Module 5 Portfolio
Breaking Bank

Final Major Project

16th August 2020

Amended for display purposes

Changes include removal of attribution table, file directory and appendix.

Links.

Game Build.

Build Video.

Stephen Warner

BIRMINGHAM CITY UNIVERSITY

Contents

Breaking Bank – The Project.

The Team.

Team Payday for project Breaking Bank consisted of three programmers: Pratik, Stanley and myself, one designer: Barney, and one producer: Christina, for a total of five members. The programming team had six main aspects to create, which include: Gameplay, AI, UI, SFX, VFX and networking. One of us focused heavily on AI and some gameplay elements, I focused on gameplay elements like the guns, interactables and the health system, and pitching in wherever needed with UI, SFX and AI. Our final member investigated network code and creating our user interface, researching SFX and implementing the majority of the game's SFX. The designer was the concept holder, they were the person who originally decided to start the project, as such they handled sourcing the art and animation assets as well as tweaking/building the levels for the team. Due to some technical issues in the first few weeks of the project we were using GitHub to host our project so that we could all collaborate together, later the programmers on the team were granted access to the Gamer Camp Perforce server meaning we could work together much easier however our designer was unable to get access to the server as well meaning that we had a challenge to overcome, which was how to get content from our designer and how to send the updates back to them. Our solution to this was to use GitHub again, however this time we used GitHub simply as a go between for transferring and updating the assets, keeping the Perforce version as the "master" version and relying on the ease of use of Perforce's version history and rollback features to ensure the build was almost always working as intended with some bugs/issues occasionally.

X Statement.

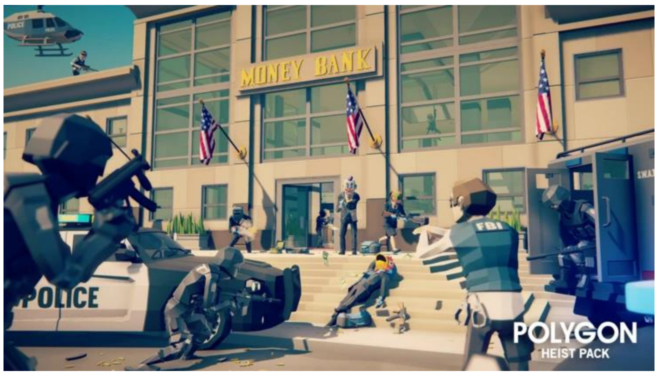
The banks are loaded, and you are not.
It is time to get the crew together, load up and spread the wealth.

Razor Statement.

Payday 2 – Setting.

Left 4 Dead 2 – Arcade Shooter.

Polygon Heist Pack – Art Style.



Elevator Statement.

Fulfil your heisting dreams in this First-Person Shooter. Play your way, brute force the defences or plan out your attack and wait for your time to strike. Want to claim your bragging rights? Go for speed and find the quickest way through the level to find out who is the fastest heister. Two things are certain: there will be a fight, and someone is getting paid.

Minimum Viable Product:

- One Level.
- Interactive AI:
  - Hostile Wave.
  - Patrolling.
- Weapons
  - 1 Gun (Rifle or Pistol).
- Interactables:
  - Drill.
  - Key cards.
  - Money Pickup.
- UI/HUD
- Detection system.
- 1 playable character with animations.

What We Actually Achieved:

- Two Levels:
  - Tutorial.
  - Main Level.
- Interactive AI:
  - Hostile Wave.
  - Patrolling.
- Weapons:
  - 1 Rifle.
  - 1 Pistol.
- Interactables:
  - Drill.
  - Money Pickup.
  - Key cards.
  - Locked doors.
  - Vault door.
  - Extraction Point.
- UI/HUD:
  - Usable front-end UI.
  - Usable In-Game UI/HUD
- Detection system.
- 1 playable character with some animations.
- SFX.
- VFX.

Project Start:

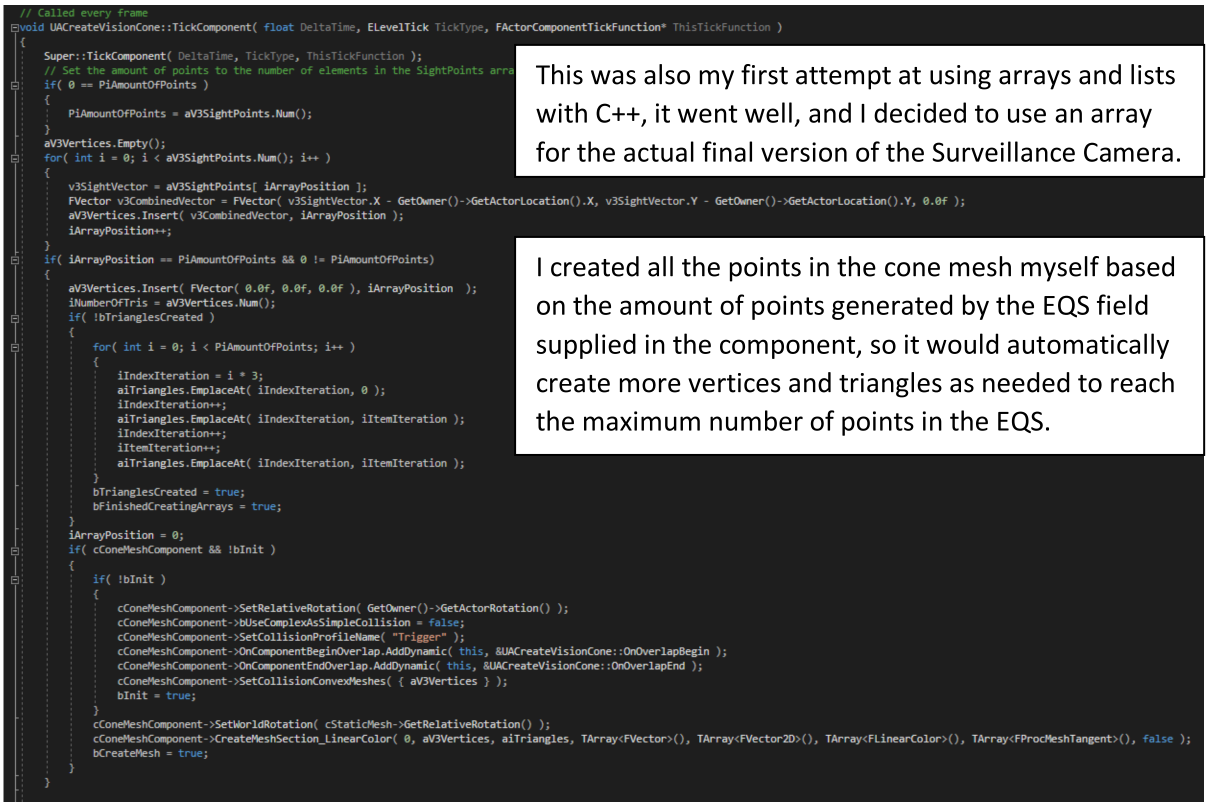
With this project, we were very ambitious at the start, planning to incorporate local area network cooperation play (LAN CO-OP), we tried to keep this as our main goal for the majority of the project life span, but after about the six week we began to realise that we needed a game before we could network it for LAN CO-OP, as testing features in the midst of attempting to incorporate networked code made error testing and also feature planning tricky. One thing we did observe as a good rule is that any time you set a value in the you must ask the server to set the value then have the server tell the client what the value should be. There will be examples of this later in this document as the majority of network code is still fully functioning, but we limited the number of players to one using the front-end UI starting the game.

Code Annotations.

A note about coding conventions, as a team we did not decide on coding conventions together, so everyone used their own, I chose to use the same conventions as we had declared in modules 3 and 4.

Early Prototyping for the Surveillance Camera.

In the very first week of module 5 I began looking into how to dynamically create a mesh which can adapt to the environment whilst also triggering collision events with objects such as the player character. To this end I looked into some experimental features of Unreal Engine, and found that it was possible to do this, however it has some issues for now, one of those issues is how the engine handles rotating of the EQS system points around a vector. The engine will attempt to keep the points horizontal regardless of the angle they are being rotated by, meaning if you rotate 90 degrees in the Z axis (Y for Unity users) so the points face up, then the points will merge into a single point. This meant the system could not be used without creating a custom rotation matrix just to handle the rotation of the points in the EQS cone. However, it works rather well if you keep the EQS field two dimensional, so it would work fine for a top down shooter/ stealth game.



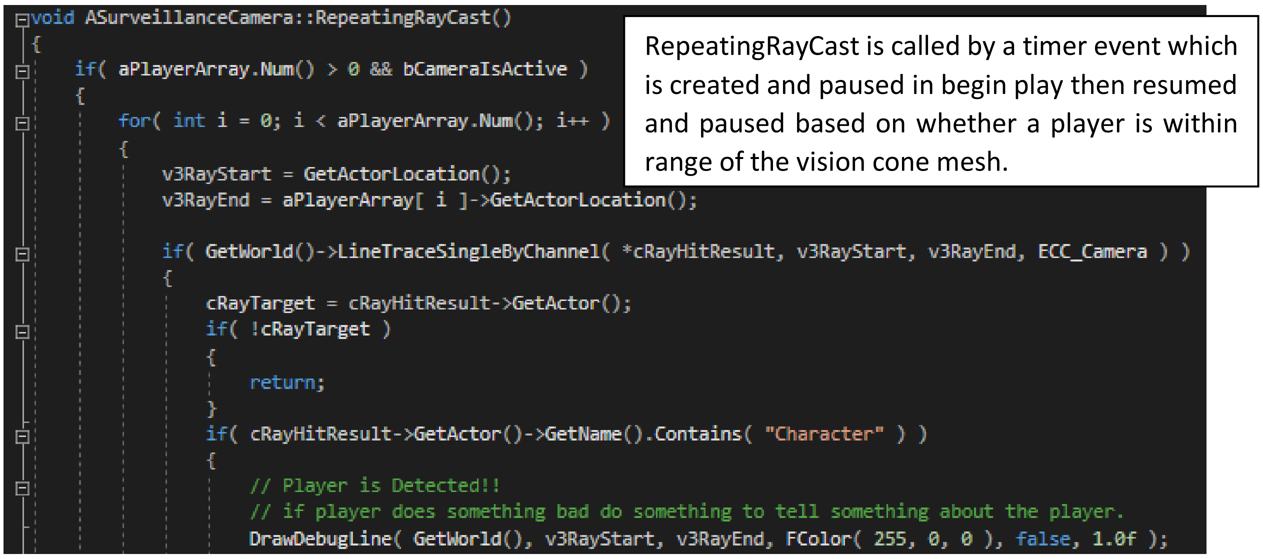
```cpp
// Called every frame
void UACreateVisionCone::TickComponent( float DeltaTime, ELevelTick TickType, FActorComponentTickFunction* ThisTickFunction )
{
    Super::TickComponent( DeltaTime, TickType, ThisTickFunction );
    // Set the amount of points to the number of elements in the SightPoints arra
    if( 0 == PiAmountOfPoints )
    {
        PiAmountOfPoints = aV3SightPoints.Num();
    }
    aV3Vertices.Empty();
    for( int i = 0; i < aV3SightPoints.Num(); i++ )
    {
        v3SightVector = aV3SightPoints[ iArrayPosition ];
        FVector v3CombinedVector = FVector( v3SightVector.X - GetOwner()->GetActorLocation().X, v3SightVector.Y - GetOwner()->GetActorLocation().Y, 0.0f );
        aV3Vertices.Insert( v3CombinedVector, iArrayPosition );
        iArrayPosition++;
    }
    if( iArrayPosition == PiAmountOfPoints && 0 != PiAmountOfPoints)
    {
        aV3Vertices.Insert( FVector( 0.0f, 0.0f, 0.0f ), iArrayPosition  );
        iNumberOfTris = aV3Vertices.Num();
        if( !bTrianglesCreated )
        {
            for( int i = 0; i < PiAmountOfPoints; i++ )
            {
                iIndexIteration = i * 3;
                aiTriangles.EmplaceAt( iIndexIteration, 0 );
                iIndexIteration++;
                aiTriangles.EmplaceAt( iIndexIteration, iItemIteration );
                iIndexIteration++;
                iItemIteration++;
                aiTriangles.EmplaceAt( iIndexIteration, iItemIteration );
            }
            bTrianglesCreated = true;
            bFinishedCreatingArrays = true;
        }
        iArrayPosition = 0;
        if( cConeMeshComponent && !bInit )
        {
            if( !bInit )
            {
                cConeMeshComponent->SetRelativeRotation( GetOwner()->GetActorRotation() );
                cConeMeshComponent->bUseComplexAsSimpleCollision = false;
                cConeMeshComponent->SetCollisionProfileName( "Trigger" );
                cConeMeshComponent->OnComponentBeginOverlap.AddDynamic( this, &UACreateVisionCone::OnOverlapBegin );
                cConeMeshComponent->OnComponentEndOverlap.AddDynamic( this, &UACreateVisionCone::OnOverlapEnd );
                cConeMeshComponent->SetCollisionConvexMeshes( { aV3Vertices } );
                bInit = true;
            }
            cConeMeshComponent->SetWorldRotation( cStaticMesh->GetRelativeRotation() );
            cConeMeshComponent->CreateMeshSection_LinearColor( 0, aV3Vertices, aiTriangles, TArray<FVector>(), TArray<FVector2D>(), TArray<FLinearColor>(), TArray<FProcMeshTangent>(), false );
            bCreateMesh = true;
        }
    }
```

This was also my first attempt at using arrays and lists with C++, it went well, and I decided to use an array for the actual final version of the Surveillance Camera.

I created all the points in the cone mesh myself based on the amount of points generated by the EQS field supplied in the component, so it would automatically create more vertices and triangles as needed to reach the maximum number of points in the EQS.

Surveillance Camera.

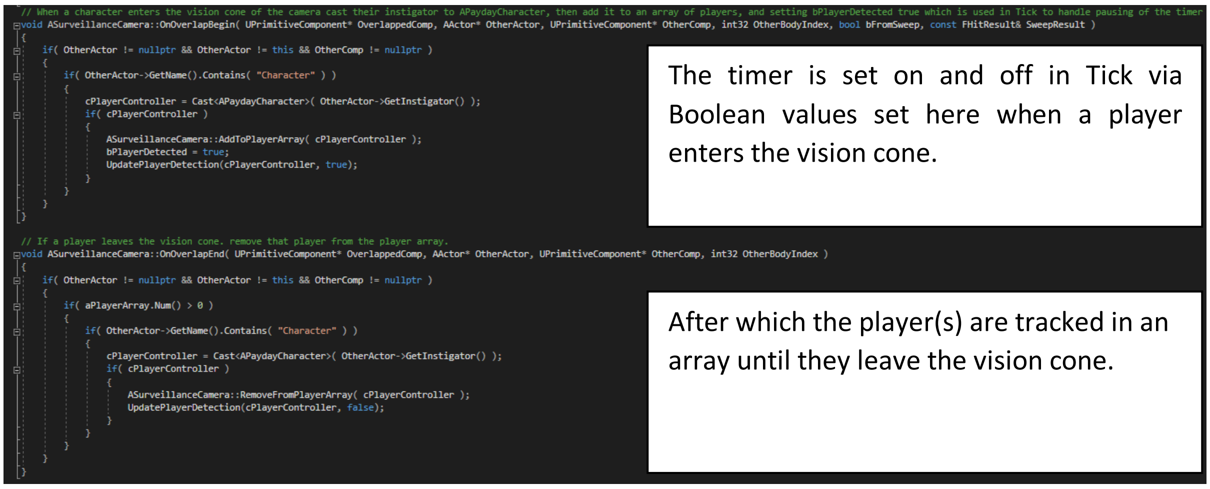
Since I realised that the EQS was not suited for three-dimensional space rotations out of the box I decided to quickly move onto creating something that could. As a result I created an object which had two static meshes, one for the vision cone and one for the prop itself, this meant that the vision cone could be altered separately from the camera to fit the area it is placed inside of. Though this had to be done manually and was not as simple to do as I would have preferred as the easiest way to alter the vision cone size is to do so before you apply any rotations to the root component/ root object, otherwise the scaling of the vision cone becomes challenging, though once scaled to requirement rotation afterwards is completely fine. I also created the surveillance camera class to serve any needs the other members of the team needed, as such I kept the functionality bare to enable easy implementation of further features later.



```cpp
void ASurveillanceCamera::RepeatingRayCast()
{
    if( aPlayerArray.Num() > 0 && bCameraIsActive )
    {
        for( int i = 0; i < aPlayerArray.Num(); i++ )
        {
            v3RayStart = GetActorLocation();
            v3RayEnd = aPlayerArray[ i ]->GetActorLocation();

            if( GetWorld()->LineTraceSingleByChannel( *cRayHitResult, v3RayStart, v3RayEnd, ECC_Camera ) )
            {
                cRayTarget = cRayHitResult->GetActor();
                if( !cRayTarget )
                {
                    return;
                }
                if( cRayHitResult->GetActor()->GetName().Contains( "Character" ) )
                {
                    // Player is Detected!!
                    // if player does something bad do something to tell something about the player.
                    DrawDebugLine( GetWorld(), v3RayStart, v3RayEnd, FColor( 255, 0, 0 ), false, 1.0f );
```

RepeatingRayCast is called by a timer event which is created and paused in begin play then resumed and paused based on whether a player is within range of the vision cone mesh.

Only if the hit actor was a player would be do any logic as we needed the camera to be aware of blocking actors like walls, floors and doors etc, so that the player wasn't being tracked through solid objects, meaning that any logic done to the player or checking the player would be done in the instance of a successful ray cast only. The entire system was designed to be easy to understand and alter when required by the other team members without me having to assist them.



```cpp
// When a character enters the vision cone of the camera cast their instigator to APaydayCharacter, then add it to an array of players, and setting bPlayerDetected true which is used in Tick to handle pausing of the timer
void ASurveillanceCamera::OnOverlapBegin( UPrimitiveComponent* OverlappedComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool bFromSweep, const FHitResult& SweepResult )
{
    if( OtherActor != nullptr && OtherActor != this && OtherComp != nullptr )
    {
        if( OtherActor->GetName().Contains( "Character" ) )
        {
            cPlayerController = Cast<APaydayCharacter>( OtherActor->GetInstigator() );
            if( cPlayerController )
            {
                ASurveillanceCamera::AddToPlayerArray( cPlayerController );
                bPlayerDetected = true;
                UpdatePlayerDetection(cPlayerController, true);
            }
        }
    }
}

// If a player leaves the vision cone. remove that player from the player array.
void ASurveillanceCamera::OnOverlapEnd( UPrimitiveComponent* OverlappedComp, AActor* OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex )
{
    if( OtherActor != nullptr && OtherActor != this && OtherComp != nullptr )
    {
        if( aPlayerArray.Num() > 0 )
        {
            if( OtherActor->GetName().Contains( "Character" ) )
            {
                cPlayerController = Cast<APaydayCharacter>( OtherActor->GetInstigator() );
                if( cPlayerController )
                {
                    ASurveillanceCamera::RemoveFromPlayerArray( cPlayerController );
                    UpdatePlayerDetection(cPlayerController, false);
                }
            }
        }
    }
}
```

The timer is set on and off in Tick via Boolean values set here when a player enters the vision cone.

After which the player(s) are tracked in an array until they leave the vision cone.

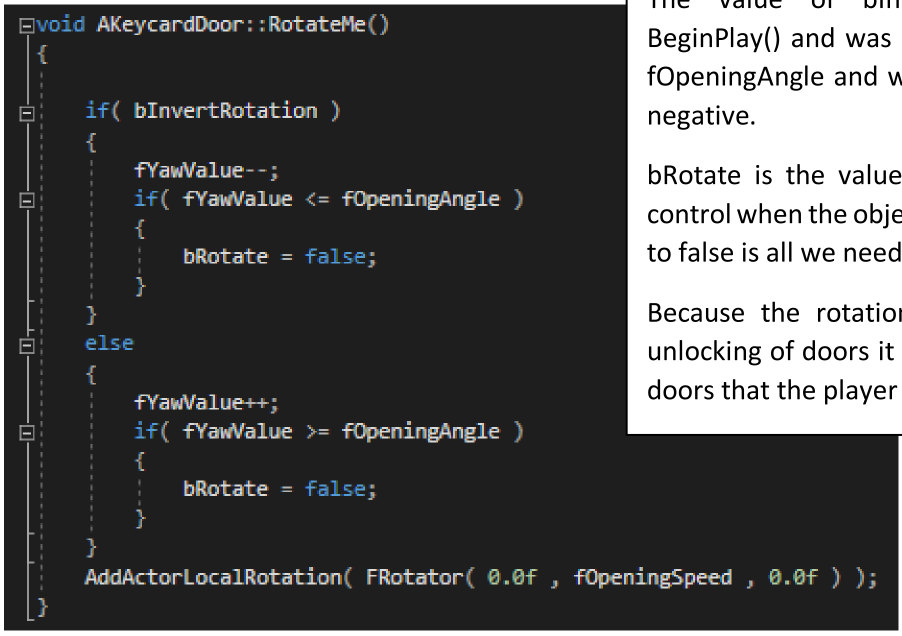
Using an array to track the number of players in the cone of vision meant that each player in range could be a ray cast target, it also meant that the class was created with multiplayer in mind. This concludes the first aspect that I contributed to the project next will be the interactable key card and locked door.

Key card & Key card Doors.

These were very simple asset for creation though whilst we were working with multiplayer in mind there was some significant issues surrounding the rotation of non-symmetrical collision boxes, for example, a door which has a collision that is slim and rectangular. Whilst working in a network code environment I ran into an issue causing the static mesh (visible object) and the collision box becoming desynchronised when rotating the root object, meaning that if you “opened” the door, the collision box might still be in the “closed” state. This is also true for constant rotation values which are set mid play, if you start rotating an object with server simulation active and then pause the rotation the collision box will continue to rotate without its static mesh rotation enabled. Locking the rotation for the object behind a Boolean value meaning true or false are the only possibilities makes this very easy to observe. Because of the network simulation there would actually be two Boolean values, one for client and one for server, if either the client or the server has a different value for that bool then the desync will occur resulting in what would at first glance appear to be something out of quantum mechanics, bool that would be true and false at the same time in the call stack for the variable. This is however not the case as we are just simulating a server and client simultaneously, which means it would just be a desync in the data for the game. This issue caused me quite a lot of headaches and took a long time to troubleshoot as it wasn’t obvious what was going on inside the engine, it didn’t help that I had also forgotten that server simulation was active in my Unreal Editor. This unexpected behaviour might be avoidable using the network properties for client only and server only functions, though I did not get to test this theory as by the time I had realised what was occurring and what caused it we had already decided that networked multiplayer was not going to be possible given our time frame left in the project at six weeks in.

Locking doors with a key card was super simple using the GameState class which is part of the base of Unreal Engine, this class allows you to store and access game specific data easily in the world and this is how I kept track of what key cards the player had collected and as a result what doors the player could open. I used EKeycardTypes to specify what type of key card was required by which door in editor, and the key cards themselves had the same values available.

So with networking out of the project I quickly prototyped a rotating door and implemented the class in a way in which the value of the rotation can be positive or negative and the class would work out which direction to rotate based on a positive or negative float value. This value and an opening speed modifier were both made available to design alteration via the editor details panel or blueprint which ever was desired, meaning I was not required to make any code changes for design to get doors to rotate in whatever direction was desired.



```
void AKeycardDoor::RotateMe()
{

    if( bInvertRotation )
    {
        fYawValue--;
        if( fYawValue <= fOpeningAngle )
        {
            bRotate = false;
        }
    }
    else
    {
        fYawValue++;
        if( fYawValue >= fOpeningAngle )
        {
            bRotate = false;
        }
    }
    AddActorLocalRotation( FRotator( 0.0f , fOpeningSpeed , 0.0f ) );
}
```

The value of bInvertRotation was set in BeginPlay() and was determined by the value of fOpeningAngle and whether that was positive or negative.

bRotate is the value which is used in Tick() to control when the object can rotate, as such setting to false is all we need to do to stop rotation.

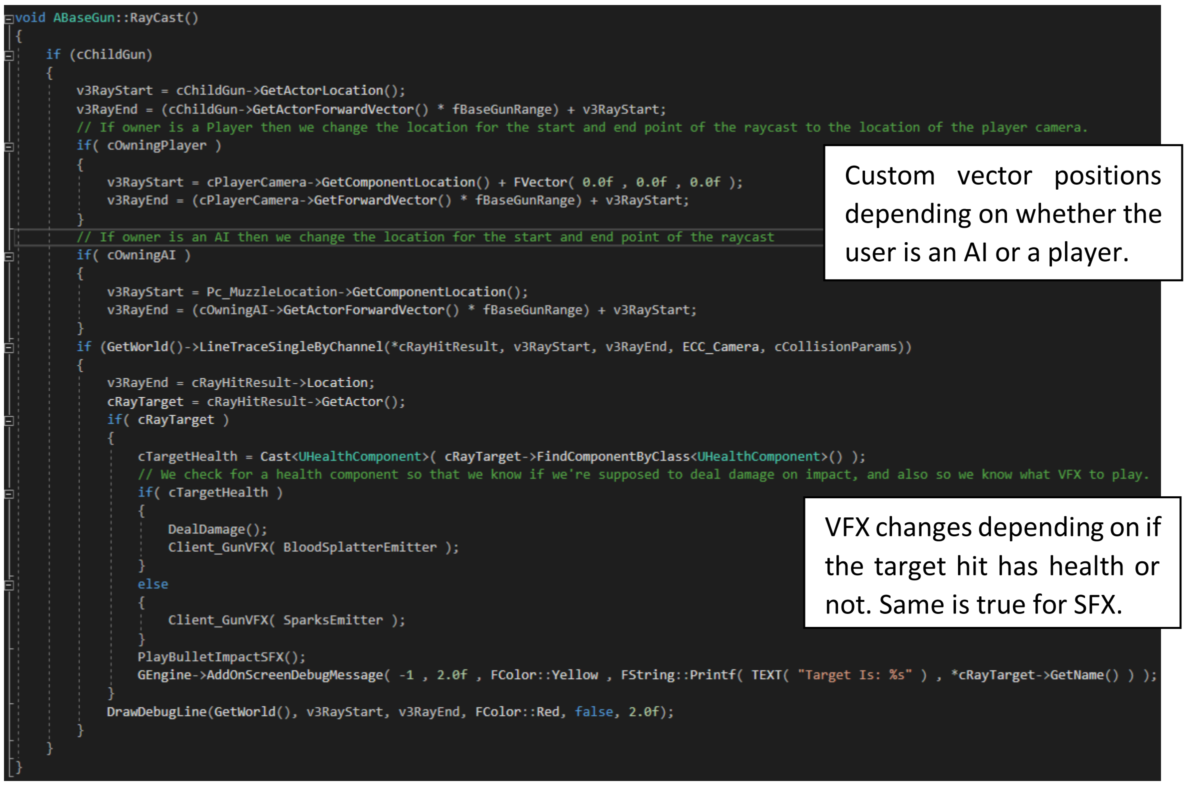
Because the rotation was independent of the unlocking of doors it was possible to have closed doors that the player could open themselves.

I like this class because of how simple it was to create and how much of an impact it had visually, allowing the player to physically open a door and then walk through it, as well as allowing the AI to walk through the now opened door. In general, keeping the code simple was quite a challenge when working around networking and pre-existing networked code.

The key cards themselves were simple, if player interacts with a key card, add that key card type to the GameState if it does not already exist and turn the key card invisible else just turn invisible. This means the players would only ever have one of each key card type at a time, it would also be possible to make a small adjustment to allow for multiple of the same type of key cards in the level and also stored in the GameState as they are stored in an array.

<u>Guns, Pistol & Rifle.</u>

The base gun class for project Breaking Bank is quite messy due to a mix between catering for AI use and working around network coding, it is however fully functional and very adjustable in blueprints / editor. As a note I now realise how I can create pre constructed blueprints inside of .cpp class files using spawn actor, I did not realise that at the time of creating the base gun class.



```cpp
void ABaseGun::RayCast()
{
    if (cChildGun)
    {
        v3RayStart = cChildGun->GetActorLocation();
        v3RayEnd = (cChildGun->GetActorForwardVector() * fBaseGunRange) + v3RayStart;
        // If owner is a Player then we change the location for the start and end point of the raycast to the location of the player camera.
        if( cOwningPlayer )
        {
            v3RayStart = cPlayerCamera->GetComponentLocation() + FVector( 0.0f , 0.0f , 0.0f );
            v3RayEnd = (cPlayerCamera->GetForwardVector() * fBaseGunRange) + v3RayStart;
        }
        // If owner is an AI then we change the location for the start and end point of the raycast
        if( cOwningAI )
        {
            v3RayStart = Pc_MuzzleLocation->GetComponentLocation();
            v3RayEnd = (cOwningAI->GetActorForwardVector() * fBaseGunRange) + v3RayStart;
        }
        if (GetWorld()->LineTraceSingleByChannel(*cRayHitResult, v3RayStart, v3RayEnd, ECC_Camera, cCollisionParams))
        {
            v3RayEnd = cRayHitResult->Location;
            cRayTarget = cRayHitResult->GetActor();
            if( cRayTarget )
            {
                cTargetHealth = Cast<UHealthComponent>( cRayTarget->FindComponentByClass<UHealthComponent>() );
                // We check for a health component so that we know if we're supposed to deal damage on impact, and also so we know what VFX to play.
                if( cTargetHealth )
                {
                    DealDamage();
                    Client_GunVFX( BloodSplatterEmitter );
                }
                else
                {
                    Client_GunVFX( SparksEmitter );
                }
                PlayBulletImpactSFX();
                GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Yellow , FString::Printf( TEXT( "Target Is: %s" ) , *cRayTarget->GetName() ) );
            }
            DrawDebugLine(GetWorld(), v3RayStart, v3RayEnd, FColor::Red, false, 2.0f);
        }
    }
}
```

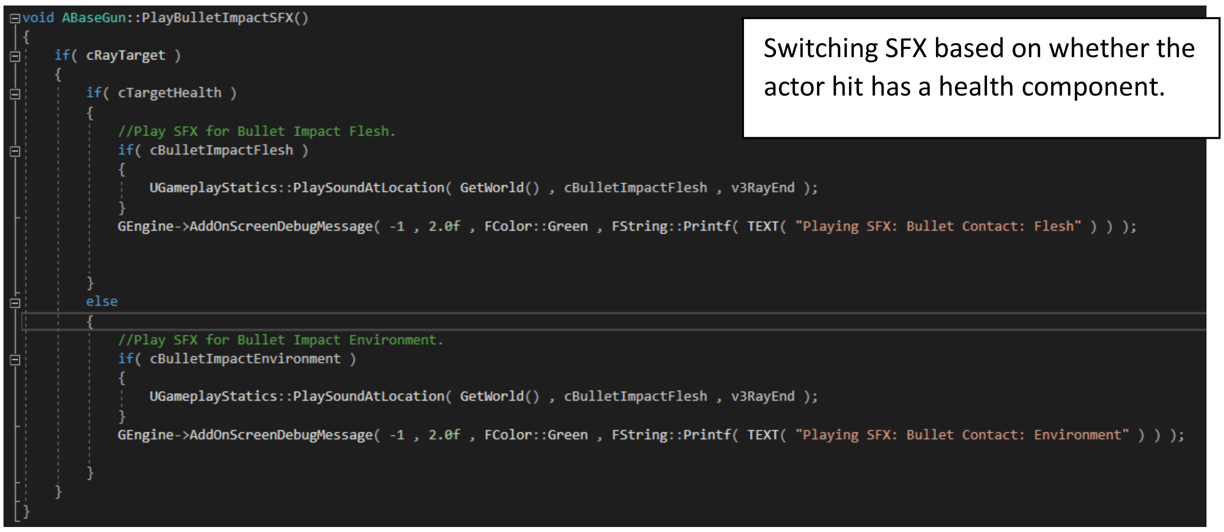
Custom vector positions depending on whether the user is an AI or a player.

VFX changes depending on if the target hit has health or not. Same is true for SFX.

Getting the guns to function properly was a long task in of itself mostly due to the lack of clarity in what was required by design, we eventually got to a stage where anything would be possible with the base gun class with minor tweaks for functionality. In its current state the base gun fires ray casts from the player camera if the user is a player and from the gun barrel locator* if the user is an AI *(Gun Barrel Locator is a separate collision box component that is configurable in editor and blueprints simply for specifying the location of the end of the gun's barrel as that would be different per gun mesh model).

Also, the base gun class uses a mix of Unreal's world timer manager and my own custom delta time clocks because during the initial creation I did not realise how to get the desired functionality out of Unreal's timer system. Though after using it with other features like the VFX and the GameTimer class I realised how to rework the base gun to use Unreal's timer system exclusively, though I am unsure as to how much of a benefit/detriment that would actually be for performance versus what I am already using in the class.

Gun SFX.



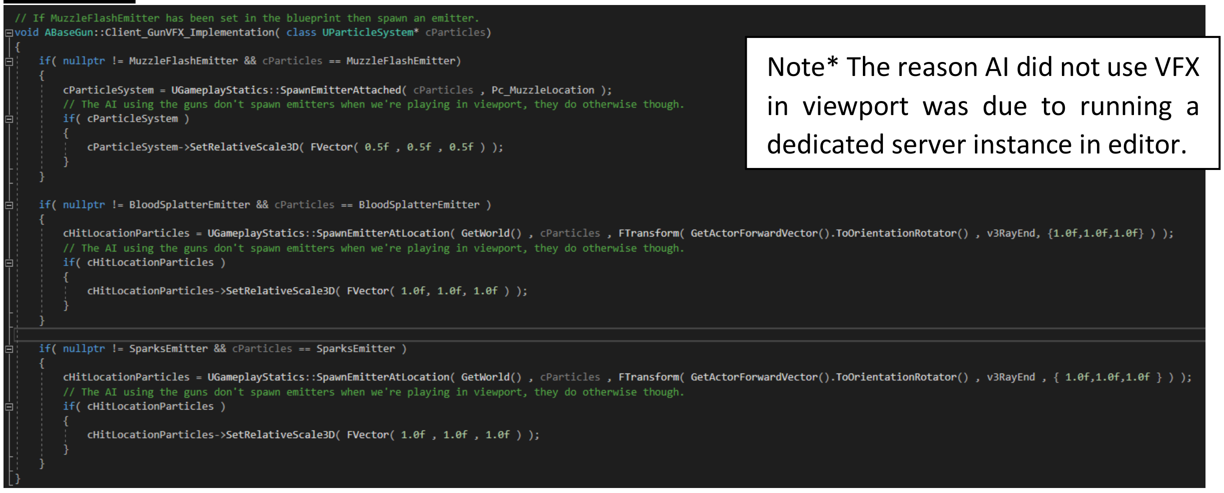
```cpp
void ABaseGun::PlayBulletImpactSFX()
{
    if( cRayTarget )
    {
        if( cTargetHealth )
        {
            //Play SFX for Bullet Impact Flesh.
            if( cBulletImpactFlesh )
            {
                UGameplayStatics::PlaySoundAtLocation( GetWorld() , cBulletImpactFlesh , v3RayEnd );
            }
            GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Green , FString::Printf( TEXT( "Playing SFX: Bullet Contact: Flesh" ) ) );


        }
        else
        {
            //Play SFX for Bullet Impact Environment.
            if( cBulletImpactEnvironment )
            {
                UGameplayStatics::PlaySoundAtLocation( GetWorld() , cBulletImpactFlesh , v3RayEnd );
            }
            GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Green , FString::Printf( TEXT( "Playing SFX: Bullet Contact: Environment" ) ) );

        }
    }
}
```

Switching SFX based on whether the actor hit has a health component.

There are actually multiple SFX functions as each does something different though it would be possible to just pass in an integer value to specify which SFX we wanted to play at that moment in time I however didn't do this as I wanted the code to be more readable as I was not intending to implement the SFX to the guns myself and the only real difference would be in compilation time of the build and it would have no impact on performance of the code.

The VFX does something similar however it is done so by "providing" a pointer to the VFX you wish to play to the function so that it can be checked for not being null and also if additional logic needs to be applied to the VFX before it is played, like changing the type of emitter that is displaying the effect for instance.

Gun VFX.



```cpp
// If MuzzleFlashEmitter has been set in the blueprint then spawn an emitter.
void ABaseGun::Client_GunVFX_Implementation( class UParticleSystem* cParticles)
{
    if( nullptr != MuzzleFlashEmitter && cParticles == MuzzleFlashEmitter)
    {
        cParticleSystem = UGameplayStatics::SpawnEmitterAttached( cParticles , Pc_MuzzleLocation );
        // The AI using the guns don't spawn emitters when we're playing in viewport, they do otherwise though.
        if( cParticleSystem )
        {
            cParticleSystem->SetRelativeScale3D( FVector( 0.5f , 0.5f , 0.5f ) );
        }
    }

    if( nullptr != BloodSplatterEmitter && cParticles == BloodSplatterEmitter )
    {
        cHitLocationParticles = UGameplayStatics::SpawnEmitterAtLocation( GetWorld() , cParticles , FTransform( GetActorForwardVector().ToOrientationRotator() , v3RayEnd, {1.0f,1.0f,1.0f} ) );
        // The AI using the guns don't spawn emitters when we're playing in viewport, they do otherwise though.
        if( cHitLocationParticles )
        {
            cHitLocationParticles->SetRelativeScale3D( FVector( 1.0f, 1.0f, 1.0f ) );
        }
    }

    if( nullptr != SparksEmitter && cParticles == SparksEmitter )
    {
        cHitLocationParticles = UGameplayStatics::SpawnEmitterAtLocation( GetWorld() , cParticles , FTransform( GetActorForwardVector().ToOrientationRotator() , v3RayEnd , { 1.0f,1.0f,1.0f } ) );
        // The AI using the guns don't spawn emitters when we're playing in viewport, they do otherwise though.
        if( cHitLocationParticles )
        {
            cHitLocationParticles->SetRelativeScale3D( FVector( 1.0f , 1.0f , 1.0f ) );
        }
    }
}
```

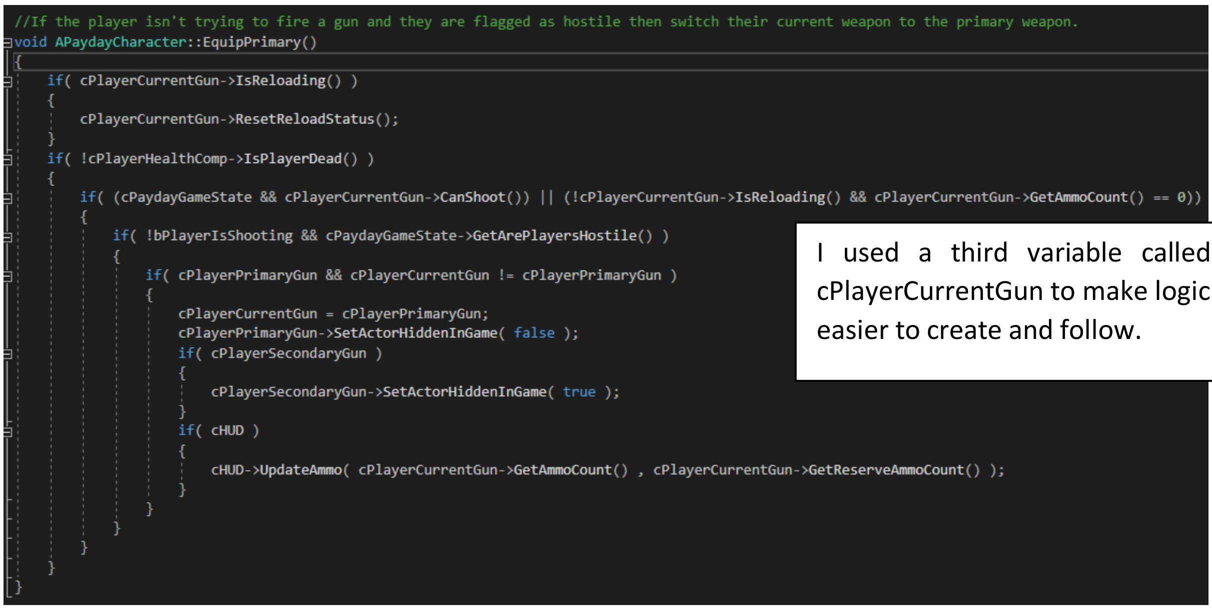
Note* The reason AI did not use VFX in viewport was due to running a dedicated server instance in editor.

Implementing the VFX this way meant that I would only have to handle logic for the VFX of the gun in one function, though this function could be coded in a nicer and more efficient way by using a switch condition on cParticles instead of if statements in theory at least as I haven't tested this. If a switch statements works as intended, then the efficiency of this function would improve as there would be no need to compare each statement to the specified variable.

*Note I forgot to adjust the name of Pc_MuzzleLocation to be cMuzzleLocation, I use Pc_VariableName because it is easier for me to read I would normally remove the underscore but I had forgotten to in this case. Also, I forgot to declare the VFX emitters as cSparksEmitter for example, that was just a mistake on my part.

The Player Character.
Equipping Weapons.

The player character was something we all edited over the course of the module, below is how I handled player weapon equipping, both for primary and secondary weapons.



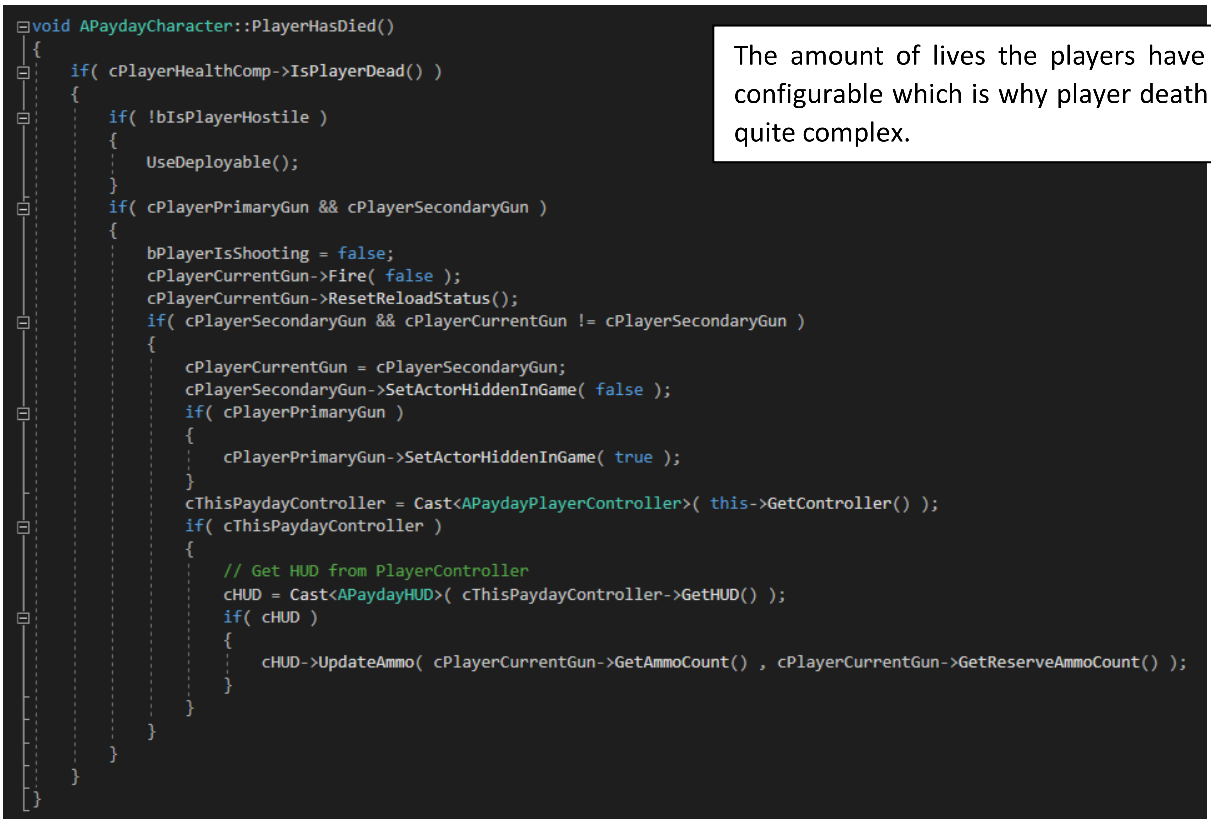
```
//If the player isn't trying to fire a gun and they are flagged as hostile then switch their current weapon to the primary weapon.
void APaydayCharacter::EquipPrimary()
{
    if( cPlayerCurrentGun->IsReloading() )
    {
        cPlayerCurrentGun->ResetReloadStatus();
    }
    if( !cPlayerHealthComp->IsPlayerDead() )
    {
        if( (cPaydayGameState && cPlayerCurrentGun->CanShoot()) || (!cPlayerCurrentGun->IsReloading() && cPlayerCurrentGun->GetAmmoCount() == 0))
        {
            if( !bPlayerIsShooting && cPaydayGameState->GetArePlayersHostile() )
            {
                if( cPlayerPrimaryGun && cPlayerCurrentGun != cPlayerPrimaryGun )
                {
                    cPlayerCurrentGun = cPlayerPrimaryGun;
                    cPlayerPrimaryGun->SetActorHiddenInGame( false );
                    if( cPlayerSecondaryGun )
                    {
                        cPlayerSecondaryGun->SetActorHiddenInGame( true );
                    }
                    if( cHUD )
                    {
                        cHUD->UpdateAmmo( cPlayerCurrentGun->GetAmmoCount() , cPlayerCurrentGun->GetReserveAmmoCount() );
                    }
                }
            }
        }
    }
}
```

I used a third variable called cPlayerCurrentGun to make logic easier to create and follow.

The logic for swapping weapons is very simple, if we are reloading our current gun, stop reloading then if we are not currently using the desired gun then swap to that gun and set the previous one hidden in game and update the HUD to display the correct ammo in the magazine and reserve ammunition pool.

I will cover player death next.

Player Death



```
void APaydayCharacter::PlayerHasDied()
{
    if( cPlayerHealthComp->IsPlayerDead() )
    {
        if( !bIsPlayerHostile )
        {
            UseDeployable();
        }
        if( cPlayerPrimaryGun && cPlayerSecondaryGun )
        {
            bPlayerIsShooting = false;
            cPlayerCurrentGun->Fire( false );
            cPlayerCurrentGun->ResetReloadStatus();
            if( cPlayerSecondaryGun && cPlayerCurrentGun != cPlayerSecondaryGun )
            {
                cPlayerCurrentGun = cPlayerSecondaryGun;
                cPlayerSecondaryGun->SetActorHiddenInGame( false );
                if( cPlayerPrimaryGun )
                {
                    cPlayerPrimaryGun->SetActorHiddenInGame( true );
                }
                cThisPaydayController = Cast<APaydayPlayerController>( this->GetController() );
                if( cThisPaydayController )
                {
                    // Get HUD from PlayerController
                    cHUD = Cast<APaydayHUD>( cThisPaydayController->GetHUD() );
                    if( cHUD )
                    {
                        cHUD->UpdateAmmo( cPlayerCurrentGun->GetAmmoCount() , cPlayerCurrentGun->GetReserveAmmoCount() );
                    }
                }
            }
        }
    }
}
```

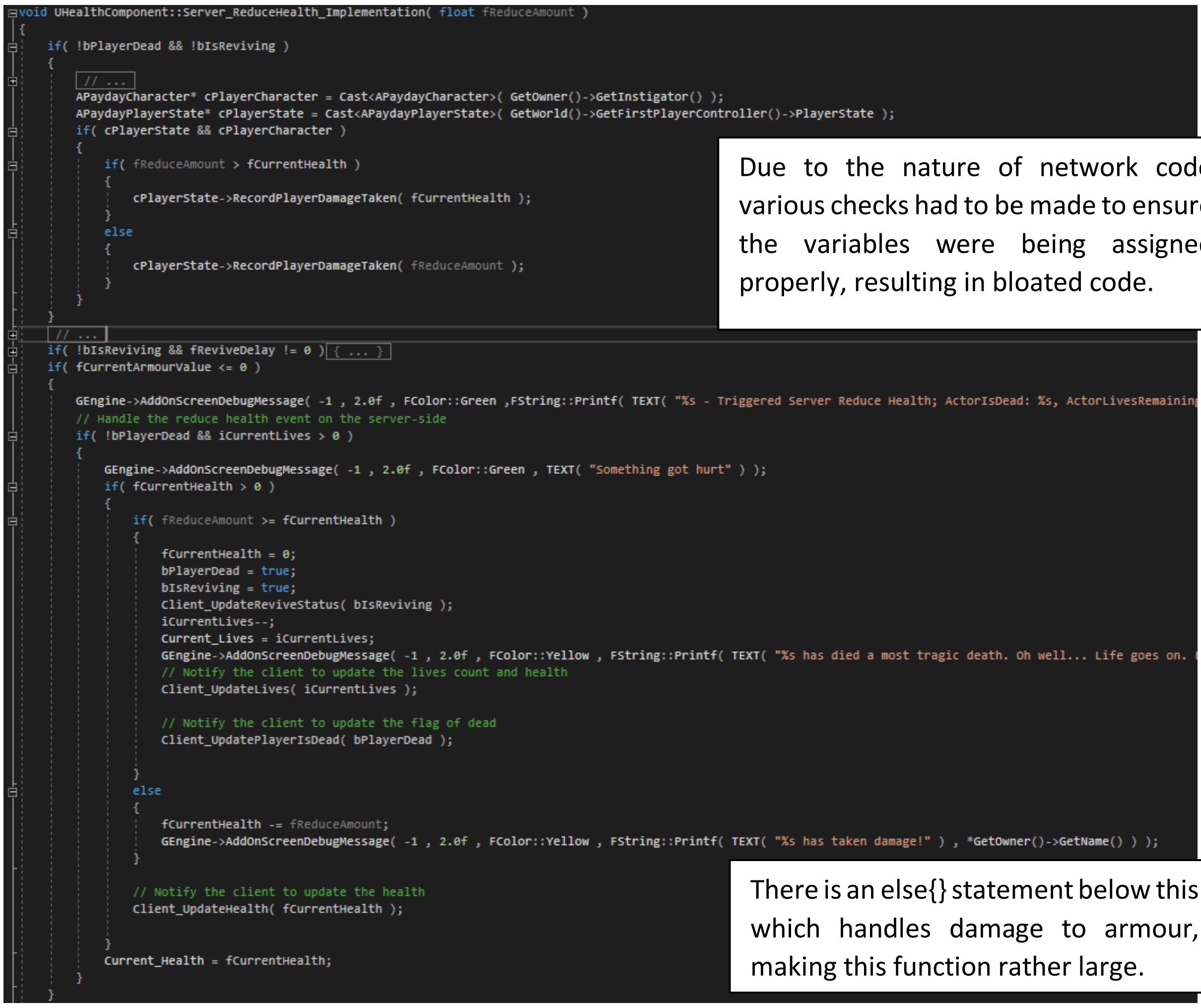
The amount of lives the players have is configurable which is why player death is quite complex.

Initially it was decided that the player would enter a downed state X number of times before they actually lost the game, this was back when we were planning for local co-op, later we decided to set the player lives to 1 meaning this function is mostly for naught, though if the player were to have multiple lives when they get “downed” they would be force equipped the pistol with whatever ammo they left in the magazine before getting downed and then have to wait until they revived before switching back to their primary weapon, whilst they are in this state the player is completely invulnerable but is free to shoot back at the AI and look around as they wish, but they are unable to move or jump.

Next up will be the health component, used by players and AI alike.

The Health Component

This is a ridiculously complex component as there is a lot of networked code involved which is redundant now because the game is not designed for multiplayer anymore, however I was asked to not remove the networking if it did not interfere with further development as such I left it in, but as a result the code became more bloated due to having to call multiple functions for client and server side handling of features.



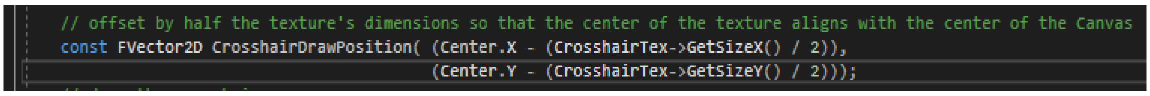
```cpp
void UHealthComponent::Server_ReduceHealth_Implementation( float fReduceAmount )
{
    if( !bPlayerDead && !bIsReviving )
    {
        // ...
        APaydayCharacter* cPlayerCharacter = Cast<APaydayCharacter>( GetOwner()->GetInstigator() );
        APaydayPlayerState* cPlayerState = Cast<APaydayPlayerState>( GetWorld()->GetFirstPlayerController()->PlayerState );
        if( cPlayerState && cPlayerCharacter )
        {
            if( fReduceAmount > fCurrentHealth )
            {
                cPlayerState->RecordPlayerDamageTaken( fCurrentHealth );
            }
            else
            {
                cPlayerState->RecordPlayerDamageTaken( fReduceAmount );
            }
        }
    }
    // ...
    if( !bIsReviving && fReviveDelay != 0 ){ ... }
    if( fCurrentArmourValue <= 0 )
    {
        GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Green ,FString::Printf( TEXT( "%s - Triggered Server Reduce Health; ActorIsDead: %s, ActorLivesRemaining
        // Handle the reduce health event on the server-side
        if( !bPlayerDead && iCurrentLives > 0 )
        {
            GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Green , TEXT( "Something got hurt" ) );
            if( fCurrentHealth > 0 )
            {
                if( fReduceAmount >= fCurrentHealth )
                {
                    fCurrentHealth = 0;
                    bPlayerDead = true;
                    bIsReviving = true;
                    Client_UpdateReviveStatus( bIsReviving );
                    iCurrentLives--;
                    Current_Lives = iCurrentLives;
                    GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Yellow , FString::Printf( TEXT( "%s has died a most tragic death. Oh well... Life goes on.
                    // Notify the client to update the lives count and health
                    Client_UpdateLives( iCurrentLives );

                    // Notify the client to update the flag of dead
                    Client_UpdatePlayerIsDead( bPlayerDead );

                }
                else
                {
                    fCurrentHealth -= fReduceAmount;
                    GEngine->AddOnScreenDebugMessage( -1 , 2.0f , FColor::Yellow , FString::Printf( TEXT( "%s has taken damage!" ) , *GetOwner()->GetName() ) );
                }

                // Notify the client to update the health
                Client_UpdateHealth( fCurrentHealth );

            }
            Current_Health = fCurrentHealth;
        }
    }
```

Due to the nature of network code various checks had to be made to ensure the variables were being assigned properly, resulting in bloated code.

There is an else{} statement below this which handles damage to armour, making this function rather large.

There are also additional features in the Health Component class to handle healing, reviving and full healing, some of those functions were never used but could have assets created to utilize them, for instance a health kit to restore lost health could easily be made to interact with a player's health component.

I also fixed a few issues with the UI one issue was the crosshair was off centre so I fixed that with a slight adjustment to the logic enabling us to swap the image to anything we desire and it will automatically be centred on the screen. At first I didn't realise there was a function that returns the X and Y values of the texture file, after I noticed that I quickly amended the above function to utilize those features so we could update the crosshair if we ever wanted to do so.



```cpp
// offset by half the texture's dimensions so that the center of the texture aligns with the center of the Canvas
const FVector2D CrosshairDrawPosition( (Center.X - (CrosshairTex->GetSizeX() / 2)),
                                       (Center.Y - (CrosshairTex->GetSizeY() / 2)));
```

The second issue I fixed enabled players to return to the main menu from the tutorial level via the extraction van, whereas before you could only leave the tutorial level via the pause menu.

I also spent a few days working out bugs relating to the AI behaviour trees and task priorities as they were causing the game to freeze under specific situations, this seems to be resolved now after investigation by myself and Pratik.

VFX.

This is a section to cover some of my notes about Unreal's VFX systems.

During my implementation of the project's VFX I tested out two different methods for handling VFX, one was to use single lifespan VFX which means they played once and terminated and then handled by Unreal at some point. The other method is to use timers to enable and disable particle spawning of the VFX when desired. From my testing neither method seems to be advantageous over the over performance wise, I was expecting to see memory usage spike when using single life VFX for gun VFX but to my surprise that was not the case, the memory usage stayed stable the entire time. The advantage that I could see is that using the timer method means you do not need multiple versions of the VFX.

Features I would have implemented if I had the time:

Player models and animations, this task would have been nice to complete and would have been a requirement for the project if we were creating a multiplayer game, however due to time constraints and having slightly more important aspects to work on I was unable to implement any meaningful animations for the player character.

Melee Combat, again something that was cut from the project due to overs scoping, this would have been a nice feature to have for when you eventually get caught reloading both your guns in a doorway with four AI running at you.

Gun Recoil and Bullet Spread, something that was not requested to be implemented by design, its just something I would have liked to include, bullet spread would be super easy to implement with the current Base Gun class as all it would need is a random float variation added into the ray cast end vector based on the number of shots fired within a given time frame.

A shotgun, because trying to shoot the AI with the rifle can be difficult at times. Also, it would be fun and look cool.

Player Sprinting / encumbered movement mechanics, this would not have been too difficult to implement in the current build as affecting the player's movement speed is simply changing a float variable in the player class.

A projectile based weapon, because then I would need to create an object pool class to manage the ammunition for the weapon.

A complete bibliography for the entire project start to finish.

Bibliography.

AnchiesesI, 2015. *Include Plugin in C++.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/c-gameplay-programming/57466-include-plugin-in-c
[Accessed 06 2020].

Andargor, 2014. *Generate Procedural Mesh.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/c-gameplay-programming/1674-generate-procedural-mesh?1552-Generate-Procedural-Mesh
[Accessed 06 2020].

Anon., 2015. *Stopping an emitter after a single run.* [Online]
Available at: https://answers.unrealengine.com/questions/222736/stopping-an-emitter-after-a-single-run.html
[Accessed 2020].

Benjaml, 2017. *Procedural Mesh only detect entering collision.* [Online]
Available at: https://answers.unrealengine.com/questions/726637/procedural-mesh-only-detect-entering-collision.html?sort=oldest
[Accessed 06 2020].

BramV, 2016. *How to set Lock Position and Lock Location on Component in C++?.* [Online]
Available at: https://answers.unrealengine.com/questions/379860/how-to-set-lock-position-and-lock-location-on-comp.html
[Accessed 06 2020].

ClaverFlav, 2014. *How to stop a particle system from spawning and remove when active particles are dead??.* [Online]
Available at: https://answers.unrealengine.com/questions/75091/how-to-stop-a-particle-system-from-spawning-and-re.html?sort=oldest
[Accessed 2020].

cplusplusgametricks, 2017. *Create C++ RPC functions by UFUNCTION().* [Online]
Available at: https://cplusplusgametricks.wordpress.com/2017/04/24/create-c-rpc-functions-by-ufunction/
[Accessed 06 2020].

CryDead, 2015. *Particle not showing in game.* [Online]
Available at: https://answers.unrealengine.com/questions/247330/particle-not-showing-in-game-1.html
[Accessed 2020].

D'areglia, Y., 2019. *Unreal Engine: Environment Query System (EQS) in C++.* [Online]
Available at: https://www.thinkandbuild.it/environment-query-system-in-c/
[Accessed 06 2020].

Eleftgerious, O., 2016. *Creating Custom EQS Generators in C++.* [Online]
Available at: https://www.orfeasel.com/creating-custom-eqs-generators-in-c/
[Accessed 06 2020].

Epic Games, Unreal Engine, Unreal Engine 4 Community Wiki, n.d. *Legacy/Procedural Mesh Component in C++:Getting Started.* [Online]
Available at:
https://www.ue4community.wiki/Legacy/Procedural_Mesh_Component_in_C%2B%2B:Getting_Started
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Custom Mesh.* [Online]
Available at: https://docs.unrealengine.com/en-US/BlueprintAPI/Components/CustomMesh/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *FEnvQueryRequest.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/AIModule/EnvironmentQuery/FEnvQueryRequest/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *FTimerManager.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Engine/FTimerManager/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Gameplay Timers.* [Online]
Available at: https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Timers/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Get Query Results as Locations.* [Online]
Available at: https://docs.unrealengine.com/en-US/BlueprintAPI/AI/EQS/GetQueryResultsasLocations/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/index.html.* [Online]
Available at: https://docs.unrealengine.com/en-US/BlueprintAPI/Components/ProceduralMesh/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *RPCs.* [Online]
Available at: https://docs.unrealengine.com/en-US/Gameplay/Networking/Actors/RPCs/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Run EQSQuery.* [Online]
Available at: https://docs.unrealengine.com/en-US/BlueprintAPI/AI/EQS/RunEQSQuery/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *UBTTask_RunEQSQuery.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/AIModule/BehaviorTree/Tasks/UBTTask_RunEQSQuery/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *UCustomMeshComponent.* [Online]
Available at: https://docs.unrealengine.com/en-

US/API/Plugins/CustomMeshComponent/UCustomMeshComponent/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *UEngine::AddOnScreenDebugMessage.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Engine/Engine/UEngine/AddOnScreenDebugMessage/1/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *UGameplayStatics::SpawnEmitterAtLocation.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Engine/Kismet/UGameplayStatics/SpawnEmitterAtLocation/1/index.html
[Accessed 2020].

Epic Games, Unreal Engine, n.d. *UGameplayStatics::SpawnEmitterAttached.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Engine/Kismet/UGameplayStatics/SpawnEmitterAttached/2/index.html
[Accessed 2020].

Epic Games, Unreal Engine, n.d. *UMG UI Designer Quick Start Guide.* [Online]
Available at: https://docs.unrealengine.com/en-US/Engine/UMG/QuickStart/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *UParticleEmitter.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Runtime/Engine/Particles/UParticleEmitter/index.html
[Accessed 2020].

Epic Games, Unreal Engine, n.d. *UProceduralMeshComponent.* [Online]
Available at: https://docs.unrealengine.com/en-US/API/Plugins/ProceduralMeshComponent/UProceduralMeshComponent/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Using a Multi Line Trace (Raycast) by Channel.* [Online]
Available at: https://docs.unrealengine.com/en-US/Engine/Physics/Tracing/HowTo/MultiLineTraceByChannel/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Using a Single Line Trace (Raycast) by Channel.* [Online]
Available at: https://docs.unrealengine.com/en-US/Engine/Physics/Tracing/HowTo/SingleLineTraceByChannel/index.html
[Accessed 06 2020].

Epic Games, Unreal Engine, n.d. *Using a Single Line Trace (Raycast) by Object.* [Online]
Available at: https://docs.unrealengine.com/en-US/Engine/Physics/Tracing/HowTo/SingleLineTraceByObject/index.html
[Accessed 06 2020].

Games, D., 2017. *Stealth Vision Cone (Procedural Mesh) - ue4 tutorial.* [Online]
Available at: https://www.youtube.com/watch?v=mN_ZtamYY1Q
[Accessed 06 2020].

Giovannini, 2015. *How to ignore actor class with line trace in C++?.* [Online]
Available at: https://answers.unrealengine.com/questions/153822/how-to-ignore-actor-class-with-line-trace-in-c.html
[Accessed 06 2020].

Kazemusha, 2016. *Print to Screen using C++.* [Online]
Available at: https://answers.unrealengine.com/questions/419732/print-to-screen-using-c.html
[Accessed 06 2020].

Kazualty, 2016. *Rotate a door on collision?.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/animation/94265-rotate-a-door-on-collision
[Accessed 06 2020].

McGuire, H., 2018. *Unreal Engine 4 C++ Tutorial: Line Trace on Fire.* [Online]
Available at: https://www.youtube.com/watch?v=xgt04-2rKV8
[Accessed 06 2020].

MrFalaranah, 2014. *Unreal Engine 4: Change Color On Particle Effects Tutorial.* [Online]
Available at: https://www.youtube.com/watch?v=8uKgMwd6jVw
[Accessed 2020].

NebulaGamesInc, 2019. *** Updated ** World Date and Clock.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/1599273-updated-world-date-and-clock
[Accessed 2020].

NebulaGamesInc, 2019. *C++ to Blueprints World Date And Time Line 263..* [Online]
Available at: https://pastebin.com/Eg6EZyRa
[Accessed 2020].

NebulaGamesInc, 2019. *Unreal Engine 4 Tutorial: World Date And Time #UE4 #RTS #Date.* [Online]
Available at: https://www.youtube.com/watch?v=7A7criIbmgc&feature=youtu.be
[Accessed 2020].

PatrykP, 2018. *Particle system problem (not showing in game).* [Online]
Available at: https://forums.unrealengine.com/development-discussion/content-creation/1429279-particle-system-problem-not-showing-in-game
[Accessed 2020].

RealSuperku, 2017. *new to C++: optional TArray parameter/ UProceduralMeshComponent.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/c-gameplay-programming/118489-new-to-c-optional-tarray-parameter-uproceduralmeshcomponent
[Accessed 06 2020].

Saikanami-, 2016. *SpawnEmitterAttached not working.* [Online]
Available at: https://answers.unrealengine.com/questions/473871/spawnemitterattached-not-working.html
[Accessed 2020].

sayer6913, 2015. *How do I make a weapon fire fully-automatic?.* [Online]
Available at: https://answers.unrealengine.com/questions/344301/how-do-i-make-a-weapon-fire-fully-automatic.html?lang=ja
[Accessed 2020].

Slowmanrunning, 2015. *Start/Stop Emitting Particle System Component.* [Online]
Available at: https://answers.unrealengine.com/questions/158065/startstop-emitting-particle-system-component.html
[Accessed 2020].

SRombauts, 2014. *Generate Procedural Mesh.* [Online]
Available at: https://forums.unrealengine.com/development-discussion/c-gameplay-programming/1674-generate-procedural-mesh/page7
[Accessed 06 2020].

SRombauts, 2016. *UE4ProceduralMesh.* [Online]
Available at: https://github.com/SRombauts/UE4ProceduralMesh/
[Accessed 06 2020].

SRombauts, 2016. *UE4ProceduralMesh/README.md.* [Online]
Available at: https://github.com/SRombauts/UE4ProceduralMesh/blob/master/README.md
[Accessed 06 2020].

Synty Studios, 2017. *Synty Polygon-Heist Pack.* [Online]
Available at: https://www.unrealengine.com/marketplace/en-US/product/polygon-heist-pack
[Accessed 06 2020].

TitanicGames, 2017. *UE4 Survival Game - Displaying Current Time (Part 9).* [Online]
Available at: https://www.youtube.com/watch?v=Z88lSlSN87o
[Accessed 2020].