# Mars Rover Design

## Key Points

|  |  |  |
| --- | --- | --- |
| Points | Source | Comments/Assumptions |
| Plateau is square or rectangular | Brief | Flat – 2d |
| Plateau is divided into a grid represented by X Y coordinates and N S W E represent directions. 0, 0 is the bottom left corner. N is up. | Brief |  |
| Rovers have cameras and robot arms | Brief | Assume rover has a camera and has a robot arm. Composition |
| Rovers move | Brief |  |
| Rovers move when it receives a move command  L – turn left 90 deg; R – Turn R 90 deg; M – Move forward one grid point | Brief | Could also take an input for a 45 degree turn. L2 = turn left 45 deg |
| First input line of 2 numbers represents upper right coordinates | Brief | Size of the plateau effectively. Discard additional numbers but continue. End if coordinates are negative or not numbers |
| Next lines are input for rovers.  2 lines for each rover  First line is position with 2 integers and compass letter | Brief | Skip rover if position is invalid |
| Next line is a string of letters to move the Rover | Brief |  |
| Rovers move sequentially | Brief | Only one rover is moving at a time |
| Output is the final position – coordinates and compass point | Brief |  |
| Rovers are expensive and should not collide or fall off the plateau | Slack | Rovers need to know the plateau size;  When a rover receives its instructions, it will calculate its route and if it will collide or fall off the plateau it will refuse to move.  Rovers will acknowledge a command that can be executed with an ok response and a command that cannot be followed with a negative response. Rovers will broadcast their position after each move (and on creation) |
| Solution should be a real product | Slack |  |

## Class Diagram

**Vehicle**

-id:String

-map: Map

-currentPosition: Position

-direction: Direction

-vehicles: Position[]

+move():String

+turn(direction: Direction):String

+broadcastPosition(): Position

+receiveVehiclePosition(): void

**MarsRover**

-instruments: Instrument[]

+calculatePath():Boolean

**Direction**

-compassPoint : String

**Position**

-x: number

-y: number

**Map**

+isPositionInMap(position:Position):Boolean

**RectangularMap** (extends Map)

-width: number

-height: number

**Instrument**

**Camera** extends Instrument

+captureImage()

+startRecording()

+stopRecording()

**RobotArm** extends Instrument

+open()

+close()

+move(x: number, y: number, z: number)