COSC2658 - Data Structures and Algorithms/COSC2469 - Algorithms and Analysis/COSC2203 - Algorithms and Analysis

Group Project (SAMPLE)

Assignment Overview:

This assignment involves designing and implementing a system for an autonomous robot to navigate through a dark maze without any prior knowledge of its structure. The robot will start at a random point and can move in four directions (UP, DOWN, LEFT, RIGHT), receiving feedback after each move (true for a successful move, false for hitting a wall, and win upon finding the exit). The maze is defined as a 2D grid with walls, empty spaces, and a single exit. The task is to develop a `Robot` class with a `navigate()` method that will guide the robot through the maze effectively and efficiently.

Preparation Advice:

1. **Understand the Problem**: Make sure you fully understand the maze structure, robot movements, and the type of feedback received after each move.
2. **Study Examples**: Review the examples/solutions shared on your course's GitHub page to understand the base code you are allowed to extend.
3. **Java Basics**: Brush up on your Java skills, especially focusing on object-oriented concepts since you cannot use the Java Collection Framework.
4. **Algorithmic Thinking**: Practice algorithmic design paradigms like brute force, greedy algorithms, and others that may be relevant to this problem.
5. **Data Structures Knowledge**: As you can't use the Java Collection Framework, you need to be comfortable creating custom data structures.

Problem Requirements:

- Create a `Robot` class without using the Java Collection Framework.
- Develop a `navigate()` method to move the robot through the maze using a `go()` method that returns movement feedback.
- Implement custom data structures to track visited locations and maze walls.
- Your solution should be as efficient as possible, minimizing the number of moves to reach the exit.
- The maze and robot's starting point are unknown to your program and will be tested with unseen mazes.

Step-by-Step Solution with Detailed Explanation:

1. Data Structures:

Since you cannot use the Java Collection Framework, you'll need to design your own data structures.

- **Maze Representation**:

```
public class MazeNode {
    boolean isWall;
    boolean isExit;
    boolean visited;
    // Constructor, getters and setters
}

public class CustomMaze {
    private MazeNode[][] grid;
    // Initialize the maze grid with the size of 1000x1000
    // Constructor and necessary methods
}
```

- **Tracking Visited Nodes**:

```
public class VisitedTracker {
    private boolean[][] visited;
    // Constructor and methods to mark visited nodes and check if a node is visited
}
```

2. Maze Navigation Algorithm:

A good algorithm for maze navigation could be a modified Depth-First Search (DFS) with backtracking.

- **DFS Algorithm**:

```
public void navigateDFS(int currentRow, int currentCol, CustomMaze maze) {
    if (!isValid(currentRow, currentCol, maze)) {
        return;
    }
    if (maze.getNodeAt(currentRow, currentCol).isExit) {
        // Exit found
        return;
    }
    maze.visit(currentRow, currentCol);
    // Recursive calls for UP, DOWN, LEFT, RIGHT
    navigateDFS(currentRow - 1, currentCol, maze); // UP
    navigateDFS(currentRow + 1, currentCol, maze); // DOWN
    navigateDFS(currentRow, currentCol - 1, maze); // LEFT
    navigateDFS(currentRow, currentCol + 1, maze); // RIGHT
    // Backtracking
    maze.unvisit(currentRow, currentCol);
}
```

  - The `isValid` method will check if the current position is within the bounds of the maze and not a wall.
  - The `isExit` attribute will be set to true for the node that represents the exit gate.

- **Backtracking Mechanism**: Backtracking is inherent to the recursive nature of the DFS. After visiting a node, the function will recursively visit adjacent nodes. If a path doesn't lead to the exit, it will backtrack.

CONTINUE....

↳ 4 cells hidden