Zarak Khan
CSC 362 AI
10/15/25

Assignment 3 Report

Problem 1

Purpose: The goal is to compare the behaviors of A* and Greedy Best-First Search, and visually observe how each algorithm generates its optimum path.

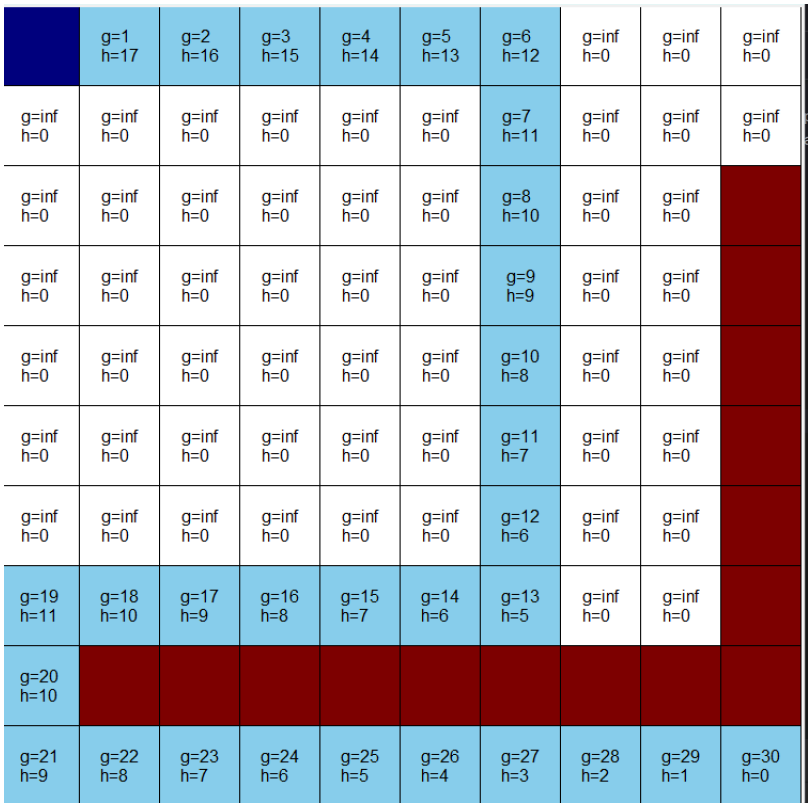
A* Search Manhattan Algorithm:

	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=inf h=0
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=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

```
### 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 A* algorithm successfully generated an optimum and efficient path to the goal. Since it considers both the cost from the start node (g), as well as the estimated cost to the goal (h), it guarantees the shortest path. It generates a path that travels straight down, to the right, completely avoiding the dead-end trap, and minimizing the total steps taken (g=18).

Greedy Best-First Search Manhattan Algorithm




```
# Update the evaluation function for Greedy Best-First Search: f(n) = h(n)
self.cells[new_pos[0]][new_pos[1]].f = self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

A clear visual difference can be observed for the Greedy Best-First Search algorithm. This algorithm only focuses on minimizing the heuristic distance (h), moving

aggressively towards the goal before hitting a dead-end and adjusting its path. The route taken is significantly longer (g=30), less efficient, and not optimal.

Problem 2

Purpose: The goal is to compare the behaviors of A* and Greedy Best-First Search using the Euclidean Distance Heuristic (instead of Manhattan distance), and visually observe how each algorithm generates its optimum path. This time, the agent is allowed to make diagonal moves (NE, NW, SE, SW) as well as the usual N, S, E, and W moves. The moves are made randomly using random.shuffle(directions).

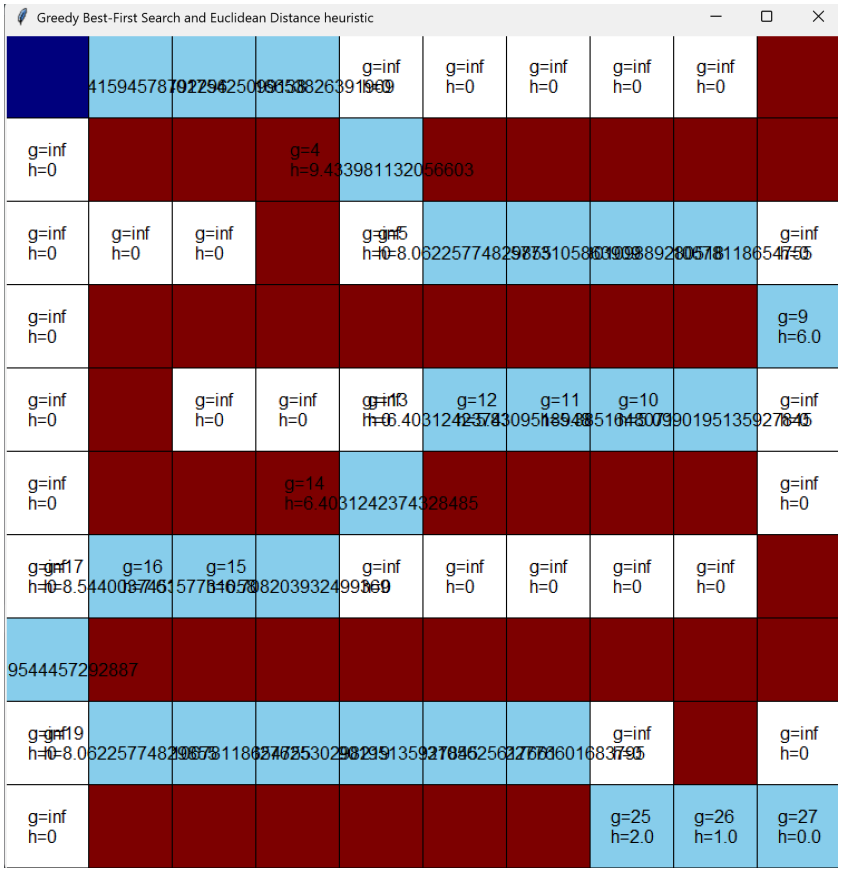
```
###########################################################################
#### Modified to use Euclidean distance: sqrt((x2 - x1)^2 + (y2 - y1)^2)
###########################################################################
def heuristic(self, pos):
    return math.sqrt((pos[0] - self.goal_pos[0]) ** 2 + (pos[1] - self.goal_pos[1]) ** 2)
```

```
#### Agent goes E, W, N, S, NE, NW, SE, SW randomly
directions = [(0, 1), (0, -1), (1, 0), (-1, 0),  # N, S, E, W
              (1, 1), (1, -1), (-1, 1), (-1, -1)]  # NE, NW, SE, SW (added diagonal movement)
random.shuffle(directions) # randomize direction, not in any specific order
```

A* Search Euclidean Algorithm:

The A* algorithm successfully generated an optimum and efficient path to the goal using the Euclidean Distance Heuristic. Since it considers both the cost from the start node (g), as well as the estimated cost to the goal (h), it guarantees the shortest path. It generates a path that travels down, and to the right, completely avoiding the much longer zig-zag path, and minimizing the total steps taken (g=16). Although it occasionally takes unnecessary diagonal steps, it remains largely optimal.

Greedy Best-First Search Euclidean Algorithm



The Greedy Best-First Search algorithm only focuses on minimizing the Euclidean distance heuristic (h) to generate a path. In this case, it aggressively moves towards the goal, but ends up taking a much longer zig-zag path (g=27). Since the agent has been misled by a path that may have seemed optimal based on the heuristic alone, it takes a much longer path than originally anticipated. The g value in this case is significantly higher compared to the A* algorithm.

Problem 3:

1.

α: This weight controls g(n), the cost from the start node to node n. A higher α value makes the algorithm prioritize paths based on the total cost travelled, causing it to take a cheaper, optimal path.

β: This weight controls h(n), the heuristic estimate of the cost to the goal node from node n. A higher β value makes the algorithm take paths that appear closer to the goal

(greedy), and they tend to ignore path costs in favor of a faster path. This bias towards nodes that appear closer to the goal leads to a less than optimal, costly path.

α	β	Observed Behavior
High	Low	Cheaper, optimal path as it prioritizes cost g(n)
Low	High	Bias towards paths/nodes that appear closer to goal. Paths taken can be costly and less than optimal.
Equal	Equal	Balances g(n) and h(n). Standard A* Search which is highly optimal and efficient.
0	High	g(n) ignored. Only focuses on h(n) to reach goal quickly (Greedy Best-First Search).
High	0	h(n) ignored. Only focuses on g(n), optimizing the path taken for the lowest possible cost.

2.

I have added beta to the parameters so I can test different values of β

```
class MazeGame:  1 usage
    def __init__(self, root, maze, beta=1.0): # beta parameter added
        self.root = root
        self.maze = maze
        self.beta = beta
```

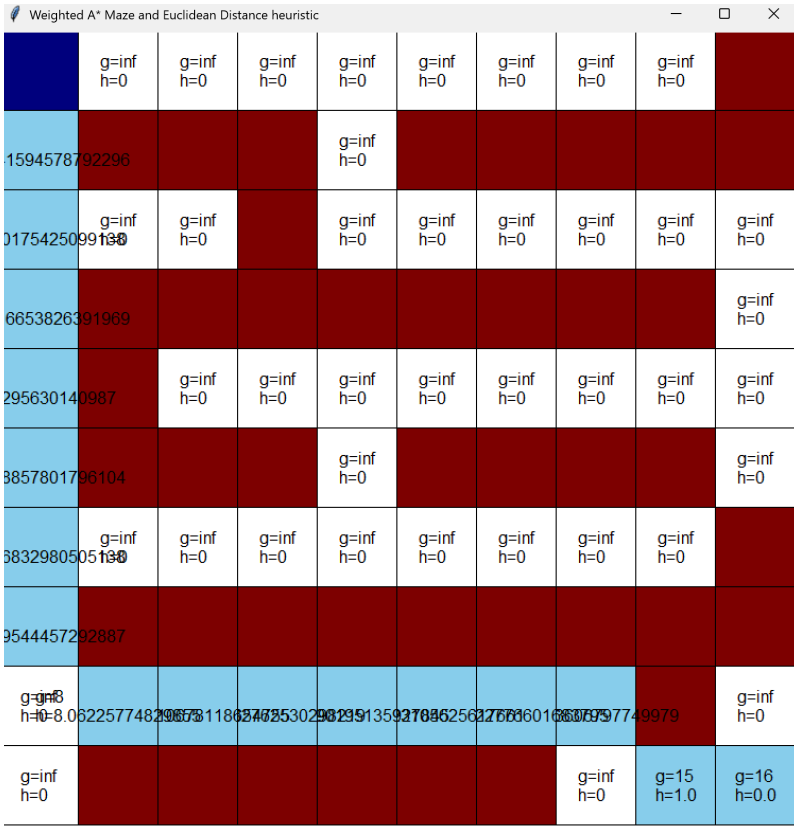
```
### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
### Added beta to weight the heuristic, to make algorithm more or less greedy
self.cells[new_pos[0]][new_pos[1]].f = new_g + self.beta * self.cells[new_pos[0]][new_pos[1]].h
self.cells[new_pos[0]][new_pos[1]].parent = current_cell
```

Test 1: β = 1.0

When β is set to 1.0, the agent takes a balanced and optimal path. The algorithm emphasizes both the path cost (g) and the heuristic estimate to the goal (h), avoiding any “shortcuts” which may lead to a longer path. Although the tendency for the algorithm to take unnecessary diagonal steps towards the goal can be observed, it does not impact the total path cost (it could in a different maze configuration).

Test 2: β = 0

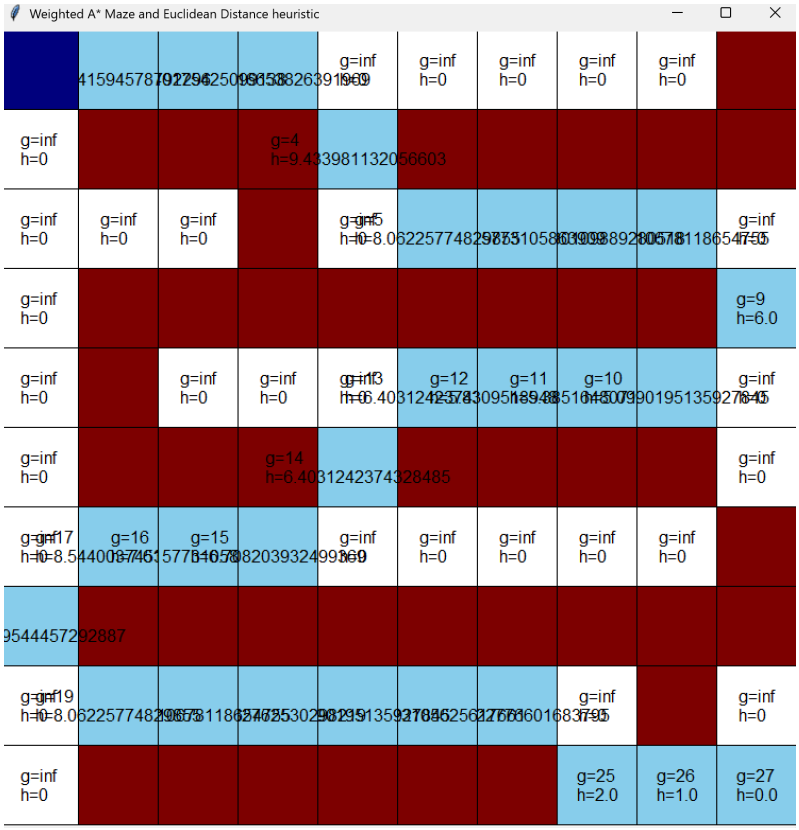
```
class MazeGame:  1 usage
    def __init__(self, root, maze, beta=0): # beta parameter added
        self.root = root
        self.maze = maze
        self.beta = beta
```



When β is set to 0, the algorithm completely ignores the heuristic (h), and only gives importance to the actual path cost (g). In this case, the agent finds the most optimal, cheapest path, avoiding any unnecessary diagonals or long paths.

Test 3: β = 10

```
class MazeGame:  1 usage
    def __init__(self, root, maze, beta=10): # beta parameter added
        self.root = root
        self.maze = maze
        self.beta = beta
```



When β is set to 10, the algorithm becomes very greedy. The estimated heuristic (h) is highly emphasized, and the actual path cost (g) is ignored. In this case, the agent aggressively moves towards the goal, but ends up taking a much longer zig-zag path (g=27). The agent takes an inefficient, non-optimal, and costly path as a result.

Conclusion:

Changing the β value affects the A* algorithms behavior. Lower β values generate cheaper, more optimal paths, while higher β values make the search greedier but less efficient and costly. It can be concluded that a balance between g(n) and h(n) (such as when β =1.0) generates the most optimal and efficient path.