ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΡΟΗ Λ - 7^{ο} ΕΞΑΜΗΝΟ

Project 1 : Πλοήγηση Robot στο χώρο με εφαρμογή του A*

ΟΜΑΔΑ:

ΤΖΙΑΒΕΛΗΣ ΝΙΚΟΣ
03112166
tziavelisnikos@yahoo.gr

ΡΙΣΣΑΚΗ ΑΓΑΠΗ
03112093
risssaki.agapi@gmail.com

ΗΜΕΡΟΜΗΝΙΑ ΠΑΡΑΔΟΣΗΣ: 27/12/15

1.ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ ΚΑΙ ΣΚΙΑΓΡΑΦΗΣΗ ΕΠΙΛΥΣΗΣ

Πρόβλημα:
Κληθήκαμε να πλοηγήσουμε δύο Robots σε χώρο με εμπόδια, διαθέτοντας την κάτοψη του χώρου. Πιο αναλυτικά, κάθε Robot , ξεκινώντας από κάποια αρχική θέση, πρέπει να περάσει από δεδομένα σημεία στον χώρο, τους ενδιάμεσου στόχους, και ύστερα να συναντηθεί με το άλλο Robot σε καθορισμένο σημείο συνάντησης, τον τελικό στόχο. Ως συνάντηση ορίζουμε την κατάσταση στην οποία κάποιο Robot βρίσκεται στο meeting point και το άλλο βρίσκεται δίπλα του την ίδια χρονική στιγμή. Επιπλεον, τα Robots πρέπει να γυρίσουν στις αρχικές τους θέσεις αφότου συναντηθούν. Φυσικά, καθ' όλη την διάρκεια των μετακινήσεων σημαντικό είναι να αποφεύγονται οι πιθανές συγκρούσεις είτε μεταξύ Robot και εμποδίου στον χώρο είτε μεταξύ των δύο Robots .

Σκιαγράφηση του τρόπου επίλυσης:
Επιλύουμε το πρόβλημα με χρήση της Java.
Αρχικά, για το Robot 1, εφαρμόζουμε τον αλγόριθμο αναζήτησης A* , για να πάρουμε κάποια διαδρομή που ξεκινάει από την δεδομένη αρχική του θέση, περνάει από όλους τους ενδιάμεσους στόχους και καταλήγει στο σημείο συνάντησης, αποφεύγοντας τα εμπόδια του χώρου. Αυτό το πετυχαίνουμε εκτελώντας τον αλγόριθμο αναζήτησης για κάθε υποδιαδρομή, δηλαδή μεταξύ της αρχικής θέσης και του πρώτου στόχου, μεταξύ του πρώτου και του δεύτερου στόχου κ.ο.κ. Η διαδρομή λαμβάνεται αν ακολουθήσουμε τον δείκτη προς τον γονέα κάθε κόμβου στο δέντρο αναζήτησης, ξεκινώντας από τον κόμβο που αντιστοιχεί στο σημείο συνάντησης.
Στη συνέχεια, για το Robot 2 εκτελούμε την ίδια διαδικασία, με επιπλέον δεδομένο την διαδρομή του Robot 1. Επομένως, το Robot 2 πρέπει να ελέγχει για πιθανά σημεία σύγκρουσης με το Robot 1 και να αποφεύγει τις συγκρούσεις κατάληλλα.
Μόλις παράξουμε τις δύο διαδρομές φροντίζουμε το Robot που φτάνει πρώτο στο σημείο συνάντησης να περιμένει το άλλο, προσθέτωντάς του stalls στο meeting point, μέχρι το άλλο να βρεθεί σε διπλανή θέση.
Τέλος, πρώτα για το Robot 1 και ύστερα για το Robot 2, με δεδομένη την διαδρομή του πρώτου, εκτελούμε τελευταία φορά τον A* ώστε να γυρίσουν και τα δύο στις αρχικές τους θέσεις, φυσικά αποφεύγοντας τις συγκρούσεις. Έτσι, λαμβάνουμε τις τελικές διαδρομές.

2.ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

- Ο χώρος καταστάσεων, στον οποίο εφαρμόζουμε την αναζήτηση μοντελοποιήθηκε μέσω της κλάσης **Node**, η οποία περιέχει τα εξής πεδία:

  - Node parent : Ο γονέας του κόμβου στο δέντρο αναζήτησης
  - Node original parent : Ο αρχικός γονέας του κόμβου στο μονοπάτι (χρήσιμο για πίσω βήματα)

- int g : Το κόστος μετάβασης από τον αρχικό κόμβο
- int h : Εκτιμώμενη απόσταση από το στόχο
- int f : Το άθροισμα των g και h
- Node r_sibling : Ο επόμενος κόμβος στο ίδιο επίπεδο του δέντρου
- Node child1 : Το πρώτο παιδί του κόμβου
- int next : 0 όταν ο κόμβος δεν επεκτείνεται άλλο
  1 διαφορετικά
  -1 όταν δε γνωρίζουμε ακόμη
- State state : Η κατάσταση του προβλήματος στο δεδομένο κόμβο

Οι βασικές μέθοδοι της κλάσης είναι:

- List<Node> find_children : Ο κόμβος πρέπει να είναι σε θέση να βρει τα παιδιά του και να τα επιστρέψει σε μορφή λίστας
- int heuristic : Ο κόμβος επίσης πρέπει να μπορεί να εκτιμήσει την απόσταση του από το στόχο

- Η κλάση **State** χρησιμοποιείται από το Node για να περιγράψει την κατάσταση στην οποία βρισκόμαστε. Περιέχει τα παρακάτω πεδία:

  - int x : Η τετμημένη του σημείου του χάρτη στο οποίο βρισκόμαστε
  - int y : Η τεταγμένη του σημείου του χάρτη στο οποίο βρισκόμαστε
  - int step : Ένας μετρητής για τα βήματα του Robot από τη στιγμή της εκκίνησης (στο σημείο εκκίνησης έχουμε step=0)

- Για τις επιμέρους διαδρομές των Robots χρησιμοποιήσαμε συνδεδεμένες λίστες από κόμβους (LinkedList<Node>)

- Η εκτέλεση του αλγορίθμου A* απαιτεί σε κάθε βήμα τη σύκριση των πιθανών κόμβων προς μετάβαση, βάσει την τιμή της παραμέτρου f. Για το σκοπό αυτό χρησιμοποιήθηκε μία ουρά προτεραιότητας (PriorityQueue<Node> queue) από την οποία ο A* κάνει κάθε φορά poll το μικρότερο στοιχείο.

- Επιπλέον, πρέπει να αποθηκεύουμε κάπου τις συντεταγμένες τις οποίες έχουμε ήδη εξετάσει, έτσι ώστε να μην ξαναεπισκεφθούμε αυτούς τους κόμβους. Για την αποθήκευση, χρησιμοποιήσαμε ένα HashSet (HashSet<Node> explored) στο οποίο η εύρεση στοιχείου μπορεί να γίνει αποδοτικά.

3.ΒΑΣΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ-ΤΕΛΕΣΤΕΣ ΓΙΑ ΤΗΝ ΥΛΟΠΟΙΗΣΗ ΤΟΥ A*

Ο Αλγόριθμος A* σε κάθε βήμα πάιρνει από μία ουρά προτεραιότητας τον κόμβο που αναμένει ότι θα οδηγήσει στο ελάχιστο μονοπάτι προς τον στόχο και , εφόσον αυτός δεν αντιστοιχεί στον στόχο, ψάχνει τους κόμβους παιδιά του με την βοήθεια του τελεστή find_children. Οι κόμβοι-παιδιά αντιστοιχούν στις θέσεις στις οποίες μπορούμε να φτάσουμε από τον γονέα σε ένα βήμα. Στην συνέχεια, ο A* προσθέτει στην ουρά τους νέους κόμβους (που αντιστοιχούν σε μερικά μονοπάτια μεγαλύτερα κατά ένα βήμα από το μονοπάτι-γονέα, που περνούν από την θέση-γονέα και οδηγούν στις θέσεις-παιδιά). Εδώ χρησιμοποιείται, επιπλέον, dynamic programming έτσι ώστε αν έχουμε βρεί πολλά μονοπάτια προς κάποια θέση να διατηρούμε πάντα μόνο το ελάχιστο από αυτά.
Για την υλοποιήση του A* χρειάστηκε να σχεδιάσουμε κατάλληλα τον τελεστή μετάβασης find_children.
Η λειτουργία του τελεστή είναι η εξής:
Ο τελεστής δρα πάνω σε δεδομένο κόμβο-γονέα. Λαμβάνει ως παραμέτρους την κάτοψη του χώρου (char[][] map) και τις συντεταγμένες του στόχου της αναζήτησης. Επιστρέφει, λίστα (List<Node> res) που περιέχει τα παιδιά του κόμβου-γονέα, δηλαδή όλες τις επιτρεπτές επόμενες θέσεις.
Η παραγωγή των κόμβων-παιδιών γίνεται ώς εξής:
Για κάθε πιθανή μετάβαση (ένα βήμα βόρεια, ένα βήμα νότια, ένα βήμα ανατολικά, ένα βήμα δυτικά) ελέγχουμε αν οδηγεί σε θέση που καταλαμβάνεται από εμπόδιο του χώρου ή τέλος του χάρτη(*). Αν ναι αγνοούμε την μετάβαση. Αν όχι, δημιουργούμε νέο κόμβο. Οι κόμβοι-παιδιά έχουν τις κατάλληλες συντεταγμένες, κόστος g = κόστος γονέα + 1, h που υπολογίζεται με χρήση της ευριστικής, δείκτη προς τον γονέα και βήμα = βήμα γονέα + 1.

(*)Κατά το διάβασμα του input προσθέτουμε όρια στον χάρτη, γράφοντας X στις θέσεις του map που αντιστοιχούν στο τέλος του χάρτη.

4.ΕΥΡΙΣΤΙΚΕΣ ΜΕΘΟΔΟΙ

Χρησιμοποιούμε δύο ευριστικές μεθόδους, έναν υπό-εκτιμητή (admissible) και έναν υπέρ-εκτιμητή(non-admissible).
Η admissible heuristic που χρησιμοποιούμε αντιστοιχεί στην manhattan distance (|x_source – x_destination| + |y_source – y_destination|) . Είναι, φυσικά, υποεκτίμηση της πραγματικής απόστασης μεταξύ δύο θέσεων αφού αντιστοιχεί στην ελάχιστη απόσταση (σε βήματα) που

μπορεί να διανυθεί κάνοντας κάθε φορά μόνο ένα βήμα βόρεια,νότια,ανατολικά ή δυτικά αλλά όχι διαγώνια. Η ελάχιστη απόσταση δεν μπορεί να επιτευχθεί πάντα λόγω των εμποδίων. Η non-admissible heuristic που χρησιμοποιούμε αντιστοιχεί στο 2*manhattan distance. Πρόκειται, προφανώς, για υπερεκτίμηση της πραγματικής απόστασης μεταξύ δύο θέσεων αφού στις περιπτώσεις που δεν έχουμε εμπόδια ενδιάμεσα, μπορεί να επιτευχθεί η ελάχιστη απόσταση που είναι το ½ της υπερεκτίμησής μας.

5.ΑΛΓΟΡΙΘΜΟΣ ΕΠΙΛΥΣΗΣ ΣΥΓΚΡΟΥΣΕΩΝ

Όπως, προαναφέρθηκε μόνο το Robot2 έχει την ευθύνη ελέγχου για πιθανές συγκρούσεις. Κάθε φορά που εξετάζει ένα νέο κόμβο, καλεί την check_for_conflict(), δεδομένης της διαδρομής του Robot1 (route_of_other).

Υπάρχουν δύο πιθανά είδη συγκρούσεων:

- Conflict case 1 : Το Robot2 εξετάζει τον κόμβο current στο βήμα current_step, ενώ το Robot1 έχει ήδη επιλέξει να επισκεφθεί τον κόμβο αυτό στο ίδιο βήμα. Για την επίλυση αυτής της σύγκρουσης, δίνουμε τη δυνατότητα στο Robot2 να περιμένει (stall) στην προηγούμενη θέση του. Αυτό θέλουμε να συμβεί μόνο αν είναι εντελώς απαραίτητο, επομένως δίνουμε στον κόμβο που κάνει stall τιμή INT_MAX ως παράμετρο f. Κατά αυτόν τον τρόπο, η διαδρομή αυτή θα επιλεχθεί μόνο αν η ουρά περιέχει μόνο στοιχεία με INT_MAX.
- Conflict case 2 : Το Robot2 εξετάζει τον κόμβο current με πατέρα τον κόμβο parent στο βήμα current_step, ενώ το Robot1 έχει ήδη επιλέξει να επισκεφθεί τον κόμβο parent, έχοντας πατέρα τον κόμβο current στο ίδιο βήμα. Με άλλα λόγια, το Robot2 σκοπεύει να πάει στη θέση του Robot1 και ταυτόχρονα το Robot1 στη θέση του Robot2 την ίδια χρονική στιγμή. Για την επίλυση αυτής της σύγκρουσης, δίνουμε τη δυνατότητα στο Robot2 να κάνει ένα βήμα πίσω, αντί του βήματος που επέλεξε (και πάλι με μικρή προτεραιότητα θέτοντας INT_MAX. Επίσης, βγάζουμε από τα explored όλα τα παιδιά του κόμβου στον οποίο οδηγούμε το Robot2, ώστε να έχει ελευθερία κίνησης για την αποφυγή.

Ο ψευδοκώδικας της check for conflict φαίνεται παρακάτω:

```
boolean check_for_conflict(current, route_of_other, queue, explored)
{
      if(Conflict case 1)
      {
            //we might want to visit this position again later
            explored.remove(current);

            //current causes conflict
            //previous position = parent of current
            f_of_stall = INT_MAX;
            x_of_stall = x of parent;
            y_of_stall = y of parent;
            step_of_stall = step of parent + 1;

            queue.add(stall);
            return true;
      }
         else if(Conflict case 2)
      {
            //current causes conflict
            //step back = parent of parent of current
         if(parent of parent of current exists)
         {
               //we might want to visit these positions again later
               explored.remove(current);
               explored.remove(children of step back position);

               f_of_go_back = INT_MAX;
               go_back_x = x of parent of parent;
               go_back_y = y of parent of parent;
               step_of_go_back = step of parent + 1;

               queue.add(step_back);
         }
         //Special case: current node cant be replaced with a step back
         //Unavoidable collision
         //Just close the path
         else
         {
             //we might want to visit this position again later
             explored.remove(current);
             current.setNext(0);
             current.setChild1(null);
         }
         return true;
      }
      return false;
}
```

6.ΜΟΡΦΗ ΑΡΧΕΙΩΝ ΕΙΣΟΔΟΥ

Τα αρχεία εισόδου έχουν την εξής μορφή:

N M
xR1 yR1
xR2 yR2
xMP yMP
targets

όπου :
N = Αριθμός γραμμών , M = Αριθμός στηλών
xR1 = Τετμημένη αρχικής θέσης Robot 1, yR1 = Τεταγμένη αρχικής θέσης Robot 1
xR2 = Τετμημένη αρχικής θέσης Robot 1, yR2 = Τεταγμένη αρχικής θέσης Robot 1
xMP = Τετμημένη τελικού σημείου συνάντησης
yMP = Τεταγμένη τελικού σημείου συνάντησης
targets = Αριθμός ενδιάμεσων στόχων

Ακολουθουν targets γραμμές τις μορφής:
xt yt
όπου xt = Τετμημένη στόχου και yt = Τεταγμένη στόχου

Τέλος, δίνεται η κάτοψη του χώρου σε N γραμμές όπου το στοιχείο j (1<=j<=M) είναι X για εμπόδιο και O για ελεύθερη θέση.

7.Testcases

Εξετάσαμε 6 διαφορετικά αρχεία εισόδου με ποικίλα χαρακτηριστικά:

- Το input1 είναι αυτό που δίνεται στην εκφώνηση της άσκησης.
- Το input2 εξετάζει εκτενώς την περίπτωση του Conflict case 2. Αναγκάζουμε το Robot2 να εισέλθει σε ένα στενό διάδρομο, τον οποίο το Robot1 διασχίζει ανάποδα. Μόλις συναντηθούν, το Robot2 εκτελεί επαναλαμβανόμενα πίσω βήματα μέχρι να βγει από το διάδρομο και να μπορέσει να αποφύγει επιτυχώς το Robot1.
- Το input3 είναι παρόμοιο με το input2, μόνο που αυτή τη φορά δίνουμε στο Robot2 την εναλλακτική να αποφύγει τελείως την προηγούμενη είσοδο στο διάδρομο με μεγαλύτερο βέβαια κόστος. Παρατηρούμε ότι το Robot2 επιτυχώς αποφεύγει τη συνάντηση με το Robot1 στο διάδρομο και τα πίσω βήματα, ακολουθώντας μία διαδρομή που αρχικά έμοιαζε χειρότερη.

- Το input4 εξετάζει την περίπτωση του Conflict case 1. Τα Robots τοποθετούνται έτσι ώστε η μόνη επιτρεπτή αρχική κίνηση και για τα δύο να οδηγεί σε σύγκρουση. Βλέπουμε ότι το Robot2 επιτυχώς περιμένει στην αρχική του θέση μέχρι να περάσει το Robot1.
- Το input5 είναι λίγο μεγαλύτερου μεγέθους με αρκετά εμπόδια στη διαδρομή. Εδώ εμφανίζονται βέβαια υποψήφια Conflicts, αλλά δεν εφαρμόζονται stalls ή πίσω βήματα, καθώς υπάρχουν εναλλακτικές διαδρομές.
- Το input6 είναι αρκετά μεγαλύτερου μεγέθους με λίγα εμπόδια, έτσι ώστε να εξετάσουμε τη συμπεριφορά του αλγορίθμου όταν υπάρχει δυνατότητα να ανοίγονται πολλοί κόμβοι.

8.ΣΤΑΤΙΣΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Τα γραφήματα ακολουθούν στην επόμενη σελίδα. Οι άξονες έχουν λογαριθμηθεί για την καλύτερη προβολή των σημείων.

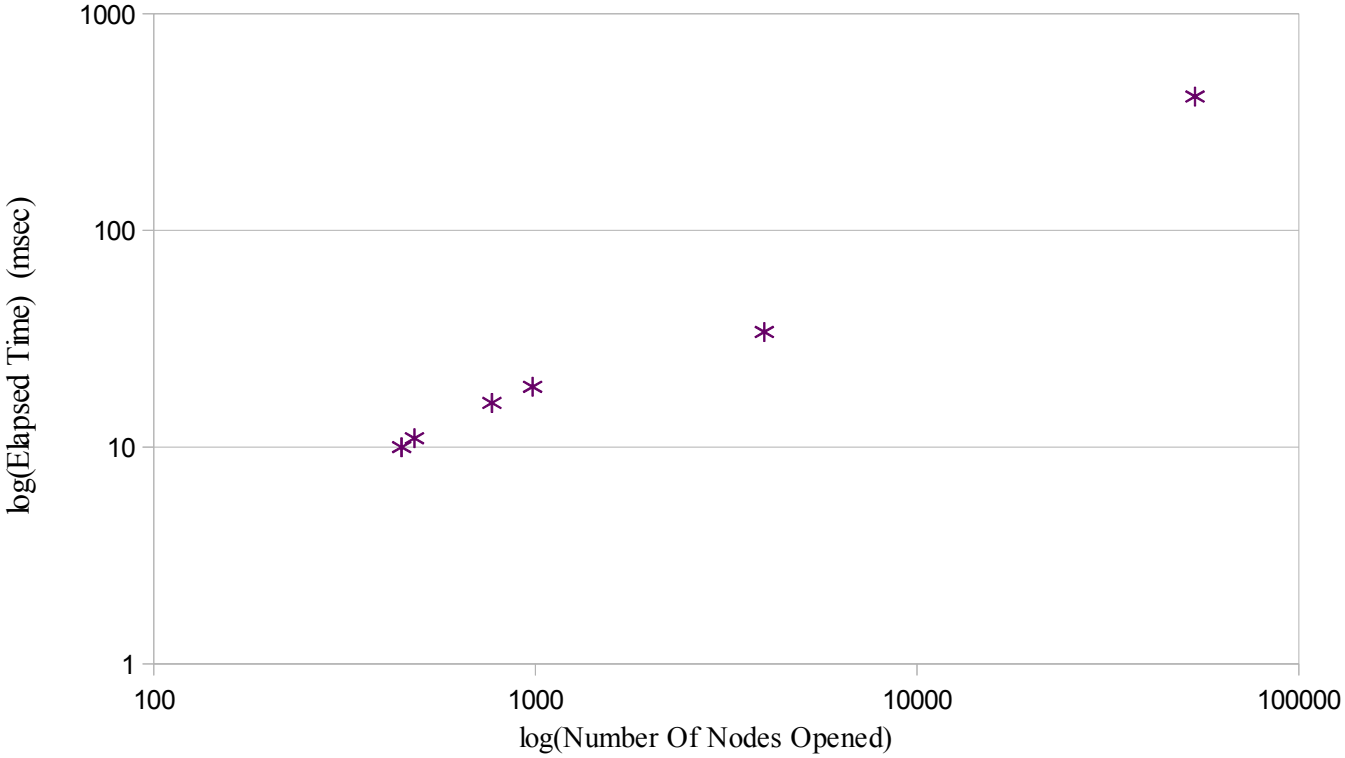
Παρατηρούμε την προφανη αναλογία που υπάρχει μεταξύ κόμβων που ανοίγονται και χρόνου εκτέλεσης.

Χρησιμοποιώντας non-addmissible heuristic παρατηρούμε ότι το τελικό μονοπάτι δεν είναι απραίτητα το βέλτιστο. Επίσης, ο αριθμός των κόμβων που ανοίγονται σε αυτήν την περίπτωση είναι μικρότερος.

9.ΒΗΜΑΤΑ ΕΚΤΕΛΕΣΗΣ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ

Τα αρχεία εισόδου, μαζί με τις αντίστοιχες εξόδους που παράγει το πρόγραμμά μας και τον πηγαίο κώδικα βρίσκονται επισυναπτόμενα σε ξεχωριστά αρχεία.

Γράφημα στην περίπτωση χρήσης admissible heuristic



Γράφημα στην περίπτωση χρήσης non-admissible heuristic