GP Coding Assignment: Evolving a Maze Solver

Background:

In this assignment, you will use **genetic programming** to evolve an algorithm that can navigate a maze. The maze will be represented as a **grid**, where each cell can be either a **wall** or an **open space**. The task is to evolve a set of instructions (using a genetic programming tree) that allows an agent to move from a **start point** to a **goal point**.

This problem will involve evolving control programs for a robot or agent to find a path through a maze, **avoiding walls** and obstacles. The agent will be able to move in four directions: **up**, **down**, **left**, or **right**. It will be represented by a **tree structure** where nodes in the tree encode movement instructions.

Problem Statement:

You are tasked with evolving a maze-solving agent using genetic programming. The agent must be able to navigate from a start position to a goal position in a maze using a sequence of movement instructions (up, down, left, right).

Input:

A maze grid, where:

0 represents an empty space (walkable).

1 represents a wall (not walkable).

The agent starts at a given position (e.g., (start_x, start_y)).

The goal is at a target position (e.g., (goal_x, goal_y)).

Task:

Evolve a program (as a genetic program tree) that can correctly navigate from the start position to the goal.

The evolved program will output a sequence of moves (up, down, left, right).

The GP should **optimize for the shortest valid path** to the goal.

Approach:

The GP algorithm will evolve individuals that represent a sequence of movement instructions to guide the agent through the maze. The movement instructions can be encoded as a series of decisions based on the agent's environment (such as "move forward", "turn left", etc.).

Detailed Requirements:

1) Maze Representation:

The maze is represented as a 2D grid with 0 for open spaces and 1 for walls.

The agent starts at a given location (start_x, start_y) and aims to reach (goal_x, goal_y).

2) Agent Actions:

The agent can move in one of four directions: up, down, left, or right.

Each agent's genome is a tree where:

Internal nodes represent decision-making **operations** (e.g., move forward, turn left, turn right).

Leaf nodes represent **conditions** (e.g., check if there is a wall in front of the agent, check if the goal is reached).

3) Fitness Function (minimization):

The fitness function measures the **path length** (number of steps) to reach the goal from the start.

Penalty should be added for invalid paths (e.g., when the agent hits a wall or loops back on itself).

The agent's fitness will be evaluated based on how well it navigates the maze.

If the agent reaches the goal, the fitness will be **0** (optimal solution). If it takes too many steps or hits walls, the fitness will increase.

Suggested fitness formula:

F = 0 if the agent reached the goal

F = s + 2d + 10w + 5l otherwise

s: number of steps taken

d: ManhattanDistance(agent_final_pos, goal)

w: number of wall hits

l = number of revisits (loops)

4) Initialization: **ramped half-and-half**, where grow & full method each deliver half of initial population

5) Parent selection: fitness proportional

6) Survivor selection: generational or steady-state

7) Crossover: exchange of subtrees (crossover rate: user-defined)

8) Optional mutation: random change in trees (mutation rate: user-defined)

9) Termination:

The algorithm terminates either when an individual finds the goal with **optimal fitness** (0 fitness) or after a **certain number of generations**.

10) Maze Generator:

```
maze = [
    [0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
    [1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
    [1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 1, 1, 1, 1, 0, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 1, 1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 1, 1, 1, 1, 0, 0, 0, 0]
]
```

Start: **(0, 0)** >> top-left of the matrix

Goal: **(9, 9)** >> bottom-right of the matrix

Suggested GP parameters:

POPULATION_SIZE = 200

MAX_GENERATIONS = 100

MAX_TREE_DEPTH_INIT = 4

MAX_TREE_DEPTH = 8

MAX_STEPS_PER_EPISODE = 60

Deliverables:

- **Python** code for the genetic programming algorithm that evolves maze-solving solutions.

- A **description** of how the genetic programming algorithm works, including the representations of individuals, fitness function, mutation, and crossover strategies.

- **Visualization**: Plot the progression of average fitness and max fitness over generations.

- Sample output: Show examples of evolved solutions, i.e., the sequences of moves generated by the GP algorithm.

Evaluation Criteria:

- Correctness: Does the agent correctly navigate the maze?

- Efficiency: Does the agent find the goal in the shortest number of steps?

- Code Quality: Is the code modular, clean, and well-documented?