INTERACTIVE AUGMENTED REALITY GAME APPLICATION

ESLAM MOHAMMED AWAD SHARIF

Session 2016/2017

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
SEPTEMBER 2016

INTERACTIVE AUGMENTED REALITY GAME APPLICATION

BY

ESLAM MOHAMMED AWAD SHARIF

SESSION 2016/2017

THE PROJECT REPORT IS
PREPARED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERSITY
IN PARTIAL FULFILLMENT
FOR

BACHELOR OF COMPUTER SCIENCE
(Specialization in Game Development)

FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY

NOVEMBER 2016

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Rights:

- The ideas in this report belong to the writer only, however, the supervisor and moderator of MMU may alter it.
- The implementations of the project is for your reference only.
- copyrights reserved for Multimedia University and its copyrights laws.
- I claim ownership of this report's content, figures, tables and diagrams EXCEPT for the cited ones .

You May **NOT:**

- In anyway reuse, have a similar, or rewrite contents of this report.
- Imply or protest in any way that the game or the idea is yours after 15/09/2016.

ESLAM MOHAMMED AWAD SHARIF
Faculty of Computing & Informatics
Multimedia University
Date: 06/02/2016

Acknowledgment

I would love to thank Dr.Junaidi for his patience with my development methods that require very little monthly updates and Dr.Wong for opening my eyes to whole new possibilities in the development of Augmented Reality application, “Use the technology to support your idea, not the opposite” is what he said in my application proposal presentation, both of your guidance through this project is highly appreciated.

Also, I’m glad to mention the enhancements this application reflected on my programming, 3D modelling and visual designing skills.

Lastly, thank Allah for giving me the strength to go through making this application and for giving me such supportive family and friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CLASS DIAGRAMS

LIST OF SEQUENCE DIAGRAMS

Abstract

In this project, development of an Augmented Reality application was requested, analysis on recent AR libraries will be performed alongside the process of which game engine will the development be on.

The focus is on producing an interactive game for mobile phones using Unity3D game engine and Vuforia’s augmented reality development kit will be the goal.

CHAPTER1

INTRODUCTION

1.1 BACKGROUND

Augmented Reality is the integration of digital information such as computer generated data such as graphics and GPS data with the user's environment in real time. Augmented Reality is different from Virtual Reality, which creates an artificial simulated environment. Augmented Reality applies digital information on top of the existing real life environment.

The term, Augmented Reality was invented by Boeing researcher from Boeing Computer Services in Seattle, Professor Thomas Caudell in 1990, Currently, AR application such as Google Glass and mobile games such as Pokemon Go are perhaps the most well-known consumer AR products. Augmented Reality (AR) has been associated with movies set, campaigns in a distant future, or situations that just seem gimmicky for the most part.

Making a game was never an easy task because the ultimate goal is to amuse people, but, building a video game with all graphics and controllers has raised the bar a little higher.

The creation of a game that interacts with player's environment takes our imagination a little further towards the future of entertaining, of course it comes with development, production and risks cost, but emerging our reality with a different (fiction) one is what we dream of, and the answer lies in Augmented Reality applications.

1.1 OBJECTIVE

Building an AR game that concludes my own game design, concept and development.

1.2 SCOPE

Draw a production pipeline to make the application development cycle versatile and produce a unique yet fun to play game.

CHAPTER 2

BACKGROUND STUDY

2.1 LITERATURE REVIEW

At the moment it's very comprehensive to grasp all the bits and pieces to develop an application in that framework, nevertheless, some of the open source libraries have made this computer vision and imagery process easier for application development.

The Augmented Reality libraries i have done comparison on have all been tested to find the most suitable one for the future development of the application, but, before the comparison, a few terms must be clear.

- **What is open source library?**

  Is computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to any one and for any purpose.

- **What are markers?**

  AR Markers are a type of 2 dimensional barcode, most probably in a perfect black and white square.

- **What is NFT?**

  Short for Natural Feature Tracking where the markers are not in black & white only and the camera needs to be trained to track these images so, it is used for images that are not easy to built as bordered markers.

- **What is camera training?**

It’s the process of training the camera on several images or just one image with specified details to improve tracking results.

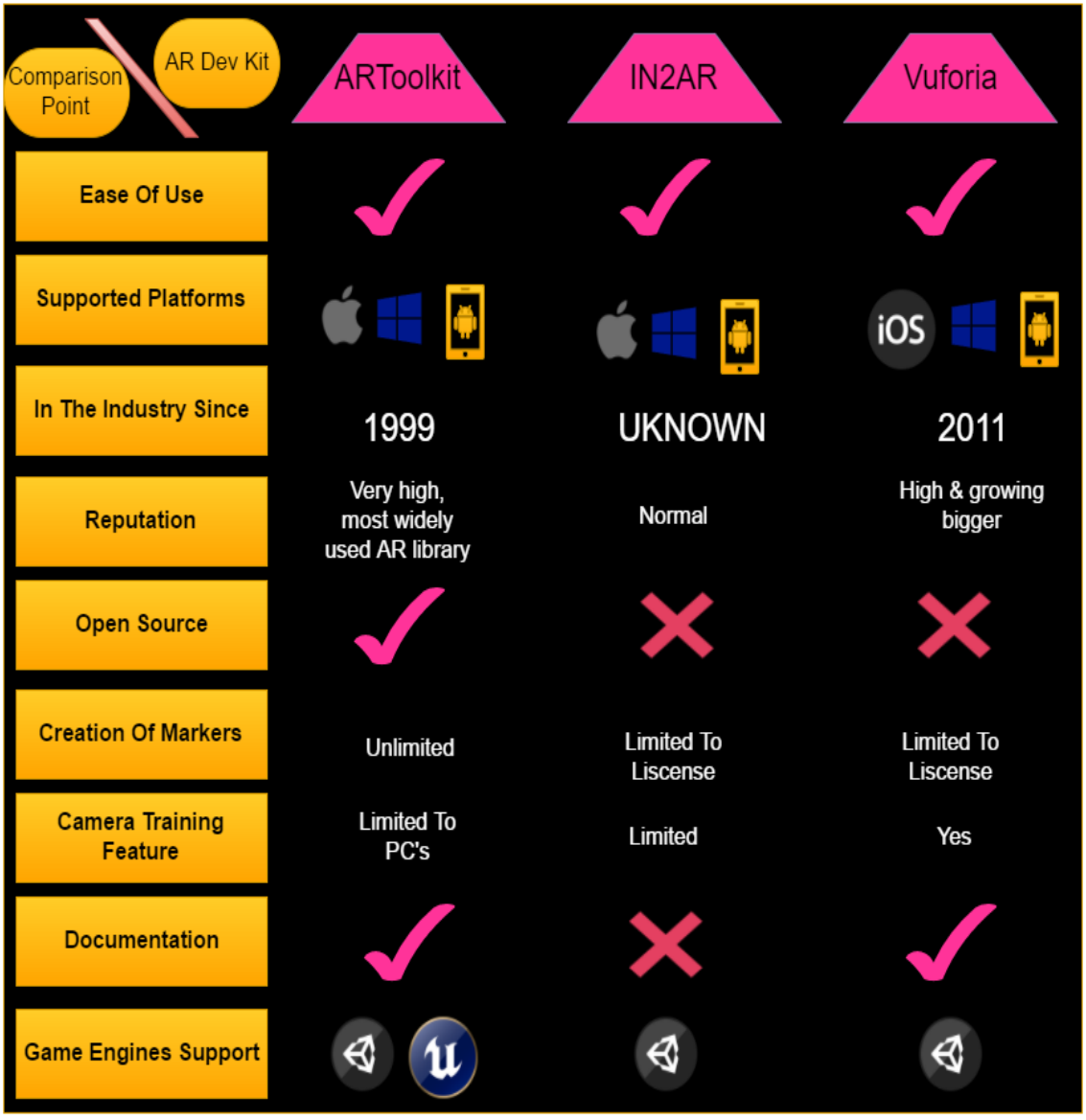


Figure 2.1 (AR SDK Comparison Table)

2.2 VUFORIA IT IS!

As shown in figure above, the wisest choice to start developing an AR application is to pick-up AR Toolkit but, overtime problems started to occur using the library with mobile phones as they provide limited access to some of the features such as camera training which is needed for a better user experience during the game.

So, as I am testing the other two libraries after I hit a wall with AR Toolkit I found that IN2AR library is good for development only if it wasn't commented on in Japanese !!, not to mention it was completely written in Flash and C# for Unity3D.

In the end, the testing leads me to choose Vuforia not because it's the last choice but because it's a very stable and feature-full AR SDK, even though the generation and storing of the markers is done online and the free license provides up to 100 markers maximum, it is great choice for the moment.

2.3 GAME ENGINES WITH VUFORIA

After choosing Vuforia as main AR SDK for the application it'll be wise to start developing the game with the library supported game engine which is Unity3D.

You can create any 2D or 3D game in unity, creating highly-optimized and beautiful game applications has never been easier. From idea to deployment with a click of a button (figuratively).

Figure 2.2 (Unity3D Logo)

Unity uses C# & Javascript as its main programming language but, in process of building this project I'll be using C# as the programming language and the targeted platform will be mobile devices.

CHAPTER 3

REQUIREMENTS

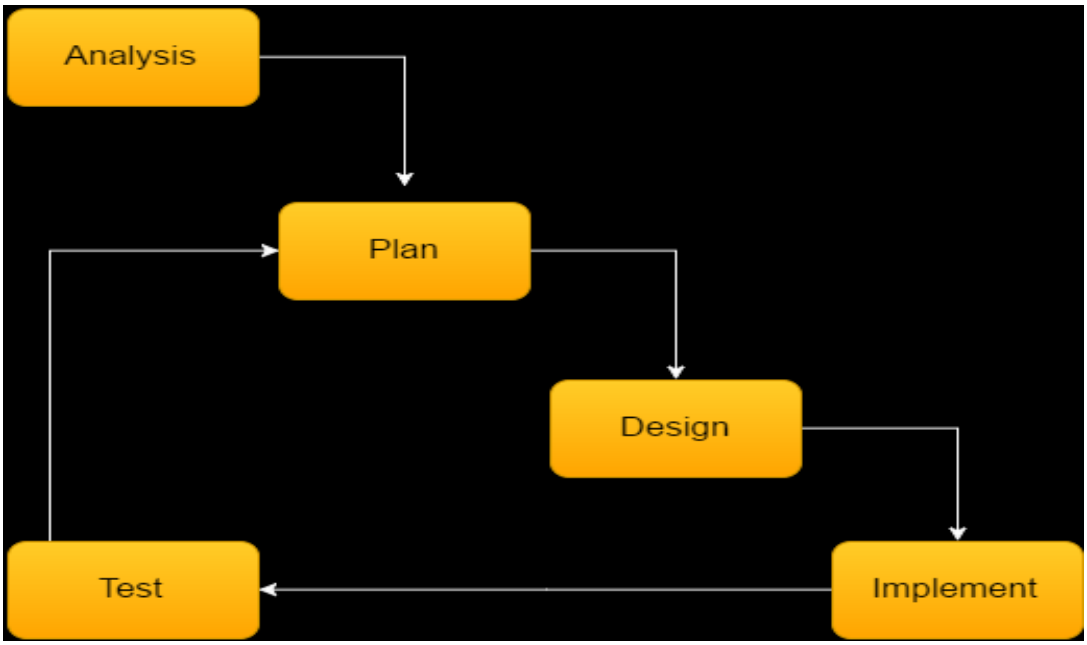


Figure 3.1 (System Requirement)

3.1 ANALYSIS

Using a machine that is fully capable of rendering high-end graphics and work without lagging or possible loss of data is important, not to mention it has to be able to work side to side with the Vuforia SDK.

Recommended development machine specs:

- Windows .
- Intel Core i5 2.4GHz.
- nVidia Graphics Card.

3.2 PLAN

To ensure users of the application get the most out of it, a minimum specs scope have to be set.

Targeted device minimum specs

- Android.
- Kitkat 4.6 (API 19).
- Quad Core chip-set processor.
- Rear Camera 10MP

CHAPTER 4

ORIGINAL GAME DESIGN

4.1 Concept

A memory game based on cards where the exciting part relies on how fast the player connects English letters with cards that represent those letters (i.e. a picture of car is the letter 'A'). And in the same sense, the player may be given a set of cards that represent a word and voice recognition will be used to confirm player's answer. Name of the game is still discussable but Brain Grinder suits the concept for now

Figure 4.1 (Game Cards Sample)

So, for example, if the player was asked to form the word “car”, he/she will have to put the cards in order from left to right to achieve that goal. And on the other hand, the game may show the cards that form word “car” and ask the player to pronounce it, so the gameplay can go either way.

4.2 MODES

- ***Solo*** (collect 100 stars to finish a difficulty level)
  - ❖ Frog (Easy) - Player will be asked to form/pronounce words of 3 - 6 letters.
  - ❖ Ant (Medium) - Player will be asked to form/pronounce words of 6 letters minimum.
  - ❖ Pigeon (Hard) - Player will be asked to form/pronounce sentences of 2 - 4 words.
  - ❖ Human (Intelligent) - Player will be asked to form/pronounce sentences of 4 words minimum.
- ***Head To Head***
  - Form Cards
    - # Input - Maximum number of words Min (3).
    - # Input - Maximum number of sentences (Min 0).
    - # Input - Timer on/off.
    - # Input - Difficulty (Frog - Human)
  - Pronounce Words
    - # Input - Maximum number of words Min (3).
    - # Input - Maximum number of sentences (Min 0).
    - # Input - Timer on/off.
    - # Input - Difficulty (Frog - Human).
  - Feeling Intelligent

Randomized set of challenges.

- ***Group VS Group*** (May or may not be implemented depending on time)
    - Choose number of team members (Min 2).
    - Pick a number up (If Min then 1 or 2).
    - Randomized gameplay.
    - Every player will be given parts of the words/sentences to form/pronounce.

4.3 RULES & ELEMENTS

- ***Rules***
    - For each correct answer the player wins 1 star, maximum of 3 stars can be won depending on how fast the player forms/pronounces the letter.
    - Every 9 stars the player gets a "Hint" helper.
    - If "Hint" is available, then it can be used by tapping on the card.
    - Player has to organize the cards from left to right to represent the given word.
    - Player has to pronounce the word represented by the cards correctly.
    - A "Timer" will be used in Head To Head or Group gameplay.
- ***ELEMENTS***
    - Physical Cards.
    - English words.
    - Voice recognition.

- Leader board.
- Timer.

4.4 IMPLEMENTATION PLAN

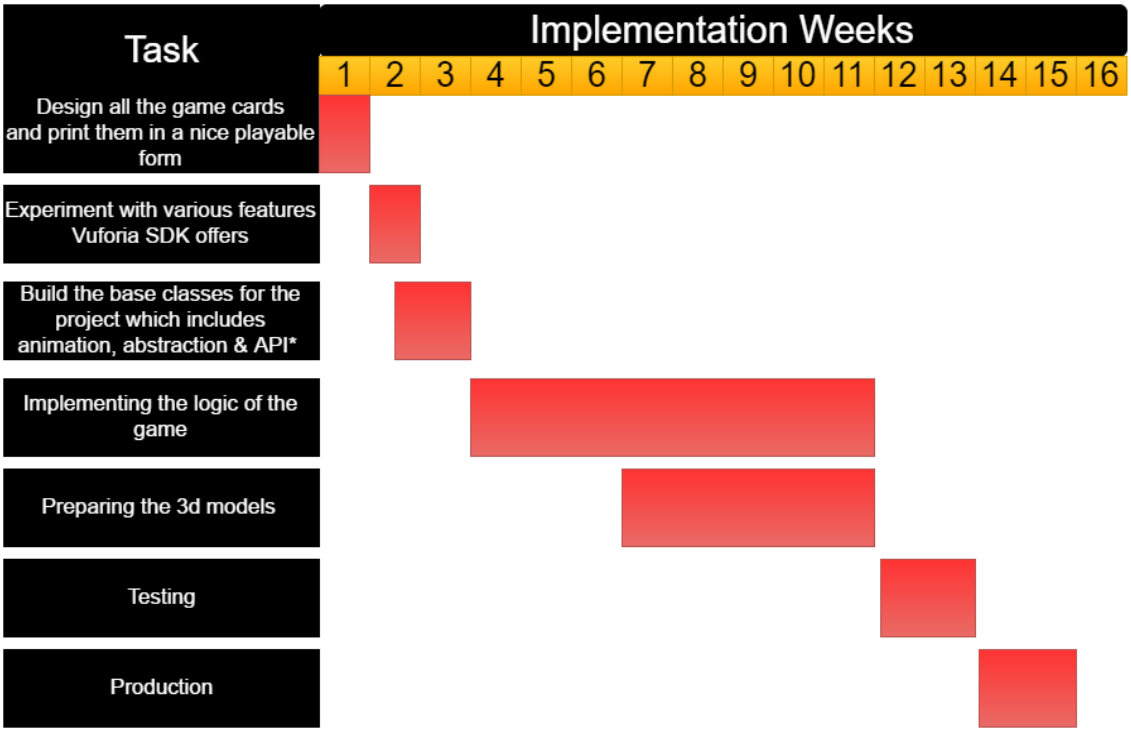


Figure 4.2 (Implementation Plan)

CHAPTER 5

3D MODELS & VISUAL DESIGNS

5.1 TOOLS

5.1.1 VECTOR EDITING

Vectr - The graphical editor helped with building the logo and gaming cards in the way I want and find easier.

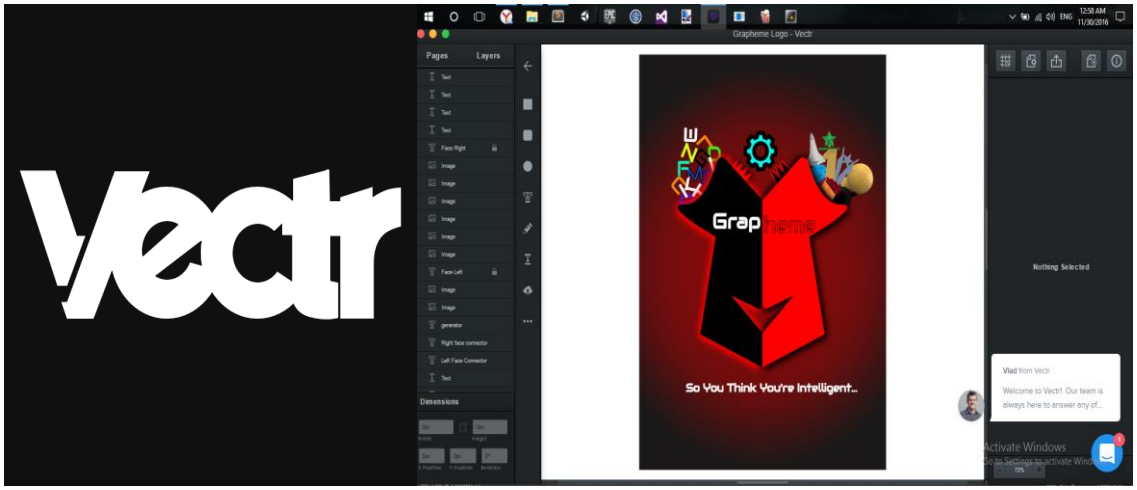


Figure 5.1 Vectr

Paint.Net - This one is more of an image manipulator than a vector editor.

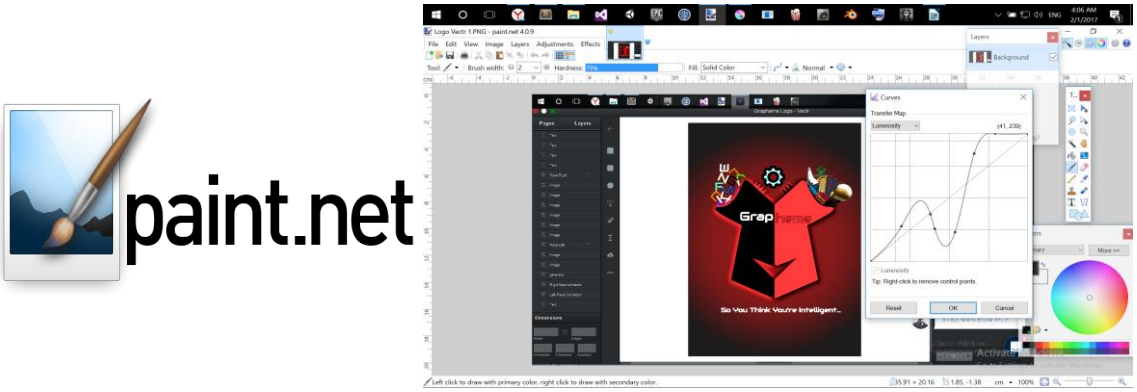


Figure 5.2 Paint.Net

Krita - A magnificent tool for scalable vector graphics editing.

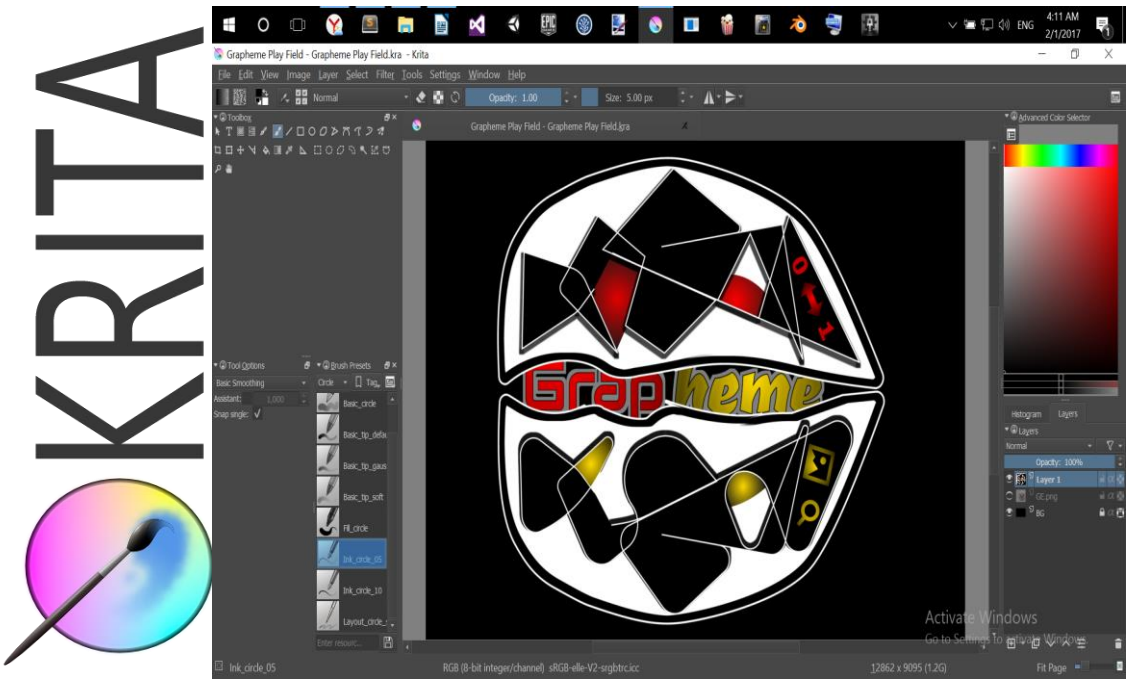


Figure 5.3 Krita

5.1.2 3D MODELING

Microsoft's 3D Builder - Has been used partially for creating simple models.

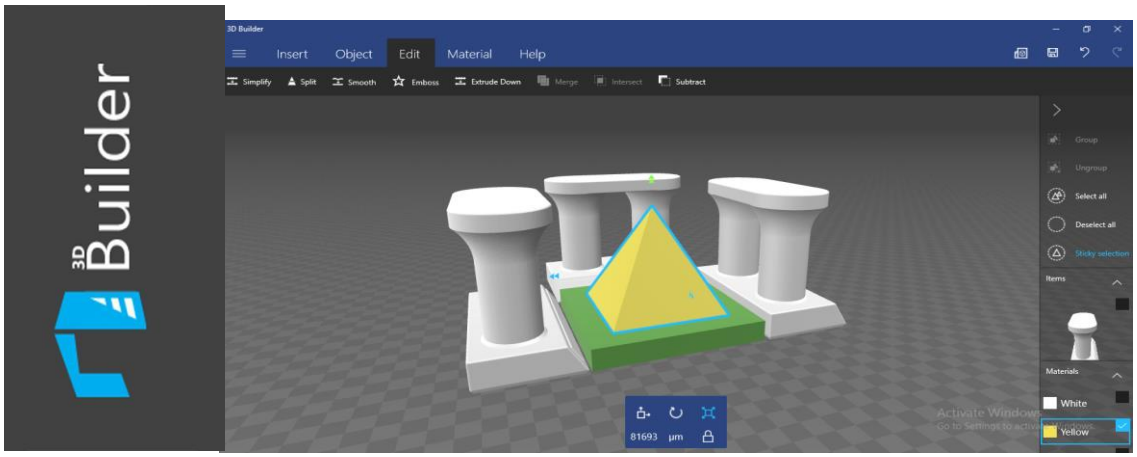


Figure 5.4 Microsoft 3DB

Blender – The popular 3D graphics creator & animator.

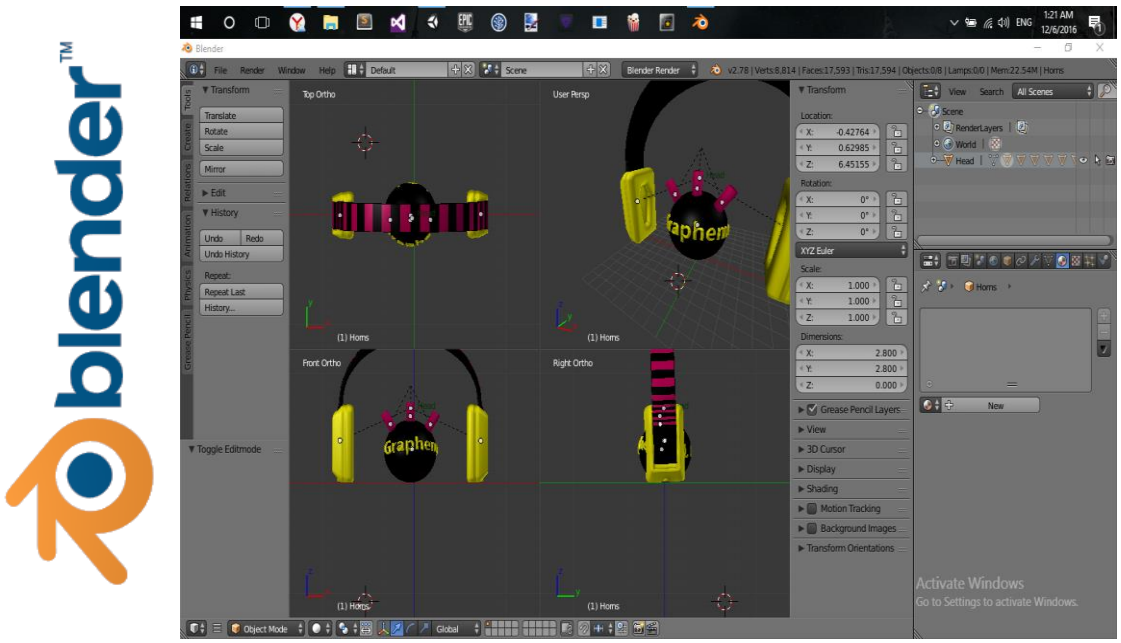


Figure 5.5 Blender

5.2 DEVELOPMENT PIPELINE

Creating the vectors or models without taking platform limitation under consideration is just a vague creation of them thus development pipeline.

Why the use of more than one SVG editor? a one might ask...to answer that i would say, first, Vectr is not a stable SVG editor as it is new in the field and it has a lot of bugs (i.e stack overflow & unsaved work exceptions) so i can't just depend on one svg editor if it's risky. Second, flexibility, using Krita when Vectr is down and vice versa kind of lowers the risks to a great minimum.

Third, Krita & Vectr offer distinguished tools and features make them great to use together. Last, some of the images exported from Krita or Vectr are distorted, and this is where Paint.net comes to work, with its quick but stable image editing tools (i.e. crop, cut, mix & resize).

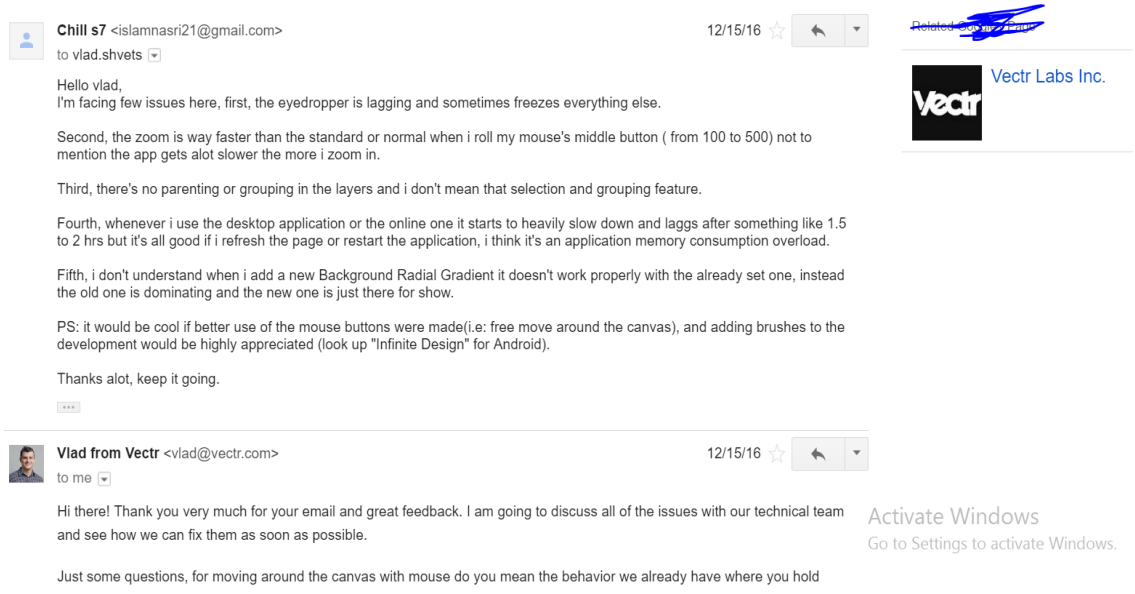


Figure 5.6 Vectr Bugs Report

As for the 3D Models, Microsoft 3D Builder is great in creating quick 3D objects with the ability of making them low Polygonal but, it doesn't help creating materials for the individual objects (at the time I'm writing this report) exported nor exporting them as '.fbx' to be well suited for importation in Unity3D, so they have to be modified with another software, which opens the door for Blender to come in handy.

Exporting models from 3DB to Blender as '.obj' and modifying their pivot, angle & materials just to be exported as '.fbx' makes it a great yet stable pipeline to depend on,

not to mention the ability to animate those models within blender as we'll see in the game.

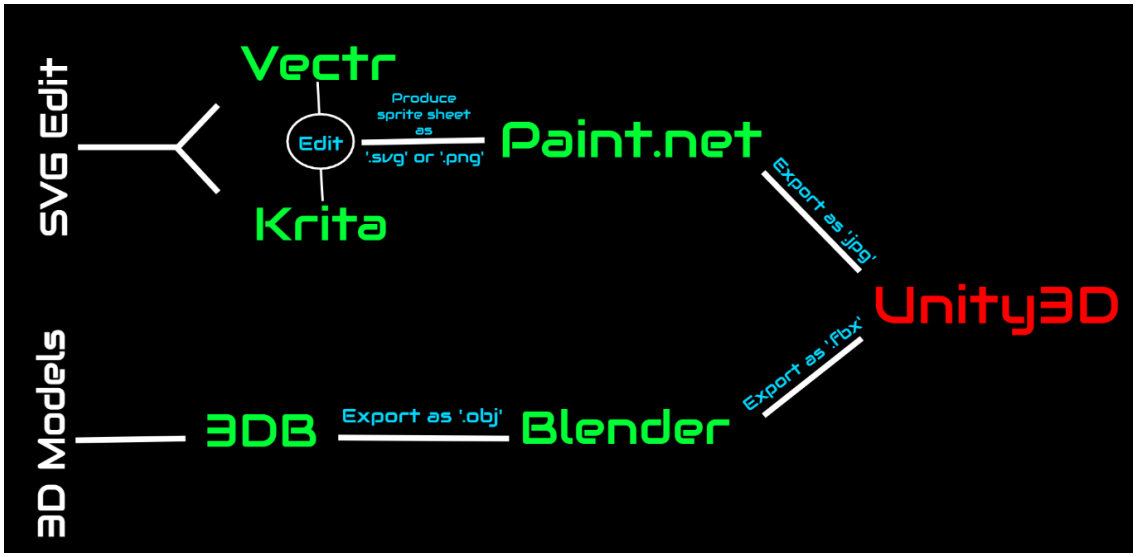


Figure 5.7 Visuals Development Pipeline

5.3 DIGITAL ASSETS

An overview of the finalized game artworks and sprites made using these awesome soft wares will be displayed in the next figures.

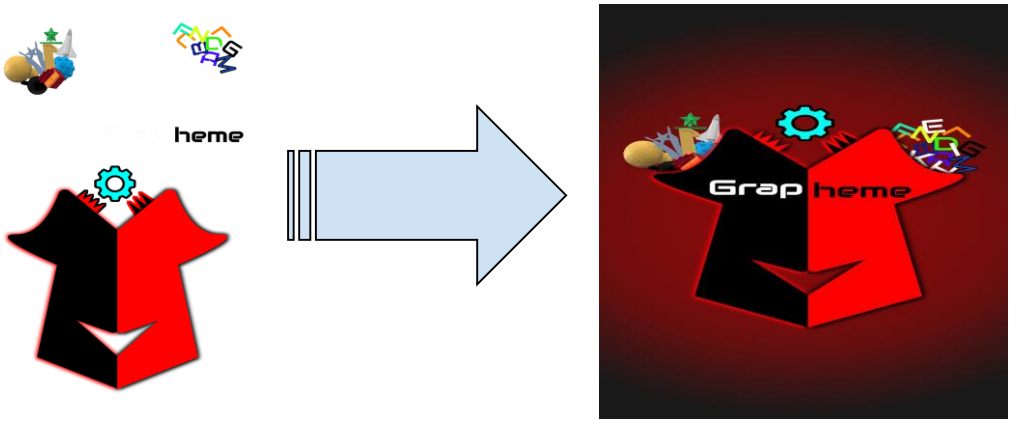


Figure 5.8 Game Logo

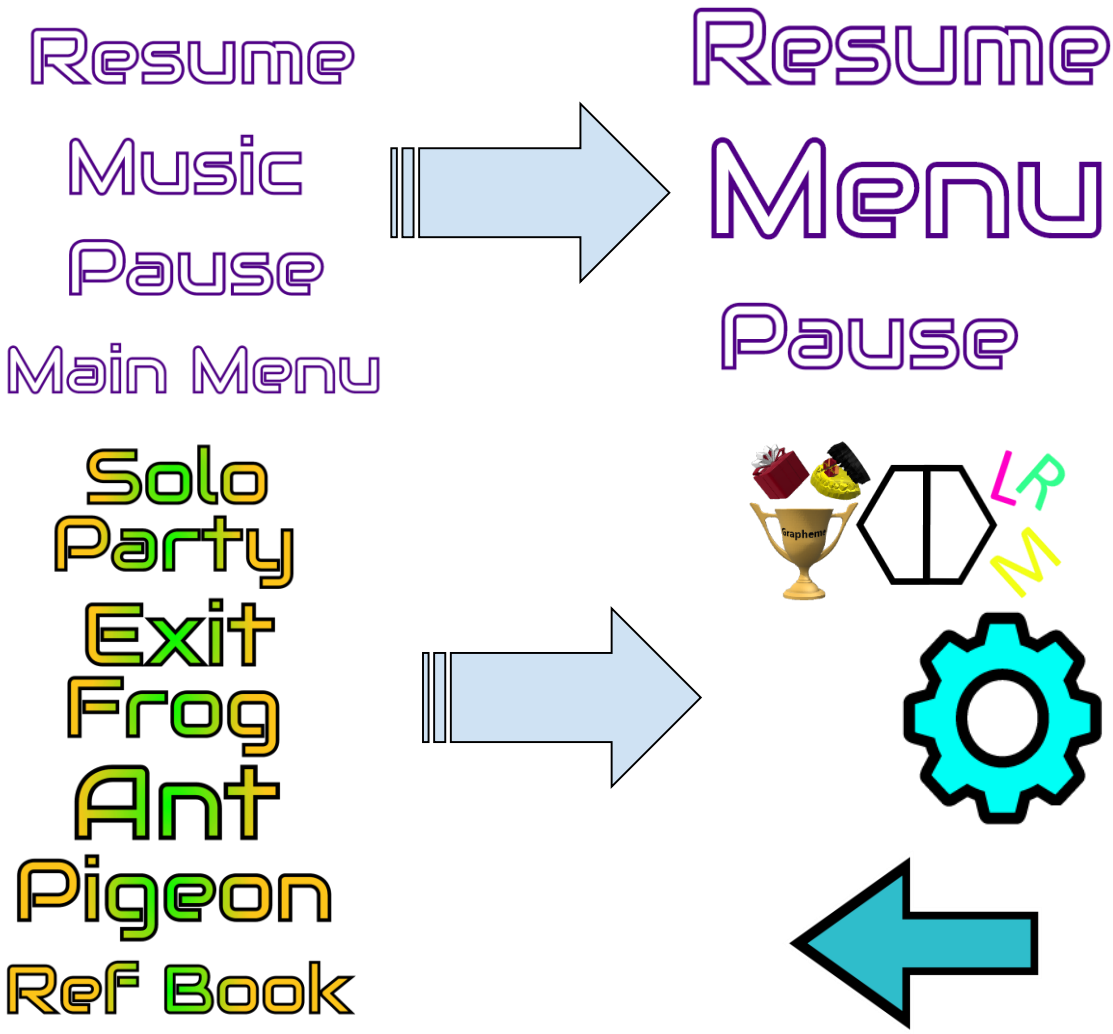


Figure 5.9 Menu Navigators

Creating 26 3d models with the development pipeline mentioned above is a tedious job where the side effects show on some of the models being a bit off or weird in their colors or looks but it still gets the job done, and at this point there's no time remodel or animate them.

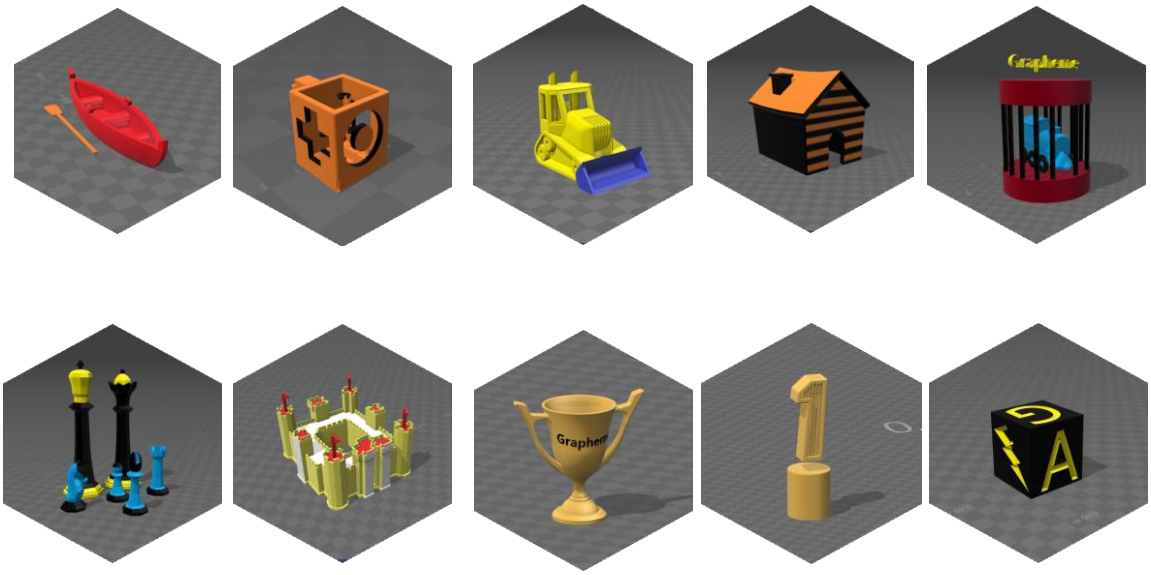


Figure 5.10 3D Models Sample

5.4 PHYSICAL ASSETS

Some of the artwork development depends on the Augmented Reality concept of how detailed an image is to be used as a tracking object, with that in mind, several images have to be produced and tested with Vuforia's online tool to ensure what's close to a perfect tracking image.

User experience is one of the important aspects in building this game so, I can't afford to have unexpected or arbitrary jitter in the object movement while the player is controlling them, we will see how the original designs have improved under this concept

but first, we'll go over few points mentioned in Vufora's Library on how to make the proper image target.

Attribute	Example
Rich in detail	Street scene, group of people, collages and mixtures of items, and sport scenes
Good contrast	Bright and dark regions, and well-lit
No repetitive patterns	A grassy field, the façade of modern house with identical windows, and a checkerboard
Format	Must be 8- or 24-bit PNG and JPG formats; less than 2 MB in size; JPGs must be RGB or greyscale (no CMYK)

Figure 5.11 Vuforia Attributes Table

- **How To Size the Image Used for an Image Target ?**

"When the image is being uploaded to the Target Manager, we recommend scaling the image to be uploaded to 320 pixels or larger width. During the resizing process, anti-aliasing is performed on the uploaded image. This act ion is perfectly acceptable for photos; however this action may not always be desirable. To avoid experiencing the anti-aliasing impact, you can ensure that the uploaded image is at least 320 pixels in width. Stretching and softening in the image due to a server-side scaling step results in a lower feature count and worse local contrast in the image. This may not be visible immediately, but such targets can result in poor target detection and tracking."

- **Feature Ditribution**

"The more balanced the distribution of the features in the image, the better the image can be detected and tracked. Verify that the yellow crosses are well-distributed across the entire image. Consider cropping the image to remove any areas without features."

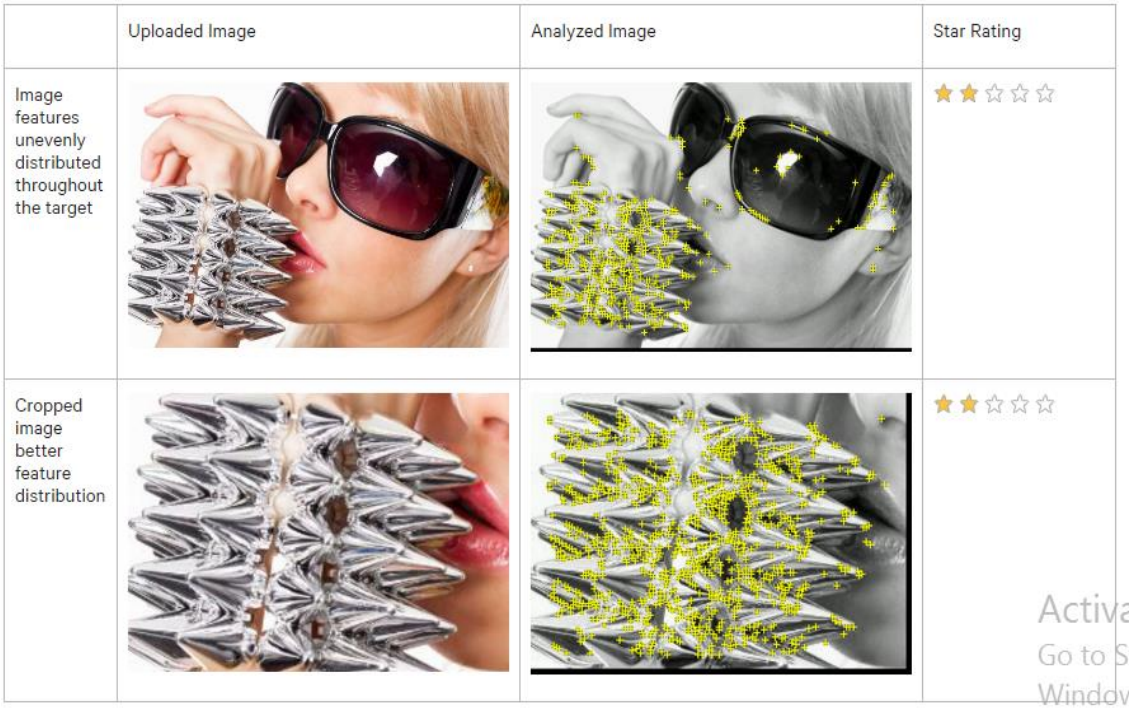
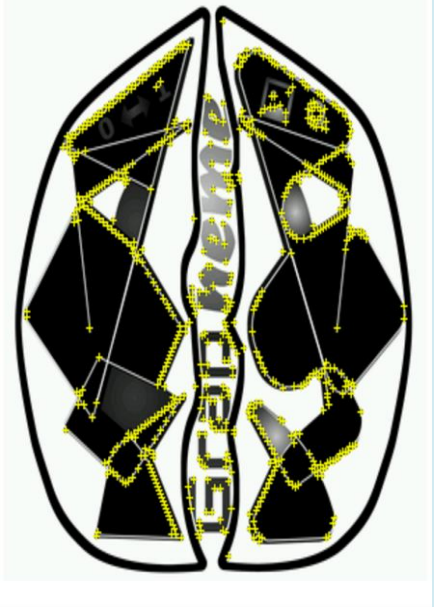
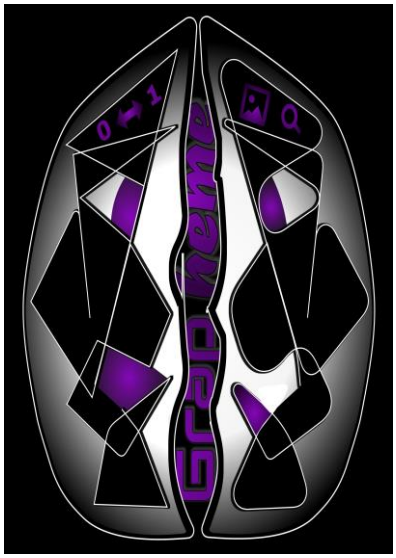
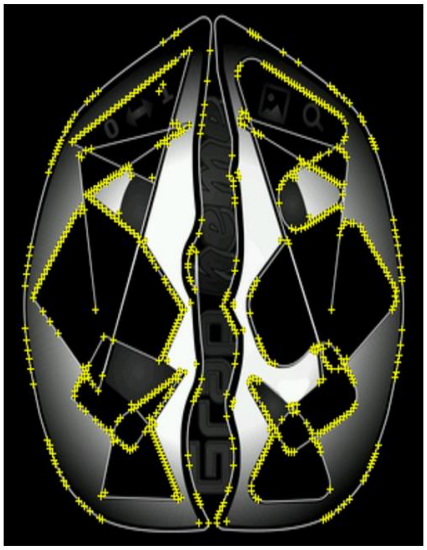


Figure 5.12 Image Feature Rating

Now, after having a close to full understanding of what an image features is, here's an overview of my design improvement over testing image features.

Uploaded Image	Analyzed Image	Star Rating
		5
		5

PlayField Grab Game	PlayField Grab Game	5
PlayField Grab Game	PlayField Grab Game	5

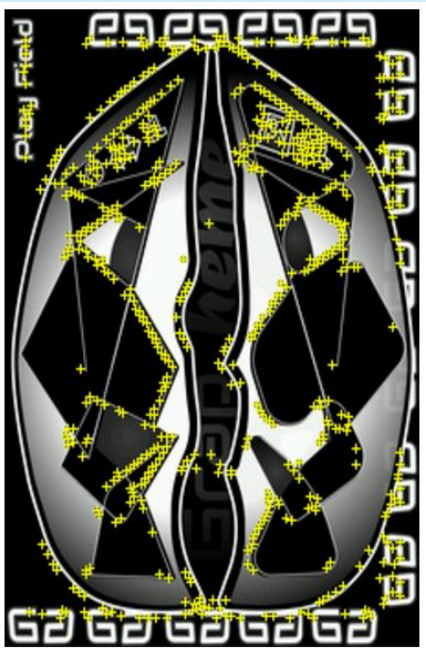
		5

Table 1 Analyzed Image Sets

As you can see, all the image targets created are feature-full after looking at their respective star rating but, that doesn't mean they're well suited for all cases & scenarios. So, after testing them out, the last design seems to rest the 3d objects on the image with less movement jitter than the first one which makes it a perfect final choice.

CHAPTER 6

SOUNDTRACK

6.1 TOOLS

I stumbled upon many free tools to create sounds and judging my little but subtle experience I choose Magix.

 # Magix Music Maker Jam (Android) - An easy tool to create music, with it's pre-looped sound samples and creative interface, making soundtracks hasn't been more fun.

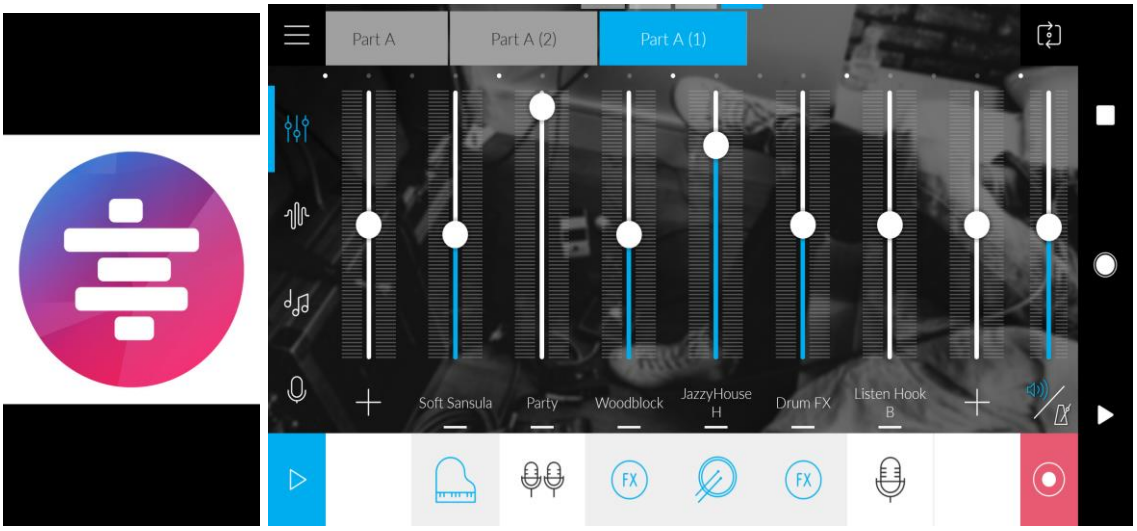


Figure 6.1 Music Maker Jam

6.2 THEME

Making music or soundtracks has always been a hobby of mine, I've been practicing over the years but I never got the chance to publish any of my work so, the theme for this game

was originally chosen to be from the days of pixelated games like mega man x & metal gear slug which we will listen to clearly in the two of the game's soundtracks which are

- Pixelated Love
- Chilling Bells

And for the 3 remaining soundtracks I just found my self losing track of the theme so I ended up making main menu tracks (or just showing off skills) in

- Dopeamine
- Feel The Harmoney
- Void Serenity

CHAPTER 7

EXPERIMENTATION PHASE

7.1 TECHNOLOGY

Due to the use of an online marker pattern maker platform, sometimes the system

can be down which will slow the productivity a bit.

Figure 7.1 Vuforia System Maintenance

Debugging on the other hand, can be a tire-full job sometimes because Unity Remote (the app provided by unity for real-time mobile application testing) doesn't connect phones camera while testing so, to know if the artwork and 3d models are properly placed over the image target printed, I have to test them on the laptop first (with web cam) then hope results are somewhat similar when game get installed on the phone.

7.2 GRAPHICAL MODELS

After building the first 10 models there was a unit measurement problem that lies within the conversion from Microsoft 3D Builder to Unity3D units and the pivot of the sculptures imported from Microsoft 3D Builder was offset by a couple of hundred meters according to Unity3D measurement, not to mention that the ".obj" files don't get imported correctly in every time as shown in the Figure 7.2.

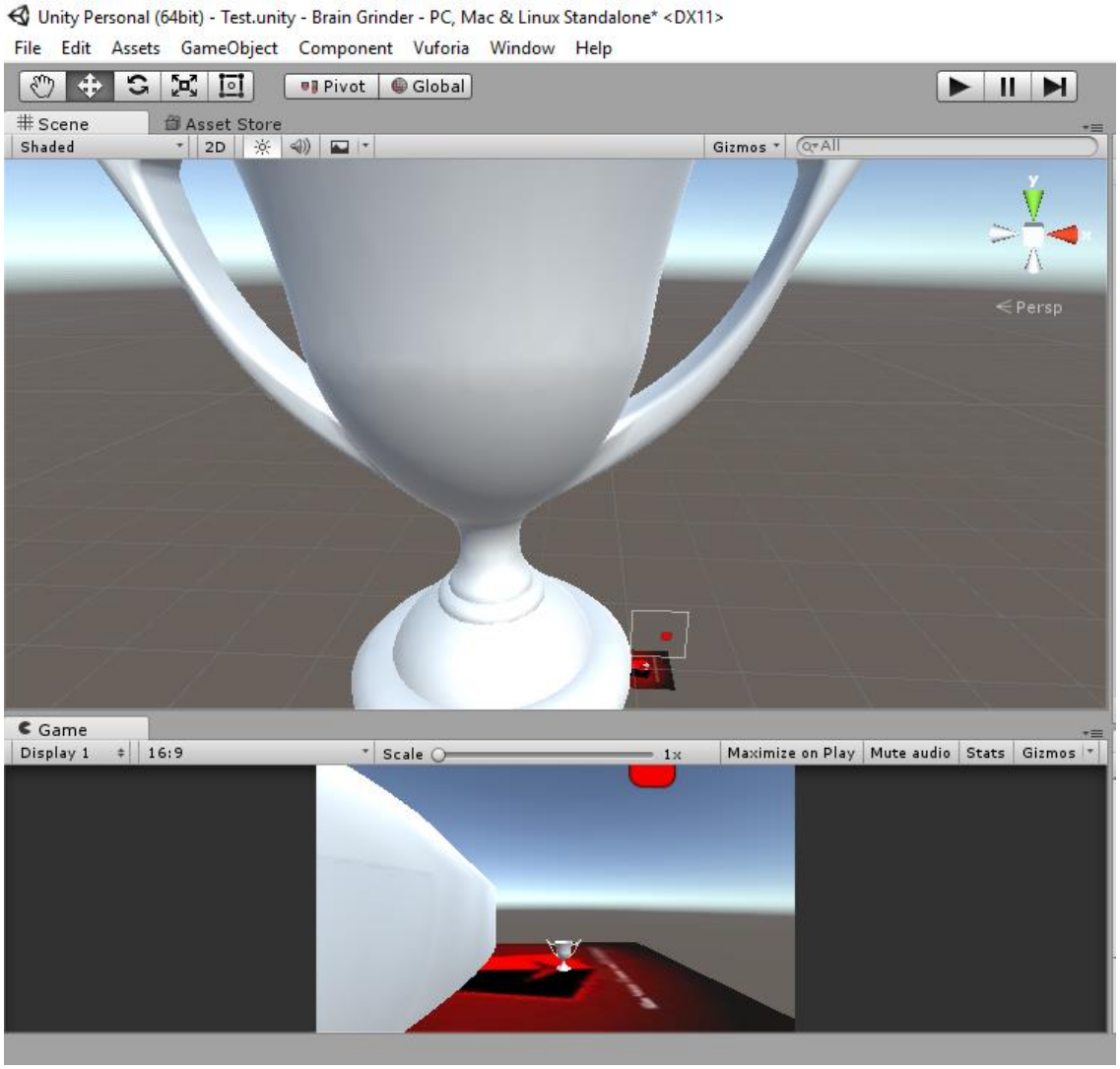


Figure 7.2 3D Models Problem

The huge model is being directly imported from 3DB with no materials so, the use of a second 3D software will be wise therefore Blender has been chosen to fix the materials, origin, and scaling of the objects created with 3D Builder and export them as ".fbx" to be more compatible with Unity3D.

Initially, all models were supposed to be animated but, to make 26 3D models animations (let's say with an "Idle" animation only) is a team work compare to a one guy job, dropping the animations part to a minimum of 1 model will do the expected for now.

7.3 GAMEPLAY & CONCEPT

In the beginning and after a bit of thought, the logo of the game can be used as gaming cards so the player could just move the card to the desired position to rearrange the shapes according to their respective letter, preparing the logo to be printed as small playing cards is shown in Figure 7.3,

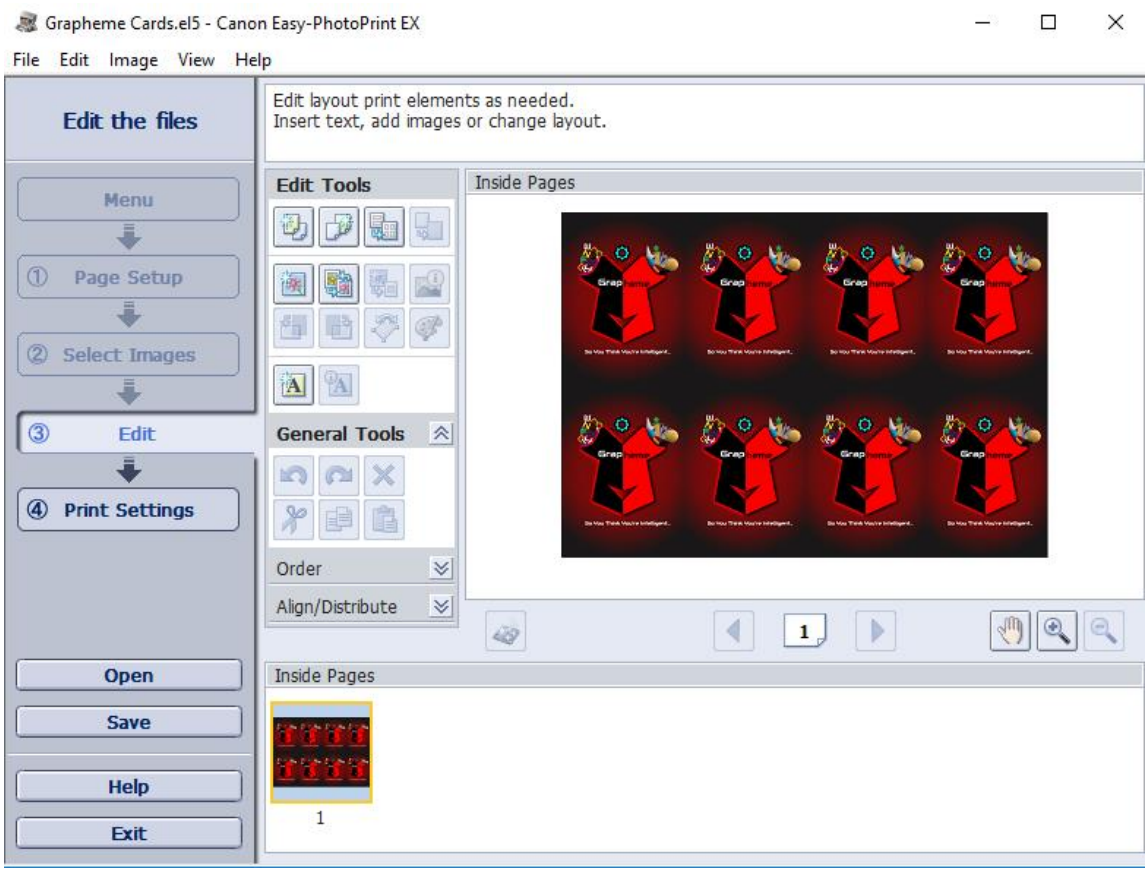


Figure 7.3 Gaming Card

but Vuforia library doesn't support that, allowing only one image target to one registered dataset of an individual image, in other words, one image can only have one object drawn on top of it as shown in Figure 7.4.

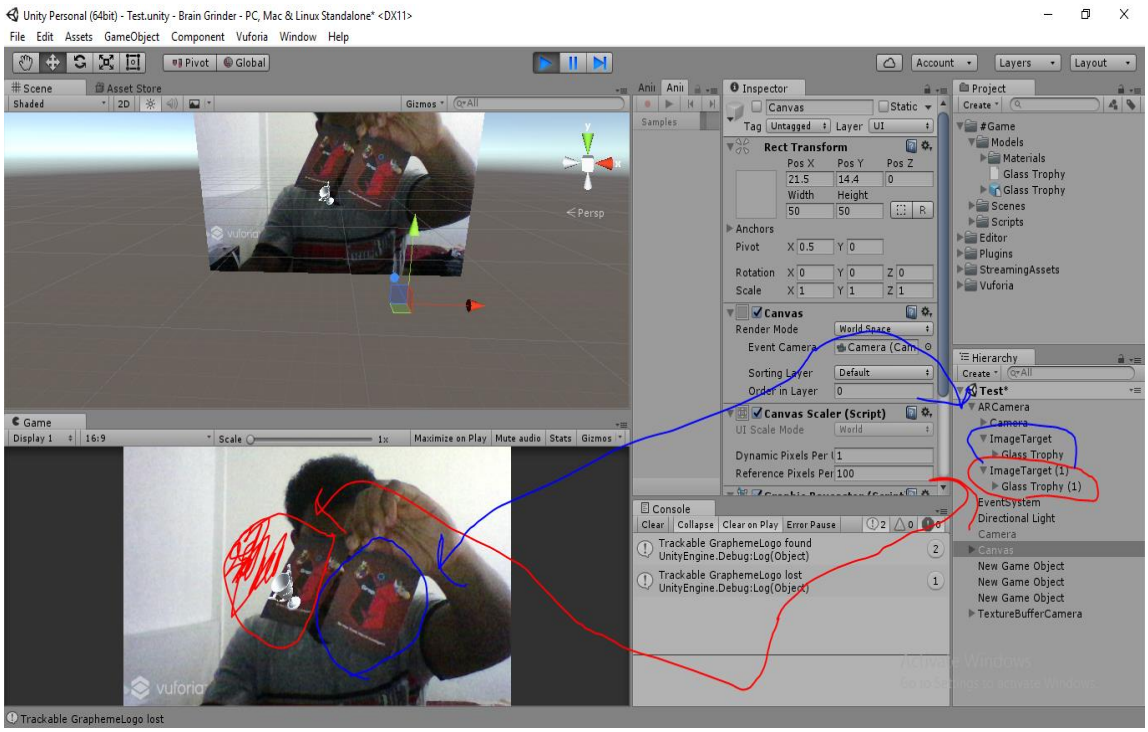


Figure 7.4 Multi Gaming Cards Issue

As displayed in the figure above, only one of the gaming cards got the 3d object activated or shown omitting the existence of any other gaming card or image with same surface which tells us that one dataset cannot be duplicated.

I tried to make this method approachable (since it's more intractable) with using Vuforia's "Multi Target" and other features but that was a dead-end in all paths. In the end, I found that holding the phone while trying to reposition the card on any surface is just a tad of unnecessary work so, after a reasonable amount of time invested in thinking of a new way that results in the same intractability sense, it hit me, just one big play-card

that gets placed on the floor is enough to do all the work which is the playfield image shown in Table 1.

CHAPTER 8

FINAL GAME DESIGN

8.1 CONCEPT

A memory game where the player is requested to form words with given shapes that represent every letter in that word. The player has to reposition given shapes on a playfield on the ground by moving in any direction. The playfield is a printed image that gets recognized by the application once the phone is pointed at it using Vuforia library.

Figure 8.1 Grapheme Playfield

8.2 NAME & GOAL

The goal of the game is to enhance the user ability to connect between their left & right brain hemisphere by connecting shapes to their respective letters.

As for The name of the game it would be “Grapheme”, it means the smallest meaningful contrastive unit in a writing system.

8.3 MODES

Initially, there is 2 modes

 # Solo – where the user tries to finish the game on all given stages.

***Frog* -** Player will be asked to form/pronounce words of 3 - 5 letters.

***Ant* -** Player will be asked to form/pronounce words of 6 – 9 letters.

***Pigeon* -** Player will be asked to form/pronounce words of 10 – 12 letters.

 # Party – Where the player competes against other players through Bluetooth.

But the development of the models, designs & compiling of the prototype took more time than expected so, I have to sacrifice the party mode for the sake of a working prototype.

8.4 RULES

The game provides two difficulty choices for the player (Medium & Hard) where

 # Medium – Represents a fixed connection between shapes(models) & their letters.

 # Hard – Represents a random connection between shapes(models) & letters that is generated at the beginning of every wave.

So, if the player choses medium difficulty, he/she is provided with a reference book that shows how models are mapped to English letters as shown in Figure 8.2.

Figure 8.2 Reference Book

On the other hand, if a one chooses the hard difficulty, shapes will get placed with letters on top of them to represent the temporary connection on the start of every wave.

8.5 ELEMENTS

The elements of the original game design still stand with minor additions like

 # Skip Word – Player can skip up to 3 words in a row.

 # Move Shape – once this button is pressed and a shape has been selected the player has to move around with their finger on screen for that shape to follow them (instead of the classic UI controls)

8.6 IMPLEMENTATION TIMELINE

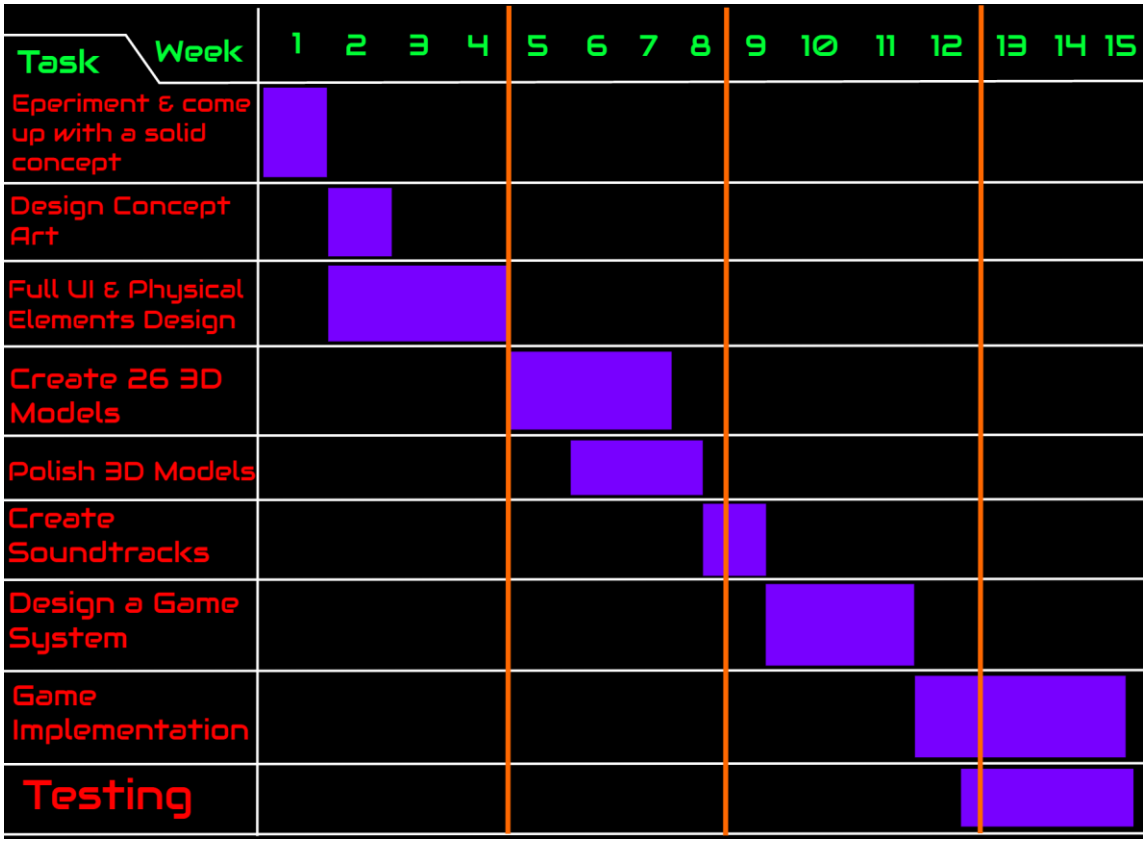


Figure 8.3 Implementation Time Line

CHAPTER 9

GAME DEVELOPMENT (OVERVIEW)

9.1 ENVIRONMENT SET-UP

Before I start the development process, I love to make sure everything is in place and properly working to avoid any un-necessary mid-progress interruptions, next segments or sessions will show you how i prepare my development environment with all necessary dependencies.

Vectr

This tool can be used online or downloaded to be installed on any machine, I prefer to use it online due to its desktop version deficiencies. All you have to do is register with an email and password to have your work saved and revisited anytime you wish to, as long as internet connection is guaranteed designing online is the best choice.

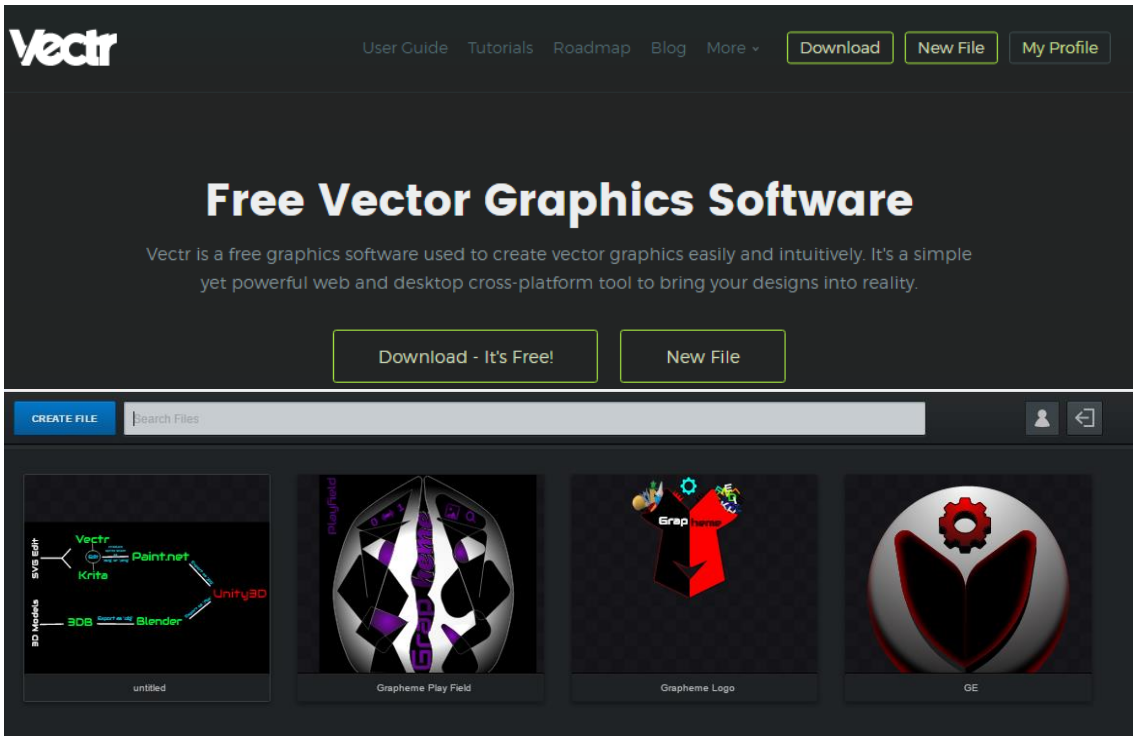


Figure 9.1 (Vectr Online Tool)

Paint.net

I consider this one the savior of my artwork because it's so easy to install and set-up not to mention it's very versatile with high pixelated images, all you have to do is get the installer from the official website (Figure 9.2) and make sure you're computer meets the minimum requirements for the application to run smoothly.

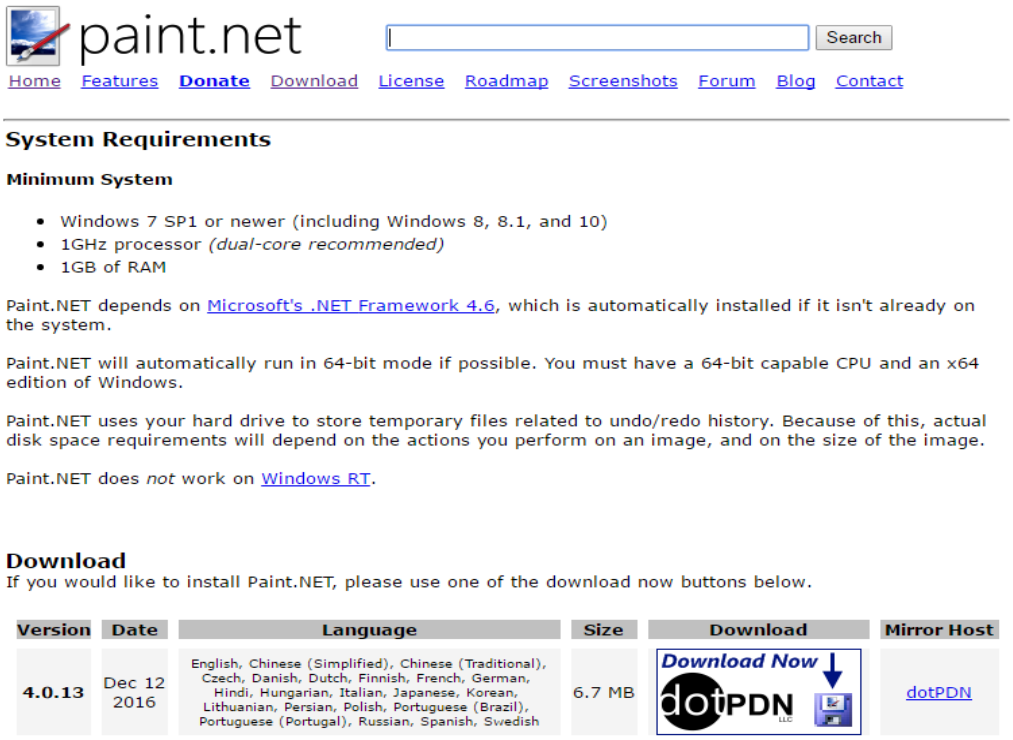


Figure 9.2 (Paint.net Official Website)

Krita

Buried in the depth of the internet behind all the popular digital image editors this tool is becoming popular for it's purpose, finding the official website wasn't something easy to do but definitely worth looking for, download the tool from the official website and make sure your computer specs meet the minimum (Figure 9.3).

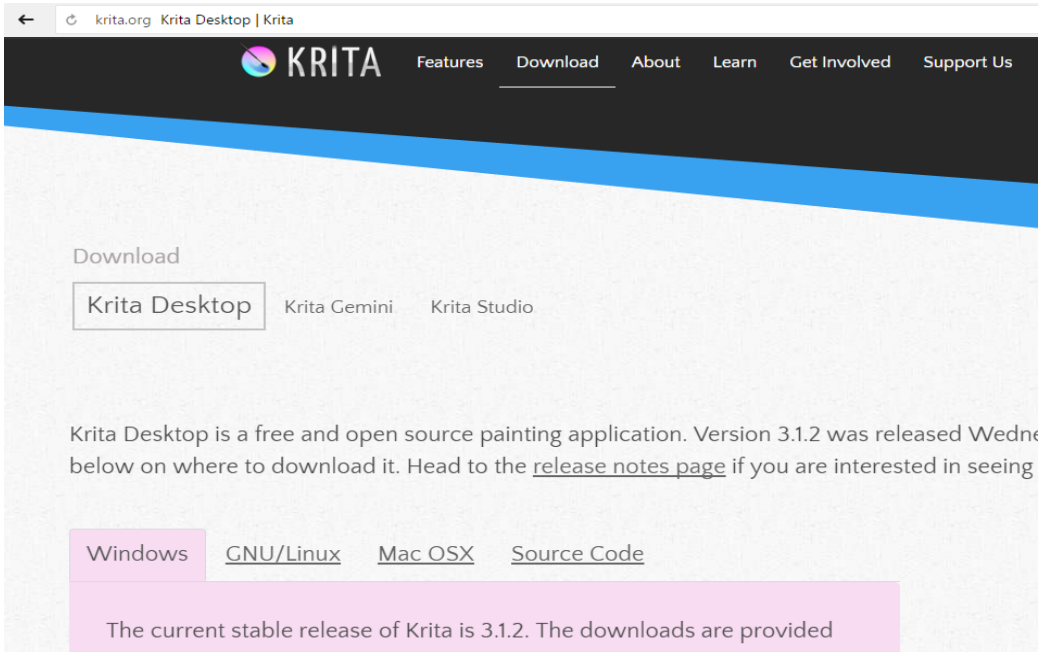


Figure 9.3 (Krita Official Website)

Microsoft 3D Builder

This tool comes pre-installed for free with every windows operating system based machine, in case you don't have, you can head out to Microsoft store and search for it. Even though it was originally created to help build 3d models for 3d printing, you'll find it pretty useful to do the basics of constructing 3d models. Follow the short tutorial (Figure 9.4) to find where the Microsoft store is on Windows OS then search for this tool.

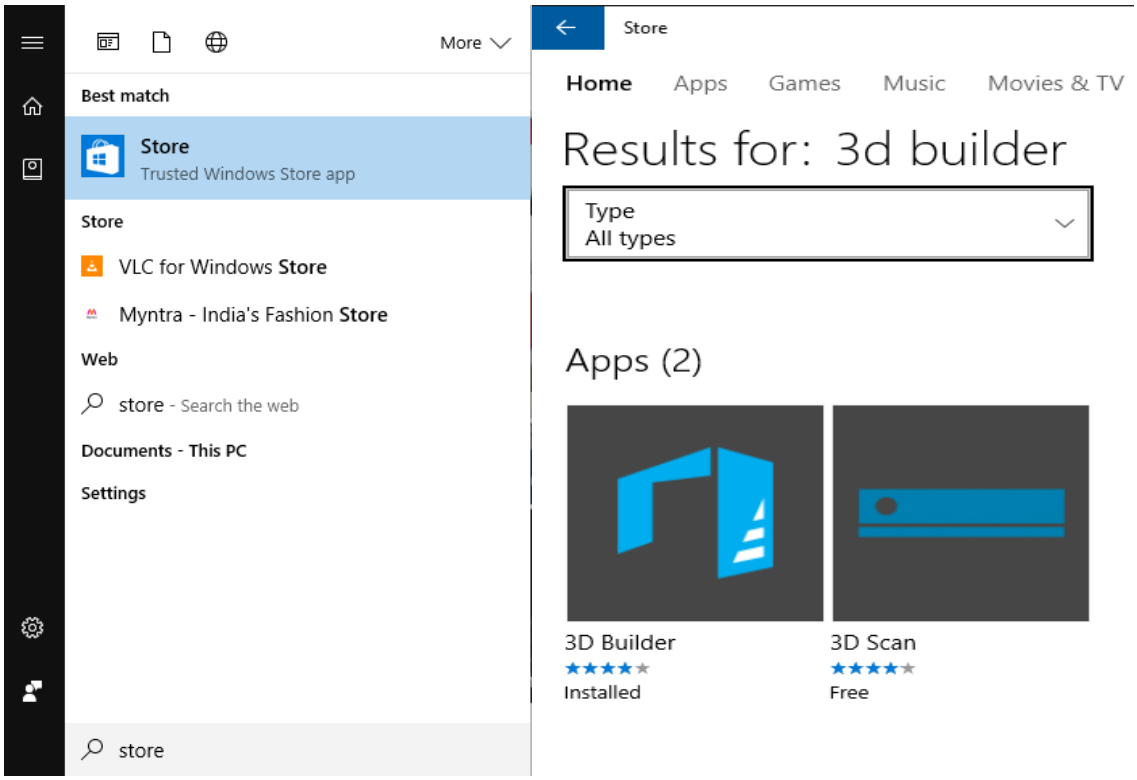


Figure 9.4 (3D Builder Installation)

Blender

Finding this tool won't be hard nowadays since it's popular and globally used, just make sure your desktop personal computer or laptop can adapt to softwares with high memory and graphics usage if you're building something graphically consuming and exhaustive.

Microsoft Visual Studio Community 2015

If you're building a game using one of the big boys (Unity3d or Unreal Engine) you need a cross-platform debugging and compiling platform that can handle all the background processing tasks without bothering you much with it and with a simple click of a button. Two of those are Mono Develop and MVS, I prefer MVS because I got used to it overtime and it's a bit tedious to change now but, if I were to develop this game on a Mac or Linux operating system based machine I would choose MonoDevelop because it's free and hey, it comes included with Unity3D as well.

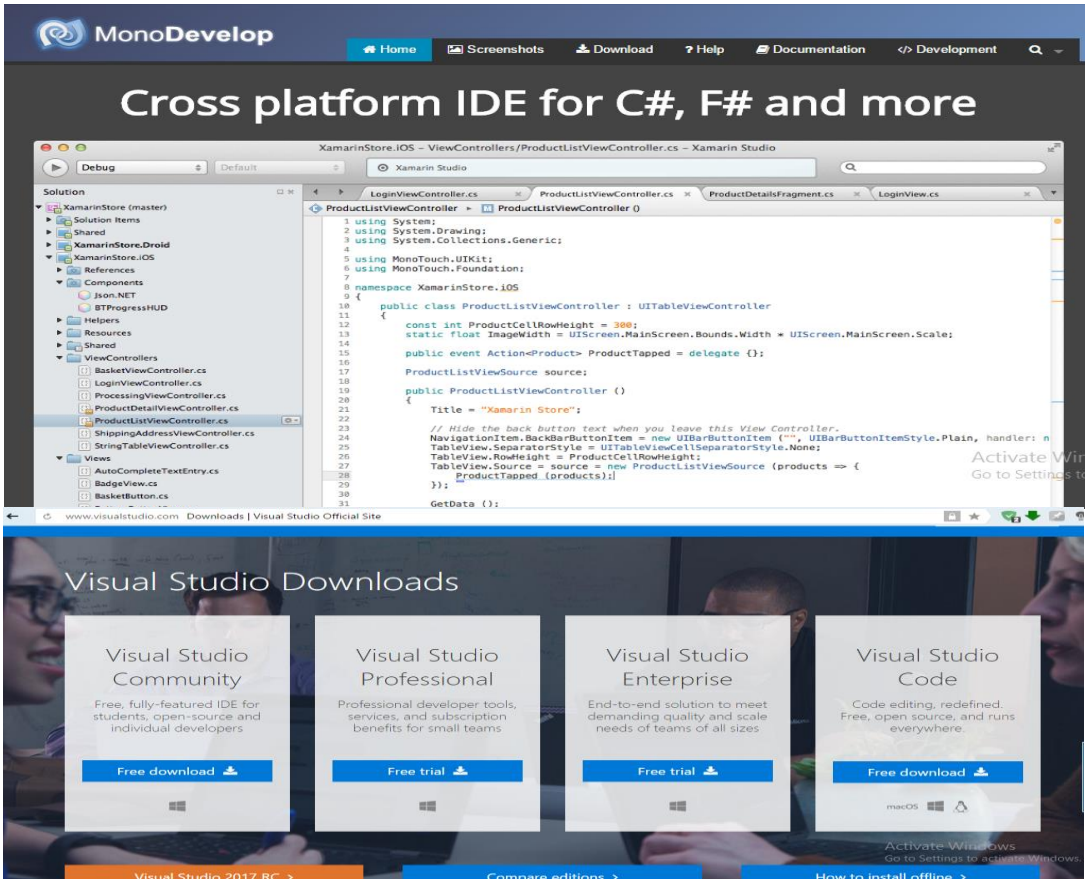


Figure 9.5 (Cross-Platform Programming IDEs)

And one more thing, if you're using a gaming engine (Unty3D in my case), MonoDevelop gets installed by default and you have to make the choice weather you want to add Microsoft Visual Studio Community or not, it's a matter of opinion in the end after all.

Unity3D

The notorious game engine that needs no introduction, with its free add-ons and periodic updates it opens the world to millions of developers to make games. Installing it from the official website and following the on screen installer instruction hasn't been easier.

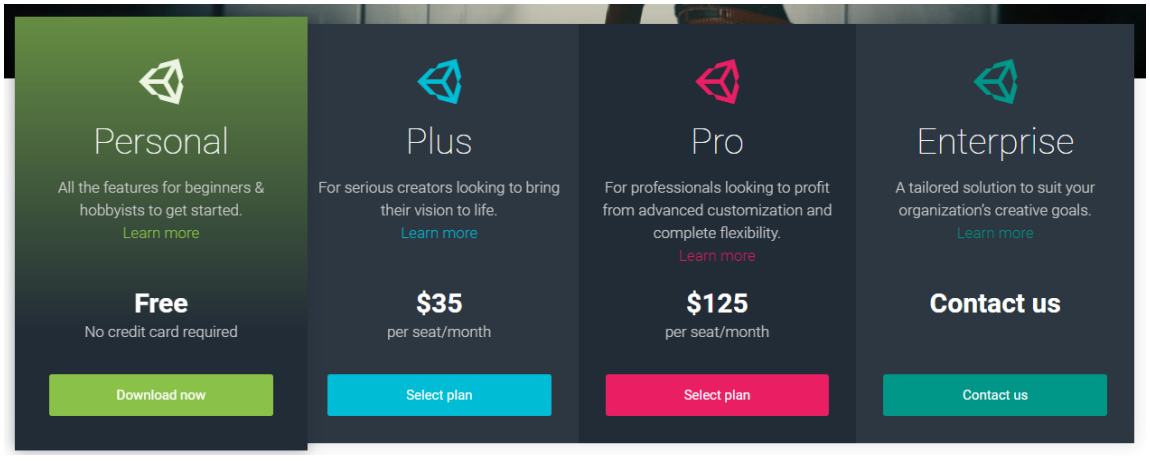


Figure 9.6 (Unity3D Website)

Plus, one of the reasons you would love Unity3D is for its multi-platform productions,

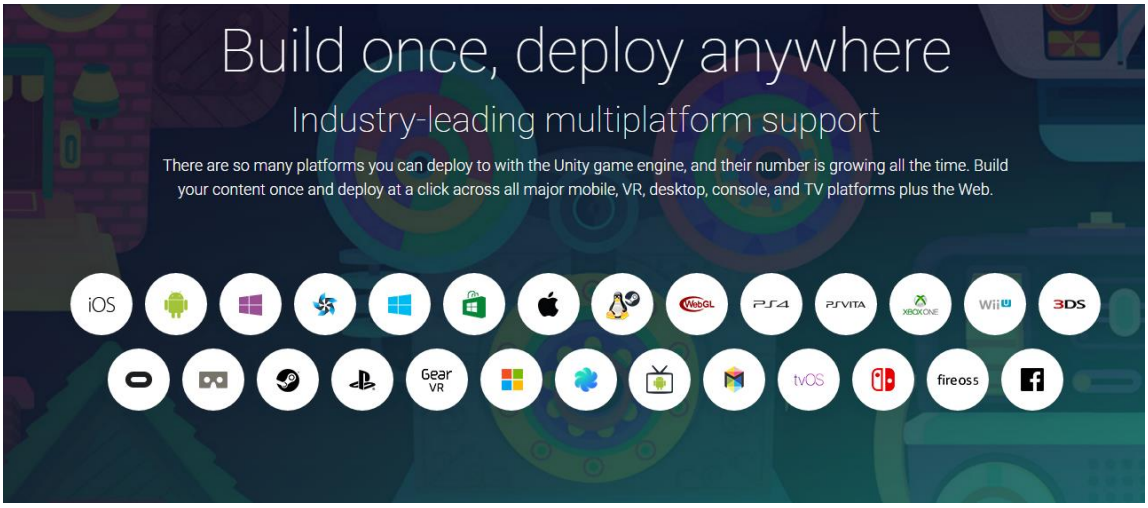


Figure 9.7 Unity3D Multi-Platform Productions

It doesn't limit your development to one platform even though that should be properly planned and well thought of right from the start but, in the majestic case of that not happening, you can still choose the specific platform you're aiming to deploy the game on while writing the code, Unity3D uses script defining symbols to distinguish between platforms (Figure 9.8), all you have to do is write your platform-specific code lines under the respective platform defining symbols (Figure 9.9).

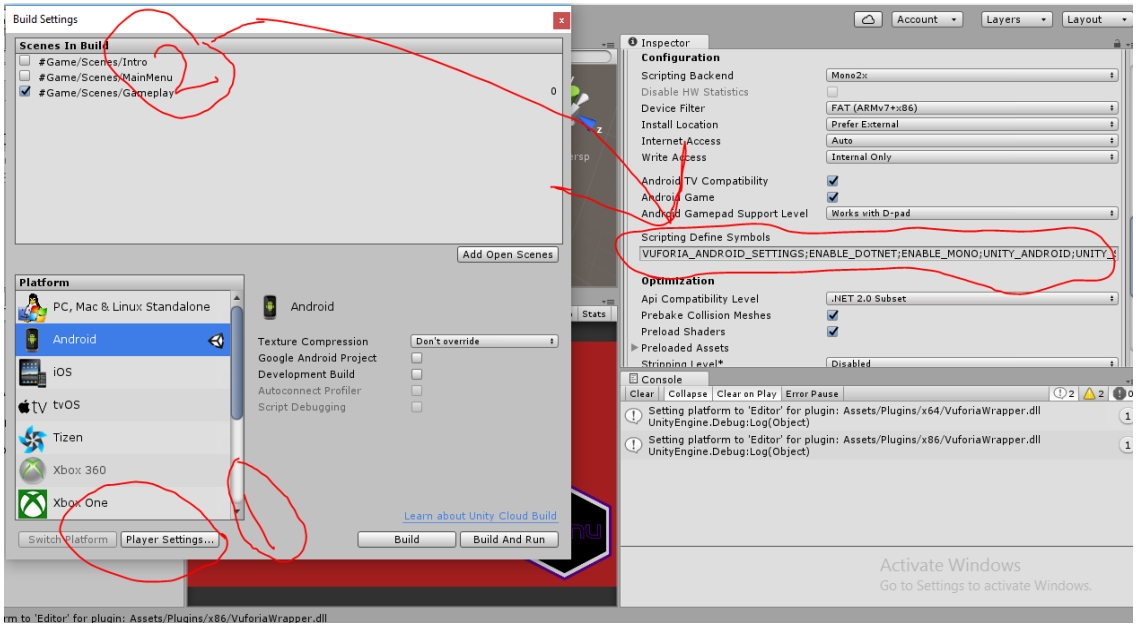
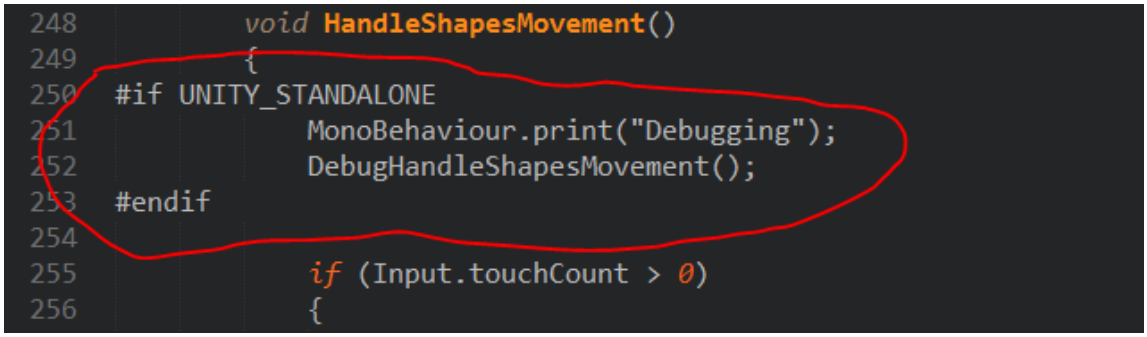


Figure 9.8 (Unity3D Script Defining Symbols)



```
248         void HandleShapesMovement()
249         {
250 #if UNITY_STANDALONE
251             MonoBehaviour.print("Debugging");
252             DebugHandleShapesMovement();
253 #endif
254
255             if (Input.touchCount > 0)
256             {
```

Figure 9.9 (Script Defining Symbols Example)

Vuforia

The free to some extent Augmented Reality library works on many platforms and it provides a Unity3D package for easy importation and use. You can get the package from the official website and double click it while the Unity3D project is working so the library gets installed or unpacked directly to your project. If the

process is confusing just follow the tutorial the little tutorial shown in (Figure 9.10).

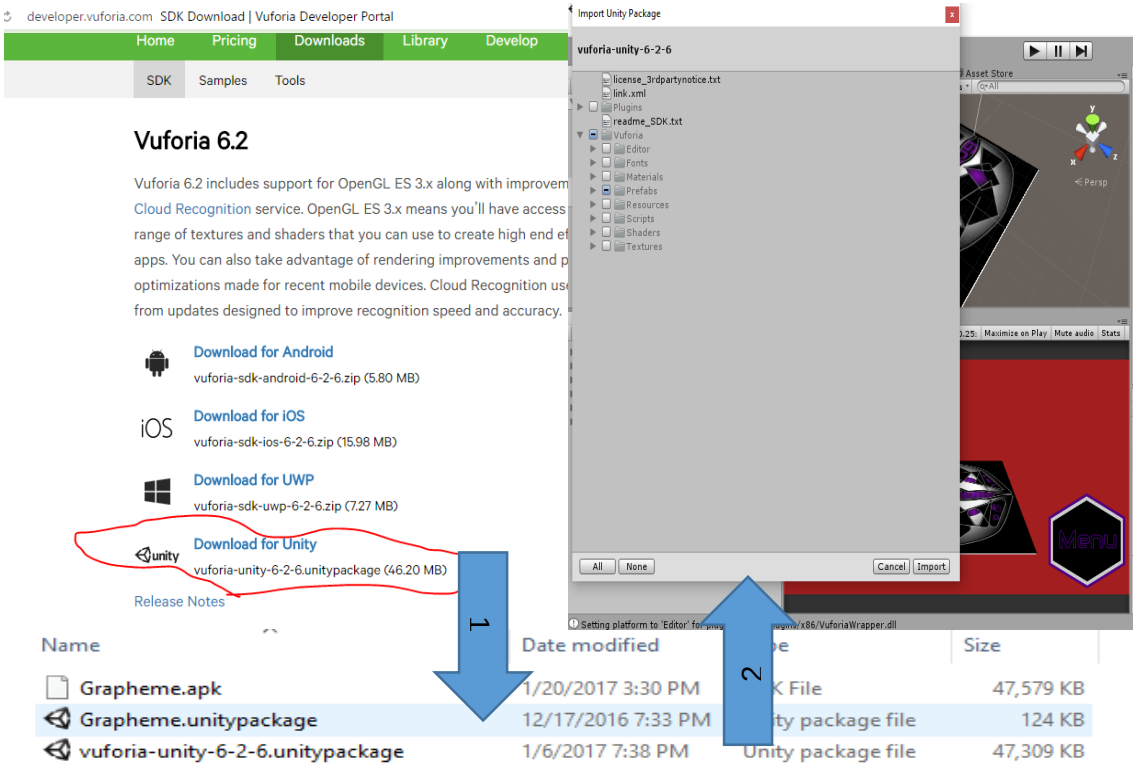


Figure 9.10 (Vuforia Library Set-up)

JSON

Before we go deep in programming I would like to explain what I used to save user data and possibly interchange it between users if the game gets bigger and there's an online hub for it. So, JSON or JavaScript (a programming language like C#) Object Notation, is a lightweight data interchange format and is a method of representing data structures, while it's mainly used by JavaScript, JSON is readable by many programs. In general, it transforms specific piece of content

(i.e. a class attributes like user_name & phone_no) to lightweight data structures that can be saved (the correct term would be “Serialized”) in what looks like a text file for later reference.

The notation of a JSON file starts with ‘{’ and ends wth ‘}’, whatever in between is defined according to data type used, so, an example of a json file would look like this

```
{
  user_name : “Dr.junaidi Abdullah”,
  phone_no : 01124536645
}
```

Inside a file named *className*.json. you can also use it to save images and assets as we’ll see in the development overview.

Folders Hierarchy

Keeping the folders organized is a huge deal to maintain efficient and stable application development and quick asset reference. Here’s how my asset folder looks like after studying perfect case hierarchy. As shown in figure below, the folder/s that start with “#” are mine to include anything in them and the rest belongs to Vuforia library.

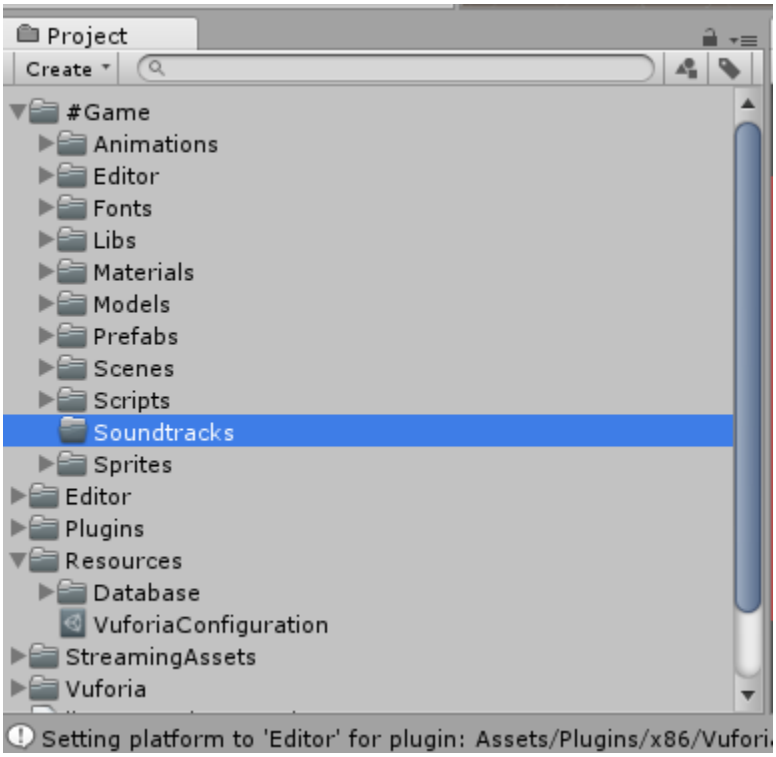


Figure 9.11 Folders Hierarchy

Now, let's talk about the special folders, "Animations" contains only the animations made using Unity3D Mecanim (Internal Animator), which are mainly UI animations. "Editor" folder works in coherence with "Scripts" but it contains scripts specially made for the Unity3D editor to ease some the testing functionalities. The rest of the folders is pretty much self-explanatory (if not we'll revisit them later).

9.2 BASE CLASSES

Before I explain the purpose of my base classes, it is very important to go over the basic programming skills:

- Abstraction – "The essence of abstractions is preserving information that is relevant in a given context, and forgetting and forgetting information that is irrelevant in the context" – John V. Guttag.

- Inheritance – is when an object or class is based on another object or class, using the same implementation (inherited from the base) or specifying a new implementation to maintain the same behavior.

- Polymorphism – It is the provision of a single interface to entities of different types. A polymorphic type is one whose operations can also be applied to values of some other type.

- Namespaces - are designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Now, and after the quick recap of the some of Object-Oriented-Programming skills, let's talk about the main classes used to develop the game.

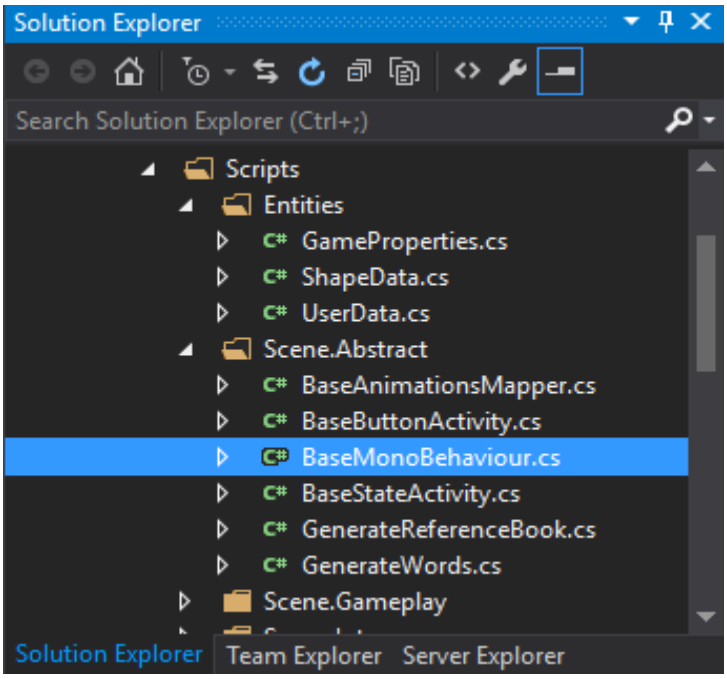
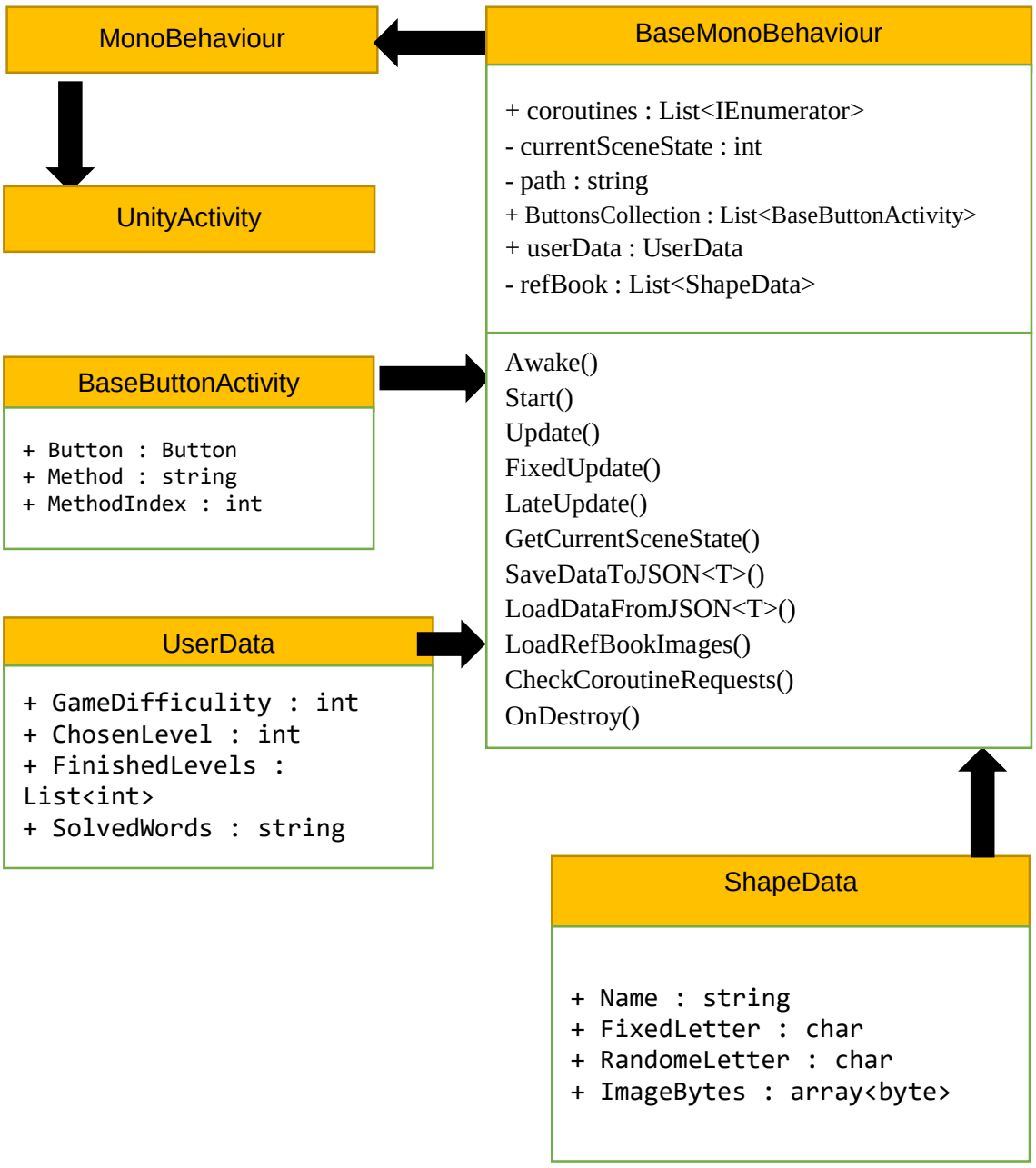


Figure 9.12 (Base Classes & Data Entities)

First, ‘BaseMonoBehaviour.cs’ script is the main controller of the game and classes that need to be serialized with unity has to extend this one because it is responsible for recording user activities and achievements in black box encapsulation form.



Class Diagram 9.1

As I mentioned, this class is responsible for almost everything in the game, the ‘currentSceneState’ depends on a predefined list of scene states declared by children of

this class and saved inside an enum, all game objects refer to this variable to know what action should be triggered at the moment, for example, if the 'currentSceneState' raises 'playerDied' flag, then animations and game logic will get triggered according to that. This class uses custom defined classes (UerData & ShapeData) as data types for special purposes that are going to be discussed later on. The last attribute is 'coroutines' which static for non "MonoBehaviour" based classes that need to run parallel with game scripts.

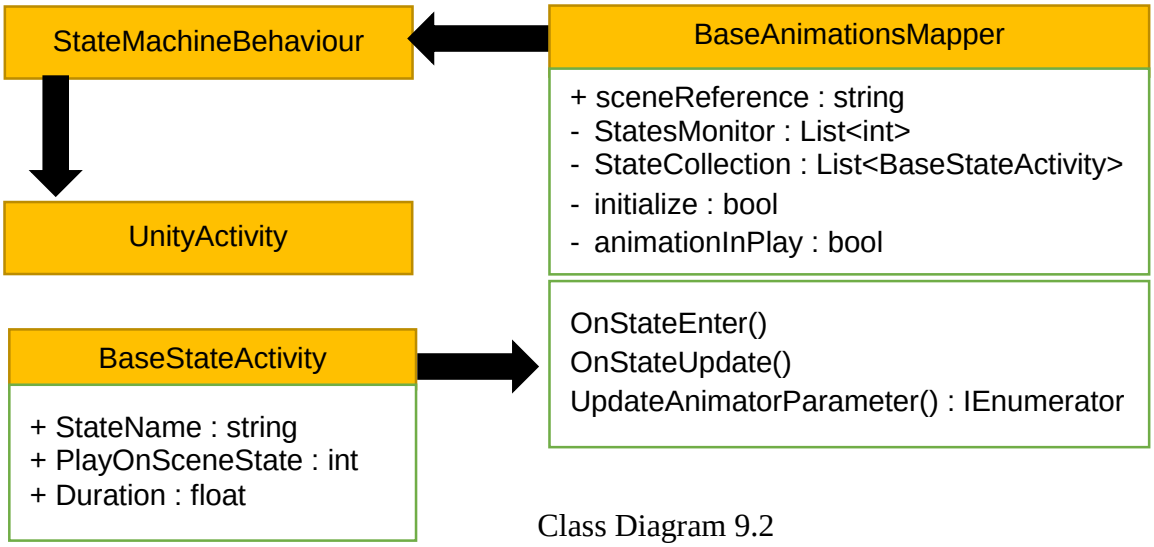
It won't be fair to talk about attributes without mentioning who uses them so, let's go over the functions and what each one means:

- Awake() – Is the one responsible for loading userData from the JSON file(I mentioned earlier) in the beginning of every session of the game, giving updated data about user difficulty choices and stage selections.
- Start() – Is made for the classes that don't need to use data retrieved from JSON files, instead just have its own implementation running (Abstract).
- (All Frame Updated Functions) – Get called every frame by unityActivity
- SaveDataToJSON<T>() – A template function to save any data in the game as JSON object in special files set in the parameters.
- LoadDataFromJSON<T>() – A template function to load JSON data from files without a specified need for its data type by passing data type and object to save them in as parameters.

- OnDestroy() – This function gets called when the game is terminated, crashed or just jumping between scenes, under any case, it terminates with saving the latest user information.
- LoadRefBookImages() – After saving the images as bytes inside JSON files to reduce game file size I need to be able retrieve and show them when needed and that's where this function come to work, calling it will fill the 'refBook' attribute with all necessary data needed to show images.

And don't forget, this class gets attached to game objects not animation layers as we'll see in the future, I'm going to talk more about it in the next chapter.

Second, 'BaseAnimationsMapper.cs' script is the father of all animations in the game, this script was independently developed because some of the scene states were jumped or foreseen due to fast frame updates in the 'Update()' function.



Class Diagram 9.2

The way it works is very simple and of course can be inherited, all it does is, first, save a custom made list (‘StatesCollection’ attribute) of animation states and their action triggering scene state (referenced by ‘SceneReference’ attribute) as shown in (Figure 9.13).

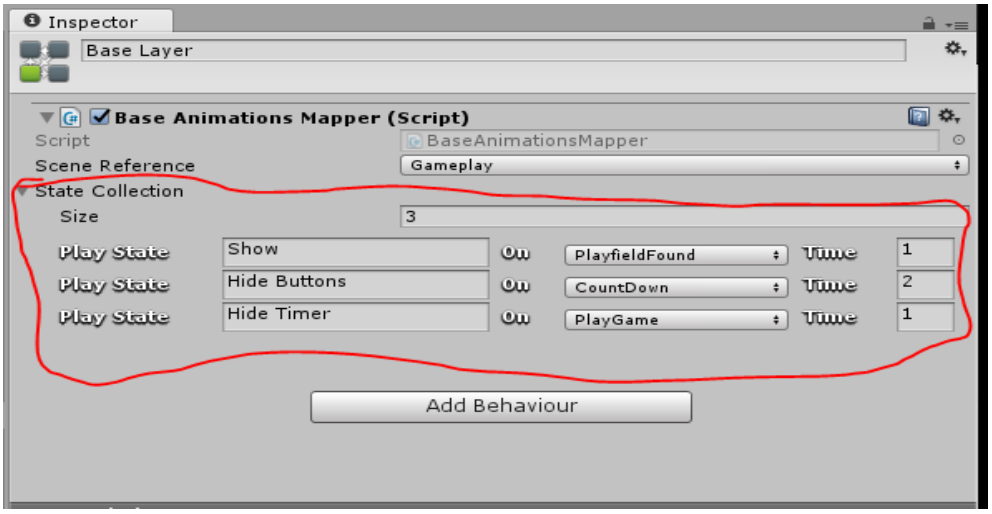


Figure 9.13 (State Collections)

Then whenever it is time to fire them the loop will run incoherence with main game loop to activate all necessary animations associated with current scene state, the way it works is explained in the coming pseudo code.

```
<psoudu>
Main loop()
{
        If (currentSceneState != 0 ) // Not 'DoNothing'
        {
                If (StatesMonitor.count <= 0)
                        StatesMonitor.Add(currentSceneState);
                Else
                If (StatesMonitor[LastIndex] != currentSceneState)
                {
                        StatesMonitor.RemoveAt(0);
                        StatesMonitor.Add(currentSceneState);
                        AnimationInPlay = false;
                }
        }

        If (animationInPlay = false AND StatesMonitor.count > 0 )
        {
                Forevery state in StateCollection Where (playOnState = StatesMonitor[0])
                {
                        Play(state);
                        Locked = true;
                }
        }
}
<!/psoudu>
```

Third, there is two important scripts (Not represented in class diagram because their work is outside of the game loop) that help maintain consistent data and minimize the size of the game file in general, which are 'GenerateReferenceBook.cs' & 'GenerateWords'. The reference book generator works by attaching It to any game object that have all the 3d models as its children, then it'll loop through each one and

create a JSON file consists of all shape necessary information that will be used later in the game, the next figure will show you how the reference book is generated.

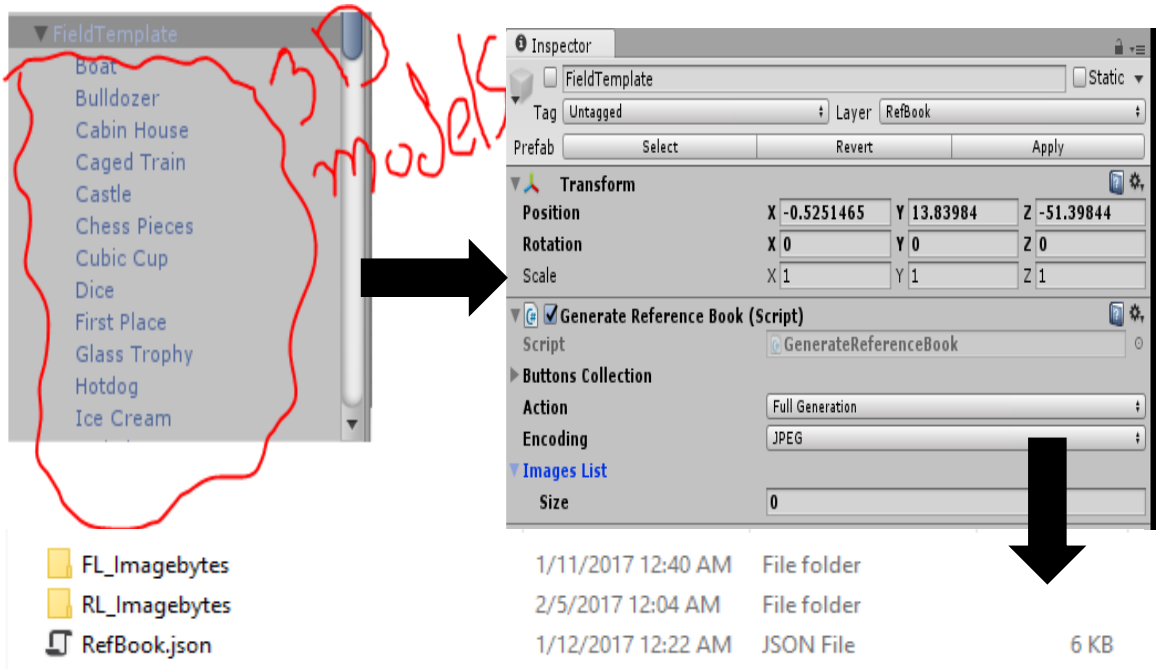


Figure 9.14 Reference Book Object

After The file is script is attached to the game object, you'll see different reference book generation options (i.e. 'FullGeneration' or 'OnlyImages') and the encoding of images that are going to be in the 'ImagesList' attribute (name is case sensetive).

When play is clicked, all the files & folders generated are saved under the unity special directory 'Application.persistentDataPath', and the next figure will demonstrate how the JSON files look like with the custom data type (or class) 'ShapeData.cs' that we've seen earlier in (Class Diagram 9.1).

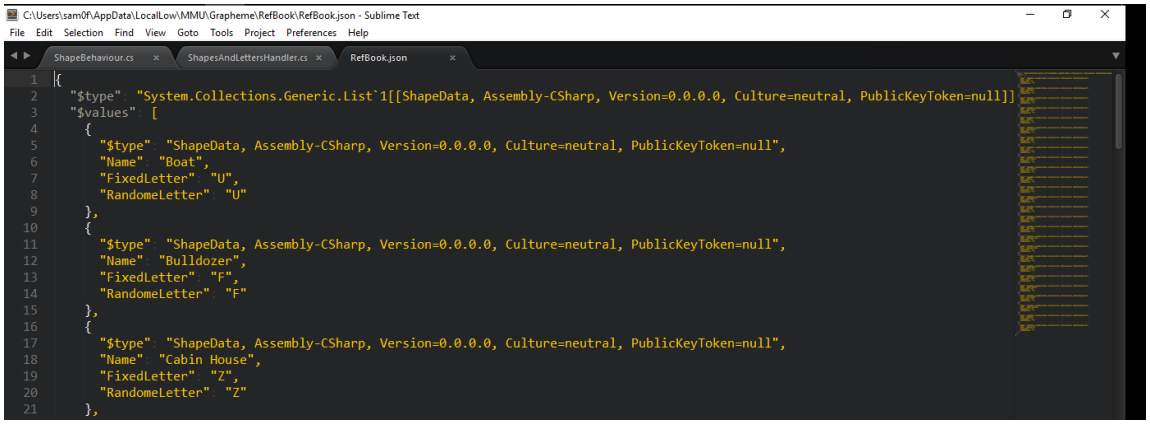


Figure 9.15 (Reference Book As JSON)

Not to mention saving images encoded in ‘JPEG’ as displayed in (Figure 9.16) in a more compact way that doesn’t consume size or space inside of a JSON file too, calling the ‘LoadRefBookImages()’ function discussed earlier loads images from these files into their respective 3d models to be shown for the user, below is the figure that shows image files for each shape and you can see final result in (Figure 8.3).

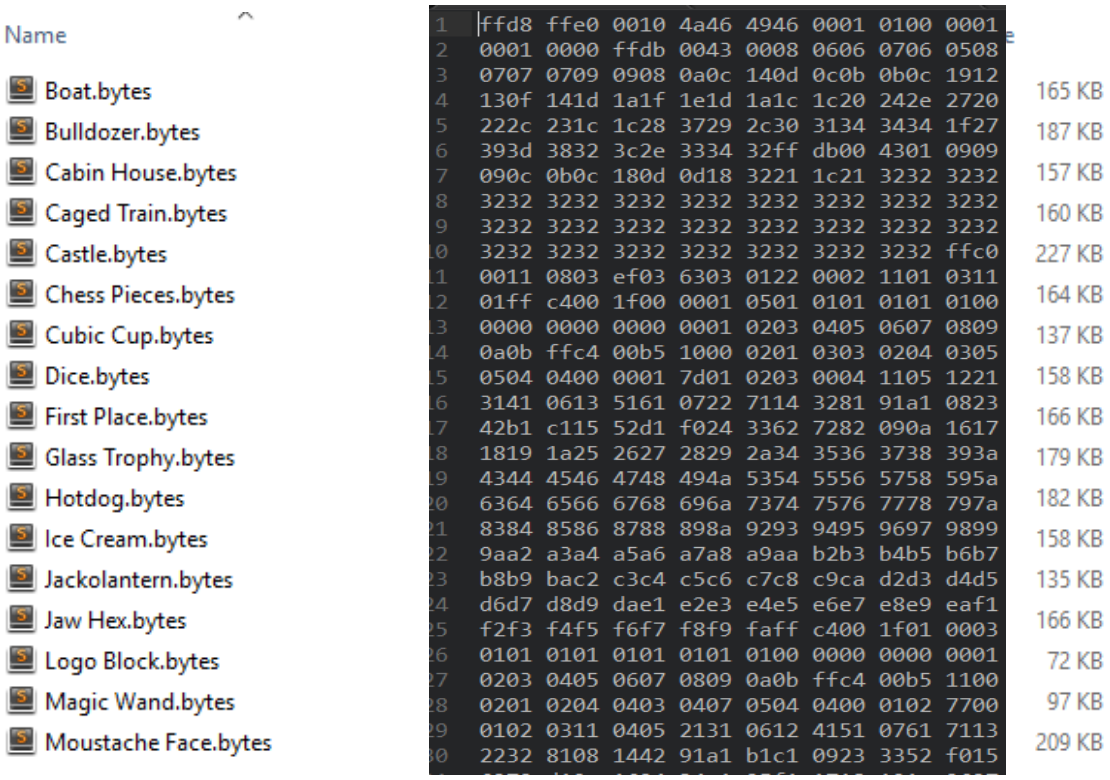


Figure 9.16 (Reference Book Images)

As shown in (Figure 9.16), an image file contains an individual image pixels colors in the data type ‘byte’ array.

Last, the remaining scripts ‘BaseButtonActivity.cs’ and ‘BaseStateActivity.cs’ have their own purpose of existence so, let’s discuss our way up to know why they’re special. Considering the first script, buttons can be created easily within unity but, the game will reach a level where there simply exist too many buttons and it’s very hard to keep track of them and their dedicated functionalities. With that in mind, it would be highly maintainable and quickly adjustable if a button system built on top of unity’s was created, and here is where the ‘BaseButtonActivity.cs’ come to action.

It works by providing the developer (me in this case) an interface that requires a button to be added to a list and it'll automatically show all the functions in a script dedicated to keep buttons activity functions.

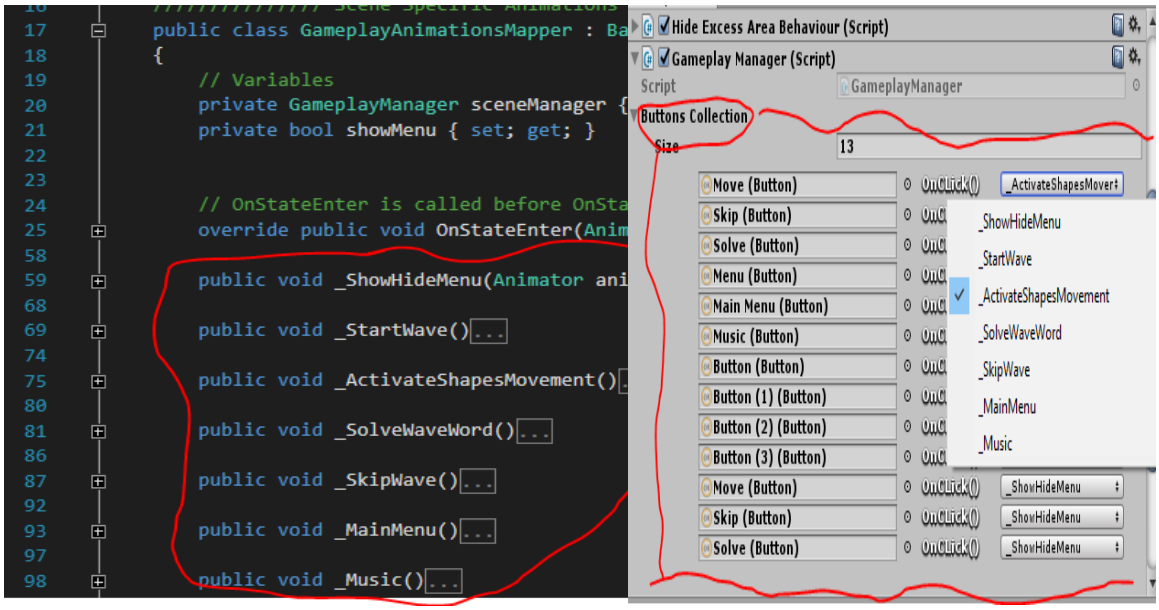
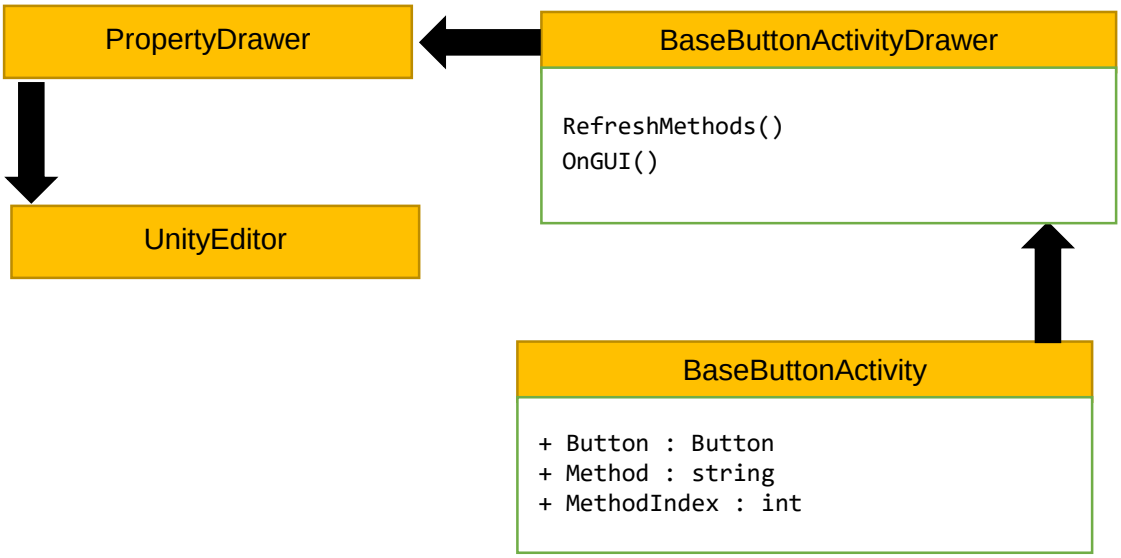


Figure 9.17 (Button Activity)

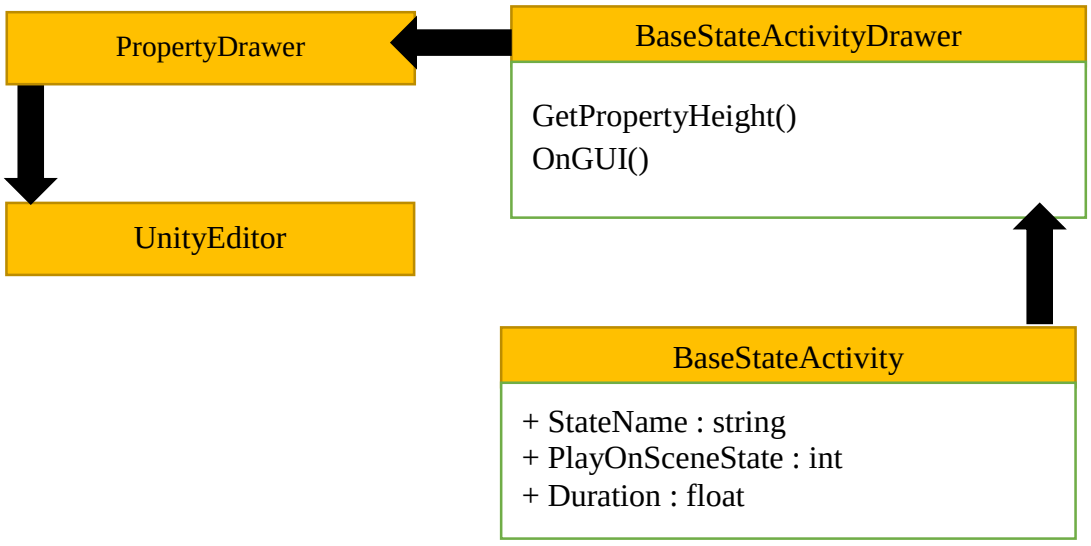
(Figure 9.17) shows that functions inside of a script show as options in a list so, how does this happen?. Unity3D provide a set of features or abstract scripts to manipulate the editor itself, and that is exactly what is happening here, take a look at (Class Diagram 3) and I'll explain afterwards.



Class Diagram 9.3

The class diagram doesn't really help show the connection between the two scripts, perhaps this example would explain it better. Imagine a very old car (the 'BaseButtonActivity.cs' script) that has to be sold in few days, everything about the car is almost broken but, it works properly only if the original owner is there (the 'BaseButtonActivityDrawer.cs' script) to drive it so, in other words, to buy the car you have to buy the driver with it otherwise it's just an ugly case of hot wheels.

With that in mind, the work of 'BaseStateActivity.cs' doesn't differ from the way 'Base ButtonActivity.cs' works at all, with an editor script to help organize and show properties of base script in elegance. The only difference would be that this one get attached to an 'Animator' layer instead of the regular game object.

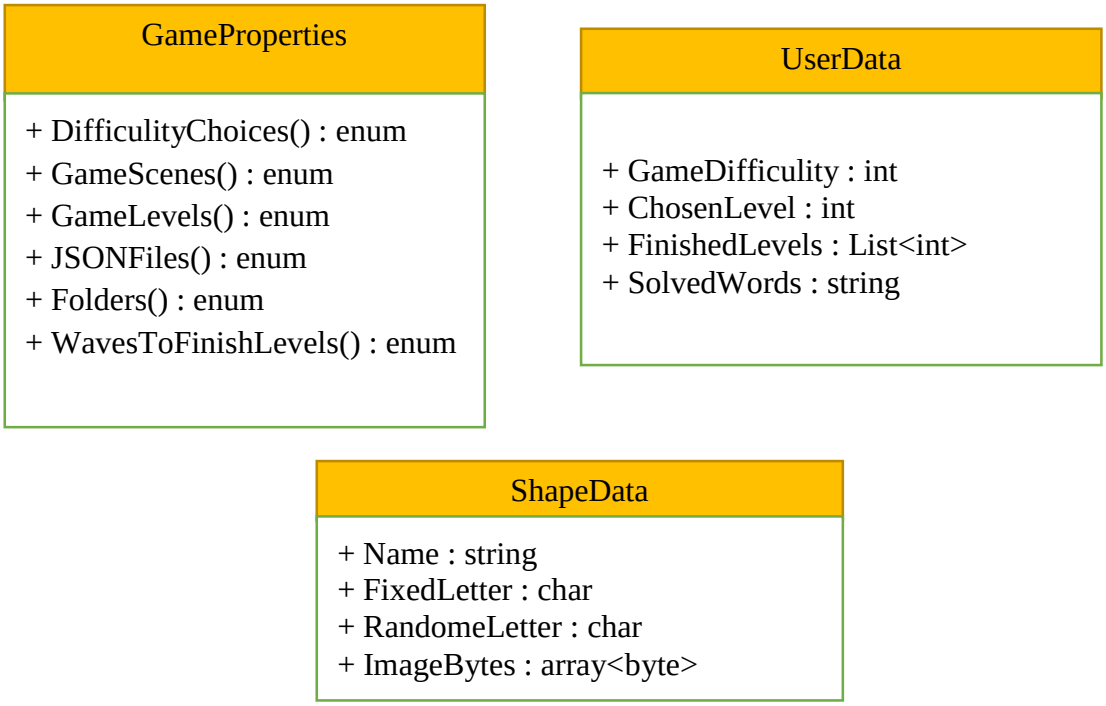


Class Diagram 9.4

9.3 DATA ENTITIES

The need for a database in any game is almost necessary, and that's where the data entity classes shine. These set of classes are built for two reasons, to keep game development persistent, and to save player's in-game progress.

They don't have any connection to other classes except for being a datatype.



Class Diagram 9.5

Now, we've seen 'UserData' and 'ShapeData' in action and it won't be fair not to go over the 'gameProperties' attributes so:

Difficulty Choices – Game's reference to all difficulty choices for user to choose from.

Game Scenes – All the scenes in the game kept in order.

Game Levels – A list of the game's main stages.

JSON Files – To save and load data to/from JSON files consistently

Folders – In case a particular folder doesn't exist, create one with a name from this list.

Waves To Finish Levels – The number of waves necessary to finish one of the Levels.

9.4 A SCENE LOGIC

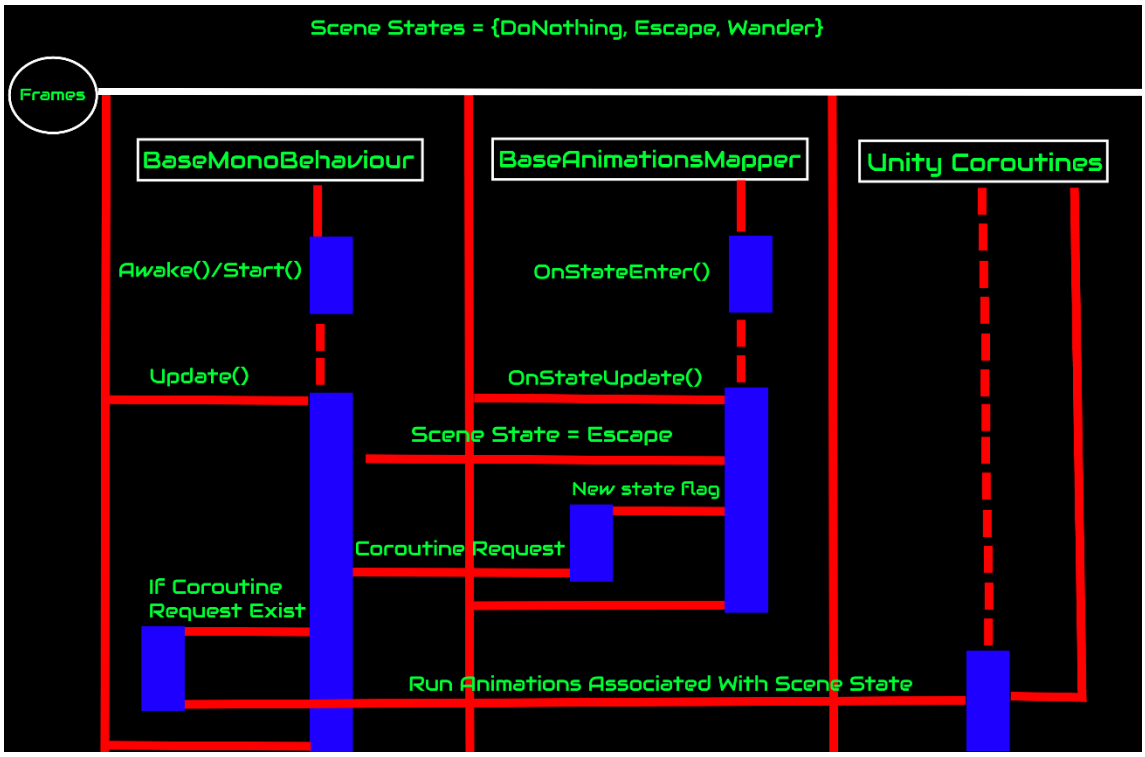
It’s only logical to separate work among team members if a match is on the line, creating more than one ‘scene’ under Unity3D framework development ensures proper division of tasks usually labeled with the name of the scene itself. With that piece of information in mind, I am going to explain the general logic behind any scene in the game before we dive in depth.

Every scene in the game falls under the roof of two rules, first, there must exist only one main camera object with ‘BaseMiniBehaviour.cs’ attached to it, and second, the ‘BaseAnimationsMapper.cs’ must be used on every object that has an ‘Animator’ component.

Now, let’s take a look at the sequence diagram that shows how a scene components run coherently assuming steady frame updates and arbitrary scene states.

Sequence Diagram 9.1

Any scene starts with the ‘DoNothing’ state therefore all animations are set to a starting value in the beginning of the game but what happens if a new state raise a flag?. The next scenario is going to demonstrate how coroutines get triggered by a new game state.



Sequence Diagram 9.2

The scenario in (Figure 47) shows how the animations mapper request a new coroutine to be run and play new animations, and that's how the game loops. If you don't understand why the coroutines don't go back to the main update loop perhaps a quick run through the meaning and purpose of coroutines will help.

When you call a function, it runs to completion before returning. This effectively means that any action taking place in a function must happen within a single frame update; a function call can't be used to contain a procedural animation or a sequence of events over time. As an example, consider the task of gradually reducing an object's

alpha (opacity) value until it becomes completely invisible is controlled by a function ‘Fade’. As it stands, the Fade function will not have the effect you might expect. In order for the fading to be visible, the alpha must be reduced over a sequence of frames to show the intermediate values being rendered. However, the function will execute in its entirety within a single frame update. The intermediate values will never be seen and the object will disappear instantly.

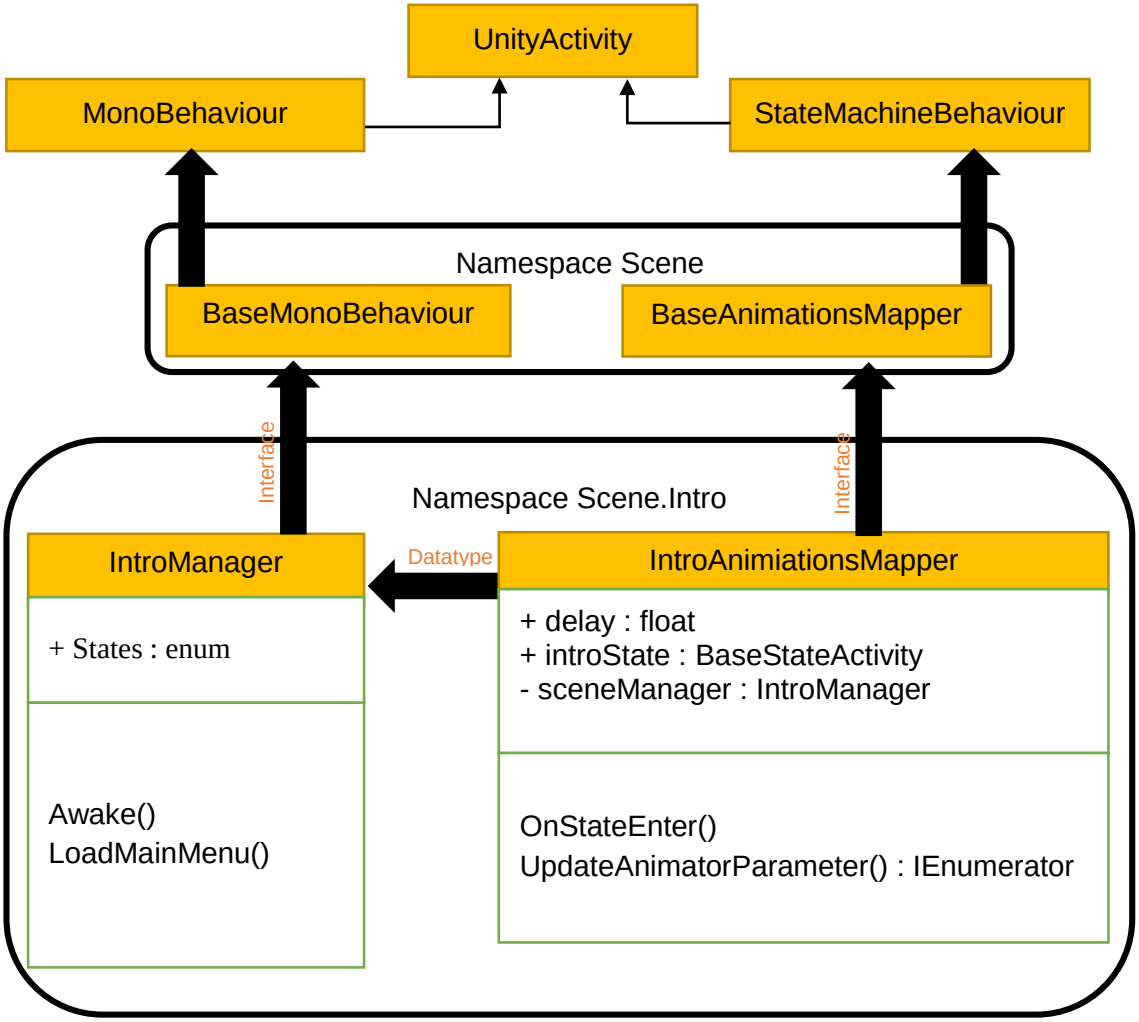
It is possible to handle situations like this by adding code to the Update function that executes the fade on a frame-by-frame basis. However, it is often more convenient to use a coroutine for this kind of task. A coroutine is like a function that has the ability to pause execution and return control to Unity but then to continue where it left off on the following frame.

CHAPTER 10

GAME DEVELOPMENT (IN DEPTH)

10.1 INTRO SCENE

This scene is responsible for the cinematic prologue of the game, where the logo slowly fade-in and fade out. The base classes structures are used as interfaces for this scene managers.

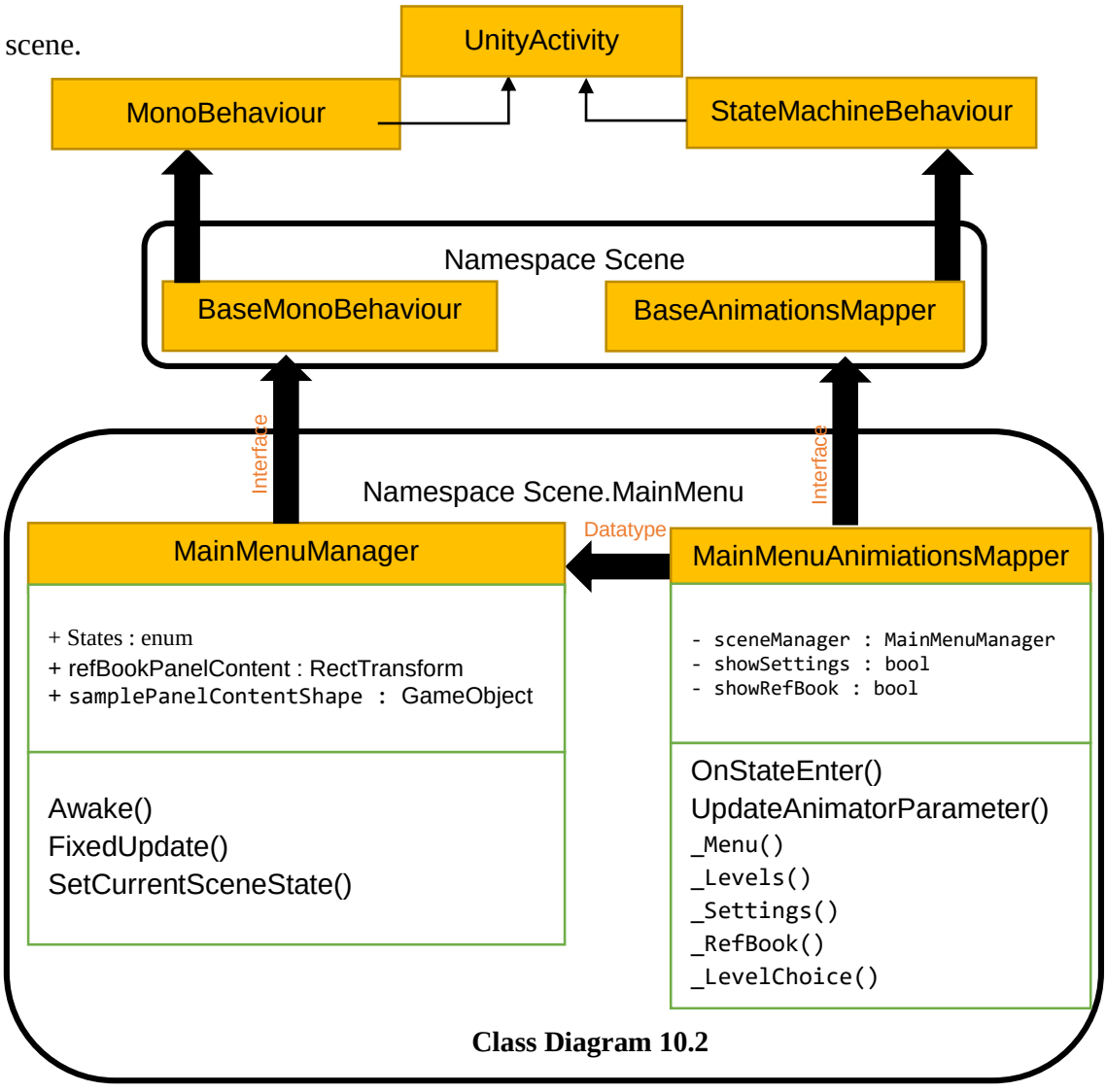


Class Diagram 10.1

The class diagram above shows that scene manager and animations mapper use base classes as interfaces to build their custom implementation on top of them, the same scenario applies to all scenes.

10.2 Main Menu Scene

After the short intro the user is presented a set of options in an elegent way to set up the game difficulty and environment in general, and that the responsibility of this scene.



Class Diagram 10.2

Do you remember when we used our base functions to store the images of our 3d shapes (models) in (Figures 42 - 44) ?!, well now it's time to fetch and place them on screen for user to inspect, otherwise what\s the use of storing them. The transforming from JSON data to UI elements happens in the 'Awake()' function but not yet displayed due to motion paralysis invoked by '_RefBook()' function as button event.

In the same chapter context (Chapter 8), (Figure 9.17) shows that a set of functions that start with an underscore '_' were displayed as choices for button events, these uniquely named functions won't exist in the base class 'BaseAnimationsMapper.cs' because every scene is case subjective and cannot be made as versatile as grouping all these functions under the base class. Explaining the functionality of each of these methods from an event point of view will go as follows:

_Menu() – Show/Hide main menu options (i.e. Solo/Exit).

_Levels() – Show/Hide stages options (i.e. Frog/Ant)

_Settings() – Show/Hide Settings of the game. For now it's just difficulty options.

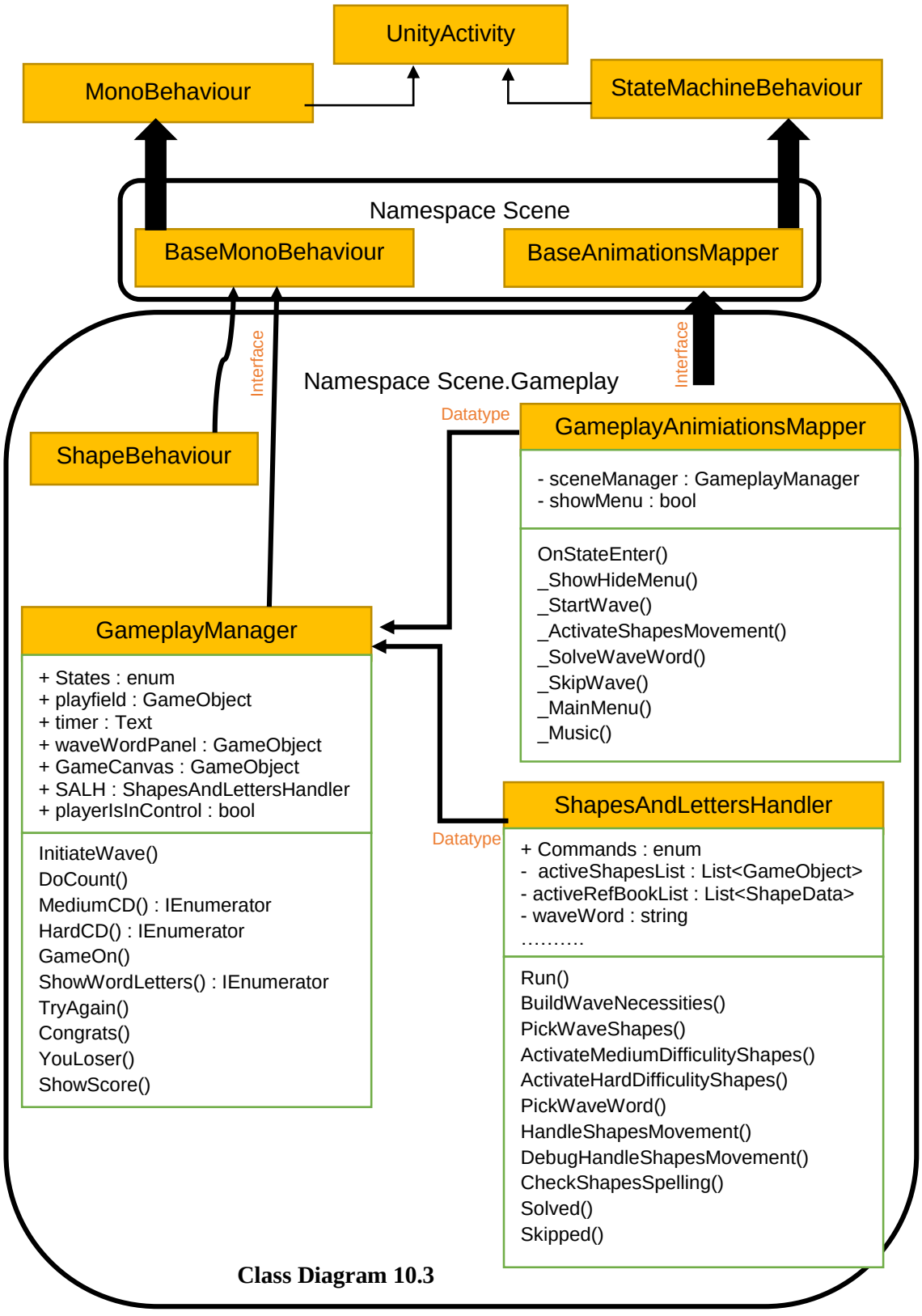
_RefBook() – Show/Hide the fixed connections between 3d shapes and English letters along with their images.

_LevelChoice() – Used in corporation with '_Setting()' to store user data.

10.3 GAMEPLAY SCENE

This scene is the essence of the game, all gameplay related activities will be initiated according to properties set by user during ‘MainMenu’ scene. Considering this scene the godfather of the game, its logic must be properly constructed and maintained to avoid memory overload and high battery power consumption.

The next diagram will include huge chunks of informations so, try to read it as if it was a favorite magazine, and I’ll explain afterwards how these classes work together and what is the purpose of making each one.



Class Diagram 10.3

Whew, it’s a lot right!, let’s break that diagram down and lift that burden off your shoulders.

The two main base classes remain the same with the addition of a sub-framework which is ‘ShapesAndLettersHandler.cs’ script, which I’ll refer to as SALH. SALH’s responsible for positioning the 3d models, fetching the current wave word from JSON file and validate user answer, all according to game logic invoked by current scene manager.

What is the need for a scene manager if SALH does it all? you ask, well, as I said before, SALH only takes care of shapes and letters related matters, the scene manager job in this case is to control the flow of the game, ensure to give user control only if possible (Not loading any assets) and congratulate user once a wave word has been solved.

CHAPTER 11

PRODUCTION

11.1 PLATFORMS

The platform determined for this hasn't changed since planning, deployment on mobile phones platforms like Android and IOS is the aim, mostly Android (for now) because of deadline limitations.

11.2 PERFORMANCE MEASURES

The final product is the trophy when developing a game, therefore, certain measures such as bug fixes, code analysis, deadline adherence and end-user experience satisfactory must be considered.

So, to enforce a well running application that doesn't crash, we're going to take the worst case scenario approach where the game is tested to the limit, then code analysis and bug fixes is performed every time memory overloads or battery power get consumed more than expected.

Unity3D provides a tool called the 'Profiler' to keep track of all games' activities, here I'm going to use it on my phone and show the results but before that, I'm going to list my phone's important specs first:

- Name – Xperia XZ
- Platform - Android OS, v6.0.1 (Marshmallow).
- CPU - Quad-core (2x2.15 GHz).
- Memory – 32 GB, 3 GB RAM.
- Battery - Li-Ion 2900 mAh battery, Stand-by time of Up to 600 h (2G) Up to 610 h (3G).

Now, the best ‘worst case scenario’ would be testing the game with on a smart phone far less capable of working on its own but this is all I got for testing at the time. Let’s observe how the profiler will work with my phone to get analysis of game’s performance (Figure 11.1).

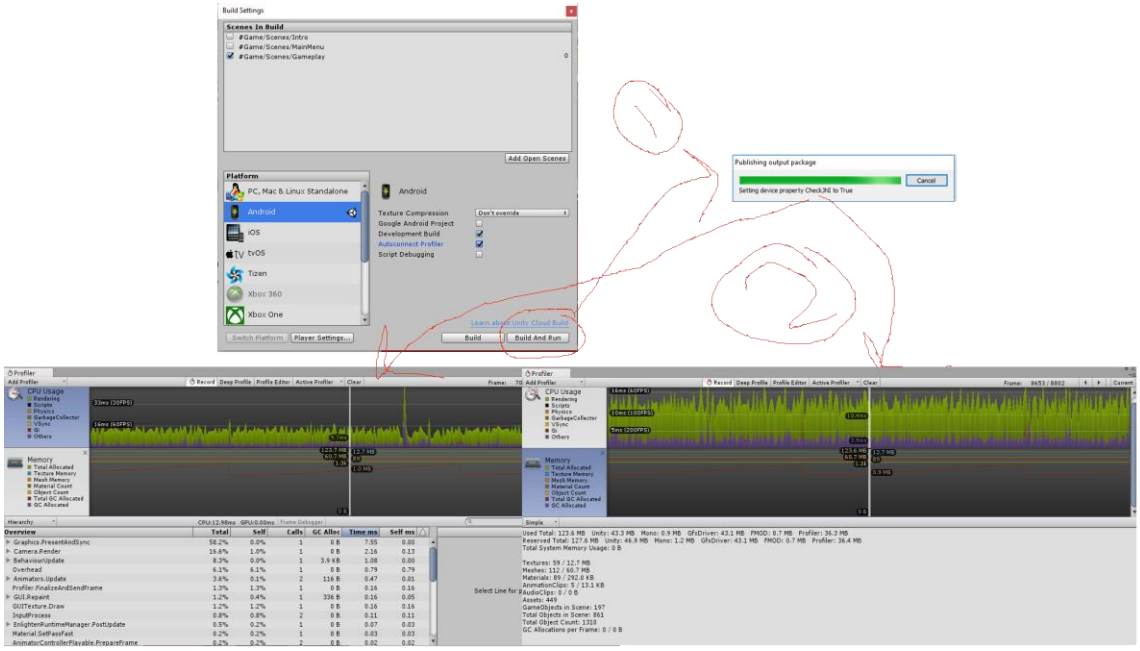


Figure 11.1 (Profiler)

After connecting your phone to your laptop or PC following the short tutorial on (Figure 11.1), you shall end up with a screen similar to mine. I still struggle with how I should process its data, nevertheless, the figure above demonstrates the memory and CPU consumption of the game in its presumable worst case which is with the 26 models on screen and player is moving them around. Even though this scenario is hypothetical and will never happen (maximum number of shapes on screen is 12 at a time) it's still great trying to break the game and test the logic of it to its best.

In the end, I agree that this testing mechanism isn't enough to experiment with entire application's crashing or bugs issues but due to prototype deadline submission, this will do just fine.

CHAPTER 12

CONCLUSION

A prototype has been built regarding this project to show the power of the technology with a consistent design that is going to be carried out from the first phase of the final year project to this (Second and final) phase, all the possible analysis on the given title have been performed.

Creating this game was fun and often lot challenging, this is my first big game project in the industry and it taught me a lot about constructing a solid game logic based on ideas and carry them out to the world using the latest technology our race has to offer.

When I started building the game 5 months ago, I was hesitant and not quite sure of how the final product will look like judging my little experience in the field but, as soon as I started using 3d modeling software and digital art tools I knew it was a learning process so, I promised myself not to stop learning, fail fast and always dream big.

Programming wise, I learned so much with Unity3D and ‘C#’, going beyond programming the game logic to building my own tools using unity editor classes helped me make my code more dynamic and versatile, which shined the light on skills I didn’t think I had.

In the end, I pray that this game shows success and attract players from different genres.

References

[1]*IN2AR, A Cross-platform Augmented Reality Engine*. (2016). *Beyondreality.nl*. Retrieved 15 September 2016, from https://www.beyondreality.nl/in2ar/

[2]*Vuforia Developer Portal* |. (2016). *Developer.vuforia.com*. Retrieved 15 September 2016, from https://developer.vuforia.com/home-page

[3]*Open Source Augmented Reality SDK | ARToolKit.org*. (2016). *Artoolkit.org*. Retrieved 15 September 2016, from https://artoolkit.org

[4]*Coherent Labs » Unity 3D Facebook integration with Coherent UI (tutorial)*. (2013). *Coherent-labs.com*. Retrieved 15 September 2016, from http://coherent-labs.com/blog/unity-3d-facebook-integration-with-coherent-ui-tutorial/

Appendix