Throughout this course we will develop a project in several stages. The project 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.

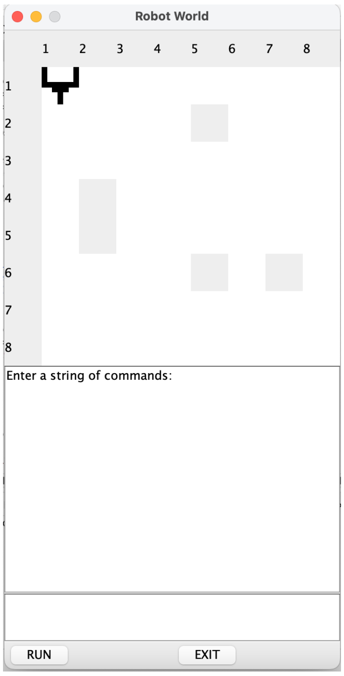

Figure 1: Initial state of the robot's world

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
- J(n): to jump forward n steps. It may jump over obstacles, but the final postion sould not have an obstacle.
- G(x,y): to go to position (x,y). Position (x,y) should not have an obstacle.

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 programs for the robot.

- A program defintion begins with the keyword `PROG` possibly followed by a declaration of variables, followed by zero or more procedure defintions, followed by a block of instructions. It ends the keyword `GORP`.
- A declaration of variables is the keyword `VAR` followed by a list of names separated by commas. A name is a string of alphanumeric characters that begins with a letter. The list is followed by `;`.
- A procedure defintion is a the word `PROC` followed by a name followed by a list of parameters within parenthesis separated by commas, followed by a block of instructions and ending with the word `CORP`.

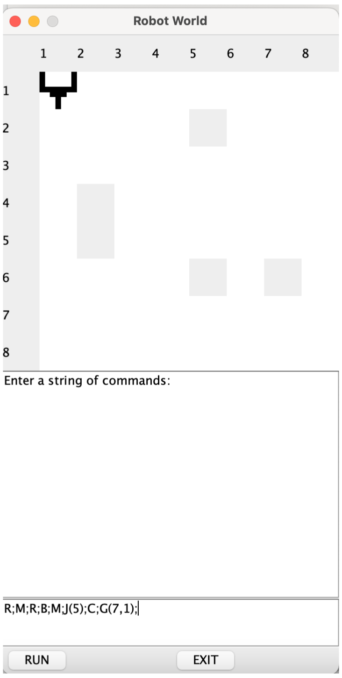

Figure 2: Robot before executing commands

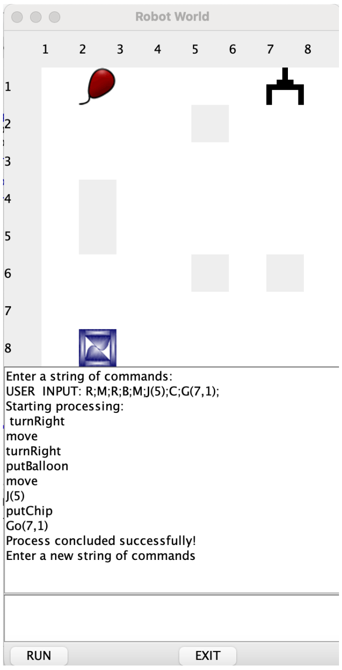

Figure 3: Robot executed commands

- A block of instructions is a sequence of instructions separated by semicolons within curly brackets.
- An instruction can be a command,a control structure or a procedure call.
  - A command can be any one of the following:
    * An assignment: `name := n` where `name` is a variable name and `n` is a number. The result of this instruction is to assign the value of the number to the variable.
    * **walk**`(n)` – where `n` is a number or a variable or a parameter. The robot should move n steps forward.
    * `jump(n)` – where `n` is a number or a variable or a parameter. The robot should jump n steps forward.
    * `jumpTo(n,m)` – where `n` and `m` are numbers, variables, or parameters. The robot should jump to position `(n,m)`.
    * `veer(D)` – where `D` can be `left`, `right`, or `around`. The robot should turn 90 degrees in the direction D.
    * **look**`(O)` – where `O` can be north, south, east or west. The robot should turn so that it ends up facing direction `O`.
    * **drop**`(n)` – where `n` is a number or a variable or a parameter. The Robot should put `n` chips from its position.
    * **grab**`(n)` – where `n` is a number or a variable or a parameter. The Robot should get `n` ballons from its position.
    * `get(n)` – where `n` is a number or a variable or a parameter. The Robot should get `n` chips from its position.
    * **free**`(n)` – where `n` is a number or a variable or a parameter. The Robot should put `n` balloons from its position.
    * `pop(n)` – where `n` is a number or a variable or a parameter. The Robot should pop `n` balloons from its position.
    * **walk**`(d,n)` – where `n` is a number or a variable or a parameter; `d` is a direction, either front, right, left, back. The robot should move `n` positions to the front, to the left, the right or back and end up facing the same direction as it started.
    * **walk**`(o,n)` – where `n` is a number or a variable or a parameter; `o` is north, south, west, or east. The robot should face `O` and then move `n` steps.
  - A control structure can be:

    Conditional: **if** `(condition)Block1` **else** `Block2 fi` – Executes `Block1` if `condition` is true and `Block2` if `condition` is false.

Conditional **if** `(condition)Block1 fi`– Executes `Block1` if `condition` is true does not do anything if it is false.

Loop: **while** `(condition)`**do** `Block od` – Executes `Block` while `condition` is true.

Repeat: `repeatTimes n Block per` – Executes `Block` `n` times, where `n` is a variable or a parameter or a number.

– These are the conditions:

  * `isfacing(O)` – where `O` is one of: north, south, east, or west
  * `isValid(ins,n)` – where *ins* can be **walk**, `jump`, **grab**, `pop`, **pick**, **free**, **drop** and `n` is a number or a variable. It is true if *ins*`(n)` can be executed without error.
  * `canWalk(d,n)` – where `D` is one of: north, south, east, or west and n is a number, a variable or a parameter.
  * `canWalk(o,n)` – where `D` is one of: right, left, front, or back and n is a number, a variable or a parameter.
  * **not** `(cond)` – where `cond` is a condition

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

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 a program 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 and tabulators are separators and should be ignored (outside of instructions).

Below we show an example of a valid program.