**Wroughtten**

**Final Report**

**Oisín Ivory**

BSc (Hons) Entertainment Systems

20083797

*22/04/2022*

**Table of Contents**

1. Abstract 3

2. Introduction 3

2.1 Game Concept 3

2.2 Unique Selling Points 3

2.3 Genre 4

2.4 Completion 4

2.5 Distribution 5

3. Features 5

3.1 Weapons 5

3.2 Weapon Bolts 5

3.2.1 Closed Bolt 5

3.2.2 Open Bolt 6

3.2.3 Bolt Action 7

3.3 Reloading system 8

3.3.1 Abstract 8

3.3.2 Magazines 8

3.3.3 Clips 9

3.4 Interaction & Storage 10

3.4.1 ‘Hands’ System 10

3.4.2 Storage System 11

3.5 Artificial Intelligence 11

3.5.1 Idle 12

3.5.2 Investigate 12

3.5.3 Patrol 12

3.5.4 Combat 12

3.5.5 Cover 13

3.6 Database 14

4. Risks, Problems & Learning 14

4.1 PHP 14

4.2 Database 15

4.3 Interfaces inside Unity 15

4.4 Animation System 16

5. Project Plan, Scope & Development 16

5.1 Development Methods 16

5.2 Asset Creation & Technologies 17

5.3 Asset Creation Pipeline 18

5.4 Experimental work 19

5.5 Audio 19

5.6 Playtesting 19

6. After WIT 20

6.1 Gameplay 20

6.2 levels 20

6.3 Weapons 20

7. Conclusion & Evaluation 21

7.1 Conclusion 21

7.2 Evaluation 21

7. SCRUM Reports 22

**1. Abstract**

Wrougtten is a first-person shooter which features unique weapon and item interaction.

**2. Introduction**

Project Rationale

This project was made as an homage to virtual reality shooters. It was my attempt to re-create the interaction players normally have with weapons and items in VR in a non-VR way.

There have been some games that have attempted to recreate this interaction namely Receiver (Wolfire Games). However, I hoped to make a faster paced game which features intelligent enemies as the enemies featured in Receiver are static turrets and incorporate more inventory management.

2.1 Game Concept

The game is based somewhere in the Soviet Union, where the player is tasked with shutting down a doomsday cult which is believed to be developing nuclear weaponry. The player starts with only a handgun and is tasked with reaching the end of the level.

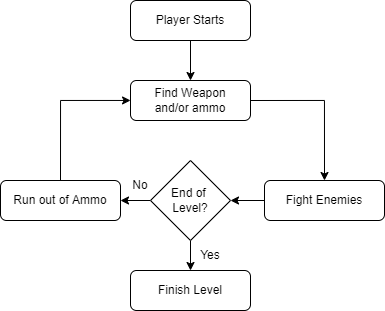
As there will not be enough ammo provided at the start, the player must salvage weapons and ammo off enemies to continue fighting through the level. Players must manage their ammo using an inventory system in which they can store and retrieve bullets and magazines for various weapons.

Figure 1 Game Loop

As players will be using many different weapons throughout the game, they must learn how to use and operate different weapon systems. All the weapon systems are simulated in the game and require the player to interact with them to use the weapon.

Players can manipulate the weapons bolt, select fire modes, load bullets manually into certain guns, use magazines in weapons and even use stripper clips all using a dynamic reloading system.

2.2 Unique Selling Points

The unique selling points of this game are the weapon and item interaction as there are few non-VR titles that implement these sort of weapon mechanics.

Where Receiver features exclusively handguns, this game will feature a broad array of weaponry and include diverse ways of interacting with weapons. It also offers a faster paced gameplay when compared to other games that feature such weapon manipulation.

2.3 Genre

While the weapon interaction will be kept grounded, the gameplay is closer to a ‘boomer shooter’, a hyper specific genre where the gameplay is more akin to shooters created in the late ‘90s such as Doom and Quake (id Software). These games are often characterized by their fast movement and having hordes of enemies and a wide variety of weapons. As of recent there has been a surge in the genre, with new games being released following these designs, such as Dusk (David Szymanski), HROT (Spytihněv)

Figure 2 Dusk by David Szymanski

2.4 Competition

The competition for this game can be split into two categories, games that feature manual weapon manipulation and games that are ‘boomer shooters’.

*Boomer Shooters*

Players that enjoy fast paced gameplay with a wide selection of weapons will have many different games to choose from, some competitors would be Dusk (David Szymanski) and Postal: Brain Damaged (Hyperstrange). These are just a few examples as there are many other games in the genre.

*Weapon Manipulation*

Figure 3 Hotdogs Horseshoes & Hand Grenades by Rust. Ltd

There are very few games that would fit into this niche genre outside of virtual reality games. The main games that this would be competing against are VR titles such as Hotdogs, Horseshoes, & Hand Grenades (Rust. Ltd), Pavlov VR (Vankrupt Games) and as mentioned earlier the non-VR title, Receiver (Wolfire Games)

2.5 Distribution

The main distribution for the game will either be on GitHub or Itch.io as a free to play game in its current form. However, if development continues past and the game becomes more refined, and more features and levels get added then there is always the possibility of a release on Steam (Valve Corporation), the most popular gaming platform for games on PC.

**3. Features**

3.1 Weapons

Throughout the game the player will come across many different weapons and weapon systems, each one with its own unique interactions. In Wroughtten using a bolt action rifle is much different than using a closed bolt weapon.

Each weapon has its own configuration of parts that make them unique, this could be using internal/external magazines, the weapon being semi-automatic or automatic and the bolting systems, close bolt, open bolt, or bolt action.

A weapon in the game is a combination of fire-modes, magazines, and the bolt system the weapon inherits. Weapons feature a fire selector, bolt, and magazine. They can also have additional features, such as a mounting point for stripper clips and the option for internal or external magazines.

3.2 Weapon Bolts

These are the main components that define how the weapon will work. They all work off an interface, allowing them to be swapped very easily, as the weapon will only interact with the interface's functions.

They also inherit common features that are present in all variants of a bolt. An example is 2 parameters, a minimum and maximum bolt distance to collect a round. Another is the held field; this determines if the bolt will continue to spring forward to be stopped. This allows for the user to easily manipulate the bolt without fighting against the spring.

Other parameters that are seen throughout the bolts are a parameter which is the area the bolt will eject a round at, once it goes past this it will eject a round if present. And a locked parameter which will freeze the bolt, not allowing it to go past the specified point.

*3.2.1 Closed Bolt*

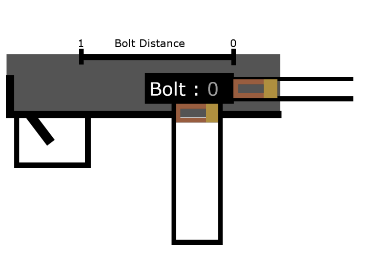
One type of weapon system in the game is closed bolt weapons. These bolts characteristics are that when there is a round ready to fire, all works parts of the weapon are forward, pulling the trigger causes the round to go off, by a firing pin or striker hitting the round.

Figure 4 Bolt starts off closed with a round in the chamber

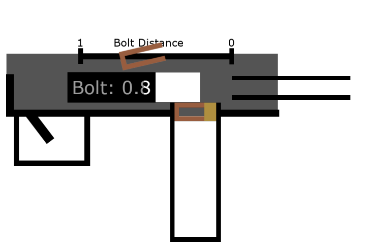
The recoil then causes the bolt to move backward, where the round is then ejected. Once the round is ejected and the bolt is pushed forward by a spring, the bolt then collects another round from the magazine, and is then ready to fire again.

Figure 5 Round is fired, recoiling the bolt backwards and reaching the ejection point

Graphical user interface, application, Teams

Description automatically generatedThe steps used to re-create this mechanism were as follows. The bolt goes into a stage of being able to accept a round, this is often when the bolt is pulled all the way back. As the bolt springs forward and passes between the collection points it will collect a round if it can.

Figure 7 Round is caught while to the bolt moves forward

Figure 6 Bolt makes it all the way back, can now take a new round

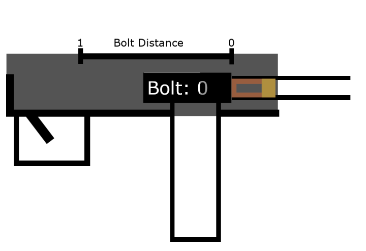
Once the bolt is all the way forward and has a round, the player can then shoot the round. The bolt then recoils all the way back. Once the bolt reaches the end, the cycle will begin again.

Figure 8 Round is chambered ready to fire again.

Only live rounds will trigger the recoil process, if a dud round is put in the bolt will attempt to fire it, if it does not successfully fire, then the bolt will not recoil and must manually be cycled using player input.

*3.2.2 Open Bolt*

Another type is open bolt. The main characteristics of this type are that the components are instead held to the rear of the receiver when ready to fire.

Figure 9 Bolt is held open waiting for the trigger to be pulled.

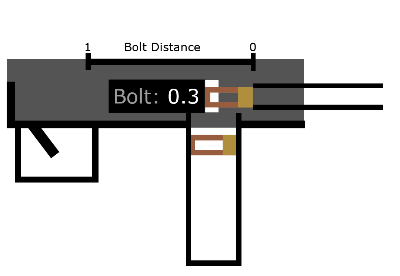
When the trigger is pulled the bolt will then travel forward and collect a round from the magazine, it is then chambered and fired. This recoils the bolt back where it is caught by the sear stopping it from sliding forward again.

Figure 10 Trigger is pulled, bolt slams forward collecting a round on the way.

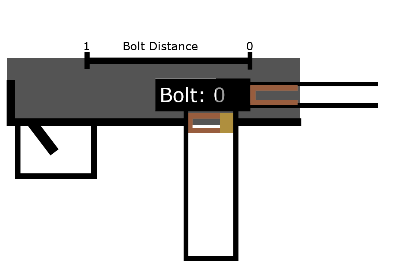
To replicate this, I used a similar setup as the closed bolt. Where the bolt has the parameter can accept round. This is set as true when the bolt is all the way back. When the bolt is drawn back it is locked at a specific stage using the locked parameter. This lock is lifted when the bolt is fired.

Figure 116 Round is chambered and instantly fired.

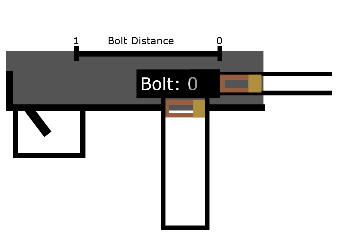
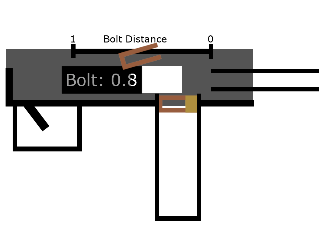
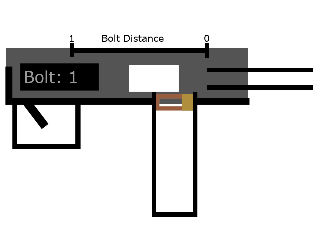
As it travels forward into the accept area of the bolt, it will then collect a round, then once the bolt is closed it will automatically fire the round, the bolt is then recoiled back. It then passes the ejection threshold which causes the casing to eject before finally reaching the back, where it locks the bolt back until ready to shoot again.

Figure 12 Bolt recoil back, ejecting the round when reaching the designated area.

Figure 14 Bolt is closed with round in the chamber

Again, if the round is a dud, then the bolt will not recoil and must be manually cycled.

Figure 13 Bolt is locked open by sear, ready to fire another round.

*3.2.3 Bolt Action*

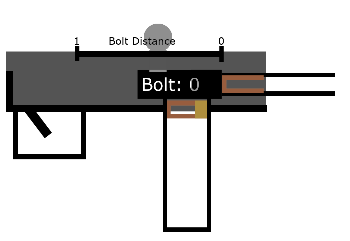
The final bolt type implemented is bolt action. This type of system is much more of a primitive system than the others. Starting open the bolt is pushed forward, collecting a round from the magazine, once the bolt is completely forward, it is then rotated, locking the bolt closed.

Figure 15 Round is fired, then bolt is rotated 90 degrees.

Then the trigger is pulled firing a round, the bolt must then be rotated and pulled back, once it is back far enough the casing is ejected and it is ready to take a new round.

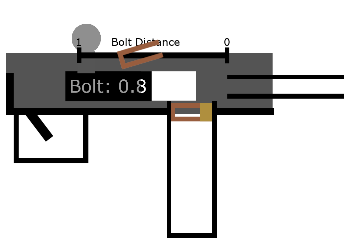
This bolt is different in many ways. Namely this bolt has 2 stages, stage 1 is the rotation of the bolt and stage 2 is pulling the bolt back and there is no recoil system.

Figure 16 Bolt is then pulled back, ejecting the round on the way.

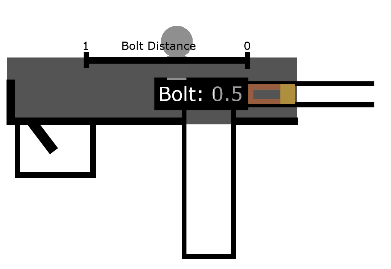
My implementation of this was done as follows; It keeps the ejection and collection variables as close bolt and open bolt. The total bolt progress is split into 2 sections, stage 1, and stage 2. During stage 1 when the bolt is closed, it will use the mouse's X input to rotate the bolt, once it is rotated completely it will then begin using the mouse’s Y input to draw the bolt back.

Figure 17 Bolt is pushed forward, catching, and chambering a round on the way.

Graphical user interface, application

Description automatically generatedOnce the bolt is completely back it can then collect a round. The player moves the bolt forward and closes it, then the weapon is ready to shoot. Once the round is fired, there is no recoil system so the player must then rotate the bolt and draw it back, ejecting the casing and becoming available to take a new round.

Figure 18 Bolt is rotated downward and is ready to fire again.

3.3 Reloading System

In Wroughtten the reloading system is designed to be as flexible as possible. Objects are split into 2 categories, loadable and reloadable. The premise is that a loadable can be loaded into a reloadable.

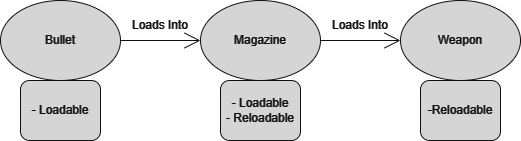
**They are reloaded by using linear interpolation over a set time, where it brings the loadable to the reloadable’s designated loading position. The loadable is detected using a box collider set up as a trigger.

Figure 19 Overview of the reloading system

***3.3.1 Abstract***

*Loadables*

These are the objects that get loaded into a reloadable. An example is bullets into a magazine. There are 3 types of loadables, Bullets, Clips and Magazines. These are used to determine the position, they move to when being loaded.

*Reloadable*

Reloadables are objects that can be reloaded. Reloadable objects take parameters for what type of loadable it will load, and the position that loadable to move to when being loaded. Loadables can also be reloadable, an example being a clip. The player can load bullets into the clip, which can then be loaded into a weapon.

***3.3.2 Magazines***

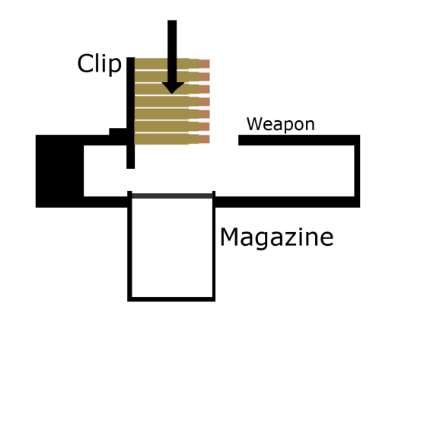
*Magazine*

Magazines are where bullets are stored, they can feed bullets into a weapon when attached. They make use of a stack to feed rounds into a weapon, following first in last out, just like their real-life counterparts.

Magazines also work off an interface as there are 2 types, internal and external. Internal magazines cannot be ejected from the weapon, whereas external magazines can.

*Magazine Acceptor*

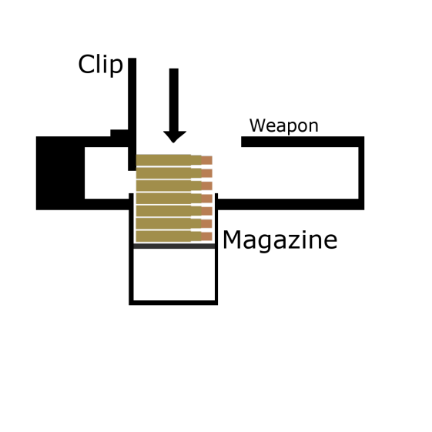
This is what is used to control the currently attached magazine if the weapon uses detachable magazines. When magazines collide with the magazine acceptor it will parent it to the weapon and then set its position to line up with the weapon. It also updates the weapons current magazine and will feed ammo from it. It can also be used to eject magazines.

***3.3.3 Clips***

*Clip*

Clips are a type of loadable used in certain weapons. They store a quantity of bullets and can all be fed into a magazine. They are often seen in weapons with internal magazines. Clips are loadable and reloadable, as the player loads bullets into them, before loading it into the weapon. Their purpose is like that of a magazine, except instead of feeding rounds into a weapon, it feeds rounds into magazines.

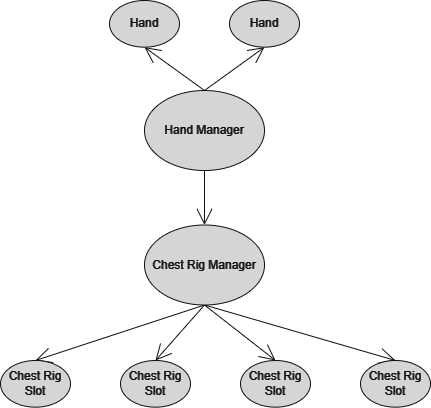
Figure 20 Clip Attached to Weapon

*******Clip Acceptor*

Clip acceptors are used to interact with clips and attach them to weapons. Once a clip is attached, if the reload key is held it takes the player's mouse's Y input. As the player drags their mouse down, it will begin feeding rounds into the weapon. To remove the clip, the player must flick their mouse up while holding R, this will then eject it from the weapon.

Figure 21 Hold reload and drag down to load magazine

3.4 Interaction & Storage

**Wroughtten features item interaction and storage. Players can pick up items using a system that I have named the ‘Hands’ system. With this system the player can pick up objects in either their left or right hand.

**3.4.1 ‘Hands’ System**

*Hand*

When picking up items, they are made as a child of the hands and their local position is set to the origin of their parent. This means moving the hand also brings the object. Picking up items will disable their rigid bodies, stopping them from being affected by gravity. The hands are controlled by a manager.

*Hand Manager*

Figure 22 Hand manager makes requests to the hands and chest rig manager

*Objects*

The hand manager controls the hands positions and the objects in them. It also allows for the ability to move hands around. The hands contain functions that allow them to use linear interpolation to move to specified positions.

*Reloading*

When pressing the reload button the hands check to see if the item in the left hand is reloadable, and then if the item in the right hand is loadable. If these conditions are met, then the hand will move to the specified position for the loadable. This will cause the item to collide with the reloadable’s box collider and the item will be loaded.

This system works the same for the clips as well, however instead of loading anything it will attach to the clip acceptor.

*Weapons*

Weapons work differently to regular objects, as they must be positioned and moved when going from hip-fire to aiming down the sight (ADS). Weapons are manipulated by picking them up in the left hand.

The weapons store a hip-fire position and an ADS position. The player can only ADS while the right hand is empty, this will move the weapon to the designated position.

The reason the weapon is moved rather than the hand, is that the recoil system implemented moves the hand to display recoil. So, when a round is successfully fired, the hand quickly moves to an offset before slowly returning to the original position.

***3.4.2 Storage System***

Items that the player wants to keep are stored in the chest-rig. The chest rig consists of 2 key features, the chest-rig slot, and the chest-rig manager. The manager is controlled by the hand manager.

*Chest-Rig Slot*

The chest rig slot acts the same as the hand, they store items in their slots, where the chest-rig manager can then retrieve and store items from them.

*Chest-Rig Manager*

Upon wanting to retrieve or store an item, when the key is pressed relative to what slot the player wants to interact with, it will either retrieve the item to either the right hand, or if it is occupied by an item, the left hand.

Items are retrieved by enabling the rigid body after they are stored, then returning the object to the hand manager, where the object is assigned to a hand and is then parented to the corresponding hand before finally being zeroed.

3.5 Artificial Intelligence

**Chart, diagram, funnel chart

Description automatically generated**The AI in wrougtten is designed to be very configurable, allowing designers to change many options to make enemies act differently from one another and give them different weapons and loot.

Figure 23 AI BehSaviour Tree

The AI are split amongst 4 different states, which control what they do, and they also have substates for other actions within those states. The AI have behavior that they will carry out regardless of state, such as searching for enemies, and keeping track of their target if they have one.

AI can be set to default to one of the states and any specific substate. If their default state is patrolling, they will patrol an area, and if their substate is seeking cover, then when in combat they will firstly seek cover.

*3.5.1 Idle*

This is the base state, where AI will stand in the one position listening and looking for any targets. Once they identify an enemy, they will then enter the combat state.

*3.5.2 Investigate*

During the investigate state, the AI will walk to a designated area, this could be the source of a gunshot. As they walk there, they will still be looking and listening for any enemies. If they find an enemy, they will swap to the combat state.

*3.5.3 Patrol*

The patrol state will make it so that the AI will follow a defined path. While patrolling they will be searching and listening for any enemies. Once they identify an enemy, they will then enter the combat state.

*3.5.4 Combat*

During the combat state, the AI will go into one of 3 combat substates. During these substates they will either maneuver around the area or attempt to attack the player.

*Shoot*

The shoot state means that the AI will attempt to attack you. During the shoot state the AI will raise their gun and fire at the player. They enter the shoot state when they have line of sight of the target and are within range. Upon losing line of sight, they will then enter the advance state.

There are a few parameters and factors that can be changed, such as how long a burst of gunfire lasts and the time between bursts. The weapon they use also affects how they act, such as the number of rounds they can fire before reloading.

*Advance*

During the advance combat state, the AI will walk towards the target. Upon getting a line of sight on the target and they are within range, they will then swap to the shoot state. If the target leaves the line of sight, they AI will advance towards their last known location.

After a time, they will exit the combat state and return to their original state, be it patrolling or idle.

*Cover*

During this state, the AI will search for cover, once it has found suitable cover it will walk to the cover, once at the cover it will then go to the shoot state.

*3.5.5 Cover*

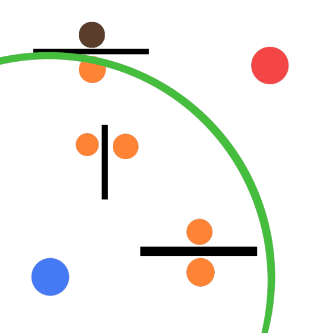
The cover system made for the AI was based on other games AI systems, namely Killzone (Guerrilla Games) and Hotdogs, Horseshoes & Hand grenades (Rust. Ltd). This system works on the premise that there are cover points placed throughout the map.

Figure 7 AI checks in a radius for cover points

These cover points consist of 2 parts, the vision evaluation, and the cover evaluation. These points dictate whether the cover point is good. This is done by casting a ray between the points and the target.

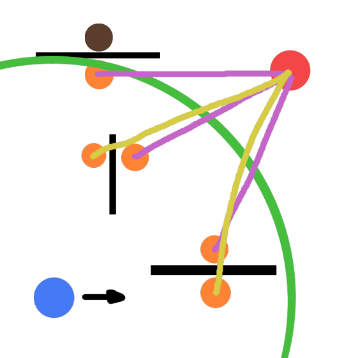
The best-case scenario for the cover point is that it has vision of the target, but the ray cast for cover is obstructed. The AI will evaluate cover points within a certain radius and will choose the best from the pool. In the event there is no cover point, they will return to the advance state.

Figure 8 AI evaluates each cover point based on cover and line of sight, then moves to the closest cover point to the AI

While the AI is in combat, they will manage what they are doing regarding cover via Enums, like how combat has its substates.

*No Cover* *point*

During this state, the AI will search for nearby cover. If there are no cover points nearby, then the AI will either shoot if they have a line of sight on the target, or they will advance toward the target. If they do find suitable cover then they will begin pathing towards it.

*Not at Cover*

While the AI is in this state, they will path towards the cover they are trying to get to; however, they will also continue searching for a better cover point, and if they find one, they will begin pathing to that instead

*At Cover*

While at cover the AI will begin by facing towards the target, then it will change the AIs combat state from cover to shoot. While at this position, the AI should begin firing at the player from cover.

3.6 Database

The database used is an SQL Database hosted for free using 000webhost.com’s services. It also makes use of phpMyAdmin, making it easy to create and manage the database by providing an easy-to-use user interface. On the database there are 2 tables, one to store users and one that will manage the scores.

Users

The user's table is a small table that consists of a username and password.

*Usernames*

The username acts as the index of the table and cannot contain duplicates. Attempting to register using a username that is registered will result in a failed attempt.

*Passwords*

The password that is stored is a SHA256 hash of the input password. The password is hashed when registering the account. When logging in it compares the username and the hash of the input password, if a matching entry is in the database, it will return a success, logging the user in.

Scores

The scores table consists of name, time, score, kills and scene. Scores are not continually added to the table but are updated based on scene and depending on if the score is greater. Each user will ideally have a high score for each scene in the game.

This was done by finding database entries that have the same name and scene as the score that was being uploaded. If there is an entry it compares the score and uploads the highest, else it will create a new entry.

Logo

Description automatically generatedThe game interacts with the database using HTTP requests. On the website there are 4 PHP files that carry specific functions. Two of the files manage registration and logging in. The others manage score uploads and get scores from the database.

**4. Risks, Problems & Learning**

4.1 PHP

Prior to doing this project I had never learned any PHP in any modules. While creating the project one module gave a very brief overview of PHP, so I decided to do more research into it. While the syntax was like C#, I was using it to parse the database.

While creating the PHP scripts for interacting with the database I had to research and learn how to parse databases in PHP. My method of parsing was using MySQLi, which is an improved version of the MySQL functions for PHP. It allows the creation of a query that can be used in the database.

4.2 Database

My original plan was to use Google’s Firebase to store scores, however I wanted to do something that was a bit lower level than that. I decided on using a MySQL database to store the users and score. To host this online I chose to use a free server provided by 000webhost. As mentioned earlier it also comes with phpMyAdmin, which made it much easier to manage the tables.

I had never incorporated any sort of database into a Unity project before, so I found this highly informative.

4.3 Interfaces inside Unity

I had never really needed to incorporate interfaces into any Unity project I had worked on before, however the systems I designed would really benefit from implementing them as interfaces to keep things as modular as possible.

While I already know exactly what and how to use interfaces, the issues arose when trying to use them inside Unity. For some reason, the Unity interface does not support the use of interfaces therefore, to use interfaces inside Unity, they must manually be assigned in scripts.

My solution to this was that the weapon’s bolt will store a reference to the weapon. When the item is spawned or the game begins, the bolt will set the weapons bolt to itself. This was the best way I found of going about it.

4.4 Animation System

Once I finished developing the AI, I had to create an animation system for it. As I am not very versed in animation due to me being more interested in the programming aspects of game development, I was not sure what methods of animation could work with the AI, as I wanted the legs animations to be independent from the torso.

Figure 9 Sarge from Quake 3 Arena, his body is split into 3 sections for animating.

While there are methods like blending trees to help with animations like that I looked to older games for inspiration, where I learned about the animation system used in Quake 3 (id Software). This game’s animations are done by splitting the body into 3 sections, where each section could be animated independently of one another.

I used this as the premise for my game and decided to split the body into 2 sections. By doing this I then added 2 different animation controllers for the torso and the legs. This allowed me to animate these sections separately from one another.

To create the animations, I used Unity’s in-built animation editor. It works like a typical animator, where the user can create keyframes which save the objects position, then the object will interpolate between each keyframe. Typically, a model is rigged, where the mesh is attached to a rig, the rig is then animated, and the mesh follows it. As I wanted to re-create an older style instead of using a rig, I opted for making each section of the models limbs different, and then animated them by positioning the limbs.

**5. Project Plan, Scope & Development**

5.1 Development Methods

*5.1.1 Pre-Production*

*Mechanics*

Before any work began on the game, I first created a GDD or ‘Game Design Document’. This document went over all the details of what was to be implemented into the game, detailing the game mechanics such as the bolt mechanics and how the AI should work. The GDD is a living document, meaning as development went on changes were made and the document was updated.

Throughout the creation of the GDD was when I began researching, looking at other games that shared similar mechanics and how they went about creating them. I also created some small example scenes in Unity testing any ideas that I had.

*Graphics*

Once I was happy with the mechanics that I wanted to implement I then began researching themes and visuals that I was going to use in my game. I was originally going to try and create highly detailed models for the game, however I wanted to create everything myself. After creating one model I then looked at how much time it took to create, before deciding to go with a lower poly art style to make it easier to create assets for the game.

The main issues that came with creating highly detailed models is that there is a lot more time spent preparing the model for the game. The amount of time that is required UV Unwrapping and then texturing highly detailed models is much larger than texturing a low poly model.

Now that I knew what art style I wanted, I began researching and looking into how people achieve the style. Mainly looking at models from games such as Quake and Dusk. The best way to re-create this style is by imposing limits on what you can make, limiting the size of the textures, the colours used and how many tris the polygon can consist of.

*Development*

When choosing to develop this game I went with Unity 3D. Unity is a free game engine that is used by professionals and hobbyists alike.

Throughout developing the systems used in this project the main development method that I used was SCRUM. This allowed for rapid prototyping when developing new systems within the game.

Before creating anything, I would begin by selecting what I wanted to develop and begin planning a sprint. Using AI as an example, I would then plan what features I want implemented and then give an estimate on how long each feature would take to implement.

I would then work on implementing the features, once implemented I would record how long it took before finally writing a SCRUM report on it.

5.2 Asset Creation & Technologies

Throughout the development of the game, I have used various applications to create the scripts, assets, and systems. The technologies I have used are as follows:

Unity

Unity is an extremely popular free game engine, used by both professionals and hobbyists alike. It includes an extremely easy to use interface, features a very flexible entity component system, and an intuitive scripting system. The programming language Unity uses is C#, a very modern and easy to pick up language.

Overall, Unity was very enjoyable to work with. Any issues that I faced were quickly resolved just by viewing the documentation. The interface is very intuitive and easy to work with.

Autodesk Maya

Autodesk Maya, or Maya for short, is a 3D asset creation application. Developed by Autodesk, it is an immensely popular 3D modelling software, which often is used in professional environments. Not only does it feature a vast array of tools that can be used when modelling. It also features some of the best UV unwrapping tools of any 3D software.

GIMP

GNU Image Manipulation Program is a free, open-source image processing application. It comes with a large variety of tools to use when creating textures. While not industry standard, it has been used in many smaller indie games. I found it to be the perfect tool for creating the textures for my game, as I could easily photo bash images and use the tools to create certain patterns, such as wear on areas of weapons or clothing.

Audacity

Audacity is a free, open-source audio editor which was used to cut down some of the audio clips and make any minor adjustments. It was quite easy to use, although I was not doing anything exceptional with the audio outside of trimming clips.

MySQL

MySQL is one of the most widely used database applications throughout the software industry. I had previously covered a module on it, so I was familiar with it, however it had been a while since then. Incorporating this into my project was defiantly a good refresher on MySQL databases.

I found that creating the queries for the database is intuitive and was done with relative ease. Any issues I had were quickly resolved by looking at the documentation.

5.3 Asset Creation Pipeline

Throughout making all the models in the game, I have developed a pipeline that worked well for me. I found following this pipeline make it extremely easy to rapidly create assets for the game. My goal with the art style was to emulate the art style of older games. This was achieved by keeping the poly count low on models, keeping the texture size small such as 32px, 64px or 128px, and finally by applying a colour palette to the images. FI used the palette used in Quake.

Research

Before beginning to model anything I started off by looking at and gathering references to use while modelling. This could be as simple as just finding images of items that I will use when creating the model.

Modelling

Diagram

Description automatically generatedOnce I had gathered enough references on what I wanted to create I then created them as image planes in Maya. I then used these images to block out the item that I was creating. Once I was happy with the block out, I then went and refined the shapes to something that I was happy with. I made sure to try keep a low poly count on the models to achieve the style I was looking for.

Figure 10 Model of a PPSH-41 and its UV

Once I was happy with the model itself, I then changed to the UV editing mode. Here I began the process of UV unwrapping the model using the vast number of tools in Maya. When I was happy with the layout of the unwrapped model, I then scaled the model to the appropriate size.

Finally, I then exported the model to Unity and exported an image file of the UV.

Texturing

During the texturing phase I begin by importing the UV of the object into GIMP. I then imported any reference images. Most of the textures are made by photo bashing, a technique by which a collection of images is used to create the texture, however there were many instances where I used the tools GIMP offers to create the texture, such as using noise to generate scratches and darker patches on any pieces.

Figure 11 Texture of the PPSH-41 created using GIMP

Once I was happy with the texture, I then went on to scale the image down to the desired size. I then applied the colour palette to the image, then exported it.

Import

When exporting to Unity from Maya it produces an FBX file. To import the file, it is as simple as dragging the file into the Unity file explorer. Once in I dragged it into the scene. This was just the untextured model.

Figure 12 PPSH-41 imported into Unity and the texture applied to it.

To apply the texture, I repeated the same as above to import it into unity. Once imported I made sure to remove the point filter from the image, this made it so that the image was sharp and there was no interpolation between the pixels.

I then dragged the image onto the model, this automatically created a new material with the image and applied it to the material.

5.4 Experimental Work

This game idea was first tested by a small prototype that I had made in the summer of 2021. The protype consisted of a single weapon and the objective was to walk through a house and shoot targets. This featured a similar weapon system; however, it was much more of a simplified system.

The bolt worked off an animation and the cycling of rounds were performed regardless of the state of the bolt, i.e., rounds were ejected and collected on the press of a button rather than the bolt's position and state.

The reloading method was similar, where the bullet must pass into a trigger to be loaded into a magazine. But instead of using linear interpolation to move to a specific position, it was done via animations. This would have made it so that every weapon would need its own unique animations. It also would have been harder to manage animations for each type of loadable.

5.5 Audio

The audio currently featured in the game is for the weapon systems. Audio is played when interacting with the bolt and firing the weapon with live ammunition. Casings also play audio when colliding with walls and the ground.

5.6 Play Testing

The playtesting done was looking at the damage that weapons do relative to their rate of fire and the type of weapon. Through playing I opted to make smaller calibre firearms such as the submachine gun shoot faster but have a lower damaging bullet, when compared to the rifle, which has a slower rate of fire but more damage per bullet and more accurate at longer ranges.

**6. After WIT**

Further development of this game would mostly look at polishing what is currently implemented and adding more weapons and levels.

6.1 Gameplay

Some things that I am not a fan of now are the weapon swaying. I would look at rebuilding the currently implemented one to work off momentum, so that turning right would tilt the weapon right. Many games have implemented this system, but most of the inspiration for this comes from the game Dusk (David Szymanski).

6.2 Levels

I would mostly been looking at expanding levels, with the possibility of starting fresh. Many games in the genre follow the convention of having an episodic release of levels, where a chapter consists of a collection of levels, often following a theme.

6.3 Weapons

Adding more weapons to levels would be great, giving the player the chance of trying out new weapons. There is also the possibility of adding more weapon types, namely lever action weapons, and single or double action revolvers.

**7. Evaluations and Conclusion**

7.1 Evaluation

The game features all the intended mechanics that was planned for. While I am happy with the state that the game is in, I feel as though there is a lot more that I could add to it and build upon. Such as adding new levels and features. The main issue with the game right now is the lack of maps, as often games like this would be split up into many levels across various themes and locations. Overall, I feel as though the game achieved its main goal, that being a game with a unique weapon and inventory system.

7.2 Conclusion

I believe that the development of this game met all its goals. It features a unique and flexible reload system. The weapon system in the game allows for modularity and being able to interchange the parts.

Throughout working on the project, I have become much more comfortable with Unity and the tools that it provides, namely the animation system and the way unity handles game object positioning, transforming, and rotating.

Outside of Unity I have also become much more proficient with GIMP. I used to mainly use photoshop, however I wanted to learn the open-source alternative. In terms of 3D modelling, I’ve felt as though I am much more proficient using the modeling and UV Unwrapping tools.

**7 SCRUM Reports**

Scrum 1: Weapon Model

Sprint 1

Created SKS weapon model in Maya.

Sprint 2

Exported UV maps and used those to texture it in photoshop.

Sprint 3

Imported into Unity.

Scrum 2: Weapon Mechanics

Sprint 1

Added Bolt movement.

Sprint 2

Created Magazines, Ammo.

Sprint 3

Joined bolt and magazine in a weapon script.

Sprint 4

Added stripper clips

Scrum 3: Hand Mechanics

Sprint 1

Added ‘simple’ hands, can now pick up certain objects.

Sprint 2

Added dynamic reloading system that adjusts based on items held.

Sprint 3

Added weapon manipulation on weapon only if it is held.

Scrum 4: Chest Rig Functionality

Sprint 1

Pick up objects and place them into a chest rig slot.

Sprint 2

Retrieve objects from chest rig.

Scrum 5: Bolt Action Weapon Model

Sprint 1

Created Mosin Nagant weapon model in Maya.

Sprint 2

Exported UV maps and used those to texture it in photoshop.

Sprint 3

Imported into Unity.

Scrum 6: Bolt Action Weapon Mechanics

Sprint 1

Added new type of bolt movement.

Sprint 2

Add new bolt to model and attach relevant scripts.

Scrum 7: Health and firing

Sprint 1

Added bullet ray casting from weapon when firing live rounds.

Sprint 2

Made bullets become useless once fired.

Sprint 3

Created a simple health system.

Sprint 4

Added targets to scene that the player can fire at.

Scrum 8: AI

Table

Description automatically generated