Mars Robot Challenge - Developer Instructions

Introduction

The following coding test is provided to you as a step through your application to Olympic Channel development department as part of the AI-Enhanced Development Team.

The exercise consists of the development of a small sample application that will illustrate your skills in the different areas involved in design, coding, good practices and patterns, as well as your ability to effectively collaborate with AI tools.

You are encouraged to use any AI tools you have access to (ChatGPT, Claude, GitHub Copilot, etc.) to assist with development, code generation, testing, and documentation. We're interested in seeing how you leverage these tools strategically while maintaining code quality and applying human judgment.

Please read the instructions carefully before beginning coding and do not hesitate to reach your point of contact in case you have any doubts or questions.

What You Are Expected to Produce and Timeline

You'll have a full week (7 days) to complete the assignment once you receive it.

The test should take from 3 to 8 hours to develop, the week is provided so that you can adapt the completion of the test to your schedule and timetable as you like. You're of course free to dedicate as much extra time as you want to it. Time of development will not be measured, only quality of code and good practices.

You will create an application with multiple execution modes (CLI, REST server, REST client) and comprehensive documentation, delivered as a Git repository with meaningful commit history.

The Problem

The Olympic Channel has recently developed a Mars Surveillance Robot with the intent of exploring the surface on Mars and explore suitability for future Olympic Sports on low gravity over there.

Once the robot has landed on Mars, the robot will enter in listening mode for instructions which will be provided as a set of commands to execute which may include moving forward (F) or backwards (B), turning (L or R), taking samples (S), extending the solar panels for getting energy (E).

At every moment, the robot contains an internal battery which provides a certain capacity. Each of the commands will consume a given quantity of the battery.

Robot Commands & Battery Consumption:

- **Move Forward (F)**: Consumes 3 battery units. Move the unit one square forward in the current facing direction

- **Move Backwards (B)**: Consumes 3 battery units. Move the unit one square backwards from the current facing direction
- **Turn Left (L)**: Consumes 2 battery units. Changes the facing direction 90º to the right
- **Turn Right (R)**: Consumes 2 battery units. Changes the facing direction 90º to the left
- **Take Sample (S)**: Consumes 8 battery units. Takes and stores a sample of whatever material is primary in the current location
- **Extend Solar Panels (E)**: Consumes 1 battery unit. Recharges 10 battery units

To better test the behavior of the robot, a functionality is required which will accept a set of commands, a starting position, a starting battery and a map of the surface composed as a n×m matrix of coordinates which will indicate the characteristics of the terrain in Mars.

Terrain Types:

- **Fe**: Ferrum. A deposit of iron
- **Se**: Selenium. A deposit of selenium
- **W**: Water. A deposit that contains water
- **Si**: Silicon. A deposit that contains silicon
- **Zn**: Zinc. A deposit that contains zinc
- **Obs**: An obstacle cell in which the robot can't go

Whenever the robot detects an obstacle ahead of the execution of the command, it must automatically apply a back off strategy *instead*, in order to continue with the execution. The robot contains a list of back off strategies to try in order, if the execution of 1 strategy results in hitting another obstacle, the robot will jump to the next strategy (battery will nonetheless be consumed):

1. E, R, F
2. E, L, F
3. E, L, L, F
4. E, B, R, F
5. E, B, B, L, F
6. E, F, F
7. E, F, L, F, L, F

Important Rules:

- **Boundaries**: Terrain boundaries are considered obstacles
- **Coordinate System**: Use standard x,y coordinates where terrain[y][x] maps to position (x,y)
- **Battery Management**: If battery is insufficient for a command but sufficient to Extend Solar Panels, automatically extend them, otherwise stop execution and return current state
- **Obstacle Detection**: Back off strategies trigger when movement commands (F/B) would hit obstacles or boundaries

The robot will then produce a simulation run of the results obtained containing:

- The set of cells visited
- The set of samples collected
- The current battery level

Input/Output Format

Input:

```
{
  "terrain": [["Fe", "Fe", "Se"], ["W", "Si", "Obs"]],
  "battery": 50,
  "commands": ["F", "S", "R", "F"],
  "initialPosition": {
    "location": {"x": 0, "y": 0},
    "facing": "East"
  }
}
```

Output:

```
{
  "VisitedCells": [{"X": 0, "Y": 0}, {"X": 1, "Y": 0}],
  "SamplesCollected": ["Fe"],
  "Battery": 32,
  "FinalPosition": {
    "Location": {"X": 1, "Y": 0},
    "Facing": "South"
  }
}
```

Core: Implementation Requirements (Required)

The result of the test should consist of:

- A console program in any object-oriented language of your choice
- A backend REST service which takes the JSON input as a POST request and returns the simulation output
- A REST client utility for testing the service
- A README file describing how to run and test your application, as well as an explanation of the decisions taken (language choice, design decisions, AI tool usage if applicable, etc.)
- You can develop your solution using Windows, Mac or Linux

The solution should produce console applications with the following execution modes:

- `obs_test input.json output.json` - CLI mode processing files
- `obs_test` - starts the REST API server (no parameters)

- `obs_test_post input.json` - REST client that posts to the server and displays formatted output

Please provide shell scripts, batch files, or executable binaries as needed for cross-platform execution.

Please focus on producing production-ready code, that is, code that does not only work but would be what you consider to be ready to be put into production.

We will value all standard good coding practices and patterns, a good design of the solution, proper error handling, and clean separation of concerns. Extra characteristics like using source control, comprehensive testing, etc., are extra points but not required for the solution.

Execution Modes (All Three Required)

1. CLI Mode:

```
obs_test input.json output.json
```

- Reads JSON input from file, processes simulation, writes JSON output to file

2. REST API Server Mode:

```
obs_test
```

- Starts web service, accepts POST requests with JSON input, returns JSON response

3. REST API Client Mode:

```
obs_test_post input.json
```

- Reads JSON input from file, makes POST request to server, displays formatted JSON output

Implementation Requirements

- Console application in your chosen language
- Same simulation logic across all execution modes
- Proper error handling for invalid inputs and network failures
- Clean separation between robot logic and interface layers

Submission Requirements

- ☐ Working solution meeting all requirements
- ☐ All three execution modes functional
- ☐ Test suite covering functionality
- ☐ README with setup instructions
- ☐ Clean Git repository with meaningful commit messages, logical commit structure, and proper .gitignore
- ☐ Brief notes on AI tool usage: which tools you used for what tasks, and examples of how you reviewed, modified, or improved AI-generated output (if applicable)

Optional Extensions

These extensions are intentionally open-ended to encourage AI tool usage for rapid prototyping and implementation.

Robot Visualization

Create a simple web-based visualization showing robot movement on the terrain. A basic 2D top-down view is sufficient - showing the grid, robot position, path taken, and samples collected. Use any web technology you prefer (HTML5 Canvas, SVG, React, etc.).

Interactive CLI Visualization

Create an interactive CLI interface where you can input commands and see the updated terrain and robot state. A basic text-based display is sufficient - showing the terrain grid as ASCII characters, robot position and facing direction, and current battery/samples. Use any CLI library or framework available in your chosen language.

Pathfinding Intelligence

Create a pathfinding system that analyzes terrain and generates optimal command sequences for the robot. Consider battery constraints, obstacle avoidance, and mission objectives. Implement algorithms for:

- Exploring the entire map and collecting samples of each terrain type
- Navigating from any point A to point B efficiently

Use any approach you prefer (A*, Dijkstra, custom algorithms, etc.). The system should integrate with your existing robot simulation.

Your Own Extensions

You can implement any extensions that you may think of. If there is a skill or technology in which you are proud of and want to share, you are more than welcome.