rover.py

```
from constants import *

class Rover:
    def __init__(self):
        self.position = []
        self.direction = 'N'
        self.boundaries = []
        self.roversplateau = []


    """
    Moves a rover one grid point.
    Rover cannot move if it is at the edge of the grid and is directed to move outside
    Or if another rover is in it's path/grid point it is trying to move into
    """
    def doMove(self):
        if self.direction == NORTH and self.position[1] < self.boundaries[1] and all(r.position !=
[self.position[0], self.position[1] + 1] for r in self.roversplateau):
            self.position = [self.position[0], self.position[1] + 1]
        elif self.direction == SOUTH and self.position[1] > -1 and all(r.position != [self.position[0],
self.position[1] - 1] for r in self.roversplateau):
            self.position = [self.position[0], self.position[1] - 1]
        elif self.direction == EAST and self.position[0] < self.boundaries[0] and all(r.position !=
[self.position[0] + 1, self.position[1]] for r in self.roversplateau):
            self.position = [self.position[0] + 1, self.position[1]]
        elif self.direction == WEST and self.position[0] > -1 and all(r.position != [self.position[0] - 1,
self.position[1]] for r in self.roversplateau):
            self.position = [self.position[0] - 1, self.position[1]]


    # Rover spin 90 degrees into the direction passed in parenthesis if valid
    def doRotate(self, dirct):
        self.direction = dirct if (VALIDCOMMANDS.find(dirct) > -1 or VALIDDIRECTIONS.find(dirct)
> -1) else self.direction


    # Move or spin rover based on the command passed
    def processCommand(self, cmd):
        if cmd == MOVE:
            self.doMove()
        elif cmd == RIGHT:
            if self.direction == NORTH:
                self.doRotate(EAST)
            elif self.direction == SOUTH:
                self.doRotate(WEST)
            elif self.direction == EAST:
                self.doRotate(SOUTH)
            elif self.direction == WEST:
                self.doRotate(NORTH)
        elif cmd == LEFT:
            if self.direction == NORTH:
```

```
            self.doRotate(WEST)
        elif self.direction == SOUTH:
            self.doRotate(EAST)
        elif self.direction == EAST:
            self.doRotate(NORTH)
        elif self.direction == WEST:
            self.doRotate(SOUTH)



def parseCommand(cmd):
    commands = cmd.splitlines()

    boundaries = commands[0].split(' ')
    commands.pop(0)

    instructions = []
    while len(commands) > 0:
        instrPerRover = commands[0:2]
        instructions.append(instrPerRover)
        del commands[0:2]

    roversOnMars = []
    for i in instructions:
        rover = Rover()
        rover.boundaries = list(map(int, boundaries))
        if len(roversOnMars) > 0:
            rover.roversplateau = roversOnMars

        for cm in i:
            a = cm.split(' ')
            for c in a:
                if len(c) > 1:
                    for char in c:
                        rover.processCommand(char)
                else:
                    isCoordPoint = c.isdigit()
                    if isCoordPoint:
                        if len(rover.position) < 1:
                            rover.position.append(int(c))
                        else:
                            rover.position.append(int(c))
                    elif VALIDCOMMANDS.find(c) > -1:
                        rover.processCommand(c)
                    elif VALIDDIRECTIONS.find(c) > -1:
                        rover.direction = c
        roversOnMars.append(rover)
    return roversOnMars
```

test_rover.py

```
from rover import *
import unittest

class TestRover(unittest.TestCase):
    def setUp(self):
        rover1 = Rover()
        rover1.direction = NORTH
        rover1.position = [1, 3]

        rover2 = Rover()
        rover2.direction = EAST
        rover2.position = [5, 1]

        rover3 = Rover()
        rover3.direction = SOUTH
        rover3.position = [2, 2]

        self.expected = [rover1, rover2, rover3]
        self.result = parseCommand('5 5\n1 2 N\nLMLMLMLMM\n3 3 E\nMMRMMRMRRM\n2 3
W\nMMLM')
        self.expected


    def test_rover_position(self):
        self.assertEqual(self.result[0].position, self.expected[0].position)
        self.assertEqual(self.result[1].position, self.expected[1].position)

    def test_rover_direction(self):
        self.assertEqual(self.result[0].direction, self.expected[0].direction)
        self.assertEqual(self.result[1].direction, self.expected[1].direction)

    #A rover cannot move if another rover is in the path that it needs to into
    def test_rover_not_crash_in_rover_path(self):
        self.assertEqual(self.result[2].direction, self.expected[2].direction)
        self.assertEqual(self.result[2].position, self.expected[2].position)



if __name__ == "__main__":
    unittest.main()
```

constants.py

```
VALIDDIRECTIONS = 'WNES'
NORTH = 'N'
SOUTH = 'S'
EAST = 'E'
WEST = 'W'
VALIDCOMMANDS = 'LRM'
LEFT = 'L'
RIGHT = 'R'
MOVE = 'M'
```