# Artificial Intelligence Assignment

# Harry O’Donnell C20379081

## Introduction

In this assignment I will explore the efficacy of different search algorithms in several different maze environments. I will be outlining the parameters for two mazes and testing the depth first search (DFS), Iterative depth search and the A\* search algorithm on these mazes to see how they perform.

## Maze Setup

The first maze I created was a 5x5 maze which was outlined as a sample maze in the assignment document. When creating the maze, I solely specified the location of the walls and set the bounds of the walls, which is 1 to 5 for both x and y in this instance.

A black square with a white square in the middle

Description automatically generated

A screenshot of a computer program

Description automatically generated

Figure 2: Maze 1

The seconds maze I used to test the search algorithm is slighted more complex, it’s a 12x12 structure with 38 walls. It was necessary to include a far more complex structure to fully test the efficacy of the search algorithm and the differences in their searching techniques, the maze can be viewed in Figure 3: Maze 2.

In order to make the set up as efficient and flexible as possible when using a new maze or different start / end positions, I have defined the walls and bounds of each individual maze in a separate prolog file.

A computer screen shot of a code

Description automatically generatedAt the start of each search algorithms Prolog code, I include which maze I’ll be using, as well as the set the bounds and walls which are defined within the including Prolog maze file.

A screenshot of a computer program

Description automatically generatedFinally, when running the algorithm, the start and end position will be set and the path the algorithm takes with be displayed. Ensuring the set up for each algorithm and maze is consistent, enables the algorithms to effectively work for many mazes with little to no changes required.

A black and white crossword puzzle

Description automatically generated

Figure 3: Maze

## Depth First Search (DFS)

**Maze 1:**



A black line drawn on a grid

Description automatically generated

**12 Moves**

**Maze 2:**



A crossword puzzle with a black and white grid

Description automatically generated**35 Moves**

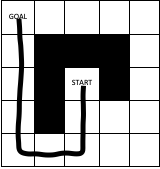
The depth first search algorithm was quite easy to implement, it a recursive algorithm which works by exploring each path to its complete depth. It’s useful to use if I was searching for an incredibly large maze which required a lot of memory but in this case, it performed poorly compared to the other algorithms in terms of the route it chose to take as well as the time it took to find its best path for maze 2, it was much slower also.

In conclusion, the DFS provides a simple memory efficient solution but in this instance due to the relatively small maze sizes, it performed poorly in terms of path taken and time to find its optimal path. This is due to the main characteristic of DFS, which is that it will go as deep as possible in a graph before then backtracking

## Iterative Depth Search (IDS)

Maze 1:





**8 Moves**

Maze 2:

A black and white crossword puzzle

Description automatically generated

**25 Moves**

The iterative depth search algorithm combines both the DFS and BFS algorithms, in my implementation I used the depth limited search with increasing depth limits, this enables the space efficiency of storing only a single path and uses the traits of BFS to find the shortest path. My implementation also uses a depth limit variable which increases after each search that isn’t successful (NewDepth is Depth + 1), this ensures that the search will eventually find its goal.

The IDS algorithm found the optimal route for the first maze and an optimal route of the seconds maze, the second maze took slightly longer to compute than the A\* but it still found the shortest path to the destination. The route it took to the second maze is slightly different to the one A\* found but it was the same number of moves.

## A\* Search

## A black and white grid with a black line Description automatically generated

## 

**8 Moves**



A black and white crossword puzzle

Description automatically generated

**25 Moves**

In my implementation of the A\* algorithm I used the Manhattan distance as the heuristic, which is a widely used heuristic for A\* which I found suitable for this maze solving problem. The problem on hand was relatively simple for the A \* algorithm to solve, so I decided to choose a computationally simple and efficient heuristic which improved the overall performance of the algorithm.

The A\* algorithm is a direct and efficient algorithm largely due to the Manhattan distance heuristic. It explores paths that are moving closer to the destination, it does this by sorting the queue based on the sum of the actual cost to reach the node (‘G’) and the estimated cost to get to the node (‘H’). This results in less nodes being explored and a more efficient shorter path.

Overall the A\* algorithm with the Manhattan distance heuristic was the most efficient algorithm for searching the mazes when it came to number of moves made and time taken to get to the destination.