|  |  |  |
| --- | --- | --- |
| Handheld Games Console | | |
| Christopher Mahon | C12380621 | Supervisor: David Carroll |

# Project statement

# What research has been done and what are the outputs?

## Background research

1. **Similar System**   
   As a part of my research, I looked into systems that similar in one way or another to my own project. For the hardware aspect of it I found two systems that I will compare it to.   
   The first of these is the Game Boy line, in particular, I will be focusing on the Game Boy Micro as it is the latest installation. Physically the system is fairly similar to what I would like for my project, as it consists of mostly the same layout of buttons and the screen. The biggest difference would be the openness of the system. Traditionally Nintendo consoles are very difficult for indie developers to create games for, as licenses were expensive and had strict regulations to apply for one.

Another console I researched is the Ouya. Although it is not a handheld console, I did felt like it was worth mentioning. The Ouya is a now discontinued 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.

References

1. **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. See Figure 2.1.

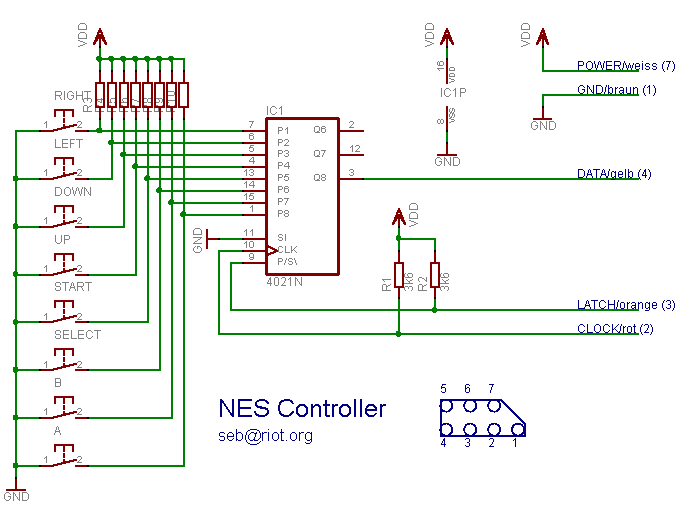
****

Figure 2.1 Nintendo Entertainment System Controller Schematic [1]

The SNES controller is an upgraded version of the NES controller. This controller uses 2 shift registers working in tandem to convert the

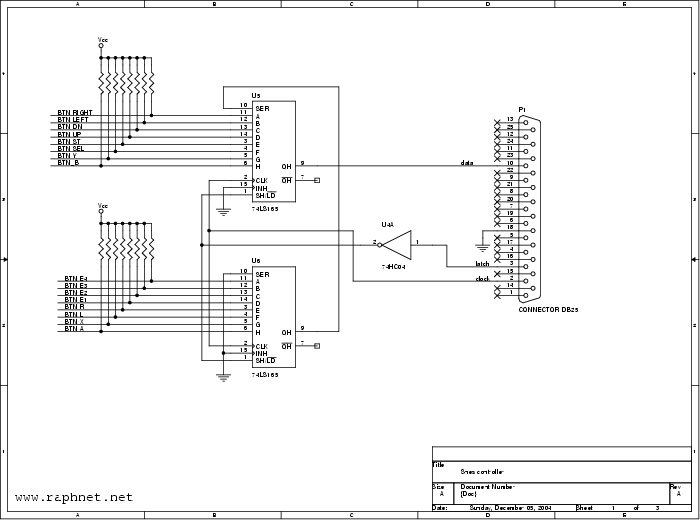
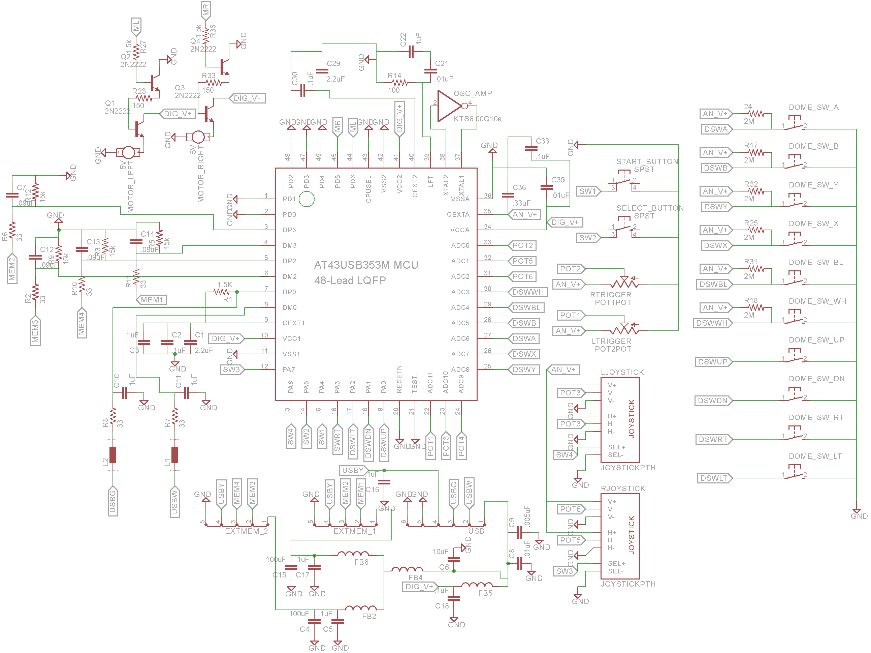
button inputs into the format required for the console to read the input [2]. See Figure 2.2. 

Figure 2.2 Super Nintendo Entertainment System Controller Schematic [3]

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 actually using a microcontroller rather than shift registers. See Figure 2.3.

  
Figure 2.3 Xbox Controller Schematic [4]

1. **2D and 3D perspective**

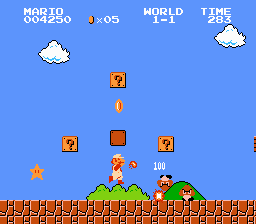
A 2D game is a game that is presented to the user only using 2 dimensions, typically in a top down or side scrolling view. The main advantage of 2D games is that they are rarely performance intensive for the system, however that will depend on the complexity of the game. As well as that, the level of complexity on developing games in 2D is much simpler as it requires less complex maths [5]. 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. 

Figure 2.4 Side Scrolling view [6]

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.

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. 

Figure 2.5 First-Person View [7]

1. **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 [8].   
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 they user 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

Role-Playing Game (RPG):

• Draws inspiration from table top games like D&D

• Player typically controls 1 or more characters through an environment

2D Examples: Final Fantasy, Golden Sun, Dragon Quest

Tactical RPGS:

• Combination of RPG and Strategy games

• Similar to RPGs as you controller a group of characters

• Incorporates strategic gameplay

2D Examples: Fire Emblem, Final Fantasy Tactics

1. **Similar Systems (Software)**

Chosen for their quality

Super Mario Bros

Similar:

* Platformer
* Side scroller
* Power Ups

Differences:

* Combat

Mega Man X

Similar:

* Platformer
* Side scroller
* Combat system

Differences:

* Limited Ammo for bonus
* Procedural Levels
* Lack of Elemental Weapons

## Alternative existing solutions to the problem you are solving eNcade, GameKid

## 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 [11]. This is an open source library written purely in Python, designed to reduce some of the issues associated with programming USB interfaces.  
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++ [13].

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.

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

Table 2.1 – Language Comparison Table

**Game Engine**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Unity | Urho3D | Allegro |
| Cross-Platform | Yes (Windows, Linux, Mac, Android, Wii U, iOS) | Yes (Windows, Mac, Linux, Android, iOS, Rasperry Pi, HTML5) | Yes () |
| Open-Source | No | Yes | No |
| Supported Languages | C#, Javascript | C++ | C, C++, Python, Java |
| Physics Management | Yes | Yes | Yes |

## Resultant findings/requirements

I’m using the Raspberry Pi for this project for two reasons. The first of these is that I’ve never used the Raspberry Pi, and I wanted to use it for this type of project. I also wanted to create something that would highlight the power of it as a device that can be easily used to create a console.

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 support to it. 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. The Game Boy line, after decades of development, have a very high quality to their controllers and have a nice feel to it.

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, Ive 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.  
For the USB controller drives, I shall be using Python 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 with the drivers and the hardware.   
For the game, I chose to use C++ as the language for 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. Unfortunately, Unity3D was not feasible to be used for this game as it does not support the Raspberry Pi.

## Bibliography (research sources)

1. <https://projects.drogon.net/nes-controller-on-the-raspberry-pi/>
2. <http://www.gamesx.com/controldata/snesdat.htm>
3. <http://www.raphnet.net/electronique/arcade_control/arcade_control_en.php>
4. <http://courses.ece.msstate.edu/ece4723/dissect/805529378889/Analysis.html>
5. <http://gamedev.stackexchange.com/questions/631/what-to-consider-when-deciding-on-2d-vs-3d-for-a-game>
6. “Super Mario Bros”, Nintendo Entertainment System, 1985, 13th of September, Nintendo, Nintendo R&D4
7. “Fallout 4”, Microsoft Windows, Playstation 4, Xbox One, 2015, 10th of November, Bethesda Softworks, Bethesda Game Studios
8. <http://www.thocp.net/software/games/reference/genres.htm>
9. The C++ Programming Language By Bjarne Stroustrup
10. <https://walac.github.io/pyusb/>
11. <https://pygame.org>
12. <http://urho3d.github.io/>
13. <http://liballeg.org/>
14. <https://unity3d.com/unity>

# Analysis: Describe clearly what your solution will do

My project when completed will consist of multiple components. These are a physical device, drivers that communicate with the controller and the OS.

# Approach and Methodology

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

|  |
| --- |
|  |

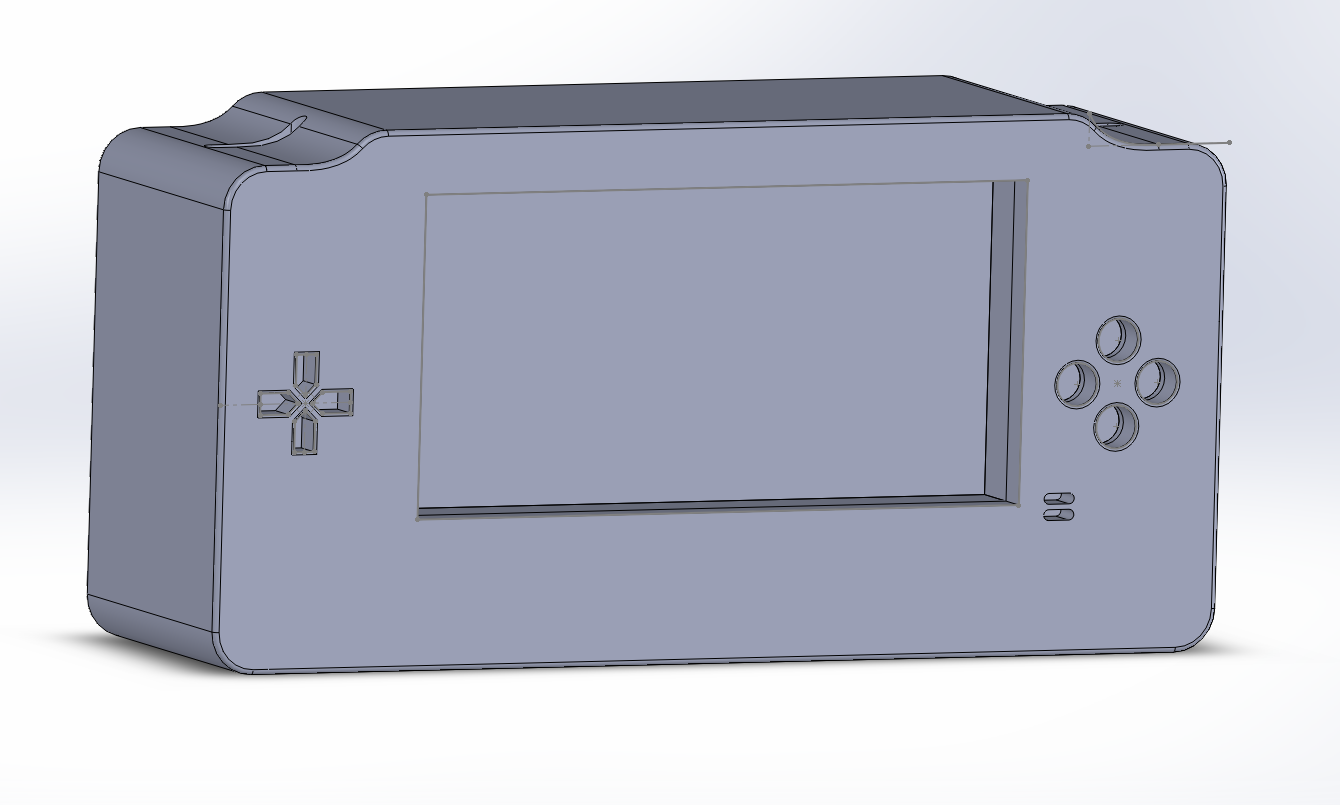
Table 4.1 – Task Priority

# 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 the 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.

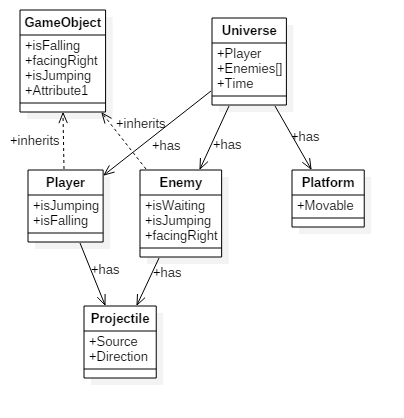
I will be implementing two basic rules that the user will need to follow, these rules are that the user has a limited number of hits before the player dies and the user will have a limited number of lives to continue a level.   
All entities will have a health bar that depletes when it is injured, either 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. 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 almost identical, 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.

   
Figure 5.3 - Class Diagram

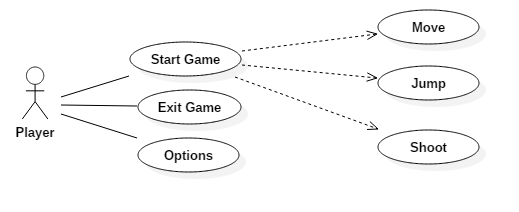
As can be seen from Figure 5.4, there is only one user that will interact with the system. The player will initially be able to begin the game, configure options such as volume, and exit the game. After the user begins the game, they will be able to move, jump and shoot the weapon.  


Figure 5.4 - Use Case Diagram

# Prototyping and Development

* Prototyping will be done in Unity due to not having hardware
* Slight movement towards familiarising myself with C++

# Testing

For this project, I shall be doing 3 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 [https://www.youtube.com/watch?v=HxFh1CJOrTU].

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.

* Who are the users

# Issues and risks

* Complexity

# Plan and future work

**Key Delieverables:** Game for R PI (TBD), Drivers(January 2016), Built console(January 2016)

# Conclusions

From the research conducted to this point, the system seems very plausible without too many changes having to be made to my initial design. The casing for the system seems to be feasible to use, although this may change depending on the dimensions of the inner hardware such as the Raspberry Pi.

* What should be here
* Identify interim conclusions viz. summary of findings thus far, plausibility of the proposed system and personal development conclusions.