Handheld Games Console

Christopher Mahon	**C12380621**	**Supervisor: David Carroll**

Contents

1. Project statement

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.

2. What research has been done and what are the outputs?

Background research

Similar Systems (Hardware)

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.

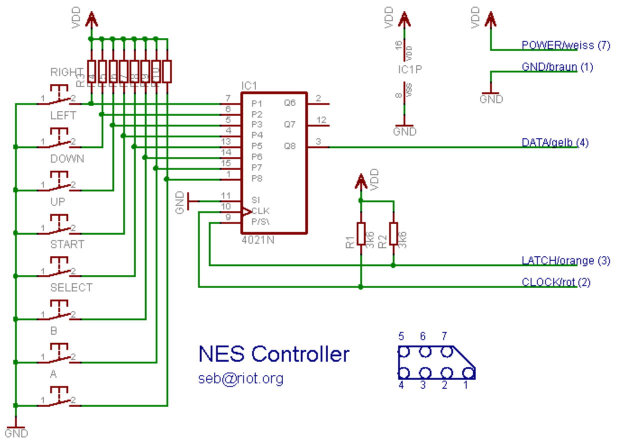

Figure 2.1 Nintendo Entertainment System Controller Schematic [3]

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.

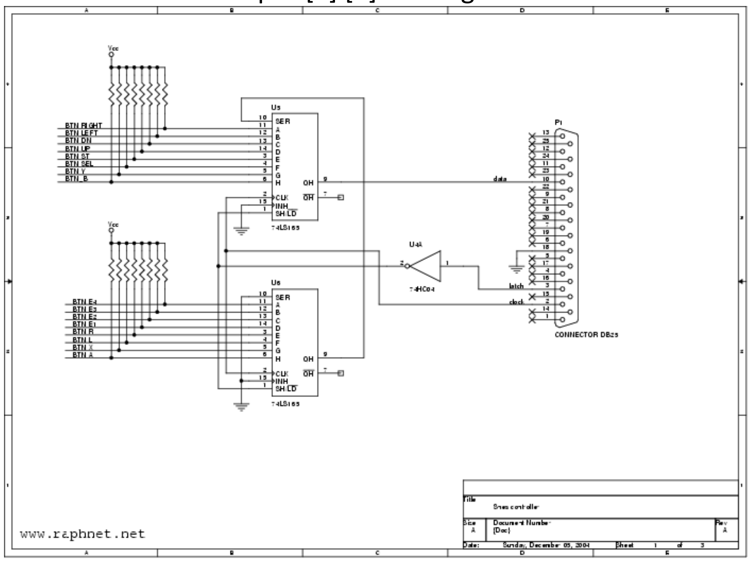

Figure 2.2 Super Nintendo Entertainment System Controller Schematic [5]

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.

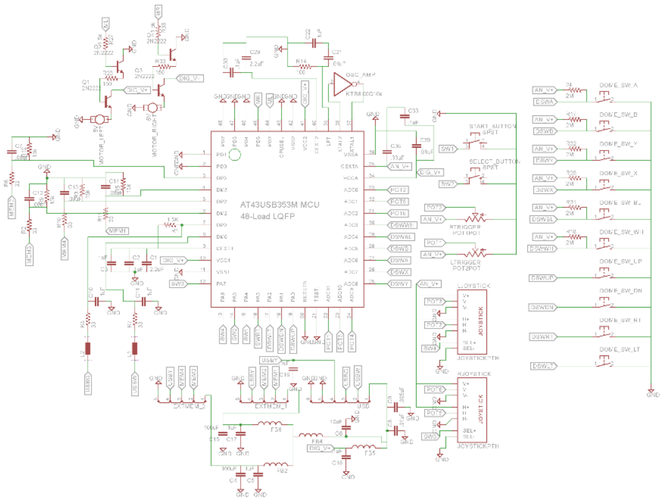

Figure 2.3 Xbox Controller Schematic [6]

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].

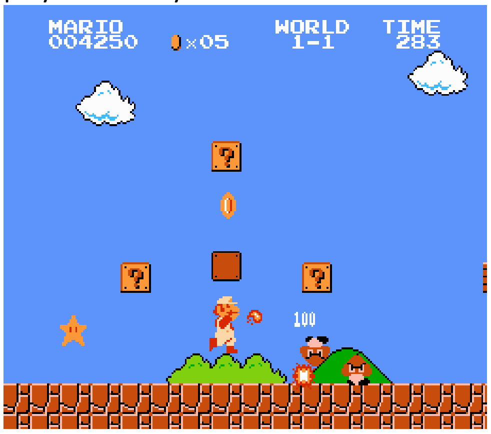

Figure 2.4 Side Scrolling view [8]

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.

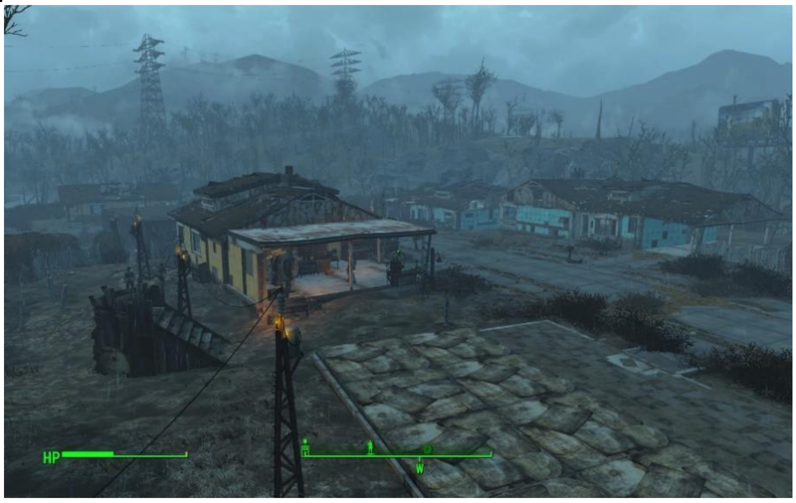

Figure 2.5 First-Person View [9]

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.

Similar Systems (Software)

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].

Alternative existing solutions to the problem you are solving

Two of the closest alternate solutions to my project are the eNcade and the GameKid. Both of these systems are self-contained that use the Raspberry Pi as the main computer of the device that are designed for playing games. They also have built in controllers to allow the user to interact with games and the OS without any additional peripherals [21] [22].
The main differences between these systems and my system is that although it is possible to develop new games for the eNcade and GameKid, the main purpose for the other systems is that they are for emulation of older consoles. As well as that, my system is the only one that has a built in touch screen for easy navigation of the OS. The main advantage of the other systems over mine is the lack of a custom OS. Although it would be optimal to have a custom OS, it is not a huge disadvantage.

Requirements	**eNcade**	**GameKid**	**My Console**
Self-Contained System	Yes	Yes	Yes
Integrated Controller	Yes	Yes	Yes
Touch Screen	No	No	Yes
Custom OS	Yes	Yes	Yes
Network Capabilities	Yes	No	Yes

Table 2.1 – Existing Solutions comparison

Technologies researched

Possible Languages

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
Cross-Platform	Yes	No	Yes

Table 2.2 – Language Comparison

Game Engine

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].

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 2.3 – Game Engines Comparison

Resultant findings/requirements

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.

As previously talked about, the controller is an important part to the system. Because of the importance of the controller, I looked into various layouts of currently existing controllers to decide on a strong layout. After researching and testing the different controller types, I decided on using a layout similar to the SNES. The SNES controller has what feels like a perfect balance of the number of buttons to how they need to be mapped. With the number of buttons, it should not be very common that there will be much requirement to have button combinations for actions to be used.

To test the system, I will be creating a game that will hopefully best utilise the hardware I have available. This will be a 2D game, as there is better support for it on the Raspberry Pi. Because of the power of the Raspberry Pi, it is much easier to create a game in 2D that performs well than it would be to create a 3D game. As well as this, 3D games benefit greatly from having access to analogue sticks to control the camera and move the player. Although they are not essential, they are still a large enough obstacle that would prevent me from creating a 3D game for the Raspberry Pi.
As well as the perspective, I looked into which genre would best test the controls of the Raspberry Pi. Although there are hundreds of genres and sub-genres I could have considered, I focused mostly on three genres; platformers, Role-Playing Games, and Tactical Role-Playing games as these were genres I was most familiar with. After looking at how each of these genres worked, I decided on using a platformer to test the system as they are typically done in real time, which better checks the responsiveness of the controller and tests both the physical design of the hardware and the drivers for it.

As I have two pieces of software that need to be written, I've looked into multiple languages from the two perspectives of USB programming and game development. Through my research, I have decided that I will be using C++ and Python for these components. I felt that Java was not suitable for my purposes, as it runs within a virtual machine and was worried that it would suffer from serious performance issues as a result.
I shall be using Python to create the USB controller drivers, as it the official language of the Raspberry Pi, I have a large amount of experience with it, and has some good support for USB programming. As well as this, it is a light weight language which will lend itself to a high latency between the system and the buttons.
For the game, I chose to use C++ as the language for game development. I have chosen it as my language for a few reasons. My main reason is that is a compiled language, which means that once the program has compiled, it is running on machine code. This is ideal as it allows the game to run as fast as possible, as performance is a huge consideration in game development.

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.

Bibliography (research sources)

1. Sarrel, M, "Game Boy Micro", http://uk.pcmag.com/game-boy-micro/26373/review/game-boy-micro, 2005, (last accessed 7 December 2015)
2. Ouya Inc., "About Ouya", https://www.ouya.tv/about/ , 2015, (last accessed 7 December 2015)

3. Gordon, "NES Controller on the Raspberry Pi", https://projects.drogon.net/nes-controller-on-the-raspberry-pi/, 2012, (last accessed 7 December 2015)
4. Christy, J. and Game Station X, "Super NES Controller Data", http://www.gamesx.com/controldata/snesdat.htm, 1999, (last accessed 7 December 2015)
5. Assénat, R., "Arcade style controller for Snes, NES, and PC", http://www.raphnet.net/electronique/arcade_control/arcade_control_en.php , 2014, (last accessed 7 December 2015)
6. Gilmers, J., Xbox controller schematic [online image], http://courses.ece.msstate.edu/ece4723/dissect/805529378889/images/xboxControllerSchematic.png, 2013. (last accessed 7 December 2015)
7. Stack Exchange Inc, http://gamedev.stackexchange.com/questions/631/what-to-consider-when-deciding-on-2d-vs-3d-for-a-game, 2010, (last accessed 7 December 2015)
8. "Super Mario Bros", Nintendo Entertainment System, 1985, 13th of September, Nintendo, Nintendo R&D4
9. "Fallout 4", Microsoft Windows, Playstation 4, Xbox One, 2015, 10th of November, Bethesda Softworks, Bethesda Game Studios
10. Stahl, T., "Video Game Genres", http://www.thocp.net/software/games/reference/genres.htm , 2005, (last accessed 7 December 2015)
11. Walac, "PyUSB", https://walac.github.io/pyusb/ , last updated 2015, (last accessed 7 December 2015)
12. Pygame Community, PyGame, https://pygame.org , 2015, (last accessed 7 December 2015)
13. Stroustrup, B., *The C++ Programming Language Fourth Edition*, Michigan, Addison-Wesley, 2013
14. Rst, "Introducing USPi – A bare metal USB driver written in C", https://www.raspberrypi.org/forums/viewtopic.php?f=72&t=92579 , 2014, (last accessed 7 December 2015)
15. Ursgraf, "LibUSBJava", http://libusb-java.ch/ , 2015, (last accessed 7 December 2015)
16. Lightweight Java Game Library Project, "LWJGL – Lightweight Java Game Library", https://www.lwjgl.org/ , 2014, (last accessed 7 December 2015)
17. Urho3D, "Urho3D", http://urho3d.github.io/, 2015, (last accessed 7 December 2015)
18. Allegro, "Allegro", http://liballeg.org/, 2015, (last accessed 7 December 2015)
19. Unity Technologies, "Unity – Game Engine, tools and Multiplatform", https://unity3d.com/unity/ , 2015, (last accessed 7 December 2015)
20. Engadget, http://www.engadget.com/products/ouya/controller/, 2013, (last accessed 7 December 2015)
21. Nzen Mods, "Nzen Mods | eNcade", http://nzenmods.com/ , 2015, (last accessed 7 December 2015)
22. RobotLovesKitty, "Gamekid", http://robotloveskitty.com/gamekid/ , 2015, (last accessed 7 December 2015)
23. "Mega Man X", Super Nintendo Entertainment System, 1993, 17th of December, Capcom, Capcom

24. “Super Mario World -- Credits Warp in 3:07.2 (Former World Record)”, [online video], 2015, https://www.youtube.com/watch?v=HxFh1CJOrTU, (last accessed 8 December 2015)

3. Analysis: Describe clearly what your solution will do

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. The goal of the game is to reach the end of a

4. Approach and Methodology

For this project, two methodologies were researched as potential ways of doing this project. The first of these two is the Rapid Application Development. The main focus of this methodology is put on development of prototypes rather than focusing on planning. This methodology is emphasises the importance of adjusting requirements as the project grows, based on knowledge gleamed from previous prototypes. The value of this is that the system is forever growing with this type of methodology, and it willing to deter from the initial design if the final result requires it. The main issue with Rapid Application Development is that it requires a huge amount of commitment from the end users as they need to be heavily involved with it throughout the entire life cycle of the project, not just at specific segments. Although not impossible, much larger systems are harder to manage by using Rapid Application Development as there is much less design documentation to follow then other traditional methods.

The other methodology that was strongly considered was the Scrum methodology. Scrum is an agile software development framework, which means that it is a flexible iterative methodology. In Scrum, all work is done in sprints, which are pre-defined time limits. At the beginning of the sprint, the work being done is planned and assigned to various members. Throughout the sprint, the team have regular scrums which are used to detail where members of the sprint are doing well or are falling behind and how any issues can be resolved. At the end of each sprint, there is a review done where the team look back on the past sprint and the work that was and was not completed. Similarly to Rapid Application Development, one of the main shortcomings of Scrum is the requirement of the end user being heavily involved with the system, however they are not quite as involved. Scrum needs a high level of commitment from all team members, or else the project will often not progress as quickly as it needs to be.

I considered both of these methodologies as they both emphasise flexible requirements and both are inherently capable of dealing with unforeseen issues that appear through development. These components were important as game development is not a rigid process and may require frequent updates of requirements, depending on new information that becomes available. I decided on the Scrum methodology because of some of the requirements I have. With the fact that my project contains a large hardware element to it, I felt that I was not able to properly use the Rapid Prototyping Methodology without constant access to the machines required. As well, I felt that Scrum fit better with treating each weekly meeting with the supervisors as a Sprint, which helped to focus my workflow.

Because of the nature of Scrum, my plan is to deliver small deliverable goals regularly, so it's much easier for myself to gauge my progress and figure out what areas need more work.

If there is a situation where time begins running short, I have worked out what priorities certain components have. My main priority tasks are getting the controller drives and the case finished as they are integral to the project. The game component I am leaving it as a medium priority component as although it is an important component of the system, its purpose is to test the function of the console. The lowest priority component of the project is wiring the controller. Although it may seem like a high priority component, I have decided that it is not as important as some other segments as the same effect can be easily created by using a pre-made controller and it is a segment of the project that will require a very specific balance to be met.

Task	**Priority**
Game	Medium
Controller Drivers	High
Casing	High
Creating the Controller	Low

Table 4.1 – Task Priority

5. 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.

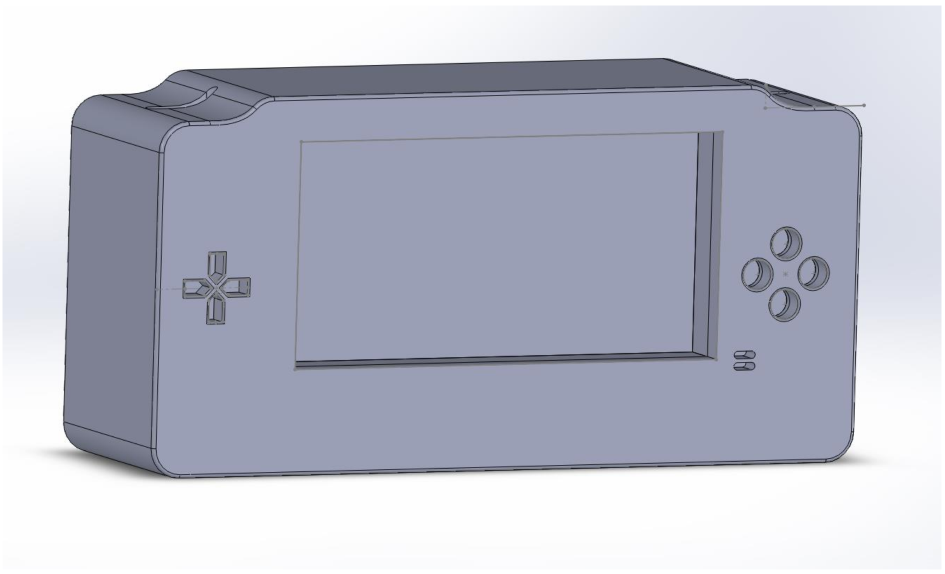
Figure 5.1 - 3D render of case design

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.

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.

Design diagrams:

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.

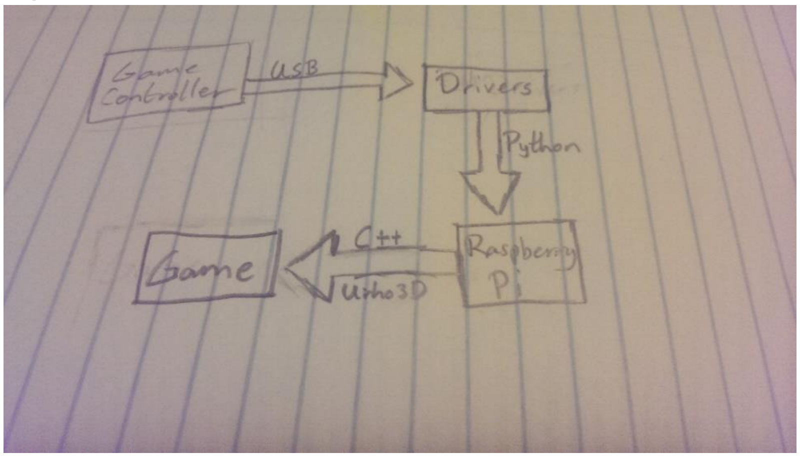

Figure 5.2 – Relational Diagram for the system

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.

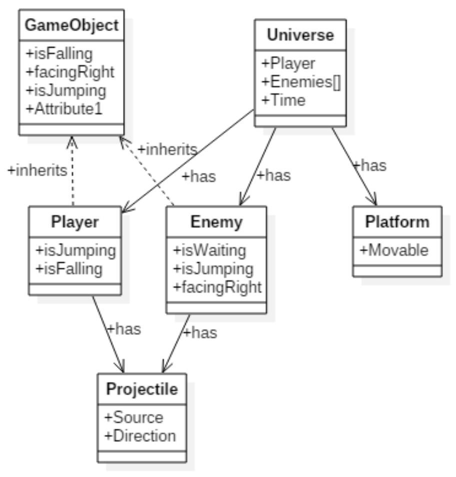

Figure 5.3 - Class Diagram

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.

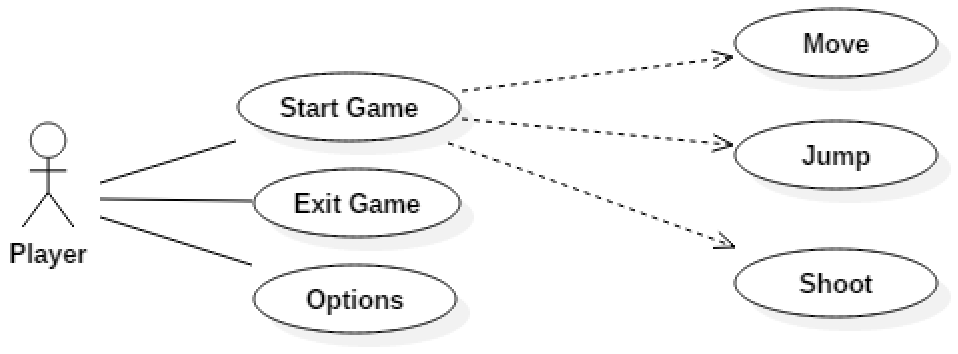

Figure 5.4 - Use Case Diagram

6. Prototyping and Development

Prototyping in this project will initially done in Unity as the hardware required was not available when prototyping began. I chose Unity over other environments as it is able to handle a large amount of the internal issues, allowing me to create a more encompassing prototype while allowing me to familiarise myself with how all the components work. While working on the prototype in Unity, I also began work on familiarising myself with C++ so as to reduce any potential issues with syntax.

My game prototype consists of a simple scene with the player in an environment they can move around in and an enemy that the player can interact with. The enemy has a basic AI that tells it to jump, shoot a bullet at the peak of its jump, and then drop back down and wait a pre-determined amount of time until it repeats the cycle. When these projectiles hit the player, it reduces their life counter by 1. When the player hits the enemy with a bullet, it is deleted and a new enemy is replaced, so as to allow more interaction testing. When the player hits into the enemy, the player is knocked back and has one health reduced from their health counter.

My case prototype is a 3D CAD drawing done in Solidworks, a specialist software used for visualising technical drawings in a 3D space. The case is a visualisation of what the ideal design would be, with slots for the d-pad and the four face buttons on the front of the case, on either side of the screen. Just below the face buttons are the menu buttons and at the top of the case will be the trigger buttons. As weight will be a big factor, it would be idea to have as many components as central as possible, so ideally it will be possible to have the Raspberry Pi right behind the screen.

7. Testing

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.

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.

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.

Test Case	Expected Result	Actual Result	Pass or Fail
Pressing the Jump Button	The player avatar will move up for a brief amount of time		
Pressing left movement button	The player avatar will move left		
Collide with enemy entity	The player avatar will lose 1 health, and move backwards		
Pressing ESC button	The game will exit		
Shooting enemy entity with projectile	The enemy entity should disappear		

Table 4.1 – Sample Test Cases

8. Issues and risks

The main issue that I have come across so far is not having access to a Raspberry Pi or the other hardware I need, which meant that my prototype will not actually run on my intended end system. Because of this, it has slightly pushed back deadlines of my original plan and means that when I get access to the hardware, I will need to re-write what I have already done for my prototype.

Because I don't have access to the controller, Ive had a few concerns regarding the timing for getting the project completed. According to the Gantt chart that I created at the start of the year, I intended on having the controller set up with drives at

Another significant problem I have ran into is the implementation of movement within the game. At first, I figured that it would not be too difficult to implement as it was just a matter of moving the player in a direction and then letting gravity move them back to a platform. I found that this was not quite correct, as while moving upwards the player character tended to jitter a lot, meaning that gravity was working against what I was trying to do.
After this I tried applying forces to control the movement of the entities. Although this did remove the jitter from upward movement, I had another issue with trying to limit the top speed of movement. Although I was able to set the top speed along either axis, I was not able to have it properly limit the speed when doing both typically cutting off one range of movement when another was already at top speed, i.e. the player wasn't able to jump and move in either direction at the same time. As well as this, the movement was not as sharp as it needed to be, causing the character to be very difficult to control.
My final solution to this issue was to re-implement the movement scheme I started with however just remove the gravity element to it and program all down movement manually. This has greatly improved the feel of the movement, and has reduced a huge number of issues I encountered. It has however increased the number of rules and constraints that need to be implemented, as I need to constantly check when the player it touching a surface and force them downwards when they aren't.

9. Plan and future work

As it is dependent on the arrival of hardware, the dates are subject to change. However ideally, the remaining components will be finished as follows. The Controller and the drivers will be finished by the 2nd of February, the Console Case will be ready by the 23rd of February and the full game will be finished by the 29th of March.

Once the prototype has been ported over to Urho3D as it currently exists, my main focus will be on getting a functional menu system. After this, I shall begin work on getting the levels to be generated procedurally. This will be accomplished by creating potential segments and then feeding them into a random number generator and getting a combination of these as the level. Once I get the level generation is done, I shall be creating various different enemies with different types of AI, like enemies that will be able to move up and down obstacles and boss like enemies. Afterwards I will be working on power ups that the user can activate and potentially become more powerful.

Once I get access to the controller, I will be able to begin working on the drivers for the controller to communicate with the Raspberry Pi. I will begin by creating a program that will read in all input gained from the controller. Once I have the data being read and stored, I shall begin looking for patterns to be able to reverse engineer the protocol and figure out which bits refer to which buttons.

Once I get access to all the hardware components, I will be able to start working on a 3D design that has accurate dimensions to what it will end up as. At the moment, the case is an over estimation of how large I imagine it will be. The main component the case is missing at the moment is the pegs that will hold the Raspberry Pi in place. Depending on how well the components will sit once all the measurements are taken, it may be necessary to consider a different case design. See Figure 5.1.

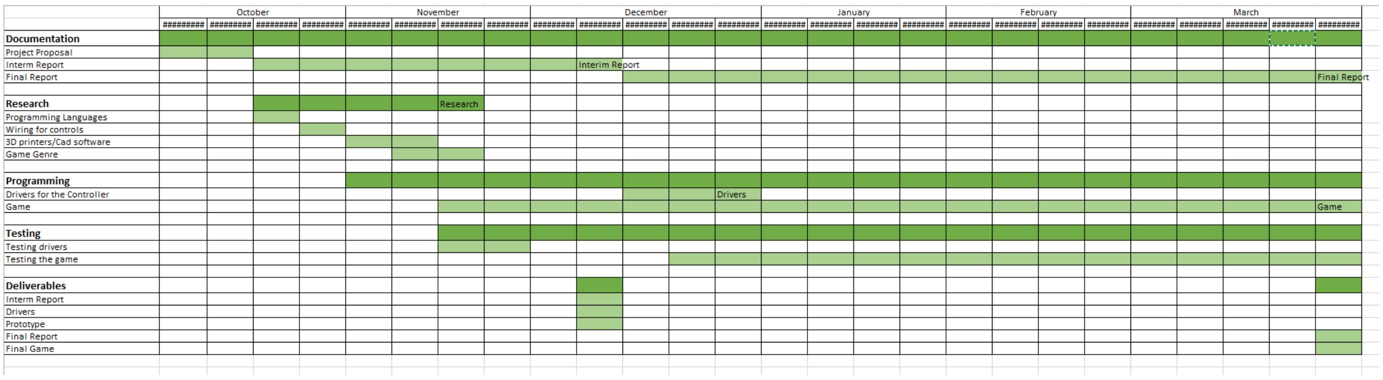

Figure 9.1 – Gantt chart

10. Conclusions

From the research conducted to this point, the system seems very plausible without too many changes having to be made to my current design and layout. The casing for the system seems to be feasible to create more or less as it stands, however the dimensions of the inner hardware such as the Raspberry Pi and the touch screen with cause it to change. The game should be feasible, as it should consist of a small number of fairly large components, and after that it will mostly consist of tweaking and small quality of life features.

Similarly to the other components, the USB programming should be feasible with my current plan for implementing it, however this has not been tested as it depends on the availability of a suitable controller.

Even just with the short time I have worked with it, I feel like I have greatly improved my understanding of the under laying hardware behind systems like my proposed system and the complexity of them and improved my ability to overcome obstacles that I come up against. I feel the issues I've encountered while trying to develop the prototype has improved my ability to identify the cause of a bug when all related variables and functions appear to be functioning correctly, and then identify multiple ways of solving bugs.