

Enhancing Immersion in Virtual Reality

Final Year Project Report

DT228

BSc in Computer Science

**Peter Flanagan**

**Dr Bryan Duggan**

School of Computing

Dublin Institute of Technology

**4/4/2017**



Abstract

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Peter Flanagan

4/4/2017

Acknowledgements

Body text

1. Introduction
   1. Overview of the project and the background behind it.
   2. Project Objectives
   3. Project Challenges
   4. Structure of the document
2. Research
   1. Eg. Research related to identifying the problem that this project solves, research into solution definition
   2. An overview of the technologies evaluated and selected or rejected and the rationale behind the key decisions.
3. Design
   1. Identification of a design methodology including why it was chosen
   2. Design of each of the project components eg: the UI, Network, Project Demonstration, source code layout
   3. Clearly identifying the list of features and use cases supported within the project.
4. Architecture & Development
   1. Overview of the system architecture and a diagram to represent all of the key elements within the architecture.
   2. Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.
   3. Identify key development components;
   4. Identification/explanation of external APIs used versus own code ; List of classes of your code etc .
5. System Validation
   1. Testing
      1. What testing was performed, why it was selected and what are the key use cases within the project.
   2. Demonstration
      1. Identify what features can be demonstrated and show screen shots or reference a video online to show the project demonstration (for audience not at demo)
6. Project Plan
   1. Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated.
7. Conclusion
   1. Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future.
8. Bibliography
9. Appendix

Introduction

Overview of the project and the background behind it.

With the launch of so many Virtual Reality solutions it is clearly an area of interest and growth in the consumer electronics sector. These devices share a common issue, they can track the position and orientation of their Virtual Reality head mount and controllers, but the player cannot freely interact with the game world by moving in real world space.

Circuit Shock is a game made in unity and written in C#. It uses a custom made Arduino based controller and a Microsoft Kinect to track the players body and controller. These are rendered in-game and in game events can trigger haptic feedback in the controller. The Kinect tracking allows the player to move in real world space and see the results rendered in game. It also allows the player to dodge in game fire in real world space and have this affect the events of the game.

Project Objectives

The main objective of this project is to create an interactive VR experience for demonstration purposes. It will focus on tracking the player’s real life actions in order to increase immersion. I intend to combine the Microsoft Kinect, Oculus rift, and a custom built Arduino based controller to create a mapping of the player and their gun in-game. This mapping is used to control the player’s in-game actions and allow the player to control the game using only real life movements. The controller should have a servo motor to allow the game to send haptic feedback to the player. The game will be procedurally generated to allow the player to continue playing for as long as they are capable. There should be only one level, as multiple levels and replay ability are primarily issues for home release games, and Circuit shock has too many hardware requirements for home release to be a viable goal.

Project Challenges

This project has many challenges, from the fact that I couldn’t find a single example of a procedurally generated rail shooter in my research to the fact that I didn’t have a pc powerful enough to run a VR experience to learning how to design and build a custom hardware controller, there were many areas where I had to create my own solutions to proceed. I had to learn to solder and build hardware elements as well as design and implement software to interact with them. I had to research the requirements of VR and build a PC to match. I had to create a movement system which allows the procedurally generated level to control the players movement.

Structure of the document

<No idea what I’m supposed to say here>

Research

## Introduction

My research for this project has been very focused on achieving my initial goal of building a working original input system. I began by researching Immersion breaking issues with VR, namely dizziness and nausea. I discovered the causes and solution to these problems and worked them into my planning process. I then started looking into known solutions to motion controls and how they may or may not apply to my project. Finally I began to research games that would suit my hardware solution while avoiding the issues inherent to VR.

## Background Research

Firstly I researched the effects of VR on players, specifically the negative side effects. Virtual reality sickness is my far the most common side effect of exposure to virtual reality, and many of the causes are outside of my control, however the intensity of in game movement relative to real life movement has been linked to the intensity of the sickness felt(1). As such I decided I should a game type will little to no movement. Furthermore I looked at different controller types common in game systems. The most common types of controllers in the last 30 years and there were two standout controller types, guns and musical instruments. The prevalence of musical Instruments can be attributed to the rise and fall of rhythm games in the 2000s(2). Gun controllers were popularised in arcades in the 1980s and 90s, and fit my requirements very well. They are intuitive to use, and using a little haptic feedback the act of shooting the gun can become a visceral experience. With my controller type chosen it was time to choose a game type for my project, there are rather a number of different types of shooting game types, from fast paced arena shooters like quake to stationary light gun based games like duck hunt. Considering the nature of virtual reality sickness I quickly ruled out the fast paced genres like arena shooter, and focused on those will little or no movement. At first I considered the likes of Space Pirate Trainer VR (3)

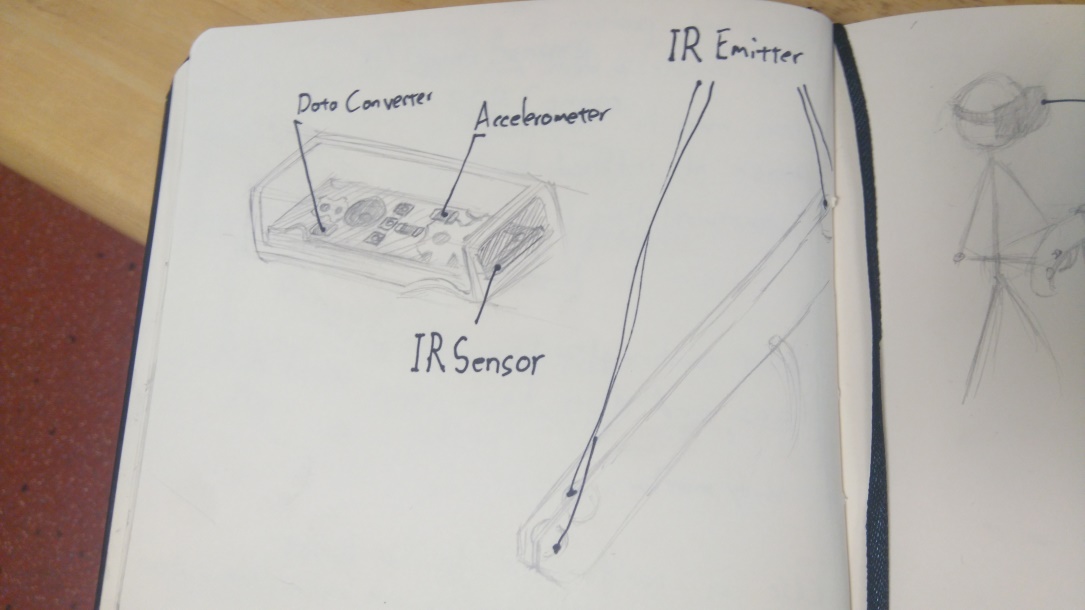
Where players were mostly stationary while they were attacked by waves of enemies until they ran out of health. At the same time I looked into older genres that have not been represented recently and I arrived at the rail shooter genre. These were games where the player slowly progressed through a level on a set track and enemies would spawn in the player’s path. It was a good fit for my requirements, limited slow movement in an intuitive game type, and it allowed the player to progress instead of just loading in harder waves of enemies. It was for this reason that I decided on the rail-shooter as my genre of choice.

## Alternative Existing Solutions to Your Problem

There are a number of solutions to the issue of motion control in VR, though these all understandably focus more on broad appeal and the ability to play a number of different game types than on a focused parallel between the real and the virtual. The HTC Vive was the first of these existing solutions I considered, it achieves position and orientation tracking using valves new lighthouse. I had the opportunity to try out the HTC Vive first hand and it was very impressed by the accuracy and intuitive nature of the position and orientation tracking. This is achieved “by flooding a room with non-visible light, Lighthouse functions as a reference point for any positional tracking device (like a VR headset or a game controller) to figure out where it is in real 3D space”(4). This approach however requires attending a training course which costs $2,975(5) , so this approach was ruled out. The PlayStation VR is an upcoming VR headset which will work with the PlayStation move to allow for motion controlled VR. As the PlayStation VR is not out yet and only works with the PlayStation 4 it was quickly decided against. The PlayStation move however was much more interesting. It uses a camera called the PlayStation Eye which “has been programmed to recognize the exact size and shape of the ball on top of the Move remote. Once the Move controller is visible to the camera, it's able to detect the exact positioning of the ball in 3-D space”(6). This use of color and scale was strongly considered but it was simply more efficient to use a simplified version of skeletal tracking with the kinect. Finally users can use the Oculus Rift, which I am using, with Oculus Touch, which I am not. Oculus Touch is an upcoming VR controller which players hold in each hand and it tracks position, orientation, and button presses. As the Touch has not been released yet not much in known about how they work. I found a very similar project by a company called Striker VR, It is the “ARENA Infinity is a haptic VR gun which can simulate various weapon fire modes and other haptic effects”(7) It seems much more similar to the goal of my project than the other VR controllers, however as it hasn’t been released yet It is hard to tell how it works.

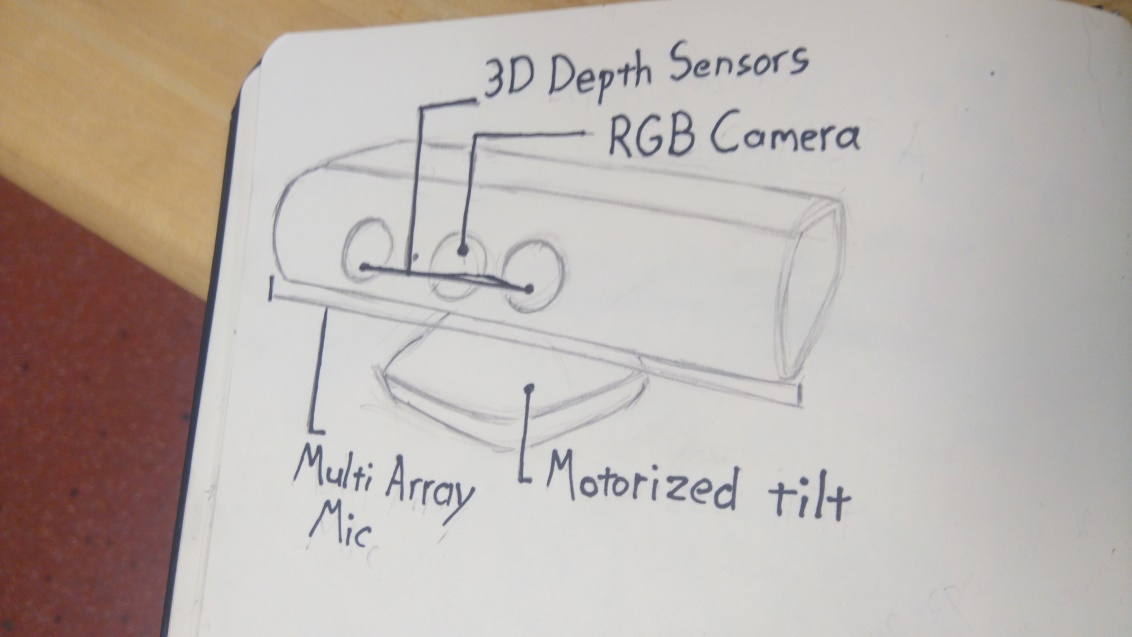
## Technologies Researched

I began by researching the Nintendo WII’s remote. It uses a remote and a sensor bar, which is “really nothing more than a shell for two infrared sources”. (8). The remove contains an IR sensor, which uses the two infrared sources to triangulate its position. As I later discovered this is essentially a precursor to valves lighthouse design. This setup has some serious limitations though, primarily that after a certain distance accuracy dropped off and position tracking becomes impossible(8). It also uses a built in accelerometer to track the forces acting on the remote, and uses a data converter to convert this analogue data into digital data for the Wii to use. The Wii gets the orientation of the remote using this and combines this with the position information to form the remotes position relative to the television.(9)



I then began research on how the Microsoft Kinect works. The kinect generates the user’s skeleton position via a two-step process. First it computes a depth map, and then it infers the skeleton from this depth map using machine learning. The depth map is constructed from three data sources, a speckle pattern, the focus of the image, and the stereo images. The speckle pattern is a known pattern, projected onto a scene and then the kinect records the distortion experienced by the pattern to define the 3-d shape. The focus of the image is used with the principle that objects further away from the focus of the camera is blurrier. It uses a special ‘astigmatic’ lens which can change its focus on the X and Y axes. It can cause a projected circle to become an ellipse whose orientation is defined by their depth. Depth from the stereo images uses the parallax between the images to generate the depth of objects in the scene.

With the depth map created stage two begins, inference of the skeleton. This is achieved via the use of a randomized decision forest, a decision tree with far too many possible decision nodes to calculate for each frame, so it uses a random subset of them to derive the body parts from the depth image. It then gets the position and orientation of each body part and creates a skeleton from this.(10)



I researched color blob tracking, similar to the PlayStation move, and found a number of reasonable solutions. It is “is one of the quickest and easiest methods for tracking an object”(11) however it requires some limitation to the design scope of the controller itself, in that a number of solid colored objects or sections must be easily visible in order to track the controller. I considered using the Microsoft Kinect for this but quickly discovered that it is much more economic time wise to use the kinect skeleton tracking to track the position of the controller. When it came to orientation tracking I first looked into Zuforia, a target based AR system, however it my use case is not directly supported, so I kept looking. I considered tracking the orientation of the gun using both of the players hands, and I prototyped this approach, however I quickly ran into issues if the player’s hands were ever hidden from the kinect camera. Finally I looked into Arduino based solutions to orientation tracking and quickly found the bno055 sensor, “an intelligent 9-axis Absolute Orientation Sensor”. An Arduino will be housed inside the controller with this and two buttons, one to register pulling the trigger and another to register pumping the handle. These will provide the state of the gun itself, while the kinect manages its location tracking. I also researched 3-d printing for the creation of the controller itself once its design and requirements are finalised.

## Other Relevant Research Done

I spent some time researching game design concepts in an effort to bolster the immersion of my virtual world. I quickly discovered the concept of the magic circle(12) which is the players acceptance of the rules of the game, similar to the suspension of disbelief for a film. The magic circle is important to establish quickly and once broken is extremely difficult to re-establish. My approach will take advantage of this as the gameplay is understood almost fully by the knowledge of how to use a gun and the conceit that you shoot the red things. This allows me to frontload the tutorial and still keeps it short enough to remain interesting. I learned the importance of having the player act through what they need to learn(13), which can be easily achieved (see Game Design Document). With the player ready quickly the primary gameplay can begin, and given that VR headsets tend to block the outside world from the player which greatly lowers the chance that the magic circle will be broken. I also read up on what makes a rail shooter, in essence it is a shooting game where the player does not control the path they are taking through the world(12).

Once I had my technical and game specific research organised I began to research software development methodologies. Firstly I looked into the waterfall software development methodology, I quickly discovered that the waterfall method is simply too rigid to be a viable contender(14), so I quickly moved on. I then looked into the lean methodology, which had a number of attractive principals. I particularly liked the focus on keeping everything changeable and minimalism. I also appreciated the focus on completing as much as possible at any given time, as I have found that often solutions are discovered while working on a project as this is when you have the most relevant knowledge of the project. The lean methodology places huge focus on the customer(15), which is not a good fit for my project as, while I will be conducting some user trails, I lack the time and the resources to research the market heavily. I finally looked at the agile crystal methodology. This foregoes the more strict processes of other methodologies in favour of focusing on the people, skills and communication of the team in question(16). I was drawn to this as I found in my time in work placement that many of the teams using the agile scrum methodologies had tweaked the process heavily to suit themselves, yet they all accepted certain aspects of the methodology which they didn’t like simply because it was part of the methodology. Furthermore for this project I am working on a team of one person with a project manager, so I felt the processes were unnecessary.

Finally I came to choose which game engine to use for the project. I looked into the Unreal Engine, Bryans Game Engine, and Unity. All three Engines support Microsoft Kinect, Arduino interaction, and VR. Unreal Engine and Unity have continuous support both from their developers and online communities and there are many helpful tutorials online to help a new developer get accustomed to the new development environment. Unreal Engine and Bryans Game Engine both use C++ as their primary language. I have been using C++ for nearly three months at time of writing, and I am beginning to come to grips with the language. Unity uses C# and JavaScript as its primary languages. I have experience with JavaScript from various web development modules I have taken over the course of my college career, and I have some limited experience with C#. My language of choice is Java; I have used java as my main programming language in both of my more recent programming modules in college and in both of my internships. C# is very similar to java(17), and I have experience with the Unity game engine myself from my time in the Games Fleadh Games Studio Challenge.

## Resultant Findings and Requirements

Ultimately my research has led me to conclude that the best solution to the issue of translating a real life object into a virtual environment is to use a Kinect to track the players hand position to get position data and an orientation sensor to map the in game gun to the custom haptic controller. This solution has the most economic requirements of those I have researched in terms of time and complexity versus quality of the final product.

I will need to connect an Arduino to handle the orientation sensor and button inputs. I will have to design the controller in auto-CAD and get it 3-d printed. This will allow me to create a solid professional looking controller without having to learn how to actually make one.

The rail shooter seems to be the best type of game to showcase the controller and entertain the players. It downplays the issues of virtual reality sickness while allowing players to move in a time tested manner. The gameplay is intuitive and allows the tutorial to be quick and effective enough to avoid hampering the experience too much while still proving without doubt that the player has the skills required to play.

I will be using the Agile Crystal methodology and breaking my time into two week sprints. This will help mitigate the considerable risk that comes from combining disparate technologies in a new a novel way. I will be using the Unity game engine to create my game.

Design

Identification of a design methodology including why it was chosen

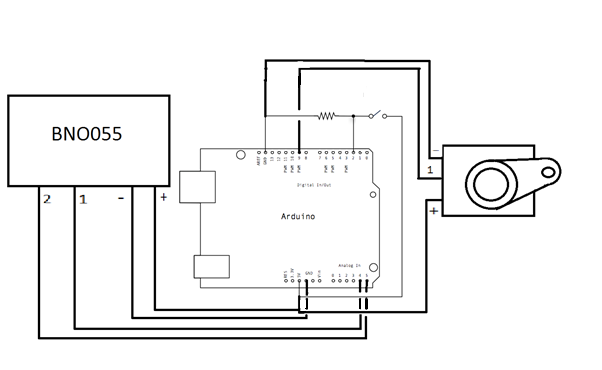
I began with a top down design methodology as I had to begin with the solution to my hardware requirements. However once I had designed a general solution and had chosen which hardware and software to use I switched to a bottom up design methodology. This was due partly to the fact that I had to wait for certain pieces of hardware to arrive so it made sense to get what I had working with the system and then build on top of it. I started by designing a system where the game would be able to track the position and orientation of a gun and built on top of it. First I built on the skeletal tracking system in the Microsoft Kinect Unity SDK to allow the player to move in the game world space via real life movement. Then I built a system to move the player along a path controller by the game. With the player able to move and be moved through game space I built enemies capable of interacting with the player. With these in place I set up the gun component to allow the player to interact with the enemies. Finally I built a system to procedurally generate content for the player using these systems.

I used a top down design methodology during the design process as I felt the only way to build a working gun controller was to divide the problem into manageable chunks. The only hardware solution I could find which solves both location and orientation tracking was valves lighthouse technology, however using this was both prohibitively expensive and required attendance of a workshop I could not attend and likely wouldn’t have been able to get invited to. Therefore I had to solve the issue myself, and I had to break the problem down. First I listed all the required features, and then I came up with a solution to each of these requirements separately.

The bottom up design methodology allowed me focus on each requirement one at a time building it up piece by piece. It seemed like the right choice as a bottom up design methodology is generally suited to system built on top of existing systems, and mine had to be built on the Kinect, which had limited access to edit its software. Furthermore I couldn’t work on the gun component of my project until the required hardware arrived, so I decided to build the project using a bottom up design methodology.

The use of both approaches allowed me a certain level of freedom to experiment as each requirement was relatively loosely coupled as a result of the design process, but were easily built upon from the development process. This meant that features could be added or altered with relative ease. For example upon implementing enemies I realized that with a little change to how I was tracking the players skeleton I could allow the player to dodge enemy fire, and this has been one of the most immersive aspects of my project.

Design of each of the project components

The Gun controller is made up for three discrete component parts, a button connected to the trigger, and BNO 055 absolute orientation sensor and a servo motor. These are connected to an Arduino board contained inside the gun controller.

The Arduino has two states. When the BNO055 sensor is not calibrated it will output ‘:NotConfigured:<System calibration level>:<Gyroscope calibration level>:<Accelerometer calibration level>:<Magnetometer calibration level>’

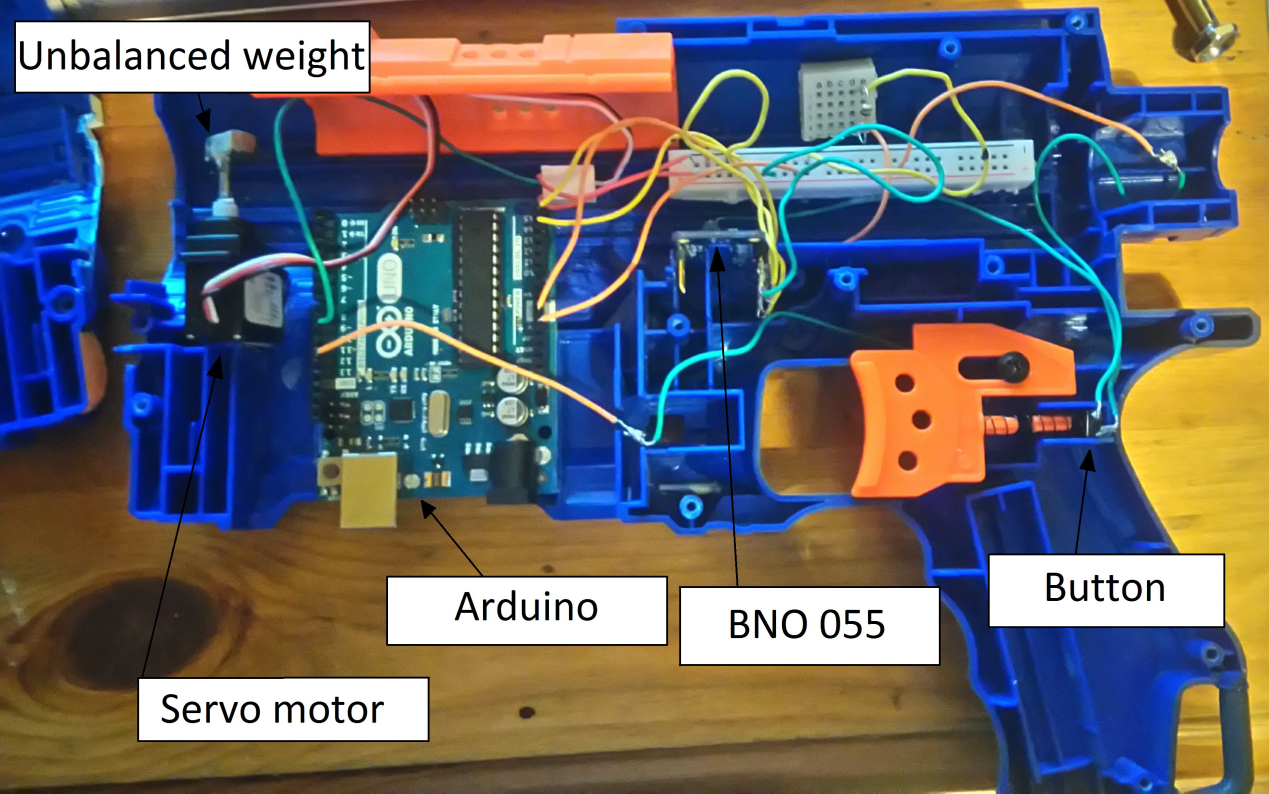
The system, gyroscope, accelerometer, and magnetometer calibration levels range from 0 (Not calibrated) to 3(Fully calibrated)

This string is displayed inside the game allowing me to calibrate the BNO sensor with the headset on. Once all four calibration levels are at 3 the Arduino will start sending the signals required to play the game.

The BNO 055 sensor is used to get the guns orientation in Euler angles. These angles are sent to unity via the serial port in the form ‘XRotation:YRotation:ZRotation’.

The button, when pressed down, will append a ‘H’ to the orientation message totalling XRotation:YRotation:ZRotation:H. Unity splits the input string via ‘:’ and if it’s split into four parts it will fire in game.

The Servo motor is controlled via a signal sent from unity via the serial port. When it receives this signal it will move the motor and upon receiving the signal again it will move back.



The Kinect is used to place the player’s skeleton in the game, and the Oculus rift camera is placed via the head position. It was used primarily as is in the Kinect Unity SDK however the camera following the head position had to be handled in a separate script as Unity does not allow users to set the position of the camera, so it had to be worked around. This was done by making the camera a child of an invisible game object, and attaching a script to move that game object to the same position as a second invisible game object, and having that second object be placed via the kinect.

The Kinect is also used in conjunction with the BNO sensor to place the gun. The guns position is calculated as follows: The position of the players preferred hand + (the current orientation of the gun x (half the length of the gun + half the height of the gun)). This was done as unity places objects via there centre, so I had to calculate the position whereby the handle of the gun would be in the players hand.

Furthermore I implemented a simple jitter filter as the kinect had far too much jitter when used initially. It simply ignores any movement smaller than a given amount.

The movement system uses a waypoint based system to define the player’s path. The path the player follows is made up of waypoint objects contained inside the rooms and each waypoint can turn the player in a given direction, they can stop the player until all the enemies in the room have been destroyed, and move the player along to the next waypoint. This system was designed to move the player slowly and steadily as to avoid any nausea or dizziness. It also had to be compatible with the procedural generation system. Each waypoint keeps track of whether or not the player has finished interacting with it.

Each room keeps track of a number of variables to be used by the levelBuilder and the enemies. It also checks each of its waypoints to see if the player has completed the room.

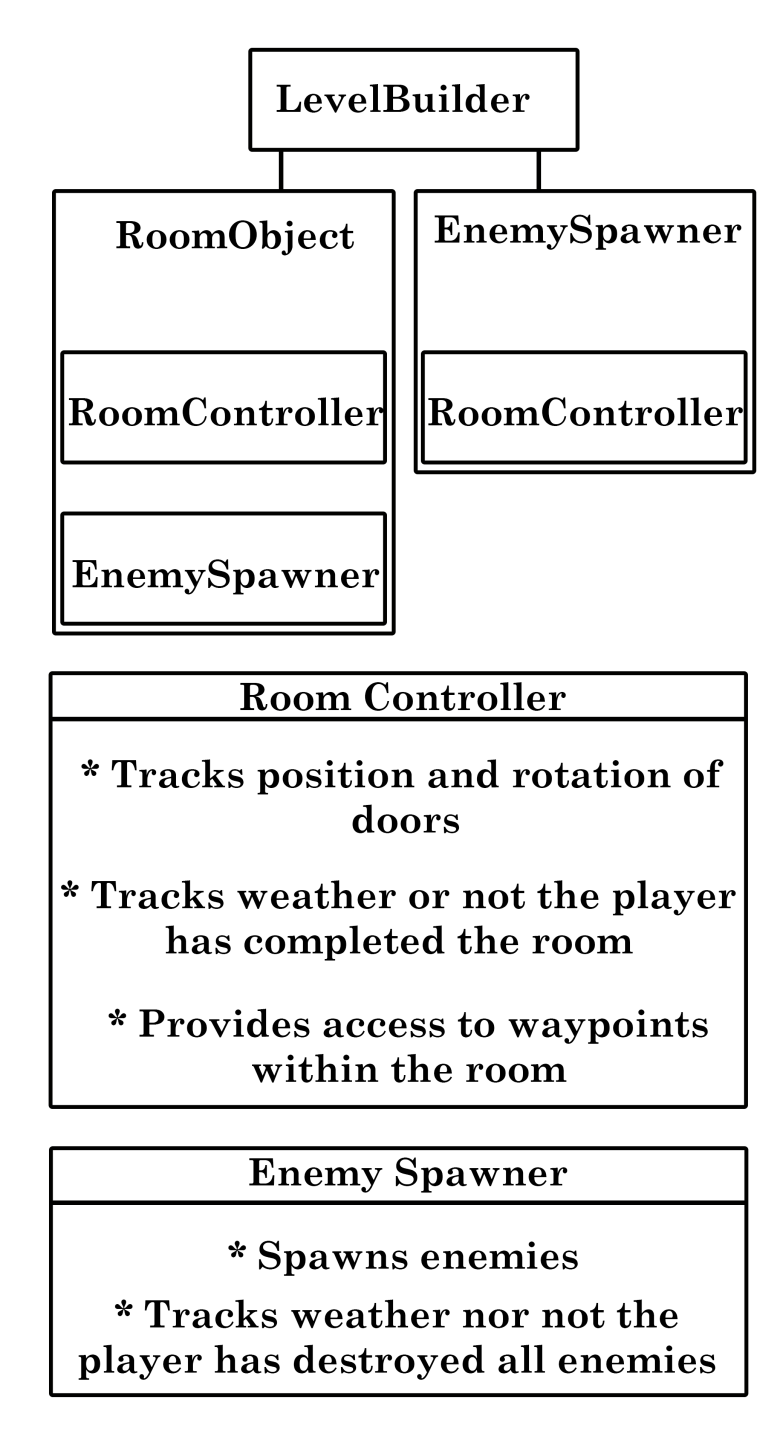
The Procedural generation system is controlled by an object called the levelBuilder. It can build two types of areas, rooms and corridors. Rooms spawn enemies for the player to fight and require the player to clear the room before progressing. Corridors do not spawn enemies; they were introduced to provide rest for the player.

The level builder creates and places the same room every time to start with, this room is a tutorial room which ensures the player knows how to aim, shoot and reload. It then places rooms one at a time connected to the previous room. This is achieved through the use of door objects within the rooms. The level builder finds the position of the last room, the relative position of the out door of the previous last room, and the relative position of the in door of the room being created. It then finds the position to use for the new room by moving the combined vector of the two doors from the position of the previous last room in the direction of the out door of the previous last room. It will then temporarily add the chosen room in the position calculated orientated to match the out door of the previous final room.

The rooms check as they’re placed to see if they collide with any previous rooms and, if they are, are destroyed and replaced. Once a room has been added the level builder will access the last waypoint of the previous room and set its next waypoint to be the first waypoint of the newly added room. It then calls the generate enemies function of the newly added room, passing in a number to represent the difficulty rating of this room. If the temporary room can go one frame without colliding with any other room objects it will be added to the array of current room objects. Finally it will add to the difficulty rating variable so that subsequent rooms are more difficult as the game progresses.

The levelBuilder also checks each of the room objects in the array of current rooms to see how many rooms have been completed by the player, and if 3 or more rooms have been completed it will delete the oldest room to clean up the scene.

Each room object has invisible boxes within which mark the areas each enemy type can spawn in. The room will randomly choose an enemy to spawn one at a time. Once an enemy is selected it will generate a random coordinate within its invisible box to act as the spawn position of the enemy. Different enemies have a different ‘cost’ associated with them and this is taken from the difficulty rating remaining. Rooms will continue to spawn enemies until the difficulty rating remaining reaches 0.



The enemy behaviour is simplistic. Enemies will wait until the player is moving towards the waypoints contained in their room. Once activated the enemies will aim at the player and, depending on their type, will either fire at the player or spawn small minion enemies which fly toward the player. They will remain doing so until the player destroys them with their gun.

Clearly identifying the list of features and use cases supported within the project.

Games, by their nature, are not suited to a wide variety of use cases. Circuit Shock is even further limited as the various hardware requirements mean it would be a poor candidate for a home release. It could possibly work in an arcade environment, but these have largely died out outside of Asia.

As such I decided to focus on one use case. Players at a demonstration, be it a formal demonstration such as the project fair or a less formal demonstration such as the State of Play Expo.

This allowed me to focus on the initial goal of the project, enhancing immersion, and ignore many aspects of games that would be required to induce and improve replay ability. Features like score tracking and multiple levels may be nice to add in the future, but they do not benefit the project from either an academic standpoint or a use case based standpoint.

The features of Circuit Shock are as follows:

Skeletal tracking is implemented via the Microsoft Kinect and has a number of features built atop it.

* Firstly the position of the players hand is determined to be used in conjunction with the orientation and size of the gun to generate the position the gun should be rendered in. The player selects their preferred had at the beginning of the game.
* It also controls the position of the VR camera, allowing the players to feel like they’re really moving around in game space.
* The entire skeleton is rendered, allowing the player to look down and see a game representation of their bodies, further increasing immersion.
* The skeletal tracking allows the player to dodge in game fire by moving in real space.
* Finally the skeletal tracking is used to allow the player to interact with certain menu objects via real life movement. The rest are activated by shooting them with the player’s gun.

The Arduino based gun controller also has a number of features, some required, some to increase immersion.

* The gun controller uses a BNO 055 absolute orientation sensor to track the orientation of the physical gun peripheral in the player’s hand. This information is passed to unity via a serial port.
* The trigger of the physical gun peripheral is linked to a button input allowing the player to fire the gun in a natural way. This buttons state is transmitted to unity alongside the orientation data.
* The gun peripheral also contains a servo motor with an unbalanced weight to allow for haptic feedback. This haptic feedback is controlled via signals sent from unity via the serial port.

Finally there are a number of software only features

* The levels are procedurally generated to allow the player to play for as long as they are able to keep their health above 0.
* The players move along a pre-set track which is stored as a number of waypoint objects. These waypoint objects are children of the rooms they’re in and control where the player will move to next, what direction the player will be facing, and whether or not the player needs to stop and fight the enemies in the room.
* The enemies are generated by each room given a number to control the difficulty of the encounter. They are activated when the player begins to move toward the first waypoint in their room, and are deactivated once the player begins to move toward a waypoint not in their room. This prevents enemies that create more enemies from making the encounter more difficult before the player has had a chance to assess and interact with them.
* The player moves inside a cart, similar to a mine-cart, which is moved and turned with the player. This reduces nausea and dizziness as it gives the player a visible marker to latch on to. It also provides a clear indicator for how to reload the gun as this is done by placing the gun inside the cart.
* The UI elements are minimal so as not to detract from the immersion, the number of bullets left before a reload is required is displayed along the side of the gun the player is holding, and their health is displayed in a bar at the top of the screen. The health bar was chosen to avoid confusion as players are very used to this contrivance.
* The level generation always starts with the same room, a tutorial of sorts. It requires the player to shoot six stationary targets before they can progress on to the game proper. This ensures that the player knows how to aim, shoot, and reload before they are in a situation with any danger.

Architecture & Development

Overview of the system architecture and a diagram to represent all of the key elements within the architecture.

Description of the architecture of the game

The player interacts with the game via real life actions. This is broken down into two types of interaction, direct interaction and interaction via the gun controller.

The player’s state is tracked by a Microsoft Kinect, this tracks the skeleton position of the player and is used to allow players to move and dodge in game space.

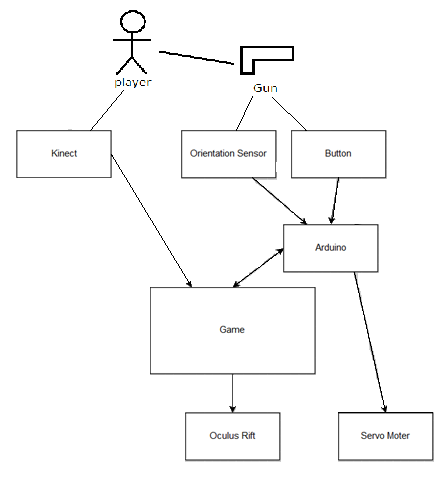
The hand positions tracked by the Kinect are used along with the orientation from the gun controller to generate the position of the in game gun.

The guns state is tracked by an Arduino; this consists of the orientation of the gun and the state of the trigger.

Game levels are built by an object called the ‘levelBuilder’, this procedurally generates rooms for the player to progress through. It also destroys old rooms and controls the rate at which rooms spawn enemies.

Rooms contain waypoints to guide the player. These waypoints can stop and/or turn the player, and set the player moving to the next waypoint. These are linked by the level builder. Rooms check if the player has passed through all their waypoints to allow the level builder to tell if the room has been completed.

Enemies are spawned by the room which contains them. This process is controlled by the level builder. They activate and deactivate based on whether the player’s next waypoint is contained within the same room as the enemies.



Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.

Gun Controller:

The first and likely the biggest issue I encountered while working on the Gun Controller was the design. I initially had no idea how one would go about tracking a gun and passing this information to Unity. I began researching this however all market tested solutions weren’t directly applicable. In order to overcome this Issue I had to split the requirements and deal with each requirement one at a time. I needed to track the position, orientation, and state of the gun. I used the Kinect to track the position of the player’s hands, along with the orientation and size of the players gun to calculate the position of the players gun.

I got the orientation of the gun from a BNO 055 sensor, which came with its own issues. Firstly it was difficult to source one. The first one I got access to was broken and couldn’t be registered by the Arduino. The second was lost in transit. The third arrived in full working order however it had to be soldered together, so I had to learn how to solder. Once I had the BNO sensor working and sending it’s orientation data to Unity I discovered that the guns orientation was not matching the gun at all. This was because the BNO sensor tracks it’s orientation on each of the three axis, however Unity sets an objects orientation around each of the three axis. Rectifying this required a lot of trial, error, and reading the Unity documentation. The BNO documentation was minimal to the point of uselessness. It was a relatively easy issue to fix, but an extremely difficult issue to diagnose.

Once I had the position and orientation working for the gun controller I discovered that turning the gun would hide the players hand from the Kinect, and also that the Kinect Unity SDK did not register whether or not it was tracking the players hands properly, nor did it provide confidence scores so I had no way of dealing with this issue from a software perspective. In order to overcome this I had to reduce the scope of the gun controller, I moved from a two handed rifle to a one handed pistol. This prevented the issue, as I had no way to directly solve it.

During testing I realised that the BNO sensor was sending the absolute rotation on the X axis, but I needed it’s rotation on the X axis relative to the direction of the Kinect. I achieved this by storing the guns initial X axis rotation and calculating its relative X rotation using its initial X rotation and current X rotation.

I used a button linked to an Arduino to track when the player pulls the guns trigger, which didn’t have any direct issues however connecting the button to the Arduino inside the gun cause a few issues. The wires provided with the Arduino were not long enough to stretch from the Arduino board to the button in the base of the trigger. I fixed this by soldering two wires together; however it impeded the trigger from moving freely. In order to fix this I had to make a space in the walls of the gun controller to allow the wire to pass unimpeded.

The Haptic feedback in the controller is achieved by using a servo motor and an unbalanced weight. The servo motor was relatively straight forward to set up, but the unbalanced weight was very difficult to source. I needed a weight which is scaled correctly for both the servo motor and the gun controller shell. In order to get the weight just right I designed the weight myself and my father fabricated it.

Kinect:

The Kinect is used to track position data. It was initially used only to track the position of the player’s hands; however this was expanded to track the players whole body. During testing it was brought to my attention that the controller only worked in the player’s right hand. I dealt with this by adding a new step to the beginning of the game where the player choses which hand they would rather use by using their hands via the Kinect.

The Kinect also had a major issue with jitter. This made aiming almost impossible as the gun would move sporadically during the time the player was pulling the trigger. It also made the VR camera move slightly in random directions every frame which destroyed the player’s immersion and caused dizziness and nausea. I implemented a simple jitter filter to deal with this, it simply checked to make sure that the difference in position between the old and new positions is greater than a given variable. This variable was altered until it felt right while playing.

Using the Kinect to control the VR camera had a number of issues. Firstly the Unity engine does not allow you to set the camera position directly, so I couldn’t have the Kinect set the camera to the players head position. To work around this I had to create an invisible ‘head’ object and an invisible camera holder object. I had the Kinect set the position of the invisible head object, made the camera a child of the camera holder object, and had the camera holder move to the invisible head object. This allowed the camera to take its place in the skeleton the Kinect creates, however this skeleton is designed to be used in the third person so I had to edit it to work in a first person setting.

Oculus rift:

The Oculus Rift worked well out of the box, however it’s rotation around the X axis was set by the Oculus Rift itself once it had been set up. This orientation difference lead to the player and their in game body representation being turned in two different directions and rendered the game unplayable. This was resolved by resetting the Oculus Rifts orientation when the game starts up.

Movement:

The movement system was designed to achieve three goals, It had to allow the player to turn, move, stop, and start as the game requires without giving the player control over their own movement, work with the procedural generation system, and prevent the player from feeling too dizzy or nauseous and interrupting the immersion.

Initially moving caused dizziness, and slowing the movement would lower but not remove the associated dizziness. In order to fix this I added a cart which moves with the player and provides a stable point to look at.

Getting the movement system to work with the procedural generation system caused a number of issues. The turning system was turning the player in seemingly random directions, this was very difficult to debug as I was miss-attributing it as an issue with the movement system. This was resolved by adding arrows to the waypoint objects that didn’t show up in the game. This allowed me to discover that the movement system was turning it correctly however the procedural generation system was creating them with the wrong orientation.

Finally the stopping and starting system to force the player to destroy the enemies in a given room was initially time based, but this lead to the player having to stand and wait for the timer to finish once the enemies were destroyed. To fix this I changed from using a time based system to checking a list of enemies held by the room to see if they have all been destroyed.

Procedural Generation:

The first issue I ran into with procedural generation was placing the rooms correctly. I had to figure out how to place the rooms dependent on what direction the player will exit the previous room. I achieved this by adding door objects where the player enters and leaves the room. This system worked well initially however it broke down when spawning too many rooms in a row as it wasn’t correctly placing rooms relative to the previous room. To deal with this I had to take the positions of the doors and multiply them by the inverse quaternion of the doors rotation. This left me with a three dimensional vector of the straight line distance from the centre of that room to its door. I then add these vectors and multiply them by the world rotation of the current out door to get the position the correct distance and orientation from the centre of the current last room. This generates a position where the rooms are just far enough away from one another that they do not collide but close enough that there is no space left between the rooms, in the direction the player will leave the previous room.

I then had to orientate the new rooms correctly. Initially I had been using the same world rotation as the previous rooms out door. This worked for the first room or two but quickly broke down. This was caused by a lack of uniformity in the room objects I was using. This was easily fixed but very difficult to diagnose as I was under the impression I was spawning the rooms incorrectly. It was only after extensive testing that I discovered the break downs in the procedural generation were occurring after the same rooms every time.

Once I had the levels being built correctly I had to figure out how to move the player through these rooms without giving the player the ability to control their own movement. This design challenge was both a quirk of the genre of game I had chosen and a design limitation I had decided upon as movement is the number one cause of dizziness and nausea in virtual reality. I wanted to control the player’s movement to ensure that it would be slow and steady, but able to navigate through a level that was being created as the player plays. In order to accommodate this I decided to hard code the waypoint objects which control movement into the rooms themselves and have the level builder link the waypoints in each room together. This also broke down after a few rooms as I was initially using the local rotation of the waypoint objects when turning the player, which worked well in testing as the waypoints weren’t children of anything, however once the waypoints were turned by their parent rooms this broke down. To remedy this I changed to using the world rotation of the waypoint objects and added invisible arrows to them so I could check their rotation in Unity’s ‘scene view’.

After this step I had a level builder building useable levels; however it had a small chance of building itself into a corner whereby the door out of the last room lead directly into a previous room. This caused it to constantly spawn and then destroy new rooms as every possible room would collide with a previous room, be destroyed, and get ready to spawn a new room. At this point I wanted to expand on the level generation and check when building rooms if they were boxing themselves into a corner; however time restrains meant this would put my project under considerable risk, so I simply limited the number of rooms that would be spawned at one time to avoid the issue.

The last major issue I ran into while working on the procedural generation was an optimisation one. When I added the enemy spawning the game would suffer from a reduced frame rate, which both caused dizziness and destroyed immersion, and the collision detection became increasingly inaccurate. To circumvent this I introduced a system to allow rooms to track the player’s progress through them and, once each waypoint within the room in question, the room could be marked as completed. I altered the level builder to destroy the oldest room in the room list when three or more rooms have been completed. This reduced the levels overheads to manageable levels and allows the game to continue in a reliable stable manner.

Identify key development components;

The key development components are the Microsoft Kinect interaction and the gun controller. These are the aspects I have attempted to leverage in order to increase immersion.

The Microsoft Kinect is used to track the player’s body and translate this to game space. This serves a number of purposes, it allows the player to move their physical bodies and see the result of that movement in game space, it allows the player to look down and see an in game representation of their own body, and it provides position data for the gun controller.

This provides a layer of immersion not usually found in games, any movement or action the player knows can be used to interact with the game naturally. There is no need to teach the player how to move as the player intrinsically knows. Furthermore it provides the player with an in game representation of their bodies mapped to the points the Kinect tracks. This can provide a simple effective way to ground the player while also allowing the player to interact with in game objects via their natural actions.

The gun peripheral allows the player to interact with the game without suffering too much from the Kinects infamous inaccuracy. It allows the use of minimal Kinect interaction while providing the player with a ubiquitous input. It also allows the game to send haptic feedback to the player via an internal servo motor. It increases the number of senses the game can output to the player to three, audio, visual, and haptic.

These components comprise the core of my project, the software only aspect of my project was designed to showcase this without causing any breaks in immersion.

Identification/explanation of external APIs used versus own code ; List of classes of your code etc .

Unity Engine

The unity engine provides a lot of functionality. I used the unity engine to handle rendering, lighting, and collision. It provided a number of graphical features, from line renderers to

<should I include the fact that you gave me the code for communicating between unity and the arduino?>

Kinect SDK

The Kinect Unity SDK comes with a number of functions and example scripts classes. I used the ‘CubemanController’ class to create the player skeleton, slightly augmented to introduce a simplistic jitter filter. It uses an object called the kinectManager which I also use in my gunPlacer class. It allows you to get the position data for bones from the Kinect. I use this to get the players hand position as it is one of the inputs for calculating the gun position.

Oculus SDK

Unity has native Oculus rift support so this worked out of the box.

Own Code

gunController

gunPlacer

The gun placer uses the KinectManager to get the position of the players preferred hand. It also gets the orientation and renderer bounds of the gun object. It then calculates the position to place the gun controller and places the gun in that position.

playerMove

The player move class moves the player as per the commands given by the waypoint objects. It can turn the player to match a given rotation. It can also move the player toward a target game object.

WaypointMovementController

The waypoint movement controller waits for the player to collide with the game object it is attached to, and once it does the waypoint activates. Once active the waypoint may tell the playerMove script to turn to match the rotation of the waypoint. It may stop the player until all the enemies in the room have been destroyed. Whether it has done one, none, or both of these it will tell the playerMove script to move the player toward the waypoints target.

cameraFollowHead

The camera follow head class is a short script used to work around the inability to set the cameras position directly in unity. This script will move it’s object to match the position of a public GameObject, the public GameObject is an invisible object placed by the Kinect, and this script is attached to an invisible object which acts as the cameras parent.

handSelectionController

The hand selection controller spawns the two option boxes the player can use to select their preferred hand. It also destroys these option boxes once a preferred hand has been selected.

handSelector

The handSelector class has a public ‘leftHanded’ Boolean and upon colliding with the player will set the preferred hand on gunPlacer and set its ‘handSelected’ Boolean to true

spawnObj

spawnObj spawns a copy of a given Game Object every 10 seconds

levelBuilder

The level builder uses variables from the various rooms roomControllers and builds the levels the player progresses through.

roomController

The room controller stores information about the room it is in control of. It also checks if all the waypoints in the room have been completed. If they have the room registers itself as completed

enemySpawner

The enemySpawner spawns enemies into the rooms their attached to. It also checks to see if all the enemies in the room have been destroyed, and if they have it marks itself completed. Waypoints use this to control their ‘fight’ Booleans.

loadScene

This script loads the scene defined in the public string component when the object it is attached to collides with a players bullet.

healthTracker

The health tracker sets the fill amount of the healthbar UI element. It also stores the players current health and lowers the players health if they collide with an enemy object

destroyAfterX

This script destroys the attached GameObject after a number of seconds defined in the Unity editor. It is used to destroy particle systems

lookAtTarget

This script turns the attached GameObject to face the target defined in the Unity editor

explodeWhenShot

This script destroys the attached GameObject when they collide with the players bullets. It also creates a particle system explosion. It is used to allow the player to destroy enemies

spawnShooter

This script creates spinning shooter enemies attached to the stalks these enemies are attached to. This script is attached to the stalk.

shootAtPlayer

This script spawns enemy bullet objects with the same orientation as the GameObject it is attached to every few seconds. The fire rate is controlled by the Unity inspector

trackStatus

This class checks if child enemy objects have been destroyed and destroys the GameObject if it has. It is used on the spinning shooter stalk

Spinner

This class spins it’s GameObject at a rate defined in the Unity inspecter

bulletDestroyOnEnemies

This class destroys the associated GameObject when it collides with an enemy GameObject. It is attached to the players bullets

batSpawner

This class spawns small bat objects

batFlight

addEnemyToRoom

moveForward

placePlayer

menuController

System Validation

Testing

Hardware testing via the Serial monitor in the arduino idk.

Personal functional testing via the Oculus rift, Kinect and gun

Personal functional testing via the unity idk.

User trials

Implementing the cart because of neasuea

Implementing the hand selection based on feedback

<Will have charts and more conclusions after more extensive user testing this weekend>

Demonstration

<will link to youtube video of me playing picture in picture>

Project Plan

Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated.

I gotta reword this from what I will use to what I have used

I am going to use the agile crystal methodology with 2 week sprints as this allows a sizeable chunk of time to spend on feature development without leaving them so long that I risk losing several weeks being stuck working on a fruitless feature that was outside my capabilities. I have ordered my backlog of features to allow for a certain margin of error when planning my sprints meaning that features can run long if I decide it’s worth the risk, but it must be a conscious decision. My team consists of myself and my project manager, and we meet face to face for weekly meeting as per the agile methodology. I also intend on using working software as my measure of progress as per the same.

I am handling design and prototyping first in an attempt to reduce the risk involved in my project as many of the different aspects run the risk be being untenable. Once I have the various prototypes working I will being working on getting the minimum viable product ready, this will help to ensure I can finish the project, while leaving a few sprints to refine and add to the minimum viable product. This will allow me to work on the various features that would be nice but are not required without risking the required ones. Once the minimum viable product has been completed I will begin user trails at the end of each sprint before planning the next sprint. This will allow me to accept some customer input and change the plan accordingly, but only after the vast majority of the risk associated with the project has been removed as the minimum viable product has been completed.

Mention having to switch from two handed gun to pistol.

Mention how sometimes college work made it very hard to follow two week sprints

I would map out all my college submissions and plan around them. I would put a greater focus on sourcing the required components earlier. I would spend more time early on researching the components, discovering I couldn’t allow for hidden body parts was a nasty surprise.

Conclusion

Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future.

<can I really get away with listing the component parts of my project again?>

I learned a lot about researching, software and hardware design, time management, and working around hardware limitations.

I would implement the ability for unity to control how long the haptics in the gun lasted. I would implement a number of environmental obstacles that must be avoided by moving in real space. I would implement a number of different gun types that could be selected by using the player’s free hand to press buttons on their cart. I would have these different guns cause different haptic results in the gun. I would also implement more enemy types.

Bibliography

1. Pappas S, 20 LSC| A, ET 2016 01:15pm. Why Does Virtual Reality Make Some People Sick? [Internet]. Live Science. [cited 2016 Nov 23]. Available from: http://www.livescience.com/54478-why-vr-makes-you-sick.html

2. The History of Music Games [Internet]. IGN. [cited 2016 Nov 23]. Available from: http://www.ign.com/articles/2014/10/21/the-history-of-music-games

3. Home [Internet]. Space Pirate Trainer. [cited 2016 Nov 21]. Available from: http://www.spacepiratetrainer.com/

4. Buckley S. This Is How Valve’s Amazing Lighthouse Tracking Technology Works [Internet]. Gizmodo. [cited 2016 Nov 20]. Available from: http://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768

5. Lang B. Valve Opens Vive’s Tracking Tech to Third-parties for Free, Details Dev Kit for Licensees [Internet]. Road to VR. 2016 [cited 2016 Nov 20]. Available from: http://www.roadtovr.com/valve-third-party-lighthouse-api-steamvr-tracking-htc-vive-dev-kit-license-royalty/

6. How PlayStation Move Works [Internet]. HowStuffWorks. 2010 [cited 2016 Nov 23]. Available from: http://electronics.howstuffworks.com/playstation-move.htm

7. Lang B. Striker VR Shows off Working Prototype of ARENA Infinity Haptic VR Gun [Internet]. Road to VR. 2016 [cited 2016 Nov 21]. Available from: http://www.roadtovr.com/striker-vr-arena-infinity-v1-prototype-haptic-vr-gun/

8. Making Sense of the Sensor Bar [Internet]. IGN. [cited 2016 Nov 20]. Available from: http://www.ign.com/articles/2006/11/29/making-sense-of-the-sensor-bar

9. Chad Tower. How the Wii Remote Works [Internet]. 2008 [cited 2016 Dec 1]. Available from: https://www.youtube.com/watch?v=ETAKfSkec6A

10. MacCormick J. kinect. pdf, 09.06. 2011. Does Kinect Work.

11. Color Object Tracking [Internet]. [cited 2016 Nov 20]. Available from: http://www.roborealm.com/tutorial/color\_object\_tracking\_2/slide010.php

12. Adams E. Fundamentals of Game Design. 3 edition. Berkeley, CA: New Riders; 2013. 576 p.

13. Extra Credits. Extra Credits: Tutorials 101 [Internet]. 2012 [cited 2016 Nov 23]. Available from: https://www.youtube.com/watch?v=BCPcn-Q5nKE

14. Software Development Methodologies [Internet]. [cited 2016 Nov 24]. Available from: http://www.itinfo.am/eng/software-development-methodologies/#chapter14

15. Poppendieck M, Poppendieck T. Lean Software Development: An Agile Toolkit. 01 edition. Boston, Mass.: Addison Wesley; 2003. 240 p.

16. Crystal Clear: A Human-Powered Methodology for Small Teams: Amazon.co.uk: Alistair Cockburn: 0785342699470: Books [Internet]. [cited 2016 Nov 24]. Available from: https://www.amazon.co.uk/Crystal-Clear-Human-Powered-Methodology-Small/dp/0201699478/ref=sr\_1\_1?ie=UTF8&qid=1480001888&sr=8-1&keywords=crystal+methodology

17. C# and Java: Comparing Programming Languages [Internet]. [cited 2016 Nov 24]. Available from: https://msdn.microsoft.com/en-us/library/ms836794.aspx

Appendix `

**Controller Design Document**

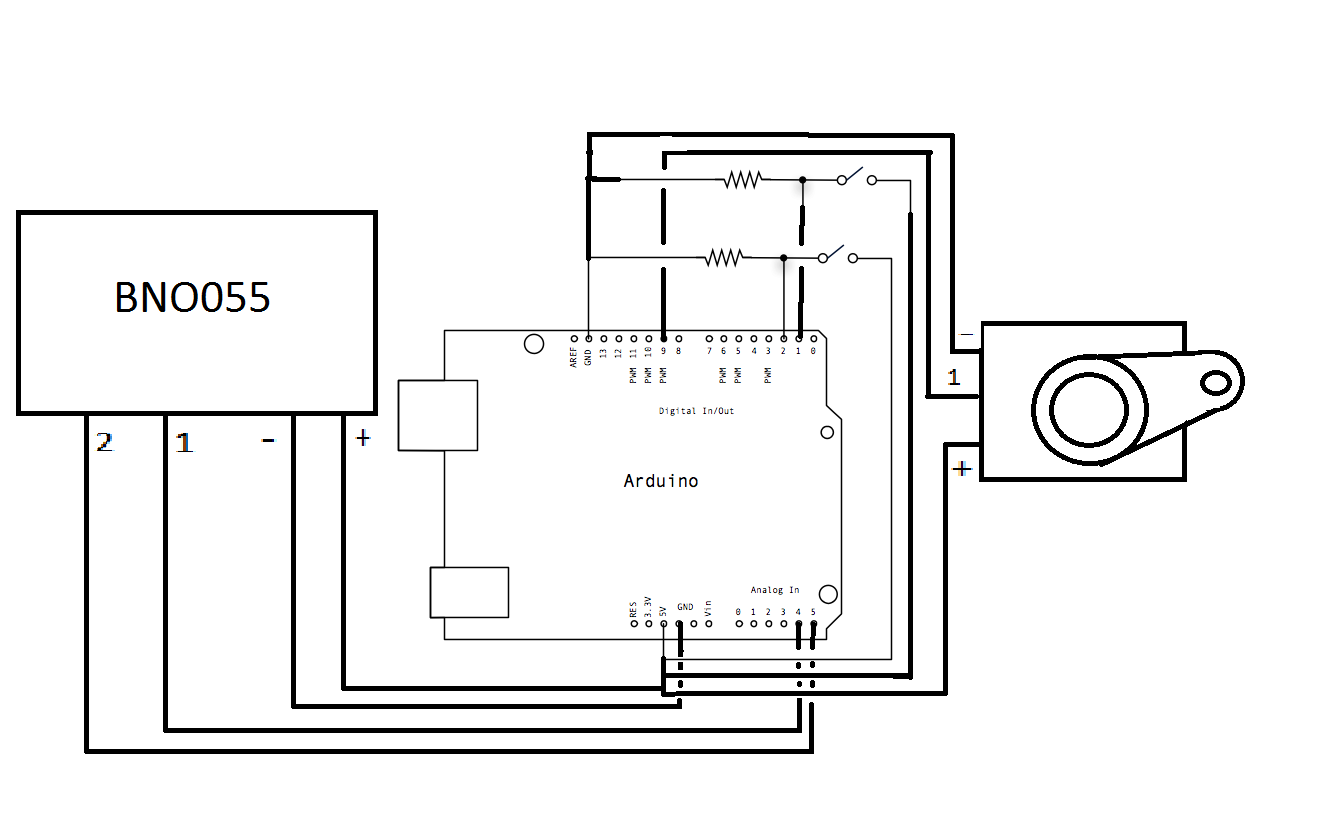
**Technologies Involved**

**Kinect**

- The Microsoft Kinect will track the position of the gun relative to the player in real space, and transfer this information to unity to map the virtual gun to. It will favour the hand closer to the player’s body where it can, and if it can track both hands it will track the distance between them.

If it loses track of the players close hand it will use the far hand and the orientation to generate the position of the close hand, this should reduce graphical errors greatly.

**Arduino**

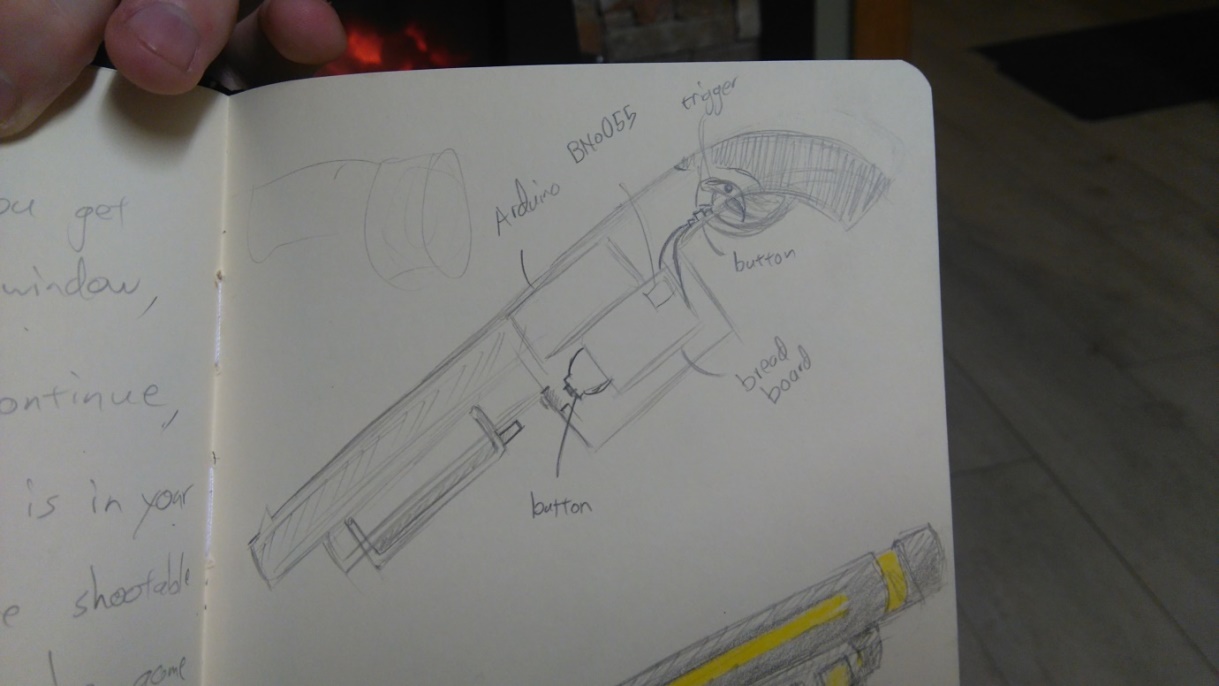


- The gun will use three circuits to track the players input. Two of these will be simple button circuits, one mapped to the virtual guns trigger, and the other linked to the in game guns reload function. These buttons will be controlled by the controllers trigger and pump respectively. The third input will be a BNO055 Absolute orientation sensor. This orientation tracking will be mapped to the in game guns orientation. The in controller output will be handled by a servo motor to provide haptic feedback.

**GUN DESIGN**

**External**

**Internal**



-The physical shell of the gun is modelled after a tommy gun. This was chosen as the large circular section in the middle of the gun provides fantastic space to house internal electronics. The trigger and pump will be placed in a manner which presses the button when extended to their limit allowing the player to control the button inputs via the controller itself.

Virtual Entanglement

**Game Design Document**



Revision: 0.0.0

GDD Template Written by: Benjamin “HeadClot” Stanley

Special thanks to Alec Markarian

Otherwise this would not have happened

[Overview](#_yj5nhqp5cf0j) 18

[Theme / Setting / Genre](#_5s48wntac2es)

[Core Gameplay Mechanics Brief](#_uzq23hfhdv6e)`

[Targeted platforms](#_kvz0cxkhwt0s)

[Project Scope](#_rdb2xo3rjh0s)

[Influences (Brief)](#_155cm8v36jpc)

[-](#_c6nxu1rzd2cc) House of the Dead

[-](#_ssiemceczw16) Tron

[-](#_31bxzkfeuvl6) Call of Duty

[The elevator Pitch](#_337xnergkz1b) 19

[Project Description (Brief):](#_z7oe7x50rpf3)

[Project Description (Detailed)](#_exbmsy55zuvb)

[What sets this project apart?](#_s4h84uy3suza)

[Core Gameplay Mechanics (Detailed)](#_a8x4s87df6uk)

[-](#_jyik8zbcjcio) Gun Tracking

[-](#_y46mn9zee60t) World Generation

[-](#_lmzwvmw5e0hr) Enemy Generation

[Story and Gameplay](#_6pmf08ssy6y0) 20

[Story (Brief)](#_ctv1wxi9dpll)

[Story (Detailed)](#_kqt2h5q76zyt)

[Gameplay (Brief)](#_ejtq4v6r30ui)

[Gameplay (Detailed)](#_cl69l94amjmx)

[Assets Needed](#_6m1256af7s3j) 21

[- 2D](#_1wb69txjqarm)

[- 3D](#_xdk2cy4n4ovn)

[- Sound](#_f8xx8iwg5gs9)

[- Code](#_ky1qxs88utre)

[Schedule](#_kmt9zaowjejr) 22

[-](#_r3fjjzh8krjg) Sprint 1

[-](#_j584764hn4bz) Sprint 2

[-](#_lbj31oz0xb3v) Sprint 3

[-](#_p0jgh8xq0o3r) Sprint 4

Concept Art 23

# 

# Overview

## Theme / Setting / Genre

- Action

- Virtual Reality

- Rail Shooter

## Core Gameplay Mechanics Brief

- Gun Tracking

- World Generation

- Enemy Generation

## Targeted platforms

- PC

## Project Scope

- Less than 200 euro

- 4.5 Months

## Influences (Brief)

### House of the Dead

Arcade Rail Shooter Video Game

- House of the dead was a series of rail shooters developed by Sega beginning in the early nineties. As the series progressed they had a focus on novelty in their gun controllers. Furthermore the house of the dead series ignored the cover mechanics popular in the genre in favor of constant progression. This series is a close match for Virtual Entanglement in terms of its functional restrictions and requirements.

### Tron

Sci-Fi Action Adventure Movie

- Tron is a major stylistic influence, the dark panels and clothes on everyone and everything with bright stripes of color is both relatively easy to implement and extremely useful in terms of clueing the player into the intent of different in game entities. The clean high contrast visuals achieve an excellent economy of time required to aesthetic achieved. This is strictly a stylistic influence, as Virtual Entanglement will not have the budget or the time to introduce any real story elements.

### Call of Duty

First Person Shooter Video Game

- Call of Duty is one of the most popular first person shooter series of all time. It stripped away many of the mechanics common to first person shooters, in favour of streamlining the core combat engagement. Specifically it introduced regenerating health, which allows for the player to experience peril when their health is low but they can return to the fray after a few seconds of hiding away to restore their health.

## The elevator Pitch

-VR Tron with a gun instead of a bike.

# What sets this project apart?

- Virtual Reality

- Specific Controller plugin

- Procedural Generation

## Core Gameplay Mechanics

### Gun Tracking

-Players will use an in game gun mapped to a real life gun controller, this will be their only way to interact with the game world.

It will use the Microsoft Kinect to tack the position and a BNO055 sensor to track the orientation of the gun.

### World Generation

- The maps players progress through will be procedurally generated each time the game is played. This will keep subsequent playthroughs engaging and prevent players from simply learning off the best way to deal with each area. A path finding script will set the players path through the game world which they will progress along at a constant rate.

### Enemy Generation

- Enemy type and placement will be chosen by a procedural generation algorithm. Certain enemies will have a higher chance to spawn in certain areas, and the frequency and difficulty of enemies spawned will increase with time.

# Story and Gameplay

## Story (Brief)

You are tasked with finding the root of a virus in the computer and destroying it.

## Story (Detailed)

You are a member of the Virtual Corps branch of the military, a select unit specialising in the use of the Entanglement Rifle, or the Ent Rifle for short, to protect vital computer systems from invasion. This rife exists in both the real and the virtual world, allowing members or the corps to interact directly with the virtual world. Players take control of members of this corps from the moment they enter the virtual world.

## Gameplay (Brief)

The player will progress along a track and fight off procedurally generated enemies in a procedurally generated environment.

## Gameplay (Detailed)

Gameplay can be split into two main sections, the tutorial and the core game.

Tutorial:

The player will being in a room with a closed door. This room will contain a window, through which the players can see people with guns similar to their own being attacked by the games enemies. In order to progress the player must shoot the door, this proves beyond doubt that the player knows how to shoot their gun. Once the player has left the starting room they themselves will be met with some the games enemies, as many enemies as the guns clip can hold, and another closed door. If the player can make it through this room we know they can both shoot and reload, both of the required skills to progress through the game.

Core:

The player will continue along a set track at a set rate. The world they progress through will be procedurally generated, with procedurally generated enemies placed throughout. The player will continue along their path with enemies attacking them when they get to close, they will have to shoot the enemies before their attacks land on the player.

When they are hit the players will lose health, which will be represented by their vision darkening and bloodlines being added at the edges of their vision. After a couple of seconds of not taking damage the player will recover any damage taken.

Players can gain combo points for each enemy slain without taking damage, these will both increase the points gained from killing enemies and increase the scale of the enemy’s death animation, providing both a functional and perceived value to the combo points.

# Assets Needed

## 2D

- Textures

- HUD Elements

## 3D

Characters List

- Flying Enemy

- Large Enemy

- Crawling Enemy

Environmental Art Lists

- Skybox

- Rooms

- Corridors

## Sound

Sound List (Ambient)

- Music

- Enemy Grunts

Sound List (Player)

- Character Hit / Collision Sound list

- Pained grunt

- Character on Injured / Death sound list

- Game over theme

## Code

Character Scripts (Player Pawn/Player Controller)

- Path finding script

- Movement script

- Object tracking script

- Gun controller script

- HUD script

- Player state/stats script

Ambient Scripts (Runs in the background)

- World generation script

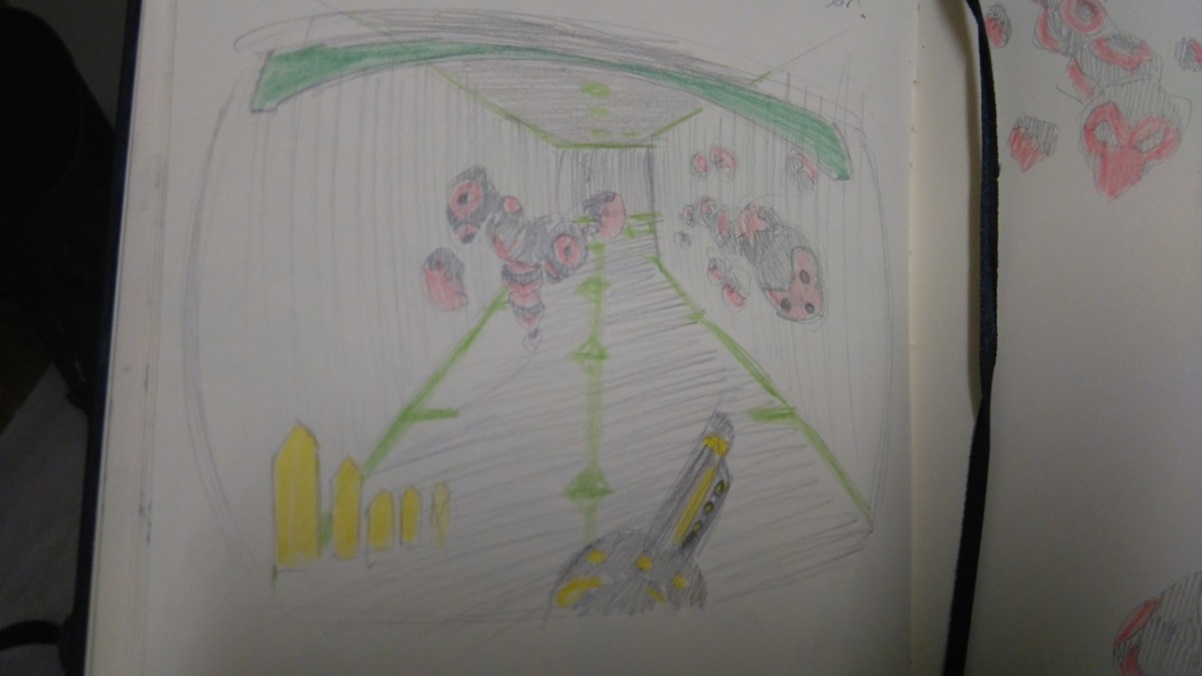
- Enemy placement script

NPC Scripts

- Wander script

- Attack script

**Concept Art**

**In game concept with heads up display (HUD)**

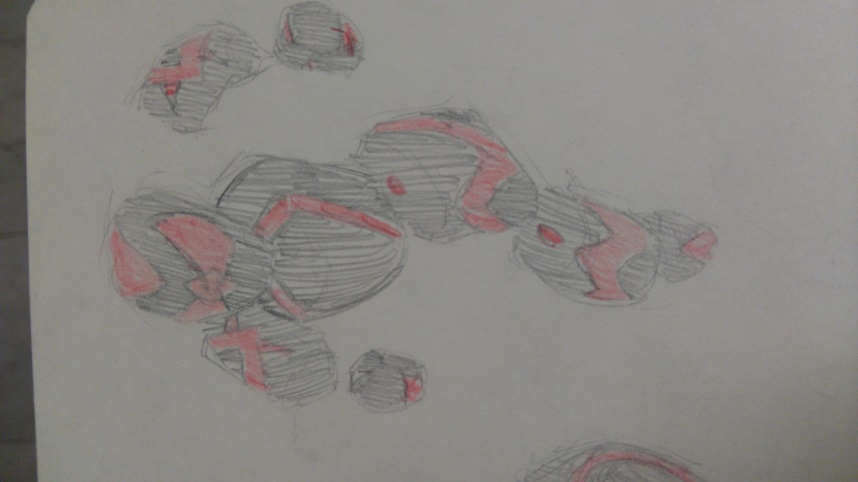
**In game concept without HUD**

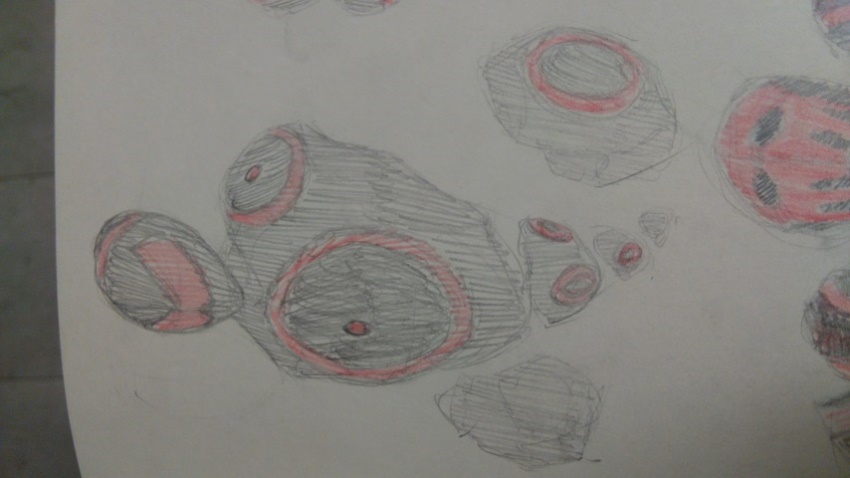
**Enemy concept art**



(Above: Large three headed enemy. Possibly a boss)

(Left: Crawler concept. Can crawl on walls and ceilings)

(Left: Ranged enemy, can fire ranged projectiles at the player.)



(Right: Melee enemy, can walk at player and damage them on contact.)