Sean Gor
10/15/25
Assignment 3 Report
Purpose: To document results from the experiments of testing paths generated by Greedy and A* algorithms, along with changing values of beta and/or alpha.

Part 1:

Code Used (Manhattan Distance heuristic):

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

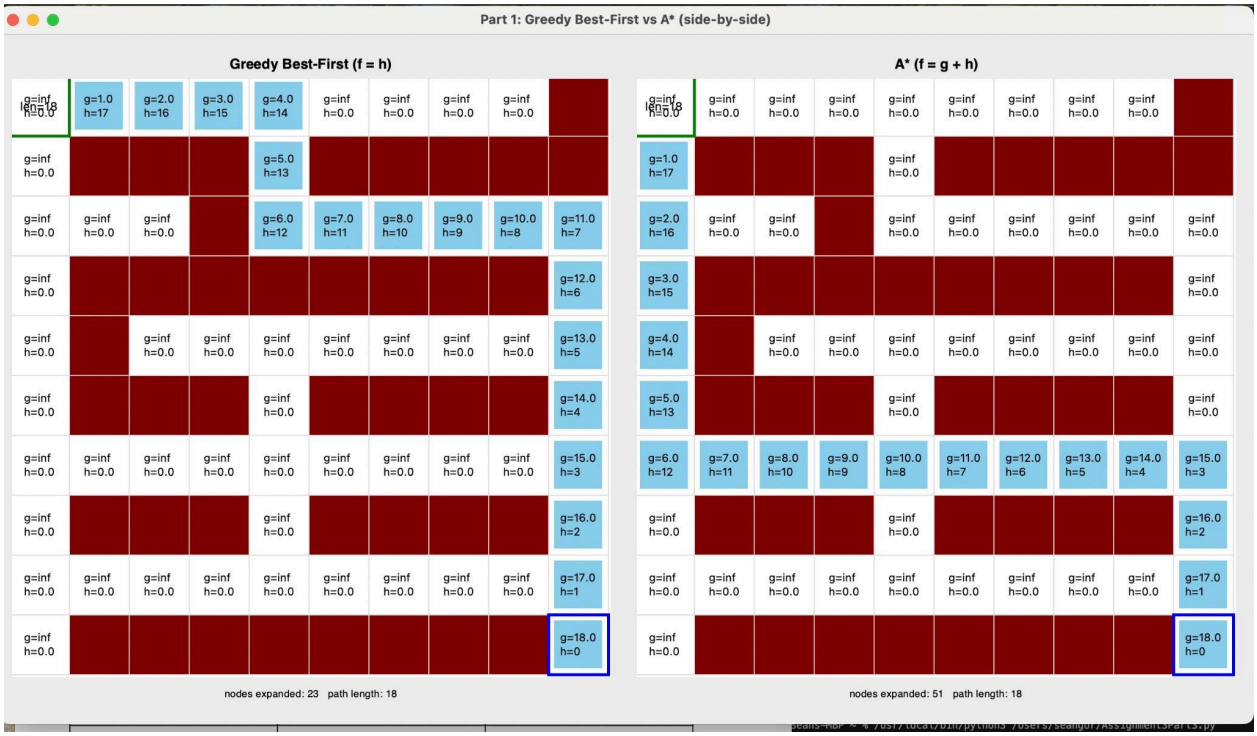
```python
        for dx, dy in ((0,1),(0,-1),(1,0),(-1,0)):
            nx, ny = u.x + dx, u.y + dy
            if not (0 <= nx < self.rows and 0 <= ny < self.cols): continue
            v = self.cells[nx][ny]
            if v.is_wall: continue

            tentative_g = u.g + 1.0
            h = self.heuristic((nx, ny))

            if self.strategy == "ASTAR":
                if tentative_g < v.g:                    # relax on g
                    v.g = tentative_g
                    v.h = h
                    v.f = v.g + v.h
                    v.parent = u
                    if (nx, ny) not in in_open:
                        pq.put((v.f, v)); in_open.add((nx, ny))
            else:  # GREEDY
                improved_g = False
                if tentative_g < v.g:
                    v.g = tentative_g
                    improved_g = True
                new_f = h                                # f = h
                if new_f < v.f or improved_g:
                    v.h = h
                    v.f = new_f
                    v.parent = u
                    if (nx, ny) not in in_open:
                        pq.put((v.f, v)); in_open.add((nx, ny))

    return expanded, 0  # no path
```

Screenshot of output chart:



Both algorithms get to the path in the same number of steps (since the path length is the same for both); however, the paths are different. In addition, the A* algorithm opens more nodes than the greedy algorithm does (according to the screenshot above). This may be because A* checks all possible paths to the goal, in order to figure out the lowest overall value of the evaluation function (both f and g). So it will consider all the nodes and their path costs so far, not just the current one. On the flip side, the Greedy Best First search algorithm (shown on the left) expands less nodes. This could be due to the fact that it minimizes the distance to the goal (h(n)), without regard to the cost (g(n)). Therefore, if it sees a short-term low cost, it will automatically go to that node without considering others (in a cumulative manner). This satisfies the overall behavior of Greedy Best First Search.

Part 2:

Code Used (shows Euclidean distance heuristic)

```
64    def heuristic(self, pos):
65        # Using Euclidean Distance for this part
66        return math.sqrt(math.pow(self.goal_pos[0] - pos[0], 2) + math.pow(self.goal_pos[1] - pos[1], 2))
```

```
116             #include diagonal directions for this part since the Euclidean Distance is being used
117             for dx, dy in ((0,1),(0,-1),(1,0),(-1,0), (1, -1), (-1, 1), (1, 1), (-1, -1)):
118                 nx, ny = u.x + dx, u.y + dy
119                 if not (0 <= nx < self.rows and 0 <= ny < self.cols): continue
120                 v = self.cells[nx][ny]
121                 if v.is_wall: continue
122
123                 tentative_g = u.g + 1.0
124                 h = self.heuristic((nx, ny))
125
126                 if self.strategy == "ASTAR":
127                     if tentative_g < v.g:                    # relax on g
128                         v.g = tentative_g
129                         v.h = h
130                         v.f = v.g + v.h
131                         v.parent = u
132                         if (nx, ny) not in in_open:
133                             pq.put((v.f, v)); in_open.add((nx, ny))
134                 else:  # GREEDY
135                     improved_g = False
136                     if tentative_g < v.g:
137                         v.g = tentative_g
138                         improved_g = True
139                     new_f = h                                # f = h
140                     if new_f < v.f or improved_g:
141                         v.h = h
142                         v.f = new_f
143                         v.parent = u
144                         if (nx, ny) not in in_open:
145                             pq.put((v.f, v)); in_open.add((nx, ny))
146
147         return expanded, 0  # no path
```

As noticed in the screenshots above, both algorithms get to the goal; however, they both take separate routes, and, as predicted, the A* version reaches the goal slightly faster than the Greedy

Best-First version. Even though the heuristic changed (Euclidean distance instead of the Manhattan distance), the overall results remained the same, because the approaches of each algorithm still remain the same. The greedy algorithm opened fewer nodes because it focuses on nodes closer to the goal, disregarding cost. The only difference is that the paths can go diagonally sometimes if it fits the strategy.

Part 3:

In this experiment, I tested certain values of Alpha (prioritizing g(n), the path cost accumulated so far and to other nodes) and Beta (prioritizing h(n), nodes that are closer to the goal without regard to cost). The results are shown below:

Section a

Table:

Alpha	Beta:	Observed Behavior
1	25	Graph turns downward later, initially stays on the horizontal path because it thinks it will reach the goal sooner.
5	16	Similar to the first one, the graph turns downward later, due to the relatively low alpha value and higher beta value. It acts more greedy (also more h(n)).
12	5	Contrary to the first two, the graph turns downward earlier, probably due to the prioritization of g(n) because the Alpha value is larger than the Beta value). It considers cost of going to nodes over ones that are solely closer to the goal.
20	1	Similar to the one above, since the alpha value is much higher than the beta value, the

		algorithm prioritizes g(n) over h(n) the graph expands more nodes and notices that staying on that horizontal track will not find the best path.

Code snippet incorporating the values of Alpha and Beta in the algorithm:

```
#### 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

        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 = (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))
```

Screenshots:

Test 1:

```
ALPHA = 1
BETA = 25
```

Test 2:

```
16    ALPHA = 5
17    BETA = 16
```

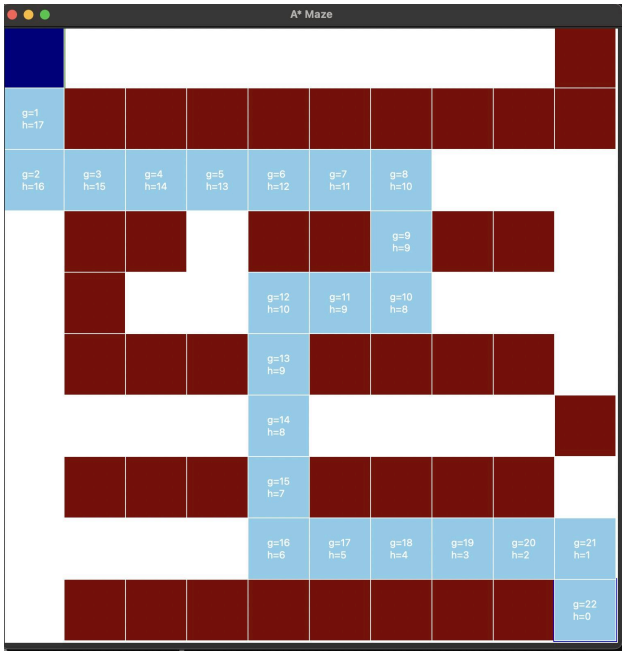
Test 3:

```
16    ALPHA = 12
17    BETA = 5
```

Test 4:

```
16    ALPHA = 25
17    BETA = 1
```

Maze for Test 1 and 2:



Maze for test 3 and 4:

Section b: **(only beta values):**

Code Used (showing how the value of Beta is incorporated in the pathfinding function):

```
133    ### Update the evaluation function for the cell n: f(n) = g(n) + h(n)
134    self.cells[new_pos[0]][new_pos[1]].f = (ALPHA * new_g) + (BETA * self.cells[new_pos[0]][new_pos[1]].h)
```

Test 1:

```
19    ALPHA = 1
20    BETA = 50
```

Test 2:

```
17    #initializing values of Alpha and Beta
18    ALPHA = 1
19    BETA = 25
```

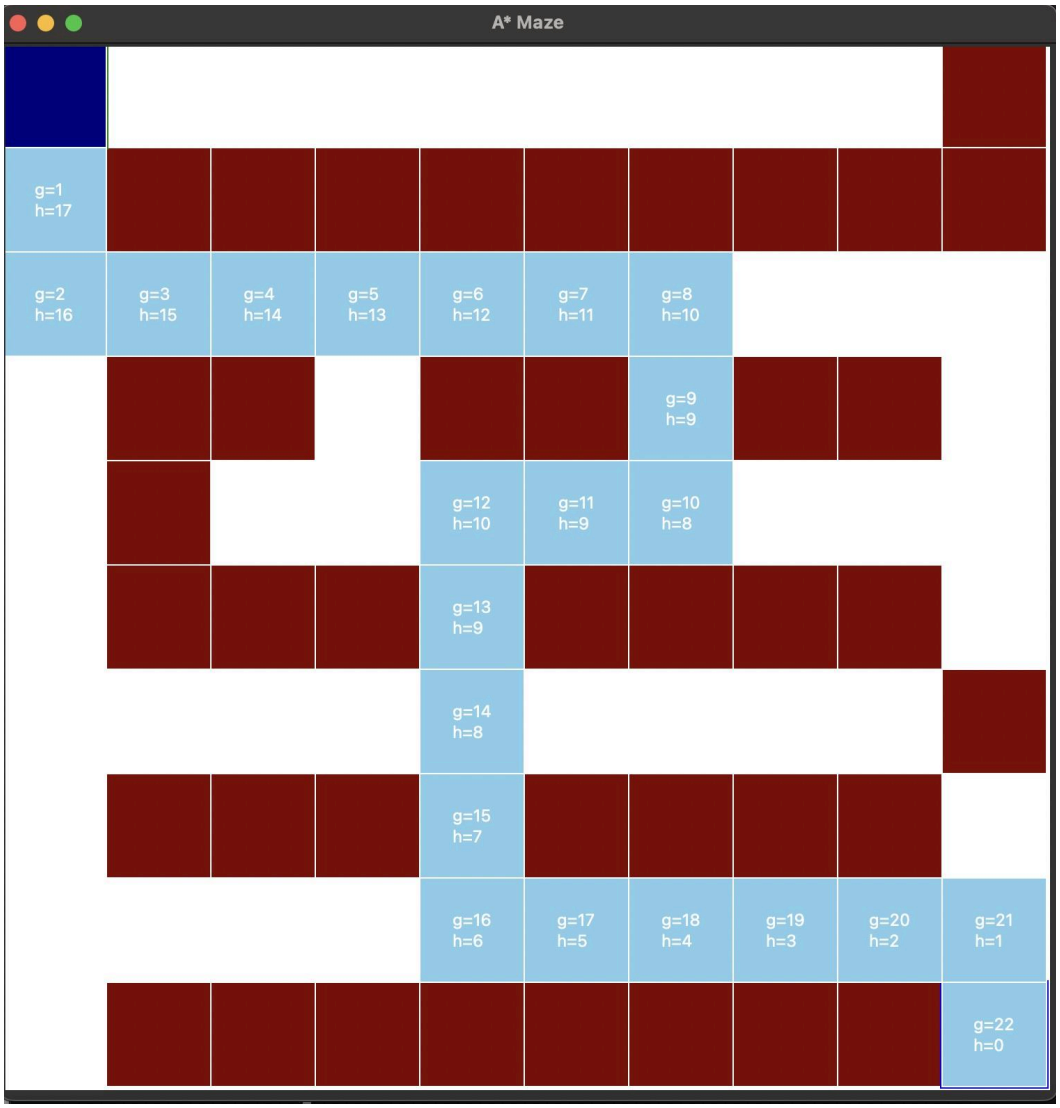
Test 3:

```
18    #initializing values of Alpha and Beta
19    ALPHA = 1
20    BETA = 10
```

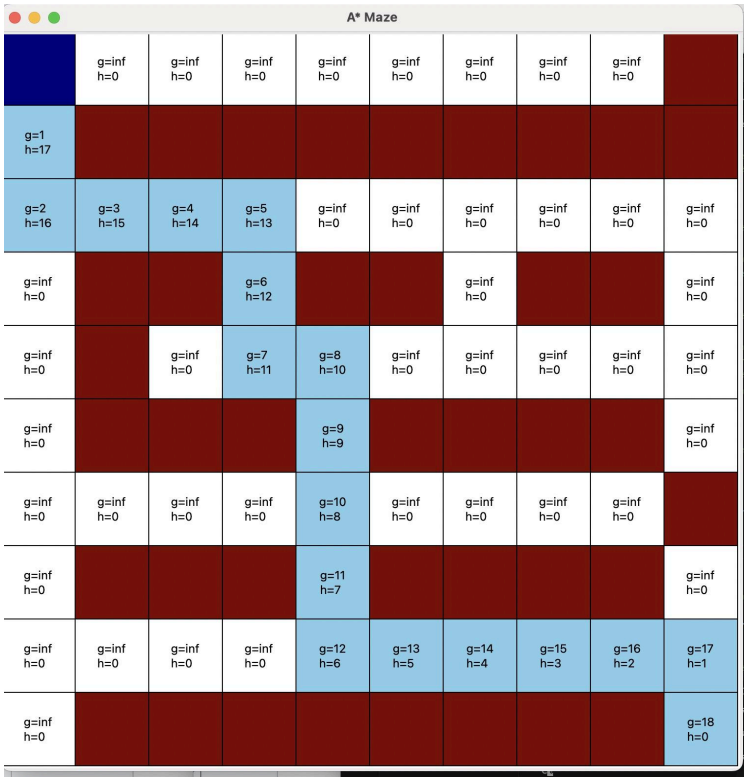
Test 4:

```
19    ALPHA = 1
20    BETA = 1
```

Picture of maze when Beta was 50 and 25:



Picture of maze generated when Beta is 10 and 1:



Alpha has the same value: For this experiment, the beta values varied, and the value of alpha was fixed at 1. As the beta was higher (like at 50 or 25), I saw that the path generated by the algorithm moved to the right more than the paths generated at lower beta values, which makes sense because as beta increases, the algorithm gets more greedy, and hence will open and follow nodes that it thinks are closer to the goal, without regard to other factors like obstacles or cost. It was hard to generate multiple paths for the beta values due to the nature of the maze; however, I did notice that when beta was 1, the algorithm turned earlier, which also makes sense because in that scenario, the algorithm would be less greedy and instead look at nodes with consideration to both cost AND distance to the goal. Since alpha and beta are the same in that scenario, the evaluation function would turn out to be $f(n) = g(n) + h(n)$.

Conclusion:

Overall, this experiment proved that paths can be different based on various values of Alpha and Beta. I saw that as Alpha was high and Beta was low, that the path finding was based on prioritizing costs (like Dijkstra's or Uniform Cost search), while when Alpha was low and Beta was high, the path finding was based on algorithms prioritizing the locally best choice (without consider to cost, much like the hill climbing or greedy algorithms). This shows that there is a difference between algorithms with a high g(n) value but low h(n) value, and vice versa.