**2D -Maze Solver Using A\* Algorithm**

SONAWAN E GAURAV  
*TY  
 PIT, VADODARA*

SAVALIYA HEMIL  
*TY  
 PIT, VADODARA*

*Abstract*—A maze is one type of puzzle where we have given starting point and destination point with obstacles. We have to find path from starting point to destination point. In this project Python Language is used and PyGame module is used for making GUI. We are using A\* Algorithm for finding minimal path from starting point to destination. The pathfinding algorithm solves the problem of determining the shortest path from origin to destination while avoiding obstacles. Agent movement is one of the most difficult challenges in designing realistic Artificial Intelligence (AI) in computer games. Pathfinding strategies are typically used as the foundation of any AI movement system. The A\* search algorithm is used in this work to find the shortest path between the source and destination on an image representing a map or a maze. Finding a way out of a maze is a fundamental computer science problem that can take many forms. Pathfinding and graph traversal make extensive use of the A\* algorithm. To test the system's performance, various map and maze images are used (100 images for each map and maze).The overall performance of the system is acceptable, and it can find the shortest path between two points on the images. More than 85% of images can find the shortest path between the two points of interest.

Keywords—PYTHON,PYGAME,A\*ALGORITHM, Artificial Intelligence.

# Introduction

Pathfinding could be used to answer the question, "How do I get from point A to point B?" Most of the time, the path from the source (current point) to the destination (next point) could include several different solutions, but if possible, the solution should be chosen must include the following objectives:

1. The route taken from point A to point B.

2. A method for getting around roadblocks.

3. The method for determining the shortest possible path.

4. A quick way to find the path.

In Computer Games path finding is very important .It is one of the hard and popular Artificial Intelligence problem. In Real World we have to use limited memory and CPU resources to solve this problem efficiently. For this reason we have to choose right path finding algorithm. In path finding we have to get from source to destination, to get around obstacles in the way, to find the shortest possible path, to find the path quickly. Pathfinding has become a popular and frustrating problem in the game industry as its importance has grown. Characters in role-playing and real-time strategy games are usually routed on a mission from their current location to a predetermined or player-determined destination.. Agent movement is one of the most difficult challenges in creating realistic AI in computer games. Pathfinding strategies are commonly used as the foundation of any AI movement system. The most common pathfinding challenge in video games is avoiding obstacles while looking for the most beneficial path through various locations. Every year, the modern computer game industry grows in size and complexity, both in terms of map size and the number of units in the industry. For path finding some of the popular algorithms are A\* Search Algorithm, BFS and DFS. Maze is a puzzled technique in which the maze is discovered by the solver by taking the most efficient route in the shortest amount of time to the destination. In fact, the maze is widely distinguished from the labyrinth because the labyrinth has a single through route with twists and turns but no branches and is not intended to be difficult to understand. The pathways and walls in the maze are fixed. The majority of maze solving algorithms are strongly related to graph theory, with mazes without loops being analogous to trees in graph theory. When there are multiple solutions to a maze, the solver can determine the shortest path from source to destination. We are implementing A\* Algorithm in this project for finding minimal path.

# Path Search ing Algorithms

Process to reach from source to destination is called Path Searching . The shortest path problem is one of the most extensively researched in computer science literature. The problem, given a weighted graph, is to find the minimum total weight path in the graph between two nodes. For various variants of the problem, several algorithms have been developed. Edges that are directed versus undirected are examples of variations. A graph is made up of nodes and arcs that connect them, and a labelled graph has one or more descriptions attached to each node that differentiates it from any other node in the graph. Blind search and heuristic search are two types of graph searches. Uniformed search which is also called Blind search where we do not have knowledge of its domain. There is another search technique which is heuristic search where there is study of the algorithms and rules of discovery and development. Heuristics can solve given problem but there is no guarantee of solution.

Why use the A\* Search Algorithm?

A\* Search Algorithm is a straightforward and efficient search algorithm for determining the best path between two nodes in a graph. It will be used to find the shortest path. Dijkstra's shortest path algorithm (Dijkstra's Algorithm) is an extension of it. The difference here is that instead of using a priority queue to store all of the elements, we use heaps (binary trees). The A\* Search Algorithm also employs a heuristic function, which provides additional information about how far we are from the goal node. This function is used in conjunction with the f-heap data structure to improve the efficiency of searching.

## Methodology

Artificial intelligence’s challenging problems in computer games.

1)Complex decision spaces: Most of the state-of-the-art computer games involve complex strategic(real-time strategy games) or believable behaviour (interactive dramas). The two kinds of behaviours share the characteristic of obtaining huge decision spaces.

2)Authoring support: Hand crafted behaviours are ultimately software code in a complex programming language, at risk from human errors.

3)Unanticipated situations : It is impossible to plan for all possible situations and player strategies that may arise during game play.

4)Knowledge engineering : Even assuming that strategies or behaviours are handcrafted, authoring these kinds of behaviour sets in a game requires a huge human engineering effort.

5)Replay ability and variability : Player might get bored of seeing the same strategies and behaviours again and again.

6)Rhetorical objective: It is possible, that human engineered behaviours or strategies do not achieve the game objectives completely.

A\* Search Algorithm

A\* is a generic search algorithm that is used to solve a variety of problems, one of which is path searching. It employs heuristic and uniform-cost search features. In path searching, the A\* algorithm returns to the most promising unexplored locations it has seen. When a location is searched for, the A\* algorithm completes when that location is the goal. The A\* algorithm is a popular path-finding algorithm in artificial intelligence.

What is Heuristic ?

We can easily calculate g, but how do we compute h?

We can compute the value of h using one of two methods:

1. Determine the exact value of h. (which is certainly time-consuming).

2. Use a variety of techniques to approximate the value of h. (less time-consuming).

Let us go over both approaches.

1. Exact heuristics

Although we can get exact values of h, it usually takes a long time.

The methods for determining h's exact value are listed below.

a. Calculate the distance between each pair of cells before using the A\* Search Algorithm.

b. In the absence of blocked cells or obstructions, we can directly determine the precise value of h using the distance formula/Euclidean Distance.

2) Approximation heuristics

There are typically three approximation heuristics for determining h:

a. Manhattan Distance

The Manhattan Distance is the sum of the absolute values of the differences between the current and goal cells' x and y coordinates.

The equation is as follows: h = abs (curr cell.x - goal.x) + abs (curr cell.y - goal.y)

When we are only allowed to move in four directions - top, left, right, and bottom - we must use this heuristic method.

b. Diagonal Distance

It is simply the greatest absolute value of the differences between the x and y coordinates of the current and goal cells.

This is summarized in the formula below -

dx = abs(curr cell.x - goal.x) dy = abs(curr cell.y - goal.y) dx = abs(curr cell.y - goal.y)

h = D \* (dx + dy) + (D2 - 2 \* D) \* min(dx, dy), where D is the node length (default = 1) and D2 is the diagonal length.

When we are only allowed to move in eight directions, such as the King's moves in chess, we use this heuristic method.

c. Euclidean Distance

Using the distance formula, the Euclidean Distance is the distance between the goal cell and the current cell:

h = sqrt ( (curr\_cell.x – goal.x)^2 + (curr\_cell.y – goal.y)^2)

# Working of a\*

Each node has three additional attributes: fitness, goal, and heuristic, denoted by the letters f, g, and h, respectively. 1) g Denote the cost of travelling from the source to the destination node (the summation of all values in the path between the source and destination). 2)h Represent the estimated cost of connecting the source and destination nodes. 3) f is the sum of g and h and is the best estimate of the cost for the path through the source node. f = g + h. The goal of g, h, and f is to determine how promising a path is up to the current node. Paths between nodes will be assigned different values. These values would typically represent the distances between the nodes. Distance is not required to be the cost between nodes. If you wanted to find the path that takes the least amount of time to traverse, the cost could be time. A\* with two lists (open list and closed list). The open list contains all nodes in the map that have not yet been completely explored. The closed list includes all nodes that have been thoroughly explored. Marker arrays, in addition to the standard open/closed lists, can be used to determine whether a state is in the open or closed list.

A\* Algorithm

1. Assume that P is the source node.

2. Give P g, h, and f values.

3. Add the source node to the list of open nodes.

4. Repetition of the following steps:

A. On the open list, look for the node with the lowest f. This node is referred to as the current node.

B. Put it on the closed list.

a) Ignore it if it is on the closed list.

b) If it isn't already on the open list, add it and then make this node the child of the current node. Take note of this node's g, h, and f values.

c) If it is already on the open list, see if this is a better path. If so, change it to the current node's parent and recalculate the g and f values.

C. For each node reachable from the current node

D. End when

a) The destination node is added to the closed list.

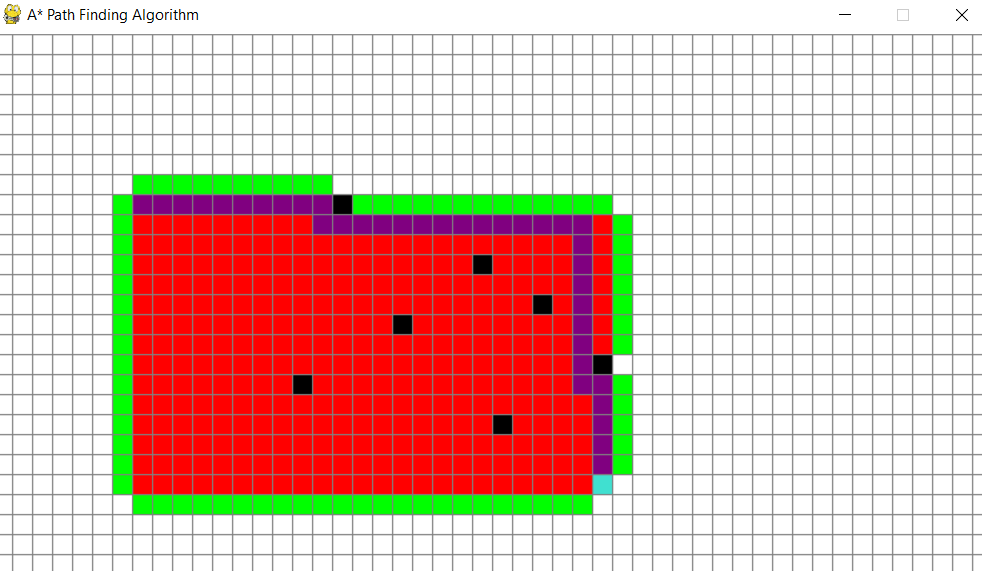
b) The destination node cannot be found, and the open list is empty.

5. Tracing from the destination node back to the source node That is the way to go.

## **Pygame**

* For creating video games Pygame a cross-platform set of Python modules is used.
* It gives computer graphics and sound libraries designed to be used with the Python programming language.
* Pygame was officially made by **Pete Shinners** to replace PySDL.
* Pygame can be used to create client-side applications that can be packaged into a standalone executable.

##### Result



Conclusion

In AI games, the basic core of the pathfinding algorithm is only a small piece of the puzzle. The most pressing issue is determining how to apply the algorithm to difficult problems. .A\* algorithm is commonly used for path finding. In this work, A\* search algorithm is used to find the shortest path between source and destination. More than 85% can find the shortest path between the selected two points. Other search algorithms can be used and each algorithm has different characteristics.

##### References

[1] Lawrence, R. and Bulitko, V. (2013) Database-Driven Real-Time Heuristic Search in Video-

Game Pathfinding. IEEE Transactions on Computational Intelligence and AI in Games, 5,

227-241. http://dx.doi.org/10.1109/TCIAIG.2012.2230632

[2] Algfoor, Z.A., Sunar, M.S. and Kolivand, H. (2015) A Comprehensive Study on Pathfinding

Techniques for Robotics and Video Games. International Journal of Computer Games

Technology, 2015, Article ID: 736138. http://dx.doi.org/10.1155/2015/736138

[3] Botea, A., Bouzy, B., Buro, M., Bauckhage, C. and Nau, D. (2013) Pathfinding in Games.

Dagstuhl Follow-Ups , 6, 21-31.

[4] Graham, R., McCabe, H. and Sheridan, S. (2015) Pathfinding in Computer Games. The ITB

Journal , 4, 57-81.

[5] Lagzi, I., Soh, S., Wesson, P.J., Browne, K.P. and Grzybowski, B.A. (2010) Maze Solving by

Chemotactic Droplets. Journal of the American Chemical Society , 132, 1198-1199.

http://dx.doi.org/10.1021/ja9076793