Mars Rover Simulation Project Report

Project Overview

The Mars Rover Simulation is a project that simulates the movement of a rover on a grid, accounting for obstacles that prevent the rover from advancing. The rover can perform the following actions:

1. Move forward based on its current direction.
2. Turn left or right to change its orientation.
3. Detect obstacles on the grid and stop its movement if one is encountered.

The simulation implements the Command design pattern to encapsulate rover actions, and the Singleton pattern to ensure a single instance of the grid is used.

Design Patterns Used

1. Command Pattern

Intent: The Command pattern is used to encapsulate all the actions that the rover can perform (moving forward, turning left, turning right, and getting its position) into separate classes. This makes it easy to queue commands and execute them without knowing the exact details of what each command does.

Why it's used: The Command pattern allows us to:

- Decouple the sender (e.g., simulation or client code) from the receiver (the rover), making the system more flexible.
- Easily extend functionality by adding new commands without modifying existing code.
- Store a list of commands and execute them in sequence, which is perfect for simulating the rover's movement.

2. Singleton Pattern

Intent: The Singleton pattern ensures that only one instance of the grid exists throughout the simulation, preventing issues that could arise from multiple instances.

Why it's used: The grid needs to be shared between the rover and the simulation, as all commands executed on the rover depend on its position on the grid. Having multiple grid instances could cause inconsistencies in obstacle placement and detection.

Detailed Code Walkthrough

1. File 1: Command Interface and Rover Class

The Rover class is responsible for storing the rover's position (x, y) and direction ('N', 'S', 'E', 'W'). It provides methods to move forward, turn left, turn right, and retrieve the current position. The rover interacts with the grid to check for obstacles before moving. The concrete command classes (MoveForwardCommand, TurnLeftCommand, TurnRightCommand, and GetPositionCommand) implement the Command interface and delegate their actions to the rover.

2. File 2: Grid Class

The Grid class is designed using the Singleton pattern to ensure that only one instance exists. It defines the grid's width and height, tracks obstacles, and provides methods to add obstacles and check for their presence. This ensures that all commands that rely on grid information (such as obstacle detection) refer to the same grid instance.

3. File 3: MarsRoverSimulation Class

This class acts as the entry point for the simulation. It initializes the grid and the rover,

then creates a list of commands. The simulation executes the commands sequentially, simulating the rover's movement
and obstacle detection in a real-world scenario. At the end of the simulation, the rover's final position and direction are printed.

Conclusion

This project effectively demonstrates the use of the Command and Singleton design patterns.
The Command pattern provides flexibility and scalability by encapsulating the rover's actions,
while the Singleton pattern ensures consistency in grid management across the simulation.