Technical Design Document

Blockout Shooter

William John Lautama

Table of Contents

Project Overview
Third Person Multiplayer Parkour Arena. That is what this game is all about. But its not just a regular third person game, it utilised parkour, fast traversal while also encouraging combat with items and unique weapons. Having the theme of the Glitch, the game has unique mechanics that capitalise on the theme such as the Chaos Mace. The game also focuses on a wall running mechanic combined to encourage high speed traversal in a parkour arena. There are also items that are present within the levels that the player can take advantage of. One of the main items are the poison trail that is in the form of a box that the player can take and a poison trail will appear behind the player that inflicts poison damage when it is hit. The aim is to implement this core mechanic efficiently and make it engaging for the player. The game is being developed for PC as it allows for smoother execution of the wall running mechanic with the keyboard, and the third person format is easier to control with a keyboard and mouse compared to consoles. This aims to provide a comfortable gaming experience for players.

Game Mechanics Overview
There are 2 main mechanics. One of the mechanics is the wall run mechanic. Wall run mechanics are very common in typical movement video games as a mechanic but usually, the format of the game is not in the form of a party game. By combining fast paced movement and shooting in a third person format, it creates an engaging and enticing experience of a mixture of movement and shooting.

Another thing that is within the game are items, specifically the poison trail item. This creates a variety of challenges to the players by making the player that has gotten the poison trail item an advantage of movement by limiting the other player's movement in the arena. There are advantages and disadvantages of the mechanic but overall, it creates an exciting way to approach the arena with the available movements.
By combining the fast paced wall run movement and the items that are available in the game, it creates unlimited possibilities of gameplay and interactions between players.

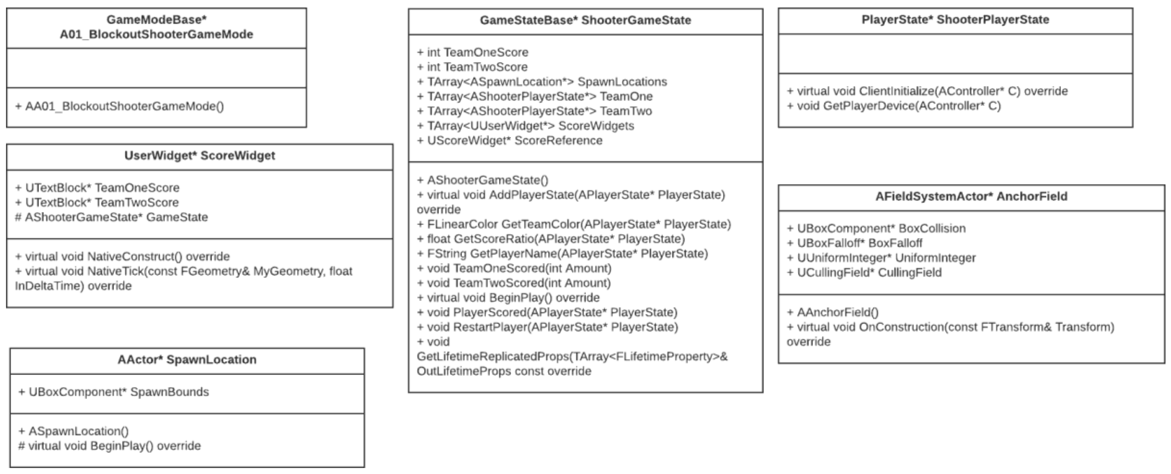
Target Platform
The platform that I am making this game for is PC. This is due to the fact that with pc, and the keyboard that are attached, the wall running mechanic can run smoother since it is very focused on pressing multiple keys on the keyboard at the same time. While this effect can be achieved in other consoles, it is harder to do so, especially on handheld and consoles that rely on the analog stick for movement.

Even though a third person format game is generally easier to control in console, the genre of action shooters are generally easier to control on PC. This is due to the aiming factor of the game being shooting and movement based therefore, by using both keyboard and mouse, more control of the game can be achieved. The item factor itself doesn’t really affect the game’s platform release but if the players are using PC to use the items, by providing extra control, the items can be more easily taken and utilised by them.

Game Mechanics

UML Diagram

The arrows are the representation of the relationship between the classes. This means, the class triggers or relates directly in activating the other class that it is pointing to.



```
GameModeBase*
A01_BlockoutShooterGameMode

+ AA01_BlockoutShooterGameMode()
```

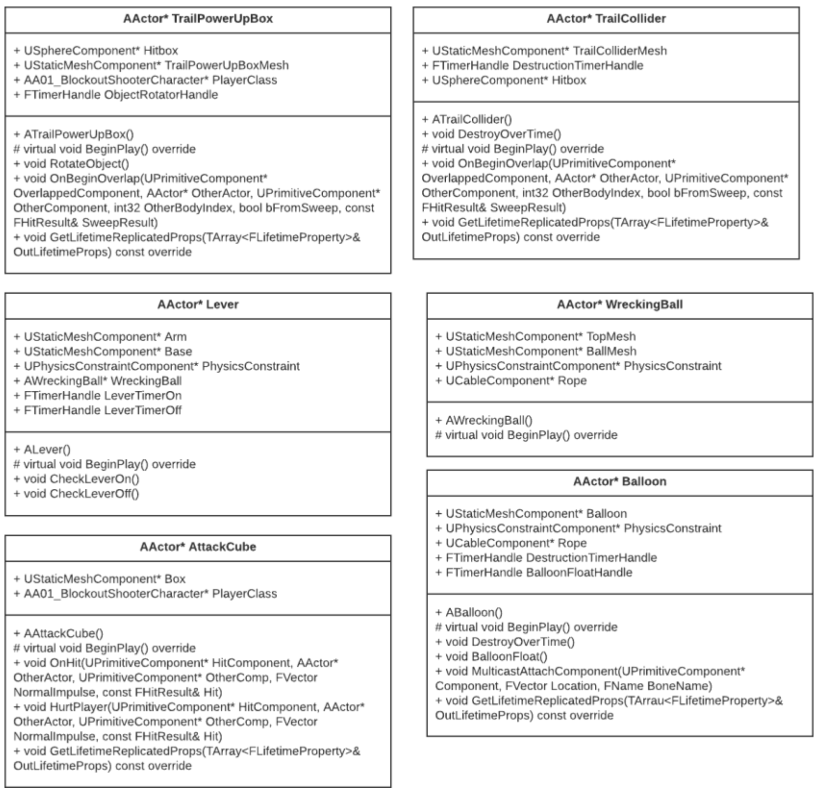
```
UserWidget* ScoreWidget

+ UTextBlock* TeamOneScore
+ UTextBlock* TeamTwoScore
# AShooterGameState* GameState

+ virtual void NativeConstruct() override
+ virtual void NativeTick(const FGeometry& MyGeometry, float
InDeltaTime) override
```

```
AActor* SpawnLocation

+ UBoxComponent* SpawnBounds

+ ASpawnLocation()
# virtual void BeginPlay() override
```

```
GameStateBase* ShooterGameState

+ int TeamOneScore
+ int TeamTwoScore
+ TArray<ASpawnLocation*> SpawnLocations
+ TArray<AShooterPlayerState*> TeamOne
+ TArray<AShooterPlayerState*> TeamTwo
+ TArray<UUserWidget*> ScoreWidgets
+ UScoreWidget* ScoreReference

+ AShooterGameState()
+ virtual void AddPlayerState(APlayerState* PlayerState)
override
+ FLinearColor GetTeamColor(APlayerState* PlayerState)
+ float GetScoreRatio(APlayerState* PlayerState)
+ FString GetPlayerName(APlayerState* PlayerState)
+ void TeamOneScored(int Amount)
+ void TeamTwoScored(int Amount)
+ virtual void BeginPlay() override
+ void PlayerScored(APlayerState* PlayerState)
+ void RestartPlayer(APlayerState* PlayerState)
+ void
GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps const override
```

```
PlayerState* ShooterPlayerState

+ virtual void ClientInitialize(AController* C) override
+ void GetPlayerDevice(AController* C)
```

```
AFieldSystemActor* AnchorField

+ UBoxComponent* BoxCollision
+ UBoxFalloff* BoxFalloff
+ UUniformInteger* UniformInteger
+ UCullingField* CullingField

+ AAnchorField()
+ virtual void OnConstruction(const FTransform& Transform)
override
```



```
AActor* TrailPowerUpBox

+ USphereComponent* Hitbox
+ UStaticMeshComponent* TrailPowerUpBoxMesh
+ AA01_BlockoutShooterCharacter* PlayerClass
+ FTimerHandle ObjectRotatorHandle

+ ATrailPowerUpBox()
# virtual void BeginPlay() override
+ void RotateObject()
+ void OnBeginOverlap(UPrimitiveComponent*
OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*
OtherComponent, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

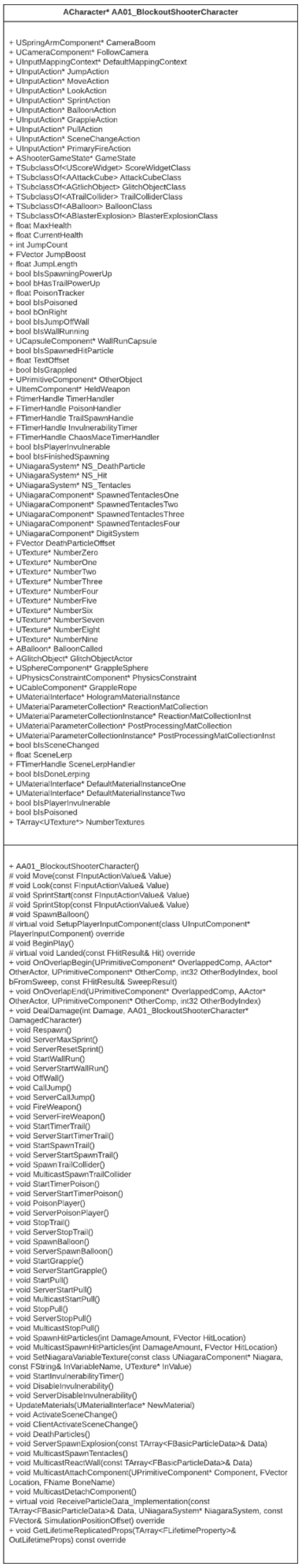
```
AActor* TrailCollider

+ UStaticMeshComponent* TrailColliderMesh
+ FTimerHandle DestructionTimerHandle
+ USphereComponent* Hitbox

+ ATrailCollider()
+ void DestroyOverTime()
# virtual void BeginPlay() override
+ void OnBeginOverlap(UPrimitiveComponent*
OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*
OtherComponent, int32 OtherBodyIndex, bool bFromSweep, const
FHitResult& SweepResult)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

```
AActor* Lever

+ UStaticMeshComponent* Arm
+ UStaticMeshComponent* Base
+ UPhysicsConstraintComponent* PhysicsConstraint
+ AWreckingBall* WreckingBall
+ FTimerHandle LeverTimerOn
+ FTimerHandle LeverTimerOff

+ ALever()
# virtual void BeginPlay() override
+ void CheckLeverOn()
+ void CheckLeverOff()
```

```
AActor* WreckingBall

+ UStaticMeshComponent* TopMesh
+ UStaticMeshComponent* BallMesh
+ UPhysicsConstraintComponent* PhysicsConstraint
+ UCableComponent* Rope

+ AWreckingBall()
# virtual void BeginPlay() override
```

```
AActor* AttackCube

+ UStaticMeshComponent* Box
+ AA01_BlockoutShooterCharacter* PlayerClass

+ AAttackCube()
# virtual void BeginPlay() override
+ void OnHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, FVector
NormalImpulse, const FHitResult& Hit)
+ void HurtPlayer(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, FVector
NormalImpulse, const FHitResult& Hit)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

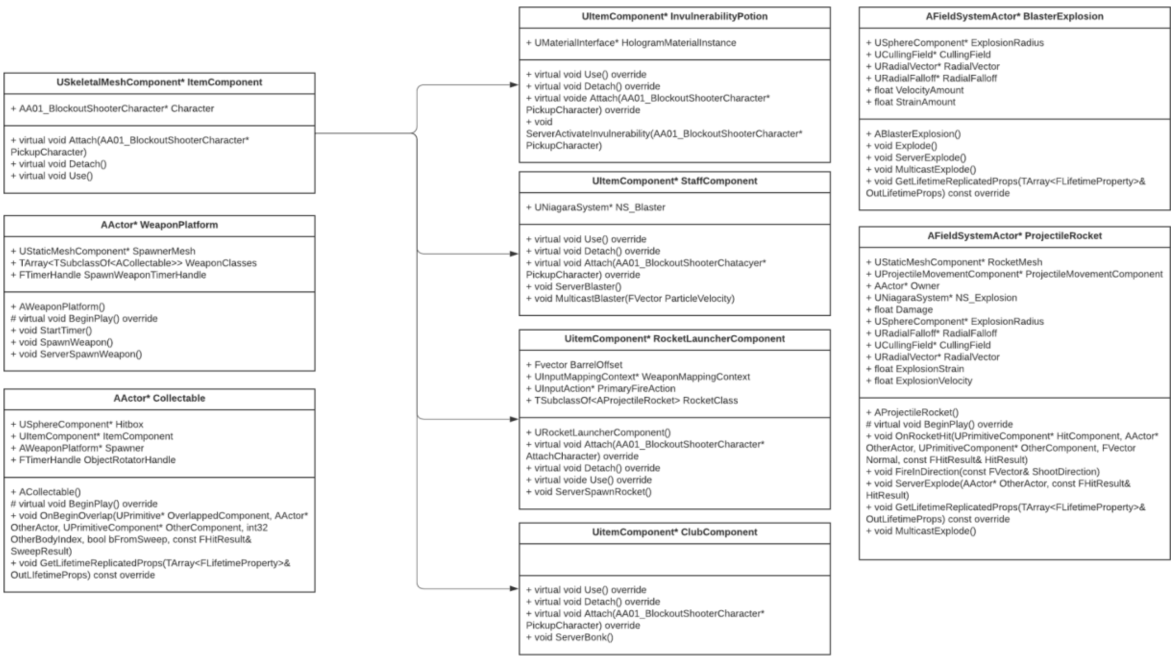
```
AActor* Balloon

+ UStaticMeshComponent* Balloon
+ UPhysicsConstraintComponent* PhysicsConstraint
+ UCableComponent* Rope
+ FTimerHandle DestructionTimerHandle
+ FTimerHandle BalloonFloatHandle

+ ABalloon()
# virtual void BeginPlay() override
+ void DestroyOverTime()
+ void BalloonFloat()
+ void MulticastAttachComponent(UPrimitiveComponent*
Component, FVector Location, FName BoneName)
+ void GetLifetimeReplicatedProps(TArrau<FLifetimeProperty>&
OutLifetimeProps) const override
```



ACharacter* AA01_BlockoutShooterCharacter

```
+ USpringArmComponent* CameraBoom
+ UCameraComponent* FollowCamera
+ UInputMappingContext* DefaultMappingContext
+ UInputAction* JumpAction
+ UInputAction* MoveAction
+ UInputAction* LookAction
+ UInputAction* SprintAction
+ UInputAction* BalloonAction
+ UInputAction* GrappleAction
+ UInputAction* PullAction
+ UInputAction* SceneChangeAction
+ UInputAction* PrimaryFireAction
+ AShooterGameState* GameState
+ TSubclassOf<UScoreWidget> ScoreWidgetClass
+ TSubclassOf<AAttackCube> AttackCubeClass
+ TSubclassOf<AGtlichObject> GlitchObjectClass
+ TSubclassOf<ATrailCollider> TrailColliderClass
+ TSubclassOf<ABalloon> BalloonClass
+ TSubclassOf<ABlasterExplosion> BlasterExplosionClass
+ float MaxHealth
+ float CurrentHealth
+ int JumpCount
+ FVector JumpBoost
+ float JumpLength
+ bool bIsSpawningPowerUp
+ bool bHasTrailPowerUp
+ float PoisonTracker
+ bool bIsPoisoned
+ bool bOnRight
+ bool bIsJumpOffWall
+ bool bIsWallRunning
+ UCapsuleComponent* WallRunCapsule
+ bool bIsSpawnedHitParticle
+ float TextOffset
+ bool bIsGrappled
+ UPrimitiveComponent* OtherObject
+ UItemComponent* HeldWeapon
+ FtimerHandle TimerHandler
+ FTimerHandle PoisonHandler
+ FTimerHandle TrailSpawnHandle
+ FTimerHandle InvulnerabilityTimer
+ FTimerHandle ChaosMaceTimerHandler
+ bool bIsPlayerInvulnerable
+ bool bIsFinishedSpawning
+ UNiagaraSystem* NS_DeathParticle
+ UNiagaraSystem* NS_Hit
+ UNiagaraSystem* NS_Tentacles
+ UNiagaraComponent* SpawnedTentaclesOne
+ UNiagaraComponent* SpawnedTentaclesTwo
+ UNiagaraComponent* SpawnedTentaclesThree
+ UNiagaraComponent* SpawnedTentaclesFour
+ UNiagaraComponent* DigitSystem
+ FVector DeathParticleOffset
+ UTexture* NumberZero
+ UTexture* NumberOne
+ UTexture* NumberTwo
+ UTexture* NumberThree
+ UTexture* NumberFour
+ UTexture* NumberFive
+ UTexture* NumberSix
+ UTexture* NumberSeven
+ UTexture* NumberEight
+ UTexture* NumberNine
+ ABalloon* BalloonCalled
+ AGlitchObject* GlitchObjectActor
+ USphereComponent* GrappleSphere
+ UPhysicsConstraintComponent* PhysicsConstraint
+ UCableComponent* GrappleRope
+ UMaterialInterface* HologramMaterialInstance
+ UMaterialParameterCollection* ReactionMatCollection
+ UMaterialParameterCollectionInstance* ReactionMatCollectionInst
+ UMaterialParameterCollection* PostProcessingMatCollection
+ UMaterialParameterCollectionInstance* PostProcessingMatCollectionInst
+ bool bIsSceneChanged
+ float SceneLerp
+ FTimerHandle SceneLerpHandler
+ bool bIsDoneLerping
+ UMaterialInterface* DefaultMaterialInstanceOne
+ UMaterialInterface* DefaultMaterialInstanceTwo
+ bool bIsPlayerInvulnerable
+ bool bIsPoisoned
+ TArray<UTexture*> NumberTextures
```

```
+ AA01_BlockoutShooterCharacter()
# void Move(const FInputActionValue& Value)
# void Look(const FInputActionValue& Value)
# void SprintStart(const FInputActionValue& Value)
# void SprintStop(const FInputActionValue& Value)
# void SpawnBalloon()
# virtual void SetupPlayerInputComponent(class UInputComponent*
PlayerInputComponent) override
# void BeginPlay()
# virtual void Landed(const FHitResult& Hit) override
+ void OnOverlapBegin(UPrimitiveComponent* OverlappedComp, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex, bool
bFromSweep, const FHitResult& SweepResult)
+ void OnOverlapEnd(UPrimitiveComponent* OverlappedComp, AActor*
OtherActor, UPrimitiveComponent* OtherComp, int32 OtherBodyIndex)
+ void DealDamage(int Damage, AA01_BlockoutShooterCharacter*
DamagedCharacter)
+ void Respawn()
+ void ServerMaxSprint()
+ void ServerResetSprint()
+ void StartWallRun()
+ void ServerStartWallRun()
+ void OffWall()
+ void CallJump()
+ void ServerCallJump()
+ void FireWeapon()
+ void ServerFireWeapon()
+ void StartTimerTrail()
+ void ServerStartTimerTrail()
+ void StartSpawnTrail()
+ void ServerStartSpawnTrail()
+ void SpawnTrailCollider()
+ void MulticastSpawnTrailCollider
+ void StartTimerPoison()
+ void ServerStartTimerPoison()
+ void PoisonPlayer()
+ void ServerPoisonPlayer()
+ void StopTrail()
+ void ServerStopTrail()
+ void SpawnBalloon()
+ void ServerSpawnBalloon()
+ void StartGrapple()
+ void ServerStartGrapple()
+ void StartPull()
+ void ServerStartPull()
+ void MulticastStartPull()
+ void StopPull()
+ void ServerStopPull()
+ void MulticastStopPull()
+ void SpawnHitParticles(int DamageAmount, FVector HitLocation)
+ void MulticastSpawnHitParticles(int DamageAmount, FVector HitLocation)
+ void SetNiagaraVariableTexture(const class UNiagaraComponent* Niagara,
const FString& InVariableName, UTexture* InValue)
+ void StartInvulnerabilityTimer()
+ void DisableInvulnerability()
+ void ServerDisableInvulnerability()
+ UpdateMaterials(UMaterialInterface* NewMaterial)
+ void ActivateSceneChange()
+ void ClientActivateSceneChange()
+ void DeathParticles()
+ void ServerSpawnExplosion(const TArray<FBasicParticleData>& Data)
+ void MulticastSpawnTentacles()
+ void MulticastReactWall(const TArray<FBasicParticleData>& Data)
+ void MulticastAttachComponent(UPrimitiveComponent* Component, FVector
Location, FName BoneName)
+ void MulticastDetachComponent()
+ virtual void ReceiveParticleData_Implementation(const
TArray<FBasicParticleData>& Data, UNiagaraSystem* NiagaraSystem, const
FVector& SimulationPositionOffset) override
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```



```
USkeletalMeshComponent* ItemComponent
+ AA01_BlockoutShooterCharacter* Character
+ virtual void Attach(AA01_BlockoutShooterCharacter*
PickupCharacter)
+ virtual void Detach()
+ virtual void Use()
```

```
AActor* WeaponPlatform
+ UStaticMeshComponent* SpawnerMesh
+ TArray<TSubclassOf<ACollectable>> WeaponClasses
+ FTimerHandle SpawnWeaponTimerHandle
+ AWeaponPlatform()
# virtual void BeginPlay() override
+ void StartTimer()
+ void SpawnWeapon()
+ void ServerSpawnWeapon()
```

```
AActor* Collectable
+ USphereComponent* Hitbox
+ UItemComponent* ItemComponent
+ AWeaponPlatform* Spawner
+ FTimerHandle ObjectRotatorHandle
+ ACollectable()
# virtual void BeginPlay() override
+ void OnBeginOverlap(UPrimitive* OverlappedComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComponent, int32
OtherBodyIndex, bool bFromSweep, const FHitResult&
SweepResult)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLIfetimeProps) const override
```

```
UItemComponent* InvulnerabilityPotion
+ UMaterialInterface* HologramMaterialInstance
+ virtual void Use() override
+ virtual void Detach() override
+ virtual voide Attach(AA01_BlockoutShooterCharacter*
PickupCharacter) override
+ void
ServerActivateInvulnerability(AA01_BlockoutShooterCharacter*
PickupCharacter)
```

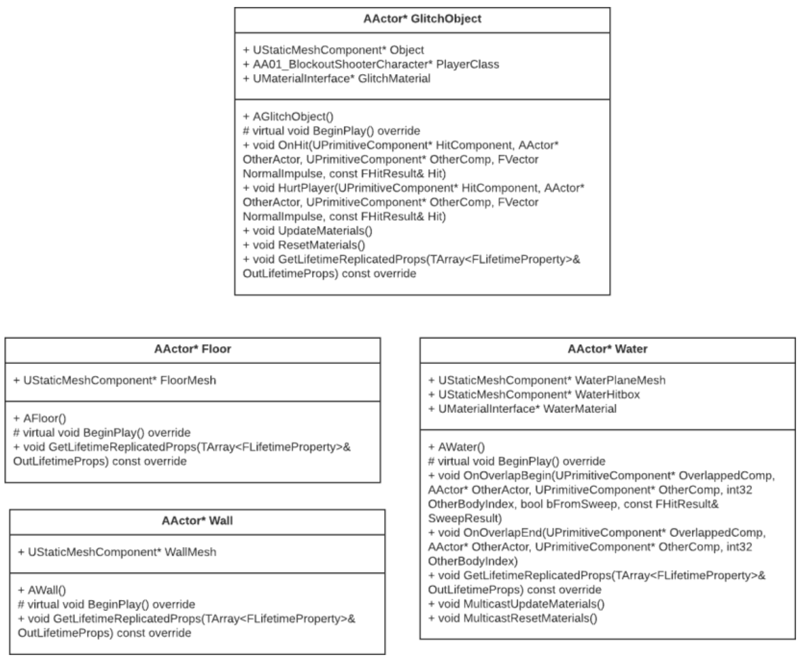
```
UItemComponent* StaffComponent
+ UNiagaraSystem* NS_Blaster
+ virtual void Use() override
+ virtual void Detach() override
+ virtual void Attach(AA01_BlockoutShooterChatacyer*
PickupCharacter) override
+ void ServerBlaster()
+ void MulticastBlaster(FVector ParticleVelocity)
```

```
UitemComponent* RocketLauncherComponent
+ Fvector BarrelOffset
+ UInputMappingContext* WeaponMappingContext
+ UInputAction* PrimaryFireAction
+ TSubclassOf<AProjectileRocket> RocketClass
+ URocketLauncherComponent()
+ virtual void Attach(AA01_BlockoutShooterCharacter*
AttachCharacter) override
+ virtual void Detach() override
+ virtual voide Use() override
+ void ServerSpawnRocket()
```

```
UitemComponent* ClubComponent
+ virtual void Use() override
+ virtual void Detach() override
+ virtual void Attach(AA01_BlockoutShooterCharacter*
PickupCharacter) override
+ void ServerBonk()
```

```
AFieldSystemActor* BlasterExplosion
+ USphereComponent* ExplosionRadius
+ UCullingField* CullingField
+ URadialVector* RadialVector
+ URadialFalloff* RadialFalloff
+ float VelocityAmount
+ float StrainAmount
+ ABlasterExplosion()
+ void Explode()
+ void ServerExplode()
+ void MulticastExplode()
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

```
AFieldSystemActor* ProjectileRocket
+ UStaticMeshComponent* RocketMesh
+ UProjectileMovementComponent* ProjectileMovementComponent
+ AActor* Owner
+ UNiagaraSystem* NS_Explosion
+ float Damage
+ USphereComponent* ExplosionRadius
+ URadialFalloff* RadialFalloff
+ UCullingField* CullingField
+ URadialVector* RadialVector
+ float ExplosionStrain
+ float ExplosionVelocity
+ AProjectileRocket()
# virtual void BeginPlay() override
+ void OnRocketHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComponent, FVector
Normal, const FHitResult& HitResult)
+ void FireInDirection(const FVector& ShootDirection)
+ void ServerExplode(AActor* OtherActor, const FHitResult&
HitResult)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
+ void MulticastExplode()
```



```
AActor* GlitchObject
+ UStaticMeshComponent* Object
+ AA01_BlockoutShooterCharacter* PlayerClass
+ UMaterialInterface* GlitchMaterial
+ AGlitchObject()
# virtual void BeginPlay() override
+ void OnHit(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, FVector
NormalImpulse, const FHitResult& Hit)
+ void HurtPlayer(UPrimitiveComponent* HitComponent, AActor*
OtherActor, UPrimitiveComponent* OtherComp, FVector
NormalImpulse, const FHitResult& Hit)
+ void UpdateMaterials()
+ void ResetMaterials()
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

```
AActor* Floor
+ UStaticMeshComponent* FloorMesh
+ AFloor()
# virtual void BeginPlay() override
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

```
AActor* Wall
+ UStaticMeshComponent* WallMesh
+ AWall()
# virtual void BeginPlay() override
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
```

```
AActor* Water
+ UStaticMeshComponent* WaterPlaneMesh
+ UStaticMeshComponent* WaterHitbox
+ UMaterialInterface* WaterMaterial
+ AWater()
# virtual void BeginPlay() override
+ void OnOverlapBegin(UPrimitiveComponent* OverlappedComp,
AActor* OtherActor, UPrimitiveComponent* OtherComp, int32
OtherBodyIndex, bool bFromSweep, const FHitResult&
SweepResult)
+ void OnOverlapEnd(UPrimitiveComponent* OverlappedComp,
AActor* OtherActor, UPrimitiveComponent* OtherComp, int32
OtherBodyIndex)
+ void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>&
OutLifetimeProps) const override
+ void MulticastUpdateMaterials()
+ void MulticastResetMaterials()
```

Movement Mechanics

This game has fairly simple movement mechanics. The basic movement (walking) is executed through W, A, S, and D. With W moving forward, A moving to the left, S moving backwards, and D moving to the right. The basic movement's max speed is 500. The game also allows the player to sprint. When the player sprints, the player's max speed is increased from 500 to 4000 making the movement experience way faster than regular walking.

The game also allows the player to jump with a custom jump that is based on the player's position. The jump's height is calculated from an FVector value that are (ActorForwardVector for X, ActorForwardVector for Y, and JumpLength which is 1.5. This is then stored to a variable called JumpBoost. The JumpBoost variable then converted into GetSafeNormal and multiplied by 1000. Then, using the LaunchCharacter function, the JumpBoost is used. When the player is wall running, instead of getting the actor forward vector, the actor's right vector based on which side the player is wallrunning is returned. The jump function is also affected with the wall running since the player will jump opposite of where the player is wall running. For example, if the wall is on the right of the player and the player is wall running, the player will jump to the left and vice versa. There is also a jump count that starts with 2 that resets every time the player wall runs. The jump count also resets after the player stops being in mid air.

The collision of the player is based on the capsule component that is attached to it. The capsule component will collide with the walls that are in the environment to prevent the player from going through the walls. When the player is wall running, the player's overall setup is also changed which will be described in more detail in the additional gameplay mechanics overview.

Controls

Mapping	Action	Description	Keybind	Modifiers
LookMapping	Look Around	Used to move the character's camera based on the mouse cursor	Mouse Movements	Modify ControllerYaw and ControllerPitch input
WalkingMapping	Move Forward	Used to move the character forward	W	SwizzleAxisValue
WalkingMapping	Move Backwards	Used to move the character backwards	S	SwizzleAxisValue, Negate

WalkingMapping	Move to the Right	Used to move the character to the right	D	Change ActorRightVector
WalkingMapping	Move to the Left	Used to move the character to the right	A	Change ActorRightVector, Negative
JumpMapping	Jump	Used to make the player jumps depends on which state the player is in (Wall Running or Normal)	Space Bar	
SprintMapping	Sprint	Used to make the player walk faster (sprint)	Shift	
FireMapping	Shoot	Used to shoot projectiles	Left Mouse Button	
BalloonMapping	Attach Balloon	Used to Spawn Balloon and Attach it to the Object that the Player was Looking At	B	
GrappleMapping	Attach Grapple	Used to Attach Grapple Into Object that the Player was Looking At	Right Mouse Button	
PullMapping	Chaos Grapple	Used to Pull Grapple to Cause Chaos Attack	Q	

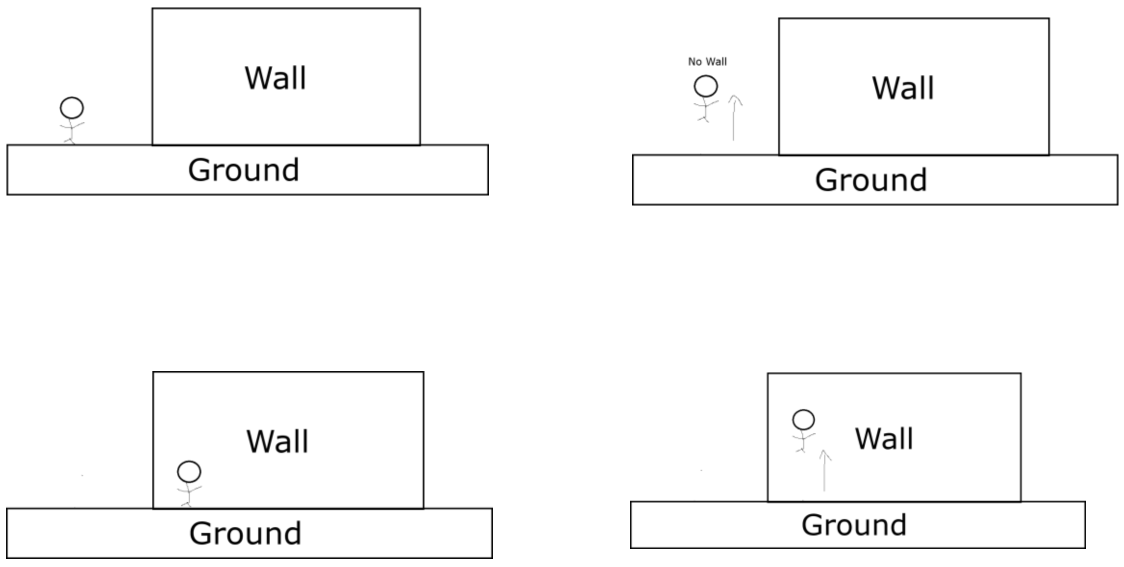
SceneChangeMapping	Outline Post Processing Activation	Used to activate / deactivate outline post processing	E	

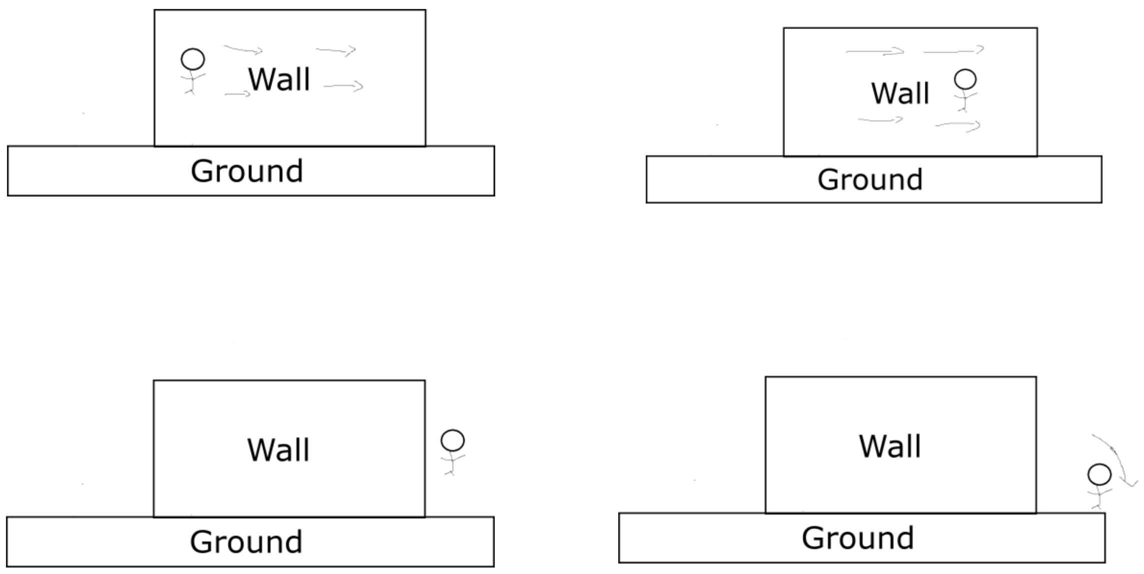
Additional Gameplay Mechanic 1 - Wall Run

Mechanic Overview

The mechanic that I included as an additional mechanic is the wall run mechanic. The way the wall run mechanic works is that it checks if the player is falling or not using a built in character movement function called IsFalling.

Then if the player is in the middle of falling or in the air, it checks if there is a wall on the side of the player. If there is no wall, nothing happens but if there is a wall, the player wall runs. The player wall runs in a horizontal direction of the wall until there are no more walls or until the player moves away from the wall.





Mechanic Description / Functionality

If the player is in the middle of falling or is in the air, it checks if there is a wall on the side of the player using a function called StartWallRun using raycast. The function StartWallRun then calls another function that is a Server RPC called ServerStartWallRun which runs on the server to prevent the player from modifying the movement code. Inside the ServerStartWallRun function, it casts a raycast that starts at the actor's location and ends on the right vector of the actor that is multiplied by 50 which is basically just the side of the player's character. The raycasts are executed based on where the wall is positioned from the player whether its left or right of the player.

If the raycasts hits, it adds a jump count to allow the player to jump, adds a boolean value to the bOnRight value based on where the wall is to the player, if it is on the right of the player then it is true, if it is left to the player then it is false. This is to make the jump functionality feel better based on which wall it is so it can propel the player opposite of the wall. Then, the bIsJumpOffWall is checked if it is false or not. If it is false, then it changes the variables. This is done to make sure that the player is not in the middle of jumping off a wall when it is trying to detect the wall.

Other than the ray cast, the player has a wall run capsule that overlaps with the wall to check more accurately. This can make the wall run more accurately and fluidly.

Then, the velocity is changed into the player's forward vector times by 2000 for the x and y, the gravity scale is changed to 0.4f, the plane constraint normal is set to 1 on the Z value but 0 on the X and Y, and the air control is set to 1.0f. The velocity is very large compared to the regular player velocity due to the fact that when the player is wall running, I want it to be faster than a regular walk which needs proper speed. The gravity scale is also lower because when the player wall runs, I want the player to not drop to the ground as fast as jumping. The air control 1 so the player can have full control of the player when wall running since it counts as being in the air. After that, if the player is touching the wall, the player will proceed to wall run with a determined velocity. The wall run is horizontal along the walls. If the player finishes wall running or if the player goes off the wall, the values that are adjusted goes back to the default values that were set from the beginning of the gameplay with a function called OffWall() that is a Server RPC.

In summary here is how the specific values work in the form of a diagram :

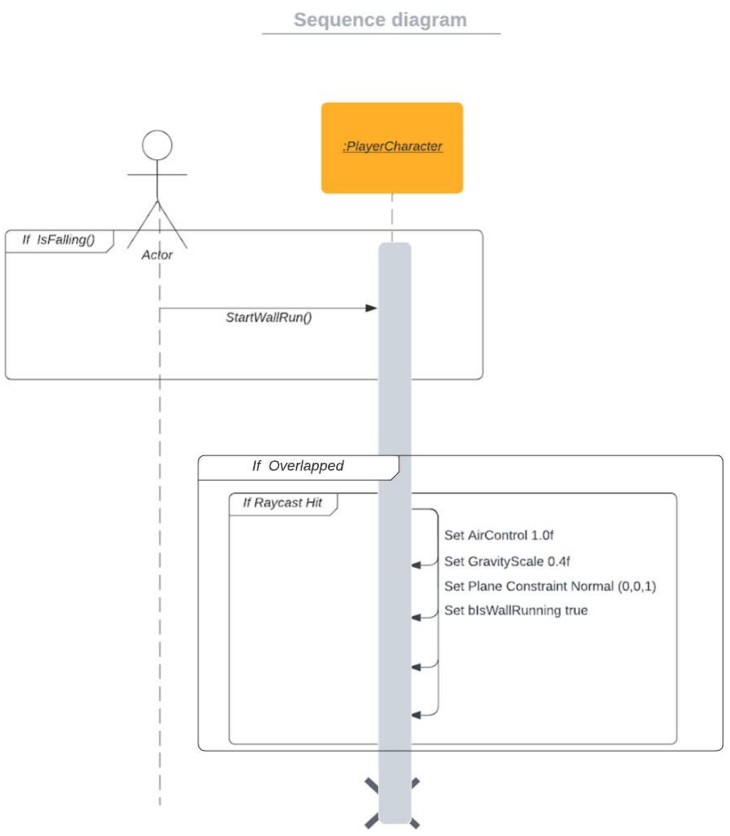
Variable	**Value**	**Description**

Raycast	Start = Actor's Location
	End = Actor's Location * 50
		Used to check for a wall on the side of the player

bOnRight	True / false depending on player's location to the wall	Used as a check variable for the position of the wall
bIsJumpOffWall	True / false depending if the player is on the wall or not	Used to make sure that the player is not in the middle of jumping off a wall when it is trying to detect the wall.
Velocity	Player's forward vector * 2000 (x, y),	Used to increase the player's speed when the player is wall running
Gravity Scale	Dynamic (1 when not wall running, 0.4 when wall running)	Used to simulate wall running with low gravity
Plane Constraint Normal	Dynamic (1 on the z value when wall running, 0 on the z value when not wall running)	Used to constraint the player when the player is wall running on the z axis.
Air Control	1.0f	Set to make the player have full control while on

		air

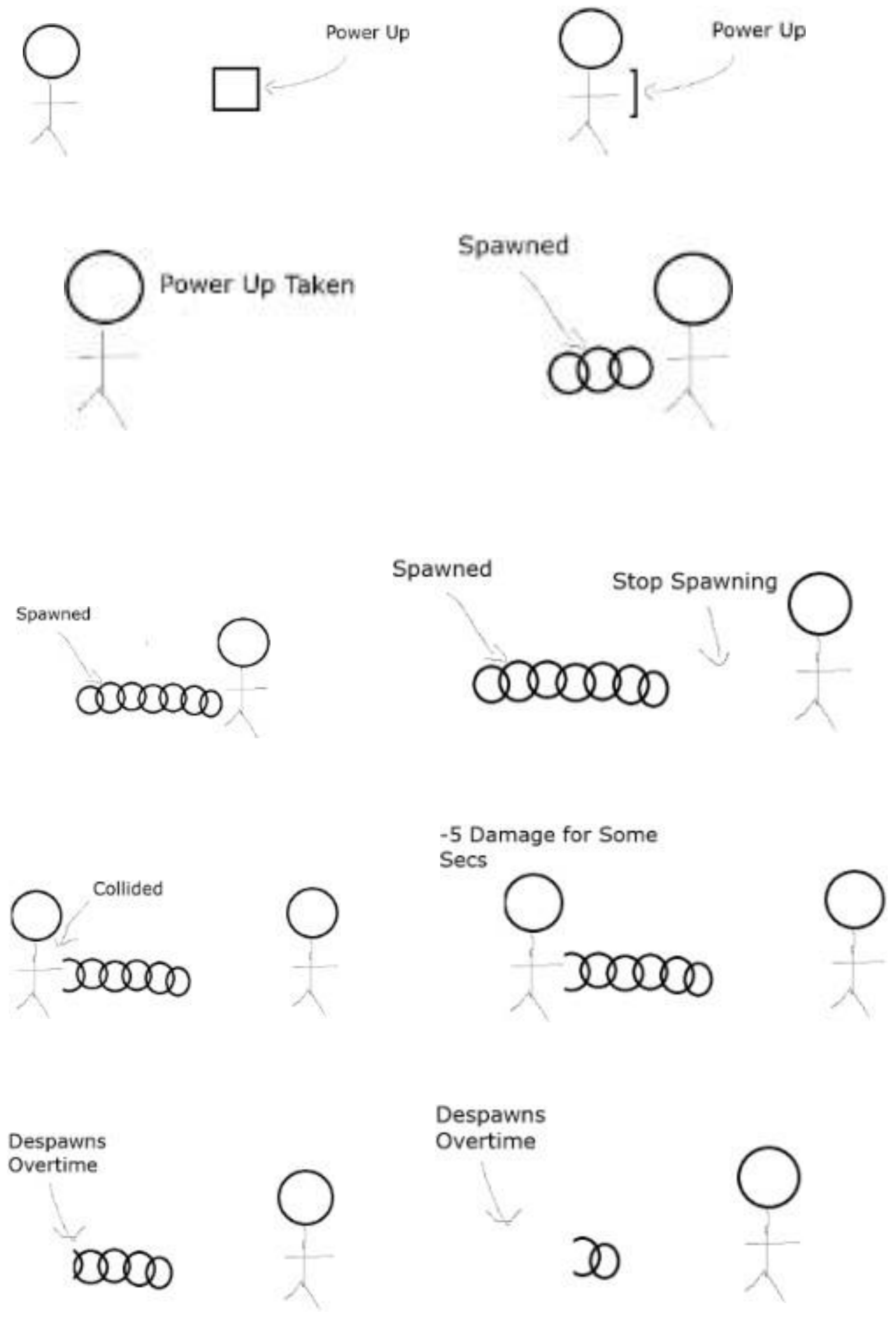
Sequence Diagram



Additional Gameplay Mechanic 2 - Poison Trail

Mechanic Overview

Another additional mechanic that I implemented is the poison trail. The poison trail (also called Collider Trail) is a powerup that the player can use. How the power up works basically is it starts as a spinning cube that spawns on the world. When the player takes it, the player will spawn a trail of poison with a looping timer of 0.01 for a couple of seconds that will be mentioned in the next section with justification. The trail will also disappear after a couple of seconds. When the player or the enemy hits the trail, it inflicts poison damage overtime for a couple of seconds. This creates a good opportunity but also encourages careful traversal through the environment.


Power Up
Power Up
Power Up Taken
Spawned
Spawned
Spawned
Stop Spawning
Collided
-5 Damage for Some Secs
Despawns Overtime
Despawns Overtime

Mechanic Description / Functionality

In this section, the details of the mechanic description of functionality will be discussed. There are 3 main classes that are being interacted with in this mechanic which are the player character class, trail collider class, and the trail power up box class.

The trail power up box class has a box mesh that can be overlapped. It also rotates with a looping timer every 0.01 seconds by 1 in the yaw. When collided, it casts the OtherActor into the player character class. Then, it checks if the cast is successful by checking if it's null or not. If it is not null, it changes the boolean variable called bHasTrailPowerUp into true and calls the function called StartTimerTrail(). It also destroys itself. This is done so that it gives an effect of the player taking the item. All of this is also replicated since the game is online multiplayer.

After it is collided it calls a looping timer called TrailSpawnHandle , it calls the SpawnTrailCollider() function every 0.01 seconds. Then, it calls the MulticastSpawnTrailCollider() function since meshes don't replicate so it needs to be called through the client and the server. In the MulticastSpawnTrailCollider, it gets the actor location, then it subtracts it with the actor forward vector times by 150 that is stored in an FVector variable called SpawnLocation. This makes sure it spawns behind the player character rather than on the player character. Then, it spawns the trail collider based on the SpawnLocation that was initialised earlier.

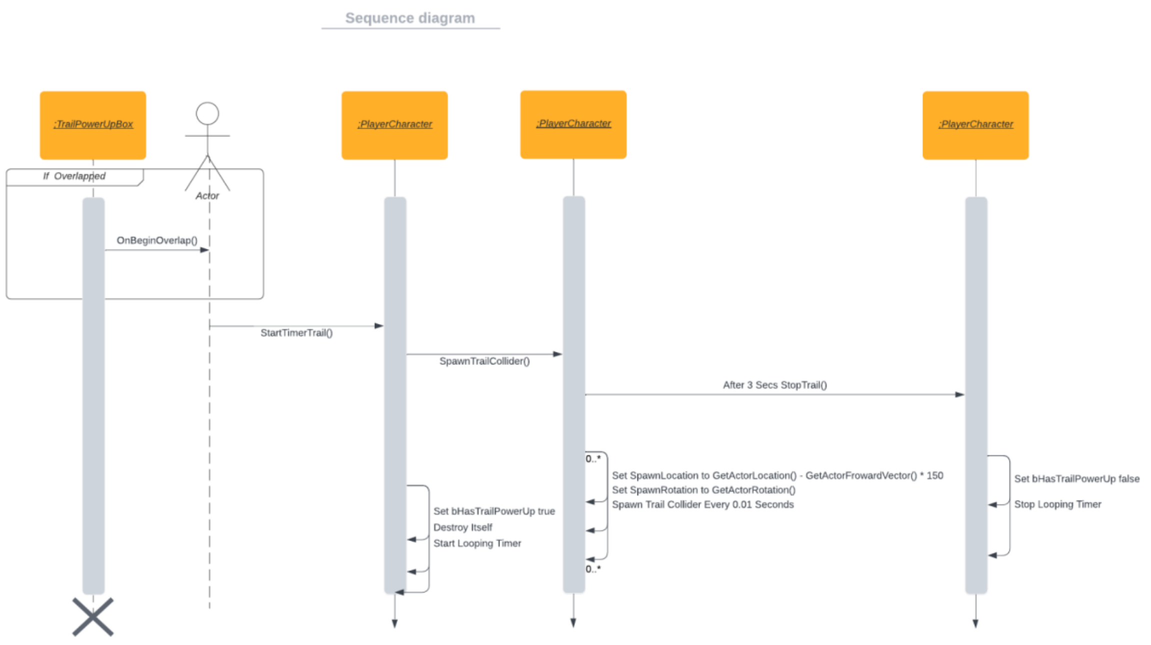
It was also mentioned that the StartTimerTrail() function was called when the player collided with the power up box. Basically what the function does it calls a server rpc called ServerStartTimerTrail(). Then, basically it starts a timer for 3 seconds that calls the function StopTrail() when it ends. In the StopTrail() function, it calls a Server RPC called ServerStopTrail() that makes the boolean variable bHasTrailPowerUp to false and also pauses the TrailSpawnHandle timer to stop the spawning of the Trail Collider.

How the trail collider works is pretty simple. The trail collider has a static mesh called TrailCollideMesh and an FTimerHandle DestrucitonTimerHandle. The collision is also enabled for overlapping with the mesh and then, it uses the same technique as the power up box, when it collides with any player, it casts the player that it collided with and then it checks if it is getting casted correctly. Then, it also checks if the character is already poisoned or not. If it is not poisoned already, set the boolean variable bIsPoisoned to true and then call the function in the player character called StartTimerPoison(). Then, the last thing is it destroys itself with Destroy().

The StartTimerPoison calls a server RPC called ServerStartTimerPoison and then it loops every 1 seconds and calls the PoisonPlayer() function. It calls a server RPC when poisoning the player, and it also deals 5 damage each time the player is poisoned using the DealDamage function that was provided in the labs. Each time the player is damaged, a float variable called PoisonTracker gets decreased by 1 each time. This happens five times since the starting number for the variable is 5. Then it checks if the PoisonTracker is less than 0, it pauses the timer so it doesn't loop again and sets bIsPoisoned to false to make it so that the player is not set as being poisoned currently.

Variable	**Value**	**Description**
Looping Rotate Timer	0.01 Seconds Looping	Used to rotate the Trail Collider Power Up Box
bHasTrailPowerUp	True / false depending on if the player has the power up or not	Used to check if the player has a power up or not
PoisonTracker	Starts as 5	Used to track the poison damage to the player

Sequence Diagram



Multiplayer

Game State & Player State

Game State

Property / Function	Description
FString GetPlayerName(APlayerState* PlayeState)	It gets the player's names in the game which are Bob and Billy. This is done to replicate the names of each player throughout the game. The use of this can be for displaying high scores with the names of each player.

Player State

Property / Function	Description
GetPlayerDevice(AController* C)	Retrieves the player's device name. This makes it so that when the players need info on their PC name, they can get it. It is also useful for developers to identify potential cheaters in games. This is done to keep all of the player's device name replicated throughout the servers.

Class Replication

Class Name	Property Name	Description
A01_BlockoutShooterCharacter	int JumpCount	This keeps track of the JumpCount of the game. This replicates it throughout the game to make sure that the amount of Jumping possible is consistent.
	FVector JumpBoost	This variable represents the amount of JumpBoost that the player is going to receive every time the player jumps. It needs to be replicated to ensure that the jumps are consistent throughout all of the players.
	Float JumpLength	This variable represents the length of the jump of each player. As mentioned before, the variable needs to be consistent in order for the player's jumps to be consistent for all players.
	Float TextOffset	This variable represents the text offset that is changed dynamically when the damage text appears on the particles. This is replicated to ensure that the offset of the floating text appears the same for all players.
	UCableComponent* GrappleRope	This variable represents the cable component used for the grapple physics constraint mechanic. This is replicated to ensure that the grapple mechanic is working perfectly and in sync with all players.
	UPhysicsConstraintComponent* PhysicsConstraint	This variable represents the physics constraint on the player. This is used to connect the grappling rope cable component to a designated object. This is replicated to make sure that all players are in sync when using the physics constraint and making sure it works properly and consistently.
	Float PoisonTracker	This variable represents the amount of times the player is poisoned overtime. This needs to be replicated to make sure that all players are poisoned in an equal and consistent manner.
	Bool bIsGrappled	This variable represents if the player already grappled or not. This needs to be replicated to make sure that all players' ability to grapple and activate the chaos physics constraint mechanic to be

		consistent throughout the game.
AttackCube	UStaticMeshComponent* Box	This variable represents the mesh of the attack cube. The variable itself needs to be replicated to make sure that the mesh is in sync with all of the players.
Balloon	UStaticMeshComponent* Balloon	This variable represents the mesh of the Balloon. The variable itself needs to be replicated to make sure that the balloon mesh stays in sync with all of the players appearance wise.
	UPhysicsConstraintComponent* PhysicsConstraint	This variable represents the physics constraint on the player. This is used to connect the balloon cable component (Rope) to a designated object. This is replicated to make sure that all players are in sync when using the physics constraint and making sure it works properly and consistently.
	UCableComponent* Rope	This variable represents the cable component used for the balloon mechanic. This is replicated to ensure that the balloon mechanic is working perfectly and in sync with all players.
Floor	UStaticMeshComponent* FloorMesh	This variable represents the mesh of the FloorMesh. The variable itself needs to be replicated to make sure that the Floor mesh stays in sync with all of the players appearance wise.
GlitchObject	UStaticMeshComponent* Object	This variable represents the mesh of the Object. The variable itself needs to be replicated to make sure that the GtlichObject mesh stays in sync with all of the players appearance wise.
TrailCollider	UStaticMeshComponent* TrailColliderMesh	This variable represents the mesh of the TrailCollider. The variable itself needs to be replicated to make sure that the TrailCollider mesh stays in sync with all of the players appearance wise.
	USphereComponent* Hitbox	This variable represents the hitbox of the TrailCollider. This needs to be replicated to make sure the collision between the player and the trail collider stays in sync between all players in the game.
TrailPowerUpBox	USphereComponent* Hitbox	This variable represents the hitbox of the TrailPowerUp Box. This needs to be replicated to make sure the collision

		between the player and the Trail Power Up box stays in sync between all players in the game.
	UStaticMeshComponent* TrailPowerUpBoxMesh	This variable represents the mesh of the TrailPowerUpBox. The variable itself needs to be replicated to make sure that the Trail Power Up mesh stays in sync with all of the players appearance wise.
Wall	UStaticMeshComponent* WallMesh	This variable represents the mesh of the Wall. The variable itself needs to be replicated to make sure that the Wall mesh stays in sync with all of the players appearance wise.
Water	UStaticMeshComponent* WaterPlaneMesh	This variable represents the mesh of the Water Plane. The variable itself needs to be replicated to make sure that the Water Plane mesh stays in sync with all of the players appearance wise.

Remote Procedure Calls

Discuss each class that has remote procedure calls implemented, outline their type and a description for their purpose.

Class Name	Function	Type	Description
A01_BlockoutShooter	ServerPoisonPlayer()	Server	This RPC is invoked on the server because the functionality is quite important. To prevent the player from adjusting how the players are poisoned and the values of the poison damage, it needs to be called from the server.
	ServerStartWallRun()	Server	This RPC is invoked on the server because I don’t want the player to be able to tinker with the code when the player is wall running. If it is not on the server, the player is able to adjust the speed, the way it works, etc which is not something that I want therefore it needs to be called from the server, it also needs to be consistent for all of the players that are present on the server.
	OffWall()	Server	This RPC is invoked on the server because I don’t want the player to be able to tinker with the code when

			the player is getting off the wall. If it is not on the server, the player is able to adjust the speed, the way it works, etc which is not something that I want therefore it needs to be called from the server, it also needs to be consistent for all of the players that are present on the server.
	ServerCallJump()	Server	This RPC is invoked on the server because I don’t want the player to be able to edit the jump values itself like height, jump boost, etc therefore it needs to be called from the server, it also needs to be consistent for all of the players that are present on the server.
	ServerResetSprint()	Server	This RPC is invoked on the server to prevent the player from editing the values of the reset sprint function since the value of the player max walk speed is present. The sprint also needs to be consistent throughout the game itself therefore it needs to be called from the server.
	ServerMaxSprint()	Server	This RPC is invoked on the server to prevent the player from editing the values of the max sprint function since the value of the max walk speed just like the reset sprint function are present in the RPC. It also needs to be consistent throughout the game itself therefore it needs to be called from the server.
	ServerStartTimerTrail()	Server	This RPC is invoked on the server to prevent the player from adjusting with the timer of the collider trail spawning. This is crucial since it can affect the power of the item and also it needs to be consistent throughout the game so it needs to be called from the server.
	ServerStartSpawnTrail()	Server	This RPC is invoked on the server to prevent the player from adjusting with the timer of the collider trail spawning. This is crucial since it can affect the power of the item and also it needs to be consistent throughout the game so it needs to be called from the server.

	ServerStopTrail()	Server	This RPC is invoked on the server to prevent the player from adjusting with the values of when the player stops trailing the poison trail mechanic. This also needs to be consistent throughout the server whether it is still trailing or not therefore it needs to be called from the server.
	ServerStartTimerPoison()	Server	This RPC is invoked on the server to prevent the player from adjusting with the timer of the collider trail spawning. This is crucial since it can affect the poison effect and also it needs to be consistent throughout the game so it needs to be called from the server.
	ServerSpawnBalloon()	Server	This RPC is invoked on the server to prevent the player from adjusting the spawn rate of the balloon and the balloon spawning overall. This is crucial since I don’t want the player to cheat and spawn as many balloons as they want through a single button press. It also needs to be consistent throughout the game therefore it needs to be called from the server.
	ServerStartGrapple()	Server	This RPC is invoked on the server to prevent the player from adjusting the values of when the player starts grappling and also where the player can grapple. It also needs to be consistent throughout the game therefore it needs to be called from the server.
	ServerStartPull()	Server	This RPC is invoked on the server to prevent the player from adjusting the strength of the pull when the player is pulling and activating the chaos grapple mechanic. It also needs to be consistent throughout the game therefore it needs to be called from the server.
	ServerStopPull()	Server	This RPC is invoked on the server to prevent the player from adjusting the strength of the pull when the player stops pulling and deactivating the chaos grapple mechanic. It also

			needs to be consistent throughout the game therefore it needs to be called from the server.
	ServerDisableInvulnerability	Server	This RPC is invoked on the server to prevent the player from adjusting the invulnerability of when it is disabled. It also needs to be consistent throughout the game therefore it needs to be called from the server.
	MulticastSpawnTrailCollider()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the meshes don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	MulticastSpawnHitParticles(int DamageAmount, FVector HitLocation)	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the particles don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	MulticastReactWall(const TArray<FBasicParticleData>& Data)	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be called on both the server and the client so all of the players can see the material change and react. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	MulticastStartPull()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the physics constraint cannot be replicated only through the server but needs to be replicated through the client and server. It is also worth mentioning that this focuses on

			some of the functionality of the game but it is not crucial if the player is able to tinker with the code since it just activates the interaction of the objects that are already called on the server.
	MulticastStopPull()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the physics constraint cannot be replicated only through the server but needs to be replicated through the client and server. It is also worth mentioning that this focuses on some of the functionality of the game but it is not crucial if the player is able to tinker with the code since it just activates the interaction of the objects that are already called on the server.
	MulticastAttachComponent(UPrimitiveComponent* Component, FVector Location, FName BoneName)	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the physics constraint cannot be replicated only through the server but needs to be replicated through the client and server. It is also worth mentioning that this focuses on some of the functionality of the game but it is not crucial if the player is able to tinker with the code since it just activates the attachment of the objects.
	MulticastDetachComponent	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the physics constraint cannot be replicated only through the server but needs to be replicated through the client and server. It is also worth mentioning that this focuses on some of the functionality of the game but it is not crucial if the player is able to tinker with the code since it just activates the attachment of the objects.
	MulticastSpawnTentacles()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the particles don't get replicated

			properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	UpdateMaterials(UMaterialInterface* NewMaterial)	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	ClientActivateSceneChange()	Client	This RPC is invoked on the Client since it is only the visual aspect that's changing and it should happen locally so each player has total control over it.
AttackCube	HurtPlayer(UPrimitiveComponent* HitComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, FVector NormalImpulse, const FHitResult& Hit)	Server	This RPC is invoked on the server since it is a crucial part of the game. It hurts the player and I don't want it so that the player can tinker with the code and adjust the damage as they want it to be. It also needs to be consistent throughout the game therefore it needs to be called from the server.
Balloon	MulticastAttachComponent(UPrimitiveComponent* Component, FVector Location, FName BoneName)	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the physics constraint cannot be replicated only through the server but needs to be replicated through the client and server. It is also worth mentioning that this focuses on some of the functionality of the game but it is not crucial if the player is able to tinker with the code since it just activates the attachment of the objects.
GlitchObject	HurtPlayer(UPrimitiveComponent* HitComponent, AActor* OtherActor, UPrimitiveComponent* OtherComp, FVector	Server	This RPC is invoked on the server since it is a crucial part of the game. It hurts the player and I don't want it so that the player can tinker with the code and adjust the damage as they want it to be. It also needs to be

	NormalImpulse, const FHitResult& Hit)		consistent throughout the game therefore it needs to be called from the server.
	UpdateMaterials()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	ResetMaterials()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
Water	MulticastUpdateMaterials()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.
	MulticastResetMaterials()	NetMulticast	This RPC is invoked on both the server and the client using NetMulticast. This is due to the fact that the materials don't get replicated properly therefore it needs to be spawned within both client and server. It is also worth mentioning that this focuses on the visual aspect of the game, not the crucial component.

Physics Constraint 1 - Balloon

Overview of Interaction

The first physics constraint interaction that I have within the game is the Balloon. The balloon attaches to the Attack Cubes that are present within the world and brings them up. After that, it also disappears overtime, releasing the Attack Cubes to the ground after 10 seconds. This creates an opportunity for the player to make a makeshift trap to the enemies by making it fall from above and hitting the player. Since the Attack Cube will have a specific amount of velocity, it will damage the player. This will add a sense of unpredictability of the level as the player attacks each other. By making it a trap, the players will need to pay more attention to the level, especially above since balloons that have been setup can just fall anytime. With a mixture of movement and items, this can create more chaos into the gameplay and improve the immersion and fun aspect of the game itself.

Interaction Description

How the Interaction Works

Overall, the way the interaction works is very simple. It uses a separate class called Balloon that is derived from the actor class. The balloon class then contains a static mesh, cable component, and a physics constraint. The static mesh simulates the balloon itself and the cable component simulates the rope that the balloon is attached to. The balloon goes up with a looping timer by 100000 on the Z value added force to simulate the balloon going up carrying the object that it will be attached to.

Everytime the balloon is spawned, a timer also starts to destroy the balloon after 10 seconds. When the player presses the “B” key, the player is able to spawn the balloon and attach it to the object that the player is looking at based on the camera’s raycast. The camera’s raycast starts on the camera and ends at the player’s camera’s forward vector multiplied by 10000 plus the start vector. When it hits, it spawns the balloon and automatically attaches it to the object using NetMulticast RPC. The physics constraint is adjusted through code dynamically whenever the player spawns the balloon. The physics constraint connects one part to the balloon and the other part to the Attack Cube specifically and it won’t spawn if it is raycasted to other objects. For now, it connects to the 0,0,0 of the object but in the future it's planning to change to the gaze of the player. A linear limit also exists with the value of 300 to prevent the balloon to just pull the cable component with no limit also with a cable length of 100.

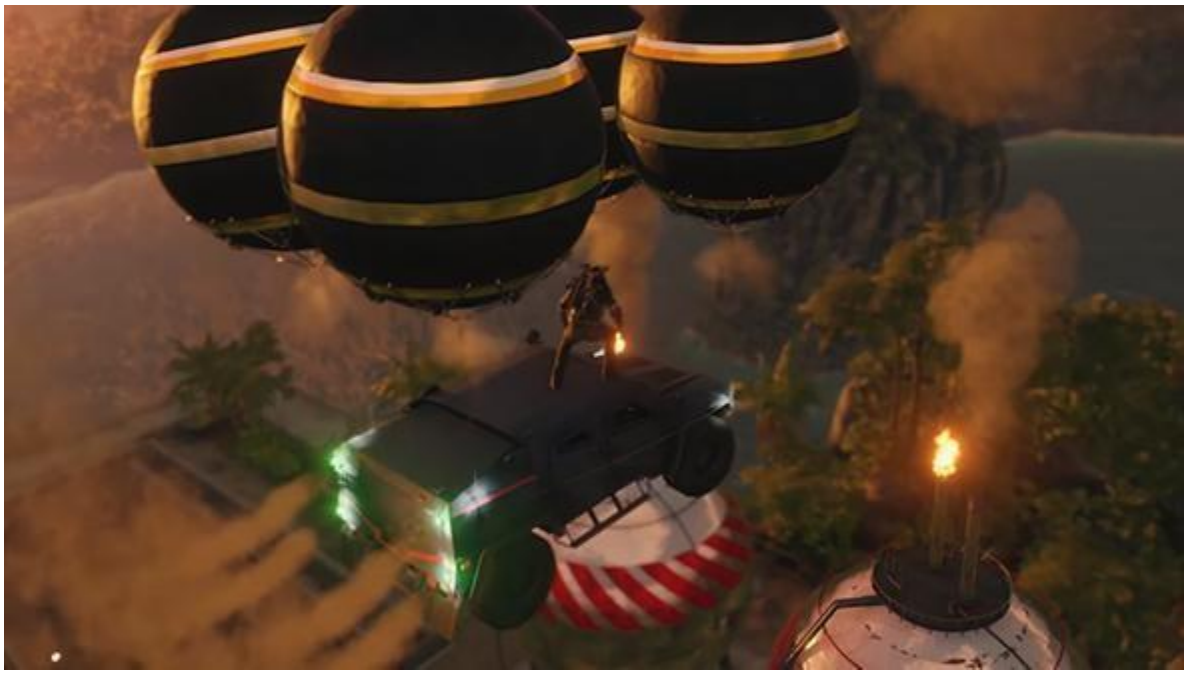
The cubes that the player can make a makeshift trap with also have a velocity detection to see if it can damage the player or not which are more than or equals to 100 on X, more than or equals to 10 on Y, more than or equals to 10 on Z.

Inspiration / Reference Images

This Physics Constraint Mechanic is inspired by some mechanics from other games. These are the screenshots.

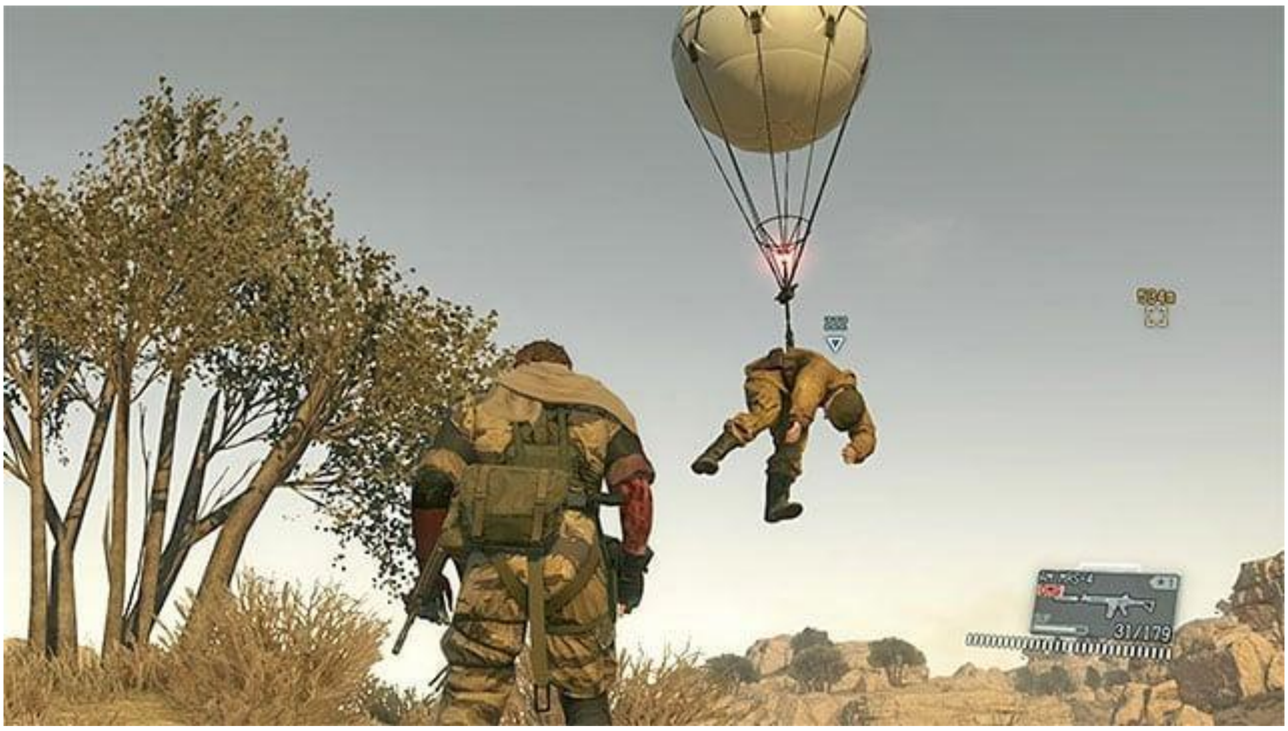
Source : https://www.allthingssmash.com/fighter-detail.php?char_id=10
https://www.allthingssmash.com/fighter-detail.php?char_id=71

These 2 characters are animal crossing characters that star in the super smash bros games. These 2 characters utilise the balloon as a form of recovery to get back to the stage. I like the way it works and even though it works differently than my mechanic, it still inspires the balloon mechanic in a way. The way it works in the game is that the player is getting carried by the balloon to safety but in my game, I wanted it to work on objects instead of players.



Source : https://www.pcgamesn.com/just-cause-4/just-cause-4-grappling-hook

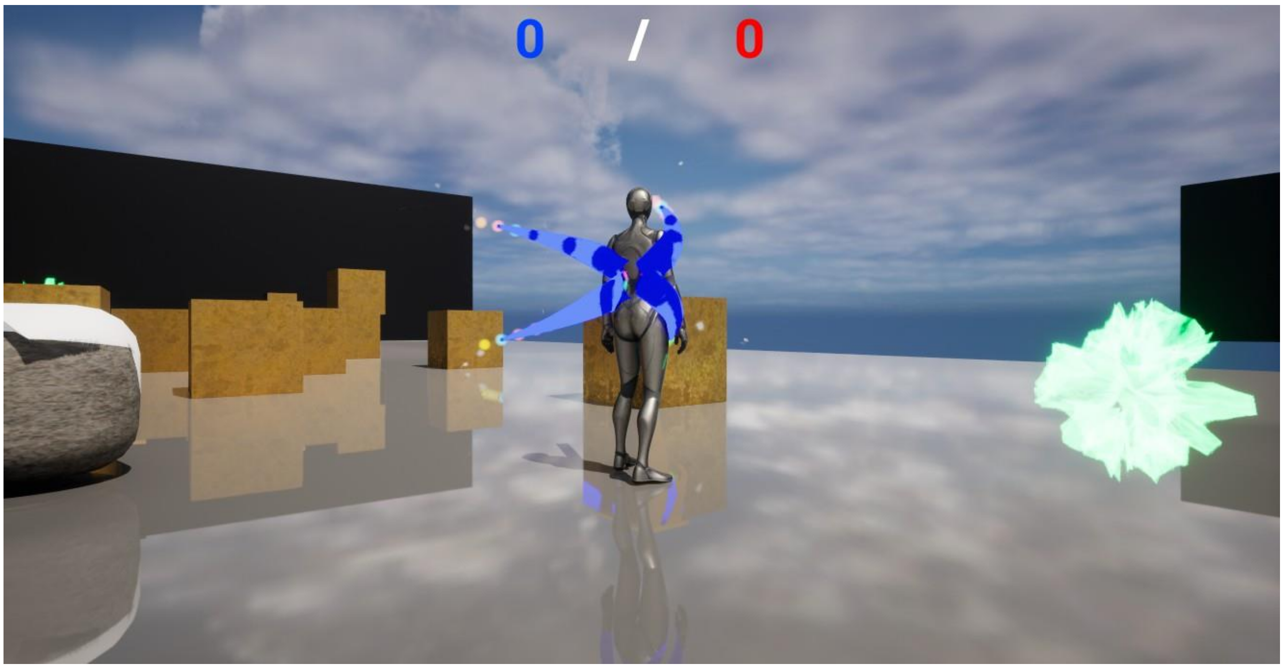
This picture shows a display of a balloon tethering mechanic from Just Cause 4. I like the idea of attaching objects into a balloon and I thought it was quite comedic, therefore I took some inspiration on the tethering part from this game.



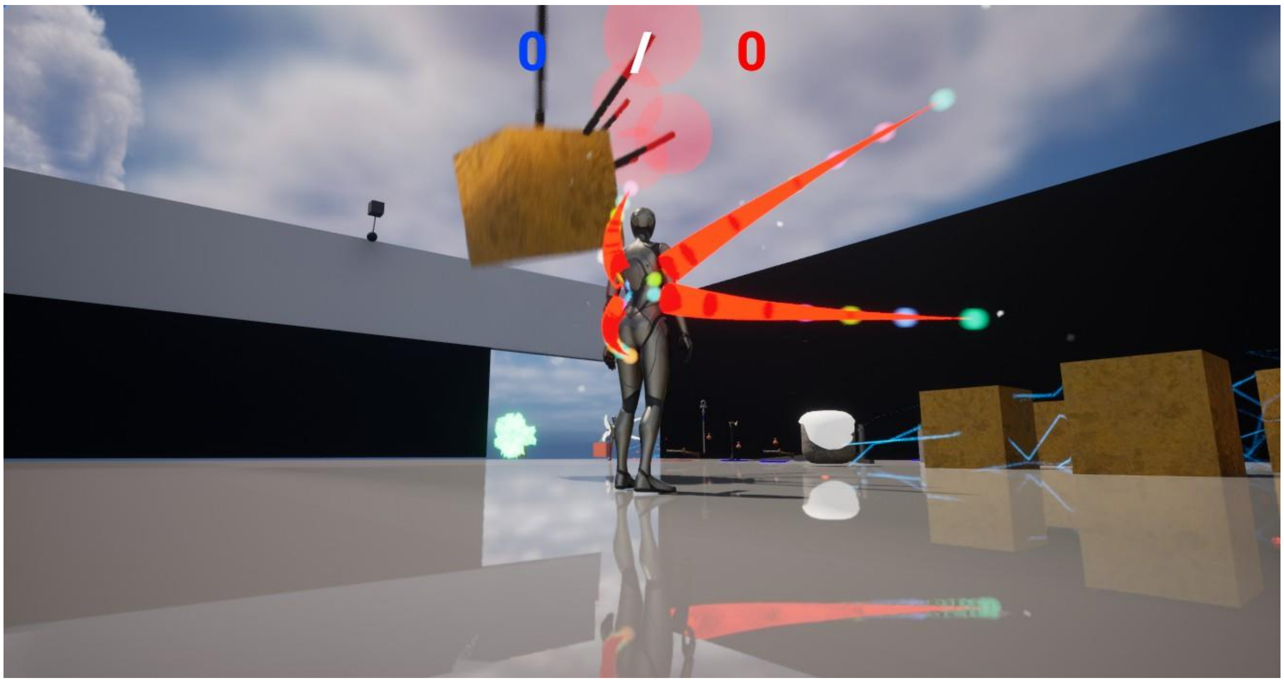
Source : https://www.gamepressure.com/mgs5thephantompain/how-does-the-fulton-work/z17b26

This picture also inspires the balloon attaching mechanic in my game. This is from the Metal Gear Solid series and it gives the player the ability to attach a balloon to objects that it has already captured. I added a twist by making the balloon in my game a makeshift trap that the player can use instead of just a capture mechanic.

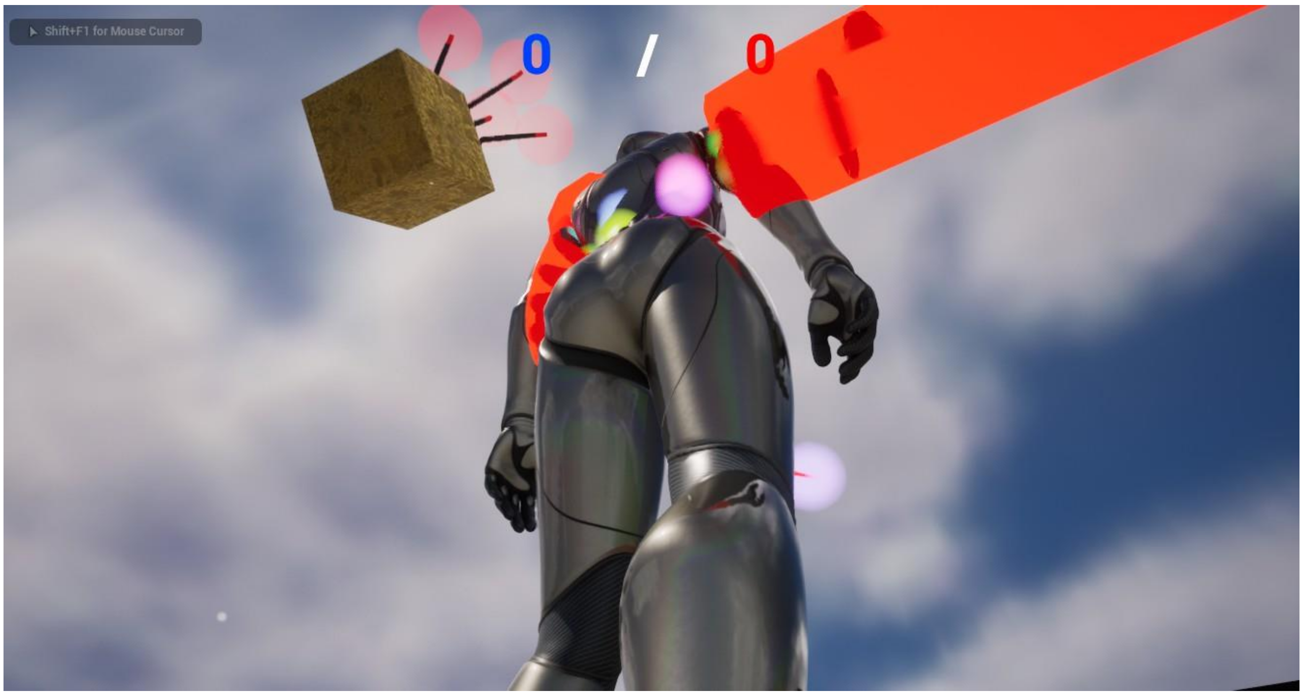
In-Engine Screenshots



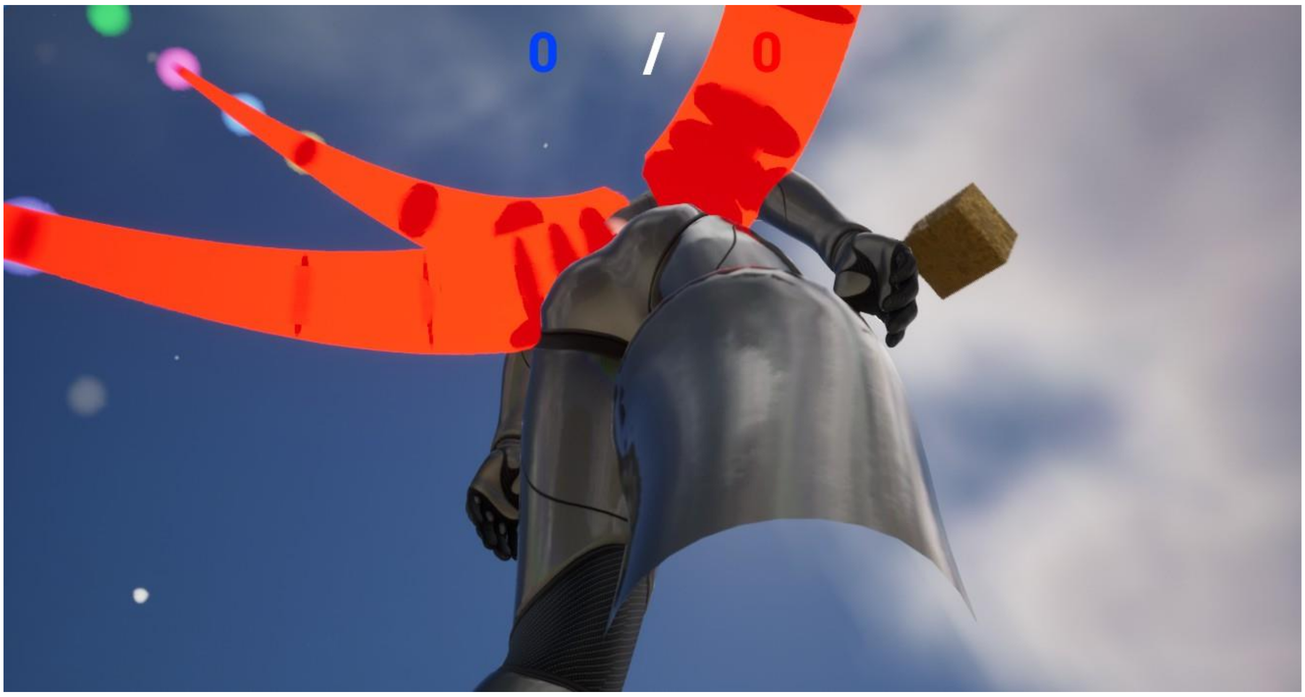
In this picture, we can see the player is looking at the box that the player can attach the balloon into.



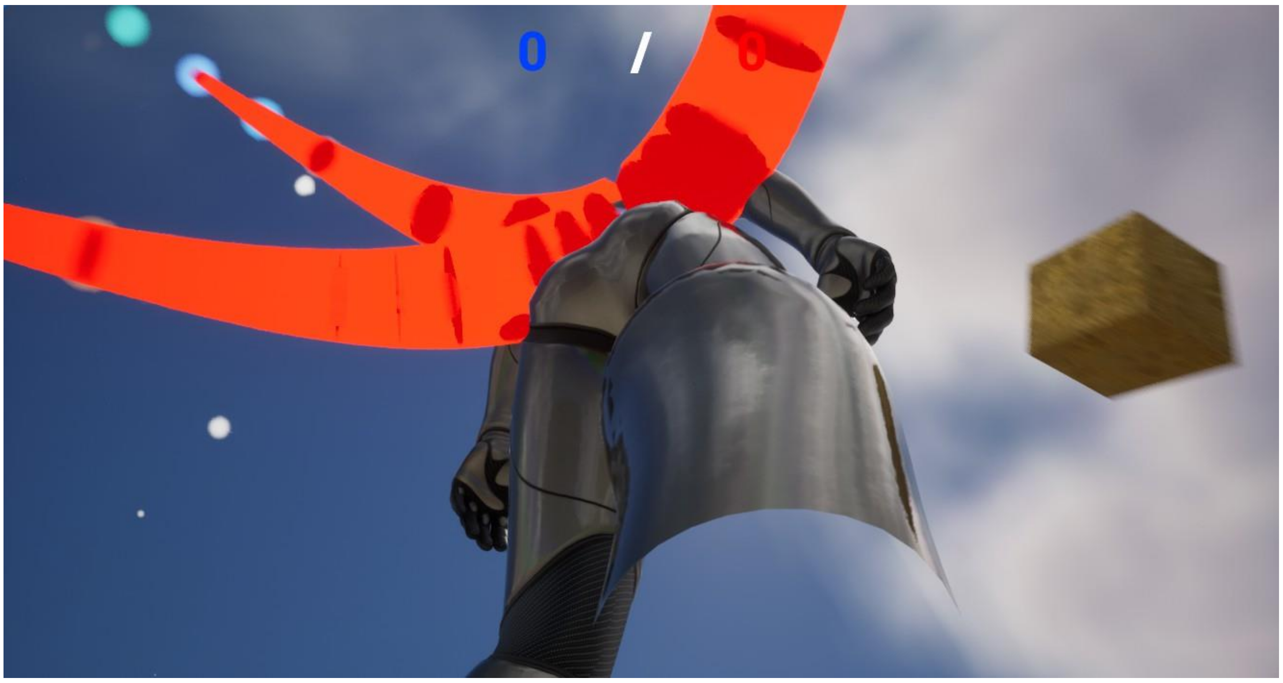
In this picture, we can see the balloon getting attached to the box and being floated up towards the sky.



This picture shows the balloon continuing to float away into the sky.



The picture shows the balloon disappeared after 10 seconds of runtime.



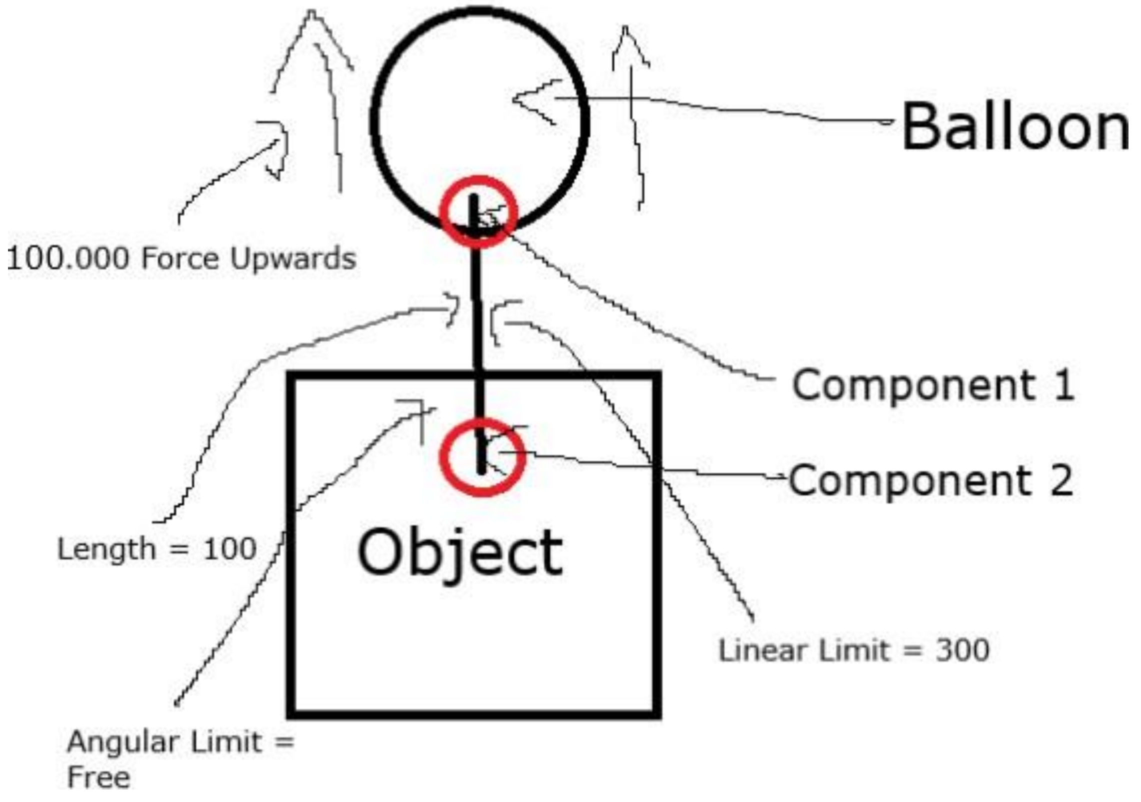
The picture shows the object that the balloons were attached to, falling to the ground since all of the balloons are gone.

Properties and Values

Property	Description of Purpose	Value
Linear Limit	The limit on the amount of linear interaction the physics constraint can sustain. This is done to limit the cable component and balloon with the object.	300
Angular Limit	The angular limit is set to free since the rotation of the object doesn't matter in the context of how the physics constraint works.	Free
Component Name 1	This component name ensures it attaches to the balloon that spawns with it.	"Balloon"
Component Name 2	This component name ensures it attaches to the object that the player's raycast hit which is the AttackCube	"Object's Name"

Balloon Force	This property is added to the balloon as a force to propel the balloon up, giving the illusion that it is a balloon going up to the sky.	FVector(0,0,100000.0)
DestructionTimer	This property is the amount of seconds the balloon will disappear after.	10 Seconds
Rope (Cable Length)	This component is the length of the cable component that is attached to the balloon and the object.	100.0

Diagram of Interaction



The interaction between the balloon and the object (Attack Cube) is fairly simple. It starts attaching to the object when it is spawned, marking it as component 2 with a linear limit 300 and free angular limit. It also pulls the object upwards by 100.000 force on the Z axis. The physics constraint connects the interaction between the balloon and the object itself.

Physics Constraint 2 - Chaos Mace

Overview of Interaction

This interaction was supposed to be a grapple interaction where the player grapples into an object and pulls it towards the player. However, I have decided to make it a chaos mace and integrate it into the gameplay by making the Glitch Objects that are present on the level into a weapon. By adding the Chaos Mace, it will add unpredictability and chaos into the gameplay which all party games should have. This will add a sense of immersion and strategy of using the items and arsenal that are provided within the game and provide a more interesting and interactive interaction between the players and the enemies.

Interaction Description

How the Interaction Works

The way it works is very simple. Basically, on the player character component, a physics constraint and a cable component is added. It uses a raycast of the player's camera when the player right clicks and when the raycast hits a Glitch Object, the cable component that is called Grapple Rope is attached to the object using the physics constraint.

Component 1 of the physics constraint is attached to a sphere component called Grapple Sphere that is located on the head of the player. Component 2 of the physics constraint is attached to the object that the raycast hits which could only work if it's a Glitch Object. It also turns bIsGrappled into true to ensure that the player is grappling into something before triggering the Chaos Mace mechanic.

When the player attaches the cable component into the Glitch Object, It attaches the other end intot he Glitch Object and changes the colour from green to blue. It also changes the movement of the Glitch Object shader making it more erratic. Then it updates the hologram material that was done in the labs which also activates Invulnerability. It also resets a timer called ChaosMaceTimerHandler and also starts a timer with the same handler to trigger a MulticastDetachComponent in 5 seconds. '

In the MulticastDetachComponent function, it resets the shader of the Glitch Object to the default one and then it detaches the end of the cable component by attaching it to the GrappleSphere which is its source itself. It also resets bIsGrappled so the player can grapple again and it disables Invulnerability. It also stops the pulling of the physics constraint. The changes that occur are the physics constraint's Linear PositionDrive and the LinearVelocityDrive are all set to false on the X, Y, and Z axis

On the BeginPlay function, the physics constraint's linear drive params are set to 1000.0f on position strength, 100.0f on velocity strength and 0 on InForceLimit. Then, it also disables the collision on the physics constraint. It is worth mentioning that these are high numbers but in order to make it have the chaos movement, it needs to have a high number. When the player holds the "Q" button, chaos will start and the object that the grapple rope is attached to will start acting in chaos and act as a chaos mace.

The changes that occur are the physics constraint linear position drive will be set to true on X, Y, and Z. It also sets the linear position target to the component location of the grapple sphere that is initialised. The Linear Velocity Drive will also be set to true on X, Y, and Z and the Linear Velocity Target will also be the grapplesphere's component location. These changes are made in a NetMulticast RPC that is called from a Server RPC. When the object that it attaches to reaches a specific velocity which is more than or equals to 100 on X, more than or equals to 10 on Y, more than or equals to 10 on Z, it will damage the player when it hits the player by 10 damage.

When the player stops holding the "Q" button, the chaos stops. The changes that occur are the physics constraint's Linear PositionDrive and the LinearVelocityDrive are all set to false on the X, Y, and Z axis. The position will also be set to the GrappleSphere Component just in case for both LinearPositionTarget and LinearVelocityTarget.

Inspiration / Reference Images

This Physics Constraint Mechanic is inspired by some media rather than game mechanics of other games since it is quite unique. Here are some of the references :

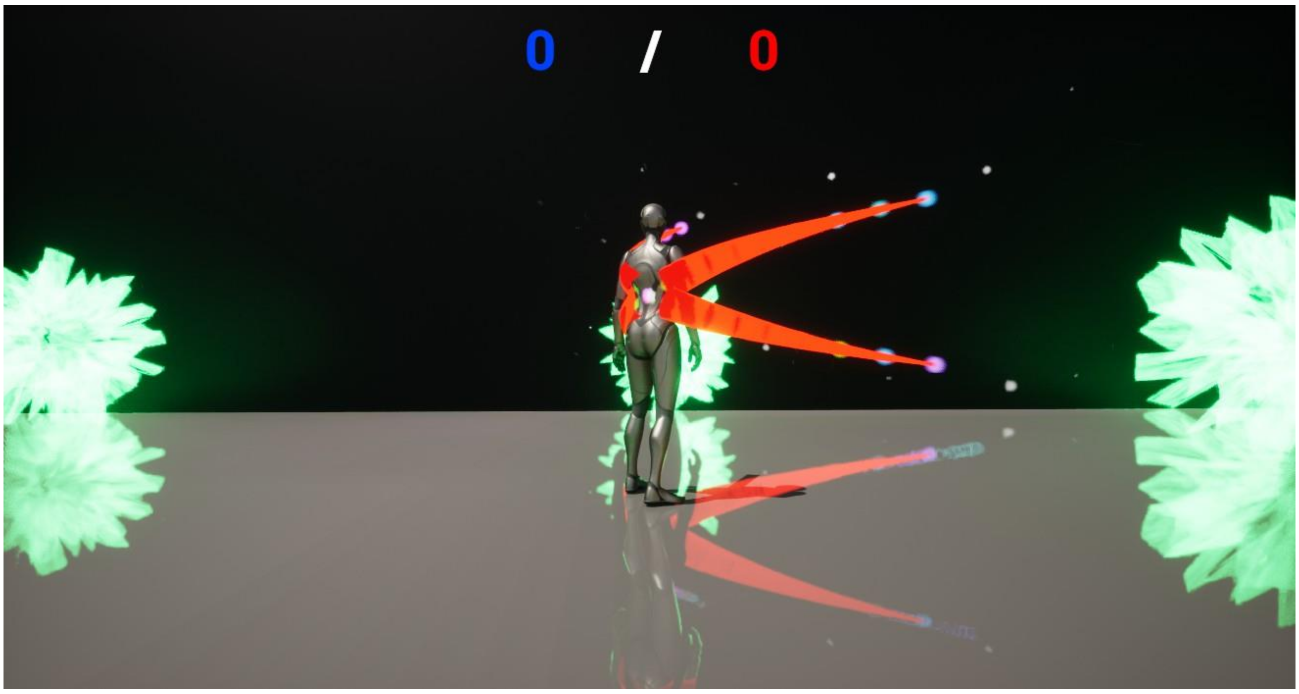
Source : https://www.cgtrader.com/3d-models/military/melee/flail-mace

I was inspired by the concept of mace in video games. I feel like maces are quite underrated and underutilised within video games as a weapon therefore, I decided to implement them. I also added a twist by not making it necessarily just a mace but making it act like it with a different theme which will be discussed in the next reference.

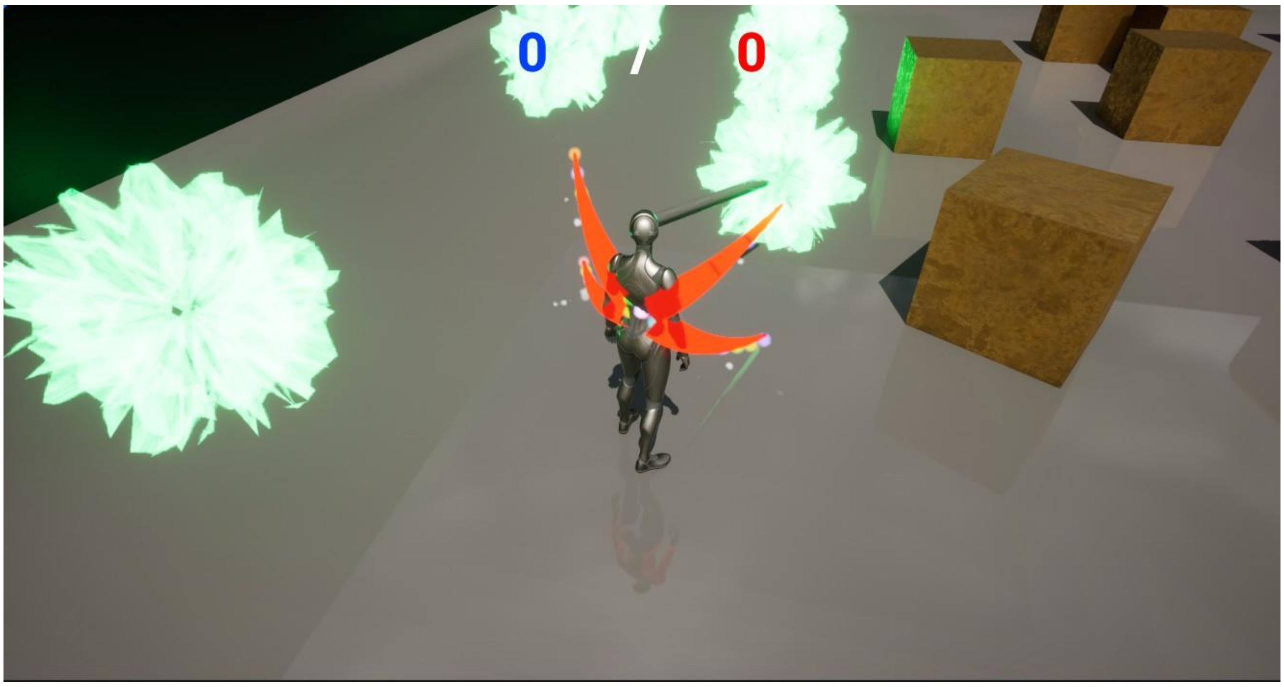
Source : https://www.ign.com/articles/how-to-watch-the-matrix-movies-in-chronological-order

I was inspired by the concept of the matrix especially in video games. The matrix inspires me to look at the concept of adding intentional bugs to video games. Therefore, when creating the chaos mace, I made it look like its buggy even but instead of it being just a buggy feature in the game, I made it so that it's a weapon that the player can use. This is heavily inspired by the matrix and how I can think outside the box on making video games.

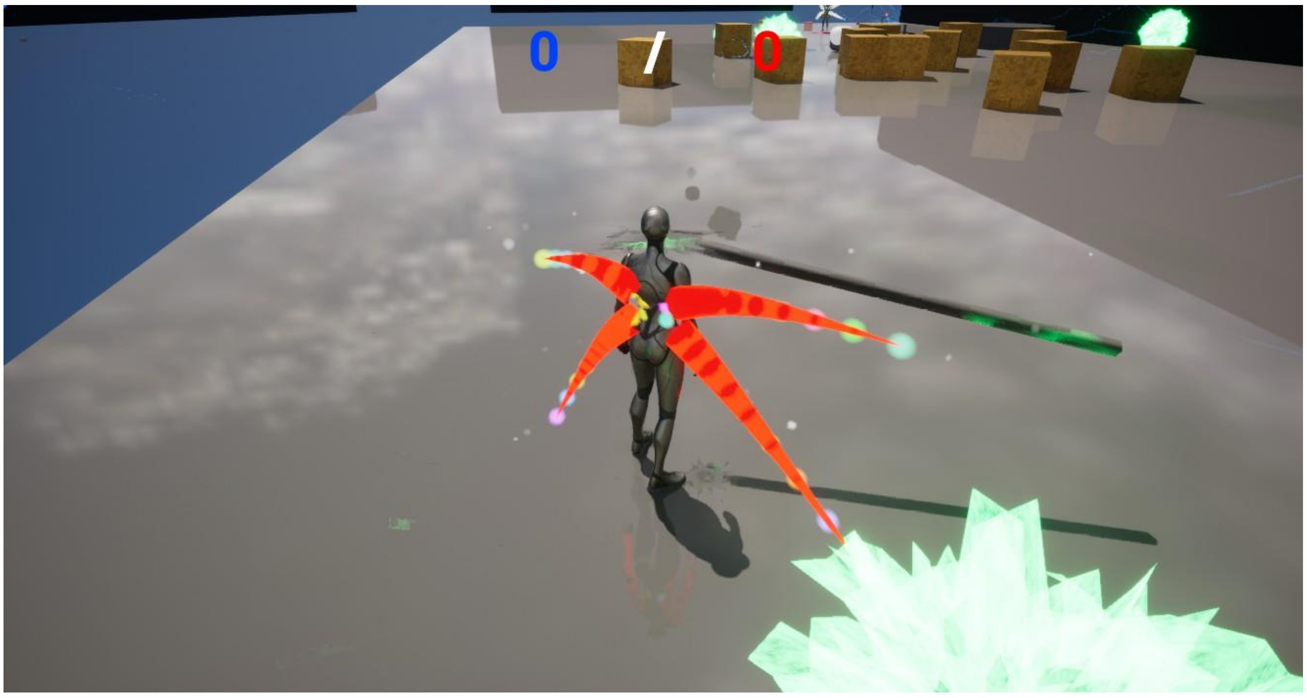
In-Engine Screenshots



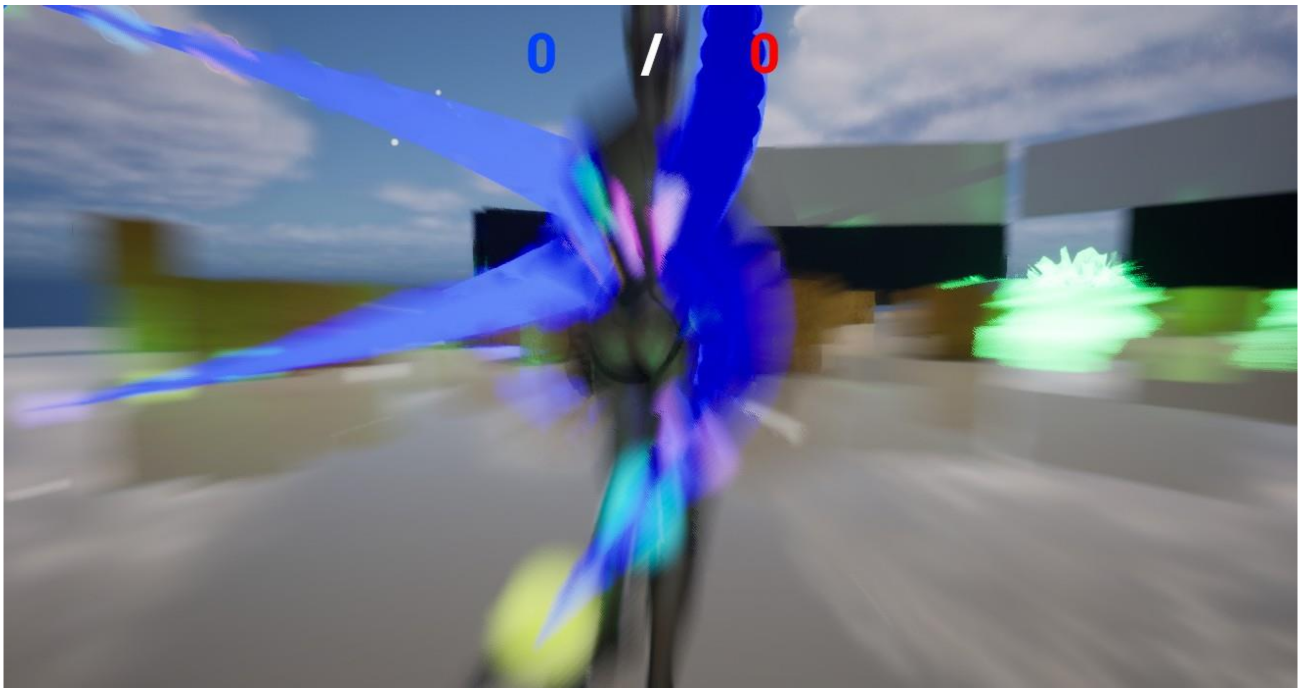
This picture shows the player standing still looking at the chaos object that the player can attach the chaos grapple to.



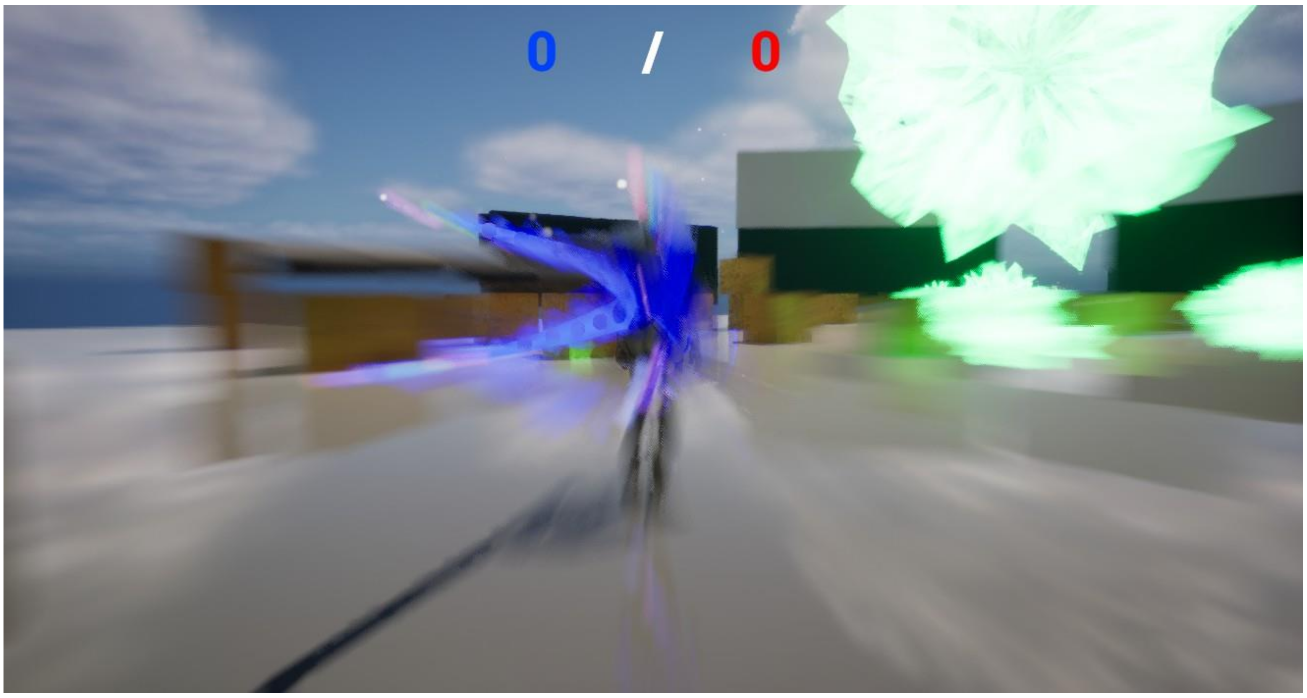
After the player presses right click towards the object, the player's grapple rope is attached to the object.



This picture demonstrates when the player holds Q, the chaos mace mechanic starts and it's getting flung around the player.



This picture demonstrates the speed and impact the chaos mace has to the gameplay. It is so fast that it is hard to capture with just a mere screenshot.



This is another screenshot showcasing the speed of the chaos mace when it is surrounding the player.

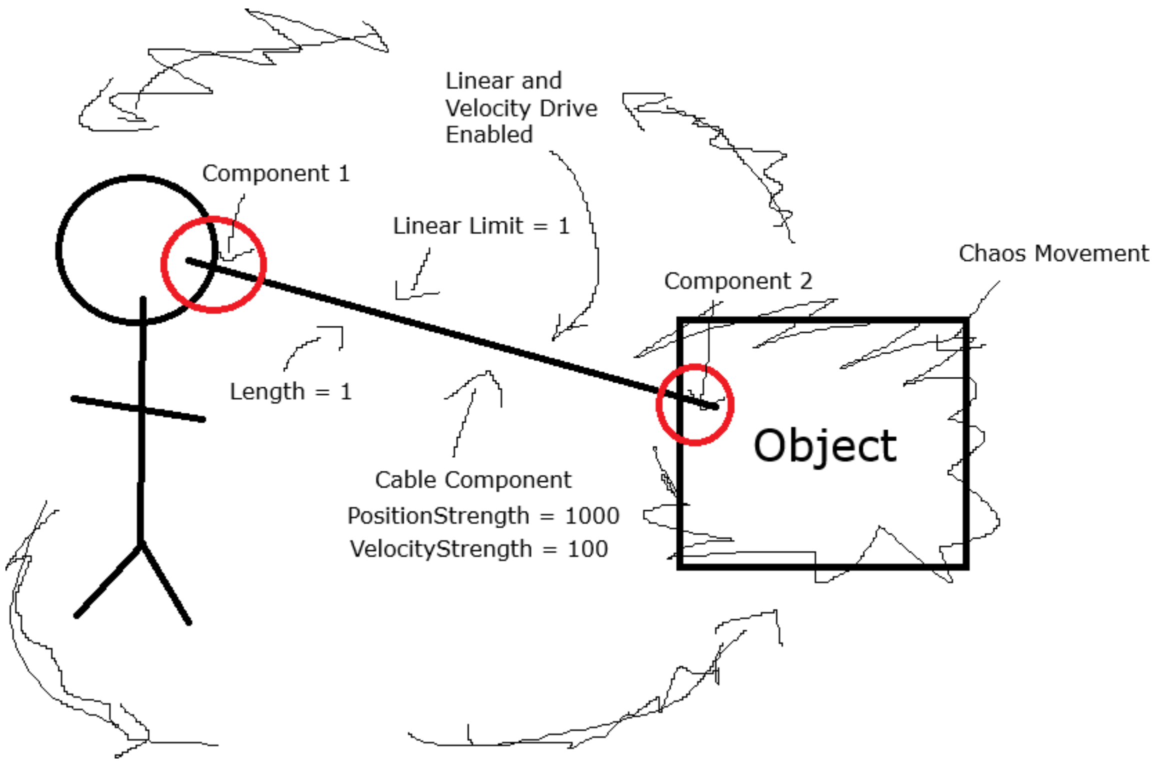
Properties and Values

Property	Description of Purpose	Value

LinearPositionDrive	Enables or disables the LinearPositionDrive of the physics constraint.	When Active = True for X,Y, and Z. When Inactive = false for X, Y, and Z
LinearVelocityDrive	Enables or disables the LinearVelocityDrive of the physics constraint.	When Active = True for X,Y, and Z. When Inactive = false for X, Y, and Z
LinearPositionTarget	Sets the linear position target of the linear motor of the physics constraint.	GrappleSphere->GetComponentLocation()
LinearVelocityTarget	Sets the linear velocity target of the linear motor of the physics constraint	GrappleSphere->GetComponentLocation()
PositionStrength	Sets the position strength of the linear motor.	1000.0f
VelocityStrength	Sets the velocity strength of the linear motor.	100.0f
DisableCollision	Enables or disables collision of the PhysicsConstraint	true
Linear Limit	The limit on the amount of linear interaction the physics constraint can sustain. This is done to limit the cable component and the object that it is connected with.	1.0
Angular Limit	The angular limit is set to free since the rotation of the object doesn't matter in the context of how the physics constraint works.	Free
Component Name 1	This component name ensures it attaches to the GrappleSphere component in the Player component.	"GrappleSphere"
Component Name 2	This component name ensures it attaches to the object that the player's raycast hit.	"Object's Name"

GrappleRope (Cable Length)	This component is the length of the cable component that is attached to the balloon and the object.	1.0

Diagram of Interaction



The cable component is connected to the grapple sphere that is located on the player’s head. This diagram shows how it works when it is triggered and the chaos mace is active. Basically, the position strength and the velocity strength is activated with the values that are shown. Then, the object goes everywhere in a chaotic manner.

Advanced Niagara Particle Effect

Niagara Particle Effect - Hit

Overview of Effect

The hit particle is a very simple particle that is combined with complex visuals of blood. This particle displays the damage that is inflicted to the players and also the blood that comes out of the damage numbers. The damage numbers change dynamically based on how much damage is inflicted to each player. The impact on this to the gameplay is quite significant.

This particle effect can tell the player how much damage the player has done to the enemy or the player itself. This increases the immersivity of the gameplay experience of the player and the enemies itself. The blood as well increases the visual queue of when the player gets damaged or damaged. This makes it very clear of when the player is damaging somebody or getting damaged with the amount of damage that was shown.

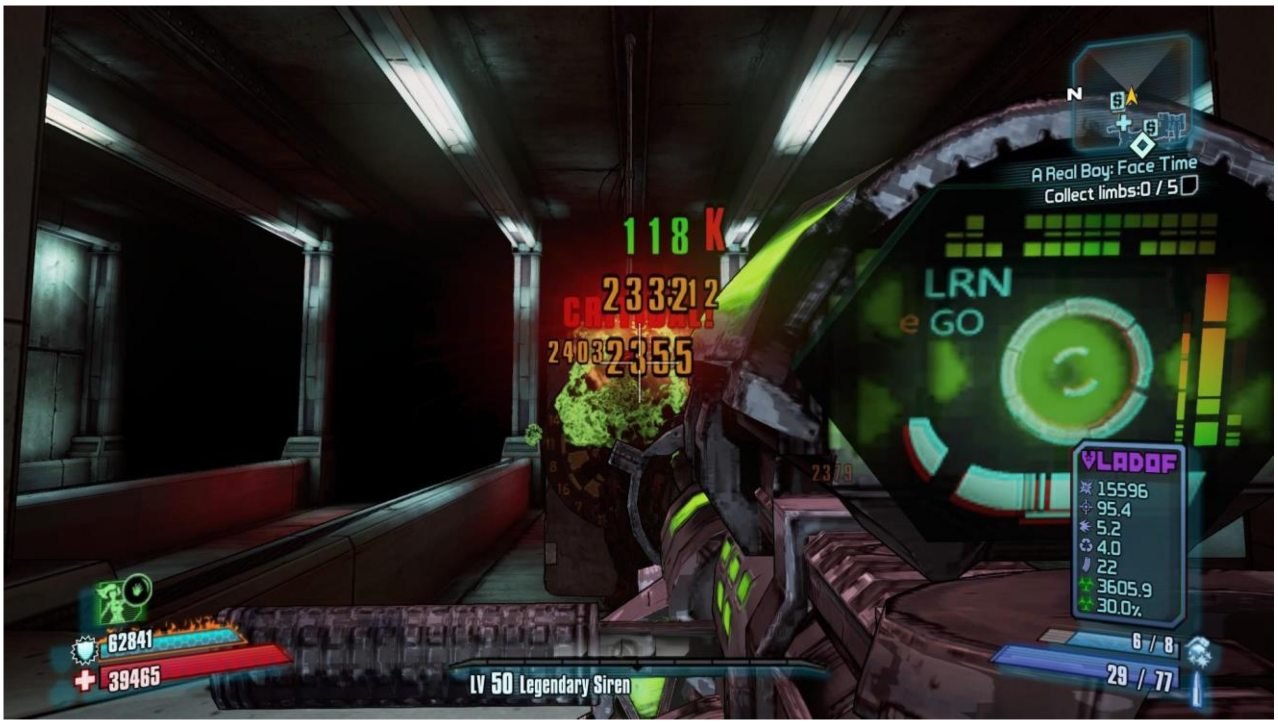
The colour of the blood is red with a white colour on the damage text that appears and fades away over time. The red coloured is due to the fact that blood is usually red coloured and the aspect of the colour red is aesthetically pleasing compared to some other colours. The white colour on the damage text is due to the accessibility of the game itself. By making it a glowing white colour, it provides ease of readability of the damage text when the players are interacting with it.

Effect Description

The effect itself is widely used in video games, especially shooters. In shooters, damage numbers usually appear after the players shoot at the enemies. This is a good indicator and makes it clear for the player to see how much damage they do to the enemy. The blood effect is also fairly common to use in games generally such as the Borderlands series, fortnite, etc. However, the way my blood effect works is that instead of it being regular blood coming out of a player, it instead blood effects raining down on the player. The damage numbers also have the colour white which is easy to read for everyone.

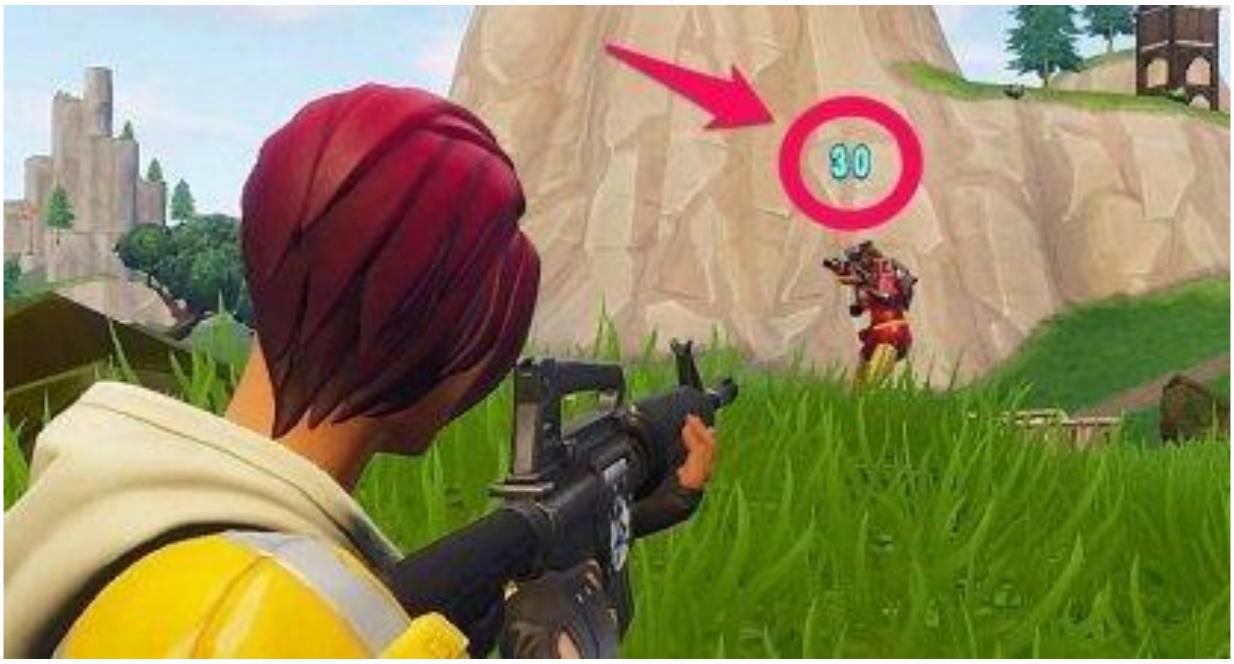
Inspiration / Reference Images:

This Advanced Particle Effect is inspired by some video games as mentioned before. Here are some reference images from the games that I was inspired by :



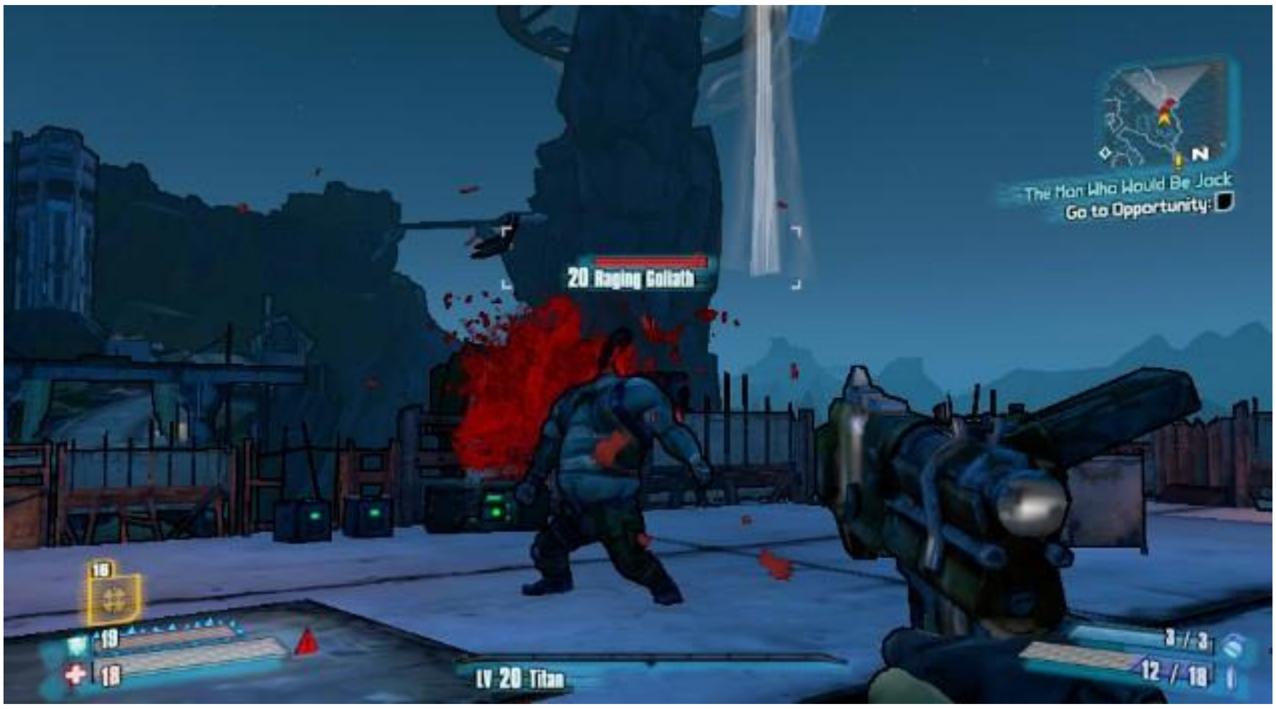
Source :
https://gaming.stackexchange.com/questions/98684/why-dont-these-e-tech-snipers-seem-as-awesome-as-the-stats-indicate

This image is from the Borderlands Series, specifically Borderlands 2. I was inspired by the simplicity of how the damage numbers react when the player shoots at an enemy. The damage numbers appear and disappear on the screen indicating how much damage the player does.



Source : https://gamewith.net/fortnite/article/show/182

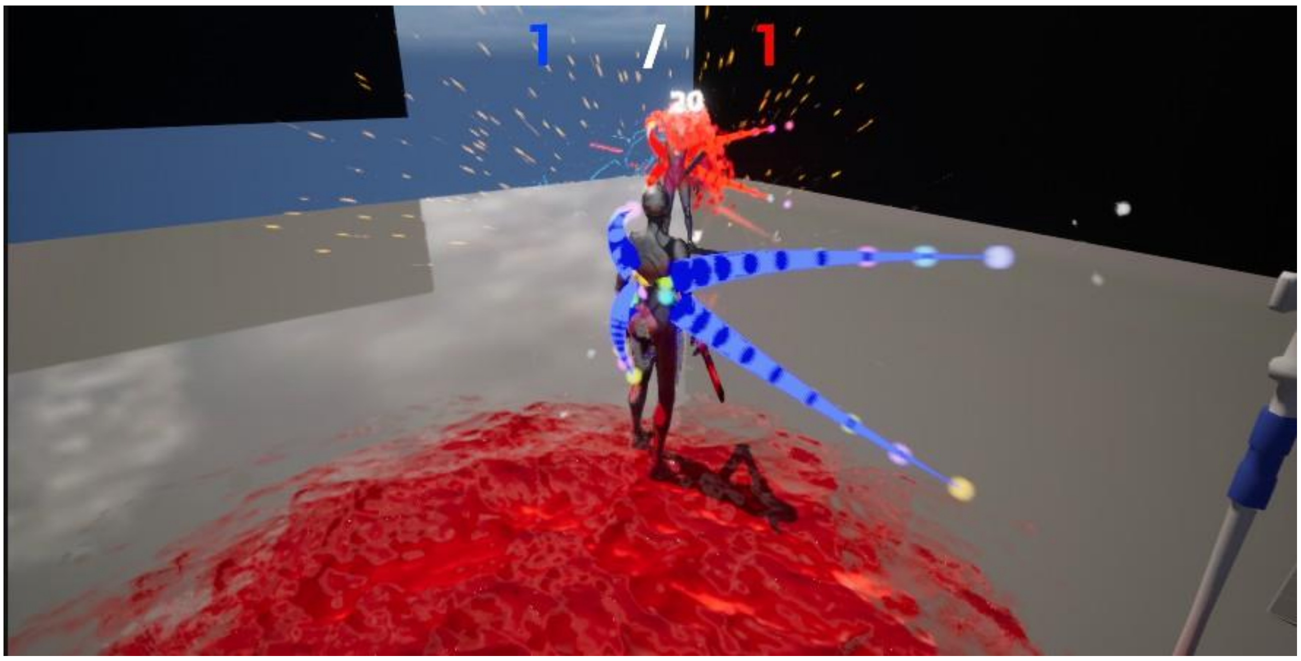
This image is from the game Fortnite. The game itself is a very popular shooter with various damage numbers that appear based on the character's shield or not. This inspires me to create my own floating damage particle as well.



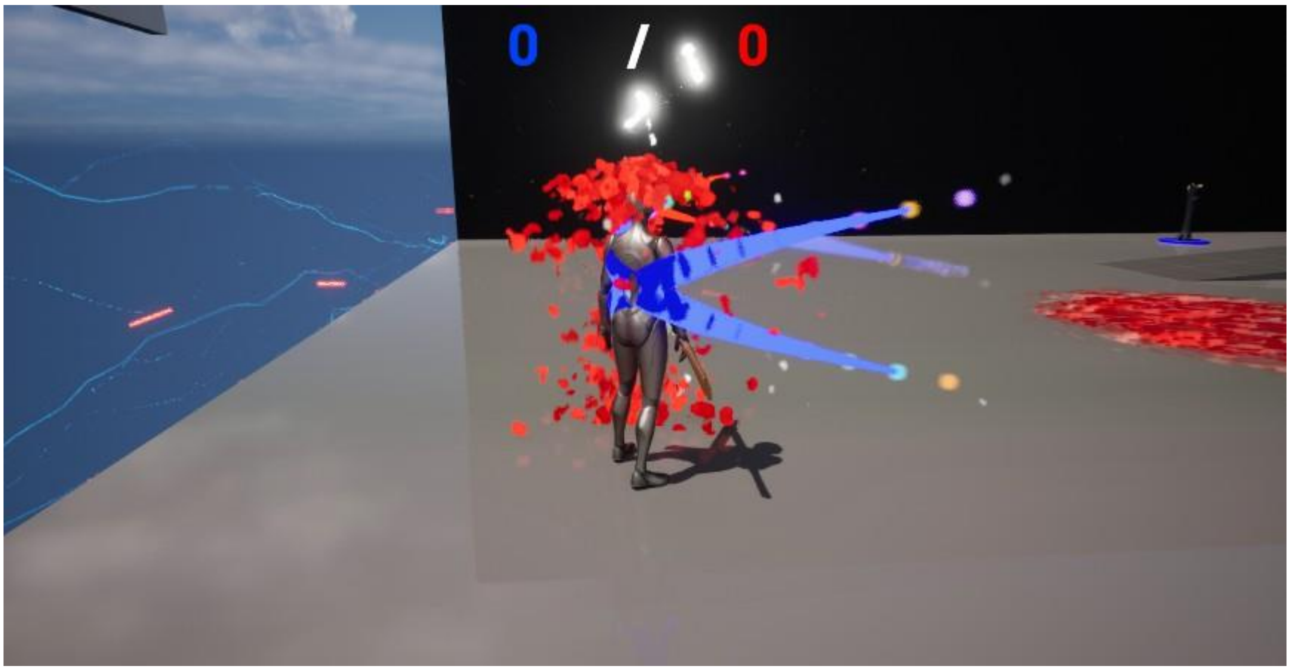
Source : https://www.pcmag.com/reviews/borderlands-2-for-playstation-vita

Going back to Borderlands again, I was inspired by how the blood works in the game. Even though I didn't necessarily implement it like the way it works in Borderlands, I was still inspired by how the texture of it works overall.

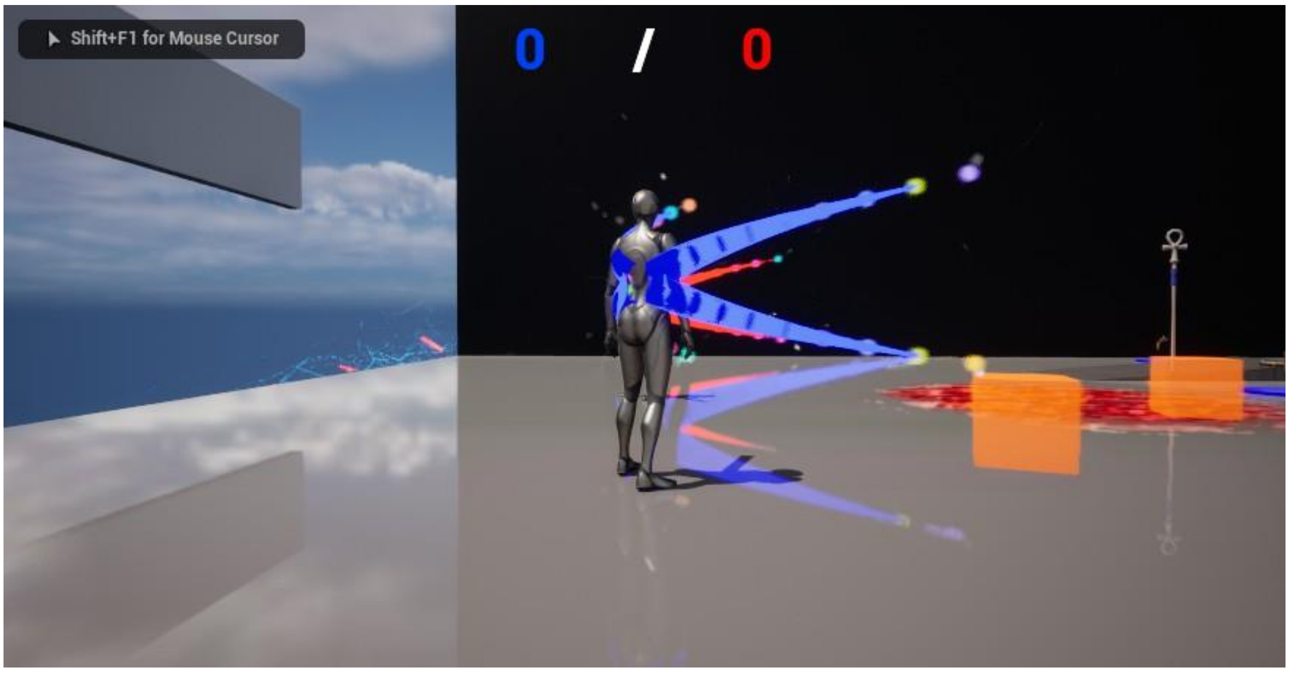
In-Engine Screenshots:



It can be seen that in the screenshot, I shot the other player with the weapon and a damage number of 20 came up. The blood particle also appeared on top of the player itself.



It can be seen on the screenshot that I made it so that it dissipates after it spawns into thin air.



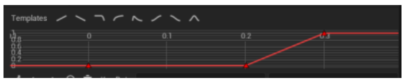
In the picture above, it can be seen that the particles are not there anymore. It's because the particle effect has already dissipated and gone from the world after it spawned with the Spawn Burst Instantaneous Module.

Properties and Values

Floating_Numbers Emitter

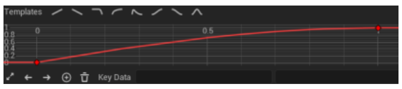
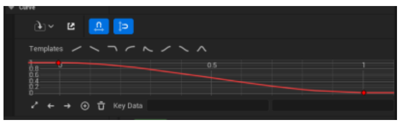
Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Emitter Deactivates but Doesn’t Die until System Does)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	1.5

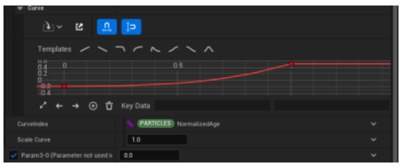
Spawn Particles in Grid	Used to spawn the particles in the grid of the world.	Y Count 256, Z Count 256
Lifetime Mode	Lifetime of the particle	Direct Set 1.5
Colour Mode	The colour of the particle	Direct Set : R 1.0, G 0.025346, B 0.0, A 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Uniform 1.0
Coordinate Space	Defines the originating space of the position before being transformed into the destination space of the emitter.	Local
Normalize Offsets	Scale both the “Offset” and “Randomize Placement within Cell Field” by the grid cell size. Which will mean that a value of 1 in the “Offset” field will produce a single cell shift in position.	true
Grid Location	Spawn particles spaced evenly on a grid	Mentioned Next Tables
Grid Location->Dimensions Definition	“Padding Per Cell” places each particle at a set distance from its neighbours	Padding Per Cell
Grid Location->XYZ Dimensions	Define the volume that the particles will occupy	X 0.25, Y 0.25, Z 0.25
Sample Texture	Sample a texture at a specific UV location and returns the color	true
Texture	The Texture to sample	user.Digit
Kill Particles	Kill Particles if the boolean input into the module is true. Allows for	Set Bool by Float Comparison -> A = Make

	particles to be dynamically killed based on a boolean conditional at any point in the execution stack.	Float from Linear color (LinearColor = STACKCONTEXT SampledColor, Channel = A), B = 0.1 Comparison Type = A Less than B
Kill Particles When Lifetime Has Elapsed	Determines if particles live forever or not	true
Curl Noise Force Strength	Scales the sampled curl noise force vector	FloatCurve :  Scale Curve = 200.0 Noise Frequency = 10.0
Acceleration	Acceleration Amount	Z = 100.0
Point Attraction Force Strength	The strength of the point attraction	15.0
Point Attraction Radius	The radius of the point attraction.	100.0
Falloff Exponent	Exponent to apply to the falloff.	0.5
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_FloatingNumbers

Blood Emitter

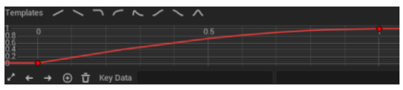
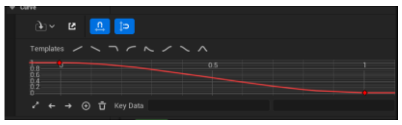
Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Burst Instantaneous	Spawns a burst of particles instantaneously	Spawn Count = 50, Spawn Time = 0.1
Lifetime Mode	Lifetime of the particle	Random Between 0.4 and 0.8
Colour Mode	The colour of the particle	Direct Set : R 1.0, G 0.0, B 0.0, A 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Uniform between 40.0 and 60.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Velocity Mode	The added velocity mode to the particle	In Cone
Velocity Speed	The added speed of the velocity of the particle	Random Range Float (Min 0.0, Max 240.0)

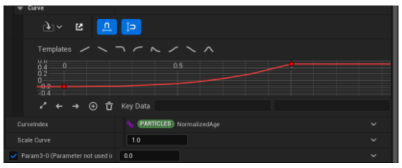
Distribution Along Cone Axis	Base the random vector to prefer adding velocity in the direction of the cone axis 0 is no bias, 1 encourages most of the random velocities to be along the axis of the cone.	0.25
Speed Fall Off From Cone Axis	Decrease the added speed as the direction deviates from the Cone Axis	0.5
Cone Axis	Axis of the Cone Velocity Mode	0.0, 0.0, 1.0
Cone Angle	Angle of the Cone Velocity Mode	50.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Rotation Rate	Scale factor on Delta Time for global speedup (or bypassing) of rotation.	Random Range Float (Minimum -100.0, Maximum 100.0, Evaluation Type : Spawn Only)
Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve -> Curve for Floats 
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.25
Scale Color	By Default, accepts the initial color as determined in the particle spawn script (Cached off as Particles Initial Color), and scales the RGB and Alpha components separately.	Scale Mode = RGB and Alpha Separately Scale RGB (X 1.0, Y 1.0, Z 1.0) Scale Alpha (Float from Curve) 

Gravity Force	Applies a gravitational force (in cm/s) to Transient.PhysicsForce	Random Range vector (Minimum (X 0.0, Y 0.0, Z -160.0), Maximum (X 1.0, Y1.0, Z 40.0) Evaluation Type : Spawn Only
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Dynamic Material Parameters	Write to the Dynamic Parameter vertex interpolator node in the material editor	Mentioned Next Tables
Dynamic Material Parameters->Tile	The override value for tiling the particle materials	Random Range Float (Min 1.0, Max 2.0) Evaluation Type : Spawn Only
Dynamic Material Parameter->Pan	The override value for speed of panning the particle materials	Random Range Float (Min -0.5, Max 0.5) Evaluation Type : Spawn Only
Dynamic Material Parameter->Dissolve	The override value for dissolving the particle materials	Curve for Floats  Curve Templates 0 0.5 0.4 0.2 0 -0.2 -0.4 Key Data CurveIndex PARTICLES NormalizedAge Scale Curve 1.0 ✓ Param3-0 (Parameter not used i 0.0
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Blood

Blood0001 Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Burst Instantaneous	Spawns a burst of particles instantaneously	Spawn Count = 50, Spawn Time = 0.0
Lifetime Mode	Lifetime of the particle	Random Between 0.2 and 1.0
Colour Mode	The colour of the particle	Direct Set : R 1.0, G 0.0, B 0.0, A 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Non-Uniform between X 30.0 Y 60.0 and X = 40.0 Y = 100.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Velocity Mode	The added velocity mode to the particle	In Cone

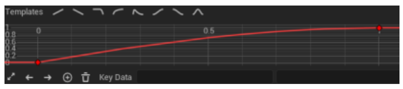
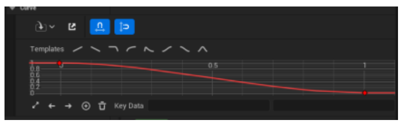
Velocity Speed	The added speed of the velocity of the particle	Random Range Float (Min 10.0, Max 300.0)
Distribution Along Cone Axis	Base the random vector to prefer adding velocity in the direction of the cone axis 0 is no bias, 1 encourages most of the random velocities to be along the axis of the cone.	0.25
Speed Fall Off From Cone Axis	Decrease the added speed as the direction deviates from the Cone Axis	0.5
Cone Axis	Axis of the Cone Velocity Mode	0.0, 0.0, 1.0
Cone Angle	Angle of the Cone Velocity Mode	100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve -> Curve for Floats 
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.25
Scale Color	By Default, accepts the initial color as determined in the particle spawn script (Cached off as Particles Initial Color), and scales the RGB and Alpha components separately.	Scale Mode = RGB and Alpha Separately Scale RGB (X 1.0, Y 1.0, Z 1.0) Scale Alpha (Float from Curve) 
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by	Active

	Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	
Dynamic Material Parameters	Write to the Dynamic Parameter vertex interpolator node in the material editor	Mentioned Next Tables
Dynamic Material Parameters->Tile	The override value for tiling the particle materials	Random Range Float (Min 0.2, Max 2.0) Evaluation Type : Spawn Only
Dynamic Material Parameter->Pan	The override value for speed of panning the particle materials	Random Range Float (Min -0.5, Max 0.5) Evaluation Type : Spawn Only
Dynamic Material Parameter->Dissolve	The override value for dissolving the particle materials	Curve for Floats  Curve Templates 0 0.5 0.4 0.2 0 -0.2 -0.4 Key Data CurveIndex PARTICLES NormalizedAge Scale Curve 1.0 Param3-0 (Parameter not used i 0.0
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Blood

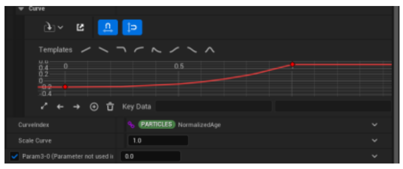
Blood0002 Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self

Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Burst Instantaneous	Spawns a burst of particles instantaneously	Spawn Count = 300, Spawn Time = 0.0
Lifetime Mode	Lifetime of the particle	Random Between 1.0 and 3.0
Colour Mode	The colour of the particle	Direct Set : R 1.0, G 0.0, B 0.0, A 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Uniform between 40.0 and 50.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Velocity Mode	The added velocity mode to the particle	In Cone
Velocity Speed	The added speed of the velocity of the particle	Random Range Float (Min 100.0, Max 410.0)
Distribution Along Cone Axis	Base the random vector to prefer adding velocity in the direction of the cone axis 0 is no bias, 1 encourages most of the random velocities to be along the axis of the cone.	0.25

Speed Fall Off From Cone Axis	Decrease the added speed as the direction deviates from the Cone Axis	0.5
Cone Axis	Axis of the Cone Velocity Mode	0.0, 0.0, 1.0
Cone Angle	Angle of the Cone Velocity Mode	50.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve -> Curve for Floats 
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.25
Scale Color	By Default, accepts the initial color as determined in the particle spawn script (Cached off as Particles Initial Color), and scales the RGB and Alpha components separately.	Scale Mode = RGB and Alpha Separately Scale RGB (X 1.0, Y 1.0, Z 1.0) Scale Alpha (Float from Curve) 
Gravity Force	Applies a gravitational force (in cm/s) to Transient.PhysicsForce	Random Range vector (Minimum (X 0.0, Y 0.0, Z -210.0), Maximum (X 1.0, Y1.0, Z -40.0) Evaluation Type : Spawn Only
Collision Enabled	Enable or disable the module's affect on the effect	enabled
Radius Calculation Type	Each particle's collision radius will be calculated for you, per frame , using the methods laid out below.	Sprite

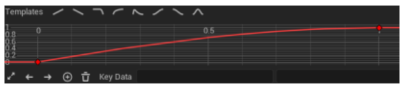
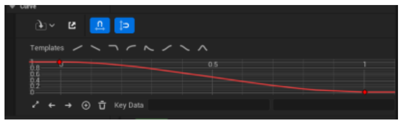
Method for Calculating Particles Radius	This enum changes the way that each particle’s collision radius is calculated.	Bounds
Particle Radius Scale	This value scales the calculated particle collision radius	1.0
Restitution	This controls the particle’s bounce coefficient. 1 will retain all of the particle’s energy along the impact normal vector and 0 will remove it.	0.0
Simple Friction	If true, fewer parameters will be used to control the friction coefficient.	true
Friction	The friction coefficient defines how quickly a particle will slow down as it slides across a surface	0.25
Enable Rest State	This will pause particles that penetrate surfaces more often than the specified rate and particles that have penetrated a surface more deeply than allowed.	true
Maximum Penetration Correction	This number specifies the maximum number of units that a particle can be pulled out of another surface before being instantaneously forced into a rest state	0.5
Percentage of Penetration Before	Particles will enter a rest state if they are found to penetrate surfaces more often than this float allows over the “Rest State Time Range”	1.0
Rest State Time Range	This is the amount of time that a particle’s interpenetrations will be	0.5

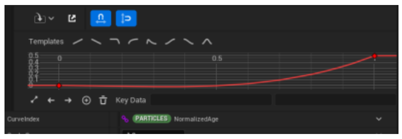
	tracked over when determining if it should enter a rest state.	
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Dynamic Material Parameters	Write to the Dynamic Parameter vertex interpolator node in the material editor	Mentioned Next Tables
Dynamic Material Parameters->Tile	The override value for tiling the particle materials	Random Range Float (Min 0.1, Max 0.2) Evaluation Type : Spawn Only
Dynamic Material Parameter->Pan	The override value for speed of panning the particle materials	Random Range Float (Min -0.1, Max 0.1) Evaluation Type : Spawn Only
Dynamic Material Parameter->Dissolve	The override value for dissolving the particle materials	Curve for Floats  Curve Templates 0.4 0.2 0 -0.2 -0.4 0 0.5 Key Data CurveIndex PARTICLES NormalizedAge Scale Curve 1.0 Param3-0 (Parameter not used i 0.0
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Blood

Blood0003 Emitter

Property	Description of Purpose	Value

Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Burst Instantaneous	Spawns a burst of particles instantaneously	Spawn Count = 50, Spawn Time = 0.1
Lifetime Mode	Lifetime of the particle	Random Between 0.5 and 1.0
Colour Mode	The colour of the particle	Random : R 0.2, G 0.0, B 0.0, A 1.0 Hue Shift Range X -0.1 Y 0.1 Saturation Range 0.8 Y 1.5 Value Range X 0.2 Y 1.0 Alpha Scale Range X 0.8 Y 2.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Uniform between 40.0 and 60.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Velocity Mode	The added velocity mode to the particle	In Cone

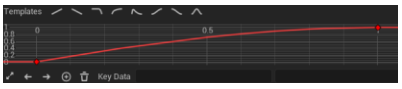
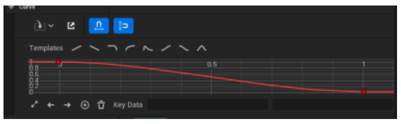
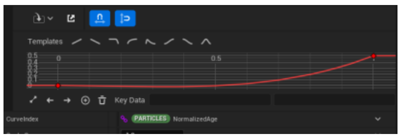
Velocity Speed	The added speed of the velocity of the particle	Random Range Float (Min 0, Max 240.0)
Distribution Along Cone Axis	Base the random vector to prefer adding velocity in the direction of the cone axis 0 is no bias, 1 encourages most of the random velocities to be along the axis of the cone.	0.25
Speed Fall Off From Cone Axis	Decrease the added speed as the direction deviates from the Cone Axis	0.5
Cone Axis	Axis of the Cone Velocity Mode	0.0, 0.0, 1.0
Cone Angle	Angle of the Cone Velocity Mode	100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve -> Curve for Floats 
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.5
Scale Color	By Default, accepts the initial color as determined in the particle spawn script (Cached off as Particles Initial Color), and scales the RGB and Alpha components separately.	Scale Mode = RGB and Alpha Separately Scale RGB (X 1.0, Y 1.0, Z 1.0) Scale Alpha (Float from Curve) 
Gravity Force	Applies a gravitational force (in cm/s) to Transient.PhysicsForce	Random Range vector (Minimum (X 0.0, Y 0.0, Z -120.0), Maximum (X 1.0,

		Y1.0, Z -90.0) Evaluation Type : Spawn Only
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Dynamic Material Parameters	Write to the Dynamic Parameter vertex interpolator node in the material editor	Mentioned Next Tables
Dynamic Material Parameters->Tile	The override value for tiling the particle materials	Random Range Float (Min 0.5, Max 1.0) Evaluation Type : Spawn Only
Dynamic Material Parameter->Pan	The override value for speed of panning the particle materials	Random Range Float (Min -0.5, Max 0.5) Evaluation Type : Spawn Only
Dynamic Material Parameter->Diss olve	The override value for dissolving the particle materials	Curve for Floats  CurveIndex PARTICLES NormalizedAge
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Blood_Opacity

Blood0004 Emitter

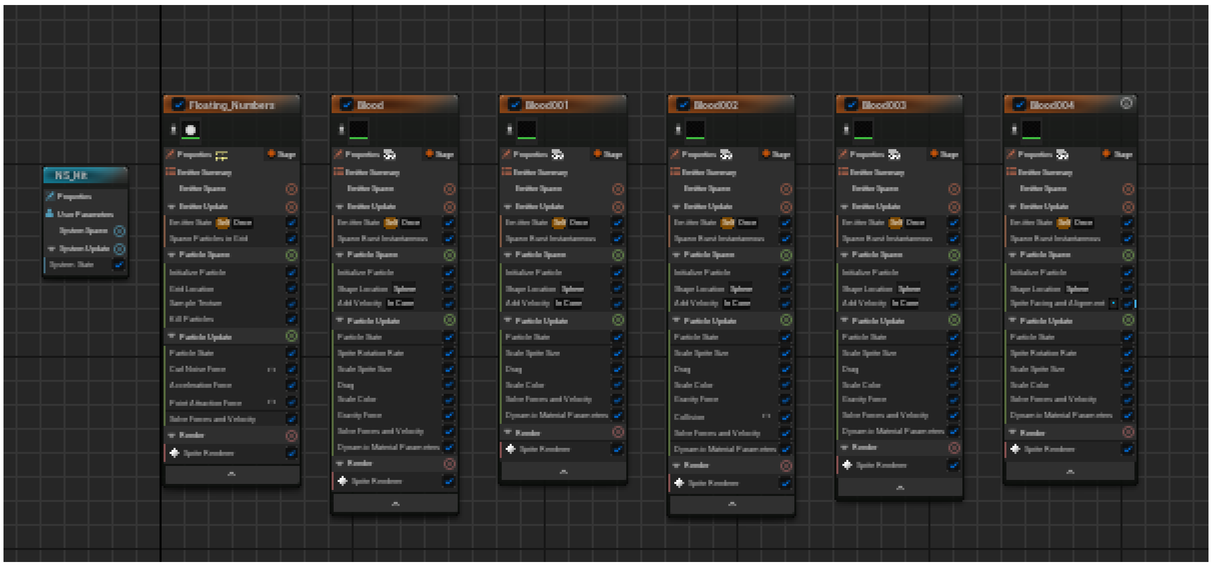
Property	Description of Purpose	Value

Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Continue (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Once
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Burst Instantaneous	Spawns a burst of particles instantaneously	Spawn Count = 50, Spawn Time = 0.1
Lifetime Mode	Lifetime of the particle	Random Between 0.8 and 1.0
Colour Mode	The colour of the particle	Direct Set : R 0.2, G 0.0, B 0.0, A 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Uniform between 40.0 and 200.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Sprite Facing and Alignment	Changes the sprite faces and alignment	X 0.0, Y 0.0, Z 1.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Rotation Rate	Scale factor on Delta Time for global speedup (or bypassing) of rotation.	Random Range Float (Minimum -100.0, Maximum 100.0, Evaluation

		Type : Spawn Only)
Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve -> Curve for Floats 
Scale Color	By Default, accepts the initial color as determined in the particle spawn script (Cached off as Particles Initial Color), and scales the RGB and Alpha components separately.	Scale Mode = RGB and Alpha Separately Scale RGB (X 1.0, Y 1.0, Z 1.0) Scale Alpha (Float from Curve) 
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Dynamic Material Parameters	Write to the Dynamic Parameter vertex interpolator node in the material editor	Mentioned Next Tables
Dynamic Material Parameters->Tile	The override value for tiling the particle materials	Random Range Float (Min 1.0, Max 2.0) Evaluation Type : Spawn Only
Dynamic Material Parameter->Pan	The override value for speed of panning the particle materials	Random Range Float (Min -0.5, Max 0.5) Evaluation Type : Spawn Only
Dynamic Material Parameter->Diss olve	The override value for dissolving the particle materials	Curve for Floats 

Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Blood

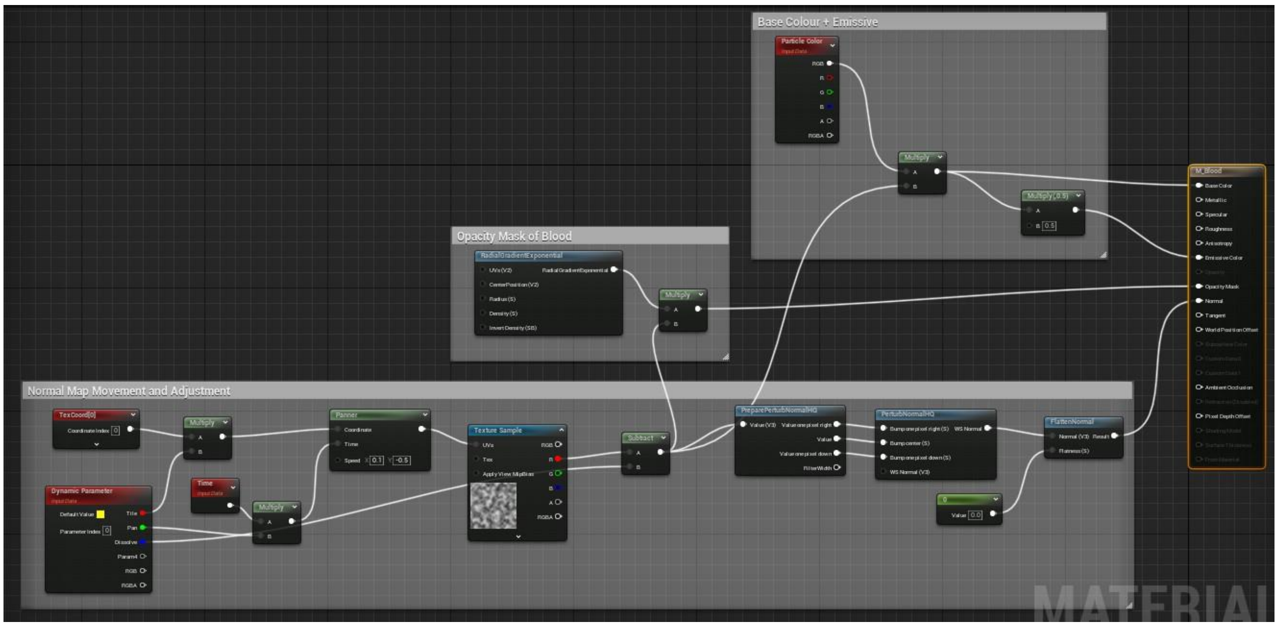
Niagara System / Emitters Breakdown



This is the overview of the emitters. This particle has a total of 6 emitters. The combined effect of the 6 emitters created the floating damage particles combined with the blood. Particle effects. Here are the details of each emitter and custom materials :

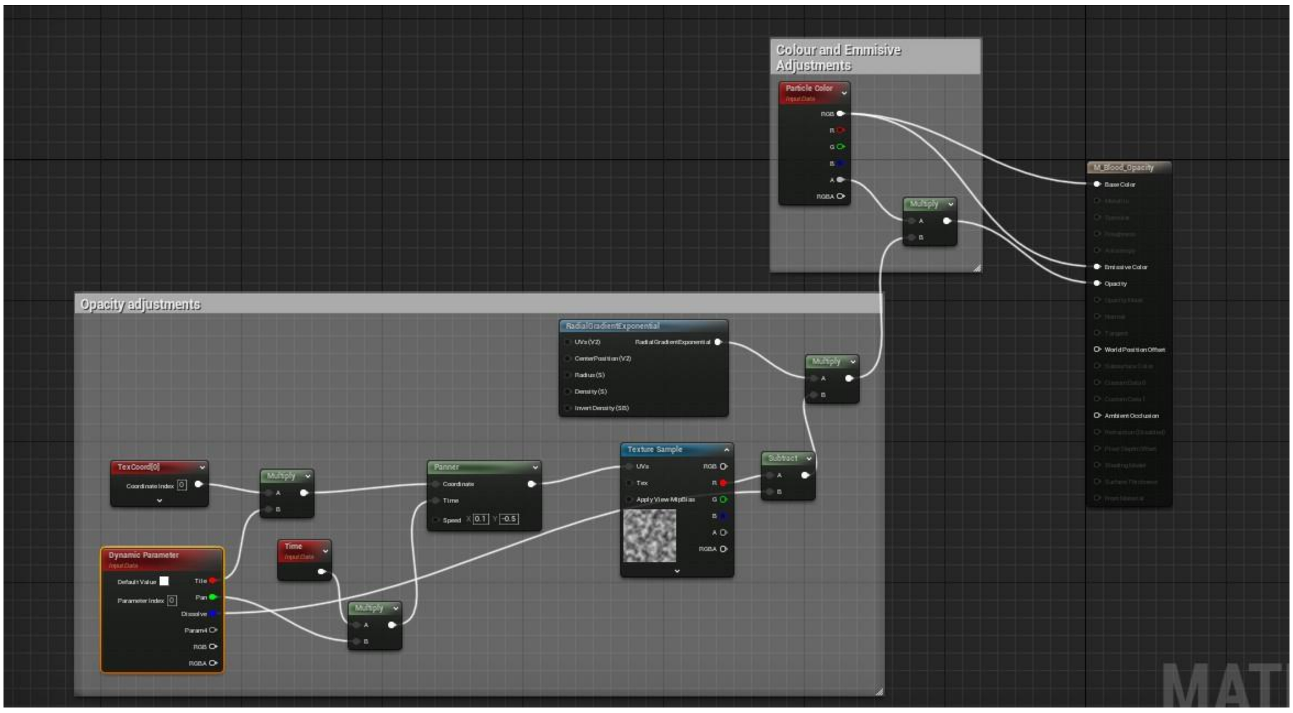
Custom Materials

Custom Material 1 - M_Blood



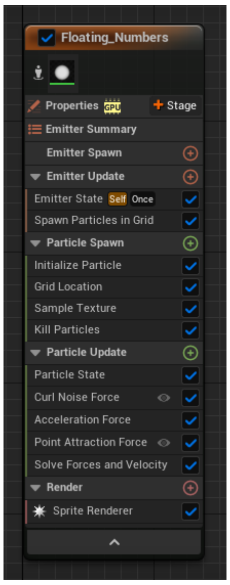
I added this custom material for the blood particles. The way it works is simple, it uses an opacity mask to make the blood sprite effect and then it also uses a dynamic parameter to adjust the values accordingly to what I want it to look like. From the dynamic parameter value, it adjusts the normal map movement.

Custom Material 2- M_Blood_Opacity



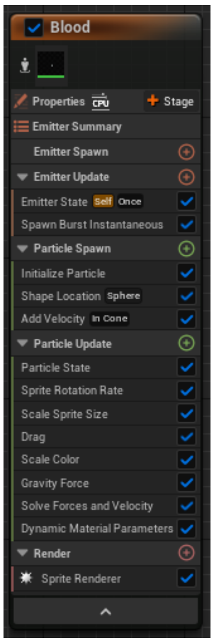
The way this custom material works is fairly similar but the difference is that the opacity of the particle is adjusted through the material instead of the normals.

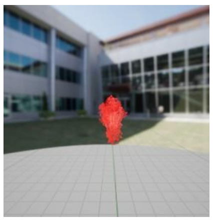
Emitters



Emitter 1- Floating_Numbers Emitter

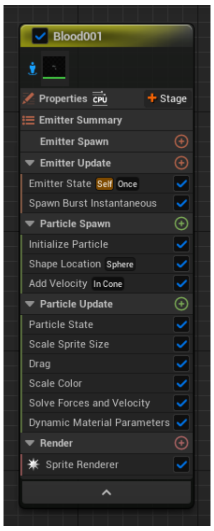
The floating number emitters are very simple. It spawns on the top of the spawn location which is the player. Then, it spawns the texture that is connected to C++ for the amount of damage that the player does to the player. Then it disappears after a while curling into thin air.


Blood
Properties CPU + Stage
Emitter Summary
Emitter Spawn
Emitter Update
Emitter State Self Once
Spawn Burst Instantaneous
Particle Spawn
Initialize Particle
Shape Location Sphere
Add Velocity In Cone
Particle Update
Particle State
Sprite Rotation Rate
Scale Sprite Size
Drag
Scale Color
Gravity Force
Solve Forces and Velocity
Dynamic Material Parameters
Render
Sprite Renderer



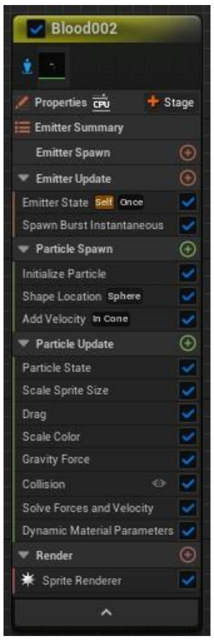
Emitter 2 - Blood Emitter

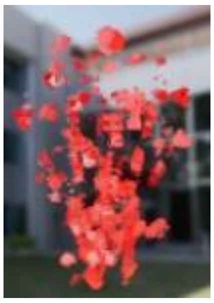
This emitter emulates a simple blood splatter. This uses a custom material that I made and splatters in a semi realistic manner. This is also combined with drag, gravity force, and the dynamic material parameters for the custom material, creating this optimal effect of the particle. This spawns with the floating damage as well splattering down to the player from above. The shape location of the spawn is a sphere which can be fairly changed but a sphere usually suffices. The sprite rotation rate is also important to give the effect of the splatter when the sprite just spawned and it rotates outwards. Velocity is also added to simulate the splatter.


Blood001
Properties CPU + Stage
Emitter Summary
Emitter Spawn
Emitter Update
Emitter State Self Once
Spawn Burst Instantaneous
Particle Spawn
Initialize Particle
Shape Location Sphere
Add Velocity In Cone
Particle Update
Particle State
Scale Sprite Size
Drag
Scale Color
Solve Forces and Velocity
Dynamic Material Parameters
Render
Sprite Renderer

Emitter 3 - Blood0001 Emitter

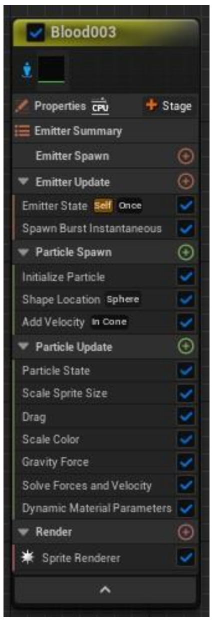
This emitter has almost the same properties as the previous emitter. The only difference is that it has less volume but spreads more. This is done to create a sense of spread of the blood instead of just a larger volume spreading into a small area.


Blood002
Properties CPU + Stage
Emitter Summary
Emitter Spawn
Emitter Update
Emitter State Self Once
Spawn Burst Instantaneous
Particle Spawn
Initialize Particle
Shape Location Sphere
Add Velocity In Cone
Particle Update
Particle State
Scale Sprite Size
Drag
Scale Color
Gravity Force
Collision
Solve Forces and Velocity
Dynamic Material Parameters
Render
Sprite Renderer



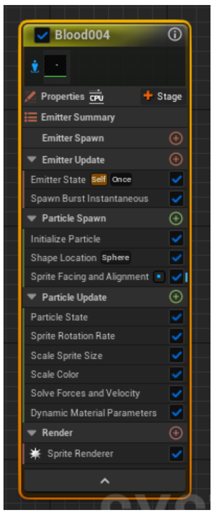
Emitter 4 - Blood0002 Emitter

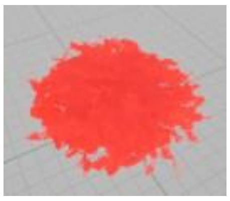
This emitter also has similar properties as the previous ones but the values of it are changed and tinkered. The difference is also that it has a more chunk-like consistency and also it collides with the environment making it look like a mixture of blood and flower petals. Since I want the effect to look grotesque yet beautiful, I added this emitter as well.



Emitter 5 - Blood0003 Emitter

This emitter has a similar value with the previous emitter but some of the values are tinkered. The difference is also in the custom material. I made it so that it has a certain type of opacity into it making it more transparent than the previous emitters. This creates a more layered effect of the blood splatter making it more immersive and look better.

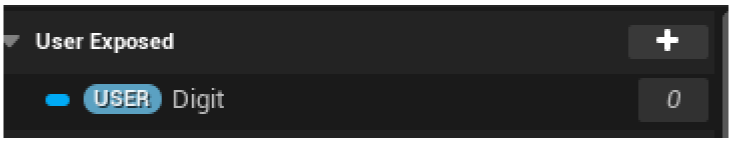




Emitter 6 - Blood0004 Emitter

This emitter is very different compared to the previous emitters. The way this emitter works is that it spawns as if it looks like a portal or a pool of blood. The way I did this is by adding a Sprite Facing and Alignment node and adjusting it accordingly. I also tinkered with the values of the modules that were provided before. This creates a more immersive effect of the blood overall.

C++ Parameters Breakdown



User Exposed

USER Digit 0

The C++ user exposed parameter is called Digit. The intended purpose of the parameter is to input the amount of damage that the player inflicted on the particles so that the particle can use the appropriate number to display to the players. This is also controlled by code. Since Niagara doesn't have a code that changes the UTexture from the particle component properly that I know of, I made a separate function.

```
void AA01_BlockoutShooterCharacter::SetNiagaraVariableTexture(const
UNiagaraComponent* Niagara,
  const FString& InVariableName, UTexture* InValue)
```

```
{
  if(!Niagara || !InValue)
  {
     return;
  }

  FName VarName = FName(*InVariableName);

  FNiagaraUserRedirectionParameterStore& OverrideParameters =
(FNiagaraUserRedirectionParameterStore&)Niagara->GetOverrideParameters();

  UNiagaraDataInterfaceTexture* Data =
(UNiagaraDataInterfaceTexture*)OverrideParameters.GetDataInterface(FNiagaraVariabl
e(FNiagaraTypeDefinition(UNiagaraDataInterfaceTexture::StaticClass()), VarName));

  if(Data)
  {
     Data->Texture = InValue;
  }
}
```

The way I spawn the particles and use the code is also fairly interesting.

```
void AA01_BlockoutShooterCharacter::MulticastSpawnHitParticles_Implementation(int
DamageAmount, FVector HitLocation)
{
  // Setup local variables
  FVector Location = HitLocation;
  TArray<int32> DamageArray;

  // Convert the DamageAmount into a string
  FString DamageString = UKismetStringLibrary::Conv_IntToString(DamageAmount);

  // Get every single string component into an array
  for(int i = 0; i < DamageString.Len(); i++)
  {
     int DividedVariable = 10 * i;
     if(DividedVariable == 0)
     {
        DamageArray.Emplace(DamageAmount % 10);
     }
     else
     {
        DamageArray.Emplace(DamageAmount / DividedVariable % 10);
     }
  }

  // Reverse the array so it matches the damage
  Algo::Reverse(DamageArray);

  // Start going through the made array
  for (auto& Number : DamageArray)
  {
     if(NS_Hit)
     {
        // Spawn the particle
        DigitSystem = UNiagaraFunctionLibrary::SpawnSystemAtLocation(GetWorld(), NS_Hit,
FVector(HitLocation.X, HitLocation.Y, HitLocation.Z + 100), FRotator(180, 180,0));

        // Add offset so the numbers dont overlap
        DigitSystem->AddLocalOffset(FVector(0,TextOffset, 0), false, nullptr);
        TextOffset += 30;
     }

     // Put the texture based on what is on the array
     if(DigitSystem)
     {
        SetNiagaraVariableTexture(DigitSystem, "user.Digit", NumberTextures[Number]);
     }
  }

  // Reset text offset when all hit particles spawned
  TextOffset = 0;
}
```

Basically, the damage that is done to the player is stored in an int32 array so first I initialised an empty int32 array. Then I converted the damage that was done to the player to string and made it so that based on the length of the string, it adds numbers to the empty array that was initialised earlier. Then, I reversed the array since I want the numbers to be shown properly and if I don't reverse it, it'll show backwards. Then, after spawning it, the textures that are spawned are based on what the number is based on the array and it will replace the DigitSystem user exposed variable based on the UTexture files that I have set up from the number 0 to number 9. There are also offsets when it spawns so that the numbers don't get spawned on top of each other. With this, it creates an optimal damage number that appears as a sprite with the adjusted UTexture files.

Destruction Aware Niagara Particle Effect

Niagara Particle Effect - Lightning

Overview of Effect

The destruction aware particle effect that I included in the game is impactful in a way to indicate when the geometry collections have collided with objects. Basically, when the wall is hit, it creates lightning sparks that come out of it. The effect itself looks aesthetically pleasing and flashy. This can give an indicator of the wall being collided and hit. The effect itself also has a large scale size that can take the player off guard. Other than being an indicator, the impact of this particle overall gives the player a more immersive experience when the player destroys the geometry collections that are present within the level.

The colour of the particles are also dominantly blue with a hint of red on some parts of the particles. This is done to create an aesthetically pleasing contrast between both of the colours rather than blending similar colours together. This gives the player extra immersion and a certain type of satisfaction when walls are being collided due to the bold combination of blue and red that even though has a very high contrast, it still works very well. Different from the typical particles that are subtle and blend with the wall or geometry collection, this creates a very flashy and contrast effect.

Effect Description

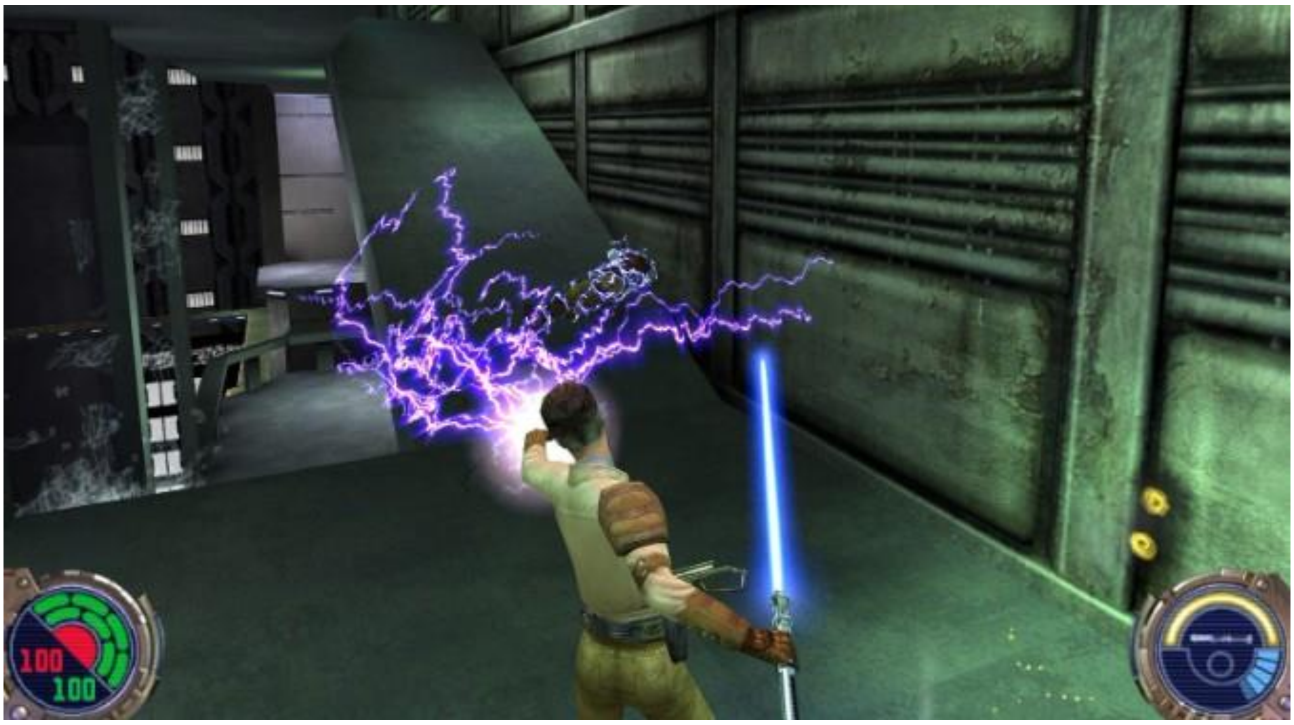
The effect itself is not as commonly used in video games as the blood and the damage numbers particles. However, the effect can still be quite effective when used correctly. The type of destruction data being listened to is called through collisions. This can spawn the lightning particle outwards from where it collided.

Inspiration / Reference Images:

For inspiration and references, there are a couple of things that I found throughout the internet. Here are the reference image that I found to make this particle effect :

Source : https://www.pinterest.ca/pin/165366617548201472/

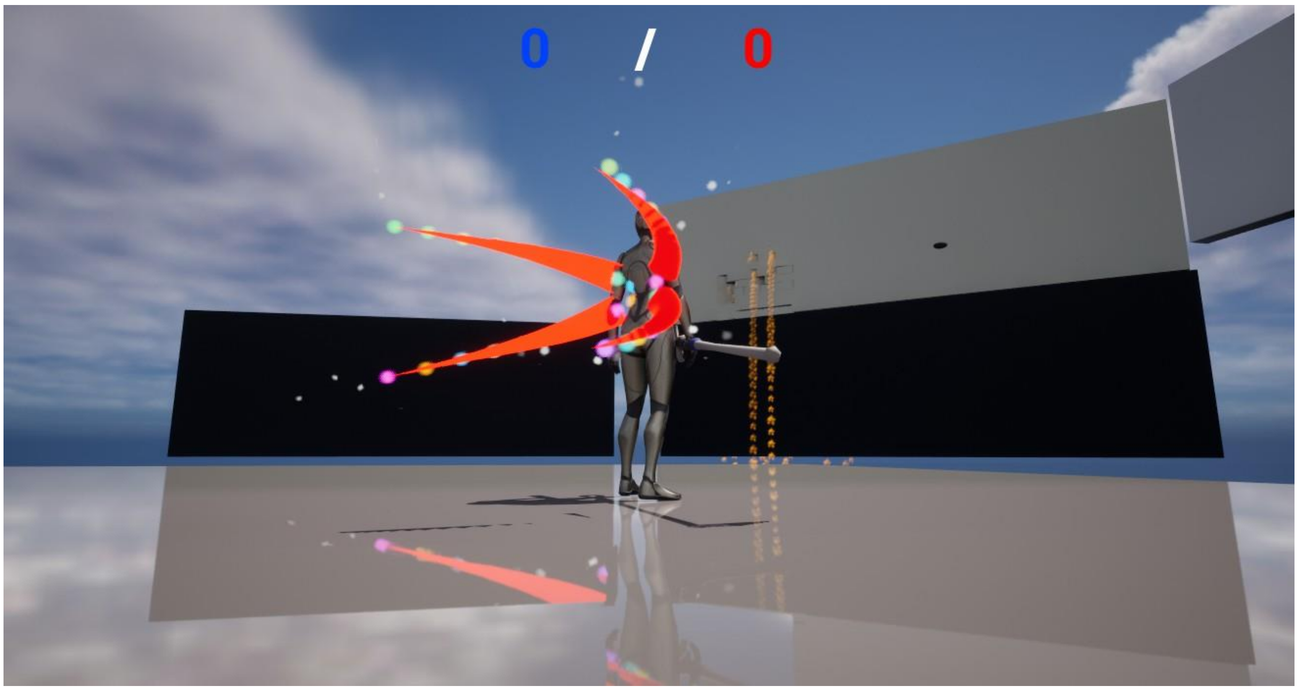
While I was scrolling through memes, I stumbled upon this meme. This endearing meme has the lightning effect that I was looking for making it suitable as a reference image.



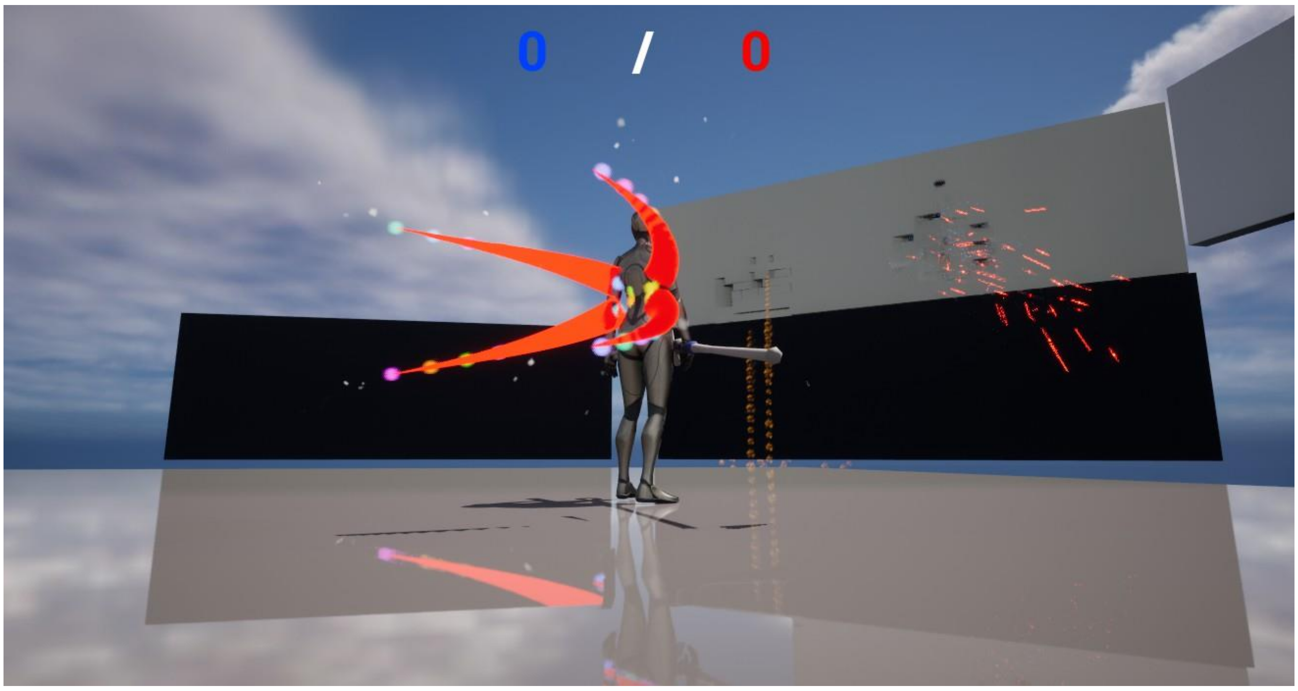
Source :
https://www.polygon.com/2019/9/4/20850148/star-wars-jedi-knight-2-jedi-outcast-nintendo-switch-nintendo-direct

This picture gives me the effect of the lightning that I want. This is an image from the game Star Wars Jedi Knight II. The lightning effect which is from a move called Force Lightning is fairly simple but effective, therefore I used this as well as a reference.

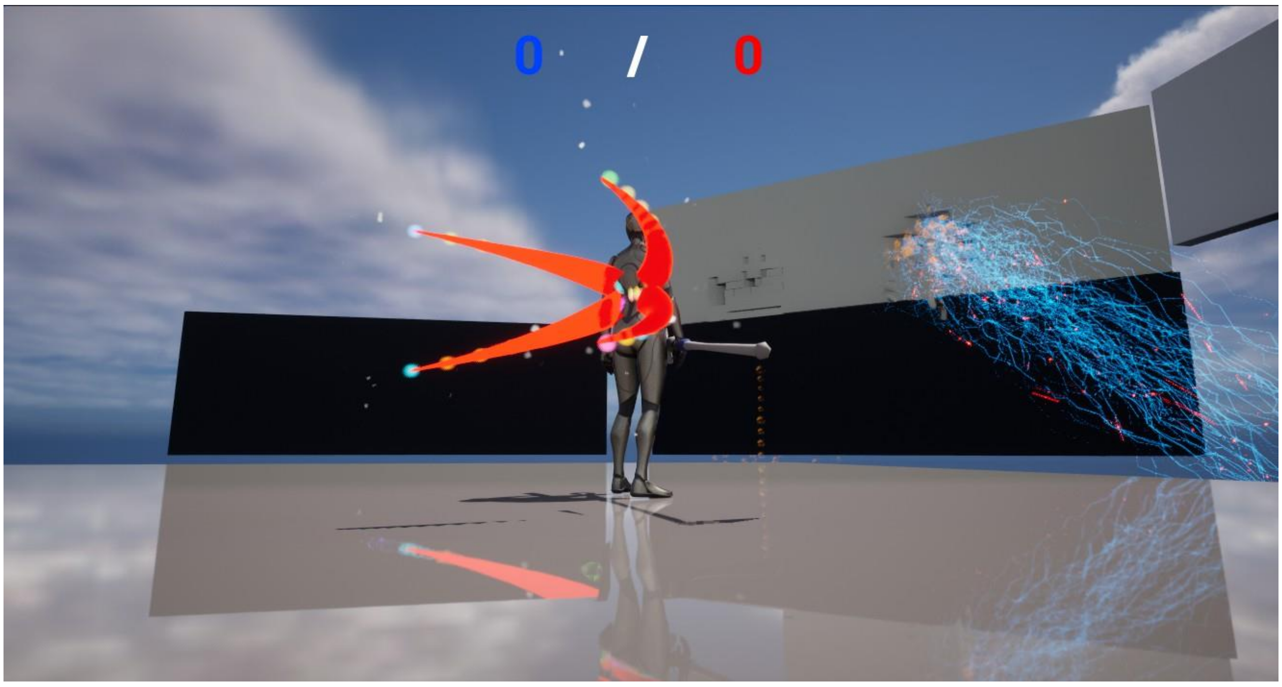
In-Engine Screenshots:



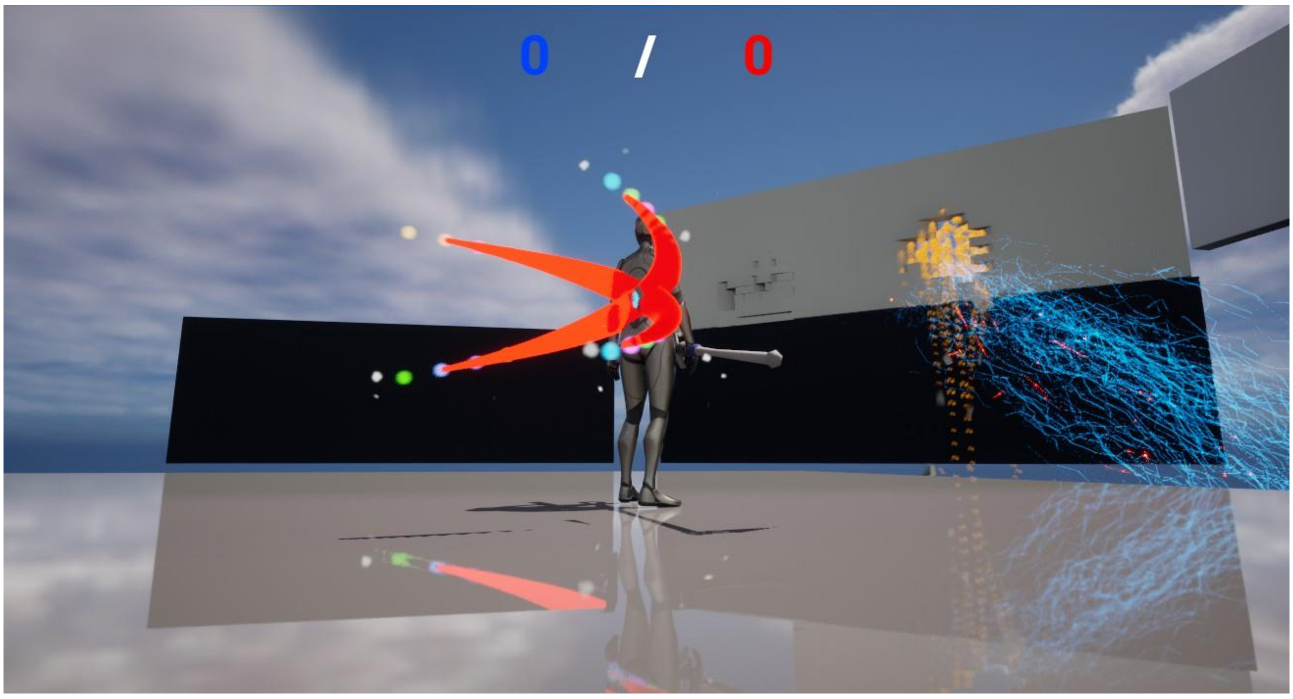
In this image, it can be seen that the projectile is going towards the wall.



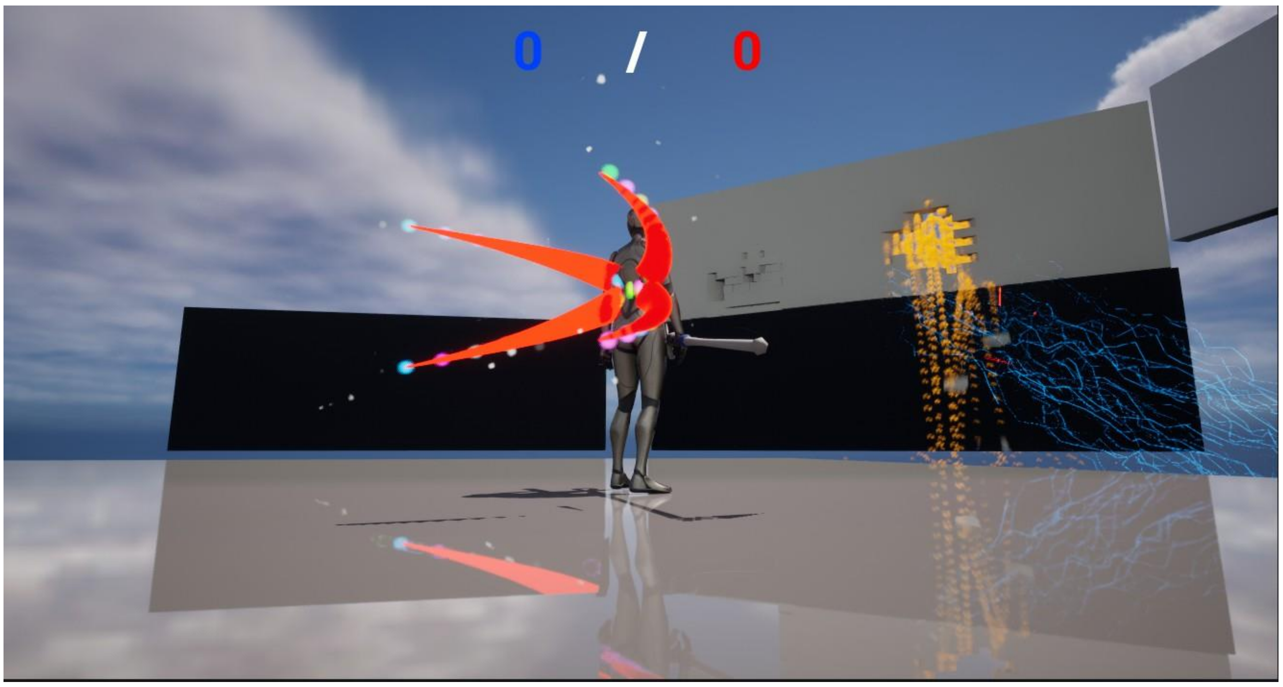
In this image, it can be seen that the wall got hit by the projectile and red sparks started coming out. This embarks the first phase of the lightning particles.



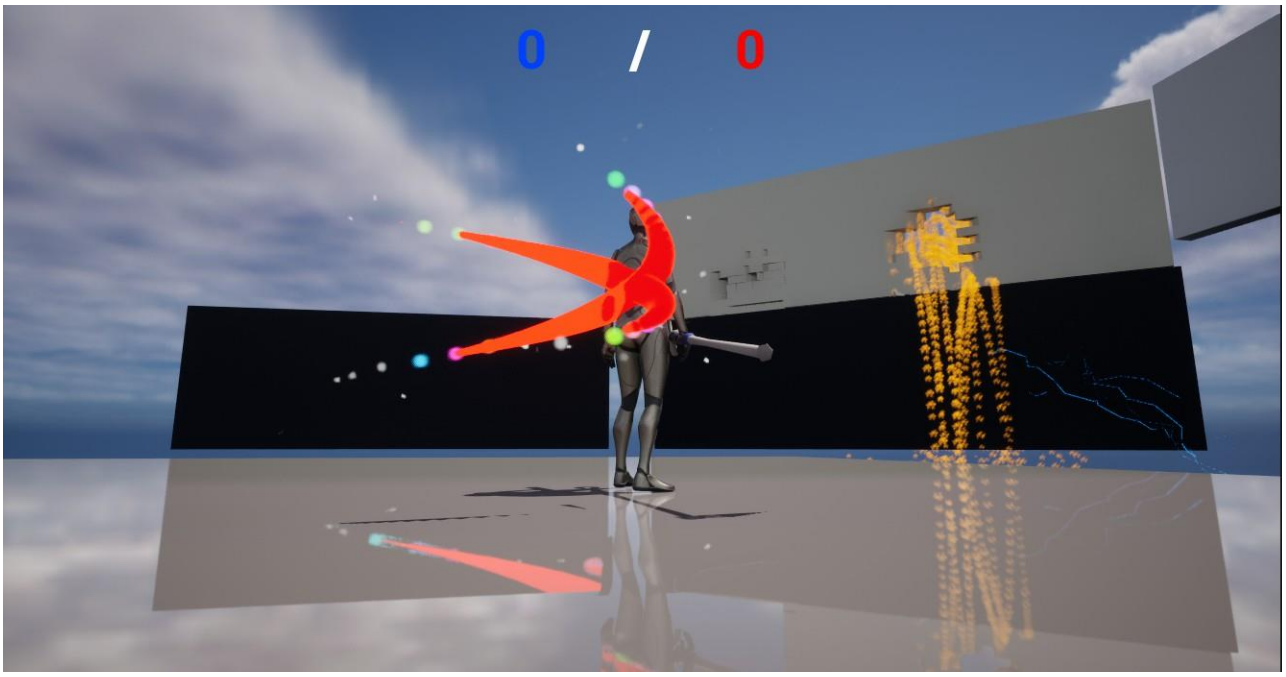
From the red sparks, the blue lightning particles are spawned. A lot of blue lightning particles are spawned from it that will dissipate overtime.



From this image, it can be seen that the lightning starts to dissipate from the source.



From this image, it can be seen that the lightning almost dissipates completely. This means the particle will disappear overtime after no collisions are spawned again.



From his image, it can be seen that the lightning has dissipated leaving only small trails left. This explains how the lightning disappears and how it can and will disappear completely after a specific amount of time.

Properties and Values

Source Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	System
Spawn from Chaos	Spawn particles based on event data from a Chaos solver	New Chaos Destruction Data (TrailLightningChaosData), Spawn Percentage Fraction = 1.0
Lifetime Mode	Lifetime of the particle	Random Between 0.1 and 0.5
Colour Mode	The colour of the particle	Direct Set : R 100.0, G 0.0, B 0.0, A 1.0

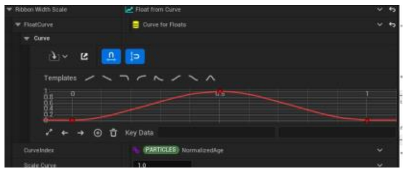
Sprite Size Mode	Determine the scale of the sprites of the particle	Uniform 15.0
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Velocity Mode	The added velocity mode to the particle	From Point
Velocity Speed	The added speed of the velocity of the particle	Random Range Float (Min 800.0, Max 1000.0)
Apply Chaos Data	Set particle position, velocity and color from a Chaos solver	New Chaos Destruction Data = TrailingLightningChaosDat a
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Gravity Force	Applies a gravitational force (in cm/s) to Transient.PhysicsForce	X = 0.0, Y = 0.0, Z = -980.0
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	0.25
Spring Force	Applies a spring-like force on a particle given certain parameters. Accumulates to Transient.PhysicsForce	Force Strength = 1.0, Spring Tightness 5.0
Curl Noise Force Strength	Scales the sampled curl noise force vector	30000.0
Curl Noise Force Noise Frequency	Modulates position to increase or decrease the rate at which curl noise is sampled.	20.0

Curl Noise Force Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.2, Y = 0.5, Z = 1.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
JItter Position Amount	Multiplies against the jitter offset to create an offset vector which is applied to particle position	50.0

Lightning Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Spawn Particles from Other Emitter	Spawn particles from another emitter	Emitter Name = Source, Spawn Rate = 100.0, Spawn Rate per Particle Cap 1000.0
Lifetime Mode	Lifetime of the particle	Random Between 1.0 and 1.0
Colour Mode	The colour of the particle	Direct Set : R 0.0, G 0.407705, B 1.0, A 1.0

Ribbon Width Mode	Determine the scale of the ribbon of the particle	Direct Set 3.0
Sample Particles from Other Emitter	Sample the attributes of particles in another emitter	enabled
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Curl Noise Force Strength	Scales the sampled curl noise force vector	80.0
Curl Noise Force Noise Frequency	Modulates position to increase or decrease the rate at which curl noise is sampled.	10.0
Curl Noise Force Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.2, Y = 0.5, Z = 1.0
Curl Noise Force Strength (2)	Scales the sampled curl noise force vector	170.0
Curl Noise Force Noise Frequency (2)	Modulates position to increase or decrease the rate at which curl noise is sampled.	3.0
Curl Noise Force Pan Noise Field (2)	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.2, Y = 0.5, Z = 1.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the	Active

	updated Particles. Velocity and Particles.Position.	
Scale Ribbon Width	Scales the initial Ribbon Width by a scale factor	Float from Curve  Ribbon Width Scale: Float from Curve FloatCurve: Curve for Floats Curve CurveIndex: PARTICLES NormalizedAge Scale Curve: 1.0
Jitter Position Amount	Multiplies against the jitter offset to create an offset vector which is applied to particle position	3.0
Ribbon Renderer	What the particles will render as	Ribbon Renderer

Collision Enabled Niagara Particle Effect

Niagara Particle Effect -Tentacles

Overview of Effect

For this particle effect, I wanted something that the player has attached to the body. It could be equipment, extra limbs, or anything. Then, I stumbled upon the idea of adding tentacles into the back of the player. This particle effect increased the interactivity and the aesthetics aspect of the game overall. Even though it doesn't necessarily do anything that can impact the overall course of gameplay such as damaging, extra abilities, etc it still packs a punch in the aesthetic department.

The interaction that it has with the environment also looks very pleasing since it can interact with walls that have a specific material in it. It can create a material change that seemingly looks like a portal that the tentacles opened. This can create a more immersive experience especially combined with the player's wall run ability. This can create a more vibrant aesthetic that adds to the already immersive wall running capabilities of the players. The colours also represent which team the player is in and this can create a sense of uniqueness to each of the players themselves.

It is also worth mentioning that the tentacle particles are spawned in fours that are located in the back of the player character. This is done to add a dynamic sense and a concept of tentacles that players are used to rather than just one single tentacle.

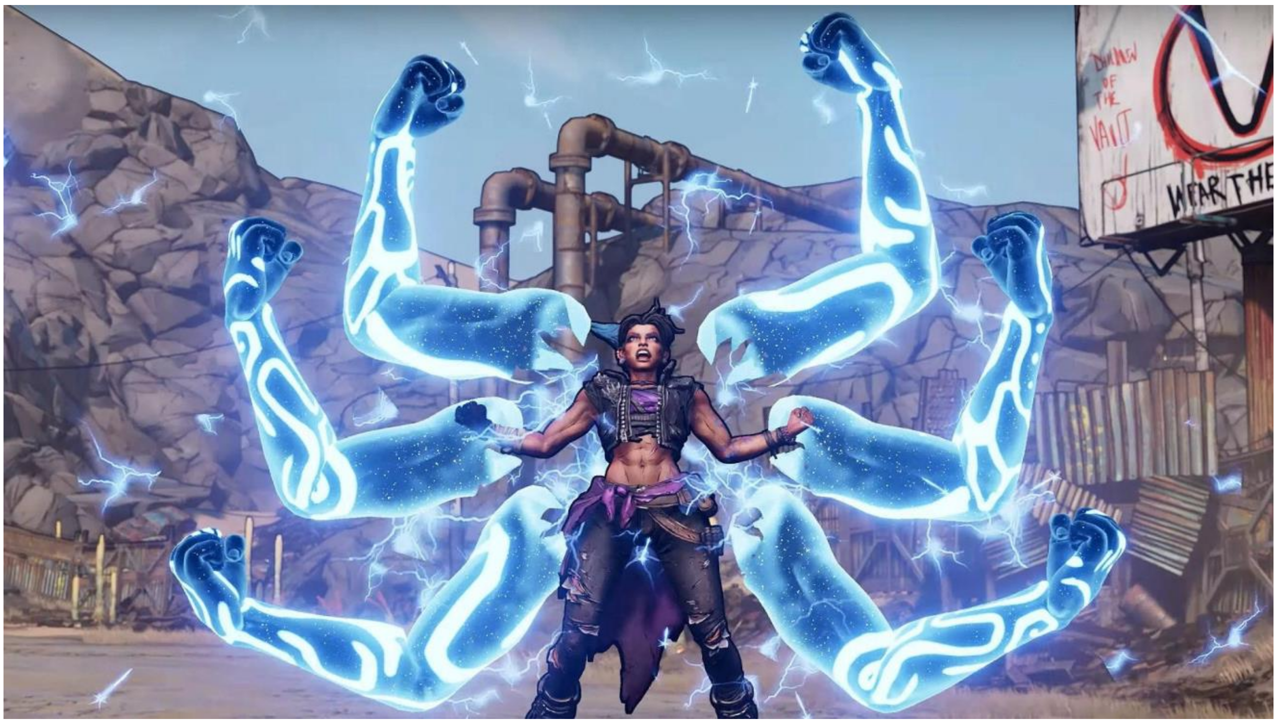
Effect Description

The particle itself has a colour between the players which are either blue or red. The particle is made out of a ribbon renderer which fits the tentacle and when it collides. The particle also changes shapes accordingly based on where the player is facing and when the particle collides with objects. The particles are inspired by tentacles that are portrayed through video games and media. Basically, the collision changes the material of specific walls with specific materials into something else based on the tentacles. The colours that are changed are from black to a mixture of green and cyan that looks textured with the noise node.

Inspiration / Reference Images:

Source : https://disney.fandom.com/wiki/Ursula

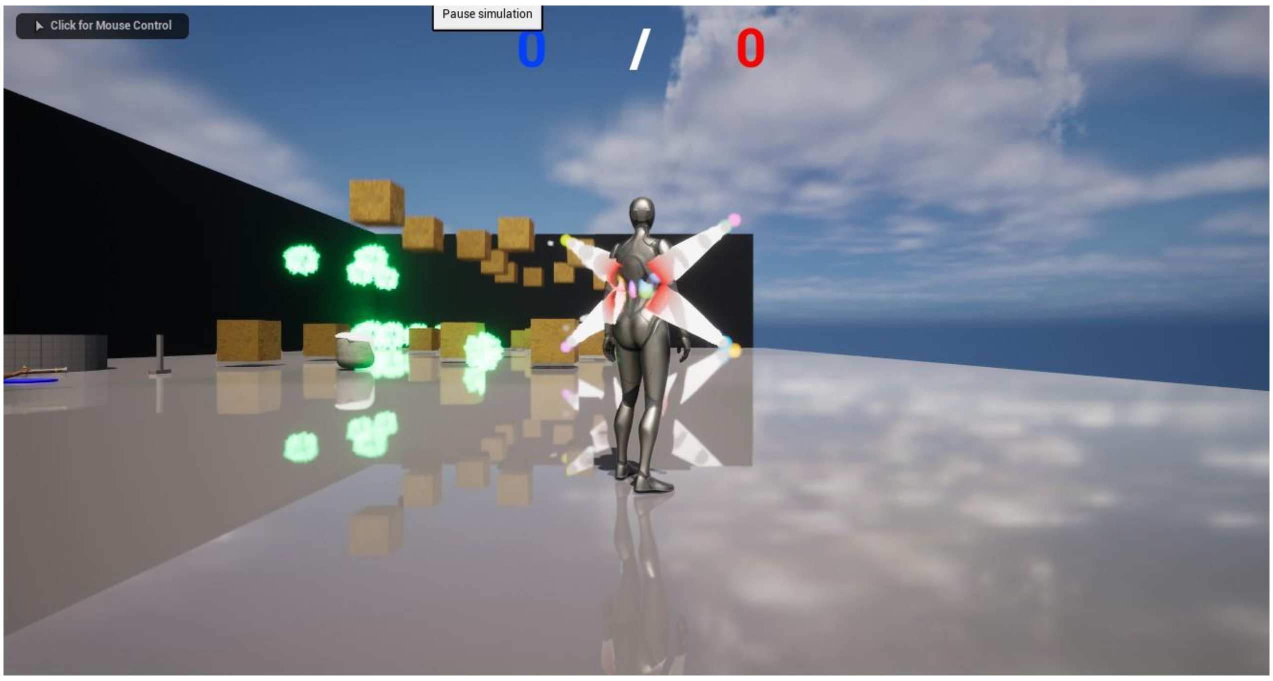
One of the inspirations is from the movie Little mermaid. The movie Little Mermaid has a character called Ursula who is a giant octopus humanoid creature. The tentacles that are attached to her body are what inspired the making of the tentacles particles.



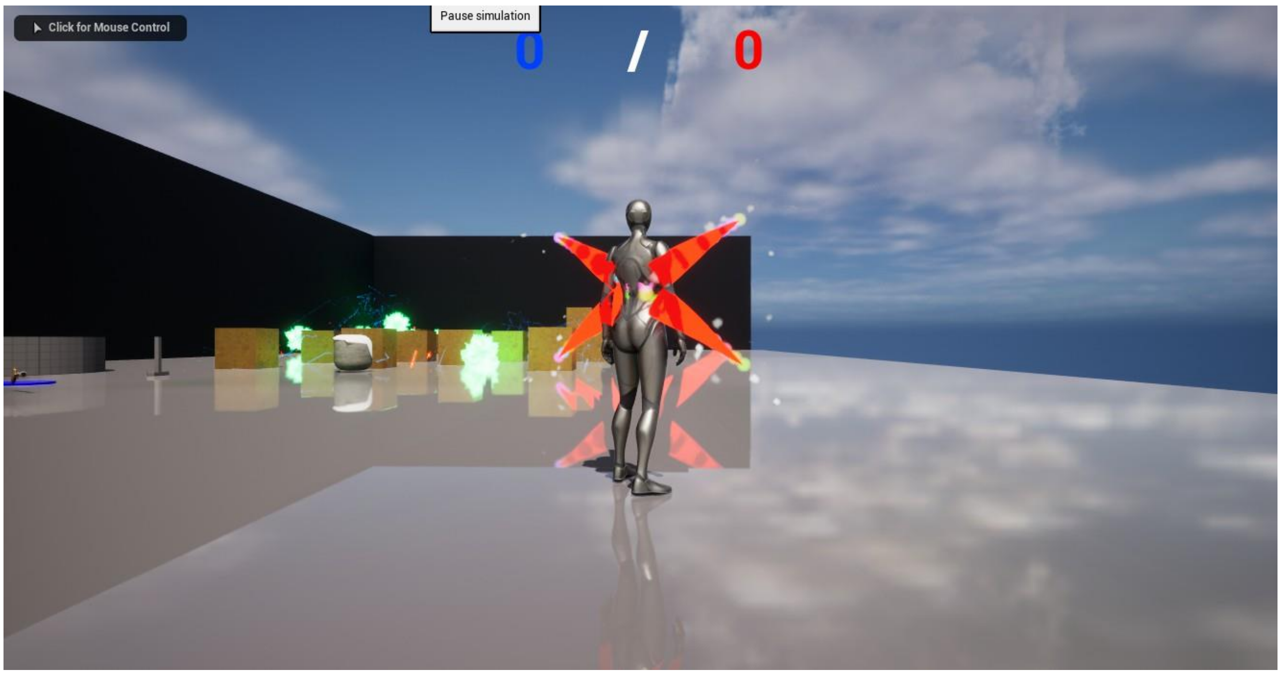
Source : https://www.pcgamesn.com/borderlands-3/amara-skill-tree

The character from Borderlands 3 named Amara has a power of extra limbs behind her back. Even though the extra limbs are not tentacles, it still inspires me to make extra limbs for my character.

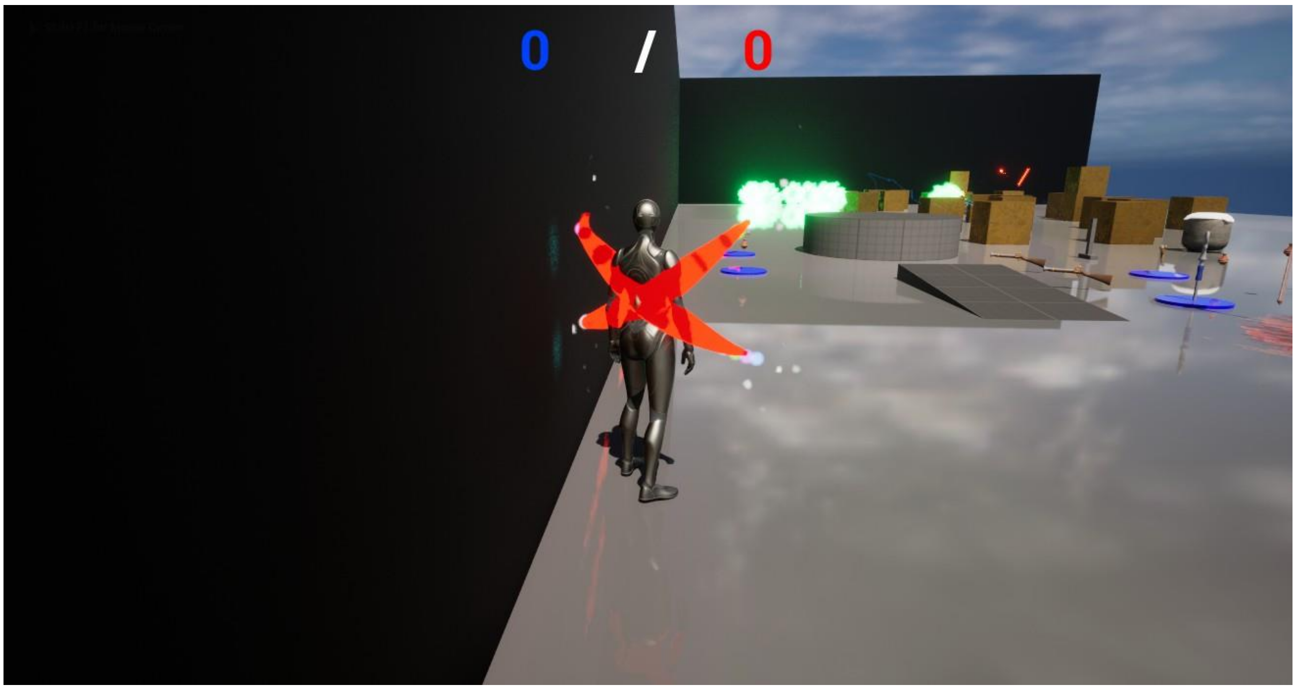
In-Engine Screenshots:



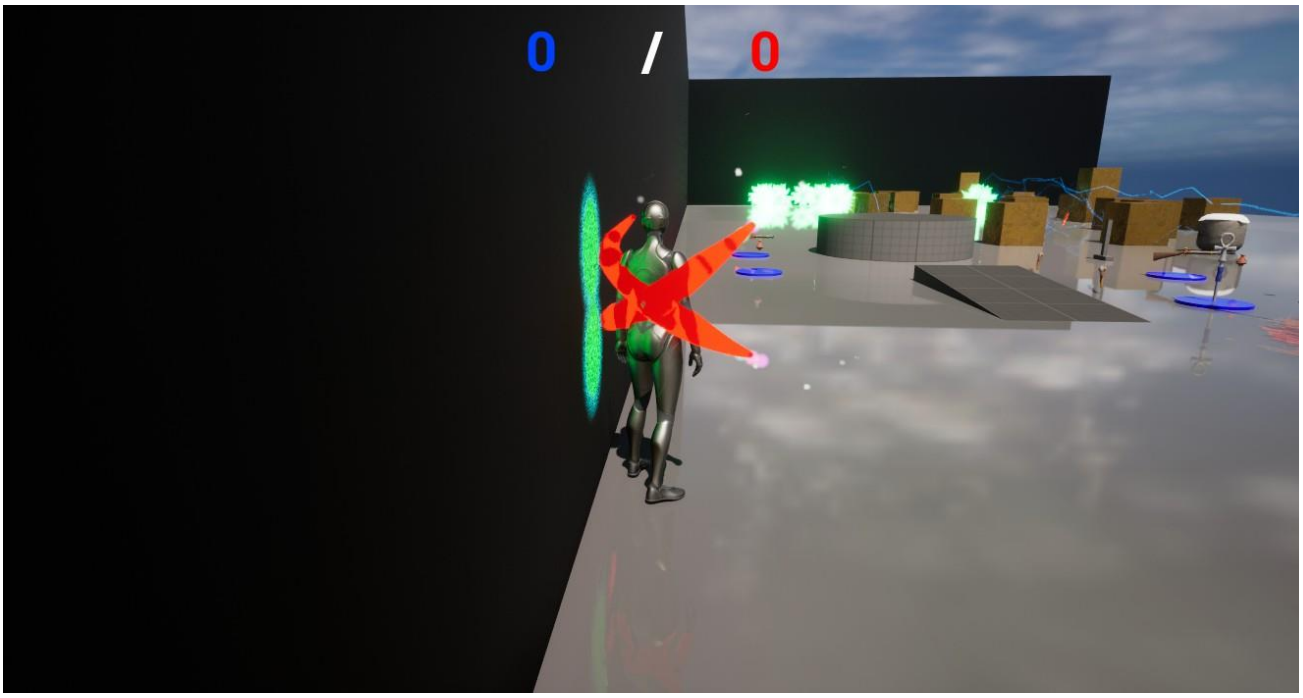
In this screenshot, it shows that the particle starts off as a white colour. Then it gradually becomes the colour of the player's team. It also spawned four of them with different rotations.



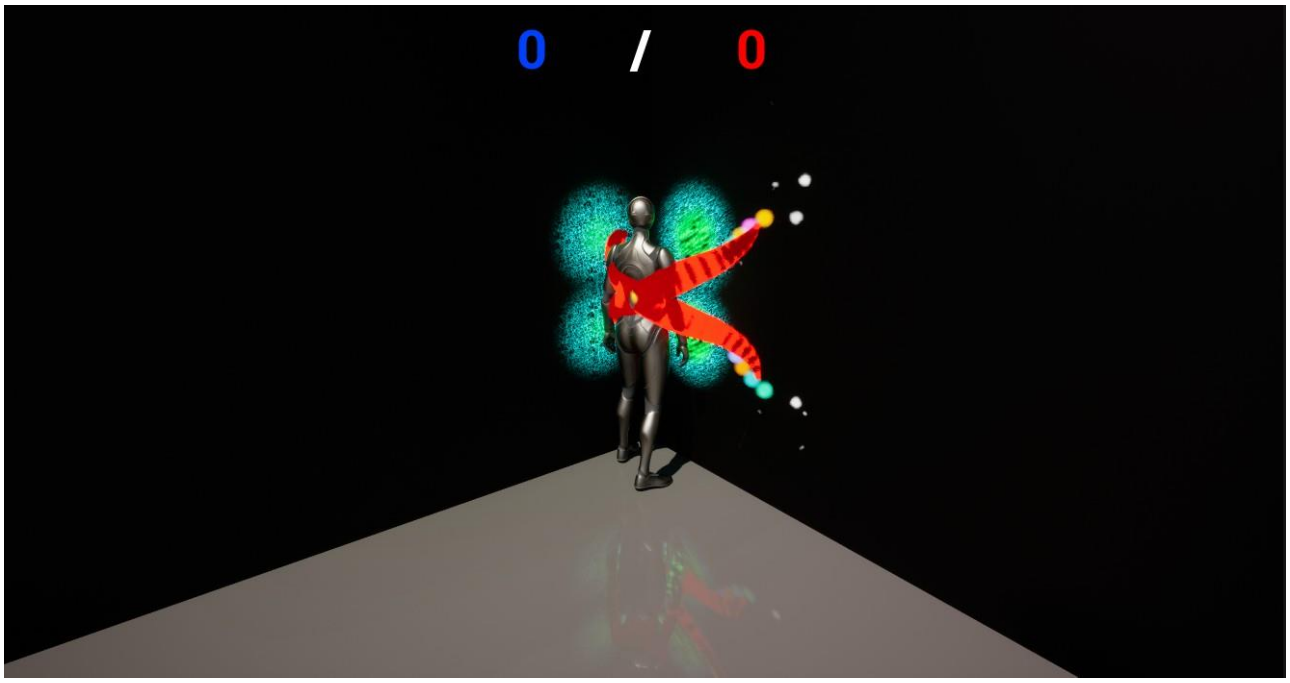
As you can see in the screenshot, the colour becomes fully red and encapsulates the whole particle.



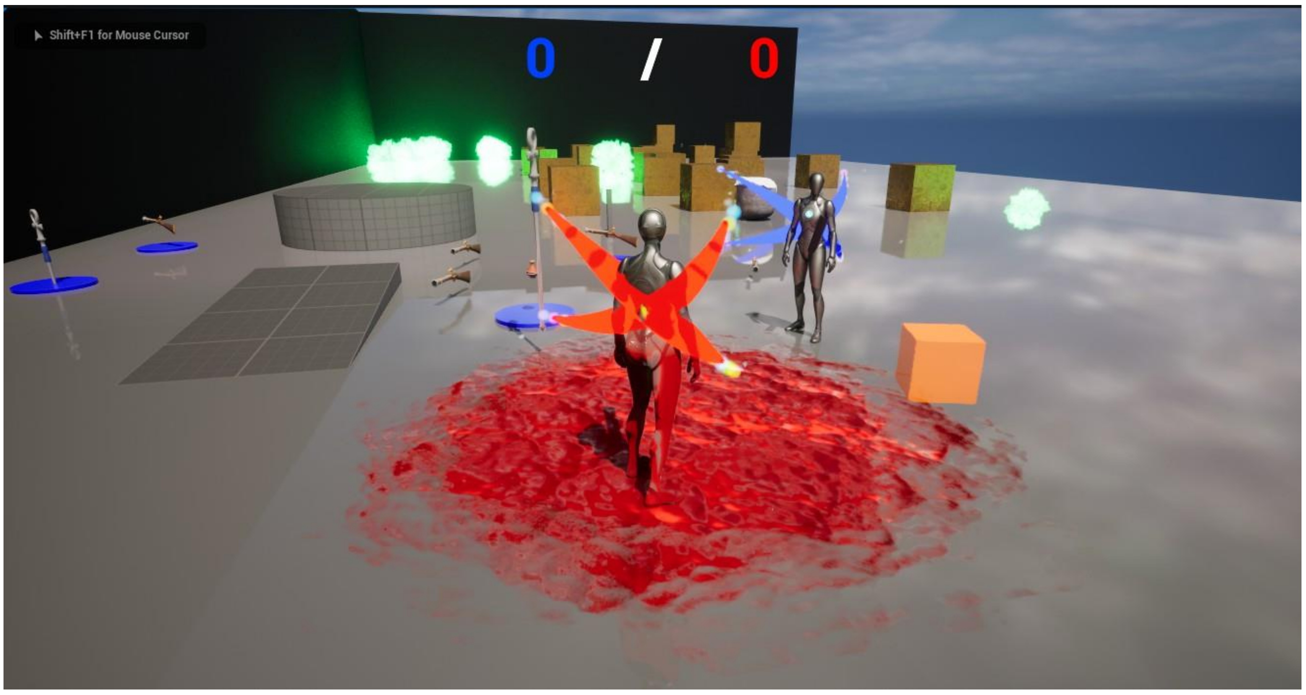
In this picture, it can be seen that the particle is starting to get close to the wall. A small colour change starts to appear when the particle is starting to touch the wall.



When the particles fully touch the wall, the wall material changes into a mix of cyan and green with a textured noise node.



It can be seen that each one of the 4 particles contribute to the changing materials of the wall, rather than just one. This proves that it didn’t use the player’s position but the particles itself.

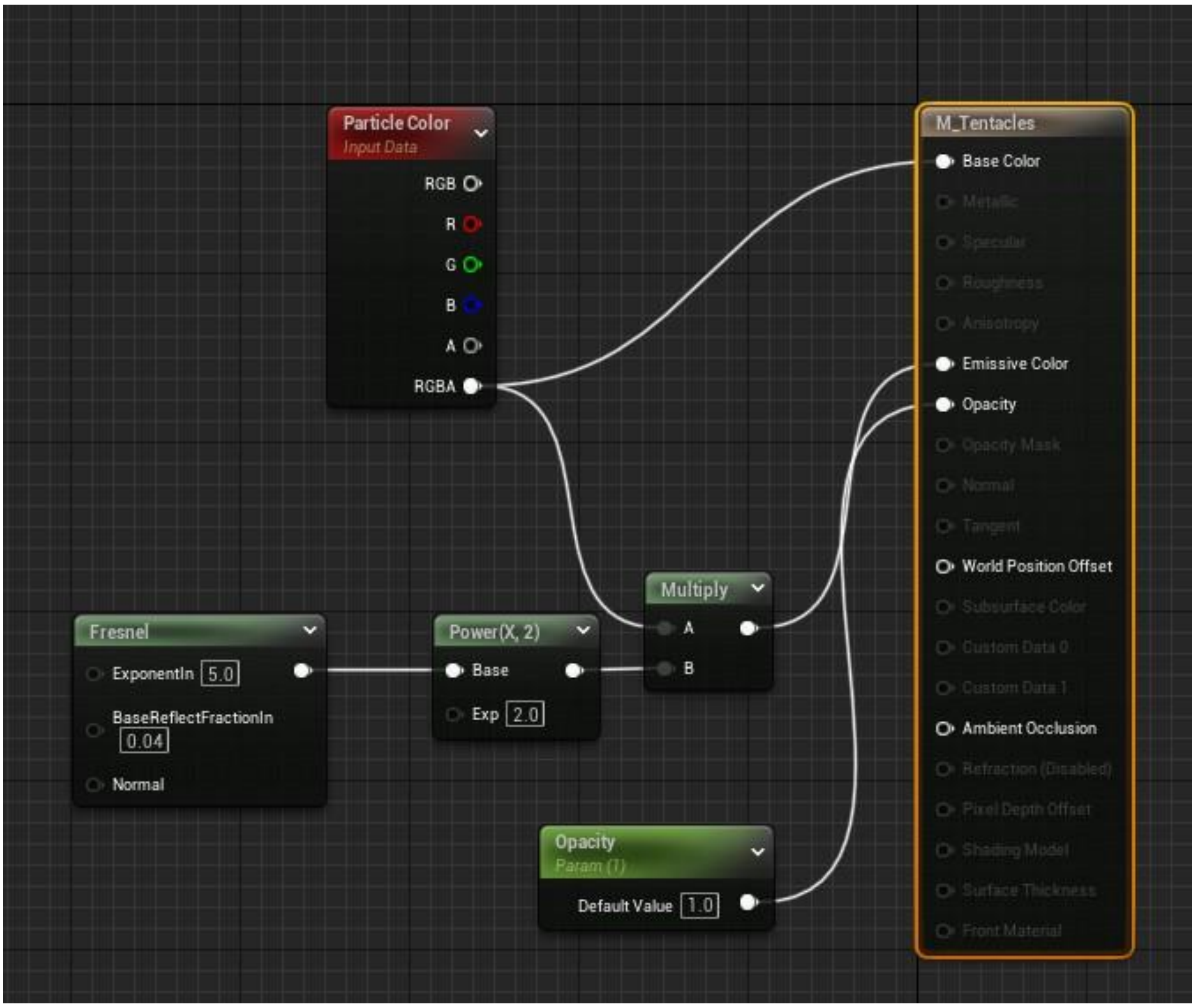


In this screenshot, it can also be seen that each player has a different colour with the player that I was controlling having the colour red and the other player having the colour blue.

Properties and Values

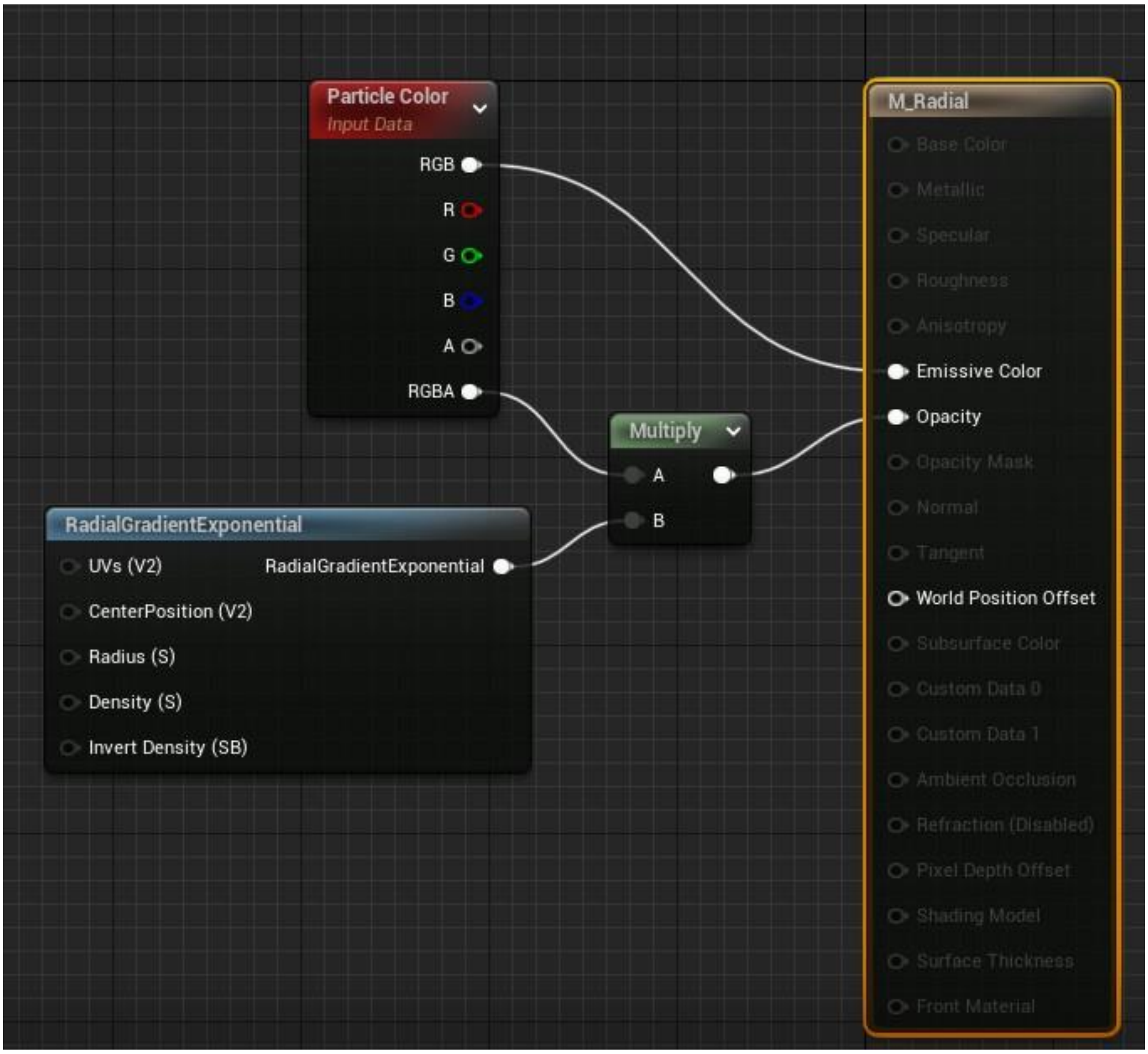
Custom Materials

M_Tentacles



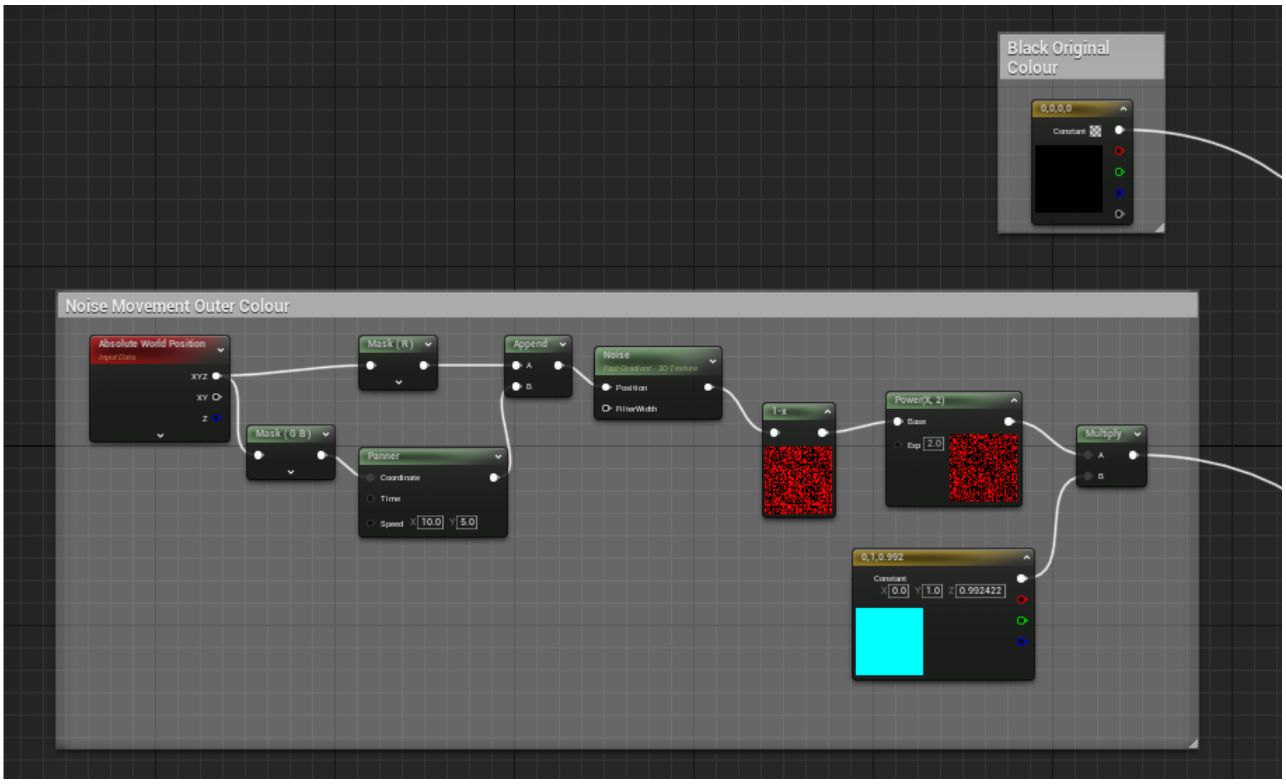
This material is used for the tentacle particles. It's fairly simple by mixing the Fresnel function and adding the particle colour. This enables it for the colour to be controlled dynamically.

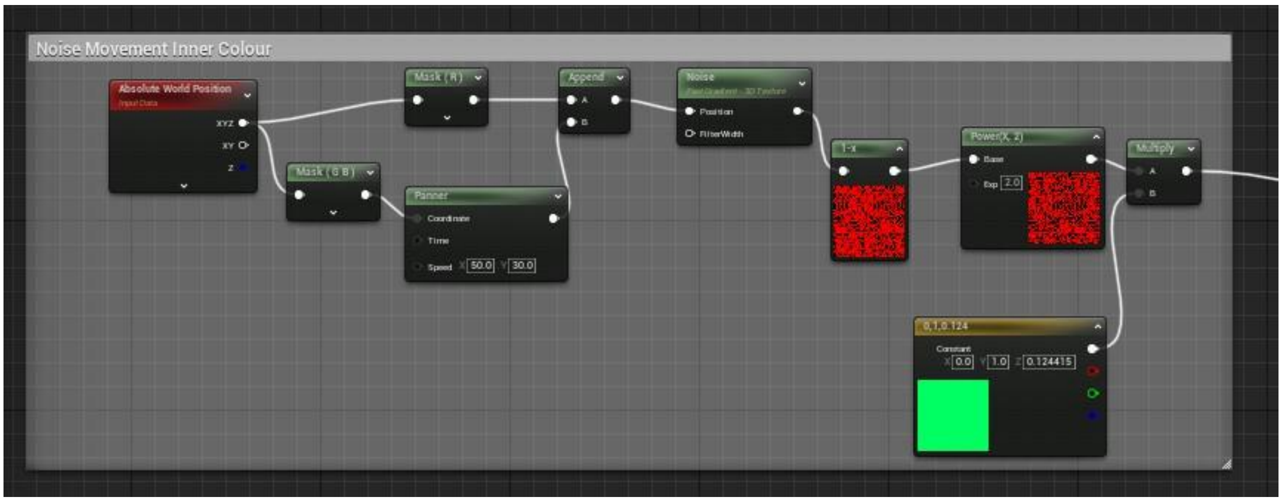
M_Radial

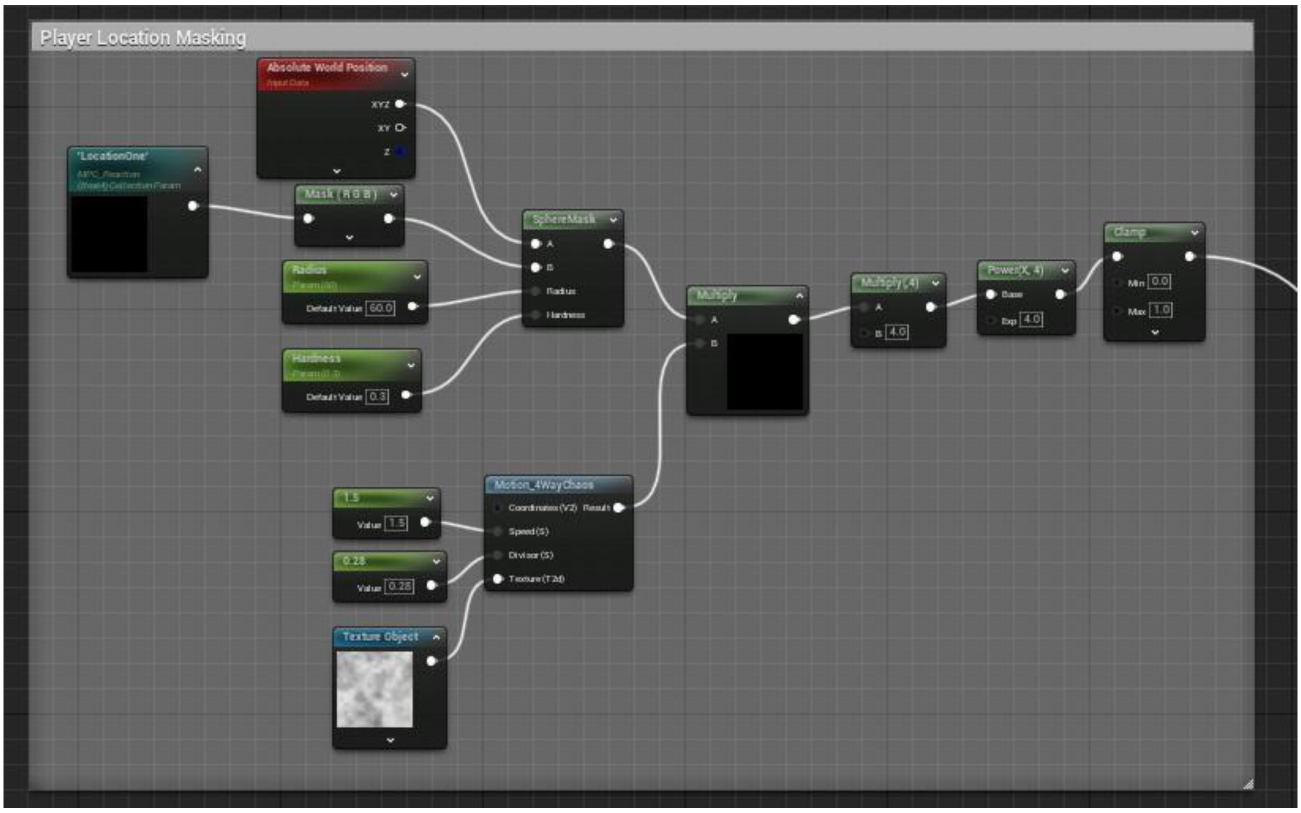


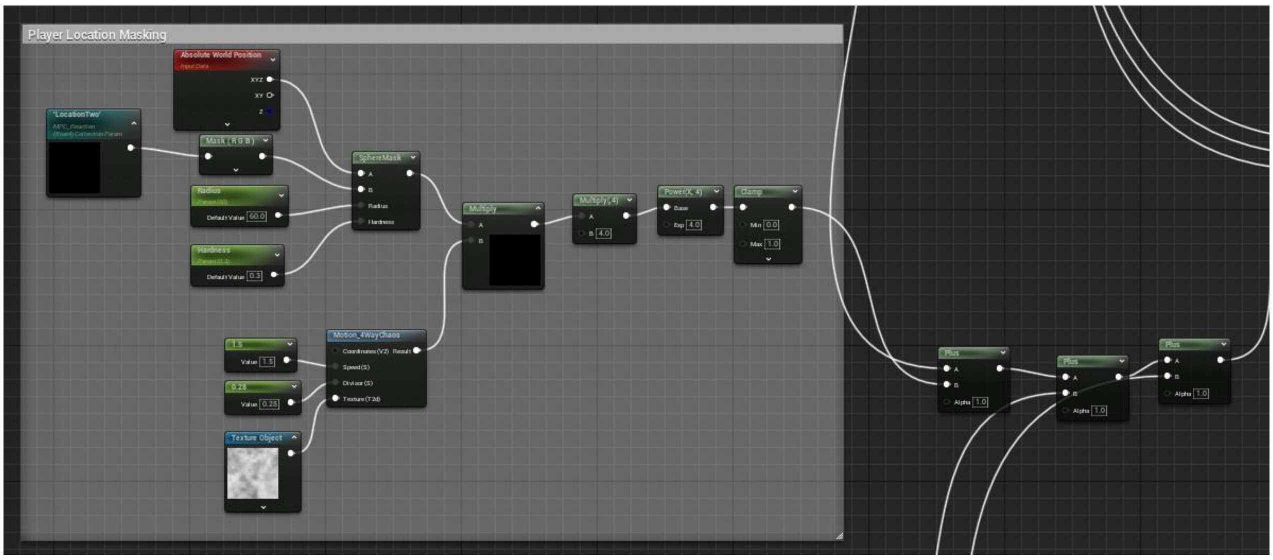
This Material is also very simple, it multiplies the RadialGradientExponential to the node for the particle dots colour.

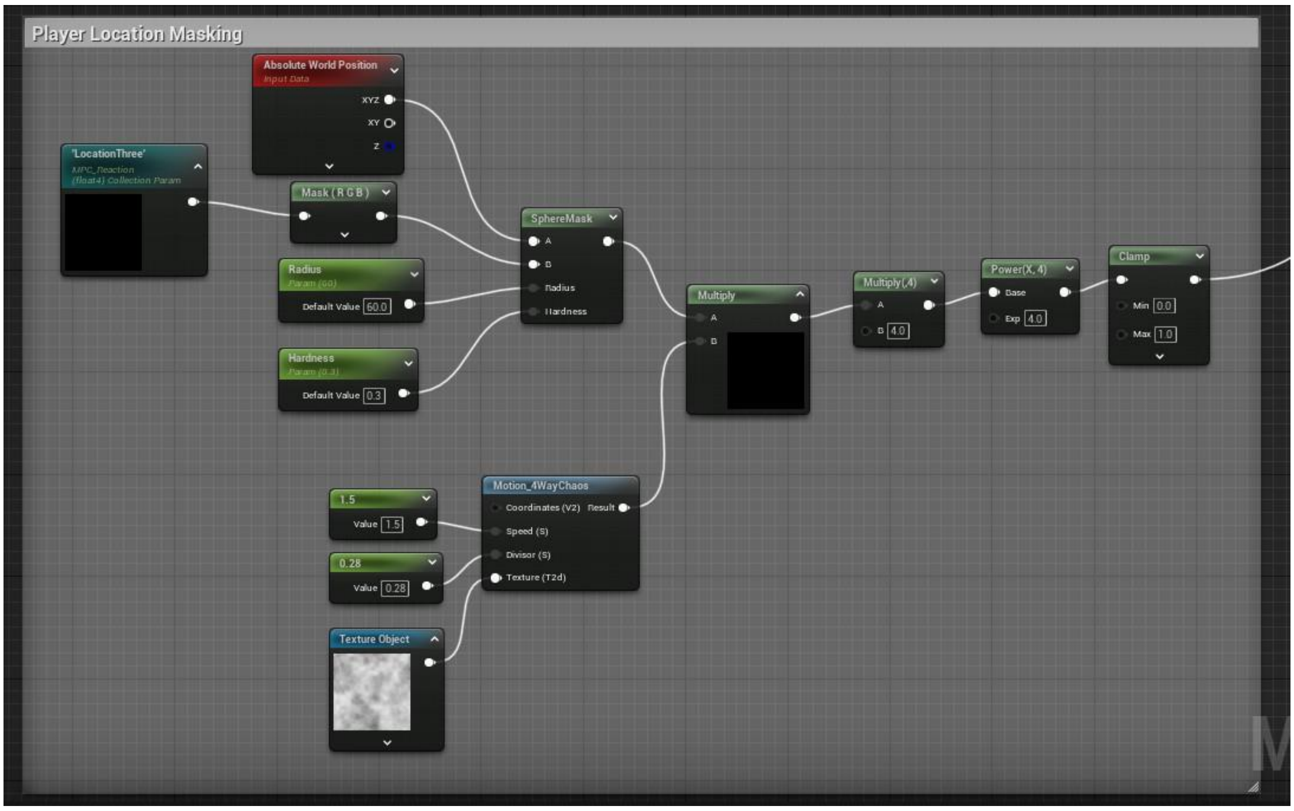
M_Reaction

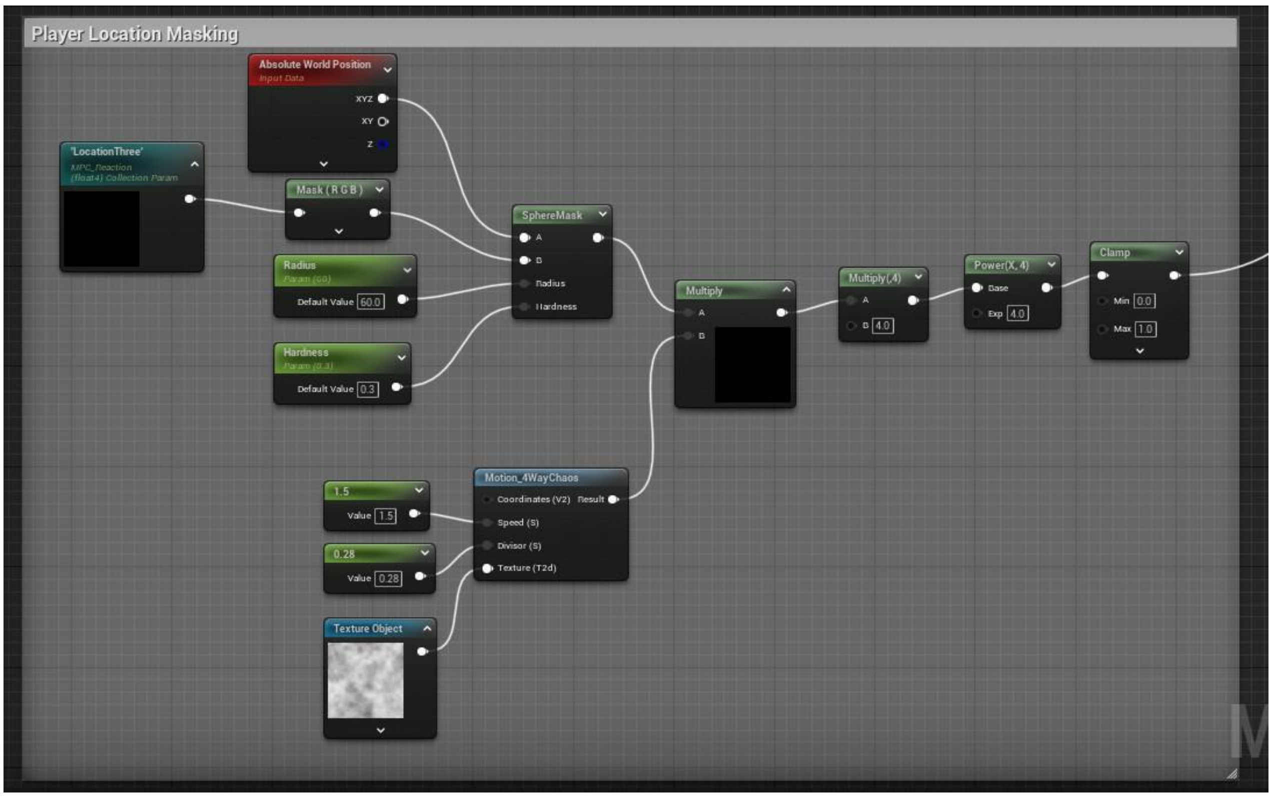

Black Original Colour
0,0,0,0
Constant
Noise Movement Outer Colour
Absolute World Position
Input Data
XYZ
XY
Z
Mask (R)
Mask (G B)
Append
A
B
Panner
Coordinate
Time
Speed X 10.0 Y 5.0
Noise
Fast Gradient - 3D Texture
Position
FilterWidth
1-x
Power(X, 2)
Base
Exp 2.0
Multiply
A
B
0,1,0.992
Constant
X 0.0 Y 1.0 Z 0.992422

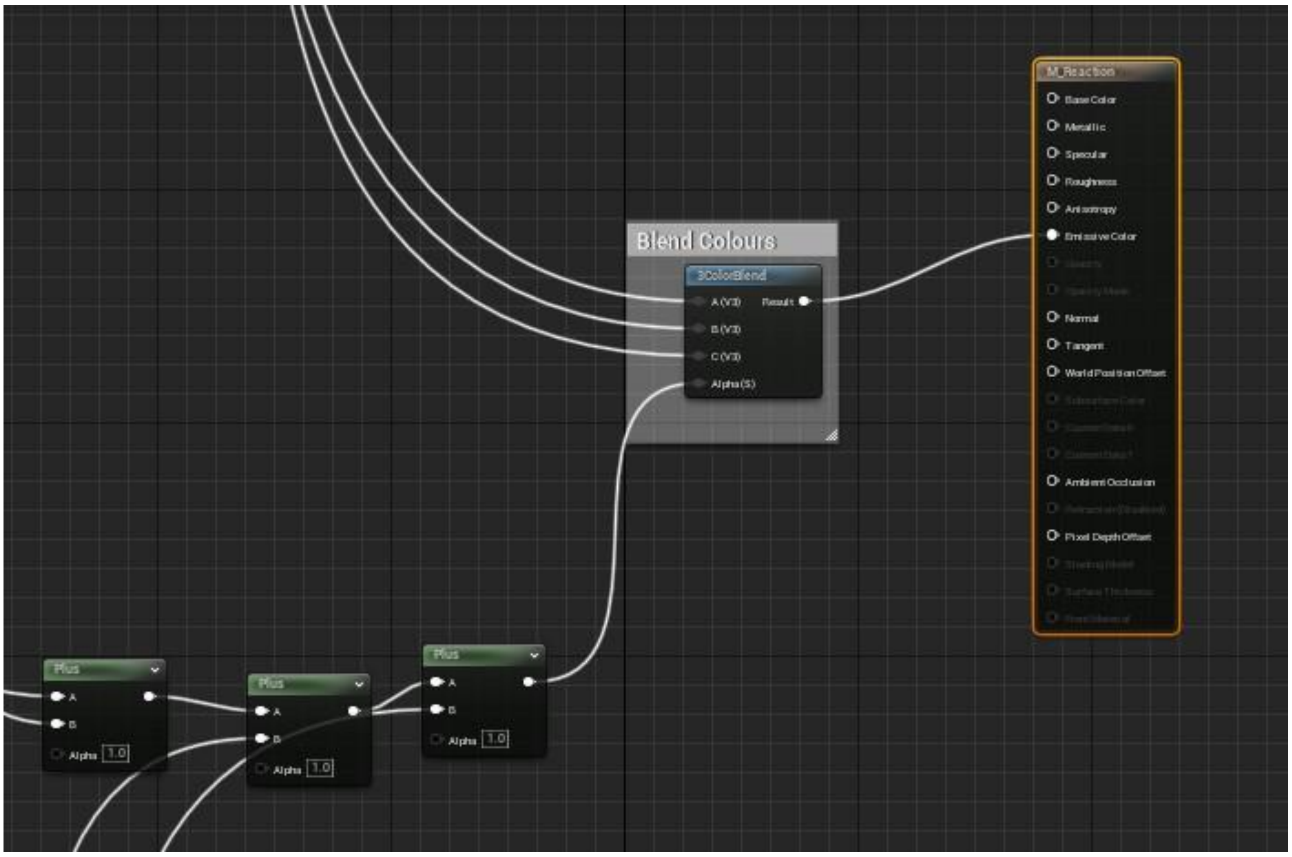

Noise Movement Inner Colour
Absolute World Position
Input Data
XYZ
XY
Z
Mask (R)
Mask (G B)
Append
A
B
Panner
Coordinate
Time
Speed X 50.0 Y 30.0
Noise
Fast Gradient - 3D Texture
Position
FilterWidth
1-x
Power(X, 2)
Base
Exp 2.0
Multiply
A
B
0,1,0.124
Constant
X 0.0 Y 1.0 Z 0.124415


Player Location Masking
Absolute World Position
XYZ
XY
Z
'LocationOne'
Mask (R G B)
Radius
Default Value 60.0
Hardness
Default Value 0.3
SphereMask
A
B
Radius
Hardness
Multiply
Multiply(,4)
B 4.0
Power(X, 4)
Base
Exp 4.0
Clamp
Min 0.0
Max 1.0
Motion_4WayChaos
Coordinates (V2)
Result
Speed (S)
Divisor (S)
Texture (T2d)
1.5
Value 1.5
0.28
Value 0.28
Texture Object


Player Location Masking
Absolute World Position
'LocationTwo'
Mask (R G B)
Radius
Default Value 60.0
Hardness
Default Value 0.3
SphereMask
Multiply
Multiply(,4)
Power(X, 4)
Clamp
Motion_4WayChaos
1.5
0.28
Texture Object
Plus
Alpha 1.0


Player Location Masking
Absolute World Position
Input Data
XYZ
XY
Z
'LocationThree'
MPC_Reaction
(float4) Collection Param
Mask (R G B)
SphereMask
A
B
Radius
Hardness
Radius
Param (60)
Default Value 60.0
Hardness
Param (0.3)
Default Value 0.3
Multiply
A
B
Multiply(,4)
A
B 4.0
Power(X, 4)
Base
Exp 4.0
Clamp
Min 0.0
Max 1.0
Motion_4WayChaos
Coordinates (V2)
Result
Speed (S)
Divisor (S)
Texture (T2d)
1.5
Value 1.5
0.28
Value 0.28
Texture Object


Player Location Masking
Absolute World Position
Input Data
XYZ
XY
Z
'LocationThree'
MPC_Reaction
(float4) Collection Param
Mask (R G B)
SphereMask
A
B
Radius
Hardness
Radius
Param (60)
Default Value 60.0
Hardness
Param (0.3)
Default Value 0.3
Multiply
A
B
Multiply(,4)
A
B 4.0
Power(X, 4)
Base
Exp 4.0
Clamp
Min 0.0
Max 1.0
Motion_4WayChaos
Coordinates (V2)
Result
Speed (S)
Divisor (S)
Texture (T2d)
1.5
Value 1.5
0.28
Value 0.28
Texture Object



Basically, this material is a bit more complicated. So it takes the generated particle's location and then it basically masks the wall based on the particle's location. Then, the sphere mask gets applied to the wall and changes the material on the wall partially.

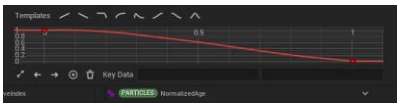
Emitters

Tentacles Emitter

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite

Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Rate	Number of particles per second to spawn	5.0
Lifetime Mode	Lifetime of the particle	Direct Set 1.0
Colour Mode	The colour of the particle	Direct Set : user.TentacleColor
Ribbon Width Mode	Determine the scale of the ribbon of the particle	Direct Set 20.0
Velocity Mode	The added velocity mode to the particle	Linear
Velocity Speed	The added speed of the velocity of the particle	X 0.0, Y 0.0, Z 100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Acceleration Force	Adds to Transient.PhysicsForce which will translate into acceleration within the solver.	X 200.0, Y 0.0, Z 0.0
Curl Noise Force Strength	Scales the sampled curl noise force vector	10.0
Curl Noise Force Noise Frequency	Modulates position to increase or decrease the rate at which curl noise is sampled.	50.0
Curl Noise Force Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.0, Y = 0.0, Z = 1.0

Collision Enabled	Enable or disable the module's affect on the effect	enabled
Radius Calculation Type	Each particle's collision radius will be calculated for you, per frame , using the methods laid out below.	Sprite
Method for Calculating Particles Radius	This enum changes the way that each particle's collision radius is calculated.	Bounds
Particle Radius Scale	This value scales the calculated particle collision radius	1.0
Restitution	This controls the particle's bounce coefficient. 1 will retain all of the particle's energy along the impact normal vector and 0 will remove it.	0.6
Simple Friction	If true, fewer parameters will be used to control the friction coefficient.	true
Friction	The friction coefficient defines how quickly a particle will slow down as it slides across a surface	0.25
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Generate Location Event	Generates an event which contains the position of the particle generating the vent.	Type : Send Rate 2.0

Scale Ribbon Width	Scales the initial Ribbon Width by a scale factor	Float from Curve  Templates 0.5 1 Key Data PARTICLES NormalizedAge
Export Particles Data to Blueprint	Export Particle Data to Blueprints or C++	Callback Handler Parameter : user.CallBack
Ribbon Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Tentacles

Colorful_Dots

Property	Description of Purpose	Value
Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Rate	Number of particles per second to spawn	10.0
Lifetime Mode	Lifetime of the particle	Direct Set 1.0
Colour Mode	The colour of the particle	Random : R 1.0, G 0.24167, B 0.0, A 1.0 Hue Shift Range X 0.0 Y

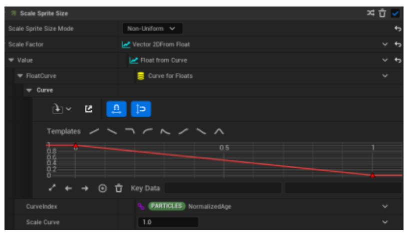
		100.0 Saturation Range 0.8 Y 1..0 Value Range X 0.8 Y 1.0 Alpha Scale Range X 0.8 Y 1.0
Ribbon Size Mode	Determine the scale of the sprites of the particle	Uniform 10.0
Velocity Mode	The added velocity mode to the particle	Linear
Velocity Speed	The added speed of the velocity of the particle	X 0.0, Y 0.0, Z 100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)
Acceleration Force	Adds to Transient.PhysicsForce which will translate into acceleration within the solver.	X 200.0, Y 0.0, Z 0.0
Curl Noise Force Strength	Scales the sampled curl noise force vector	10.0
Curl Noise Force Noise Frequency	Modulates position to increase or decrease the rate at which curl noise is sampled.	50.0
Curl Noise Force Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.0, Y = 0.0, Z = 1.0
Collision Enabled	Enable or disable the module's affect on the effect	enabled

Radius Calculation Type	Each particle’s collision radius will be calculated for you, per frame , using the methods laid out below.	Sprite
Method for Calculating Particles Radius	This enum changes the way that each particle’s collision radius is calculated.	Bounds
Particle Radius Scale	This value scales the calculated particle collision radius	1.0
Restitution	This controls the particle’s bounce coefficient. 1 will retain all of the particle’s energy along the impact normal vector and 0 will remove it.	0.6
Simple Friction	If true, fewer parameters will be used to control the friction coefficient.	true
Friction	The friction coefficient defines how quickly a particle will slow down as it slides across a surface	0.25
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active
Generate Location Event	Generates an event which contains the position of the particle generating the vent.	Type : Send Rate 2.0

Particle_Dots

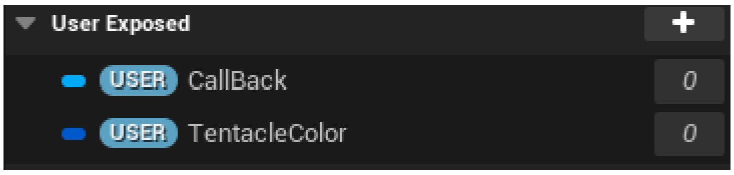
Property	Description of Purpose	Value

Life Cycle Mode	Determines whether the life cycle (Managing looping, age, and death) of the Emitter is calculated by the system that owns it, or by the emitter itself.	Self
Inactive Response	Determines what happens when the emitter itself enters an inactive state	Complete (Let Particles Finish then Kill Emitter)
Loop Behavior	Determines what happens when the Loop Duration is exceeded and what values are calculated.	Infinite
Loop Duration	Establishes the duration of the emitter life cycle.	2.0
Spawn Rate	Number of particles per second to spawn	10.0
Lifetime Mode	Lifetime of the particle	Direct Set 1.0
Colour Mode	The colour of the particle	Random : R 1.0, G 0.24167, B 0.0, A 1.0 Hue Shift Range X 0.0 Y 0.0 Saturation Range 0.8 Y 2.0 Value Range X 0.1 Y 1.0 Alpha Scale Range X 0.1 Y 1.0
Sprite Size Mode	Determine the scale of the sprites of the particle	Random Uniform Between 0.1 and 10.0
Velocity Mode	The added velocity mode to the particle	Linear
Velocity Speed	The added speed of the velocity of the particle	X 0.0, Y 0.0, Z 100.0
Particle State	Manages Particle Age / Lifetime	Kill Particles When Lifetime Has Elapsed (true)

Scale Sprite Size	Takes the initial sprite scale as set in the spawn script, and scales it by a scale factor	Uniform Curve –> Curve for Floats  Scale Sprite Size Scale Sprite Size Mode: Non-Uniform Scale Factor: Vector 2DFrom Float Value: Float from Curve FloatCurve: Curve for Floats Curve CurveIndex: PARTICLES NormalizedAge Scale Curve: 1.0 Templates Key Data 1 0.8 0.6 0.4 0.2 0 0.5
Gravity Force	Applies a gravitational force (in cm/s) to Transient.PhysicsForce	X 0.0, Y 0.0, Z 24.734362
Curl Noise Force Strength	Scales the sampled curl noise force vector	500.0
Curl Noise Force Noise Frequency	Modulates position to increase or decrease the rate at which curl noise is sampled.	50.0
Curl Noise Force Pan Noise Field	Moves the sample position via Emitter Age plus a random float generated by the determinism flag to create a more random feeling sample	X = 0.0, Y = 0.0, Z = 1.0
Drag	Applies Drag directly to particle velocity and/or rotational velocity, irrespective of Mass.	1.0
Rotational Drag	Reduces each particle's rotational velocity	1.0
Solve Forces and Velocity	Takes the values accumulated into Transient.PhysicsForce, Multiplies by Engine.DeltaTime and adds to the current Velocity. Outputs the updated Particles. Velocity and Particles.Position.	Active

Event Handle Properties Source	Source of the emitter and event	Emitter : Tentacles Event LocationEvent
Receive Location Event	Receives a Location event, such as one generated by the "Generate Location Event" Module. Optionally writes that event payload directly to the receiving particles' attributes	Enabled
Shape Primitive	The shape of what the particle will spawn into	Sphere
Sphere Radius	The radius of the sphere primitive	8.0
Sprite Renderer Material	The material used to render the particle/ Not that it must have the Use with Niagara Sprites flag checked.	M_Radial

C++ Interaction Description


User Exposed

USER CallBack 0

USER TentacleColor 0

Basically the way the C++ interaction works is very simple. It takes the user.CallBack function to trigger the collision between the particles and the environments. It also takes the user.TentacleColor function to change the colours based on the players. I also made a material parameter collection to store the vector of the location that was provided from the collision.

```
void AA01_BlockoutShooterCharacter::MulticastReactWall_Implementation(const
TArray<FBasicParticleData>& Data)
{
  for(int i = 0; i < Data.Num(); i++)
  {
      if(SpawnedTentaclesOne)
      {
         FLinearColor TentacleColor = GameState->GetTeamColor(GetPlayerState());
         TransparentMatCollectionInst =
GetWorld()->GetParameterCollectionInstance(TransparentMatCollection);

         if(TransparentMatCollectionInst)
```

```
        {
           TransparentMatCollectionInst->SetVectorParameterValue("LocationOne",
SpawnedTentaclesOne->GetComponentLocation() + FVector(0,0,30.0));

SpawnedTentaclesOne->SetNiagaraVariableLinearColor(FString("TentacleColor"),Tentac
leColor);

           TransparentMatCollectionInst->SetVectorParameterValue("LocationTwo",
SpawnedTentaclesTwo->GetComponentLocation() - FVector(0,0,30.0));

SpawnedTentaclesTwo->SetNiagaraVariableLinearColor(FString("TentacleColor"),Tentac
leColor);

           TransparentMatCollectionInst->SetVectorParameterValue("LocationThree",
SpawnedTentaclesThree->GetComponentLocation() + FVector(0,0,30.0));

SpawnedTentaclesThree->SetNiagaraVariableLinearColor(FString("TentacleColor"),Tent
acleColor);

           TransparentMatCollectionInst->SetVectorParameterValue("LocationFour",
SpawnedTentaclesFour->GetComponentLocation() - FVector(0,0,30.0));

SpawnedTentaclesFour->SetNiagaraVariableLinearColor(FString("TentacleColor"),Tenta
cleColor);
        }
     }
  }
}
```
I also added offsets to when the particle sends the location of the material parameter collection. I made it so that since each particles' location is technically one location, I made it so that it is distinct and it is offset so that it touches the edges of the particles rather than the supposedly middle part of the particles. It also takes the colour of the player's team and then it sends the value when it collides. Since the particle technically collides with the player's mesh, it changes the colour straight away when the player starts moving.

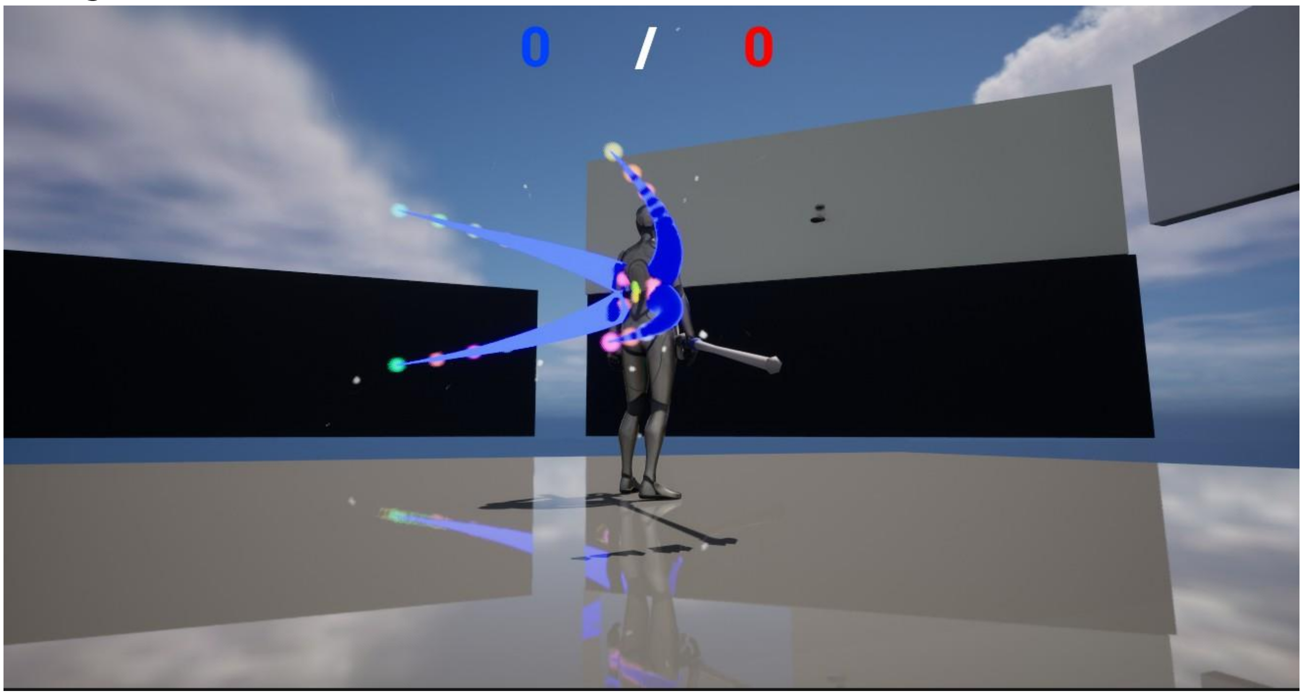
The impact of the C++ interaction is that although it is not really complicated, it provides visual feedback when the player's tentacles touch the wall and also provides an identifier of the tentacles colour to the player's team.

Custom Geometry Collection

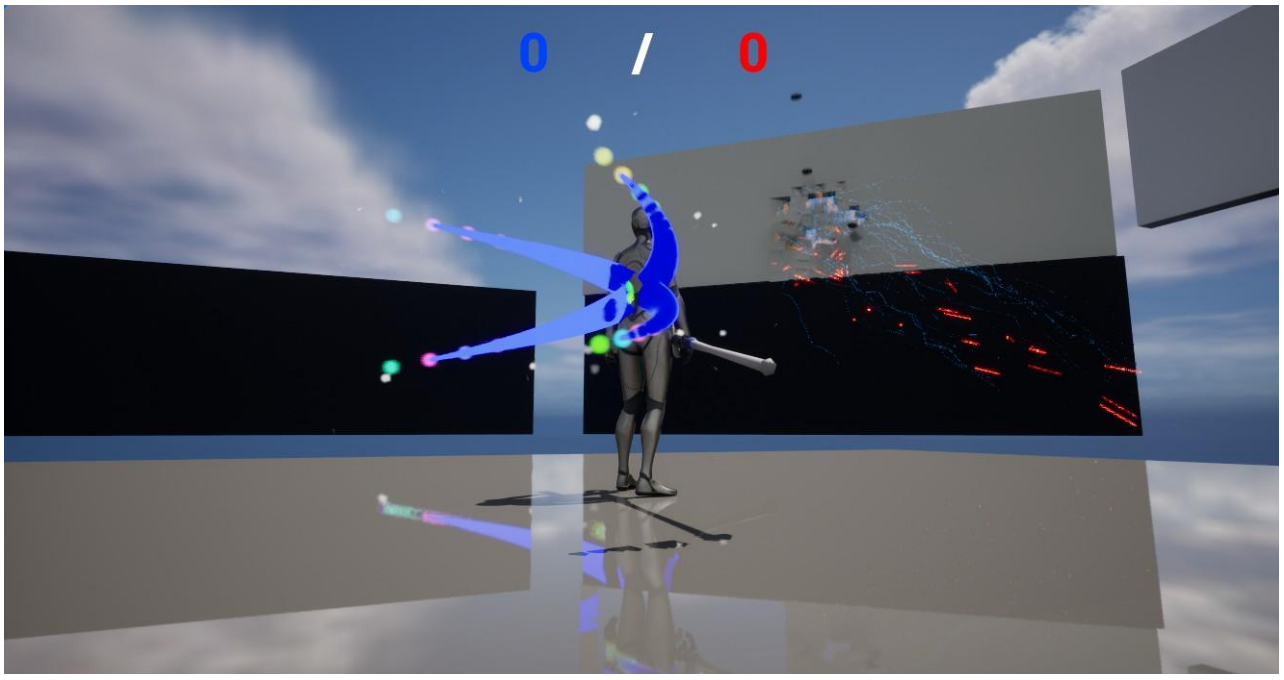
Overview of Effect

The effect that I made for my custom geometry collection is very simple. It's basically a basic wall but instead of a regular fracturing, it simulates bricks. The robustness of the bricks and the brick shapes of the wall adds this effect of immersion when the player shoots it. This creates a very pleasing and satisfying wall to break within the game.

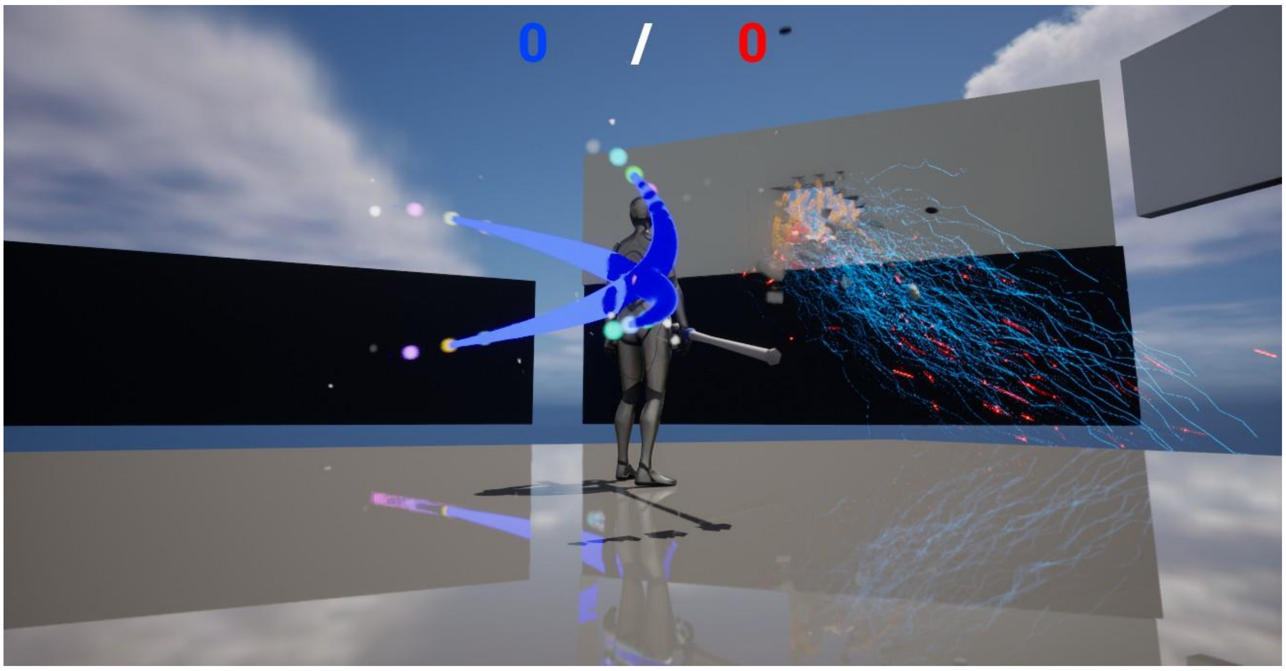
In-Engine Screenshots



This screenshot shows how the wall looks and when the wall is about to get hit by projectiles.



This screenshot shows when the wall gets hit by a projectile. As it can be seen, a very clear brick outline is visible with bricks about to fall.



The screenshot shows the bricks all falling over after being hit by the projectile. It is very clear that the shapes of the bricks are squares showcasing that they are in fact bricks.

Shader Effects

Shader Effect 1 - Stylized Water Shader

Overview of Effect

The effect is a very standard stylized water shader. It has a very stylized style with adjustable colours. The colours that I used for it are the blue colour for the standard water, the orange colour for the lava colour, and the black colour for the noir water. This effect uses a noise node with a texture of the number 0 to simulate the stylized circles effect of the water. The water shader's dynamic capabilities are triggered through a hitbox that is attached to the water component. When the player enters the hitbox, the opacity changes to 0.5 and the wave direction of it also changes to a faster movement. All of this affects the visual aspect of the plane itself but it does not change the mesh.

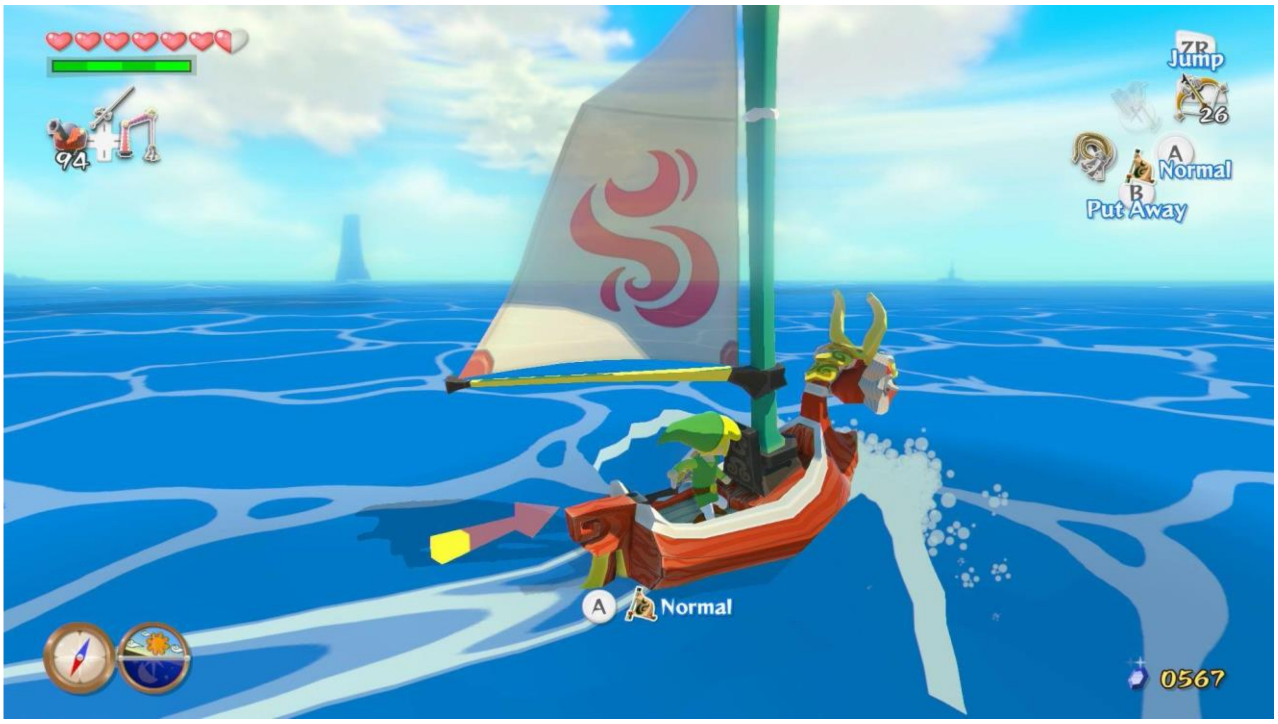
The intended impact on the gameplay is that it simulates water really well and can increase the immersiveness of the gameplay. This can lead to a more enjoyable experience when the player is playing the game. The other effect of the C++ interaction of the shader is indication. With the opacity change and the speed of the wave changing, it can be a very clear indicator that a player is inside the water. All of this creates an aesthetically pleasing water shader that is implemented in the gameplay.

Effect Description

As mentioned before in the overview, the look of the shader is very stylized. It is not similar to a typical water shader where it usually has low opacity and reacts like actual water but it goes the other direction. The colour of it is adjustable with the default colour being blue. The other colours that I put in the game are orange and black. The lines of the stylized part of the water has a white colour with a 0 texture that is attached to a noise node, making it random. There are also movements of the water shader. The movements are the wave movements and the colour movements. The colour movements are meant to simulate small wave movements while the wave movements are there to simulate the wave movements.

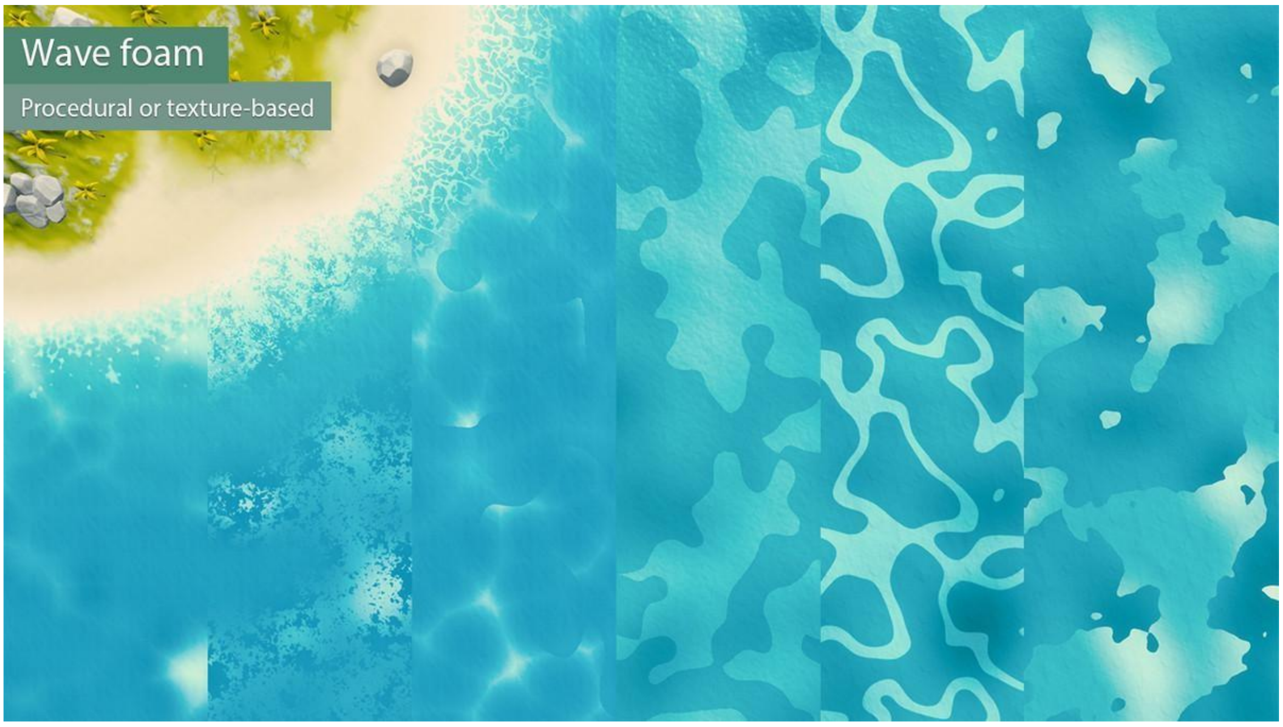
Inspiration / Reference Images:

There are a lot of games that use the stylized water shader in their games. Some of the games that I got the inspiration from are :



Source : https://danielilett.com/2020-04-05-tut5-3-urp-stylised-water/

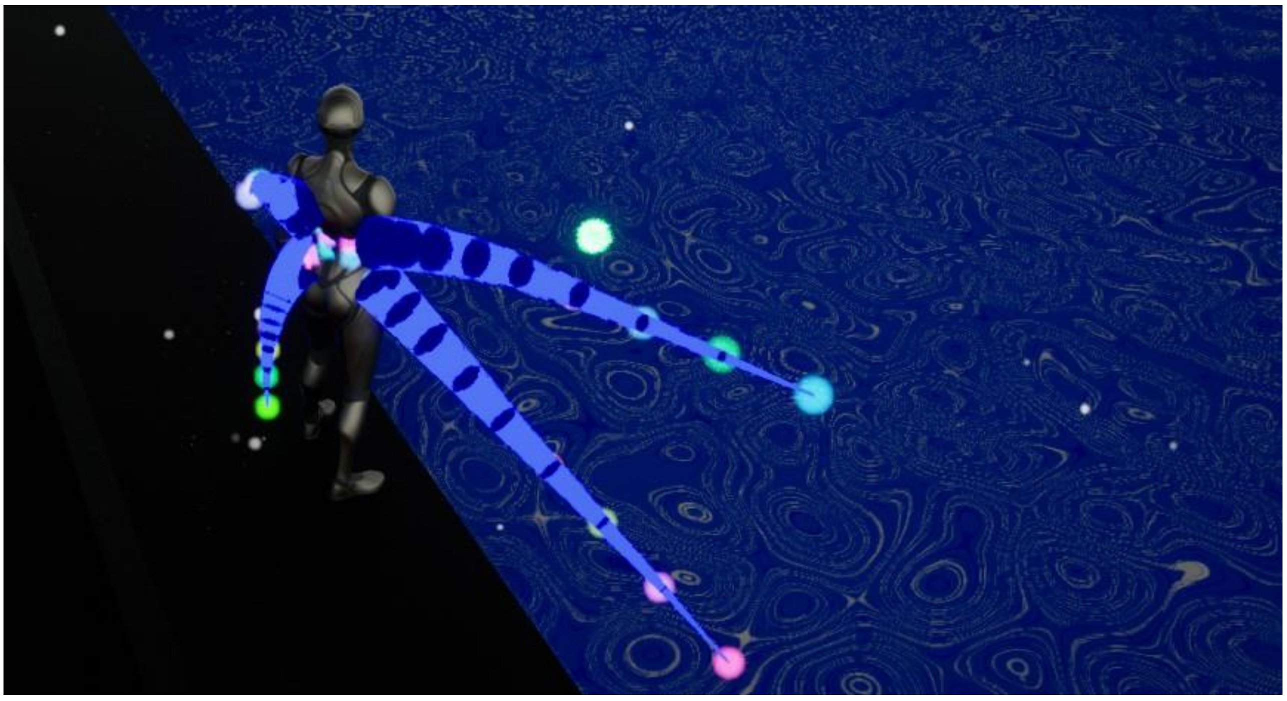
The Legend of Zelda: The Wind Waker has a very stylized art direction in it. The water shader inspired me to make my own with a similar feel in it. By similar feel, I meant the style of the lines, and also the water colour itself.



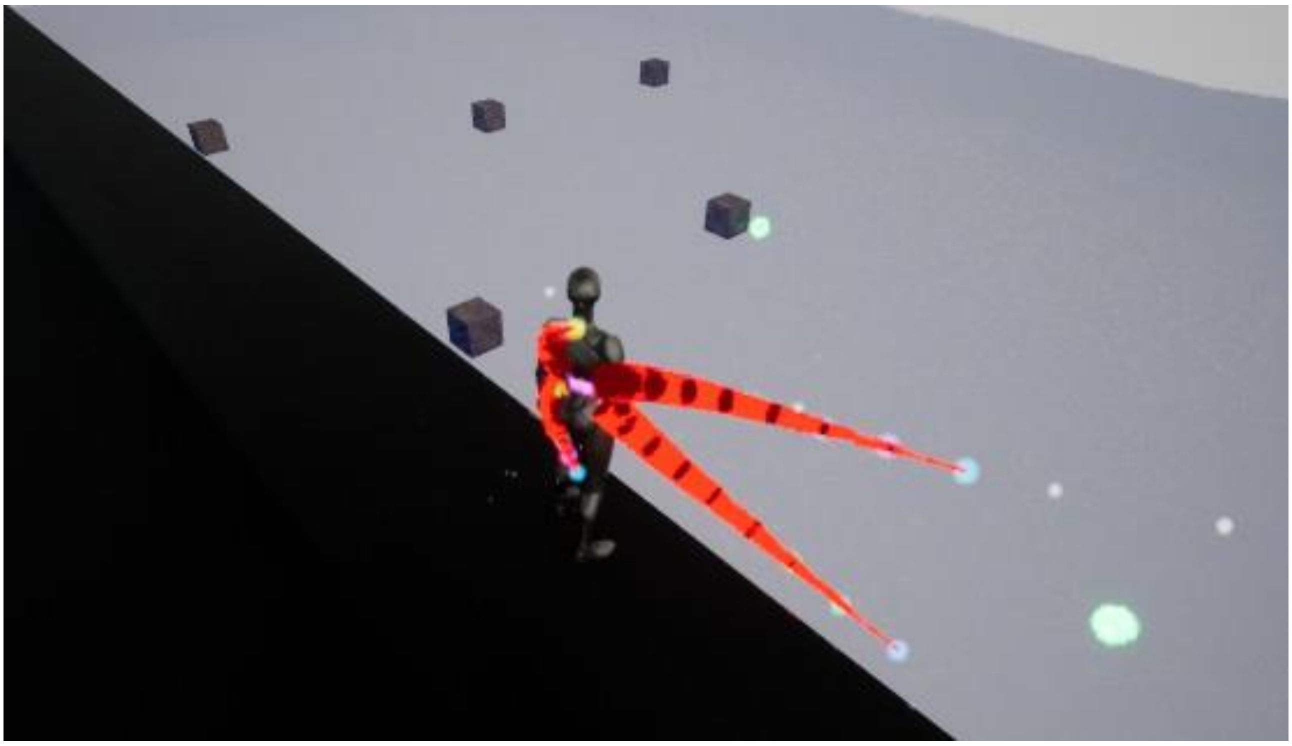
Source : https://assetstore.unity.com/packages/vfx/shaders/stylized-water-shader-71207

I also got some inspiration from the unity asset store that showcases stylized water shaders. These water shaders put me in the right direction on where I want my water shader to head.

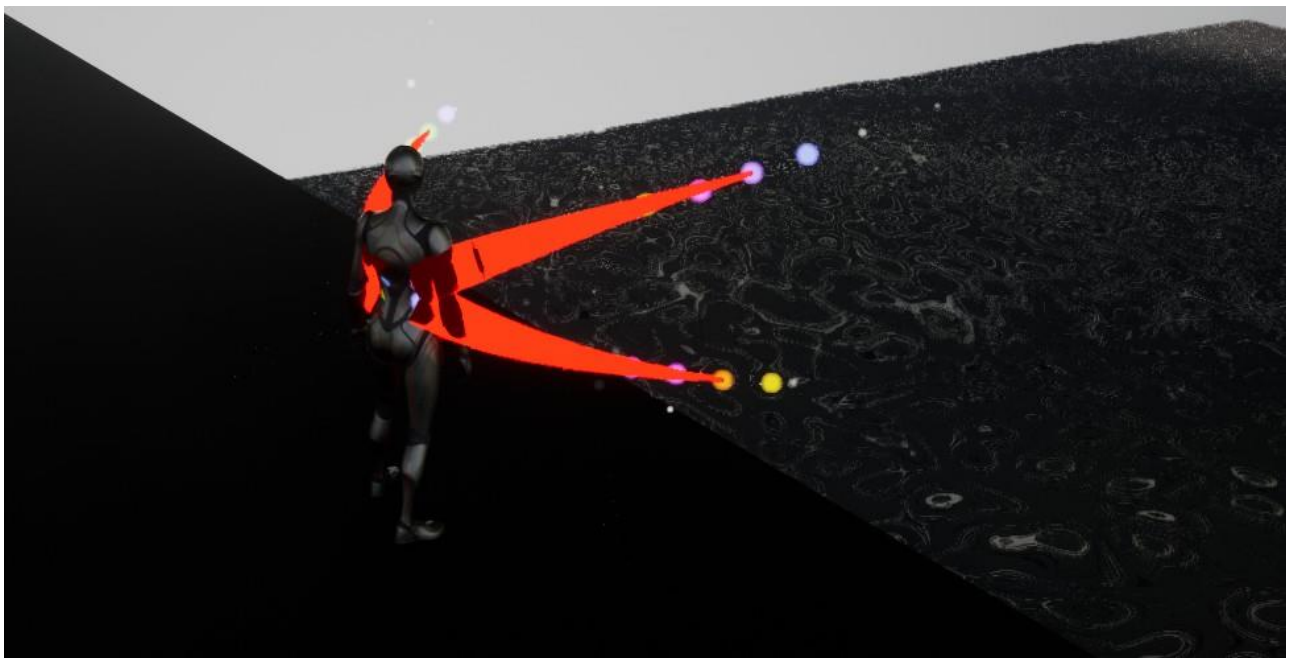
In-Engine Screenshots:



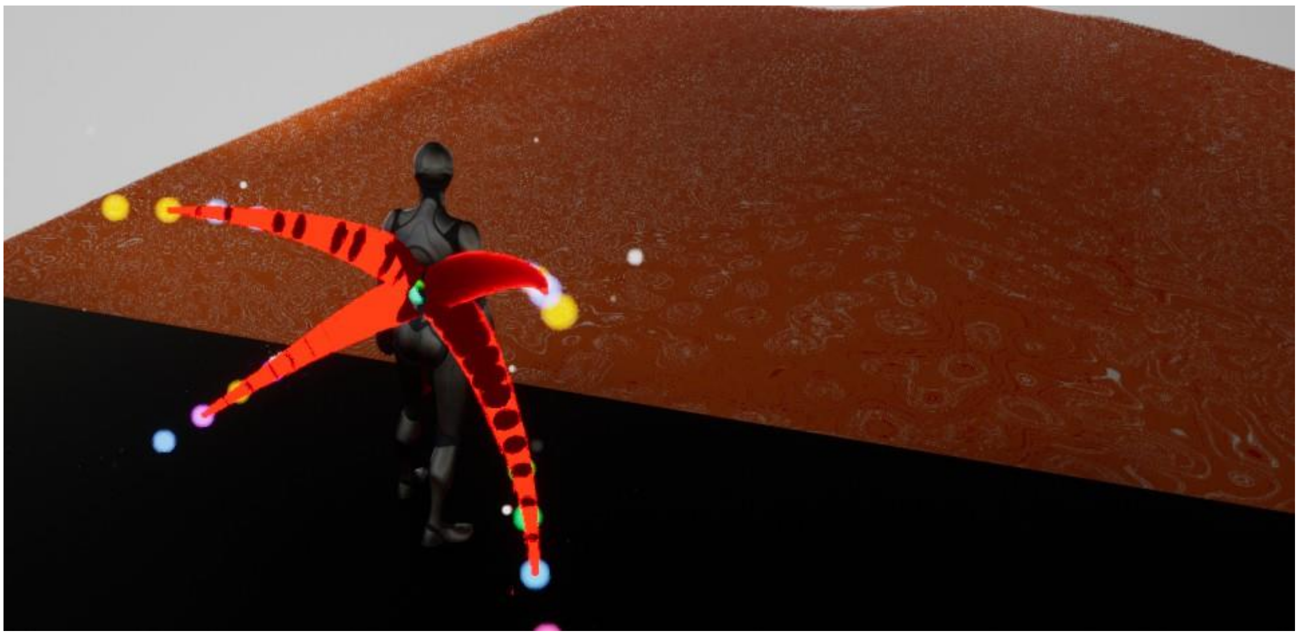
This is the state of the water shader when it is not touched or entered. This has a blue colour which is the default colour.



This screenshot showcases the water when a player is inside the water. In terms of opacity, it can clearly be seen that it is way lower. The wave's speed has also increased though it is not clear to be seen in the screenshot.



This is one of the variations of the water shader which is the noir water. It has overall the same values and behaviour with the normal water shader with the only difference being black coloured.



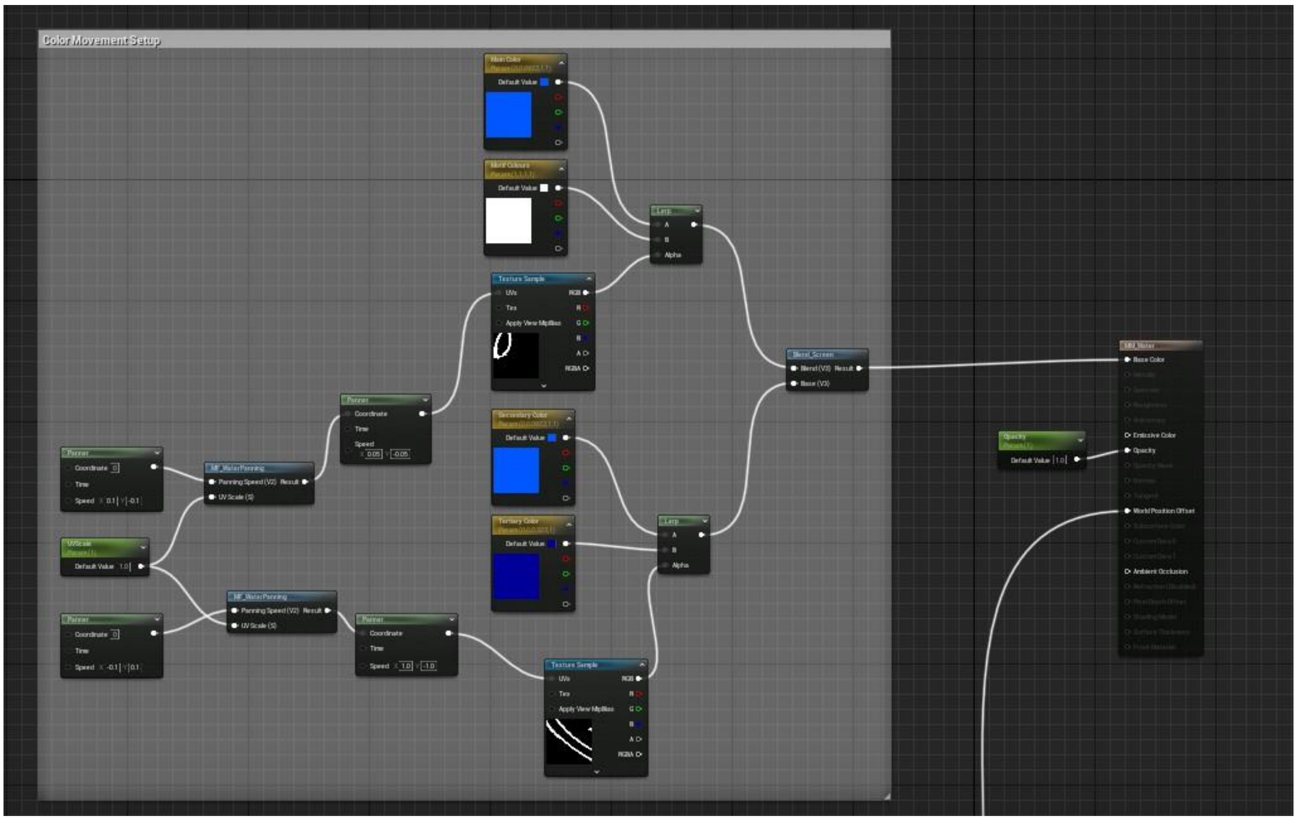
The other variation of this has the colour orange to simulate a stylized lava.

Properties and Values

Property	Description of Purpose	Value
Main Color	Used to control the main colour of shader dynamically.	Float4 (0, 0.923, 1, 1)
Motif Colours	Used to control the stylized part of the colour of the shader dynamically	Float4 (1, 1, 1, 1)
Secondary Color	Used to control the secondary colour of the shader dynamically	Float4 (0, 0.0923, 1, 1)
Tertiary Color	Used to control the tertiary colour of the shader dynamically	Float4 (0, 0, 0.323, 1)
UVScale	Used to control the UV scaling dynamically	Scalar (1.0)
WaveDirection	Used to control the direction of the wave dynamically	Float4 (1, 1, 1, 0.001)
WaveOffset	Used to control the offset of the wave dynamically	Float4 (0, 0, 10, 1)
Opacity	Used to control the opacity of the shader	Scalar (1.0)

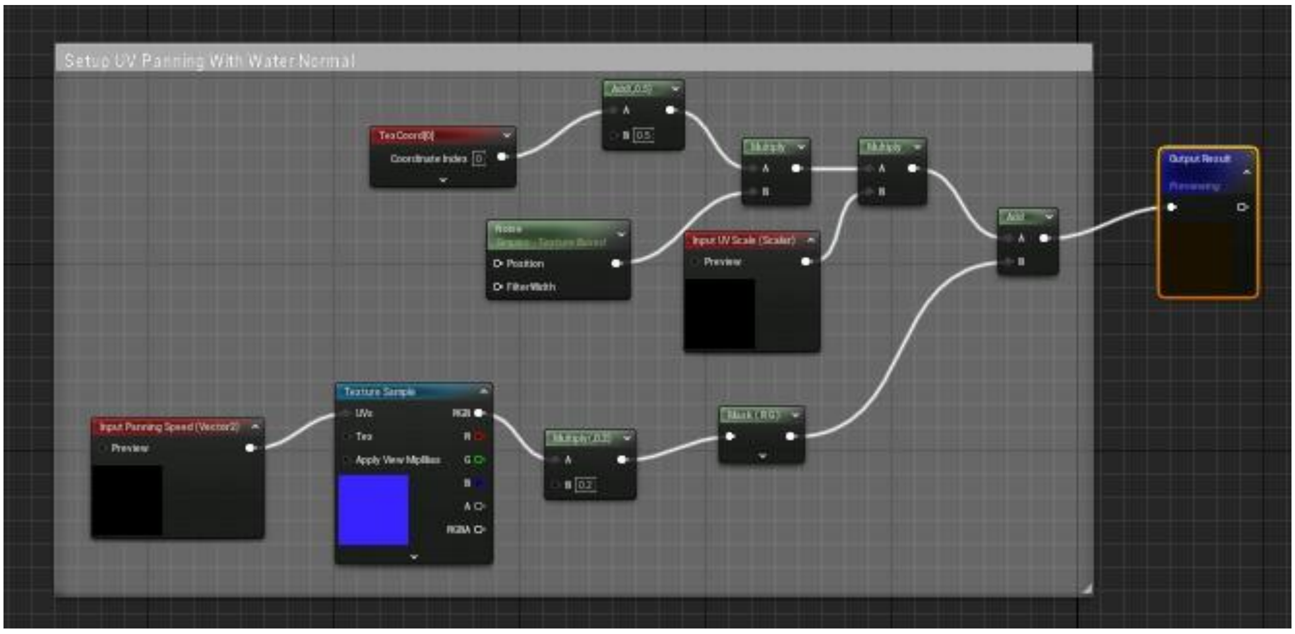
	dynamically	

Node Graph

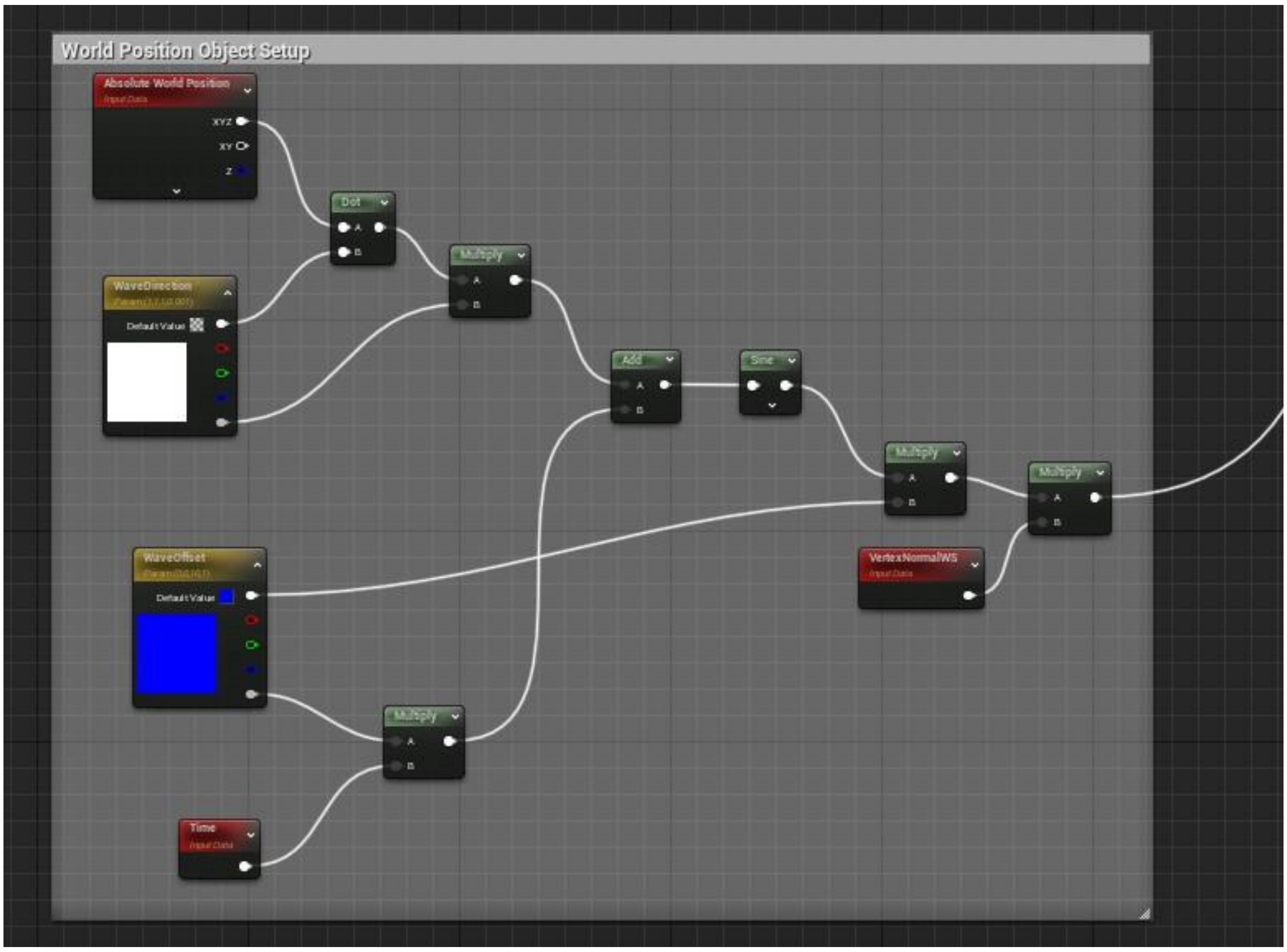


These node graphs are for the colour movement of the shader. It uses 4 different colours for each component. It uses the texture sample of 0 that is combined through a Panner that uses a custom material function called MF_WaterPanning. With all the 4 different colours, it is combined using a material function called Bled_Screen that is put into the Base Colour.

MF_WaterPanning



In this material function that was used called MF_WaterPanning, It uses the Noise node to add variation into the texture sample. It also uses the blue texture sample as the default to set up the panning speed with the Noise node that is multiplied with the input UV scale that was set up. The TexCoord also plays a vital role in this function in creating a proper text coordinate of the shader.



These nodes are used to set up the positioning of the water shader. I wanted to adjust the movement and this plays a vital role in it. It starts with the Absolute World Position that is combined with the dot product of the wave direction and then multiplied and added with the wave offset that is multiplied with the Time node. This is then multiplied into the VertexNormalWS node and then connected into the World Position Offset. This causes the object to have an illusion of it moving around simulating a wave even though it's just the shader doing the work.

Shader Effect 2 - Glitcher

Overview of Effect

The effect is called the Glitcher because it looks like it's glitching. This is very in theme with the whole game with the Chaos Mace mechanic and the glitching theme. The colour of this shader is Green with spikes that look like it's coming in and out creating a glitch effect. The Glitcher is very in line with the Chaos Mace mechanic. It reacts whenever the player attaches the Chaos Mace into the object and then it reacts by making the movement of the glitching effect faster and changing the colour to blue which is controlled by C++.

This effect is very simple but it is quite effective. It creates the impact of the Chaos Mace mechanic and also makes the game more immersive overall. Since the theme of the game is “A Glitch in the System”, the Glitcher is a perfect addition of a shader. The shader also reacts differently to how the mesh is shaped. If the mesh is a cube, then the shader will leave gaps in the object making the object extra glitchy. If the mesh is shaped like a sphere, then the shader will have sharp edges that are going in and out of the mesh. This can create a variety of objects that can adhere to the game mechanics for future development.

Effect Description

The shader has a default colour of green. The Shader also has an effect that looks glitchy with spikes going in and out if put into a sphere mesh. The shader also changes dynamically based on what actions the player does to the mesh with the shader attached mainly connected to the Chaos Mace mechanic. When the player attaches the cable component into the object, it enables the player to trigger the Chaos Mace mechanic. There are also indicators that can indicate this through the shader which is the colour changing from green to blue and the speed of the spikes going in and out.

Inspiration / Reference Images:

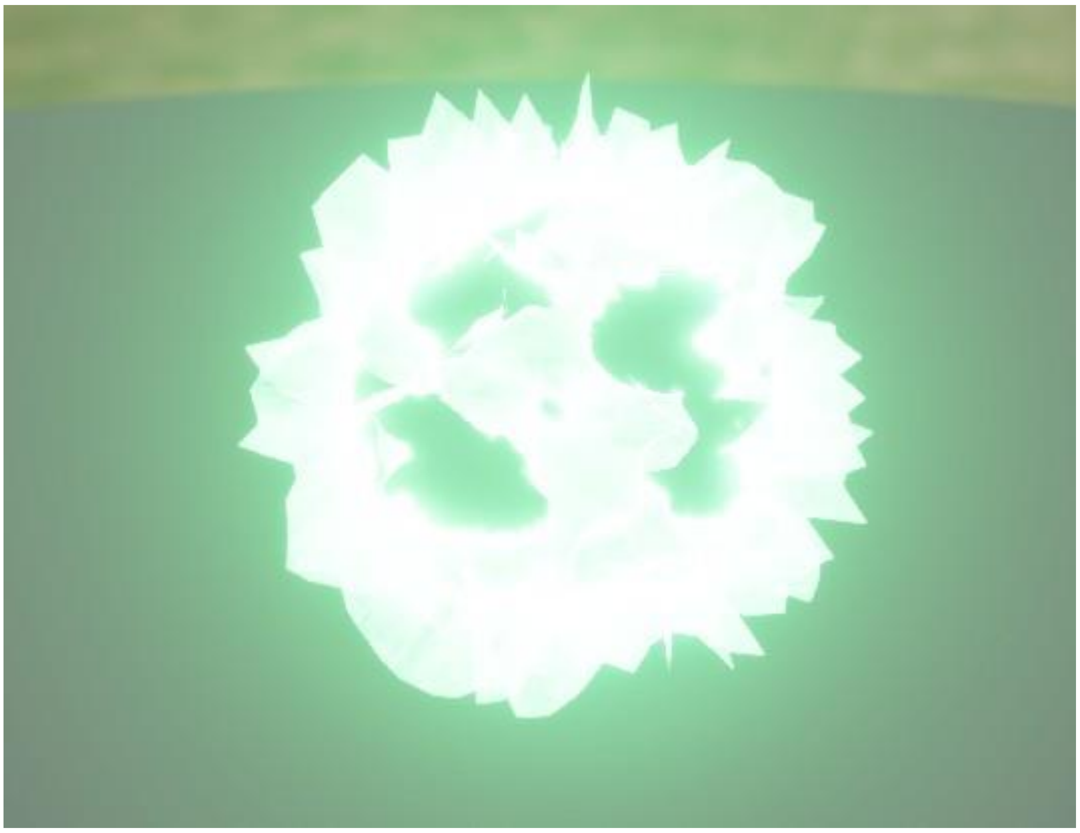
One of the things I was inspired by is the ball ray that causes static electricity. When I was growing up, I used to go to science museums and these things were always interesting to me. Therefore, I wanted to make a ball of lightning that looks similar to this. Then after some development, it inspired the Glitcher even if it looks vastly different.

Source : https://www.amazon.com.au/Katzco-Plasma-Ball-Electricity-Plug/dp/B089KTFXJB

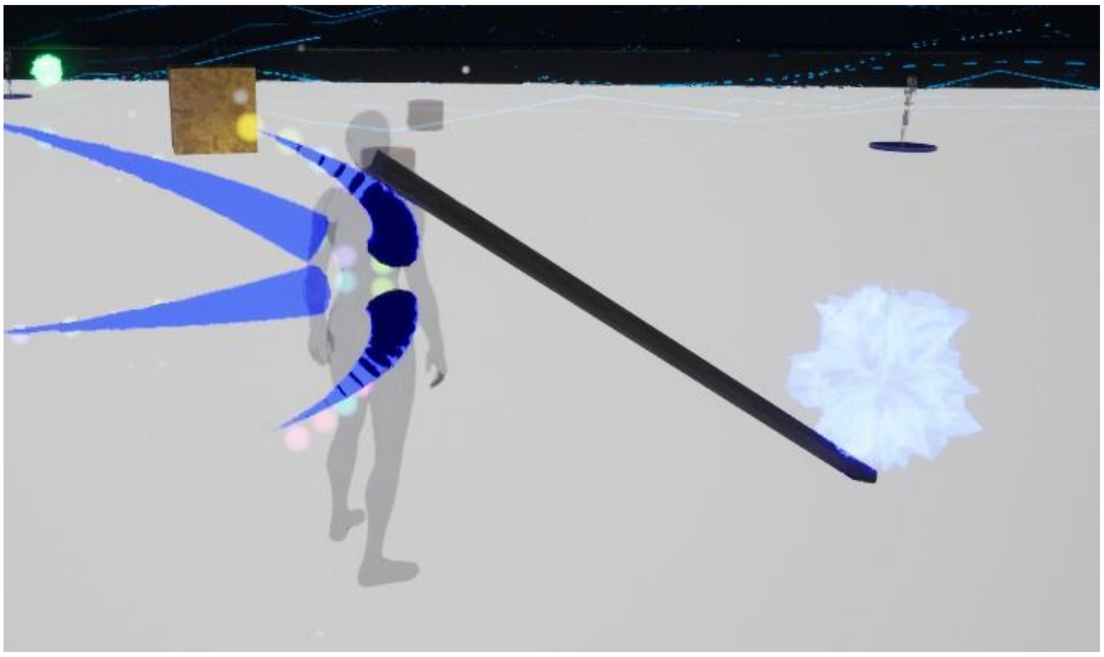
This ball of blue energy is what inspired the shader as well. I took the inspiration of the energy aspect of it making it look like a blue ball of energy. The Glitcher has an energy ball feel into it that was heavily inspired by this image.

Source : https://www.pinterest.com.au/pin/ball-blue-energy-rays-in-the-ball-of-blue-energy-aff-blue-ball-energy-ball-rays-ad--862439397373703391/

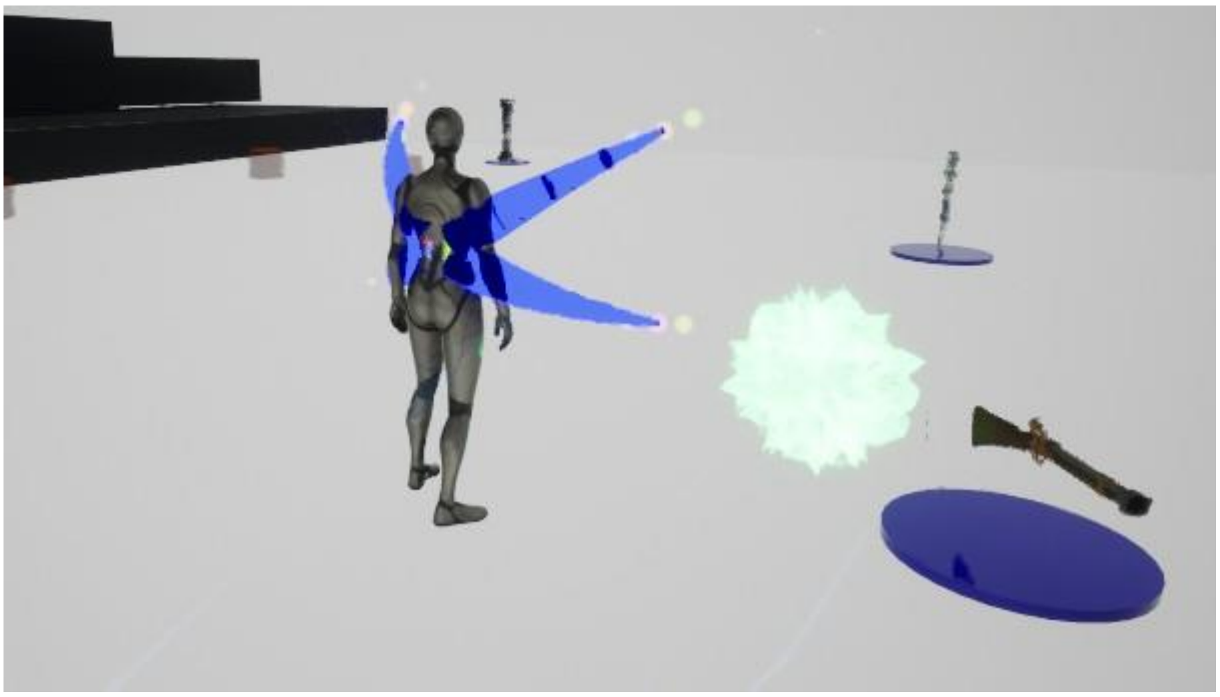
In-Engine Screenshots:



This is the default view of the Glitcher mesh. It's hard to be shown as a screenshot, but the Glitcher is moving constantly. It can also be seen that the default colour of it is green.



In this picture, it can be seen that The Glitcher is attached to the cable component. This is because the player has attached it and the player is now able to do the Chaos Mace mechanic. It is hard to tell through the screenshot but the movement of the Glitcher is way faster than before. As it can be seen, the colour changes from green to blue.

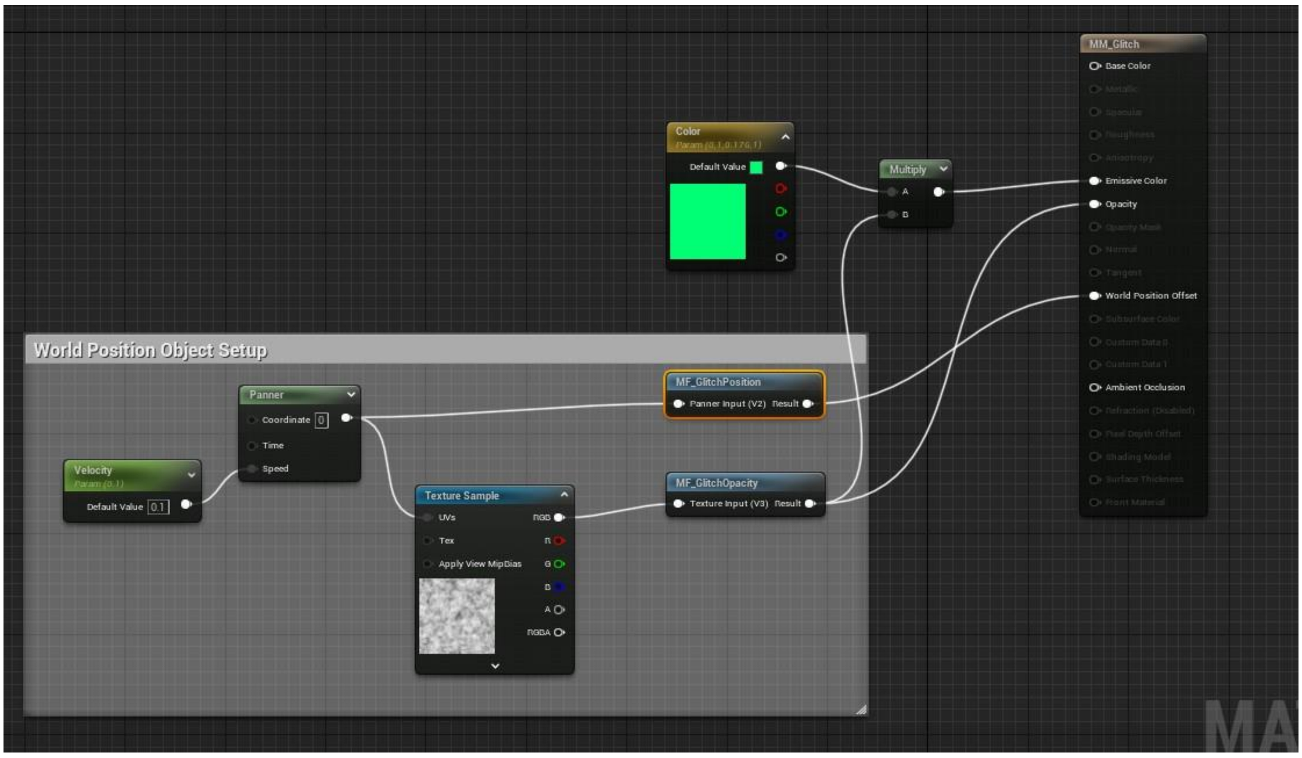


In this picture, it can be seen that the Glitcher is back to the default colour. This is because the duration of the Chaos Mace mechanic has ended.

Properties and Values

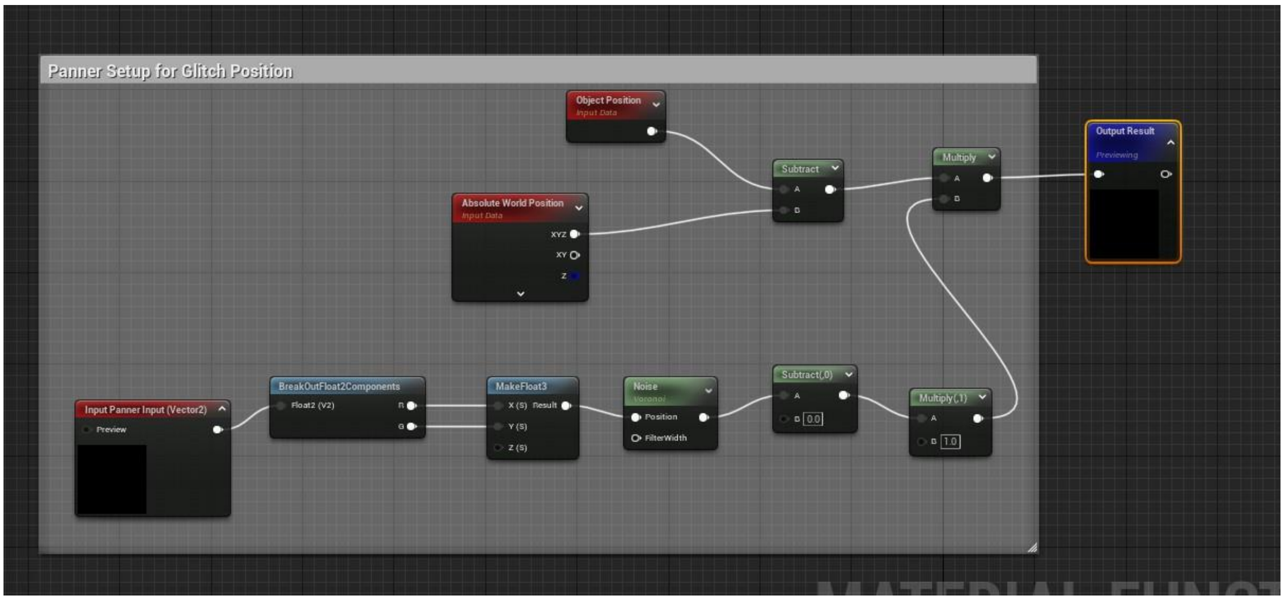
Property	Description of Purpose	Value
Color	Used to control the color of the shader dynamically	Float4 (0, 1, 0.176, 1)
Velocity	Used to control the velocity of the shader's movement dynamically	Scalar (0.1)

Node Graph



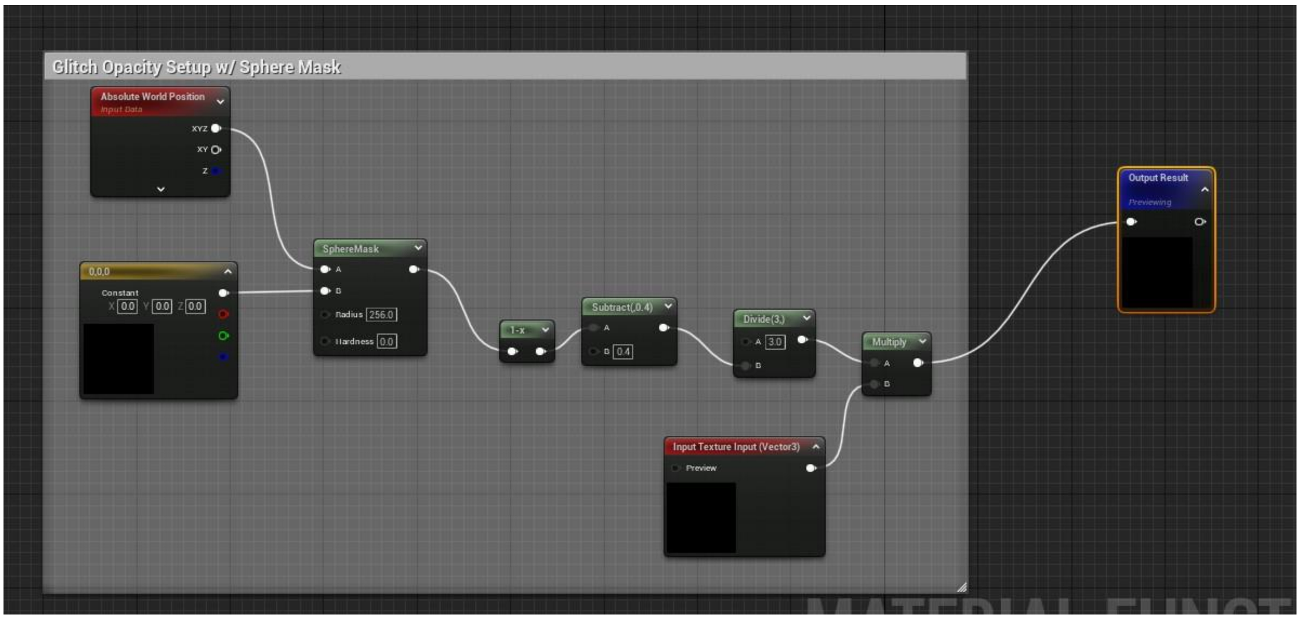
The way the node works on the surface is quite simple but there are custom material functions that I made behind the scenes which are the MF_GlitchPosition and MF_GlitchOpacity. Basically, it sets up the World Position of the object with the Velocity being dynamic with the default value of 0.1. It is also combined with a Panner and the Texture Sample T_Perlin_Noise to provide the noise of the object. The colour is also fairly simple with a green colour that is multiplied with the opacity that goes into the Emissive Color. For the World position, it is connected to the World Position Offset and for the MF_GlitchOpacity, it is connected to the Opacity node.

MF_GlitchPosition



This material functions sets up the positioning of the object shader. It starts from the Panner Component that is the Vector2 input component that broke out. Then it is turned into a Float3 and put into the Noise node to provide the variance of the material. Then it is subtracted with 0 for now but it can be changed later on when I want to make a variety of the material. Then all that is multiplied with the position of the object that was subtracted by the Absolute World Position of the object.

MF_GlitchOpacity



For the glitch opacity, it is quite simple. Basically it uses a SphereMask that is inverted. Then, it is subtracted by 0.4 and 3.0 divided by it. Then, it is multiplied with the input texture that was T_Perlin_Noise to provide the nice opacity that the shader has.

Post Processing Effects

Local Post Processing Effect - Underwater

Overview of Effect

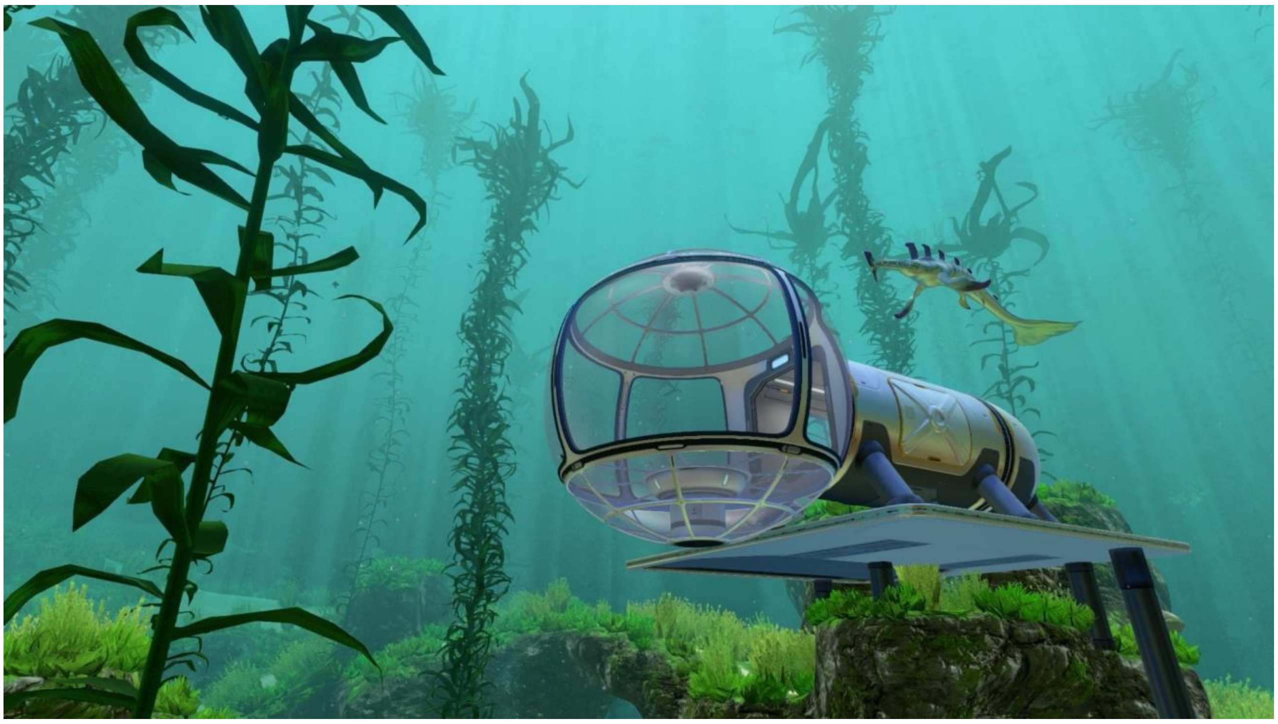
The post processing effect that is present in the scene making it a local post processing effect that I made is Underwater. This effect simulates the player being under the water. Since the water shader is using a plane as its mesh, making a post processing under it is very crucial. By adding the extra local post processing under the water shader, it can simulate the player being under the water. When the player is under the water, the player's screen will turn to a shade of blue with small water bubbles going upwards. This is commonly used in video games that have water in them.

Although there aren't C++ interactions present, the effect is still highly effective in the game setting. Without the post processing, the water shader effect can be ineffective since it being a single plane can look quite obvious. This can make the game experience more immersive and aesthetically pleasing. This can also make the whole game more coherent in terms of art direction. This can also make the player have more enjoyment when fighting underwater since the overall scene looks vastly different than the regular camera on the surface. Even if the change is only on the camera, it still is highly effective.

Effect Description

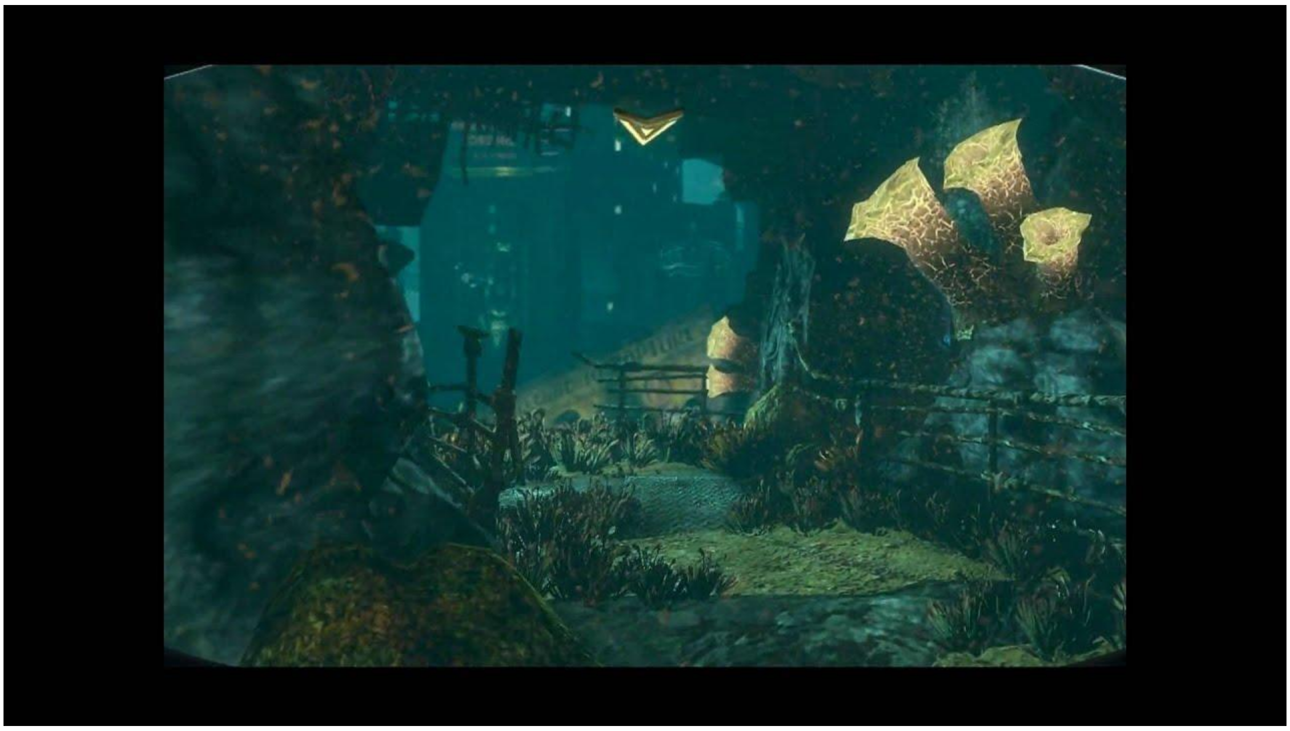
What the effect looks like is very simple. It has a very simple blue shade of colour that overlays the camera. There are also simple water movements that go upwards to simulate breathing underwater. This water movement is quite subtle and fast because I don't want it to obstruct the player's view to the point it obstructs the gameplay of the player. I want it to be visible but not too visible.

Inspiration / Reference Images:



Source : https://www.facebook.com/subnautica/photos/a.1158919240790871/1158920067457455/?type=3

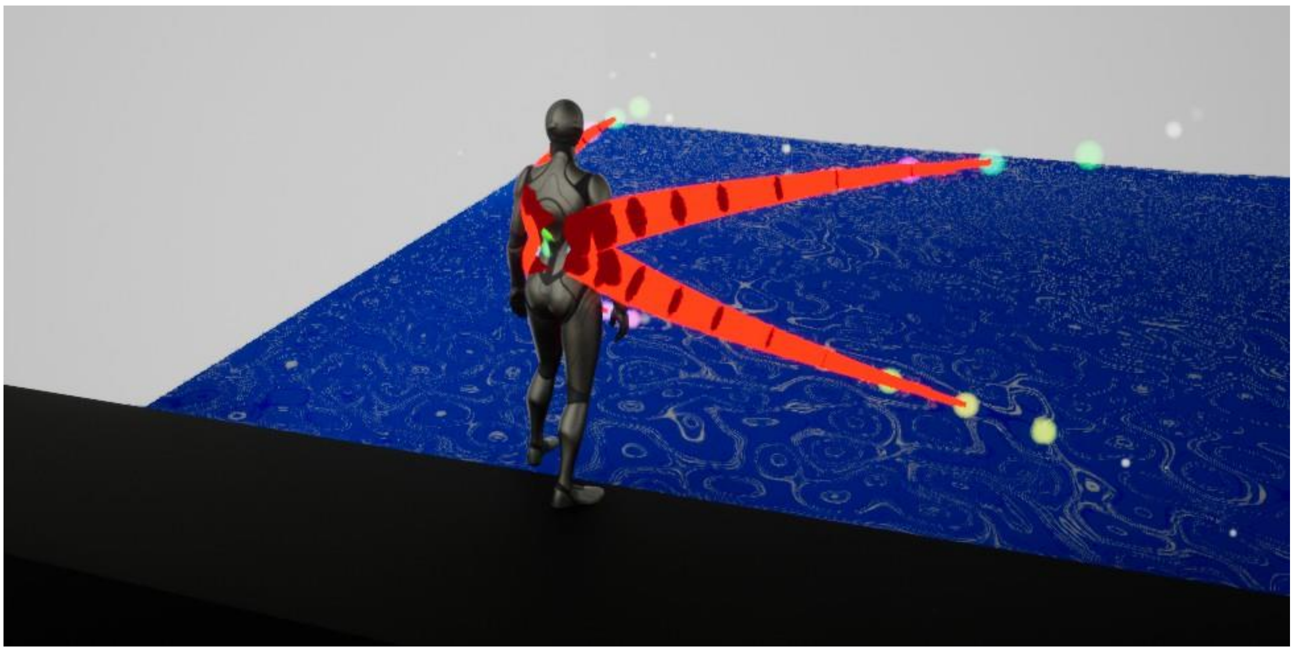
Subnautica is one of the games that I could think of when it comes to underwater post processing. I am not 100% sure on how they do it, I don't even know if it is actually a post processing but all I know is that the colour of it is where I want my underwater post processing to go.



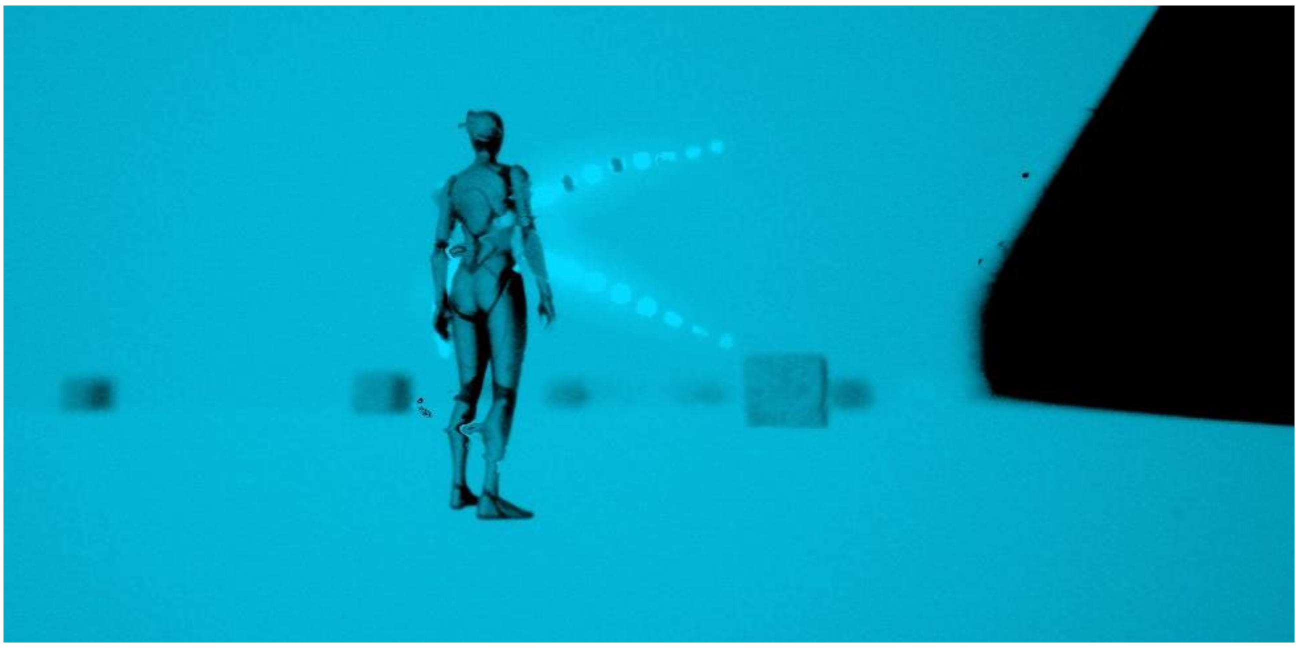
Source : https://www.youtube.com/watch?v=BxUL3-CzZos

Another game that I got the inspiration from is Bioshock 2. This game has Underwater Sections That inspired my post processing effect. The blue mixed with green colour of the post processing is perfect for reference of my post processing effect.

In-Engine Screenshots:



From the picture, it can be seen that the player is outside the water therefore, there are no underwater post processing benign applied.



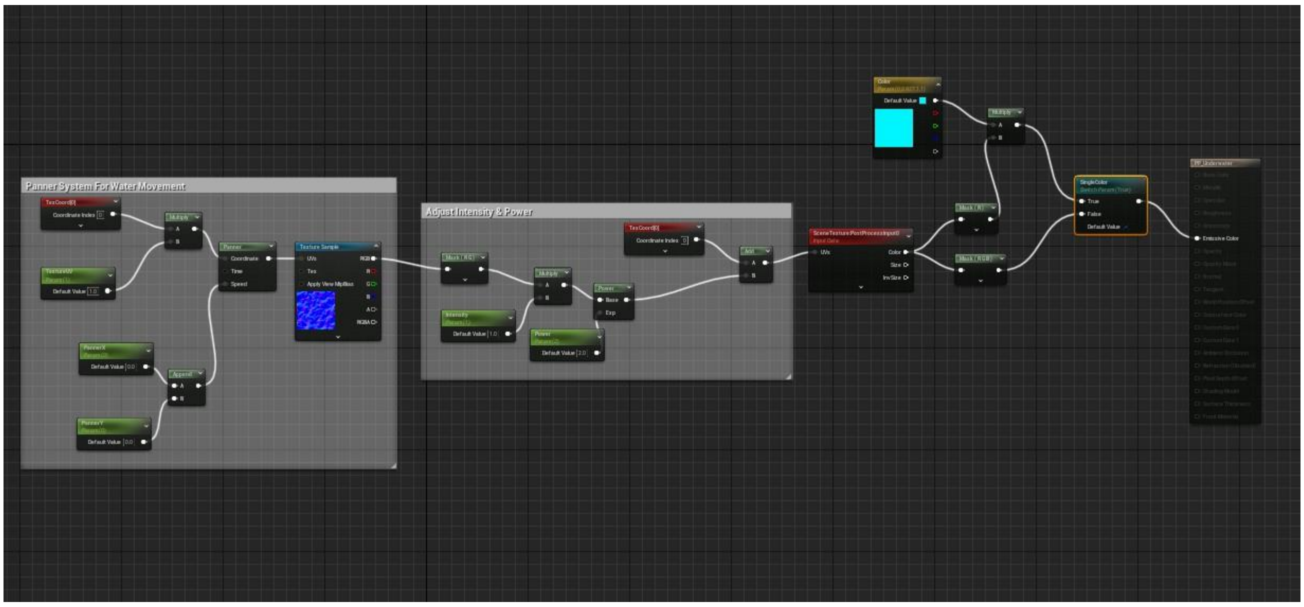
When the player enters the water, it can be seen that the post processing is in effect. The whole scene becomes this blue colour shade and some areas are a bit distorted.

This is due to the fact that it is from the movement of the post processing effect that describes the unpredictable and wavy nature of the water.

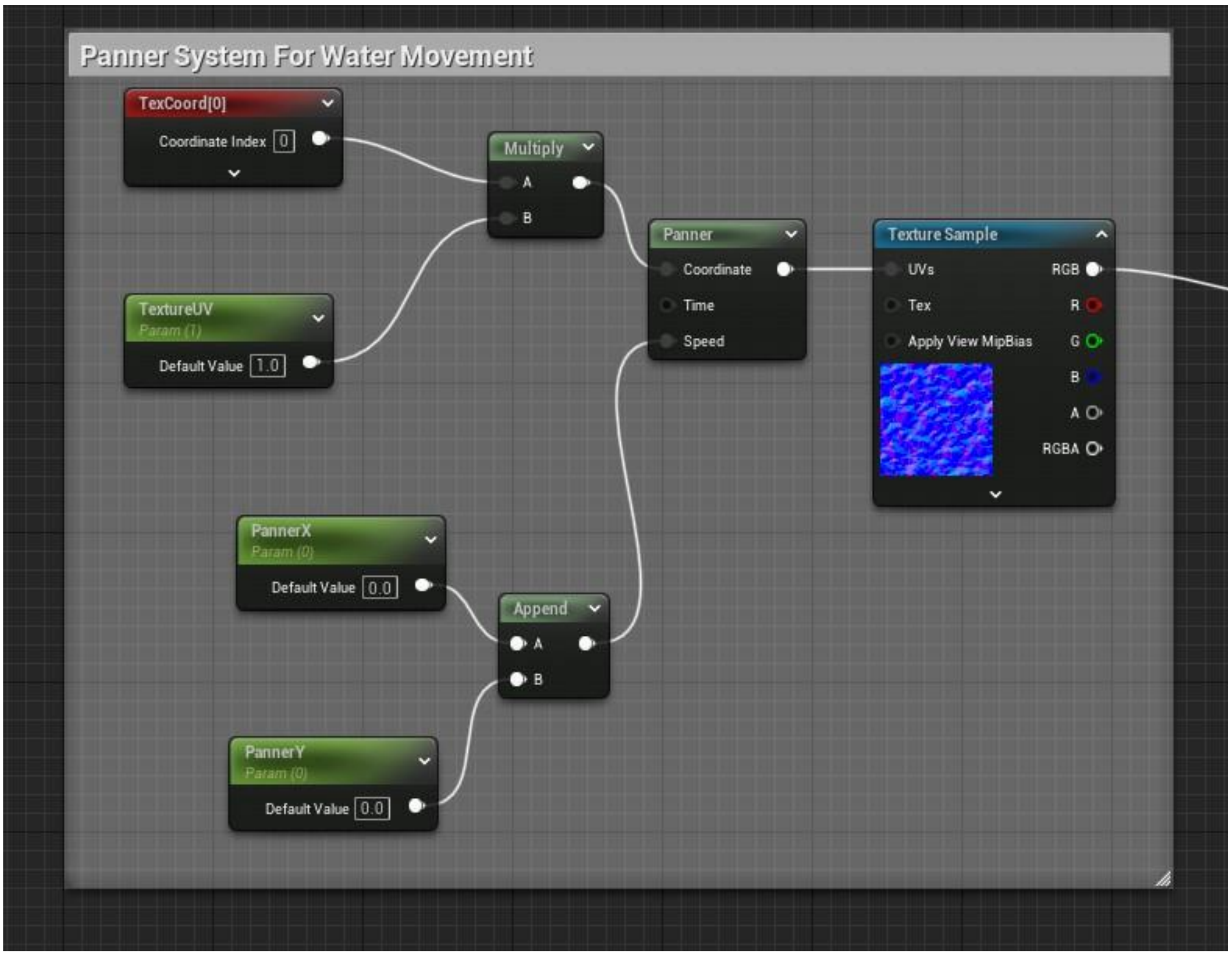
Properties and Values

Property	Description of Purpose	Value
Color	Used to control colour of post processing effect dynamically	Float4 (0, 0.927, 1, 1)
TextureUV	Used to control the UV of the post processing effect dynamically	Scalar (1.0)
PannerX	Used to control the X part of the panner of the post processing effect dynamically	Scalar (0.0)
PannerY	Used to control the Y part of the paneer of the post processing effect dynamically	Scalar (0.0)
Intensity	Used to control the intensity of the paneer of the post processing effect dynamically	Scalar (1.0)
Power	Used to control the power of the post processing effect dynamically	Scalar (2.0)
SingleColor	Used to blend and control the colours of the scene and the post processing effect	Static Switch Parameter (True)

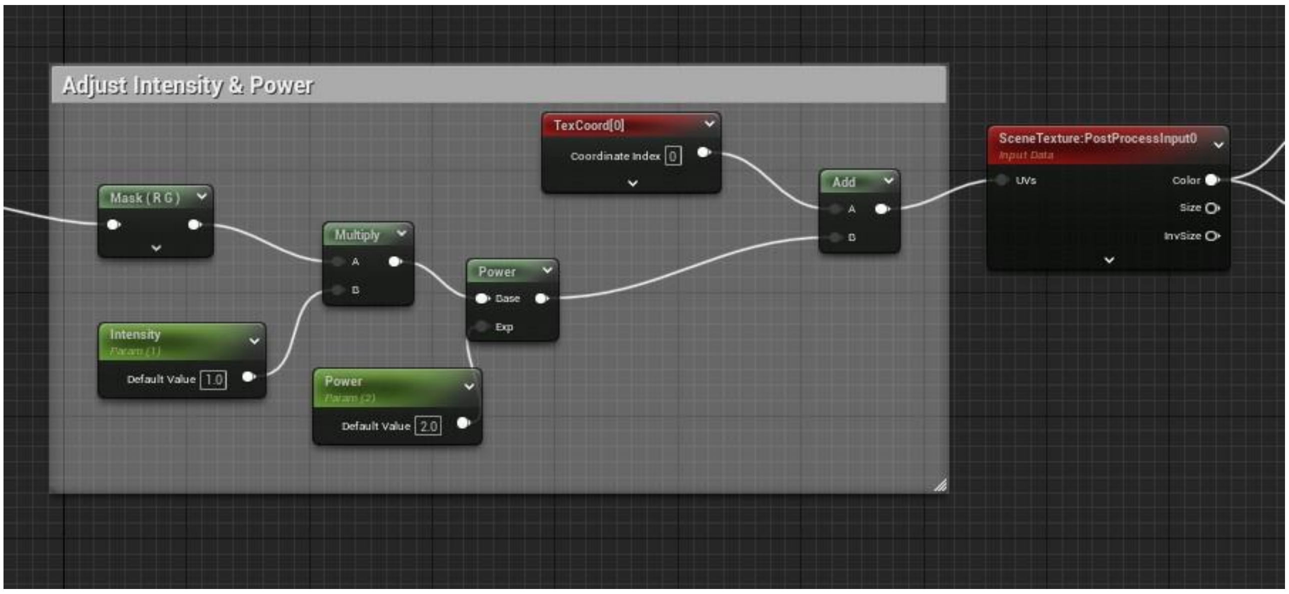
Node Graph



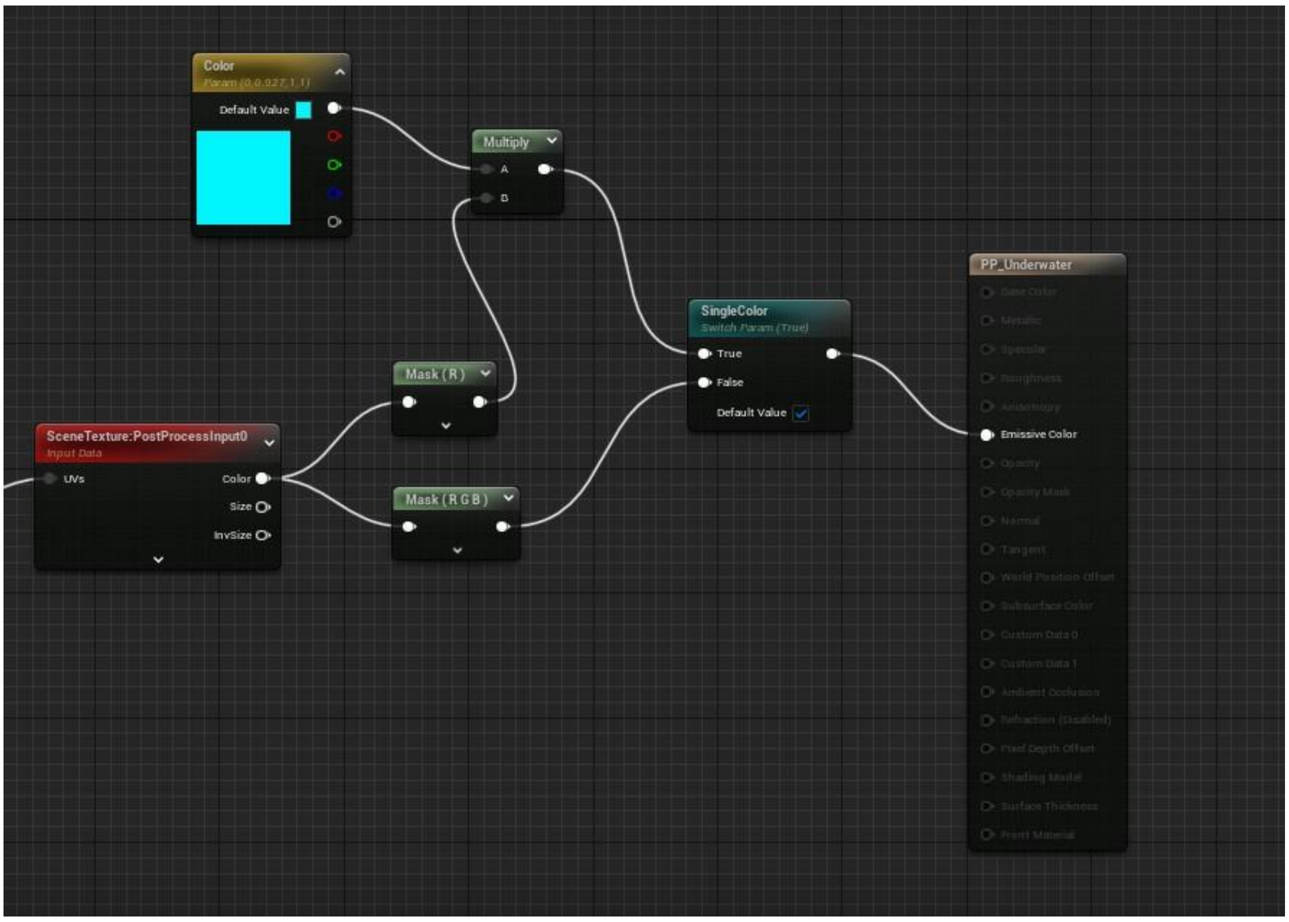
This is the whole post processing graph, every section will be explained in detail.



This section is for the paneer system for the water movement. The paneer system uses a value from the scalar of TextureUV and the texture coordinate that is multiplied together. It is then added to the coordinate of the paneer with the speed that is adjustable. Then, it is added to the Texture Sample which uses T_Water_N normal texture.



This section is for adjusting the intensity and the power of the post processing effect. It can be seen that the Multiply and the Power Node are highly adjustable. That is because those nodes are the one responsible for the intensity and power of the processing effect. It is then added to the Post Processing Input.



The last part of the section is blending in the colour of the whole post processing and the water movement effect with the static switch parameter node. This is then connected to the Emissive Colour part of the main node because that's the only available node when setting the main node into post processing.

Global Post Processing Effect - Outline

Overview of Effect

The global post processing is called Outline. The way this post processing works is that it activates from the get go but the player can control whether to turn it on or off. The post processing gives a green outline to each object's vertices making things easier to see and making everything looks very stylized.

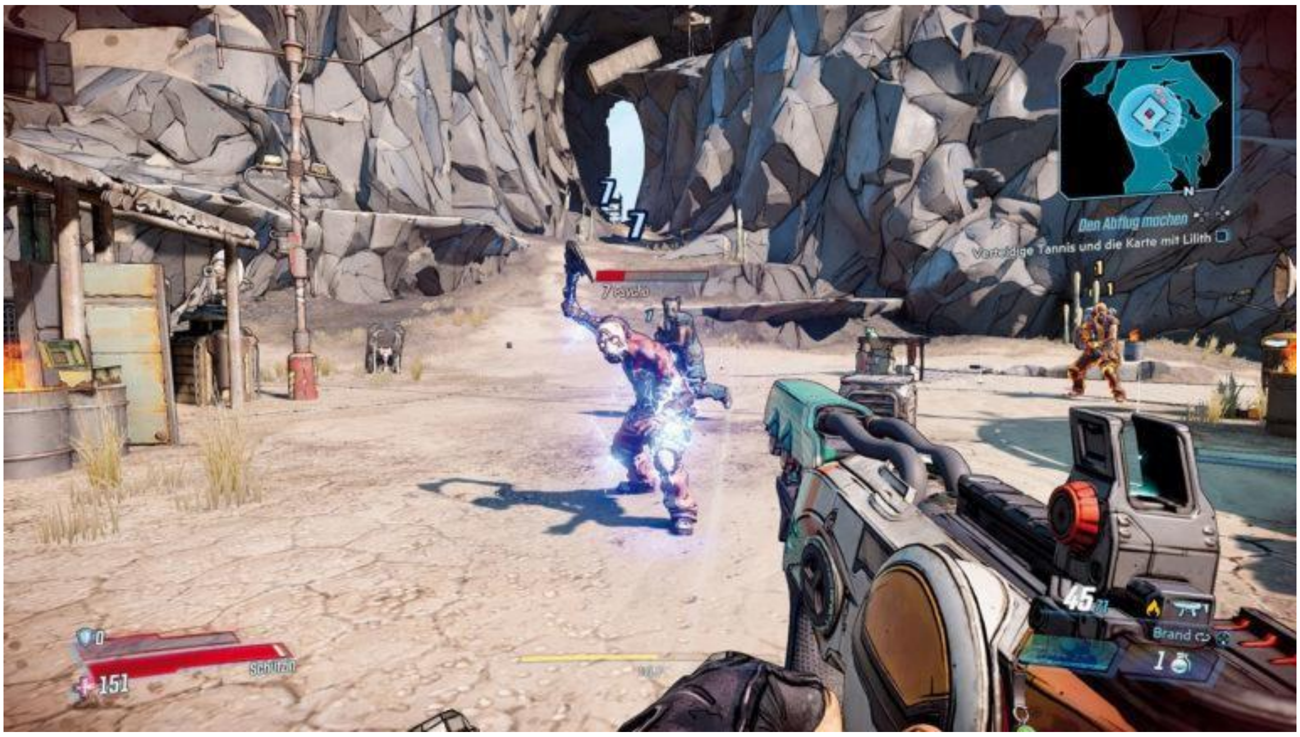
Since the game itself is inspired by the matrix, I decided to go with the colour green for the outline of the objects. Other than the outline, all of the other colours are black. This gives the player the sense of immersion and feeling that they are transported into a different world. The way the post processing activates and deactivates it is also unique. It basically deactivates by spreading outwards from the player in the form of a circle. The player can do this by pressing the E key. The player can also activate it back and the way it activates is it does the exact same thing but in reverse. It comes to the player in the form of a circle which can be done by pressing the E key again.

This creates a sense of control of when the player wants to see the different world. This can also be used to identify objects that are hard to see without the post processing. For example, if an object has a very harsh lighting and is really hard to see, it would be best if the player can activate the post processing and see the object's outline.

Effect Description

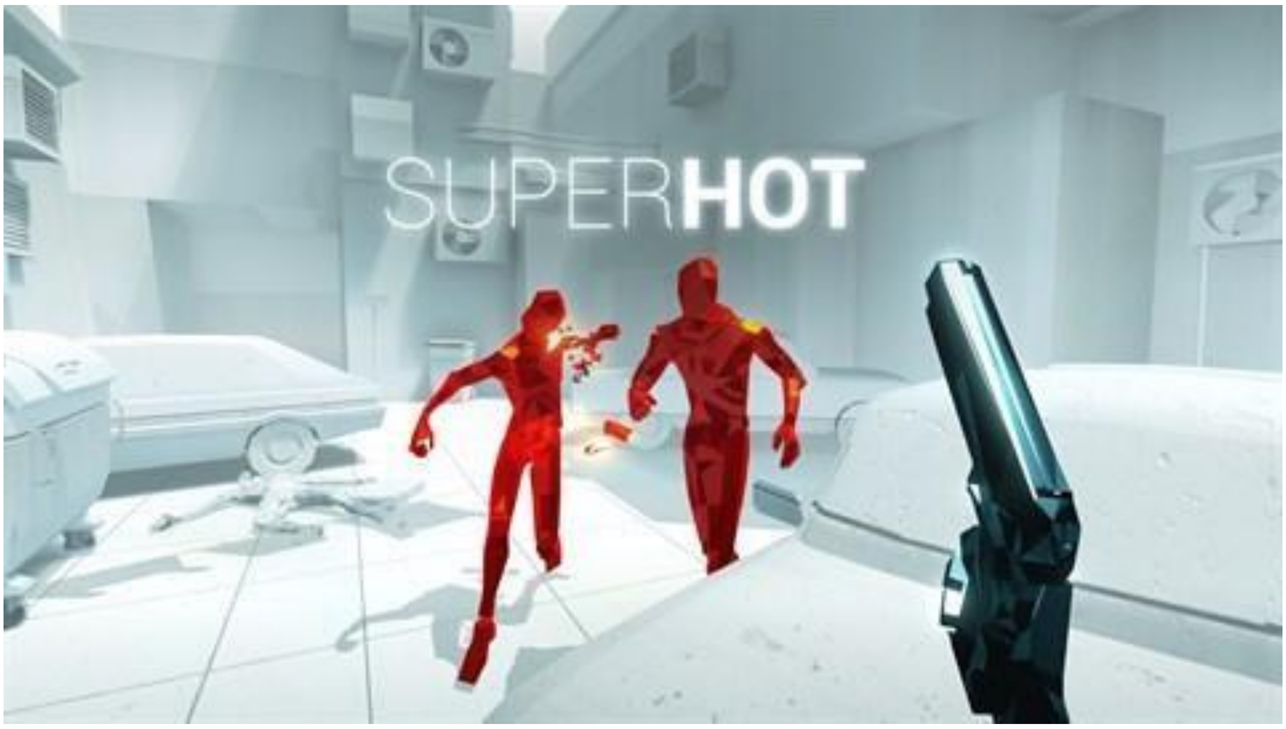
The effect looks very different from a typical filter on a camera. The effect makes everything black except for the outlines of the object. The outline of the objects have the colour green on it. The effect can also be controlled through the E key by deactivating and activating it. The way it deactivates is it goes outwards from the player in the form of a circle. The way it activates is the same way but in reverse, it comes to the player in the form of a circle.

Inspiration / Reference Images:



Source : https://www.maniac.de/tests/borderlands-3-im-test-ps4-xbox-one/

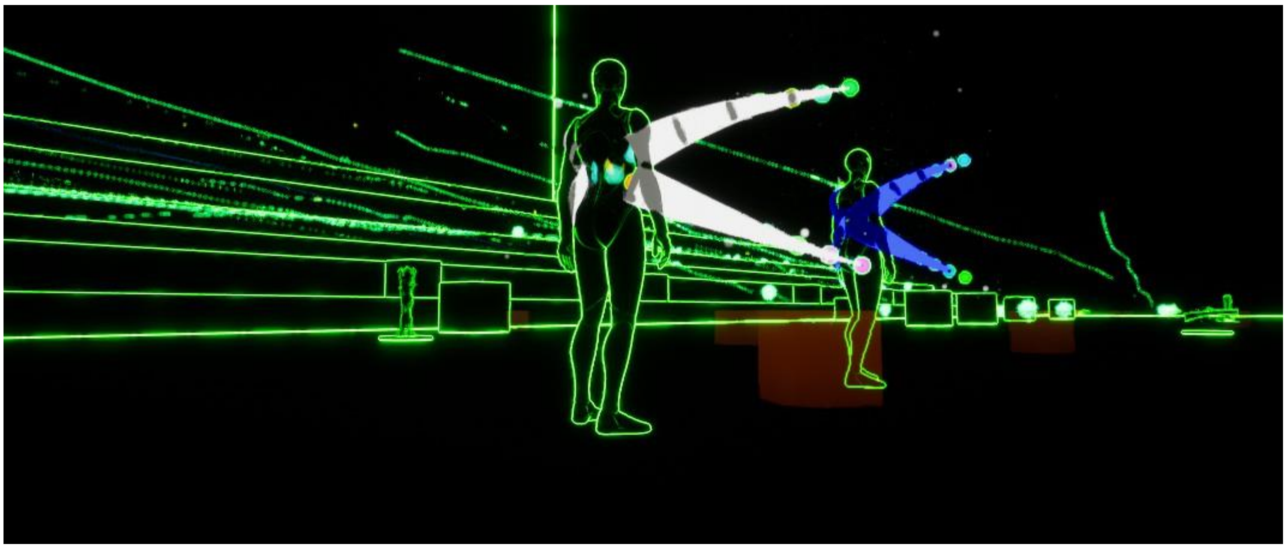
One of the games that have an outline post processing is Borderlands. In this image which comes from Borderlands 3, it has outlines in everything. This style is called Cel Shading which is not what I did for my post processing effect. But I was inspired by the outline aspect of the post processing effect.



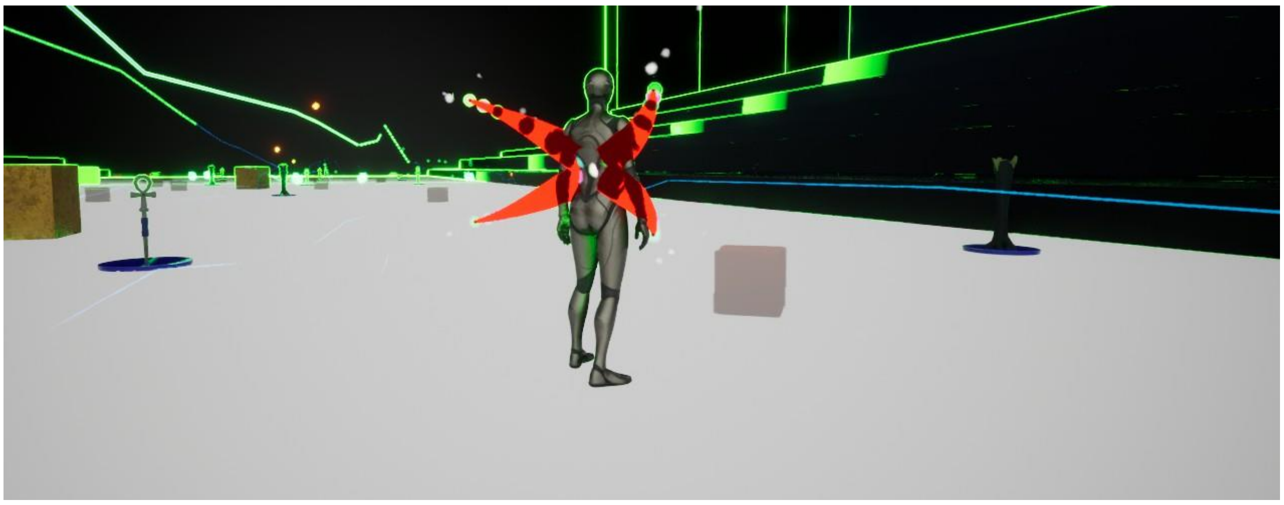
Source : https://www.xbox.com/en-AU/games/store/superhot-windows-10/9NV17MJB26PG

The game that I was also inspired by is Superhot. Even though superhot doesn’t have a post processing that I was inspired by, I was inspired more from the style of the game. It's very simple but it is still charming. It inspired the simplicity aspect of my post processing effect by making everything simple with the outline and nothing else.

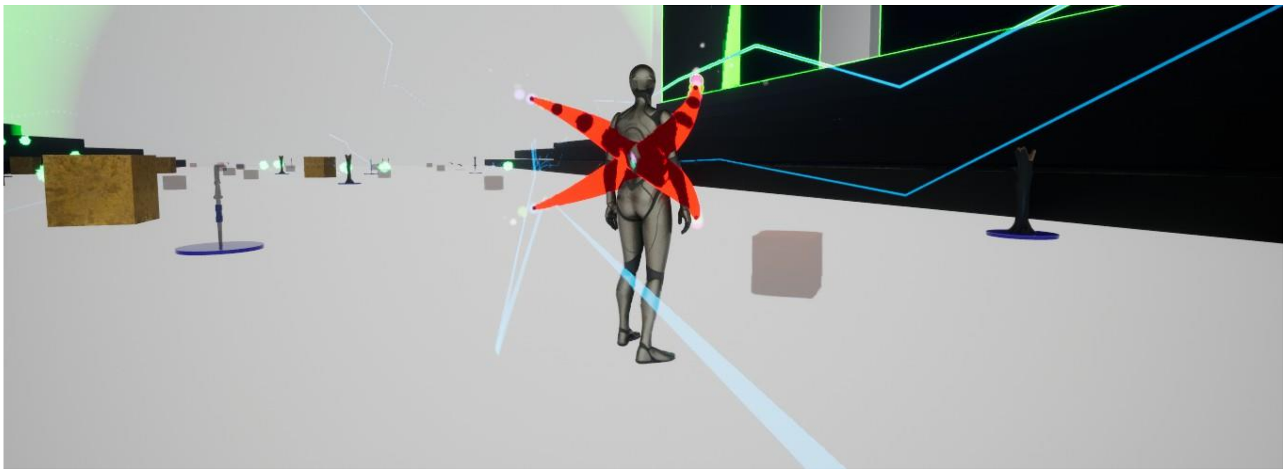
In-Engine Screenshots:



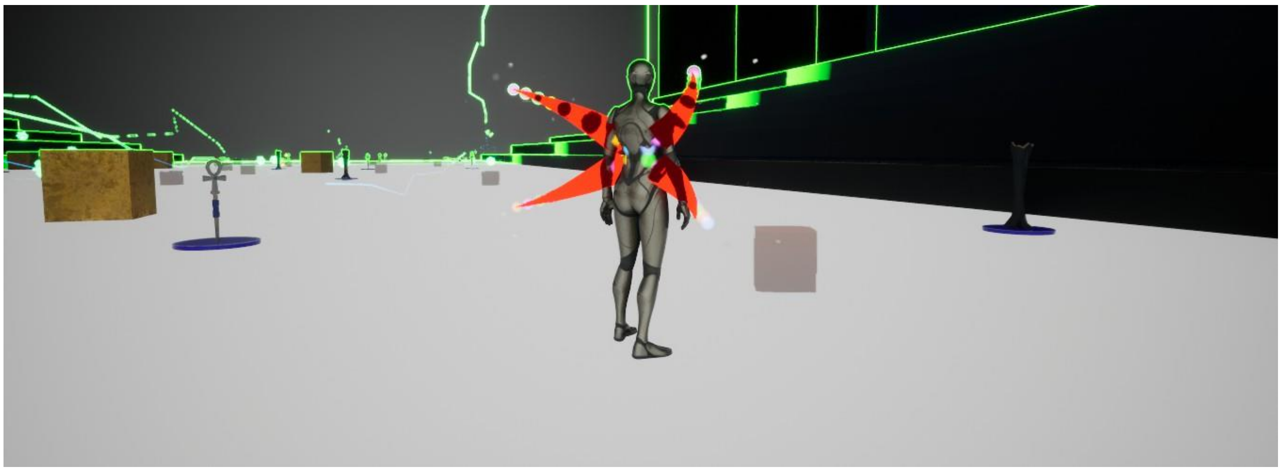
By default the post processing is on. As it can be seen, everything is black except for some objects and particles. All of the objects also have a green outline in them.



In this picture, it can be seen that the post processing effect is going away from the player. This is the deactivation process of the post processing.



In this picture, the post processing is going away from the player. It's more wider than the previous picture showing the dynamic movement of the deactivation of the post processing.



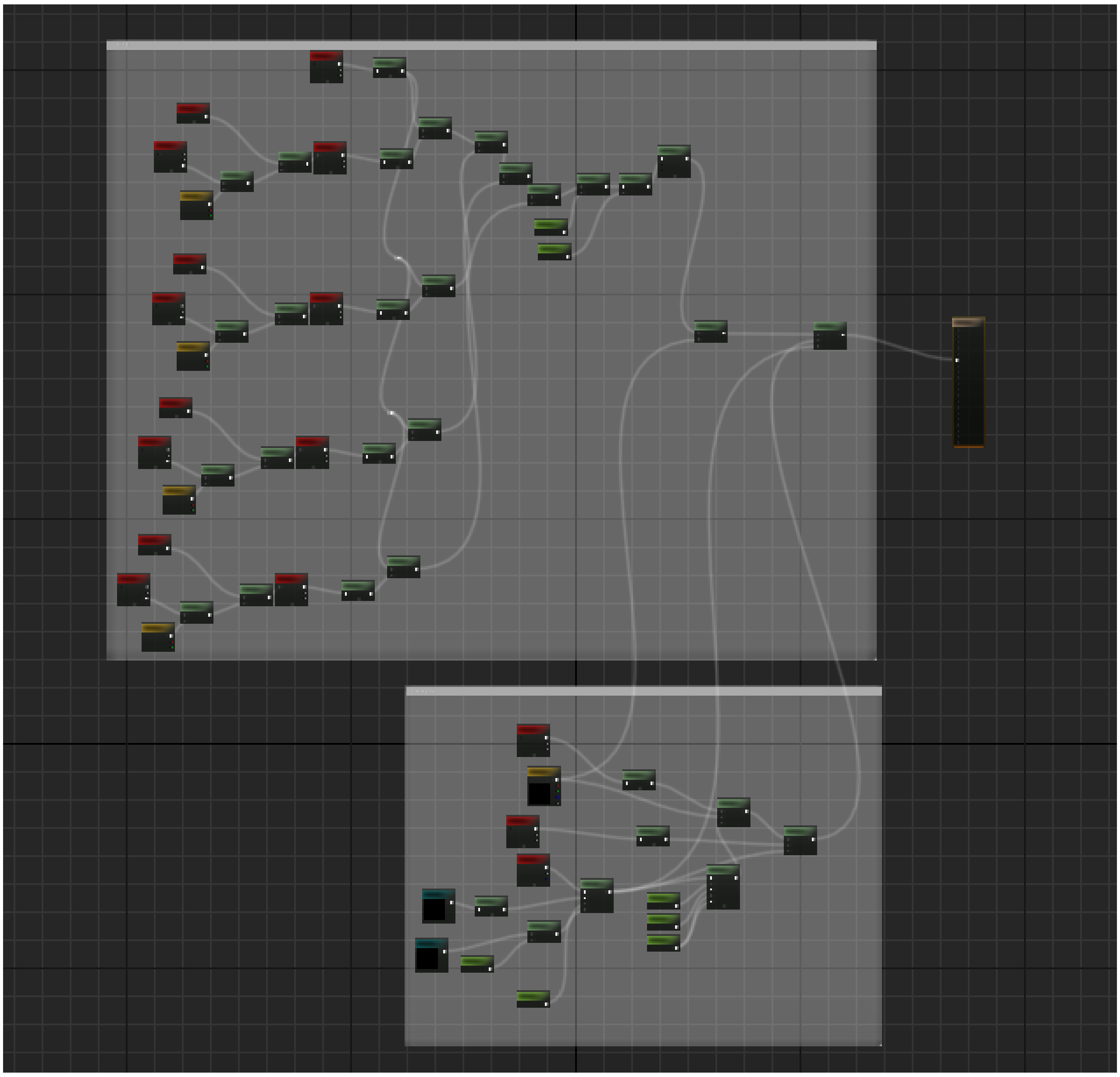
In this picture, the post processing is coming back to the player. It's in the process of coming back.

Properties and Values

Property	Description of Purpose	Value
Line Depth	Used to control the depth of the outline of the post processing effect dynamically	Scalar (0.0)
Power	Used to control the power of the outline post processing effect dynamically	Scalar (0.0)
Outline Color	Used to control the Outline Color of the post processing effect dynamically	Float4 (0, 0, 0, 0)
MPC_Outline	Used to control the Position and the Radius of the Post processing effect	Position (Vector), Radius (Scalar)

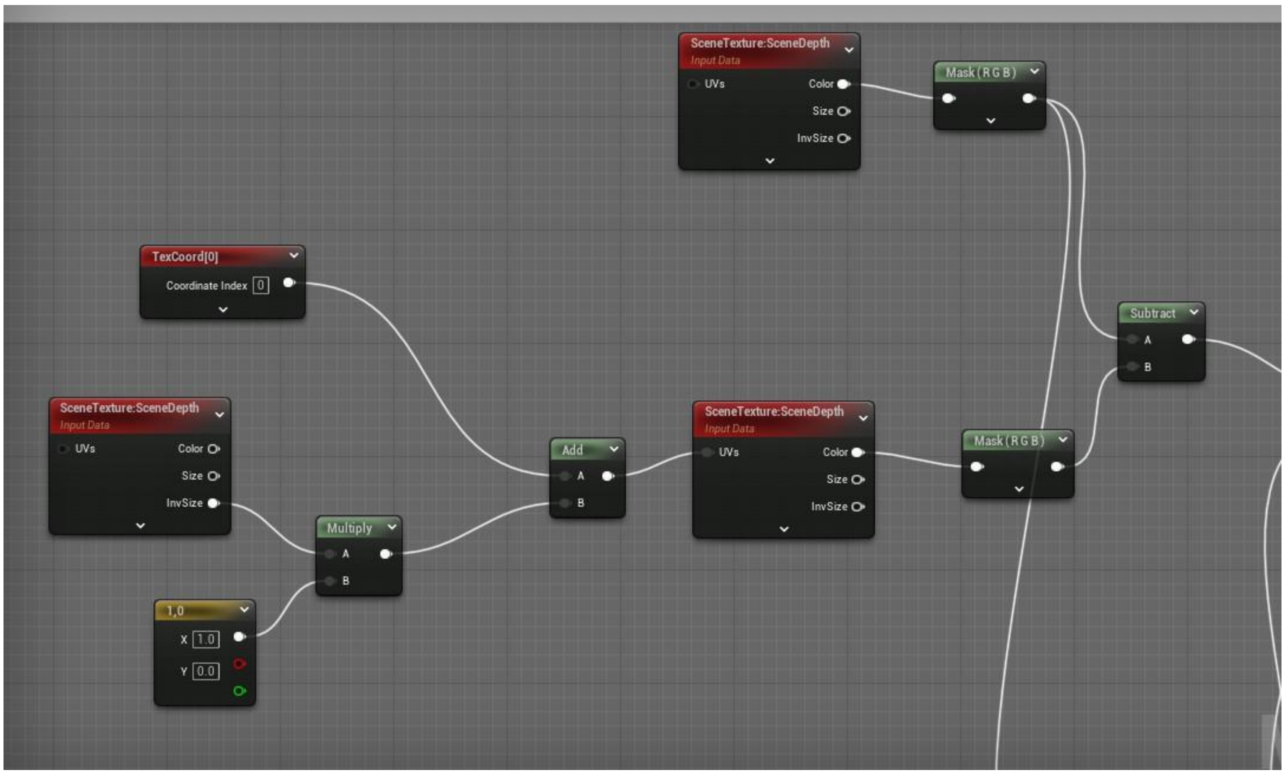
	dynamically through Material Parameter Collection	
Radius	Used to control the radius of the post processing effect’s sphere mask dynamically	Scalar (0.0)
SceneLerpHandler	A Timer that handles the timer transitioning of the post processing	FTimerHandler
SceneLerp	The value that changes the radius of the scene dynamically	Min 0, Max 6000
bIsDoneLerping	Checks if the scene is done changing or not	false

Node Graph

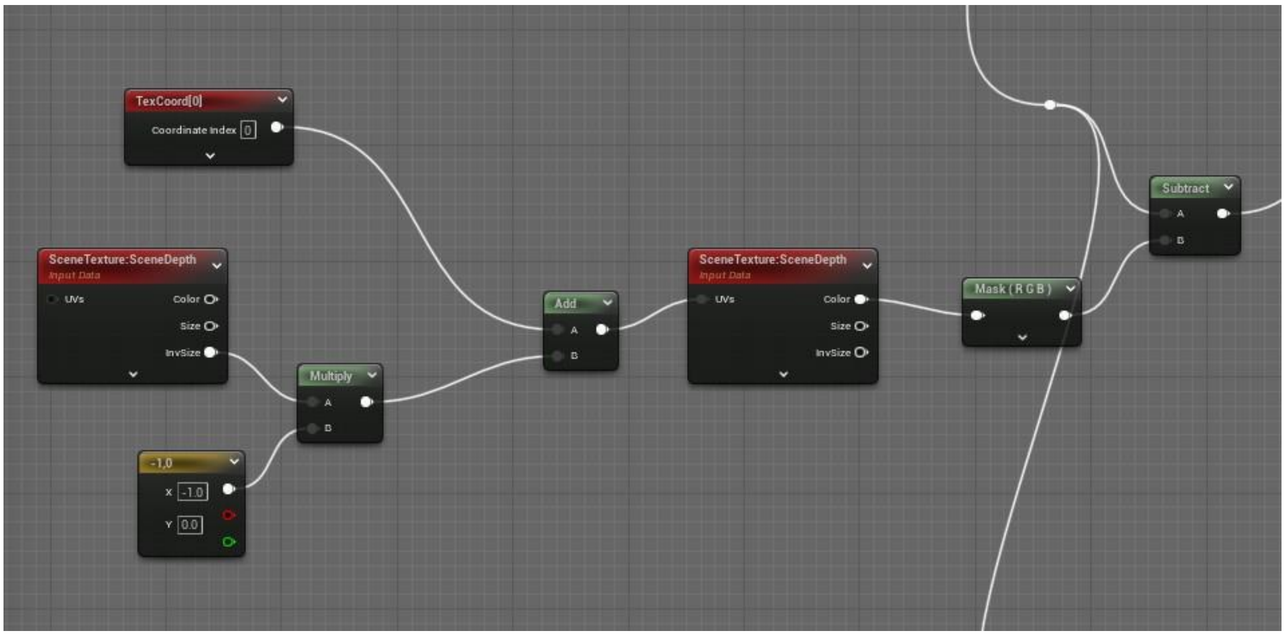


This is the overall preview of the whole node. Even though it looks like there’s a lot going on, it's simpler than it seems. Each section will be explained thoroughly.

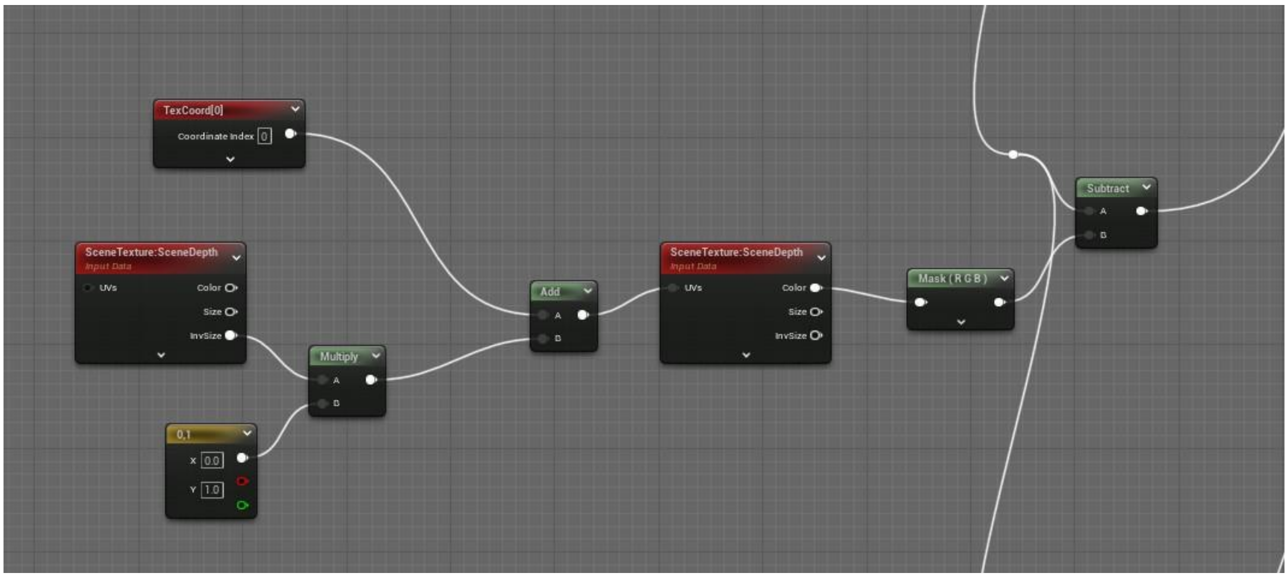
To make sure only the outline is present, I need to subtract everything and leave the outline.



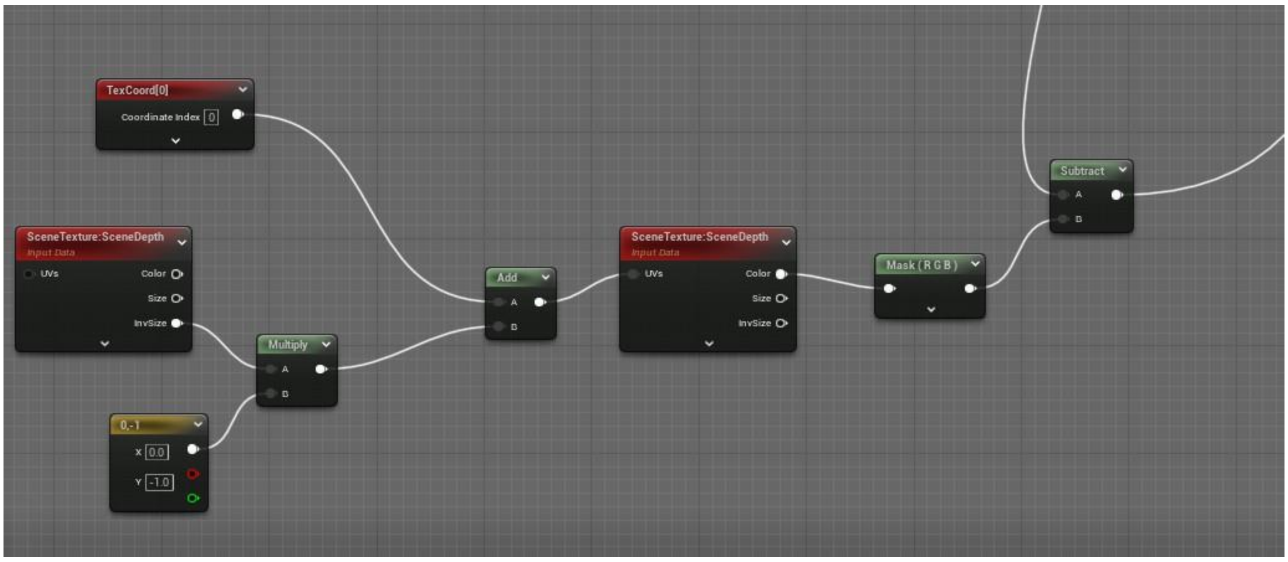
From this node, it can be seen that I multiplied the scene depth with 1, 0. Then I added it with the texture coordinate and the UV of the scene depth again. Then I mask it with the RGB. From this, I subtracted it from the full scene.



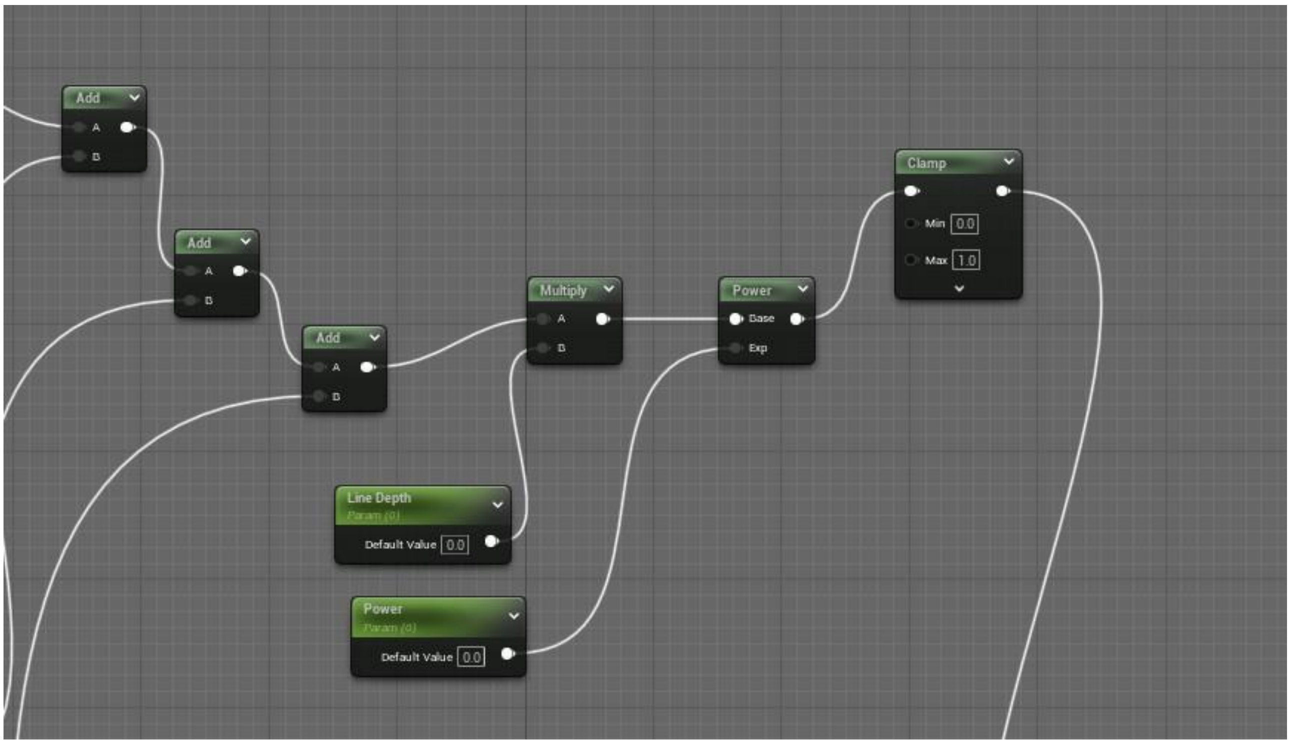
This screenshot has the same function, the only difference is that it's subtracted by -1, 0.



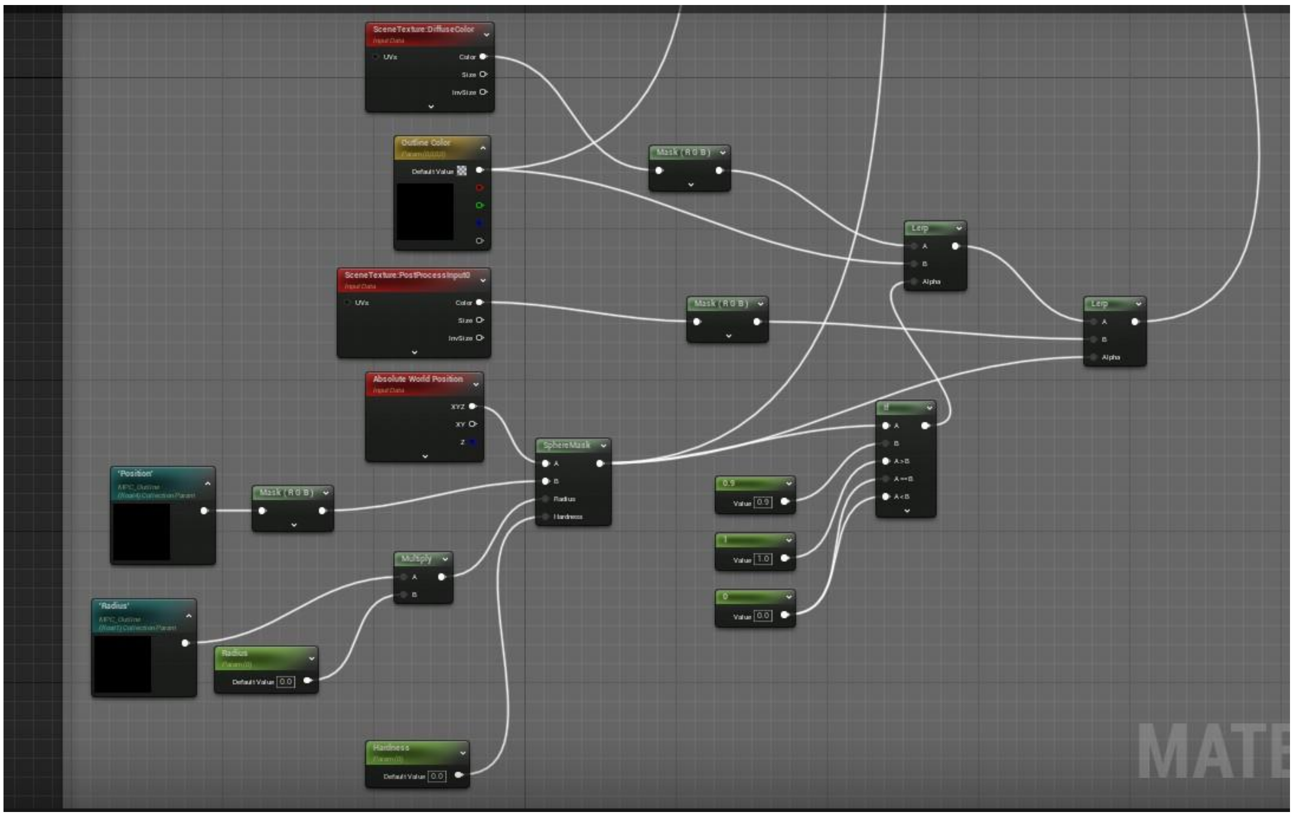
This screenshot has the same function, the only difference is that it's subtracted by 0, 1.



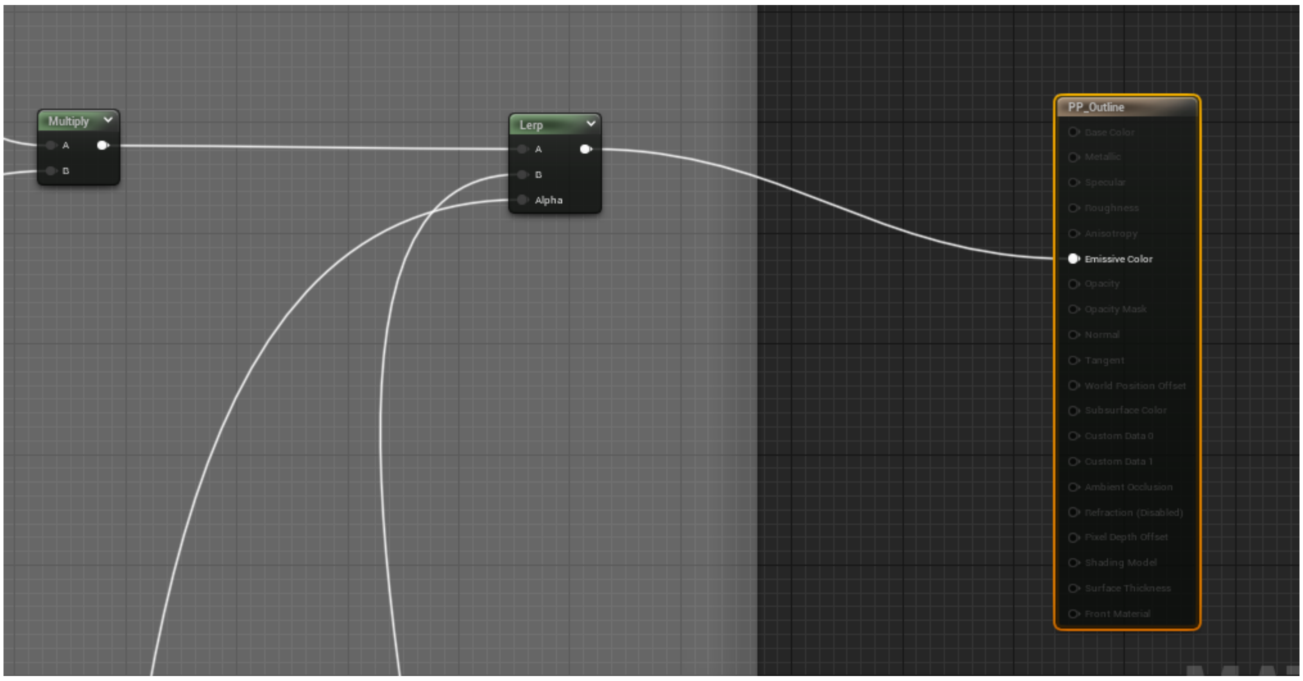
This screenshot has the same function, the only difference is that it's subtracted by 0, -1



All of the things that were subtracted earlier are all added and multiplied with the line depth and with the power and then clamped.



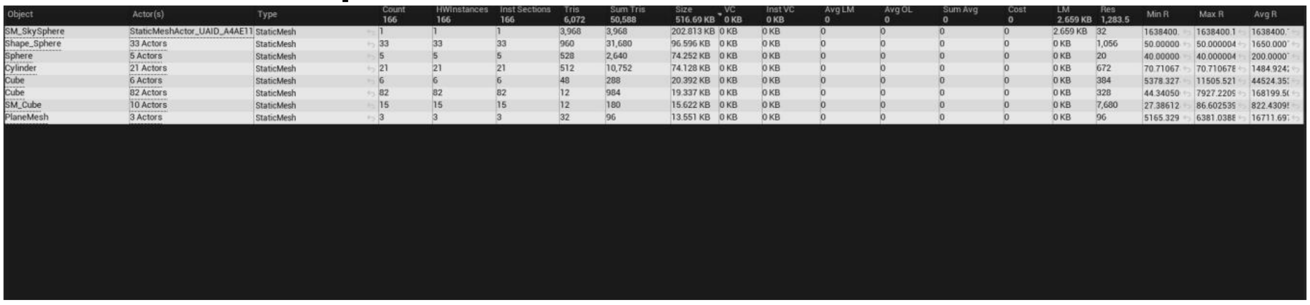
This part is for the masking part of the post processing effect. It gets the radius that is adjustable and also the position of the player through the Material parameter collection. It then uses a SphereMask that is also combined with the Absolute World Position. Then it goes through an if statement to determine whether it's activated or not. It also goes through a lerp with the adjustable outline colour and diffuse colour function. All of that is then lerp with the mask being in the alpha and then multiplied further.



The multiply is multiplied with the previous functionality of subtraction combined with Outline Colour. The lerped part of the B is connected to the result of the masking and the alpha is connected from the SphereMask.

Optimisation

Statistics Auditor Report



Object	Actor(s)	Type	Count	HWInstances	Inst Sections	Tris	Sum Tris	Size	VC	Inst VC	Avg LM	Avg OL	Sum Avg	Cost	LM	Res	Min R	Max R	Avg R
			166	166	166	6,072	50,588	516.69 KB	0 KB	0 KB	0	0	0	0	2.659 KB	1,283.5			
SM_SkySphere	StaticMeshActor_UAID_A4AE11	StaticMesh	1	1	1	3,968	3,968	202.813 KB	0 KB	0 KB	0	0	0	0	2.659 KB	32	1638400.	1638400.1	1638400.
Shape_Sphere	33 Actors	StaticMesh	33	33	33	960	31,680	96.596 KB	0 KB	0 KB	0	0	0	0	0 KB	1,056	50.00000	50.000004	1650.000
Sphere	5 Actors	StaticMesh	5	5	5	528	2,640	74.252 KB	0 KB	0 KB	0	0	0	0	0 KB	20	40.00000	40.000004	200.0000
Cylinder	21 Actors	StaticMesh	21	21	21	512	10,752	74.128 KB	0 KB	0 KB	0	0	0	0	0 KB	672	70.71067	70.710678	1484.924
Cube	6 Actors	StaticMesh	6	6	6	48	288	20.392 KB	0 KB	0 KB	0	0	0	0	0 KB	384	5378.327	11505.521	44524.35
Cube	82 Actors	StaticMesh	82	82	82	12	984	19.337 KB	0 KB	0 KB	0	0	0	0	0 KB	328	44.34050	7927.2209	168199.5
SM_Cube	10 Actors	StaticMesh	15	15	15	12	180	15.622 KB	0 KB	0 KB	0	0	0	0	0 KB	7,680	27.38612	86.602539	822.4309
PlaneMesh	3 Actors	StaticMesh	3	3	3	32	96	13.551 KB	0 KB	0 KB	0	0	0	0	0 KB	96	5165.329	6381.0388	16711.69

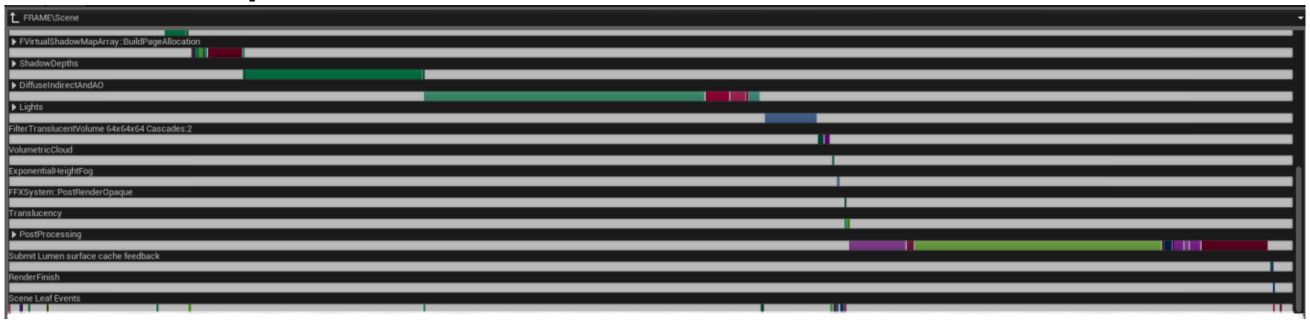
This is one of the test cases report which is the Statistics auditor Report. It checks how many tris and polygons are present within the scene. This helps with identifying which object has too many tris and needs to be optimised in terms of modelling. From the screenshot, it can be seen that there are no objects with too many tris reaching millions. The only concern for the game is the tris of the Shape_Sphere. The tris of the Shape_Sphere, though not a lot, still racks up to 31,680 which is high compared to the second highest which is the Cylinder which is 10,752. This can come from the amount of actors that are present within the scene as it can be seen with Shape_Sphere having 33 actors. Even if this is not a problem yet, this can still cause potential problems if it is replaced with another mesh.

There are a couple of solutions to solve the potential problem that may occur. One of the solutions is to lessen the number of actors that are present within the scene with the mesh that was referred to. We can use Shape_Sphere for example. By reducing the number of the actors from 33 into around 20. This however, can affect the gameplay and the use of the actors.

It can lessen the tris by a huge amount which will lessen the workload of the computer. Another potential solution that can be executed is also to lessen the tris of the model. By making the sphere a lower poly sphere, it can lower the tris immensely and can support the game's performance more. Though it may reduce the appeal of the mesh itself.

In conclusion, based on the report, currently the game is in good shape in regards to tris, however it is still worth noting that there are still some potential problems present. These problems can be potentially solved by lowering the tris of the model or deleting the actors themselves.

GPU Profiler Report


FRAME\Scene
FVirtualShadowMapArray::BuildPageAllocation
ShadowDepths
DiffuseIndirectAndAO
Lights
FilterTranslucentVolume 64x64x64 Cascades:2
VolumetricCloud
ExponentialHeightFog
FFXSystem::PostRenderOpaque
Translucency
PostProcessing
Submit Lumen surface cache feedback
RenderFinish
Scene Leaf Events

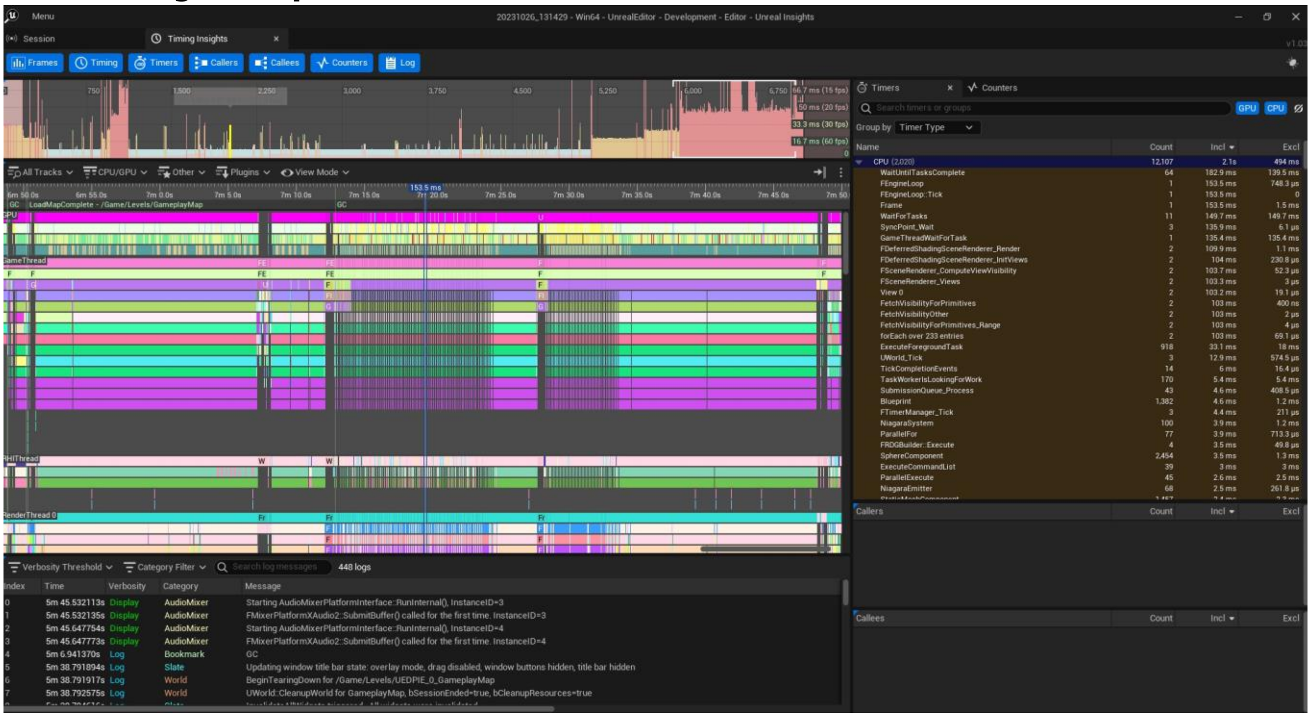
This is one of the test cases from the Unreal Engine reports. It contains useful information about the workload of the GPU based on the game's runtime. As it can be seen, there are 2 things that can be optimised more which are the shadows and the post processing. These 2 things are very GPU intensive since they use HLSL that utilises the GPU.

The potential causes of these things are because of the post processing that is implemented within the game. One of which is the dynamic post processing which can use a lot of GPU power. The shadows are also affected since the shadows of the object will render differently based on the post processing that is applied. Although it is hardly noticeable if the player uses a high level graphics card. This can cause some issues for computers with a low level graphics card.

There are a couple of potential fixes that can be done to this. The most straightforward one is to lessen the post processing effect for the whole game. The global post processing that is currently implemented has a lot of work behind the scenes with Unreal's material nodes. By lessening the effect overall, it can cause some aesthetic issues making it look less appealing but it can optimise the game better. The other potential solution is to disable the post processing when the post processing is deactivated. The way it works right now is that the post processing is not fully disabled but rather pushed back outside of the map. This means the post processing is still active in the background even if the player is not seeing it. Again, this can cause issues with how the game looks since it can reveal the skyboxes that are clashing with the game's art direction but it could heavily optimise the game in the right way.

Overall, the solution to this problem just comes by adjusting the post processing. The shadows then will solve itself once the post processing problem is solved which can lead to better GPU performance.

Unreal Insights Report



Name	Count	Incl	Excl
CPU (2,020)	12,107	2.1s	494 ms
WaitUntilTasksComplete	64	182.9 ms	139.5 ms
FEngineLoop	1	153.5 ms	748.3 µs
FEngineLoop::Tick	1	153.5 ms	0
Frame	1	153.5 ms	1.5 ms
WaitForTasks	11	149.7 ms	149.7 ms
SyncPoint_Wait	3	135.9 ms	6.1 µs
GameThreadWaitForTask	1	135.4 ms	135.4 ms
FDeferredShadingSceneRenderer_Render	2	109.9 ms	1.1 ms
FDeferredShadingSceneRenderer_InitViews	2	104 ms	230.8 µs
FSceneRenderer_ComputeViewVisibility	2	103.7 ms	52.3 µs
FSceneRenderer_Views	2	103.3 ms	3 µs
View 0	2	103.2 ms	19.1 µs
FetchVisibilityForPrimitives	2	103 ms	400 ns
FetchVisibilityOther	2	103 ms	2 µs
FetchVisibilityForPrimitives_Range	2	103 ms	4 µs
forEach over 233 entries	2	103 ms	69.1 µs
ExecuteForegroundTask	918	33.1 ms	18 ms
UWorld_Tick	3	12.9 ms	574.5 µs
TickCompletionEvents	14	6 ms	16.4 µs
TaskWorkerIsLookingForWork	170	5.4 ms	5.4 ms
SubmissionQueue_Process	43	4.6 ms	408.5 µs
Blueprint	1,382	4.6 ms	1.2 ms
FTimerManager_Tick	3	4.4 ms	211 µs
NiagaraSystem	100	3.9 ms	1.2 ms
ParallelFor	77	3.9 ms	713.3 µs
FRDGBuilder::Execute	4	3.5 ms	49.8 µs
SphereComponent	2,454	3.5 ms	1.3 ms
ExecuteCommandList	39	3 ms	3 ms
ParallelExecute	45	2.6 ms	2.5 ms
NiagaraEmitter	68	2.5 ms	261.8 µs

Callers	Count	Incl	Excl

Callees	Count	Incl	Excl

Verbosity Threshold · Category Filter · 448 logs

```
Index  Time            Verbosity  Category    Message
0      5m 45.532113s   Display    AudioMixer  Starting AudioMixerPlatformInterface::RunInternal(), InstanceID=3
1      5m 45.532135s   Display    AudioMixer  FMixerPlatformXAudio2::SubmitBuffer() called for the first time. InstanceID=3
2      5m 45.647754s   Display    AudioMixer  Starting AudioMixerPlatformInterface::RunInternal(), InstanceID=4
3      5m 45.647773s   Display    AudioMixer  FMixerPlatformXAudio2::SubmitBuffer() called for the first time. InstanceID=4
4      5m 6.941370s    Log        Bookmark    GC
5      5m 38.791894s   Log        Slate       Updating window title bar state: overlay mode, drag disabled, window buttons hidden, title bar hidden
6      5m 38.791917s   Log        World       BeginTearingDown for /Game/Levels/UEDPIE_0_GameplayMap
7      5m 38.792575s   Log        World       UWorld::CleanupWorld for GameplayMap, bSessionEnded=true, bCleanupResources=true
```

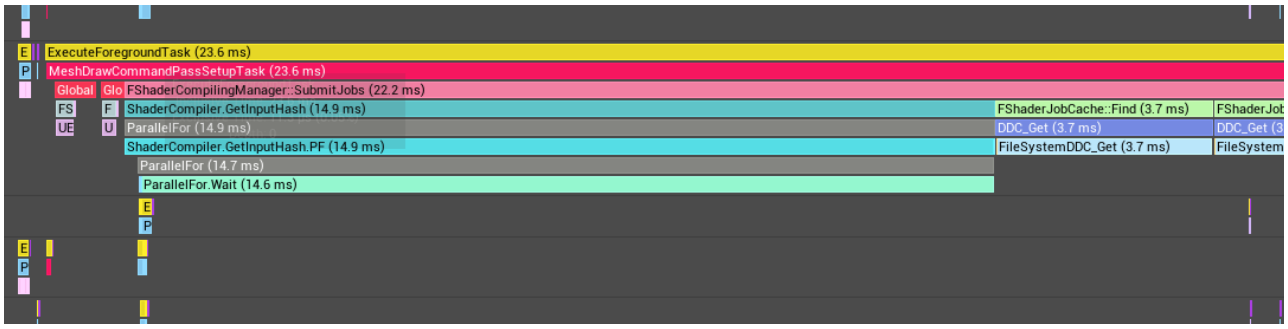
This is the Unreal Insights report. It shows where the frames are bad when the game is running. As it can be seen above, the game is running at an inconsistent pace. The game spikes at the start which is quite normal but then the game spikes a lot in the middle and then spikes at the end.

There are a few things that can cause this. It can be because the player was emitting particle systems which can be quite taxing to the computer. Another thing that can cause it is chaos destruction and post processing. The chaos destruction is heavy in terms of visuals and the post processing is also taxing. Another thing that might be an issue is the wall running mechanic. When the player wall run, it checks through the overlap and the ray trace functions which can be quite intensive behind the scenes.

From these potential problems there are a couple of potential solutions for each problem. For the particle system, less particle systems can be implemented. By less, I mean the spawn rate and the amount of particle systems that spawns in the world. This can make the game's performance better though it can sacrifice the particle's visual aesthetic. Another potential solution is adding an LOD system to the chaos destruction of when it is out of the player's view. This can give the game more performance without sacrificing anything since it is not gonna be visible when it is out of view. It can also be destroyed when it is out of view. The last potential solution is adjusting the wall running mechanic and changing the way it is coded. This solution is considered to be difficult since changing something that’s already working can lead to worse results of it not working. While time consuming, this can be a great solution to solving performance issues for things that are behind the scenes.

Timing Sections Report

Timings Section 1


ExecuteForegroundTask (23.6 ms)
MeshDrawCommandPassSetupTask (23.6 ms)
Global
Glo
FShaderCompilingManager::SubmitJobs (22.2 ms)
FS
F
ShaderCompiler.GetInputHash (14.9 ms)
FShaderJobCache::Find (3.7 ms)
FShaderJob
UE
U
ParallelFor (14.9 ms)
DDC_Get (3.7 ms)
DDC_Get (3
ShaderCompiler.GetInputHash.PF (14.9 ms)
FileSystemDDC_Get (3.7 ms)
FileSystem
ParallelFor (14.7 ms)
ParallelFor.Wait (14.6 ms)
E
P

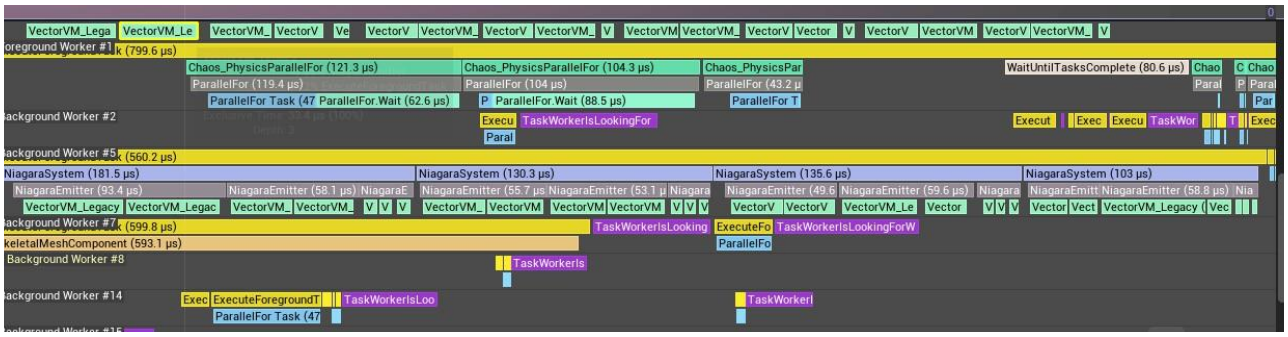
In one of the frames that spiked, this result has been retrieved. It can be seen that the ShaderCompiler is using 14.9 milliseconds of data which is quite a lot. This can be quite taxing to the computer which can cause performance issues.

The potential cause of this is the shaders that are present in the world. The shaders that are present are the Glitcher shaders and the water shader. The water shader shouldn’t be a problem since there are only 3 present but the Glitcher could cause some issues. This is due to the fact that the Glitch Objects are scattered throughout the map and everywhere.

There are a couple of potential solutions for the issue with the shaders. One of the potential solutions is lessening the movement and the dynamic change of the shader. By changing the values and the nodes in the shaders. Another change that is also possible is lessening the actors with the custom shaders. This should be a last resort option since lessening the actor can impact the gameplay of the game itself. The last potential solution that could work is also making the shader disappear when the camera is not rendering them. This could solve the problem of impacting the gameplay while also solving the problem of the extensive shader use in the game.

In conclusion, having a lot of shaders in the scene is not a good idea since it can cause extensive strain into the game. The potential cause of the shader problem from the report is most likely the Glitcher shader. This can potentially be fixed by lessening the number of actors with the shader, lessening the movement and complexity of the shader, and making the shader disappear when it is not in the player’s sight. All of these are potential solutions but there is no guarantee that any of them will work perfectly.

Timings Section 2



In this screenshot, there are a couple of things that are present but I am going to focus on the NiagaraSystem. The Niagara System is triggered 4 times with more than 100 nanoseconds of use which is quite significant.

The potential problem for this is the particle effect since particle systems are made with the Niagara System. The particle effects in discussion are the tentacle particles. This is due to the fact that the tentacle particles are 4 separate particle systems that are attached to the player. By knowing that it is 4 particle systems that function almost identically, I can pinpoint that, that is the major cause of what causes the usage of the system. It is also worth mentioning that everytime it collides, it triggers a set of code and it collides almost every time with the player itself which can be a huge problem of optimization. This is because of how the code works when the tentacles are interacting with the walls it uses a primitive collision that goes between 2 modes : collide or don’t collide.

There are several potential fixes that can be done to the particle system so that it is more optimized. One of the potential fixes is just to delete the particle system. The tentacles are not essential to the gameplay system so deleting it wouldn’t affect the gameplay loop. The downside of this is that it sacrifices the aesthetics of the game. Another potential fix is lowering the spawn rate of the tentacles. By lowering the spawn rate, the collision will be less as well. This can make the game more optimized but this will sacrifice the tentacle’s visuals. The last potential fix that can be done is adding a custom collision to the tentacles. Adding this can be quite trivial since this means adjusting the code of the NiagaraSystem itself to detect collision more efficiently which may need extensive knowledge of the game engine itself. Though it is time consuming, these potential fixes are worth it to increase the game’s performance and upgrade the immersivity for the players that are playing the game.