Kevin Scott
Dr. Forouraghi
CSC 362
10/21/25

1. **Problem 1 (10 points)**

   Modify **AStarMaze** to compare the behaviors of the **Greedy Best-First** and **A*** search algorithms. You need to modify the maze configuration so you can visually observe differences in the optimum paths generated by the two algorithms. Your report should include a side-by-side comparison of the two approaches similar to the graph shown below along with your explanation. You only need to draw the shortest paths and not the highlighted frontiers.

   The only changes that I had to make to the initial code for the A* version was to change the maze configuration. For the Greedy Best-First algorithm I had to edit the section where moving incurs a unit cost and I essentially set that unit cost to 0. Thus, making the algorithm greedy. The A* algorithm is most definitely more optimal than the Greedy one.

```
################################################################
#### Modify the wall cells to experiment with different maze
#### configurations.
################################################################
maze = [
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
```

A*

```
### Update the evaluation function for the cell n: f(n) = h(n)
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h # g(n) = 0
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

```
#### The cost of moving to a new position is 1 unit
new_g = current_cell.g
```

g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=8 h=10									
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

Greedy

```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

```
    #### The cost of moving to a new position is 1 unit
    new_g = current_cell.g + 1
```

g=0 h=17	g=0 h=16	g=0 h=15	g=0 h=14	g=0 h=13	g=0 h=12	g=0 h=11	g=0 h=10	g=0 h=9	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=8	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=7	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=6	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=5	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=4	
g=0 h=11	g=0 h=10	g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	
g=0 h=10									
g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2	g=0 h=1	g=0 h=0

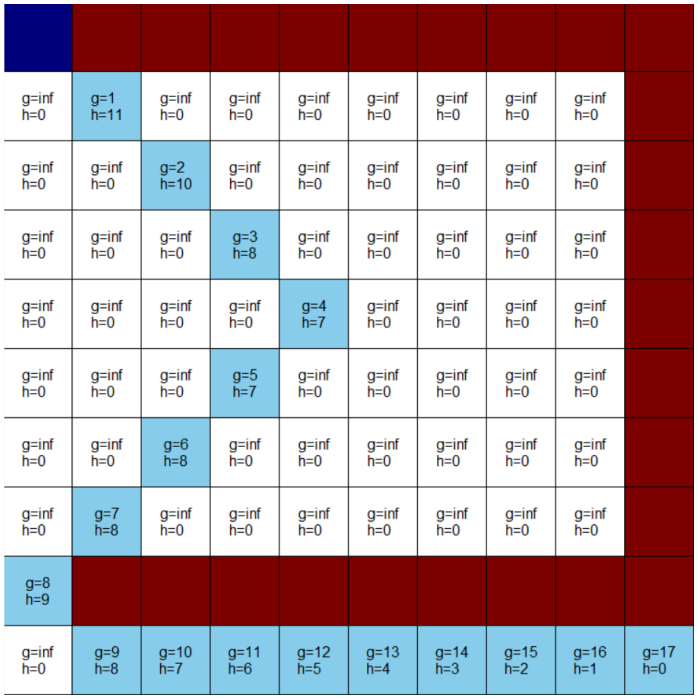
2. **Problem 2 (10 points)**

Repeat the above experiment but this time:

- Use the Euclidean Distance heuristic.
- The agent is allowed to make diagonal moves (i.e., NE, NW, SE, SW) in addition to the usual N, S, E, and W moves.
- The moves are made randomly and not in any specific order.

To make the Euclidean distance heuristic work I had to make changes to the Manhattan heuristic, making it match this formula $\sqrt{(x2 - x1)2 + (y2 - y1)2}$. After I did that I had to add the diagonal moves by editing the portion denoting where the agent can move. I then repeated the process from Problem 1 and made the second one Greedy. The A* one followed a very predictable pattern, going diagonally towards the goal until the h-value reached equilibrium and then it focused mostly on g and started going towards the edge, around the wall until it reached the goal. The Greedy algorithm started the same as the A* one but it then kept going towards the goal essentially until it reached the wall on the bottom and then followed the wall all the way to the goal.

A*



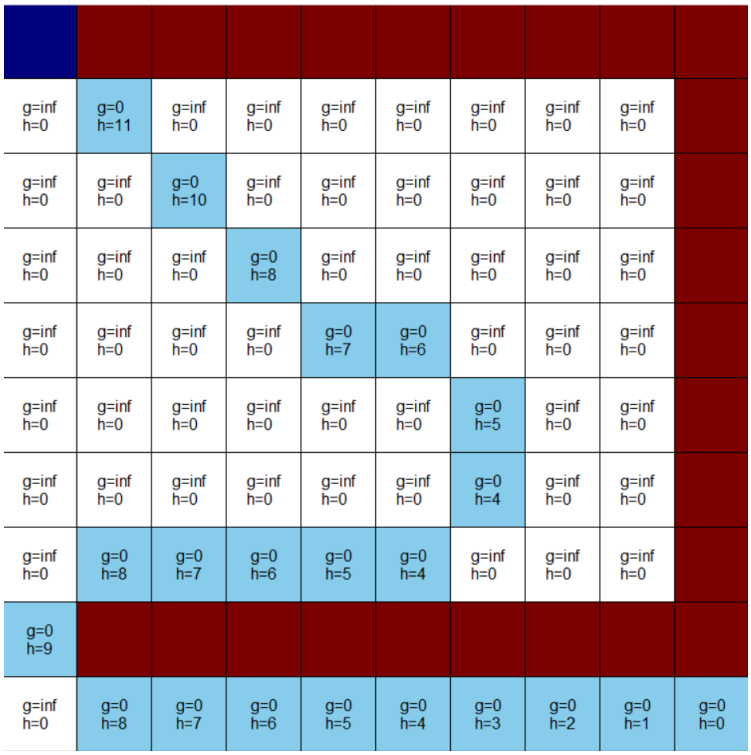

```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

```
#### The cost of moving to a new position is 1 unit
new_g = current_cell.g + 1
```

```
#### Agent goes E, W, N, S, NW, NE, SW, SE whenever possible
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0), (1, -1), (1, 1), (-1, -1),  (-1, 1)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)
```

```
############################################################
#### Euclidean distance
############################################################
def heuristic(self, pos):
    dist = (abs(pos[0] - self.goal_pos[0]) ** 2 + abs(pos[1] - self.goal_pos[1]) ** 2) ** 0.5
    return round(dist)
```

Greedy




```
### Update the evaluation function for the cell n: f(n) = h(n)
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.cells[new_pos[0]][new_pos[1]].h # g(n) = 0
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

```
#### The cost of moving to a new position is 1 unit
new_g = current_cell.g
```

```
#### Agent goes E, W, N, S, NW, NE, SW, SE whenever possible
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0), (1, -1), (1, 1), (-1, -1),  (-1, 1)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)
```

```
########################################################
#### Euclidean distance
########################################################
def heuristic(self, pos):
    dist = (abs(pos[0] - self.goal_pos[0])**2 + abs(pos[1] - self.goal_pos[1])**2)**0.5
    return round(dist)
```

3. The evaluation function in **AstarMaze** is defined as **f(n) = g(n) + h(n)**. A weighted version of the function can be defined as:

f(n) = α x g(n) + β x h(n) where

1. Explain how different values of α and β affect the A* algorithm's behavior. Tabulate your results:

α (g-weight)	β (h-weight)	Observed Behavior	Description
1	1	Standard A*	Balanced: finds shortest path optimally
0.5	1	More Greedy	Faster but less optimal: prioritizes goal proximity
1	0.5	More Cautious	Expands more nodes: slower but safer, avoids detours
0	1	Greedy Best-First	Ignores path cost: may reach goal quickly but with longer total path
1	2	Strong Goal Bias	Highly greedy: focuses on heuristic, often non-optimal paths
2	1	Dijkstra-like	Almost ignores heuristic: explores broadly but finds true shortest path

2. β can be considered the algorithm's bias towards states that are closer to goal. Run the algorithm for various values of the bias to determine what changes, if any, are observed in the optimum path. Include screenshots of the path for each specific value of β along with your explanation.

The following images represent the table I have above. The first image is what you get if you do α = 1 and β = 1 but also what you get if you do 1 and 0.5, and also 2 and 1. The second Image is what you get if you have 0.5 and 1, and 1 and 2. The last image is what you get when you do 0 and 1 (the Greedy Best-First)

g=1 h=17	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=8 h=10									
g=9 h=9	g=10 h=8	g=11 h=7	g=12 h=6	g=13 h=5	g=14 h=4	g=15 h=3	g=16 h=2	g=17 h=1	g=18 h=0

g=1 h=17	g=2 h=16	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=inf h=0	g=3 h=15	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=inf h=0	g=4 h=14	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=inf h=0	g=5 h=13	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=inf h=0	g=6 h=12	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=inf h=0	g=7 h=11	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=9 h=11	g=8 h=10	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	
g=10 h=10									
g=11 h=9	g=12 h=8	g=13 h=7	g=14 h=6	g=15 h=5	g=16 h=4	g=17 h=3	g=18 h=2	g=19 h=1	g=20 h=0

g=0 h=17	g=0 h=16	g=0 h=15	g=0 h=14	g=0 h=13	g=0 h=12	g=0 h=11	g=0 h=10	g=0 h=9	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=8	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=7	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=6	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=5	
g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=inf h=0	g=0 h=4	
g=0 h=11	g=0 h=10	g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	
g=0 h=10									
g=0 h=9	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=0 h=3	g=0 h=2	g=0 h=1	g=0 h=0