Slogo Team 2 Design Document

Team Members: Sean Wareham, Elder M. Yoshida, Ryan Fishel, Scott Valentine, Ellango Jothimurugesan

Git Repository: https://github.com/srwareham/SLogo.git

INTRODUCTION:

SLogo is an implementation of a much simplified version of Logo. "Simple Logo", which retains the features most commonly used by beginning users so that it can be used to provide an introduction to computer programming. Our program will be split into modules of Model and View so that we can split into subteams and work independently. The View team will be handled by Elder and Sean, and the Model team will be handled by Ellango, Ryan, and Scott.

SAMPLE WORKFLOW:

Example: The user enters ‘fd 50’

The view has an ActionListener that waits for the user to click the Submit button for commands that they have entered, and the actionPerformed method runs myController.sendString(“fd 50”). The Controller then hands off the action to the Parser, by calling myParser.generateInstruction(“fd 50”), which returns an object of type ForwardInstruction to the Controller. The Controller then tells the instruction to execute itself, with runInstruction(parsedInstruction, myModel), and tells the Model to update with the command by creating a new Vector v and moving the turtle with myTurtle.translate(v). The Canvas will then paint the result of this action.

Example 2: The user enters “fwd 50”, an invalid code

The process for forwarding the String “fwd 50” to the Controller class until it reaches the function myParser.generateInstruction(“fwd 50”). At this point the Parser will not be able to find the instruction “fwd” in its function map and then it will throw an IllegalInstructionException from the exceptions package. It then tells the Model to tell the View to display the error with myModel.displayMessage(exception.toString())

Main.java

1. new View()
2. new Model(view)
3. new Controller(Model)
4. view.setController(controller)

Model:

* Control Package
  + Controller.java
    - Tells Model and Parser to do appropriate actions. It serves as a layer between the View and the concrete actions it requests.
      * public Instruction sendString(String s)
        + Sends the string to the parser and gets an Instruction as a return type.
      * public void saveState(FileWriter fw)
        + Tells the Parser to save user-defined variables and functions
      * public void loadState(File f)
        + Reads in the file and updates instance variables of the Parser
      * public void clear()
        + Tells the Parser and Model to clear their state.
      * public void runInstruction(Instruction instr, Model myModel)
        + executes the specific instruction and correspondingly updates the model.
  + Parser.java
    - Contains all variables and user-defined functions
    - Converts strings from the console into Instruction.
      * public Instruction generateInstruction(String s)
        + The Parser takes in a String and returns an Instruction object as the output
      * public void saveWorkspace(FileWriter fw)
        + Saves the current workspace (variables) to a data file
      * public void loadWorkspace(File f)
        + The Parser takes in a data file and stores the variables that had been saved
* Simulation
  + Model.java
    - Contains:
      * myTurtle
      * Collection <Line> myLines
      * myView.
    - public void update(Dimension bounds)
      * Updates the turtle’s orientation and adds lines
    - public void paint( Graphics2D pen)
      * paints sprites contained in model (turtle and lines)
    - public void addLine(Line line)
      * adds a new line to the collections of lines in the model
    - public void clearLines()
      * clears all lines from model
    - public void displayMessage(String s)
      * Tells the View to display a string.
  + Turtle.java
    - public void update(Bounds bounds)
      * checks if turtle is in bounds
      * moves the turtle and adds lines if necessary
      * updates ValueText with coordinate position and heading
    - public void togglePen()
      * flips boolean isPenDown
      * this is used to determine if turtle is painting or not
    - Extends sprite with own update and paint methods
  + Line.java
    - extends sprite with own paint method
    - Keeps track of start and end locations
* Instructions
  + Instruction.java
    - An abstract class for all instructions in the SLogo library.
    - All instructions (e.g. forward, sum) inherit from Instruction
      * public abstract execute (Model model)
        + Invokes the instruction and performs whatever command it does by editing the state of the model.
* Exceptions (all inherit from Exception)
  + OutOfBoundsException
    - Overrides toString()
  + IllegalInstuction
    - Overrides toString()
  + IncorrectFileFormat
    - Overrides toString()

View:

* View.java
  + Highest level of the view. All visible elements are contained within this.
  + Contains: menu bar, buttons, text fields, and the Canvas
  + Methods:
    - public void displayText(String text)
      * This method will be used to display textual information to the user via the GUI. This data can be exemplified by the Turtle’s current position/ heading as well as Error Messages
    - public void setController(Controller controller)
      * sets the controller
      * this is needed for the sequence of objects created in Main.java
* Canvas.java
  + Is the working environment for turtle.
  + Is where objects in the simulation are painted.
  + Methods:
    - public void paintComponent(Graphics2D pen)
      * This method paints all Graphics objects such as the turtle and the lines
    - public step()
      * Tells the model to update the simulation.

GUI Aspects:

View:

The view is the wrapper comprised by everything visible to the user. Specifically, the view contains the file menu, the canvas, the history, the console, the on-screen buttons, and the on-screen sliders.

File Menu:

There is a swing file menu within the view that containing the options: “New” “Save” “Load” “Clear” and “Quit”

“New” will open a new view instance. “Save” will save the current workspace--the variables and user defined methods only-- by opening a file explorer and prompting for a name. “Load” will open a file explorer and allow a user to choose a file of user defined variables and functions set into the current workspace. “Clear” will clear all on-screen material as well as any user defined variables or functions. The canvas, the history, and the console are all cleared

Canvas:

The canvas will be the onscreen representation of the turtle and all lines drawn. The canvas will also contain a textual representation of the turtle in the form of (x-coordinate, y-coordinate, heading based on degrees counter clockwise from the positive x-axis). Coordinates are based from the origin,the center of the canvas, (0,0).

Console:

The console is a java swing text area composed of multiple lines where a user can enter text.

The user can enter multiple lines of commands that will not be processed until the “submit” button is pressed. Commands entered in this text field will be applied to the simulation’s current state. That is, if the same commands are submitted twice, they will be repeated as though they were new commands.

Buttons:

* Submit

The “Submit” will call Controller.sendString(String input) and pass the content of the Console in as the input

* Up

The “Up” button will call Controller.sendString(String input) and pass in the String (“fd ” + DISPLACEMNET\_MAGNITUDE) from the “Displacement Slider”. This simply sends the command as though it were entered through the console. Commands sent via Buttons are logged in the command history.

* Down

The “Down” button will call Controller.sendString(String input) and pass in the String (“fd ” + -DISPLACEMENT\_MAGNITUDE). This will function much like the “Up” button

* Rotate Left

The “Rotate Left” button will call Controller.sendString(String input) and pass in the String (“Left ” + ANGLE\_MAGNITUDE)

* Rotate Right

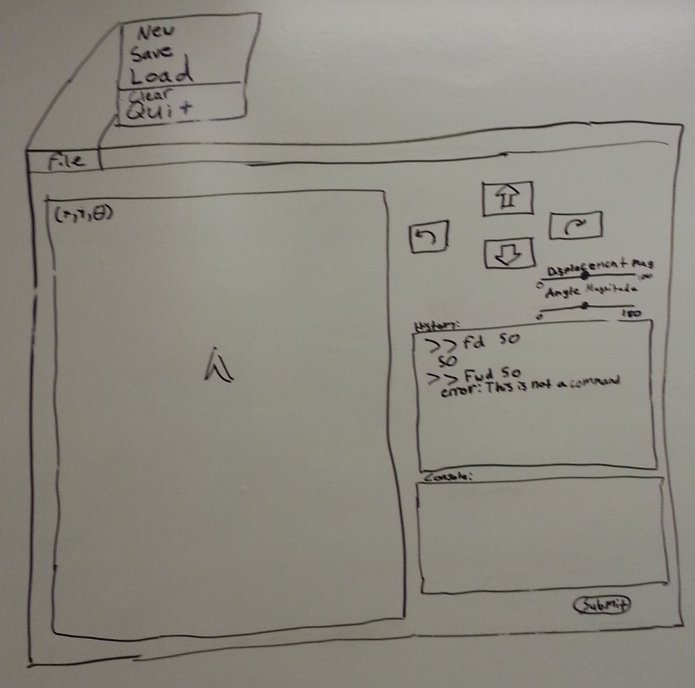
The “Rotate Left” button will call Controller.sendString(String input) and pass in the String (“Right ” + ANGLE\_MAGNITUDE)

Sliders:

The sliders are part of the View class. They are created using the swing package and will be used to change the angle and magnitude of the effects of the on-screen buttons.

Sample Photos:

**Sample UI**



**UML DIAGRAM (photos)**

