Universidad de los Andes

ISIS-1106 Lenguajes y máquinas
Project 1
Fecha: September 12, 2021

The project work for the course consists of managing and operating a language to program a robot in a two-dimensional world. The robot is able to move in the world (delimited by an $n \times n$ matrix); the robot moves from cell to cell. Cells are indexed by rows and columns. The top left cell is indexed as (1,1). North is top; West is left. The robot interacts (picks and puts down) with two different types of objects (chips and balloons). Additionally, note that the robot cannot move on, or interact with obstacles in the world (gray cells).

Robot Description

In this project, Project 1, we will use JavaCC to build an interpreter for the Robot Language introduced in Project 0.

Figure 1 shows the robot facing North in the top left cell. The robot carries chips and balloons which he can put and pickup. Chips fall to the bottom of the columns. If there are chips already in the column, chips stack on top of each other (there can only be one chip per cell). Balloons float in their cell, there can be more than one balloon in a single cell.

The attached Java project includes a simple JavaCC interpreter for the robot.[1] The interpreter reads a sequence of instructions and executes them. An instruction is a command followed by an end of line.

A command can be any one of the following:

- move(n): to move forward n steps
- turnright(): to turn right
- Put(chips,n): to drop n chips
- Put(balloons,n): to place n balloons
- Pick(chips,n): to pickup n chips
- Pick(balloons,n): to grab n balloons
- Pop(n): to pop n balloons

[1] The given interpreter is used for a different robot language, but can be used as a starting point for your own interpreter.

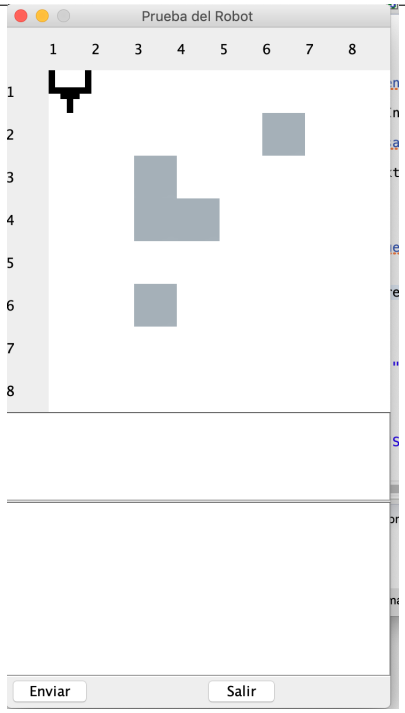

Figure 1: Initial state of the robot's world

The interpreter controls the robot through the class `uniandes.lym.robot.kernel.RootWorldDec`

Figure 2 shows the robot before executing the commands that appear in the text box area at the bottom of the interface.

Figure 3 shows the robot after executing the aforementioned sequence of commands. The text area in the middle of the figure displays the commands executed by the robot.

Below we define a language for commands and blocks. All commands are separated by new lines.

- A command can be any one of the following:
  - `MOVE n` – where `n` is a number or a previously defined variable. The robot should move n steps forward.
  - `RIGHT n` – where `n` is a number or a previously defined variable. The robot should turn n degrees clockwise.
  - `LEFT n` – where `n` is a number or a previously defined variable. The robot should turn n degrees counter-clockwise.
  - `ROTATE n` – where `n` is a number or a previously defined variable. The robot should turn n degrees in any direction (at random).

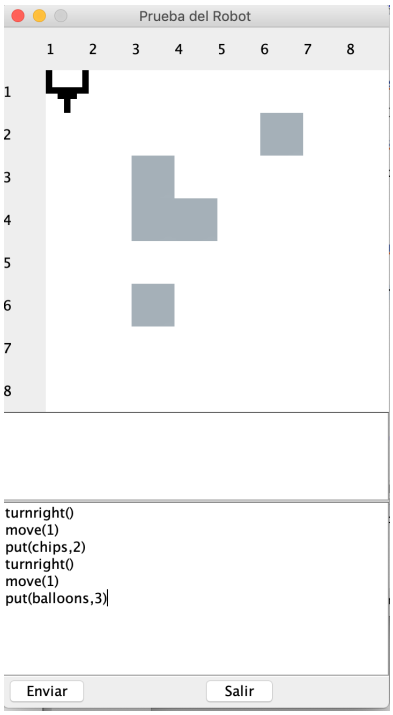

Figure 2: Robot before executing commands

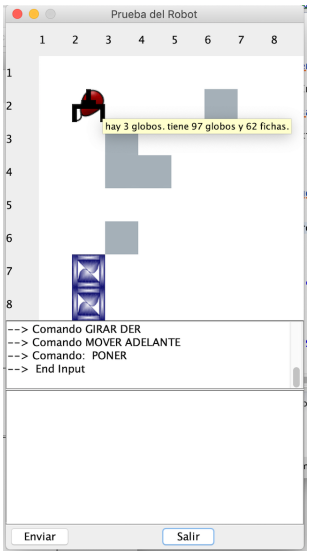

Figure 3: Robot executed commands

- `LOOK O` – where `O` can be `N`, `E`, `W`, or `S`. The robot should turn so that it ends up facing direction North if `O` is `N`, East if `O` is `E`, West if `O` is `W`, and South if `O` is `S`.
- `DROP n` – where `n` is a number or a previously defined variable. The Robot should drop `n` chips.
- `FREE n` – where `n` is a number or a previously defined variable. The Robot should let go of `n` balloons.
- `PICK n` – where `n` is a number or a previously defined variable. The Robot should pickup `n` chips.
- `POP n` – where `n` is a number or a previously defined variable. The Robot should pop `n` balloons.
- `CHECK O n` – where `O` is either `C` for chips, or `B` for balloons, and `n` is a previously defined variable or a number. The robot check if there are `n` chips or balloons (depending on the value of `O`) in the robot's position.
- `BLOCKEDP` – is a boolean predicate to check if the robot is blocked (it can or cannot move forward)
- **NOP**. the robot does not do anything.
- A block of commands: `(BLOCK commands)` where commands is simply a sequence of one or more commands (separated by new lines).
- Iterative instructions: `REPEAT n [commands]`, where `n` is a variable or a number describing the number of times the commands inside the `[]` will repeat, and `commands` is a sequence of basic commands separated by new lines.
- A conditional command: `IF expr [ commands ]`, where `expr` is a boolean expression, and `commands` is a sequence of basic commands separated by new lines
- `DEFINE n val` – defines a new variable `n` assigning it value `val` (an integer). Note that variable names need to be lowercase.
- `TO f :param OUTPUT expression END`. Functions are defined between the `TO` and `END` keywords, giving them a name `f` and a list of space separated parameters each defined by the colons before its name (as in `:param`). The inner works of a function are given as an expression or block of commands in its `OUTPUT`.

Task 1. The task of this project is to modify the parser defined in the JavaCC file `uniandes.lym.robot.control.Robot.jj` (you must **only** send this file), so that it can interpret the new language for the robot. You may not modify any files in the other packages, nor `uniandes.lym.robot.control.iterprete.java`.

An example of a valid input for the robot is shown below. You may assume that the name space is maintained throughout the execution of the examples. This is to say: you can enter each instruction separately, and definitions will be stored throughout the execution of the whole program.

```
ROTATE 3

IF BLOCKEDP [MOVE 1
NOP]

(BLOCK
IF BLOCKEDP [MOVE 1
NOP]
LEFT 90
)

DEFINE one 1

TO foo :c :p
OUTPUT
   DROP :c
   FREE :p
    MOVE one
END
foo 1 3


TO goend
OUTPUT IF !BLOCKEDP [
   (BLOCK MOVE 1
          goend)
   NOP
  ]
END
```