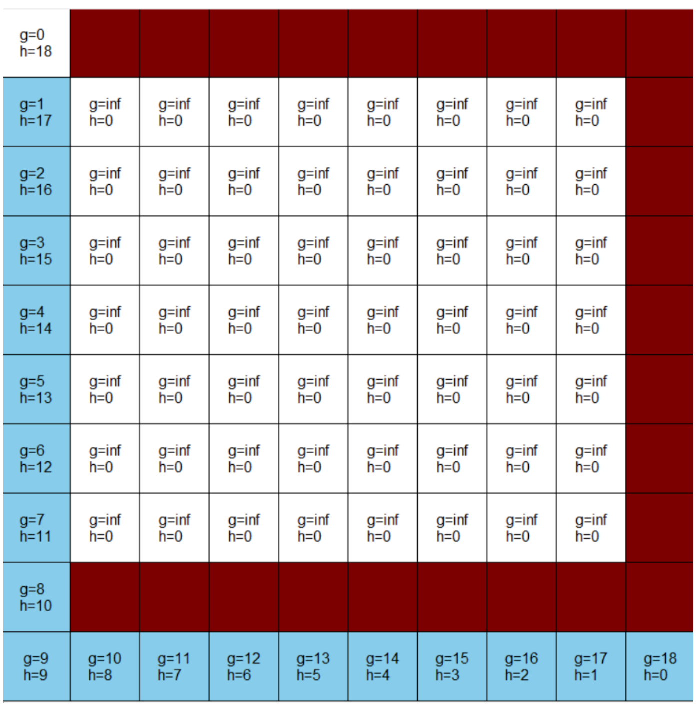
Meadow Monticello
Dr. Forouraghi
CSC 362
21 October 2025

Homework 3

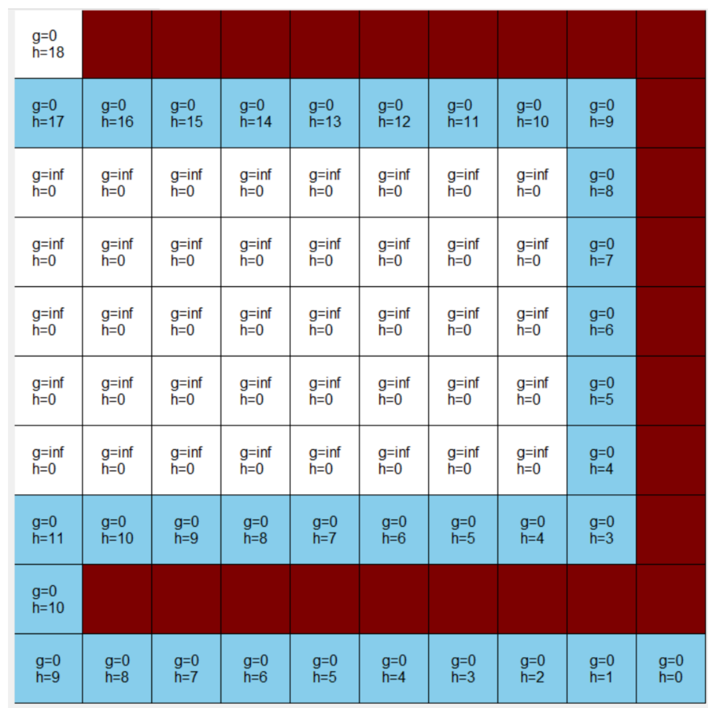
1. Modify AStarMaze to compare the behaviors of the Greedy Best-First and A* search algorithms. You need to modify the maze configuration so you can visually observe differences in the optimum paths generated by the two algorithms. Your report should include a side-by-side comparison of the two approaches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

A*




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

Greedy




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

`### 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`
	`### Update the evaluation function for the cell n: f(n) = h(n)`
	`self.cells[new_pos[0]][new_pos[1]].f = self.cells[new_pos[0]][new_pos[1]].h # g(n) = 0 because it's gre`
	`self.cells[new_pos[0]][new_pos[1]].parent = current_cell`

In the A* algorithm the path is optimal. G increases 1 with every step.	The greedy best first algorithm's path is not optimal. In order to make these changes so it would produce the path above, I removed the +1 from the code above. I also noted that g(n) would then equal 0 in this case. Therefore, f(n) = h(n).

Also I changed the maze from the initial code to match the configuration in the problem description. See below:

```
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]
]
```

2. 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.

A*

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

The A* algorithm took a predictable route. A* looks at both g and h. When h plateaued, the algorithm focused solely on g.

Greedy

g=0 h=13									
g=inf h=0	g=0 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=inf h=0	g=0 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=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=inf h=0	g=0 h=7	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=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=inf h=0	g=inf h=0	
g=inf h=0	g=0 h=8	g=0 h=7	g=0 h=6	g=0 h=5	g=0 h=4	g=inf h=0	g=inf h=0	g=inf h=0	
g=0 h=9									
g=inf h=0	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

The greedy best first algorithm only looks at h. Because of this, the path is unorthodox and not efficient.

```
#### Agent goes E, W, N, S, NE, NW, SE, SW 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)

    if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][new_pos[1]].is_wall:

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

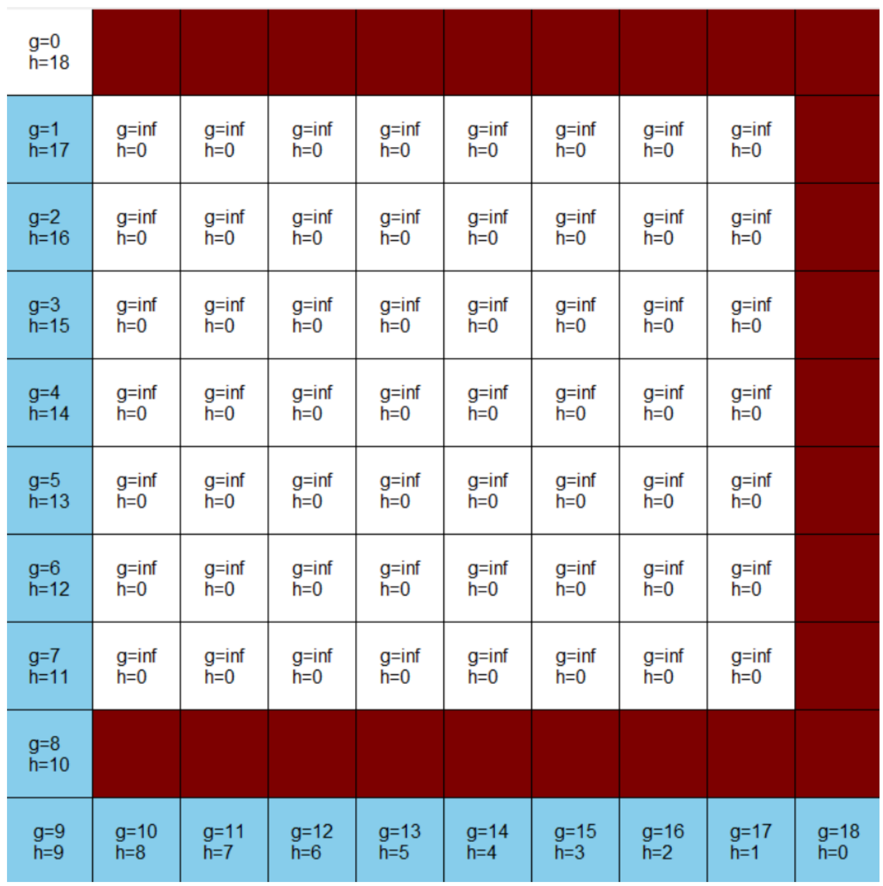
Both the A* and greedy best first search have more coordinates in problem 2 then problem 1 to account for the additional directions (diagonal moves).

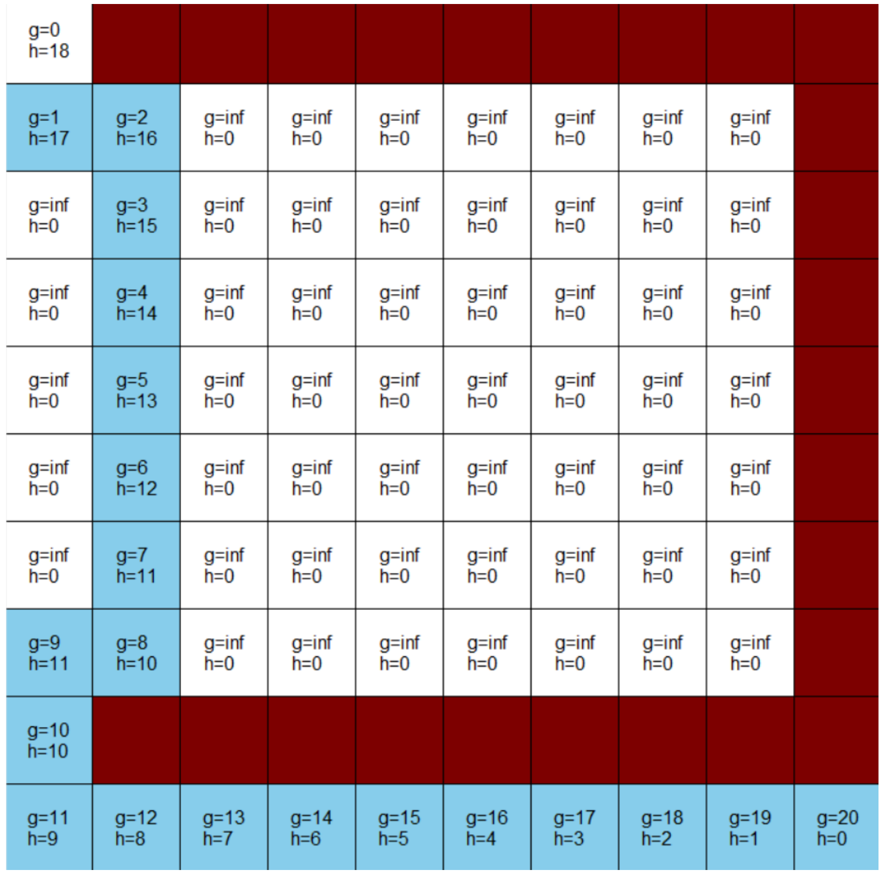
This is the change made in the code for greedy best first algorithm. Note how there is no +1 after the current_cell.g.

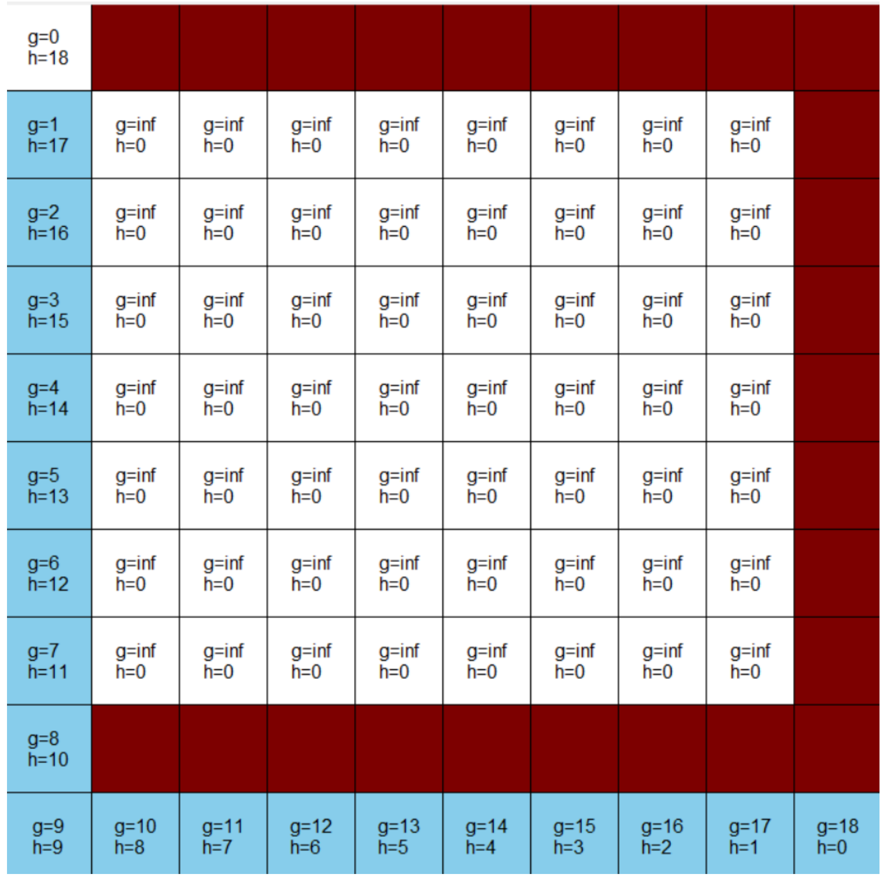
```
##############################################################
#### Euclidean distance
##############################################################
def heuristic(self, pos):
    dist = (abs(pos[0] - self.goal_pos[0]) ** 2 + abs(pos[1] - self.goal_pos[1]) ** 2) ** 0.5
    return round(dist)
```

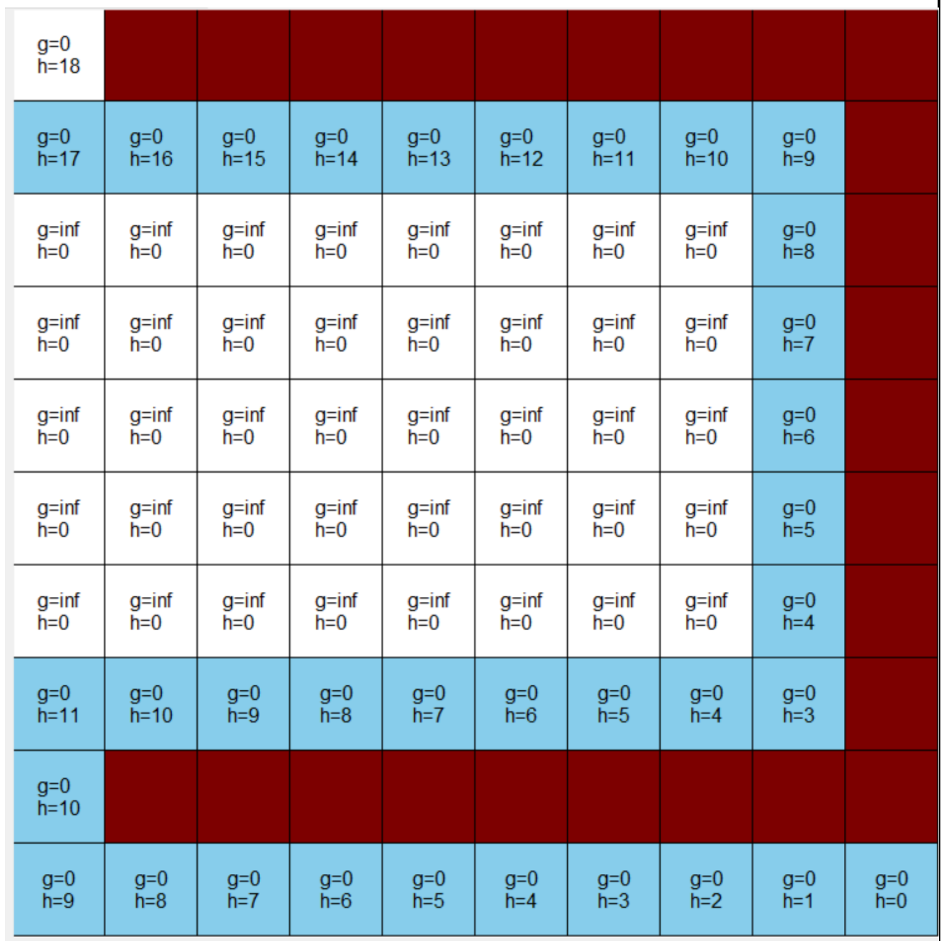
I had to change the distance formula to match the euclidean distance formula. The change is shown above.

3. Part 1

α	β	Observed Behavior	Interpretation	Output
1	1	A*	Optimal, shortest path	

.5	1	Leaning greedy	Focuses on proximity to goal	

1	.5	More cautious	Slower solution, but safe and straightforward route	

0	1	Greedy best first	Long total path and squares visited	


```
##############################################################
#### Manhattan distance
##############################################################
def heuristic(self, pos):
    return (abs(pos[0] - self.goal_pos[0]) + abs(pos[1] - self.goal_pos[1]))
```

I reverted back to Manhattan distance.

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

    if 0 <= new_pos[0] < self.rows and 0 <= new_pos[1] < self.cols and not self.cells[new_pos[0]][new_pos[1]].is_wall:

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

I am using the 4 cardinal directions again (no diagonals). I also reincluded +1 in the cost.

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

#### Add the new cell to the priority queue
open_set.put((self.cells[new_pos[0]][new_pos[1]].f, new_pos))
```

The screenshots of the mazes above were obtained from changing the alpha and beta in this code accordingly.