Vir Desai

720329873

Collaborators: None

HW1

1. Solving Mazes with Search

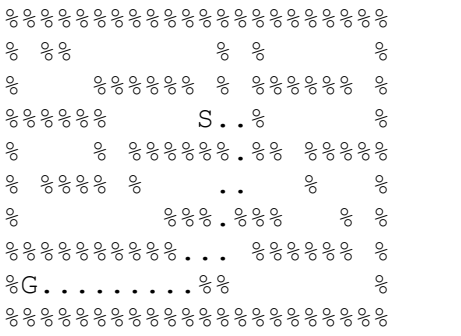
As mentioned in class and in the assignment document, the Depth-first, Bredth-first, Greedy best-first, and A* search algorithm are basically the same and only differ in how they manage the frontier. Depth-first does not have to deal with finding a better path to a state already on the frontier because it terminates once the goal is found on its path. Therefor the implementation of it in my code is fairly trivial with the frontier being a last-in-first-out data structure which has values added to it if any of the ordered locations (Up, Left, Right, Down) from the current node are open based on my parameters for open. If a different order of locations (sat Down, Up, Left, Right) is used when appending to the frontier, a different outcome for path cost and number of nodes explored results. The breadth-first search I used is also fairly trivial as it appears almost the same as the depth-first search with the addendum of the data structure being a first-in-first-out structure in this situation for the frontier.

The greedy best-first and A* search both use the first-in-first-out frontier data structure that was utilized in the breadth-first search. The handlings of the frontiers for both of these algorithms are not the same as for the depth-first and breadth-first ones. I created individual functions to handle the frontiers for both of these which essentially do the same thing but with one difference. As both of these algorithms are based off of the cost associated with the positions, my position nodes changed from two unit lists to three. The third is the cost associated with the location. The difference between the algorithms is that the cost for the greedy best-first search is just the Manhattan distance between the current location and goal location whereas the cost for the A* search is the current path cost of the node + the Manhattan distance from that position to the location. Both of these frontiers were managed by checking to see if the up, down, left, and right, directions were open in my parameters for appending that directional node to the frontier. The frontier would then be instantiated with more nodes and I used a cool sort function to get them in the order I wanted of the least cost node being at index 0 in the list (frontier.sort(key=lambda x:x[2])).

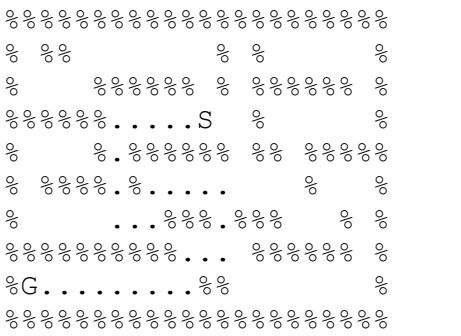
<u>Small Maze</u> → Optimal = 20

```
%%%%%%%%%%%%%%%%%%%%
% %%          % %       %
%     %%%%%% % %%%%%% %
%%%%%%      S..%       %
%     % %%%%%%.%% %%%%%
% %%%% %      ....%...%
%          %%% %%%...%.%
%%%%%%%%%%....%%%%%%.%
%G.........%%........%
%%%%%%%%%%%%%%%%%%%%
```

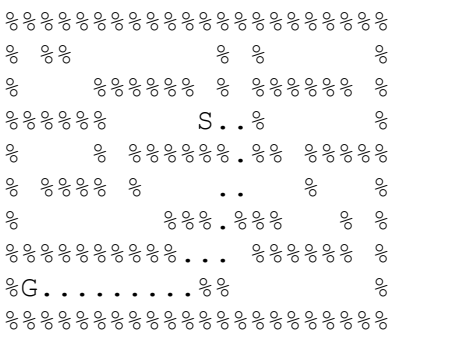
```
Depth First Search
Path Cost: 37
Number of Nodes: 38
```



Breadth First Search
Path Cost: 20
Number of Nodes: 171

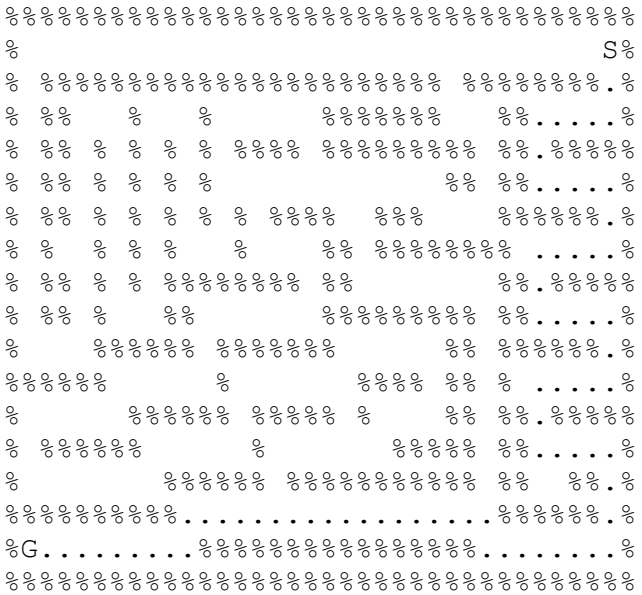


Greedy Best First Search
Path Cost: 29
Number of Nodes: 40

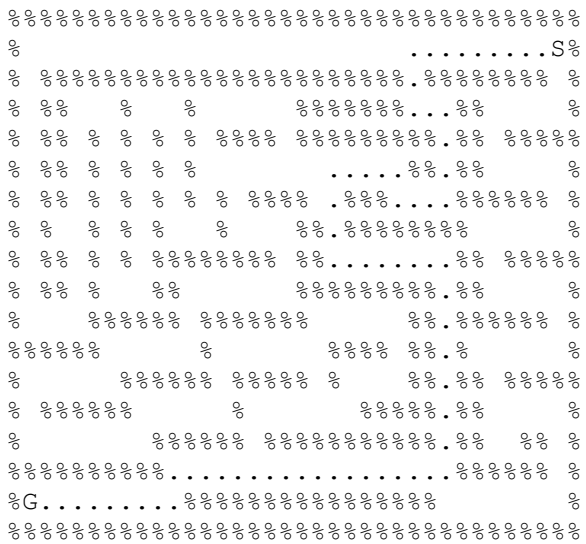


A* Search
Path Cost: 20
Number of Nodes: 130

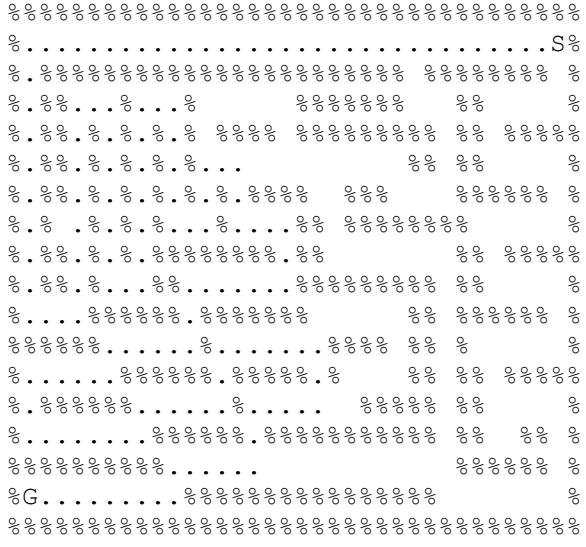
Medium Maze → Optimal = 68



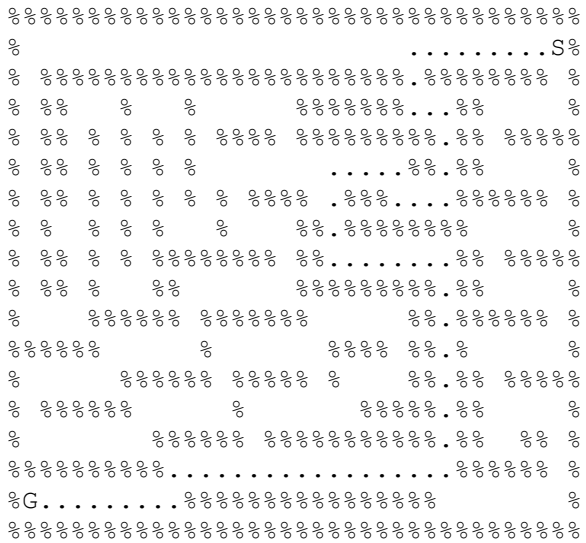
Depth First Search
Path Cost: 74
Number of Nodes: 78



Breadth First Search
Path Cost: 68
Number of Nodes: 395

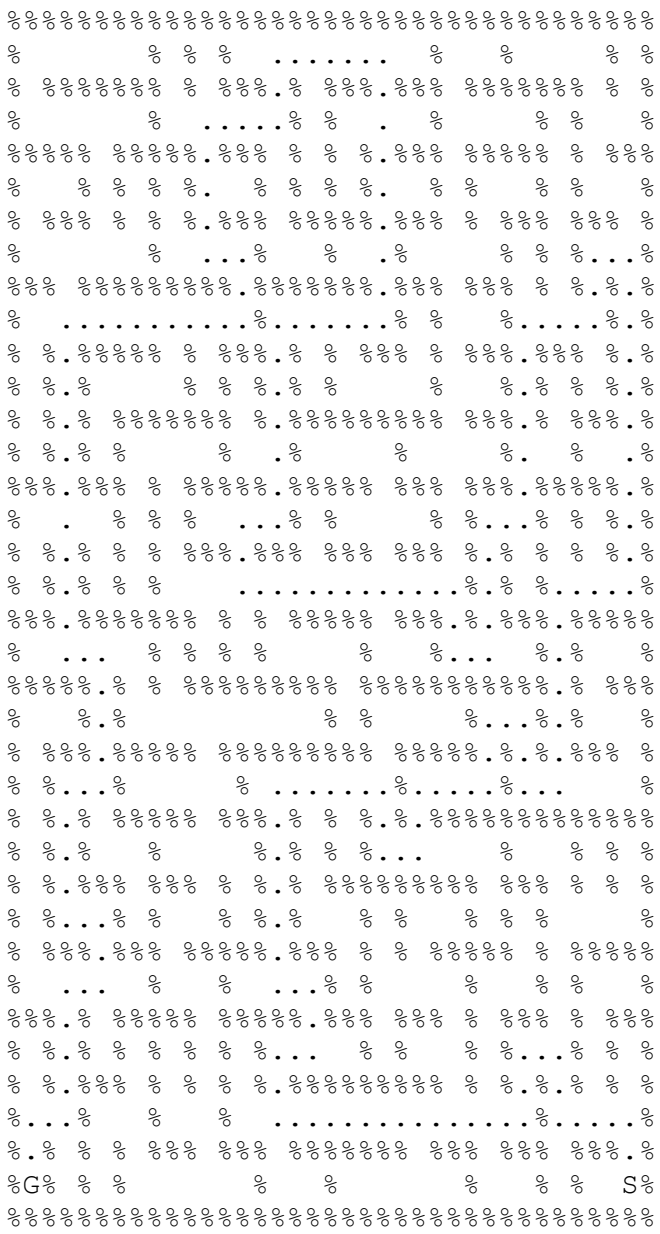


Greedy Best First Search
Path Cost: 152
Number of Nodes: 158

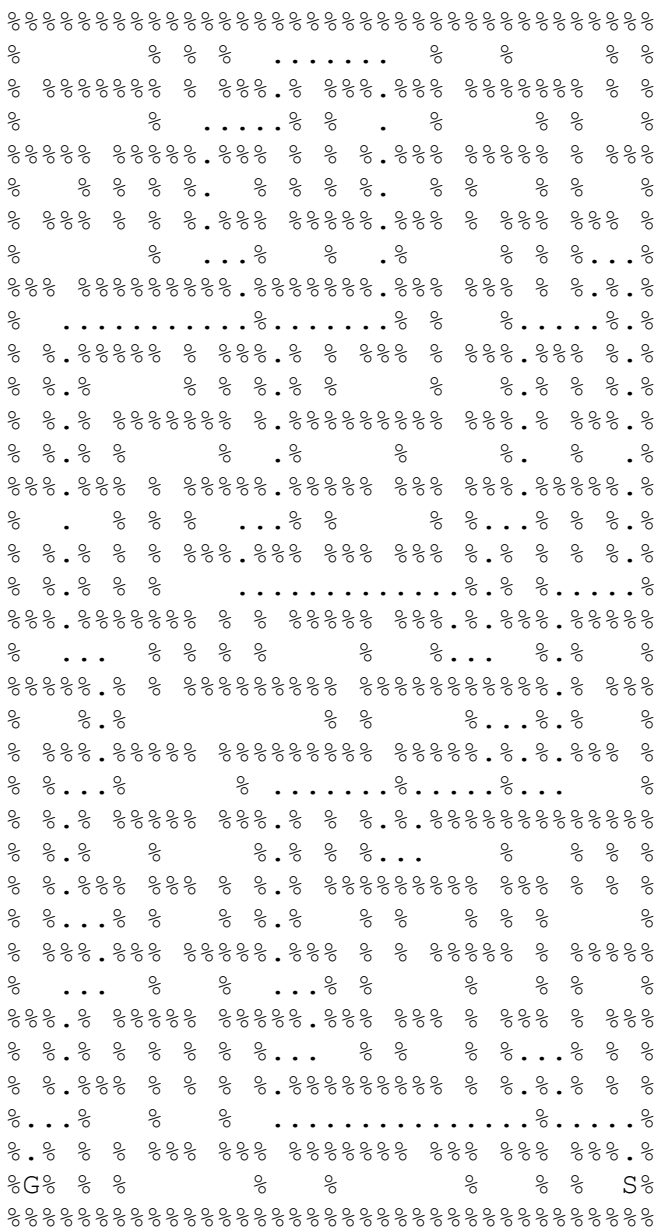


A* Search
Path Cost: 68
Number of Nodes: 125

Big Maze → Optimal = 211

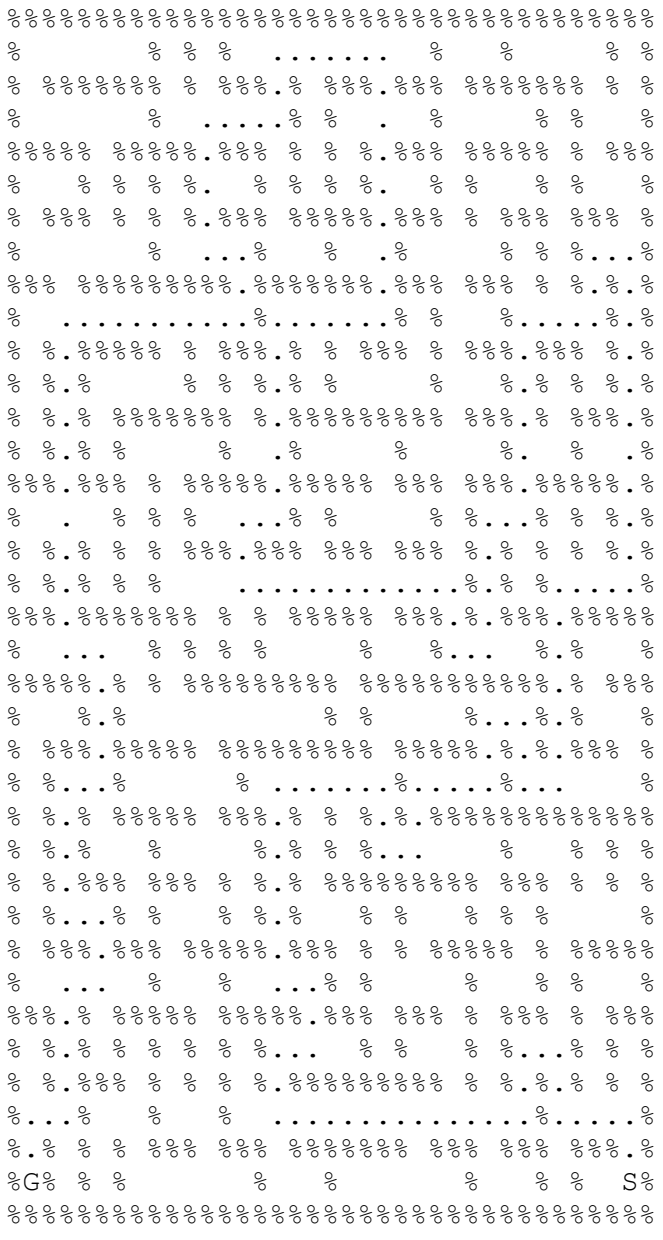


Depth First Search
Path Cost: 211
Number of Nodes: 365

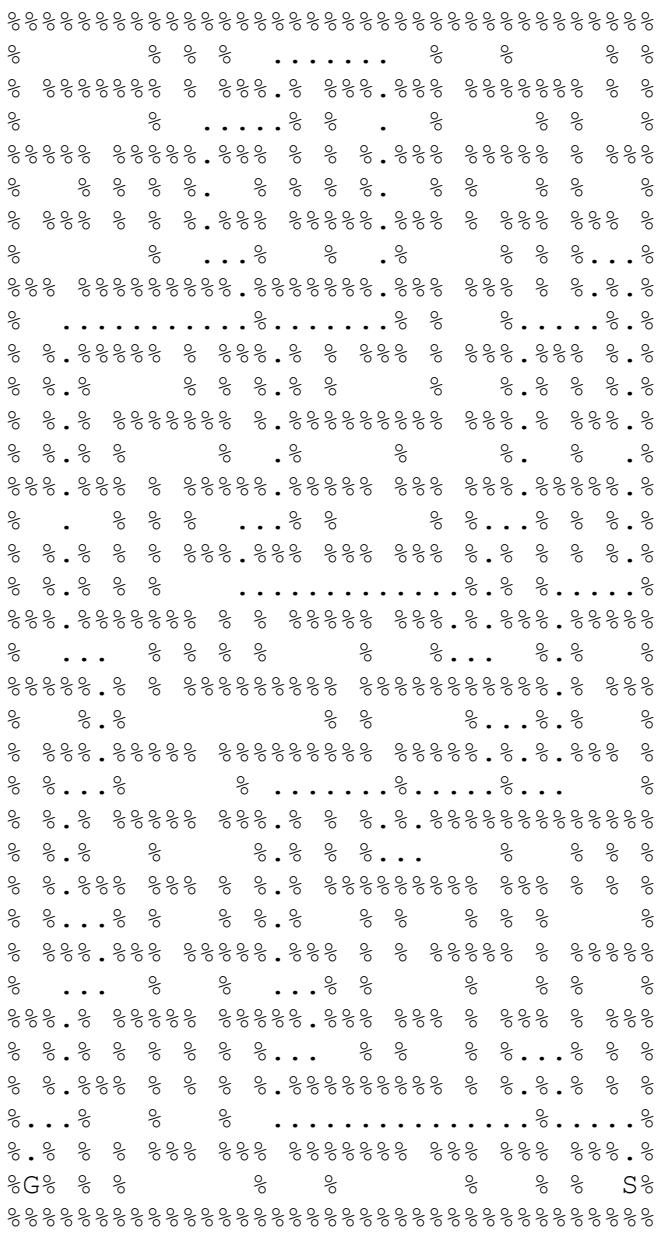


Breadth First Search
Path Cost: 210
Number of Nodes: 626

There was some issue here with the path cost because it is 1 less than the rest of the algorithms when it should be 211.



Greedy Best First Search
Path Cost: 211
Number of Nodes: 657



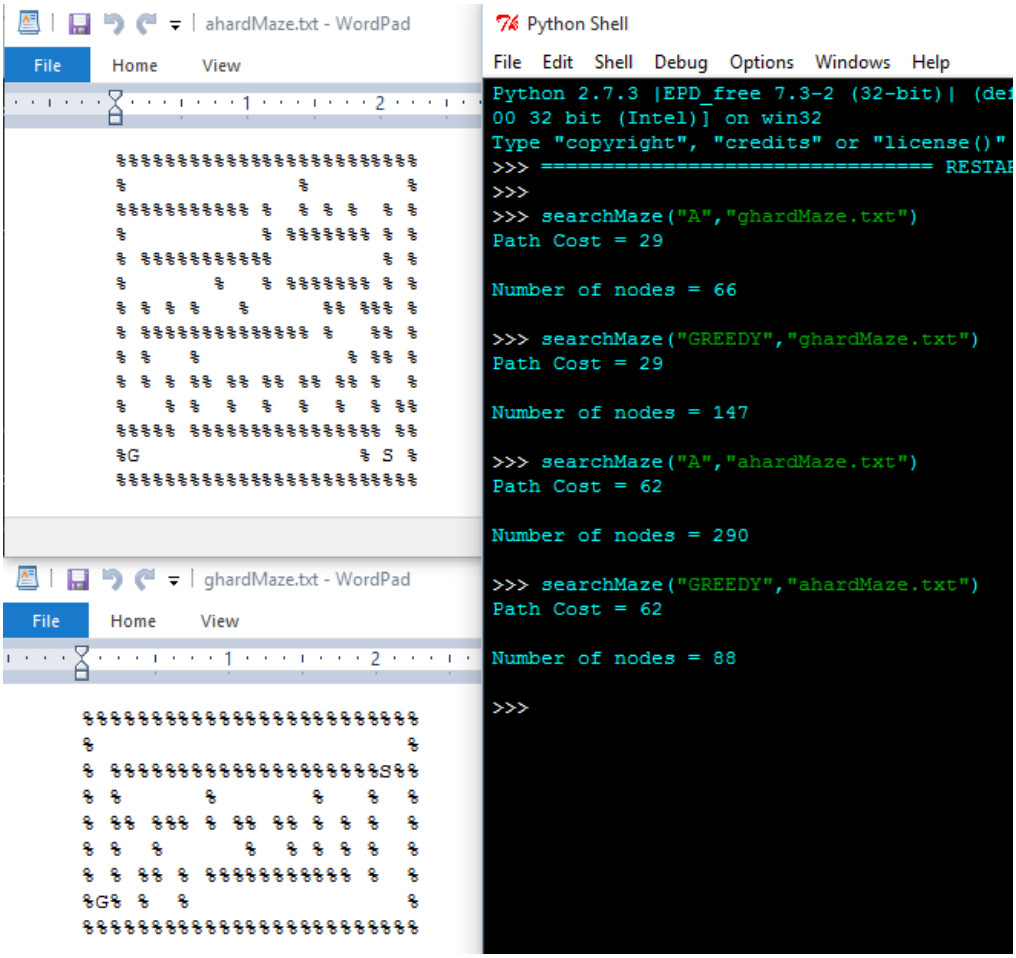
A* Search
Path Cost: 211
Number of Nodes: 1097

2. Designing “difficult mazes

Shown below are the mazes completed paths as created by my A* search and Greedy best-first search algorithms for the two different mazes. They are both the same in terms of the path which they took to get to the solution. The image below those mazes shows the console output on the two different mazes as well as the images of the blank maze documents.

```
%%%%%%%%%%%%%%%%%%%%%%
%                    %      ...%
%%%%%%%%%%%  %   %  %  %  .%.%
%                %  %%%%%%%.%.%
%  %%%%%%%%%%%.........%.%
%.......%...%.%%%%%%%  %.%
%.% %  %...%...    %% %%%.%
%.%%%%%%%%%%%%%  %    %%.%
%.%...%               % %%.%
%.%.%.%%  %%  %%  %%  %%..%
%...%.%   %   %   %   %  %.%%
%%%%%.%%%%%%%%%%%%%%%%.%%
%G....                 % S %
%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%
%.....................  %
%.%%%%%%%%%%%%%%%%%%S%%
%.%       %         %    %
%.%% %%%  %  %%  %%  %  %  %  %
%.%   %       %   %  %  %  %   %
%.%  %%  %  %%%%%%%%%%  %   %
%G%  %   %                  %
%%%%%%%%%%%%%%%%%%%%%%
```



The maze above on the left is the maze which is harder for the A* search algorithm (290 nodes explored compared to 88) and the maze on the right is harder for the Greedy Best-First algorithm (147 nodes explored compared to 66). The maze which is hard for the A* search algorithm is easy for the Greedy Best-First algorithm because the greedy algorithm is expanding the nodes in the frontier with just the lowest straight line Manhattan distance from the goal. The A* search on the other hand is adding that Manhattan distance with the distance that the particular nodes on the frontier have already traveled and it then sorts and picks the next frontier node with the lowest of that cost. This makes the greedy algorithm go straight for the goal node as there are not very many obstacles in the way after the beginning section but the A* algorithm searches through many of the intermediary crevices which are false paths. The other maze is the opposite as there are deep paths and crevices leading to a Manhattan distance very close to the goal node with the issue of a wall blocking the path. A* is taking into account that the correct first move up is only slightly less cost efficient that down at the start and moves on to that quickly as the downward false path rises in node cost but the upper route lowers.

3. Eating all the cheese

For this section I was able to get my code to run for the small, tricky, and medium cheese inputs. The medium cheese requires a bit of waiting but does get solved in a reasonable amount of time. The big input was taking too long so I decided to not wait for it to complete. My A* search for this part is changed in terms of instead of executing until the goal is found, as is done in part 1, I have a list of all the coordinates of the cheeses and the function terminates when all the elements in the goal list have been explored. Unlike the frontier in part 1 for this algorithm which I was sorting based on the Manhattan distance + the cost to get to that node, I implemented a recursive formula to find the shortest total path between the Manhattan distances of all the paths. My handling of this computed the Manhattan distance from the current location node a goal node and then to the closest goal node from that and recursively found the shortest path for that node. For each current location I was at, the Manhattan distance to a goal node was calculated and the shortest total Manhattan distance path was then computed to the remaining nodes from there and then the current location to the next goal node until the smallest total Manhattan distance path cost was found.

This heuristic I believe is admissible because the cost that is finally used after all that mess gets handled is guaranteed to be less than what the actual cost would be to reach all the cheese. My function added the coordinates above, below, to the left, and to the right, to the frontier if those positions were not walls or already explored. By already explored I mean that I had a tag on my nodes referencing the cost calculated to accomplish the goal and if the node attempting to be added to the frontier had the same coordinates and cost as already explored it would not be added. This check was for eliminating possible loops that the function could go into. Due to the way I calculated my cost for each node, the execution of this A* search does not explore many nodes. In fact, my code was returning that the path cost to get between all the nodes was greater than the number of nodes that were explored in my algorithm for the small cheese input. The path costs I believe that I was computing were not working for some reason in my algorithms as they were in the previous parts so I wrote another function to track the paths between the labeling systems which were made to come up with those numbers. My path cost for the small cheese maze was 47 and the number of nodes explored was 39, for the tricky cheese maze the cost was 72 and the number of nodes explored was 106, and for the medium cheese maze the cost was

242 and the number of nodes explored was 259. Obviously from my deduction I noticed that these are not optimal solutions but only slightly not for the small and tricky mazes. Below is the console output of the cheese inputs as well as the text file outputs that were produced.

```
>>> searchMaze("CHEESE","smallcheese.txt")
Path Cost = 47

Number of nodes = 39

>>> searchMaze("CHEESE","trickycheese.txt")
Path Cost = 72

Number of nodes = 106

>>> ================================ RESTARI
>>>
>>> searchMaze("CHEESE","mediumcheese.txt")
Path Cost = 242

Number of nodes = 259
```

Small Cheese

```
%%%%%%%%%%%%%%%%%%%
%g           654S 1%
%h%%f%%d%%9%%7%% %2%
% %% %ecba8      %3%
%%%%%%%%%%%%%%%%%%%
```

Tricky Cheese

```
%%%%%%%%%%%%%%%%%%%
%2           56%   %
%3%%1%%4%%8%%7%% % %
%        S       % %
%%%%%%%%%%%%%%%%% %
%dcba9             %
%%%%%%%%%%%%%%%%%%%
```

Medium Cheese

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ponmlkjihgfe%%%%%----------%
%%%q%--z%%%-dcb-----%-%---%-%%%
%--r%%%y%-%%%%a%-%%%%%-%%%---%
%-%stuwx%--6789%------%-----%-%
%-%%%v%%%%%5%%%%%%-%%%-%-%%%%%
%-----%----4321S----%---%-----%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```