ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ
Τεχνητή Νοημοσύνη
1η Άσκηση
Ακ. έτος 2011-2012

Γερακάρης Βασίλης Α.Μ.: 03108092
Λύρας Γρηγόρης Α.Μ.: 03109687

10 Φεβρουαρίου 2012

Υλοποίηση αλγορίθμου A*

1.1 Παρουσίαση προβλήματος και σκιαγράφηση λύσης

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε τον αλγόριθμο αναζήτησης A* με σκοπό να οδηγηθούν με βέλτιστο τρόπο 2 ρομπότ σε ένα προκαθορισμένο σημείο συνάντησης, αποφεύγοντας τις πιθανές συγκρούσεις.

Έχοντας ως δεδομένο ότι τα δύο ρομπότ έχουν γνώση της θέσης και του πλάνου του άλλου ρομπότ, καθώς και πλήρη γνώση της αίθουσας, επιλέξαμε να κάνουμε μια παρατήρηση /παραδοχή που διευκολύνει σημαντικά την δομή του προγράμματός μας:

Αντί τα 2 ρομπότ να πραγματοποιούν τις κινήσεις/επιλογές τους εναλλάξ, θεωρούμε, χωρίς βλάβη της γενικότητας, ότι στις συγκρούσεις το 2ο ρομπότ θα έχει μια 'nice' συμπεριφορά, παραχωρόντας τη βέλτιστη κίνηση στο 1ο ρομπότ. Με τον τρόπο αυτό, μπορούμε να εκτελέσουμε σειριακά τον αλγόριθμο A*, πρώτα για το 1ο ρομπότ και στη συνέχεια για το 2ο, έχοντας ήδη δεδομένες τις κινήσεις του 1ου.

Ο χώρος των καταστάσεων μας δίνεται ως ένα grid με O (στις θέσεις που επιτρέπεται η κίνηση) και X (στις θέσεις όπου βρίσκονται εμπόδια). Η βασική ιδέα πίσω από τον αλγόριθμό μας είναι ότι οι θέσεις με X κωδικοποιούνται ως '-1', οι θέσεις με O με '0', ενώ η θέση στην οποία θα βρίσκεται το 1ο ρομπότ στο k-οστό βήμα με 'k'.

Με τον τρόπο αυτό, μπορούμε να περιορίσουμε τις επιλογές του 2ου ρομπότ σε κάθε βήμα, αναγκάζοντας το να ψάξει εναλλακτικό μονοπάτι ή να μείνει στάσιμο γι'αυτό το βήμα (αν το συμφέρει). Καταφέρνουμε έτσι να αποφύγουμε τις πιθανές συγκρούσεις, ενώ ταυτόχρονα δε θυσιάζουμε τη βελτιστότητα της λύσης μας.

Για την υλοποίηση του παραπάνω αλγορίθμου, επιλέχθηκε ως κατάλληλη γλώσσα η Python, επειδή μας δίνει τη δυνατότητα να επικεντρωθούμε στα πλέον σημαντικά κομμάτια του προβλήματος (αλγόριθμος και στρατηγική αναζήτησης), να γράψουμε πυκνό και ευανάγνωστο κώδικα, ενώ με τη χρήση ενός extension module όπως το Psyco έχουμε αντίστοιχο χρόνο εκτέλεσης όπως αν γράφαμε σε μια γλώσσα χαμηλότερου επιπέδου.

1.2 Επιλογή Δομών Δεδομένων

Στην υλοποίησή μας επιλέξαμε να χρησιμοποιήσουμε λίστες λιστών της Python, όπου ο κάθε κόμβος στι λίστα του A* είχε 4 στοιχεία:

1. Το άθροισμα ευριστικής και κόστους (h + c)
2. Το κόστος (c)
3. Την τετμημένη του σημείου (x)
4. Την τεταγμένη του σημείου (y)

Επιπλέον, χρησιμοποιείται ένα λεξικό (dict) προκειμένου να πραγματοποιηθεί η αναδόμηση της επιλεγμένης διαδρομής ενός ρομπότ. Τα dictionaries είναι associative arrays, ένας τύπος αντικειμένων που μοιάζει με λίστα, αλλά όπου σε μοναδικά κλειδιά αντιστοιχίζονται (όχι απαραίτητα μοναδικές) τιμές.

1.3 Βασικές συναρτήσεις-τελεστές

2

1.4 Ευριστικές μέθοδοι

- Δεδομένου ότι δουλεύουμε σε δισδιάστατο χώρο με μόνο οριζόντια και κάθετη κίνηση (όχι διαγώνια), επιλέχθηκε ως πιο έγκυρος και αποδοτικός υποεκτιμητής (admissible heuristic) η απόσταση Manhattan:

$$ManhDist = |x - x_T| + |y - y_T|$$

  Η χρήση υποεκτιμητή μας εγγυάται τη βελτιστότητα της λύσης.

- Ένας λειτουργικός υπερεκτιμητής (non-admissible heuristic), είναι το άθροισμα των τετραγώνων των αποστάσεων από τον προορισμό:

$$SqDist = (x - x_T)^2 + (y - y_T)^2$$

  Αν χρησιμοποιήσουμε ένα υπερεκτιμητή, θα επεκτείνουμε πιθανώς πολύ λιγότερους κόμβους κατά την αναζήτηση μας. Το χρονικό κέρδος αυτό όμως αντισταθμίζεται με την πιθανότητα να μην είναι βέλτιστη η λύση που προκύπτει, αφού ορισμένοι βέλτιστοι κόμβοι αποφεύγονται λόγω του αυξημένου συνολικού κόστους που εισάγεται με τον υπερεκτιμητή.

1.5 Έλεγχος και επίλυση συγκρούσεων

Όπως εξηγήσαμε πριν, ο αλγόριθμός μας εκτελείται πρώτα για το ένα ρομπότ, και στη συνέχεια για το δεύτερο, σε ένα τροποποιημένο grid. Στην περίπτωση που στο μέτωπο αναζήτησης του 2ου για το k-οστό βήμα είναι η θέση που πήγε το 1ο ρομπότ στο k βήμα, ανιχνεύεται πιθανή σύγκρουση. Το 2ο ρομπότ έχει πλέον 2 επιλογές:

1. Να περιμένει (wait) για ένα γύρο, επιτρέποντας στο 1ο να προχωρήσει και έπειτα ακολουθώντας το (μιας και οι επιλογές του 1ου είναι εξ'ορισμού βέλτιστες), δίνοντας στο σημείο της σύγκρουσης κόστος $c + 1$

2. Να αναζητήσει κάποια εναλλακτική πορεία προς το σημείο συνάντησης.

Από τις 2 αυτές επιλογές, τελικά θα διαλέξει αυτή που έχει το λιγότερο κόστος (σε αριθμό βημάτων)

Εκτελώντας τον αλγόριθμο A* 2 φορές, ώστε στη μία το ρομπότ A να παραχωρεί προτεραιότητα στις συγκρούσεις, ενώ στη 2η το B να παραχωρεί προτεραιότητα και επιλέγοντας αυτό που παράγει μικρότερο αριθμό βημάτων τερματισμού, καταλήγουμε στο τελικά βέλτιστο αποτέλεσμα της αναζήτησης.

Algorithm 1 Conflict resolution

procedure resolve($newGrid[x][y], currStep$)
  if $newGrid[x][y] == currStep$ **then**
    $cost_1 \leftarrow (len(Astar((x, y), target)) + 1)$
    $path_1 \leftarrow (path(Astar((x, y), target)))$
    $newgrid[x][y] \leftarrow Invalid$
    $cost_2 \leftarrow len(Astar(currPos, target))$
    $path_1 \leftarrow (path(Astar((currPos, target)))$
    if $cost_1 < cost_2$ **then**
      $cost \leftarrow cost_1$
      $path \leftarrow path_1$
    else
      $cost \leftarrow cost_2$
      $path \leftarrow path_2$

1.6 Χρόνος εκτέλεσης ανά μέγεθος εισόδου

Nodes	Times

Nodes	Times

1.7 Εκτύπωση βημάτων-συγκρούσεων-εναλλακτικών δρόμων σε παράδειγμα εκτέλεσης

6

1.8 Πηγαίος κώδικας

Here be dragons!

- Οι συναρτήσεις που καλούνται για την ανάγνωση της εισόδου:

```
#!/usr/bin/python
#/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
#
#* File Name : inparser.py
#
#* Purpose : Input parsing of state space
#
#* Creation Date : 24-12-2011
#
#* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
#
#* Created By : Greg Liras <gregliras@gmail.com>
#
#* Created By : Vasilis Gerakaris <vgerak@gmail.com>
#
#_.-._._._._._._._._._._._._._._._._._._._._._._._._._._._._.*/

def retbools(st):
    a=[]
```

```
    for i in st:
        #'X' marks obstacle, 'O' marks open space
        if i == 'X':
            a.append(-1)
        elif i=='O':
            a.append(0)
    return a

def parseInput(f):
    lines = int(f.readline().split()[0])
    robo1_initstate = tuple(map(int,f.readline().split()))
    robo2_initstate = tuple(map(int,f.readline().split()))
    target = tuple(map(int,f.readline().split()))
    text = map(retbools,f.readlines())
    return (target,robo1_initstate,robo2_initstate,text)
```

- Οι ευριστικές που χρησιμοποιούνται στον αλγόριθμο:

```
#!/usr/bin/python
#/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
#
#* File Name : heuristics.py
#
#* Purpose :
#
#* Creation Date : 09-02-2012
#
#* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
#
#* Created By : Greg Liras <gregliras@gmail.com>
#
#* Created By : Vasilis Gerakaris <vgerak@gmail.com>
#
#_.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.*/

def manhattanDist(point1,point2):
    return abs(point1[0][0]-point2[0])+abs(point1[0][1]-point2[1])

def squaredDist(point1,point2):
    return (point1[0][0]-point2[0])**2 + (point1[0][1]-point2[1])**2
```

- Το κυρίως σώμα του A* :

```
#!/usr/bin/python
#/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
#
#* File Name : astar.py
#
#* Purpose : Main body of A* algorithm
#
#* Creation Date : 24-12-2011
#
#* Last Modified : Fri 10 Feb 2012 17:50:59 EET
#
#* Created By : Greg Liras <gregliras@gmail.com>
#
#* Created By : Vasilis Gerakaris <vgerak@gmail.com>
#
#_.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.*/


def nextNodes((a,b)):
    return [((a-1,b),(a,b)),((a,b-1),(a,b)),((a+1,b),(a,b)),((a,b+1),(a,b))]

def putinlist(starque,(h,c,xy)):
    if not starque:
        starque.append((h,c,xy))
        return starque

    for i in range(len(starque)):
        (sh,sc,sxy) = starque[i]
        if sxy == xy:
            starque.pop(i)
            (sh,sc,sxy) = min((sh,sc,sxy),(h,c,xy))
            starque.append((sh,sc,sxy))
            return starque
```

```
34
35  def astar(startpoint,finishpoint,grid,heuristic,robotID=1):
36      ancestors={}
37      #ancestors is a dictionary which stores the ancestors of each point
38      #this will be used in the end to rebuild the path
39      passedlist=[]
40      passedlist.append(startpoint)
41      #passedlist contains nodes that have been processed already
42      starque=[]
43      #num of rows
44      sizex=len(grid)
45      #num of columns
46      sizey=len(grid[0])
47      possible = map(lambda x:(heuristic(x,finishpoint)+1,1,x),nextNodes(startpoint))
48      #each point has these characteristics
49      #(heuristic,cost,((x,y),father))
50      for (h,c,((x,y),father)) in possible:
51          #checking for bounds and then checking if grid[x][y]==True
52          #now should be grid[x][y]==0 ??
53          if x >= 0 and y >= 0 and x < sizey and y < sizex and grid[x][y] != -1:
54              passedlist.append((x,y))
55              ancestors[(x,y)]=father
56              starque.append((h,c,(x,y)))
57
58      ind = starque.index(min(starque))
59      (h,c,(x,y)) = starque.pop(ind)
60      #Conflict detection on first step
61      while(grid[x][y] == c + 1 ):
62          print "Conflict in [%d,%d] on step %d , Robot #%d recalculating.." %(x,y,c,robotID)
63          starque.append((h+1,c+1,(x,y)))
64          ind = starque.index(min(starque))
65          #Consideration of alternative path
66          print "Robot #%d trying.." %(robotID),starque[ind][2][0],starque[ind][2][1]
67          (h,c,(x,y)) = starque.pop(ind)
68
69      #ind = starque.index(min(starque))
70      #find index of touple with the lowest heuristic+cost
71      #nxt = starque.pop(ind)
72      nxt = (h,c,(x,y))
73      #remove if from the queue
74      #and store it in nxt
75      #nxt = (minh,c,((x,y),father))
76      current = nxt[2]
77      currentCost = nxt[1]
78      #current cost is the cost so far that is stored in nxt
79      #goal = zip( *nextNodes( finishpoint ) )[0]
80      #print goal
81
82      while(current!=finishpoint):
83          #until you find the end
84          possible = map(lambda x:(heuristic(x,finishpoint)+currentCost+1,currentCost+1,x),nextNodes(current))
85          #find the next possible list
86          for (h,c,((x,y),father)) in possible:
87              if x >= 0 and y >= 0 and x < sizex and y < sizey and grid[x][y] != -1:
88                  #check what is in possible list, make sure its sane
89                  #starque = putinlist(starque,(h,c,(x,y)))
90                  if(x,y) not in passedlist:
91                      passedlist.append((x,y))
92                      #if I havend passed this so far
93                      #then store it and insert the coordinates in ancestors dictionary
94                      ancestors[(x,y)]=father
95                      starque.append((h,c,(x,y)))
96                  #if I have passed this already then I can reach this with a lower cost
97                  #so i don't need to save x,y
98
99          ind = starque.index(min(starque))
100         (h,c,(x,y)) = starque.pop(ind)
101         #Conflict detection for all steps
102         while(grid[x][y] == c + 1 ):
103             print "Conflict in [%d,%d] on step %d , Robot #%d recalculating.." %(x,y,c,robotID)
104             starque.append((h+1,c+1,(x,y)))
105             ind = starque.index(min(starque))
106             #Consideration of alternative paths
107             print "Robot #%d trying.." %(robotID),starque[ind][2][0],starque[ind][2][1]
108             (h,c,(x,y)) = starque.pop(ind)
```

```
        #find index of touple with the lowest heuristic+cost
        nxt = (h,c,(x,y))
        #remove if from the queue and store it in nxt
        #nxt = (minh,c,((x,y),father))
        current = nxt[2]
        currentCost = nxt[1]
        #current cost is the cost so far that is stored in nxt
    print "Found it after %d expanded nodes" %len(passedlist)

    finalists=[]
    i =finishpoint
    finalists.append(i)
    #starting from the end build the path list
    #following the directions in the ancestors dictionary
    while i !=startpoint:
        i = ancestors[i]
        finalists.append(i)
    #reverse the path so it starts from the beginning
    finalists.reverse()
    #put it in the queue
    return (finalists, len(passedlist))
```

- Οι συναρτήσεις που σχετίζονται με την εμφάνιση και τροποποίηση του χώρου καταστάσεων:

```
#!/usr/bin/python
#* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
#
#* File Name : grids.py
#
#* Purpose : Functions related with grid modifying & output
#
#* Creation Date : 24-12-2011
#
#* Last Modified : Fri 10 Feb 2012 04:17:27 PM EET
#
#* Created By : Greg Liras <gregliras@gmail.com>
#
#* Created By : Vasilis Gerakaris <vgerak@gmail.com>
#
#_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._.*/

lgrid=[]
blockChar = unichr(0x258A)
joinedColor = "0;32"
names = "Vasilis Gerakaris - Gregory Liras"

#Choose what will be printed depending on element in position
def revertMap(b):
    if b=="@" or b=="#":
        return b
    elif b>=0:
        return " "                  #unused space gets blankspace
    elif b<0:
        return "\033[41m \033[0m"   #obstacle gets red solid box
    else:
        return b

def flushgrid(grid):
    global lgrid
    lgrid = []
    for i in grid:
        lgrid.append(map(revertMap,list(i)))

def designpath(color,(sx,sy),(fx,fy),finalists):
    global lgrid
    for (x,y) in finalists:
        if ( lgrid[x][y].startswith("\033")):       #if both robots use position, color with green
            lgrid[x][y] = "\033["+joinedColor+"m*\033[0m"
        else:                                       #else keep designated robot color
            lgrid[x][y]="\033["+color+"m*\033[0m"
    lgrid[sx][sy]="S"
    lgrid[fx][fy]="F"

def printpath():
```

```
    print "\033[47m"+(" "*(len(lgrid[0])+2))+"\033[0m"
    for i in lgrid:
        print "\033[47m \033[0m"+"".join(i)+"\033[47m \033[0m"
    print "\033[47m\033[1;34m"+(" "*(len(lgrid[0])+2-len(names)))+names+"\033[0m"

def printgrid((cx,cy),(fx,fy),grid):
    global lgrid
    if not lgrid:
        for i in grid:
            lgrid.append(map(revertMap,i))
    lgrid[cx][cy]="@"
    lgrid[fx][fy]="#"

    for i in lgrid:
        print "".join(i)
    print "----"

def modifygrid(finalists,grid):
    i=1
    for (x,y) in finalists:
        grid[x][y]=i        #robot 1 marks its steps on the grid to
        i+=1                #be unusable (on same turn) by robot 2
    return grid
```

- Ο controller που καλεί τις παραπάνω συναρτήσεις για να παραχθεί το τελικό αποτέλεσμα:

```
#!/usr/bin/python
#* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
#
#* File Name : controler.py
#
#* Purpose : 1st assignment in Artificial Intelligence
#
#* Creation Date : 24-12-2011
#
#* Last Modified : Fri 10 Feb 2012 04:44:06 PM EET
#
#* Created By : Greg Liras <gregliras@gmail.com>
#
#* Created By : Vasilis Gerakaris <vgerak@gmail.com>
#
#_.-._._._._._._._._._._._._._._._._._._._._._._._._._._._._.-.*/

import sys

from inparser import parseInput
from astar import astar
import heuristics
from grids import flushgrid,printpath,designpath,modifygrid

def main():
    if len(sys.argv) < 3:
        print "Usage: %s <inputfile> <mode (A/N)>" %sys.argv[0]
        return -1
    f=open(sys.argv[1],"r")
    (target,r1,r2,initialfield) = parseInput(f)
    f.close()
    field = map(list,initialfield)
    modeCheck = sys.argv[2]
    while modeCheck != "A" and modeCheck != "N" and modeCheck != "a" and modeCheck != "n":
        print "Wrong mode, choose A for admissible heuristic or N for non-admissible"
        modeCheck = raw_input('Enter A or N --->')
    if modeCheck == "A" or modeCheck == 'a':
        print "Using Manhattan Distance as admissible heuristic"
        heuristic = heuristics.manhattanDist
    else:
        print "Using Square Distances as non-admissible heuristic"
        heuristic = heuristics.squaredDist
    print "\nLEGEND:"
    print "\033[1;34m Robot1 path \n\033[1;33m Robot2 path \n\033[0;32m Joined path \033[0m"
    print "\n \033[0;34m ======= Robot 2 plays 'nice' ======= \033[0m"
    total=0
    (finalists1,nodes) = astar(r1,target,field,heuristic,1)
    total += nodes
    field = modifygrid(finalists1,field)
    (finalists2,nodes) = astar(r2,target,field,heuristic,2)
```

```
    total += nodes
    flushgrid(field)
    designpath("1;34",r1,target,finalists1)
    designpath("1;33",r2,target,finalists2)
    printpath()
    max1 = max(len(finalists1),len(finalists2))
    print "Max length in steps:", max1-1
    print "\t Robot 1 took:\t\t", len(finalists1)-1, " steps"
    print "\t Robot 2 (nice) took:\t", len(finalists2)-1, " steps"
    flushgrid(field)
    print "\n \033[0;34m ======= Robot 1 plays 'nice' ======= \033[0m"
    field = map(list,initialfield)
    modeCheck = sys.argv[2]
    (finalists2,nodes) = astar(r2,target,field,heuristic,2)
    total += nodes
    field = modifygrid(finalists2,field)
    (finalists1,nodes) = astar(r1,target,field,heuristic,1)
    total += nodes
    flushgrid(field)
    designpath("1;34",r1,target,finalists1)
    designpath("1;33",r2,target,finalists2)
    printpath()
    max2 = max(len(finalists1),len(finalists2))
    print "Max length in steps:", max2-1
    print "\t Robot 1 (nice) took:\t", len(finalists1)-1, " steps"
    print "\t Robot 2 took:\t\t", len(finalists2)-1, " steps"
    print "\n ===== RESULT ====="
    if max1 < max2:
        print "1st strategy (Robot 2 plays nice) is optimal"
    elif max2 < max1:
        print "2nd strategy (Robot 1 plays nice) is optimal"
    else:
        print "Both strategies yield same result"
    print "Total nodes considered:", total


if __name__=="__main__":
    main()
```