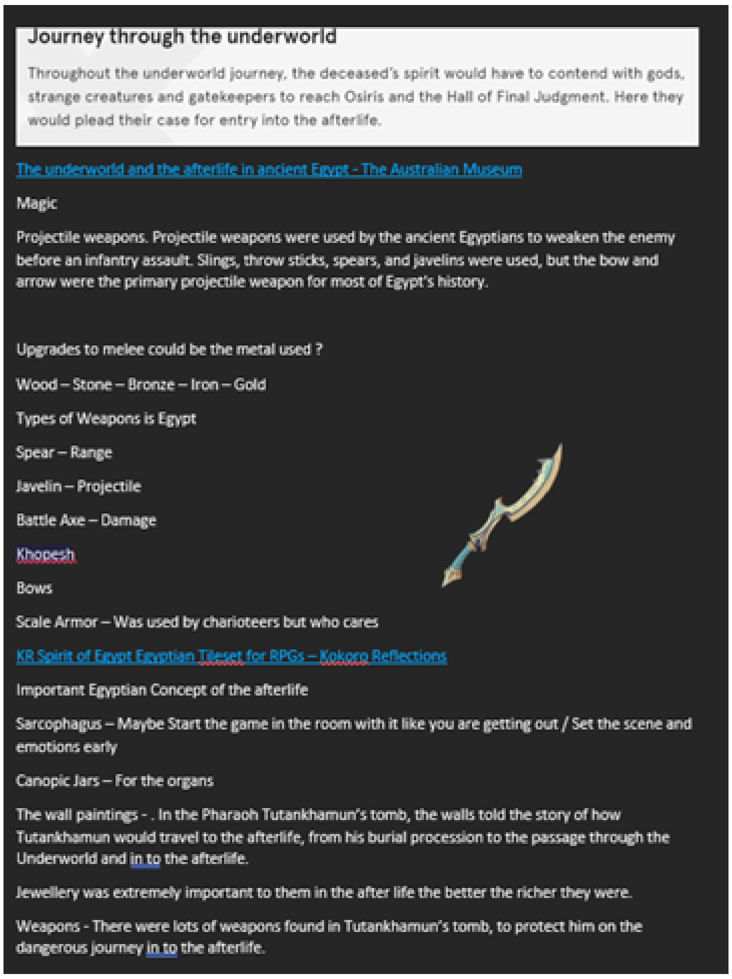
Week 1 – Forum Post

Tuesday: We started with a group chat to do with the genre style and high-level game aspects as what we wanted to do with the game. This included ideas about story and how to portray this as well as a using a PCG based system. We then started to think about the requirements that it would need and the most challenging points in the development. One of these things for development side was the sprite creation. We then discussed the engine and using OpenGL and the lower-level side of the project including a room creation tool. Roles were then assigned, and I volunteered to be Scrum Leader. For this I will do a weekly recap and review of what people have been up to.

Between Tuesday and Thursday goals for people to get knowledge able of Roguelike games was agreed upon. Mostly me as I have played many in my life and needed to update my knowledge on how they work. As well as this research into Egyptian mythology as we had discussed the story and theme of the game being based around the Gods and afterlife.



Journey through the underworld

Throughout the underworld journey, the deceased's spirit would have to contend with gods, strange creatures and gatekeepers to reach Osiris and the Hall of Final Judgment. Here they would plead their case for entry into the afterlife.

The underworld and the afterlife in ancient Egypt - The Australian Museum

Magic

Projectile weapons. Projectile weapons were used by the ancient Egyptians to weaken the enemy before an infantry assault. Slings, throw sticks, spears, and javelins were used, but the bow and arrow were the primary projectile weapon for most of Egypt's history.

Upgrades to melee could be the metal used ?

Wood – Stone – Bronze – Iron – Gold

Types of Weapons is Egypt

Spear – Range

Javelin – Projectile

Battle Axe – Damage

Khopesh

Bows

Scale Armor – Was used by charioteers but who cares

KR Spirit of Egypt Egyptian Tileset for RPGs – Kokoro Reflections

Important Egyptian Concept of the afterlife

Sarcophagus – Maybe Start the game in the room with it like you are getting out / Set the scene and emotions early

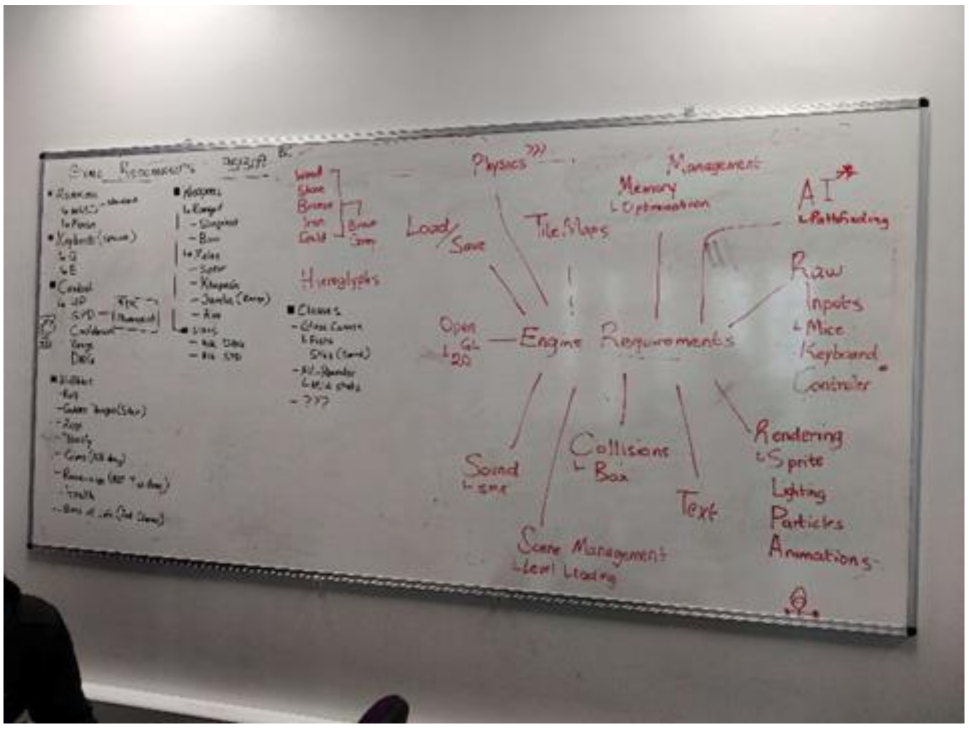
Canopic Jars – For the organs

The wall paintings - . In the Pharaoh Tutankhamun's tomb, the walls told the story of how Tutankhamun would travel to the afterlife, from his burial procession to the passage through the Underworld and in to the afterlife.

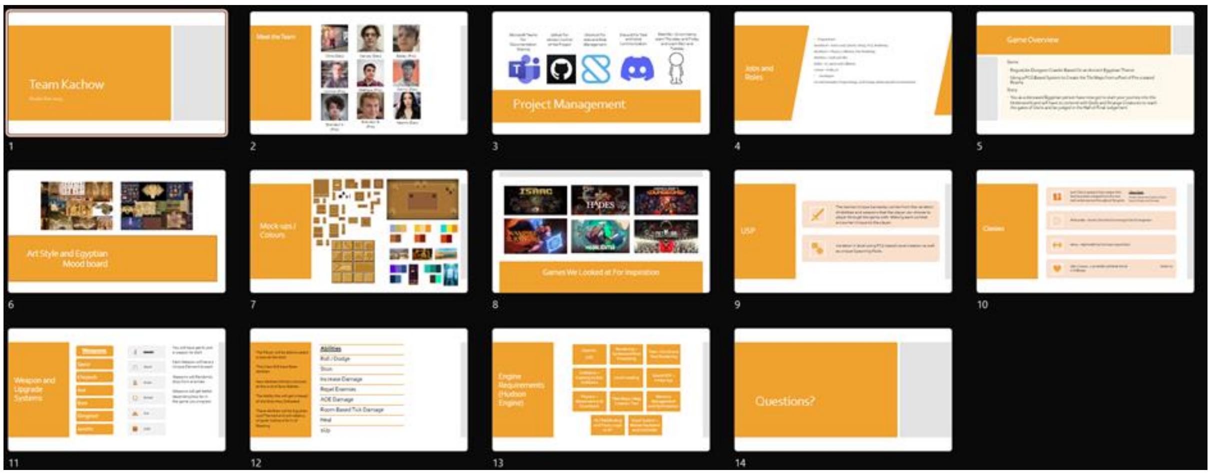
Jewellery was extremely important to them in the after life the better the richer they were.

Weapons - There were lots of weapons found in Tutankhamun's tomb, to protect him on the dangerous journey in to the afterlife.

Thursday: This was when the full team start discussing Engine Requirements and more detailed systems in the game. This was extremely important to get everyone on the same page of what we were doing. Along with more technically decision on the PCG such as room shapes and sizes and how we are going to make this work. As Scrum lead and Brandon being remote, I tried to lead this discussion and make sure everyone was getting their feedback. At the end of this day the GDD was started, and some roles were assigned in the Dev side of the team and the Tech Team had a separate detailed talk about the engine and start the TDD.



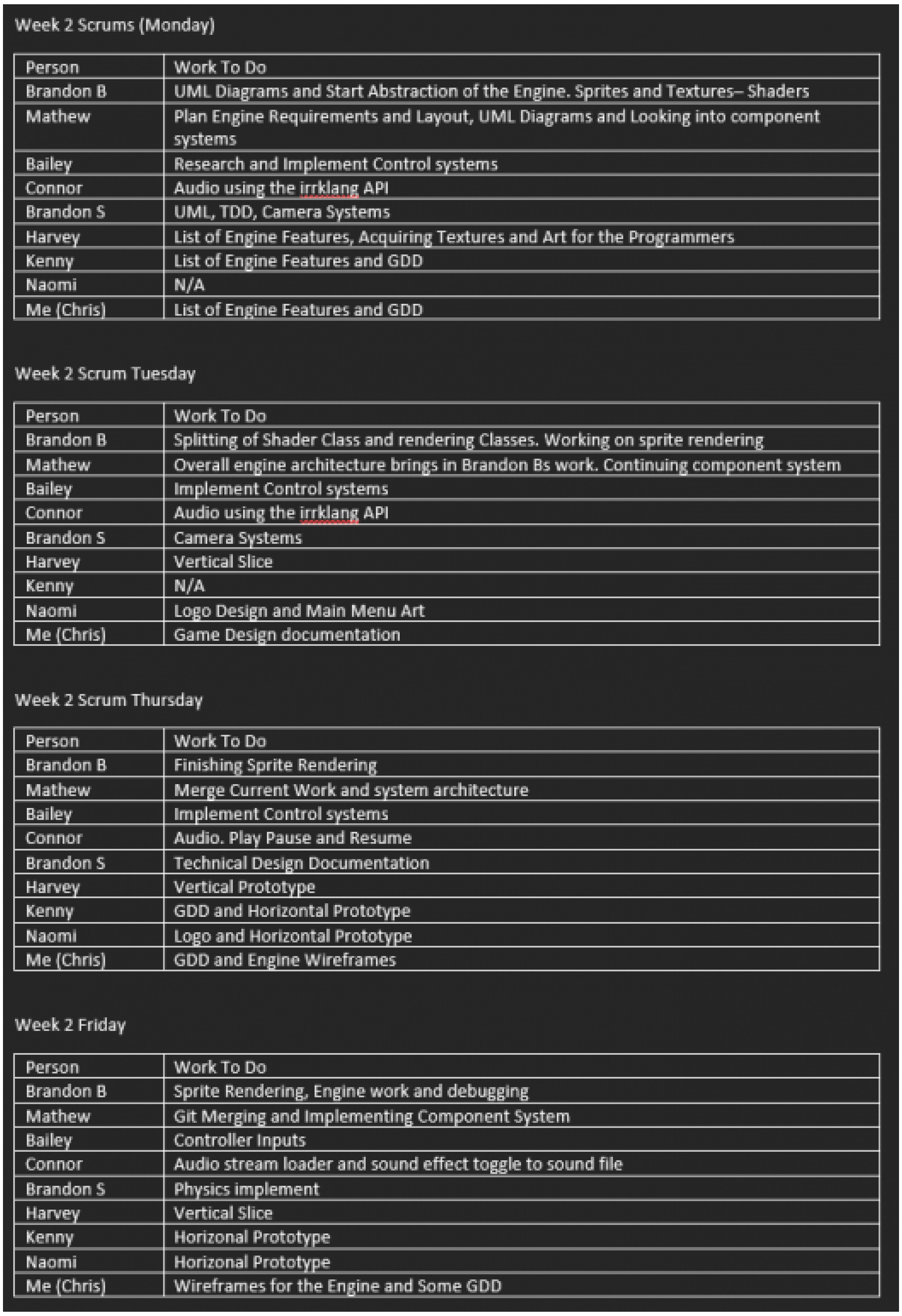
Friday: This was the play I produced the Pitch documentation. As there wasn't a lot of detail to look at on blackboard I looked back at my Mobile Dev and replicated that format again adding the Engine requirements. I had some help with the Mock-ups and Colours schemes by Harvey who had made a mock-up in Unity of what the aims for the visuals are. In the day we also created a MoSCoW document and placed each task into a column.


Team Kachow
Meet the Team
Project Management
Jobs and Roles
Game Overview
Art Style and Egyptian Mood board
Mock ups / Colours
Games We Looked at For Inspiration
USP
Classes
Weapon and Upgrade Systems
Engine Requirements (Hudson Engine)
Questions?

Friday Afternoon: In the afternoon Shaun came through and I had a conversation about Forums and Documentation. After this he suggested we had a look through last years Forums to get inspiration, during this analysis of there forums the scope of the game really became an issue. Looking over it we had planned to much as the Must section and after seeing this, Me, Harvey , Brandon S and Brandon B had a late evening call and rediscusses the scope for the engine and the scope for the final game and the MoSCoW document was amended. This will now be a large discussion point on Monday's morning meeting.

Must Have	Should Have	Could Have	**Won't** Have
Weapons 1 melee/ 1 range	Classes	Unique Boss Mechanics	Local Multiplayer

Enemies 1 melee/ 1 range	Multiple Bosses	Minimap	Difficulty
Movement	Multiple Tile Sets	Money	
Abilities 2x	Tiers Of Weapon	Shop	
Passives	Loot Pools		
Map Creation	Spawn Pools		
Menus	Scaling Enemy Stats		
HUD	Boss		
Spirits			


Week 2 Scrums (Monday)

Person	Work To Do
Brandon B	UML Diagrams and Start Abstraction of the Engine. Sprites and Textures– Shaders
Mathew	Plan Engine Requirements and Layout, UML Diagrams and Looking into component systems
Bailey	Research and Implement Control systems
Connor	Audio using the irrklang API
Brandon S	UML, TDD, Camera Systems
Harvey	List of Engine Features, Acquiring Textures and Art for the Programmers
Kenny	List of Engine Features and GDD
Naomi	N/A
Me (Chris)	List of Engine Features and GDD

Week 2 Scrum Tuesday

Person	Work To Do
Brandon B	Splitting of Shader Class and rendering Classes. Working on sprite rendering
Mathew	Overall engine architecture brings in Brandon Bs work. Continuing component system
Bailey	Implement Control systems
Connor	Audio using the irrklang API
Brandon S	Camera Systems
Harvey	Vertical Slice
Kenny	N/A
Naomi	Logo Design and Main Menu Art
Me (Chris)	Game Design documentation

Week 2 Scrum Thursday

Person	Work To Do
Brandon B	Finishing Sprite Rendering
Mathew	Merge Current Work and system architecture
Bailey	Implement Control systems
Connor	Audio. Play Pause and Resume
Brandon S	Technical Design Documentation
Harvey	Vertical Prototype
Kenny	GDD and Horizontal Prototype
Naomi	Logo and Horizontal Prototype
Me (Chris)	GDD and Engine Wireframes

Week 2 Friday

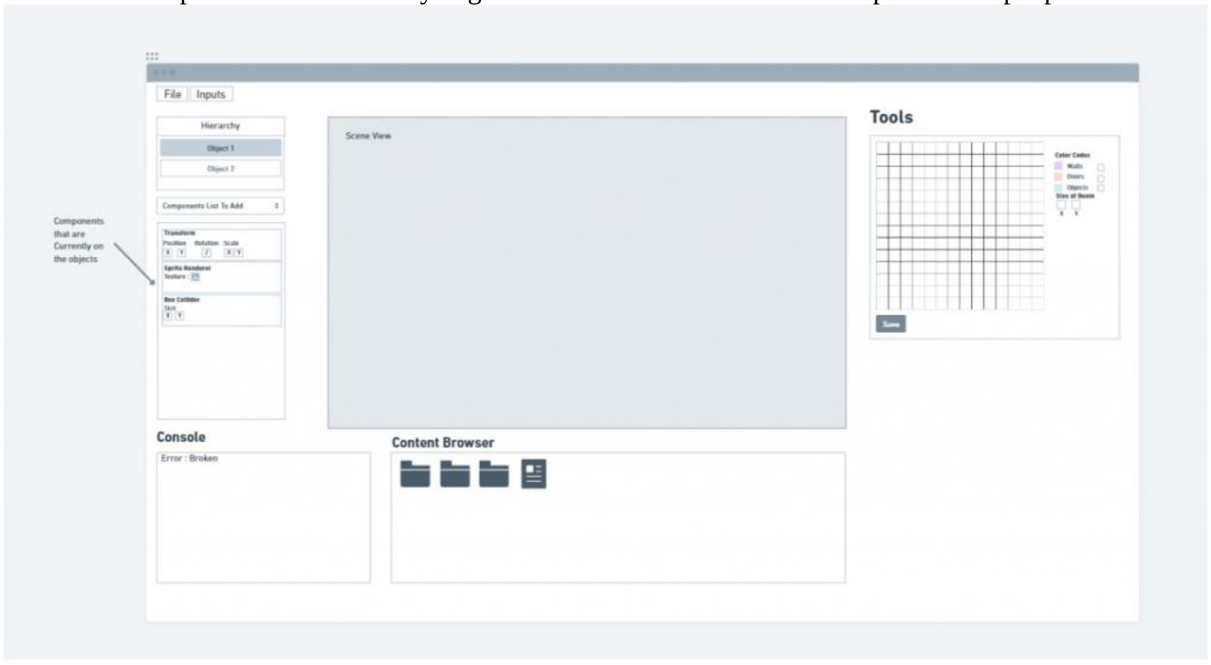
Person	Work To Do
Brandon B	Sprite Rendering, Engine work and debugging
Mathew	Git Merging and Implementing Component System
Bailey	Controller Inputs
Connor	Audio stream loader and sound effect toggle to sound file
Brandon S	Physics implement
Harvey	Vertical Slice
Kenny	Horizonal Prototype
Naomi	Horizonal Prototype
Me (Chris)	Wireframes for the Engine and Some GDD

Week 2 – Forum Post

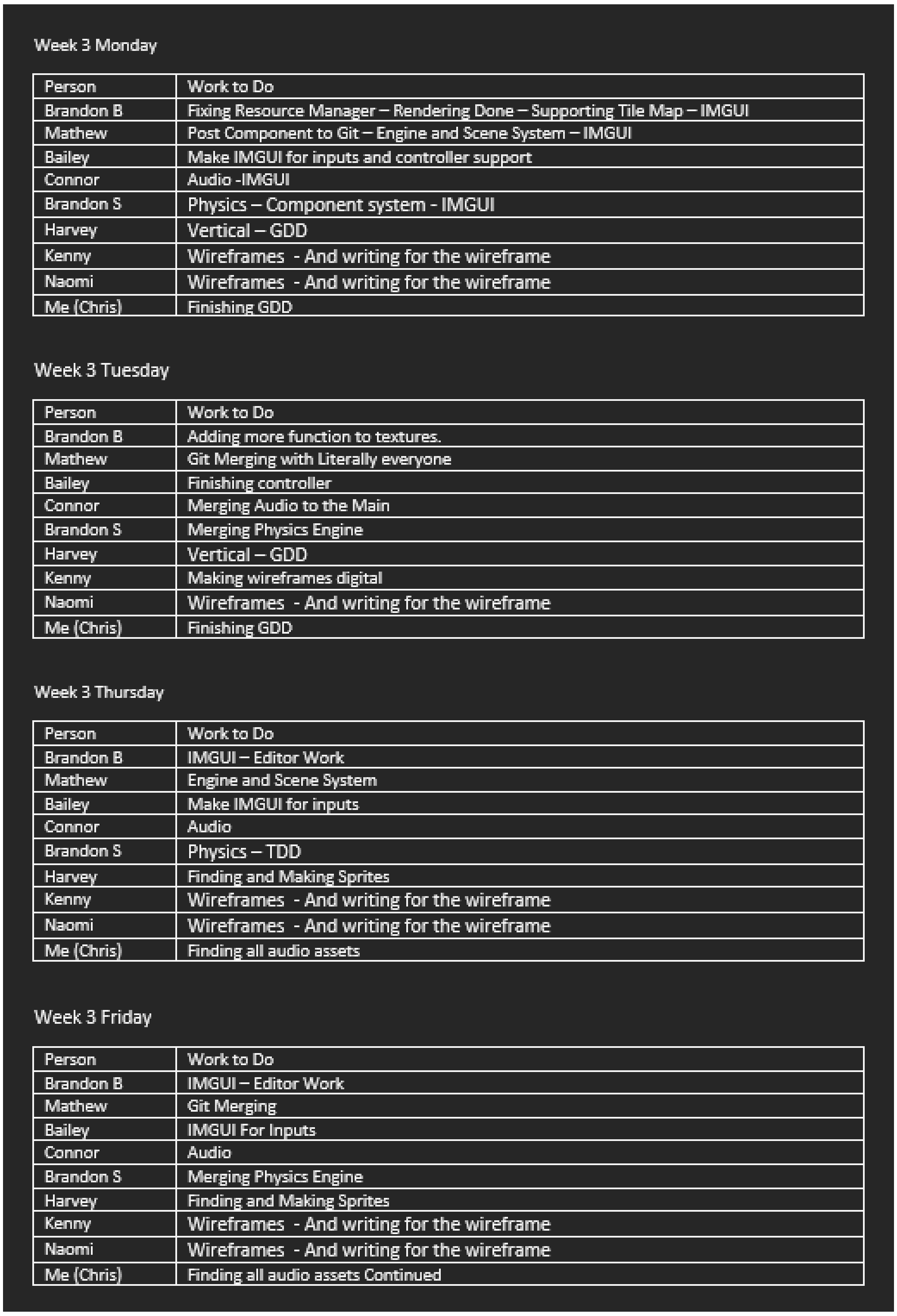
Monday: Monday started with a recap and review of the first week and as well as working out the new tasks and goals for the week. The programmers are now starting proper coding and now trying to bring the framework together and start to develop the engine UI. On Monday I first made detailed list of engine features that are going to the programmers to help them decide which ones they can produce in the project.

Tuesday : After this I start filling out the GDD documentation about the player class enemies’ weapons and outlining the story and the theme. This was a quick turnaround so we could start making crucial decisions early and support the programmers with their decisions.

Thursday and Friday : I carried on work on the GDD now getting later into the documents about the control system and health as well as this a programmers needed a wireframes of the engine overall look. I took a lot of inspiration from the unity engine and tried to make it as basic as possible so people can use it.



Scrum Meeting Week 3 Summary



Week 3 Monday

Person	Work to Do
Brandon B	Fixing Resource Manager – Rendering Done – Supporting Tile Map – IMGUI
Mathew	Post Component to Git – Engine and Scene System – IMGUI
Bailey	Make IMGUI for inputs and controller support
Connor	Audio -IMGUI
Brandon S	Physics – Component system - IMGUI
Harvey	Vertical – GDD
Kenny	Wireframes - And writing for the wireframe
Naomi	Wireframes - And writing for the wireframe
Me (Chris)	Finishing GDD

Week 3 Tuesday

Person	Work to Do
Brandon B	Adding more function to textures.
Mathew	Git Merging with Literally everyone
Bailey	Finishing controller
Connor	Merging Audio to the Main
Brandon S	Merging Physics Engine
Harvey	Vertical – GDD
Kenny	Making wireframes digital
Naomi	Wireframes - And writing for the wireframe
Me (Chris)	Finishing GDD

Week 3 Thursday

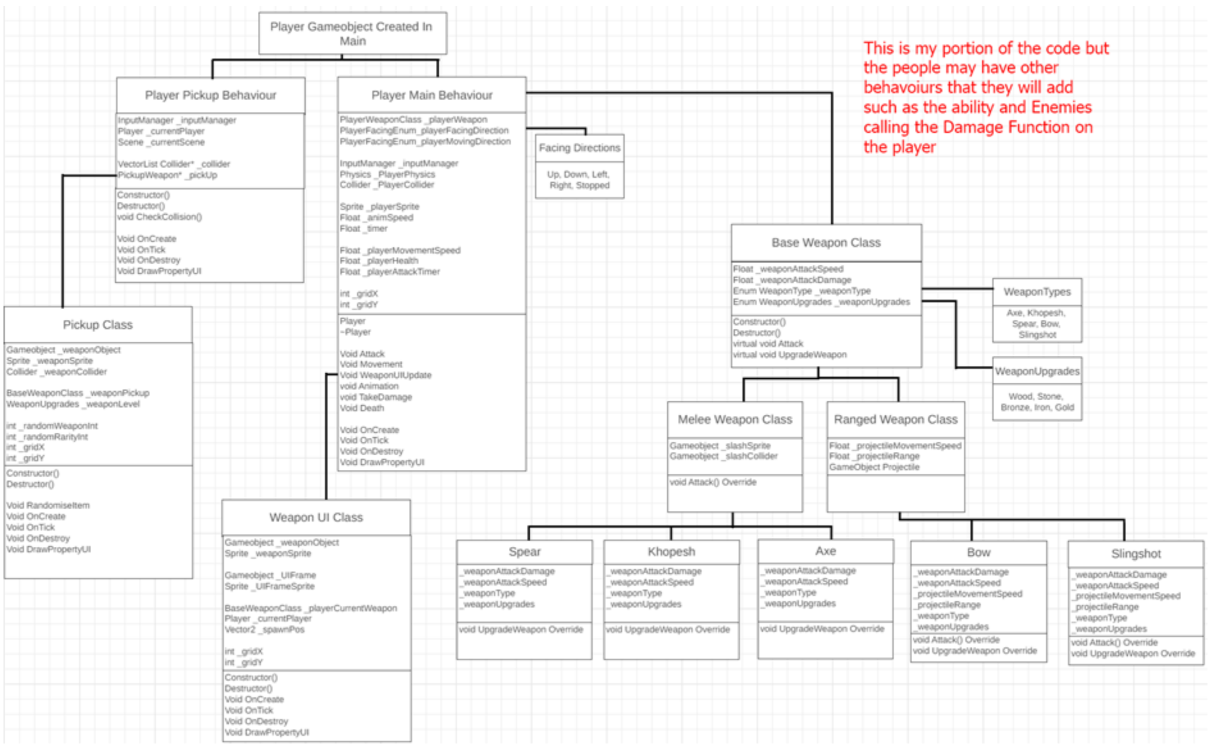
Person	Work to Do
Brandon B	IMGUI – Editor Work
Mathew	Engine and Scene System
Bailey	Make IMGUI for inputs
Connor	Audio
Brandon S	Physics – TDD
Harvey	Finding and Making Sprites
Kenny	Wireframes - And writing for the wireframe
Naomi	Wireframes - And writing for the wireframe
Me (Chris)	Finding all audio assets

Week 3 Friday

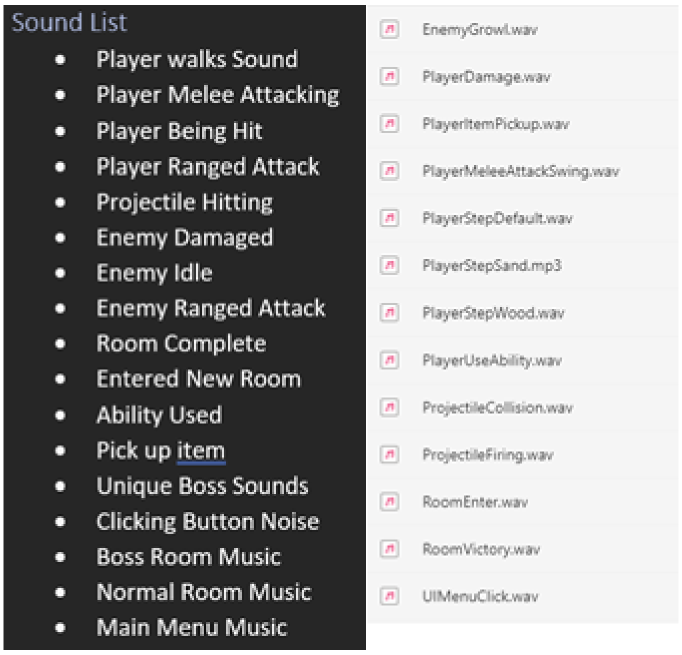
Person	Work to Do
Brandon B	IMGUI – Editor Work
Mathew	Git Merging
Bailey	IMGUI For Inputs
Connor	Audio
Brandon S	Merging Physics Engine
Harvey	Finding and Making Sprites
Kenny	Wireframes - And writing for the wireframe
Naomi	Wireframes - And writing for the wireframe
Me (Chris)	Finding all audio assets Continued

Posted February 5
Week 3 Forum Post

This week was the start of planning how we are going to do the game. I start by finishing the GDD finish all the gameplay details and outlining each weapon and classes values. After this I needed to plan how the game is going to run by making a UML diagram. I have been put in charge to the Player movement so Up ,Down , Left and right. Combat so each of the weapons and how the weapons inherits there values as well as the upgrade system of the weapons as well as the function of attacking the player and doing damage. And finally the drop system that will have a weapon drop from a enemy and the player picking up and equipping it. This will also need a UI for which weapon the player is holding.



As well as this I went and found the audio we are going to need to be in the game.


Sound List

- Player walks Sound
- Player Melee Attacking
- Player Being Hit
- Player Ranged Attack
- Projectile Hitting
- Enemy Damaged
- Enemy Idle
- Enemy Ranged Attack
- Room Complete
- Entered New Room
- Ability Used
- Pick up item
- Unique Boss Sounds
- Clicking Button Noise
- Boss Room Music
- Normal Room Music
- Main Menu Music

EnemyGrowl.wav
PlayerDamage.wav
PlayerItemPickup.wav
PlayerMeleeAttackSwing.wav
PlayerStepDefault.wav
PlayerStepSand.mp3
PlayerStepWood.wav
PlayerUseAbility.wav
ProjectileCollision.wav
ProjectileFiring.wav
RoomEnter.wav
RoomVictory.wav
UIMenuClick.wav

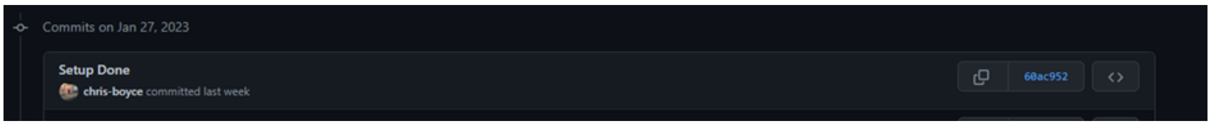
At the end of this week we should start to have a game engine so we can start the development of the game

Week 4 Forum Post

So, week 4 has started and the engine is yet to be finished. This is due to a few things taking longer than planned we planned for it to be done by Wednesday but it turned out even later with me not being able to make a branch till the Friday. This means from a dev side we have a week less meaning it will need to be picked up on week 5.

This week start with me finish the UML diagram as well as testing some inheritance and C++ in a separate project to remind myself how to get it to work. This was really basic stuff as I didn't want to invest a lot of time outside the engine.

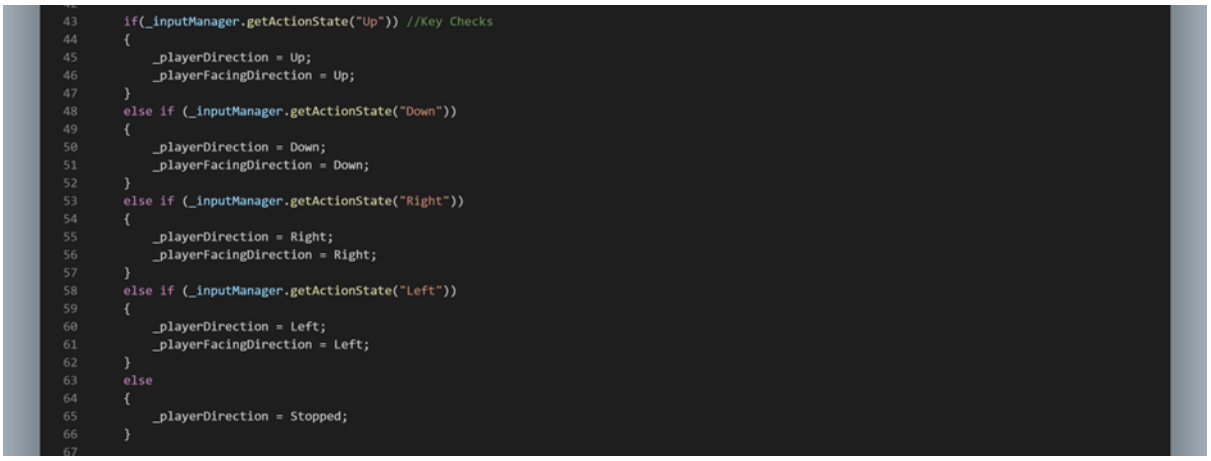
Then we waited and on Friday it was time to start. We made a Fork of the engine that we made branches off. I start with player movement. At this time the engine still didn't have keyboard inputs meaning I had to test adding velocity to the player by changing a Enum value and relaunching the project. This was then changed to add IMGUI buttons to make development easier.

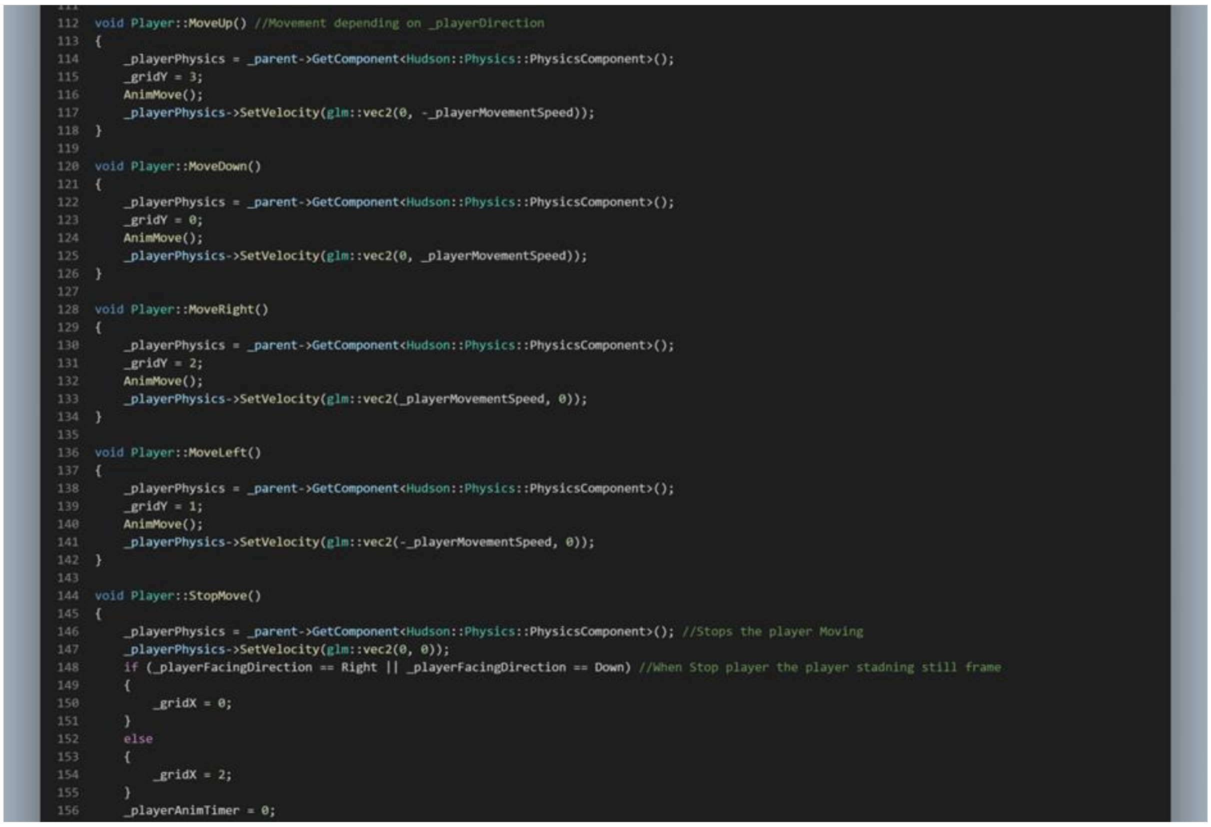

Commits on Jan 27, 2023

Setup Done
chris-boyce committed last week

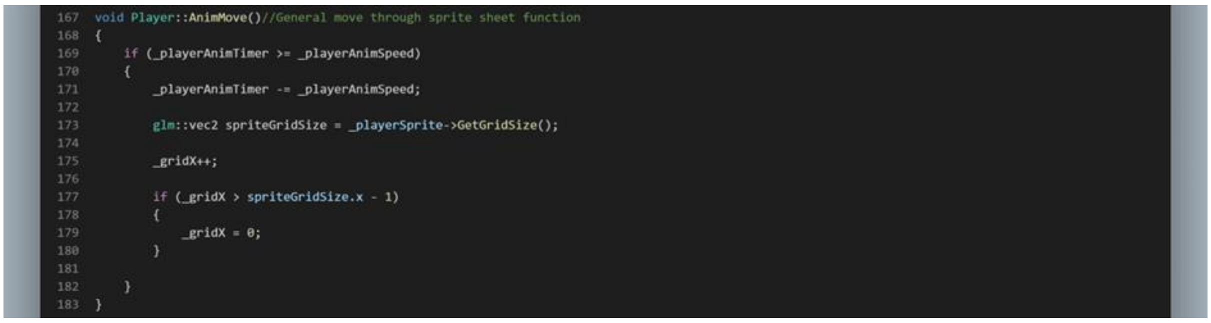
60ac952

Over the week 4 weekend I carried on with the project and got a lot of the movement system working and the start of the firing system. This system works off getting the keyboard inputs and changing the Enum of the moving direction and the facing direction, if there are no inputs the players facing direction stays the same but the players movement is set to stopped. The velocity is then set in the Ontick function and the player will the move. Finally the animation is called and it will run through the sprite sheet to make it look like it is walking


```
if(_inputManager.getActionState("Up")) //Key Checks
{
    _playerDirection = Up;
    _playerFacingDirection = Up;
}
else if (_inputManager.getActionState("Down"))
{
    _playerDirection = Down;
    _playerFacingDirection = Down;
}
else if (_inputManager.getActionState("Right"))
{
    _playerDirection = Right;
    _playerFacingDirection = Right;
}
else if (_inputManager.getActionState("Left"))
{
    _playerDirection = Left;
    _playerFacingDirection = Left;
}
else
{
    _playerDirection = Stopped;
}
```



```
void Player::MoveUp() //Movement depending on _playerDirection
{
    _playerPhysics = _parent->GetComponent<Hudson::Physics::PhysicsComponent>();
    _gridY = 3;
    AnimMove();
    _playerPhysics->SetVelocity(glm::vec2(0, -_playerMovementSpeed));
}

void Player::MoveDown()
{
    _playerPhysics = _parent->GetComponent<Hudson::Physics::PhysicsComponent>();
    _gridY = 0;
    AnimMove();
    _playerPhysics->SetVelocity(glm::vec2(0, _playerMovementSpeed));
}

void Player::MoveRight()
{
    _playerPhysics = _parent->GetComponent<Hudson::Physics::PhysicsComponent>();
    _gridY = 2;
    AnimMove();
    _playerPhysics->SetVelocity(glm::vec2(_playerMovementSpeed, 0));
}

void Player::MoveLeft()
{
    _playerPhysics = _parent->GetComponent<Hudson::Physics::PhysicsComponent>();
    _gridY = 1;
    AnimMove();
    _playerPhysics->SetVelocity(glm::vec2(-_playerMovementSpeed, 0));
}

void Player::StopMove()
{
    _playerPhysics = _parent->GetComponent<Hudson::Physics::PhysicsComponent>(); //Stops the player Moving
    _playerPhysics->SetVelocity(glm::vec2(0, 0));
    if (_playerFacingDirection == Right || _playerFacingDirection == Down) //When Stop player the player stadning still frame
    {
        _gridX = 0;
    }
    else
    {
        _gridX = 2;
    }
    _playerAnimTimer = 0;
```



```
void Player::AnimMove()//General move through sprite sheet function
{
    if (_playerAnimTimer >= _playerAnimSpeed)
    {
        _playerAnimTimer -= _playerAnimSpeed;

        glm::vec2 spriteGridSize = _playerSprite->GetGridSize();

        _gridX++;

        if (_gridX > spriteGridSize.x - 1)
        {
            _gridX = 0;
        }

    }
}
```

Week 5 Forum Post

LET’S GOOOOOOOO. Time to start working on the game quickly.

My TODO List :

Projectile System / Firing

Melee Combat / Attack

Inheritance off all the weapons / Setting Up the Base Class, Melee Class, and Ranged Class

Stats for the player / set up a bunch of Variables.

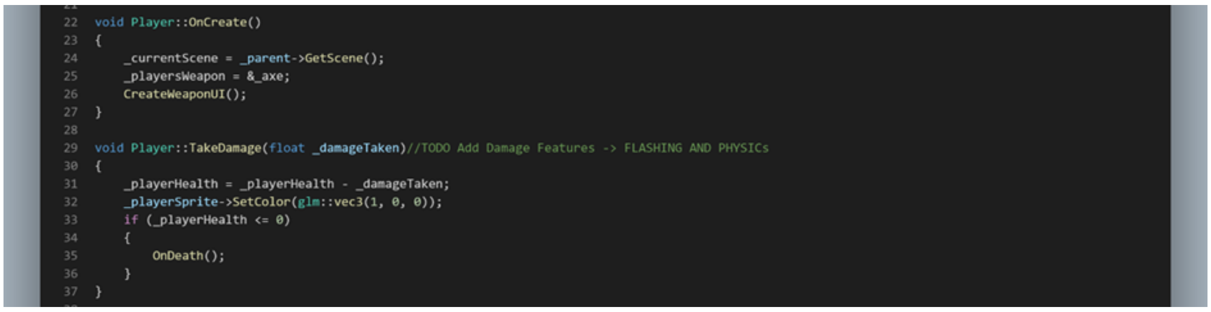
Weapon Levelling / Set up the base stats and the Upgrade Functions to changes the stats.

Pick Ups / Setting up some class that interacts and saves the Stats to the player

Pick up UI / Some sort of UI class that displays the weapons.

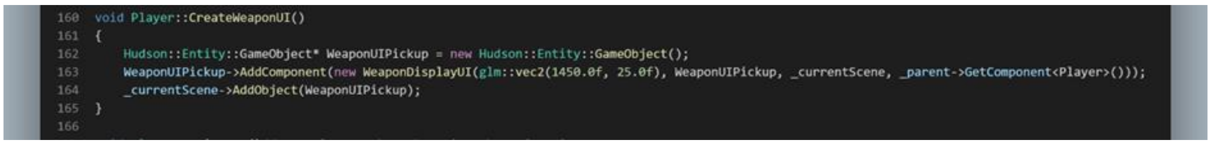
Player Class

Added the take damage and the variables to hold the “Base Weapon Class”. Which will be shown later on how it works. But it literally just stores a class of the weapon on the player.



```
void Player::OnCreate()
{
    _currentScene = _parent->GetScene();
    _playersWeapon = &_axe;
    CreateWeaponUI();
}

void Player::TakeDamage(float _damageTaken)//TODO Add Damage Features -> FLASHING AND PHYSICs
{
    _playerHealth = _playerHealth - _damageTaken;
    _playerSprite->SetColor(glm::vec3(1, 0, 0));
    if (_playerHealth <= 0)
    {
        OnDeath();
    }
}
```

Added to the player a Create UI because it is going to create the UI from the player and show which weapon that the player is going to hold.

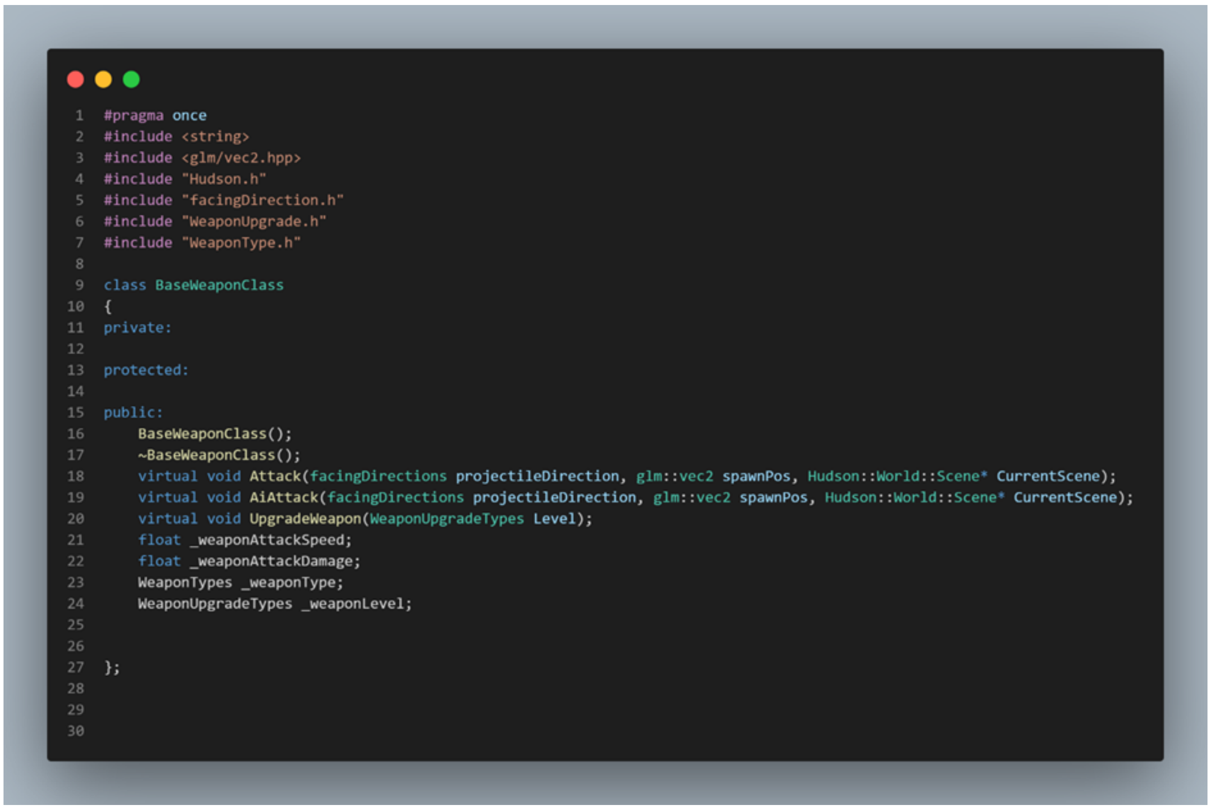


```
void Player::CreateWeaponUI()
{
    Hudson::Entity::GameObject* WeaponUIPickup = new Hudson::Entity::GameObject();
    WeaponUIPickup->AddComponent(new WeaponDisplayUI(glm::vec2(1450.0f, 25.0f), WeaponUIPickup, _currentScene, _parent->GetComponent<Player>()));
    _currentScene->AddObject(WeaponUIPickup);
}
```

Apart from that lots of little bits got changed in the player such as adding an attack function to call the weapon attack function and adding keyboard inputs into the game.

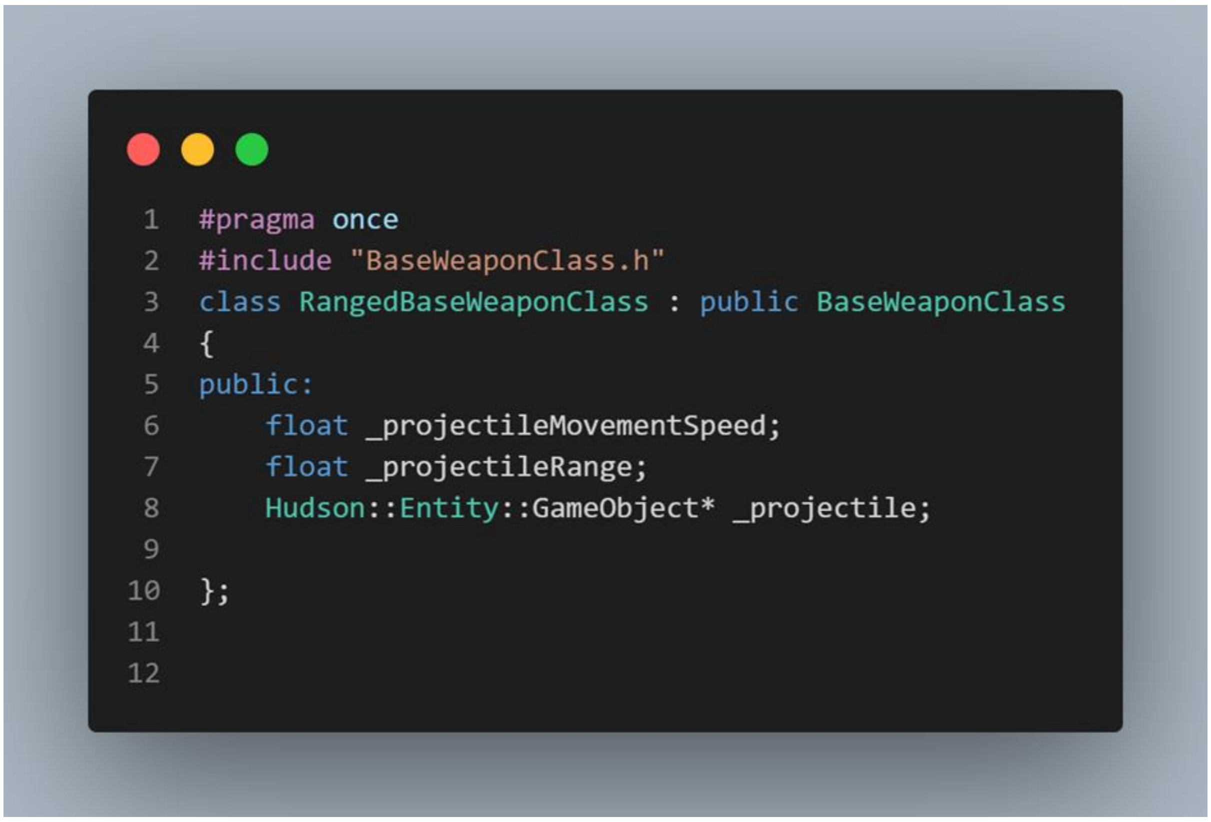
Base Weapon Class

The base weapon class had been set up with all the functions and variables that are going to be attached to each of the weapons. I worked on splitting these weapons into ranged and melee base classes.



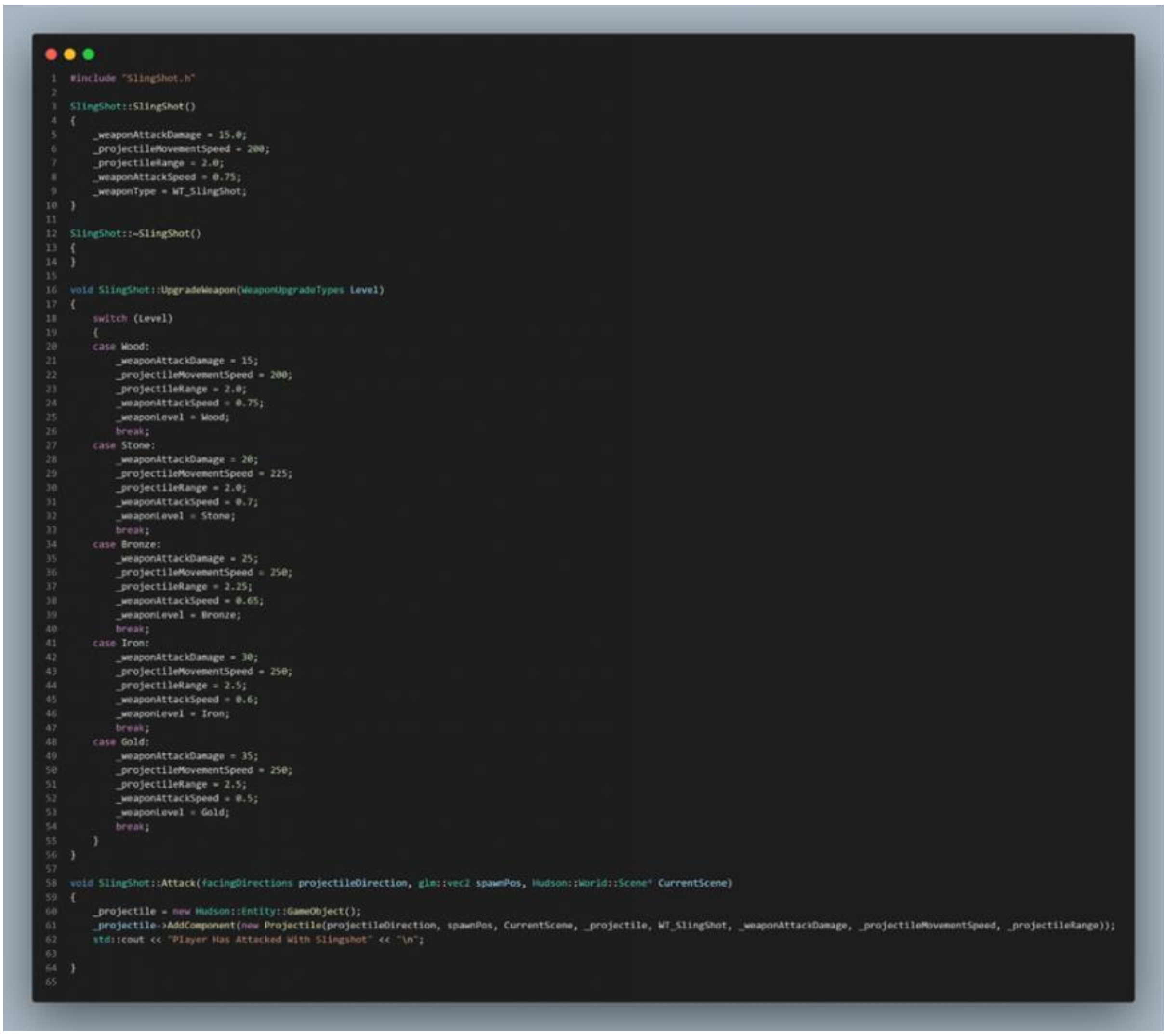
```cpp
#pragma once
#include <string>
#include <glm/vec2.hpp>
#include "Hudson.h"
#include "facingDirection.h"
#include "WeaponUpgrade.h"
#include "WeaponType.h"

class BaseWeaponClass
{
private:

protected:

public:
    BaseWeaponClass();
    ~BaseWeaponClass();
    virtual void Attack(facingDirections projectileDirection, glm::vec2 spawnPos, Hudson::World::Scene* CurrentScene);
    virtual void AiAttack(facingDirections projectileDirection, glm::vec2 spawnPos, Hudson::World::Scene* CurrentScene);
    virtual void UpgradeWeapon(WeaponUpgradeTypes Level);
    float _weaponAttackSpeed;
    float _weaponAttackDamage;
    WeaponTypes _weaponType;
    WeaponUpgradeTypes _weaponLevel;


};



```

The base ranged class adds all the variables for the ranged variables so it can fire them once called



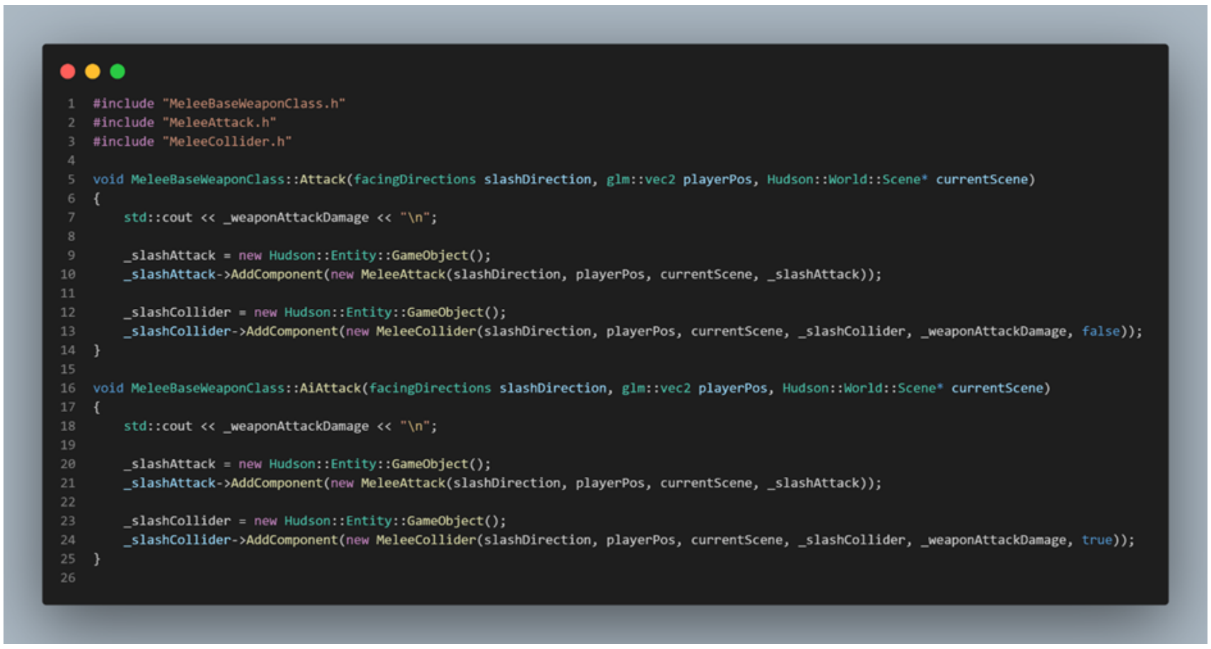
```cpp
#pragma once
#include "BaseWeaponClass.h"
class RangedBaseWeaponClass : public BaseWeaponClass
{
public:
    float _projectileMovementSpeed;
    float _projectileRange;
    Hudson::Entity::GameObject* _projectile;

};


```

This is an example of how the “Slingshot” is set up. Each of the variables are set and the attack function makes a projectile and passes all the values it needs into the create the projectile. The Attack and upgrades are overrides due to the slight difference in each weapon. This CPP is similar throughout each weapon class.



```cpp
#include "SlingShot.h"

SlingShot::SlingShot()
{
    _weaponAttackDamage = 15.0;
    _projectileMovementSpeed = 200;
    _projectileRange = 2.0;
    _weaponAttackSpeed = 0.75;
    _weaponType = WT_SlingShot;
}

SlingShot::~SlingShot()
{
}

void SlingShot::UpgradeWeapon(WeaponUpgradeTypes Level)
{
    switch (Level)
    {
    case Wood:
        _weaponAttackDamage = 15;
        _projectileMovementSpeed = 200;
        _projectileRange = 2.0;
        _weaponAttackSpeed = 0.75;
        _weaponLevel = Wood;
        break;
    case Stone:
        _weaponAttackDamage = 20;
        _projectileMovementSpeed = 225;
        _projectileRange = 2.0;
        _weaponAttackSpeed = 0.7;
        _weaponLevel = Stone;
        break;
    case Bronze:
        _weaponAttackDamage = 25;
        _projectileMovementSpeed = 250;
        _projectileRange = 2.25;
        _weaponAttackSpeed = 0.65;
        _weaponLevel = Bronze;
        break;
    case Iron:
        _weaponAttackDamage = 30;
        _projectileMovementSpeed = 250;
        _projectileRange = 2.5;
        _weaponAttackSpeed = 0.6;
        _weaponLevel = Iron;
        break;
    case Gold:
        _weaponAttackDamage = 35;
        _projectileMovementSpeed = 250;
        _projectileRange = 2.5;
        _weaponAttackSpeed = 0.5;
        _weaponLevel = Gold;
        break;
    }
}

void SlingShot::Attack(facingDirections projectileDirection, glm::vec2 spawnPos, Hudson::World::Scene* CurrentScene)
{
    _projectile = new Hudson::Entity::GameObject();
    _projectile->AddComponent(new Projectile(projectileDirection, spawnPos, CurrentScene, _projectile, WT_SlingShot, _weaponAttackDamage, _projectileMovementSpeed, _projectileRange));
    std::cout << "Player Has Attacked With Slingshot" << "\n";

}
```

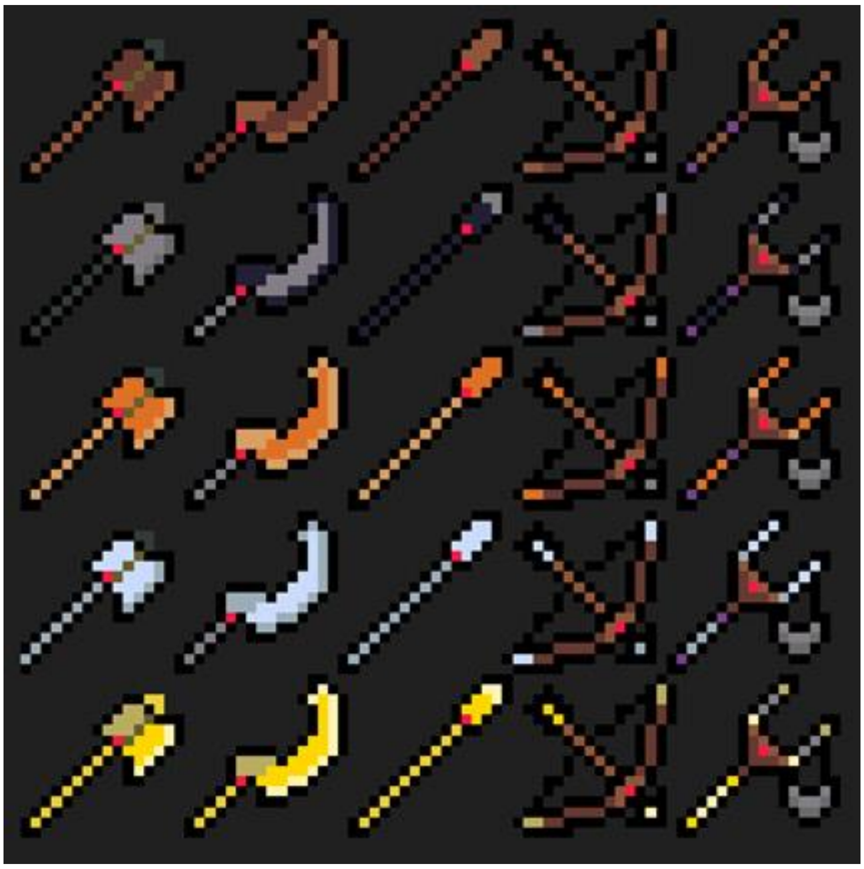
The melee base class is a bit different. It work by making 2 game objects one that makes the sprite and one that makes the collider the collider will do the damage function and the sprite just runs through the animations. This is because a small issue in the engine on deleting the collider but not the sprite so making 2 very similar objects was the best way to do this.

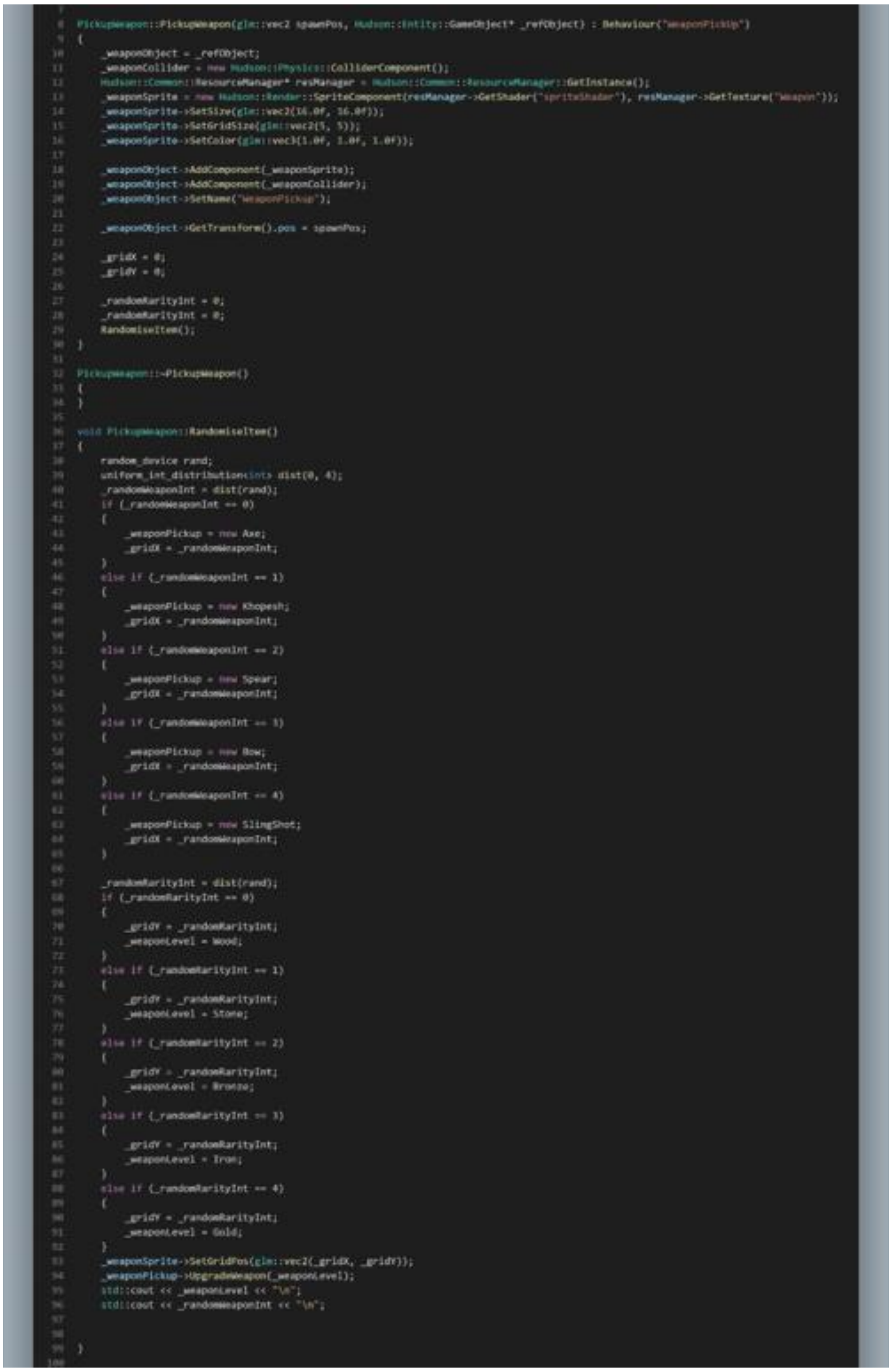


```cpp
#include "MeleeBaseWeaponClass.h"
#include "MeleeAttack.h"
#include "MeleeCollider.h"

void MeleeBaseWeaponClass::Attack(facingDirections slashDirection, glm::vec2 playerPos, Hudson::World::Scene* currentScene)
{
    std::cout << _weaponAttackDamage << "\n";

    _slashAttack = new Hudson::Entity::GameObject();
    _slashAttack->AddComponent(new MeleeAttack(slashDirection, playerPos, currentScene, _slashAttack));

    _slashCollider = new Hudson::Entity::GameObject();
    _slashCollider->AddComponent(new MeleeCollider(slashDirection, playerPos, currentScene, _slashCollider, _weaponAttackDamage, false));
}

void MeleeBaseWeaponClass::AiAttack(facingDirections slashDirection, glm::vec2 playerPos, Hudson::World::Scene* currentScene)
{
    std::cout << _weaponAttackDamage << "\n";

    _slashAttack = new Hudson::Entity::GameObject();
    _slashAttack->AddComponent(new MeleeAttack(slashDirection, playerPos, currentScene, _slashAttack));

    _slashCollider = new Hudson::Entity::GameObject();
    _slashCollider->AddComponent(new MeleeCollider(slashDirection, playerPos, currentScene, _slashCollider, _weaponAttackDamage, true));
}

```

<u>Weapon Pickup System</u>

This weapon class works by getting a “BaseWeaponClass” set up and giving It a random int that selects the weapon type and weapon level. At the end of this is will display the correct sprite to the player that I made by taking Baileys weapons and recolouring them. This sets up all the values for the next script.

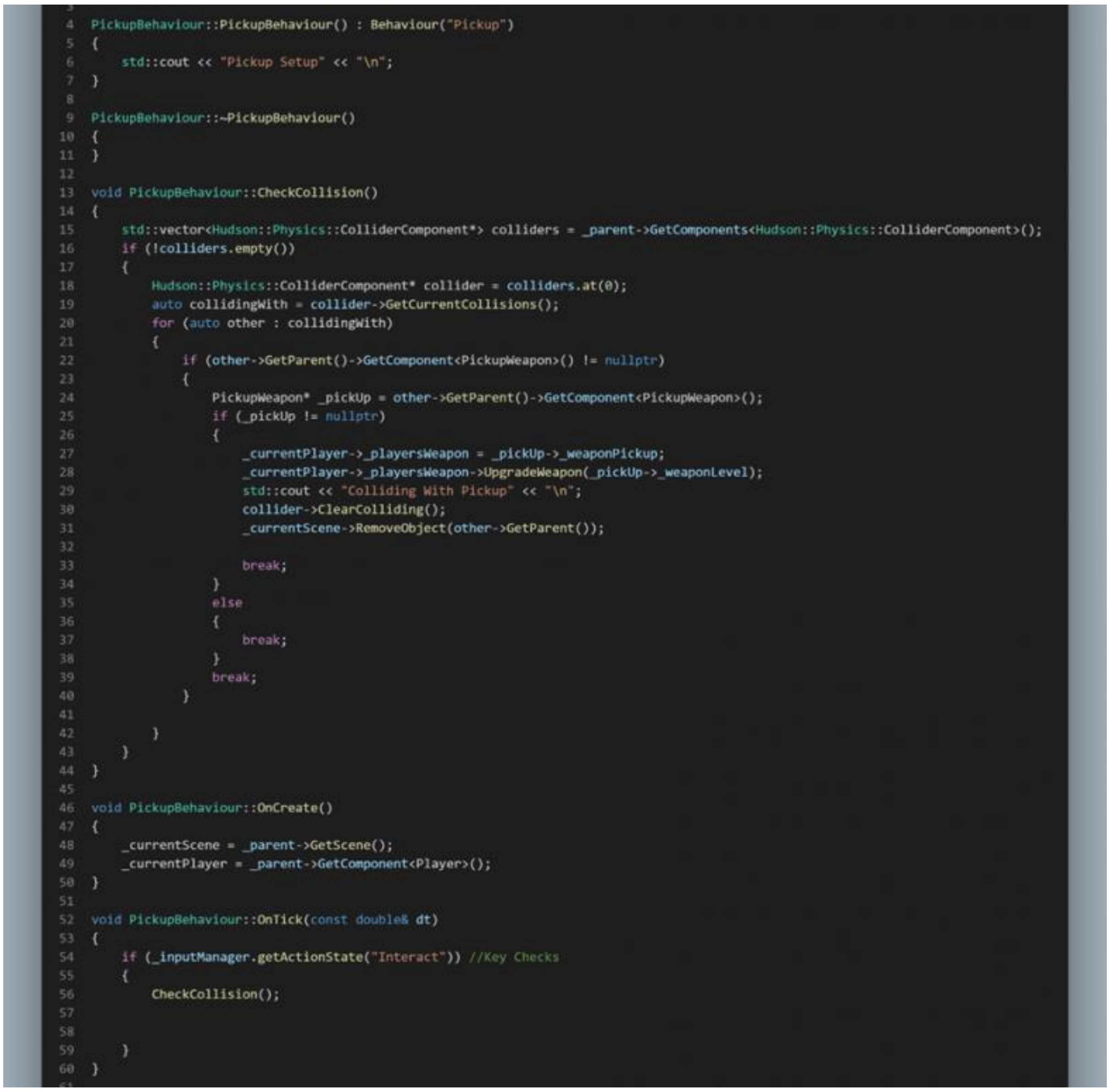




```
PickupWeapon::PickupWeapon(glm::vec2 spawnPos, Hudson::Entity::GameObject* _refObject) : Behaviour("weaponPickup")
{
    _weaponObject = _refObject;
    _weaponCollider = new Hudson::Physics::ColliderComponent();
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    _weaponSprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("Weapon"));
    _weaponSprite->SetSize(glm::vec2(16.0f, 16.0f));
    _weaponSprite->SetGridSize(glm::vec2(5, 5));
    _weaponSprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));

    _weaponObject->AddComponent(_weaponSprite);
    _weaponObject->AddComponent(_weaponCollider);
    _weaponObject->SetName("WeaponPickup");

    _weaponObject->GetTransform().pos = spawnPos;

    _gridX = 0;
    _gridY = 0;

    _randomRarityInt = 0;
    _randomRarityInt = 0;
    RandomiseItem();
}

PickupWeapon::~PickupWeapon()
{
}

void PickupWeapon::RandomiseItem()
{
    random_device rand;
    uniform_int_distribution<int> dist(0, 4);
    _randomWeaponInt = dist(rand);
    if (_randomWeaponInt == 0)
    {
        _weaponPickup = new Axe;
        _gridX = _randomWeaponInt;
    }
    else if (_randomWeaponInt == 1)
    {
        _weaponPickup = new Khopesh;
        _gridX = _randomWeaponInt;
    }
    else if (_randomWeaponInt == 2)
    {
        _weaponPickup = new Spear;
        _gridX = _randomWeaponInt;
    }
    else if (_randomWeaponInt == 3)
    {
        _weaponPickup = new Bow;
        _gridX = _randomWeaponInt;
    }
    else if (_randomWeaponInt == 4)
    {
        _weaponPickup = new SlingShot;
        _gridX = _randomWeaponInt;
    }

    _randomRarityInt = dist(rand);
    if (_randomRarityInt == 0)
    {
        _gridY = _randomRarityInt;
        _weaponLevel = Wood;
    }
    else if (_randomRarityInt == 1)
    {
        _gridY = _randomRarityInt;
        _weaponLevel = Stone;
    }
    else if (_randomRarityInt == 2)
    {
        _gridY = _randomRarityInt;
        _weaponLevel = Bronze;
    }
    else if (_randomRarityInt == 3)
    {
        _gridY = _randomRarityInt;
        _weaponLevel = Iron;
    }
    else if (_randomRarityInt == 4)
    {
        _gridY = _randomRarityInt;
        _weaponLevel = Gold;
    }
    _weaponSprite->SetGridPos(glm::vec2(_gridX, _gridY));
    _weaponPickup->UpgradeWeapon(_weaponLevel);
    std::cout << _weaponLevel << "\n";
    std::cout << _randomWeaponInt << "\n";


}
```

Weapon Pickup Behaviour

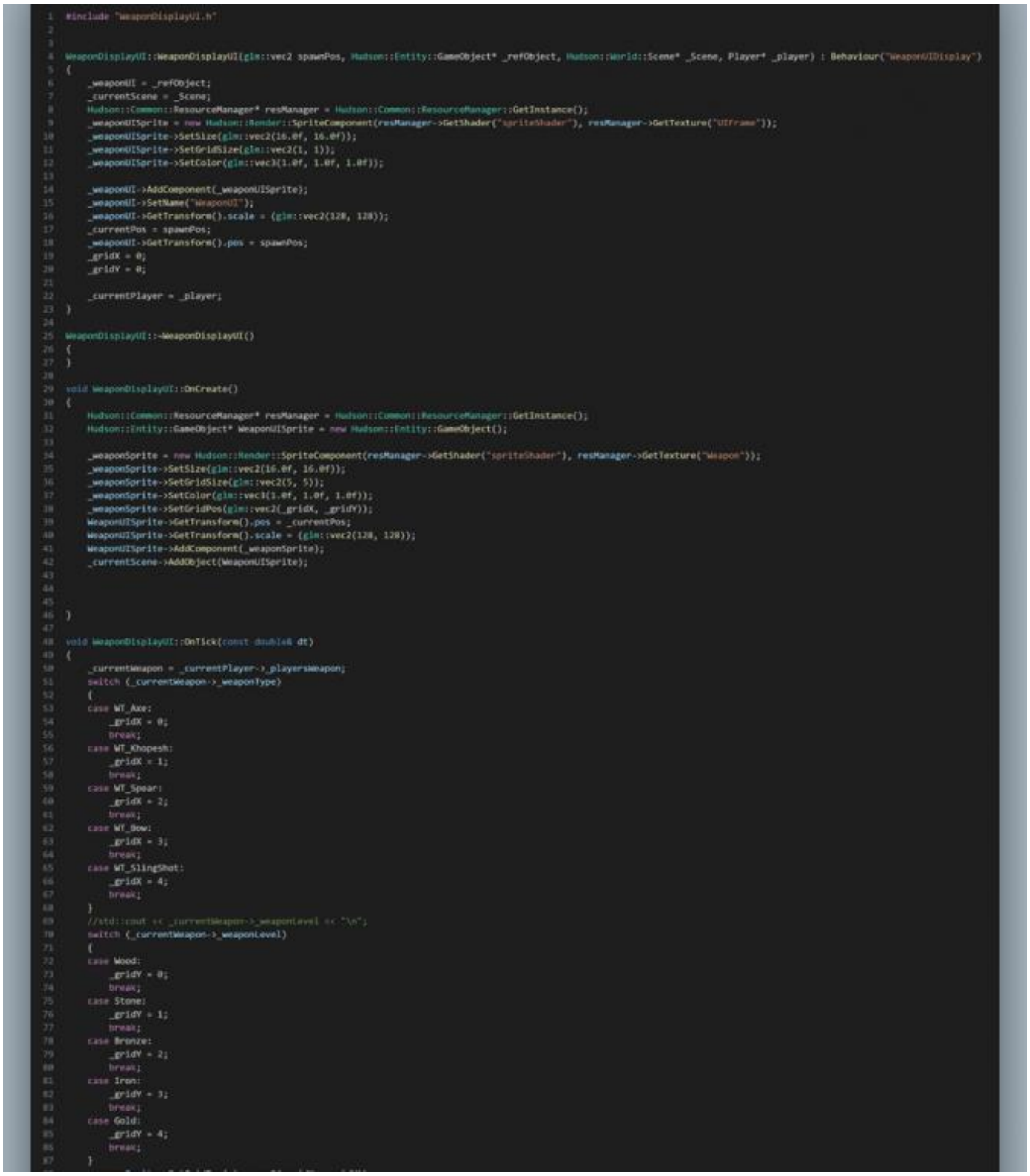
This script is attached to the player object. When it collides with a pickup and the player presses “F” it will then get the objects values and apply them to the player current weapon variable and change the weapon that the player is holding. It then deletes the object.



```cpp
PickupBehaviour::PickupBehaviour() : Behaviour("Pickup")
{
    std::cout << "Pickup Setup" << "\n";
}

PickupBehaviour::~PickupBehaviour()
{
}

void PickupBehaviour::CheckCollision()
{
    std::vector<Hudson::Physics::ColliderComponent*> colliders = _parent->GetComponents<Hudson::Physics::ColliderComponent>();
    if (!colliders.empty())
    {
        Hudson::Physics::ColliderComponent* collider = colliders.at(0);
        auto collidingWith = collider->GetCurrentCollisions();
        for (auto other : collidingWith)
        {
            if (other->GetParent()->GetComponent<PickupWeapon>() != nullptr)
            {
                PickupWeapon* _pickUp = other->GetParent()->GetComponent<PickupWeapon>();
                if (_pickUp != nullptr)
                {
                    _currentPlayer->_playersWeapon = _pickUp->_weaponPickup;
                    _currentPlayer->_playersWeapon->UpgradeWeapon(_pickUp->_weaponLevel);
                    std::cout << "Colliding With Pickup" << "\n";
                    collider->ClearColliding();
                    _currentScene->RemoveObject(other->GetParent());

                    break;
                }
                else
                {
                    break;
                }
                break;
            }

        }
    }
}

void PickupBehaviour::OnCreate()
{
    _currentScene = _parent->GetScene();
    _currentPlayer = _parent->GetComponent<Player>();
}

void PickupBehaviour::OnTick(const double& dt)
{
    if (_inputManager.getActionState("Interact")) //Key Checks
    {
        CheckCollision();


    }
}
```

Weapon Display UI

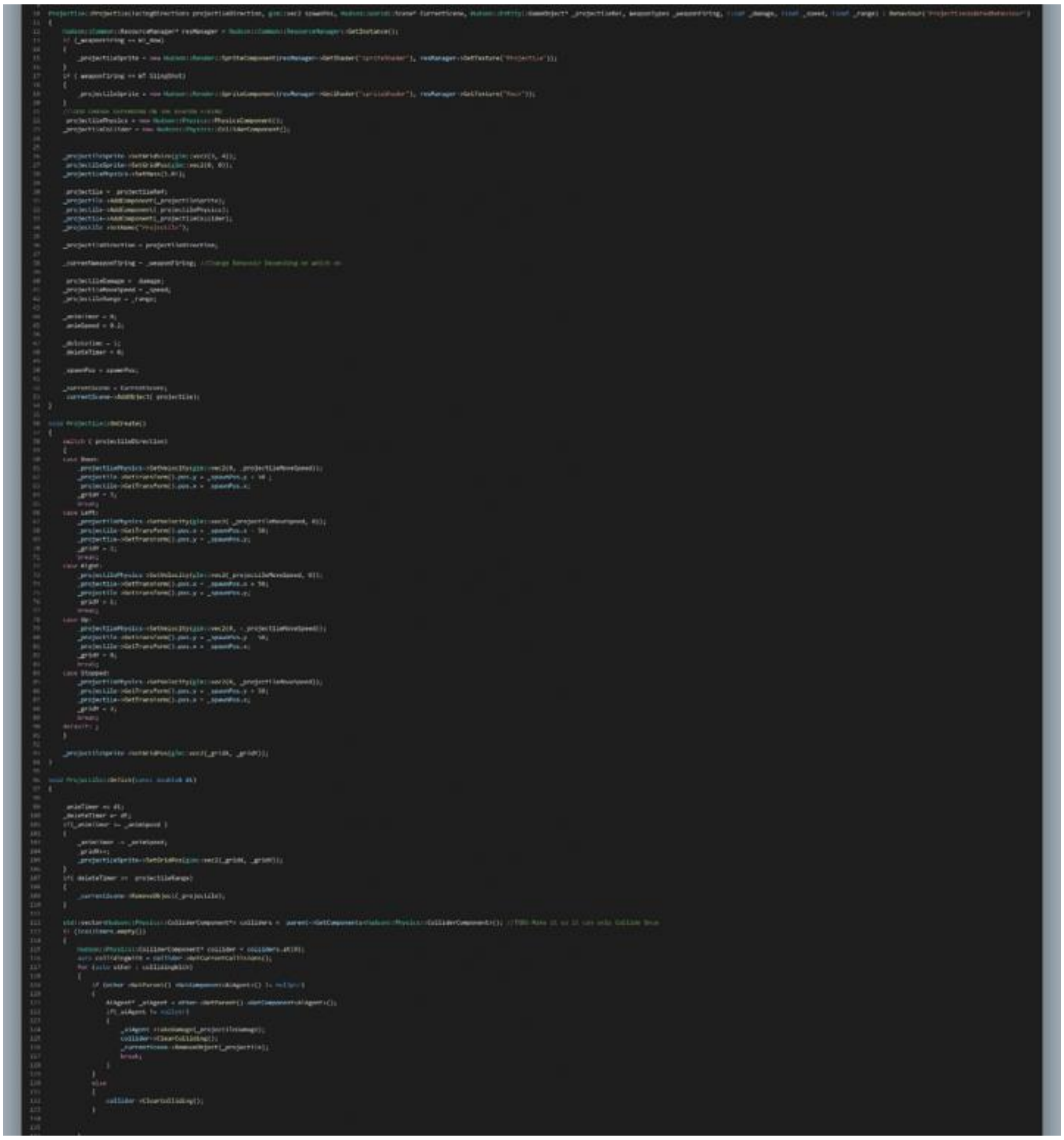
The weapon UI that is made in the player gets the value of the player weapon and sets the UI sprite sheet to the correct weapon the player is holding. This also makes a UI Frame that goes round the object that makes it look like a UI element.



```cpp
#include "WeaponDisplayUI.h"


WeaponDisplayUI::WeaponDisplayUI(glm::vec2 spawnPos, Hudson::Entity::GameObject* _refObject, Hudson::World::Scene* _Scene, Player* _player) : Behaviour("WeaponUIDisplay")
{
    _weaponUI = _refObject;
    _currentScene = _Scene;
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    _weaponUISprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("UIFrame"));
    _weaponUISprite->SetSize(glm::vec2(16.0f, 16.0f));
    _weaponUISprite->SetGridSize(glm::vec2(1, 1));
    _weaponUISprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));

    _weaponUI->AddComponent(_weaponUISprite);
    _weaponUI->SetName("WeaponUI");
    _weaponUI->GetTransform().scale = (glm::vec2(128, 128));
    _currentPos = spawnPos;
    _weaponUI->GetTransform().pos = spawnPos;
    _gridX = 0;
    _gridY = 0;

    _currentPlayer = _player;
}

WeaponDisplayUI::~WeaponDisplayUI()
{
}

void WeaponDisplayUI::OnCreate()
{
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    Hudson::Entity::GameObject* WeaponUISprite = new Hudson::Entity::GameObject();

    _weaponSprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("Weapon"));
    _weaponSprite->SetSize(glm::vec2(16.0f, 16.0f));
    _weaponSprite->SetGridSize(glm::vec2(5, 5));
    _weaponSprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));
    _weaponSprite->SetGridPos(glm::vec2(_gridX, _gridY));
    WeaponUISprite->GetTransform().pos = _currentPos;
    WeaponUISprite->GetTransform().scale = (glm::vec2(128, 128));
    WeaponUISprite->AddComponent(_weaponSprite);
    _currentScene->AddObject(WeaponUISprite);



}

void WeaponDisplayUI::OnTick(const double& dt)
{
    _currentWeapon = _currentPlayer->_playersWeapon;
    switch (_currentWeapon->_weaponType)
    {
    case WT_Axe:
        _gridX = 0;
        break;
    case WT_Khopesh:
        _gridX = 1;
        break;
    case WT_Spear:
        _gridX = 2;
        break;
    case WT_Bow:
        _gridX = 3;
        break;
    case WT_SlingShot:
        _gridX = 4;
        break;
    }
    //std::cout << _currentWeapon->_weaponLevel << "\n";
    switch (_currentWeapon->_weaponLevel)
    {
    case Wood:
        _gridY = 0;
        break;
    case Stone:
        _gridY = 1;
        break;
    case Bronze:
        _gridY = 2;
        break;
    case Iron:
        _gridY = 3;
        break;
    case Gold:
        _gridY = 4;
        break;
    }
```

Combat

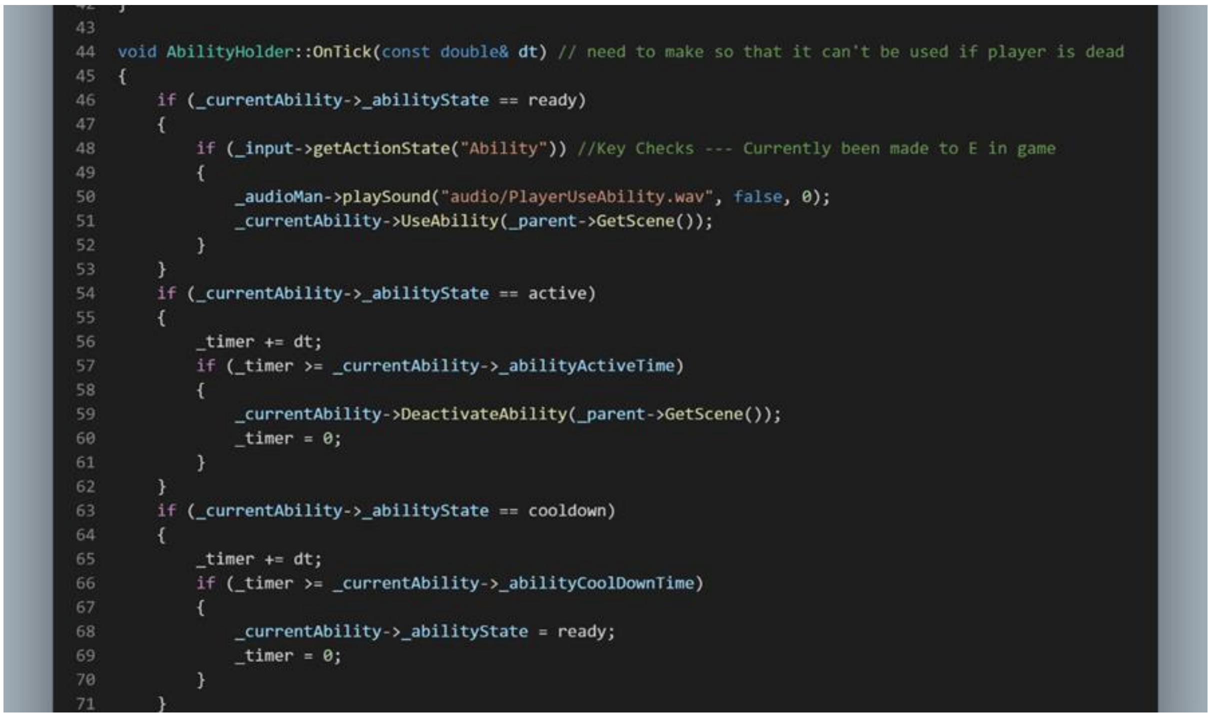
This script is called by the projectile weapons and makes a new game object the projectile and sets all the variables that are passed in such as the direction the speed the damage and range and then send it on its way. It also contains the collision code for when it hits an enemy and delivers the damage to them. The Melee works in the same way by getting everything passed into the it then delivering the damage.



```
[illegible]
```

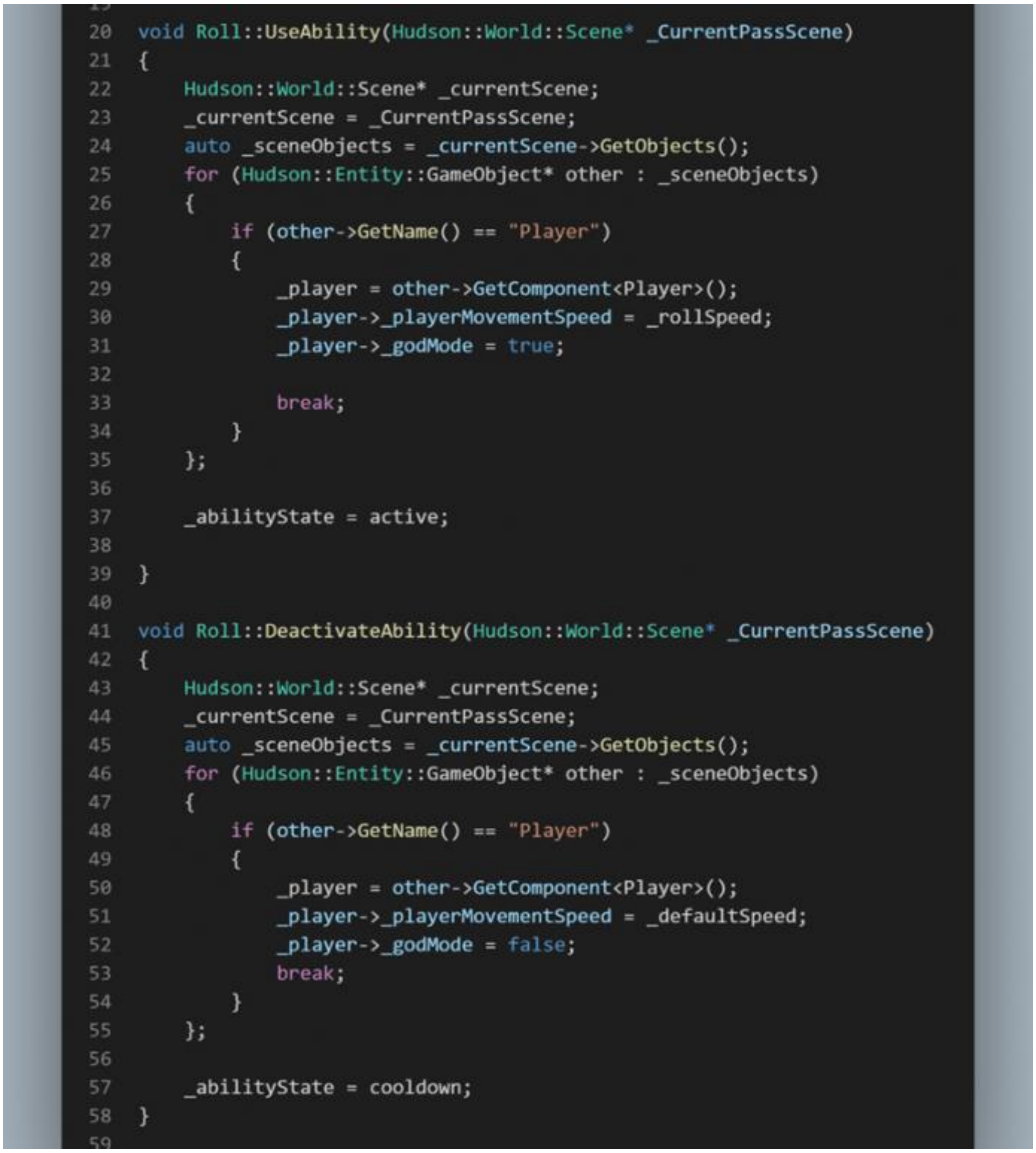
Week 6 Forum Post

I started this week by getting Kenny sorted to make abilities this was by editing the Base Behaviour Class up and setting the “Format” for abilities to use by using the Enum that he had set up. After this I started by making a roll and stun class and testing this with the player. This on being activated and then activating the ability. The timer then starts and after the timer stops the ability is deactivated the ability. Then a cooldown starts. Which is now represented with a UI highlight.



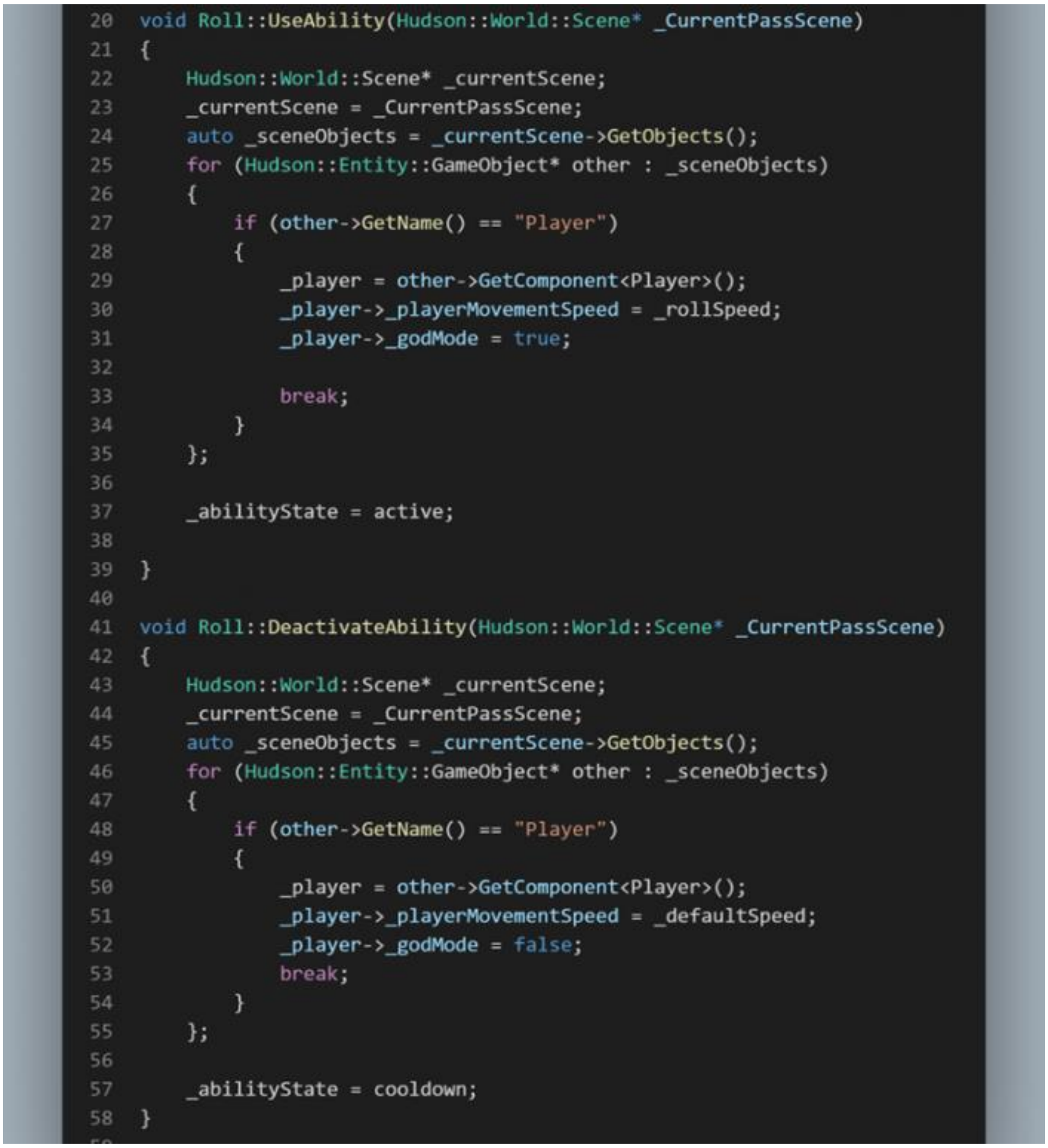
```
void AbilityHolder::OnTick(const double& dt) // need to make so that it can't be used if player is dead
{
    if (_currentAbility->_abilityState == ready)
    {
        if (_input->getActionState("Ability")) //Key Checks --- Currently been made to E in game
        {
            _audioMan->playSound("audio/PlayerUseAbility.wav", false, 0);
            _currentAbility->UseAbility(_parent->GetScene());
        }
    }
    if (_currentAbility->_abilityState == active)
    {
        _timer += dt;
        if (_timer >= _currentAbility->_abilityActiveTime)
        {
            _currentAbility->DeactivateAbility(_parent->GetScene());
            _timer = 0;
        }
    }
    if (_currentAbility->_abilityState == cooldown)
    {
        _timer += dt;
        if (_timer >= _currentAbility->_abilityCoolDownTime)
        {
            _currentAbility->_abilityState = ready;
            _timer = 0;
        }
    }
```

Roll



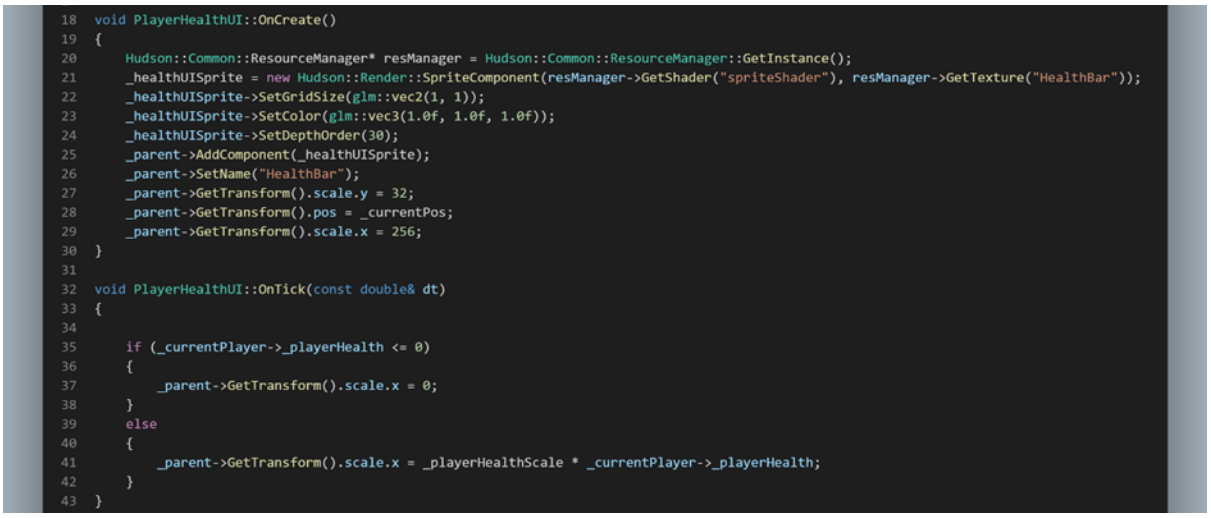
```cpp
void Roll::UseAbility(Hudson::World::Scene* _CurrentPassScene)
{
    Hudson::World::Scene* _currentScene;
    _currentScene = _CurrentPassScene;
    auto _sceneObjects = _currentScene->GetObjects();
    for (Hudson::Entity::GameObject* other : _sceneObjects)
    {
        if (other->GetName() == "Player")
        {
            _player = other->GetComponent<Player>();
            _player->_playerMovementSpeed = _rollSpeed;
            _player->_godMode = true;

            break;
        }
    };

    _abilityState = active;

}

void Roll::DeactivateAbility(Hudson::World::Scene* _CurrentPassScene)
{
    Hudson::World::Scene* _currentScene;
    _currentScene = _CurrentPassScene;
    auto _sceneObjects = _currentScene->GetObjects();
    for (Hudson::Entity::GameObject* other : _sceneObjects)
    {
        if (other->GetName() == "Player")
        {
            _player = other->GetComponent<Player>();
            _player->_playerMovementSpeed = _defaultSpeed;
            _player->_godMode = false;
            break;
        }
    };

    _abilityState = cooldown;
}
```

Stun



```
void Roll::UseAbility(Hudson::World::Scene* _CurrentPassScene)
{
    Hudson::World::Scene* _currentScene;
    _currentScene = _CurrentPassScene;
    auto _sceneObjects = _currentScene->GetObjects();
    for (Hudson::Entity::GameObject* other : _sceneObjects)
    {
        if (other->GetName() == "Player")
        {
            _player = other->GetComponent<Player>();
            _player->_playerMovementSpeed = _rollSpeed;
            _player->_godMode = true;

            break;
        }
    };

    _abilityState = active;

}

void Roll::DeactivateAbility(Hudson::World::Scene* _CurrentPassScene)
{
    Hudson::World::Scene* _currentScene;
    _currentScene = _CurrentPassScene;
    auto _sceneObjects = _currentScene->GetObjects();
    for (Hudson::Entity::GameObject* other : _sceneObjects)
    {
        if (other->GetName() == "Player")
        {
            _player = other->GetComponent<Player>();
            _player->_playerMovementSpeed = _defaultSpeed;
            _player->_godMode = false;
            break;
        }
    };

    _abilityState = cooldown;
}
```

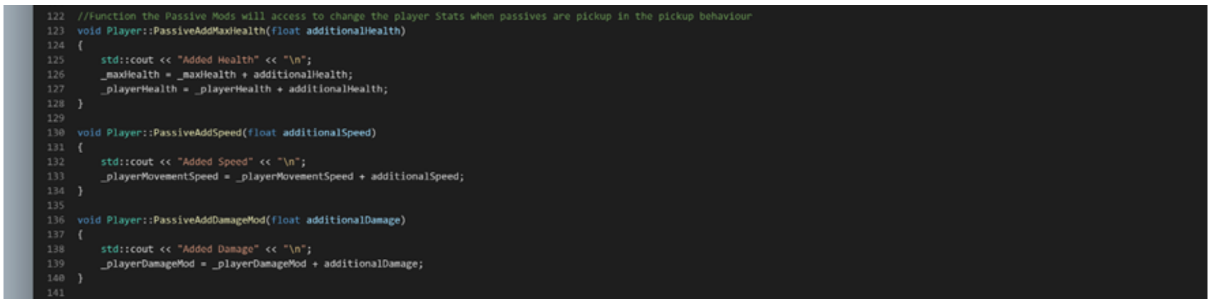
I then after this added the health bar to the game. Which is a texture that get scaled to represent the players health in the game. This was very simple and effective, but sometimes that’s the best way. It then updates it every frame.



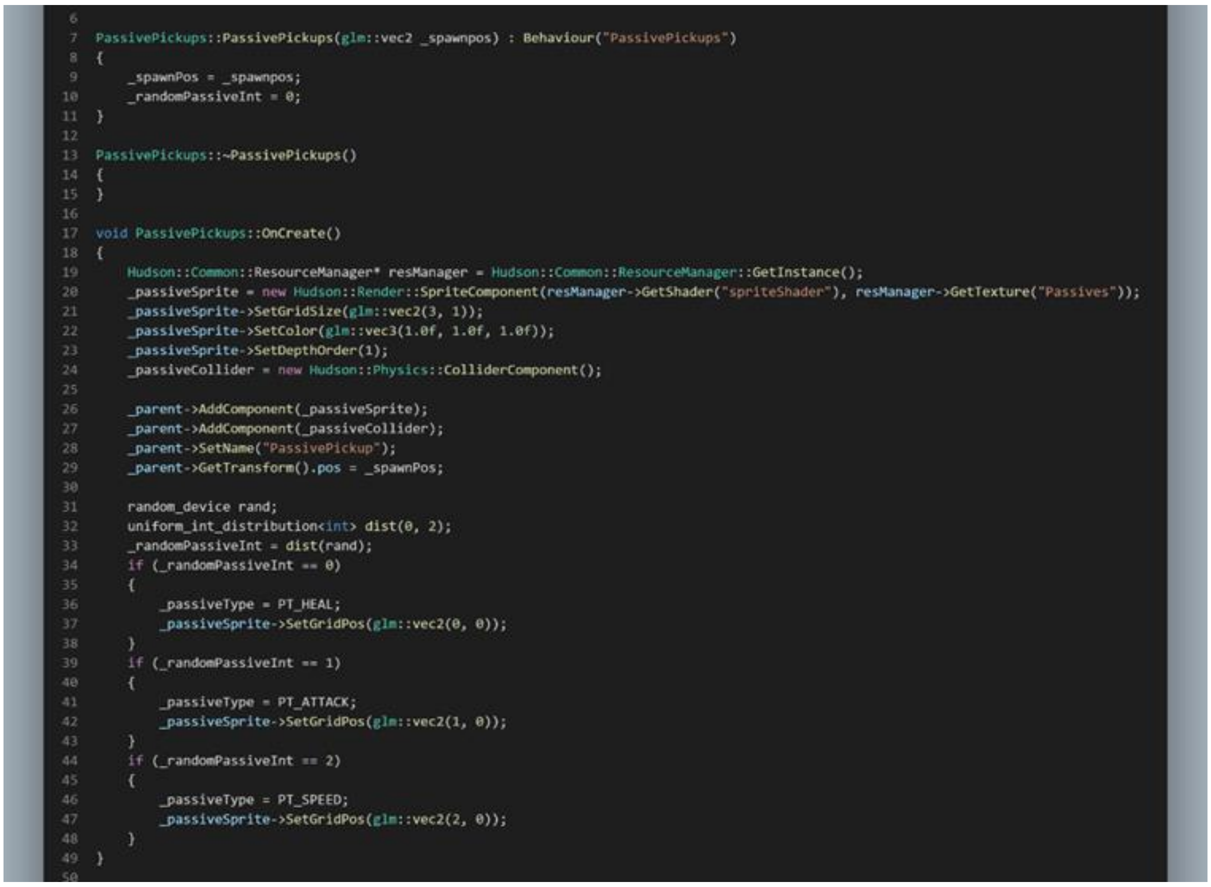
```cpp
void PlayerHealthUI::OnCreate()
{
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    _healthUISprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("HealthBar"));
    _healthUISprite->SetGridSize(glm::vec2(1, 1));
    _healthUISprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));
    _healthUISprite->SetDepthOrder(30);
    _parent->AddComponent(_healthUISprite);
    _parent->SetName("HealthBar");
    _parent->GetTransform().scale.y = 32;
    _parent->GetTransform().pos = _currentPos;
    _parent->GetTransform().scale.x = 256;
}

void PlayerHealthUI::OnTick(const double& dt)
{

    if (_currentPlayer->_playerHealth <= 0)
    {
        _parent->GetTransform().scale.x = 0;
    }
    else
    {
        _parent->GetTransform().scale.x = _playerHealthScale * _currentPlayer->_playerHealth;
    }
}
```

Adding Passives – New Pickups

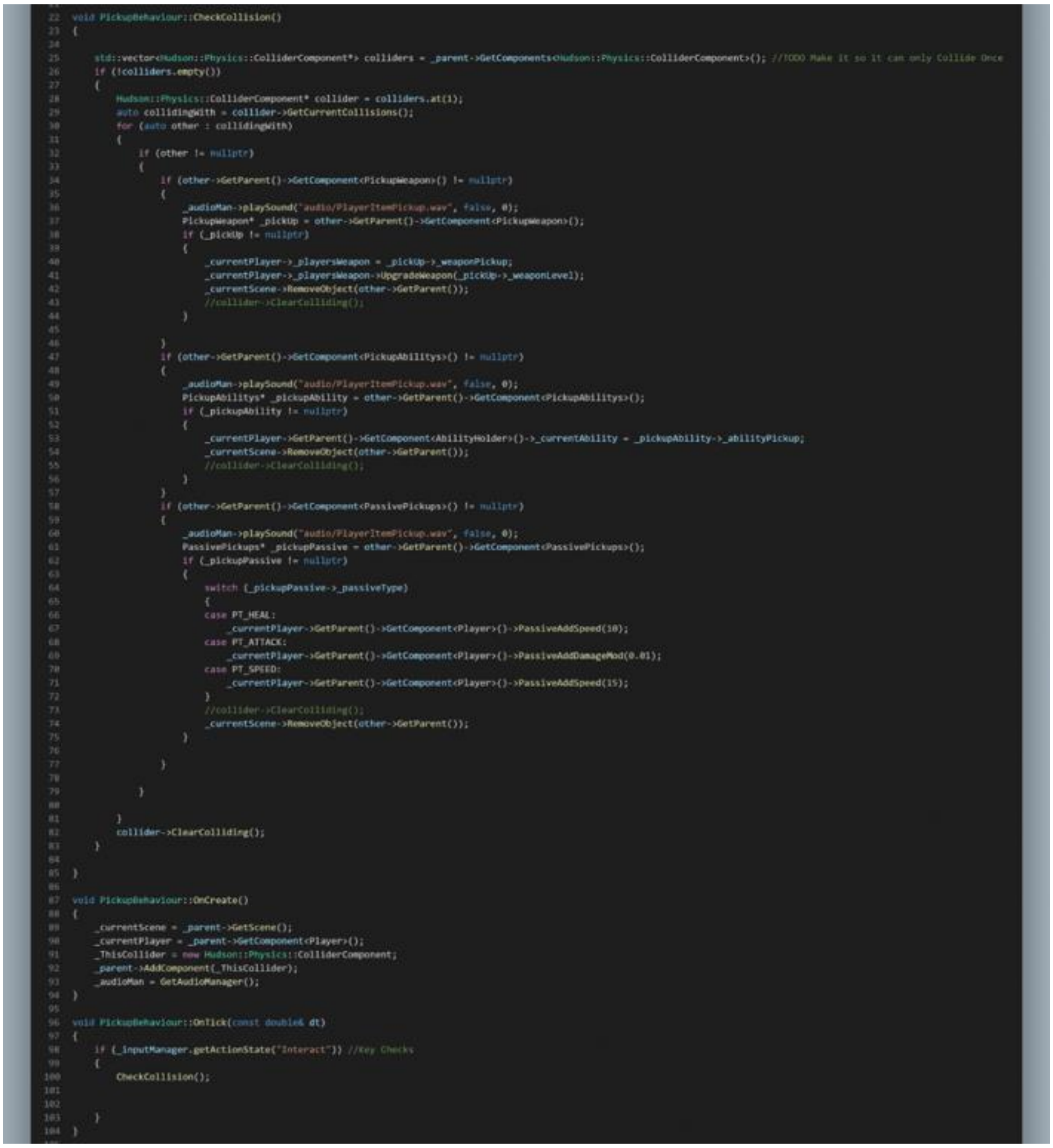
The game now needed to add passive ability. A Damage Up, Health Up and a Speed Up. I made a ability drop that worked exactly the same as the weapon drop and I add it to the functions to the player that changes there stats as well as the pickup ability function to let the player pick them up. I also made the sprites for these. I'm not an artist.



```cpp
//Function the Passive Mods will access to change the player Stats when passives are pickup in the pickup behaviour
void Player::PassiveAddMaxHealth(float additionalHealth)
{
    std::cout << "Added Health" << "\n";
    _maxHealth = _maxHealth + additionalHealth;
    _playerHealth = _playerHealth + additionalHealth;
}

void Player::PassiveAddSpeed(float additionalSpeed)
{
    std::cout << "Added Speed" << "\n";
    _playerMovementSpeed = _playerMovementSpeed + additionalSpeed;
}

void Player::PassiveAddDamageMod(float additionalDamage)
{
    std::cout << "Added Damage" << "\n";
    _playerDamageMod = _playerDamageMod + additionalDamage;
}

```



```cpp

PassivePickups::PassivePickups(glm::vec2 _spawnpos) : Behaviour("PassivePickups")
{
    _spawnPos = _spawnpos;
    _randomPassiveInt = 0;
}

PassivePickups::~PassivePickups()
{
}

void PassivePickups::OnCreate()
{
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    _passiveSprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("Passives"));
    _passiveSprite->SetGridSize(glm::vec2(3, 1));
    _passiveSprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));
    _passiveSprite->SetDepthOrder(1);
    _passiveCollider = new Hudson::Physics::ColliderComponent();

    _parent->AddComponent(_passiveSprite);
    _parent->AddComponent(_passiveCollider);
    _parent->SetName("PassivePickup");
    _parent->GetTransform().pos = _spawnPos;

    random_device rand;
    uniform_int_distribution<int> dist(0, 2);
    _randomPassiveInt = dist(rand);
    if (_randomPassiveInt == 0)
    {
        _passiveType = PT_HEAL;
        _passiveSprite->SetGridPos(glm::vec2(0, 0));
    }
    if (_randomPassiveInt == 1)
    {
        _passiveType = PT_ATTACK;
        _passiveSprite->SetGridPos(glm::vec2(1, 0));
    }
    if (_randomPassiveInt == 2)
    {
        _passiveType = PT_SPEED;
        _passiveSprite->SetGridPos(glm::vec2(2, 0));
    }
}

```



```cpp
void PickupBehaviour::CheckCollision()
{

    std::vector<Hudson::Physics::ColliderComponent*> colliders = _parent->GetComponents<Hudson::Physics::ColliderComponent>(); //TODO Make it so it can only Collide Once
    if (!colliders.empty())
    {
        Hudson::Physics::ColliderComponent* collider = colliders.at(1);
        auto collidingWith = collider->GetCurrentCollisions();
        for (auto other : collidingWith)
        {
            if (other != nullptr)
            {
                if (other->GetParent()->GetComponent<PickupWeapon>() != nullptr)
                {
                    _audioMan->playSound("audio/PlayerItemPickup.wav", false, 0);
                    PickupWeapon* _pickUp = other->GetParent()->GetComponent<PickupWeapon>();
                    if (_pickUp != nullptr)
                    {
                        _currentPlayer->_playersWeapon = _pickUp->_weaponPickup;
                        _currentPlayer->_playersWeapon->UpgradeWeapon(_pickUp->_weaponLevel);
                        _currentScene->RemoveObject(other->GetParent());
                        //collider->ClearColliding();
                    }

                }
                if (other->GetParent()->GetComponent<PickupAbilitys>() != nullptr)
                {
                    _audioMan->playSound("audio/PlayerItemPickup.wav", false, 0);
                    PickupAbilitys* _pickupAbility = other->GetParent()->GetComponent<PickupAbilitys>();
                    if (_pickupAbility != nullptr)
                    {
                        _currentPlayer->GetParent()->GetComponent<AbilityHolder>()->_currentAbility = _pickupAbility->_abilityPickup;
                        _currentScene->RemoveObject(other->GetParent());
                        //collider->ClearColliding();
                    }
                }
                if (other->GetParent()->GetComponent<PassivePickups>() != nullptr)
                {
                    _audioMan->playSound("audio/PlayerItemPickup.wav", false, 0);
                    PassivePickups* _pickupPassive = other->GetParent()->GetComponent<PassivePickups>();
                    if (_pickupPassive != nullptr)
                    {
                        switch (_pickupPassive->_passiveType)
                        {
                        case PT_HEAL:
                            _currentPlayer->GetParent()->GetComponent<Player>()->PassiveAddSpeed(10);
                        case PT_ATTACK:
                            _currentPlayer->GetParent()->GetComponent<Player>()->PassiveAddDamageMod(0.01);
                        case PT_SPEED:
                            _currentPlayer->GetParent()->GetComponent<Player>()->PassiveAddSpeed(15);
                        }
                        //collider->ClearColliding();
                        _currentScene->RemoveObject(other->GetParent());
                    }

                }

            }

        }
        collider->ClearColliding();
    }

}

void PickupBehaviour::OnCreate()
{
    _currentScene = _parent->GetScene();
    _currentPlayer = _parent->GetComponent<Player>();
    _ThisCollider = new Hudson::Physics::ColliderComponent;
    _parent->AddComponent(_ThisCollider);
    _audioMan = GetAudioManager();
}

void PickupBehaviour::OnTick(const double& dt)
{
    if (_inputManager.getActionState("Interact")) //Key Checks
    {
        CheckCollision();


    }
}
```

Small Changes

Added Flashing back to the player after the update that changed the damage function.

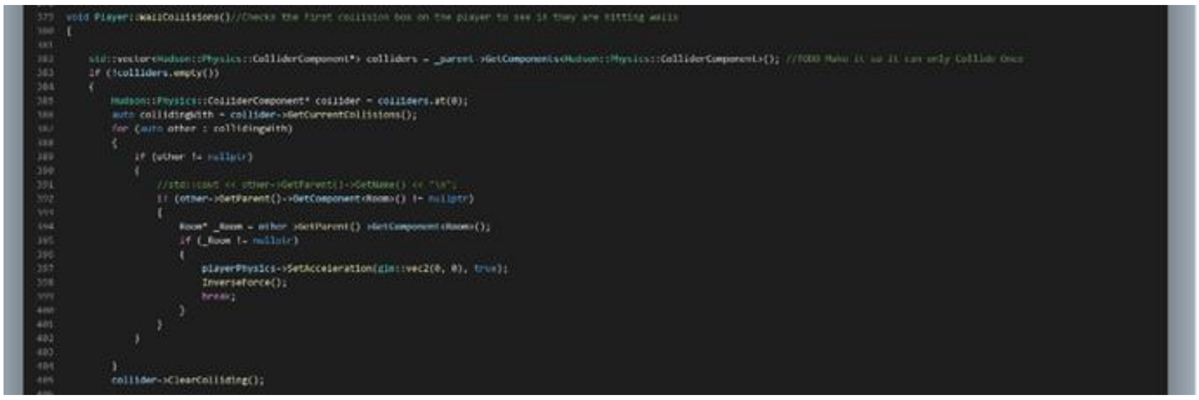
Object drop rates for the weapons. So now that wood and stone drop at a high percentage and Bronze Iron and Gold drop at a much lower rate.

And added a Maxhealth variable to set the players current health.

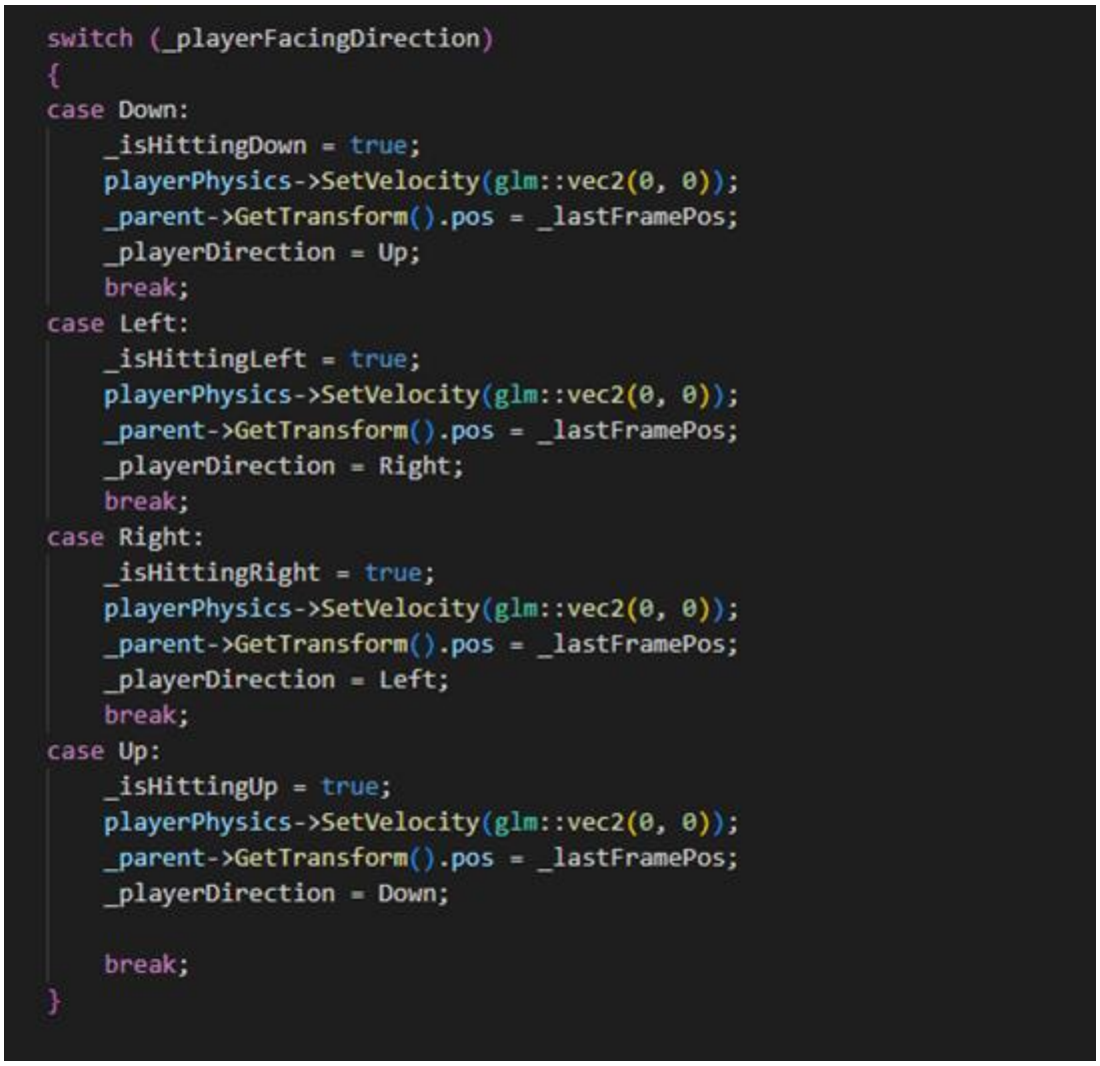
Finally fixed the bomb to call the correct function.

<u>Wall Collisions with the player</u>

Wall collisions work for a box collision on the player and checking it is a wall. It then run a function that calls an inverse force as well as setting the player back to there last frame. This took a lot of trial and error throughout a long night of test and this was the best it come out as. Functionally it works but does look a bit framey. This later down the road cause a lot of issues when implementing the room loader but a Thursday Fix solved all these problems. Thanks to Mathew helping me out. This was a to do with the collider code being a pain to implement.



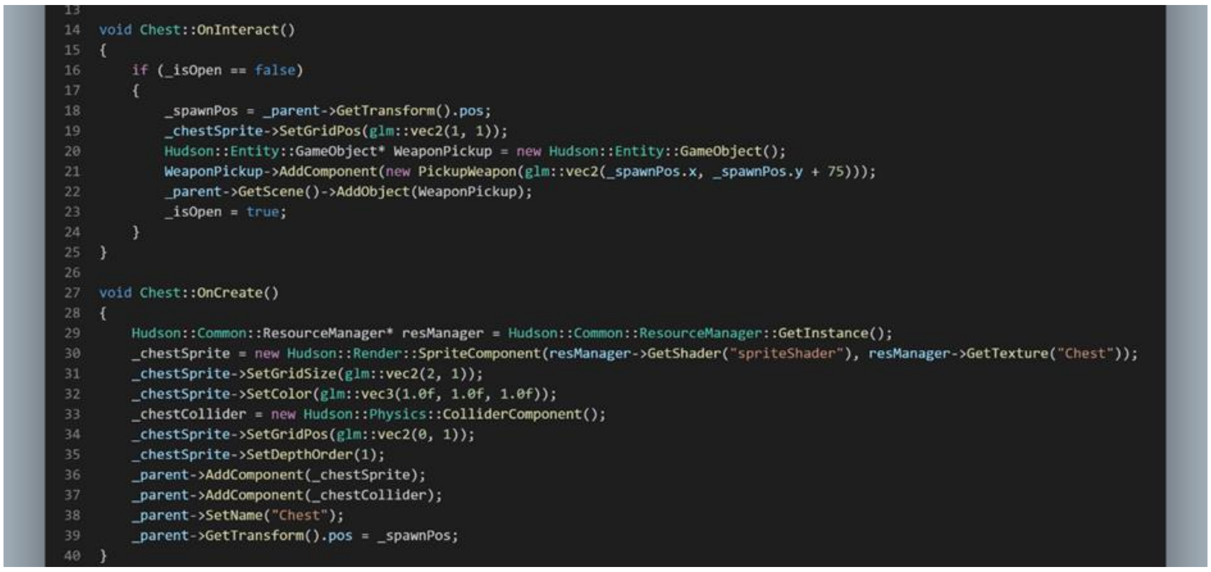
```
void Player::WallCollisions()//Checks the first collision box on the player to see if they are hitting walls
{

    std::vector<Hudson::Physics::ColliderComponent*> colliders = _parent->GetComponents<Hudson::Physics::ColliderComponent>(); //TODO Make it so it can only Collide Once
    if (!colliders.empty())
    {
        Hudson::Physics::ColliderComponent* collider = colliders.at(0);
        auto collidingWith = collider->GetCurrentCollisions();
        for (auto other : collidingWith)
        {
            if (other != nullptr)
            {
                //std::cout << other->GetParent()->GetName() << "\n";
                if (other->GetParent()->GetComponent<Room>() != nullptr)
                {
                    Room* _Room = other->GetParent()->GetComponent<Room>();
                    if (_Room != nullptr)
                    {
                        playerPhysics->SetAcceleration(glm::vec2(0, 0), true);
                        InverseForce();
                        break;
                    }
                }
            }

        }
        collider->ClearColliding();
```



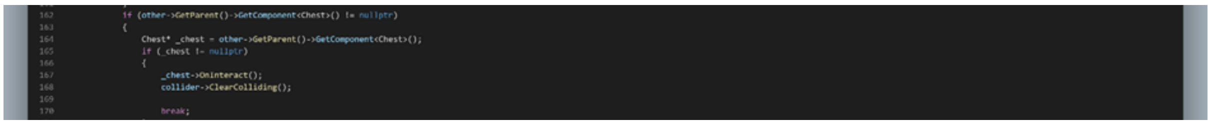
```
switch (_playerFacingDirection)
{
case Down:
    _isHittingDown = true;
    playerPhysics->SetVelocity(glm::vec2(0, 0));
    _parent->GetTransform().pos = _lastFramePos;
    _playerDirection = Up;
    break;
case Left:
    _isHittingLeft = true;
    playerPhysics->SetVelocity(glm::vec2(0, 0));
    _parent->GetTransform().pos = _lastFramePos;
    _playerDirection = Right;
    break;
case Right:
    _isHittingRight = true;
    playerPhysics->SetVelocity(glm::vec2(0, 0));
    _parent->GetTransform().pos = _lastFramePos;
    _playerDirection = Left;
    break;
case Up:
    _isHittingUp = true;
    playerPhysics->SetVelocity(glm::vec2(0, 0));
    _parent->GetTransform().pos = _lastFramePos;
    _playerDirection = Down;

    break;
}
```

Adding a Chest

Added a chest that when the player attacks will spawn a weapon drop. This will later be added to the Tile editor so the developer can place them into the level, as part of the design. This literal set up a sprite and then when a open function is called to in a collision will open the sprite as well as spawn a new object.



```cpp
void Chest::OnInteract()
{
    if (_isOpen == false)
    {
        _spawnPos = _parent->GetTransform().pos;
        _chestSprite->SetGridPos(glm::vec2(1, 1));
        Hudson::Entity::GameObject* WeaponPickup = new Hudson::Entity::GameObject();
        WeaponPickup->AddComponent(new PickupWeapon(glm::vec2(_spawnPos.x, _spawnPos.y + 75)));
        _parent->GetScene()->AddObject(WeaponPickup);
        _isOpen = true;
    }
}

void Chest::OnCreate()
{
    Hudson::Common::ResourceManager* resManager = Hudson::Common::ResourceManager::GetInstance();
    _chestSprite = new Hudson::Render::SpriteComponent(resManager->GetShader("spriteShader"), resManager->GetTexture("Chest"));
    _chestSprite->SetGridSize(glm::vec2(2, 1));
    _chestSprite->SetColor(glm::vec3(1.0f, 1.0f, 1.0f));
    _chestCollider = new Hudson::Physics::ColliderComponent();
    _chestSprite->SetGridPos(glm::vec2(0, 1));
    _chestSprite->SetDepthOrder(1);
    _parent->AddComponent(_chestSprite);
    _parent->AddComponent(_chestCollider);
    _parent->SetName("Chest");
    _parent->GetTransform().pos = _spawnPos;
}
```



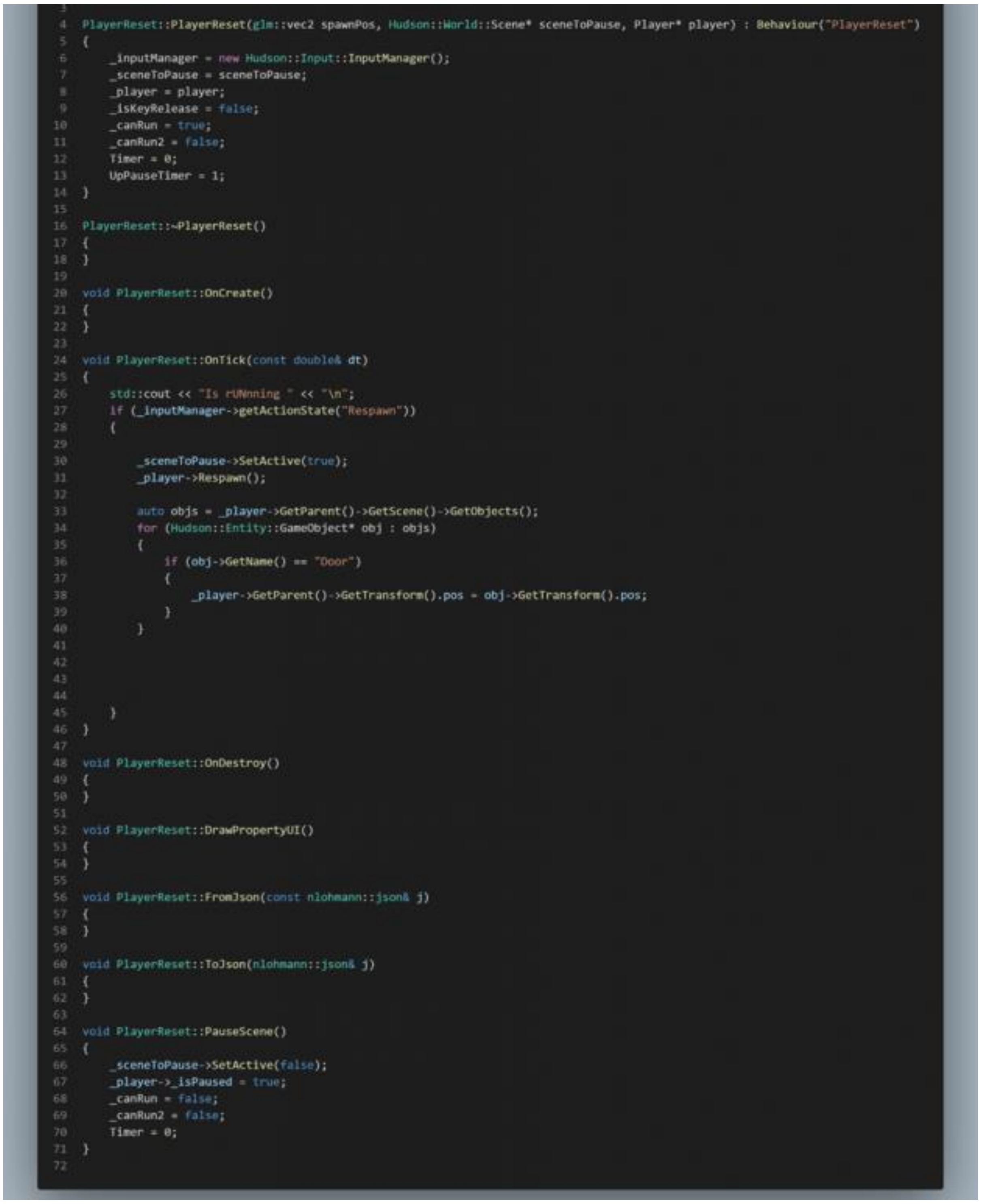
```cpp
if (other->GetParent()->GetComponent<Chest>() != nullptr)
{
    Chest* _chest = other->GetParent()->GetComponent<Chest>();
    if (_chest != nullptr)
    {
        _chest->OnInteract();
        collider->ClearColliding();

        break;
```

Finally there was the Room making in the tool that had been made. This was a not a fun experience but needed doing. Props to Kenny for getting a majority of the rooms made. We made 6 on Thursday but a lot got reset in later build so he did go back and make more.

Me and Harvey sat and did the weapon balancing making sure the damage and abilities all had the correct stats throughout all of the weapon level, making the cooldown correct on the ability and passives adding a fun amount.

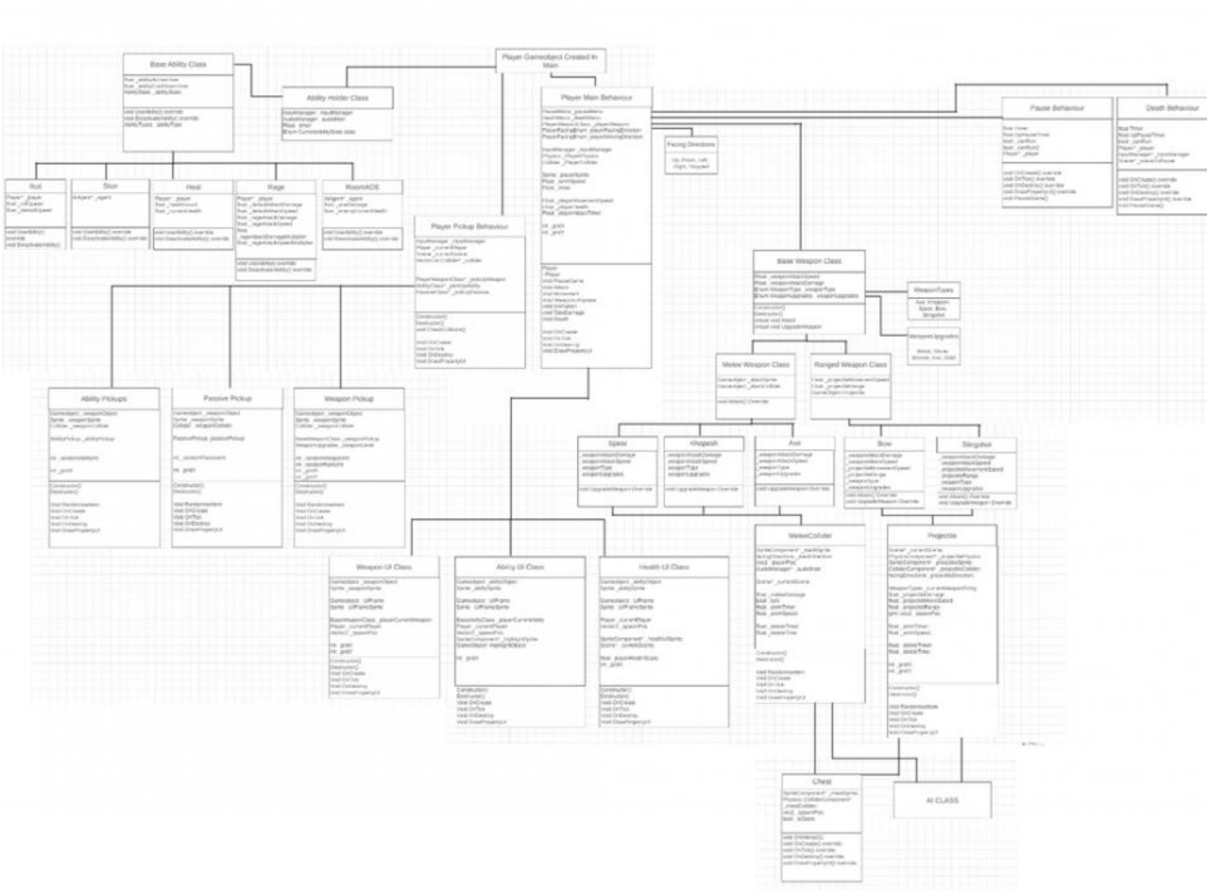
Friday

A good handful days of not sleeping, I added a pause menu and a reset on the players death this uses the same sort of code and waits for a keypress to carry on. This is set to K to respawn and ESC to pause the game. This was done by using the “Chris Method” which was teleporting the door to the player to generate the new room and then resetting the stats of the player. This functionally works and was the only way not to have some sort of memory issues, and as it was the day the time way wasn’t an issue. I also helped bailey with some of the Boss Stuff, just some collisions issues and magic to get it work.



```cpp

PlayerReset::PlayerReset(glm::vec2 spawnPos, Hudson::World::Scene* sceneToPause, Player* player) : Behaviour("PlayerReset")
{
    _inputManager = new Hudson::Input::InputManager();
    _sceneToPause = sceneToPause;
    _player = player;
    _isKeyRelease = false;
    _canRun = true;
    _canRun2 = false;
    Timer = 0;
    UpPauseTimer = 1;
}

PlayerReset::~PlayerReset()
{
}

void PlayerReset::OnCreate()
{
}

void PlayerReset::OnTick(const double& dt)
{
    std::cout << "Is rUNning " << "\n";
    if (_inputManager->getActionState("Respawn"))
    {

        _sceneToPause->SetActive(true);
        _player->Respawn();

        auto objs = _player->GetParent()->GetScene()->GetObjects();
        for (Hudson::Entity::GameObject* obj : objs)
        {
            if (obj->GetName() == "Door")
            {
                _player->GetParent()->GetTransform().pos = obj->GetTransform().pos;
            }
        }




    }
}

void PlayerReset::OnDestroy()
{
}

void PlayerReset::DrawPropertyUI()
{
}

void PlayerReset::FromJson(const nlohmann::json& j)
{
}

void PlayerReset::ToJson(nlohmann::json& j)
{
}

void PlayerReset::PauseScene()
{
    _sceneToPause->SetActive(false);
    _player->_isPaused = true;
    _canRun = false;
    _canRun2 = false;
    Timer = 0;
}

```

Critical Evaluation to come bright and early on Friday afternoon.



-

<u>Critical Reflect</u>

The final project as a whole is now completed, the game has got many floors in the system and potential crashes, but we spent the last night ironing them out. The main crash now is to do with the sound engine which is impossible to fix due to the problem being in the DLL in Irklang library. Which it to late to explore. Looking at the project as a whole I'm proud of what the team has produced, it is still under the scope which we set out but development time for each feature was ridiculously late with the whole engine not being ready till the 30th of January. This meant dev time was squeeze and meant Harvey and Me on the dev side a handful of all nights and many 18 hours+ days of just coding to get things to work.

The engine has many floors and holes in how it works and many design decisions where not taking into account from the start. This is due to none of us ever working on an engine before and not understanding the small details from the start. I'm not going to bring anyone from the engine team down as everything they made in there is beyond my scope. If you left it to me to make an engine id have a box spinning and nothing else. So, anything is better than nothing.

The game also has some glitches and not intended features such as the player flash not working and the rooms all being extremely similar as the tool had to be fixed and files got reworked but the fundament mechanics of a rouge like are there such as melee combat, ranged combat, 25 unique weapons, abilities and passives which add to the complexity and depth of the game. The room selection is the best we could do with 48 hours left till the hand ins as the tool literal was done Thursday night. It we had more time Id love to make more and implement a better system, but it was out of my hands.

My performance in the project was a new level. I stopped living and become one with for the last week or so. I learnt many new things about C++ openGL teamwork and Github. I personally wish that there was more time but the time I had was used and structured to the best of my ability and constantly pushing the boundaries of the engine and the game. The player and all the upgrades for the weapon upgrades, abilities and passives which I worked on all hit the original scope and I'm proud of myself for that, even taking other people jobs that they weren't going to finish such as the in game UI the pause menu and wall collision which was all not originally my to work to do.

I've had a good time on this project, but I don't want to look at code for a good week now and want to sleep.