CS575: Final Project Report

Project Title: MAZE GAME
Team Member(s): Mohammed Arif Mohammed Ali Khan, Kartik Khergade, Disha Shetty

I. PROBLEM

To figure out a path from one point to the chosen destination block in n*n matrix. Basically, the user has the liberty to choose the size of the matrix and has to find a path to reach at his destination given that the chosen path is optimal. Now here the user can only move to 2 directions from its source: either forward or down (given the block is the first one). Now along the way there might be weights or cost placed in each block and whenever the user comes to that block, the weights or the costs gets added. There are many algorithms which we can use to calculate the optimum path but the ones which we are going to use are A* algorithm along with the visualisation.

II. ALGORITHMS

A. Maze Generation – Recursive Backtracking

Recursive Backtracking is one of the easiest maze generator algorithm. The first step of this algorithm is to choose an initial cell and check whether there is a possibility of visiting the neighbouring blocks of the chosen cell, if yes; then pick any random neighbour and put in the grid and mark the path to the neighbour. Repeat the process till we come to a point where a block does not have any neighbours to mark. We mark the once visited cell as visited and then. We keep on doing this until we reach the starting cell and empty and that will generate our resulting maze.

B. Maze Generation – Recursive Division

The "recursive division" algorithm is one that must be implemented as a wall adder. This algorithm is particularly fascinating because of its fractal nature: you could theoretically continue the process indefinitely at progressively finer and finer levels of detail. Initially, you start with an empty field. Then bisect the wall, either horizontally or vertically and add a single passage through the wall and keep on doing this for the areas of either sides of the wall. Call recursively till the maze reaches its desired location.

C. Maze Generation – Sidewinder

Side winder algorithm is closely related to the Binary tree algorithm but manages to get away with only one side being spanned by a passage. In sidewinder, we start from block place at 0^{th} row and 0^{th} column and work our way through towards the east side and randomly decide whether you need to create a passage at east or north side of the block (for the last block you cannot create a passage at the east side as the maze goes out of bound; same for the top row of the maze). Continue the above point till all the rows have been processed.

D. Maze Solving – A Algorithm*

A* is considered to be one of the efficient algorithm for path-finding or graph traversals. This algorithm is efficient to find the shortest path between two points. Unlike BFS and DFS, A* is said to be informed search algorithm i.e. it will take into account the location of the goal before deciding the next move. A cost or weight is associated with these block and A* will try to take the one with the minimum cost. The heuristic function $f(n) = g(n) + h(n)$ wherein f(n) is the cost of the block; g(n) is the actual cost from the start block to n and h(n) is the heuristic function that estimates the cost of the cheapest pat from n to the goal.

III. SOFTWARE DESIGN AND IMPLEMENTATION

A. Software Design

Here we have created separate modules for each algorithm and have imported all those modules in driver script which is main.py and then used them there; main.py when executed, asks user to choose from algorithms they want to generate the maze, after the user is asked to enter the dimensions of the maze they want generated. Then the generated maze is displayed, and after pressing any key, the maze is removed from the screen and user is asked if they want the maze solved through A* Algorithm. If yes, then the solution of maze is displayed on the screen. We have used arrays, lists, priority queues, matrices and dictionaries thorough the implementations of algorithms and driver code.

Recursive Backtracking generates the maze by doing the following steps:
1) Select a random point to start.
2) Choose a random adjacent point. Only create a passage if that point is not been visited before.
3) Continue doing same process over and over again until there are no more adjacent points to choose from.
4) Then trace back your steps until you are at a point from where you can choose an adjacent point again.
5) The generation of maze is completed when we are at starting cell again.

Working of Recursive Division:
1) Start in an empty grid with no walls.
2) Create a wall in the random place bisecting the grid into two.

3) Create a passage thorough the wall at a random place and you will get the valid maze every time you complete these two steps.
4) Repeat steps 2 and 3 until you have a maze with desired detail.

Working of sidewinder:
Sidewinder considers one row of the grid at a time so these steps are for every row:
1) Assign ever point in the row a different set number.
2) For each point randomly decide whether to create a passage to the East of the point.
3) If the passage to the east is created add the point to the set of previous point.
4) If the passage to the east is not created, randomly pick one point from the current set, create a passage leading North and Go to the next set of the row and do steps 2,3 and 4 again until row is completed.

A* Algorithm works as follows:
It's working is as follows:
1) Start at the given starting point.
2) Select an open neighbouring point having least f value to go, f value of a point depends upon the distance of that point from goal or end point.
3) Repeat step 2 until you've reached the goal or end point.

B. Implementation and Tools Used

Here, A* is implemented in Python as per the algorithm. For the visualization purpose, we use mathplotlib package from Python.

.

IV. PROJECT OUTCOMES

- https://drive.google.com/file/d/1ufOV-V0Ji5NHWjkrFFga5z69GJHlKM7z/view?usp=share_link

V. REFERENCES

[1] https://www.educative.io/answers/what-is-the-maze-problem
[2] https://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm#:~:text=The%20%E2%80%9Crecursive%20division%E2%80%9D%20algorithm%20is,Begin%20with%20an%20empty%20field
[3] https://weblog.jamisbuck.org/2011/2/3/maze-generation-sidewinder-algorithm.html
[4] https://en.wikipedia.org/wiki/A*_search_algorithm
[5] https://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm
[6] https://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking
[7] https://www.youtube.com/watch?v=elMXlO28Q1U&t=282s&ab_channel=EasyLearnTutorial