Game mechanics development project report (CMP302)

# Summary

This mechanic is an alternate ammo type mechanic that allows the player to shoot different types of bullets that has different effects on the enemies depending on the ammo type. It is similar to the mechanic seen in Call of Duty: Black Ops 3’s (Activision 2015) zombie game mode with the Double Pack a Punch feature. My feature includes multiple different bullets that can be fired Including:

* Basic Ammo: Will disappear on collision
* Fire Ammo: Once it hits the enemy it will spawn a lava pool below the enemy’s feet that damages any enemy to step in it
* Thunder Ammo: Sends the enemy flying into the sky before killing the enemy
* Turned Ammo: Sets the Enemy to target the other enemies instead of the player

A video demonstrating these systems is available here: ​

LINK VIDEO HERE

# Requirement Specification

## Introduction

### Purpose

I created this mechanic to be a very base level implementation of the alternate ammo type mechanic. The code is easy to build upon with new ammo types and different effects for the ammo so can be easily progressed and implemented into future games and projects.

## Overall Description

### Product Perspective

The Primary aim of this project was to create a working alternate ammo type system with each bullet type having different effects on hitting enemies. Other parts of the product such as the moving mannequins are there to provide testing for the player and developer.

### Product Functions

1. Allow the user to pick up weapon and shoot
2. Allow the user to see what bullet was shot through the UI
3. Allow the user to target the training dummies to test the bullet effects
4. Allow a designer or programmer to derive from the already created bullet types to create new effects
5. Allow a designer or programmer to create new bullet types and implement them into the system.

### User Classes & Characteristics

This project is aimed at a software engineer, or games designer who is familiar with the Unreal Engine 5 product pipeline.

A software engineer should not be required to extend on spells that have already been created, although one might be required to engineer new bullet types depending on their gameplay complexity.

### Design & Implementation Constraints

I did my best to stick fully to cpp programming for my implementation in ue5. This helped keep my blueprints very concise and easy to follow.

## System Features

### Ammo Type: Fire Ammo

**Description:**

A Bullet shot by the player. Once the bullet hits the target it spawns a lava pool bellow the enemy that damages any enemy to step in it. After a period of time the pool of lava disappears.

**Stimulus / response sequences**​:

For gameplay, the bullet functionality is accessed by mouse input.

For development, the FireAmmoActor will open the cpp for the corresponding bullet type and the blueprints can be accessed by BP\_FireAmmo.

**Functional requirements**:​

*REQ-1: Valid target*

The Fire Ammo requires an enemy for the bullet to take effect and do its sequence.

*REQ-2: Bullet selected*

The bullet must be chosen by the random number generator for it to fire.

*REQ-3: Fire input*

There must be “fire” or “trigger” input by the player for the bullet to fire.

### Ammo Type: Thunder Ammo

**Description:**

A Bullet shot by the player. Once the bullet hits the target the target is launched into the air before being killed once it reaches a certain height.

**Stimulus / response sequences**​:

For gameplay, the bullet functionality is accessed by mouse input.

For development, the ThunderAmmoActor will open the cpp for the corresponding bullet type and the blueprints can be accessed by BP\_ThunderAmmo

**Functional requirements**​:

*REQ-1: Valid target*

The Fire Ammo requires an enemy for the bullet to take effect and do its sequence.

*REQ-2: Bullet selected*

The bullet must be chosen by the random number generator for it to fire.

*REQ-3: Fire input*

There must be “fire” or “trigger” input by the player for the bullet to fire.

### Ammo Type: Turned Ammo

**Description:**

A Bullet shot by the player. Once the bullet hits the target the target targets the enemies instead of the player.

**Stimulus / response sequences**​:

For gameplay, the bullet functionality is accessed by mouse input.

For development, the TurnedAmmoActor will open the cpp for the corresponding bullet type and the blueprints can be accessed by BP\_TurnedAmmo

**Functional requirements**​:

*REQ-1: Valid target*

The Fire Ammo requires an enemy for the bullet to take effect and do its sequence.

*REQ-2: Bullet selected*

The bullet must be chosen by the random number generator for it to fire.

*REQ-3: Fire input*

There must be “fire” or “trigger” input by the player for the bullet to fire.

### Ammo Type: Standard Ammo

**Description:**

A Bullet shot by the player. Once the bullet hits anything the bullet disappears

**Stimulus / response sequences**​:

For gameplay, the bullet functionality is accessed by mouse input.

For development, the ProjectileAmmoActor will open the cpp for the corresponding bullet type and the blueprints can be accessed by BP\_FirstPersonProjectile

**Functional requirements**​:

*REQ-1: Valid target*

The Fire Ammo requires an enemy for the bullet to take effect and do its sequence.

*REQ-2: Bullet selected*

The bullet must be chosen by the random number generator for it to fire.

*REQ-3: Fire input*

There must be “fire” or “trigger” input by the player for the bullet to fire.

### Test Dummy

**Description:**

A test dummy is a practice target for the test environment that responds to the players bullets.

**Stimulus / response sequences**​:

For gameplay, the player has no direct control over the dummy. Dummies will respond to bullets the player shoots.

For development, the BP\_TestDummy will open the blueprint and the TestDummyActor will open the cpp for it.

**Functional requirements**​:

*REQ-1: Enemy Tag*

*The Dummy will have an enemy tag to make the bullets respond to hitting them*

*REQ-2: AI Control*

Test dummies target the player and constantly move towards them

### Test Map

**Description:**

A test area for the player to test their bullets in.

**Stimulus / response sequences**​:

For gameplay, the player will explore this area but will not be able to leave the walls.

For development, the map file will open in the unreal engine editor.

**Functional requirements**​:

*REQ-1: Test Dummies*

The test area must have a set of test dummies for the player to use their bullets on.

*REQ-2: Enclosed area*

The area must be closed so the player cannot leave this area, since this is a test range.

### Player character

**Description:**

The player character must be able to move around the map according to the player input. The player must also be able to shoot bullets.

**Stimulus / response sequences:**

In game, the player character will respond to the correct movement keys (W/A/S/D) to move around the scene.

In editor, the BP\_FirstPersonCharacter blueprint will open in the blueprint editor and the FirstPersonCharacterCPP will open the cpp.

**Functional requirements:**

*REQ-1: Respond to input*

The player character must respond to input to move.

### Weapon

**Description:**

The weapon must spawn bullets when input is pressed and be attached to the player character

**Stimulus / response sequences:**

In game, the weapon can be picked up by walking over it and can be shot using left mouse button.

In editor, the BP\_Weapon\_Component blueprint will open in the blueprint editor and the WeaponCompCPP will open the cpp.

**Functional requirements:**

*REQ-1: Be Picked Up*

The player character must be able to pick it up to use it.

REQ-2: Shoot

The weapon must respond to user input by shooting

REQ-3: Change Bullet Types

The weapon must randomly generate a bullet type to shoot

## Other non-functional requirements

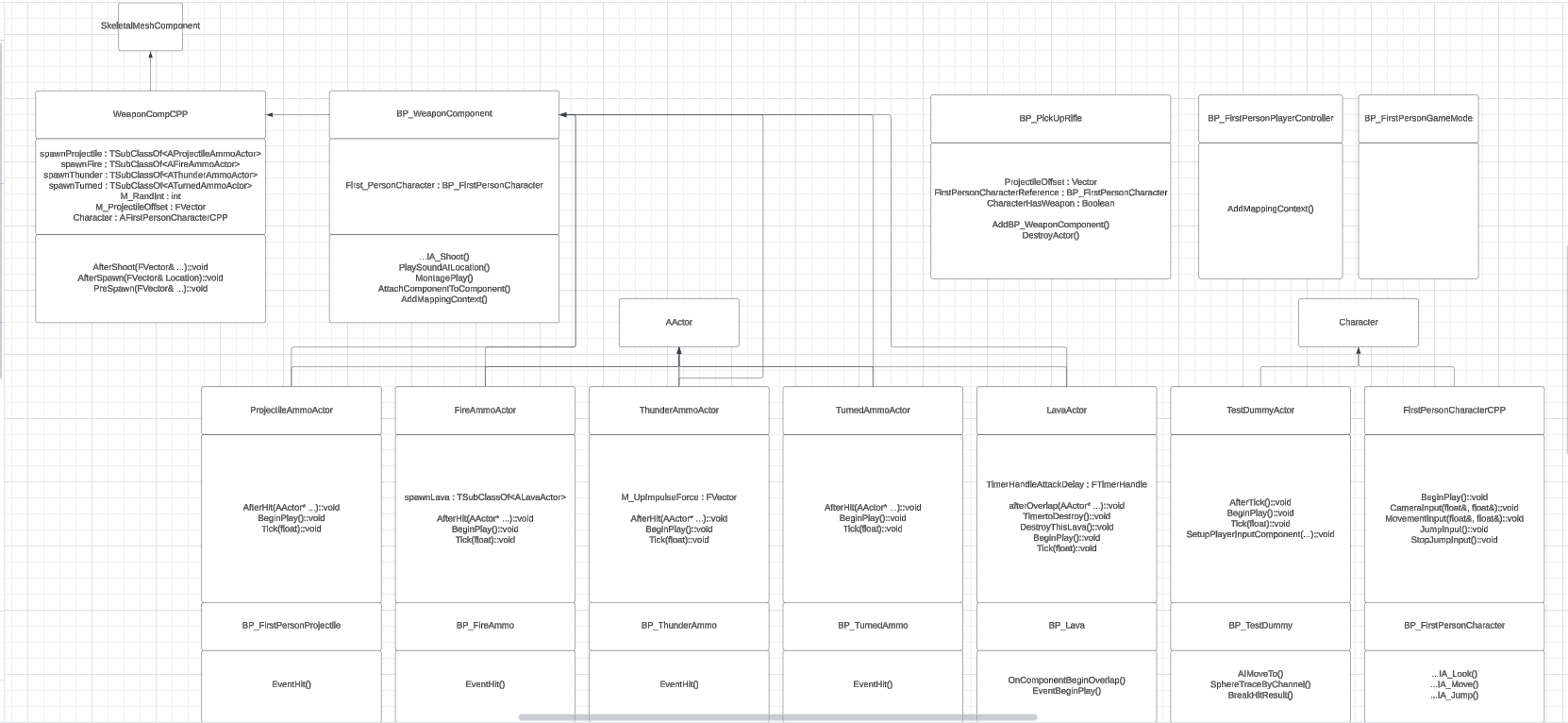
### Software quality

The software implementation must be kept at a high quality and concise standard. All C++ code will be written with proper object-orientation in mind, but also towards the Unreal Engine 5 online coding standard. (​*Coding Standard*​)

While the development of this project was in a waterfall manner, testing and iteration was performed regularly to assure the validity and safety of the code base.

# Method

## UML Diagram



*This is readable if you zoom in, sorry for making it small*

## Technical Discussion

Throughout my code I had to spawn actors within my cpp code, to do so I used subclasses of actors. This is because trying to spawn the class would spawn the whole class whereas all my models were stored in the blueprint. So, I used a subclass to spawn make sure to spawn the blueprint of the actor by setting the variable to the BP instead of the CPP. I did this for each ammo type and for the Lava in the Fire ammo type. In my code I also had to make use of tags. One part that I had to learn was how to add and remove however this proved to be easy enough by just getting the tags array and rewriting the index I wanted to change. I also made good use of casting in my code where I created a new variable of the type I wanted to cast to and putting it in an if statement to make sure the cast worked before trying anything. Then using the new variable, I could use the functions of the new variables type to do as I wished to the variable I was casting from.

## Development

My development was straightforward for the most part, I started in a first-person blueprint project. From there I created all my ammo types and mechanic functionality in blueprint before then trying to convert as much as I could to cpp.

# Conclusion:

For the most part I accomplishes what I set out to do for this project. The only shortfall being that I couldn’t find a way to make the AIMoveTo function work in cpp. I would have really liked to of managed to make everything I created myself (not given by unreal) in cpp but I fell short on time. Another thing I would of liked to of done with more time would be add more bullet types, such as a poison mist that does tick damage over time or one that makes all enemies within a radius get slowed majorly. Other than that all my implementation went right and I am happy with the mechanic I created.

## References: