Genesis Development Environment Design Document

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and design of Genesis Development Environment.

1.2 Scope

The main project goal is to design and implement a system that provide basically 2 functions ; visual programming through dragging and dropping graphical object to generate code constructs corresponding to these objects.

Genesis Development Environment is an integrated development environment intended for genesis programming language users. GDE provides functionalities like a file browser to be able to navigate your project files, provides an editor to be able to type down your code, save it and run it. And mainly the visual programming module which generates the code in the editor by moving graphical objects on a canvas and been able to run a simple animation that depicts the algorithm written in the editor.

1.3 Overview

So going through this document I will talk about different modules in this software from the perspective of a user and from a technical point of view going into every technical detail.

1.4 References

Dea, Carl. Java FX 2.0 Introduction By Example. Apress, n.d.

Nouard, Arnaud. In-Side FX. 2013.
<http://arnaudnouard.wordpress.com/category/javafx/undecorator/>.

Oracle Java FX 2.0. n.d. <http://www.oracle.com/technetwork/java/javafx/overview/index.html>.

(Dea) This is one of the well-formed books for Java FX newbies as it provides various simple examples.

(Oracle Java FX 2.0) This is the main Java FX website as it provides download links, documentation, examples, samples and articles.

(Nouard) I used this package as a back stage for my application; it provides the ability to maximize, minimize, full screen and controls the applications look and feel through calling some simple packages from the code.

1.5 Definitions and Acronyms

GDE: Genesis Development Environment

2. SYSTEM OVERVIEW

The system is built on the basis of the MVC architecture. Genesis runtime files represent the background model and then there lies the View and the controller classes.

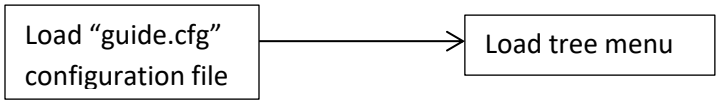
Basically the user drags and drops graphical objects on a canvas to be able to generate genesis code in the code editor area in which eventually he will be able to play a simple animation that demonstrates that algorithm being run.

Technically the menu on the left is loaded from a text file that simply describes the genesis languages. In the tools sub tree is a connector objects that when you drag it connects the two genesis constructs that you've created before. The third sub tree is a file browser that allows you locate your genesis files.

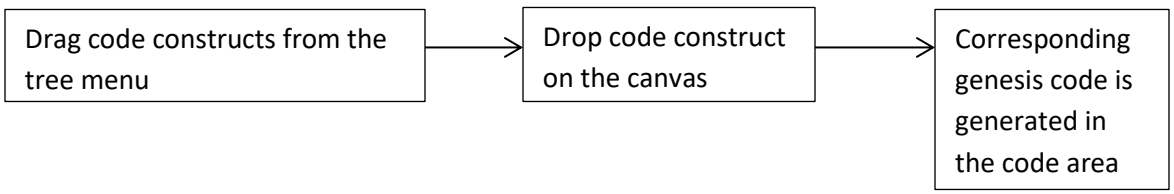
3. SYSTEM ARCHITECTURE

3.1 Architectural Design

From a high level perspective;

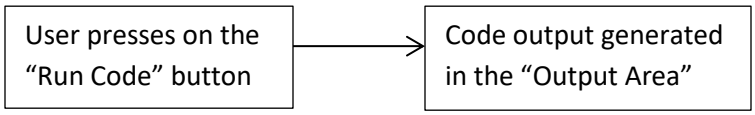


This idea of loading the tree menu from a configuration file came to me when I studied and observed how the "GUIDE" code works. This way of loading the tree menu made perfect sense in case new constructs were being added to the original Genesis code.

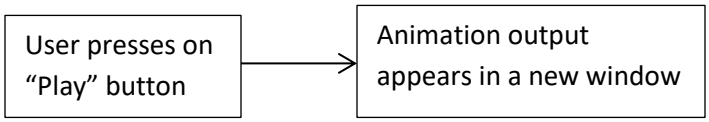


When dragging a code construct from the tree menu a corresponding image is dropped on the canvas.

Once this image is dropped and appears on the screen a window pops up asking the user for some values to be able to generate the right code on the screen.



When the user presses on this button the "GenesisInterpreter" class is called and eventually the "Evaluator" class and the output is shown in the "OutputArea".



When play button is pressed the "EvaluatorXML" class is called which holds the code responsible for animations.

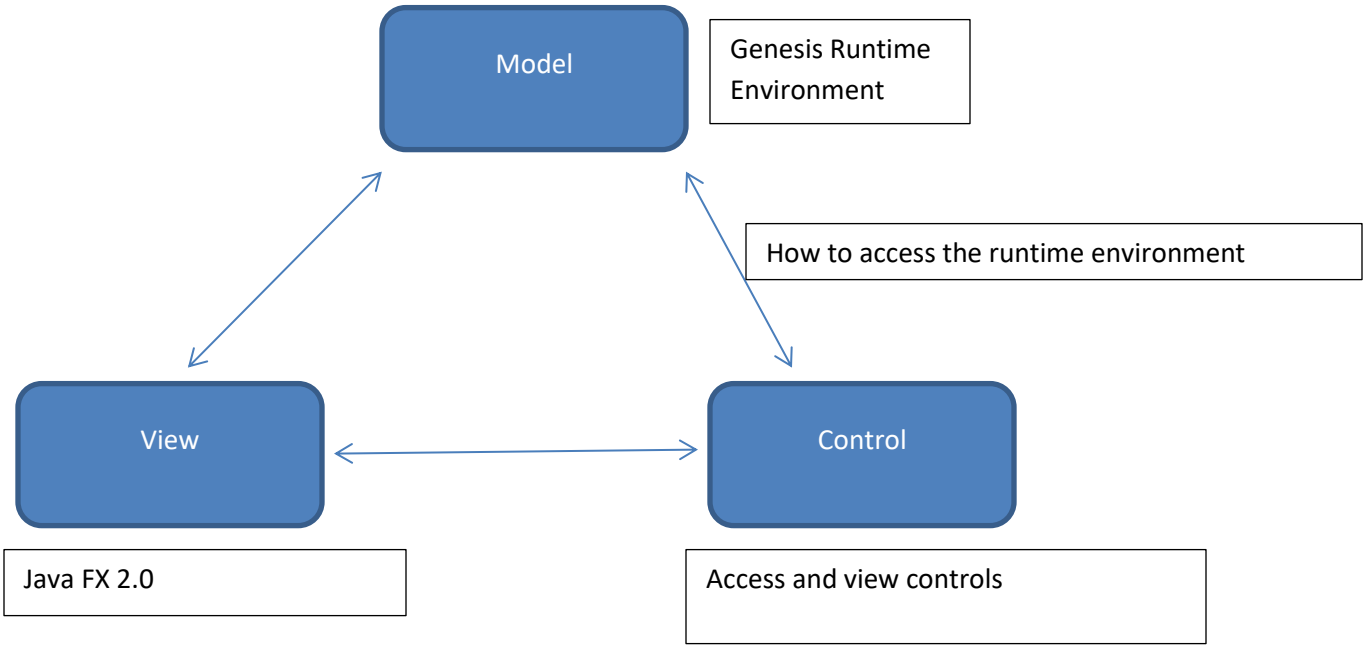
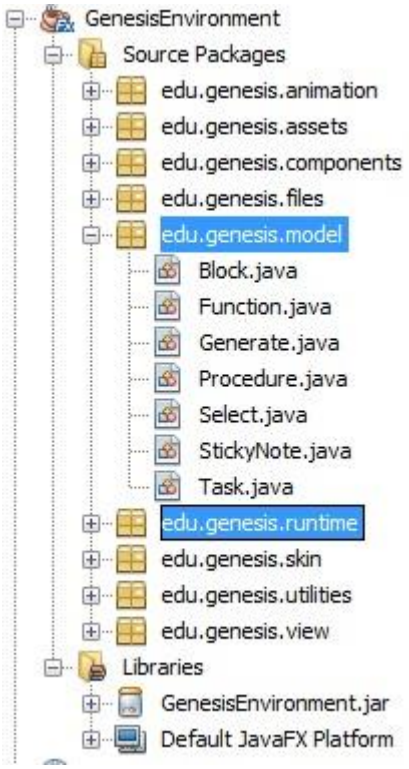
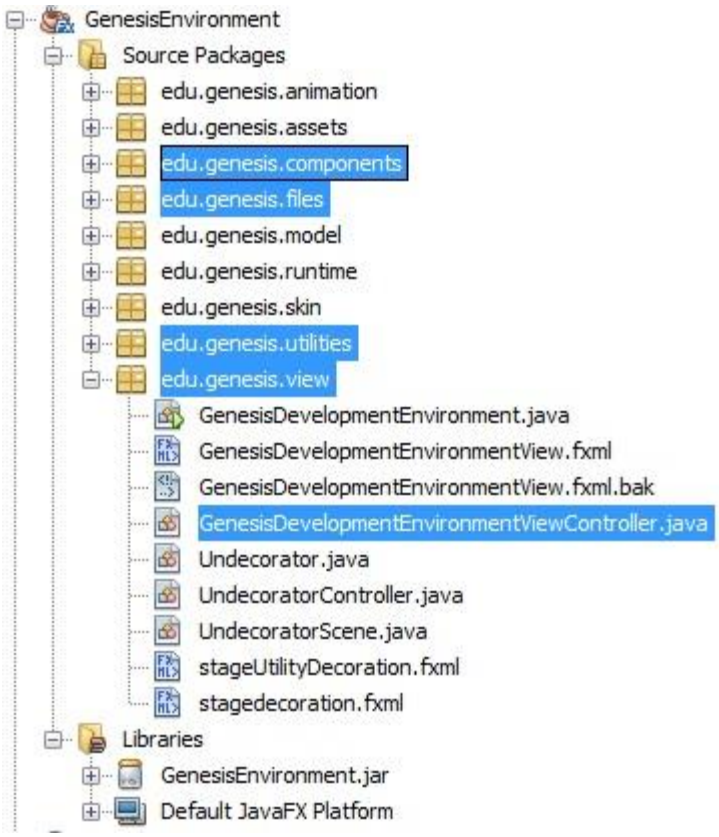


Fig 1: Model view controller architecture as it is demonstrated in this project.

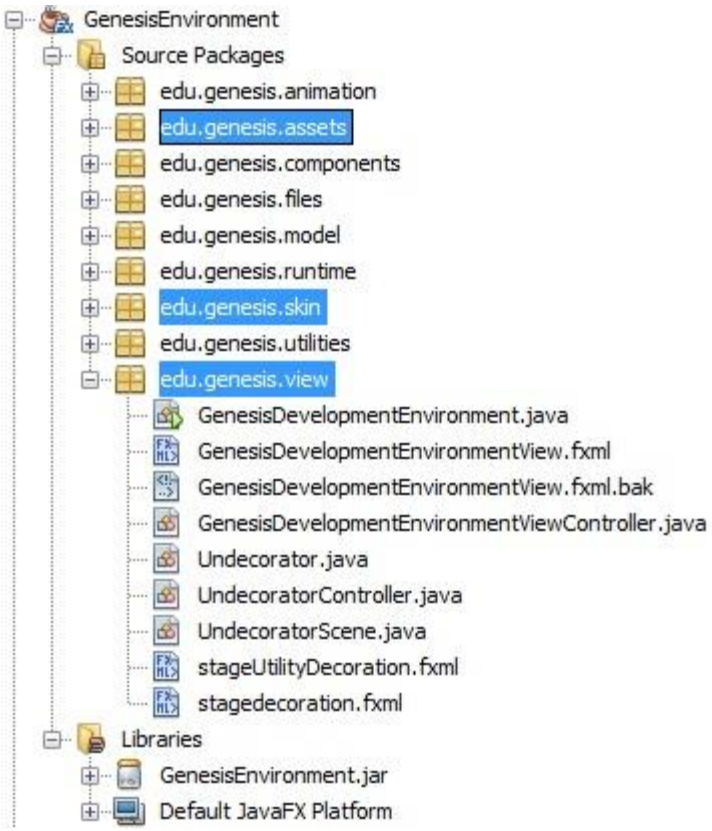
Model :


GenesisEnvironment
Source Packages
edu.genesis.animation
edu.genesis.assets
edu.genesis.components
edu.genesis.files
edu.genesis.model
Block.java
Function.java
Generate.java
Procedure.java
Select.java
StickyNote.java
Task.java
edu.genesis.runtime
edu.genesis.skin
edu.genesis.utilities
edu.genesis.view
Libraries
GenesisEnvironment.jar
Default JavaFX Platform

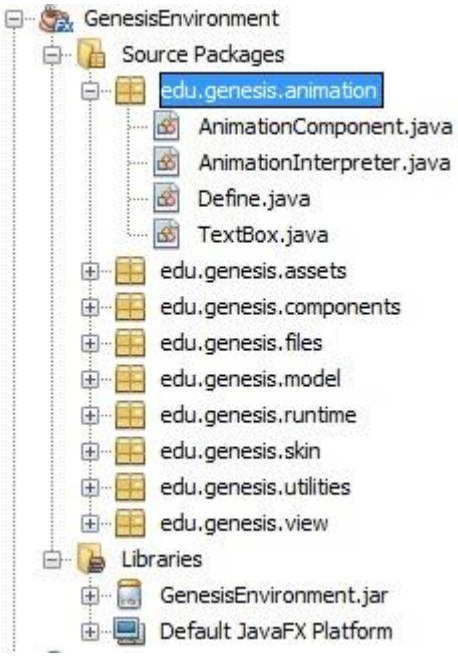

GenesisEnvironment
Source Packages
edu.genesis.animation
edu.genesis.assets
edu.genesis.components
edu.genesis.files
edu.genesis.model
edu.genesis.runtime
edu.genesis.skin
edu.genesis.utilities
edu.genesis.view
GenesisDevelopmentEnvironment.java
GenesisDevelopmentEnvironmentView.fxml
GenesisDevelopmentEnvironmentView.fxml.bak
GenesisDevelopmentEnvironmentViewController.java
Undecorator.java
UndecoratorController.java
UndecoratorScene.java
stageUtilityDecoration.fxml
stagedecoration.fxml
Libraries
GenesisEnvironment.jar
Default JavaFX Platform

Controller:

View:



Animation:



3.2 Decomposition Description

Genesis Interpreter is the entry point to Genesis Runtime Environment where a genesis file is passed to the “callGenesis()” method as a result the “EvaluatorXML” class is called to produce an xml file that describes an animation procedure of graphical objects representing the algorithm that was written.

This XML file then is parsed and interpreted by the “AnimationInterpreter” class to graphical objects and animation paths depending the values in the XML file “edu/genesis/files/output.xml”

I created a new Evaluator Class in the Genesis Runtime Environment to run in a separate mode called the XML mode when you pass the argument as “-x”.

```
public void playButton(ActionEvent event) {
    List<String> argsList = new ArrayList<>();
    argsList.add("-x");
    File file = ButtonEventsHandler.saveButton(event);
    argsList.add(file.getAbsolutePath());
    String[] args = new String[argsList.size()];
    for(int i = 0 ; i < args.length ; i++){
        args[i] = argsList.get(i);
    }
    GenesisInterpreter.callGenesisXML(args);
    File outputFile = new File("src/edu/genesis/files/output.xml");
    AnimationInterpreter ai = new AnimationInterpreter(outputFile);
}
```

GenesisInterpreter.java

```
                    } catch (Exception ie) {
                        // eof
                        moreInput = false;
                    }
                } while (tryAgain);
            }
            if (empty) {
                EvaluatorXML.stopAt.add(0);  // default to a 0
            }
        }
        try {
            interpreter.ax.evalProgram(tn);
        } catch (Exception e) {
            e.printStackTrace();
            EvaluatorXML.printError("Your algorithm is incorrect in some unknown way.\n"
                    + "> If you have the time, please mail a copy of your algorithm to:\n"
                    + ">\n>      morell@cs.atu.edu\n\n"
                    + "> Include in your mail the version of Genesis you are running.\n"
                    + "> To obtain this type:\n"
                    + ">    run -v   // on a Unix-like system\n"
                    + "> or\n"
                    + ">    run +v   // on a Windows system\n"
                    + "\n>\n> Thanks!");
            outputArea.appendText("Your algorithm is incorrect in some unknown way.\n"
```

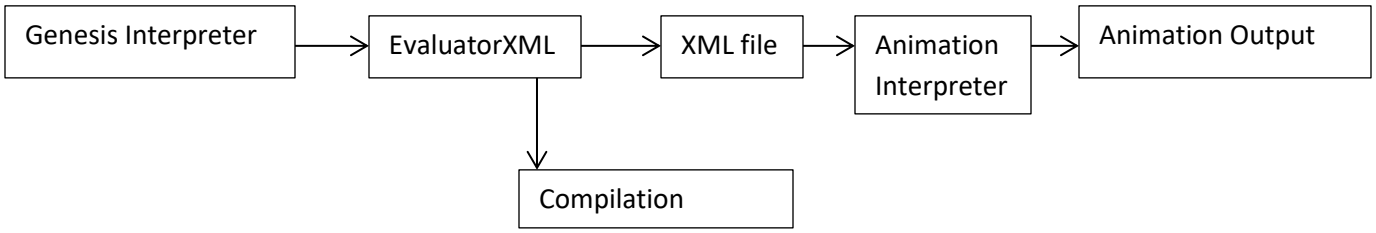
EvaluatorXML.java

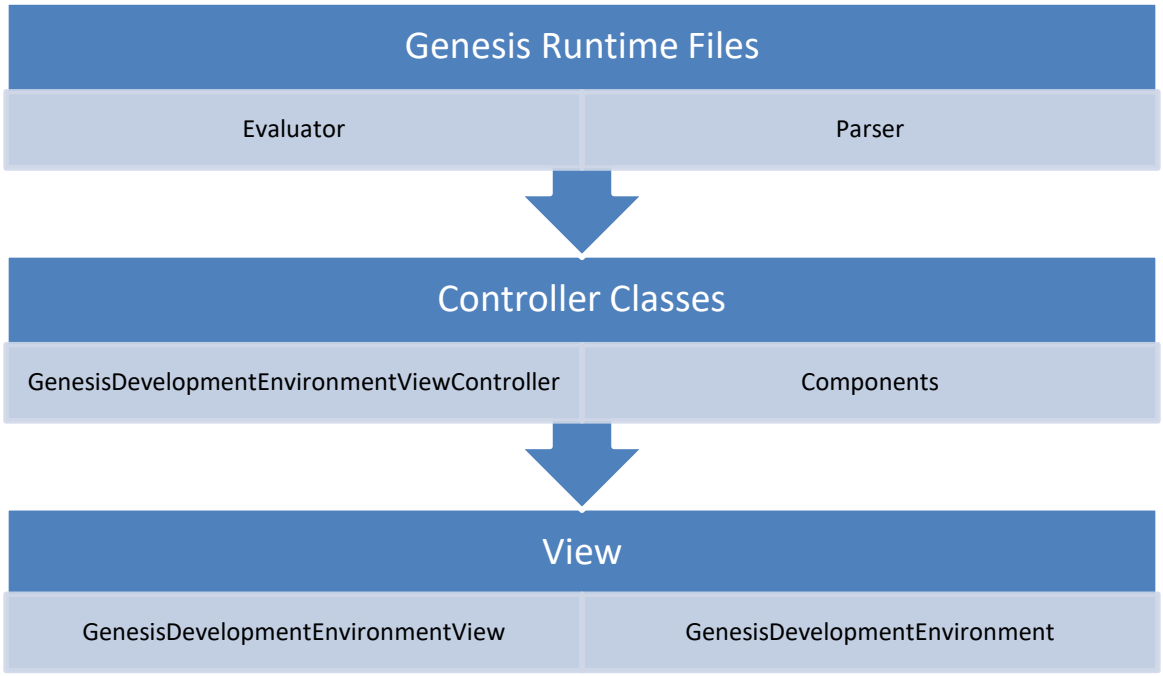
```
        printError(msg, tn);
    }
    // ---------------- begin eval routines ------------------ //

    public void evalProgram(Node tn) {
        root = (TreeNode) tn;
        if (xml) {
             try {
                docFactory = DocumentBuilderFactory.newInstance();
                docBuilder = docFactory.newDocumentBuilder();
                doc = docBuilder.newDocument();
                rootElement = doc.createElement("world");
                doc.appendChild(rootElement);
          } catch (ParserConfigurationException pce) {
          }
        }
        evalStmtList(tn);
        if (xml) {
            try {
                TransformerFactory transformerFactory = TransformerFactory.newInstance();
                Transformer transformer = transformerFactory.newTransformer();
                DOMSource source = new DOMSource(doc);
```

Output.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<world>
    <define>
        <name>a</name>
        <type>box</type>
        <outline>solid</outline>
        <location x="20" y="20"/>
        <value>a</value>
    </define>
    <display>a</display>
    <define>
        <name>n</name>
        <type>arrow</type>
        <outline>noborder</outline>
        <location x="20" y="60"/>
        <value>n</value>
    </define>
    <display>n</display>
    <define>
        <name>1</name>
        <type>box</type>
        <outline>solid</outline>
```




Genesis Runtime Files
Evaluator
Parser
Controller Classes
GenesisDevelopmentEnvironmentViewController
Components
View
GenesisDevelopmentEnvironmentView
GenesisDevelopmentEnvironment

3.3 Design Rationale

Designing this kind of an application is not quite an easy task as the backend is the genesis runtime environment but in the end MVC made the most sense because based on the user interaction data were being viewed.

4. DATA DESIGN

4.1 Data Description

As mentioned the data backend of the system is the genesis runtime environment. The output of running the code is stored in the output code area in an xml file that describes the animation to be generated. Xml files are easy to parse and then store the xml values in a hash map and according to each value a graphical object appears on the screen.

```
private static void populateGraphicalObjects(final Group group) {
    Set set = xmlMap.entrySet();
    Iterator i = set.iterator();
    TextBox box = null;
    double count = 0;
    while (i.hasNext()) {
        Map.Entry me = (Map.Entry) i.next();
        if(me.getKey().toString().contains("define")){
            box = new TextBox(me.getValue().toString(), "10", "10", count , count);
            group.getChildren().add(box);
            count = count + 20;
        }
    }
```

So this method creates a “TextBox” graphical object if the tag “define” is there in the file.

5. COMPONENT DESIGN

Main program logic:

void main(){

load guide.cfg file;

load the tree from the file;

set drag and drop properties;

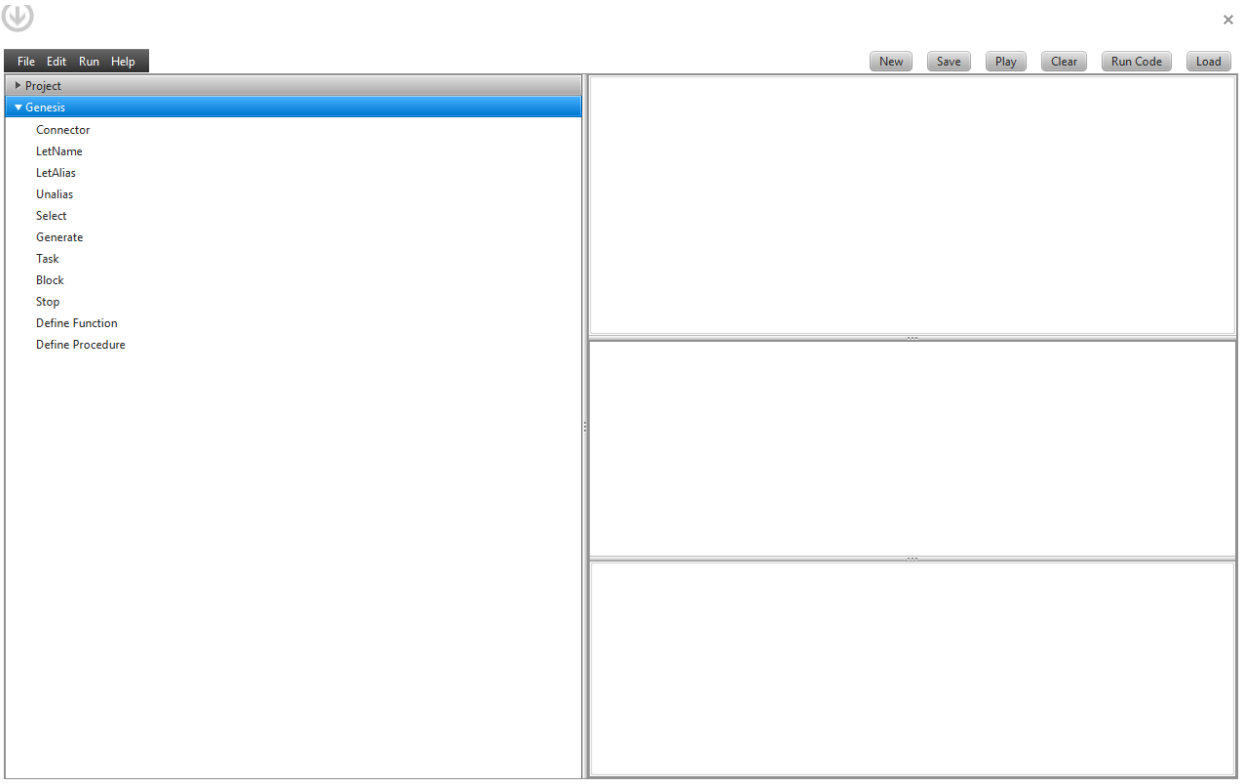
start drag and drop;

on drop generate code;

```
        if(button ==  run){
                run genesis in debug mode;
        }
        If(button == play){
                Run genesis xml mode;
                Animate();
        }
}
Genesis(){
Debug(){
Run genesis → generate output in outputArea
}
XML(){
Run genesis → generate xml output
}
}
Animate(){
intpretXml();
generateAnimation();
}
```

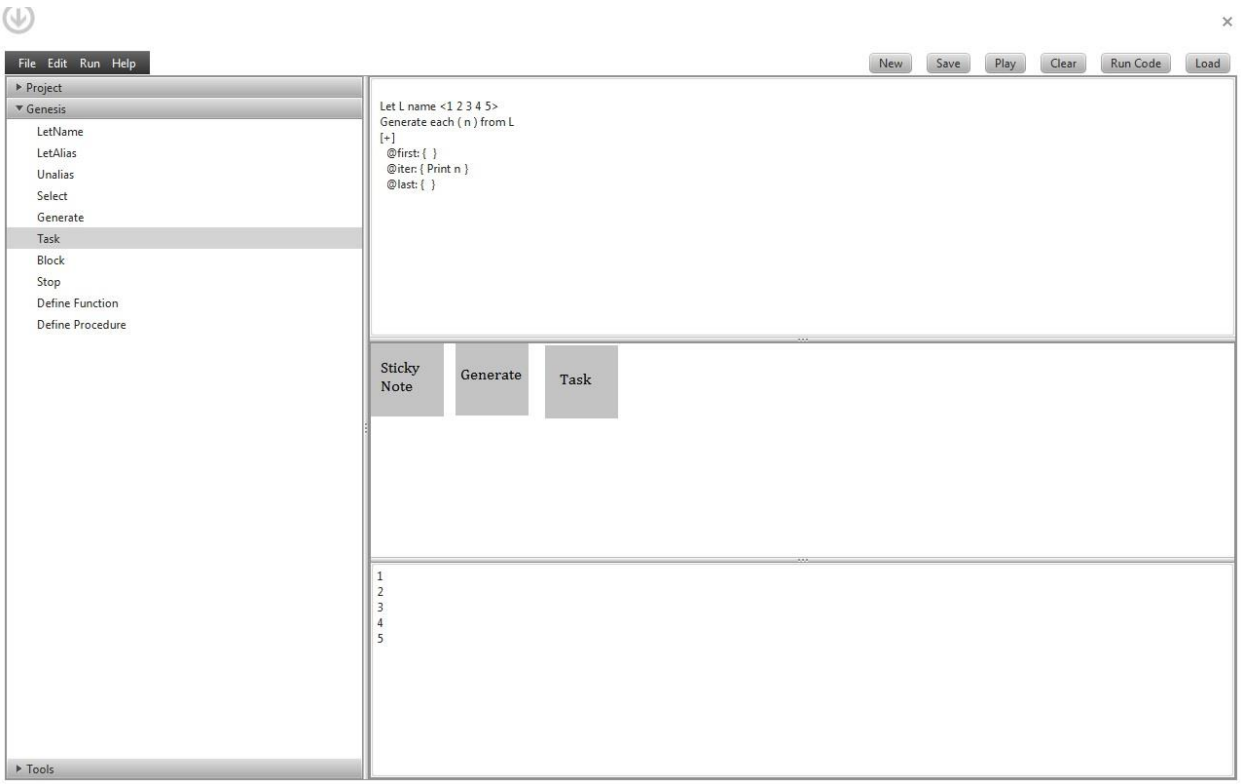
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

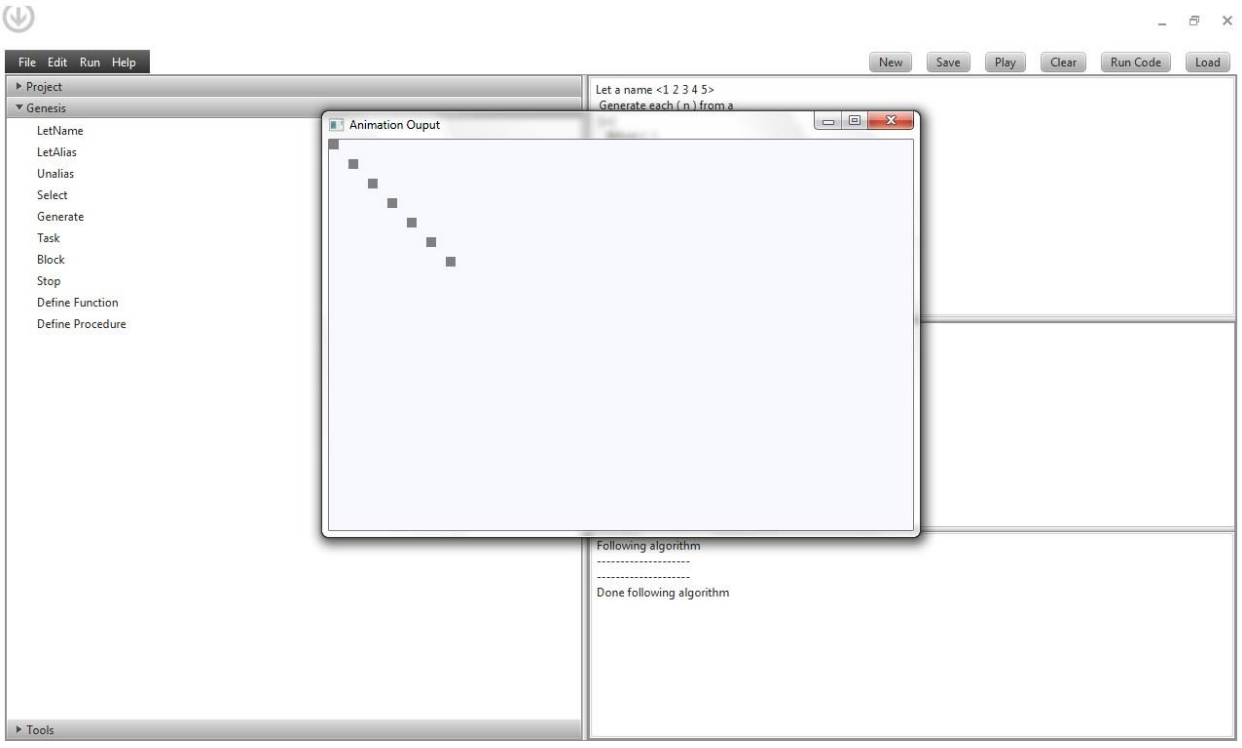


The genesis menu offers genesis language constructs that you can use to create an algorithm.

You can drag and drop on the canvas area.



Expected output when you run your code without animation



Expected output when running in animation mode

7. REQUIREMENTS

Java 7 is required for this application to run

Netbeans IDE version 7.2

Java FX 2.2

Genesis Runtime Files

8. APPENDICES

9. Testing

9.1 Example Algorithms

Algorithm 1 :

Let list name <1 2 3 4 5 6 7>

Let prime name <>

Let fact name <>

Generate each (n) from list

[+] when ((n%2) != 0)

    append n onto prime

let fact (n) name function{

  Let ans name 1

  select n = 0 -> Let ans name 1

        n > 0 -> Let ans name (n * fact(n -1))

  Return ans

}

Generate each (n) from prime

```
[+]
        @first: { }
        @iter: { append fact (n) onto fact }
        @last: {}

Let Swap(alias m,alias n) name procedure {
  Let t name m; Let m name n; Let n name t
}

Let Sort (L) name procedure {
  Generate each(m) from L while on(next(m))
  [+]
  @i:{
    Generate each(n) from next(m)
    [+] when ( n < smallest )
                @first:
                        Move (smallest) to (m);
                @iter:
                        Move (smallest) to (n)
                @last:
                        Swap(m,smallest)
  }
}

Sort (fact)
```

```
Print "-------------------------"

Print "Original List"

Print list

Print "Prime List"

Print prime

Print "Factorial List"

Print fact
```

Algorithm 2:

```
Let L name <1 2 3 4 5>

Generate each (n) from L

[+]

        @first: Let s name 0

        @iter: Let s name s + n

        @last: Print "Sum = " s

[+]

        Print "Double " 2*n
```

Algorithm 3:

```
Let a name <1 2 3 4 5>

 Generate each ( n ) from a

 [+]

   @first: {  }

   @iter: { Print n }
```

@last: { }

Algorithm 4:

Let L name <1 2 3 4 5>

Generate each (n) from L

[+]

@first: Let s name 0

@iter: Let s name s + n

@last: Print "Sum = " s

9.2 Scenarios

Generate Code Process

Menu --> Drag & Drop --> Dialog Box pops up --> XML Generation --> XML to Code --> Code to Textarea

Generate Animation Process

Code compilation in trace mode --> Print runtime scope to a XML File --> Scope XML to Code XML --> Animation

Scenarios

1- User starts up the system and starts dragging a sticky note to the canvas and sets its name and value
2- The code is generated as “Let a name 5” in the code area
3- The user decides to save the code to a file.

10. Future Developments

How to add a new Genesis construct to the tool.

First you have to describe this component in the guide.cfg file according to the pattern being followed in that file.

```
label =  LetName
format =  Let _StickyNote_ name _expression_ \n
toolTip = Label a value
action = 1
image = file:///C:/Users/Laptop/Documents/NetBeansProjects/GenesisEnvironment/src/edu/genesis/assets/stickynote.jpg
class = edu.genesis.components.LetNameDialog
```

Where label is the name that is going to appear in the tree view showing genesis constructs.

and format is how this construct looks like. Tooltip is just to describe the construct.

Action is a unique Id number across the file. Image is the exact location of your image that will be dragged on to the canvas. And finally class is the component that you should create using the java fx framework as shown before that would popup when that user drags and drops the image on the canvas

Then you need to create a model class for that component.

Since the GenesisDevelopmentEnvironmentViewController.java is the main controller so every component that you create should extend this class.

Threads :

When running the animation. 2 threads should start the animation thread and the genesis output thread.

2 Paths:

1 – Writing the code → Save → Compile → Animate

2 – Drag Objects → Save → Interpreted Code → Compile

For 1 – Outputs are : Animation and Compilation log

Table of Contents

1. INTRODUCTION 1
1.1 Purpose 1
1.2 Scope 1
1.3 Overview 1
1.4 References 1
1.5 Definitions and Acronyms 2
2. SYSTEM OVERVIEW 2
3. SYSTEM ARCHITECTURE 2
3.1 Architectural Design 2
3.2 Decomposition Description 7
3.3 Design Rationale 11
4. DATA DESIGN 11
4.1 Data Description 11
5. COMPONENT DESIGN 11
6. HUMAN INTERFACE DESIGN 13
6.1 Overview of User Interface 13
..... 14
7. REQUIREMENTS 15
8. APPENDICES 15
Technologies used ..... **Error! Bookmark not defined.**
9. Testing 15
9.1 Example Algorithms 15
9.2 Scenarios 18
10. Future Developments 18