Artificial Intelligence: Search Algorithms

Carsen Ball
Brett Layman

October 8, 2018

Abstract

In this paper we compare four search algorithms: Depth First Search (DFS), Breadth First Search (DFS), Greedy Best, and A*. We discuss the significant design and implementation decisions that were made, and their potential effects on the results. We ran each algorithm on 3 different mazes, recording the length of the path taken to the solution, and the number of Nodes (spaces) that were visited (expanded). A* appears to perform the best of the algorithms that find the optimal solution (A* and BFS), while Greedy Best appears to perform the best of the algorithms that aren't guaranteed to find the optimal solution (Greedy Best and DFS).

1 Software Architecture

1.1 Overall structure

For this project, 4 algorithms were implemented, and tested on 3 different mazes. We used Python for our programming language. To represent the maze, we read in a text file, and stored it in a two dimensional array. For each item in the array that did not represent a wall, an instance of the `Node` class was created. The `Node` class is used to represent a position on the map. Upon initialization, each Node stores the character, which could be a ., *, or P and the x and y coordinates of the position. All Nodes are initialized with a visited value of False. Whether or not the Node is the start Node is represented by a Boolean value. When performing Greedy Best and A* searches, the Nodes also have the cost so far, a compare value, and a compare function, which allows the algorithm to compare two nodes based on a heuristic. For A*, the start Node's cost so far value is set to 0. These additional details for the heuristic based searches were added by having a separate subclass for Nodes called: `HeuristicNode`

The `SearchAlg` class acts as a driver for the program, and performs all of the necessary set up before initiating the search. At run time, the map is initialized as previously explained. After each position on the map is represented as a Node, the Nodes are connected to their neighbors using their x and y coordinates. When performing Greedy Best and A*, the distance to the goal

is also calculated and stored in the Node. This distance is calculated using Manhattan distance. The frontier is initialized, and the starting node is added to the frontier. Then the search is started by calling the constructor of the `Search` class, passing in the frontier.

The `Search` class encapsulates much of the functionality needed for the searches. This class stores and manipulates the frontier, which is represented as a stack for DFS, a queue for BFS, and a Priority Queue for Greedy Best and A*. A subclass of the `Search` class was created for each type of search, and each subclass contains different `add` and `remove` methods for manipulating the frontier. The `Search` class also contains variables for counting the number of nodes expanded (visited) and the length of the resulting path.

1.2 Algorithm Details

All four algorithms use a frontier to decide the order in which nodes are expanded, but whereas DFS and BFS use a simple stack and queue for the frontier respectively, the Greedy Best and A* algorithms rely on a heuristic to set the priority of nodes in the frontier. For Greedy Best, the heuristic is the Manhattan distance to the goal. For A*, the heuristic is the cost of the path taken to get to the current node, plus the Mannhattan distance to the goal. To keep track of this measure, when a Node is added to the frontier, the cost so far of the Node being added is updated to be one plus the cost so far of the Node expanded from.

When a Node is removed from the frontier, the node is expanded, meaning that its neighbors are added to the frontier. The order in which neighbors are added is: left, right, top, bottom. This means that when the frontier is a stack (as in DFS), they are removed in the order: bottom, top, right, left. A goal check is performed when nodes are added to the frontier. This prevents goal states from becoming hidden in the frontier and not being discovered until they are removed. The process of expanding frontier Nodes, and adding new nodes to the frontier is continued until the goal state is found.

We were able to trace a path back from the goal to the start by using references. Each Node contains a reference to where it was expanded from, thus creating a linked list of the path taken to that node.

One key aspect of A* that was unique from the other algorithms, was it's use of Tree search. DFS, BFS and Greedy Best all used Graph search to avoid re-visiting explored nodes. This became problematic with A*, because it preventing optimal solutions from replacing sub-optimal ones. So Tree search was used to allow for re-exploration of neighbor Nodes. When a Node is revisited from a different parent, the path from that parent is compared to it's previous parent, and if the cost is less, the true cost is updated and references are changed to follow the optimal solution.

2 Results

In this section we explore the performance of each algorithm on three mazes. When comparing the algorithms, we used both efficiency: number of nodes expanded, and optimality: did the algorithm find the shortest path.

Greedy Best was the most efficient for every maze, expanding much less nodes in many cases, but it only found the optimal solution in 2 of the 3 mazes, so proved to be somewhat unreliable in that regard.

A* was always optimal (as theoretically expected), and was more efficient than BFS and DFS for 2 of the 3 mazes. However, it's efficiency lagged behind Greedy best. In terms of being guaranteed to find the optimal solution, and doing so with reasonable efficiency, A* appears to be a good choice.

DFS never found the optimal solution, but was more efficient than A* in one case, and more efficient than BFS for all mazes. BFS was optimal in all cases (as expected), but also had the worst efficiency in all cases.

In conclusion, we recommend using A* when guaranteeing optimality is important, and Greedy Best when efficiency is more important than optimality.

We provide the results of the algorithms by showing the path taken for each algorithm on the three mazes, as well as bar charts comparing efficiency and path lengths of each algorithm. In each maze image the path taken is represented by a "P", with the start represented as an "S" and the end as a "*". Each node that has been expanded is represented by a "." .

3 Individual Contributions

3.1 Carsen

Brett started programming the assignment by reading in the maze and representing the maze with Nodes. He also programmed DFS and BFS. I then extended some of the code Brett wrote to write A* and Greedy Best. We both helped each other debug the search algorithms. Because Brett did a lot of the initial heavy lifting, I produced the results.txt file, created the result graphs and screen shots as well. I wrote a base line report, Brett made changes and amendments and we both proof read the report.

The biggest challenge was getting A* to find the optimal path. Figuring out we needed to revisit nodes fixed the issue. I think it is interesting that Breadth first, being a very simple algorithm found the shortest path along with A*.

3.2 Brett

I wrote the code to represent the maze as a grid of Nodes connected to their neighbors. I also wrote the code for DFS and BFS, and helped to convert A* from Graph search to Tree search. I proof read and added some details to the paper.

Like Carsen, I also found it challenging to get A* to find the optimal solution. We discussed this issue several times, trying to figure out where the algorithm

was going wrong. Finally, after working through some hypothetical situations, we came up with a solution, which was using Tree Search for A*. Another challenge was trying to understand the difference between expanded nodes and frontier nodes, and how each can effect the path taken in different ways. At first, it was difficult to understand the path that DFS took in the open maze, but after considering the frontier nodes that it was avoiding based on Graph search, the path made more sense.

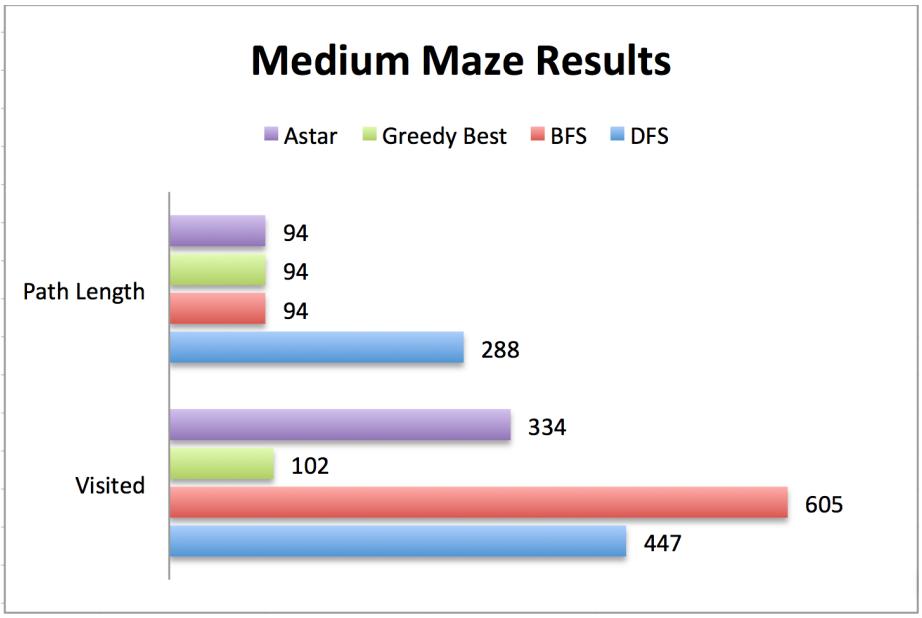

Figure 1: Medium Maze Results

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%PPPPPP    % %PPPPPPPP     %PPPPP%...........%PPPP         %     *%
%P%%%%P%% % %P%%%%%P%%%%%P%%%P%.%%%%.%%.%%%P%%P%% %%% % % %P%
%P%    PP%    %PPPPP%PPPPPPP%PPP%.......%PPPPP% PP% % %    % %P%
%P% %%%P% %%%%.%%P%%%%%%%%%P%%%%%.%%%.%P%%%.%.%P% % %%%%% %P%
%P%    %P  %     PPP%  PPPPP%PPPP ..%...%P .%.%.%P%      %   %P%
%P% %%%P%%% %%%P%%% %P%%%P%%%%P%%.%.%%%P%.%%%.%P%%% %%% %%%P%
%P  %  PPP%     P%   %P%PPP     PP%.%.%PPP%.....%PPP% %   %.%P%
%P%%% %%%P%%%%%P%%%%%P%P%%% %%%P%%%%%P%%%%%%.%%%%P%%% %%%.%P%
%P     %. PPPPP%PPPPPPP P% % %  PPPPPPP%PPP%.....%PPP  %...%P%
%P%%%%%.%%%%%P%%%%%.%%%P% % % %%%%%%%%%P%P%.%%%.%%%P%%%.%%%P%
%P%    %.%...%P .......%P%    % %      %  P%P%.%PPPPPPP....%PPP%
%P% %%%.%.%%%P%%%%%.%.%P%%%% % % %%% %P%P%%%P%%%%%.%%%%%P%%%
%P  %...%.%. P ...%.%.%PPP%    % %    % %P%PPPPP%.....%PPP%P% %
%P%%%.%%%.%.%P%%%.%%%.%%%P% %%% %%% % %P%%%%%%%%%.%%%P%P%P% %
%P%...%...%.%PPP%...%.%PPP% %      %   %PPP%PPPPP%.%PPP%PPP% %
%P .%%%.%.%.%.%P% %.%.%P%%% % %%%%%%% %%%P%P%%%P%.%P%%%%%%% %
%P%...%.%...%.%PPP%...%P%                PPPP%P%PPP%.%P%       %
%P%%%.%%%.%%%.%%%P%.%%%P% %%% % %%%%%%P%%%%P%P%%%%%P% % %%% %
%PPP%.........%PPP%.%PPP% %   %        PP%PPP%P%PPPPP% % %   %
%%%P%%%.%.%%%%%P%%%%%P%%%% %%% %%%%%%%P%P%%%P%P%%%%%%% % %%%
%SPP  %.......%PPPPPPP       %          %PPP  %PPP         %   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 2: Path Taken By Depth First Search On Medium Maze

Figure 3: Path Taken By Breadth First Search On Medium Maze

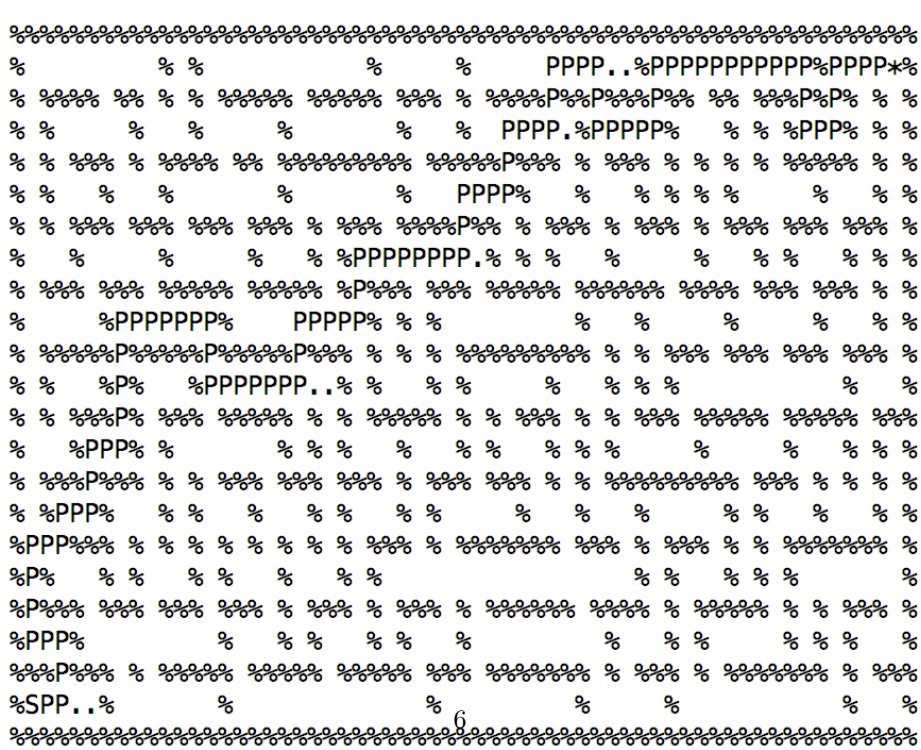

Figure 4: Path Taken By Greedy Best First Search On Medium Maze

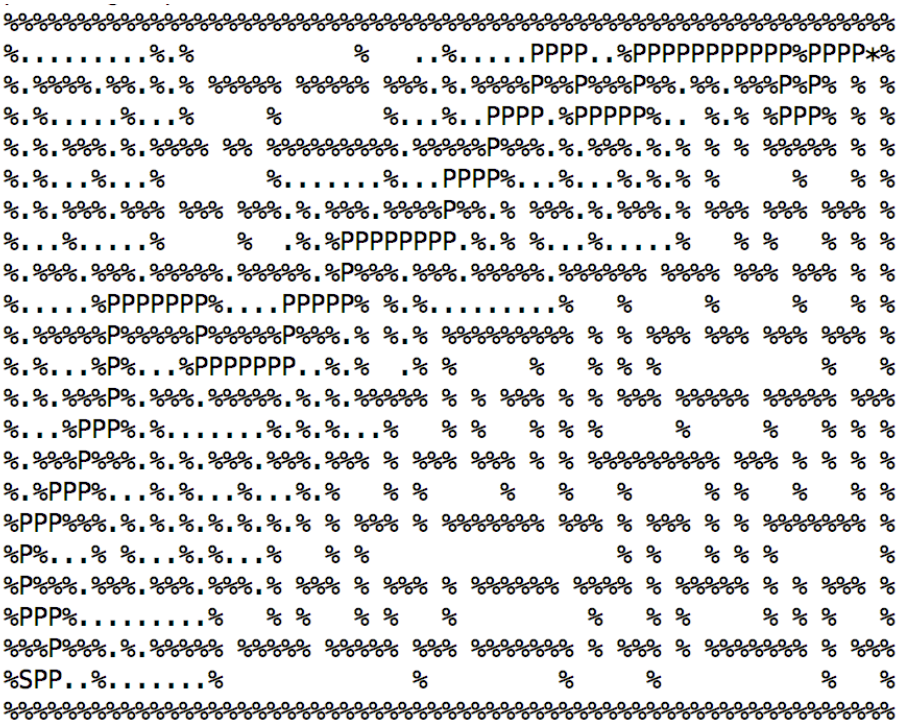

Figure 5: Path Taken By AStar Search On Medium Maze

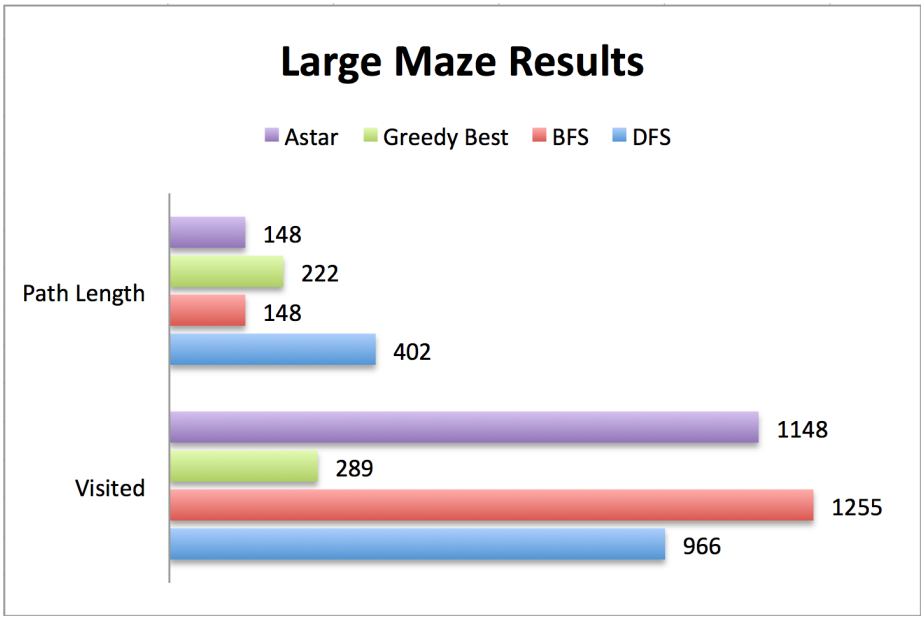

Figure 6: Large Maze Results

Figure 7: Path Taken By Depth First Search On Large Maze

Figure 8: Path Taken By Breadth First Search On Large Maze

Figure 9: Path Taken By Greedy Best On Large Maze

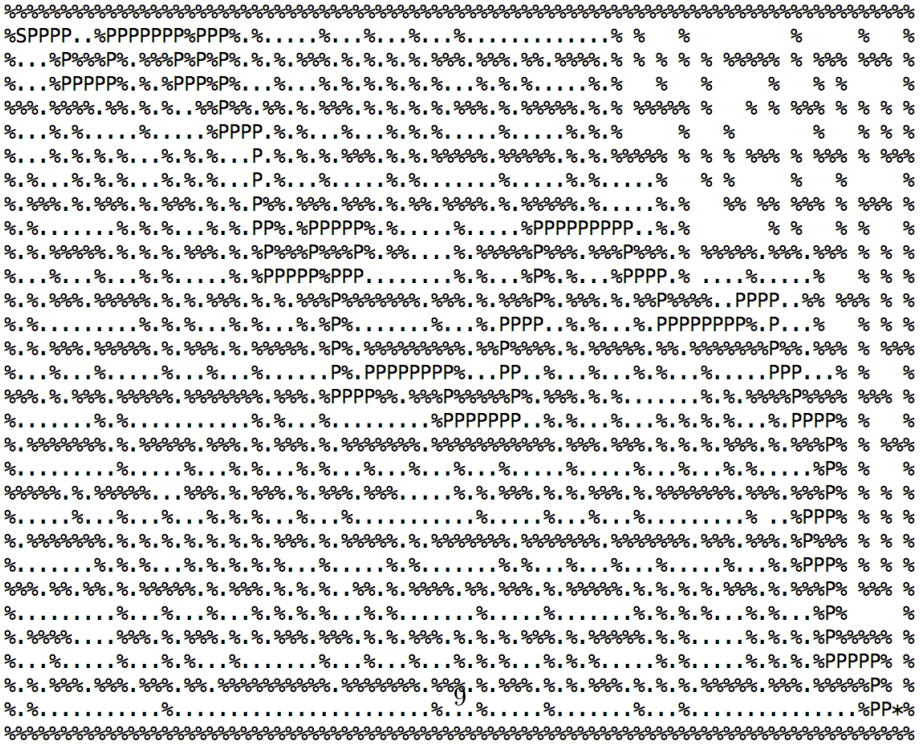

Figure 10: Path Taken By AStar On Large Maze

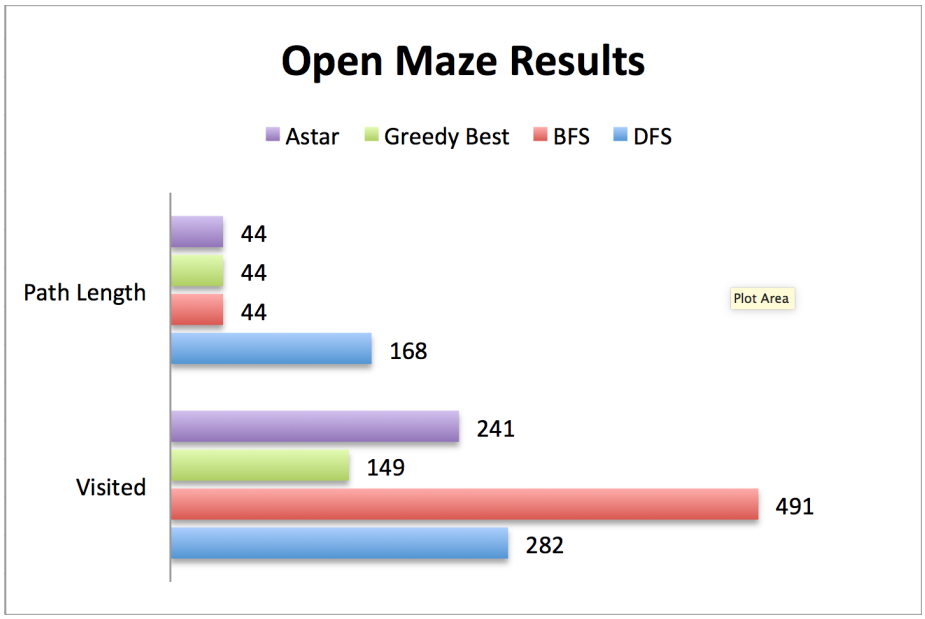

Figure 11: Open Maze Results

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      PPP PPP PPP PPP%S PPP .......%
%      P P P P P P P P%P P P .......%
%      P P P P P P P P%P P P .......%
%      P P P P P P P P%P P P .......%
%      P P P P P P P P%PPP PP ......%
%      P P P P P P P P%%%%%%P ......%
%      P P P P P P P PP PPP%P ......%
%      P P P P P P P  P P P%P ......%
%      P P P P P P P  P P P%P ......%
%      P P P P P P P  P P P%P ......%
%      P PPP PPP PPP. P P P%P ......%
%      P %%%%%%%%%%%% P P P P ......%
%      P %            P P P P ......%
%      P %            P P P P ......%
%      P %            P P P P ......%
%      P %            P P P P ......%
%      P %            P P P P ......%
%      P*%            PPP PPP.......%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 12: Path Taken By Depth First Search On Large Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            .........%SPPPPP.......%
%           ..........%.....P.......%
%          ...........%.....P.......%
%         ............%.....P.......%
%       ..............%.....P.......%
%      ...............%%%%%%P.......%
%     .....................%P.......%
%    ......................%P.......%
%   .......................%P.......%
%  ........................%P.......%
% ......PPPPPPPPPPPPPP.....%P.......%
%  .....P%%%%%%%%%%%%PPPPPPPP.......%
%   ....P%..........................%
%    ...P%..........................%
%     ..P%..........................%
%      .P%..........................%
%       P%..........................%
%       *%..........................%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 13: Path Taken By Breadth First Search On Large Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     %S            %
%                     %P..          %
%                     %P..          %
%                     %P...         %
%                     %PPPPPP       %
%                     %%%%%%P       %
%                          %P       %
%                          %P       %
%                          %P       %
%                          %P       %
%       PPPPPPPPPPPPPP     %P       %
%       P%%%%%%%%%%%%PPPPPPPP       %
%       P%..............            %
%       P%...............           %
%       P%................          %
%       P%................          %
%       P%..................        %
%       *%...................       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 14: Path Taken By Greedy Best On Large Maze

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     %S......       %
%                     %P......       %
%                     %P......       %
%                     %P......       %
%                     %PPPPPP.       %
%                     %%%%%%P.       %
%                          %P.       %
%                          %P.       %
%                          %P.       %
%                          %P.       %
%       PPPPPPPPPPPPPP.....%P.       %
%       P%%%%%%%%%%%%PPPPPPPP.       %
%       P%....................       %
%       P%....................       %
%       P%....................       %
%       P%....................       %
%       P%....................       %
%       *%....................       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 15: Path Taken By AStar On Large Maze