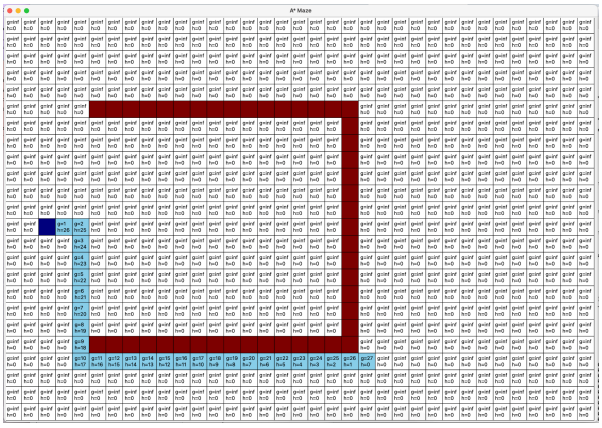
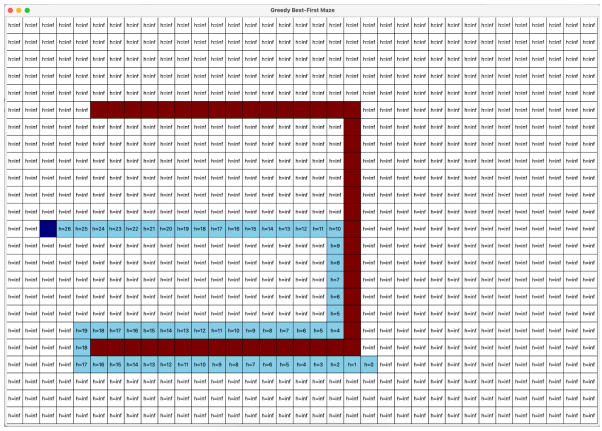
Sarah Groark
Artificial Intelligence - Assignment 4

Question 1

A*	Greedy Best-First
	
The A* algorithm utilizes the evaluation function (f) that takes both the actual cost (g) and heuristics (h) into consideration. In this case, one move costs 1 unit that is added to the g value of the cell, while the heuristic distance is calculated using the Manhattan distance. The algorithm then takes into account the summation of these values, which is then assigned to the value f (the evaluation function), and chooses the neighboring cells with the most reduced f value.	The Greedy Best-First algorithm, in contrast, only makes moves based on heuristic values of neighboring cells. The evaluation function for this search approach omits the actual cost (g) and only includes heuristic values (h). In this sense, the agent makes moves to neighboring cells with the lowest heuristic value. Resultantly, the agent moves closer towards the goal state without accruing the actual costs of the move, which explains why the agent above moves closer towards the goal state (heuristically) but has to redirect once encountering an obstacle.

```
if new_g < self.cells[new_pos[0]][new_pos[1]].g:
    ### Update the path cost g()
    self.cells[new_pos[0]][new_pos[1]].g = new_g

    ### 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) = 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

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

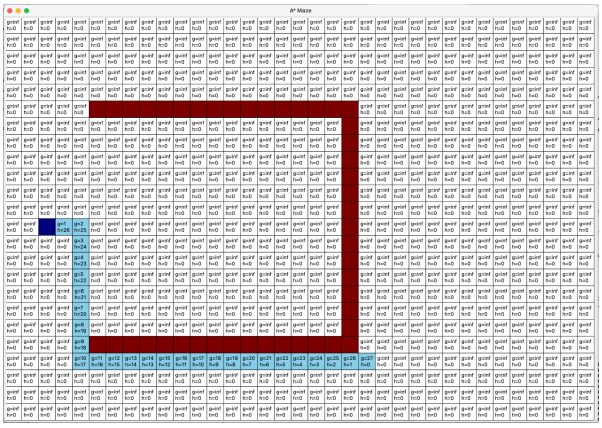
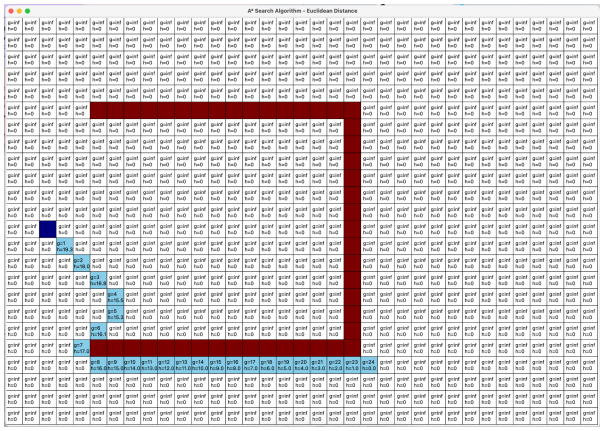
In regular A*, the evaluation function is set to equal the sum of the actual cost and heuristic of the cell.

```
if new_h < self.cells[new_pos[0]][new_pos[1]].h:
    ### Update the path cost g()
    #self.cells[new_pos[0]][new_pos[1]].g = new_g

    ### 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) = g(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))
```

In Greedy Best-First, the evaluation function is set to equal just the value of the cell's calculated heuristic.

Question 2

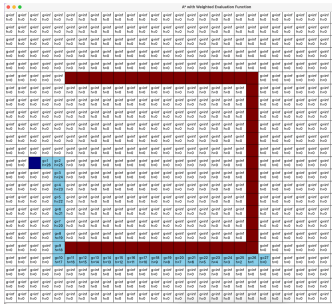
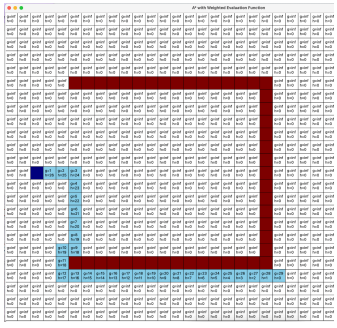
Changing the heuristic definition from Manhattan distance to Euclidean distance allows for diagonal moves in addition to the typical N, S, E, W moves that were included previously. The move cost remained constant at 1 unit per move.

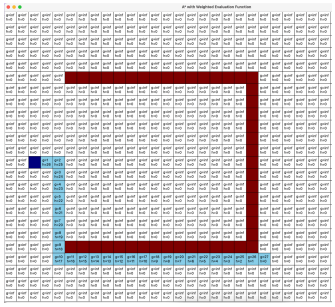
A*- Manhattan Distance	A* - Euclidean Distance
	
Manhattan Distance functions efficiently for north, south, east, and west agent moves. It calculates the sum of the absolute difference of two points – the current cell being explored and the goal state – to define the heuristic.	Use of the Euclidean Distance allows for diagonal moves in addition to the standard N, S, E, W agent moves. It is used to calculate the shortest distance between two points (the current cell and the goal state).
Equation: $d = \|x_1 - y_1\| + \|x_2 - y_2\|$	Equation: $d = [(x_1 - y_1)^2 + (x_2 - y_2)^2]^{1/2}$
`` ## #### Manhattan distance ## def heuristic(self, pos): return (abs(pos[0] - self.goal_pos[0]) + abs(pos[1] - self.goal_pos[1])) ```	``` ## #### Euclidean distance ## def heuristic(self, pos): return round(math.sqrt((pos[0] - self.goal_pos[0])**2 + (pos[1] - self.goal_pos[1])**2),1) ```
`` #### 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) ```	``` diagonal_directions = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1),(1, -1),(-1, 1),(-1, -1)] #### Agent goes E, W, N, and S, whenever possible for dx, dy in diagonal_directions: new_pos = (current_pos[0] + dx, current_pos[1] + dy) ```

This implementation allows for just north, south, east, and west moves (as it just initiates the changing of one coordinate, rather than two).	The implementation of the above 8 coordinates allows for both diagonal moves and the standard north/south/east/west agent moves (since both coordinates are changed).

Question 3

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

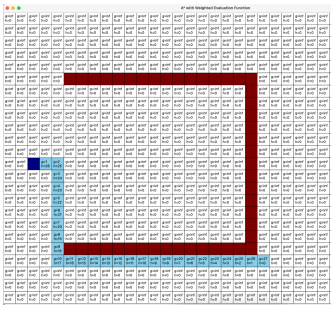
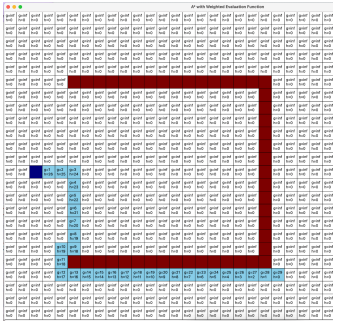
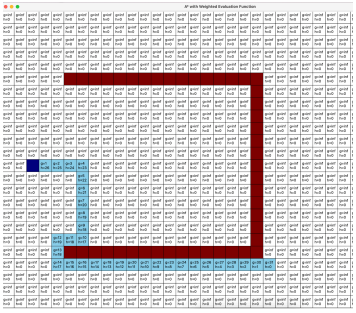
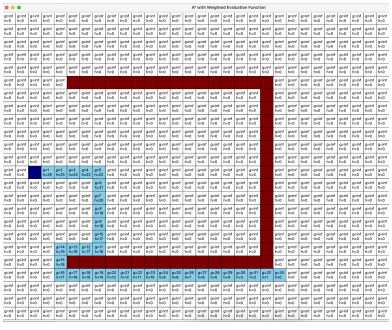
α	**β**	**Observed Behavior**	**Output**
1	1	No change from regular A*.	
1	2	Favors heuristics slightly more over actual cost values.	

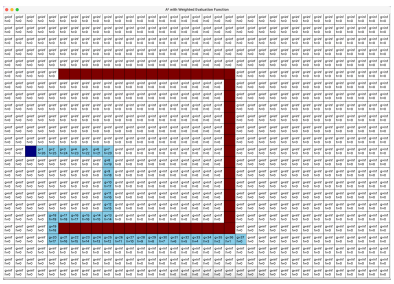
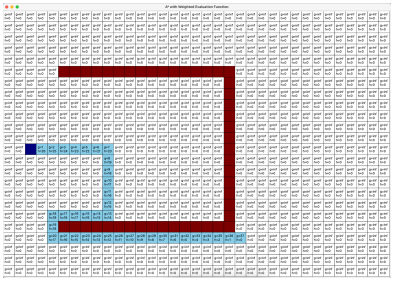
2	1	No change from 1,1.	

2. **β** can be considered the algorithm's bias towards states that are closer to the 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.

Code implementation:

```
##############################################################
#### A* Algorithm
##############################################################
def find_path(self, alpha = 1, beta = 2):
    open_set = PriorityQueue()


      ..............

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
```

β	Observation	Output
1	Behaves the same as regular A*.	
2	Slightly favors heuristics in the evaluation function - will make moves closer towards goal state.	
3	Favors states closer to goal state (more so than lower values of β)	
4	Favors heuristics and states closer to the goal slightly more than β = 3.	

15	Favors moves towards the goal state slightly more than β = 4.	
150	Behaves the same as β = 15.	

For increased β values, the algorithm begins to behave more similarly to a greedy best-first search algorithm. However, as β increases, the cost of finding the goal also increases, although it does find the solution relatively faster.