Maze Generator & Solver Documentation

Authors:

- Ahmed Magdy Elhussiney ID: 221001359

- Amr Hamouda ID: 221001776

- Mohamed Goma ID: 221001823

Our Project is about generating a Random Maze by DFS algorithm and solving it my BFS and A\* Algorithm that consider the Maze as a graph and trying to get the path to the end point. We stored the nodes of graph in a list, and we display it as a gride and change its color while traversing it to display the path that the Algorithm takes.

For DFS generating Maze Algorithm:

First, we set up a grid of cells where each cell starts as a wall. Begin at a starting cell and mark it as part of the maze (visited). then utilized a stack to keep track of the path. From the current cell, if it has any unvisited neighboring cells, it chooses one randomly, removes the wall between the current and chosen cell, and push the current cell onto the stack before moving to the chosen cell. Mark the new cell as visited and continue from this cell. If the current cell has no unvisited neighbors, we backtrack by popping the last cell from the stack and making it the current cell. Repeat this process until the stack is empty, indicating all possible paths have been explored. While iterate through the cells, we used Pygame to draw the maze generation in real-time by updating the display after each cell is visited or a wall is removed, allowing us to visualize the maze construction dynamically.

BFS () function:

The “BFS ()” function implements the Breadth-First Search (BFS) algorithm to find a path from the start cell to the end cell in a maze. It initializes a deque (double-ended queue) with the start cell and a dictionary ‘came\_from’ to keep track of the path. The function enters a loop that continues until the queue is empty. In each iteration, it dequeues the current cell and checks if it is the end cell; if so, it breaks the loop. It then examines the neighboring cells (top, right, bottom, left) that are accessible (i.e., do not have walls). For each valid neighbor that has not been visited, it appends the neighbor to the queue, draws a rectangle on the screen to visualize the BFS progress, and updates the ‘came\_from’ dictionary to record the current cell as the predecessor of the neighbor. The color used for visualization gradually changes by incrementing ‘colorBFS\_Range’. The screen is updated, and the program pauses briefly to create an animation effect. Finally, the function returns the ‘came\_from’ dictionary, which contains the path from the start to the end cell.

For the A\* Function:

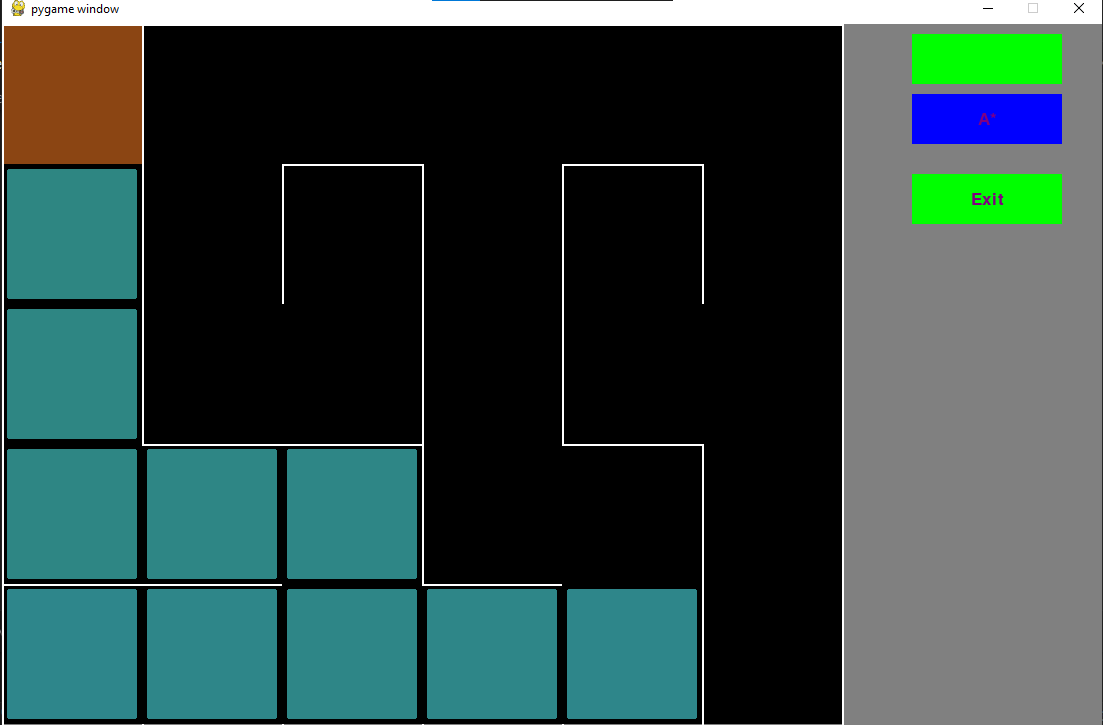
We execute the A\* algorithm to solve a maze. It begins by creating a grid of cells, where each cell represents a node in the maze. The function ‘heuristic’ calculates the Manhattan distance between two nodes. In the ‘a\_star’ function, a priority queue is used to manage nodes to explore, with ‘g\_score’ and ‘f\_score’ dictionaries tracking the cost from the start node and the estimated cost to the end node, respectively. The algorithm explores nodes, updating scores and paths based on neighboring cells that are not blocked by walls. If a lower-cost path to a neighbor is found, the neighbor's scores are updated, and it is added to the queue for further exploration. Visual feedback is provided by drawing the path on the screen using Pygame, and the function returns the ‘came\_from’ dictionary, which can be used to reconstruct the path from start to end.

Snapshots From Program:

A screenshot of a computer

Description automatically generatedGenerating The Maze:

Solving Maze By BFS:



A screenshot of a game

Description automatically generated

A screenshot of a computer game

Description automatically generatedMaze Solved by BFS:

Solving Maze by A\*:A screenshot of a computer game

Description automatically generated

A screenshot of a computer game

Description automatically generatedMaze Solved by A\*:

Video for our Program:

<https://youtu.be/9O894IfWV7I>

References

- Pygame Documentation (https://www.pygame.org/docs/)

- Python Standard Library (https://docs.python.org/3/library/)

- Maze Generation Algorithms

(https://en.wikipedia.org/wiki/Maze\_generation\_algorithm)

- A\* Search Algorithm

(https://en.wikipedia.org/wiki/A\*\_search\_algorithm)

- [Breadth-First Search Algorithm

(<https://en.wikipedia.org/wiki/Breadth-first_search>)

We got some ideas about how to stablish the GUI of the algorithm from this video:

<https://youtu.be/Ez7U6jU0q5k?si=llso5Whq8Twr9E8a>