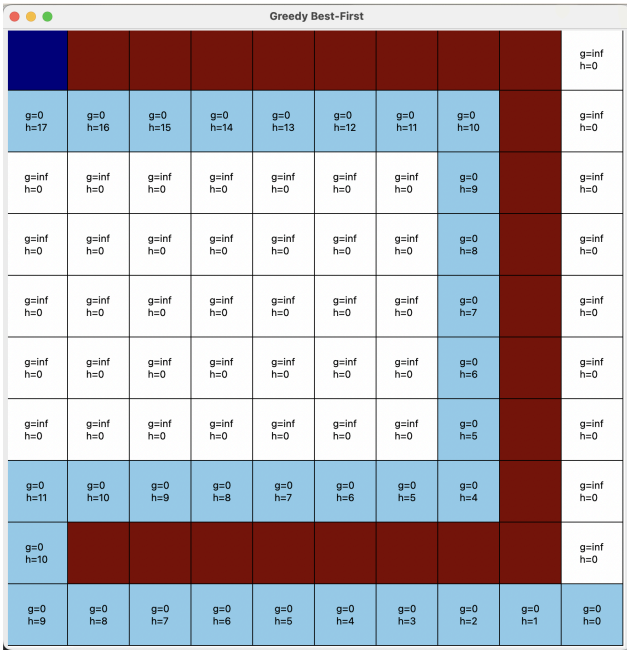
Erin Zahner

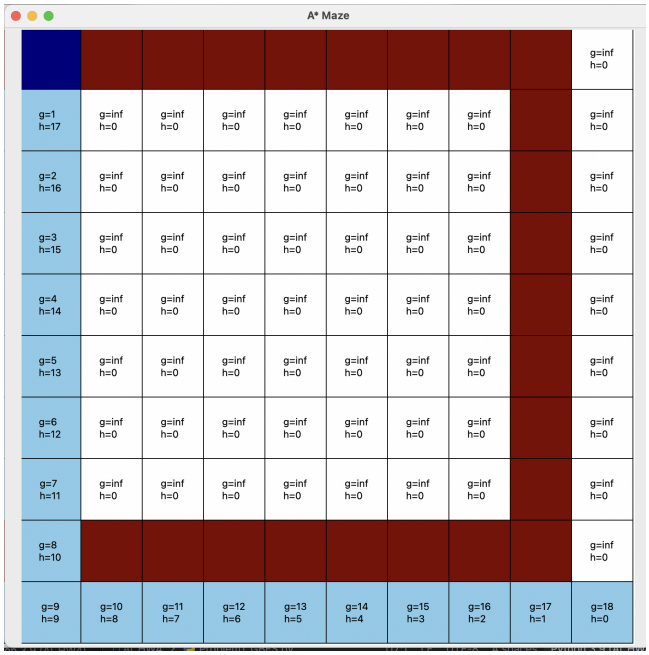
AI - HW 4

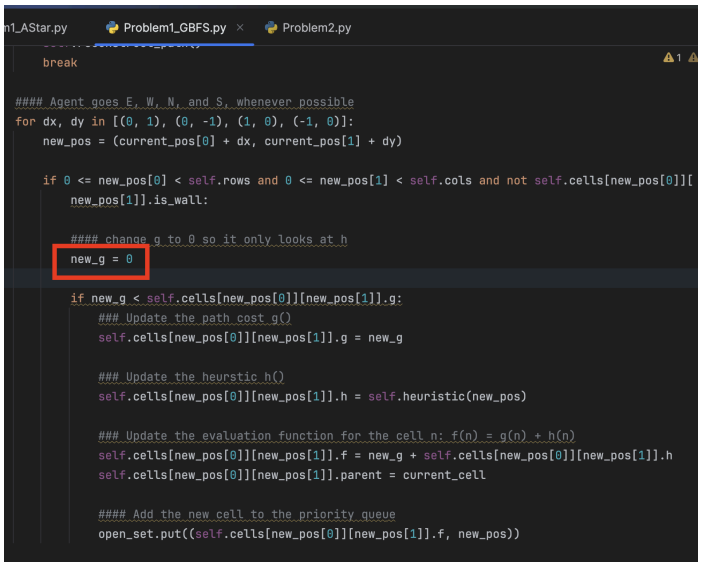
Problem 1

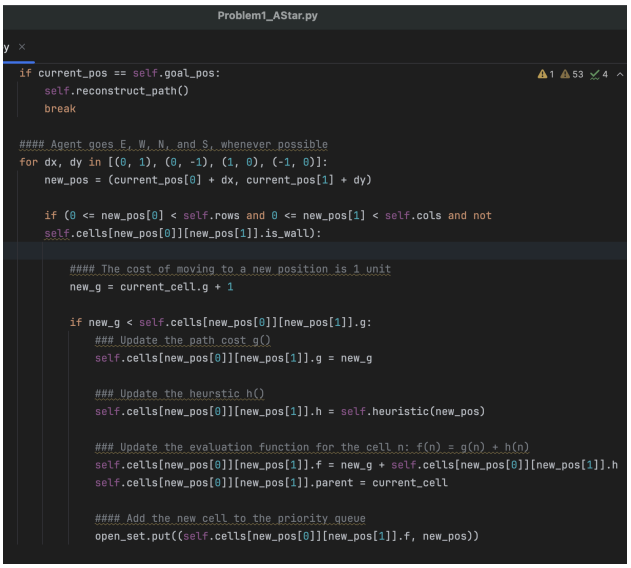
Greedy Best-First



A*



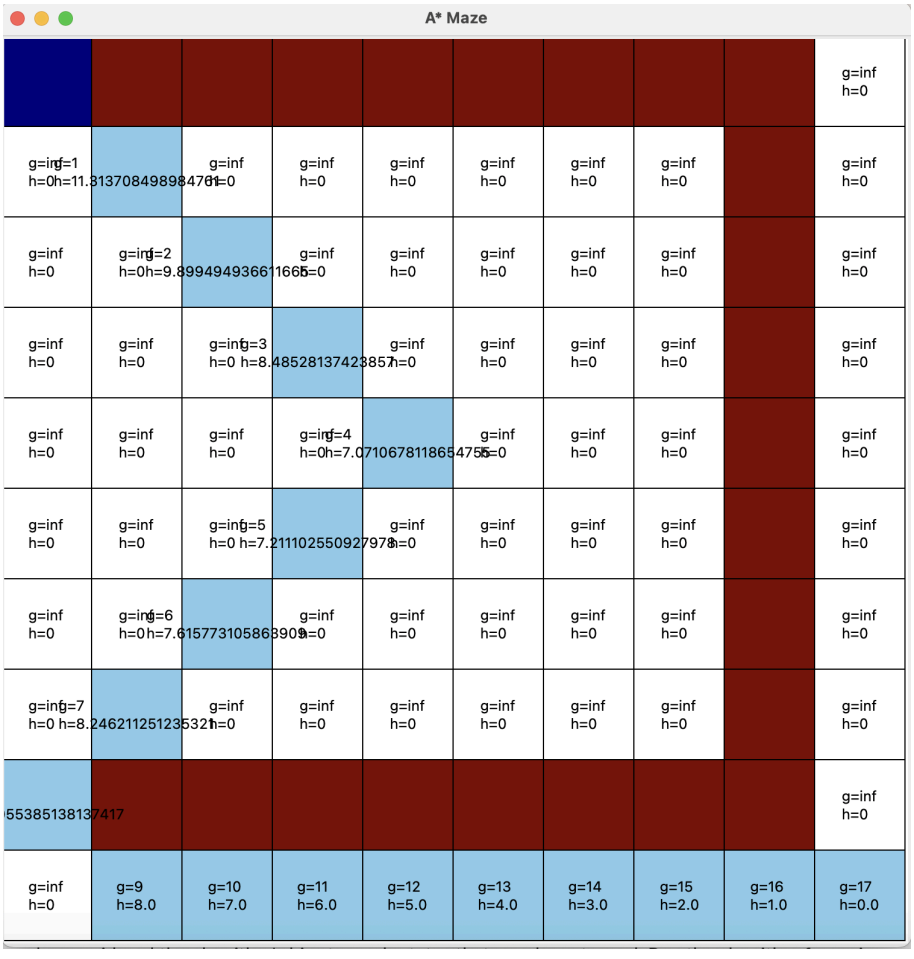




Explanation

To change the algorithm from an A* algorithm, in which the agent looks at both the actual cost and the heuristic, to a Greedy Best-First algorithm, which only cares about having the lowest heuristic value, all we have to do is change one line. Enclosed in a red box in my screenshot is the single line that had to change, the g value was changed to always be 0, forcing the agent to only look at the heuristic values. By doing this, I was able to leave the rest of the code alone and make minimal changes. In the pictures of the two mazes, you can see that with the Greedy Best-First algorithm, the agent takes a worse path because it is only looking at the heuristics, not the true costs. In contrast, the A* takes the best path because it looks at the bigger picture of the algorithm.

Problem 2



```
#############################################################
#### Euclidean distance
#############################################################
3 usages
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, and S, NE, NW, SE, SW whenever possible
moves = [(0, 1), (0, -1), (1, 0), (-1, 0), (1, 1), (1, -1), (-1, 1), (-1, -1)]
random.shuffle(moves)

for dx, dy in moves:

    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
```

Explanation

To change the heuristic from Manhattan distance to Euclidean, it was a pretty simple change. All I had to do was take out the absolute value and replace it with math.sqrt and square each term by typing **2. With these simple changes to the formula, the algorithm is now using the Euclidean distance as the heuristic, rather than the Manhattan distance. The next change was fairly simple as well. To allow the agent to move through the maze using diagonal moves, all we had to do was add these moves to the list available. These moves are (1, 1) - NE, (1, -1) - NW, (-1, 1) - SE, and (-1, -1) - SW. To ensure that one move was not favored over others due to the order they

were entered, I put all the moves into a list and randomly shuffled them. Now, before the loop, the directions are put in a random order to ensure less bias.

Problem 3 - Part 1

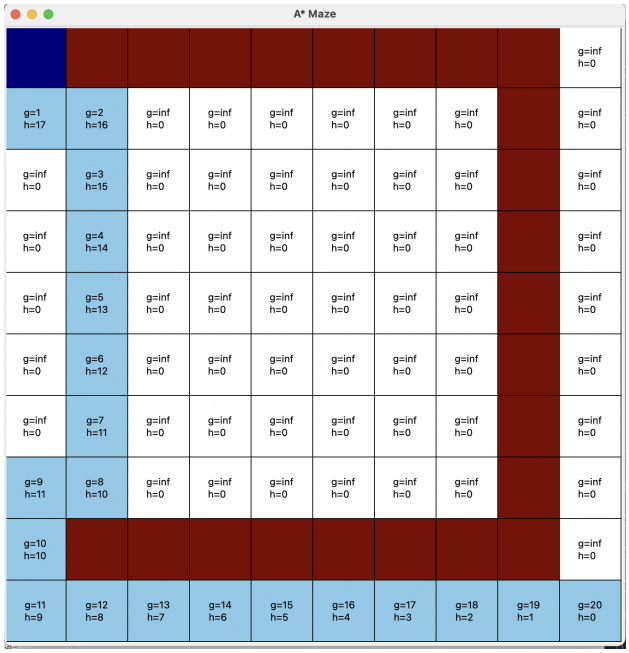
α	β	Observation
1	1	Base Case: same as with no weights
1	2	Some bias towards h(), leading it to move right when not necessary
1	3	Even more bias towards h(), pushing the path right even more, making it less optimal
2	1	Looks the same as base case
3	2	Same as base case
3	4	Same as base case
3	5	Same as 1,2; bias for h()

Explanation

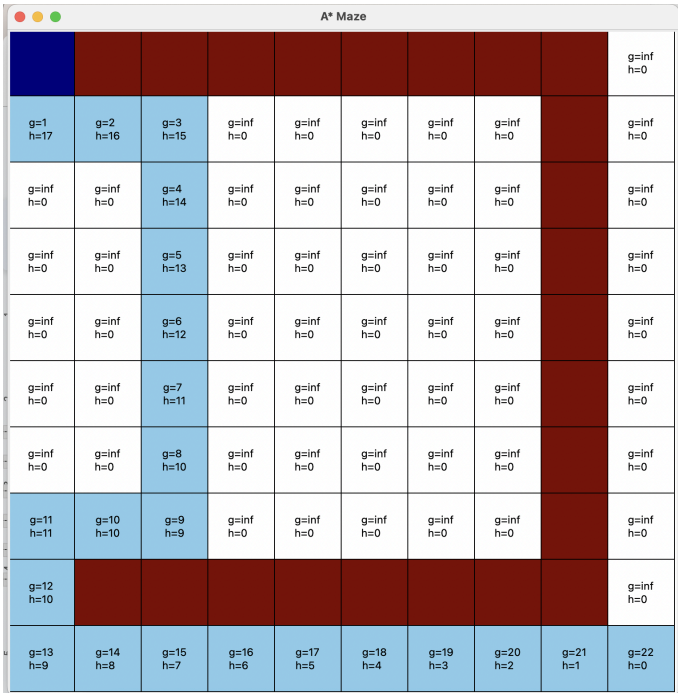
To test the impact of weights on the g() and h() in the function, I added two new variables that could be changed when you call the maze function. In the code, you will see w_g and w_h (weight_g, weight_h). These are initialized in the constructor and used when calculating the function f(n). When calling the maze function, I would set them equal to the weights I wanted to test, for example, “game = MazeGame(root, maze, w_g=1, w_h=2)”. It was interesting to see the difference in the behavior of the agent based on the weights provided. When the h() was more weighted such as when α was 1 and β was 2 (seen in the first picture below), the path went right to favor the heuristic over the actual cost. However, when the weights were reversed and α was 2

and β was 1, the path looked the same as the original. Furthermore, when I tried to adjust the weights more, if the g() was weighted, it took more for the h() to overpower it. For example, when α was 3 and β was 4, the path was still favoring g(), despite h() having more weight.

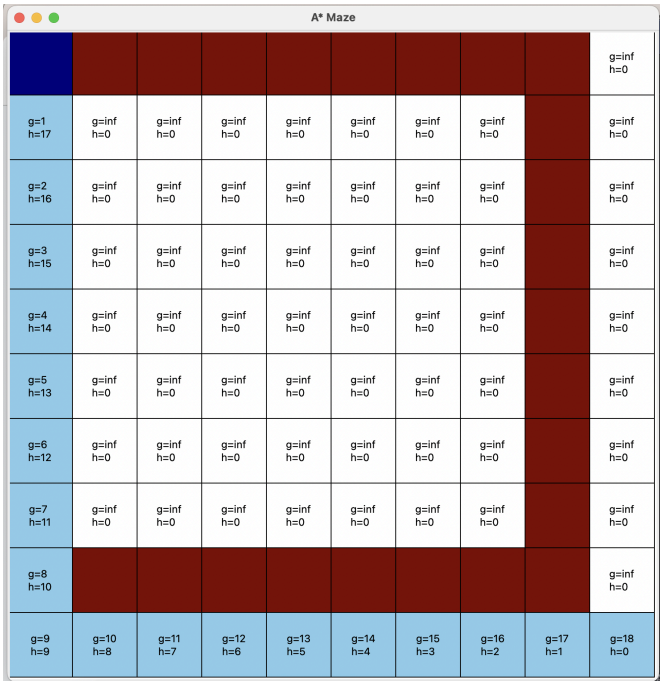
α = 1 and β = 2



α = 1 and β = 3



α = 2 and β = 1 and α = 3 and β = 4

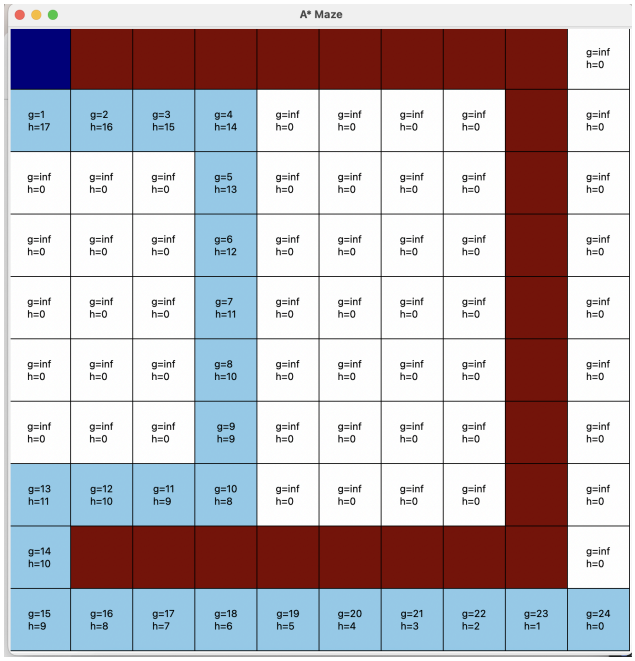


Problem 3 - Part 2

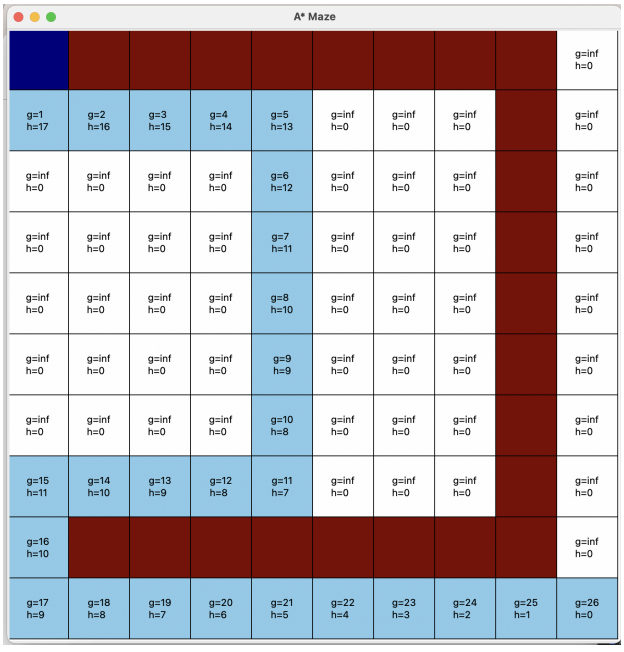
For part 2 of problem 3, I tested the effect of just changing the β value. The results of this were very interesting. While at first there was an obvious change when β was increased, as seen in the photos above, the effect eventually plateaus. When going from a weight of 1, to 2, to 3, there is a change every time to favor h() even more, however, when the weight reaches 4 and 5, it does not change until it increases to 6 (result seen below). Similarly, once the weight reaches 6, nothing changes until the weight is increased to 12. The effect of the weight becomes less severe as the weight becomes greater, in the sense that it takes a greater and greater weight to make a difference. After 12, there seems to be no possible weight that will make the path favor the heuristic to any greater extent. I tried all the way up to a weight of 10,000,000,000 and the path

did not change. It seems that as long as g() is also being considered, favoring the heuristic can only go so far. We saw in part one that when we only look at the h() value, the path goes as right as possible before turning around, however, even when the h() value is favored by 10,000,000,000, it does not have the same effect as only looking at the heuristic.

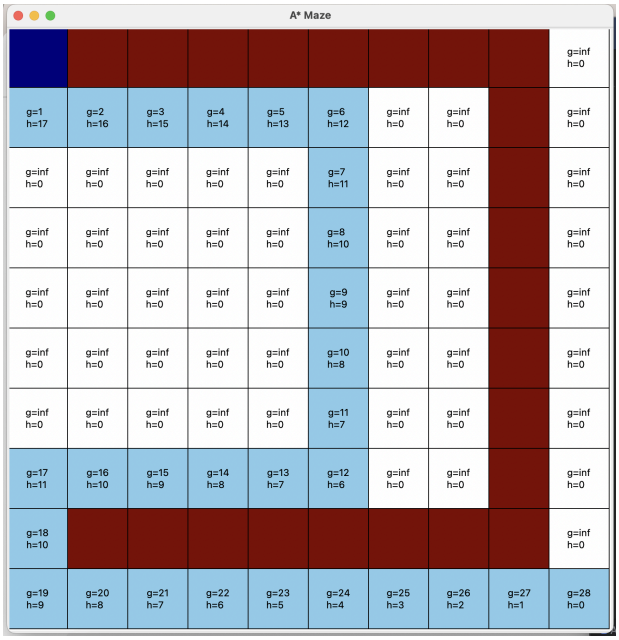
β = 4-5



β = 6-12



β = 12-10,000,000+



Conclusion:

From the tests done on the performance of finding the best path in maze based on A* vs Greedy Best-First, Euclidean distance using diagonals, and the effect of weights, we can see that there are ways to optimize or manipulate the choices an agent will make. The first test revealed to us that the A* algorithm is much better at finding the optimal shortest path compared to the Greedy Best-First. In the second test, we were able to see that by using diagonals jumps and Euclidean distance, the agent was able to find a path that was one jump shorter than when we used Manhattan distance with no diagonal moves. Finally, we saw that changing the weights of g() or h() has the potential to change the path a significant amount, at least to a point. While changing the weight of h() from 1-5 has a big change, the effect plateaus drastically after that. This shows that there agent will only trust the heuristic to a certain extent, even when it has a weight of 10,000,000.