##
# Author: Michael Wood
# Purpose: HW4 Technical Report
# Class: AI, CSC 362, Spring 2024
##

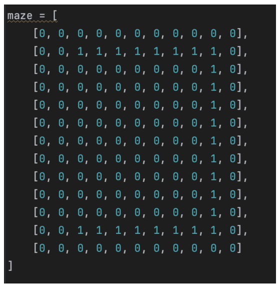
##
# PROBLEM 1
##

Description:

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.

Experiment:

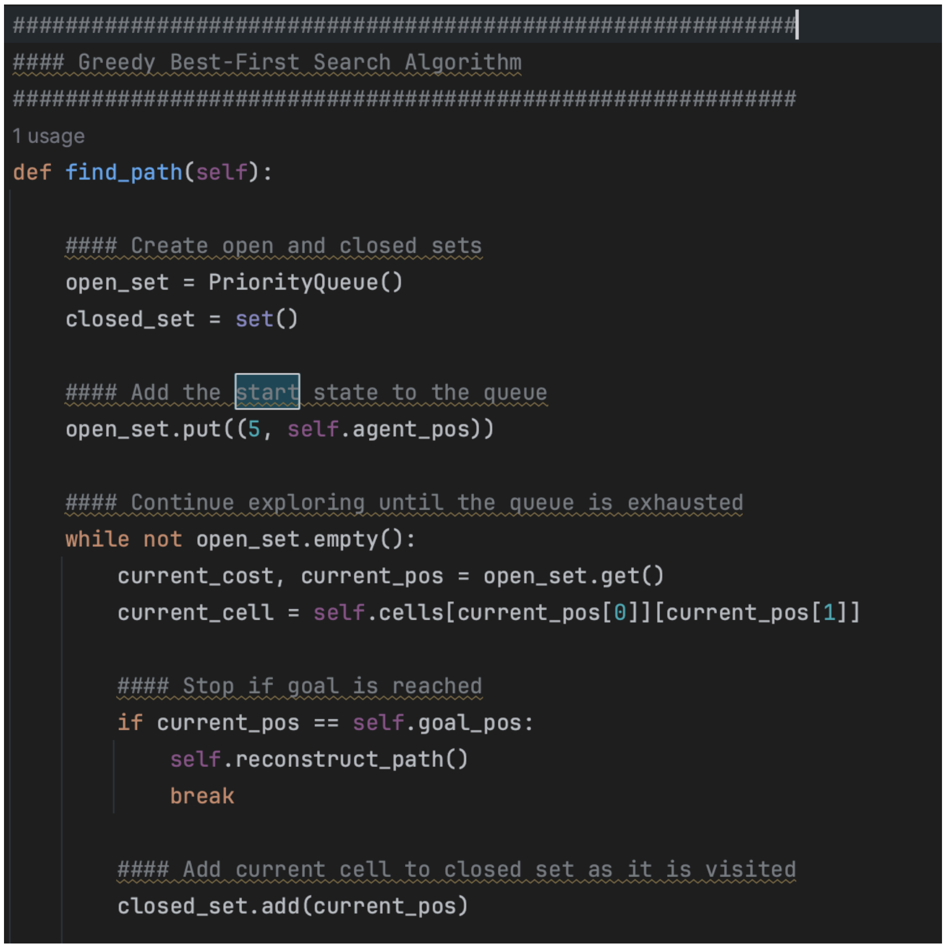
- I modified the AStarMaze algorithm provided in canvas. Much of the code regarding the GUI and the actual algorithm are taken from those files. The only thing I changed for the A* algorithm for each of these problems was the size of the maze and the formation of the walls and empty blocks as such (0=empty, 1=wall):



```
maze = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
```

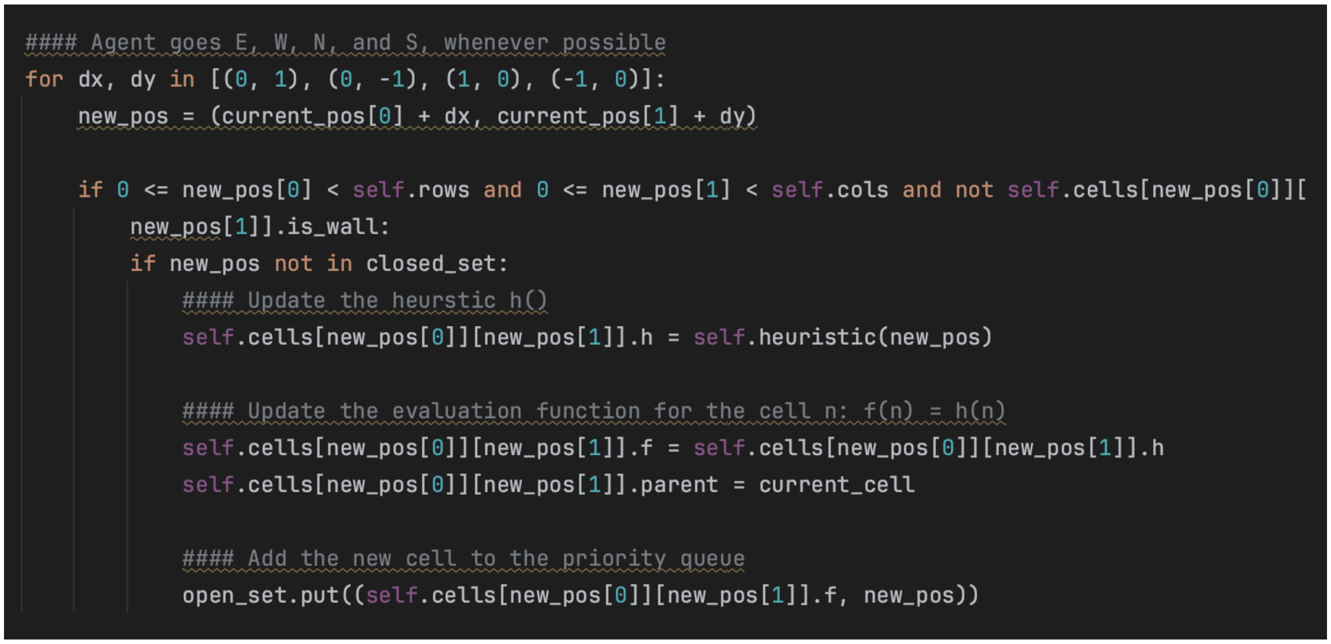
- For the Greedy Best-First Search algorithm, the only thing I changed was the actual algorithm to find the path in the maze (shown below on page 2):

1.



```
############################################################
#### Greedy Best-First Search Algorithm
############################################################
1 usage
def find_path(self):

    #### Create open and closed sets
    open_set = PriorityQueue()
    closed_set = set()

    #### Add the start state to the queue
    open_set.put((5, self.agent_pos))

    #### Continue exploring until the queue is exhausted
    while not open_set.empty():
        current_cost, current_pos = open_set.get()
        current_cell = self.cells[current_pos[0]][current_pos[1]]

        #### Stop if goal is reached
        if current_pos == self.goal_pos:
            self.reconstruct_path()
            break

        #### Add current cell to closed set as it is visited
        closed_set.add(current_pos)
```

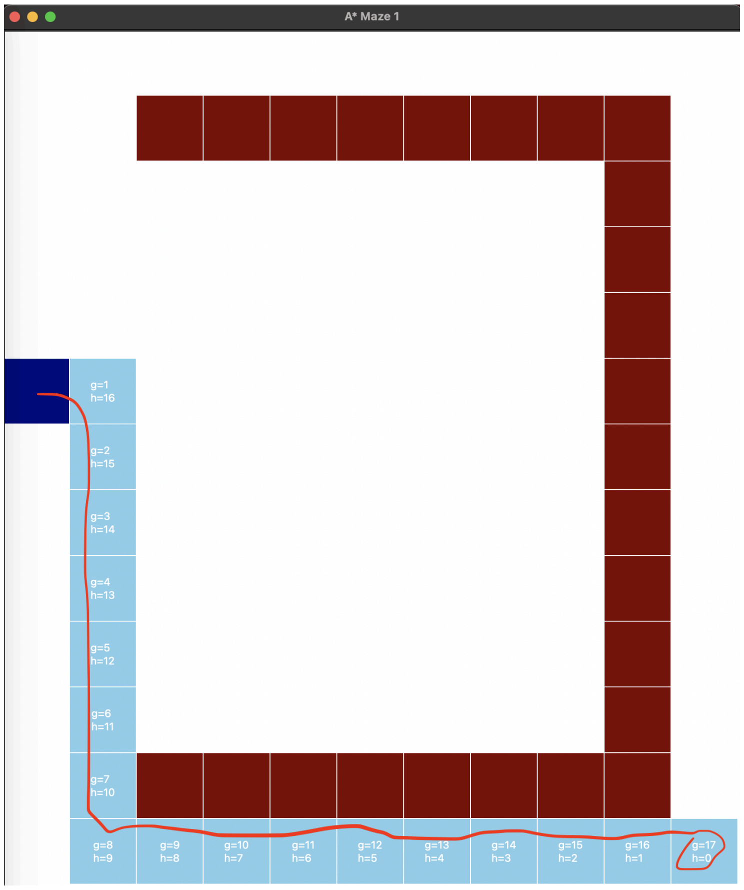
2.



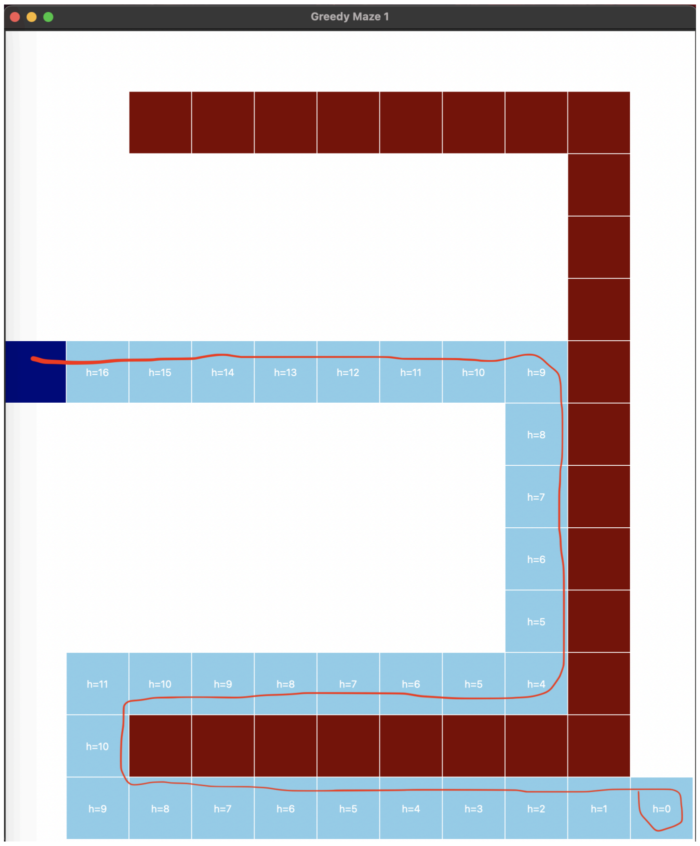
```
#### Agent goes E, W, N, and S, whenever possible
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
    new_pos = (current_pos[0] + dx, current_pos[1] + dy)

    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:
        if new_pos not in closed_set:
            #### Update the heurstic h()
            self.cells[new_pos[0]][new_pos[1]].h = self.heuristic(new_pos)

            #### 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
            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))
```

<u>Results</u> (red line was drawn separately for clarification of the path):

- A* (result of Problem1_AStar.py):



- GBFS (result of Problem1_Greedy.py):



Conclusion:

- The A* algorithm displays the more accurate and optimal result here. It uses the function f(n) = g(n) + h(n) using both the path cost and the heuristic cost of each state. It finds the value of f(n) of all blocks around it and takes the minimum cost path if the block has not been visited yet. The GBSF algorithm uses a similar method, but it only takes in the heuristic cost instead: f(n) = h(n). This calculation does not take into account the walls in the maze and only goes for the minimum heuristic cost from the current block to the goal state. From these results, the A* algorithm is more efficient than the GBFS algorithm.

###

# PROBLEM 2

###

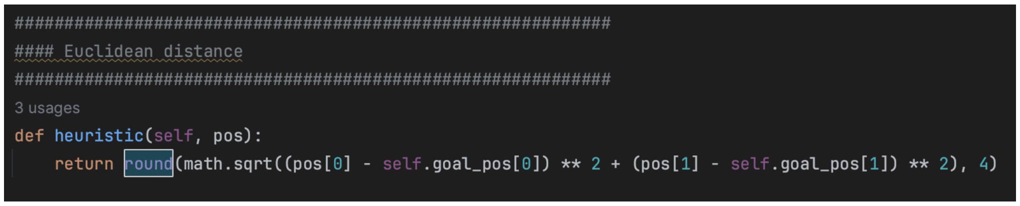
Description:

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.

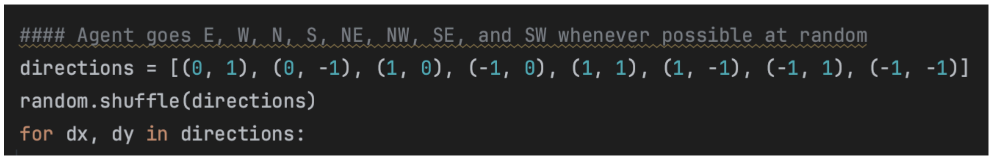
Experiment:

- I modified each of the algorithms to include the new requirements.
- I used the Euclidean Distance heuristic:
  - Changed h(n) calculation (for both A* and GBFS):



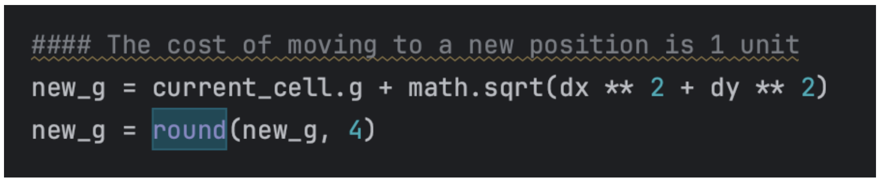
```
############################################################
#### Euclidean distance
############################################################
3 usages
def heuristic(self, pos):
    return round(math.sqrt((pos[0] - self.goal_pos[0]) ** 2 + (pos[1] - self.goal_pos[1]) ** 2), 4)
```

  -
- I allowed the agent to move in any direction, including diagonal directions
- I randomized the order of those directions as well:
  - Updated the directions and randomized them using the random.shuffle() function (in both A* and GBSF Algorithms):



```
#### Agent goes E, W, N, S, NE, NW, SE, and SW whenever possible at random
directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]
random.shuffle(directions)
for dx, dy in directions:
```

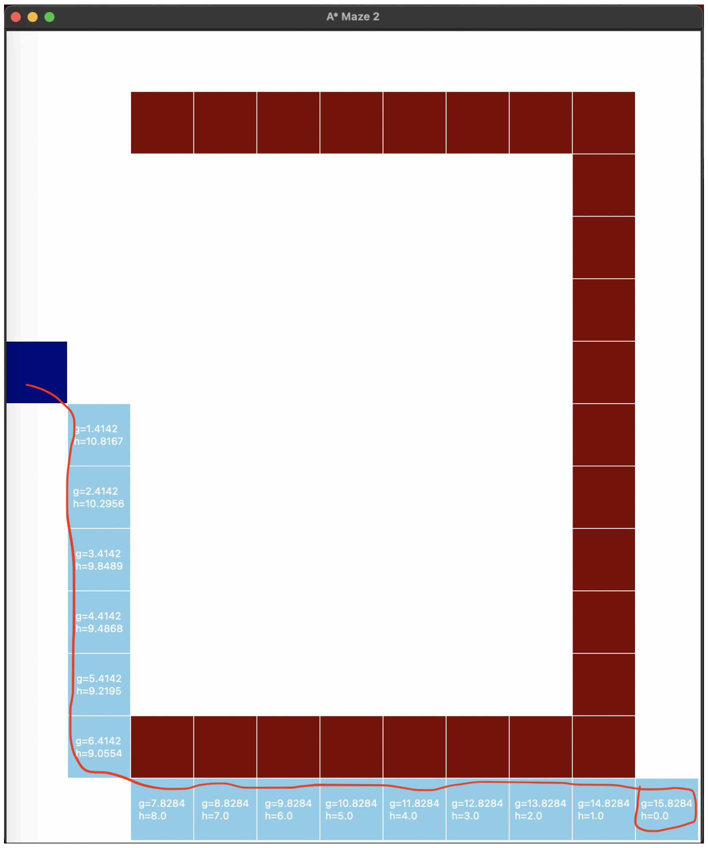
  - Changed g(n) calculation (only in A*):



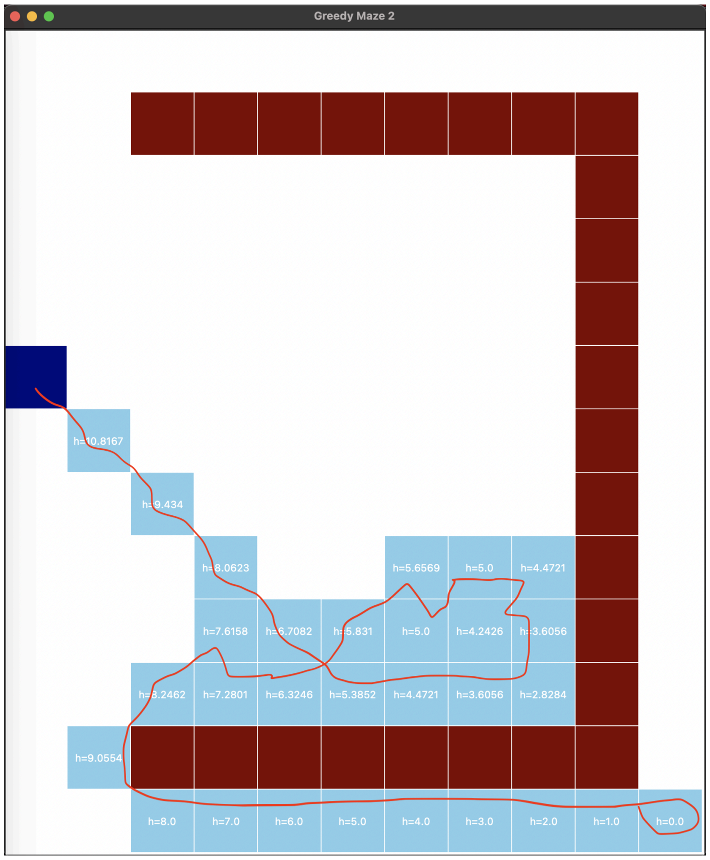
```
#### The cost of moving to a new position is 1 unit
new_g = current_cell.g + math.sqrt(dx ** 2 + dy ** 2)
new_g = round(new_g, 4)
```

<u>Results</u> (red line was drawn separately for clarification of the path):

- A* (result of Problem2_AStar.py):



- Greedy Best-First Search (result of Problem2_Greedy.py):



Conclusion:

- Once again, the A* algorithm seems more accurate and optimal than the GBFS algorithm here. The GBSF algorithm takes an even longer path using the Euclidean distance heuristic. Since it only takes the shortest path to the goal state and ignores walls, the GBSF algorithm almost has to circle back to find as it hits the wall. The A* algorithm reaches the goal state much quicker with the path cost in mind along with the heuristic cost.

##

# PROBLEM 3

##

Description:

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 . g(n) + β . h(n) where α, ≥0

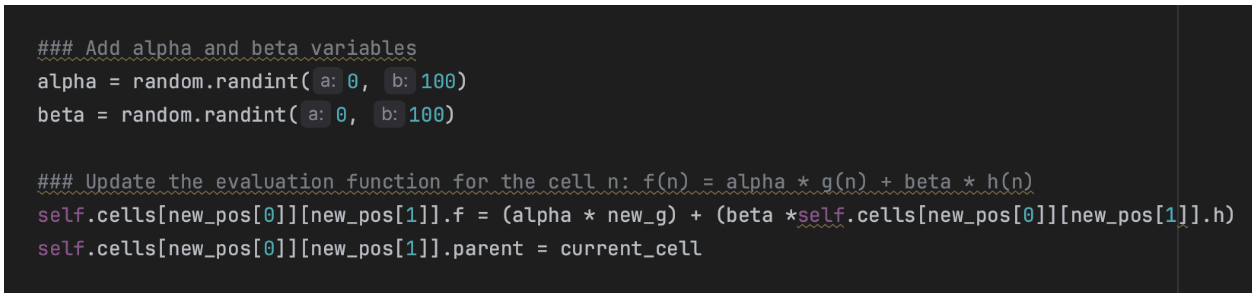
1. Explain how different values of α and β affect the A* algorithm's behavior. Tabulate your results:

α	β	**Observed Behavior**
…	…	…

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.

Experiment:

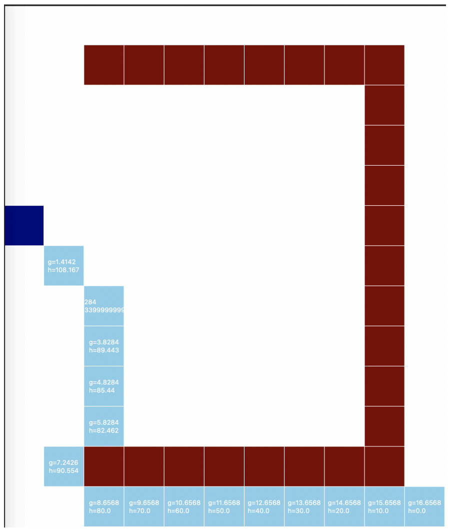
- I simply modified the equation in the A* algorithm used in Problem 2 by adding the alpha and beta values (the random number generators are placeholders for specific values to be placed in later):

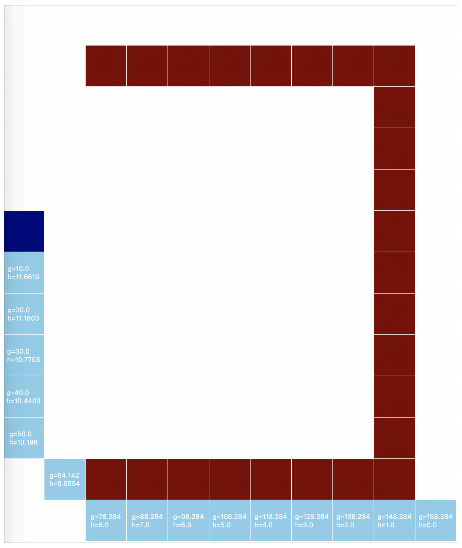


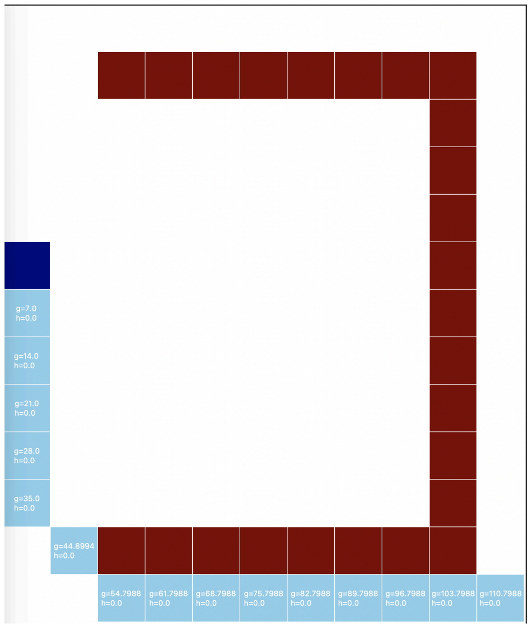
```
### Add alpha and beta variables
alpha = random.randint( a: 0,  b: 100)
beta = random.randint( a: 0,  b: 100)

### Update the evaluation function for the cell n: f(n) = alpha * g(n) + beta * h(n)
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
```

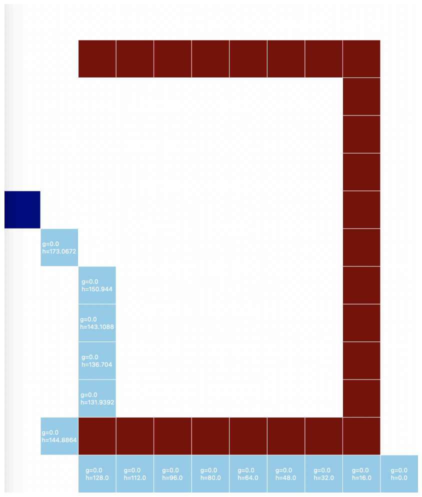
<u>Results</u>:

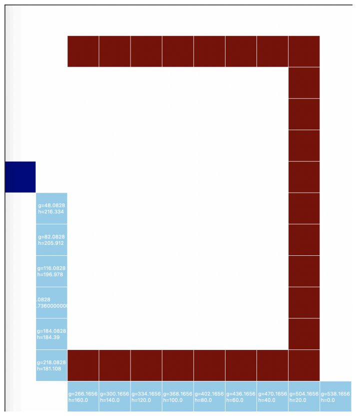
α	β	**Observed Behavior**

1	10	 g=1.4142
		h=108.167
		284
		3399999999
		g=3.8284
		h=89.443
		g=4.8284
		h=85.44
		g=5.8284
		h=82.462
		g=7.2426
		h=90.554
		g=8.6568
		h=80.0
		g=9.6568
		h=70.0
		g=10.6568
		h=60.0
		g=11.6568
		h=50.0
		g=12.6568
		h=40.0
		g=13.6568
		h=30.0
		g=14.6568
		h=20.0
		g=15.6568
		h=10.0
		g=16.6568
		h=0.0
		● The path did change slightly here with an increased bias of 10
		● One more diagonal move to the right and back than the original path

10	1	 g=10.0
		h=11.6619
		g=20.0
		h=11.1803
		g=30.0
		h=10.7703
		g=40.0
		h=10.4403
		g=50.0
		h=10.198
		g=64.142
		h=9.0554
		g=78.284
		h=8.0
		g=88.284
		h=7.0
		g=98.284
		h=6.0
		g=108.284
		h=5.0
		g=118.284
		h=4.0
		g=128.284
		h=3.0
		g=138.284
		h=2.0
		g=148.284
		h=1.0
		g=158.284
		h=0.0
		● With an alpha score of 10, this path does not move diagonally to the left until the second to last row of the maze

7	0	 g=7.0 h=0.0
		g=14.0 h=0.0
		g=21.0 h=0.0
		g=28.0 h=0.0
		g=35.0 h=0.0
		g=44.8994 h=0.0
		g=54.7988 h=0.0
		g=61.7988 h=0.0
		g=68.7988 h=0.0
		g=75.7988 h=0.0
		g=82.7988 h=0.0
		g=89.7988 h=0.0
		g=96.7988 h=0.0
		g=103.7988 h=0.0
		g=110.7988 h=0.0
		● A bias of 0 also displays this path of not moving diagonally until the second to last row

0	16	 g=0.0 h=173.0672
		g=0.0 h=150.944
		g=0.0 h=143.1088
		g=0.0 h=136.704
		g=0.0 h=131.9392
		g=0.0 h=144.8864
		g=0.0 h=128.0
		g=0.0 h=112.0
		g=0.0 h=96.0
		g=0.0 h=80.0
		g=0.0 h=64.0
		g=0.0 h=48.0
		g=0.0 h=32.0
		g=0.0 h=16.0
		g=0.0 h=0.0
		● An alpha of 0 and a bias of 16 displays this path of moving to the right diagonally and back one more time

34	20	 g=48.0828 h=216.334
		g=82.0828 h=205.912
		g=116.0828 h=196.978
		.0828 .7360000000
		g=184.0828 h=184.39
		g=218.0828 h=181.108
		g=266.1656 h=160.0
		g=300.1656 h=140.0
		g=334.1656 h=120.0
		g=368.1656 h=100.0
		g=402.1656 h=80.0
		g=436.1656 h=60.0
		g=470.1656 h=40.0
		g=504.1656 h=20.0
		g=538.1656 h=0.0
		● With these values, it provided the same path as the original one

Conclusion:

- It seems that the alpha and beta values do not affect the A* algorithm that much in this scenario. The algorithm is built to find the minimum cost of f(n) = g(n) + h(n) at each block, and adding the alpha and beta values would not seem to change that as long as they remain greater than or equal to 0.

###
# OVERALL CONCLUSION
###

- The A* algorithm is much more efficient at reaching the goal state than the GBFS algorithm in these cases. The Greedy algorithm only takes in the heuristic cost, so it does not guarantee that the goal state will even be found in a particular scenario. With the path cost and the heuristic cost, the A* algorithm can be more guaranteed to find the correct solution.
- It is interesting that allowing the algorithms more freedom such as moving diagonally helped both of them find the goal state in less steps even if the Greedy path may look a bit more confusing.
- Increasing the bias did not seem to affect the A* algorithm too much, but it is interesting to note that it did have some effect on the final path chosen. I imagine in a much larger space, it would affect the result of the algorithm a lot more.