Path Finder in Maze

What’s the shortest route from our place to the nearest park?
How do we find a way through a maze?
Can we program a game character to find the exit avoiding enemies?

Finding a path from source to destination avoiding obstacles and minimizing the cost (time, distance, risks, fuel, path).

Pathfinding is closely related to the shortest path problem, within graph theory, which examines how to identify the path that best meets some criteria (shortest, cheapest, fastest, etc) between two points in a large network.

Introduction:

The Path Finder in Maze module is designed to visualize complex maze generation and implement four different shortest path algorithms. The module is developed using Python and utilizes the pygame package for graphical representation. The maze generation is accomplished through Randomized Prim's Algorithm, while the four shortest path algorithms implemented are Dijkstra's Algorithm, Breadth First Search, A* Algorithm, and Q-Learning.

Maze Generation:

The module employs Randomized Prim's Algorithm to generate mazes. This algorithm starts with a grid of cells and gradually builds walls between adjacent cells, forming a maze. The algorithm randomly selects a starting cell, marks it as part of the maze, and adds its neighboring walls to a list. It then proceeds by repeatedly choosing a random wall from the list, removing it, and marking the cell on the opposite side as part of the maze. This process continues until all cells are included in the maze.

Shortest Path Algorithms:

The Path Finder in Maze module implements four different shortest path algorithms:

a. **Dijkstra's Algorithm:**
   Dijkstra's Algorithm is a graph search algorithm that finds the shortest path between nodes in a graph. It works by maintaining a priority queue of nodes and their tentative distances from the start node. The algorithm explores the graph by repeatedly selecting the node with the smallest tentative distance, updating the distances of its neighbors, and marking the visited nodes.

b. **Breadth First Search:**
   Breadth First Search is another graph traversal algorithm that explores all the vertices of a graph in breadth-first order. It starts at the initial node and systematically explores the adjacent nodes before moving to the next level of nodes. By doing so, it finds the shortest path in an unweighted graph.

c. **A* Algorithm:**
   The A* Algorithm combines the best features of Dijkstra's Algorithm and heuristic search. It uses an evaluation function that estimates the total cost of a path by combining the cost to reach the current node and an estimated cost to reach the goal. A* Algorithm intelligently explores the graph by prioritizing nodes with lower total costs, leading to efficient pathfinding.
d. **Q-Learning:**
   Q-Learning is a reinforcement learning technique that uses a table of Q-values to make decisions in an environment. It is based on the principle of trial and error learning, where an agent interacts with the environment, receives rewards or penalties, and updates its Q-values accordingly. Q-Learning can be applied to maze-solving problems to learn the optimal path from a start state to a goal state.

Conclusion:

The Path Finder in Maze module provides a comprehensive platform for visualizing maze generation and implementing four different shortest path algorithms. It utilizes Python and the pygame package to create an interactive environment where users can observe the maze generation process and compare the effectiveness of various pathfinding algorithms. This module is beneficial for educational purposes, algorithm exploration, and understanding the concepts of maze solving and pathfinding.