**FINAL REPORT**

**ARTIFICIAL INTELLIGENCE**

NAME: PRATYAY GUCHHAIT

SECTION: K18KK

ROLL NUMBER: 67

REGISTRATION NO.: 11808521

**Abstract**

The project (“Path Finding Maze”) is to design an algorithm to solve a given maze using A\* algorithm, which is one of the better algorithms to find between a start point to an end point. The A\* algorithm is a greedy approach to find a path as it searches for the lowest f(n) value and gives an optimal path.

Node (also called State) 🡪 All potential position or stops with a unique identification.

Transition 🡪 The act of moving between states or nodes.

Starting Node 🡪 Where to start searching.

Goal Node 🡪 The target to stop searching.

Search Space 🡪 A collection of nodes, like all board positions of a board game.

Cost 🡪 Numerical value (say distance, time, or financial expense) for the path from a node to another node.

F(n)🡪The total cost of reaching the final point.

G(n)🡪Actual cost of reaching the final point.

H(n)🡪Estimated cost reaching the final point.

**Introduction**

A maze is a puzzled way which consists of different branch of passages where the aim to reach the destination by finding the most efficient route within the shortest possible time. Artificial Intelligence plays a vital role in defining the best possible way of solving any maze effectively. Graph theory appears as an efficient tool while designing proficient maze solving techniques. Graph is a representation or collection of sets of nodes and edges. This concept is deployed in solving unknown maze consisting of multiple cells.

**Related Work**

* **Depth First Search (DFS):** It is an uninformed graph search algorithm. As from the name we can depicts, it first chooses one of the branch and starts searching until the end of the branch(leaf node) , when goal state is not reached it then backtracks and searches our branches. This is one of the drawbacks of this algorithm that it keeps on searching on side until it gets to the end of the branch. Using this algorithm in problems like solving a maze will take lot of time and maynot give us the best path to goal state.
* **A\* Algorithm:** It is an informed search algorithm. It finds the optimal path between the initial and the final state. The algorithm calculates f(n) value of every node using h(n) and g(n) values. This is one of the advantages of the algorithm ,it takes lesser time to reach a goal state and that to using optimal path. It searches for the node giving a lesser f(n) value and explores those nodes ignoring the nodes giving higher f(n) values.

**Implementation**

First, we will create a class node and call the function :

(1) **Class ‘Node’** that can be used to create an object for every node with information as the parent node, current position in the maze, and cost values (g, h & f).

(2) We need to define a **returnPath() function** that will return the path from start to end node that A\*.

We will establish a **search() function** which will be the drive the code logic:

(3.1) Initialization of all variables.

(3.2) Add the starting node to the “**yetToVisitList[]**”. Define a stop condition to avoid an infinite loop. Define movement in terms of relative position.

Repeat the following till stop criteria have met:

(3.3) Look for the lowest f cost square on the “**yetToVisitList[]**”. This square becomes the current square. We also check max iteration reached or not

(3.4) Check if the current square is the same as target square (meaning we have found the path)

(3.5) Use the current square and check four squares adjacent to this current square to update the children node. If it is not movable or if it is on the “**visitedList[]**”. ignore it. Otherwise, create the new node with the parent as the current node and update the position of the node.

(3.7) Check all the children node created to see

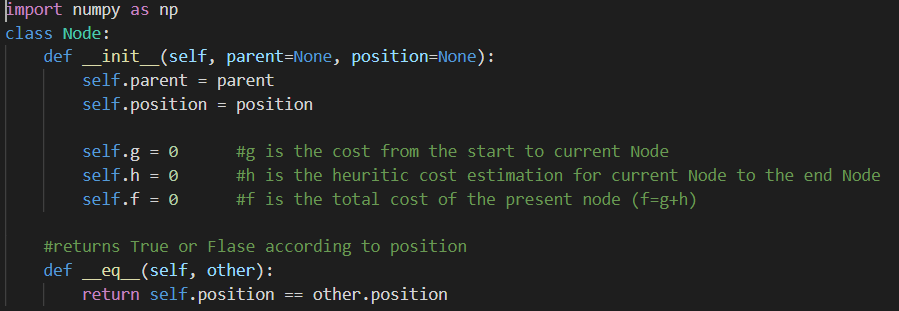
**·** If it isn’t on the “**yetToVisitList[]**”, add it to the “**yetToVisitList[]**”. Make the current square the parent of this square. Record the f, g, and h costs of the square.

**·** If it is in the “**yetToVisitList[]**” already, check to see if this path to that square is better, using g cost as the measure. A lower g cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the g and f scores of the square.

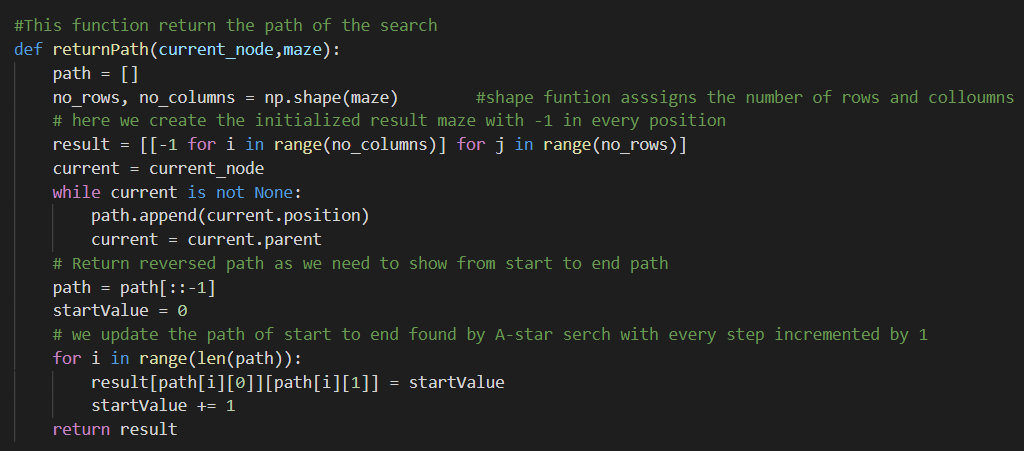
(4) **Main Program:** We will define maze, start, and end position. Then we will use the search function, and if a path exists, we can print the path from path function.

**Detailed Walk Through the Code**

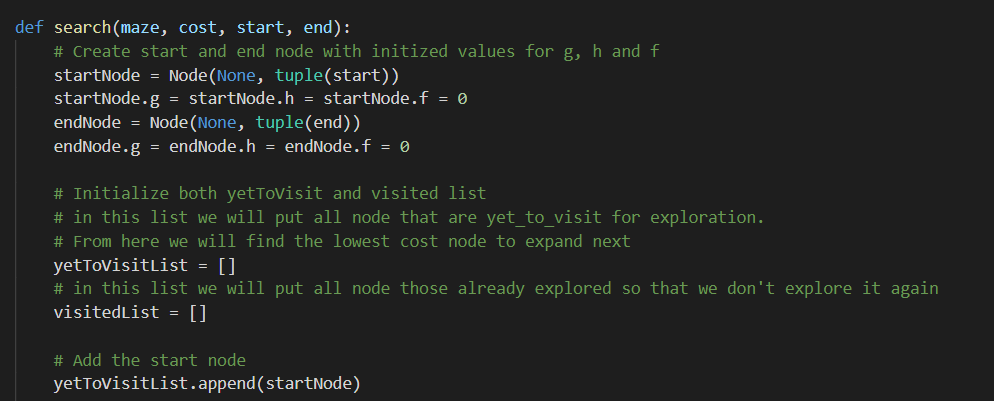
First, we will create a class for a node that will contain all the attributes associated with the node like the parent of the node, position of the node, and all three costs (g,h & f) for the node. We initialize the node and build a method for checking the equality of the node with another node.



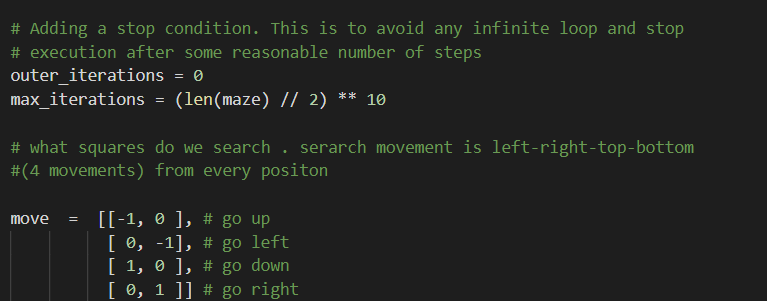
Now we will build the path function, which will be used to return the path from the start node to the target node (end node).



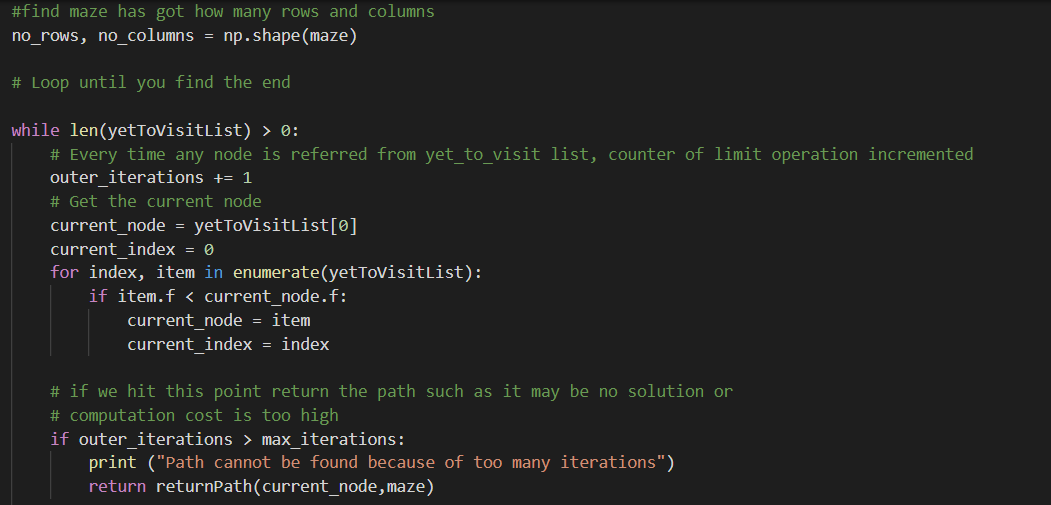
Now we will define the search function, which has multiple steps. The first step will be to initialize nodes and lists that we will use in the function.



Add the starting node to the “**yetToVisitList[]**”. Define a stop condition to avoid an infinite loop. Define movement in terms of relative position, which will be used to find the child node and other relative positions.



Now we use the current node by comparing all f cost and selecting the lowest cost node for further expansion. We also check max iteration reached or not, Set a message and stop execution (avoid infinite loop).



**Output/Screenshot**

