Spring 2017

Game Collection Development and Marketing

Todd R. Locker Jr
The University of Akron, TRL43@zips.uakron.edu

Honors Research Project
Department of Computer Science
Game Collection Development and Marketing
Todd R Locker Jr
Fall 2016 – Spring 2017

Abstract

The reasons for completing this project include expanding my knowledge of programming features and methodology, and to learn about game platform development and design. This was done by using the Java programming language to create a game platform which can host a variable number of games. One of the most notable features that was utilized is the JavaFX platform. Much programming experience was gained from this platform and all the features and methods it provides for customization. The application that was produced is graphical user interface based and created with event-driven programming. The result of this project was a custom game platform and a couple of games that can be played from the platform.

Background and Motivation

Throughout my whole life, I have enjoyed playing games on a variety of platforms. I enjoy experimenting with the mechanics of games, trying to figure out how something may be coded. The artwork put into many modern games is tremendous and something I find fascinating. Up until just a couple of years ago, the inner workings of games were a mystery to me. After beginning college at the University of Akron, I discovered my fondness for computer science and thus got the opportunity to explore and learn about this long hobby of mine. I have also successfully acquired a full-time position for after graduation dealing with software development using object-oriented programming and many of the other skills I acquired here at the University of Akron. I wanted to choose a project that would advance my object-oriented skills and other various skills.

Introduction

The focus of this project was to gain insight into the steps of developing games, and to gain broad experience in creating games and marketing them with the use of a website. To do this, an application was created that is a “game platform” which hosts multiple games. Some of the games that can be added to and played in the platform were also developed as part of this project. To gain experience in the marketing aspect, a simple website was created similarly to how modern game developers create their websites for users to download and play the games created by them.

Methods and Results

Considering all the requirements that must be fulfilled to complete this project, the Java programming language was chosen to write the game platform. The biggest reason for this decision was due to the need for a graphical user interface (GUI) aspect within the language. An integrated development environment (IDE) called Eclipse was selected to be used to write the code. Eclipse is the most popular IDE used for Java due to all the functionality it provides. The first step taken in the project was to research ways to achieve the goals set. Through research, a discovery was made of Oracle's recent release of a new Java platform called JavaFX. JavaFX replaces the older Swing platform which was previously the best way to create a GUI using Java. Utilizing JavaFX was a perfect way to explore something new and unfamiliar, which was one of the main foci of this project.

The GUI is needed in this project to simplify, direct, and constrain the user of the application. For most occasions when the user is required to have input into the application, it is best to provide the user with a simple and straightforward way to do so. This is to help with usability, and is also a great way to prevent errors caused by incorrect input. If the user has a defined set of choices, the likelihood of incorrect input drastically decreases. The goal of creating an efficient and effective GUI revolves around two key aspects: look and feel. JavaFX provides a multitude of features and customizations allowing developers to modify these two aspects as desired.

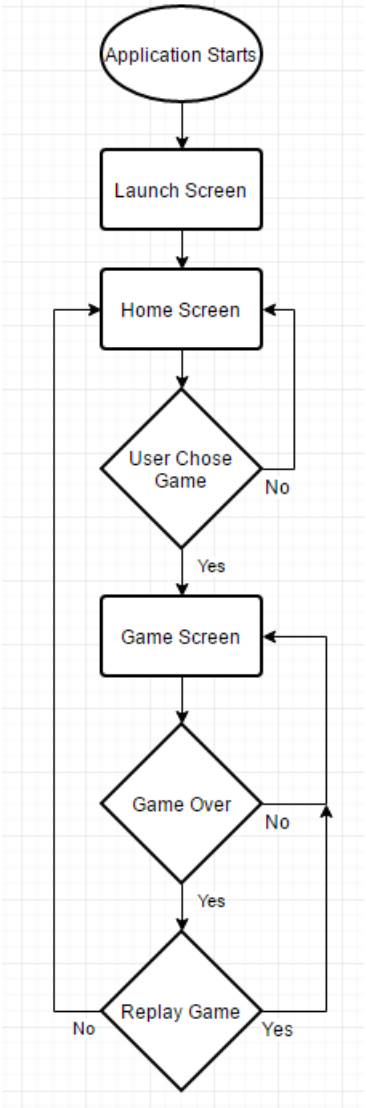
JavaFX provides built in functionality for handling applications, which can be done by extending the Application class. This class provides the initialization, entry point, and termination of the application. The application is terminated either when all the GUI stages are closed, or the exit method is explicitly called. The type of programming that is used due to the utilization of this functionality and the GUI is called event-driven programming. Event-driven programming is basically programming that relies on actions made by the user to determine the flow of the application. While the application is running, the code is in an infinite loop called the event loop. Code for all this functionality can be seen in figure 1 below.

Figure 1 – Code for launching and initializing the application

```
public class Main extends Application {

	public static Stage currentStage;

	public static void main(String[] args) {
		launch(args);
	}

	@Override
	public void start(Stage primaryStage) throws Exception {
		currentStage = primaryStage;
		primaryStage.setTitle("Game Collection");
		primaryStage.centerOnScreen();
		primaryStage.setResizable(false);
		primaryStage.setScene(new LaunchScene());
		primaryStage.show();
	}
```

After deciding what methods and tools would be used first to implement this project, a high-level design was created. This high-level design was used to diagram the architecture of the application to provide a visual representation of what would like to be accomplished. Despite having tools and a design chosen for this application, it was noted that flexibility must be maintained to account for any challenges that arise during the process. The high-level design diagram can be seen below in figure 2.

Figure 2 – High level diagram of the flow of the application



Within JavaFX, there are 3 basic layers that are used to structure the application. The first layer is called the **stage**. This layer contains everything that will be in the application. It can be thought of as the window that the application opens up in. The second layer is called the **scene**. Scenes are what contains the actual objects created in the application. When switching views or tasks, a new scene can be created and then used to replace the old scene currently in the stage. The third layer is called **nodes**. Nodes are the objects that are added to scenes. Examples of nodes include buttons, text, graphics, lists, and fields. For each layer, there are methods available to set a variety of options such as positioning and dimensions. Objects called layouts are used in scenes to control how nodes are placed within the scene. Layouts can be placed inside other layouts, allowing for a lot of customization when it comes to placing and sizing nodes.

As can be seen in the high-level diagram above, there are three main scenes that are part of the application. These scenes do not represent methods or classes, but rather illustrate the flow of the application as the user progresses through each part. The 3 scenes are referred to as the **launch scene**, the **home scene**, and the **game scene**. The following paragraphs will explain what the purpose of each scene is and describe the internal structure of each scene in more detail.

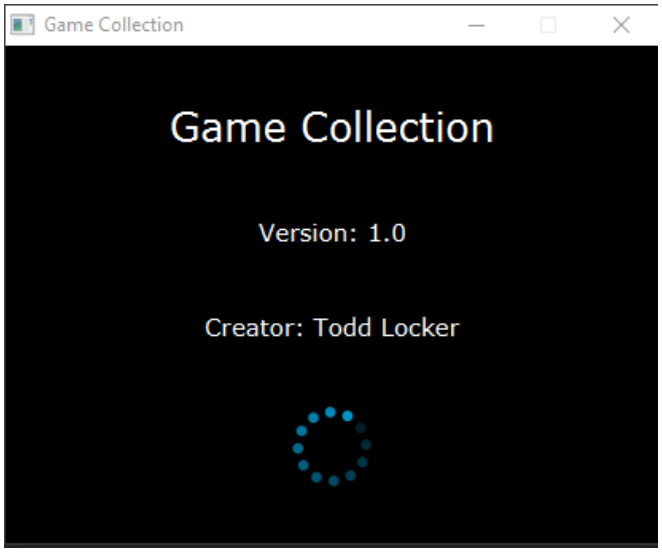
The first scene encountered is the launch scene. On this scene, the title of the application is displayed along with the version and author of the application. There is also a loading animation. JavaFX provides a variety of progress controls

in the forms of text, images, and animations that can be used to inform the user of the progress of loading something. The decision was made to utilize a progress indicator that does not quantify the progress, but rather simply informs the user that the application is loading. This was because this scene is displayed for a set amount of time, and quantifying progress is something that is very difficult to do, as even large software companies run into problems doing so correctly. There are no actions for the user to take in this scene other than to wait a second for the next scene to replace this scene. This scene has its own designated class which extends the JavaFX class Scene. It is created in the Main class simply by using the code '*new LaunchScene()'* to call the constructor. The launch scene itself and the code to create the progress indicator described earlier are shown below in figures 3-a and 3-b.

Figure 3-a – Code to create the progress indicator

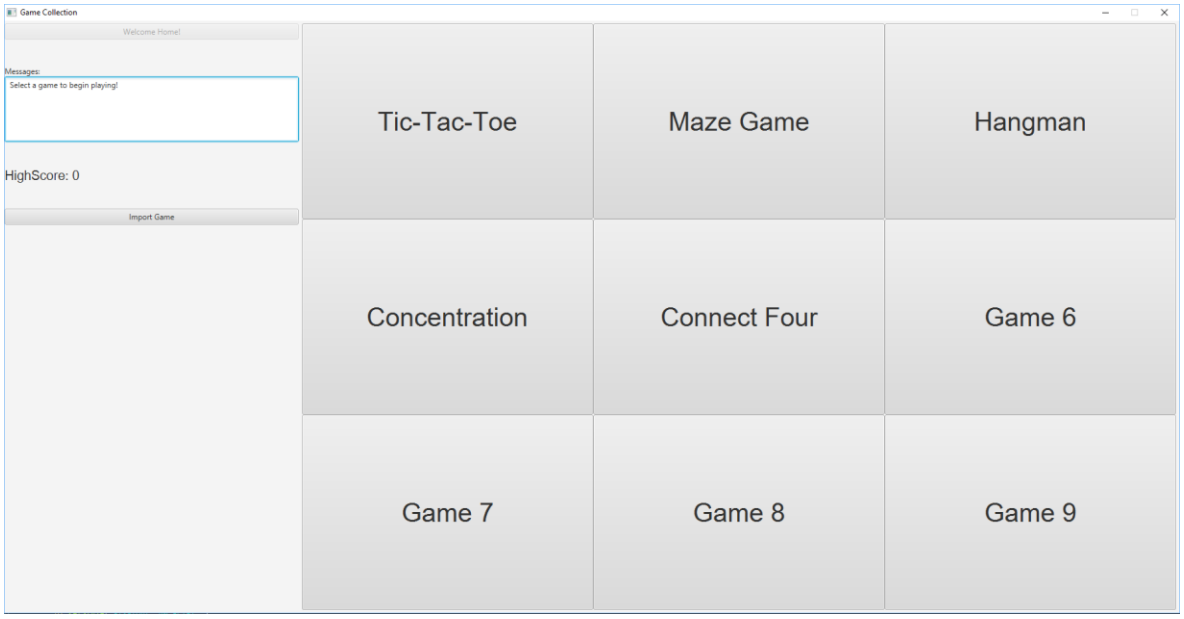
```
// Set the launch scene progress indicator
ProgressIndicator indicator = new ProgressIndicator;
indicator.setProgress(-0.1);
indicator.setMaxSize(50, 50);
indicator.setLayoutY(100);
```

Figure 3-b – The launch scene



The second scene the user will encounter is the home scene. This is the main part of the game platform. The home scene is divided into two sections. The section on the left, taking up 25% of the scene and referred to as the “overview area”, displays a message on how to play a game. It also displays scores of the games within the library. The right section taking up the other 75% of the scene, referred to as the “play area”, contains buttons, each labeled with the name of a game. When the user clicks on a button, they will begin to play the game. This scene is in its own class called *HomeScene*. The class extends a custom abstract class called *GameApp* which contains the constructor code to set up the base grid layout of the scene, an abstract method to get the overview area, and an abstract method to get the play area. A screenshot of the home scene is shown below in figure 4.

Figure 4 – The home scene



The third main scene in this application is the game scene. This is the scene that the user will play some of the games in. If a game is imported from outside this application, the game is run using the exec method and will not appear in the game scene. The game scenes are set up with similar layout to the home scene. This is to provide efficiency and simplicity for the user. To achieve this standard, each game scene extends the custom abstract class *GameApp*, just as the home scene does. There will be many game scenes, as each game makes up its own class and is called only when the corresponding event is triggered. The overview area contains a back button to return to the home scene. It also contains instructions for playing the game. Third, there is a running list of system generated messages for the user to inform them about various events,

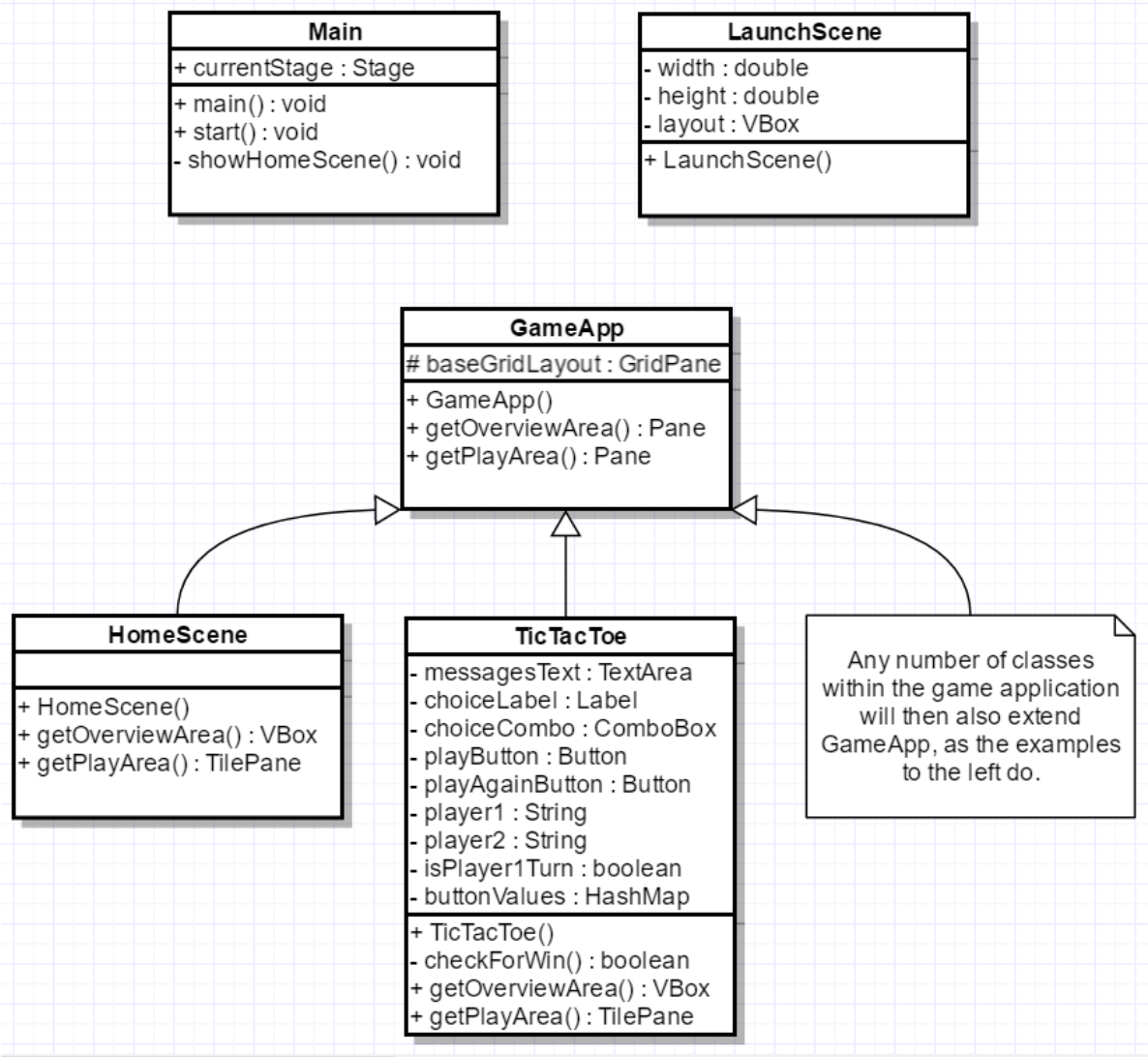
warnings, or information about their current game. Code for how a game may be called from the home scene is shown below in figure 5.

Figure 5 – Example of how a button to launch a game is created

```
Button b1 = new Button("Tic-Tac-Toe");
        b1.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
        b1.setStyle("-fx-font: 40 arial;");
        b1.setOnAction(new EventHandler<ActionEvent>(  {
                @Override
                public void handle(ActionEvent e) {
                        TicTacToe G = new TicTacToe();
                        Main.currentStage.setScene(G);
                }
        });
```

The Unified Modeling Language (UML) class diagram below in figure 6 shows examples of the classes used within the game platform. The arrows represent that the class *extends* the class the arrow is pointing to. There can be a variable number of these classes. This design decision was made to allow for consistency and simplicity in creating and adding to the platform.

Figure 6 – UML diagram



Discussion and Future Work

The project completed was a very good learning experience. Having no prior knowledge of JavaFX, discovering it and its many features have helped improve my programming skills. Also, the work on the website created has strengthened my skills in HTML, CSS, and JavaScript. This project provides many opportunities for expansion and refinement. Future work includes creating more games to add to the game platform, or improving upon existing games. The logic could be improved to provide more statistics about each game or averages for the games. Functionality could also be improved to make the GUI more dynamic and customizable by the user.

References

"Java Platform, Standard Edition (Java SE) 8." *http://docs.oracle.com/javase/8/javase-clienttechnologies.htm.* Oracle, 2016. Web. April 2017.

"JavaFX Tutorial." *http://www.tutorialspoint.com/javafx/*. Tutorials Point, 2017. Web. April 2017.

"Eclipse." *https://eclipse.org/*. Eclipse, 2017. Web. April 2017.

"HTML5 Tutorial." *https://www.w3schools.com/html/default.asp*. W3Schools, 2017. Web. April 2017.

"CSS Tutorial." *https://www.w3schools.com/css/default.asp*. W3Schools, 2017. Web. April 2017.

"gliffy." *https://www.gliffy.com/products/online/*. Gliffy, Inc, 2017. Web. April 2017.

Holub, Allen. "Allen Holub's UML Quick Reference." *https://holub.com/uml/*. Allen Holub, 2017. Web. April 2017.

Ambler, Scott. "UML 2 Class Diagrams: An Agile Introduction." *http://www.agilemodeling.com/artifacts/classDiagram.htm*. Agile Modeling, 2014. Web. April 2017.

Pankaj. "Java Design Patterns – Example Tutorial." *http://www.journaldev.com/1827/java-design-patterns-example-tutorial*. JournalDev, 2016. Web. April 2017.

Pawlan, Monica. "What Is JavaFX?" *http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm*. Oracle, 2013. Web. April 2017.