# Introduction

## Coding Test

The following coding test is provided to you as a step through your application to Olympic Channel development department as part of the Backend Developer.

The exercise consists on the development of a small sample application that will illustrate your skills on the different areas involved in design, coding, good practices and patterns.

Please do 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 should have received

As part of this assignment you should have received the following files:

* OCS\_CodingTest\_Backend\_2.pdf: This document
* Test\_run\_1.json and test\_sol\_1.json: A sample input and output for testing.
* Test\_run\_2.json and test\_sol\_2.sjon: Another sample input and output for testing.

Note the provided test files are just examples and should not be considered representative of comprehensive testing.

# What you are expected to produce and how long you’ll have

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

The test should take from 2 to 6 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.

The result of the test should consist on:

* A console program in C#, Java, node.js, Kotlin or any object-oriented language of your choice.
* A backend REST service which takes the json file as a POST request and returns the output.
* A readme file describing how to run and test your application, language used as well as an explanation of the decisions taken (language of choice, design decisions, etc)
* You can develop your solution using windows, mac or linux.
* The solution should produce a console application named obs\_test which can be executed using “obs\_test” command line (please provide shell scripts or batch files as needed) which takes a single input path as the input and a single output path as the output. Therefore, the solution should either:
* obs\_test c:\mypath\jsonInput.json c:\output\jsonOutput.json to run the console program with input and output.
* obs\_test without parameters to run the REST API

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, etc. Extra characteristics like using source control, containers, etc, are extra points but not required for the solution.

# 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.

* 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 nxm matrix of coordinates which will indicate the characteristics of the terrain in mars.

The terrain described on the map can contain the following:

* 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

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.

Examples

Example – Input

{

  "terrain": [

    ["Fe", "Fe", "Se"],

    ["W", "Si", "Obs"],

    ["W", "Obs", "Zn"]

  ],

  "battery": 50,

  "commands": [

    "F", "S", "R", "F", "S", "R", "F", "L", "F", "S"

    ],

  "initialPosition": {

    "location" : {

      "x" : 0,

      "y" : 0

    },

    "facing" : "East"

  }

}

The terrain represents a multidimensional array describing the resources on the terrain.

The array represents the terrain in a direct way, with 0 based indexes, that is, in the example above, for X = 1 and Y = 0 we have “Fe” and for X = 0 and Y = 1 we have “W”.

Battery represents the starting battery for the robot.

Commands represents the set of commands to execute.

InitialPosition represents the initial position of the robot. Location (X, Y) represents the starting position of the robot in the 0 based array as described above.

Facing represents the initial facing direction of the Robot, there are 4 possible valid values:

* North,
* East
* South
* West

Example – Output

Running the Example above will produce the following output:

{

    "VisitedCells": [

        {

            "X": 0,

            "Y": 0

        },

        {

            "X": 1,

            "Y": 0

        },

        {

            "X": 1,

            "Y": 1

        },

        {

            "X": 0,

            "Y": 1

        },

        {

            "X": 0,

            "Y": 2

        }

    ],

    "SamplesCollected": [

        "Fe",

        "Si",

        "W"

    ],

    "Battery": 8,

    "FinalPosition": {

        "Location": {

            "X": 0,

            "Y": 2

        },

        "Facing": "South"

    }

}

* **VisitedCells** fields contains all the fields that the robot has traversed on its execution without any repetitions.
* **SamplesCollected** contains all the fields that the robot has taken samples of, including duplicates, that is, If the robot samples Water twice then it should appear twice.
* **Battery** contains the remaining battery of the unit after the execution
* **FinalPosition** contains the final position of the robot after the execution.

# How to deliver the results

In order to deliver the results please answer to the email with either:

* The solution attached, including instruction on how to execute and test it. Please make sure you don’t include elements susceptible of being intercepted by the antivirus (i.e. executable binaries). Ideally the solution should be able to be run in the simulator by npm run or npm start.
* A link to a private GitHub repository (or similar) with enough information in it to execute and test your solution (i.e. via de usual readme.md format on GitHub)