Project AstroVault

Technical Plan

Jack Storm

EGD-220-03 | Project 3 | Team 2
Final

A collection of technical categorizations outlining the structure behind *Project AstroVault* from Team 2.

1.0 Table of Contents

2.0 Introduction

2.1 Document Overview

This document contains the technical plans for our *Project AstroVault.* In it, you will find details on the technology used, the artist pipeline, the design pipeline, the systems involved with making the game along with other documentation including sprint schedules and agile development.

2.2 Document Revisions

This section of the document contains notes about the different revisions to the technical plan over the course of development.

2.2.1 Revision – 03/23/2014

This is the first version of the technical plan. The document itself was entirely reformatted to address the new game and most of the sections are drafts and or missing.

2.2.2 Revision – 3/30/2014

The technology section has been expanded upon to incorporate the Citrus Engine. Blank Sections have been filled in, completing the first full draft of this document.

2.3 Special Thanks

Thanks goes out to the members of the team and members of QA for making this game possible. We would also like to mention special thanks to Peter Wehr. I would like to mention special thanks to the following:

Jason Sidor, for his help with Technical Design Documents.

David Johnston, for his help with Citrus.

Dan Mclean, for his help with Starling.

Stephen Lane-Welsh, for use of his Android Device.

Jacob A Storm
Lead Programmer
March 23rd, 2014

2.4 Abbreviations and Terms

Common abbreviations and terms used in the document will be addressed in this section.

- Adobe AIR - Adobe's versatile platform
- Android – Google's mobile operating system
- AS3 – ActionScript 3
- Cirtus Engine – Mobile Engine combining multiple frameworks
- Feathers – GUI framework for mobile
- Flash - Adobe's versatile platform
- GUI – Graphical User Interface
- IDE – Integrated Development Environment
- iOS – Apple's mobile operating system
- Nape – Physics Engine
- SDK – Source Development Kit
- Starling – Open source mobile framework derived from Sparrow
- QA – Quality Assurance
- OS – Operating System

3.0 Technology

This section contains the information about Technology used in this product.

3.1 Target Platforms

Primary – Android, using Adobe AIR mobile

Alternative – iOS, using Adobe AIR mobile

The Android OS is a mobile platform made by Google. One of the primary draws to this platform is that it runs on over a billion tablets and phones around the world. This means that our potential consumers come from a large pool. Another draw to the Android OS is how open it is. The OS itself is open source and as long as we has access to the free Android SDK, we can freely develop for it with no cost. Because Android is such a widely used mobile OS, there is a lot of documentation and resources out there on getting our project to work with Android devices. Another plus is that a developer's certificate is free.

iOS is our other potential development platform. It is created for Apple's mobile devices. In contrast to Android, iOS is a closed platform and only worked on by Apple. This is a positive, in that the base system for our game would be very stable. The problem with the iOS is that we would need to get a developer's certificate, and unlike Android, it isn't free.

Because of the draws of Android, we decided to shift our development of *Project AstroVault* to an Android based device.

One of the requirements of this project is that we need to make a Flash based product. That is where the Adobe AIR runtime comes in. Using AIR, we will be able to build AS3 projects and have it run natively on a variety of products. For our purposes, that would be Andoid.

3.2 Target Devices

Primary – Droid RAZR M, Galaxy Nexus, Droid DNA and other Android phones

Alternative – Android Tablets

Because of the free developer's certificate for Android development, along with the fact that the Lead Programmer as access to multiple Android devices, we have chosen to develop for those Android devices over iOS counter parts. More specifically, the focus of our builds will center on Android phones and not tablets. The reasoning behind this is that more people are likely to own an Android powered phone over a tablet.

The downside to developing for devices that use such is open platform like Android, is that each hardware company that makes Android devices have there own take on what an Android device should be. This means that there are a variety of different resolutions that we will encounter in the Android ecosystem, whereas iOS devices don't have it as bad. There are a few ways we can go about handling this, we will most likely go with smart resizing of the game, with will be discussed further in Section 5.0.

3.3 Development Tools

Primary – FlashDevelop

Alternative – Flash Builder

When it comes to an environment to work in, we have a few options. We could go all out and use a text editor coupled with a compiler for AIR or we could use an IDE. Life is too short, so we are going to use an IDE. When it comes to IDEs we have two main options we can use though out our production cycle.

We could use Flash Builder, which is built by Adobe and is designed to work extremely well with Adobe's Flash products, which includes Flash and Adobe AIR. A positive about using Flash Builder is that it is cross platform and can run on multiple OSes. Another positive, is that it is well supported by Adobe and a lot of documentation and tutorials are made for Flash Builder. The downside is, that since Flash Builder is a professional product it costs money.

The alternative for Flash Builder is FlashDevelop. FlashDevelop is a free, open source IDE that supports AS2, AS3 and other languages like HAXE. The positive draw to this IDE is that it is free compared to its counter part. FlashDevelop has an active community that constantly monitors the development of their IDE and keeps well documented tutorials. The disadvantage that FlashDevelop has is that it is only for Windows, so there is no cross platform development.

We have decided to go with using FlashDevelop over Flash Builder. FlashDevelop is free, and most of the team uses Windows for development, so we have no need for the cross platform capabilities of Flash Builder.

3.4 Frameworks

Primary – Citrus Engine, with Starling, Feathers and Nape.

Alternative – FlashPunk, Vanilla AS3, StarlingPunk

3.4.1 Base Frameworks

Choosing a framework to work with is an important decision to make. Depending on what framework we chose, we could have an easy time implementing features and systems in to the game, or it could be near impossible.

Our first choice for a frame work to work with is just using none at all and sticking with using vanilla AS3 to develop our game. We could go with this, but there are better alternatives we can use.

FlashPunk is my personal favorite. It streamlines the process and does much of the back end work for us and allows us to go straight into making a game. The problem with FlashPunk, is that it is optimized for desktop applications, and isn't really cut out for mobile. Which is a shame, since the framework is really nice. We could try to run with it, but since we need the game to run well, and there is a high chance we will run into optimization problems if we stick with this as our framework.

Starling is another option to go with. It is an open source framework backed by Adobe and is derived from Sparrow. Sparrow was designed for iOS. Both are designed for making mobile applications and have built in support for multi-touch on other mobile specific features. A key feature about Starling is that its structure resembles vanilla AS3, so a person content with AS3 will easily be able to pick up Starling. Starling also has support for animations and content scaling, which will be key when working with multiple devices and resolutions. It also gives us access to Stage3D, which allows for us to render directly on the GPU, helping with the optimization issues we would run in with FlashPunk. The downside to Starling compared to FlashPunk is that there is still a fair amount of back end stuff needed to be taken care of.

StarlingPunk is an attempt of taking the base of Starling and putting the FlashPunk overlay on it. It would be the best of both worlds. However, as of the time of this writing, StarlingPunk is still fairly unstable and does not have all of the features implemented in.

Based on the scope of this game and that it needs to work on mobile, we decided that Starling would be the best choice for us as a base framework

3.4.2 Sub-Frameworks

Nape is a physics framework that helps out with simple two dimensional physics. We may or may not utilize this physic system, but it is included with the version of the Citrus Engine. If we need it it is there.

Feathers helps out with GUI elements on a mobile device. It is light weight and helps us deal with the menus and other interface controls we need, streamlining the process for us.

3.4.3 Engine

Citrus Engine is the framework that combines graphical and physics frameworks all in to one neat package. With Citrus, we have the choice of Starling with Feathers or Away3D as the graphical component. For physics we have the choice of Box2D, Nape, and Away physics. Because of our familiarity with Starling, and the simplicity of Nape, we chose the package that came with Citrus, Starling, Feathers, and Nape.

4.0 Art Pipeline

This section addresses the Artist Pipeline and how the art gets into the game.

4.1 Multiple Asset Resolutions

One of the major issues with Android development is with the variations in screen sizes. The screen types can be boils down to three different levels of screen sizes; Low Definition, Standard Definition, and High Definition. This means that we will need to find away to address the different assets for the different devices.

The way proposed to the group is to use vector based art. This way we can make large assets and scale them down to the right size without having the artists redraw each of the different sizes for the different displays. It is easier to scale an image down than it is to scale that image up when trying to maintain the quality of the image.

These assets will be exported to their own files, preferably in a .png format, and placed in folders representing their sizes. For example, the folders will be named something along the lines of *Assets1x*, *Assets2x*, and *Assets3x*.

4.2 Sprite Sheet Packer

The next step in this process is generating a file to use as an atlas, or a sprite sheet. It is better to have textures of larger dimensions and by powers of 2 than it is to have textures of smaller dimensions. That is where a utility called Sprite Sheet Packer comes in. Sprite Sheet Packer allows for us to take the multiple assets that were generated and pack them into a larger sprite sheet. Upon packing, Sprite Sheet Packer generates two files; the first being the texture file and the second is a data file that has the information on each of the sprites within the larger file. This data file is then used to help generate the Texture Atlas.

4.3 Texture Atlas

Starling helps us out here with its Texture Atlas class. The texture atlas is what helps us store and retrieve the different textures used in the game. It takes two parameters, the sprite sheet and the data file. Since Sprite Sheet Packer generates both of these files for us, it helps us out significantly. Different atlases can be made for different parts of the game. One atlas file can be made for the characters animations, one for background elements, and one for collidable objects and so on. This process would be done for each of the different asset sizes.

4.4 Asset Manager

This is the most important part of the pipeline because this is how the art actually gets used in the game. Starling helps out again in this portion with its built in atlas. The atlas is a great way to manage files, sounds, textures, and atlases. They way this would work out is that Starling will determine which Assets size to use for the game and load in the correct files (Refer to Section 6.0 for more information). By utilizing the asset manager and keeping a set naming scheme for the assets, the artists will only need to update the data files for the sprite sheets to see the art in the game.

5.0 Design Pipeline

This section addresses the Design Pipeline

5.1 Deep in the Code

The Design Pipeline is a tricky subject. Right now we don't really have anything set up for a level loading process or anything like that. However, even with out a level editor or another type of editor for the game, the designers are still heavily involved in the process, even without tool sets. Since this project technically only has one programmer, the designers have chipped in to help programming. With the designers working closely with the programmer and with the code, the will have a lot of first hand influence with how the game turns out.

6.0 Systems

This section address the multiple systems found in the game

6.1 Controls

6.1.1 Touch Events

Half of the game involves tapping on the screen, either to navigate the menus, or to build up momentum for the player. Starling helps out once again, with its Touch Events. This allows for us to make an event occur each time the player touches the screen.

The Touch Events are handled with TouchPhases. By utilizing the BEGAN, MOVED, STATIONARY, and ENDED states of a TouchPhase, we can gather the input data we need in order to make an in game event occur. By setting Starling.multitouchEnabled to true, we can incorporate multitouch elements into the game with ease. Gestures and swiping motions are done by tracking the path of the finger from TouchPhase.BEGAN to TouchPhase.ENDED.

6.1.2 Accelerometer

The space portion of the game utilizes the accelerometer on the device. The accelerometer is accessed through the Flash API by importing the sensors and special events associated with the accelerometer. Activating the accelerometer is as simple as creating a new accelerometer and attaching an accelerometer event to it. With the accelerometer in place we can access information from three axes; X, Y and Z. Using this data, we can incorporate it into the games logic.

6.2 Device Scaling

6.2.1 Stretching

In dealing with multiple Android devices, we need a way to scale the stage to fit each of the devices. The first, most simple way to do it is just by stretching the stage to fit whatever device it is on. This is a horrible thing to do, if we want the game to look nice. Stretching distorts the images on the screen and makes the game look terrible.

6.2.2 Letterboxing

The second way we can go about scaling, is by letterboxing. What is done here, is centering the image in the middle of the screen. This way the image will stay crisp and not distorted. Another step we can take in this section is that we can scale up the image by zooming in to the largest size we can get with out cropping. By doing this, we can keep the aspect ratio the same on all devices. This is better than stretching, but there is a better way to do all of this.

6.2.1 Smart Scaling

The final way, and the way we decide to proceed involves the idea of support different stage sizes. The idea is to have a base stage size that you use to align all of the objects in the game. Usually this is split into three general sizes; LD, SD, and HD. To determine which asset type we use, we need to determine the scale factor for the device. This is determined by finding the factor of the screen size divided by the stage size. Using this scale factor, we then choose the correct asset resolutions. From here on out we use percentages to determine the assets location on the screen.

6.3 Animation

Animations in the game would be handled by the Citrus engine, using its Animation Sequence class. This class allows for us to store all animations related to an object using Texture atlases. The Citrus animations make use of Starling's Juggler class which controls all animations and allows for some nice tweening effects.

6.4 Menus

This is how we navigate through the game. By utilizing the Feathers framework, the user interface will be easy to incorporate into the game and shouldn't prove difficult.

6.5 Particles

Particles systems add polish to a game. The Starling framework gives access to a powerful, but simple particle system. The particle emitter takes a special file that hold all the information for a particle effect. We load in that file, along with the texture we want to use for the particle and viola, the effect is now in the game. We just need to mess with the emitter location in the code, but everything else can be done outside of the code base.

7.0 Sprints

This section contains details about each sprint cycle

7.1 Sprint 1 – 3/17/14

During the first sprint cycle we had an entirely different idea for the game Sleepy Cat Adventure Game. It was going to be, what we called, a procedurally generated nap labyrinth. We decided on using Android devices as our platform. Research was done for using FlashPunk as the framework behind our game. A build was created and run on a RAZR M to stress test FlashPunk's mobile abilities. We were able to generate roughly 50 animated entities and keep a steady 30 frames a second.

7.2 Sprint 2 – 3/24/14

The idea of the game was switch from Sleepy Cat Adventure Game to what is now known as *Project AstroVault*. This meant that the Technical Plan had to be rewritten to serve the new game. Much of it is still a work in progress. After testing the FlashPunk builds, I decided that flashpunk wasn't going to cut it since it didn't have built in support for Accelerometer and multi-touch. Much of this week was spent researching Starling and getting a build on the phone utilizing starling and the accelerometer.

7.3 Sprint 3 – 3/31/14

After reviewing the new direction for the game, Ryan and I decided to utilize the Citrus Game Engine and make use of its camera system and sprite system. This week has been spent researching the Citrus engine and seeing how it incorporates the Starling framework. Since the Citrus Game Engine houses Starling, the switch was not difficult. The setbacks this week were as follows:

- My phone, the one we used for testing purposes broke.
- Teaching a designer how to use SVN.
- Figuring out how to properly work Texture Atlases

7.4 Sprint 4 – 4/7/14

This sprint cycle we kicked into full gear. We needed to play catch up. Ryan started building the core gameplay and implementing the phase switching in the game. We have developed a deeper understanding of the citrus engine as well as starling. Parallax and text fields have been implemented. Art assets are now in the game and can be scaled on the fly. Touch controls have been implemented and can easily be expanded upon. The game currently runs on jellybean 4.2.

7.5 Sprint 5 – 4/14/14

This sprint was spent trying to figure out how to increase our loading times on mobile. We researched ways to up performance, stagger loading, and tried a preloader. Eventually we were able to get the loading to work better and were able to get almost straight into the game.

7.6 Final Sprint – 4/23/14

Everything must get in. Asteroids now spawn in space. There is a splash screen along with a menu. Asteroids affect your speed in space and slow you down enough to end the level sooner if you get hit. There is a user interface which helps people understand what to do when playing the game. Sound effects are being implemented.

8.0 References

This section contains a list of references used for this documentation

- http://www.android.com/meet-android/ - Android Info
- http://wiki.starling-framework.org/ - Starling reference wiki
- http://www.adobe.com/products/air.html – Information about AIR
- http://www.adobe.com/products/flashplayer.html – Information about Flash
- http://www.adobe.com/products/flash-builder.html – Information about Flash Builder
- http://useflashpunk.net/ - Information about FlashPunk
- http://citrusengine.com/ - Information about Citrus Game Engine