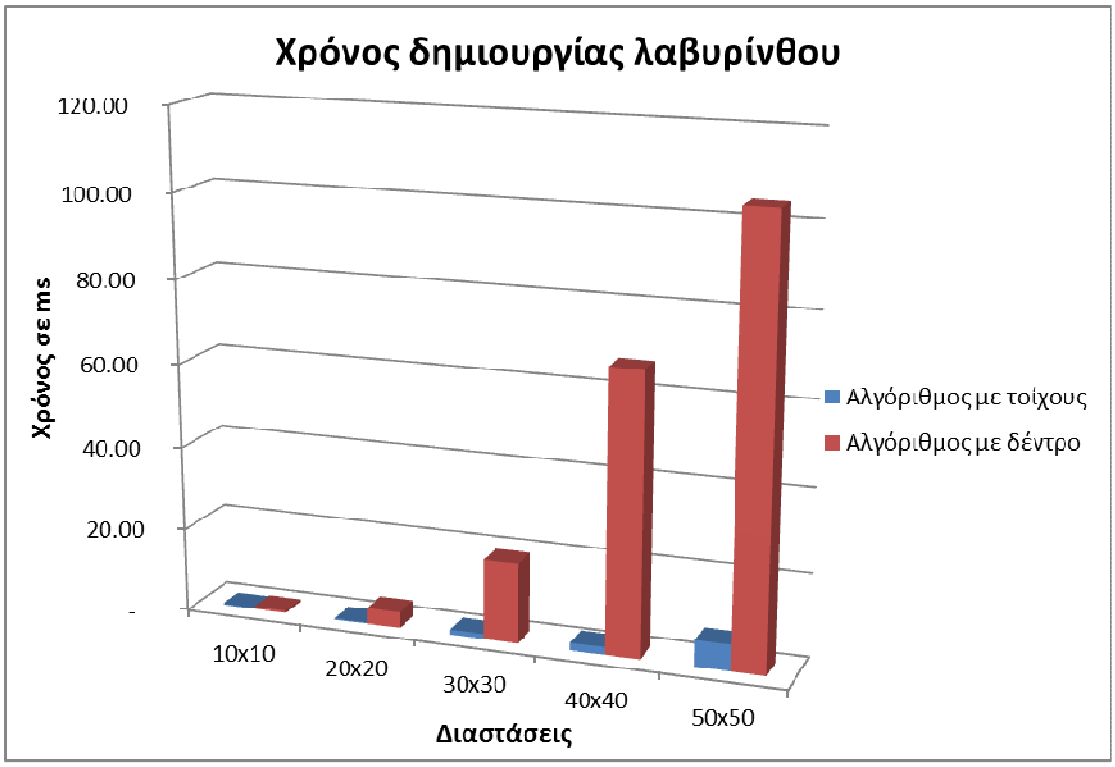
Για τον αλγόριθμο με δέντρο:



Γράφημα 2

Παρατηρούμε ότι ο αλγόριθμος με δέντρο στις μεγάλες διαστάσεις γίνεται πολύ απρόβλεπτος στο χρόνο δημιουργίας λαβυρίνθων. Στις διαστάσεις 40x40 κυμαίνεται από 20 ms μέχρι 170 ms, πράγμα που δείχνει μεγάλη αστάθεια και πολυπλοκότητα. Αντίθετα, στις μικρές διαστάσεις οι τιμές δεν κυμαίνονται και υπάρχει γενικά σταθερότητα.

Συγκριτικά για τους δύο αλγορίθμους:



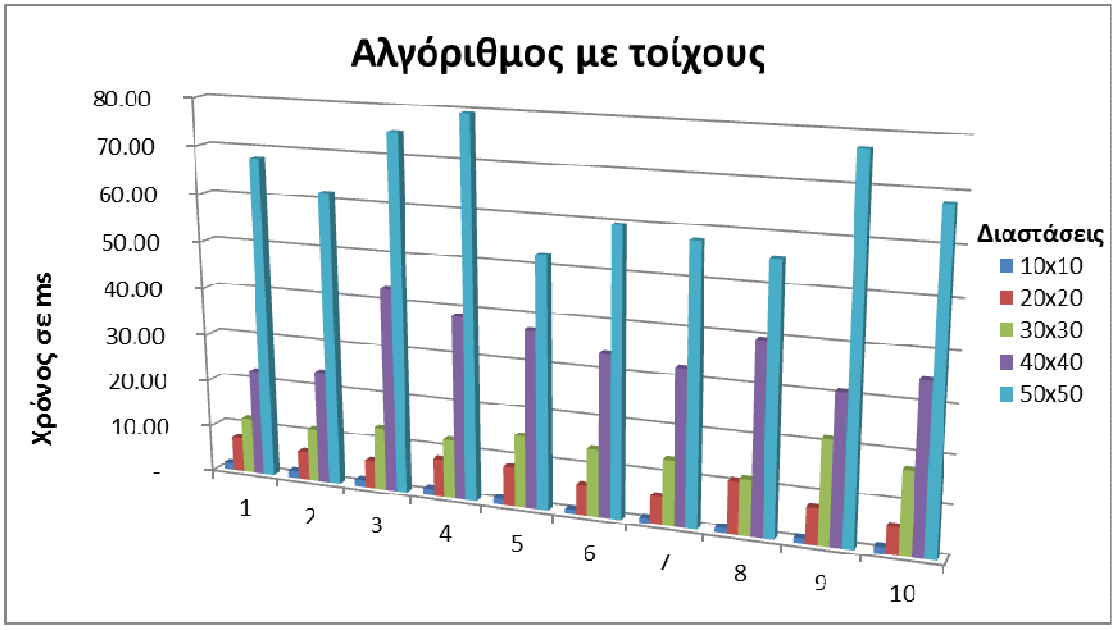
Γράφημα 3

Στο γράφημα αυτό γίνεται σύγκριση του μέσου όρου του χρόνου δημιουργίας για κάθε διάσταση. Παρατηρούμε την μεγάλη διαφορά των δύο αλγορίθμων όσον αφορά τον χρόνο δημιουργίας. Η μέγιστη μέση τιμή του αλγορίθμου με τοίχους, στις διαστάσεις 50x50 φυσικά, είναι 6 ms, ενώ για τον αλγόριθμο με δέντρο είναι 104 ms. Αυτό δείχνει πόσο πιο απλή είναι η κατασκευή του αλγορίθμου με τοίχους και πόσο πιο πολύπλοκη είναι του αλγορίθμου με δέντρο, χωρίς βέβαια αυτό να σημαίνει ότι οι λαβύρινθοι του πρώτου δεν είναι αρκετά πολύπλοκοι ή ότι οι λαβύρινθοι του δεύτερου είναι υπερβολικά αργοί. Απλώς είναι θέμα προτεραιοτήτων και προτιμήσεων του χρήστη.

ΧΡΟΝΟΣ ΕΠΙΛΥΣΗΣ ΛΑΒΥΡΙΝΘΩΝ

Παρακάτω παραθέτω μερικά διαγράμματα που αφορούν τους χρόνους επίλυσης των λαβυρίνθων των δύο αλγορίθμων. Παρόλο που ο αλγόριθμος επίλυσης είναι ο αλγόριθμος εύρεσης μονοπατιού του Dijkstra, δηλαδή δεν έχει καμία σχέση με τους αλγόριθμους δημιουργίας, μια στατιστική μελέτη του χρόνου επίλυσης θα μας δείξει πόσο δύσκολο είναι να λυθούν οι λαβύρινθοι που δημιουργούμε, συνεπώς την πολυπλοκότητά τους.

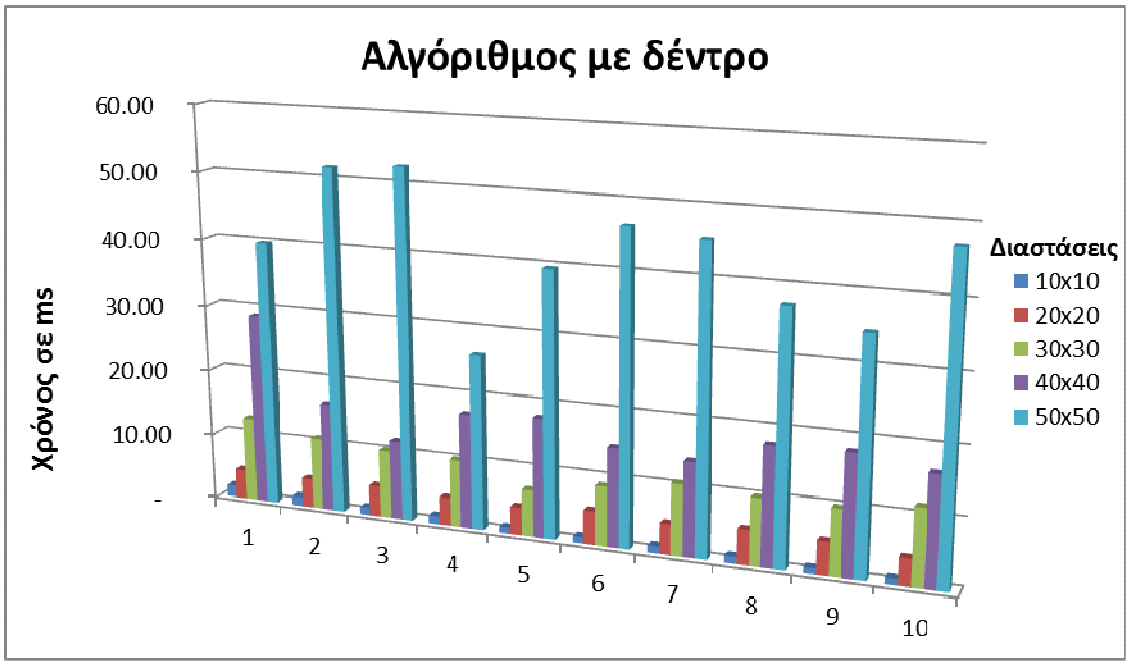
Για τον αλγόριθμο με τοίχους:



Γράφημα 4

Στο παραπάνω γράφημα φαίνεται ο χρόνος επίλυσης ενός λαβυρίνθου, με αλγόριθμο δημιουργίας τον αλγόριθμο με τοίχους. Όπως βλέπουμε οι τιμές αυξάνονται αναλογικά με τις διαστάσεις και είναι σχεδόν σταθερές μεταξύ ίδιων διαστάσεων. Συνεπώς, δείχνει, όπως και στη διαδικασία δημιουργίας, σταθερότητα και συνέπεια όσον αφορά την επίλυση του λαβυρίνθου. Μέγιστος χρόνος επίλυσης στα στατιστικά μας είναι 80 ms.

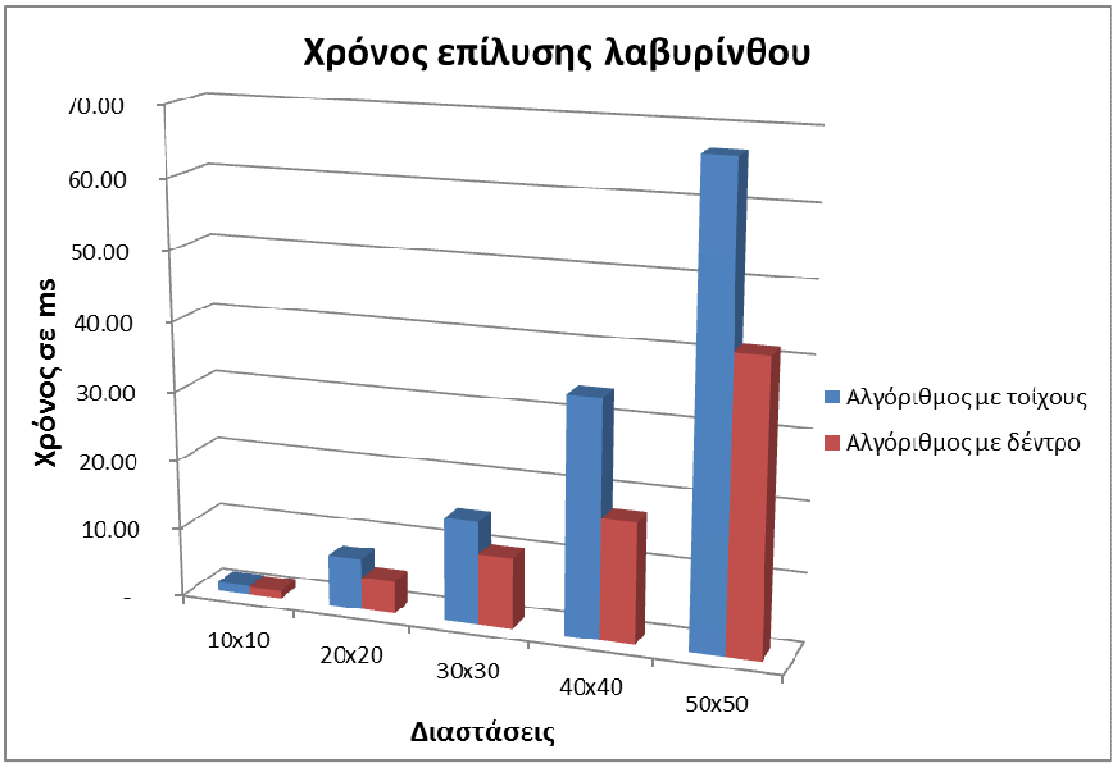
Για τον αλγόριθμο με δέντρο:



Γράφημα 5

Παρομοίως και στον αλγόριθμο με δέντρο υπάρχει σταθερότητα μεταξύ ίδιων διαστάσεων και οι χρόνοι επίλυσης αυξάνονται αναλογικά με τις διαστάσεις. Μέγιστος χρόνος επίλυσης είναι 52 ms που σημαίνει ότι είναι πιο γρήγορη η επίλυσή του σε σχέση με τον αλγόριθμο με τοίχους κατά 28 ms.

Συγκριτικά για τους δύο αλγορίθμους:



Γράφημα 6

Παρατηρούμε στο παραπάνω γράφημα ότι ο αλγόριθμος με δέντρο επιλύεται ταχύτερα του αλγορίθμου με τοίχους. Ο μέγιστος μέσος χρόνος επίλυσης του αλγορίθμου με τοίχους είναι 67 ms, ενώ του αλγορίθμου με δέντρο 47 ms. Είδαμε νωρίτερα ότι σε χρόνους δημιουργίας λαβυρίνθων ο αλγόριθμος με τοίχους υπερτερούσε του δεύτερου, ενώ σε χρόνους επίλυσης συμβαίνει το αντίθετο. Αυτό σημαίνει ότι τα μονοπάτια του αλγορίθμου με δέντρο είναι πιο βατά και βρίσκονται πιο εύκολα απ' τον αλγόριθμο επίλυσης / εύρεσης μονοπατιού. Αυτό δε σημαίνει βέβαια ότι ο αλγόριθμος με τοίχους δεν είναι σωστός προγραμματιστικά ή σχεδιαστικά. Και πάλι βέβαια εξαρτάται απ' το λόγο που το θέλει ο χρήστης. Αν χρειάζεται δύσβατους και ταχείς στη δημιουργία λαβυρίνθους επιλέγει τον αλγόριθμο με τοίχους, αν χρειάζεται πιο βατά μονοπάτια και ταχεία επίλυση επιλέγει τον αλγόριθμο με δέντρο.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Συνοπτικά, στην εργασία αυτή έχω χρησιμοποιήσει δύο αλγορίθμους κατασκευής λαβυρίνθων τον «αλγόριθμο με τοίχους» και τον «αλγόριθμο με δέντρο», οι οποίοι είναι βελτιωμένες παραλλαγές του αλγορίθμου Aldous-Broder. Επίσης, για αλγόριθμο επίλυσης χρησιμοποίησα τον αλγόριθμο συντομότερου μονοπατιού του Dijkstra. Με συγκρίσεις που έκανα μεταξύ των δύο αλγορίθμων κατέληξα στο συμπέρασμα ότι ο αλγόριθμος με τοίχους υπερτερεί κατά πολύ του αλγόριθμου με δέντρο στο χρόνο κατασκευής λαβυρίνθου, αλλά υστερεί στο χρόνο επίλυσης. Τέλος, φαίνεται οπτικά και προγραμματιστικά, ότι ο αλγόριθμος με δέντρο κατασκευάζει πολυπλοκότερους λαβυρίνθους με περισσότερες διαδρομές.

Συνεπώς, με βάση τις συγκρίσεις αυτές προτείνω την χρήση του αλγορίθμου με δέντρο για τις περισσότερες περιπτώσεις καθ’ ότι δημιουργεί πολυπλοκότερους λαβυρίνθους και η επίλυσή του είναι ταχύτερη του αλγορίθμου με τοίχους. Ωστόσο, σε περιπτώσεις που υστερούμε σε επεξεργαστική ισχύ και χρειαζόμαστε ταχεία δημιουργία λαβυρίνθων προτείνω τη χρήση του αλγορίθμου με τοίχους.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Adrian Fisher Design Ltd. (n.d.). *Mirrormaze*. Ανάκτηση από http://www.mirrormaze.com

Aldous, D. (1990). The random walk construction of uniform spanning trees. *SIAM Journal on Discrete Mathematics*, 450-465.

Berg, C. (2002). *Maze Types and Topology: A Summary*. Ανάκτηση από Amazeing Art: http://amazeingart.com/maze-faqs/maze-types.html

Broder, A. (1989). Generating random spanning trees. *In Proc. 30th Ann. IEEE Symp. on Foundations of Computer Science*, (σσ. 442-453).

Labyrinthos. (n.d.). Ανάκτηση από http://www.labyrinthos.net

Levitin, A. (2003). *Introduction to the design & analysis of algorithms.* Addison-Wesley.

MU, F. o. (2008). *Mazes*. Ανάκτηση από Faculty of Informatics Masaryk University: http://www.fi.muni.cz/~xfoltin/mazes/

Pedersen, H. and Singh, K. (2006). *Organic labyrinths and mazes*. In Proceedings of the 4th international symposium on Non-photorealistic animation and rendering (NPAR '06). ACM, New York, NY, USA, 79-86. http://doi.acm.org/10.1145/1124728.1124742

Pullen, W. (2005). *Maze Classification*. Ανάκτηση από Astrolog.org: http://www.astrolog.org/labyrnth/algrithm.htm

Roberts, E. S. (2006). *Thinking Recursively with Java* (20th Anniversary Edition εκδ.). John Wiley.

Roberts, F. S., & Tesman, B. (2009). *Applied combinatorics.* CRC Press.

Sagan, H. (1994). *Space-Filling Curves.* New York: Springer-Verlag.

Sedgewick, R. (2003). *Algorithms in Java, Part 5: Graph Algorithms* (3rd εκδ.). Addison-Wesley Professional.

Thomas H. Cormen, C. E. (2009). *Introduction to Algorithms* (3rd εκδ.). MIT Press.

Wikipedia. (n.d.). Ανάκτηση από Maze: http://en.wikipedia.org/wiki/Maze#Mazes_in_psychology_experiments

Γκέσος, Π. (2006). *Κατασκευή και Επίλυση Λαβυρίνθου*. Ανάκτηση από Τα Ανοικτού Κώδικα Προγράμματα μου: http://programs.agiasofia.gr/maze/algorithm/

ΧΡΗΣΙΜΑ LINKS

Κατασκευή και επίλυση λαβυρίνθου:

http://programs.agiasofia.gr/maze/algorithm/

Neohumanism: Maze generation algorithm:

http://neohumanism.org/m/ma/maze_generation_algorithm.html

Mazeworks: How to build a maze:

http://www.mazeworks.com/mazegen/mazetut/index.htm

Introduction to Mazes and Labyrinths:

http://gwydir.demon.co.uk/jo/maze/intro/index.htm

Welcome to Walter's Maze Mansion!:

http://www.astrolog.org/labyrnth/maze.htm

Wikipedia: Maze generation algorithm:

http://en.wikipedia.org/wiki/Maze_generation_algorithm

Maze generation: Algorithm recap:

http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

Maze generation: Recursive backtracking:

http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking

Dijkstra's Algorithm:

http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijkstraAlgor.htm

Wikipedia: Maze:

http://en.wikipedia.org/wiki/Maze

Pathfinding Dijkstra's Algorithm:

http://www.gamedeveloperskills.com/?p=13