ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

1Η ΑΣΚΗΣΗ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Ακ.Έτος 2013-2014,7ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

<u>Στοιχεία Ομάδας Φοιτητών:</u>

Αφεντουλίδης Γρηγόρης (03107760)

Κολοτούρος Δημήτρης (03107116)

1. Το πρόβλημα που κληθήκαμε να επιλύσουμε σε αυτήν την άσκηση αφορούσε την διαδοχική αναζήτηση στο χώρο καταστάσεων πιθανών διαδρομών προς κάποιο ορισμένο στόχο με τη χρήση του αλγορίθμου A*. Συγκεκριμένα μας δίνεται ως είσοδος μία κάτοψη ενός χώρου καθορισμένων διαστάσεων και οι θέσεις ενός Robot "κυνηγού"(Robot 1) και ενός Robot ''κυνηγημένου''(Robot 2). Το κυνήγι ορίζεται από γύρους όπου σε κάθε έναν από αυτούς το Robot 2 κάνει μία τυχαία κίνηση κατά ένα μόνο βήμα και αμέσως μετά το Robot 1 οργανώνει την αναζήτηση του στο χώρο καταστάσεων (που είναι οι πιθανές θέσεις μέσα στο δωμάτιο) προσπαθώντας να βρεί ένα μονοπάτι προς το Robot 2 με βάση την τελευταία θέση που αυτό κατέχει. Το αποτέλεσμα αυτής την αναζήτησης έχει δύο πιθανά αποτελέσματα, είτε δεν υπάρχει πιθανή διαδρομή οπότε έχουμε αδιέξοδο (π.χ λόγων εμποδίων μέσα στο χώρο),είτε υπάρχει διαδρομή και τότε το Robot 1 προχωράει σύμφωνα με τα 2 πρώτα βήματα της επιστρεφόμενης διαδρομής. Τελικά αναμένουμε ότι το Robot 1 κάποια στιγμή θα φτάσει το Robot 1 και πλέον το πρόβλημά μας έχει λυθεί.Τέλος για διαφορετικά μεγέθη input δοκιμάσαμε να λύσουμε το πρόβλημα με admissible και inadmissible ευριστικές συναρτήσεις και εξάγαμε κάποια εδιαφέροντα στατιστικά στοιχεία σε σχέση με τις συνολικά κατασκευασθείσες καταστάσεις και τον χρόνο εκτέλεσης κάθε φορά.
Στο κομμάτι της υλοποίησης επιλέξαμε να προγραμματίσουμε σε Ruby που είναι μια σύγχρονη αντικειμενοστραφής γλώσσα σεναρίων που προσφέρει μεγάλη εκφραστικότητα στο προγραμματιστή καθώς και ένα ισχυρό core library,με αντίκτυπο όμως την ταχύτητα εκτέλεσης καθώς είναι interpreted γλώσσα που δεν υπόκειται σε βελτιστοποιήσεις κώδικα αφού δεν παρεμβάλεται κάποιος compiler.

2. Η πρώτη δομή δεδομένων που χρειστήκαμε ήταν αυτή που θα αναλάμβανε να αναπαριστά την κάτοψη του χώρου του προβλήματος. Για αυτό το σκοπό χρησιμοποιήσαμε ένα διδιάστατο πίνακα χαρακτήρων όπου η κάθε κατειλλημένη από εμπόδιο θέση ήταν ένα χαρακτήρας "x" και η κάθε ελεύθερη ένας χαρακτήρας "o". Στο κύριο script όπου διαβάζουμε αρχικά την είσοδο αυτή η δομή φτιάχτηκε από το παρακάτω κομμάτι κώδικα:

```
map = Array.new(dim1.to_i)            #dim1 is the number of lines
dim1.to_i.times do |i|
map[i] = f1.readline.split(//)[0..-2]
end
```

Ο χάρτης αυτός έπειτα περνάει ως παράμετρος σε όσα κομμάτια κώδικα τον χρειάζονται.

Επίσης η αναπαράσταση κάθε κατάστασης στην αναζήτηση A* έγινε με την κλάση **Node**.Για κάθε μία καινούργια κατάσταση αρχικοποιούμε ένα καινούργιο αντικείμενο αυτής της κλάσης όπου έχει μεταβλητές υπόστασης που αναφέρουν την θέση στο χώρο,την τιμή της ευριστικής συνάρτησης του κόμβου,το κόστος μετάβασης,ένα δείκτη στον κόμβο-κατάσταση πατέρα και το συνολικό τρέχον κόστος που είναι το άθροισμα ευριστικής και κόστους μετάβασης.Ο κώδικας:

```
class Node
attr_accessor :x,:y,:heuristic,:access_cost,:parent,:next,:t_cost
  def initialize(dimx,dimy,hf,ac,p)
    @x,@y,@heuristic,@access_cost,@parent = dimx,dimy,hf,ac,p
    @next=nil
    @t_cost = @heuristic + @access_cost
  end
end
```

Επίσης η μεταβλητη @next μας βοηθάει να φτιάξουμε μια συνδεδεμένη λίστα από κόμβους-καταστάσεις όταν πλέον έχει βρεθεί η διαδρομή, αφού σε αυτήν καταχωρούμε τον απόγονο ενός δεδομένου κόμβου πάνω στο ζητούμενο μονοπάτι.

Ακόμα μέσα στην κλάση Astar που αναλαμβάνει να τρέξει τον εν λόγω αλγόριθμο χρησιμοποιήσαμε ένα διδιάστατο πίνακα ίδιου μεγέθους με το χάρτη του χώρου με Boolean τιμές(αρχικοποιημένο με false) που αναφέρουν αν κατά την διάρκεια της αναζήτησης έχουμε επισκεφτεί ένα δεδομένο κόμβο. Αυτό υλοποιήθηκε όπως φαίνεται παρακάτω:

```
@visited = Array.new(@xmap){Array.new(@ymap,false)}
```

Ουσιαστικά με αυτή τη δομή καταφέραμε να έχουμε ένα closed set καταστάσεων που μας απέτρεπε να μπαίνουμε σε loops κατά την διάρκεια εκτέλεσης του αλγορίθμου.

Επίσης μία ακόμα δομή που χρειάστηκε να υλοποιήσουμε ήταν αυτή της ουράς προτεραιότητας ελαχίστου που είναι απαραίτητη για την υλοποίηση του A*. Αυτή υλοποιήθηκε ως μία κλάση με όνομα **MinHeap** που όπως φανερώνει και το όνομά της χρησιμοποιεί ένα heap ελαχίστου εσωτερικά. Σκοπός της φυσικά είναι αφού έχουμε ένα αντικείμενο αυτής, να

μας παρέχει ένα API όπου θα μπορούμε να αναφερόμαστε σε βασικές λειτουργίες της που έχουμε ανάγκη. Αυτό το API αποτελείται από τις μεθόδους **pop,push,get και empty?** με τις 2 πρώτες να είναι για να βγάλουμε ένα στοιχείο από την ουρά και αντίστοιχα να εναποθέσουμε ένα(στην περίπτωσή μας node objects) ενώ η get επιστρέφει την ουρά ολόκληρη και η empty? αναφέρει αν είναι άδεια ή όχι κάποια δεδομένη στιγμή. Ο κώδικάς της κλάσης επειδή είναι εκτενής δεν δίνεται εδώ.

Τέλος κατασκευάσαμε και μία δομή που αναπαριστά τα Robot που είναι ουσιαστικά μια κλάση με το όνομα Robot η οποία έχει μεταβλητές θέσης και μια μεταβλητή ώστε να αποθηκεύει εσωτερικά την κάτοψη του χώρου του προβλήματος. Σε αυτήν ενσωματώνονται και 2 μέθοδοι οι **report** και **moveRandom** με την μεν πρώτη να αναλαμβάνει να ανακοινώνει την τρέχουσα θέση του Robot ενώ η δεύτερη δεδομένης της τρέχουσας θέσης και των επιτρεπόμενων κινήσεων γύρω από αυτήν διαλέγει ισοπίθανα πάντα μια νέα θέση (ουσιαστικά η μέθοδος αυτή είναι αναγκαία μόνο για το Robot 2).

3. Η υλοποίηση του A* στο πρότυπο του αντικειμονοστραφούς ύφους που από την φύση της προωθεί αυστηρά και η ruby υλοποιήθηκε με την μορφή μια κλάσης. Αντικείμενα αυτή της κλάσης γίνονται instantiate παρέχοντάς τους ένα δεδομένο πρόβλημα αναζήτησης κάθε φορά. Συγκεκριμένα λοιπόν δίνεται ο ‘’χάρτης’’ του προβλήματος οι αρχικές συντεταγμένες και οι τελικές συντεταγμένες του προβλήματος και η κλάση παρέχει την μέθοδο **run** που αναλαμβάνει να επιλύσει το πρόβλημα με τον αλγόριθμο A* χρησιμοποιώντας εσωτερικά μεθόδους-τελεστές που παρέχει το σώμα της κλάσης.Παραθέτουμε τις μεθόδους αυτές με μία συνοπτική εξήγηση παρακάτω:

Μέθοδος startNode
Η συγκεκριμένη μέθοδος δημιουργεί τον αρχικό κόμβο-κατάσταση του προβλήματος από όπου θα ξεκινήσει η αναζήτηση στο δέντρο καταστάσεων.Ο αρχικός κόμβος-κατάσταση έχει μηδενική τιμή κόστους μετάβασης και πατρικό κόμβο με τιμή nil.

Μέθοδος calculateHeuristic
Η μέθοδος αυτή αναλαμβάνει κατά περίπτωση να υπολογίσει την τιμή της ευριστικής συνάρτησης που έχουμε ορίσει για το πρόβλημα και να την επιστρέψει στον καλούντα.

Μέθοδος check
Με τη μέθοδο αυτή ελέγχουμε αν έχουμε φτάσει στο στόχο της αναζήτησης ώστε να τερματιστεί ο αλγόριθμος.

Μέθοδος sameNodes
Με την sameNodes ελέγχουμε αν από ένα κόμβο κατάσταση αν κάποιος από τους κόμβους που επιχειρούμε να κάνουμε expand είναι πατρικός του συγκεκριμένου κόμβου που εξετάζουμε.

Οι τελεστές μετάβασεις από μία κατάσταση σε κάθε μία από τις γειτονικές 4 δεν έγιναν με κάποια μέθοδο αλλά με κώδικα μέσα στο σώμα της μεθόδου run της κλάσης A*. Έτσι για κάθε δεδομένο κόμβο-κατάσταση του προβλήματος ελέγχουμε πρώτα άμα είναι πιθανός κόμβος μετάβασης ως προς το αν είναι ένα εμπόδιο του δωματίου,δεν ανήκει στο closed set κόμβων που έχουμε ήδη επισκεφτεί και τέλος δεν είναι πατρικός του συγκεκριμένου κόμβου και αν και οι τρεις παραπάνω συνθήκες ικανοποιούνται τότε ο κόμβος γίνεται expand και εισάγεται στην ουρά προτεραιότητας αφού πρώτα υπολογιστεί η τιμή της ευριστικής συνάρτησης που του αναλογεί και αντίστοιχα και το κόστος μετάβασης.

Τέλος το main script της άσκησης είναι το RoomChase.rb από όπου ελέγχουμε επαναληπτικά αν το Robot 1 τελικά έφτασε το Robot 2 και αν όχι τότε αναλαμβάνουμε να κινήσουμε πρώτα το δεύτερο και να οργανώσουμε μία νέα αναζήτηση για το δεδομένο στιγμιότυπο του προβλήματος. Επίσης κρατάμε στατιστικά από κάθε τρέξιμο του αλγορίθμου όσο αναφορά το πλήθος καταστάσεων που εξετάστηκαν και τα αθροίζουμε συνολικά ενώ κρατάμε και το συνολικό χρόνο εκτέλεσης του προγράμματος.

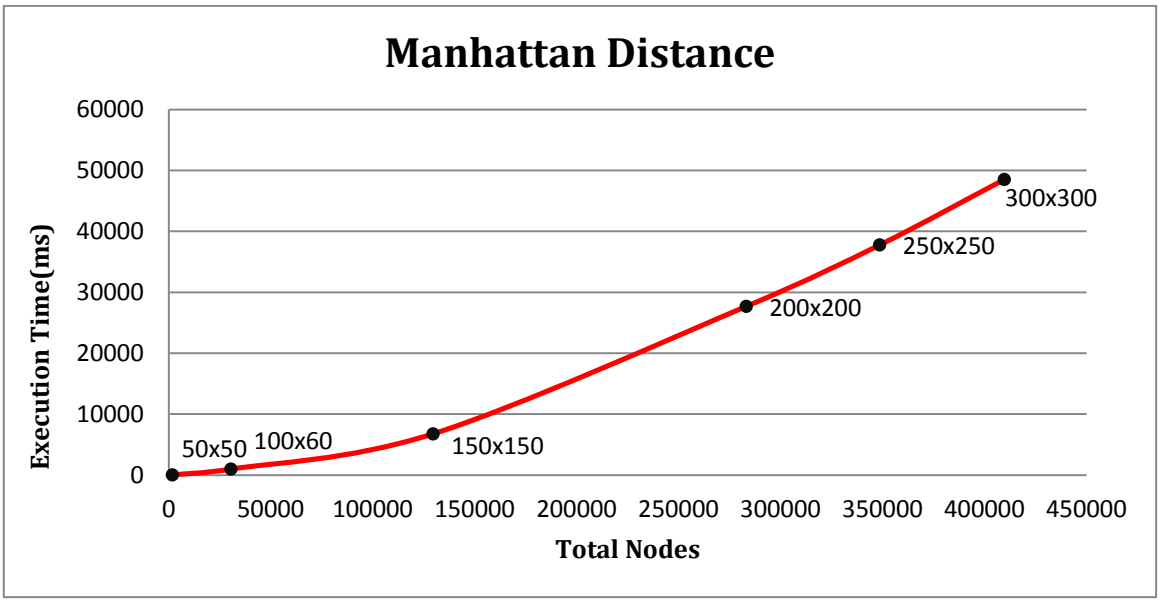
4. Οι ευριστικές συναρτήσεις που χρησιμοποιήσαμε ήταν αρχικά η Manhattan distance για την υλοποίηση με admissible heuristic που μας ζητείται ενώ για την αντίστοιχη με inadmissible χρησιμοποιήσαμε το τετράγωνο της ευκλείδιας απόστασης.

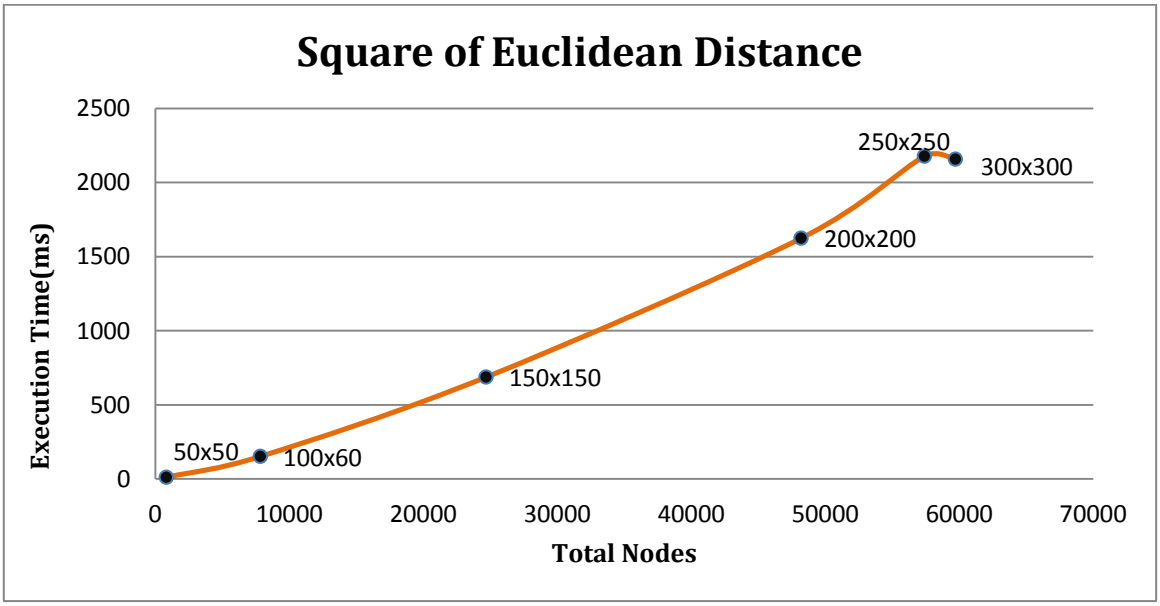
5. Για τα στατιστικά στοιχεία που μας ζητήθηκαν κατασκευάσαμε μέσω ενός script 6 διαφορετικά αρχεία που αντιστοιχούσαν σε κατόψεις χώρων με μεγέθη 50x50,100x60,150x150,200x200,250x250 και 300x300 αντίστοιχα.Οι επιλογές για τις αρχικές θέσεις των 2 robot έγιναν αυθαίρετα από εμάς.Παρακάτω παραθέτουμε το σύνολο των καταστάσεων καθώς και τους χρόνους εκτέλεσης για κάθε ένα από τα αρχεία εισόδου και για κάθε ευριστική αντίστοιχα.

Admissible Heuristic (Manhattan Distance)		
Input Files	*Total Nodes*	*Exexution Time(ms)*
input 1	1.744	24
input 2	30.481	974
input 3	129.620	6.757
input 4	283.236	27.690
input 5	348.665	37.776
input 6	409.802	48.522

Inadmissible Heuristic (Square of eucl.distance)		
Input Files	*Total Nodes*	*Exexution Time(ms)*
input 1	823	12
input 2	7.842	153
input 3	24.687	688
input 4	48.204	1.625
input 5	57.411	2.178
input 6	59.723	2.158

Τα παραπάνω αποτελέσματα σε γραφήματα:





Παρατηρώντας τα παραπάνω αποτελέσματα παρατηρούμε πολύ σημαντική διαφοροποίηση στον αριθμό των εξεταζόμενων συνολικά καταστάσεων και εντελώς ισοδύναμα και στο χρόνο εκτέλεσης,εφ'όσον ο χρόνος είναι προφανώς ανάλογος των καταστάσεων που αναγκαζόμαστε να εξετάσουμε για να λύσουμε το πρόβλημα, με την περίπτωση των εκτελέσεων με inadmissible ευριστική να υπερτερούν.

Με μία πρώτη ανάλυση αυτό είναι λογικό γιατί στην περίπτωση της αναζήτησης στο χώρο καταστάσεων με μία υπερεκτιμήτρια αναμένουμε να γίνεται μία αναζήτηση που οδηγείται να προχωρήσει σε βάθος μέσα στο δέντρο της αναζήτησης παρά να κάνει backtracking συχνά συνεχίζοντας από προηγούμενες καταστάσεις σε υψηλότερα επίπεδα του δέντρου. Αυτό μας οδηγεί να σκεφτούμε πως σε ένα instance του προβλήματος όπου υπάρχει λύση η αναζήτηση αυτού του είδους θα οδηγηθεί μέχρι το τέρμα της και θα την ανακαλύψει πολύ γρήγορα και χωρίς να αναγκαστεί να προσθέσει πολλές καταστάσεις,πόσο μάλλον όταν υπάρχουν αρκετές λύσεις στο πρόβλημα. Η αναζήτηση αυτή φαντάζει πολύ ελκυστική γιατί παρέχει μία λύση αρκετά γρήγορα ενώ ο μόνος τρόπος να αργήσει φαντάζει να είναι όταν θα επιλέγει συνέχεια λάθος διαδρομές και θα αργεί πολύ να καταλάβει ότι δεν αποτελούν λύσεις πράγμα που είναι σχετικά απίθανο,παρόλα αυτά θυσιάζει την βελτιστότητα της διαδρομής αφού δεν εγγυάται ότι η λύση θα είναι πράγματι η βέλτιστη.

Αντίθετα η admissible ευριστική συνάρτηση προτιμάει να κάνει μια ποιο διεξοδική αναζήτηση με μεγαλύτερο πλάτος συνόρου με σκοπό να εντοπίσει την βέλτιστη λύση για το πρόβλημα.Αυτό προφανώς την αναγκάζει να κάνει επέκταση πολλών καταστάσεων και άρα να είναι σημαντικά πιο αργή.

Γενικά τα παραπάνω στατιστικά δεν είναι και τα πλέον ενδεδειγμένα για την εξαγωγή ασφαλών συμπερασμάτων καθώς δεν λαμβάνουμε υπόψη διάφορες παραμέτρους που υφίστανται από την φύση του συγκεκριμένου προβλήματος και μπορούν να αλλοιώσουν τα αποτελέσματα,όπως το γεγονός ότι η κίνηση του Robot 2 είναι μεν τυχαία στο μέτρο του δυνατού,αλλά τίποτα δεν μας αποκλείει ότι η κίνηση του είναι απευθείας πάνω στο Robot 1 που το κυνηγάει πράγμα που διαφοροποιεί πολύ και το χρόνο εκτέλεσης αλλά και τις καταστάσεις που δημιουργούνται,ενώ τέλος το script που μας δημιούργησε τις αίθουσες πρόσθετε με μία τυχαιότητα εμπόδια που μπορεί να επηρέαζαν τον τρόπο που έτρεχε ο αλγόριθμος σε κάθε μία από τις περιπτώσεις.