Game mechanics development project report (CMP302)

By Joseph Roper - 1901881

Summary

The mechanics I have made are revolved around improving the first-person shooter experience, I have taken ideas from well-known FPS games such as Counter-Strike: Global Offensive and applied them together.

1. The first equipped gun can shoot bullets that penetrate through walls
    a. I have three different materials that can be shot through, each one has a different density, stopping the bullets after colliding with too many objects.
2. The second gun behaves as a shotgun
    a. Shoots multiple bullets at once representing a shotgun spread
    b. Has a circular cross hair attached to represent bullet spread that can be changed in size to affect the bullets trajectory
    c. Bullets cluster towards the centre of the crosshair to promote realism.
3. User can peak/lean around corners while shooting

A video demonstrating these systems is available here: https://youtu.be/jvw3MJIjEr0

Requirement Specification

Introduction

Purpose

Many games try to perfect the shooter experience each year by making new mechanics to promote a realistic shooter. The purpose of this project was to investigate how those mechanics could be structured in code and implemented into one project working along side each other to later use in my own shooting game.

The project is a basic framework for an FPS title, the components delivered include a shotgun with a realistic bullet spread and crosshair not found in most recent games, accurate bullet penetration and user movement. The mechanics are designed to be easily implemented into new and current games and built upon if desired.

Overall Description

Product Perspective

The primary aim of this project was to deliver realistic gun behaviours into a game environment.

Product Functions

1. Allow the user to switch between gun modes with the number keys
2. Allow the user to see an accurate crosshair for aiming
3. Allow the user to target different materials to test the bullet penetration effects
4. Allow the user to change the shotgun bullet spread in real time using the mouse wheel
5. Allow the user to peak around corners while aiming using the ‘Q’ and ‘E’ keys

User Classes & Characteristics

This project is aimed at a software engineer, or games designer who is familiar with the Unreal Engine 4 product pipeline.

A software engineer should not be required to extend on the weapon mechanics, although a designer may want to change the gun model, and the other aesthetics to fit their game.

Design & Implementation Constraints

Some basic constraints are specified in the coursework implementation document.

The developer has decided to self-impose the constraint of not using the Unreal Gameplay Framework for the weapon mechanics. UE4 provides a gameplay framework for projectiles. However, the developer wanted to do this outside of the gameplay framework for the sake of extensibility and versioning safety.

Since there is no artist assigned to the project, the developer used unreal engines in house free models.

System Features

Bullet Penetration

Description:

A gun that can be shot by the player at objects. When shot at an object it leaves a bullet hole, depending on the material of the object the bullet is able to pass through and come out of the other side. After a set period of time the bullet holes will disappear like they do in most games to save processing power and improve performance.

Stimulus / response sequences:

For gameplay, the gun that enables bullet penetration is automatically equipped when the program is first opened. Although to access it the user can press ‘1’ on their keyboard. The gun is aimed by moving the mouse and to shoot it is the mouses left click.

Functional requirements:

REQ-1: Valid material

The ‘Bullet Penetration’ requires a valid target to pass through. In this instance it is any object with the plaster, wood or concrete material applied to it.

REQ-2: Gun selected

The gun must be selected by the appropriate key bind before it can be cast.

REQ-3: Fire input

There must be “fire” input by the player for the gun to shoot.

Shotgun Bullet Spread

Description:

A gun that can be equipped and shot by the player. Once the shotgun is selected the default cross hair will disappear and a circle will appear on screen. When the gun is fired multiple bullets will be shot and land within the new crosshair, each bullet will spawn a bullet hole like the first gun.

Stimulus / response sequences:

For gameplay, the shotgun is not equipped when the program is first opened to access it the user can press ‘2’ on their keyboard. The gun is aimed by moving the mouse and to shoot it is the mouses left click. When using the mouses scroll wheel the user can change the radius of the bullet spread and crosshair.

Functional requirements:

REQ-1: Gun selected

The gun must be selected by the appropriate key bind before it can be cast.

REQ-2: Fire input
There must be “fire” input by the player for the gun to shoot.

Player Leaning

Description:
The user can lean left and right to peak around corners providing an advantage when in combat as the user can stay behind cover while firing at an enemy.

Stimulus / response sequences:

For gameplay, the leaning mechanic can be accessed with the ‘Q’ and ‘E’ key bindings.

Functional requirements:

REQ-1: Correct Input
The user must use the correct key bindings to use the mechanic

Test Map

Description:
A test area for the player to explore and test the mechanics created.

Stimulus / response sequences:

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

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

REQ-1: Enclosed area
The area must be closed so the player cannot leave this area, since this is a test range.

REQ-2: Penetrative Materials
The area must have objects with the materials created applied to test the bullet penetration mechanic.

REQ-2: Cover Objects
The area must have objects which will allow the player to take cover behind and aim at targets for the leaning mechanic demonstration.

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 their weapon

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 player blueprint will open in the blueprint editor.

Functional requirements:
REQ-1: Respond to input
The player character must respond to input to move and fire the weapon.

Other non-functional requirements

Performance requirements

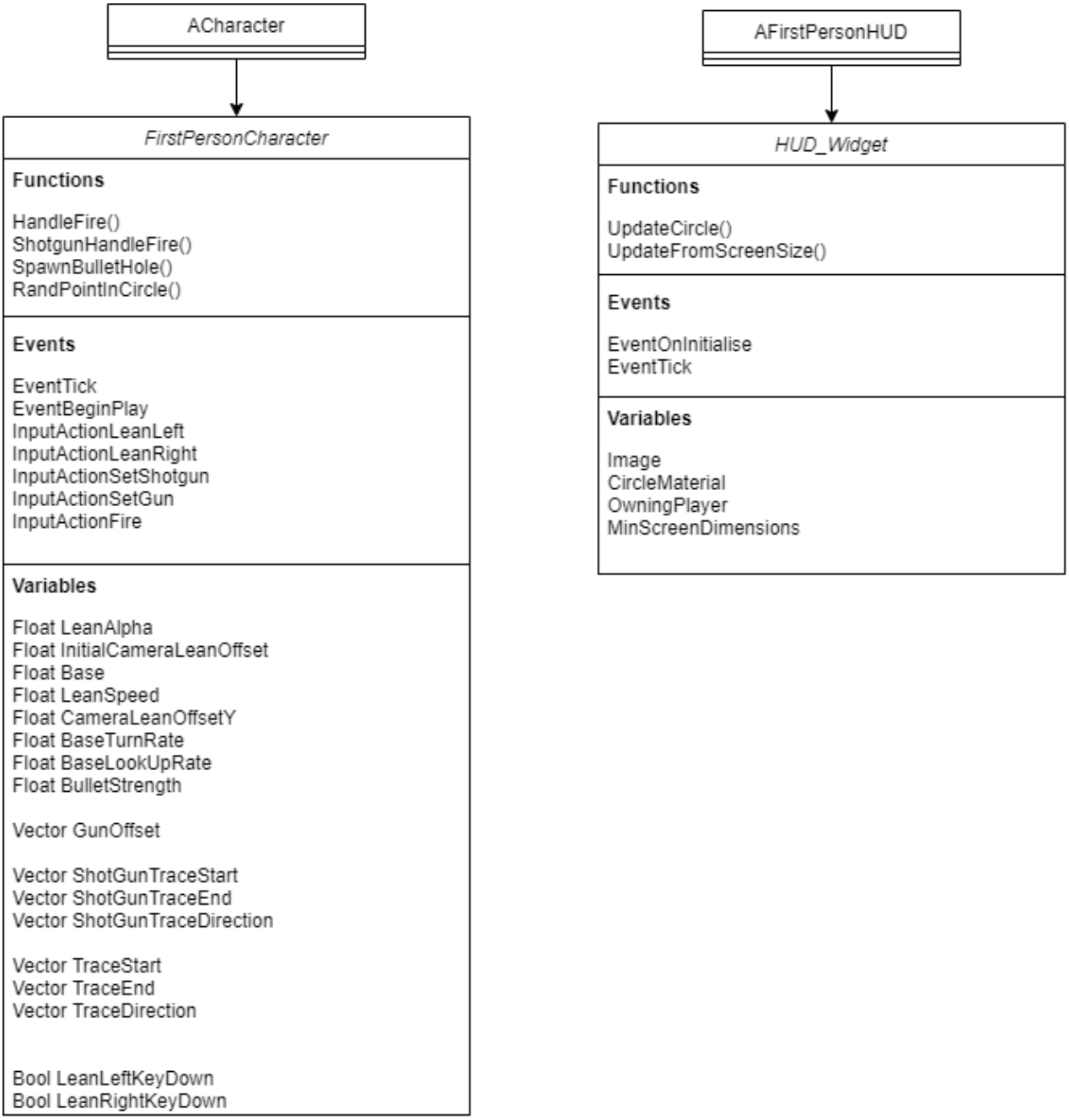
While the game is in play, there must be a stable frame rate for the mechanics to function correctly. A low frame rate could cause effects like the bullet trajectory being miscalculated, when it should be a smooth animation. Low frame rates could also delay response to input actions.

Software quality

The software implementation must be kept at a high quality and concise standard. The developer strongly believes in the KISS principle (KISS) for development. All C++ code will be written with proper object-orientation in mind, but also towards the Unreal Engine 4 online coding standard. (*Coding Standard*)

While the development of this project used the agile methodology, testing and iteration was performed regularly to assure the validity and safety of the code base.

UML Diagram



Any functions that use the parameters '(...)', are standard Unreal Engine functions that have three or more parameters, and as such were cut from the UML diagram for the sake of readability.

Technical Discussion:

Bullet Penetration

For the bullet penetration to work I first needed to adapt some materials. To do this I created 3 new surface types: plaster, wood, and concrete. I then created the corresponding physics

materials and set it to the surface types created. I found three materials that matched the description of the types created and applied the physics material. In my scene I have multiple objects with the materials created applied, these will be shot at by the player.

When the left mouse button is pressed and the default cross hair is in use the handle fire function in run, this is where my bullet penetration code will be made. In this function I first play the shoot animation and sound and then go to the actual code changed. I first set the bullet strength float to 1 by default. This value will decrease with every object hit based on the materials density. When the strength gets to 0 the bullet will stop. I then created an ignore list which will take in actors that have already been hit so that they are excluded from the line trace calculation. For the calculation of the bullet line, I create three vectors:

Trace Start, Trace End and Trace Direction. For trace start I just set it to the cameras location as that where we want the bullet to come from. For the direction I used the forward vector of the camera which is where the player is looking. To calculate the trace end I multiplied

To calculate the end of the bullet line I had to multiply the trace direction by an arbitrary value for the bullets distance, I used 1000000.0f. This was added together with the trace start vector. This basically calculates where the bullet is going to go for how long and from where.

The trace vectors calculated are then used by the line trace node to actually create the bullet line, the ignore list was added. If the line hits anything it spawns the bullet hole onto the object hit.

To do this I created a function called SpawnBulletHole. This function takes in the hit result as a parameter. Within the function it uses the spawn decal node which takes the parameter along with the hit location. However, the texture needs to be rotated so that it is flush with the object hit this is done by rotating the texture along the x axis using the impact normal as a base. I then set the life span of the bullet hole to be deleted after 10 seconds to save computing power. When implementing this feature the rendering of the bullet hole kept being paused when I moved away from it, to fix this I made it, so the texture fades out at a larger distance using the set fade node.

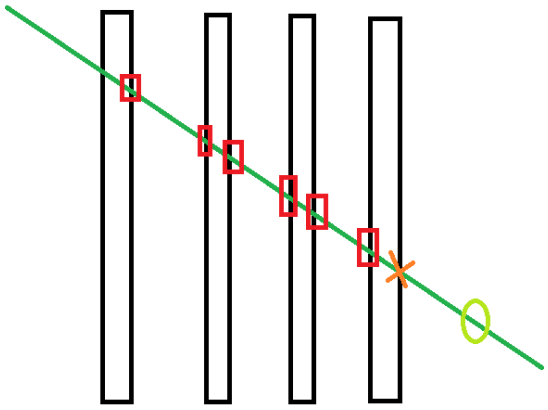
After the bullet hole is spawned, I need to check if it should pass though the object hit. To do this I retrieve the surface type hit and if it has one of the special materials applied, it will have a density value. This density value is taken away from the bullet's strength. So, if the density is 0.7 it will take that away from the bullet strength which is 1 resulting in the bullet strength of 0.3 which is greater than 0 so the bullet penetrates the object hit. If the strength was less than 0 the bullets trajectory will be stopped. If the bullet is stopped the code will do nothing afterwards however if it passes through it needs to calculate another bullet hole for the bullets exit location.

The bullets exit location is calculated by first adding the actor that was just hit onto the ignore list. This is achieved by performing a line trace in the opposite direction from the bullet lines end onto the bullets exit location on the object. I first started by getting the distance of the new trace by

retrieving the max size of the object penetrated and multiplying it by 2 to make sure that the object will be hit no matter the bullets angle.

To calculate the new trace, I need to make two new vectors the trace start and end. To calculate the trace start I need to get the direction of the line multiplied by the trace distance added onto the entry location. If you look at the diagram the bullet is travelling from left to right, we want the new line trace to go in the opposite direction so from the bottom right to the top left until it hits the penetrated object. The light green circle is our new start location which has just been calculated.

To get the trace end we want it to be the trace direction value multiplied by the opposite of the trace distance, so it goes from right to left added onto the trace start. Like the diagram at the orange cross a new hit will be achieved and that object will be added to an ignore list. If the object that has been hit is not the original object penetrated like in the diagram the line trace code will be repeated until it is hit, each loop the trace start will be updated to the new impact point so that the next hit can be calculated without fail.



When the line trace exit location is calculated a new bullet hole will be applied. The code is then repeated from the original line trace each time starting from the new impact points location until the bullet is stopped.

Shotgun Spread

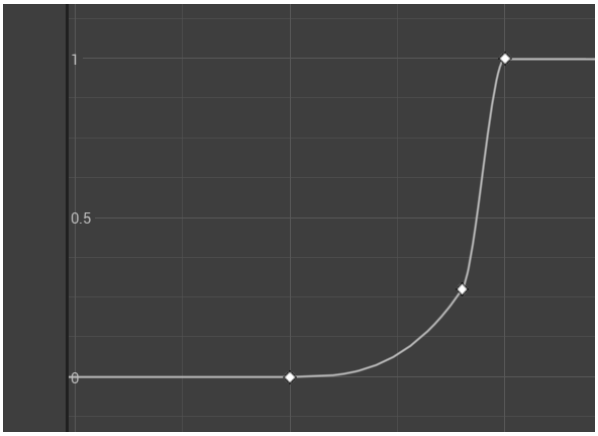
For the shotgun spread mechanic I have two main activities happening the creation of the new cross hair and the shotgun fire effect.

I am first going to explain how I made the fire mechanic. When the left mouse is pressed it checks whether the default crosshair is active based off a bool, I created call DefaultCrosshairBool if true it means the normal crosshair is applied onto the screen when false that means the shotguns crosshair is active. If the bool is false, my shotgun fire function is run.

Within this function I first play the shoot animation and sound and then go to the actual code changed. I first start by creating a for loop which is based off the number of bullets per shot. I have set the number of bullets float to be 25 meaning the code is ran 25 times spawning 25 bullets.

I then calculate a random point in a circle based off the bullet angle which is set to be the same as the crosshairs. I retrieve the tangent of the bullet angle and multiply it with the bullet distance to get the radius of the circle. Within this the bullets angle is set to be a random angle within the circle by setting it to a float in the range of 0 – 360.

I then calculate the bullets distance from the centre of the circle based off the spread curve I made, which will make it so the bullets cluster towards the middle like an actual shotgun. It does this by making it more likely to get a value returned within the curve on the picture. So, I get a float from the curve and times it against the radius of the circle and use that as the max range from 0.



These values are then used to calculate point x and y. To get point x I got the cosine of the bullets angle multiplied with the distance from the centre. For the y I did the same however I got the sine of the angle.

I then had to calculate the bullet line, to do this I made three vectors Trace Start, Trace End and Trace Direction. For trace start I just set it to the cameras location as that where we want the bullet to come from. For the direction I used the forward vector of the camera which is where the player is looking.

To calculate the end of the bullet line I had to add three vectors. The first vector was the trace direction multiplied by the bullet distance which was then added together with the trace start vector. This basically calculates where the bullet is going to go for how long and from where. The second vector uses the cameras right vector and multiplies it against point x from the circle function. Finally, the third vector is the cameras up vector times against point y. These two vectors are used to pinpoint where in the crosshair the bullet should land.

The trace vectors calculated are then used by the line trace node to create the bullet line. If the line hits anything a bullet hole is applied to that location using the spawn bullet hole function discussed previously.

To create the cross hair for the shot gun I started with creating a new material, initially I was going to use a texture for the crosshair but because I wanted the user to be able to change the size of the crosshair it wouldn't work. As if I was to scale the texture up the thickness of the crosshair would also get larger resulting in the players view to get obstructed when aiming.

To do this I created 2 radial gradient nodes and changed the radius of them to be the thickness of the line multiplied by the scale both user changeable values. I the subtracted these nodes and acquired a circle. I then set the material to the user interface in the setting so that it could be applied to the screen and made it translucent.

I then made a HUD widget and added an image to the middle of the screen and set it to the material. To add this to the viewport when in game I went to the player blueprint and on the begin play event I attached a create HUD widget node and set the HUD widget created to it then used the add to viewport node.

To make it so you can change the size of the crosshair I first added the mouse wheel as an input. When the mouse wheel axis is moved it gets the value and adds the bullet angle onto it. The float made is then clamped so that the player can only change the crosshair size within a set range. This new value is then set as the bullet angle and used to calculate the bullet lines, but it is also used in the HUD widget to calculate the crosshair size. In the HUD widget the scale of the circle is set by getting the minimum screen dimensions divided with the radius of the circle. This value is then multiplied by the tangent of the bullet angle to get the new size of the circle.

Player Leaning

To make the player lean mechanic I first imported some premade animations from Ref 2*. These will be used to simulate the leaning effect based on the user's input. I then created a blend space named LeanAimOffset for the animations to be used and applied into the first-person player skeleton. I adjusted the scale of the lean to only fall in between -1 to 1, this is the lean alpha. If the lean alpha is equal to -1 the player will be leaning to the left, at 0 the player will be standing upright in its idle phase and at 1 the player will be leaning right. In the first-person players animation blueprint I inserted the blend space created and made a new variable to set the lean alpha. Within the first-person character blueprint, I created a getter function to retrieve the new lean alpha variable created.

For the inputs I created two new input actions for key bindings 'Q' and 'E' when pressed these set bools based off whether the key is pressed(true) or unpressed(false).

For the lean calculations I made a function called handle tick which is ran every game update and takes in the game's delta time. Within this function I have it so if the lean right bool is true it

calculates the lean angle based off the lean speed times delta time, I did the same for the left lean bool with just the small change of multiplying the lean speed by -1 to reverse the direction of the animation. If the bools are false meaning nothing is pressed the lean alpha is reset to 0 based off the lean angle reversed back to the start by using the same calculation used to lean.

Along side the lean calculations I also rotate the camera. For this I created a function called offset camera in this function I first have to detach the camera from the players body so only the camera moves. I then created a select node based off the lean alpha, if the alpha value is less than 0 the player is leaning left, and it runs the false code. Else the player is leaning to the right. If true(left) the cameras lean offset which is a float is taken away from the initial cameras lean offset float, if false it adds the offsets together. Whether true or false I need to interpolate between the current cameras rotation and the rotation calculated so that the gameplay isn't jarring. To do this I used a lerp node which takes in the initial camera rotation, the rotation calculated and the absolute value of the lean alpha, this value is then set as the cameras y position.

I then reattach the cameras children including the players body back onto the camera using the attach to component node.

Development Process:

For my project I first made every mechanic using the unreal blueprint functionality. This process was very smooth however I did miss the ease of being able to do simple calculations and set variables in a couple of lines rather than using multiple nodes and connecting them all.

I found converting my project from blueprints to C++ very difficult due to the lack of documentation on the unreal website for corresponding nodes in the C++ format. The change from easily creating a variable in blueprints to having to set every flag for a variable in C++ is a lot. I didn't expect the rise in difficulty from recreating blueprints in C++.

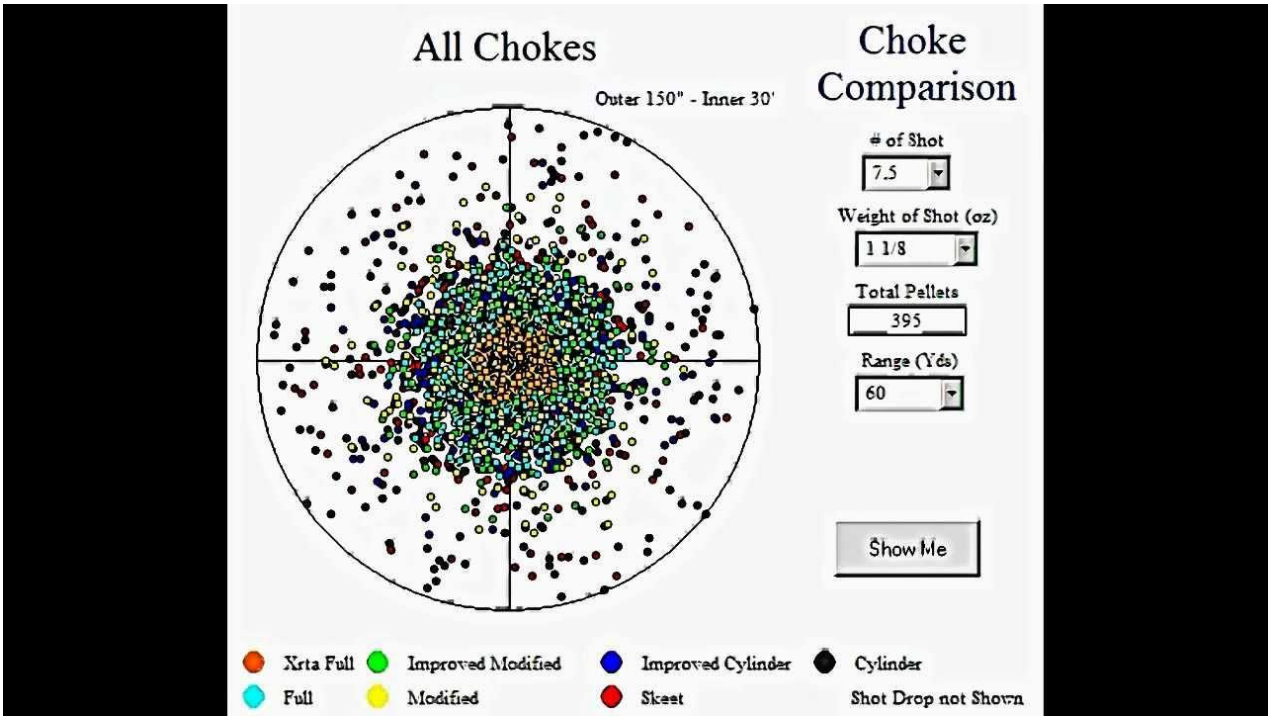
The bullet penetration mechanic functions solely from C++, the shotgun bullet spread mechanic is mainly in code apart from some of the crosshair functionality. For the leaning mechanic I decided to leave this in blueprints as it uses a lot of animations and visual representation which seems best suited for blueprints, rather than importing all of the animations into C++

Conclusion:

In conclusion I am very pleased with the functionality of my three mechanics and believe they would make great additions to any first-person shooter title.

The shotgun spread behaves like it does in real life by implementing the curve so the bullets cluster but also allowing for stray shots improves player immersion. I believe I have improved upon the average game shotgun like you would see in Counter-Strike: Global Offensive. To show this I have

set the number of bullets in my round to 8 the same as in the CSGO game. From the images my result matches that of a real-life shotgun fire more due to more bullets being focused around the centre rather than being scattered across the target. I also think that I have made a better crosshair for player information which can help with their aim as then can easily tell where the bullets will land.



The bullet penetration mechanic works perfectly and allows the designer to easily add new materials for bullet penetration. In most FPS games with similar collision mechanics only one or two objects can be shot through limiting the player experience and realism. However, if my mechanic was to be fully implemented within a game it would allow the designer to have every bit of map architecture capable of using this feature, which would add a lot of gameplay functionality which I don't believe has been available before.

The leaning mechanic was a very simple addition but worthwhile as I believe it adds to the player experience and would be well received if featured in an FPS game.

If I was to improve on anything it would be to add the bullet penetration feature onto the shotgun however I was unable to achieve this due to time restraints, however if I was to use these mechanics for a future project, I would revisit the mechanic addition and achieve good results. As when testing to do this I came close nevertheless chose to remove it as it wasn't perfectly working, and I'd rather show off code with no gameplay bugs. Again, if I had more time, I would've changed the testing area map and added shootable enemies but despite that I believe I am able to show off my mechanics perfectly well with the testing area created.

References:

How to reference from character to HUD:

www.youtube.com. (n.d.). *Unreal Engine - Calling variable values from character to a HUD*. [online] Available at: https://www.youtube.com/watch?v=-QNqOtG2vjk&t=2s&ab_channel=PRDVEntertainment [Accessed 20 Jan. 2022].

Player leaning animations:

Dropbox. (n.d.). *Animations.zip*. [online] Available at: https://www.dropbox.com/s/7ilcnukyy6v1a58/Animations.zip?dl=0 [Accessed 20 Jan. 2022].

C++ convert help:

GameDev.tv. (2020). *Resolved - Error: Install a version of .NET Framework SDK at 4.6.0 or higher*. [online] Available at: https://community.gamedev.tv/t/resolved-error-install-a-version-of-net-framework-sdk-at-4-6-0-or-higher/144998 [Accessed 20 Jan. 2022].

How to set a vector in C++:

answers.unrealengine.com. (n.d.). *How to set a vector? [C++] [SIMPLE] - UE4 AnswerHub*. [online] Available at: https://answers.unrealengine.com/questions/74246/how-to-set-a-vector-c-simple.html [Accessed 20 Jan. 2022].

How to use curve in C++:

answers.unrealengine.com. (n.d.). *Get Float Value from Curve in C++ - UE4 AnswerHub*. [online] Available at: https://answers.unrealengine.com/questions/87120/get-float-value-from-curve-in-c.html [Accessed 20 Jan. 2022].

Multiplying vectors in C++:

docs.unrealengine.com. (n.d.). *FVector::operator**. [online] Available at: https://docs.unrealengine.com/4.26/en-US/API/Runtime/Core/Math/FVector/op_mul/2/ [Accessed 20 Jan. 2022].

docs.unrealengine.com. (n.d.). *FVector*. [online] Available at: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Core/Math/FVector/ [Accessed 20 Jan. 2022].

Counterstrike image:

www.youtube.com. (n.d.). *CS:GO Shotguns Tutorial*. [online] Available at: https://www.youtube.com/watch?v=Fefm-ro61Rk&ab_channel=TheWarOwl [Accessed 20 Jan. 2022].

shotgun image:

www.youtube.com. (n.d.). *Shotgun Pattern Diameter Comparison: 30-60 Yards*. [online] Available at: https://www.youtube.com/watch?v=21fasmPUjIw&ab_channel=RandyWakeman [Accessed 20 Jan. 2022].

Hud to C++:

Redd.it. (2022). [online] Available at: https://preview.redd.it/8g0t8bbh8wk61.png?width=916&format=png&auto=webp&s=49e44043644cfc42c50a637cdaf764e2b35006eb [Accessed 20 Jan. 2022].

Play animation montage C++:

www.youtube.com. (n.d.). *Unreal Engine C++ #4 - Playing Animation Montages*. [online] Available at: https://www.youtube.com/watch?v=ftjflBJwFAU&ab_channel=CodeLikeMe. [Accessed 20 Jan. 2022].

Play Sound C++:

www.youtube.com. (n.d.). *Unreal Tutorial C++ - Series - Player Character - The Punch - Part 5 - Play Sounds*. [online] Available at: https://www.youtube.com/watch?v=LT9xAPOp03Y&t=90s&ab_channel=JollyMonsterStudio [Accessed 20 Jan. 2022].

Spawn Decal C++:

answers.unrealengine.com. (n.d.). *How to spawn a decal in C++? - UE4 AnswerHub*. [online] Available at: https://answers.unrealengine.com/questions/768669/how-to-spawn-a-decal-in-c.html [Accessed 20 Jan. 2022].