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 Just-in-time compiler όπως ο Psycho έχουμε ίδιο 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

5

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 14:33:24 EET
#
#* Created By : Greg Liras <gregliras@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 : Thu  9 Feb 2012 22:09:12 EET
#
#* Created By : Greg Liras <gregliras@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 14:27:19 EET
#
#* Created By : Greg Liras <gregliras@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

def astar(startpoint,finishpoint,grid,heuristic):
    ancestors={}
    #ancestors is a dictionary which stores the ancestors of each point
    #this will be used in the end to rebuild the path
    passedlist=[]
    passedlist.append(startpoint)
    #passedlist contains nodes that have been processed already
    starque=[]
    #num of rows
    sizex=len(grid)
    #num of columns
    sizey=len(grid[0])
    possible = map(lambda x:(heuristic(x,finishpoint)+1,1,x),nextNodes(startpoint))
    #each point has these characteristics
    #(heuristic,cost,((x,y),father))
    for (h,c,((x,y),father)) in possible:
        #checking for bounds and then checking if grid[x][y]==True
        #now should be grid[x][y]==0 ??
        if x >= 0 and y >= 0 and x < sizey and y < sizex and grid[x][y] != -1:
            passedlist.append((x,y))
            ancestors[(x,y)]=father
            starque.append((h,c,(x,y)))

    ind = starque.index(min(starque))
    (h,c,(x,y)) = starque.pop(ind)
    #Conflict detection on first step
    while(grid[x][y] == c + 1 ):
        print "Conflict in [",x,",",y,"] on step", c, ", Robot #2 recalculating.."
        starque.append((h+1,c+1,(x,y)))
```

```
62          ind = starque.index(min(starque))
63          #Consideration of alternative path
64          print "Robot #2 trying..",starque[ind][2][0],starque[ind][2][1]
65          (h,c,(x,y)) = starque.pop(ind)
66
67      #ind = starque.index(min(starque))
68      #find index of touple with the lowest heuristic+cost
69      #nxt = starque.pop(ind)
70      nxt = (h,c,(x,y))
71      #remove if from the queue
72      #and store it in nxt
73      #nxt = (minh,c,((x,y),father))
74      current = nxt[2]
75      currentCost = nxt[1]
76      #current cost is the cost so far that is stored in nxt
77      #goal = zip( *nextNodes( finishpoint ) )[0]
78      #print goal
79
80      while(current!=finishpoint):
81          #until you find the end
82          possible = map(lambda x:(heuristic(x,finishpoint)+currentCost+1,currentCost
83          #find the next possible list
84          for (h,c,((x,y),father)) in possible:
85              if x >= 0 and y >= 0 and x < sizex and y < sizey and grid[x][y] != -1:
86                  #check what is in possible list, make sure its sane
87                  #starque = putinlist(starque,(h,c,(x,y)))
88                  if(x,y) not in passedlist:
89                      passedlist.append((x,y))
90                      #if I havend passed this so far
91                      #then store it and insert the coordinates in ancestors dictio
92                      ancestors[(x,y)]=father
93                      starque.append((h,c,(x,y)))
94                  #if I have passed this already then I can reach this with a lower
95                  #so i don't need to save x,y
96
97          ind = starque.index(min(starque))
98          (h,c,(x,y)) = starque.pop(ind)
99          #Conflict detection for all steps
100         while(grid[x][y] == c + 1 ):
101             print "Conflict in [",x,",",y,"] on step", c, ", Robot #2 recalculating
102             starque.append((h+1,c+1,(x,y)))
103             ind = starque.index(min(starque))
104             #Consideration of alternative paths
105             print "Robot #2 trying..",starque[ind][2][0],starque[ind][2][1]
106             (h,c,(x,y)) = starque.pop(ind)
107
108
109         #find index of touple with the lowest heuristic+cost
110         nxt = (h,c,(x,y))
```

```
111             #remove if from the queue and store it in nxt
112             #nxt = (minh,c,((x,y),father))
113             current = nxt[2]
114             currentCost = nxt[1]
115             #current cost is the cost so far that is stored in nxt
116         print "Found it after %d expanded nodes" %len(passedlist)
117 
118         finalists=[]
119         i =finishpoint
120         finalists.append(i)
121         #starting from the end build the path list
122         #following the directions in the ancestors dictionary
123         while i !=startpoint:
124             i = ancestors[i]
125             finalists.append(i)
126         #reverse the path so it starts from the beginning
127         finalists.reverse()
128         #put it in the queue
129         return (finalists, len(passedlist))
```

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

```
1  #!/usr/bin/python
2  #/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
3  #
4  #* File Name : grids.py
5  #
6  #* Purpose : Functions related with grid modifying & output
7  #
8  #* Creation Date : 24-12-2011
9  #
10 #* Last Modified : Fri 10 Feb 2012 10:04:46 EET
11 #
12 #* Created By : Greg Liras <gregliras@gmail.com>
13 #
14 #_._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._._.*/
15 
16 lgrid=[]
17 blockChar = unichr(0x258A)
18 joinedColor = "0;32"
19 names = "Vasilis Gerakaris - Gregory Liras"
20 
21 #Choose what will be printed depending on element in position
22 def revertMap(b):
23     if b=="@" or b=="#":
24         return b
25     elif b>=0:
26         return " "                  #unused space gets blankspace
27     elif b<0:
```

```
28          return "\033[41m \033[0m"    #obstacle gets red solid box
29      else:
30          return b
31
32  def flushgrid(grid):
33      global lgrid
34      lgrid = []
35      for i in grid:
36          lgrid.append(map(revertMap,list(i)))
37
38  def designpath(color,(sx,sy),(fx,fy),finalists):
39      global lgrid
40      for (x,y) in finalists:
41          if ( lgrid[x][y].startswith("\033")):        #if both robots use position,
42              lgrid[x][y] = "\033["+joinedColor+"m*\033[0m"
43          else:                                        #else keep designated robot co
44              lgrid[x][y]="\033["+color+"m*\033[0m"
45      lgrid[sx][sy]="S"
46      lgrid[fx][fy]="F"
47
48  def printpath():
49      print "\033[47m"+(" "*(len(lgrid[0])+2))+"\033[0m"
50      for i in lgrid:
51          print "\033[47m \033[0m"+"".join(i)+"\033[47m \033[0m"
52      print "\033[47m\033[1;34m"+(" "*(len(lgrid[0])+2-len(names)))+names+"\033[0m"
53
54  def printgrid((cx,cy),(fx,fy),grid):
55      global lgrid
56      if not lgrid:
57          for i in grid:
58              lgrid.append(map(revertMap,i))
59      lgrid[cx][cy]="@"
60      lgrid[fx][fy]="#"
61
62      for i in lgrid:
63          print "".join(i)
64      print "----"
65
66  def modifygrid(finalists,grid):
67      i=1
68      for (x,y) in finalists:
69          grid[x][y]=i          #robot 1 marks its steps on the grid to
70          i+=1                  #be unusable (on same turn) by robot 2
71      return grid
```

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

```
1  #!/usr/bin/python
2  #/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
```

```
#
#* File Name : controler.py
#
#* Purpose : 1st assignment in Artificial Intelligence
#
#* Creation Date : 24-12-2011
#
#* Last Modified : Fri 10 Feb 2012 14:30:12 EET
#
#* Created By : Greg Liras <gregliras@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,field) = parseInput(f)
    f.close()
    modeCheck = sys.argv[2]
    while modeCheck != "A" and modeCheck != "N" and modeCheck != "a" and modeCheck
        print "Wrong mode, choose A for admissible heuristic or N for non-admissibl
        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
    print "\n \033[0;34m ======= Robot 2 plays 'nice' ======= \033[0m"
    total=0
    (finalists1,nodes) = astar(r1,target,field,heuristic)
    total += nodes
    field = modifygrid(finalists1,field)
    (finalists2,nodes) = astar(r2,target,field,heuristic)
    total += nodes
    flushgrid(field)
    designpath("1;34",r1,target,finalists1)
    designpath("1;33",r2,target,finalists2)
```

```
52      printpath()
53      max1 = max(len(finalists1),len(finalists2))
54      print "Max length in steps:", max1-1
55      print "\t Robot 1 took:\t\t", len(finalists1)-1, " steps"
56      print "\t Robot 2 (nice) took:\t", len(finalists2)-1, " steps"
57      flushgrid(field)
58      print "\n \033[0;34m ======= Robot 1 plays 'nice' ======= \033[0m"
59      (finalists2,nodes) = astar(r2,target,field,heuristic)
60      total += nodes
61      field = modifygrid(finalists2,field)
62      (finalists1,nodes) = astar(r1,target,field,heuristic)
63      total += nodes
64      flushgrid(field)
65      designpath("1;34",r1,target,finalists1)
66      designpath("1;33",r2,target,finalists2)
67      printpath()
68      max2 = max(len(finalists1),len(finalists2))
69      print "Max length in steps:", max2-1
70      print "\t Robot 1 (nice) took:\t", len(finalists1)-1, " steps"
71      print "\t Robot 2 took:\t\t", len(finalists2)-1, " steps"
72      print "\n ===== RESULT ====="
73      if max1 < max2:
74          print "1st strategy (Robot 2 plays nice) is optimal"
75      elif max2 < max1:
76          print "2nd strategy (Robot 1 plays nice) is optimal"
77      else:
78          print "Both strategies yield same result"
79      print "Total nodes considered:", total
80
81
82  if __name__=="__main__":
83      main()
```