HW 3

Title: AI Assignment 3

Name: Brennen Cramp

Date: 10/14/2025

Problem 1:

First, the implementation of Greedy Best-First Search had to be created by modifying the A\* algorithm and a visited set needed to be created to stop cycles and redundant path traversing. One thing to note is that Greedy BFS does not factor in the path cost [g(n)] thus f(n) = h(n) so the section where it would decide what to add to the open\_set priority queue is not based on the new path cost value but by if the cell had already been visited (seen in *Figure 3*). The maze was also altered to show how the optimal paths are found for each search algorithm. The result for using Greedy BFS is seen in *Figure 1*, where the path traverses 27 cells including the start and goal cells. Next, the A\* search was executed on the same maze configuration where the optimal path traverses 19 cells (seen in *Figure 2*) including the start and goal cells which is a shorter path than Greedy BFS. A\* search is the optimal decision as opposed to Greedy BFS since A\* considers both the path cost and the Manhattan Distance heuristic to search for the optimal pathing.

A screenshot of a computer

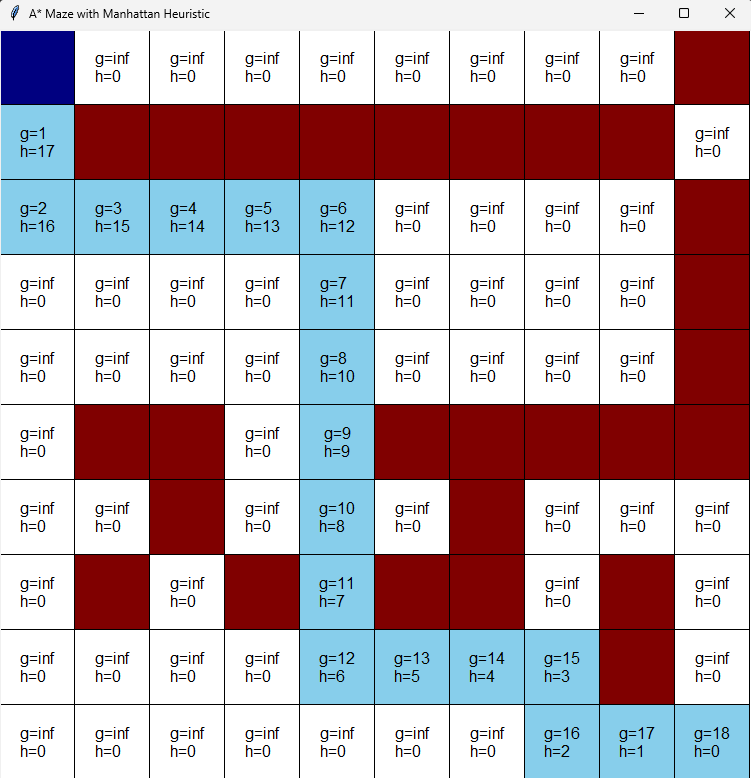
AI-generated content may be incorrect.

Figure : Greedy BFS with Manhattan Heuristic Figure : A\* with Manhattan Heuristic

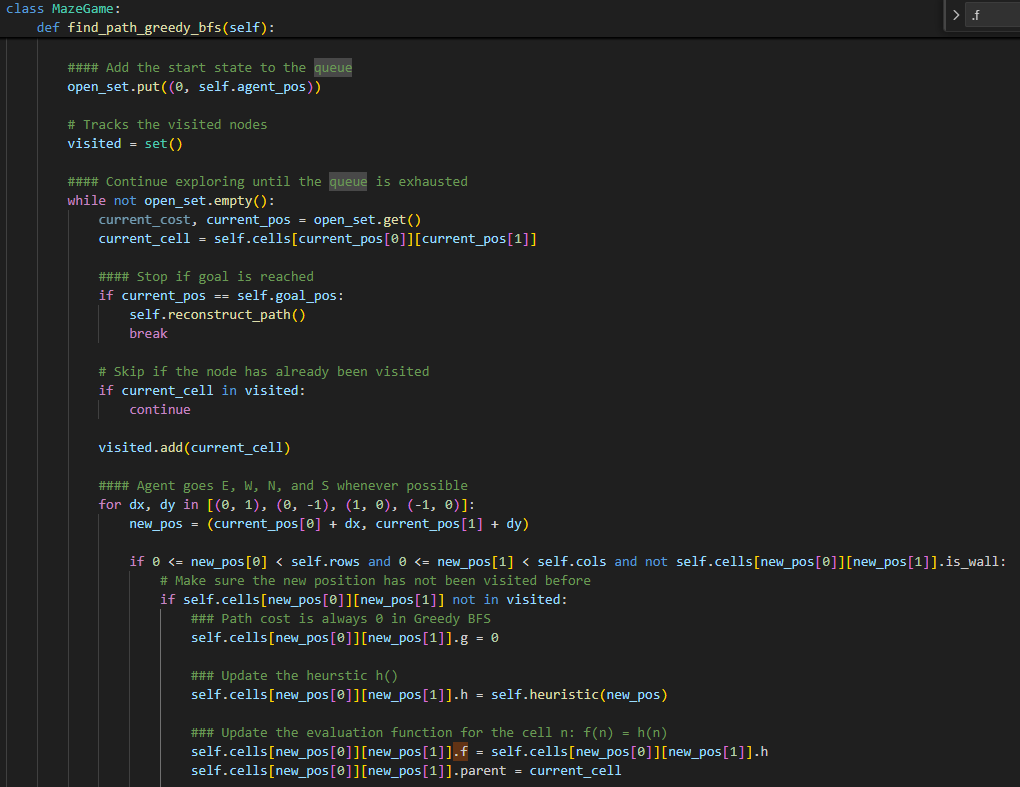


Figure : Find Path Function using Greedy BFS

Problem 2:

To change the heuristic and add randomizing of the moves list, both the “math” and “random” libraries were imported (seen in *Figure 6*) to use the sqrt() and shuffle() functions. The implementation of the Euclidean Distance for the heuristic caused the f(n) and h(n) values to have many digits after the decimal when displayed on the graph so to control this, the round() function was used to cap the decimal places to the thousandths (seen in *Figure 9*). The same algorithm functions used to find the paths and maze configurations from Problem 1 were also used to solve this problem. After changing the heuristic to use the Euclidean Distance (seen in *Figure 7*), allowing the agent to move diagonally (seen in *Figure 8*), and making the move list randomized to iterate over (seen in *Figure 8*), the mazes were generated. In Figure 4, Greedy BFS was used with the Euclidean Heuristic which traversed 24 cells, including the start and goal cells, as opposed to the A\* implementation (seen in Figure 5) which with the inclusion of travelling diagonally had the optimal path with 13 cells traversed. Using the Euclidean Distance as a heuristic instead of the Manhattan Distance proved to be the optimal decision since the path found when using Greedy BFS decreased by 3 cells (27 to 24) and the path found during A\* search decreased by 6 cells (19 to 13). Additionally, A\* search proved to be more optimal than Greedy BFS due to Greedy BFS not factoring in the path cost and sheerly operating on the heuristic value which led the pathing to go an inopportune route first.

A screenshot of a computer game

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Figure : Greedy BFS with Euclidean Heuristic Figure : A\* with Euclidean Heuristic

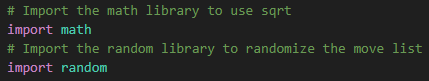


Figure : Added Import Statements

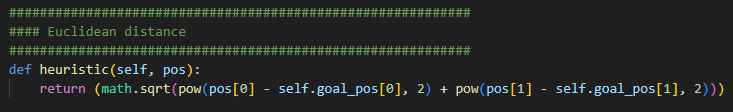


Figure : Euclidean Distance Heuristic

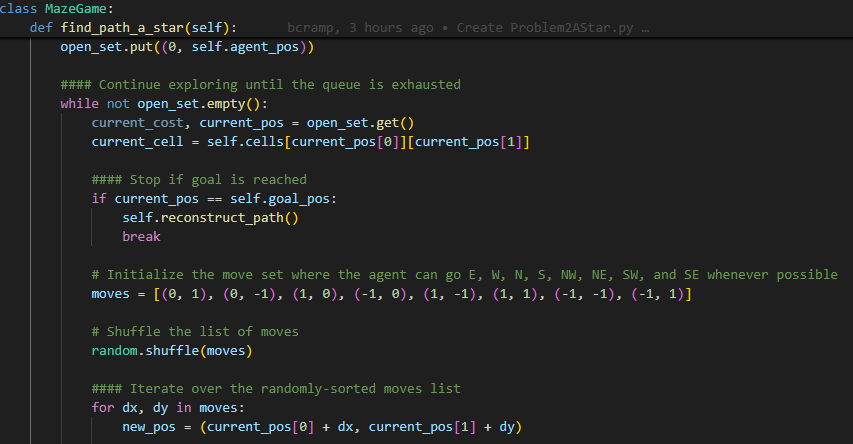


Figure : Find Path Function with Increased Directions and Randomizing Move Set

A computer screen shot of a computer code

AI-generated content may be incorrect.

Figure : Rounds h(n) and f(n) to have 3 Decimal Places

Problem 3:

The program used in Problem 1 (Manhattan Distance) for A\* search was utilized to test Problem 3 with the same maze configuration. The file was modified with class variables for α (“alpha”) and β (“beta”), which can be seen in *Figure 12*, and applied to the spots in the code where f(n) is being calculated (seen in *Figure 13*), meaning the actual g(n) and h(n) calculations were not changed. Instead of using f(n) = g(n) + h(n) for calculating f(n), f(n) = a \* g(n) + b \* h(n) was utilized. Test coefficient values were included in the table below and observed behavior of the resulting maze for the unique pairing of coefficients. When changing the coefficients, only two paths were observed for the resulting maze which can be seen in *Figure 10* and *Figure 11*. While testing, it was seen that changing the α-value did not affect the optimal path found if β was never greater than α \* 3. However, when β > α \* 3, the path was affected by visiting 2 more cells, straying off the optimal path where it seemed the agent was lost for 3 cells before returning to the optimal path. Since β can be considered the algorithm's bias towards states that are closer to goal, once β > α \* 3 was reached at any point, as seen in the table below (α=1 and β=4; α=1 and β=100; α=1 and β=1000; α=0 and β=1; α=0 and β=1000; α=7 and β=22), this means that the algorithm is relying more on the heuristic. This was observed earlier in how Greedy BFS only relied on the heuristic, so the algorithm is leaning into more “greedy” behavior once that threshold of β > α \* 3 was reached.

|  |  |  |
| --- | --- | --- |
| α | β | Observed Behavior |
| 1 | 1 | The search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 2 | 1 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 3 | 1 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 1 | 2 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 1 | 3 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 1 | 4 | Search path increased and took a unique route with 21 cells (incl. start/goal) visited where it went 1 cell farther to the right and came back to nominal path when it hit a wall going down (*Figure 11*). |
| 1 | 100 | Search path traversed the same 21 cells (incl. start/goal) as done when α=1 and β=4 (*Figure 11*). |
| 1 | 1000 | Search path traversed the same 21 cells (incl. start/goal) as done when α=1 and β=4 (*Figure 11*). |
| 0 | 0 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 0 | 1 | Search path traversed the same 21 cells (incl. start/goal) as done when α=1 and β=4 (*Figure 11*). |
| 0 | 1000 | Search path traversed the same 21 cells (incl. start/goal) as done when α=1 and β=4 (*Figure 11*). |
| 1 | 0 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 1000 | 0 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 7 | 21 | Search path stayed the same with 19 cells (incl. start/goal) visited (*Figure 10*). |
| 7 | 22 | Search path did the same 21 cells (incl. start/goal) as done when α=1 and β=4 (*Figure 11*). |

A screenshot of a computer game

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Figure : Expected Optimal Path Figure : Path Traversed when β > α \* 3

A screenshot of a computer program

AI-generated content may be incorrect.

Figure : Class Variables to set Coefficients

A computer screen shot of a program code

AI-generated content may be incorrect.

Figure : New Calculation for f(n)

Problem 4:

Leet code accepted submission for the “Path with Minimum Effort (1631)” problem:

A screenshot of a computer

AI-generated content may be incorrect.

I used Dijkstra's algorithm by first initializing a grid of all the efforts as infinity so when actual values are found, the smallest effort will replace it and then iterating over the open\_set (priority queue) to traverse the neighboring cell heights passed in. The next step was to do basic boundary checking and if those checks passed, I had to iterate over the possible moves that can be applied to the current position. If the position is valid, I calculated the effort difference and minimized the maximum effort where it will be checked if the new effort is less than the effort currently stored at the new position; if the new effort is less than the effort at the new position, I replace it and add the new position to the open\_set to keep expanding and find other efforts to find the path where the max difference between the neighboring cells has been minimized.