In the Prolog implementation of the mazes, the maze itself is represented by defining its size and any blocked cells. The starting point and goal point are also defined. The rules for moving in each direction are defined using the move predicate, which checks whether a move in a particular direction is valid and not blocked by any obstacles.

In the implementation of the depth-first search algorithm, the dfs predicate is defined as the main entry point. It calls the dfs_helper predicate, which recursively searches for a path from the starting point to the goal. The algorithm visits each unvisited neighboring cell and marks it as visited. If the goal is found, the path is returned. Otherwise, the algorithm continues searching until all possible paths have been explored.

In the implementation of the iterative deepening search algorithm, the ids predicate is defined as the main entry point. It calls the ids_helper predicate, which recursively searches for a path from the starting point to the goal with an increasing depth limit. The algorithm visits each unvisited neighboring cell and marks it as visited. If the goal is found within the depth limit, the path is returned. Otherwise, the algorithm increases the depth limit and continues searching until all possible paths have been explored.

In the implementation of the A* search algorithm, the astar predicate is defined as the main entry point. It uses the Manhattan distance heuristic function to estimate the distance from the current cell to the goal. The algorithm keeps track of the cost of the path from the starting point to the current cell and the estimated cost from the current cell to the goal. It selects the cell with the lowest estimated cost and continues searching until the goal is found.

For the smaller maze all the algorithms found a short path to the goal though the A* was slightly longer more than likely due to inefficiencies in the heuristic function. Over all they found a fast path so were quite successful.

The larger maze had some problems . the DFS found a very long path due to the way its functions but was successful. The IDS and A* failed initially. The allowable depth had to be increased for the IDS and then it found a path far shorter then the DFS. The A* function failed on the larger maze and had to be redone as the heuristic function was unable to handle the larger maze. Though is not works the it is still inefficient as it does not find the absolute shortest path as it should. Though this was notices I could not get it to get the absolute shortest path so the current working version was used.

The performance of the different algorithms depends on the size and complexity of the maze. The depth-first search algorithm is simple to implement but may get stuck in loops and may not find the shortest path such as in the second maze. The iterative deepening search algorithm is a more robust version of the depth-first search algorithm that always finds the shortest path, but it can be slower due to the repeated searches at different depths which is why the one used to take so long to run. The A* search algorithm is the most efficient and effective algorithm for finding the shortest path, but it requires the use of a heuristic function, which can be difficult to design for complex mazes. Problems were found initially as the heuristic function worked for the simple maze but could not handle the larger more complicated one and had to be changed.

In terms of memory usage, the iterative deepening search algorithm uses the most memory due to the repeated searches at different depths. The depth-first search algorithm and A* search algorithm use less memory because they only need to store the current path and a few other variables.

In conclusion, the choice of algorithm depends on the size and complexity of the maze, as well as the desired trade-off between efficiency and accuracy. The depth-first search algorithm is simple and fast

but may not always find the shortest path. The iterative deepening search algorithm is more robust but slower and uses more memory. The A* search algorithm is the most efficient and effective but requires the use of a heuristic function, which can be difficult to design.