**MAZE SOLVER -**

**ARTIFICIAL INTELLIGENCE**

**PROJECT**

**A Report on:**

**MAZE SOLVER PROJECT**

Towards fullfilment of Artificial Intelligence

CSE Department

PES UNIVERSITY

**TEAM MEMBERS**

* Gourishankar (01FB16ECS123)
* Prathith (01FB16ECS270)
* Akilesh.S (01FB17ECS701)­
* Harshit.J.Dave (01FB17ECS712)
* Swaroop Raj Sheety (01FB17ECS729)

**INTRODUCTION**

For our project, we created a maze and having ball character solve the maze. The character has a manual algorithm encoded to implement A\* algorithm. Our program language of choice is  JAVA on a NetBeans IDE. We plan to test each AI’s efficiency by testing run time, analysing cost vs benefit, and blind surveys.

In order to complete this project, our group wrote our code using JAVA as our language of choice, utilising the NetBeans IDE. We chose to share our code over GitHub, but heavily dependent on Google Drive. The creation of the maze was executed using cellular automata, within an integer array. In the array, zeros represent, represent walls and one’s empty corridors. Later, threes represent the path the actor takes to solve the maze.

**IMPLEMENTATION**

**PROBLEM STATEMENT**

# To implement A\*(star) pathfinding algorithm to solve a maze in Java.

# A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

**A\* Properties:**

* It is complete; it will always find a solution if it exists.
* It can use a heuristic to significantly speed up the process.
* It can have variable node to node movement costs. This enables things like certain nodes or paths being more difficult to traverse, for example an adventurer in a game moves slower across rocky terrain or an airplane takes longer going from one destination to another.
* It can search in many different directions if desired.

**Explanation:**

What A\* Search Algorithm does is that at each step it picks the node according to a value-‘**f**’ which is a parameter equal to the sum of two other parameters – ‘**g**’ and ‘**h**’. At each step it picks the node/cell having the lowest ‘**f**’, and process that node/cell.

**g** = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.  
**h** = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way. There can be many ways to calculate this ‘h’.

**Exact Heuristics** –

We can find exact values of h, but that is generally very time consuming.

1) Pre-compute the distance between each pair of cells before running the A\* Search Algorithm.

2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance.

**Approximation Heuristics –**

There are generally three approximation heuristics to calculate h –

**1) Manhattan Distance –**

* It is nothing but the sum of absolute values of differences in the goal’s x and y coordinates and the current cell’s x and y coordinates respectively, i.e.,
* **h** = abs (current\_cell.x – goal.x) +

abs (current\_cell.y – goal.y)

* When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom)

**2) Diagonal Distance-**

* It is nothing but the maximum of absolute values of differences in the goal’s x and y coordinates and the current cell’s x and y coordinates respectively, i.e.,
* **h** = max { abs(current\_cell.x – goal.x),

abs(current\_cell.y – goal.y) }

* When to use this heuristic? – When we are allowed to move in eight directions only (similar to a move of a King in Chess)

**3) Euclidean Distance-**

* As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula
* **h** = sqrt ( (current\_cell.x – goal.x)2 +

(current\_cell.y – goal.y)2 )

* When to use this heuristic? – When we are allowed to move in any directions.

### **F score**

The f score is simply the addition of g and h scores and represents the total cost of the path via the current node.

***f(n)=g(n)+h(n)***

***// Pseudo Code of our project A\* in java:***

**Limitations:**  
Although being the best pathfinding algorithm around, A\* Search Algorithm doesn’t produce the shortest path always, as it relies heavily on heuristics / approximations to calculate – h

**Applications**:

Tower defence is a type of strategy video game where the goal is to defend a player’s territories or possessions by obstructing enemy attackers, usually achieved by placing defensive structures on or along their path of attack.

A\* Search Algorithm is often used to find the shortest path from one point to another point. You can use this for each enemy to find a path to the goal.

One example of this is the very popular game- Warcraft III

**Time Complexity**  
Considering a graph, it may take us to travel all the edge to reach the destination cell from the source cell.

The worst case time complexity is O(E), where E is the number of edges in the graph

**Auxiliary Space** In the worst case we can have all the edges inside the open list, so required auxiliary space in worst case is O(V), where V is the total number of vertices.

**//Screenshots of our project:**