An evaluation of JavaFX as 2D game creation tool

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 th 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
- distribution license

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

Authors' contact:

1. Introduction

JavaFX [1] is a GUI (Graphic User Interface) framework created by Sun Microsystems, based on a script language that merges XML[2] definitions with embedded Javascript-like[3] code. JavaFX can use pure Java classes integrated in the scripts, what make possible to enhance existing Java applications with a modern look and feel. With the perspective of an rich user experience, our study is based on the creation of a video game.

This guideline 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 explain the current audio support of JavaFX, the difficulties found to deploy the game and some license restrictions of the JavaFX.

1.1 Web application complexity

Web applications filled a gap of good presentation for the end user and solved the critical problem of application distribution, but as the time goes the continuous expectation of a more intuitive experience by the end user force HTML based applications to its limits, making lots of different technologies to be tied together in order to build richer user experience. What in the beginning was a good solution to easily distribute applications using plain HTML, become a technology nightmare with HTML, JavaScript, CSS, Flash [4] all together, excluding from the list the server side technologies as JSP/Servlets, PHP and ASP.

We believe that this proliferation of web technologies , pushed a natural movement to reduce the number of technologies need in order to solve the same problems. This movement has taken place in the last years with a new way to build an web presentation layer, where the technologies come with a rich set of possibilities in only one box, a report from Zapthink [5] adds to the need of a rich user experience the expectation of decentralizing computing so the user has best cost/IT assets available, and explain that these two forces were the key for the RIA (Rich Internet Application) born.

1.2 RIA Technologies

The main players of these new way to build applications are : Adobe Flex [6]/ AIR [7], Silverlight [8] and JavaFX. All can run over web browsers [9] and also run as desktop applications, moving the applications to a client-server like model, with a difference that the server communication now is based on web services.

When comparing these three solutions based on the age of the technology, Adobe Flash based solutions, that are Adobe flex / One should be considered the most mature solution since Flash was introduced in 1996 [10] and Adobe Flex itself was introduced in 2005 some years before Silverlight (2008) and also before JavaFX (2008).

One of the main advantages and obstacles of RIA technologies is the fact that a player needs to setup at the user machine in order to work, Flash technologies have an clear advantage on this race, most of the PC's (Personal Computer's) have a flash player installed.

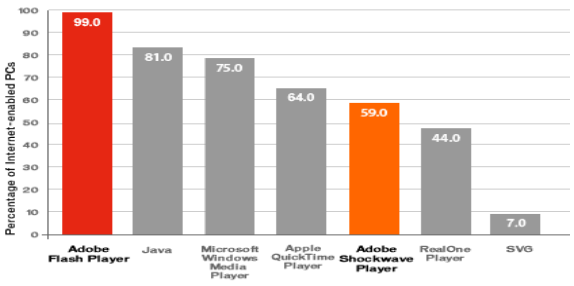

Image 1: Flash player penetration[11]

But even with this great penetration in the client side, when we consider number of job offerings as market adoption number, we see at image 2 that both Adobe Flex and Silverlight from Microsoft are getting a similar performance in jobs offering numbers, even with the hype of JavaFX in the 2008 Java One, the official 1.0 was in December of 2008 followed some months later by a 1.1 release, based on this we will have to wait a little bit more to see the market reaction to JavaFX.

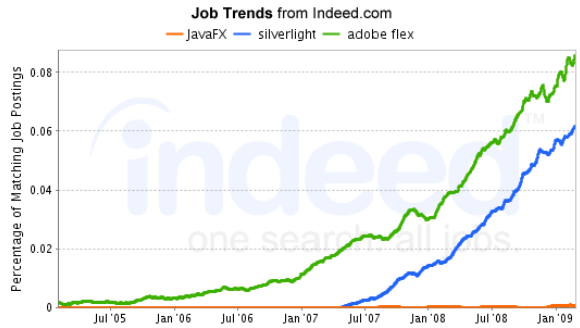

Image 2: Job offers in RIA market

We strongly believe that the current Java community can embrace JavaFX technology based on the current number of job offers of Java, see Image 3, compared with other technologies related to Flash and Microsoft (C#, Vbscript) and Javascript, we can see the potential growth of JavaFX as a RIA option for the Java community.

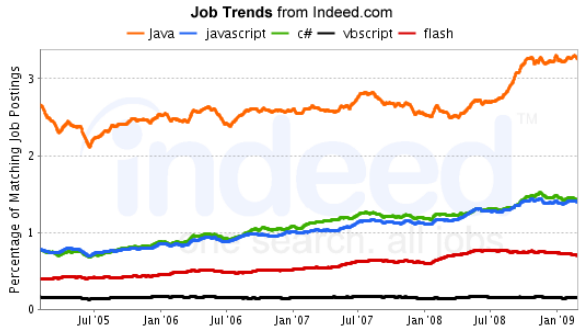

Image 3: Java job offers and other technologies [13]

Currently game development in Java have some options beside the pure Java development or applet based games, some frameworks are available in the market, Jmonkey [14], JPCT [15], Pulpcore [16] and GTGE [17], all use Java code for the game development, the possibility of use an script language as JavaFX can increase the productivity of the games, and low the cost of maintenance.

2. Development process

For this experience we created a port of an existing game used for the Global Game Jam 2009 [18], that was originally created in XNA, and all the source code of the game is available for download [19] using subversion.

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 a 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 that the game will use.

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 an TimeLine [20] 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 [12] 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();
}
}
```

One special features of the language to highlight is the Duration type that allows the usage of milliseconds, seconds and minutes or combination of the three, features like this and the simplicity of the created code 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.

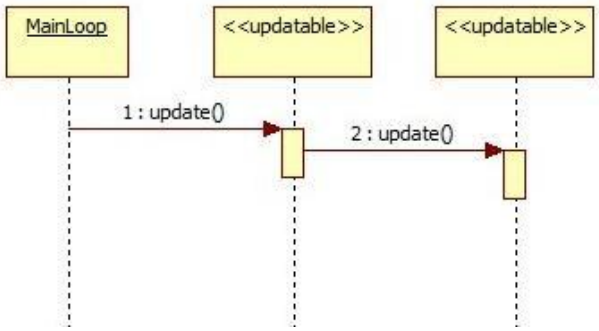

Image 4: Main loop sequence diagram

In order to use the Game class the Main.fx script creates an instance of it and call the play() method that starts the timeline and keep the game in constant loop.

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

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

Note that the instruction

```
var game: Game = Game {}
```

Creates an instance of the Game class and the initial state of this object is defined inside the curly brackets and the run method is the JavaFX implementation of the Java main() method.

2.1 Design team integration

Game development teams need someone to fill the artistic role, in order to create the environment, the GUI, the characters and NPC's (non player characters). JavaFX offers support uncompressed and compressed bitmaps files, beside that offers a production suite integration with some popular design tools as Adobe Photoshop [21] and Adobe Illustrator [22] beside that also offer an converter that reads SVG [23] files and convert then to FXZ files, that are the standard format used for images definition at JavaFX and can be read in the code.

As the design can work in parallel with the programming team, we can have colored rectangles working as game characters during the development process, this can be achieved within most of existing game frameworks. With the use an SVG files converted to FXZ format is possible to add an 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 image 4 show the character created in the SVG format after imported to the JavaFX project, and the code bellow make the change in part of the object.

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

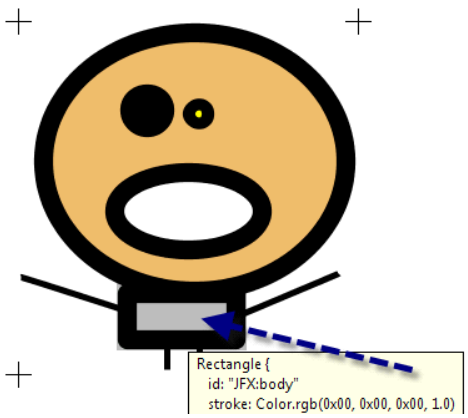

Image 5: Imported SVG file

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

FXZ files are .Fx files compressed with ZIP [24] algorithm, can most of the times are result of the conversion from an SVG file. SVG files that are XML based can describe for example an Circle within small number of parameters, but depending of how the artist build the Circle it can be defined by hundreds of nodes, this demonstrate of how intense can be the processing of FXZ files every time we need then in the game. This is more clear when we have complex illustrations composed by several components, and every call of FXDLoader.load() [12] will force the parsing of the FXZ files, in order to avoid this we implement an caching mechanism to load the FXZ files only once and reuse.

```
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);
    }

}
```

The key of the caching mechanism implementation is the Duplicator [12] 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 [12] 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;
    }
}
```

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();
      }
    }
    ...
}
```

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 an script based polygon.

One workaround to this restriction is presented by Silveira Neto [25] where a bounding box smaller than the game object itself is created so part of the object really overlaps the collision target when the collision happens, offering to the player the visual feedback of the collision.

Image 6: bounding box smaller than the image

3. issues

3.1 Audio support

The current version of the engine use an third party video/audio decoder created by On2 [26][27], that offers support to multiple video formats, and claims that offers support to mp3 [28] files. After some tests with different mp3 files encoded in different frequencies and different quality JavaFX wasn't able to play most of the combinations, the table1 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

For sound effects that multiple files are used, the possibility of use 16bits against 96bits when encoding the audio files would the audio footprint of the game six times smaller.

An solution found for this mp3 restricted support were to implement a version of MediaPlayer that support Mp3 by using the integration of the Jlayer library [29]. This implementation [30] were made using a combination of an Java and JavaFX implementation of AbstractAsyncOperation. The same solution can be used to add support to OGG [31] files and WAV [32] files or any other audio format.

3.2 Deploy of the game

We tested the Applet and webstart [33] deploy of JavaFX applications. For the Applet or the webstart 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 on line when first runs the game. This restrictions is a roadblock to the usage of JavaFX as solution to create standalone games, where the user don't need internet connection to play.

Using the webstart 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 webstart solution is the fact that even with Java Runtime Environment installed webstart application files are not automatically run, what adds an extra complexity to the end users, in order to execute the webstart 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. Pulpcore Other game solution based on applets, solved this issue by creating an small applet with less than 200k, size depends on messages or custom actions in the loader, that loads the real application applet, and can show to the user a feedback of the percentage of the download.

4. Related work

Despite the fact of JavaFX is new and still with some bugs, there some casual games developed with this technology. There are few game implementations in JavaFX at this time, Pacman [34] and Brick Breaker [35] are examples of the use of the technology in 2D game creation.

Image 7 Pacman clone


Image 8 Brick break

5. Conclusions

Based on the fact that Java language itself has a good popularity (see image 3), and 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 2D game development challenges

We see as expansions to this research the investigation of multi-player games created with web technologies specially 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

[1] JavaFX, http://javafx.com/
[2] XML, http://www.w3.org/TR/REC-xml/
[3] Javascript, http://pt.wikipedia.org/wiki/JavaScript
[4] Flash, http://www.adobe.com/products/flash/
[5] Rich Internet Applications: Market Trends and Technologies, http://www.zapthink.com/report.html?id=ztr-ws112
[6] Flex, http://www.adobe.com/devnet/flex/
[7] Adobe AIR, http://www.adobe.com/products/air/
[8] Silverlight, http://silverlight.net/
[9] Wikipedia - Rich Internet Applications, http://en.wikipedia.org/wiki/Rich_Internet_application
[10] Wikipedia Flash Player, http://en.wikipedia.org/wiki/Adobe_Flash

[11] Flash Player penetration , http://www.adobe.com/products/player_census/flashplayer/
[12] Job offers in RIA, http://www.indeed.com/jobtrends?q=JavaFX,+silverlight,+adobe+flex
[13] Job offers Java against JavaScript, C#, VbScript, Flash, http://www.indeed.com/jobtrends?q=Java,+javascript,+c%23,+vbscript,+flash
[14] Jmonkey, http://www.jmonkeyengine.com/
[15] Jpct, http://www.jpct.net/
[16] Pulpcore, http://www.interactivepulp.com/pulpcore/
[17] GTGE game engine, http://www.goldenstudios.or.id/products/GTGE/
[18] Global Game Jam, http://globalgamejam.org/
[19] Project source code subversion, SUPRESSED – HAS INFORMATION ABOUT THE AUTHOR
[20] JavaFX API documentation, http://java.sun.com/javafx/1/docs/api/index.html
[21] Adobe Photoshop, http://www.adobe.com/products/photoshop/photoshop/?promoid=DTENB
[22] Adobe Illustrator, http://www.adobe.com/products/illustrator/
[23] SVG, http://www.w3.org/TR/SVG/
[24] ZIP, http://en.wikipedia.org/wiki/ZIP_(file_format)
[25]How to create a RPG like game, http://silveiraneto.net/2008/12/08/javafx-how-to-create-a-rpg-like-game/
[26] JavaFx media support, http://javafx.com/docs/articles/media/player.jsp
[27] On2 technologies, http://www.on2.com/
[28] MP3, http://en.wikipedia.org/wiki/MP3
[29] Jlayer - pure Java mp3 library, http://www.javazoom.net/javalayer/javalayer.html
[30] JavaFx MediaPlayer using Jlayer, SUPRESSED – HAS INFORMATION ABOUT THE AUTHOR
[31] OGG, http://en.wikipedia.org/wiki/Ogg
[32] WAV, http://en.wikipedia.org/wiki/WAV
[33] webstart, http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp
[34] JavaFX game - Pacman, http://www.javafxgame.com/javafx-pac-man-article-5/
[35] JavaFX game - Brick breaker, http://javafx.com/samples/BrickBreaker/