Gabriel Brown
CSC 412 Intro to AI

Programming Assignment 1: Maze Search

While Python was the preferred programming language, and plenty of material was provided to work with the project in Python, I ultimately decided to make my own version of this assignment in Java. I began by making a function that would read in the maze text files and interpret them correctly. This function will take in the maze text file and output a 2D array of positions which ultimately represents our maze. The positions class is an extensive class as it holds a lot of unique data, but most importantly, it holds the characters of each coordinate on the maze. We also store the row and column of each coordinate, the parent position, or the previous position, and all necessary data for our A* heuristic such as our cost and whether or not we have already visited the square.

We have two other objects which includes a “player” and an “endpoint.” Both are pretty similar and contain a position themselves. The main difference between the two objects is that the player can move and change positions while the endpoint obviously cannot. In both our player and endpoint constructor, you must pass a character “symbol” as well as the position array maze. We do this in order to have our constructor find our position for us by looking for the symbol associated with each. For the player, our symbol is “P” and for the endpoint, our symbol is “.”.

Once these are found, we are ready to begin our breadth first search. For this project, we chose breadth first search in order to compare its best outcome to A* search. To do this, we used a queue to hold our possible moves. Using a check moves function, we ensure our moves are valid by making sure the position character is not a “%” and also within the bounds of our array. If both are true, we add the path. Following this, we simply follow our queue, which travels through our maze pretty much how water would, moving into each new area at the same time, filling every single space it finds available, until it finds the end. What is most important is adding our previous space, the space that basically “found” the movable space into our previous space

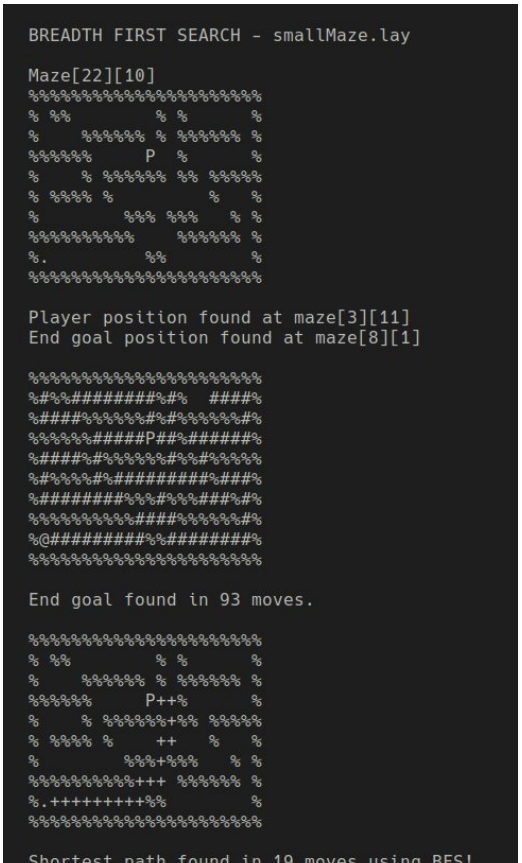
in our position class This allows us to backtrack in order to find the most efficient path in both our breadth first search and A star search.

For the A star search, I decided to add a cost integer to our position class. This class has a function which passes in the searcher and endpoint and it uses their positions to calculate our cost. We are using the Manhattan Distance so we are simply adding up the difference in rows and columns between two positions, then adding them up. So our cost is the sum of the differences between our searchers position, our nodes position, and our endpoint's position. After checking for valid positions, and collecting their cost, we evaluate them and choose the lowest cost position to move to before erasing it from our list. We continue doing this, choosing the lowest value, until we reach our endpoint.

Now lets get into our results. The '@' symbol is the head of our search, arriving at the endpoint. The '#' is every position the searcher goes to in search of the endpoint, and the '+' symbol marks the optimal path.

Part 1:

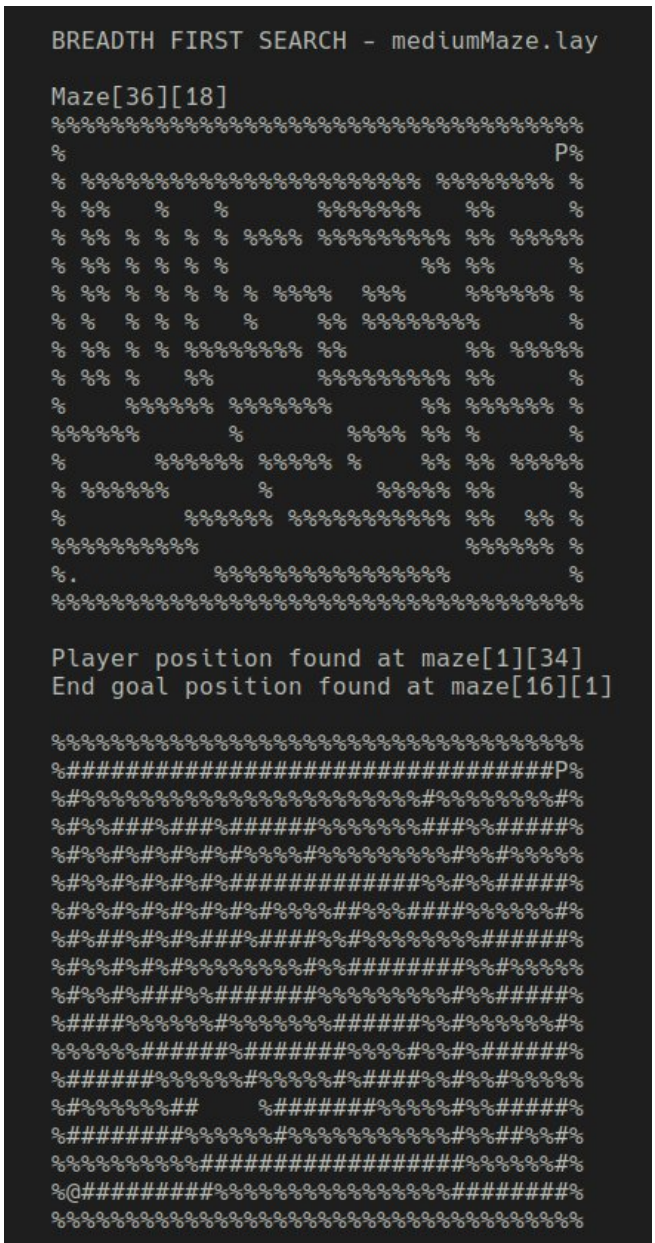
Small Maze

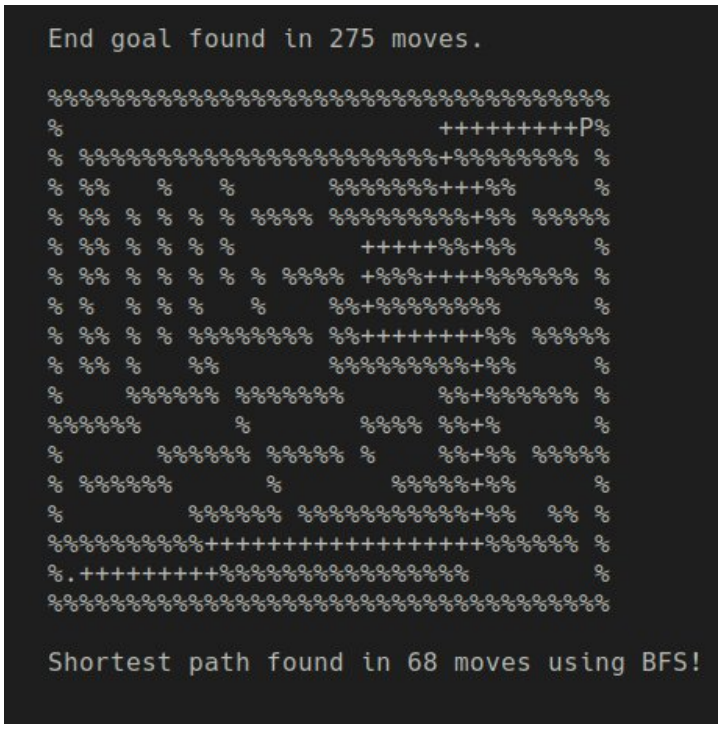


Gabriel Brown
CSC 412 Intro to AI

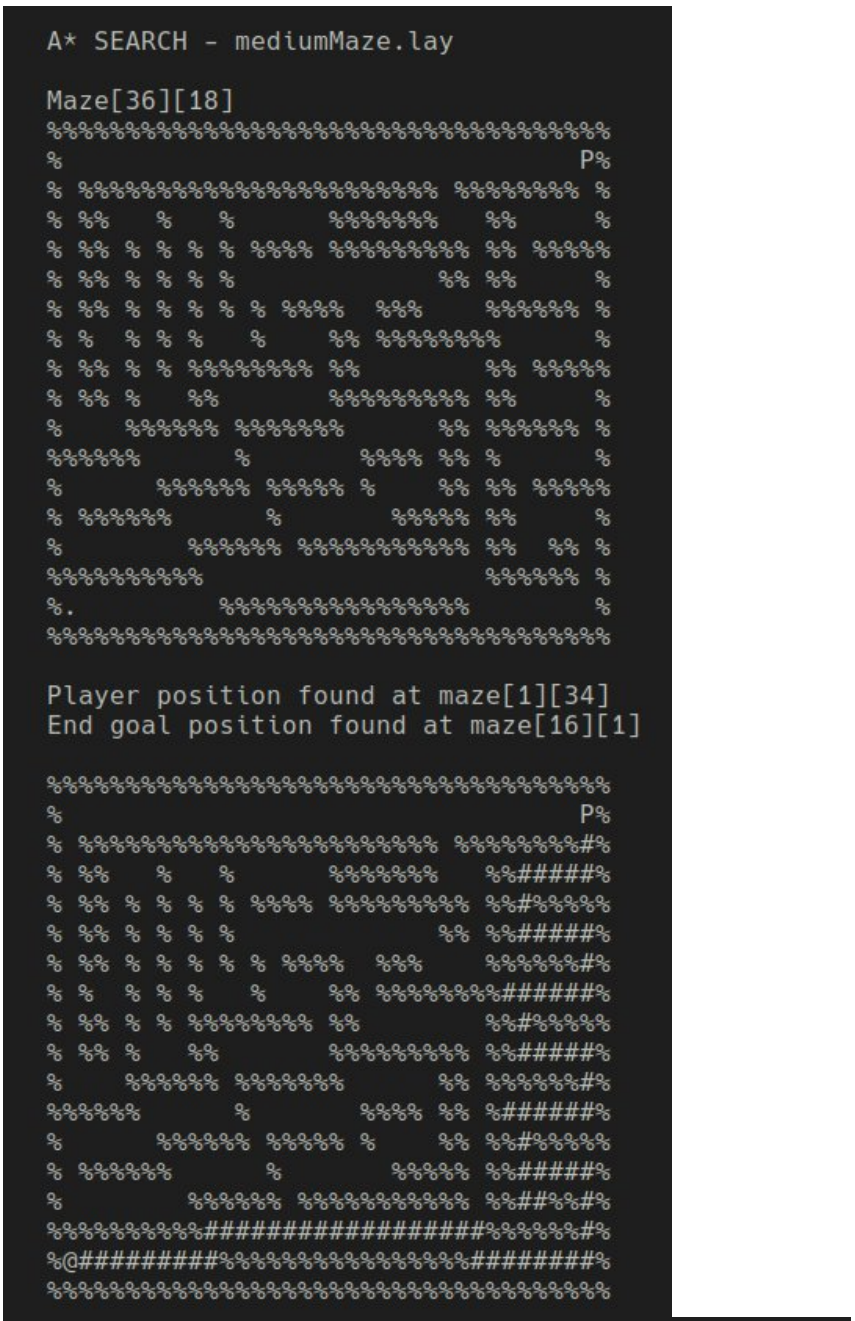
The small maze has some pretty interesting results. Because the player is essentially placed right in the middle of the maze while the endpoint is in a corner, the breadth first search nearly has to exhaust every single option in order to find the endpoint. 93 moves had to be made to find the endpoint, but the identified shortest path is 19 moves. With A star search, we make less than half the moves made for breadth first search, but our shortest path is quite a bit longer at 29 moves. In conclusion, A star was much more efficient with half the moves, but the shortest found path is not an insignificant amount longer than the path found with breadth first search.

Medium Maze





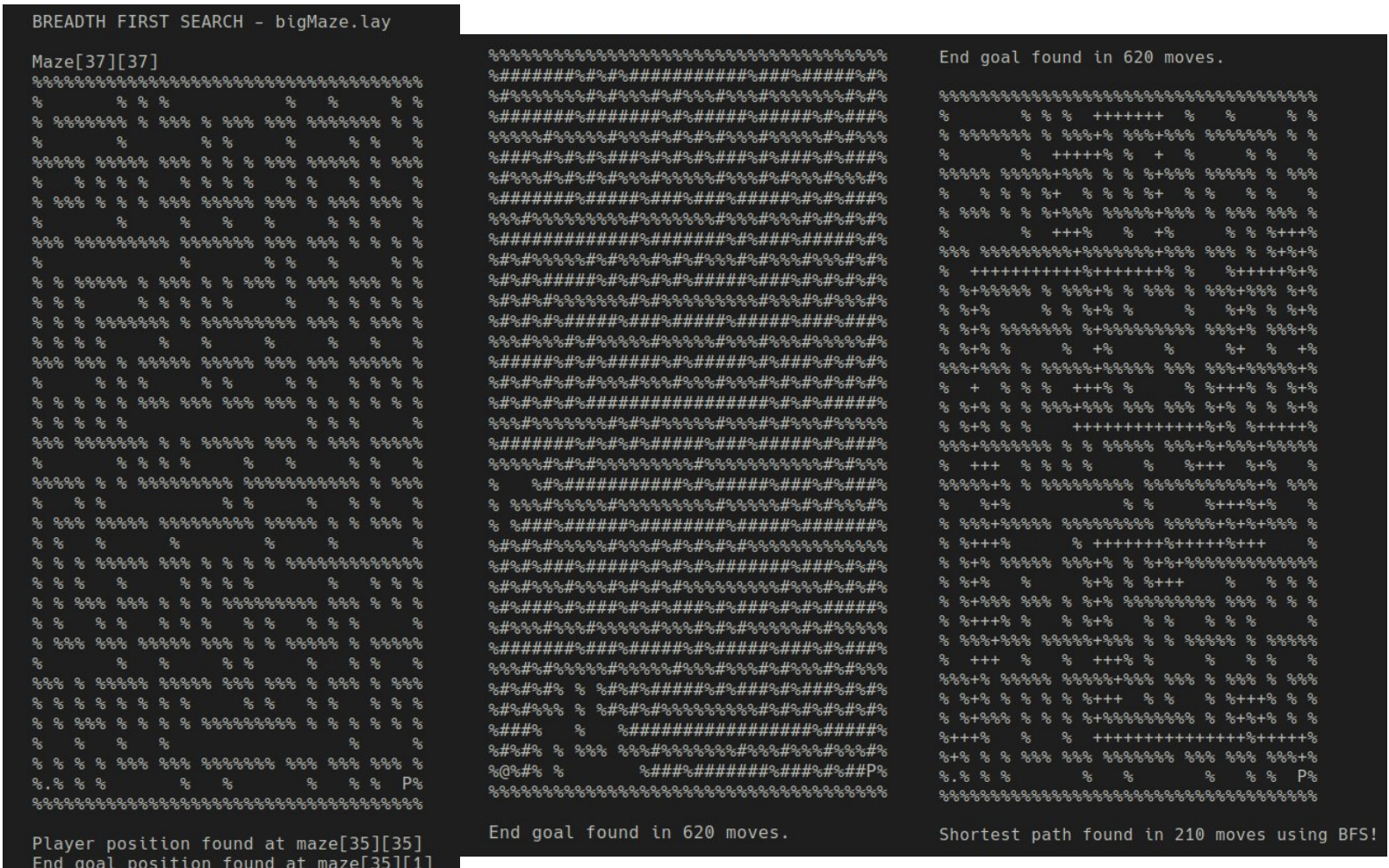
Gabriel Brown
CSC 412 Intro to AI



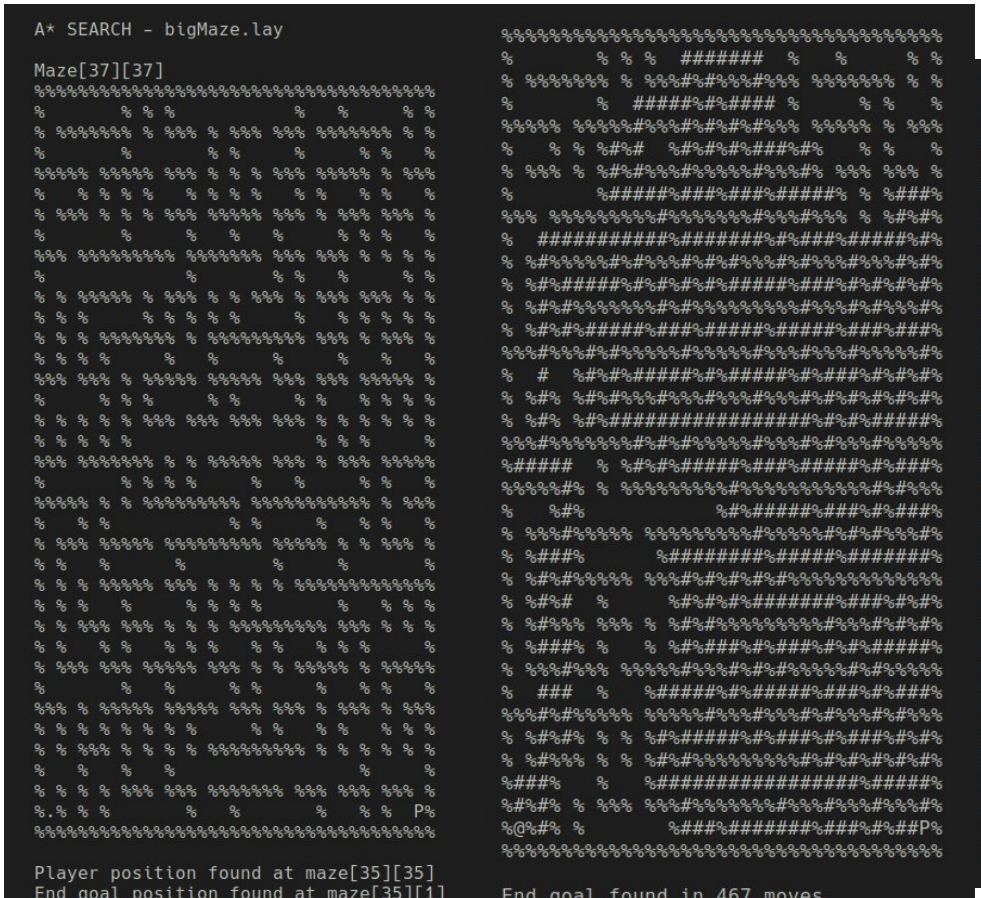
Gabriel Brown
CSC 412 Intro to AI

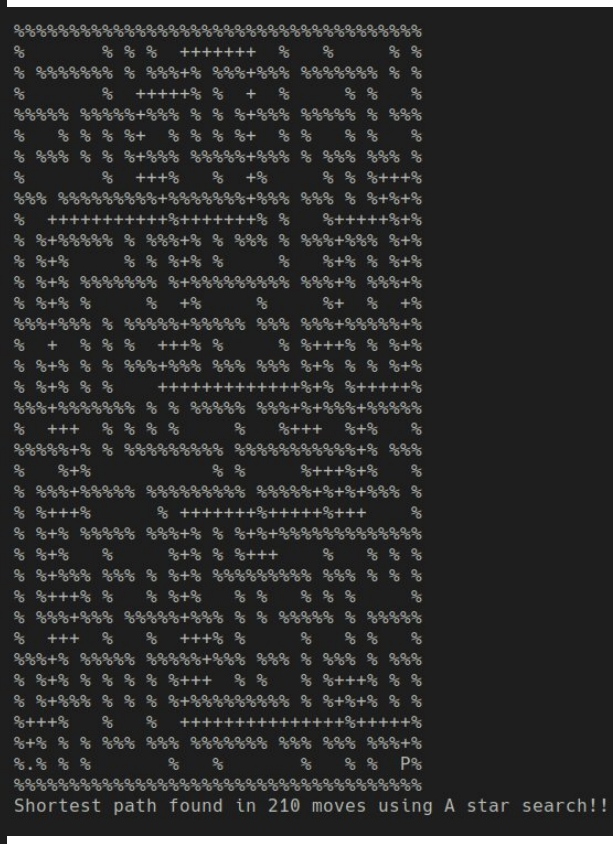
The medium maze begins to show the power of the A star search. While our breadth first search took 275 moves to find the endgoal, our A star search only took 82 moves, about a third of the moves needed. Breadth first search did ultimately come out on top with a shortest path of only 68 moves, but our A start search wasn't far behind at all with a shortest path of 76 moves.

Big Maze



Gabriel Brown
CSC 412 Intro to AI



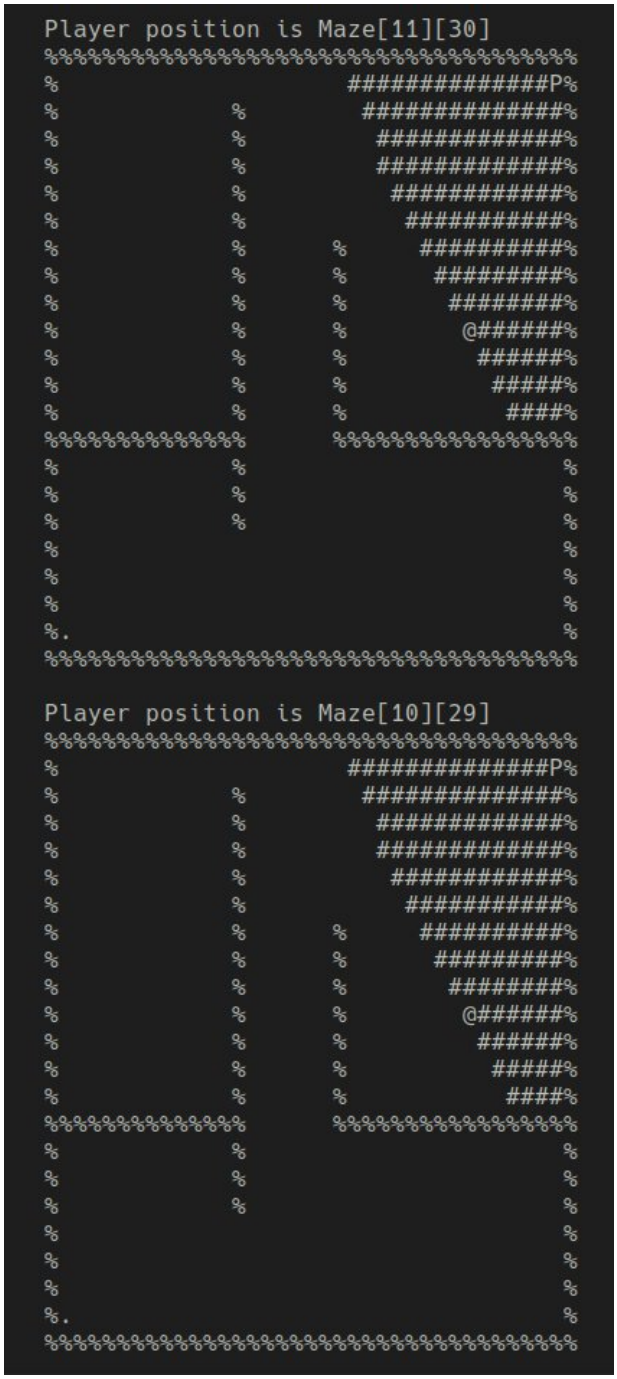


The big maze is a pretty interesting case too. This maze is in some sense designed to not work well with our heuristic, at least in terms of efficiency. This is because of how far we must travel up the maze to reach our end goal, and as we can see, this is inevitable by the design of the maze. It is interesting to note though, that both found the exact same path, the most optimal path, using this maze. So, when comparing both, A star is the absolute winner as it found the exact same optimal path in significantly less moves. While it took bfs 620 moves, it took a star 467 moves, and both found a path that takes 210 moves to reach.

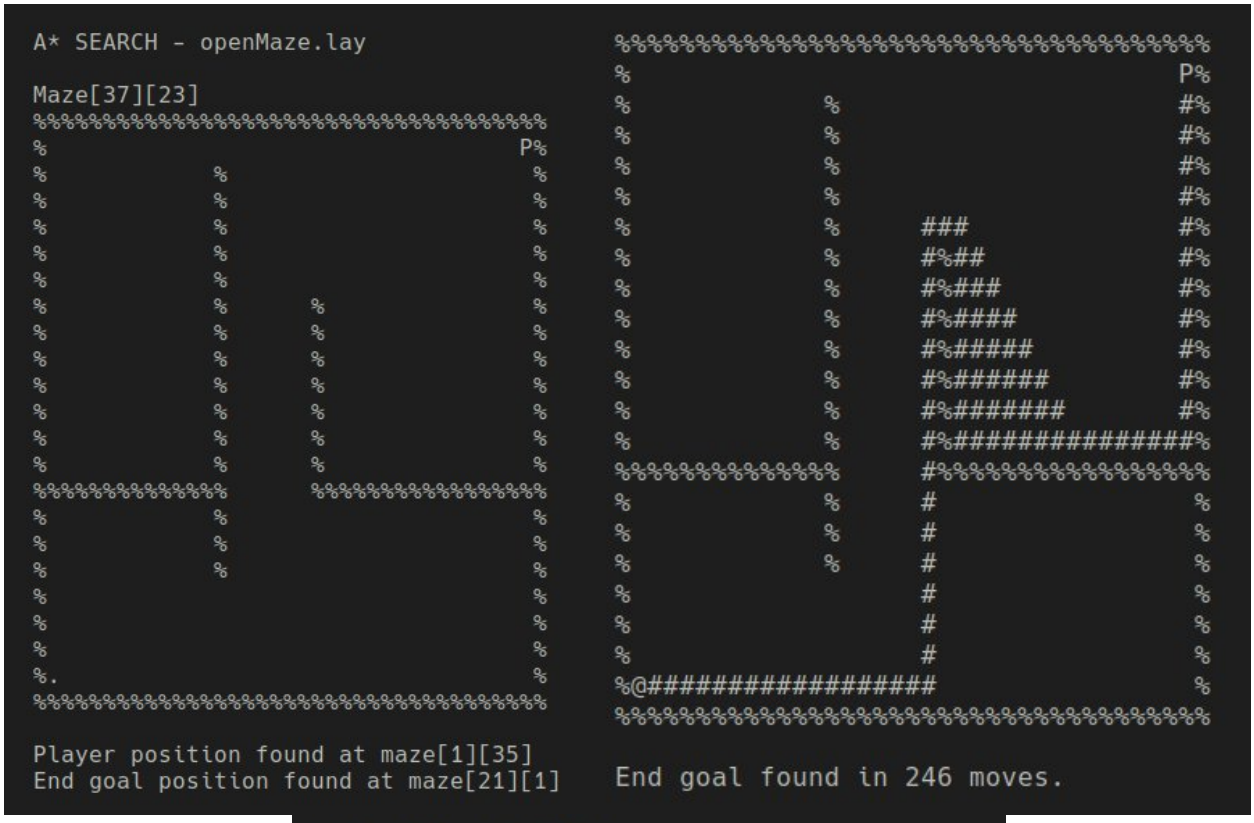
Open Maze

For breadth first search with the open maze, the operation takes so long that the code never completes. Because of how moves are added, the queue gets exponentially larger, with a

large amount of moves that have already been completed, as the moves were loaded into the queue before the searcher moved there. Because of this, breadth first search is impossible to use here, and would take a ridiculous amount of time to complete. Attached will be some pictures of the attempts to run the breadth first search. This particular shot is from printing the maze after every move, after running the program for a minute.



The A Star search was obviously much more successful.

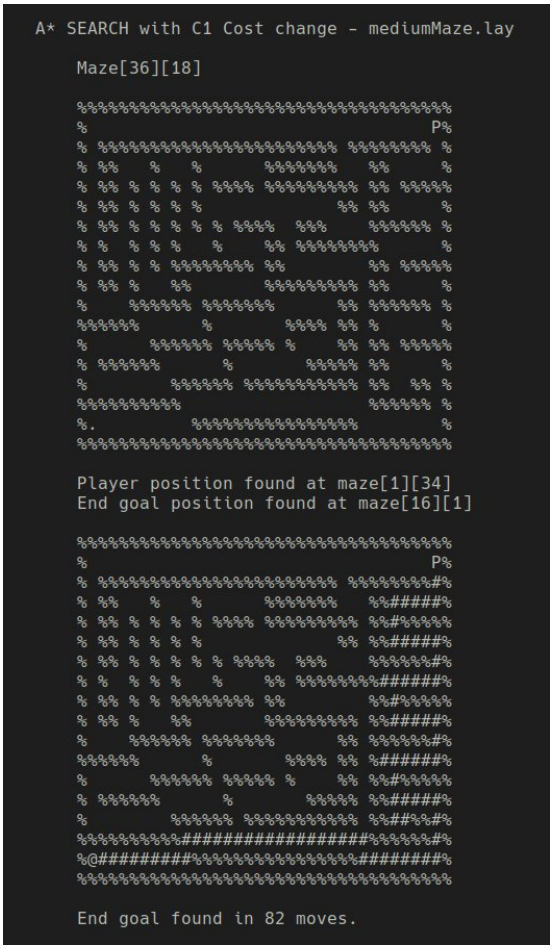


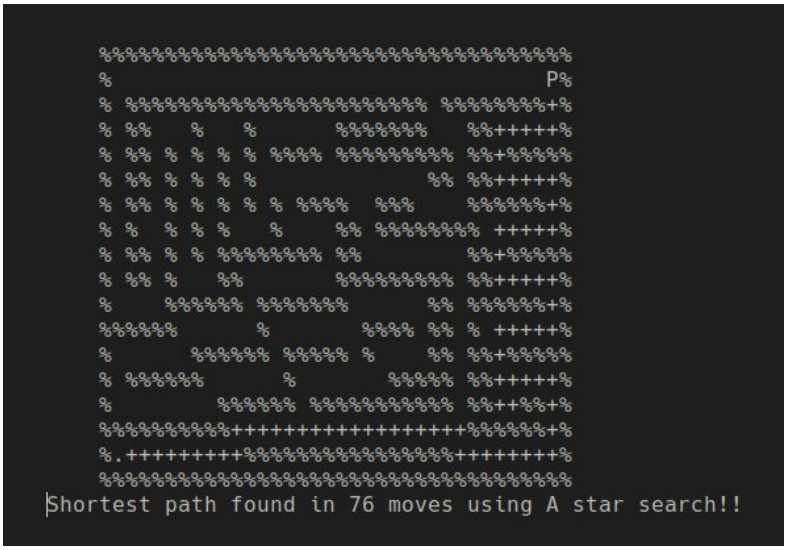
The A star search really shines here. The a star search is a must when you have much more open maps or a lot of different directions to go very often. As we can see it almost only travels on the optimum path, diverging every once in a while when there is a large wall to

overcome. While the breadth first search had an uncountable amount of moves, the end goal for our open search with a star was found in only 246 moves, and the optimal path was found to be only 68 moves.

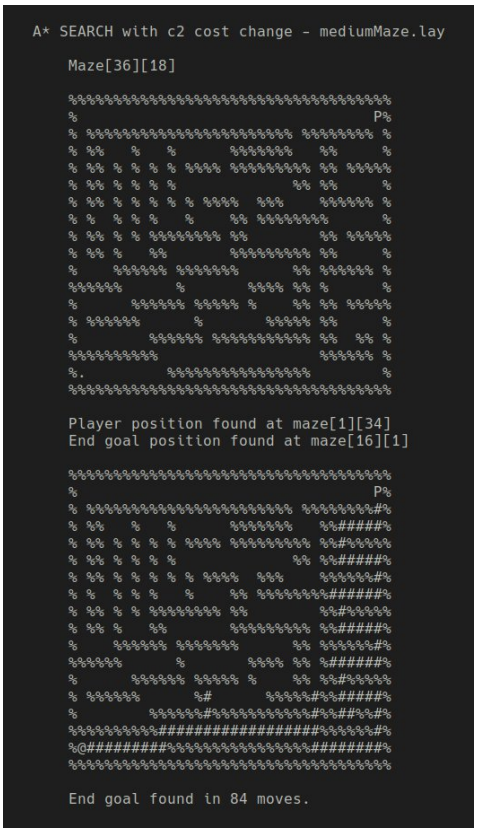
Part 2

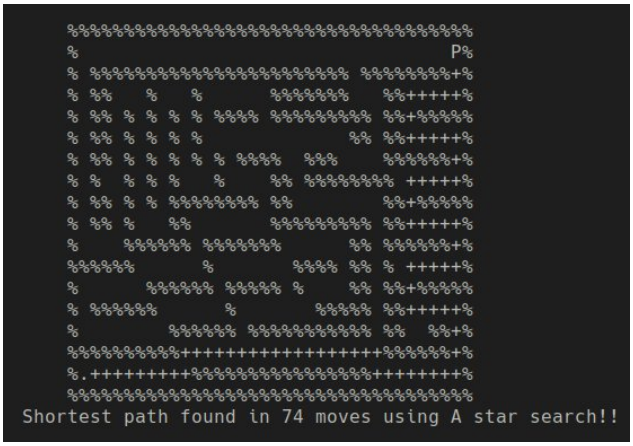
1. C1 = 1/2^x: This change to the cost definitely influenced the player to go to the right. It was even slightly less efficient because of this, but ultimately, the path does not look that different from the norm.





2. C2 = 2^x. This did cause the searcher to favor the right, but it did ultimately favor going down just the same. We can definitely see that it took more moves to find the endpoint though.





Part 3

For this “tricky” search I simply added a function to my endpoint that allowed me to check whether or not the searcher had reached the current endpoint, and if it had, it should re-check for endpoints in case there were any more and this worked perfectly. An added benefit was that sometimes it would hit the dots on the way to another dot, which allowed me to clear it simply by virtue of landing on it by design!

Gabriel Brown
CSC 412 Intro to AI

```
* SEARCH - trickySearch.lay

Maze[20][7]

%%%%%%%%%%%%%%%%%%%%
%.            ..%   %
%.%%.%%.%%.%%.%% % %
%         P       % %
%%%%%%%%%%%%%%%%%% %
%.....              %
%%%%%%%%%%%%%%%%%%%%

Player position found at maze[3][9]
End goal position found at maze[1][1]
End goal position found at maze[1][13]
End goal position found at maze[1][14]
End goal position found at maze[2][1]
End goal position found at maze[2][4]
End goal position found at maze[2][7]
End goal position found at maze[2][10]
End goal position found at maze[2][13]
End goal position found at maze[5][1]
End goal position found at maze[5][2]
End goal position found at maze[5][3]
End goal position found at maze[5][4]
End goal position found at maze[5][5]

%%%%%%%%%%%%%%%%%%%%
%################%###%
%#%%#%%#%%#%%#%%#%#%
%########P#######%#%
%%%%%%%%%%%%%%%%%%#%
%@#################%
%%%%%%%%%%%%%%%%%%%%

End goal found in 76 moves.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%
%.                ..%   %
%.%%.%%.%%.%%.%% % %
%    @###P          % %
%%%%%%%%%%%%%%%%%% %
%.....               %
%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
%.                ..%   %
%.%%.%%.%%.%%.%% % %
%   @####P         % %
%%%%%%%%%%%%%%%%%% %
%.....               %
%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
%.                ..%   %
%.%%@%%.%%.%%.%% % %
%   #####P         % %
%%%%%%%%%%%%%%%%%% %
%.....               %
%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
%.                ..%   %
%.%%#%%.%%.%%.%% % %
%  @#####P         % %
%%%%%%%%%%%%%%%%%% %
%.....               %
%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
%.                ..%   %
%.%%#%%@%%.%%.%% % %
%  ######P         % %
%%%%%%%%%%%%%%%%%% %
%.....               %
%%%%%%%%%%%%%%%%%%%%%%%%
```

IMPORTANT NOTE!!!!

All of this code can be seen perfectly just like this from within the homework. I specifically wrote it for this assignment and its good enough that it basically writes this whole report out. I urge you to take a look and please reach out to me if you have any issues because this should work flawlessly.