|  |  |  |
| --- | --- | --- |
| **CS 114 Intro Software Eng** | **Short Team Project** | **Spring 2015** |
|  | **KAREL THE ROBOT** |  |

Karel is an educational programming language that was created to introduce programming to beginners. Its creator Richard E. Pattis named the language after Karel Capek. Karel Capek was a Czech writer, and his science-fiction play R.U.R. (Rossum’s Universal Robots) introduced the word robot.

In this programming tool, Karel the robot exists in a small world that has streets, avenues, walls and beepers. He can understand five comments: *move*, *turnleft*, *putbeeper*, *pickbeeper*, and *turnoff*. However, programmers practicing the language can generate new functions using the existing ones. A program is written to control Karel in his world. Programmers can also create control-flows using pre-defined test statements.

In this project, you are asked to implement a similar application. This application should create Karel’s world using the programmer’s input text file, and control Karel’s movements reading user’s program file. The requirements and specifics of “Karel the Robot” application are below. However, you are free to add new features. You will work on this project as a team. For this project you are expected to submit your source code, its executable and README file. Each student should upload these files to lms.dwc.edu. Your README file should include a short description of the application, name of programmers, system requirements, instructions if necessary, and known bugs. You will be asked to present your work as a power point presentation.

The presentation can be about sharing your experience and work with the audience. You can talk about your design choices, risks that you took, advantages and disadvantages of your choices, and any unique feature you might have added to the application. Please include a short run of your application. The presentation is expected to be for 15 minutes and you can have five minutes to show your program.

**Karel’s World:**

The specifics of his world are defined in a simple text file. This file should be loaded when the application starts and be displayed on the Windows Form. Rules of use of this simple text file (you may choose to create this file with Notepad++) are:

1. The file defines the size of the world, the position of walls and beepers and Karel’s initial position.
2. Avenues and streets correspond to Cartesian coordinates. Avenues are vertical lines (x coordinates). Their number increase to the right. Streets are horizontal lines (y coordinates). Their number increases to the North. When locations and motion commands are used avenues are entered before streets.
3. Directions are
   1. North: 1
   2. East: 2
   3. South: 3
   4. West: 4
4. The commands are not case sensitive. These are the commands to build his world:
   1. World (might be written as WORLD, world –case insensitive- etc): defines the size of Karel’s world.

*World num\_avenues num\_streets*

*Ex: World 6 6*

*His world has 6 avenues and six streets*

* 1. Beepers: defines the location and the number of beepers.
     + Beepers avenue street num\_beepers
     + Beepers 3 2 1
     + This places one beeper at the intersection of 3rd avenue and third street.
  2. Wall: defines the location and direction of walls. Each wall is one block.
     + Wall avenue street direction
     + Wall 2 2 4
     + This places a block of wall at {2,2} in West direction.
  3. Robot: defines the starting point and direction for Karel.
     + Robot avenue street direction num\_beepers
     + Robot 4 2 1 0
     + Karel starts at {4,2} and faces North. He has zero beepers in his bag.

Example:

World 8 8

Beepers 3 3 1

Robot 4 2 1 0

Wall 2 3 1

**Karel’s Language:**

Karel responds to only few comments. However, these can be expanded. There are primitive and test comments. Test calls return true or false.

**Primitives:**

1. move: Karel moves one intersection forward.
2. turnleft: Karel turns left.
3. pickbeeper: Take a beeper from the intersection and keep it in the bag
4. putbeeper: Take a beeper from the bag and leave it at the current intersection.
5. turnoff: Karel turns off, and program ends.

**Tests:**

front\_is\_clear, front\_is\_blocked, left\_is\_clear, left\_is\_blocked, right\_is\_clear, right\_is\_blocked, next\_to\_a\_beeper, facing\_north, not\_facing\_north, facing\_south, not\_facing\_south, facing\_east, not\_facing\_east, facing\_west, not\_facing\_west, any\_beepers\_in\_beeper\_bag, no\_beepers\_in\_beeper\_bag

**Clarification on Walls:**

Make a grid or array of cells to hold Karel. The robot moves from cell to cell. Use a paintbrush in Visual Studio Graphics to draw lines between cells to represent walls. Use the object Pen to draw lines to represent the walls.

For example, declare:

Graphics ^ g;

Pen^ blackPen;

Then later create these objects as:

g=panel1->CreateGraphics();

blackPen=gcnew System::Drawing::Pen(Color::Black);

draw using the command

g->DrawRectangle(blackPen,gridRect); //gridRect is an object that is the cell of the array

gridRect made with the following command:

Rectangle gridRect = Rectangle(x,y, Cellsize-1,Cellsize-1); //Cellsize is the number pixels in a cell

Is there a g->drawLine() command in Visual Studio?

**Double Buffering:**

All solutions are to use the simple double buffering scheme documented on lms.dwc.edu. This is the technique that uses two graphics objects and the view command.