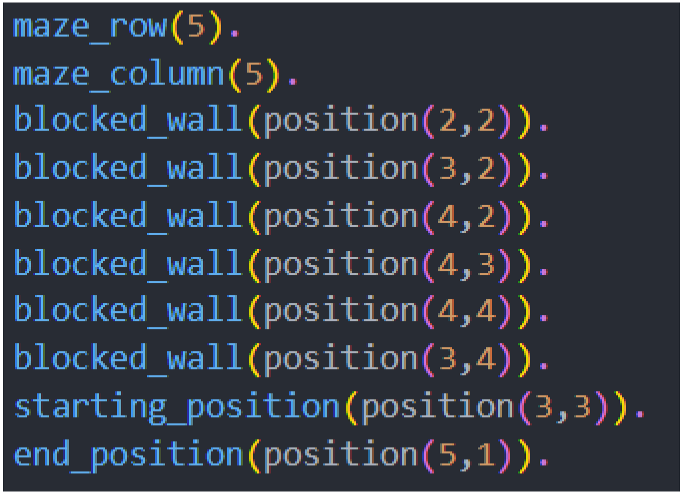
Artificial Intelligence Maze Assignment

Lecturers Name:	Dr. Svetlana Hensman
Student Name and Number:	Djibril Coulybaly C18423664
Programme:	DT228 BSc in Computer Science
Module:	Artificial Intelligence
Submission Date:	01st September 2022

Maze Design Decisions

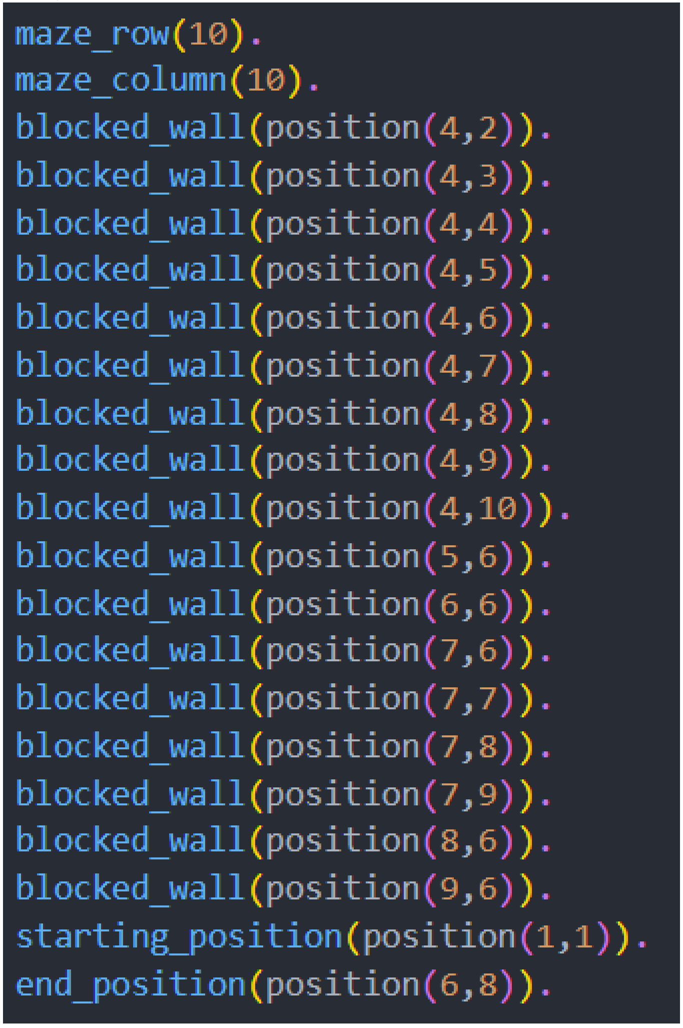
To begin, the 2 sample mazes that were noted in the CA description were implemented as follows:

Sample Maze 1



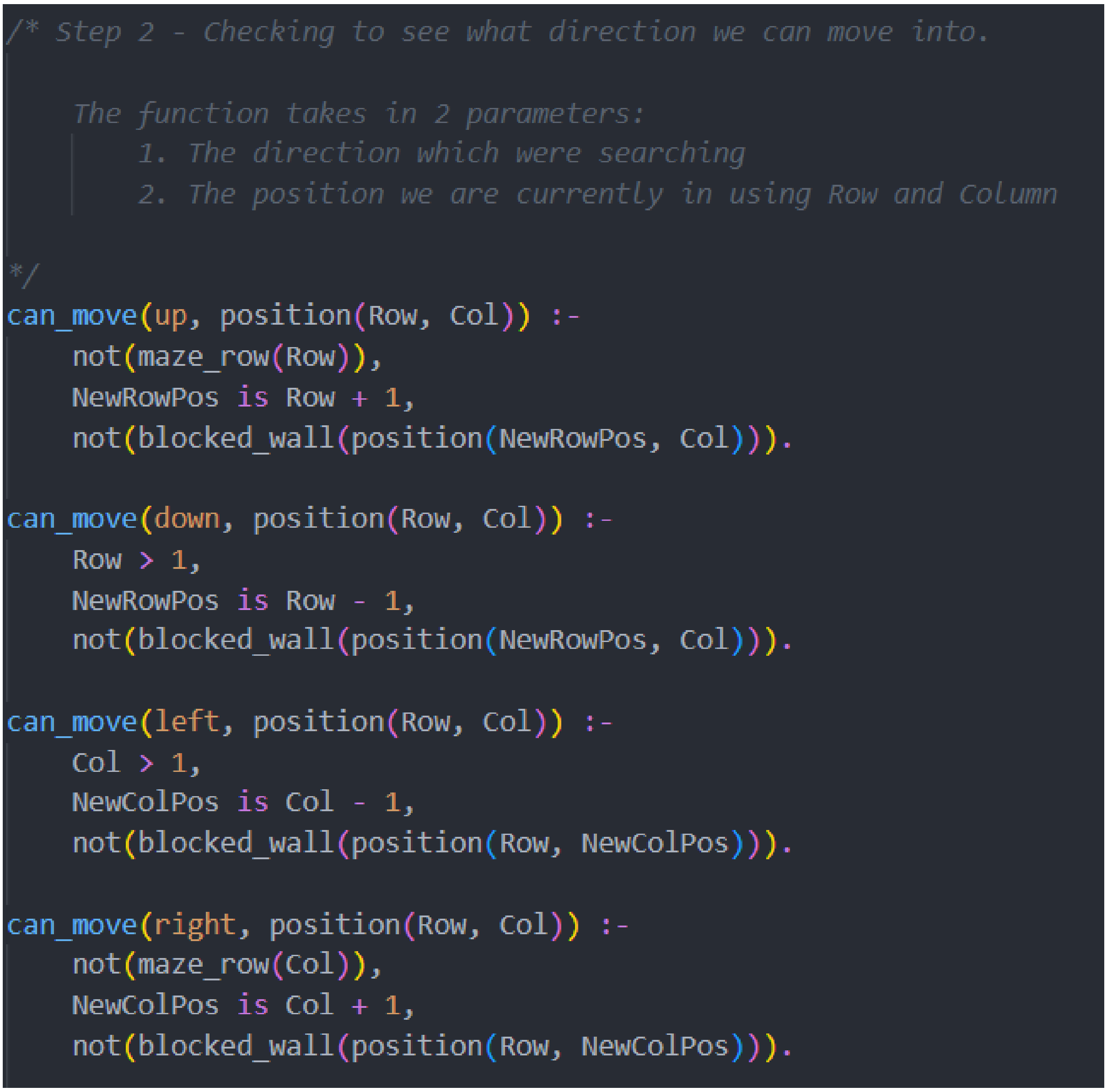
```
maze_row(5).
maze_column(5).
blocked_wall(position(2,2)).
blocked_wall(position(3,2)).
blocked_wall(position(4,2)).
blocked_wall(position(4,3)).
blocked_wall(position(4,4)).
blocked_wall(position(3,4)).
starting_position(position(3,3)).
end_position(position(5,1)).
```

Sample Maze 2

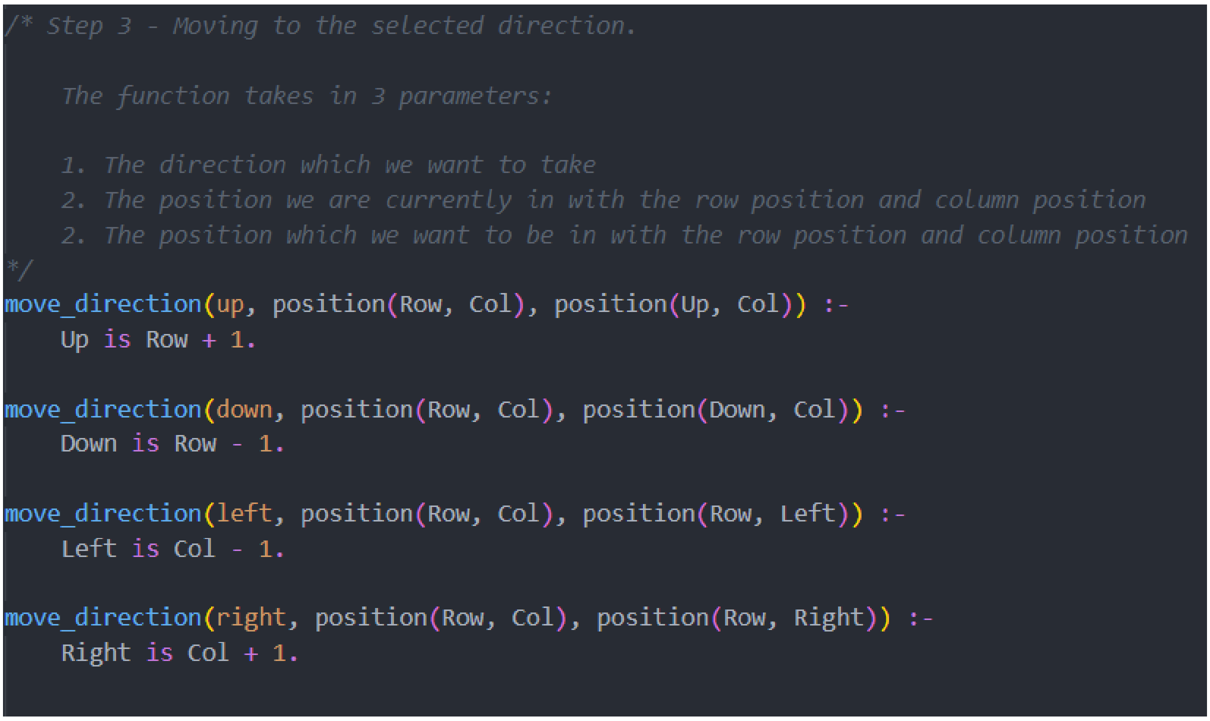


```
maze_row(10).
maze_column(10).
blocked_wall(position(4,2)).
blocked_wall(position(4,3)).
blocked_wall(position(4,4)).
blocked_wall(position(4,5)).
blocked_wall(position(4,6)).
blocked_wall(position(4,7)).
blocked_wall(position(4,8)).
blocked_wall(position(4,9)).
blocked_wall(position(4,10)).
blocked_wall(position(5,6)).
blocked_wall(position(6,6)).
blocked_wall(position(7,6)).
blocked_wall(position(7,7)).
blocked_wall(position(7,8)).
blocked_wall(position(7,9)).
blocked_wall(position(8,6)).
blocked_wall(position(9,6)).
starting_position(position(1,1)).
end_position(position(6,8)).
```

Next, the following two steps were a common factor that was implemented across each search algorithm to conduct the direction of navigating the maze:



```
/* Step 2 - Checking to see what direction we can move into.

    The function takes in 2 parameters:
        1. The direction which were searching
        2. The position we are currently in using Row and Column

*/
can_move(up, position(Row, Col)) :-
    not(maze_row(Row)),
    NewRowPos is Row + 1,
    not(blocked_wall(position(NewRowPos, Col))).

can_move(down, position(Row, Col)) :-
    Row > 1,
    NewRowPos is Row - 1,
    not(blocked_wall(position(NewRowPos, Col))).

can_move(left, position(Row, Col)) :-
    Col > 1,
    NewColPos is Col - 1,
    not(blocked_wall(position(Row, NewColPos))).

can_move(right, position(Row, Col)) :-
    not(maze_row(Col)),
    NewColPos is Col + 1,
    not(blocked_wall(position(Row, NewColPos))).
```



```
/* Step 3 - Moving to the selected direction.

    The function takes in 3 parameters:

    1. The direction which we want to take
    2. The position we are currently in with the row position and column position
    2. The position which we want to be in with the row position and column position
*/
move_direction(up, position(Row, Col), position(Up, Col)) :-
    Up is Row + 1.

move_direction(down, position(Row, Col), position(Down, Col)) :-
    Down is Row - 1.

move_direction(left, position(Row, Col), position(Row, Left)) :-
    Left is Col - 1.

move_direction(right, position(Row, Col), position(Row, Right)) :-
    Right is Col + 1.
```

How each search algorithm was individually implemented will be discussed in the next chapter. To run each of the search algorithms, consult each of the search algorithms (saved in .pl format) and execute the following commands:

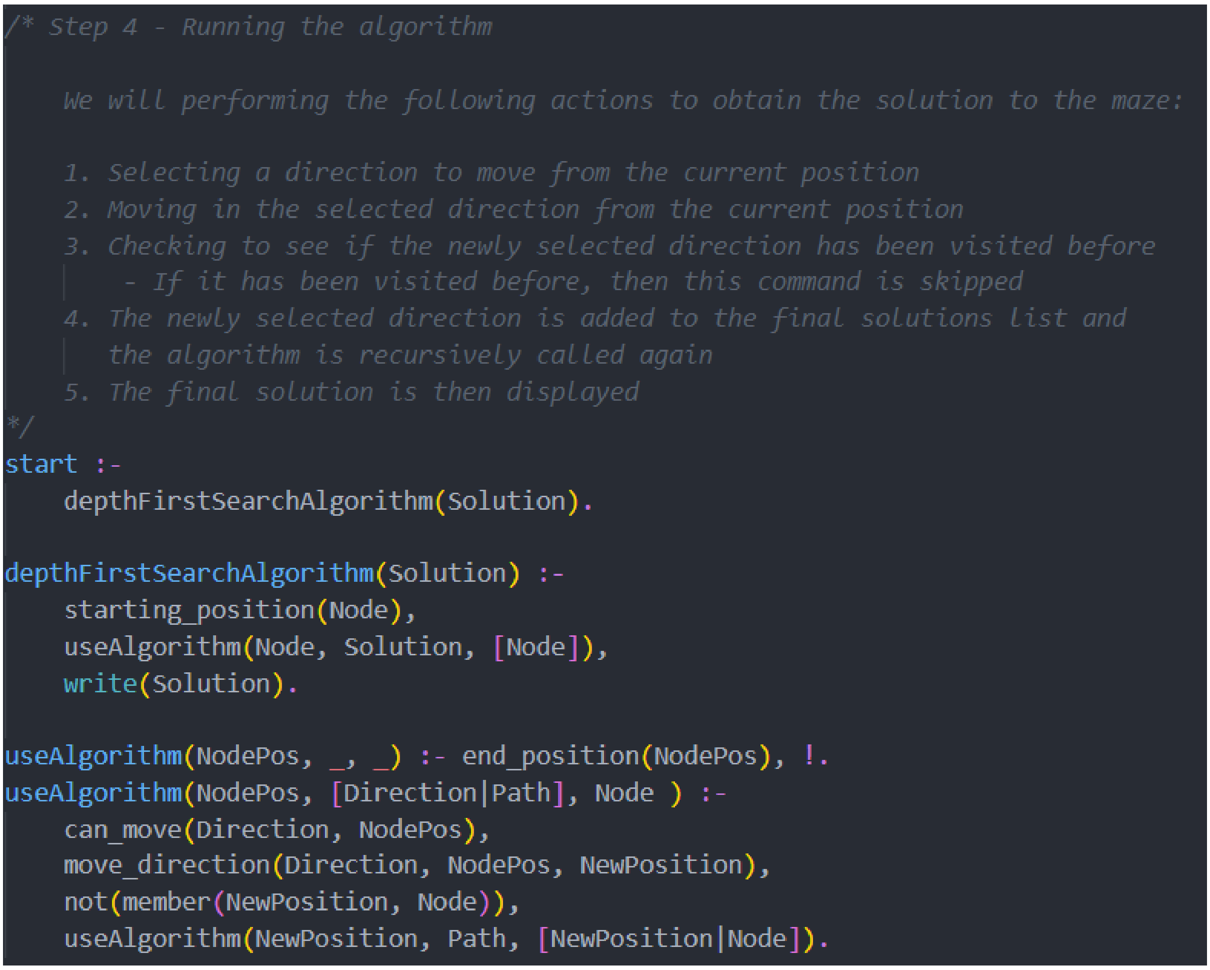
Search Algorithm	**Command for running the program**	**Command for viewing performance time**
Depth First Search	solveMaze.	time((solveMaze)).
Iterative Deep Search	solveMaze.	time((solveMaze)).
A* Search	solveMaze.	time((solveMaze)).

Search Algorithms Implemented

Depth First Search

Algorithm Path and Performance

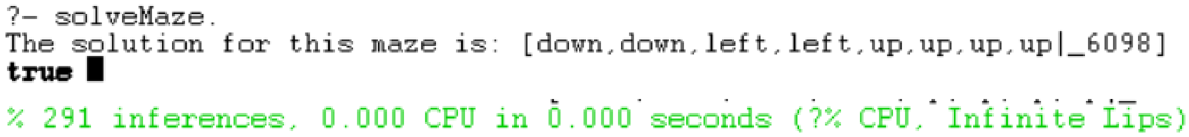
The following screenshot illustrates the though-process of implementing the depth first search algorithm



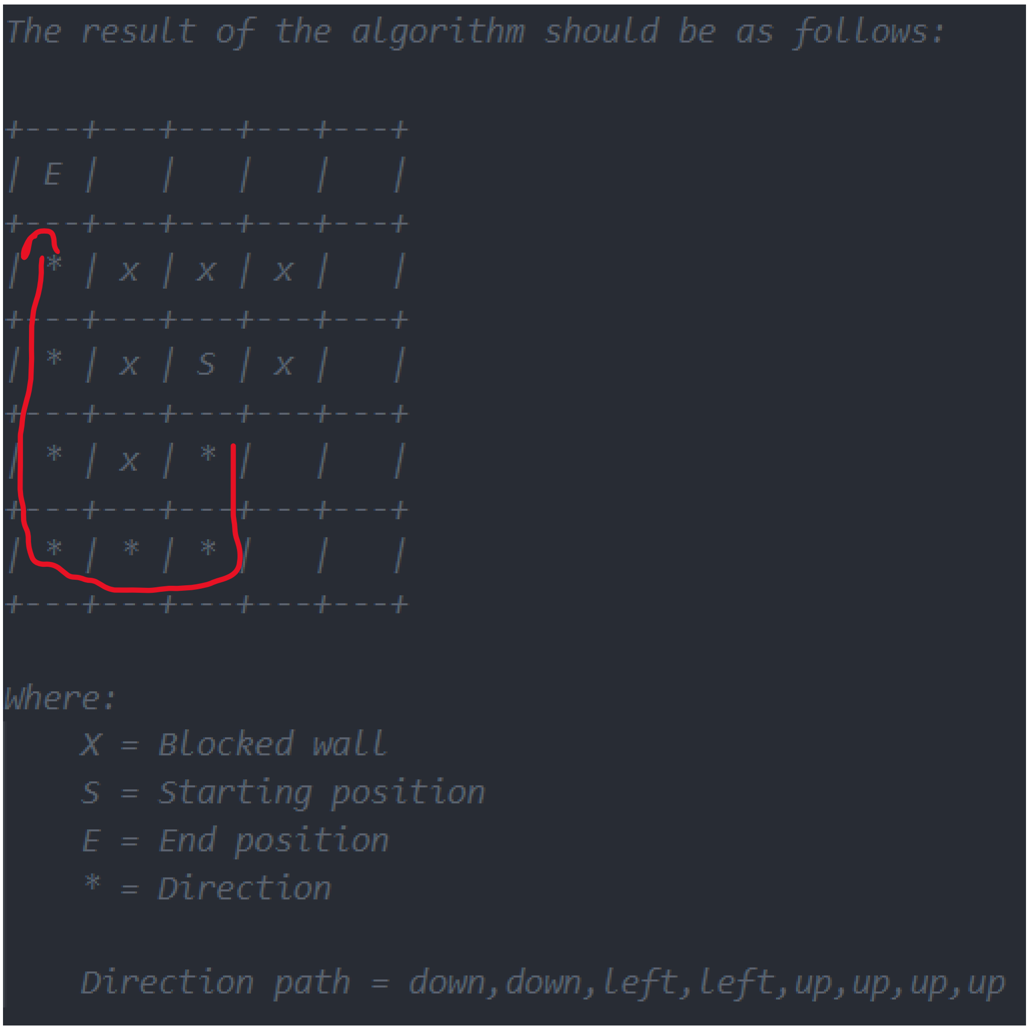
```
/* Step 4 - Running the algorithm

    We will performing the following actions to obtain the solution to the maze:

    1. Selecting a direction to move from the current position
    2. Moving in the selected direction from the current position
    3. Checking to see if the newly selected direction has been visited before
        - If it has been visited before, then this command is skipped
    4. The newly selected direction is added to the final solutions list and
       the algorithm is recursively called again
    5. The final solution is then displayed
*/
start :-
    depthFirstSearchAlgorithm(Solution).

depthFirstSearchAlgorithm(Solution) :-
    starting_position(Node),
    useAlgorithm(Node, Solution, [Node]),
    write(Solution).

useAlgorithm(NodePos, _, _) :- end_position(NodePos), !.
useAlgorithm(NodePos, [Direction|Path], Node ) :-
    can_move(Direction, NodePos),
    move_direction(Direction, NodePos, NewPosition),
    not(member(NewPosition, Node)),
    useAlgorithm(NewPosition, Path, [NewPosition|Node]).
```

Sample Maze 1

The following is the output obtained from executing the depth first search algorithm on Sample Maze 1, alongside the performance time and a figure of the path from the starting position to the end position.



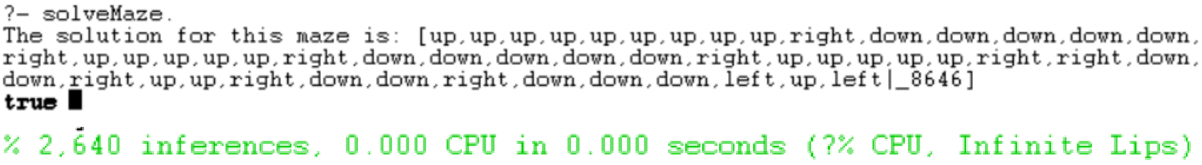
```
?- solveMaze.
The solution for this maze is: [down,down,left,left,up,up,up,up|_6098]
true
% 291 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
```



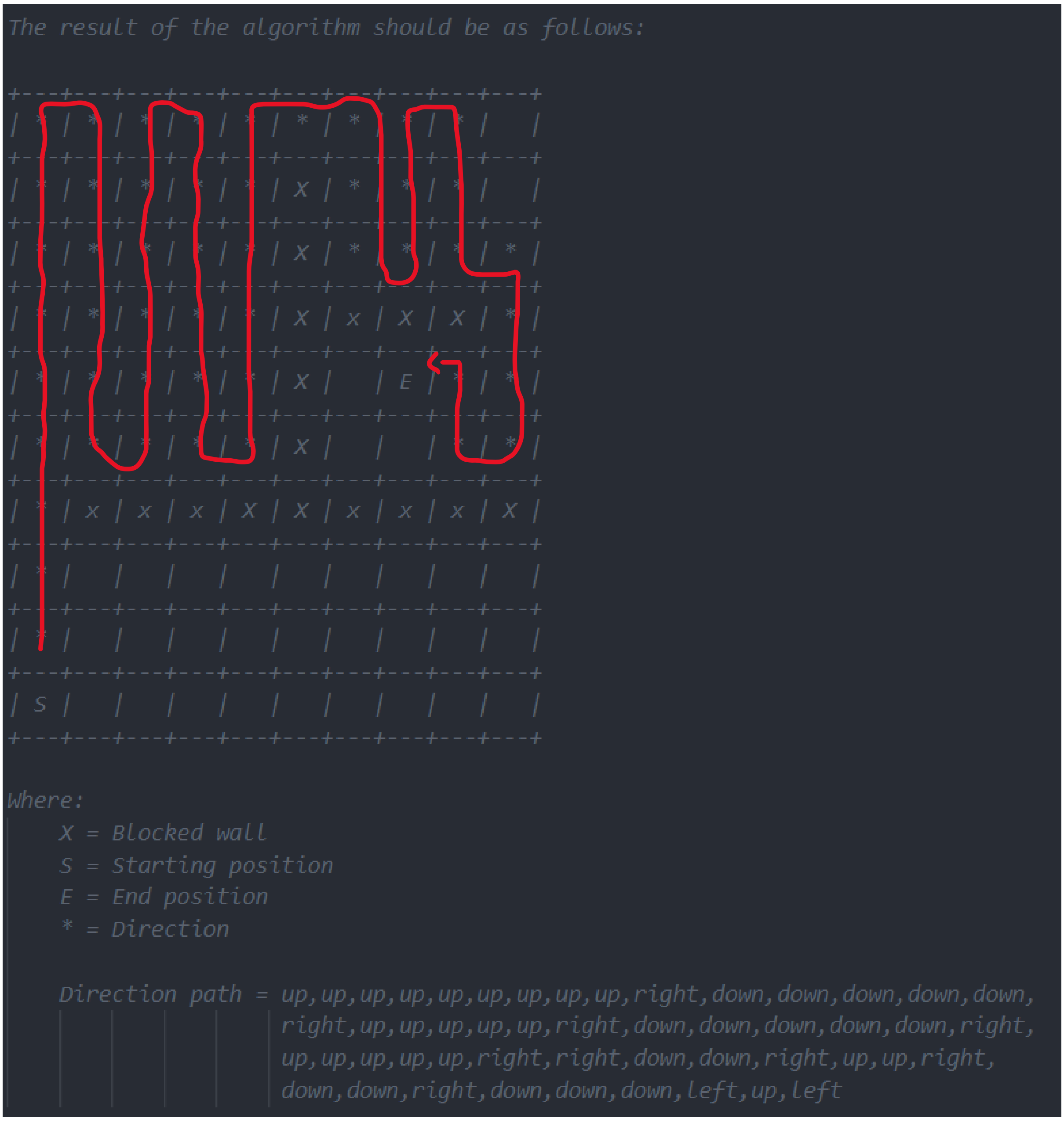
```
The result of the algorithm should be as follows:

+---+---+---+---+---+
| E |   |   |   |   |
+---+---+---+---+---+
| * | x | x | x |   |
+---+---+---+---+---+
| * | x | S | x |   |
+---+---+---+---+---+
| * | x | * |   |   |
+---+---+---+---+---+
| * | * | * |   |   |
+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = down,down,left,left,up,up,up,up
```

Sample Maze 2

The following is the output obtained from executing the depth first search algorithm on Sample Maze 2, alongside the performance time and a figure of the path from the starting position to the end position.



```
?- solveMaze.
The solution for this maze is: [up,up,up,up,up,up,up,up,up,right,down,down,down,down,down,
right,up,up,up,up,up,right,down,down,down,down,down,right,up,up,up,up,up,right,right,down,
down,right,up,up,right,down,down,right,down,down,down,left,up,left|_8646]
true 

% 2,640 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
```

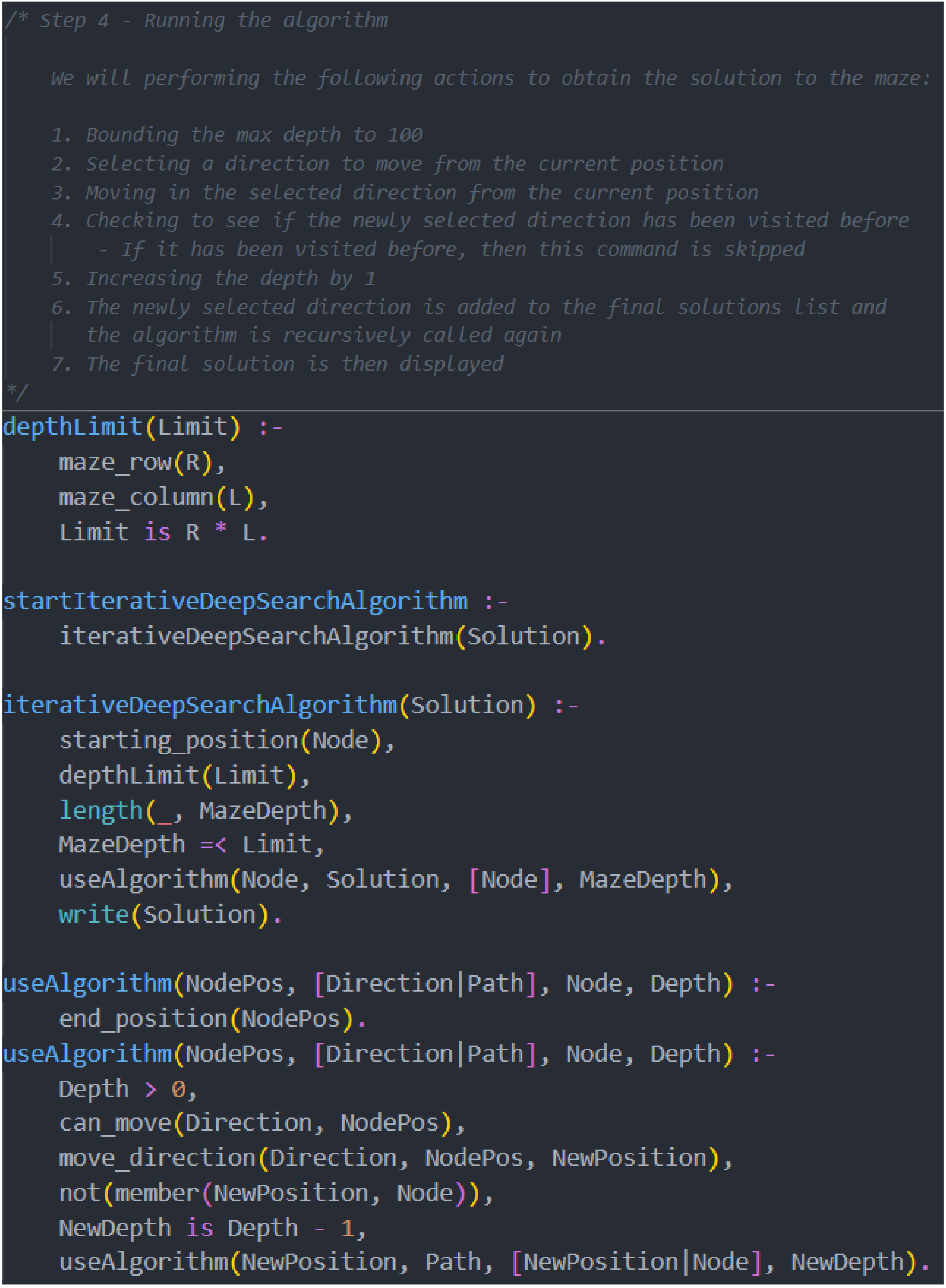


```
The result of the algorithm should be as follows:


+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | * | * | * | * |   |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X | * | * | * |   |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X | * | * | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X | x | X | X | * |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X |   | E | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X |   |   | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * | x | x | x | X | X | x | x | x | X |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| S |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = up,up,up,up,up,up,up,up,up,right,down,down,down,down,down,
                     right,up,up,up,up,up,right,down,down,down,down,down,right,
                     up,up,up,up,up,right,right,down,down,right,up,up,right,
                     down,down,right,down,down,down,left,up,left
```

Iterative Deep Search

Algorithm Path and Performance

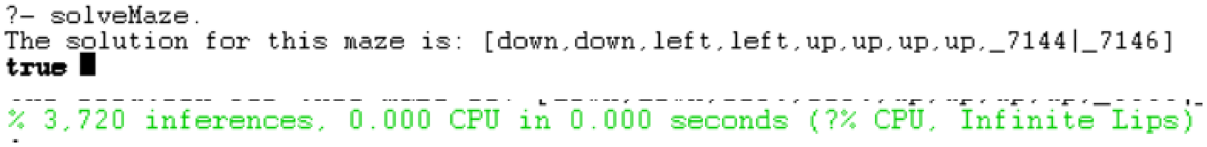
The following screenshot illustrates the though-process of implementing the iterative deep search algorithm



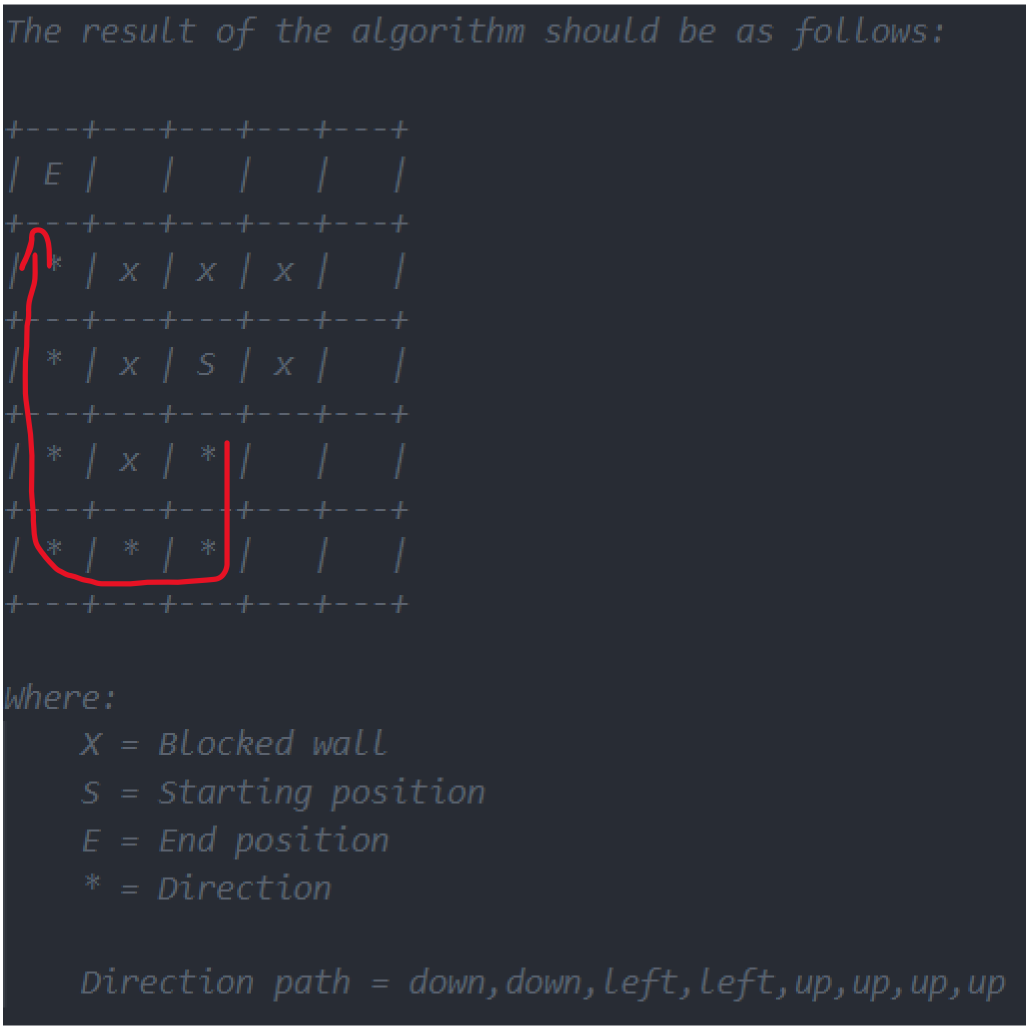
```
/* Step 4 - Running the algorithm

    We will performing the following actions to obtain the solution to the maze:

    1. Bounding the max depth to 100
    2. Selecting a direction to move from the current position
    3. Moving in the selected direction from the current position
    4. Checking to see if the newly selected direction has been visited before
        - If it has been visited before, then this command is skipped
    5. Increasing the depth by 1
    6. The newly selected direction is added to the final solutions list and
       the algorithm is recursively called again
    7. The final solution is then displayed
*/
depthLimit(Limit) :-
    maze_row(R),
    maze_column(L),
    Limit is R * L.

startIterativeDeepSearchAlgorithm :-
    iterativeDeepSearchAlgorithm(Solution).

iterativeDeepSearchAlgorithm(Solution) :-
    starting_position(Node),
    depthLimit(Limit),
    length(_, MazeDepth),
    MazeDepth =< Limit,
    useAlgorithm(Node, Solution, [Node], MazeDepth),
    write(Solution).

useAlgorithm(NodePos, [Direction|Path], Node, Depth) :-
    end_position(NodePos).
useAlgorithm(NodePos, [Direction|Path], Node, Depth) :-
    Depth > 0,
    can_move(Direction, NodePos),
    move_direction(Direction, NodePos, NewPosition),
    not(member(NewPosition, Node)),
    NewDepth is Depth - 1,
    useAlgorithm(NewPosition, Path, [NewPosition|Node], NewDepth).
```

Sample Maze 1

The following is the output obtained from executing the iterative deep search algorithm on Sample Maze 1, alongside the performance time and a figure of the path from the starting position to the end position.



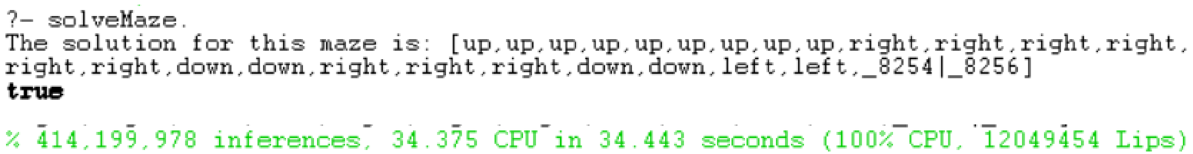
```
?- solveMaze.
The solution for this maze is: [down,down,left,left,up,up,up,up,_7144|_7146]
true █
% 3,720 inferences, 0.000 CPU in 0.000 seconds (?% CPU, Infinite Lips)
```



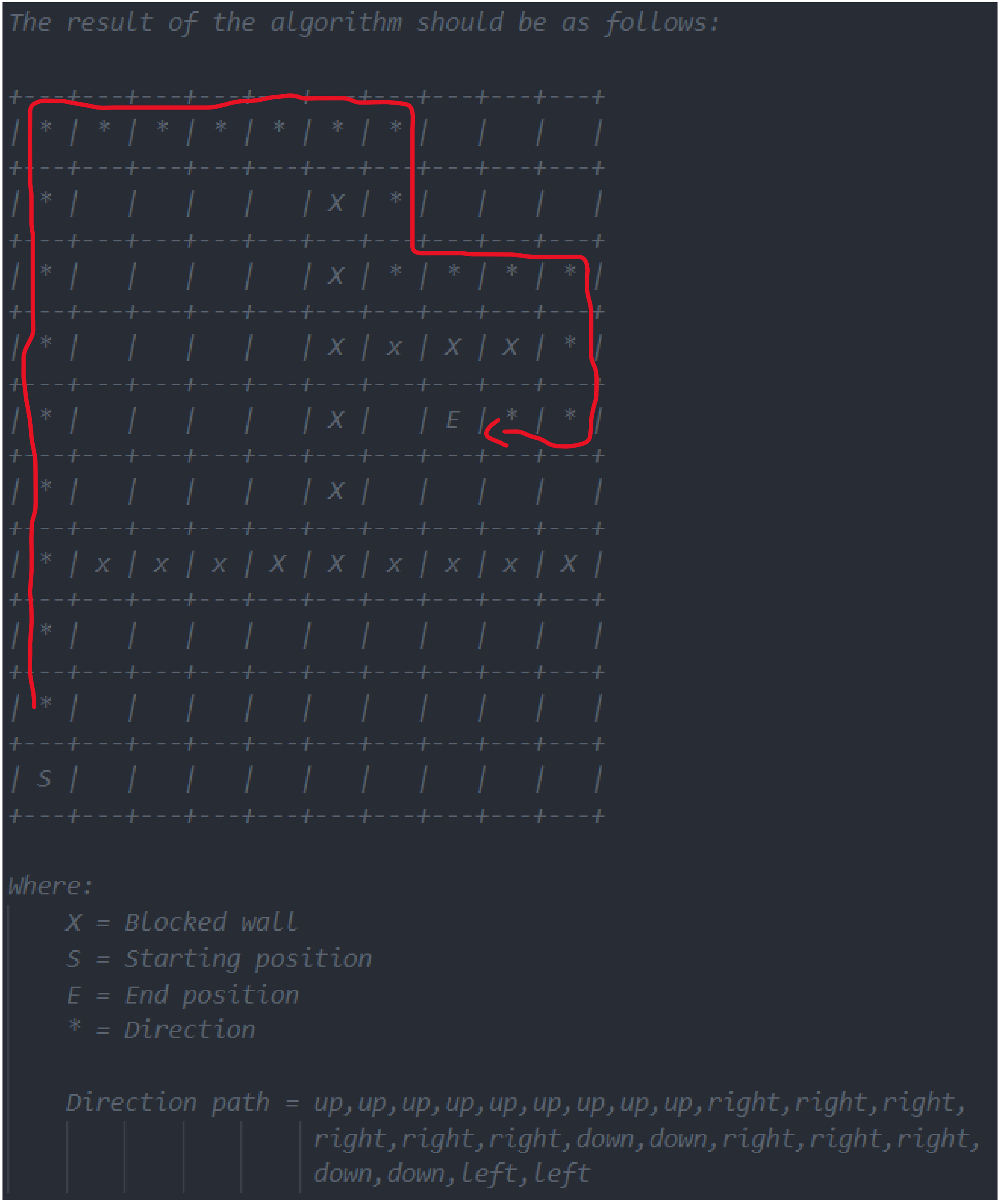
```
The result of the algorithm should be as follows:

+---+---+---+---+---+
| E |   |   |   |   |
+---+---+---+---+---+
| * | x | x | x |   |
+---+---+---+---+---+
| * | x | S | x |   |
+---+---+---+---+---+
| * | x | * |   |   |
+---+---+---+---+---+
| * | * | * |   |   |
+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = down,down,left,left,up,up,up,up
```

Sample Maze 2

The following is the output obtained from executing the iterative deep search algorithm on Sample Maze 2, alongside the performance time and a figure of the path from the starting position to the end position.



```
?- solveMaze.
The solution for this maze is: [up,up,up,up,up,up,up,up,up,right,right,right,right,
right,right,down,down,right,right,right,down,down,left,left,_8254|_8256]
true

% 414,199,978 inferences, 34.375 CPU in 34.443 seconds (100% CPU, 12049454 Lips)
```

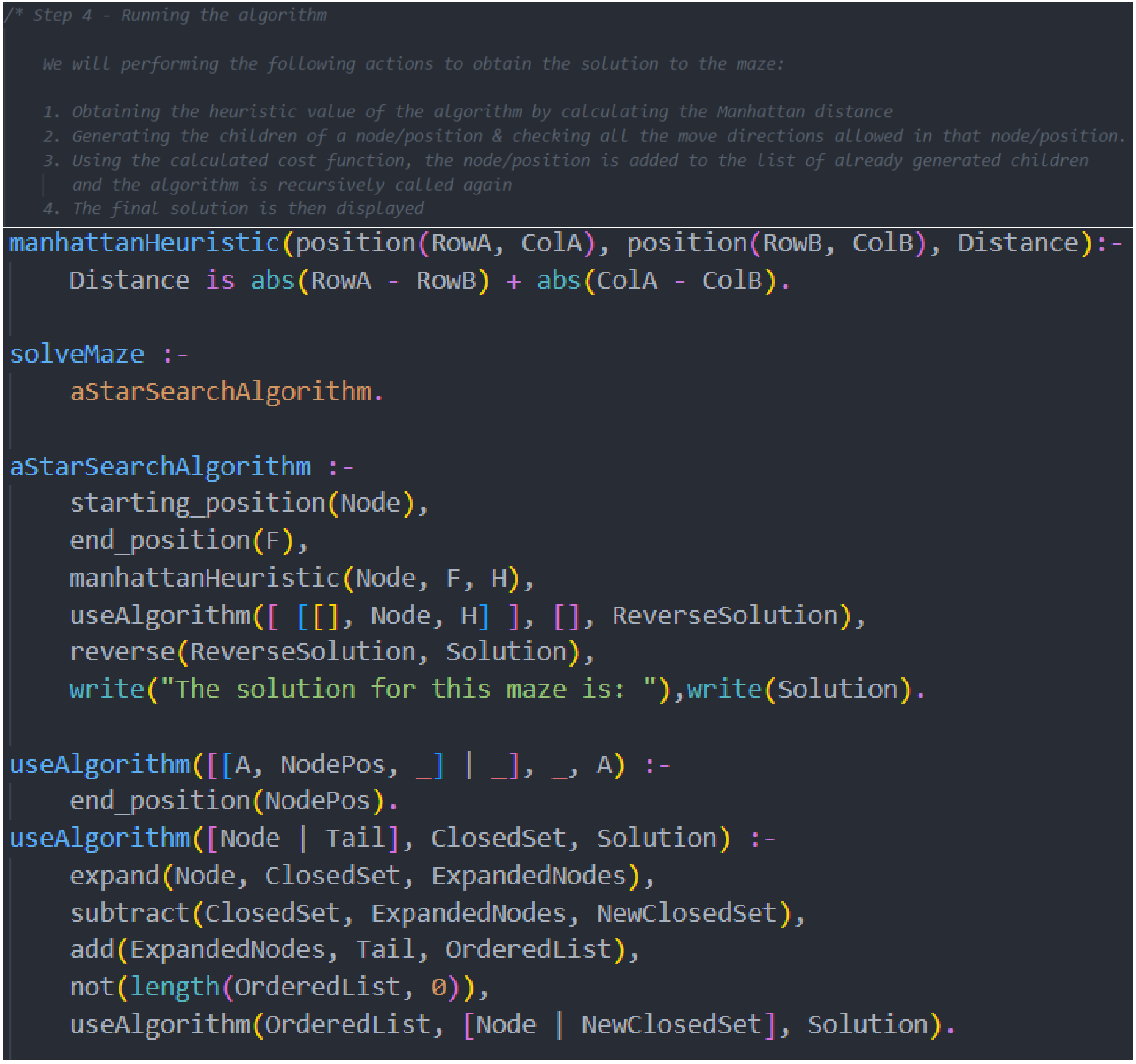


```
The result of the algorithm should be as follows:

+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | * | * |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X | * |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X | * | * | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X | x | X | x | * |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X |   | E | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * | x | x | x | X | X | x | x | x | X |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| S |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = up,up,up,up,up,up,up,up,up,right,right,right,
                     right,right,right,down,down,right,right,right,
                     down,down,left,left
```

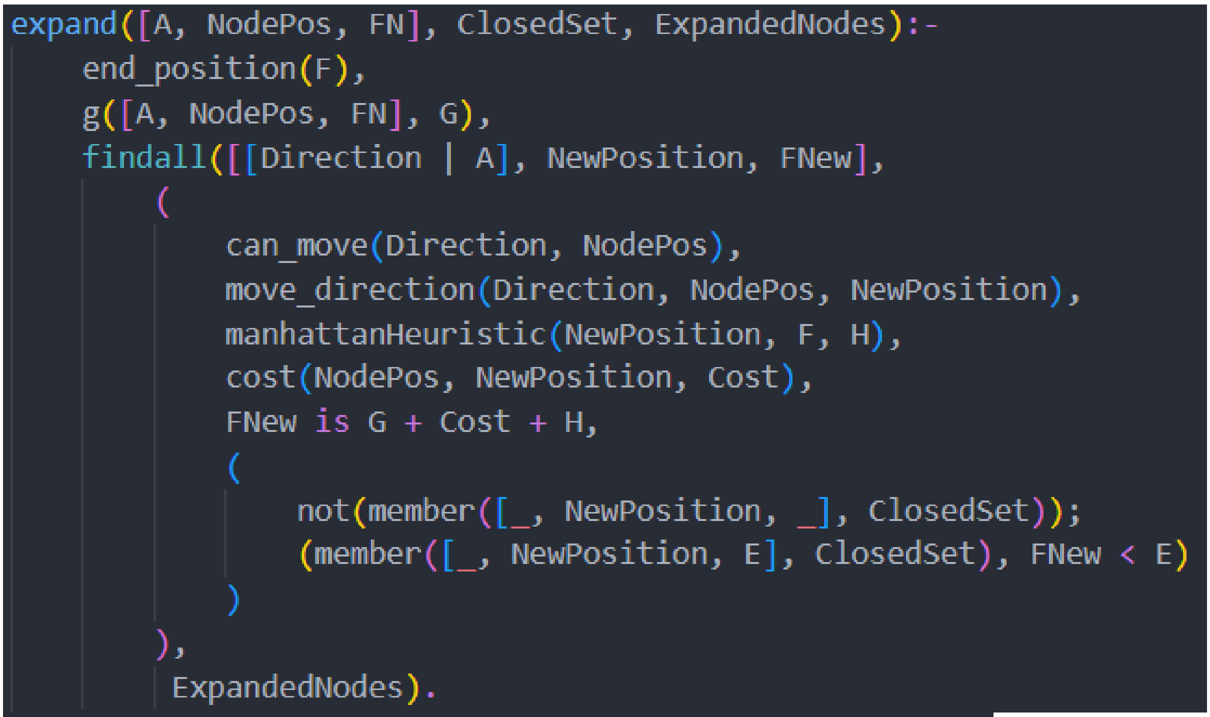
A* Search

Algorithm Path and Performance

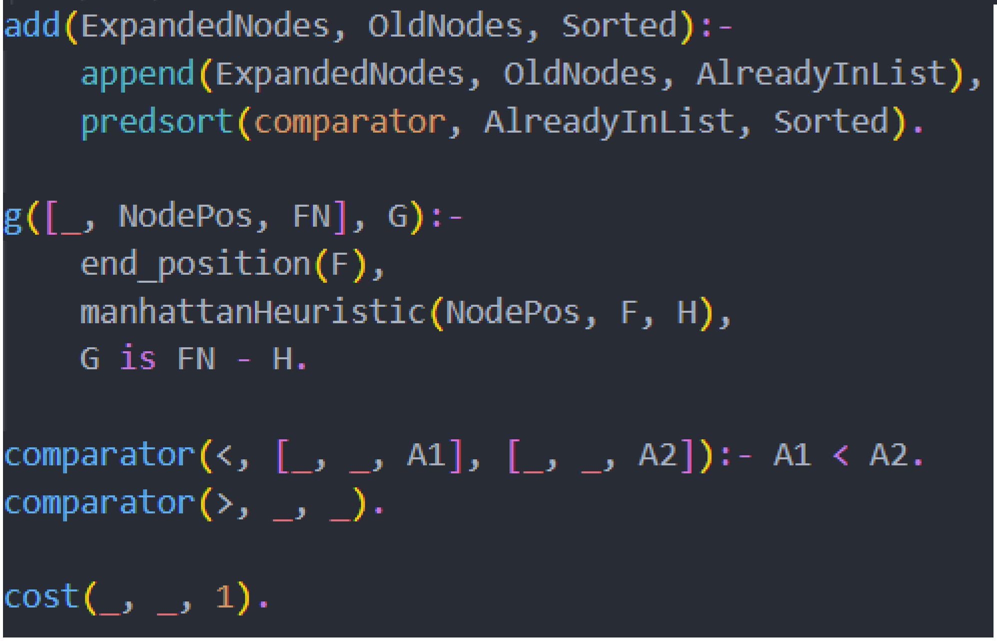
The following screenshot illustrates the though-process of implementing the A* search algorithm



```
/* Step 4 - Running the algorithm

   We will performing the following actions to obtain the solution to the maze:

   1. Obtaining the heuristic value of the algorithm by calculating the Manhattan distance
   2. Generating the children of a node/position & checking all the move directions allowed in that node/position.
   3. Using the calculated cost function, the node/position is added to the list of already generated children
      and the algorithm is recursively called again
   4. The final solution is then displayed
manhattanHeuristic(position(RowA, ColA), position(RowB, ColB), Distance):-
    Distance is abs(RowA - RowB) + abs(ColA - ColB).

solveMaze :-
    aStarSearchAlgorithm.

aStarSearchAlgorithm :-
    starting_position(Node),
    end_position(F),
    manhattanHeuristic(Node, F, H),
    useAlgorithm([ [[], Node, H] ], [], ReverseSolution),
    reverse(ReverseSolution, Solution),
    write("The solution for this maze is: "),write(Solution).

useAlgorithm([[A, NodePos, _] | _], _, A) :-
    end_position(NodePos).
useAlgorithm([Node | Tail], ClosedSet, Solution) :-
    expand(Node, ClosedSet, ExpandedNodes),
    subtract(ClosedSet, ExpandedNodes, NewClosedSet),
    add(ExpandedNodes, Tail, OrderedList),
    not(length(OrderedList, 0)),
    useAlgorithm(OrderedList, [Node | NewClosedSet], Solution).
```



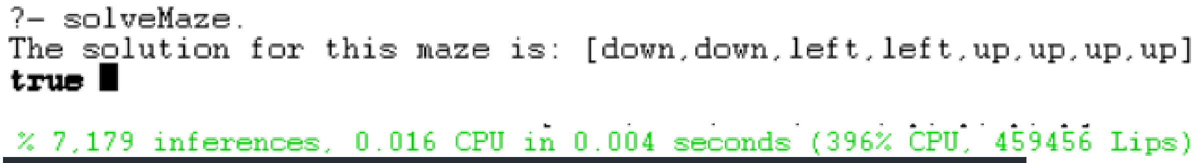
```
expand([A, NodePos, FN], ClosedSet, ExpandedNodes):-
    end_position(F),
    g([A, NodePos, FN], G),
    findall([[Direction | A], NewPosition, FNew],
        (
            can_move(Direction, NodePos),
            move_direction(Direction, NodePos, NewPosition),
            manhattanHeuristic(NewPosition, F, H),
            cost(NodePos, NewPosition, Cost),
            FNew is G + Cost + H,
            (
                not(member([_, NewPosition, _], ClosedSet));
                (member([_, NewPosition, E], ClosedSet), FNew < E)
            )
        ),
           ExpandedNodes).
```



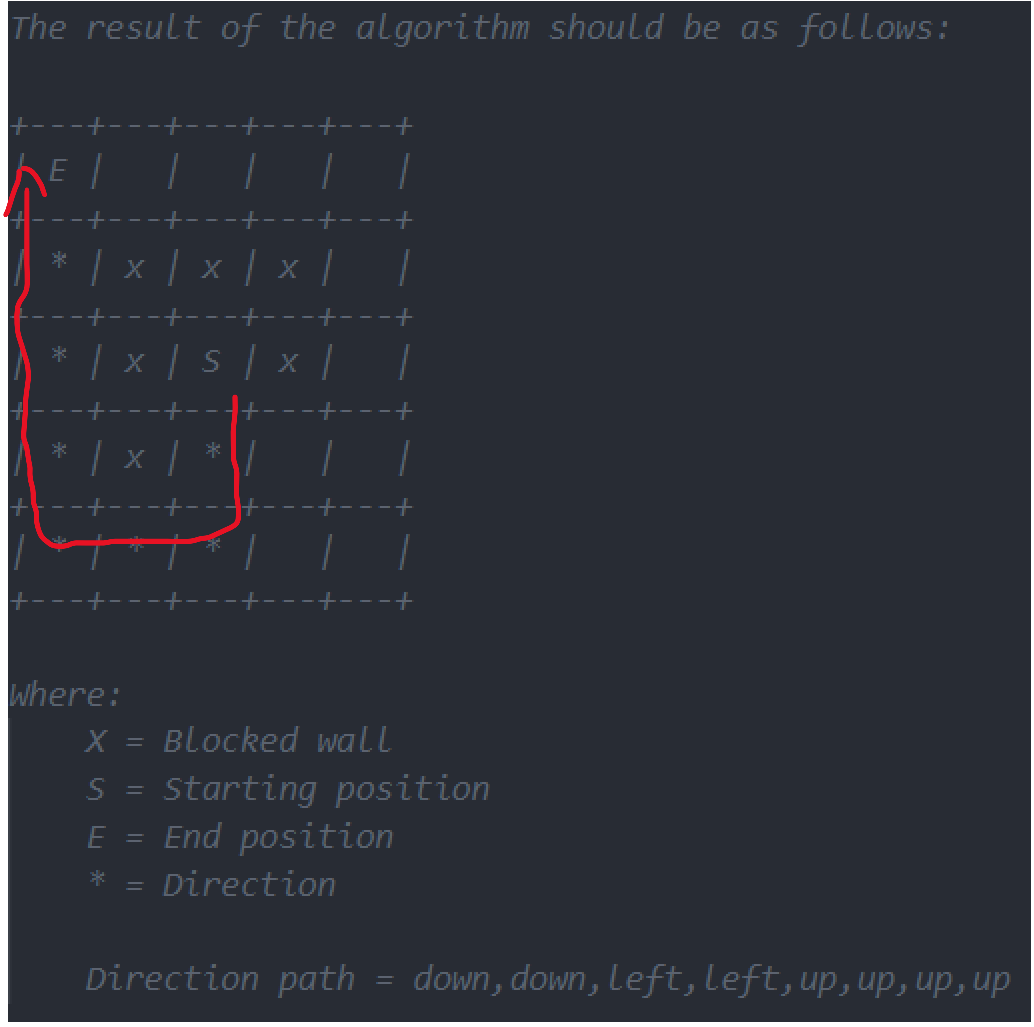
```
add(ExpandedNodes, OldNodes, Sorted):-
    append(ExpandedNodes, OldNodes, AlreadyInList),
    predsort(comparator, AlreadyInList, Sorted).

g([_, NodePos, FN], G):-
    end_position(F),
    manhattanHeuristic(NodePos, F, H),
    G is FN - H.

comparator(<, [_, _, A1], [_, _, A2]):- A1 < A2.
comparator(>, _, _).

cost(_, _, 1).
```

Sample Maze 1

The following is the output obtained from executing the A* search algorithm on Sample Maze 1, alongside the performance time and a figure of the path from the starting position to the end position.



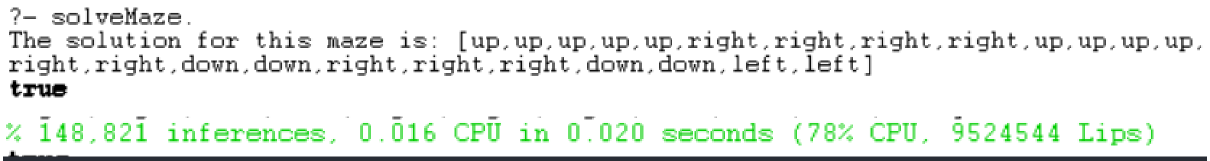
```
?- solveMaze.
The solution for this maze is: [down,down,left,left,up,up,up,up]
true

% 7,179 inferences, 0.016 CPU in 0.004 seconds (396% CPU, 459456 Lips)
```



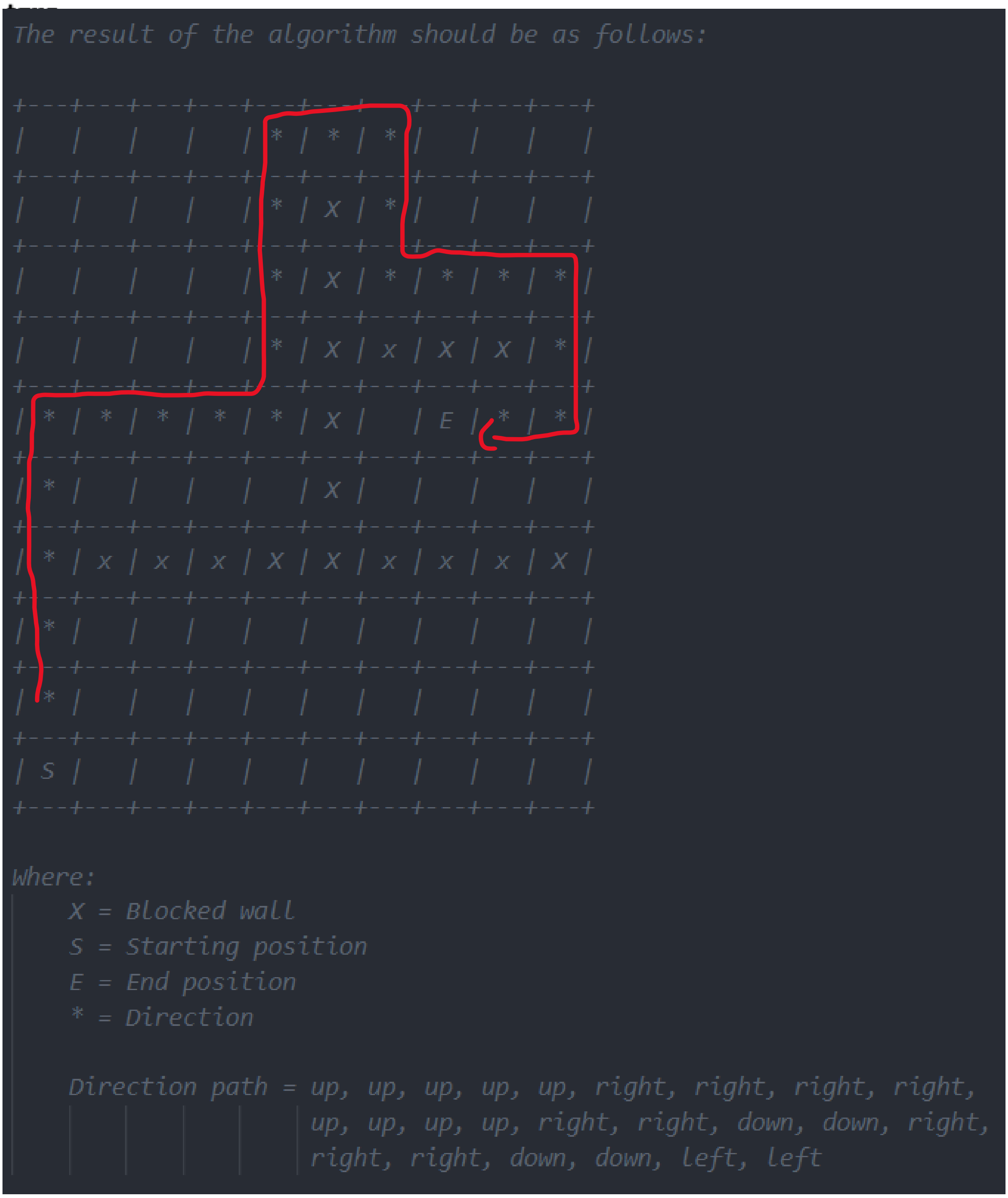
```
The result of the algorithm should be as follows:

+---+---+---+---+---+
| E |   |   |   |   |
+---+---+---+---+---+
| * | x | x | x |   |
+---+---+---+---+---+
| * | x | S | x |   |
+---+---+---+---+---+
| * | x | * |   |   |
+---+---+---+---+---+
| * | * | * |   |   |
+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = down,down,left,left,up,up,up,up
```

Sample Maze 2

The following is the output obtained from executing the A* search algorithm on Sample Maze 2, alongside the performance time and a figure of the path from the starting position to the end position.



```
?- solveMaze.
The solution for this maze is: [up,up,up,up,up,right,right,right,right,up,up,up,up,
right,right,down,down,right,right,right,down,down,left,left]
true

% 148,821 inferences, 0.016 CPU in 0.020 seconds (78% CPU, 9524544 Lips)
```



```
The result of the algorithm should be as follows:

+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * | * | * |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * | X | * |   |   |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * | X | * | * | * | * |
+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   | * | X | x | X | X | * |
+---+---+---+---+---+---+---+---+---+---+
| * | * | * | * | * | X |   | E | * | * |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   | X |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * | x | x | x | X | X | x | x | x | X |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| * |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+
| S |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+

Where:
    X = Blocked wall
    S = Starting position
    E = End position
    * = Direction

    Direction path = up, up, up, up, up, right, right, right, right,
                     up, up, up, up, right, right, down, down, right,
                     right, right, down, down, left, left
```