

ARTIFICIAL INTELLIGENCE

EXPERIMENT NO: 5

DEVELOPING BEST FIRST SEARCH AND A*Algorithm FOR REAL WORLD PROBLEM

Nikith Kumar Seemakurthi

RA1911003020480

Aim:

To develop best first search and A*algorithm for a real world problem.

Algorithm:

1. Initialize an array representing amaze with 0's and 1's.
2. Declare start and end points representing the top left and bottom right corner of the maze.
3. Initialize two array representing open_list and closed_list
4. Starting with the start point, check all the adjacent points.
5. If an adjacent point is a walkable terrain, check all its adjacent points, and keep track of the path.
6. Continue the process till you find the endpoint.
7. Print the path

Source Code:

```
class Node():  
    def __init__(self, parent = None, position = None):  
        self.parent = parent  
        self.position = position  
        self.g = 0  
        self.h = 0  
        self.f = 0  
  
    def __eq__(self, other):  
        return self.position == other.position  
  
def astar(maze, start, end):  
    start_node = Node(None, start)
```

```
start_node.g = start_node.h = start_node.f = 0
end_node = Node(None, end)
end_node.g = end_node.h = end_node.f = 0
```

```
open_list = []
closed_list = []
```

```
open_list.append(start_node)
```

```
while len(open_list)>0:
```

```
    current_node = open_list[0]
    current_index = 0
    for index,item in enumerate(open_list):
        if item.f<current_node.f:
            current_node = item
            current_index = index
```

```
    open_list.pop(current_index)
    closed_list.append(current_node)
```

```
if current_node == end_node:
```

```
    path = []
    current = current_node
    while current is not None:
```

```
        path.append(current.position)
        current = current.parent
    return path[::-1]
```

```

children = []
for new_position in [(0,-1),(0,1),(-1,0),(1,0),(-1,-1),(-1,1),(1,-1),(1,1)]:
    node_position = (current_node.position[0]+new_position[0],
                    current_node.position[1]+new_position[1])

    if node_position[0] > (len(maze)-1) or node_position[0] < 0 or node_position[1] >
(len(maze[len(maze)-1])-1) or node_position[1]<0:
        continue

    if maze[node_position[0]][node_position[1]] != 0:
        continue
    new_node=Node(current_node,node_position)

    children.append(new_node)

for child in children:
    for closed_child in closed_list:
        if child == closed_child:
            continue

        child.g = current_node.g+1
        child.h = ((child.position[0]-end_node.position[0])**2) + ((child.position[1]-
end_node.position[1])**2)
        child.f = child.g+child.h

        for open_node in open_list:
            if child == open_node and child.g > open_node.g:
                continue

        open_list.append(child)

def main():
    maze=[[0,0,0,0,1,0,0,0,0],
[0,0,0,0,1,0,0,0,0],

```

```

[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,1,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0]]

```

```
start = (0,0)
```

```
end = (7,6)
```

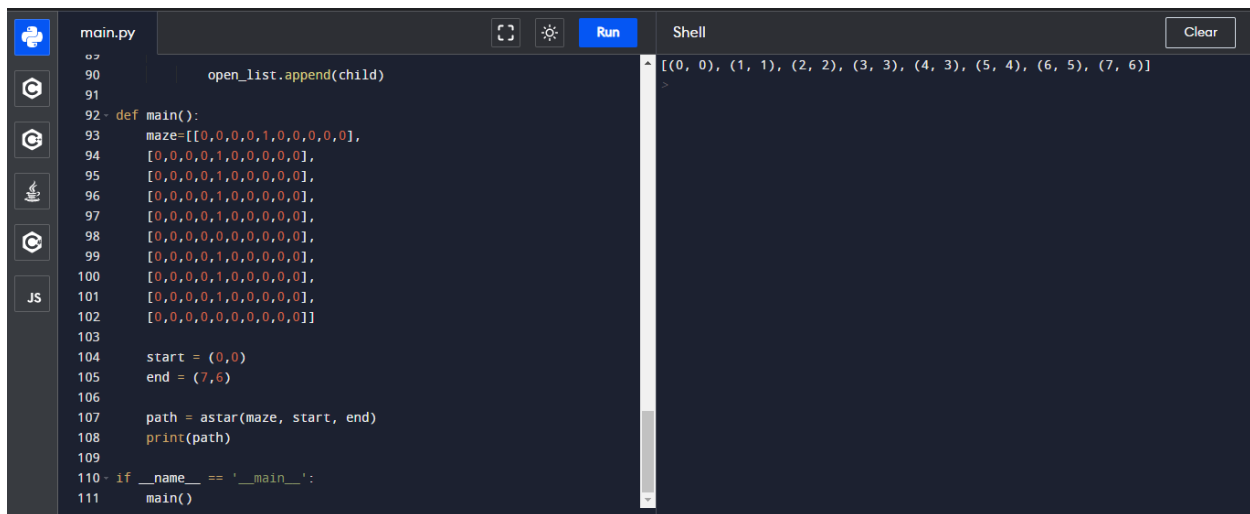
```
path = astar(maze, start, end)
```

```
print(path)
```

```
if __name__ == '__main__':
```

```
    main()
```

Output:



The screenshot shows a code editor with a file named 'main.py'. The code defines a 10x10 maze, sets a start point at (0,0) and an end point at (7,6), and uses the A* algorithm to find a path. The output in the shell window shows the path as a list of coordinates: [(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6)].

```

main.py
90     open_list.append(child)
91
92 - def main():
93     maze=[[0,0,0,0,1,0,0,0,0,0],
94           [0,0,0,0,1,0,0,0,0,0],
95           [0,0,0,0,1,0,0,0,0,0],
96           [0,0,0,0,1,0,0,0,0,0],
97           [0,0,0,0,1,0,0,0,0,0],
98           [0,0,0,0,0,0,0,0,0,0],
99           [0,0,0,0,1,0,0,0,0,0],
100          [0,0,0,0,1,0,0,0,0,0],
101          [0,0,0,0,1,0,0,0,0,0],
102          [0,0,0,0,0,0,0,0,0,0]]
103
104     start = (0,0)
105     end = (7,6)
106
107     path = astar(maze, start, end)
108     print(path)
109
110 - if __name__ == '__main__':
111     main()

```

```

Shell
Clear
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6)]

```

Result:

Thus an A* path finding algorithm was implemented for a real world problem.