**Question 1**

**A\* Algorithm**



**Greedy Best First Search Algorithm**

A screenshot of a game

Description automatically generated

**Explanation**

In the first maze, I used the A\* algorithm to obtain the best optimized path using the function f(n) = g(n) + h(n), with g(n) being the path cost and h(n) being the heuristic. The distance was calculated using the Manhattan Distance formula. In the second maze, I used a greedy best first search algorithm to find a path in which the only thing considered was which node was the best to go to at that time. I accomplished this by removing path cost entirely from the A\* algorithm that I used previously, leaving me with f(n) = h(n), with h(n) being the heuristic. In the A\* algorithm, you can see how it ended up choosing a path that was shorter than the one chosen by the greedy best first search algorithm because it considered both path cost and the heuristic. However, with the GBFS algorithm, it ended up choosing a longer path because it only focused on the distance of its position from the goal node without considering the path cost. This difference can be seen in the path that it decided to take to get to the goal node. In the GBFS algorithm, you can see how it was only focused on shortening the distance between the goal node and its current position. The second that it got to (2, 3), it began to move horizontally because that would be shortest path if there weren’t any walls. This caused the GBFS algorithm to pick a path that was longer than needed.

**Question 2**

**A\* Euclidean Distance Algorithm**

A screenshot of a computer

Description automatically generated

**Greedy Best First Search Euclidean Distance Algorithm**

A screenshot of a computer

Description automatically generated

**Explanation**

In A\* algorithm, I calculated the distance between the position of the agent and the goal node using the Euclidean distance formula and allowed the agent to make diagonal moves. For the GBFS, I did the same as the A\* algorithm but removed g(n) from the equation so that it only focused on the heuristic. In the GBFS algorithm, you can see the agent actively trying to shorten the distance between the goal node and the current position, especially in the straight line it takes toward the goal node after (2, 0) before it corrects itself whereas, in comparison, the A\* algorithm chooses the best path instead of just making a beeline to the goal node.

**Question 3**

**Part 1**

|  |  |  |
| --- | --- | --- |
| **α** | **β** | **Observed Behavior** |
| 3 | 1 | no change from optimum path |
| 6 | 1 | no change from optimum path |
| -3 | 1 | increased amount of square needed to reach goal node, seems to be taking the least optimal path |
| -6 | 1 | Same as -3 |
| 1 | 4 | increased amount of square needed to reach goal node, seems to be focusing on just lessening the distance between goal node and current position without regard to walls |
| 1 | 6 | further increased amount of squares needed ot reach goal node, even more focused on just lessening distance |
| 1 | 3 | brings the path closer to the optimum path |
| 1 | 2 | brings the path closer to the optimum path |

**Explanation**

In these trials, I increased and decreased the values of both beta and alpha one at a time in order to see how beta and alpha affected the path that was decided on. The heuristic that I used was Manhattan distance. To fully observe the effects, I kept one of the variables constant while increasing and decreasing the other variable to see how the path changed in comparison to the optimum path.

In my observations I found that, with lower values of alpha, the algorithm tends to aim for less optimal paths and makes more moves to lessen the distance toward the goal node without considering obstacles. Increasing the value of alpha would bring the path closer to the optimum path. With higher level of beta, however, it seems the algorithm is aiming directly for the goal node without considering the walls that might be preventing it from reaching that goal node. If the value of beta decreases though, it brings the path closer to the optimum path.

**Part 2**

When increasing the bias towards states that are closer to the goal, the path chosen deviates from the optimum path and goes more based off of how close each node is to the goal without considering the obstacles whereas, when the bias is decreased, it actually ends up bringing the path closer to the optimum path because it doesn’t put as much weight on that heuristic.

**Beta = 6, Alpha = 1**

**A screenshot of a game

Description automatically generated**

* **The path is far (3 columns away) from the optimum path**

**Beta = 4, Alpha = 1**

**A screenshot of a game

Description automatically generated**

* **Path is now 2 rows away from the optimum path**

**Beta = 2, Alpha = 1**

**A screenshot of a computer game

Description automatically generated**

* **Paths is now 1 row away from the optimum path**

**Alpha = - 3, Beta =1**

**A screenshot of a computer game

Description automatically generated**

**Alpha = - 6, Beta =1**

**A screenshot of a game

Description automatically generated**

**Alpha = 3, Beta =1**

**A screenshot of a game

Description automatically generated**

**Alpha = 6, Beta =1**

**A screenshot of a computer game

Description automatically generated**

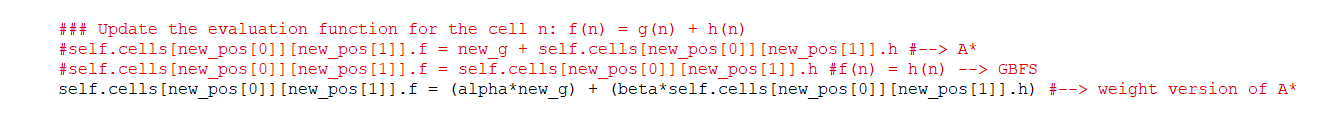
**Code Snippets**

**Code used for Euclidean and Manhattan Distance**

**A group of colorful lines

Description automatically generated with medium confidence**

**Formulas used to define f(n) for A\*, GBFS, and weighted A\***

****

**Full find\_path function**

**A screenshot of a computer

Description automatically generated**

* **The line to add a new cell to the priority queue is cut off**