Artificial Intelligence Project: Maze Search

Authors:
Logan Bonney
Cody Stoner

Implementation:

This program was written in Python consisting of multiple parts. We decided to have a class mazes that just strictly dealt with reading the maze from the provided files and creating a node for each location on the maze. The next class we implemented was Search. This holds all of the code for doing BFS, DFS, A*, and Greedy algorithms. Within this class we used the node class to be able to traverse in given ways of each search algorithm. All the algorithms handle these nodes in similar ways, with difference mainly being the data structure used to organize them. Each search algorithm uses essentially the following pseudocode.

```
1   vistedNodes = []
2   while(queue.length > 0):
3       node = queue.next()
4       vistedNodes.append(node)

5       If(node.value == '*'):
6           print("Goal found")
7           Return true

8       for neighbor in node.neighbors:
9           If(vistedNode.contains(neighbor) == False and neighbor.value != '%'):
10              neighbor.previous = node
11              queue.add(neighbor)
```

There is also code to keep track of the number of nodes expanded small differences between algorithms. After the goal is found the path is found by following neighbor.previous back to the start.

Depth-First Search (DFS)

DFS adds neighbors to a stack whenever it moves to a new space. The nodes are then removed and added until the goal is found or there are no more nodes to search. DFS performed the best in terms of cost on the open maze but the final solution took many steps more than any others.

Breadth-First Search (BFS)

As seen BFS and DFS are very similar, but BFS uses a queue instead of a stack. So then we just had the top node dequeued from the que then all nodes adjacent nodes, that

haven't been visited. This was just repeated until the que was empty. So once the finish node was found then the que was empty.

Greedy Best-First Search (GREEDY)

Greedy search is different from BFS and DFS due to the knowledge of the goal state. When a node is expanded then it will expand to a node in the frontier. This is calculated using the manhattan distance from the given node in the frontier to the node at the goal state. We implemented this using a built in heap data structure with python (heapq). Using this allows a node to be added to frontier, then the node is added to the queue. As more nodes get added on then the first nodes in the queue are popped off. This is repeated until the goal state is found.

A* (ASTAR)

A* uses a min heap sorted by the distance to the goal added the distance traveled to get to the current position. This way it is always looking for the path with the least total expected cost. This algorithm performed fairly well in most cases but there is a bug that causes it to expand more nodes than should be possible in open spaces.

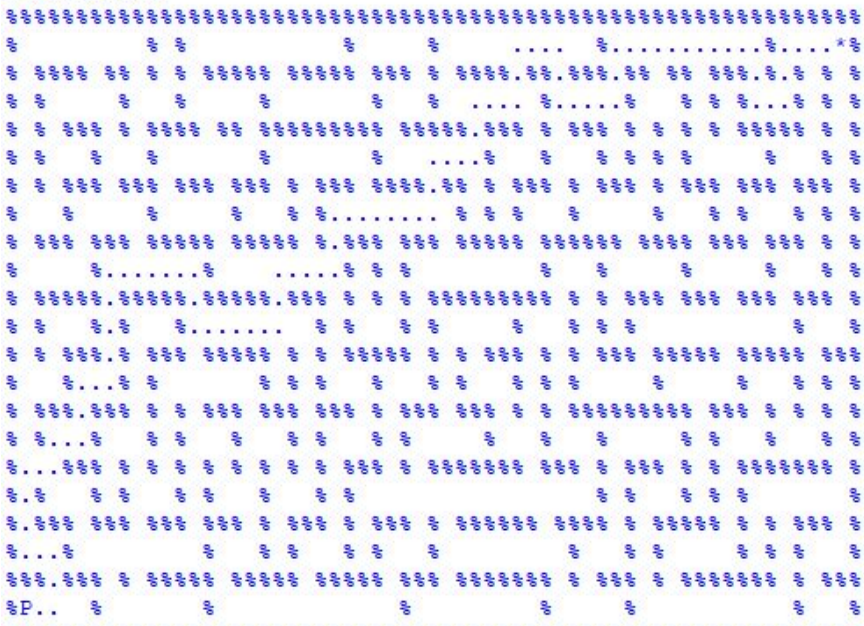
Contributions:

Logan was responsible for both the DFS and the A* search algorithms. Cody was responsible for both the BFS and the Greedy search algorithms. Both of us implemented various other methods such as node and read maze files so that we could further ease the implementations of the different searching methods.

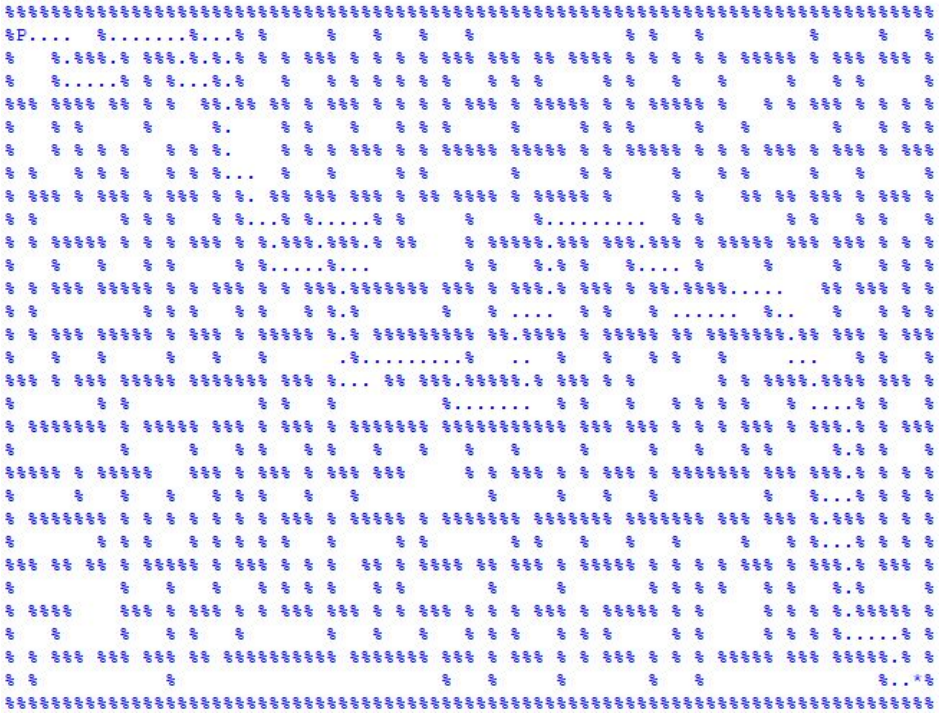
Outputs:

```
Breadth First Search:
Cost: 509, Steps 43
Solved Maze:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    %P              %
%                    %.              %
%                    %.              %
%                    %.              %
%                    %......        %
%                    %%%%%%.        %
%                         %.        %
%                         %.        %
%                         %.        %
%                         %.        %
%       ..................%.        %
%       .%%%%%%%%%%%%     ...        %
%       .%                           %
%       .%                           %
%       .%                           %
%       .%                           %
%       .%                           %
%       *%                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

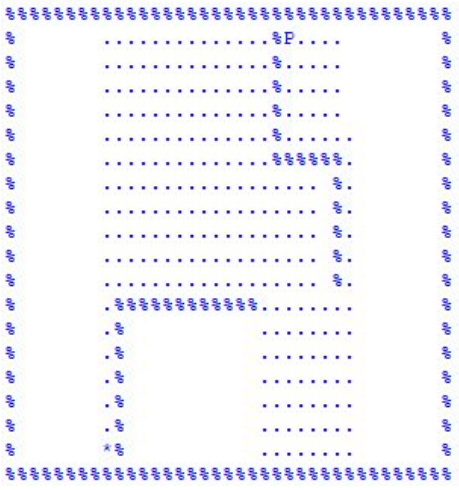
Breadth First Search:
Cost: 609, Steps 93
Solved Maze:



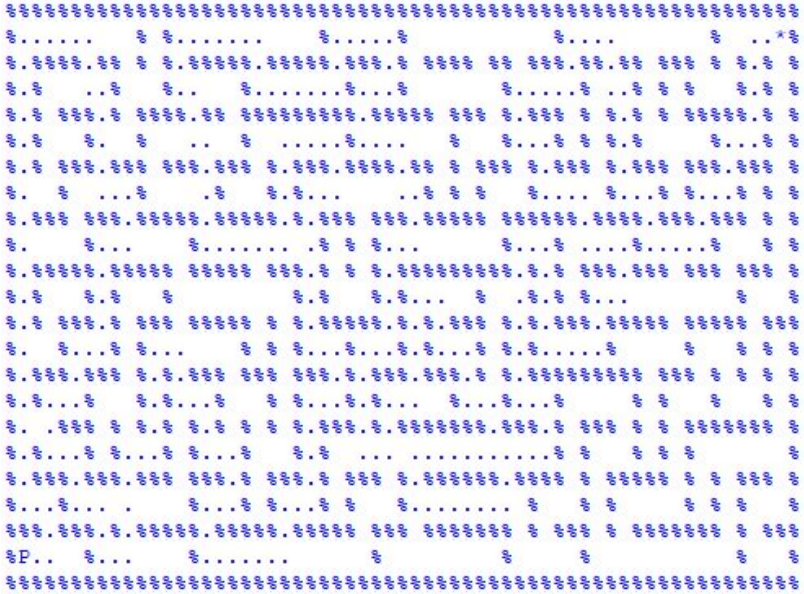
Breadth First Search:
Cost: 1258, Steps 147
Solved Maze:



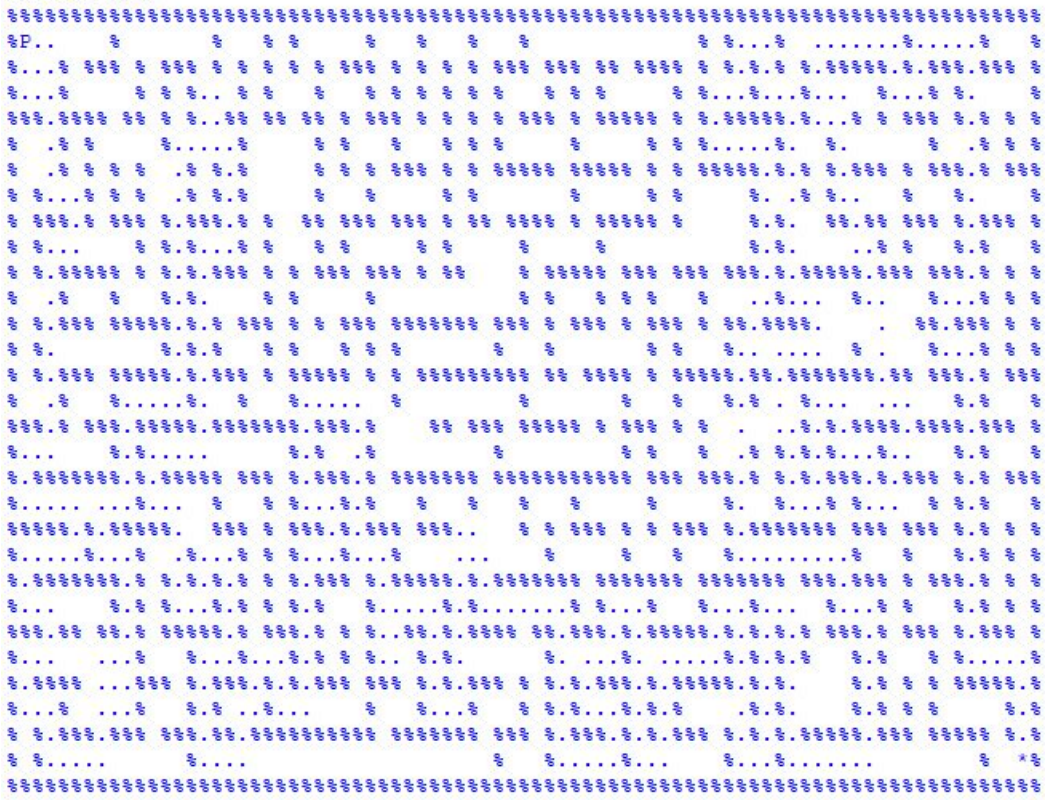
Depth First Search:
Cost: 268, Steps 267
Solved Maze:



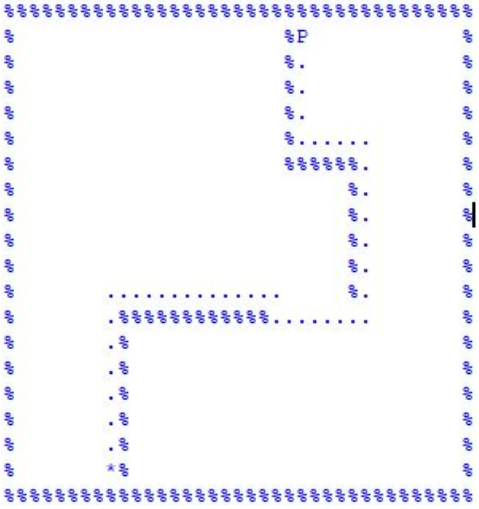
Depth First Search:
Cost: 523, Steps 309
Solved Maze:



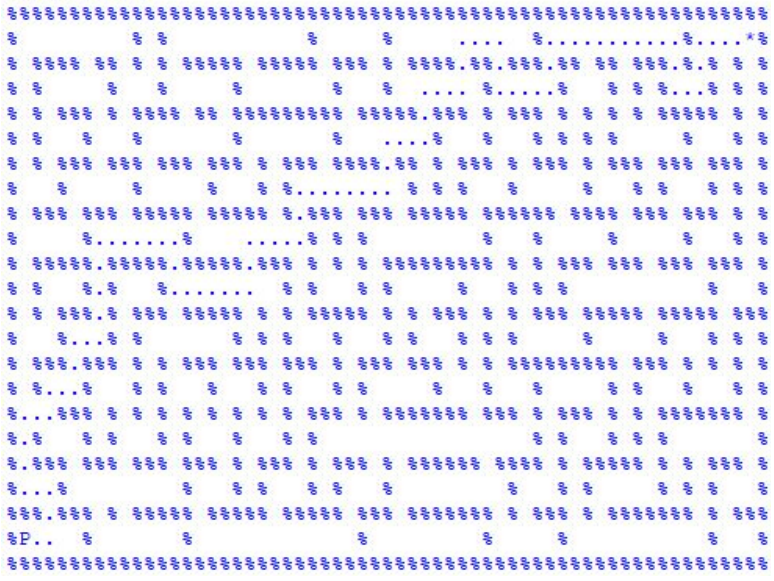
Depth First Search:
Cost: 1129, Steps 461
Solved Maze:



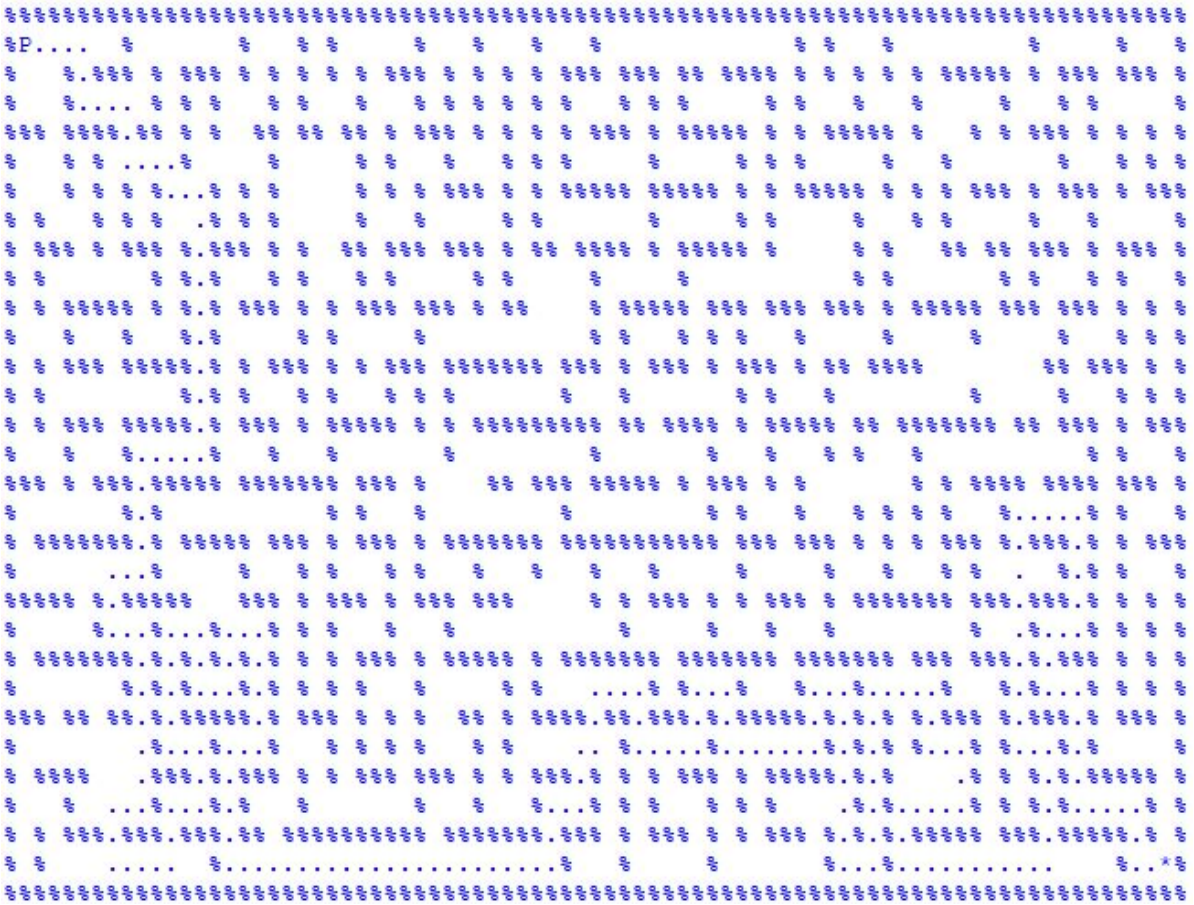
Greedy Search:
Cost: 14630, Steps 43
Solved Maze:



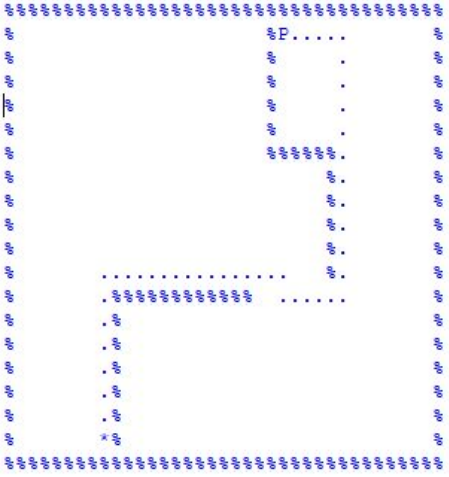
Greedy Search:
Cost: 102, Steps 93
Solved Maze:



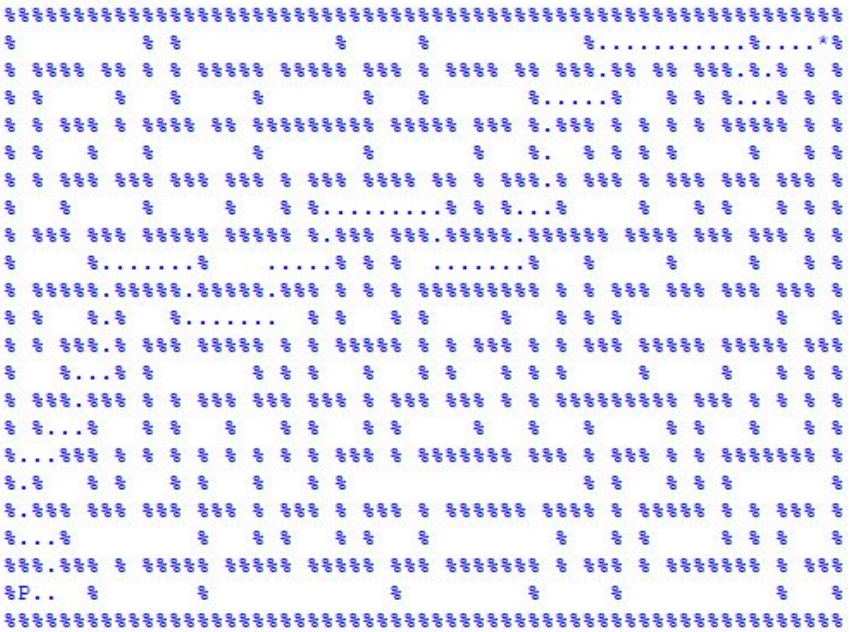
Greedy Search:
Cost: 325, Steps 221
Solved Maze:



A* Search:
Cost: 84615, Steps 43
Solved Maze:



A* Search:
Cost: 354, Steps 93
Solved Maze:



A* Search:
Cost: 2521, Steps 147
Solved Maze: