Artificial Intelligence Project One: Report

Bryan Plant
Department of Computer Science
Montana State University, Bozeman

Keely Weisbeck
Department of Computer Science
Montana State University, Bozeman

Spencer Cornish
Department of Computer Science
Montana State University, Bozeman

1. Introduction

The purpose of this project was to research and apply four general-purpose search algorithms to solve a maze. Breadth-First Search (BFS), Depth-First search (DFS), Greedy best-first search(Greedy), and A* search were used to solve three different mazes of varying sizes and difficulty. In order to evaluate the performance of each search algorithm, we tracked the solution cost, the number of expanded nodes, and the captured path for each algorithm for each maze.

2. Algorithms and Implementation Strategies

The program was written in the Python programming language and consisted of eight main parts. The first is the main.py class which was responsible for the initial setup and running of the entire program. Next is the maze.py class. This class initialized each maze so it could be explored by the different algorithms. The maze class used Tinker, a GUI package built into Python, to draw and display the different mazes as well as the different paths taken by the algorithms. The maze class also found and stored the start and end location of each maze. Another major part of the program was the node.py class. This class allowed us to define each point on the maze as a 'node' which contained an x and y coordinate as well as the previous node visited. This information allowed us to easily track each algorithms progress through the mazes. There is also the algorithm.py class which is the base class for the algorithms and provides helper functions. Finally, there is a class for each of our four algorithms which inherit from the base algorithm class.

2.1 Depth-First Search (DFS)

The Depth-First Search (DFS) algorithm is most commonly used to traverse graphs and trees in a depth-ward motion. DFS uses a stack to keep track of the next nodes to visit. DFS starts at the root node. The node is marked as visited, put on the stack, and an arbitrary unvisited adjacent node is found, marked as visited, and a new adjacent node is found. This process is repeated until there are no adjacent nodes that are unvisited. At this point, the top node on the stack is selected and checked for any unvisited adjacent nodes. If there is an unvisited adjacent node then it is marked as visited and placed on the stack. This process is repeated until the stack is empty.

2.1.1 Implementation

For this project, the basic description of DFS described above needed to be modified to find the solution to a maze. The "root node" was represented by the start location of the maze, and the algorithm finishes when the end location of the maze is found or when the stack was empty.

2.2 Breadth-First Search (BFS)

Breadth-First Search, like depth-first search, is a tree and graph search algorithm that explores a tree or graph in layers. BFS starts by placing the root node in a queue. The basic process is similar to DFS except BFS uses a queue instead of a stack. In

BFS the top node is taken from the queue and all adjacent unvisited nodes, the next 'layer' of the graph, are added to the queue. This process is repeated until the queue is empty.

2.2.1 Implementation

Because both breadth-first search and depth-first search are so similar, the implementation of each algorithm was also very similar. Similar to the modification made to DFS, BFS ends when the end location of the maze is found or when the queue is empty.

2.3 Greedy best-first search (Greedy)

Greedy best-first search is an informed search algorithm. The algorithm is similar to the previous two, except it has information about the goal state. When it expands a node, it expands the frontier node that is the most promising according to a heuristic. The heuristic, in this case, is the manhattan distance from each node on the frontier to the goal state.

2.3.1 Implementation

The implementation of Greedy best-first search was very similar to BFS and DFS. A python list was used as the data structure for the queue, and the python heapq library was used to make the list act as a priority queue. When a new node is added to the frontier, the node is added to the priority queue as a tuple of (distance from node to goal, node). This allows the priority queue to be sorted by how far each node is from the goal state using manhattan distance. This means that in order to get the next node to be expanded, the first node from the list is popped off. When a node is expanded, it is checked for the goal state and the algorithm returns if the goal state is found.

2.4 A*

A* is another informed search algorithm. When a node is expanded, a heuristic is computed for frontier nodes, and attempts to minimize two functions summed together. One of these functions is a cost from the start node of the search to the current node, the other is typically an estimate to the goal node. A* never overestimates, since it will always backtrack to find the best possible path to the goal.

2.4.1 Implementation

The implementation of A* was very similar to Greedy best-first search. The only major difference between the implementation of these algorithms is the use of a different heuristic. For Greedy, a simple heuristic of the manhattan distance to the goal was used, whereas with A* the length of the current path added to the estimated manhattan distance to the goal was used.

3. Results

The solution costs and the number of nodes expanded for each algorithm on each maze are shown below. An image of each completed path is also given where red is the start state, yellow is the goal state, green is the final path, and blue is the expanded nodes.

3.1 Depth-First Search (DFS)

Maze	**Solution Cost**	**Number of Expanded nodes**
Open	125	273
Medium	117	164
Large	443	830

Table 3.1.1: a description of results for the depth-first search algorithm on each maze

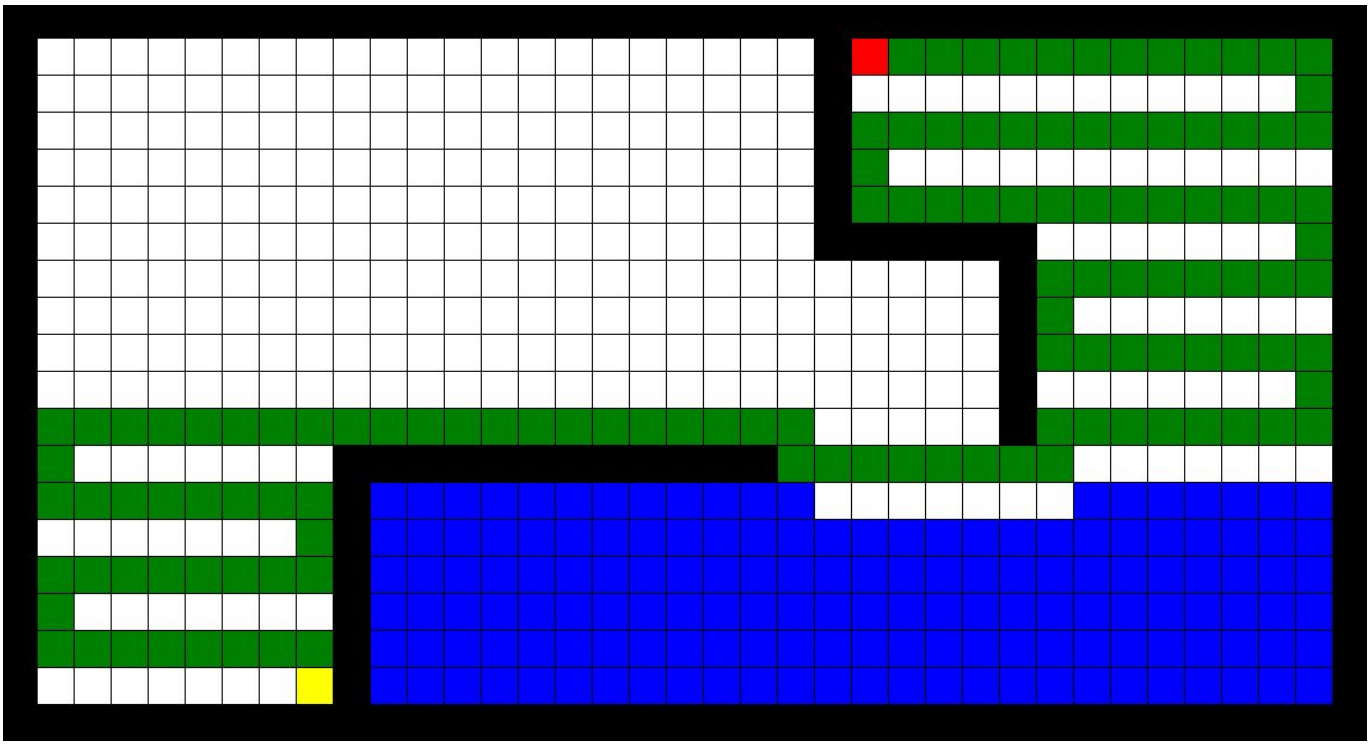

Image 3.1.1: Captured path for Open Maze

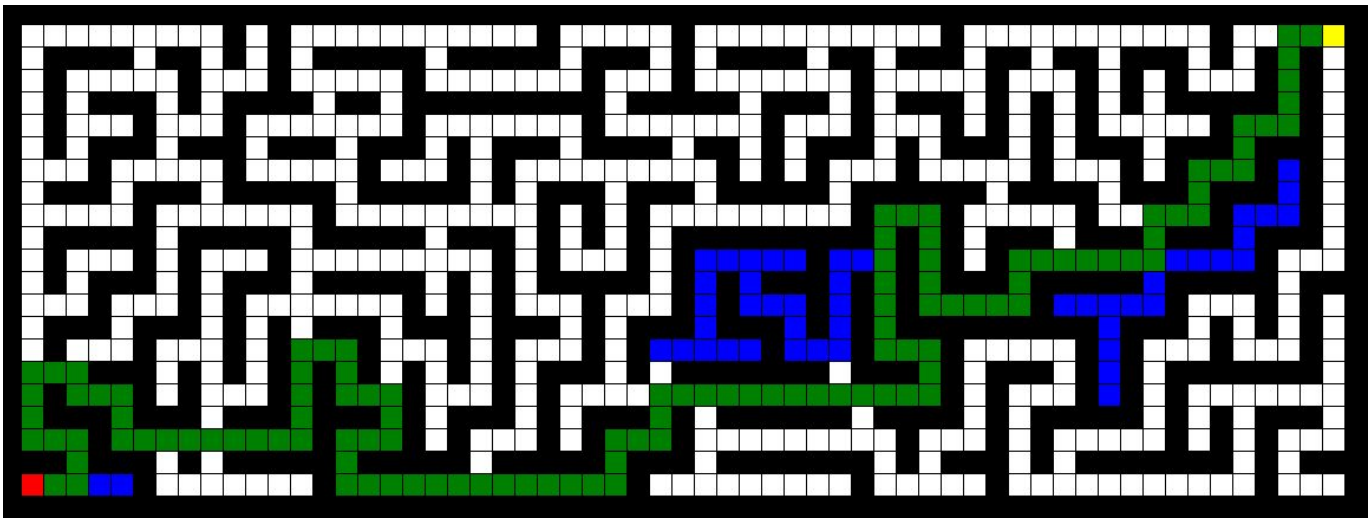

Image 3.1.2: Captured path for Medium Maze

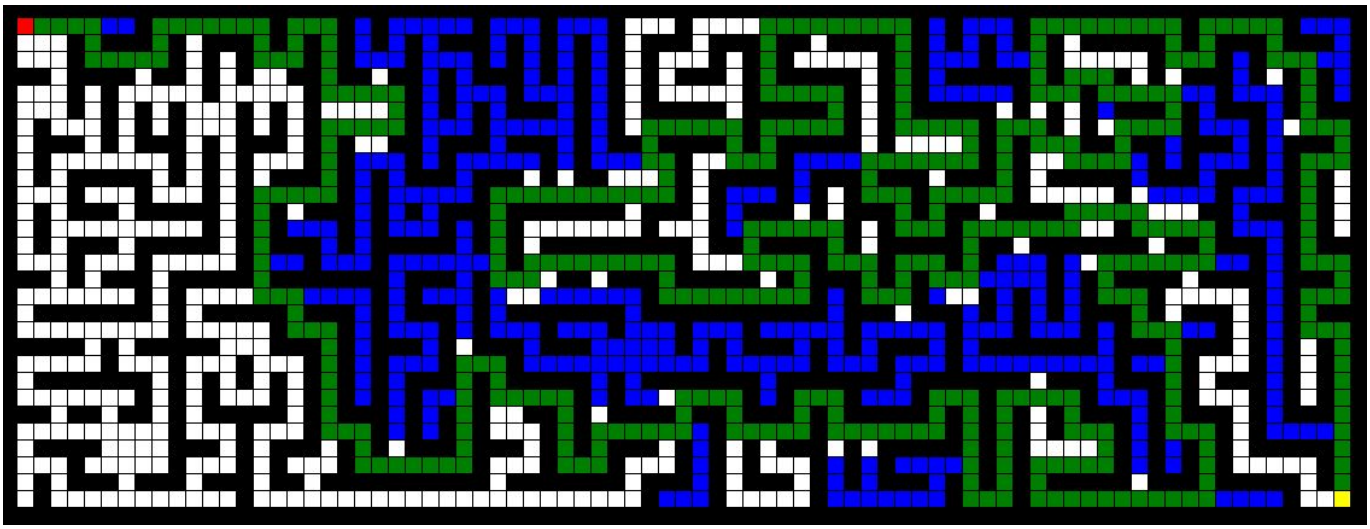

Image 3.1.3: Captured path for Large Maze

3.2 Breadth-First Search (BFS)

Maze	Solution Cost	Number of Expanded noded
Open	45	505
Medium	95	608
Large	149	1255

Table 3.2.1: a description of results for the breadth-first search algorithm on each maze

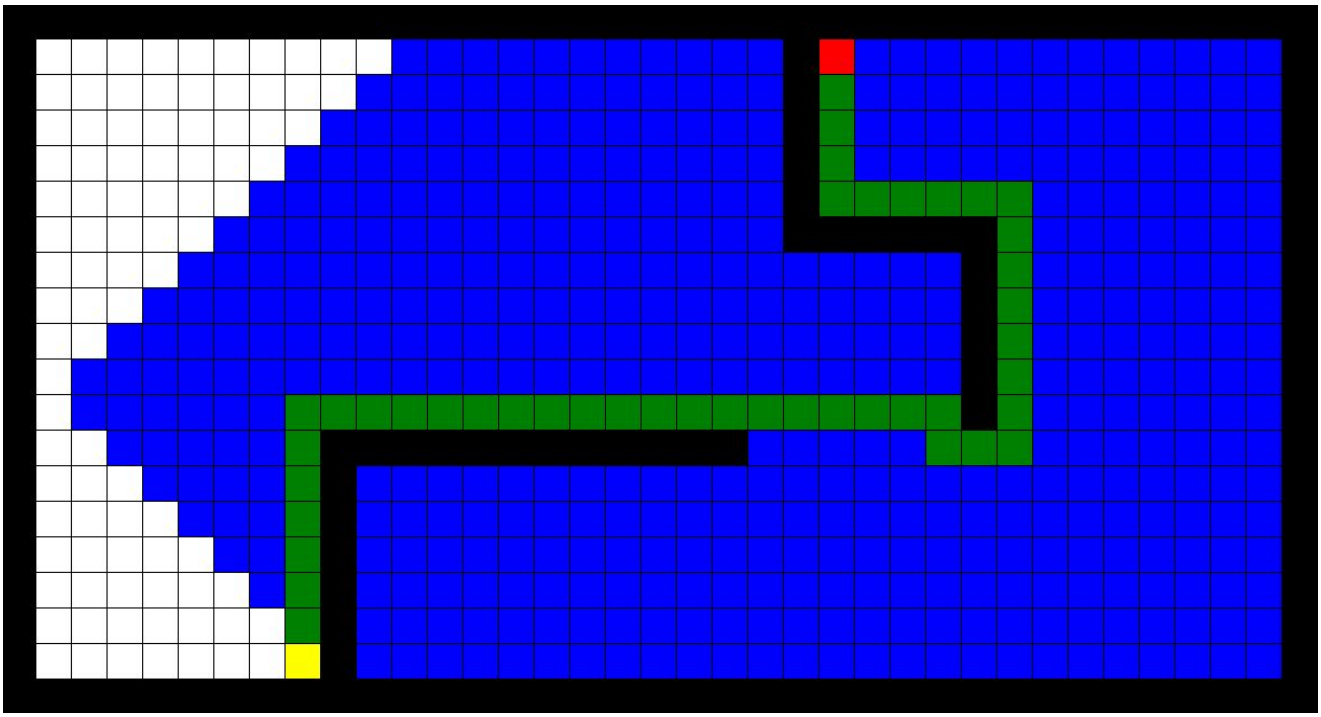

Image 3.2.1: Captured path for Open Maze

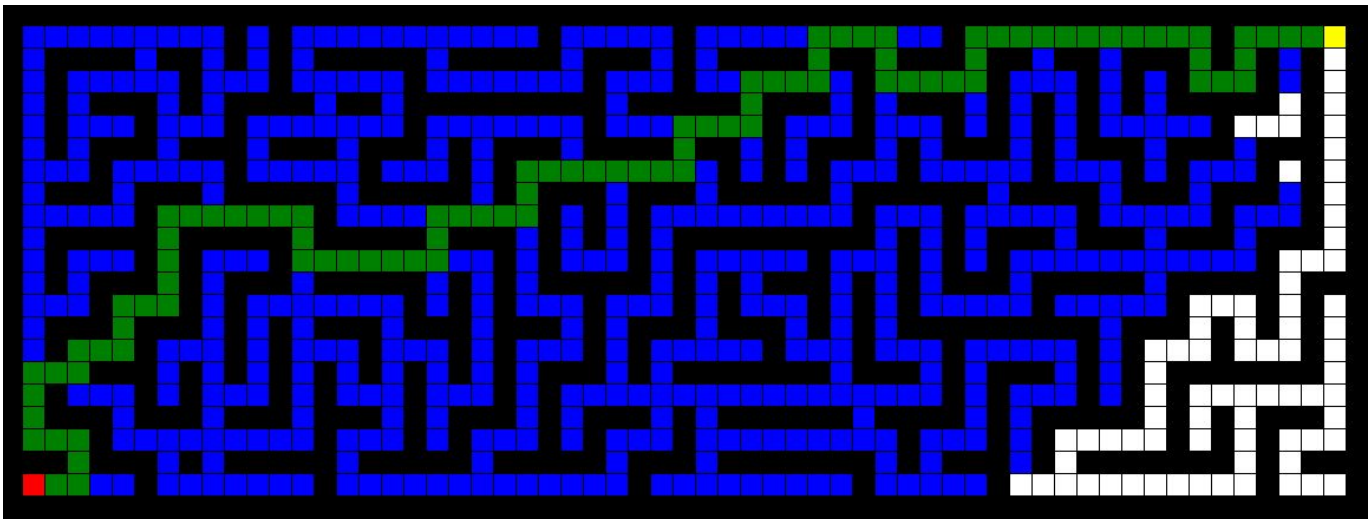

Image 3.2.2: Captured path for Medium Maze

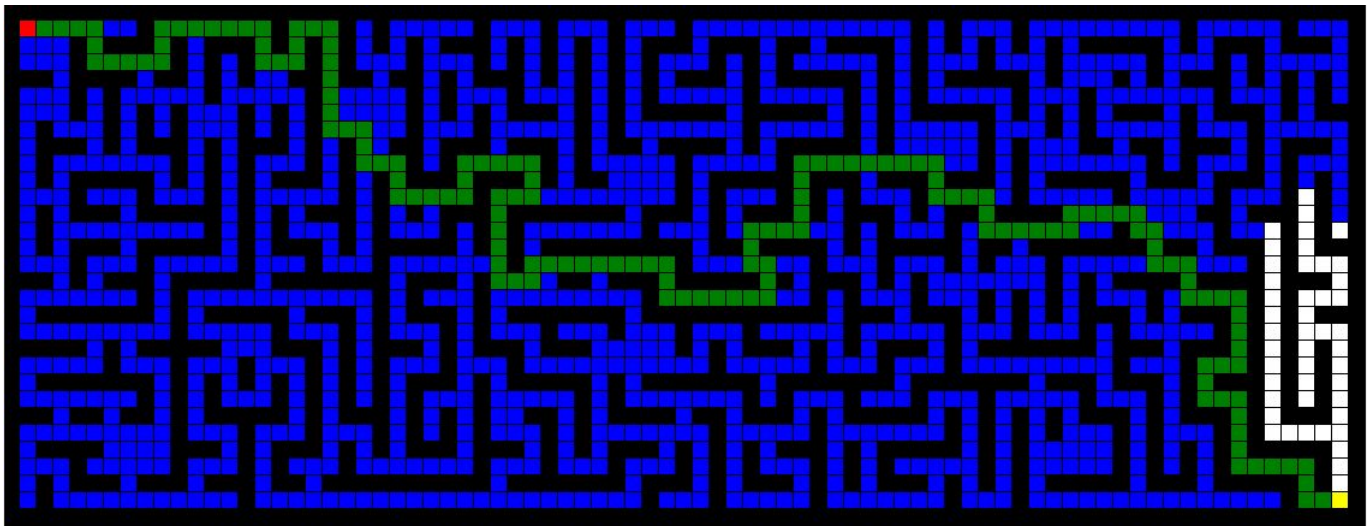

Image 3.2.3: Captured path for Large Maze

3.3 Greedy best-first search (Greedy)

Maze	Solution Cost	Number of Expanded nodes
Open	45	150
Medium	95	103
Large	223	289

Table 3.3.1: a description of results for the Greedy best-first search algorithm on each maze

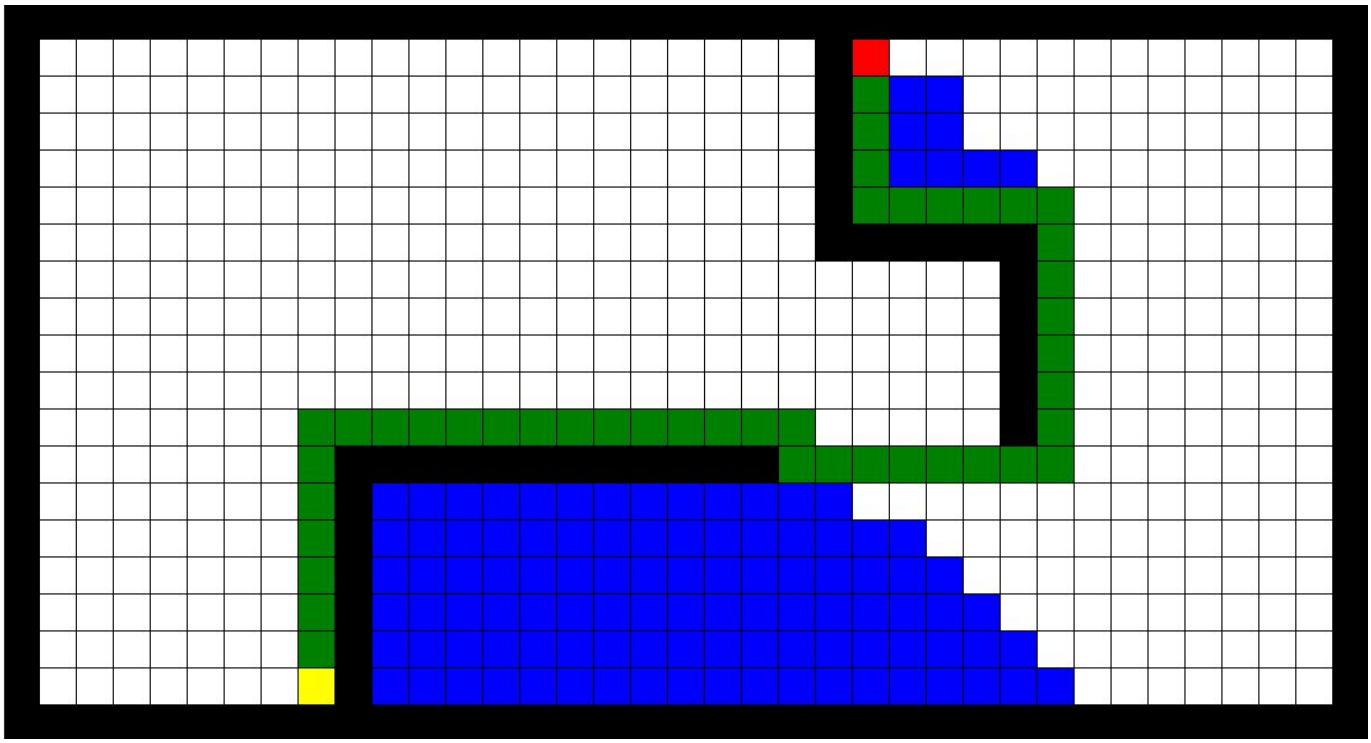

Image 3.3.1: Captured path for Open Maze

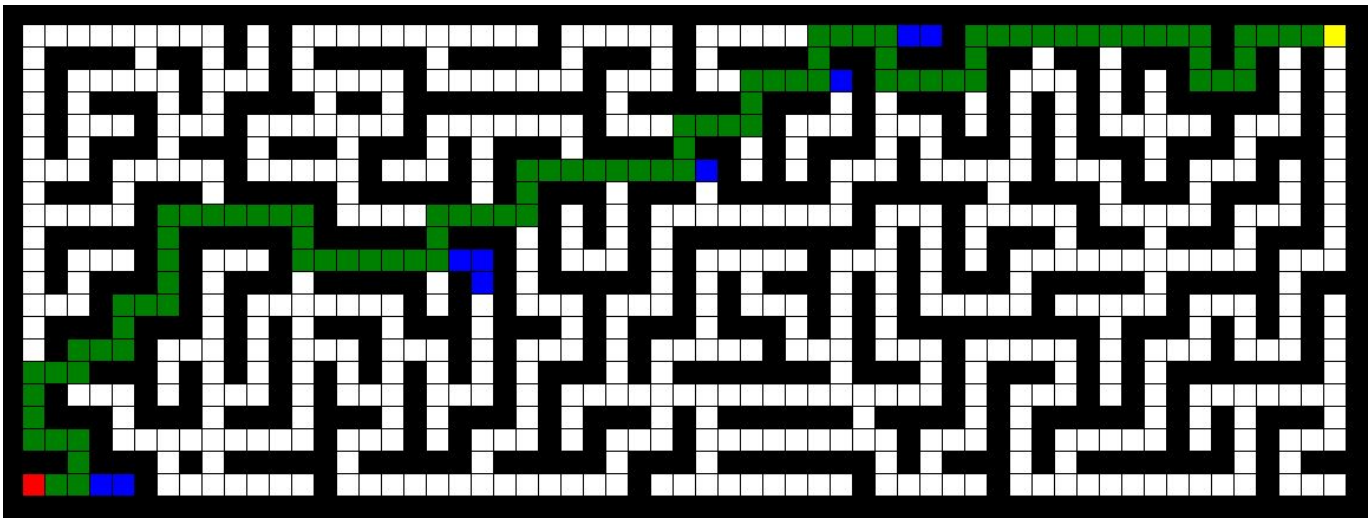

Image 3.3.2: Captured path for Medium Maze

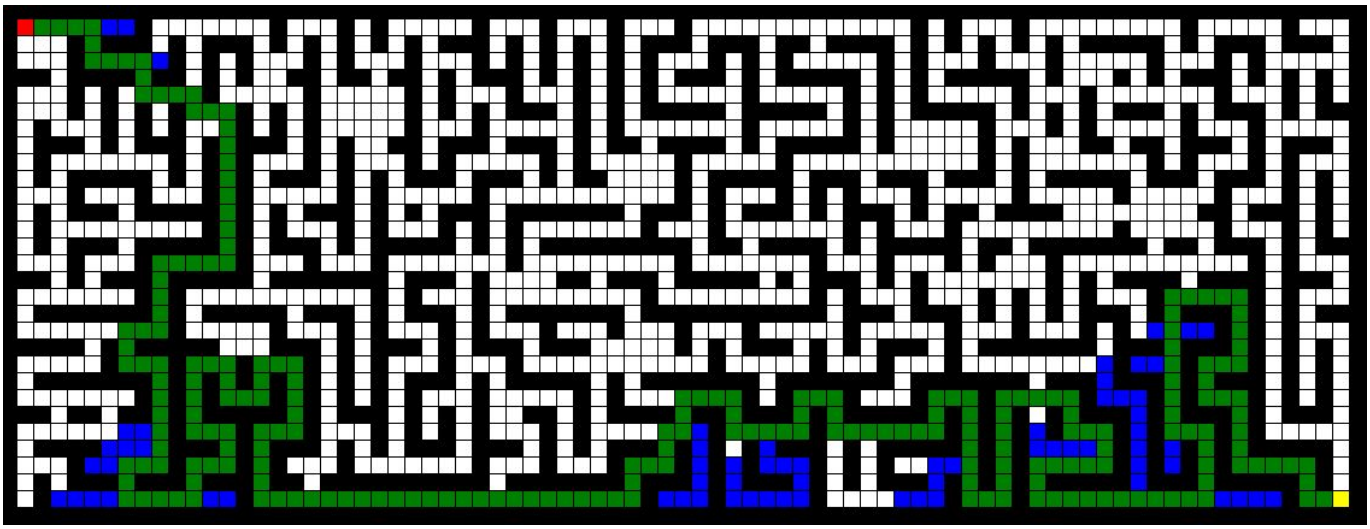

Image 3.3.3: Captured path for Large Maze

3.4 A*

Maze	Solution Cost	Number of Expanded nodes
Open	45	191
Medium	95	313
Large	149	1184

Table 3.4.1: a description of results for the A search algorithm on each maze*

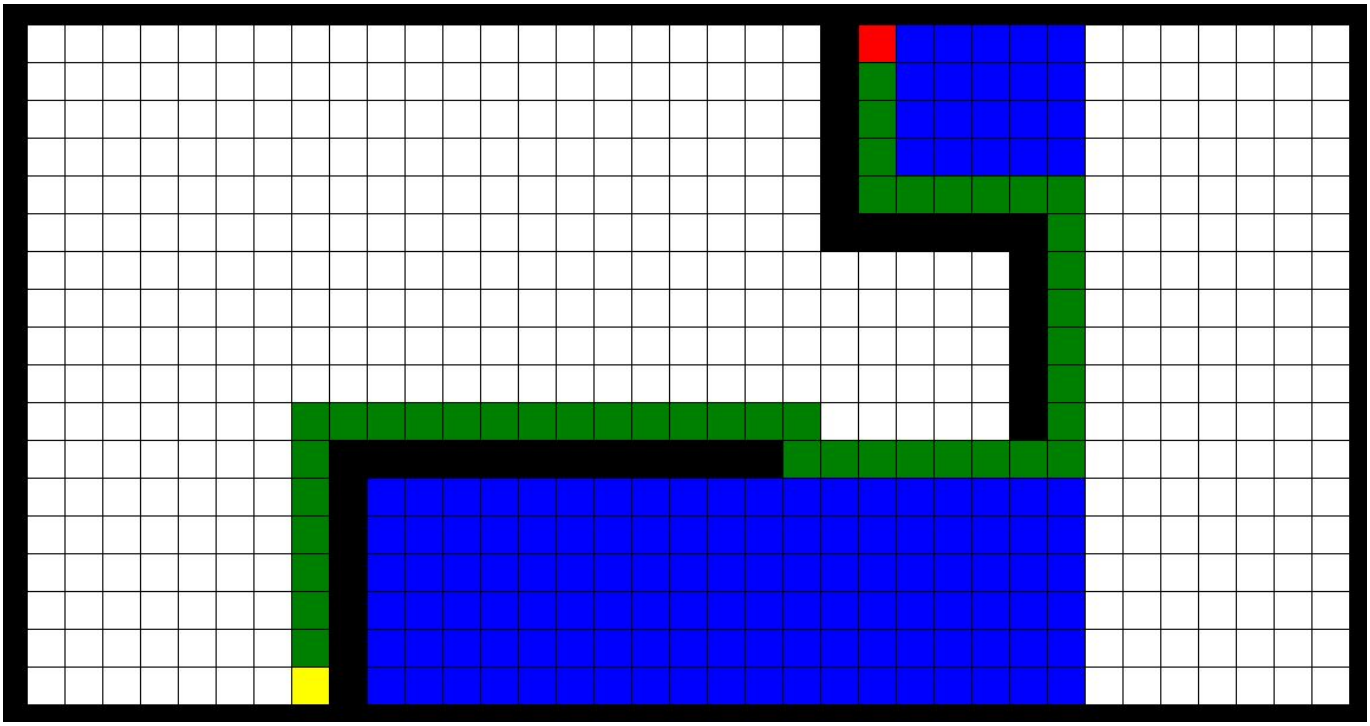

Image 3.4.1: Captured path for Open Maze

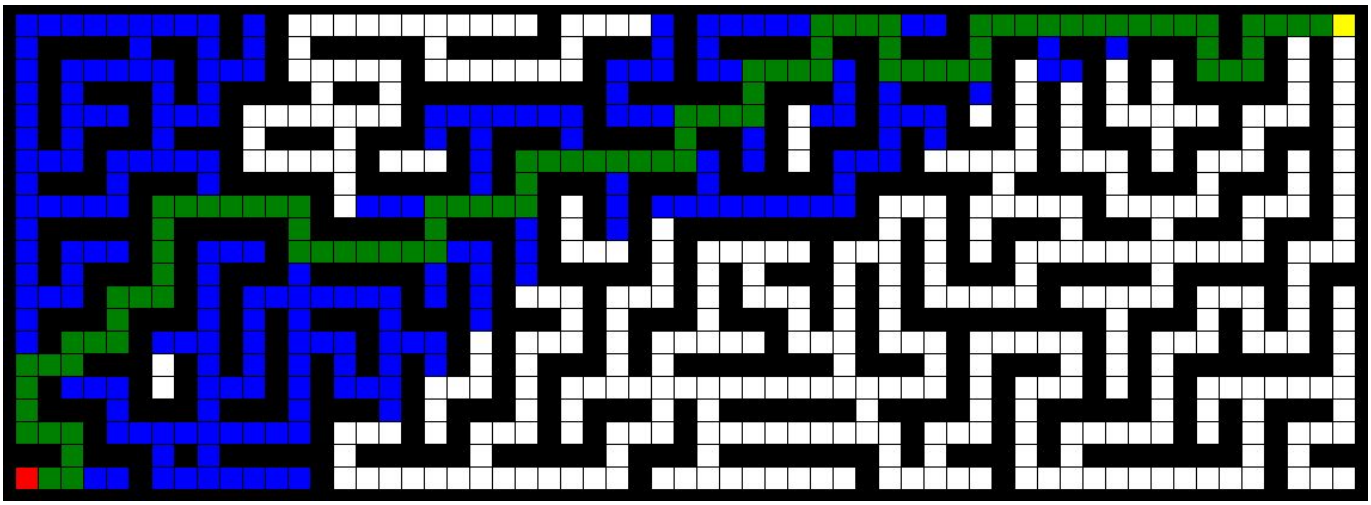

Image 3.4.2: Captured path for Medium Maze

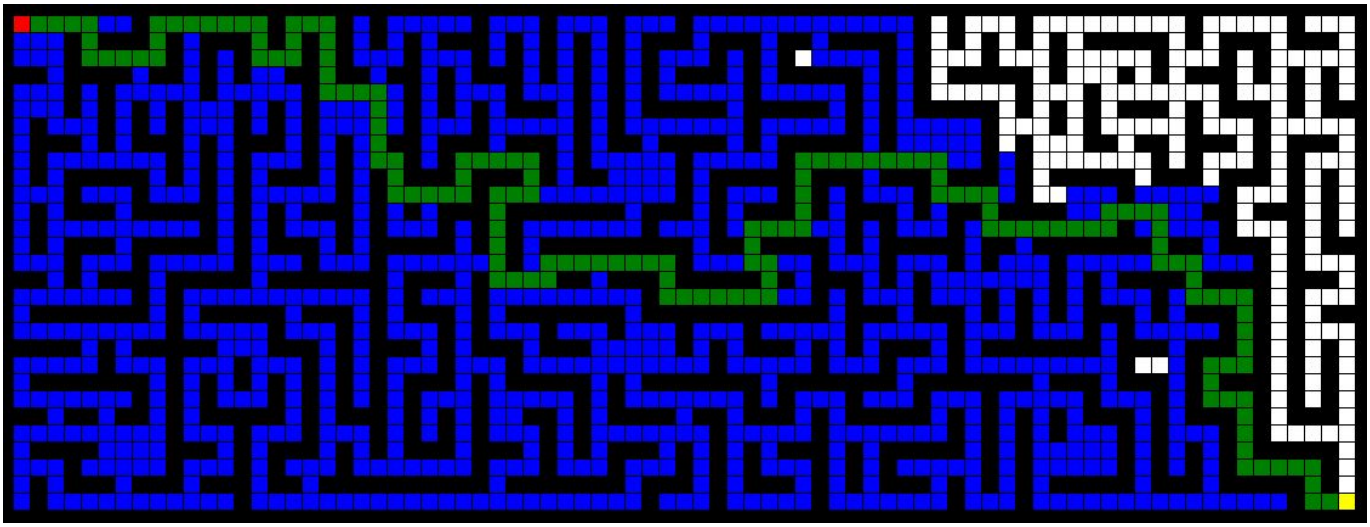

Image 3.4.3: Captured path for Large Maze

4. Contributions

The project was split into 3 parts based on the algorithms and each group member was assigned a part to implement. Every team member contributed to the writing of this paper.

4.1 Bryan Plant:

Bryan was in charge of implementing the basic structure of the program and Greedy.

4.2 Keely Weisbeck:

Keely was in charge of implementing DFS and BFS. The algorithm implementations are very similar, so the team decided these algorithms should be implemented by the same person.

4.3 Spencer Cornish:

Spencer was in charge of implementing A*.