Maze Router

Using A* Algorithm in Python Language

Ramy ElGendi - 900170269

Ali Moussa - 900160311

TABLE OF CONTENTS

A* Algorithm

What is A*?

Best-first search algorithm to find the goal node with shortest time/distance

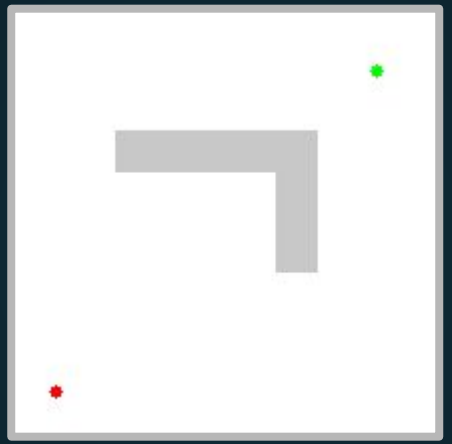
$$f(n) = h(n)+g(n)$$

n - next node path

h(n) - heuristic function

g(n) - cost from start to node

A* Illustration in general



https://upload.wikimedia.org/wikipedia/commons/5/5d/Astar_progress_animation.gif

Code breakdown

01

FileParsing.py

Deals with taking an input file and parsing the data into lists

File Parsing Class Constructor

```
def __init__(self, filename):  # Class Constructor
    self.filename = filename
    self.netList = []
    self.netNames = []
```

This class constructor takes 1 parameter: file address, and parses the file.

netList - List of lists of integers that stores the paths of each net

netNames - List of strings that stores the names of each net

Ex:
net1 (1,0,0) (1,0,10) ....
net2 (1,1,1) (1,2,4) ....
netlist = [[[1,0,0],[1,0,10],...], [[1,1,1],[1,2,4],...]]
netNames = [“net1”, “net2”]

Libraries:

os - Used to check if file exists

```python
def Exists(self):
    if os.path.isfile(self.filename):
        print(self.filename + " has been imported!")
        return True
    return False
```

Exists Function

This function uses the library: os.path to check if the file the user wants to parse exists or not

```python
def Parse(self):
    file = open(self.filename, 'r')
    lines = file.readlines()

    for line in lines:  # net1 (1, 10, 20) (2, 30, 50) (1, 5, 100)
        netList = []
        self.netNames.append(line[:line.index('(')])
        modified = line[line.index('(') + 1:].split('(')
        for x in modified:
            if ')' in x:
                i = x.replace(')', '')
            if '\n' in i:
                i = i.replace('\n', '')
            netList.append(i.split(','))
        self.netList.append(netList)

    return self.netList
```

Parse Function

This function takes a file and reads it line by line

(each line representing a net)

and saves the paths found in netList

```python
def OutputList(self, pathsList, file_out):
    out = open(file_out, "w")
    for i, line in enumerate(pathsList):
        out.write(self.netNames[i] + " ")
        for path in line:
            out.write("(" + str(path[2]) + " " + str(path[0]) + " " + str(path[1]) + ") ")

        out.write("\n")

    return True
```

OutputList Function

This function takes a paths list in the format: [[1,0,0],[1,0,1],...,[1,10,10] and matches each list in pathsList with the net name in netNames list

02

AStar.py

Routing algorithm class

AStar Class Constructor

```
def __init__(self, via=1, height=1000, width=1000, layers=10):
    self.height = height
    self.width = width
    self.layers = layers
    self.via = via
    self.grid = np.empty([width, height], dtype=list)  # Creati

    for x in range(height):  # Initializing Empty Grid
        for y in range(width):
            for z in range(layers + 1):  # Creating the amount
                if z == 0:
                    self.grid[x][y] = []
                else:
                    self.grid[x][y] += [0]
```

Libraries:

numpy as py - Used to create the grid (faster than normal array)

This class constructor takes 4 parameters:

- **via** - The number of vias to be used if necessary (to minimize the usage of via, it is set to 1)

- **height** - the “y” of the grid
- **width** - the “x” of the grid
- **layers** - the “z” of the grid (amount of layers)

Here, a 2D grid is created, each point in the grid is broken down into a list of points (representing the layers)

```python
def H_Fn(self, x1, y1, z1, x2, y2, z2):  # xyz1s are the source, xyz2s are the target
    return abs(x2 - x1) + abs(y2 - y1) + abs(z2 - z1)
```

H_Fn Function

Heuristic function using manhattan distancing

```python
def Path(self, z1, x1, y1, z2, x2, y2):
    # Initialization
    FinalPath = [[x1, y1, z1]]
    GCost = [0]
    H = [self.H_Fn(x1, y1, z1, x1, y1, z1)]
    F = [GCost[0] + H[0]]

    self.grid[x1][y1][z1 - 1] = 1

    CurrentNode = FinalPath[0]
    timeout = 0
    while not CurrentNode == [x2, y2, z2] and timeout < 1000000:
        timeout += 1
        # Check if target==source
        CurrentNode_, f, g = self.Next(CurrentNode[0], CurrentNode[1], CurrentNode[2], x2, y2, z2)

        CurrentNode = CurrentNode_
        self.grid[CurrentNode[0]][CurrentNode[1]][CurrentNode[2] - 1] = 1
        FinalPath.append(CurrentNode)
        F.append(f)
        GCost.append(g)
    if timeout < 1000000:
        return FinalPath
    else:
        return []
```

Path Function

Function used to calculate and return a list of path of nodes from source to target

FinalPath - List of the final path selected from source to target
GCost g(n) - cost from initial path to current
H h(n) - heuristic function for next node
F f(n) - determines which is the next node
timeout - To stop searching if no path can be found

Astar Class Functions

Condition for first 2 layers to make sure that next node is reachable

```
def Next(self, x1, y1, z1, x2, y2, z2):  # xyz1s are the source, xyz2s are the target
    F_final = self.height * self.width
    Final = []
    if z1 == 1:  # Source Node is in layer 1
        if x1 - 1 >= 0 and self.grid[x1 - 1][y1][z1] != 1 and self.grid[x1 - 1][y1][z1 + 1] != 1:
            Final.append([x1 - 1, y1, z1 + 1])
        if x1 >= 0 and self.grid[x1][y1 - 1][z1 - 1] != 1:
            Final.append([x1, y1 - 1, z1])
        if x1 + 1 < self.height and self.grid[x1 + 1][y1][z1] != 1 and self.grid[x1 + 1][y1][z1 + 1] != 1:
            Final.append([x1 + 1, y1, z1 + 1])
        if y1 + 1 < self.width and self.grid[x1][y1 + 1][z1 - 1] != 1:
            Final.append([x1, y1 + 1, z1])
    elif z1 == 2:  # Source Node is in layer 2
        if z2 == 2:
            z = z2 - 1
        else:
            z = z2

        if x1 - 1 >= 0 and self.grid[x1 - 1][y1][z1 - 1] != 1:
            Final.append([x1 - 1, y1, z1])
        if y1 - 1 >= 0 and (self.grid[x1][y1 - 1][0] != 1 or self.grid[x1][y1 - 1][2] != 1):
            Final.append([x1, y1 - 1, 1])
        if x1 + 1 < self.height and self.grid[x1 + 1][y1][1] != 1:
            Final.append([x1 + 1, y1, z1])
        if y1 + 1 < self.width and (self.grid[x1][y1][0] != 1 or self.grid[x1][y1 + 1][2] != 1):
            Final.append([x1, y1 + 1, z1])
```

Condition to support multi layer.

```
else:  # Support for multi layer (NOT TESTED)

    if x1 - 1 >= 0 and self.grid[x1 - 1][y1][0] != 1 and self.grid[x1 - 1][y1][1] != 1:
        Final.append([x1 - 1, y1, z1 - 1])
    if y1 - 1 >= 0 and self.grid[x1][y1 - 1][2] != 1:
        Final.append([x1, y1 - 1, z1 + 1])
    if x1 + 1 < self.height and self.grid[x1 + 1][y1][0] != 1 and self.grid[x1 + 1][y1][1] != 1:
        Final.append([x1, y1 + 1, z1 + 1])
    if y1 + 1 < self.width and self.grid[x1][y1 + 1][2] != 1:
        Final.append([x1, y1 + 1, z1 + 1])
```

Next Function Breakdown

```
if len(Final) == 0:
    return (x1, y1, z1), 0, 0
else:
    for node in Final:
        if z1 == node[2]:
            F = 1 + self.H_Fn(node[0], node[1], node[2], x2, y2, z2)
        else:
            F = self.via + self.H_Fn(node[0], node[1], node[2], x2, y2, z2)

        if F < F_final:  # Picking the lowest
            F_final = F
            Node = node
            G = F - self.H_Fn(node[0], node[1], node[2], x2, y2, z2)

return Node, F_final, G
```

After making sure there are next cell, and source cell is not target cell, we find the lowest path (according to f(n)), and returns the next node, cost and f(n)

03

visual.py

Visualizing the final path on graph

Visual Class Constructor & Functions

```python
class visual():
    def __init__(self, pathsList):  # Class Constructor
        self.pathsList = pathsList

    def figure(self):
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')

        for i in self.pathsList:
            ax.plot([v[0] for v in i], [v[1] for v in i], [v[2] for v in i])
        ax.set_xlabel('Width')
        ax.set_ylabel('Height')
        ax.set_zlabel('Layers')
        ax.set_xticks([0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000])
        ax.set_yticks([0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000])
        ax.set_zticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
        plt.title("A* Router Simulation\n")
        plt.show()
```

This class constructor takes a list of 3 coordinates

Figure Function

This function prints the pathsList using matplotlib. Each net is represented in a color

Libraries:

matplotlib.pyplot as plt - Used to 3D to plot the grid

04

Simulation.py

The main file that links all of the classes

Simulation.py

1) Importing classes

```
import FileParsing
import sys
import AStar
import visual
```

2) Input File Parsing

```
inputFile = input("Enter address for your input file. [Format: /Users/ramyelgendi/Desktop/AStarRouter/input.txt] :\n")
# Parsing Input
file = FileParsing.FileParsing(inputFile)
if not file.Exists():
    print("Parsing file does not exist!")
    sys.exit()
netList = file.Parse()
print("Input has been parsed successfully!")
```

3) Routing

```
# Create Grid
router = AStar.AStar(1, 1000, 1000, 6)

pathsList = []
print("Beginning simulation!")
for line in netList:
    Path = []
    for i, node in enumerate(line):
        try:
            source = node
            target = line[i + 1]
            netPath = router.Path(int(source[0]), int(source[1]), int(source[2]), int(target[0]), int(target[1]),
                                  int(target[2]))
            Path = Path + netPath
        except IndexError:
            pass
    pathsList.append(Path)
```

4) Creating output file & graph

```
# Writing Output
outputFile = inputFile.replace('.txt', '_out.txt')
status = file.OutputList(pathsList, outputFile)
if status:
    print("Completed Successfully! ")

    while True:
        choice = input("Do you want to see results on a graph? (yes/no): \n")
        if choice == "yes":
            visual = visual.visual(pathsList)
            visual.figure()  # Draw
            sys.exit()
        elif choice == "no":
            sys.exit()
        else:
            print("Invalid Entry! Please type yes/no only")


else:
    print("Failed! ")
    sys.exit()
```

SIMULATION

TEST CASES

```
net1 (1, 10, 20) (2, 30, 50) (1, 5, 100)
net2 (2, 100, 200) (1, 300, 50)
net3 (1, 100, 50) (2, 300, 150) (2, 50, 50) (1, 2, 2)
net4 (3, 200, 30) (4, 60, 10)
net5 (4, 0, 100) (4, 300, 200)
net6 (6, 100, 100) (6, 100, 900)
net7 (1, 20, 30) (1, 40, 10)
net8 (1, 700, 70) (1, 800, 800) (2, 900, 900)
```

input.txt

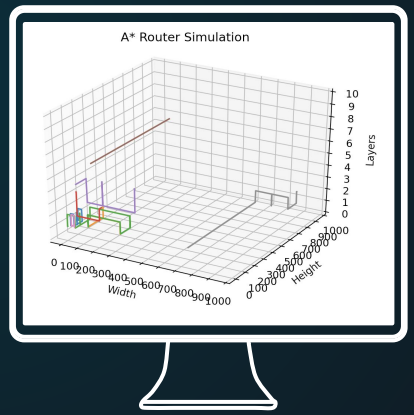
```
net1  (1 10 20) (2 11 20) (2 12 20) (2 13 20) (2 14 20) (2 15 20) (2 16 20) (2 17 2
net2  (2 100 200) (1 100 199) (1 100 198) (1 100 197) (1 100 196) (1 100 195) (1 10
net3  (1 100 50) (2 99 50) (2 100 50) (1 100 51) (2 101 51) (2 102 51) (2 103 51) (
net4  (3 200 30) (3 200 29) (3 200 28) (3 200 27) (3 200 26) (3 200 25) (3 200 24)
net5  (4 0 100) (4 0 101) (4 0 102) (4 0 103) (4 0 104) (4 0 105) (4 0 106) (4 0 10
net6  (6 100 100) (6 100 101) (6 100 102) (6 100 103) (6 100 104) (6 100 105) (6 10
net7  (1 20 30) (1 20 29) (1 20 28) (1 20 27) (1 20 26) (1 20 25) (1 20 24) (1 20 2
net8  (1 700 70) (1 700 71) (1 700 72) (1 700 73) (1 700 74) (1 700 75) (1 700 76)
```

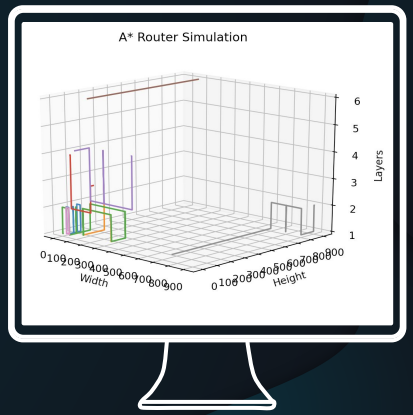
input_out.txt

```
Enter address for your input file. [Format: /User
input.txt
input.txt has been imported!
Input has been parsed successfully!
Beginning simulation!
Completed Successfully!
Do you want to see results on a graph? (yes/no):
yes

Process finished with exit code 0
```

Simulation





Graph Visualization (Rotatable 3D)

THANKS!

Credits:

Ramy ElGendi - 900170269

Ali Moussa - 900160311