An evaluation of JavaFX as a 2D game creation tool

Hamilton Lima Jr
Media Lab – UFF

Fábio Corato de Andrade
Media Lab - UFF

Anselmo Montenegro
Media Lab - UFF

Esteban Clua
Media Lab - UFF

Abstract

With the current growth in the user experience and the existence of multiple publishing platforms, the investigation of new game creation tools that simplify the development process, is important to reduce costs and increase the overall quality of the products.

Based on this perspective, we present an analysis of the JavaFX technology as a tool for 2D game development. For instance, we will focus the evaluation on the following features: deployment, scripting support, vector graphics support, flexible main loop, sprite caching, collision handling, audio support and distribution license.

Keywords: 2d games, JavaFX, tool evaluation, RIA

Authors' contact:
{hlima,anselmo,esteban}@ic.uff.br
fabio.corato@ig.com.br

1. Introduction

JavaFX [JAVAFX] is a GUI (Graphic User Interface) framework created by Sun Microsystems, based on a script language that merges XML definitions with embedded JavaScript-like code. JavaFX can use pure Java classes integrated in the scripts, which allows the enhancement of existing Java applications with a modern look and feel. With the perspective of a rich user experience, our study is based on the creation of a video game.

This analysis is based on the experience of creating a side scrolling platform video game using the JavaFX script technology. During the experiment of the game creation several decisions were made in order to accommodate the technology and the expected results of the game, most of these decisions are described in the paper separated in two main topics: The process of development and issues found during this process.

At the process of development topic, we will go over the integration with the design team, the sprites caching management, the organization of the scripts files and the collision handling. The Issues found topic will describe the current audio support of JavaFX, the difficulties to deploy the game and some license restrictions.

2. Development process

For this experience we created a port of an existing game submitted to the Global Game Jam 2009, called Tiny Soldiers the Rise of Mosquito [TINYSOLDIERS], that is a side scrolling game created in XNA.

2.1 Game main loop

Most of the available samples of JavaFX are organized in only one big script due to the simplicity of the samples, in our case we will enforce that the separation of script files will allow an object oriented organization of the script files. Our main script will be the Main.fx file that will have all the declaration of the instances used in the game.

First of all this script will have the instance of the Game object that is an extension of the Stage class.

The Game.fx class uses a TimeLine [JAVAFXAPI] object to control the game main loop; by using this type of object we can control the speed of the game by changing the time parameter of the Keyframe object inside the TimeLine

```
public class Game extends Stage {
 public var tick: Timeline = Timeline {
   repeatCount: Timeline.INDEFINITE
   keyFrames: [ KeyFrame {
       time: 10ms
       action: function() {
         mainLoop();
   }}]
 };

 public function mainLoop(){ }
 public function play(){ tick.play(); }
}
```

Code 1: *Creation of the main loop*

The "KeyFrame" object have an attribute "time" that is from the type Duration, that allows the usage of milliseconds, seconds and minutes or combination of the three. Language features like this add simplicity to the code and make the development process more intuitive.

The mainLoop() method check for objects that extends the "Updatable" class and call update() method of each one propagating the game tick for the objects that need update in the game.

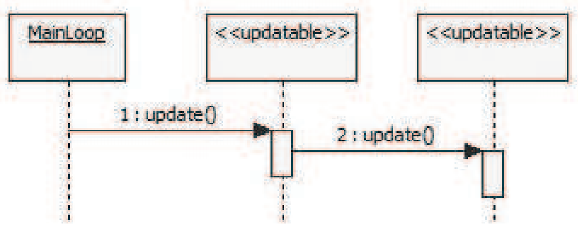

Figure 1: *Main loop sequence diagram*

In order to use the “Game” class, the Main.fx script creates a “Game” instance and call the play() method, this starts the timeline and keep the game in constant loop.

```
var game: Game = Game {
  title: "Tiny Soldiers ... "
  x: 0
  y: 50
  width: 800
  height: 600
  scene: Scene {
    content: bind currentGroup
  }
  fullScreen: false
}

function run(__ARGS__ : String[]) {
  game.play();
  soundtrack.play();
}
```

Code 2: *The Instance of the Game class*

The run() method of the “Game” instance is the JavaFX implementation of the Java main() method. In order to create the initial state of the “Game” object, all attribute changes and instance attribute creation should be defined inside the curly brackets.

2.1 Design team integration

Game development teams need someone to fill the artistic role. This role will create the environment, the GUI, the characters and NPC's (non player characters) and all visual behavior of the game. To support this role JavaFX offers support to uncompressed and compressed bitmaps files. In addition to images support, a production suite integrates JavaFX with design tools as Adobe Photoshop [PHOTOSHOP] and Adobe Illustrator [ILLUSTRATOR], beside that also offer a converter that reads SVG [SVG] files and convert then to FXZ files, that are the standard format used for images definition at JavaFX and can be read directly in the code.

As the design team can work in parallel with the programming team, we can have colored rectangles working as game characters during the development process; what can be achieved within most of existing game frameworks. With the use an SVG files converted to FXZ format is possible to add a unique identifier attribute to individual elements of drawings, usually know as ID, and these are the reference used in the script to manipulate the images that can be changed by the design team without any sort of change in the code.

The Figure 2 shows the character that was created in the SVG format and imported to the JavaFX project. After importing the SVG to the project we can use the method lookup() to retrieve references of individual parts of the image, and manipulate it, see a sample in the Code 3.

```
(player1.lookup("JFX:body")
   as Rectangle).fill = Color.BLUE;
```

Code 3: *Changing one object fill color*

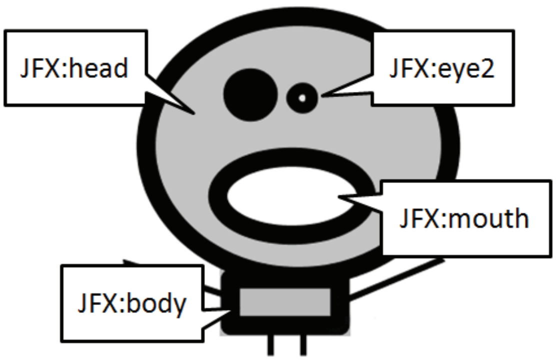

Figure 2: *Imported SVG file*

This approach allow the use of the FXZ files as integration artifact that won't need any intervention from the programming team in order to work properly in the game, this integration allow the use of complex objects and have its visual aspect changed by programming instead of having multiple sprites for the different states of the visual object.

2.2 Sprite caching

Vector images in the SVG format, are compressed with ZIP algorithm to create the FXZ files. SVG files can be as simple as a circle with a center point and a radius, or can be as complex as hundreds of shapes and points, build together to make complex illustrations, that will be read and interpreted by JavaFX every time the FXZ file is used by the game.

When complex illustrations composed by several shapes and fills, are used in the JavaFX scripts, a call to FXDLoader.load() will force the parsing of the FXZ file, as most of the times some illustrations are reused in the game, there is a need to avoid all this parsing every time the illustration is used, The Code 4 show the caching mechanism implemented to load the FXZ files only once and enable the reuse of the objects.

```
public class NodeFromFXZPool {
 var cache: HashMap = new HashMap();
 public function get(source:String):Node{
  var content:
    Node = cache.get(source) as Node;

  if( content == null){
    content =
      FXDLoader.load(source) as Node;
    cache.put(source, content);
  }
  return Duplicator.duplicate(content);
 }
}
```

Code 4: *Sprite caching implementation*

The key of the caching mechanism implementation is the Duplicator [JAVAFXAPI] class that uses and existing Node definition and create an independent copy of it.

2.3 Scripts Organization

In order to organize the code in classes we separated the classes in .fx files, but this created an problem when defining the different Scene [JAVAFXAPI] objects of the game, because we need to change the main game instance in order to indicate Scene changes. This would be simple to achieve if the scope of the event handling methods belong to the objects created, but in JavaFX the scope of the created methods belong to the script where the method were created.

In this example we have the definition of an instance of the HowToPlayGroup that will be show as current scene of the game and when the onClick method is call the currentGroup is changed to a value that is declared in the Main.fx

```
var   howToPlayScene:   HowToPlayGroup   =
HowToPlayGroup {
    onClick: function(){
        currentGroup = menuScene;
    }
}
```

Code 5: *Use setting the Group callback*

In order to create this we build a callback solution in the scene definitions to avoid coupling with the main script,

```
public class HowToPlayGroup extends Group
{
  public var onClick: function(): Void;
  ...
  onMouseClicked: function(e){
    if( onClick != null ){
      onClick();
    }
  }
  ...
}
```

Code 6: *Creating the callback in the Group*

With this callback strategy scenes can be treated as independent artifacts that can be created and tested without the main script, reducing external dependencies and the coupling to the main Stage.

2.4 Collision Handling

In JavaFX games in order to check the collision of game elements we use the Rectangle.intersect() method that is provided with the language. There is a possibility of having more complex collision handling, this need additional implementation, that could be iterating over the existing points from an imported SVG file or from a script based polygon.

One workaround to this restriction is presented by Silveira Neto [NETO, SILVEIRA 2008], and we can see at Figure 3, where a bounding box is created at the coordinates x=4 and y=25 inside the game object itself, so part of the object really overlaps the collision target when the collision happens, offering to the player the visual feedback of the collision.

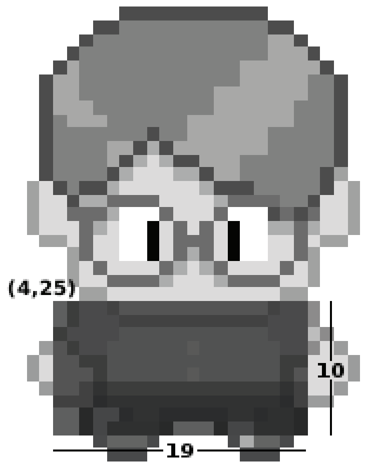

Figure 3: *Bounding box smaller than the image*

3. Issues

3.1 Audio support

The current version of JavaFX use a video/audio decoder created by On2 [ON2]. This decoder offers support to multiple video formats, and claims that offers support to MP3 files. After some tests with different MP3 files encoded with different frequencies and different quality, JavaFX wasn't able to play most of the combinations; Table 1 shows some tested combinations.

Frequency	Encoding	Duration	File size	Result
48000Hz	128bits	> 1sec	456K	Failure
48000Hz	**96bits**	**> 1 sec**	**341K**	**Success**
48000Hz	32bits	> 1sec	114K	Failure
48000Hz	16bits	> 1 sec	57K	Failure

Table 1: *MP3 combinations tests*

In order to solve this MP3 restriction, we implemented a new version of MediaPlayer class, integrating the Jlayer library [JLAYER] objects in AbstractAsyncOperation objects. With this solution all the combinations described in the Table 1 was play with success. The same solution can be used to add support to OGG, WAV or any other audio format.

3.2 Deploy of the game

We tested the Applet and Web Start [WEBSTART] deploy of JavaFX applications. For the Applet or the Web Start deploy the user needs to download the JavaFX Runtime environment that can't be released with the application due to license restrictions, that force the end user to be online at the first run of the game. This restriction is a roadblock to the usage of JavaFX as solution to create standalone games, where the users don't need internet connection to play.

Using the Web Start solution the end user is forced to handle dialogs in English, without an option to translate to the user's language; this is a serious restriction for publishing games to the general public in special for the Brazilian market. Other issue on using the Web Start solution is the fact that even with Java Runtime Environment installed Web Start application files are not automatically executed, what adds an extra complexity to the end users, in order to execute the Web Start file.

Another issue when using the JavaFX as an Applet is the fact that the whole applet must be downloaded before the user can start the interaction, this generate a high level of frustration due to the fact that Applet download don't give the user a feedback of the percentage of the download. A Java Game engine named Pulpcore [PULPCORE] based on applets solved this issue, for plain Java Applet games, by creating a small applet with less than 200k in size, which loads the real application applet, and can show to the user a feedback of the percentage of the game download.

4. Related work

Despite the fact that JavaFX is new and still with some bugs, there some casual games developed, there is a Pacman clone [PACMAN] and the Brick Breaker [BRICK] that can be found at the JavaFX documentation.

5. Conclusions

Based on the fact that JavaFX offers a full integration with existing Java code, we can assume that JavaFX has a good chance to be the next natural GUI framework choice for Java games and applications.

Related to the evaluation itself, we can conclude that JavaFX can be used to create game applications with some restrictions. The current audio support has restrictions related to MP3 files and no OGG files support. Only controlled environments where the users can make sure that access to the Internet is available, is the expected deployment scenario in order to download the JavaFX runtime environment.

Follow the summary of the JavaFX evaluation using the 2D game development challenges table:

Deployment	Online required, >10mb runtime download
Scripting support	Full with JavaFX script
Vector graphics support	Can be imported to framework script
Flexible main loop	Created with TimeLine
Sprite caching	Can be done
Collision handling	Rectangles only
Audio support	Poor mp3 support, no OGG
Distribution license	Can't distribute standalone runtime, need download

Table 2: Summary of JavaFX evaluation

We see as expansions to this research the investigation of multi-player games created with web technologies especially with JavaFX. Other future research related to JavaFX would be the investigation of 3D possibilities, in particular the integration with Jmonkey or Java3D engine.

References

BRICK, Brick Breaker JavaFX game sample, http://javafx.com/samples/BrickBreaker

ILLUSTRATOR, http://www.adobe.com/products/illustrator

JAVAFX, http://javafx.com

JAVAFXAPI, JavaFX API documentation, http://java.sun.com/javafx/1/docs/api/index.html

JLAYER, Pure Java mp3 library, http://www.javazoom.net/javalayer/javalayer.html

NETO, SILVEIRA 2008, How to create a RPG like game, http://silveiraneto.net/2008/12/08/javafx-how-to-create-a-rpg-like-game

ON2, On2 technologies, http://www.on2.com

PACMAN, JavaFX Pacman clone, http://www.javafxgame.com/javafx-pac-man-article-5

PHOTOSHOP, http://www.adobe.com/products/photoshop

PULPCORE, http://www.interactivepulp.com/pulpcore/

SVG, http://www.w3.org/TR/SVG

TINYSOLDIERS, Tiny Soldiers the Rise of Mosquito, Global Game Jam 2009, http://globalgamejam.org/games/tiny-soldiers-rise-mosquitos

WEBSTART, http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp