Introduction

The goal of this project was to develop a GUI and API based off of the Logo programming language. We wanted to make our API flexible and extensible so that adding new features would be relatively simple. For the front end, our GUI was designed so that setting turtle images, pen colors, and creating multiple workspaces/tabs was easily implemented. Adding exceptions and changing console output to reflect errors in user input was very flexible. Our ability to add new functionality through menubar items and mouseListeners was shown to be extremely effective, when implementing new features like adding new tabs/workspaces and closing current tabs. For the backend, adding commands, adding new features to the parsing algorithm, and changing the languages of the input commands was easily extensible.

The core architecture of our program was centered around an interpreter, which would receive the strings from the input text box in the View interface and then interpret these strings. After interpreting these strings, the commands would be created and then executed based on the order they were entered in by the user.

We designed our view to be extremely simple and clean. That way, a peer could easily pick up our program and experiment with it without getting bogged down by extraneous features and panels in our window. When implementing new features, instead of adding new buttons to the main frames, we added the functionality to a drop down menu in the menu bar with mouseListeners to keep our view relatively simple and elegant. When adding new features that required panels, we opted to do the work in a pop up panel that would update the necessary information in the back end and then update the main view accordingly.

Overview

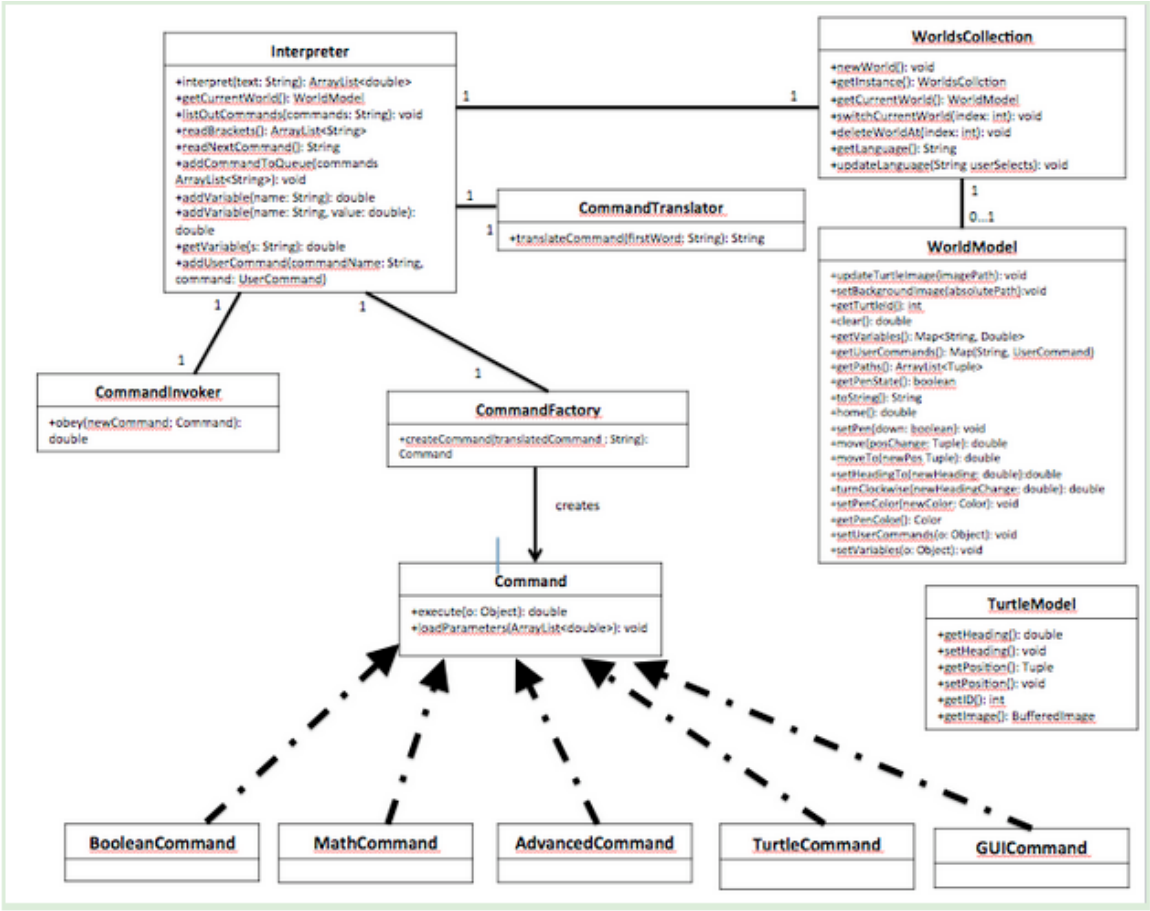
In the frontend of our project, we have a class, SlogoFrame, which encompasses a module of panels, along with a menubar. This is a singleton class, which encloses our WorldGraphicsPanel, StatsPanel, ConsolePanel and MenuBar. SlogoFrame has a primary method, populateDisplays, which calls addNewTab(). addNewTab() creates a new StatsPanel, WorldGraphicsPanel, and Console Panel. Each in this module of JPanels serves a different purpose within the user interface. Firstly, the WorldGraphics Panel utilizes a graphics 2D engine. The primary method in this class is paint, and paint draws the border on the panel, draws the trails of the turtle, and draws the turtle. Then StatsPanel displays the commands and variables created by the user in that specific world. The ConsolePanel has text areas for user input history, user input, and the console output. It encloses a public instance of Interpreter, and takes the user input text, evaluates it using the interpreter, and then outputs the result to the console output. Within the MenuBar, we have a variety of Menus: such as FileMenu, TurtleMenu, HelpMenu, and LanguageMenu, which allow the user to make selections and change preferences on the IDE.

We also have a module of classes called exceptions. This package contains a variety of exceptions which inherit from the class SlogoException. SlogoException extends exception, and has a simple constructor which takes in an alertString and prints it to the console. Each of the classes which inherit from SlogoException utilize this constructor in order to print their corresponding alert message.

The Interpreter Class is vital to our project. In the evaluateCommand method, it parses a string input, in order to get the name of the command and the parameters. The interpreter class has a variety of methods which allow it to first determine if the string input by the user is legitimate, and then throw an exception if not. For example, these methods include isVariable() and isCommand(). The interpreter class also contains readBrackets, which allows evaluateCommands to evaluate advancedCommands. Once the evaluateCommand method has parsed the string input, it uses the strings to translate the strings using the Command Translator, and then instantiate newCommands using the CommandFactory. The CommandInvoker has an obey method which calls the execute method of a given command.

The Commands module also plays a vital role. Within the commands package, there are multiple packages such as math commands and boolean commands. We have a superclass Command, and then MathCommand, BooleanCommand, AdvancedCommand, and TurtleCommand all inherit from this superclass. For each of the Slogo commands, a number of commands are available for use. For example TanMathCommand and SinMathCommand extend MathCommand. The classes have a private instance variable called parameters, which is an arraylist of doubles called. The commands also are run using an execute method which returns the desired output given the parameters.

Finally, we have the model module. This is composed of the WorldsCollection class, the WorldModel Class, and the TurtleModel class. The WorldsCollection class contains an arraylist of world models. It is a singleton class, which instantiates the class when getInstance() is called, unless it has already been instantiated. The WorldsCollection class allows you to delete worlds, switch the world, or return the current world. It also allows you to update the language of the entire IDE, across all of the world models. The WorldModel stores the turtles of that particular world in the form of instances of the TurtleModel class, along with their trails. It also holds the variables and commands created by the user, and the pen state (color, and showing or not showing).



User Design Interface

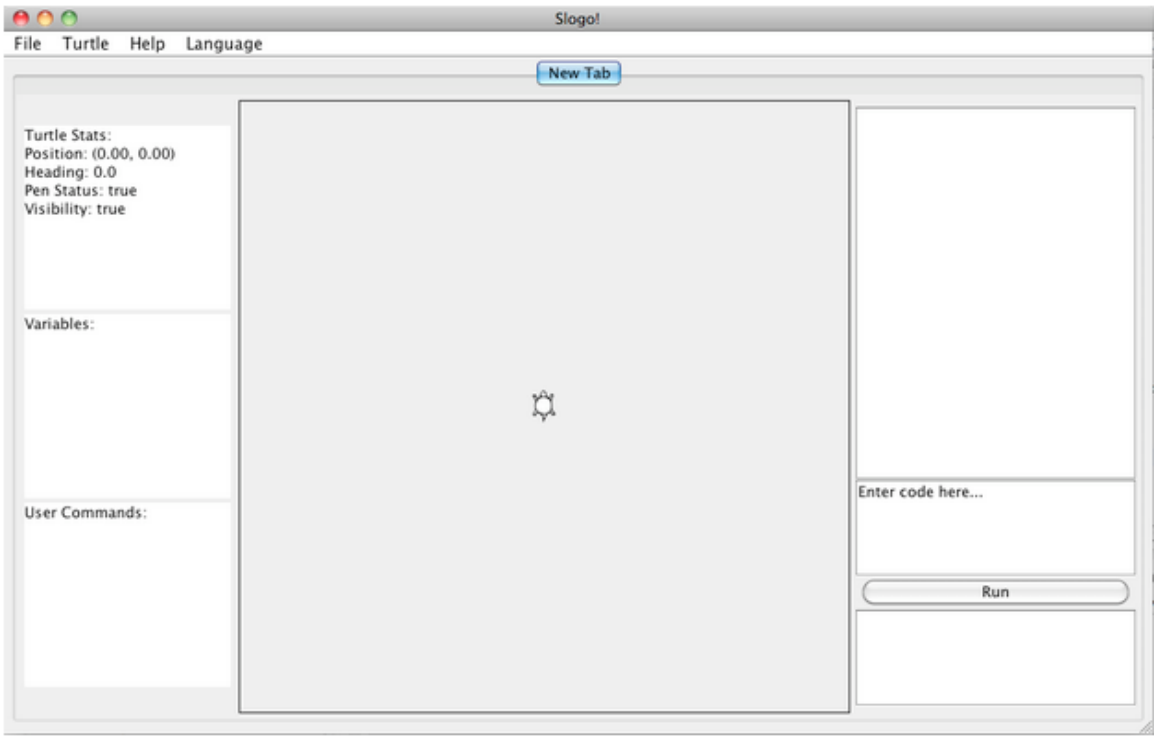
The GUI has several important components:

· A console panel which contains a text box for SLogo command input, a run button to execute commands, a console box to display exceptions, and a command history box

· A world graphics panel that displays the turtles and paths

· A stats panel that displays the current turtle state

Aside from these basic elements, there are tabs for multiple independent environments and drop-down menu selectors for various other operations. The menu selectors allow for changing of the SLogo command language, display of command help, options for setting the turtle image and pen color, setting of the background, adding and closing tabs, and saving and loading workspaces. These components are tiled vertically. This allows for users to have ample space to view the history, turtle state, and enter multiple lines of SLogo commands. The overall window's width is greater than its height. A screenshot is below.



The console is located on the bottom right, and can display exception descriptions and evaluation return values longer than one line. The SLogo command entry box is above the run button. This box highlights the text in the box after the run button is clicked so the user can easily delete the text and type new commands into the box. All commands that are run are sent to the history box located on the upper right, regardless of whether or not the command is valid. The variables box is of ample size to list many variables and can scroll. The same applies with the user commands box. However, the user commands box cannot scroll horizontally, which results in truncation of longer user commands. In retrospect, this should have been fixed. Another bug in the GUI occurs when the command history box becomes long enough to scroll. This results in a strange window packing situation in which some GUI elements disappear until the window is resized to be slightly larger. Aside from this, the GUI is very intuitive to use and provides a clean interface for the user to control the turtle environment.

Design Details

For the front-end:
SlogoFrame serves as the container for the different panel components as well as the menu bar. In addition, it keeps track of data and handles creation/switching/deletion of tabs to make sure that the user is on the correct workspace. This was mainly created at first simply to contain the different gui panels, though eventually this naturally became a tabbedPane container as well. For extension, the code is extremely flexible to the addition of different panels.

WorldGraphicsPanel, the central panel, is one of the two displays for the results of user commands. This obtains the current world from the WorldsCollection and paints the trails and turtles.

StatsPanel, the left panel, reflects the numeric data for turtle stats as well as world variables. These are all obtained from the current world as well. This was the simplest way to incorporate instant numeric feedback for the user without having to manage multiple windows.

ConsolePanel, the right panel, controls the majority of the user input/output. The top text box displays the history of user-input commands for that world. The middle textbox is the input box for user commands. When done entering the command, the button below it allows the user to run the command, thus fulfilling the most basic requirement of running commands through the window. The last textbox has the console output.

Within the **MenuBar**, we have a variety of Menus: such as FileMenu, TurtleMenu, HelpMenu, and LanguageMenu, which allow the user to make selections and change preferences on the IDE. This is simply a way for us to manage many of the user commands/preferences and the menus to us seem intuitive.

For the backend:
Our **Exceptions** package contains subclasses of SlogoException extends exception which has a simple constructor which takes in an alertString and returns it indirectly to the consolePanel when thrown. Each of the classes which inherit from SlogoException utilize this constructor in order to print their corresponding alert message.

The **Interpreter** parses and interprets user input. Because of our wish to parse huge user commands, we essentially made our review of user input as a stream rather than a DOM model. By evaluating word by word (with words defined as separated by space), we were able to free a lot of RAM for other purposes. With every word, we attempt to parse its value either by returning its literal/stored value or by parsing the command - and its parameters - through the creation of Commands. The CommandInvoker has an obey method which calls the execute method of a given command.

The **Commands** classes were made so that each command could be initiated as needed through reflection and then simply used to load different parameters every time it is run. The basic structure has execute that runs and then returns values based on the specs outlined by the commands sheet provided by instructor.

The Model classes act as the Model/Controller of the MVC design. The logic and backend are contained in the actual WorldsCollection class, the WorldModel Class, and the TurtleModel class. The WorldsCollection class contains an arraylist of world models. It is a singleton class, which instantiates the class when getInstance() is called, unless it has already been instantiated. The **WorldsCollection** class allows you to delete worlds, switch the world, or return the current world. It also allows you to update the language of the entire IDE, across all of the world models. The WorldModel stores the turtles of that particular world in the form of instances of the TurtleModel class, along with their trails. It also holds the variables and commands created by the user, and the pen state (color, and showing or not showing).