Battleship

We are going to play a special version of the very well known game Battleship. The purpose of the game is to sink all of your opponent's ships before your opponent sinks yours.

In this special version of the game, there will be only one board and the game will be played automatically after a sequence of commands is interpreted. Furthermore, unlike in the original game, here the ships can move between one turn and the other.

Considerations

- The board of the game will be rectangular.
- For simplicity the ships will occupy only one square.
- Each ship position and location is represented by a combination of 'x' and 'y' coordinates and a letter representing one of the four cardinal compass points.
- An example position might be 0, 0, N, which means the ship is in the bottom left corner and facing North.
- In order to move a ship, a simple string of letters will be sent. The possible letters are 'L', 'R' and 'M'. 'L' and 'R' make the ship spin 90 degrees left or right respectively, without moving from its current spot. 'M' means move forward one grid point, and maintain the same heading.
- To try to sink a ship (that is, to shoot at a square of the board) the coordinates of the square will be sent.
- A square can't be occupied by more than one ship but a during a movement a ship can navigate through a square occupied by a ship.

Input

- The first line of the input is the upper-right coordinates of the board, the lower-left coordinates are assumed to be 0, 0. The format of the input will be `(X, Y)`. This sets the size of the board.
- The second line of the input is information about the initial deployment of the ships. It will be a space separated list of positions and orientations: `(X1, Y1, O1) (X2, Y2, O2) … (XN, YN, ON)`
- The rest of the input will contain different lines which will contain either ship movements or shots:
    - Ship movement. The format will be the position of the ship followed by the list of movements: `(X, Y) LLRRLRM`
    - Shot. The format will be the position shot: `(X, Y)`

Output

The output for each ship should be its final coordinates and heading and if they are sink or not.

Example test case

Test Input:

```
(5, 5)
(1, 2, N) (3, 3, E)
(1, 2) LMLMLMLMM
(2, 3)
(3, 3) MRMMRMRRM
(1, 3)
```

Expected Output:

```
(1, 3, N) SUNK
(4, 1, E)
```

Technical considerations

- You can deliver any kind of application (command line application, web application or even mobile application).
- You can use your preferred programming language to solve this exercise.
- You need to provide some instructions so we can run the exercise in our own environment (Mac or Linux, no Windows please as we don't have any Windows box).

What you need to submit

- Source code
- Unit tests that cover ~~most~~ all of the functionality
- Validation against input data
- Demonstrated error handling
- Comments as deemed necessary
- Documentation (as necessary)
- Any assumptions that you have made

Optionally, you can also include:

- Brief explanation of your solution (such as design considerations)
- Github url of the repo where you have the exercise
- Class diagrams