The project work for the course consists of managing and operating a language to program a factory 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 0, we will try to understand the robot language. That is, given a program for the robot, we will like to see if this satisfies the language specification and robot behavior, explained in the following.

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 interpreter for the robot. The interpreter reads a sequence of instructions and executes them. An instruction is a command followed by ";".

A command can be any one of the following:

- M: to move forward

- R: to turn right

- C: to drop a chip

- B: to place a balloon

- c: to pickup a chip

- b: to grab a balloon

- P: to pop a balloon

The interpreter controls the robot through the class `uniandes.lym.robot.kernel.RootWorldDec`
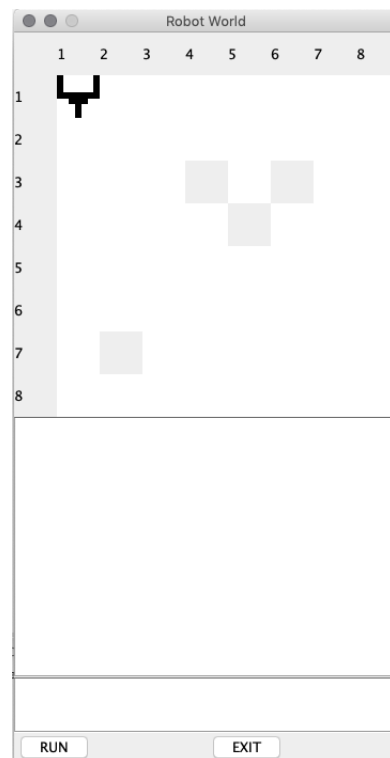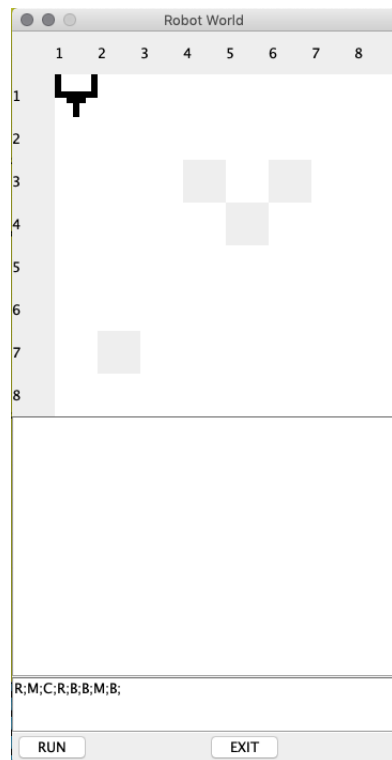
Figure 1: Initial state of the robot's world

Figure 2: Robot before executing commands

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.
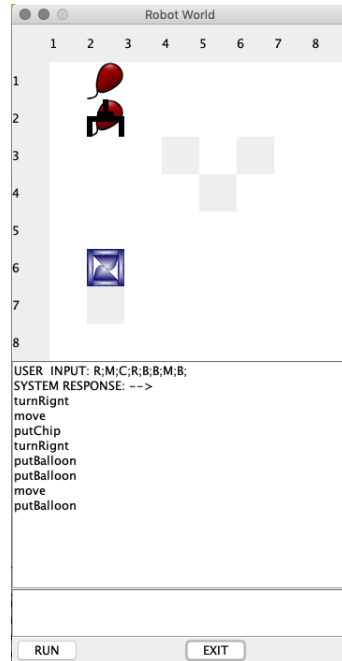

Figure 3: Robot executed commands

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).

  - **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 numbers describe 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 use Python or Java to implement a simple yes/no parser. The program should read a text file that contains commands and definitions for the robot, and verify whether the syntax is correct.

You must verify that used function names and variable names have been previously defined or in the case of functions, that they are the function's arguments. You must allow recursion.

Spaces, newlines, and tabulators are separators and should be ignored.

An example of a valid input for the robot would be the following:

```
1 ROTATE 3

3 IF BLOCKEDP [MOVE 1
4 NOP]

6 (BLOCK
7 IF BLOCKEDP [MOVE 1
8 NOP]
9 LEFT 90
10 )

12 DEFINE one 1

14 TO foo :c :p
15 OUTPUT
16    DROP :c
17    FREE :p
18     MOVE n
19 END
20 foo 1 3


23 TO goend
24 output IF !BLOCKEDP [
25    (BLOCK MOVE 1
26          goEnd)
27   NOP
28   ]
29 END
```