Toy Robot Simulation Analysis

Objetives

Design a solution that let take control over the free movement of a multiples toy robot through a complete set of commands and over a playground.

The solution must allow send commands directly to the robot from shell commands or from a file.

Optionally could be added a remote console to handle the robot remotely.

Requirements

- The playground should be by default and square **of 5x5 units** with **origin in the left bottom corner**.
- Initially there's no obstacles and collisions shall not be detected.

Commands

The commands available in the current version will be:

1. **HELP:** Show help of the use available commands.
2. **USE [Id]:** Set the active Robot over the commands are executed. USE or USE 0 deactivate the current active robot so next PLACE command create a new one.
3. **LIST:** List all robots existing in the parking, show the Ids and the position if it is placed on the table.
4. **PLACE x,y,F:** Generate and place a Robot on the table and initialize it by setting an initial position a facing to a specified direction. Initialize a log movement history. Implies an USE Id command so all commands after this one will assume as the active Robot the Id created by the PLACE command, until an USE command be executed with another Id.
5. **MOVE:** Move, the active Robot, and one step in the facing direction. Based on a configurable **StepSize**.
6. **LEFT:** Rotate the face of the active Robot 90º to the left.
7. **RIGHT:** Rotate the face of the active Robot 90º to the right.
8. **REPORT:** Show the current position of the active Robot in format [x,y,F]
9. **SHOWLOG:** Show the movement history since the last PLACE command for the active Robot. In a format of a sequence of [x,y, F]
10. **REBOOT [Id]:** Reset the active Robot to the origin position and clear the log history.
11. **REMOVE [Id]:** Remove active the Robot from the playground clearing the movement history. This command does not destroy the Robot, only remove from the playground. So it can place again by using a PLACE command. Using a PLACE command without a previously USE Id command create a new Robot.
12. **DESTROY [id]:** Destroy active the Robot. Eliminate all of his data and liberate resources used by it.
13. **QUIT:** Liberate all the resources used by the application and exit.

Arguments and parameters

- "**F**" will be based on cardinal points: **North, South, East and West.**

Movement

The movement of the robot is produced by facing the robot to a specified direction using the LEFT and RIGHT commands any times needed to get to face that direction, and sending a MOVE command any times needed to move the robot forward a specific number of steps.

Movement orientation will refer to cardinal points: **North, South, East and West**.

Constraints

List of constraints for this version:

- Multiple robots at a time are allowed, but could be configure to 1.
- Only 10 robots could be placed at a time. (**Configurable**).
- Movements over the limits of the playground are allowed. (**Configurable**)
- No obstacles on the playground.
- Could be placed more than one Robot at a time, but collisions are not detected. (**Configurable**)
- All commands will be ignored if a robot exit but it's not placed in the playground.
- All movement commands which cause a position outside the playground will be ignored.

Specifications

This project will be developed based on **.NET Core 2.1** because of its **crossplatform** feature among other interesting capabilities. The use of .NET Framework 4.7 does not add any relevant plus features and limit the publishing only on windows platforms.

The architecture of the Robot simulator solution trying to separate responsibilities and layers, parts are:

1. **Robot:** Class library responsible to handle the robot behaviour and apply the changes to the robot, is used directly by the controller library.
2. **Controller:** Class library which can handle Robots, configure the environment and apply the constraints to commands. The controller is also responsible of parsing commands input.
3. **Configuration:** Class library responsible to load and parse configuration values to the controller options.
4. **Shell Client:** A console application to received input commands and handle Robots using the controller.
5. **Commander API:** A **Rest web service** as a commander http API.
6. **Remote Client:** Any type of application which can handle the Robot using the commander API through SSL.

Because .NET Core applications need a launcher, the deployment provides 2 launch batch files: one for window and one for Linux. But is possible running the application using simply the next code: "**dotnet Lm.ToyRobot.Shell.dll**" from inside the deployment folder.

Log

A history of movements will be save for each Robot instance, recording movements from the PLACE command.

Movements history will be cleared by the **REBOOT**, **REMOVE** or **DESTROY** commands.

In this version the log will save the next data:

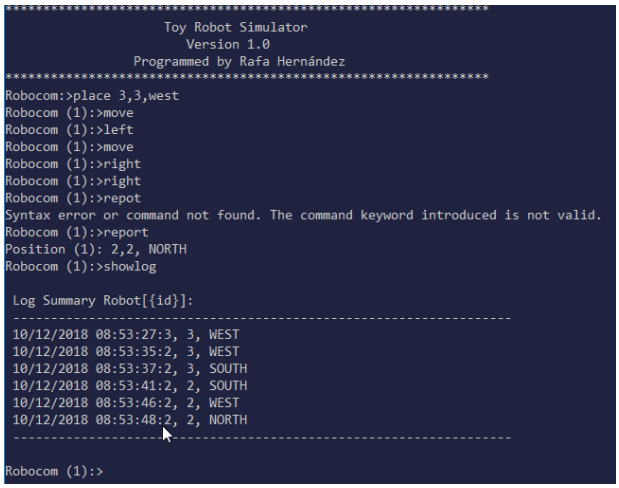
1. **Date and time** of the entry.
2. **Position** of the Robot.

Shell

Prompt shows the current active robot.

Samples

A sample of movement could be:



A sample handle multiple robots.