

Øving 3 TDT4136

Sondre Foslien, NTNU

October 6, 2017

Task 1:

```
1 from PIL import Image, ImageDraw
2 from heapq import heappush, heappop
3
4
5 class Tile:      #tile class for every tile generated
6     xPos = 0
7     yPos = 0
8     g = 0
9     h = 0
10    f = 0
11    parent = None
12
13    def __init__(self, x, y, parent):      #initialize parent and coordinates
14        self.xPos = x
15        self.yPos = y
16        self.parent = parent
17
18    def __cmp__(self, rhs):      #overload cmp operator, compares only f value
19        if rhs == None: return False
20        if self.f > rhs.f: return 1
21        if self.f == rhs.f: return 0
22        if self.f < rhs.f: return -1
23
24    def __eq__(self, rhs):      ##overload == operator to compare x and y coordinates
25        if rhs == None: return False
26        if self.xPos == rhs.xPos and self.yPos == rhs.yPos: return True
27        return False
28
29    def manhattanDist(self, end):      #calculates manhattan dist from self to given tile
30        #calculates |x_2 - x_1| + |y_2 - y_1|
31        return abs(self.xPos - end.xPos) + abs(self.yPos - end.yPos)
32
33
34
35 class Board:
36     x, y = 0,0      #initialize variables to save board size
37     board = []      #initialize board for saving input file
38
39    def __init__(self, filename):
40
41        file = open(filename, 'r')      #open boardfile
42        for line in file:      #iterate through file
43            l = list(line[:-1])      #make string into list & removing the last \n
44            if self.x == 0: self.x = len(l)      #if x is not set it is set to the length of input list
45            if 'A' in l:      #if there is an element in input list with value A save as startTile
46                self.startTile = Tile(l.index('A'),(len(self.board)), None)
47            if 'B' in l:      #if there is element in input list with value B save as endTile
48                self.endTile = Tile(l.index('B'),(len(self.board)), None)
49            self.board.append(l)      #append the input list to board when finished investigating it
50            self.y+=1      #increment y for every input list
51
52    def aStar(self):
53        c, o = [], []      #initialize closed and open lists
54
55        #give startTile right h and f value, and append to open list
56        self.startTile.h = self.startTile.manhattanDist(self.endTile)
57        self.startTile.f = self.startTile.h + self.startTile.g
58        o.append(self.startTile)
59
60        while o:      #while there is elements in open-list
61            x = heappop(o)      #pop the best element(smallest f) from
62            if x in c: continue      #if element is already closed continue
63            heappush(c, x)      #push element into closed list
64
65        #if we are at endTile we are finished, return good values
```

```

66         if x == self.endTile: return x, True
67         succ = self.genSucc(x)           #find successors to x
68         for s in succ:                   #for all successors
69             if s in c: continue          #if already closed continue
70             g = x.g + 1                  #increment g value from parent
71
72         #if s not opened or has better value than already existing opened s
73         if s not in o or g < o[o.index(s)].g:
74             s.g = g                      #update g and calculate new h
75             s.h = s.manhattanDist(self.endTile)
76             s.f = s.g + s.h              #update h
77             if s not in o:               #if not opened push into heap at right place
78                 heappush(o, s)
79         return None, False               #if we didn't reach endTile return false
80
81
82     def genSucc(self, current):
83         succ = []                       #initialize successor list
84         x, y = current.xPos, current.yPos #get current coordinates
85
86         #array for iterate over successors
87         array = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
88         for x_, y_ in array:            #iterate through all sucessor coordinates
89
90             #if inside board and not a wall append to successor list
91             if 0 <= x_ < self.x and 0 <= y_ < self.y:
92                 if self.board[y_][x_] == '#': continue
93                 succ.append(Tile(x_, y_, current))
94         return succ
95
96     def reconstructPath(self, current):
97         #reconstruct path by starting at a current tile
98         path = [(current.xPos, current.yPos)]
99
100         while current.parent: #and working your way backwards through parents until there is none
101             path.append((current.parent.xPos, current.parent.yPos))
102             current = current.parent
103         return path
104
105     def getColor(self, symbol):
106         #returns color value based on which symbol
107         if symbol == "A":
108             return (255, 0, 0)
109         elif symbol == "B":
110             return (0, 255, 0)
111         elif symbol == ".":
112             return (255, 255, 255)
113         elif symbol == "#":
114             return (0, 0, 0)
115
116     def printBoardGraphics(self, path):
117         img = Image.new("RGB", (70*self.x+1, 70*self.y+1), "white") #draw background image with 70*x by 70*y
118         resolution
119         idraw = ImageDraw.Draw(img) #make possible to draw
120         for x in range(0, len(self.board)): #iterate through all elements in board
121             for y in range(0, len(self.board[0])):
122                 c = self.getColor(self.board[x][y]) #get color of current element
123                 #draw 70x70 rectangle with right color and a black outline
124                 idraw.rectangle([(y*70, x*70), (y*70+70, x*70+70)], fill=c, outline=(0,0,0))
125
126                 #if coordinate is in path draw smaller square on top to show path
127                 if (y, x) in path:
128                     c = (107, 97, 255)
129                     idraw.rectangle([(y*70+20, x*70+20), (y*70+50, x*70+50)], fill=c, outline=(0,0,0))
130         img.save("pictures/board-1-3.png") #save board
131
132     def main():
133         #make board, run aStar and save graphics
134         a = Board("boards/board-1-3.txt")
135         current, success = a.aStar()
136         if success:
137             print "success"
138             path = a.reconstructPath(current)
139             print path
140             a.printBoardGraphics(path)
141
142 if __name__ == "__main__":
143     main()

```


Task 2:

```
1 from PIL import Image, ImageDraw
2 from heapq import heappush, heappop
3
4
5 class Tile:      #tile class for every tile generated
6     xPos = 0
7     yPos = 0
8     g = 0
9     h = 0
10    f = 0
11    cost = 0
12    parent = None
13
14    def __init__(self, x, y, parent, cost):    #initialize parent and coordinates
15        self.xPos = x
16        self.yPos = y
17        self.parent = parent
18        self.cost = cost
19
20    def __cmp__(self, rhs):    #overload cmp operator, compares only f value
21        if rhs == None: return False
22        if self.f > rhs.f: return 1
23        if self.f == rhs.f: return 0
24        if self.f < rhs.f: return -1
25
26    def __eq__(self, rhs):    ##overload == operator to compare x and y coordinates
27        if rhs == None: return False
28        if self.xPos == rhs.xPos and self.yPos == rhs.yPos: return True
29        return False
30
31    def manhattanDist(self, end):    #calculates manhattan dist from self to given tile
32        #calculates |x2 - x1| + |y2 - y1|
33        return abs(self.xPos - end.xPos) + abs(self.yPos - end.yPos)
34
35
36 class Board:
37     x, y = 0,0    #initialize variables to save board size
38     board = []    #initialize board for saving input file
39
40    def __init__(self, filename):
41        file = open(filename, 'r')    #open boardfile
42        for line in file:    #iterate through file
43            l = list(line[:-1])    #make string into list & removing the last \n
44            if self.x == 0: self.x = len(l)    #if x is not set it is set to the length of input list
45            if 'A' in l:    #if there is an element in input list with value A save as startTile
46                self.startTile = Tile(l.index('A'), (len(self.board)), None, 0)
47            if 'B' in l:    #if there is element in input list with value B save as endTile
48                self.endTile = Tile(l.index('B'), (len(self.board)), None, 0)
49            self.board.append(l)    #append the input list to board when finished investigating it
50            self.y+=1    #increment y for every input list
51
52    def aStar(self):
53        c, o = [], []    #initialize closed and open lists
54
55        #give startTile right h and f value, and append to open list
56        self.startTile.h = self.startTile.manhattanDist(self.endTile)
57        self.startTile.f = self.startTile.h + self.startTile.g
58        o.append(self.startTile)
59
60        while o:    #while there is elements in open-list
61            x = heappop(o)    #pop the best element(smallest f) from
62            if x in c: continue    #if element is already closed continue
63            heappush(c, x)    #push element into closed list
64
65            #if we are at endTile we are finished, return good values
66            if x == self.endTile: return x, True
67            succ = self.genSucc(x)    #find successors to x
68            for s in succ:    #for all successors
69                if s in c: continue    #if already closed continue
70                g = x.g + s.cost    #calculate value of g for successor
71
72            #if s not opened or has better value than already existing opened s
73            if s not in o or g < o[o.index(s)].g:
74                s.g = g    #update g and calculate new h
75                s.h = s.manhattanDist(self.endTile)
76                s.f = s.g + s.h    #update h
77                if s not in o:    #if not opened push into heap at right place
```

```

78         heappush(o, s)
79     return None, False #if we didn't reach endTile return false
80
81 def genSucc(self, current):
82     succ = [] #initialize successor list
83     x, y = current.xPos, current.yPos #get current coordinates
84
85     #array for iterate over successors
86     array = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
87     for x_, y_ in array: #iterate through all sucessor coordinates
88
89         #if inside board get cost associated with tile and append to successor list
90         if 0 <= x_ < self.x and 0 <= y_ < self.y:
91             cost = self.getCost(self.board[y_][x_])
92             succ.append(Tile(x_, y_, current, cost))
93     return succ
94
95 def reconstructPath(self, current):
96     #reconstruct path by starting at a current tile
97     path = [(current.xPos, current.yPos)]
98
99     while current.parent: #and working your way backwards trough parents until there is none
100         path.append((current.parent.xPos, current.parent.yPos))
101         current = current.parent
102     return path
103
104 def getColor(self, symbol):
105     #returns color value based on which symbol
106     if symbol == "A": return (255,0,0)
107     elif symbol == "B": return (0,0,0)
108     elif symbol == "w": return (0,0,255)
109     elif symbol == "m": return (128,128,128)
110     elif symbol == "f": return (0,102,51)
111     elif symbol == "g": return (51,255,51)
112     elif symbol == "r": return (204,102,0)
113     elif symbol == "#": return (0,0,0)
114
115 def getCost(self, symbol):
116     #returns cost based on input symbol
117     if symbol == "w": return 100
118     elif symbol == "m": return 50
119     elif symbol == "f": return 10
120     elif symbol == "g": return 5
121     elif symbol == "r": return 1
122     elif symbol == "B" or symbol == "A" or symbol == "." or symbol == "#": return 1
123
124 def printBoardGraphics(self, path):
125     img = Image.new("RGB", (70*self.x, 70*self.y), "white") #draw background image with 70*x by 70*y resolution
126     idraw = ImageDraw.Draw(img) #make possible to draw
127     for x in range(0, len(self.board)): #iterate trough all elements in board
128         for y in range(0, len(self.board[0])):
129             c = self.getColor(self.board[x][y]) #get color of current element
130             #draw 70x70 rectangle with right color and a black outline
131             idraw.rectangle([(y*70,x*70),(y*70+70,x*70+70)], fill=c, outline=(0,0,0))
132
133             #if coordinate is in path draw smaller square on top to show path
134             if (y,x) in path:
135                 c = (107,97,255)
136                 idraw.rectangle([(y*70+20,x*70+20),(y*70+50,x*70+50)], fill=c, outline=(0,0,0))
137     img.save("pictures/board-2-4.png") #save board
138
139 def main():
140     #make board, run aStar and save graphics
141     a = Board("boards/board-2-4.txt")
142     current, success = a.aStar()
143     if success:
144         path = a.reconstructPath(current)
145         print path
146         a.printBoardGraphics(path)
147
148 if __name__ == "__main__":
149     main()

```


Task 3:

Task 3.1 Astar

```
1 from PIL import Image, ImageDraw, ImageFont
2 from heapq import heappush, heappop
3
4
5 class Tile:      #tile class for every tile generated
6     xPos = 0
7     yPos = 0
8     g = 0
9     h = 0
10    f = 0
11    cost = 0
12    parent = None
13
14    def __init__(self, x, y, parent, cost):      #initialize parent and coordinates
15        self.xPos = x
16        self.yPos = y
17        self.parent = parent
18        self.cost = cost
19
20    def __cmp__(self, rhs):      #overload cmp operator, compares only f value
21        if rhs == None: return False
22        if self.f > rhs.f: return 1
23        if self.f == rhs.f: return 0
24        if self.f < rhs.f: return -1
25
26    def __eq__(self, rhs):      ##overload == operator to compare x and y coordinates
27        if rhs == None: return False
28        if self.xPos == rhs.xPos and self.yPos == rhs.yPos: return True
29        return False
30
31    def manhattanDist(self, end):      #calculates manhattan dist from self to given tile
32        #calculates |x2 - x1| + |y2 - y1|
33        return abs(self.xPos - end.xPos) + abs(self.yPos - end.yPos)
34
35
36 class Board:
37     x, y = 0,0      #initialize variables to save board size
38     board = []      #initialize board for saving input file
39
40    def __init__(self, filename):
41        file = open(filename, 'r')      #open boardfile
42        for line in file:
43            l = list(line[:-1])      #iterate through file
44            #make string into list & removing the last \n
45            if self.x == 0: self.x = len(l)      #if x is not set it is set to the length of input list
46            if 'A' in l:      #if there is an element in input list with value A save as startTile
47                self.startTile = Tile(l.index('A'), (len(self.board)), None, 0)
48            if 'B' in l:      #if there is element in input list with value B save as endTile
49                self.endTile = Tile(l.index('B'), (len(self.board)), None, 0)
50            self.board.append(l)      #append the input list to board when finished investigating it
51            self.y+=1      #increment y for every input list
52
53    def aStar(self):
54        c, o = [], []      #initialize closed and open lists
55
56        #give startTile right h and f value, and append to open list
57        self.startTile.h = self.startTile.manhattanDist(self.endTile)
58        self.startTile.f = self.startTile.h + self.startTile.g
59        o.append(self.startTile)
60
61        while o:      #while there is elements in open-list
62            x = heappop(o)      #pop the best element(smallest f) from
63            if x in c: continue      #if element is already closed continue
64            heappush(c, x)      #push element into closed list
65
66            #if we are at endTile we are finished, return good values
67            if x == self.endTile: return x, True, o, c
68            succ = self.genSucc(x)      #find successors to x
69            for s in succ:      #for all successors
70                if s in c: continue      #if already closed continue
71                g = x.g + s.cost      #calculate value of g for successor
72
73                #if s not opened or has better value than already existing opened s
74                if s not in o or g < o[o.index(s)].g:
75                    s.g = g      #update g and calculate new h
```

```

75         s.h = s.manhattanDist(self.endTile)
76         s.f = s.g + s.h           #update h
77         if s not in o:             #if not opened push into heap at right place
78             heappush(o, s)
79         return None, False         #if we didn't reach endTile return false
80
81     def genSucc(self, current):
82         succ = []                  #initialize successor list
83         x, y = current.xPos, current.yPos #get current coordinates
84
85         #array for iterate over successors
86         array = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
87         for x_, y_ in array:        #iterate through all sucessor coordinates
88
89             #if inside board get cost associated with tile and append to successor list
90             if 0 <= x_ < self.x and 0 <= y_ < self.y:
91                 if self.board[y_][x_] == '#': continue
92                 cost = self.getCost(self.board[y_][x_])
93                 succ.append(Tile(x_, y_, current, cost))
94         return succ
95
96     def reconstructPath(self, current):
97         #reconstruct path by starting at a current tile
98         path = [(current.xPos, current.yPos)]
99
100         while current.parent: #and working your way backwards through parents until there is none
101             path.append((current.parent.xPos, current.parent.yPos))
102             current = current.parent
103         return path
104
105     def getColor(self, symbol):
106         #returns color value based on which symbol
107         if symbol == "A": return (255,0,0)
108         elif symbol == "B": return (0,0,0)
109         elif symbol == "w": return (0,0,255)
110         elif symbol == "m": return (128,128,128)
111         elif symbol == "f": return (0,102,51)
112         elif symbol == "g": return (51,255,51)
113         elif symbol == "r": return (204,102,0)
114         elif symbol == "#": return (0,0,0)
115
116     def getCost(self, symbol):
117         #returns cost based on input symbol
118         if symbol == "w": return 100
119         elif symbol == "m": return 50
120         elif symbol == "f": return 10
121         elif symbol == "g": return 5
122         elif symbol == "r": return 1
123         elif symbol == "B" or symbol == "A" or symbol == "." or symbol == "#": return 1
124
125     def printBoardGraphics(self, path, opened, closed):
126         img = Image.new("RGB", (70*self.x, 70*self.y), "white") #draw background image with 70*x by 70*y resolution
127         idraw = ImageDraw.Draw(img) #make possible to draw
128         for x in range(0, len(self.board)): #iterate through all elements in board
129             for y in range(0, len(self.board[0])):
130                 c = self.getColor(self.board[x][y]) #get color of current element
131                 #draw 70x70 rectangle with right color and a black outline
132                 idraw.rectangle([(y*70,x*70),(y*70+70,x*70+70)], fill=c, outline=(0,0,0))
133
134                 #if coordinate is in path draw smaller square on top to show path
135                 if (y,x) in path:
136                     c = (107,97,255)
137                     idraw.rectangle([(y*70+20,x*70+20),(y*70+50,x*70+50)], fill=c, outline=(0,0,0))
138                 elif (y,x) in closed:
139                     c = (0,0,0)
140                     font = ImageFont.truetype("arial.ttf", size=40)
141                     idraw.text([(y*70+20,x*70+20)], "X", c, font)
142                 elif (y,x) in opened:
143                     c = (0,0,0)
144                     font = ImageFont.truetype("arial.ttf", size=100)
145                     idraw.text([(y*70+15,x*70+5)], "+", c, font)
146         img.save("pictures/Astar board-1-4.png") #save board
147
148     def main():
149         #make board, run aStar and save graphics
150         a = Board("boards/board-1-4.txt")
151         current, success, opened, closed = a.aStar()
152         o, c = [], []
153         if success:
154             path = a.reconstructPath(current)

```



```

155     for element in opened:
156         o.append((element.xPos, element.yPos))
157     for element in closed:
158         c.append((element.xPos, element.yPos))
159     a.printBoardGraphics(path, o, c)
160
161 if __name__ == "__main__":
162     main()

```

Task 3:2 BFS

```

1 from PIL import Image, ImageDraw, ImageFont
2 from heapq import heappush, heappop
3
4
5 class Tile:      #tile class for every tile generated
6     xPos = 0
7     yPos = 0
8     g = 0
9     h = 0
10    f = 0
11    cost = 0
12    parent = None
13
14    def __init__(self, x, y, parent, cost):    #initialize parent and coordinates
15        self.xPos = x
16        self.yPos = y
17        self.parent = parent
18        self.cost = cost
19
20    def __cmp__(self, rhs):    #overload cmp operator, compares only f value
21        if rhs == None: return False
22        if self.f > rhs.f: return 1
23        if self.f == rhs.f: return 0
24        if self.f < rhs.f: return -1
25
26    def __eq__(self, rhs):    ##overload == operator to compare x and y coordinates
27        if rhs == None: return False
28        if self.xPos == rhs.xPos and self.yPos == rhs.yPos: return True
29        return False
30
31    def manhattanDist(self, end):    #calculates manhattan dist from self to given tile
32        #calculates |x2 - x1| + |y2 - y1|
33        return abs(self.xPos - end.xPos) + abs(self.yPos - end.yPos)
34
35
36 class Board:
37     x, y = 0,0    #initialize variables to save board size
38     board = []    #initialize board for saving input file
39
40    def __init__(self, filename):
41        file = open(filename, 'r')    #open boardfile
42        for line in file:    #iterate through file
43            l = list(line[:-1])    #make string into list & removing the last \n
44            if self.x == 0: self.x = len(l)    #if x is not set it is set to the length of input list
45            if 'A' in l:    #if there is an element in input list with value A save as startTile
46                self.startTile = Tile(l.index('A'), (len(self.board)), None, 0)
47            if 'B' in l:    #if there is element in input list with value B save as endTile
48                self.endTile = Tile(l.index('B'), (len(self.board)), None, 0)
49            self.board.append(l)    #append the input list to board when finished investigating it
50            self.y+=1    #increment y for every input list
51
52
53
54    def bfs(self):
55        c, o = [], []    #initialize closed and open lists
56
57        #give startTile right h and f value, and append to open list
58        self.startTile.h = self.startTile.manhattanDist(self.endTile)
59        self.startTile.f = self.startTile.h + self.startTile.g
60        o.append(self.startTile)
61
62        while o:    #while there is elements in open-list
63            x = o.pop(0)    #pop the best element(smallest f) from
64            if x in c: continue    #if element is already closed continue
65            c.append(x)    #push element into closed list
66
67            #if we are at endTile we are finished, return good values
68            if x == self.endTile: return x, True, o, c

```

```

69         succ = self.genSucc(x)           #find successors to x
70         for s in succ:                   #for all successors
71             if s in c: continue           #if already closed continue
72             g = x.g + s.cost              #calculate value of g for successor
73
74             #if s not opened or has better value than already existing opened s
75             if s not in o or g < o[o.index(s)].g:
76                 s.g = g                   #update g and calculate new h
77                 s.h = s.manhattanDist(self.endTile)
78                 s.f = s.g + s.h           #update h
79                 if s not in o:             #if not opened push into heap at right place
80                     o.append(s)
81         return None, False                #if we didn't reach endTile return false
82
83     def genSucc(self, current):
84         succ = []                         #initialize successor list
85         x, y = current.xPos, current.yPos #get current coordinates
86
87         #array for iterate over successors
88         array = [(x-1, y), (x+1,y), (x, y-1), (x, y+1)]
89         for x_,y_ in array:               #iterate through all sucessor coordinates
90
91             #if inside board get cost associated with tile and append to successor list
92             if 0 <= x_ < self.x and 0 <= y_ < self.y:
93                 if self.board[y_][x_] == '#': continue
94                 cost = self.getCost(self.board[y_][x_])
95                 succ.append(Tile(x_, y_, current, cost))
96         return succ
97
98     def reconstructPath(self, current):
99         #reconstruct path by starting at a current tile
100         path = [(current.xPos, current.yPos)]
101
102         while current.parent: #and working your way backwards trough parents until there is none
103             path.append((current.parent.xPos, current.parent.yPos))
104             current = current.parent
105         return path
106
107     def getColor(self, symbol):
108         #returns color value based on which symbol
109         if symbol == "A": return (255,0,0)
110         elif symbol == "B": return (0,0,0)
111         elif symbol == "w": return (0,0,255)
112         elif symbol == "m": return (128,128,128)
113         elif symbol == "f": return (0,102,51)
114         elif symbol == "g": return (51,255,51)
115         elif symbol == "r": return (204,102,0)
116         elif symbol == "#": return (0,0,0)
117
118     def getCost(self, symbol):
119         #returns cost based on input symbol
120         if symbol == "w": return 100
121         elif symbol == "m": return 50
122         elif symbol == "f": return 10
123         elif symbol == "g": return 5
124         elif symbol == "r": return 1
125         elif symbol == "B" or symbol == "A" or symbol == "." or symbol == "#": return 1
126         return 1
127
128     def printBoardGraphics(self, path, opened, closed):
129         img = Image.new("RGB", (70*self.x, 70*self.y), "white") #draw background image with 70*x by 70*y resolution
130         idraw = ImageDraw.Draw(img) #make possible to draw
131         for x in range(0, len(self.board)): #iterate trough all elements in board
132             for y in range(0, len(self.board[0])):
133                 c = self.getColor(self.board[x][y]) #get color of current element
134                 #draw 70x70 rectangle with right color and a black outline
135                 idraw.rectangle([(y*70,x*70),(y*70+70,x*70+70)], fill=c, outline=(0,0,0))
136
137                 #if coordinate is in path draw smaller square on top to show path
138                 if (y,x) in path:
139                     c = (107,97,255)
140                     idraw.rectangle([(y*70+20,x*70+20),(y*70+50,x*70+50)], fill=c, outline=(0,0,0))
141                 elif (y,x) in closed:
142                     c = (0,0,0)
143                     font = ImageFont.truetype("arial.ttf", size=40)
144                     idraw.text([(y*70+20,x*70+20)], "X", c, font)
145                 elif (y,x) in opened:
146                     c = (0,0,0)
147                     font = ImageFont.truetype("arial.ttf", size=100)
148                     idraw.text([(y*70+15,x*70+5)], "*", c, font)

```

```

149         img.save("pictures/bfs_board-1-4.png")           #save board
150
151
152 def main():
153     #make board, run bfs and save graphics
154     a = Board("boards/board-1-4.txt")
155     current, success, opened, closed = a.bfs()
156     o, c = [], []
157     if success:
158         path = a.reconstructPath(current)
159         for element in opened:
160             o.append((element.xPos, element.yPos))
161         for element in closed:
162             c.append((element.xPos, element.yPos))
163         a.printBoardGraphics(path, o, c)
164
165 if __name__ == "__main__":
166     main()

```

Task 3:3 Dijkstra

```

1 from PIL import Image, ImageDraw, ImageFont
2 from heapq import heappush, heappop
3
4
5 class Tile:      #tile class for every tile generated
6     xPos = 0
7     yPos = 0
8     g = 0
9     h = 0
10    f = 0
11    cost = 0
12    parent = None
13
14    def __init__(self, x, y, parent, cost):      #initialize parent and coordinates
15        self.xPos = x
16        self.yPos = y
17        self.parent = parent
18        self.cost = cost
19
20    def __cmp__(self, rhs):      #overload cmp operator, compares only f value
21        if rhs == None: return False
22        if self.g > rhs.g: return 1
23        if self.g == rhs.g: return 0
24        if self.g < rhs.g: return -1
25
26    def __eq__(self, rhs):      ##overload == operator to compare x and y coordinates
27        if rhs == None: return False
28        if self.xPos == rhs.xPos and self.yPos == rhs.yPos: return True
29        return False
30
31    def manhattanDist(self, end):      #calculates manhattan dist from self to given tile
32        #calculates |x2 - x1| + |y2 - y1|
33        return abs(self.xPos - end.xPos) + abs(self.yPos - end.yPos)
34
35
36 class Board:
37     x, y = 0,0      #initialize variables to save board size
38     board = []      #initialize board for saving input file
39
40    def __init__(self, filename):
41        file = open(filename, 'r')      #open boardfile
42        for line in file:      #iterate through file
43            l = list(line[:-1])      #make string into list & removing the last \n
44            if self.x == 0: self.x = len(l)      #if x is not set it is set to the length of input list
45            if 'A' in l:      #if there is an element in input list with value A save as startTile
46                self.startTile = Tile(l.index('A'), (len(self.board)), None, 0)
47            if 'B' in l:      #if there is element in input list with value B save as endTile
48                self.endTile = Tile(l.index('B'), (len(self.board)), None, 0)
49            self.board.append(l)      #append the input list to board when finished investigating it
50            self.y+=1      #increment y for every input list
51
52
53
54    def dijkstra(self):
55        c, o = [], []      #initialize closed and open lists
56
57        #give startTile right h and f value, and append to open list
58        self.startTile.h = self.startTile.manhattanDist(self.endTile)

```

```

59     self.startTile.f = self.startTile.h + self.startTile.g
60     o.append(self.startTile)
61
62     while o:
63         x = heappop(o)
64         if x in c: continue
65         heappush(c, x)
66
67         #if we are at endTile we are finished, return good values
68         if x == self.endTile: return x, True, o, c
69         succ = self.genSucc(x)
70         for s in succ:
71             if s in c: continue
72             g = x.g + s.cost
73
74             #if s not opened or has better value than already existing opened s
75             if s not in o or g < o[o.index(s)].g:
76                 s.g = g
77                 s.h = s.manhattanDist(self.endTile)
78                 s.f = s.g + s.h
79                 if s not in o:
80                     heappush(o, s)
81         return None, False
82
83     def genSucc(self, current):
84         succ = []
85         x, y = current.xPos, current.yPos
86
87         #array for iterate over successors
88         array = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
89         for x_, y_ in array:
90
91             #if inside board get cost associated with tile and append to successor list
92             if 0 <= x_ < self.x and 0 <= y_ < self.y:
93                 if self.board[y_][x_] == '#': continue
94                 cost = self.getCost(self.board[y_][x_])
95                 succ.append(Tile(x_, y_, current, cost))
96         return succ
97
98     def reconstructPath(self, current):
99         #reconstruct path by starting at a current tile
100         path = [(current.xPos, current.yPos)]
101
102         while current.parent:
103             path.append((current.parent.xPos, current.parent.yPos))
104             current = current.parent
105         return path
106
107     def getColor(self, symbol):
108         #returns color value based on which symbol
109         if symbol == "A": return (255,0,0)
110         elif symbol == "B": return (0,0,0)
111         elif symbol == "w": return (0,0,255)
112         elif symbol == "m": return (128,128,128)
113         elif symbol == "f": return (0,102,51)
114         elif symbol == "g": return (51,255,51)
115         elif symbol == "r": return (204,102,0)
116         elif symbol == "#": return (0,0,0)
117
118     def getCost(self, symbol):
119         #returns cost based on input symbol
120         if symbol == "w": return 100
121         elif symbol == "m": return 50
122         elif symbol == "f": return 10
123         elif symbol == "g": return 5
124         elif symbol == "r": return 1
125         elif symbol == "B" or symbol == "A" or symbol == "." or symbol == "#": return 1
126
127     def printBoardGraphics(self, path, opened, closed):
128         img = Image.new("RGB", (70*self.x, 70*self.y), "white")
129         idraw = ImageDraw.Draw(img)
130         for x in range(0, len(self.board)):
131             for y in range(0, len(self.board[0])):
132                 c = self.getColor(self.board[x][y])
133                 idraw.rectangle([(y*70,x*70),(y*70+70,x*70+70)], fill=c, outline=(0,0,0))
134
135                 #if coordinate is in path draw smaller square on top to show path
136                 if (y,x) in path:
137                     c = (107,97,255)

```

```

139         idraw.rectangle([(y*70+20,x*70+20),(y*70+50,x*70+50)], fill=c, outline=(0,0,0))
140     elif (y,x) in closed:
141         c = (0,0,0)
142         font = ImageFont.truetype("arial.ttf", size=40)
143         idraw.text([y*70+20,x*70+20], "X", c, font)
144     elif (y,x) in opened:
145         c = (0,0,0)
146         font = ImageFont.truetype("arial.ttf", size=100)
147         idraw.text([y*70+15,x*70+5], "*", c, font)
148
149     img.save("pictures/djikstra board-1-1.png")
150
151 def main():
152     #make board, run bfs and save graphics
153     a = Board("boards/board-1-1.txt")
154     current, success, opened, closed = a.dijkstra()
155     o, c = [], []
156     if success:
157         path = a.reconstructPath(current)
158         for element in opened:
159             o.append((element.xPos, element.yPos))
160         for element in closed:
161             c.append((element.xPos, element.yPos))
162         a.printBoardGraphics(path, o, c)
163
164 if __name__ == "__main__":
165     main()

```

Board 1-3

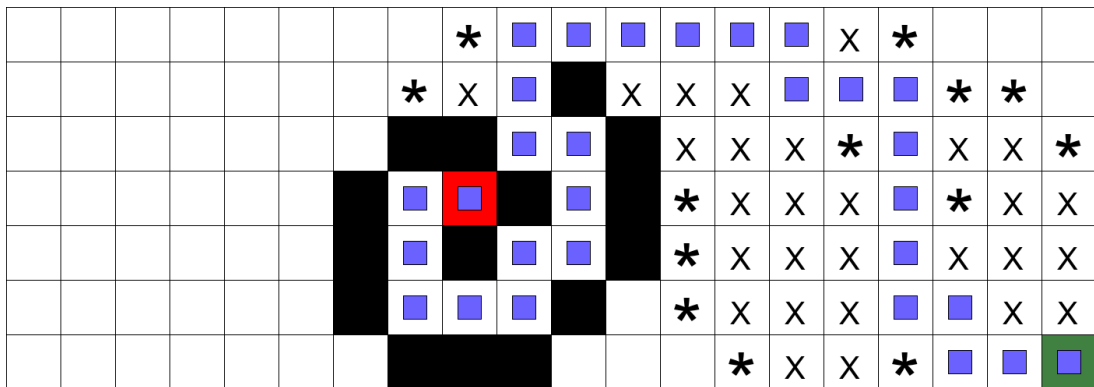


Figure 1: Board 1-3 with A* search

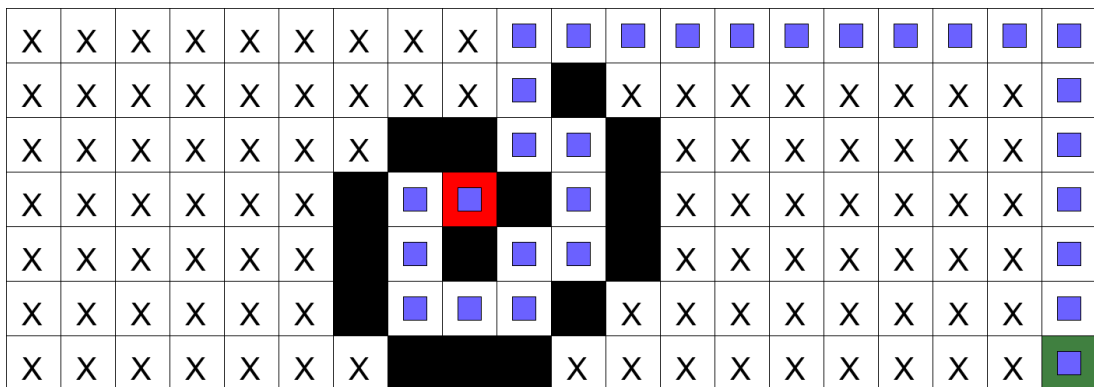


Figure 2: Board 1-3 with BFS search

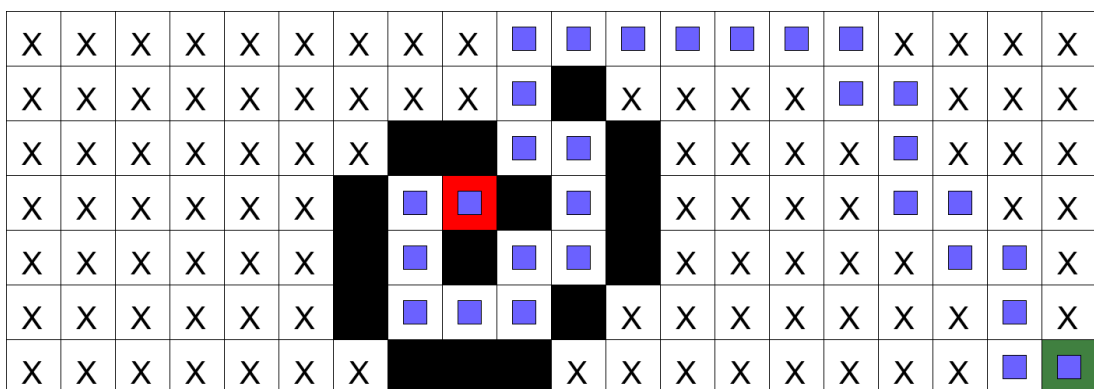


Figure 3: Board 1-3 with Dijkstra search

Board 2-4

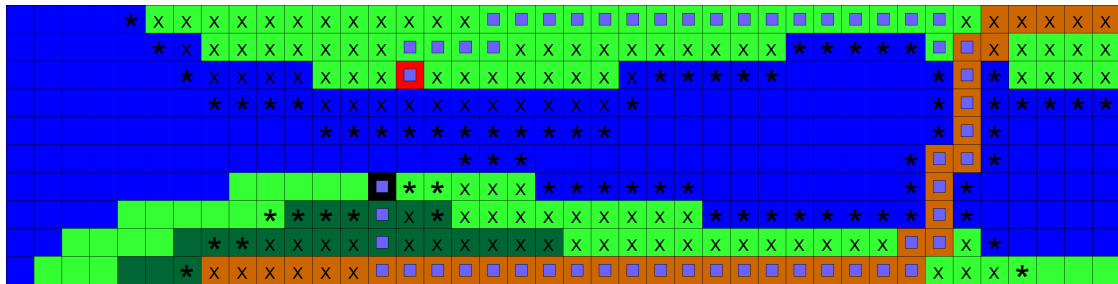


Figure 4: Board 2-4 with A* search

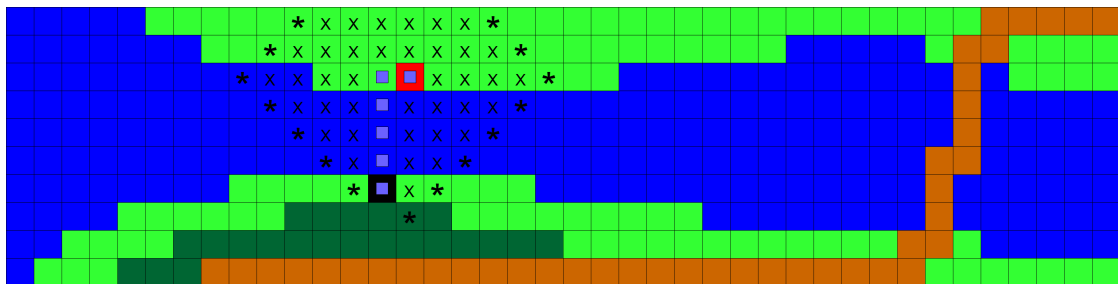


Figure 5: Board 2-4 with BFS search

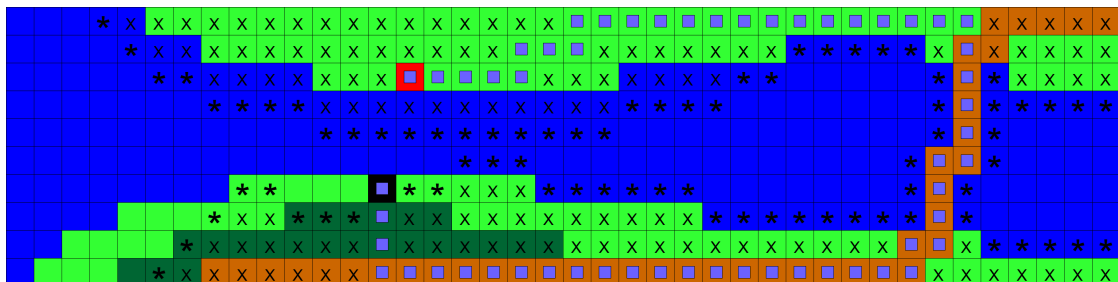


Figure 6: Board 2-4 with Dijkstra search

Comments:

Board 1-3 has no weights and therefore even BFS finds a optimal route. BFS and Dijkstra opens and processes a lot of more tiles than A*. This is the kind of task where A* shines. It heads directly en route to the endpoint, compared to Dijkstra which basically has found the shortest path to every tile on the map by the time it's finished. The path that Dijkstra and A* finds are equally as good, they are just different because of the implimentation.

Board 2-4 is weighed and therefore BFS does not do a good job. It finds a path to the endpoint, which is simply the first way it sees. But it is a lot more costly to go this way. Dijkstra and A* basically finds the same way. This is a task where Djistra does comparatibily a lot better than last time. The number of opened and closed tiles is about the same as A*.