ΠΛΗ417 – Τεχνητή Νοημοσύνη
1η Προγραμματιστική εργασία – Μέρος Α

LAB41744570
Μανώλης Πετράκος ΑΜ 2014030009

Για την υλοποίηση του πρώτου μέρους, έχουν δημιουργηθεί τρεις νέες κλάσεις. Στην abstract κλάση *State* υπάρχουν γενικές μέθοδοι και μεταβλητές που χρησιμοποιούνται σε offline και online αλγόριθμους. Στην κλάση *OfflineState* υλοποιούνται οι αλγόριθμοι BFS, DFS και A* ενώ στην κλάση *OnlineState* υλοποιείται ο αλγόριθμος LRTA*.

Η main μέθοδος παραμένει στην κλάση *GridGenerator* και έχει προστεθεί ένα μενού επιλογής αλγορίθμου. Αν επιλεχθεί ένας offline αλγόριθμος καλείται η μέθοδος *CalculateStepsOffline*. Σε αυτή, κατασκευάζεται μία κατάσταση για την αρχή του δέντρου αναζήτησης, γίνεται έλεγχος ότι η αρχή δεν συμπίπτει με το τέρμα και καλείται ο κατάλληλος αλγόριθμος. Τέλος, εκτυπώνει το κόστος αναζήτησης και αν βρέθηκε λύση εξάγει το βέλτιστο μονοπάτι μαζί με το κόστος του. Αν επιλεχθεί ένας online αλγόριθμος χρησιμοποιείται η μέθοδος CalculateStepsOnline για την αντίστοιχη λειτουργία.

1) Αλγόριθμοι BFS, DFS και A*

Εξήγηση προσέγγισης:

Στους offline αλγόριθμους, κάθε κατάσταση αντιστοιχεί σε ένα κόμβο στο δέντρο αναζήτησης. Μια θέση του Grid μπορεί να βρίσκεται σε παραπάνω από μία καταστάσεις, αλλά κάθε κατάσταση προσπελάζεται μια φορά. Για αυτό, ως κόστος εκτέλεσης ενός αλγόριθμου θεωρείται το πλήθος των καταστάσεων, δηλαδή το μέγεθος του δέντρου του. Και οι τρεις αλγόριθμοι συνεχίζουν να ψάχνουν και μετά την εύρεση μια λύσης, καθώς μπορεί να υπάρχει κάποια πιο φθηνή. Επιστρέφουν το κόμβο του δέντρου που αντιστοιχεί σε αυτήν και σε περίπτωση που δεν υπάρχει λύση, επιστρέφουν ένα κενό κόμβο.

Η δημιουργία νέων καταστάσεων γίνεται με το ίδιο τρόπο και στους τρεις αλγορίθμους. Η μέθοδος *CreateChildren* δοκιμάζει όλες τις πιθανές κινήσεις (πάνω, κάτω, αριστερά, δεξιά) και για αυτές που είναι αποδεκτές, κατασκευάζεται μια νέα κατάσταση. Ο έλεγχος των κινήσεων γίνεται στην μέθοδο *IsValidMove* και οι λόγοι απόρριψης μιας κίνησης είναι:

- Οδηγεί εκτός των ορίων του Grid.
- Καταλήγει σε τείχος.
- Έχει βρεθεί φθηνότερη λύση από το αποτέλεσμα της κίνησης.

- Καταλήγει σε μία θέση από όπου έχει ξαναπεράσει το μονοπάτι της, δηλαδή επιστρέφει σε μια κατάσταση πρόγονο.
- Για την θέση που καταλήγει, υπάρχει ήδη μια κατάσταση με μικρότερο ή ίσο κόστος. Δηλαδή έχει βρεθεί ένα φθηνότερο μονοπάτι προς αυτήν τη θέση.

Ο αλγόριθμος BFS υλοποιείται στην συνονόματη μέθοδο *BFS*. Κατασκευάζεται μια λίστα που λειτουργεί σαν FIFO ουρά και εισάγεται ο αρχικός κόμβος. Δημιουργούνται τα παιδιά του, ελέγχονται αν βρίσκονται στον στόχο και εφόσον δεν είναι μπαίνουν στο τέλος της ουράς. Αυτή η διαδικασία συνεχίζεται μέχρι να μην μπορούν να δημιουργηθούν νέες καταστάσεις και να αδειάσει η ουρά. Επιστρέφει τον κόμβο που αντιστοιχεί στο τέρμα με το φθηνότερο μονοπάτι.

Ο αλγόριθμος DFS υλοποιείται με παρόμοιο τρόπο, με μόνη διαφορά ότι η λίστα λειτουργεί σαν stack, δηλαδή τα παιδιά ενός κόμβου μπαίνουν στην αρχή της. Έτσι, ένα παιδί ελέγχεται μετά τους απόγονους του προηγούμενου παιδιού.

Ο αλγόριθμος A* διαχωρίζει τις καταστάσεις σε αυτές που έχει ελέγξει αν είναι στον στόχο και έχει κατασκευάσει τους απόγονους τους, και σε αυτές που έχει δημιουργήσει αλλά δεν έχει ελέγξει. Επιλέγει από την δεύτερη κατηγορία την κατάσταση με το ελάχιστο εκτιμώμενο κόστος, ελέγχει αν είναι η φθηνότερη λύση και εφόσον είναι την κρατάει. Ως εκτιμώμενο κόστος, θεωρείται το κόστος μέχρι να φτάσει σε αυτό το σημείο συν την απόσταση Manhattan από αυτό το σημείο ως το στόχο. Αν η κατάσταση δεν βρίσκεται στο τέρμα, κατασκευάζονται τα παιδιά της. Απορρίπτονται αυτά που η θέση τους υπάρχει σε άλλη κατάσταση στην πρώτη κατηγορία, καθώς έχει γίνει ήδη η απαραίτητη επεξεργασία για αυτήν. Επίσης, απορρίπτονται αυτά όπου υπάρχει αντίστοιχη κατάσταση στην δεύτερη κατηγορία με φθηνότερο μονοπάτι. Αν το μονοπάτι είναι ακριβότερο, την αντικαθιστούν. Οι καταστάσεις-παιδιά που παραμένουν μπαίνουν στην δεύτερη κατηγορία, ενώ η κατάσταση-πατέρα εισάγεται στην πρώτη. Αυτή η διαδικασία συνεχίζεται μέχρι να αδειάσει η δεύτερη κατηγορία ακόμα και αν έχει βρεθεί μια λύση καθώς σε περίπτωση που υπάρχουν βήματα με μεγάλες διαφορές κόστους, η πρώτη λύση μπορεί να μην είναι η βέλτιστη.

Τέλος υπάρχει η μέθοδος *ExtractSolution* η οποία κατασκευάζει το μονοπάτι της βέλτιστης λύσης. Κρατάει τις θέσεις των καταστάσεων ξεκινώντας από αυτήν με την βέλτιστη λύση και ανεβαίνοντας μέχρι την ρίζα του δέντρου. Χρησιμοποιείται και για τους τρεις αλγορίθμους.

Απόδοση αλγόριθμων:

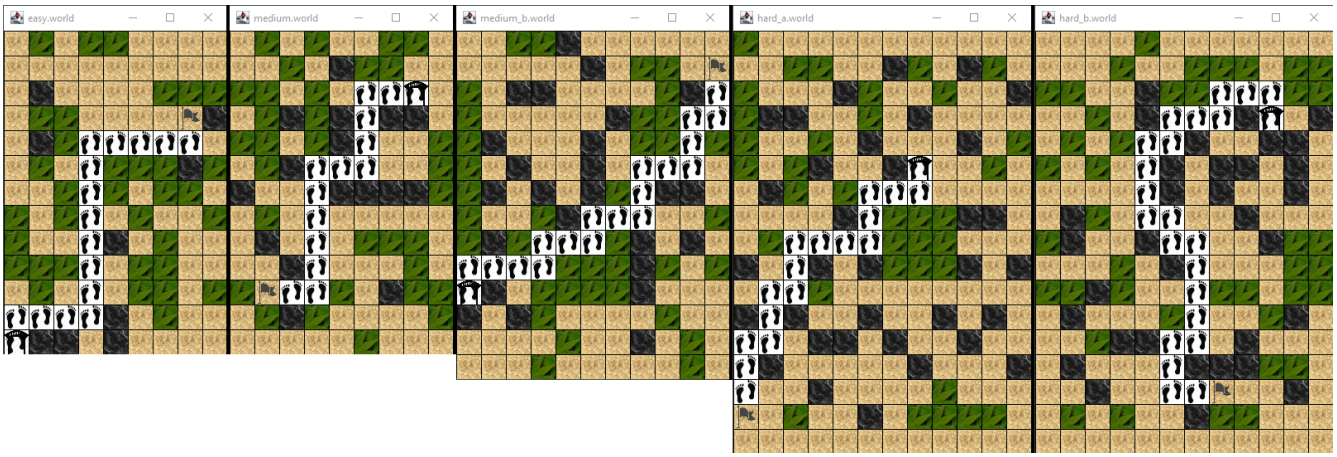
	BFS / Grass Cost = 2					BFS / Grass Cost = 10				
Κόστος αναζήτησης	252	134	236	176	261	299	226	363	247	516
Κόστος λύσης	19	20	21	18	31	40	45	37	26	62

	DFS / Grass Cost = 2					DFS / Grass Cost = 10				
Κόστος αναζήτησης	1234	805	784	907	1357	1120	904	518	1293	1579
Κόστος λύσης	19	20	21	18	31	40	45	37	26	62

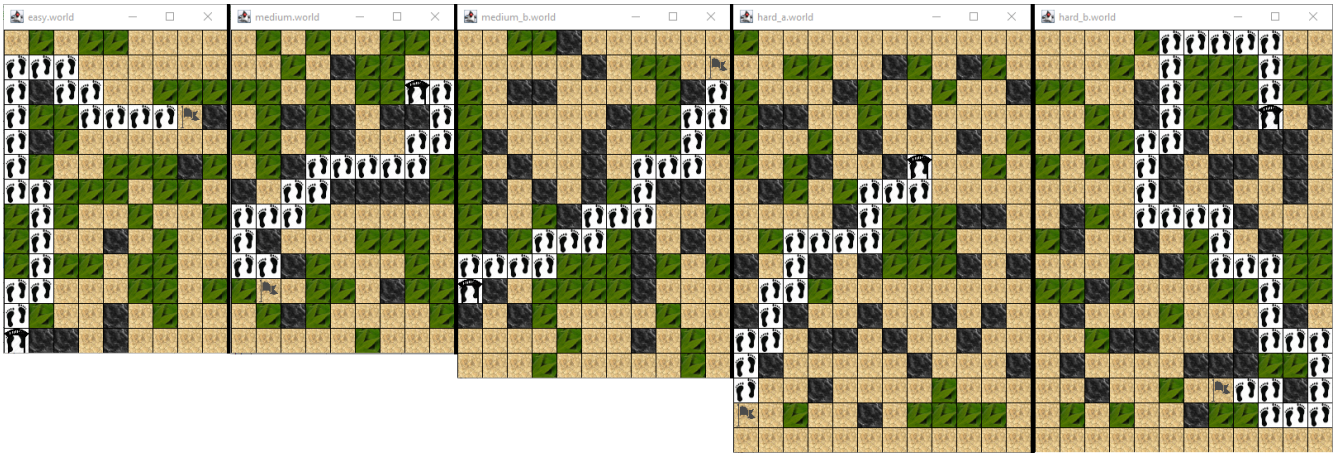
	A* / Grass Cost = 2					A* / Grass Cost = 10				
Κόστος αναζήτησης	121	114	146	196	193	111	102	131	179	175
Κόστος λύσης	19	20	21	18	31	40	45	37	26	62

Εικόνες Κόσμων:

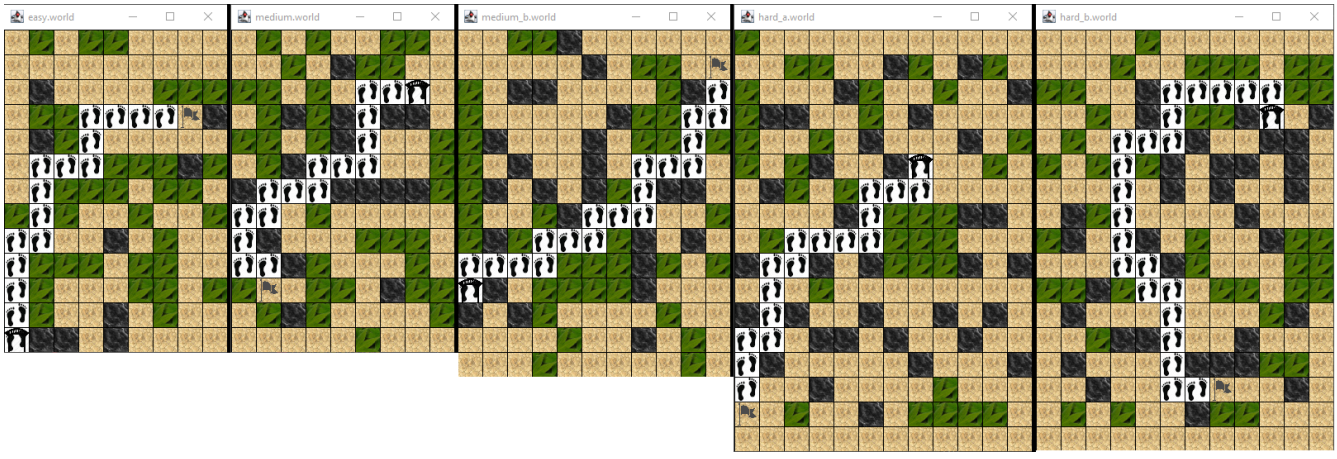
BFS / Grass Cost = 2



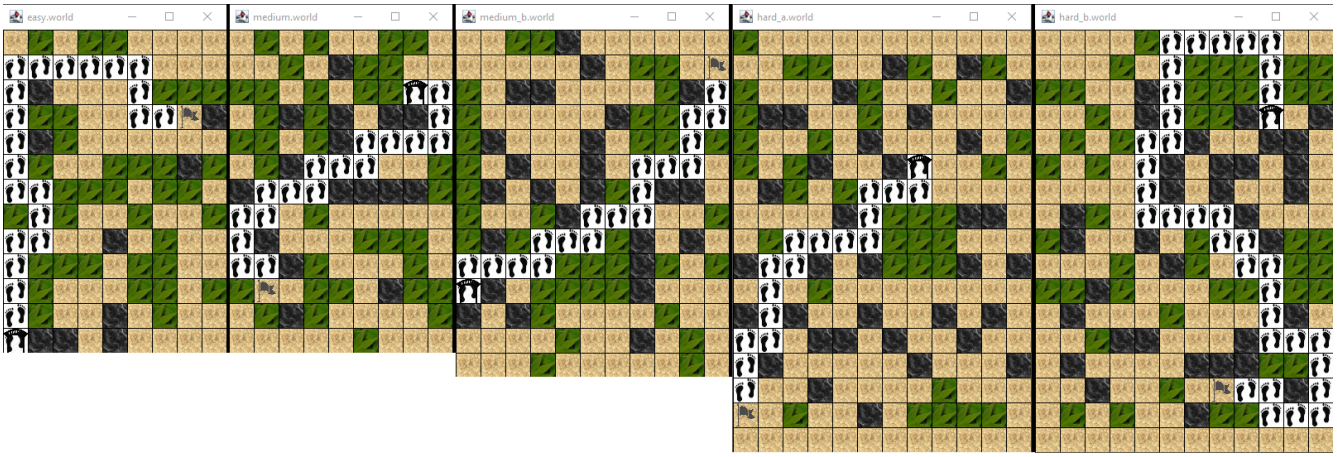
BFS / Grass Cost = 10



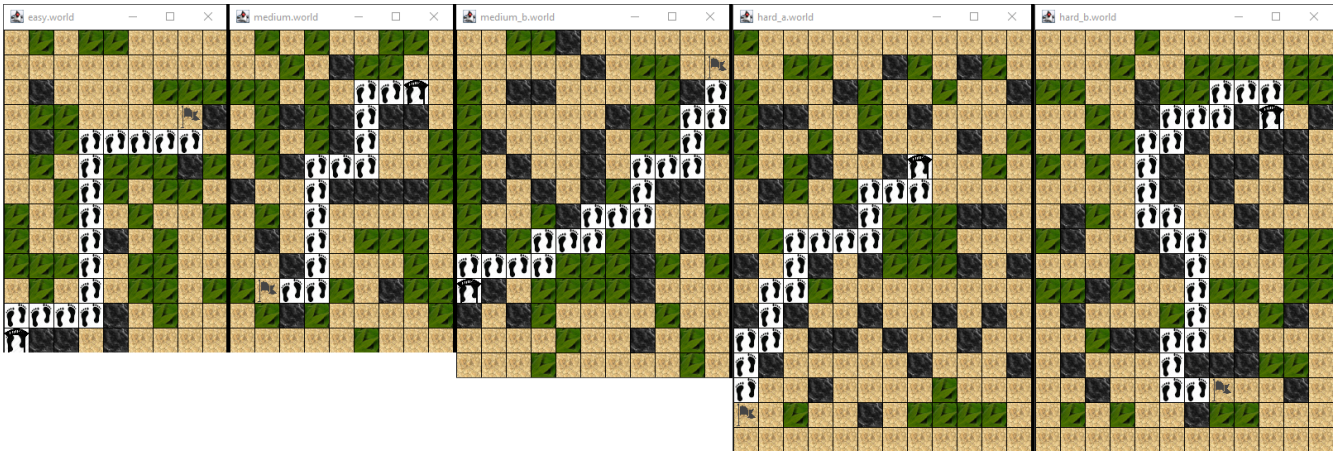
DFS / Grass Cost = 2



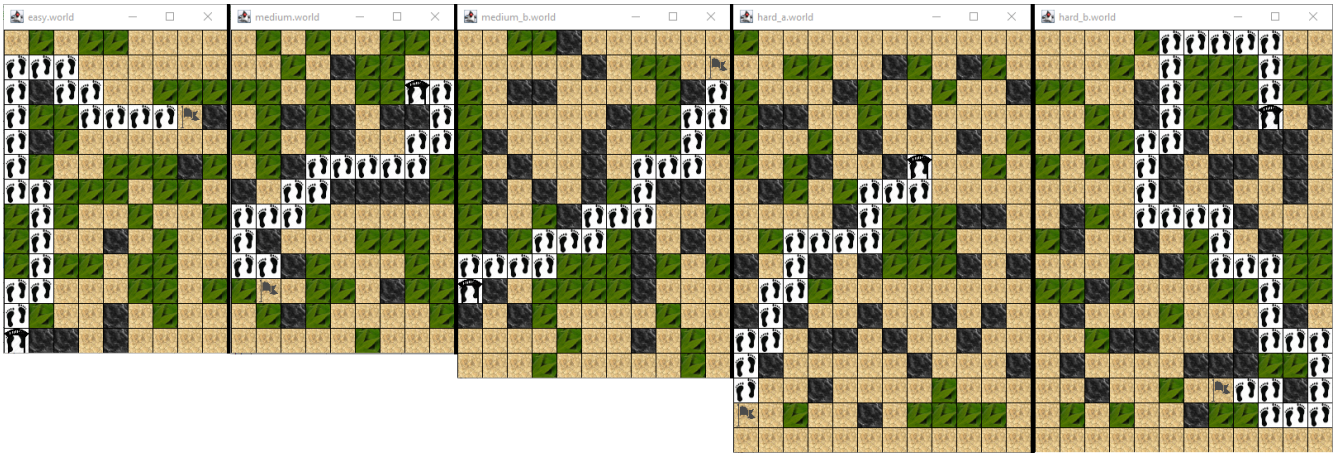
DFS / Grass Cost = 10



A / Grass Cost = 2*



A / Grass Cost = 10*



Από τις μετρήσεις και τις εικόνες φαίνεται ότι και οι τρεις αλγόριθμοι βρίσκουν την βέλτιστη λύση, αλλά με αρκετά διαφορετική απόδοση. Ο DFS έχει διαρκώς την χειρότερη επίδοση ενώ ο A* την καλύτερη. Επίσης ο A* επηρεάζεται ελάχιστα σε σχέση με τους άλλους από το μέγεθος του κόσμου και το κόστος των κελιών.

1) Αλγόριθμος LRTA*

Εξήγηση προσέγγισης:

Ο αλγόριθμος LRTA* δημιουργεί ένα γράφο καταστάσεων, όπου η κάθε μία αντιστοιχεί σε μια θέση ενός κόσμου. Για κάθε θέση κατασκευάζει το πολύ μία κατάσταση (όταν την ανακαλύπτει πρώτη φορά), αλλά μπορεί να την προσπελάσει πολλές φορές. Για αυτό, ως κόστος του αλγόριθμου θεωρείται το πλήθος των κινήσεων που θα κάνει μέχρι να φτάσει στον στόχο.

Ο αλγόριθμος ξεκινάει βλέποντας ποιες είναι οι δυνατές κινήσεις που μπορεί να κάνει στην θέση που βρίσκεται. Σε σχέση με τους offline αλγόριθμους, οι περιορισμοί είναι πιο ελαστικοί και οι λόγοι απόρριψης είναι αν καταλήγουν εκτός του κόσμου ή σε τείχος που έχει ήδη ανακαλυφθεί. Για τις κινήσεις που γίνονται δεκτές, υπολογίζεται το εκτιμώμενο κόστος των θέσεις που καταλήγουν. Αν η θέση είναι ανεξερεύνητη, το εκτιμώμενο κόστος είναι ίσο με την απόσταση Manhattan από αυτήν έως το τέρμα. Αντίθετα, εάν η θέση είναι γνωστή, το εκτιμώμενο κόστος είναι το κόστος της θέσης συν την απόσταση Manhattan της υπόλοιπης διαδρομής. Αν η μικρότερη εκτίμηση είναι μεγαλύτερη από την εκτίμηση της θέσης που βρίσκεται πριν την κίνηση, η δεύτερη παίρνει την τιμή της πρώτης, καθώς είναι το ελάχιστο ρεαλιστικό κόστος για αυτήν.

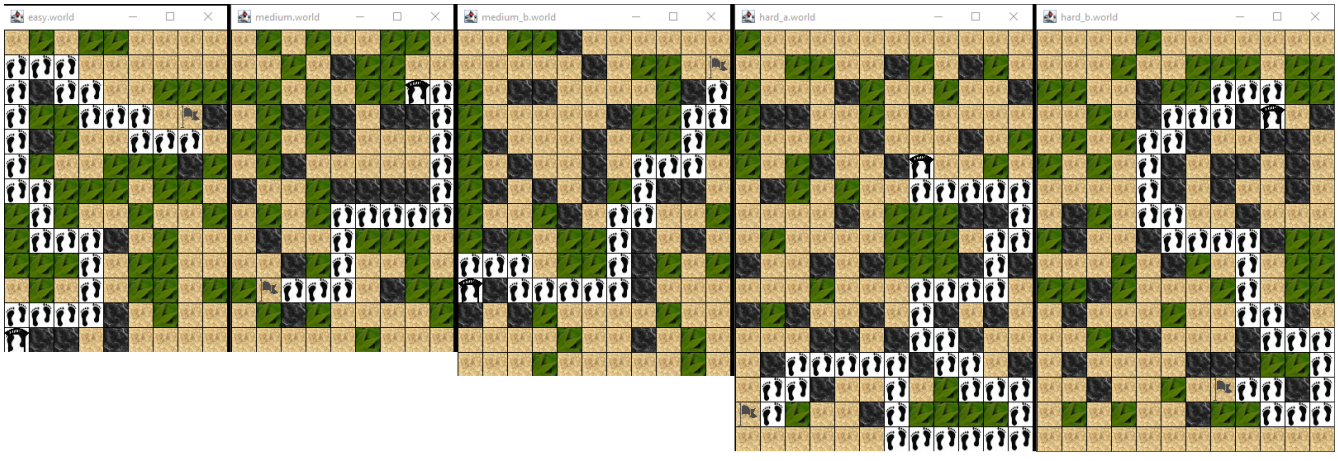
Τότε, ο αλγόριθμος πηγαίνει στην φθηνότερη γειτονική θέση και εάν δεν έχει ξαναπεράσει από αυτή, την ανακαλύπτει. Αν δει ότι έφτασε σε τείχος, υποχρεωτικά γυρνάει πίσω. Αν φτάσει σε χόρτα και μόλις ανακάλυψε την θέση ή δεν έχει εξερευνήσει τους υπόλοιπους γείτονες της προηγούμενης, μειώνεται το εκτιμώμενο κόστος της προηγούμενης. Αυτό γίνεται για να δοθεί κίνητρο στον πράκτορα να εξερευνήσει περισσότερο και να βρει μονοπάτια με λίγα χόρτα.

Ο αλγόριθμος επαναλαμβάνει αυτή τη διαδικασία μέχρι να βρεθεί το τέρμα, ενώ αν δεν βρεθεί συνεχίζει συνέχεια. Τέλος, υπάρχει η μέθοδος *ExtractPath* για την κατασκευή του καλύτερου μονοπατιού από το ιστορικό του αλγόριθμου καθώς και τον υπολογισμό του κόστους του.

Απόδοση αλγόριθμου και εικόνες κόσμων:

	LRTA* / Grass Cost = 10				
Κόστος αναζήτησης	234	336	137	73	809
Κόστος λύσης	46	61	75	39	103

LRTA / Grass Cost = 10



Πηγές:
BFS, DFS
A*
Manhattan Distance
LRTA*