Contents

[Chapter 1 – Introduction 2](#_Toc443903450)

[1.1 Introduction 2](#_Toc443903451)

[1.2 Background 2](#_Toc443903452)

[1.3 Project Specification 2](#_Toc443903453)

[1.4 Conclusion 3](#_Toc443903454)

[Chapter 2 – Similar Systems 3](#_Toc443903455)

[2.1 Introduction 3](#_Toc443903456)

[2.2 Overview of Game Genres 3](#_Toc443903457)

[2.3 Game Concept 3](#_Toc443903458)

[2.4 Similar Systems 3](#_Toc443903459)

[2.5 Conclusion 3](#_Toc443903460)

[Chapter 3 – Technology Research 3](#_Toc443903461)

[3.1 Introduction 3](#_Toc443903462)

[3.2 Game Summary 3](#_Toc443903463)

[3.3 Development Methodology 3](#_Toc443903464)

[3.4 Game Engine Research 3](#_Toc443903465)

[3.4.1 Urho3D 3](#_Toc443903466)

[3.4.2 Allegro 5 3](#_Toc443903467)

[3.4.3 Unity 4](#_Toc443903468)

[3.5 Game Engine Comparison 4](#_Toc443903469)

[3.7 Conclusion 4](#_Toc443903470)

[Chapter 4 – Design 5](#_Toc443903471)

[4.1 Introduction 5](#_Toc443903472)

[4.2 Storyboard 5](#_Toc443903473)

[4.3 Use Case 5](#_Toc443903474)

[4.4 Development Methodology 5](#_Toc443903475)

[4.5 Case Design 5](#_Toc443903476)

[4.6 Conclusion 5](#_Toc443903477)

[Chapter 5 – Implementation 5](#_Toc443903478)

[5.1 Introduction 5](#_Toc443903479)

[5.2 Game Implementation 5](#_Toc443903480)

[5.3 Controller Drivers Implementation 5](#_Toc443903481)

[5.4 Conclusion 5](#_Toc443903482)

[Chapter 6 - Testing & Issues 5](#_Toc443903483)

[6.1 Introduction 5](#_Toc443903484)

[6.2 Game Testing 5](#_Toc443903485)

[6.3 Controller Testing 5](#_Toc443903486)

[6.4 Test Evaluation 5](#_Toc443903487)

[6.5 Logged Issues 5](#_Toc443903488)

[6.6 Conclusion 5](#_Toc443903489)

[Chapter 7 – Project Evaluation & Conclusion 6](#_Toc443903490)

[7.1 Introduction 6](#_Toc443903491)

[7.2 Project Evaluation 6](#_Toc443903492)

[7.3 Future Work 6](#_Toc443903493)

[7.4 Conclusion 6](#_Toc443903494)

[Bibliography 6](#_Toc443903495)

# Chapter 1 – Introduction

## Introduction

*TBD*

Handheld games consoles have been around since the 1970s and 80s. Throughout the years, these consoles have been extremely popular, however they are often rather expensive and often have development for them restricted.   
My intention for this project is to demonstrate using a Raspberry Pi, that it is very possible a relatively cheap handheld console while making it an open environment for anyone to develop software for it. I shall be developing a game to run on the system to allow user to test the controls of the system.

## Background

*TBD*

## Project Specification

My project when completed will consist of multiple components. These is a physical device, drivers that communicate with the controller and the OS, and a game which is used to test the physical hardware.

The physical device is the core of the project, and will be the combination of the Raspberry Pi, the touchscreen, the controller and a power supply all within a case. The Raspberry Pi is the main computer of it, and the user will use the touchscreen to interact with the OS system. The main functions of the controller will be to control a game and other software designed to use it. The console has a fairly simple design, as it consists of a touch screen and a directional pad, four face buttons, 2 shoulder buttons and a start and select button.

The drivers will be how the controller input will be converted into a format that the system and game can understand. It will read in the input according to how the protocol dictates, parse the data it receives and then output it to the OS and any other programs that are listening for it.

The game will be a basic 2D platformer that is designed to test the latency and accuracy of the controller and drivers.

## Conclusion

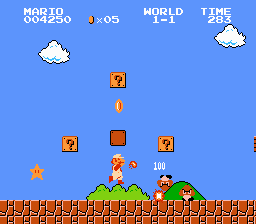
*TBD*

# Chapter 2 – Similar Systems

## 2.1 Introduction

## 2.2 Overview of Game Genres

### 2D and 3D perspective

A 2D game is a game that is presented to the user only using 2 dimensions, typically in a top down or side scrolling view. The main advantage of 2D games is that they are rarely performance intensive for the system, however that will depend on the complexity of the game. As well as that, the level of complexity on developing games in 2D is much simpler as it requires less complex maths. The main limitation of 2D games is the movement range, as a player can only ever move in the four cardinal directions: up, down, left and right [7]. 

*Figure 2.4 Side Scrolling view [8]*

A 3D game is a game that is presented to the user using the x, y and z axes, typically in a first of third person perspective. The main advantage to using 3D is that you provide the player with a much greater range of movement. The main issue with 3D is that it is much more difficult to work with, as it has a much greater level of complexity then its 2D counterpart [7].

There is much greater support for 2D over 3D on the Raspberry Pi, as it the Raspberry Pi is a much weaker system and would be harder to run 3D games without encountering performance issues.



*Figure 2.5 First-Person View [9]*

### Genres

Platformer is a sub-genre of the Action genre. They are most often categorised as the player moving from one location to another through a dangerous environment while trying to complete puzzles or avoiding various obstacles such as holes and enemies [10].   
Although they can exist in both 2D and 3D games, however they are more commonly 2D games. Popular examples of such games are the Mega Man, Super Mario Bros and Sonic the Hedgehog series.

Role-Playing Games (RPGs) is one of the sub-genres of the Adventure genre. This type of game draws inspiration from classical table top games such as Dungeons & Dragons. The typically involve a large quest line that the player is to follow, some form of equipment management system and a levelling system. In more recent years, this genre has evolved to inherently include open world aspects, which allow users to choose how quickly or slowly they progress through the story. Popular 2D games series in this genre include the Final Fantasy and Dragon Quest series [10].

Tactical Role-Playing Games are a hybrid of the Role-Playing Game and the Strategy genres. These games are similar to RPGs in the sense that you control a character or a group of characters, however they incorporate more of a strategic gameplay [10]. The more well-known games of this genre, such as Final Fantasy Tactics and Fire Emblem, use an isometric grid based combat system.

## 2.3 Game Concept

Super Mario Bros is a highly acclaimed game created by Nintendo for the NES. It is a side-scrolling platformer game which consists of the user travelling across a 2D world to defeat the final boss. There is a wide variety of the types of levels that the player must traverse through which include over world, underwater and castle levels. The user must either jump on top or jump over enemies to pass them. While traversing these levels, the player is able to find power ups which help them, like the fire flower which allows the user to shoot a projectile to take out most enemies from a range [8].

Mega Man X is another highly acclaimed game created by Capcom for the SNES. This game is considered a huge success for its well-designed controls and layout. The game is a side-scrolling platformer which consists of 8 highly different levels that can be completed in any order. Once those 8 levels are completed, there is one final level that the user must go through. The combat system is the player shooting projectiles at enemies, similar to the Fire Flower in Super Mario Bros. Each of the bosses of the 8 levels have a weakness to a specific weapon that can be unlocked by killing another one of the bosses, which makes it so that although there is a set path the player is supposed to follow, they are not required to [23].

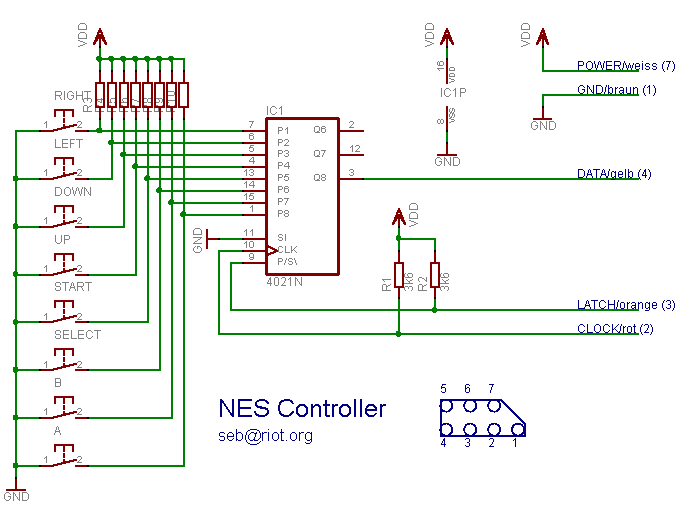
## 2.4 Similar Systems

The Game Boy line is a series of handheld consoles created by Nintendo. This line of consoles ran from 1989 to 2005 with the series being discontinued in favour of the Nintendo DS. The latest in this line of consoles was the Game Boy Micro. In terms of the functional hardware, this console was almost identical to its predecessor with the same button layout and a similar under laying hardware. With this system the intention behind it was not necessarily to improve on the performance of the series, but to continue working on improving the comfort of their systems. From a developer stand point, Nintendo consoles are traditionally very difficult to begin developing for as the regulations for getting a developer kit are very strict and very expensive [1].

The Ouya is a now discontinued game console that was released in 2013, that was designed to be an open source platform for developers to make games without needing any specialist equipment or licenses. It also intended to provide free games and software to some extent to all users, either through demos or by supporting in game purchases. [2].

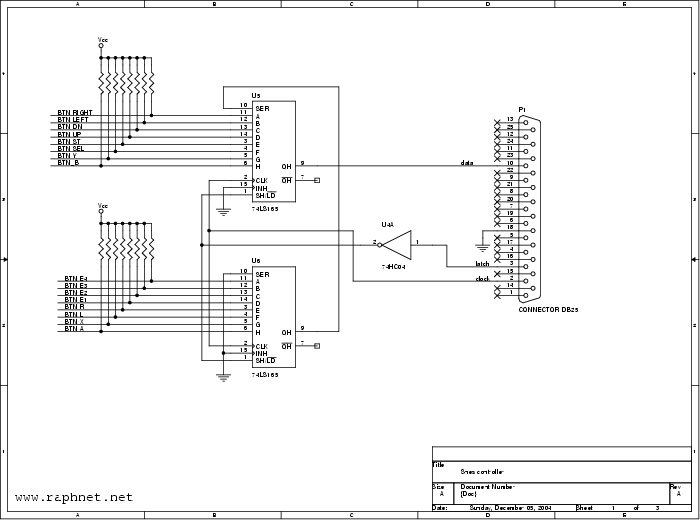
### Different Controller Styles

The NES used a controller that consisted of 8 buttons. Internally it uses an 8-bit shift register to convert the 8 inputs into the format required to be output into the proprietary connector used by NES controller [3]. See Figure 2.1.

****

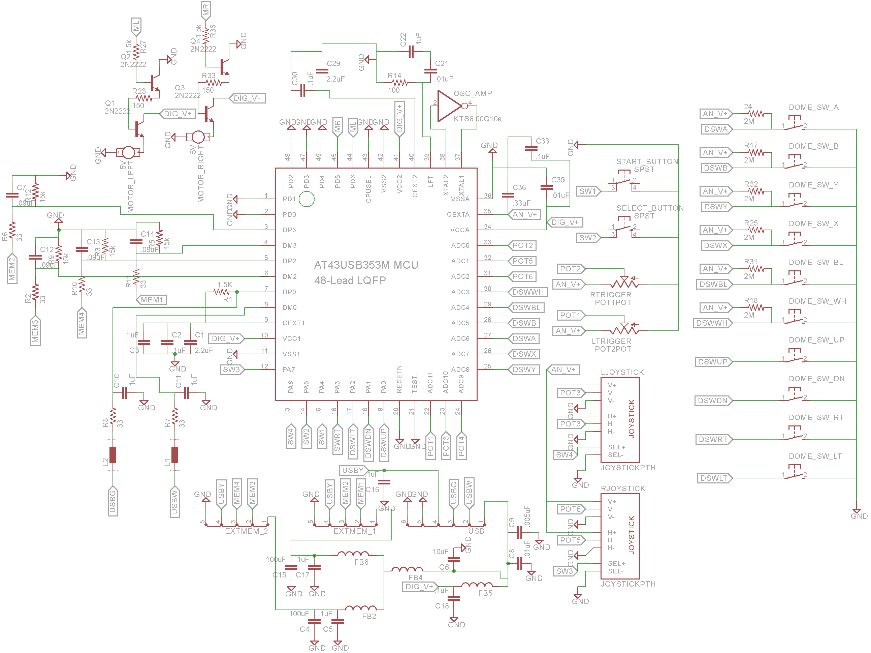
*Figure 2.1 Nintendo Entertainment System Controller Schematic [3]*

The SNES controller is an upgraded version of the NES controller. This controller uses 2 shift registers working in tandem to convert the button inputs into the format required for the console to read the input [4] [5]. See Figure 2.2.



*Figure 2.2 Super Nintendo Entertainment System Controller Schematic [5]*

The Xbox controller uses a relatively similar concept. With the buttons being fed into a device which converts the raw input from the buttons being pressed and outputs data that can be read by the console. In this case because of the greatly increased complexity due to analogue sticks, it’s using a microcontroller rather than shift registers [6]. See Figure 2.3.

  
*Figure 2.3 Xbox Controller Schematic [6]*

## 2.5 Conclusion

# Chapter 3 – Technology Research

## 3.1 Introduction

## 3.2 Game Summary

## 3.3 Development Methodology

## 3.4 Game Engine Research

Python is an interpreted programming language, which means that programs written in it are able to be run without requiring the user to compile the program into an executable format, although it is still possible to create an executable by using third party software. It is a general use programming languages, which means that it has a large number of practical uses, and has a wide variety of libraries available for it.   
As Python is an open-source language, it has a large amount of third party libraries available for it.   
For USB programming, Python has a library called PyUSB. This is an open source library written purely in Python, designed to reduce some of the issues associated with programming USB interfaces [11].  
For game development, Python has a few options available. Pythons primary game development library is PyGame, as it is incredibly easy to pick up and is very versatile [12]. Allegro is a game Engine that can be used with Python, however it is not heavily supported as it is designed for use with C and C++ [18].

C+ is a complied programming language which means that when written, the file then needs to be compiled to be able to be run. C++ is a versatile language, and can be used on a variety of platforms such as Windows and Unix-based systems and it has a large amount of support as a result [13].  
There is a specifically made library called Uspi that is designed to programming USB interfaces in C and C++. This is an open source library that is currently in development which supports a few different interfaces and different devices such as gamepads, keyboards and Ethernet controllers [14].   
For game development, C++ has access to engines such as Urho3D and Allegro. Urho3D is an open source 2D and 3D game engine implemented in C++. Urho3D is compatible with the Raspberry Pi and supports devices like Gamepads and touch screens [17]. Allegro is a 2D graphical library that is designed for use with C & C++ [18].

Java is another compiled programming language that is compatible with the Raspberry Pi, which runs inside of the Java virtual machine which is available for almost every platform. The main reason for this is so that a single compiled file can be deployed on almost any system without needing to be recompiled, as the variations in the systems are handled by the virtual machine itself.   
For USB development, there is a wrapper for Java called Libusb Java. The purpose of this is to allow the use of the Libusb libraries within Java, and is compatible with any devices that Libusb is compatible with [15].   
Lightweight Java Game Library is an open source Java library that give access to popular APIs such as OpenGL and OpenAL. This game library was used in popular games such as Minecraft by Mojang and is implemented by some game engines [16].

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | **Python** | **C++** | **Java** |
| Graphical libraries available | Allegro, PyGame | Urho3D, Allegro | Lightweight Java Game Library |
| USB libraries available | PyUSB | Uspi | Libusb |
| Relative difficulty (Due to experience etc) | Easy | Medium | Medium |
| Complied language | No | Yes | Yes |

Urho3D is an open source game engine that can be deployed on the Raspberry Pi. Urho3D can be used for both 2D and 3D game development. Games can be written in with it using C++ and has an integrated 2D physics engine, although it may be required for the user to implement their own improvements on the physics engine depending on their needs. Games created using this engine can be deployed to a multitude of platforms such Windows, Raspberry Pi, Android and iOS with little to no refactoring. Urho3D supports various input devices such as keyboards, joysticks and touch screen input [17].

Allegro is a 2D graphical library that can be deployed on the Raspberry Pi. Allegro is a library that only deals with 2D graphics, however it can be implemented alongside a 3D library such as OpenGL. Although not an engine itself, it does have some low level physics and collider implementations [18].

Unity is a hugely popular game engine. Unity supports both 2D and 3D game development, which separate physics engines dedicated to each one. It is a relatively new emergence into the market as a popular engine and has gained huge traction in the area, and is gaining large amounts of external support, now being used as an official environment for systems such as the PlayStation 4, Xbox One and Wii U [19].

## 3.5 Game Engine Comparison

For the game engine, I shall be using Urho3D for the game, as it inherently supports C++ and has a large support for various different platforms. Although it has greater cross-platform compatibility, Unity was not feasible to be used for this game as it does not support the Raspberry Pi.

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements** | **Unity** | **Urho3D** | **Allegro** |
| Compatible with Raspberry Pi | No | Yes | Yes |
| Open-Source | No | Yes | Yes |
| Supported Languages | C#, Javascript | C++ | C, C++ |
| Physics Management | Yes | Yes | Yes |

*Table 3.5.1 – Game Engines Comparison*

## 3.7 Conclusion

I’m using the Raspberry Pi for this project because I felt that its power and size would be very suitable for this type of project, and has been something Ive been wanting to do for a while.

Instead of the Raspberry Pi I had considered using an ARM microprocessor, however with the time constraints of the project I did not feel comfortable using it and risking not being able to do anything else with the project past constructing it.

For my system, I will be aiming for the openness in game development like with the Ouya, and the physical feel of the Game Boy. I feel that a system with both key features will be very popular compared to some other systems on the market.   
The openness would allow any developer to pick up the system and deploy games in an easy to use way, and would allow continuous growth for the system as developers and support the growth of new developers.  
The feel of the console is integral with the usability of the system. One of the biggest short comings of the Ouya is that the controller wasn’t very highly regarded as a well-designed device, with complaints of short life and poor structural integrity [20]. The Game Boy line, after years of experience, have a very high quality to their controls.

# Chapter 4 – Design

## 4.1 Introduction

## 4.2 Storyboard

I will be implementing two basic rules that the user will need to follow, these rules are that the user has a limited number of hits before the player dies and the user will have a limited number of lives to continue a level.   
All entities will have a health bar that depletes when it is injured, either by contact with enemies or when hit by a projectile. When the health bar reaches zero on an enemy, the enemy gets deleted. When the health bar reaches zero on the player, it triggers a game loss event, and reduces their lives by 1.  
The user will have a limited number of lives that they can use. Through the level, you will be able to find various checkpoints. When the user loses a life and their life count is not equal to zero, they will be able to respawn at one of the checkpoints. When their life count is equal to zero and they lose a life, they will get a game over and be forced to restart the level from the beginning. This life count can be replenished with items found in the game, similar to the 1-UP mushrooms found in the Super Mario Bros series.

The game mechanics are mostly fairly simple, however they can get complex when combined together. The player will be able to move left and right, jump, shoot a projectile and collect pickups. The pickups will allow the user to shoot different a different weapon to their usual weapon. As well as this, the player will be able to interact with the world using the touch screen by moving platforms around in the environment. The player will always be under the influence of gravity, and rarely able to leave its influence. One exception is the wall jumping mechanic, where the user is able to slide down a wall slowly and propel themselves off of the wall.

The various enemies in the game will require some form of AI to function. Although they will vary depending on each enemy, most of them will share certain key features. For example, all enemies in the game will share some form of pattern in their movements. For example, there will be enemies that will jump, and at the peak of the jump will shoot out a projectile. This is designed to allow the players predict the actions of enemies and remove some unnecessary unpredictability from the game.

## 4.3 Use Case

## The system consists of multiple components as seen in Figure 5.2. The game controller will be connected to the Raspberry Pi through USB. The input of this controller is fed to the system by the controller drives written in Python. The output of the drivers is passed from the system to the game which will be used to control the game which is written in C++ using the Urho3D engine. C:\Users\Chris\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMAG0020.jpg Figure 5.2 – Relational Diagram for the system

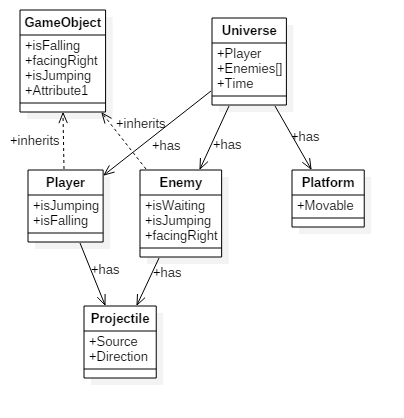
As can be seen from Figure 5.3, the game will revolve around 4 main classes; the Universe class, the Player Class, the Enemy Class and the Projectile Class. The Universe manages terrain generation, the player and enemy classes. The Player and Enemy Classes are similar as they both inherit from GameObject, varying only in how the movement is done. The projectile class is the same for both the enemies and player, varying only in who the source is. The platform class has one main attribute, whether the user can interact with it using the touch controls. As well as this, the platform will be what the user interacts with by walking.  


Figure 5.3 - Class Diagram

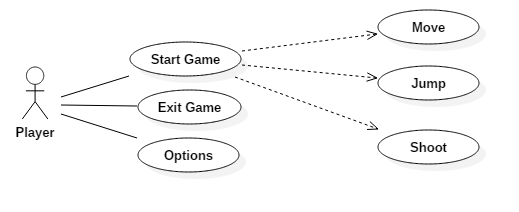
As can be seen from Figure 5.4, there is only one user that will interact with the system at a time. The player will initially be able to begin the game, configure options such as volume, and exit the game. After the user begins the game, they will be able to move, jump and shoot their weapon.  


Figure 5.4 - Use Case Diagram

## 4.4 Development Methodology

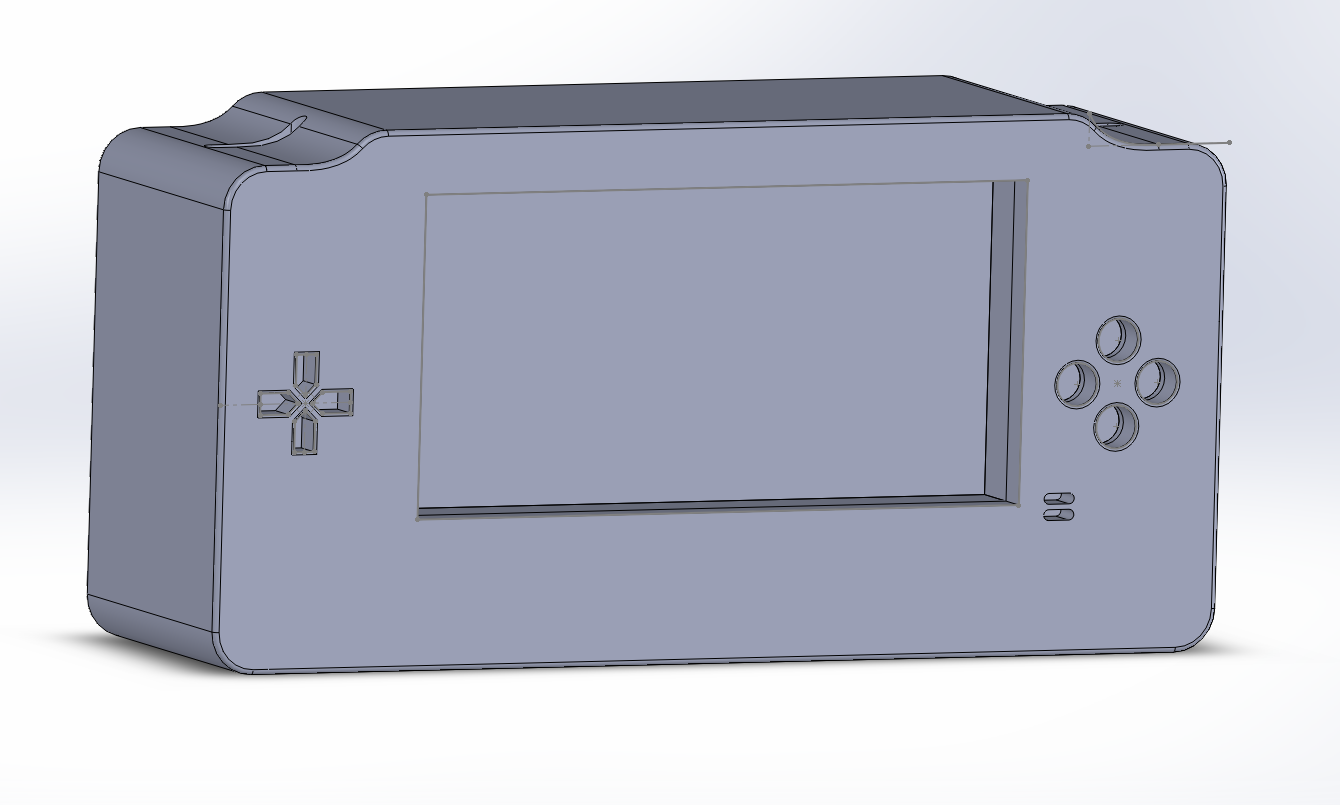
For designing the game, I used the Ernest Adams Design Methodology. This Methodology includes these following elements as a guideline for designers to follow: Define game rules, Define game mechanics and Define game AI.

## 4.5 Case Design

There are some usability requirements that needed to be taken into careful consideration when designing the casing of this consoles. Comfort is a huge concern for this device, as consoles are typically used for extended periods of time. As a result, I have designed the console to have slightly curved corners so as to prevent the user from injuring their hands on a pointed edge.

Another requirement that is important to take into consideration is that all of the main buttons should be usable from a single holding position. This is to allow users to play games comfortably without having any issues.

The final main requirement for the console is that the weight should be supported evenly enough that the user is able to comfortably able to hold it in one hand. As the console makes use of a touch screen for navigation throughout the system, it is important that the user is able to hold the system in one hand so as to allow the user to navigate without having to place the console on a flat surface.

  
Figure 5.1 - 3D render of case design

## 4.7 Prototyping

## 4.6 Conclusion

# Chapter 5 – Implementation

## 5.1 Introduction

## 5.2 Game Implementation

# Urho3D

* Difficult to set up, lots of time wasted with no results what so ever.
* When finally set up, incredibly difficult to use.

# Allegro 5

* Issue running on Raspberry Pi, Screen not displaying as intended
* Working on Windows Command Line
* Ways of locating error, check in Raspberry Pi VM for same issue. If issue not present, most likely an issue with the touch screen drivers
* Consider Allegro 4, issues may not exist

After having finished creating the drivers for the controller in Python, it came time to create a way of integrating the program I had written in Python into C++ so it could be used for controlling the game. After brief research into the various APIs that allowed this, I decided on using the one provided by Python itself. Once I had decided on which API to use, I needed to figure out how to use it. I learned that as it was in its current form, my Python drivers were not usable as they were not easily callable as functions. As well as this, I figured that I might need a way to store the state of my drivers, else every time I try to read the state of the controller, I would need to recreate the connection to the controller. Although it may not be a big deal on other systems, it is incredibly inefficient use of the otherwise very valuable and limited resources of the Raspberry Pi.

## 5.3 Controller Drivers Implementation

To find details about the controller, I ran a program that finds vendor and product ids of all devices plugged in. I ran it while it was plugged in, then unplugged it and ran again to find what changed. The device that disappeared is the controller  
namefinder.py

Hexes for the controller are 0x81f(Vendor ID) and 0xe401 (Product ID)

For determining what buttons correspond to which bit being fed in, I created a script that prints out the data being put in and from that I can decide which button is which by pressing the buttons. Given the player is not restricted to pressing only 1 button, it is important to make sure that all combinations of buttons are accounted for. Through testing, I’ve learned that only buttons that share the same bits will actually interfere with each other. Because there are many buttons that share bits, it’s important to make sure all combinations of buttons is accounted for. A, B, X and Y (and L, R, Start and Select) all add up to the combination value minus the beginning value (ie AB = 47+79-15).

buttontesting.py

Bits and stuff:  
Nothing pressed = [127, 127, 128, 128, 128, 15, 0, 0]  
A = [127, 127, 128, 128, 128, 47, 0, 0]  
B = [127, 127, 128, 128, 128, 79, 0, 0]  
X = [127, 127, 128, 128, 128, 31, 0, 0]  
Y = [127, 127, 128, 128, 128, 143, 0, 0]  
L = [127, 127, 128, 128, 128, 15, 1, 0]  
R = [127, 127, 128, 128, 128, 15, 2, 0]  
Up = [127, 0, 128, 128, 128, 15, 0, 0]  
Down = [127, 255, 128, 128, 128, 15, 0, 0]  
Left = [0, 255, 128, 128, 128, 15, 0, 0]  
Right = [255, 127, 128, 128, 128, 15, 0, 0]  
Start = [127, 127, 128, 128, 128, 15, 32, 0]  
Select = [127, 127, 128, 128, 128, 15, 16, 0]

Combinations:  
AB = [127, 127, 128, 128, 128, 111, 0, 0]  
BY = [127, 127, 128, 128, 128, 207, 0, 0]  
YX = [127, 127, 128, 128, 128, 159, 0, 0]  
XA = [127, 127, 128, 128, 128, 63, 0, 0]  
YA = [127, 127, 128, 128, 128, 175, 0, 0]  
XB = [127, 127, 128, 128, 128, 95, 0, 0]

ABY = [127, 127, 128, 128, 128, 239, 0, 0]  
BYX = [127, 127, 128, 128, 128, 223, 0, 0]  
YXA = [127, 127, 128, 128, 128, 291, 0, 0]  
ABX = [127, 127, 128, 128, 128, 127, 0, 0]  
ABXY = [127, 127, 128, 128, 128, 255, 0, 0]

Start Select = [127, 127, 128, 128, 128, 15, 48, 0]  
LR = [127, 127, 128, 128, 128, 15, 3, 0]  
L Start = [127, 127, 128, 128, 128, 15, 33, 0]  
R Start = [127, 127, 128, 128, 128, 15, 34, 0]  
L Select = [127, 127, 128, 128, 128, 15, 17, 0]  
R Select = [127, 127, 128, 128, 128, 15, 18, 0]

L R Start = [127, 127, 128, 128, 128, 15, 35, 0]  
L R Select = [127, 127, 128, 128, 128, 15, 19, 0]  
L Start Select = [127, 127, 128, 128, 128, 15, 49, 0]  
R Start Select = [127, 127, 128, 128, 128, 15, 50, 0]  
L R Start Select = [127, 127, 128, 128, 128, 15, 51, 0]

After having finished creating the drivers for the controller in Python, it came time to create a way of integrating the program I had written in Python into C++ so it could be used for controlling the game. After brief research into the various APIs that allowed this, I decided on using the one provided by Python itself. Once I had decided on which API to use, I needed to figure out how to use it. I learned that as it was in its current form, my Python drivers were not usable as they were not easily callable as functions. As well as this, I figured that I might need a way to store the state of my drivers, else every time I try to read the state of the controller, I would need to recreate the connection to the controller. Although it may not be a big deal on other systems, it is incredibly inefficient use of the otherwise very valuable and limited resources of the Raspberry Pi.

## 5.4 Conclusion

# Chapter 6 - Testing & Issues

## 6.1 Introduction

For this project, I shall be doing three types of testing on the game I am developing: functionality testing, compatibility testing and regression testing. In addition to this method of testing, I will have the game create a log file that will give me some information that may be useful for debugging purposes.

Functionality testing consists of using the piece of software in an attempt to discover. I shall be implementing functionality testing in two stages. Stage 1 consists of me testing the game personally to try to detect and remove some of the more obvious bugs. Stage 2 will consist of outside users testing the game to find some of the less obvious bugs.

Regression testing consists of re-testing a system after previous bugs have been fixed. This will be done similarly to functionality testing, where by I’ll have myself and external users testing the system to find new bugs that have appeared from previous bug testing.

## 6.2 Game Testing

It’s important to get large amounts of people testing the system from different viewpoints, particularly in games testing, as there are a huge amount of different combination of things that can create bugs and glitches to happen. For example, there was a bug that was discovered in Super Mario World at the beginning in 2015 that allows players to finish the game in under 3 minutes time [24].

For testing the system initially, the users I will be including will be experts in the area. Although it would be ideal to get as many testers as possible to test the system, because the system is limited to specific hardware, there will be a very small test group as it will require me to physically be present while they test it. As well as getting the feedback from the user, I also plan to have it so the game will be recorded for use by myself so I will be able to see the bugs occur without needing to be physically present.

Because of how the system will need to be tested, I shall be filling in various test cases in accordance to how the player uses the system. I feel it is important that the user is not made aware of the content of the test cases (See Table 7.1), as that knowledge could greatly influence their thoughts and actions and impede their potential ability to test the system from a different viewpoint.

## 6.3 Controller Testing

For the hardware, I will be using the game to test the system. Mostly, this just entails using the game on the system, with the hopes of detecting any issues with the Raspberry Pi, the controller or the drivers. As well as this I shall be running stress tests on the system by running the game and any other piece of software that would be logical to have running at the same time for an extended period of time to detect any potential issues.

## 6.4 Test Evaluation

## 6.5 Logged Issues

### 6.5.1 Test 1

Initial Testing of the game and controller

### 6.5.2 Test 2

### 6.5.3 Test 3

## 6.6 Conclusion

# Chapter 7 – Project Evaluation & Conclusion

## 7.1 Introduction

## 7.2 Project Evaluation

## 7.3 Future Work

Doing Stuff, in the future!

## 7.4 Conclusion

# Bibliography