Technical Design Document for BlockoutShooter

Jiangye Song

Contents

Project Overview

Blockout Shooter is an engaging 1 vs 1, third-person shooter game designed for PC. This game introduces a unique and colorful mechanics to the traditional shooter genre, making it interesting to a wide range of players. The central mechanic of Blockout Shooter revolves around color-matching and strategic gameplay within a dynamic arena filled with colorable blocks.

Game Mechanics Overview

Blockout Shooter introduces an innovative blend of color-matching combat and strategic gameplay in a 2-player, third-person shooter experience. Players are assigned specific colors, and they can only use their weapon when their standing on blocks that have the same color with them (or uncolored blocks). Additionally, power-ups can be acquired through defeating opponents or pushing the trees, enhancing the overall depth and excitement of the gameplay. These mechanics combine to create a dynamic and engaging multiplayer experience.

Target Platform

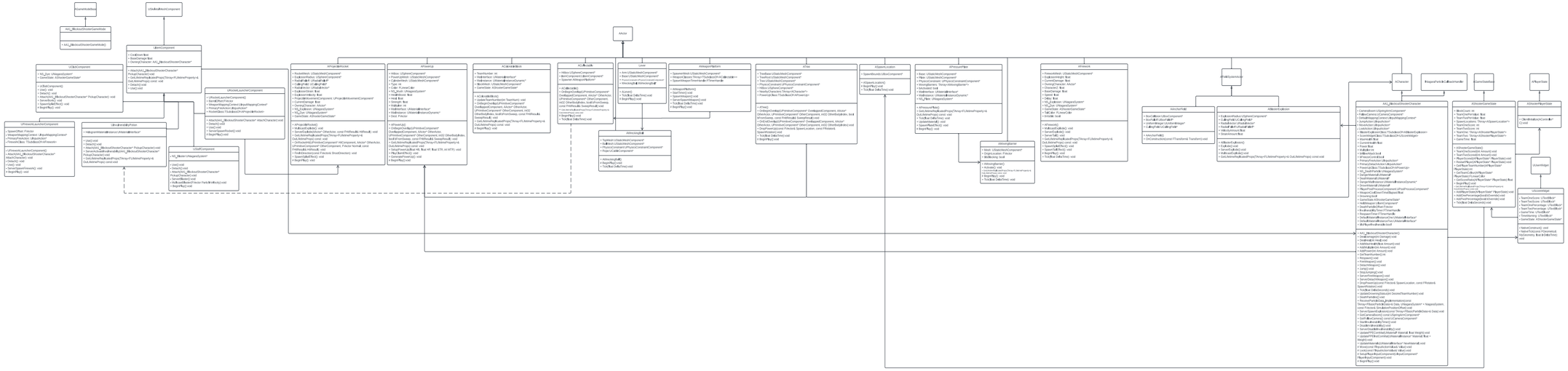
The primary target platform for Blockout Shooter is PC, leveraging the precision of mouse and keyboard controls for an optimal gaming experience. If pursued, mobile platform can be a target platform because touch controls can be applied on TPS. On mobile platforms, the game will utilize intuitive touch controls for aiming and movement, ensuring accessibility and engagement. However, target platform will not include consoles because it will be hard to aim with a controller, and the gameplay did not optimize for the controllers.

Game Mechanics

UML Diagram

This game used the template of Third Person as a starting point. Classes provided by unreal and do not contain any modification will not show its attributes and functions in this UML, shows as a simple class shape.

This diagram is created via Lucidchart.



Movement Mechanics

The movement mechanics of the players did not change in compared to original Unreal Engine's template in normal case. The gravity scale is 1.75, the maximum walk speed is 500 cm/s, the max jump count is 1, and the max step height is 45.0 cm. These defaults values are already good enough to create a fair and enjoyable arena.

However, if the player is on an enemy-colored colorable block, they start to drown. Their maximum walking speed will decrease to 200 cm/s, and the gravity scale raises to 5. This is aimed to give the player a penalty to tell player the importance of dyeing blocks and tell player to avoid the enemy-colored colorable blocks as much as possible.

Player characters can step on all the static meshes, geometric collection objects (including the wall window and fragile obstacles) and physical-constrain enabled actors (including lever, wrecking ball, and pressure plate) in the map, and they cannot step on item components, collectables, particle effect, etc. Player characters can step on powerups, but the powerups will be consumed when player characters touches their hitboxes.

Colorable blocks collision profile is made as overlap all in the game and player characters are not actually walking on it. Instead, the player characters are stepping on the meshes below the colorable blocks (so colorable block is more like a tile).

Controls

Mapping	Action	Description	Keyboard Control Binding	Modifiers
Walking Mapping	Move Forward	Used to move the player forward relative to the camera direction	W Up	Swizzle Input Axis Values (YXZ)
	Move Left	Used to move the player left relative to the camera direction	A Left	Negate (XYZ)
	Move Backward	Used to move the player backward relative to the camera direction	S Down	Swizzle Input Axis Values (YXZ) Negate (XYZ)
	Move Right	Used to move the player right relative to the camera direction	D Right	None
Jumping Mapping	Jump StopJumping	Used to make the player jump	Space	None
Looking Mapping	Look	Used to rotate the follow camera of the player	Mouse Movement	Negate(Y)

Firing Mapping	Fire Weapon	Used to use the weapon that player is currently holding	MouseLeftClick	None
Detach Mapping	Detach Weapon	Remove the weapon that player is currently holding	Q	None
Fullscreen	Toggle Fullscreen	Used to switch the game window between Fullscreen and windowed	F11	N/A

Movement input will be translated into a 2D vector referred to as "MovementVector" within the Unreal Engine. Unreal will initially determine the forward direction based on the camera rotation. Then, it will calculate both the right and forward directions through the GetUnitAxis function. These directions will be combined with the MovementVector values to dictate the character's movement.

Similarly, aiming input will be transformed into a 2D vector known as "LookAxisVector." Unreal Engine will directly apply the values of this vector to the controller, enabling precise control over the player's aim within the game.

The input of jumping will set bPressedJump to true, it will then start counting the jump key hold time which is used to decide the jump height of the player. Eventually, the DoJump function in CheckJumpInput will perform the jump.

When the player presses the weapon fire button, FireWeapon function will be called. If the player is holding a weapon and they’re allowed to attack, it will then call the Use function of that weapon.

Additional Gameplay Mechanic 1

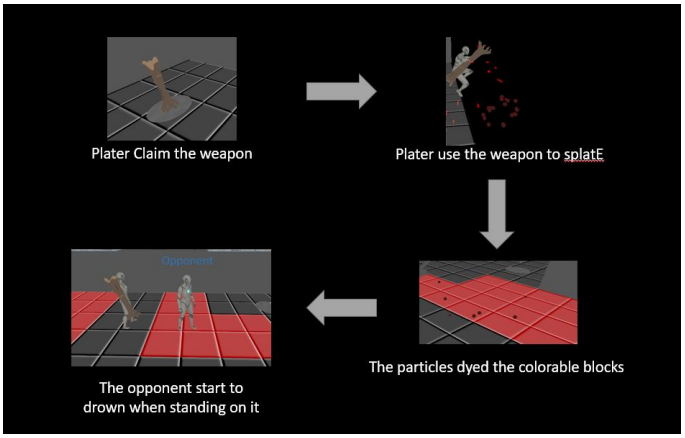
The Splatoon-like colorable blocks.

Mechanic Overview

The Colorable Block mechanic is a fundamental element of Blockout Shooter, and the inspiration is from Splatoon by Nintendo. In the arena, players can use their weapons to unleash color blasts that dye colorable blocks within the game world to match their assigned color. These colorable blocks play a crucial role in shaping the battlefield and influencing player strategy.

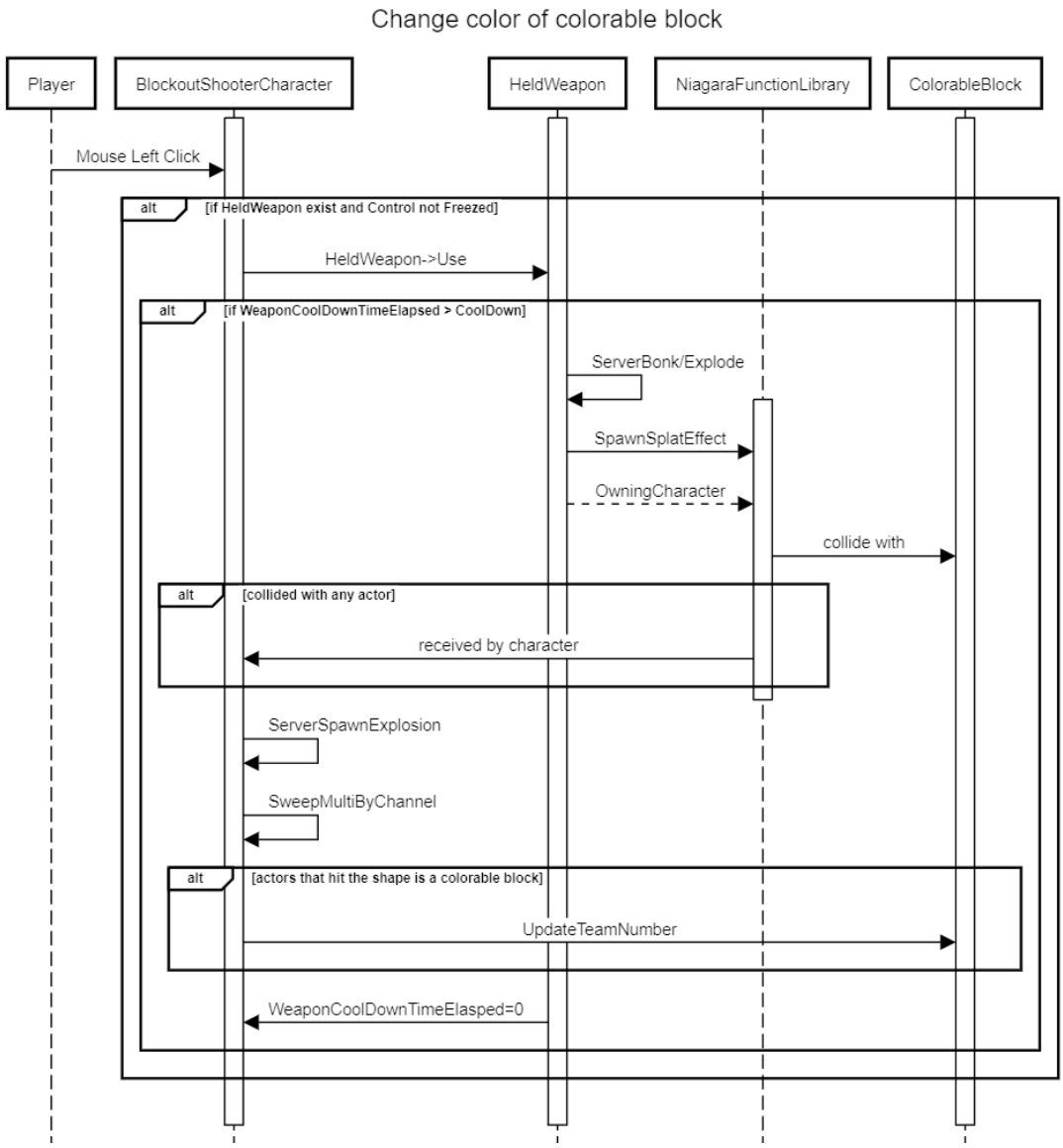
Mechanic Description / Functionality

Colorable blocks serve as both a tactical advantage and a potential obstacle. These blocks initially start as uncolored and won't affect any player's side. When a player uses their weapon, the weapon will collision enabled particles, and each of the particle create a sphere collision with a radius of 10, which will dye all the colorable blocks it collided to the player's assigned color (blue for team one and red for team two for now) by calling UpdateColor function from them. While a player is colliding with the colorable block, the block will call the UpdateDrowningStatus of the player to notify player to check whether the color is their enemy's color. The player will set drowning status to themselves if they are on an enemy-colored colorable block and will remove (set to false) the drowning status if they are not. The movement values will be changed whilst the drowning status is updated, the exact value of the changes is described in Movement Mechanics.

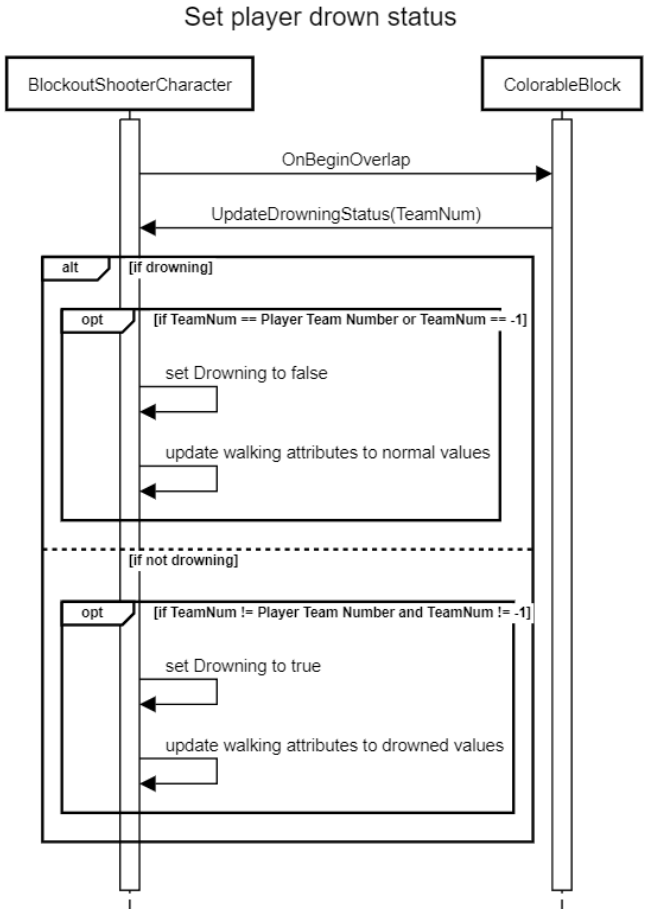


Sequence Diagram

Change color of colorable block



Set player drown status



Additional Gameplay Mechanic 2

Power-ups that change bullets into different types.

Mechanic Overview

The Power-Up mechanic in Blockout Shooter draws inspiration from classic arcade games like Bomberman, introducing a variety of power-up items to enhance player abilities.

These power-ups are represented by mushroom within the Blockout Shooter game, each offering unique benefits when collected by players. However, once the player died, they lose all the power-ups they claimed on respawn!

Mechanic Description / Functionality

Blockout Shooter features different types of power-ups, each denoted by a distinct mushroom color:

- Red Mushroom: Increases the player's attack power. When a player uses a weapon, the weapon's damage is recalculated, incorporating the player character's Power attribute, which determines the extent of the damage

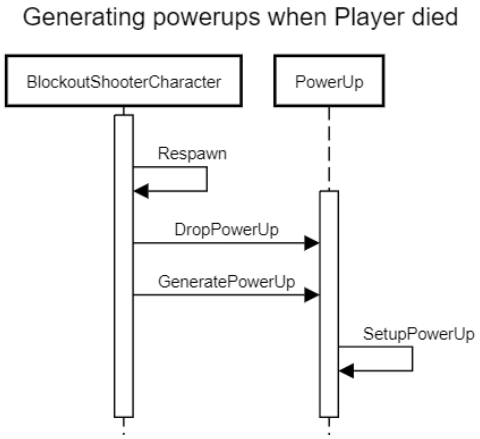
boost (damage = base damage * power). This enhanced damage output empowers players to engage their opponents more effectively.
** Keep in mind: The damage of a staff is made to always 0, as it's too advantageous for dyeing blocks .*

- Green Mushroom: Boosts the player's hit points (HP increase by 40-80), or increasing both current and maximum HP (HP increase by 25-50, with a 0-30 health boost), thus providing survivability advantages.
- Yellow Mushroom: Combines the benefits of increased HP and enhanced attack power, offering a well-rounded advantage to the player.
- Blue Mushroom: Amplifies the "multiplier" of the player's weapon. The impact of this multiplier varies based on the player's choice of weapon:
    - Club: The player gains a 300 more strength from each multiplier when pushing opponents. This enhanced strength allows the player to push their opponents off cliff more easily.
    - Rocket Launcher: The player's weapon spawns [Multiplier] rockets upon use, with each rocket deviating by a 30-degree angle from the others. This creates a wider area of effect and increases the area damage potential.
    - Staff: The player's weapon spawns [3 * Multiplier] blaster shots upon use. This rapid-fire rate provides an efficient way to dye the colorable blocks, making it become a comeback weapon when the other player has already occupied a lot of blocks.
    - Firework Launcher: The player's weapon casts a [Multiplier] number of fireworks in sequence, with a 0.4-second delay between each firework. This allows for a dazzling and visually striking attack that can disrupt opponents and add a level of unpredictability to engagements.

The ability provided by each power-up is randomly selected from the above, which adds an element of unpredictability to the game. This randomness is achieved through a function called "GeneratePowerUp" within the Powerup class. It utilizes the "RandRange" function to determine the power-up type, which then executes the "SetupPowerUp" function with appropriate parameters based on the random selection.

Sequence Diagram

Generating powerups when Player died



Multiplayer

Game State & Player State

Blockout shooter game contains player states which are held by each player, and a game state which stores the teams by storing the player states in different arrays.

Game State

Function	Description
AShooterGameState()	Constructor that initializes the score of both teams to 0.
TeamOneScored(int)	Increase the score of Team 1.
TeamTwoScored(int)	Increase the score of Team 2.
PlayerScored(APlayerState*)	Increase the score of a team by one depends on the PlayerState.
RestartPlayer(APlayerState*)	Respawn the player to the spawn points according to their team number.
GetPlayerTeamNumber(APlayerState*)	Return the team number of a player by searching their PlayerState from the TArrays.
GetTeamColour(APlayerState*)	Return the color of the team by searching their PlayerState from the TArrays.
GetScoreRatio(APlayerState*)	Return the current ratio of the current team in compare with the other team.
AddPlayerState(APlayerState*)	Add the player to a team by inserting their PlayerState to the teams' TArray.
AddOnePercentage(bool)	Increase the percentage of team one which represent the percentage of colorable blocks dyed by team one, decrease team two percentage when bool is true.

AddTwoPercentage(bool)	Increase the percentage of team two which represent the percentage of colorable blocks dyed by team two, decrease team one percentage when bool is true.

Property	Description
TeamOneScore	Team 1's score in int. Team score should showed consistently in each client.
TeamTwoScore	Team 2's score in int. Team score should showed consistently in each client.
TeamOne	A TArray that contains all PlayerStates that owned by Characters in TeamOne. Team members should contains the same players in each client.
TeamTwo	A TArray that contains all PlayerStates that owned by Characters in TeamTwo. Team members should contains the same players in each client.
SpawnLocations	A TArray that contains all the spawn locations to respawn the player characters.
BlockCount	Represent the number of colorable blocks of the game, used to calculate the amount to increase/decrease the percentage of each team.
GameTime	The countdown time of the match in the game.

Player State

Property / Function	Description
ClientInitialize(AController*)	Called by Controller when its PlayerState is initially replicated. Add this player state to the GameState.

Class Replication

Class	Property Name	Description
BlasterExplosion	VelocityAmount	The velocity that will used to spawn the explosion.
	StrainAmount	The strain that will used to spawn the explosion.
Collectable	Hitbox	The collision hitbox for the collectable item.
	ItemComponent	The component represents the collectable item. All clients should have the same item component for each item.
Firework	FireworkMesh	The mesh of the firework.
	OwningCharacter	The player character who launched the firework, not replicating may cause it became null for some client.
	bVisible	
ProjectileRocket	RocketMesh	The mesh of the rocket.
	OwningCharacter	The player character who launched the rocket.
	ProjectileMovementComponent	The component responsible for the rocket's movement, replicate to make the actor move smoothly with the same trail.
InvulnerabilityPotion	HologramMaterialInstance	The material instance controlling the appearance of the invulnerability potion's holographic effect.
ItemComponent	OwningCharacter	The player character who collected the item.
MovingBarrier	Mesh	The mesh of the barrier.
	bIsBlocking	A Boolean indicating whether the barrier is currently blocking (or trying to block) player movement, replicate to make sure the barrier is at the same state for all clients.
PressurePlate	MatInterface	The material instance of the material interface.
	MatInstance	The material interface of the material.
ColorableBlock	TeamNumber	The team number of the team that represented by the current dyed

		color. Initially -1 represented the block is uncolored.
	BlockMesh	The mesh of the block.
	MatInstance	The material instance of the material interface.
	MatInterface	The material interface of the material.
PowerUp	PowerUpMesh	The mush mesh of the power up.
	CylinderMesh	Lower part of the mush mesh.
	Hitbox	The collision hitbox of the power up.
	MatInstance	The material instance of the material interface.
	MatInterface	The material interface of the material.
	HealthBoost	The amount of the health boost that will be applied to the player when the player is taking this power up.
	Heal	The amount of the instant heal that will be applied to the player when the player is taking this power up.
	Strength	The amount of the strength that will be applied to the player when the player is taking this power up.
	Multiplier	The amount of the multiplier that will be applied to the player when the player is taking this power up.
	Type	The type of the power up.

Remote Procedure Calls

Class Name	Function	Type	Description
ClubComponent / BlasterExplosion / Firework / ProjectileRocket	ServerBonk / ServerExplode	Server, Reliable	Complete the action on the server to make the damage result is synchronized, as well as the net multicast functions inside it.
RocketLauncherComponent / FireworkLauncherComponent	ServerSpawnFirework / ServerSpawnRocket	Server, Reliable	Complete the action on the server to make the actors created across all nodes.
ClubComponent / Firework / ProjectileRocket	SpawnSplatEffect	NetMulticast,Reliable	Play the splat effect on all client and server sides.
ColorableBlock	UpdateTeamNumber(int)	Server, Reliable	Update the team number of the team that represented by the current dyed color.
BlockoutCharacter	DropPowerUp(FVector&, FRotator&)	Server, Reliable	Create a power up on the specified location with specified rotation on the map.
	UpdateDrowningStatus(int)	Server, Reliable	Update the status of the drowning status of the current player.
	DeathParticles	NetMulticast, Server, Reliable	Play the Death particle effect on all nodes.
	ServerFireWeapon	Server, Reliable	Use weapon on the server to make it synchronized.
	ServerDetachWeapon	Server, Reliable	Remove weapon the action on the server to make sure the weapon destroyed on all nodes.

	ServerSpawnExplosion	Server, Reliable	Spawn explosion caused by the blaster or dye particles when they collided with something.
	ServerDisableInvulnerability	Server, Reliable	Disable the invulnerability of a player character.
	UpdatePPEComMat / UpdatePPEInstComMat	Client, Reliable	Update the post process effect for client's follow camera's ppe component according to their character status, must not synchronized.
	UpdateMaterials	NetMulticast, Reliable	Change/reset the player material, need to have the same material for all players to see
PowerUp	SetupPowerUp(float,float,float,int)	Server, Reliable, Multicast	Setup the mesh attachment and save the values of its properties.
	GeneratePowerUp	Server, Reliable	Randomly select the type of the power up.
Tree	DropPowerUp(FVector&, FRotator&)	Server, Reliable	Create a power up on the specified location with specified rotation on the map.

Physics Constraint 1 - Lever

A Lever to control the wrecking ball.

Overview of Interaction

The lever is a physical object within the game world that serves the purpose of introducing dynamic, and its interaction is based on physics constraints and the tick function. Upon activation, the wrecking ball will start swinging back and forth due to physics. The swinging motion of the wrecking ball can be observed visually in the game environment. It is designed to hit and break the wall on the edge of the map.

Interaction Description

How the Interaction Works

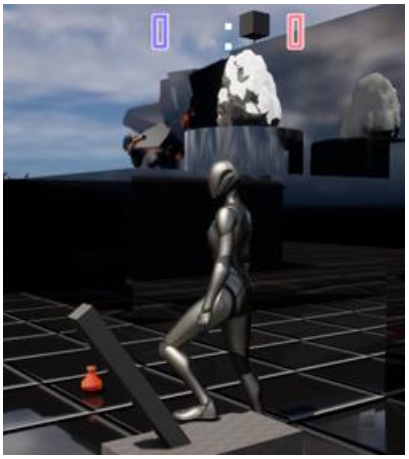
The wrecking ball does not have gravity at the start so it cannot swing. When the lever is pulled by a player, the tick function detects the angle and enables the gravity of the wrecking ball that is selected in the detailed panel. The swinging wrecking ball collided with the wall on the edge of the map, exerting a force on it and started to collapse.

Inspiration / Reference Images

The inspiration for the lever and wrecking ball physics interaction in Blockout Shooter is from the game Human: Fall Flat. This game is known for its physics-based puzzles, reveal the fun for unpredictable outcomes from interactions.

(Retrieved from: https://www.youtube.com/watch?v=_4EhMRdT1XI)

In-Engine Screenshots

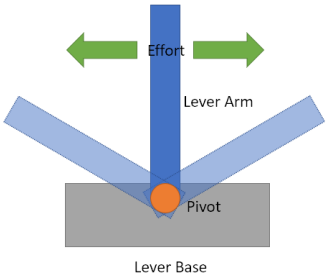


The player broke the wall by activating the wrecking ball on the corner of the map by pushing the lever.

Properties and Values

Property	Description of Purpose	Value
Angular Limit	The limit on amount of rotation that can be applied as part of lever.	30 Degrees
Limit Stiffness	The bounce force of the lever when a limit is exceeded.	50
Angular Motor Target Orientation	The rotation the lever tries to rotate to (in this game, the original orientation).	FRotator(0,0,0)
Angular Motor Target Orientation strength	The strength at which the lever tries to rotate to a rotation	50

Diagram of Interaction



Physics Constraint 2 - Tree

A Tree that provides powerups.

Overview of Interaction

The Tree actor introduces a strategic gameplay element into Blockout Shooter, emphasizing player decisions and risk management. While it does add dynamism to the game environment, its primary purpose is to create opportunities for strategic gameplay. Players can push the Tree to earn power-ups from them, but this action might also expose them to attacks from the opponent. Moreover, the thrust force generated by the Tree actor can push the player outside the arena, resulting in their elimination, adding a layer of risk to the interaction.

Interaction Description

How the Interaction Works

As the tree does not have a physical collision, a tree root mesh is added and shows as a cube in the engine but will not be visible in the game. When player is trying to push the tree, it is actually pushing the cube (tree root) which also drives the tree to rotate the destination rotation.

Tracking by the tick function, when the Tree actor is pushed, it responds by bouncing back. The extent and direction of this bounce are determined by the magnitude and direction of the force applied by the player. The vector is calculated by vector subtraction, but the z-axis force will force set to 500. The tick function will also create a random power-up near the tree.

To prevent player from being thrust to an extremely high or far place, the velocity is limited in BlockoutShooterCharacter tick function by 1000.

Inspiration / Reference Images

None.

In-Engine Screenshots

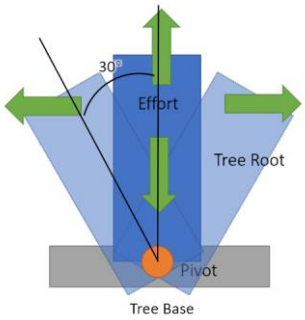
The player is thrust out from arena when pushing the trees.

Properties and Values

Property	Description of Purpose	Value
Angular Limit	The limit on amount of rotation that can be applied as part of lever.	30 Degrees

Angular Motor Target Orientation	The rotation the tree tries to rotate to (in this game, the original orientation).	FRotator(0,0,0)
Angular Motor Target Orientation Strength	The strength at which the tree tries to rotate to the rotation.	100
Angular Motor Target Velocity	The target angular velocity for the motor.	FVector(0)
Angular Motor Target Velocity Strength	The strength to reach the target angular velocity.	5

Diagram of Interaction



Physics Constraint 3 - PressurePlate

Overview of Interaction

The PressurePlate is a dynamic element within the Blockout Shooter game, designed to introduce interactivity, strategy and change the game environment. Its primary function is to influence the state of "MovingBarrier" objects in the game world. Upon activation, it initiates a cascading effect by altering the "bIsBlocking" Boolean attribute of the associated MovingBarriers. This interaction leads to changes in the position and behavior of these barriers, dynamically affecting player movement and strategy.

Interaction Description

How the Interaction Works

The operation of the PressurePlate interaction is based on a sequence of events. When a player or object (like wall / box fragments) step on/hit with the PressurePlate, it activates a trigger event. The PressurePlate then change the "bIsBlocking" Boolean attribute of the associated MovingBarriers. This attribute determines whether the barriers are actively blocking player movement or hidden beneath the ground.

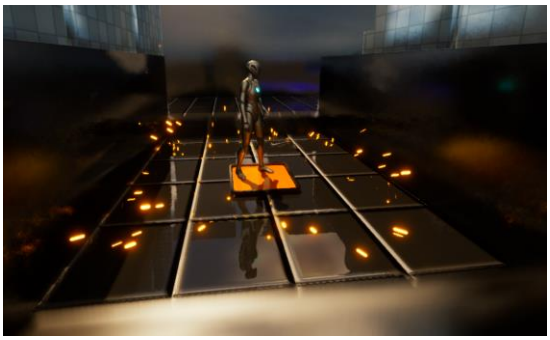
Inspiration / Reference Images

The inspiration for the PressurePlate interaction in Blockout Shooter comes from the game Superliminal. In Superliminal, players often encounter pressure plates as mechanisms for opening doors and progressing through the game world. These

pressure plates require players to step on / place objects on them to trigger their effects.

(Retrieved from: my own recordings)

In-Engine Screenshots

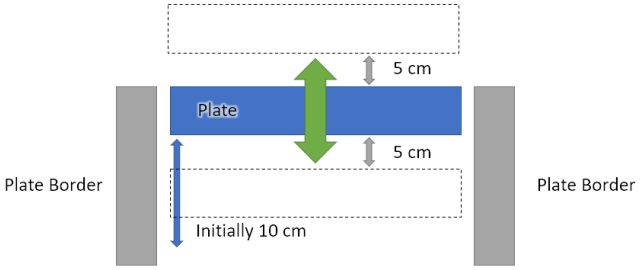


The player stepped on the pressure plate to lower the moving barrier.

Properties and Values

Property	Description of Purpose	Value
Linear Limit Z Motion Limit	The limit on amount of displacement that can be applied as part of lever.	5 cm
Linear Motor Position Target	The bounce force of the lever when a limit is exceeded.	FVector(0,0,0)
Linear Motor Position Target strength	The strength at which the lever tries to rotate to a rotation (in this game, the original position).	50

Diagram of Interaction



Advanced Niagara Particle Effect

Niagara Particle Effect - NS_Mush

NS_Mush is the power-up picking effect in BlockoutShooter.

Overview of Effect

The NS_Mush Niagara particle effect is designed to enhance the visual aspects of power-up mushroom pickups within the game. This effect provides a captivating visual representation of the power-up acquisition. It communicates to players that they have collected a power-up and accentuates the significance of this in-game action.

Effect Description

At the center of the effect, the SingleLoopingParticle emitter represents the location where the power-up mushroom was collected. This emitter features particles that dynamically grow (0-0.2 seconds) and shrink (0.3-0.5 seconds) in size, mimicking the pulsating effect of a power-up. The color of these particles matches that of the power-up mushroom, creating a visual connection between the effect and the collected item.

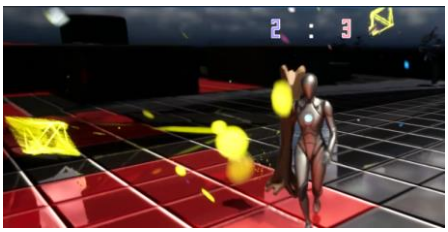
A mesh burst emitter complements the central effect by creating a burst of mesh particles that resemble the contour of a diamond. These mesh particles share the color of the power-up mushroom and rotate gracefully. As time progresses, these particles confluence back toward the center eventually, aligning with the shrinking effect of the SingleLoopingParticle emitter.

There are two BlowingParticles emitters operating simultaneously to add an extra layer of visual excitement. One emits confetti in random colors, adding a vibrant and dynamic aspect to the effect. The other emitter blows confetti in a color that matches the power-up mushroom's color, reinforcing the thematic connection between the effect and the collected power-up.

In addition to the previously described elements, the SingleLoopingParticle and the mesh burst emitters now include a point attraction force. This position is set as a parameter "Dest," represents a potential location where the player might be situated at that moment. As the effect progresses, the attraction force gradually increases from zero, pulling towards the specified "Dest" location.

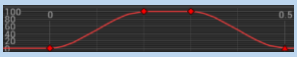
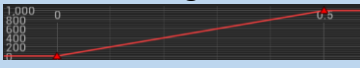
Inspiration / Reference Images:

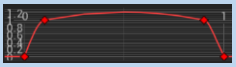
In-Engine Screenshots:



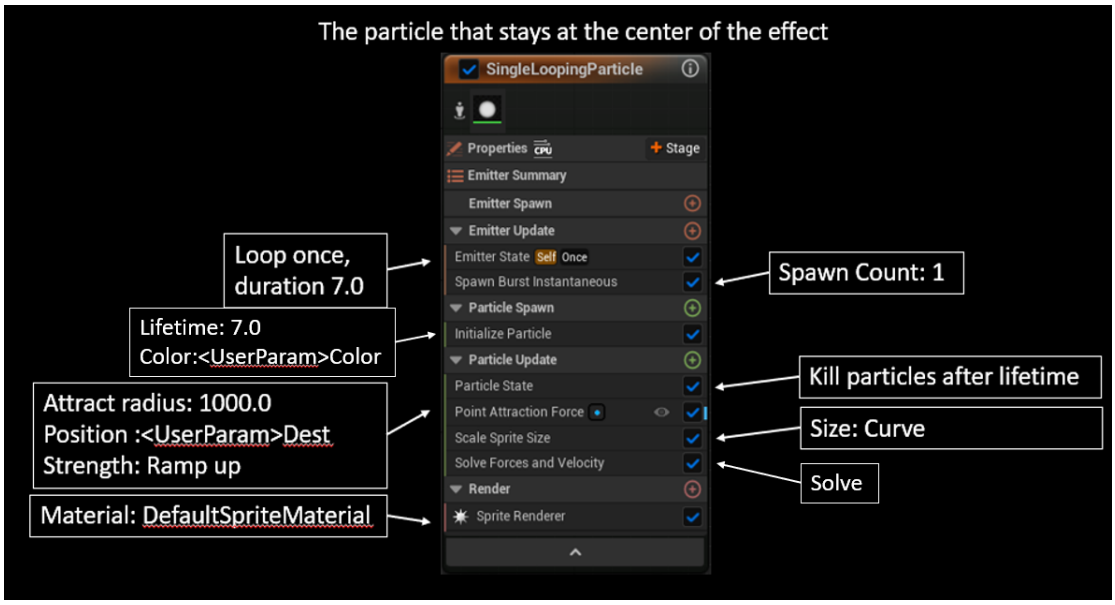
The player claimed a yellow mush.

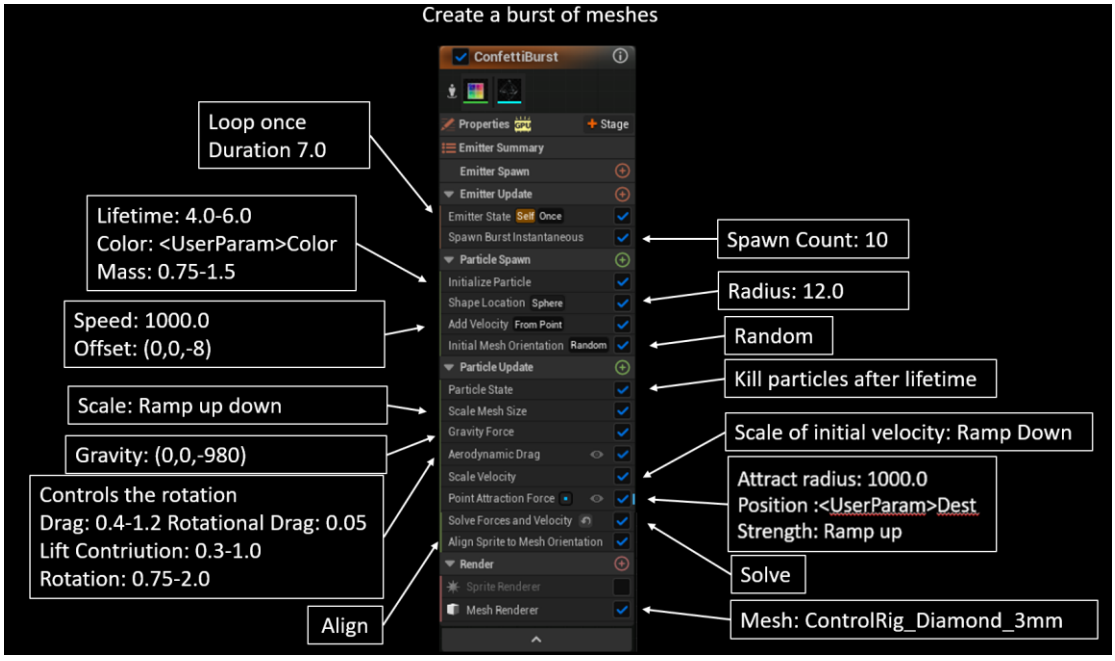
Properties and Values

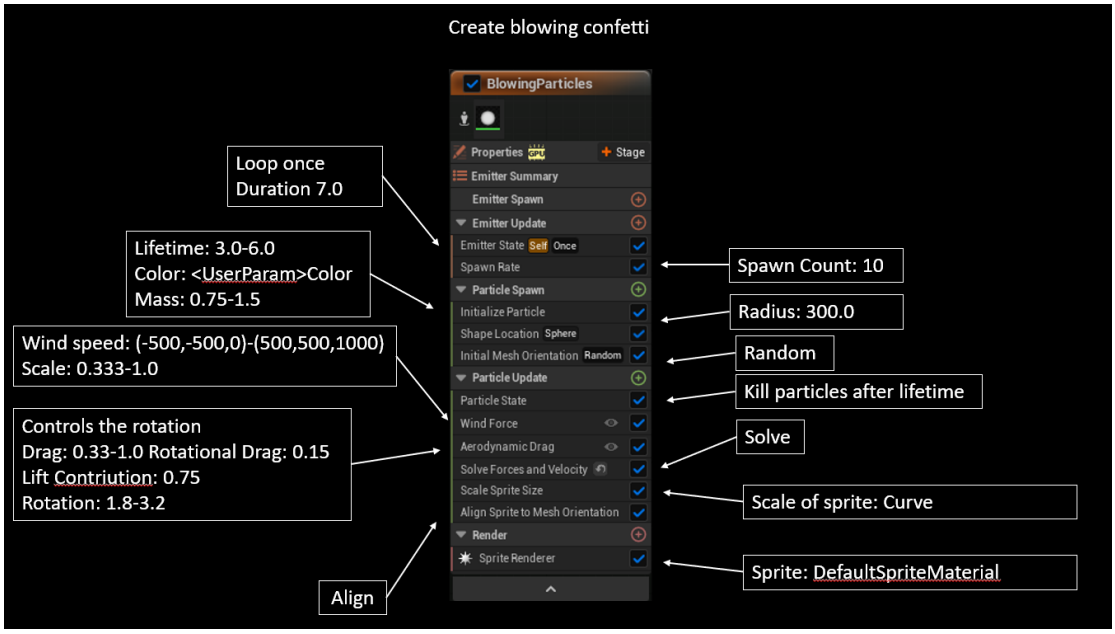
Emitter	Property	Description of Purpose	Value
SingleLoopingParticle	Scale Sprite Size	Controls the size of the sprites.	Scale Factor: 
	Point Attraction Force	Control the strength and location of the attraction force.	Strength:  Position: Dest (Parameter)
ConfettiBurst	Shape Location [Sphere]	Controls the location of the meshes.	Radius: 12.0f
	Add Velocity [From Point]	Controls the strength of force of the burst.	Velocity Speed: 1000.0f
	Scale Mesh Size	Controls the size of the meshes.	Scale Factor: Ramp Up Down Curve
	Aerodynamic Drag	Controls the rotation of the meshes.	Aerodynamic Drag: Random Range Float 0.4-1.2
	Point Attraction Force	Control the strength and location of the attraction force.	Strength:  Position: Dest (Parameter)
BlowingParticles	Wind Force	Applies a wind speed to the particles	Random Range Vector (-500,-500,0)-(500,500,1000)

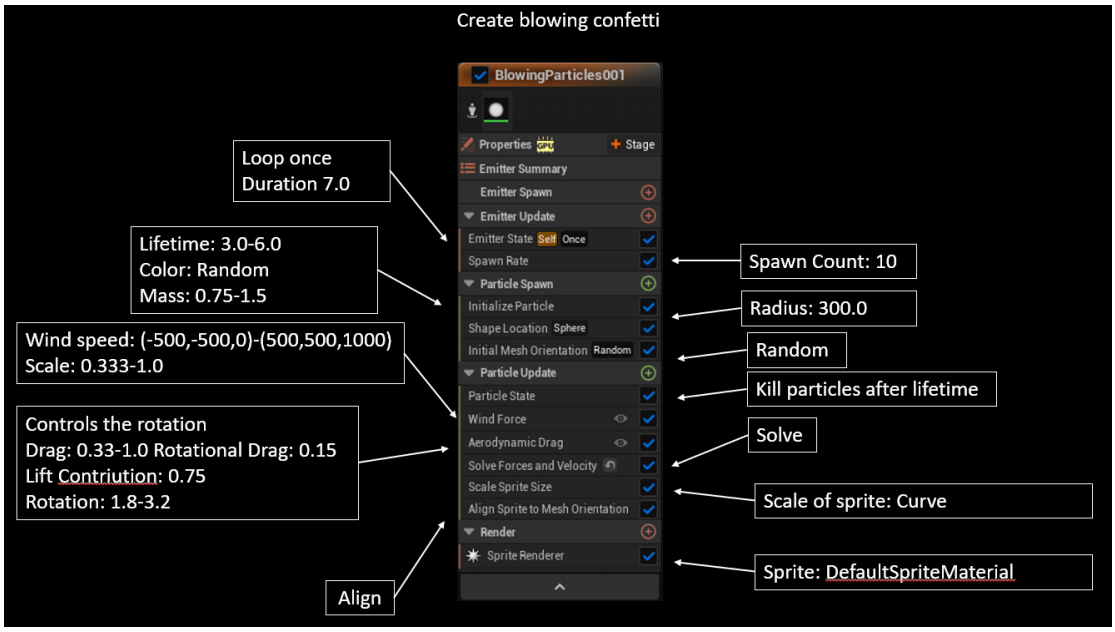
	Scale Sprite Size	Controls the size of the sprites.	Scale Factor: 

Niagara System / Emitters Breakdown









C++ Parameters Breakdown

Provide detailed information on the C++ exposed parameters used for creation of the particle system.

Parameter	Description
Color	The color of the mush mesh of the power up that is taken. Used as color of confetti, particle, and mesh.
Dest	The potential location of the player after 0.4 seconds. Used for point attraction force Vector.

Destruction Aware Niagara Particle Effect

Niagara Particle Effect – NS_TrailingPiece

NS_TrailingPiece is a fire trailing effect.

Overview of Effect

The NS_TrailingPiece Niagara particle effect enhances the visual impact of fragile actors' destruction within Blockout Shooter. This effect responds to the collapse of fragile actors (glass wall / fragile obstacles), creating a more chaotic view. Its purpose is to intensify the sense of chaos of destruction, which attracts the player to break them more, enhancing the overall gameplay experience.

Effect Description

The NS_TrailingPiece effect listens for Chaos destruction data triggered from the world. When a fragment is marked for destruction, the NS_TrailingPiece effect generates 5 additional fragments behind the fragments from the fragile actors. It applies itself a velocity (from point) to make each fragments moves more randomly.

Inspiration / Reference Images:

The NS_TrailingPiece Niagara particle effect is commonly seen in games, movies, and the reality when structures crumble or explode.

In-Engine Screenshots:

The player broke the wall using a staff.

Properties and Values

Property	Description of Purpose	Value
Spawn from Chaos	Spawn particles based on event data from a chaos solver.	Parameter in Chaos Destruction Data Spawn Percentage Fraction: 5.0
Apply Chaos Data	Set position, velocity, and color from a chaos solver.	Parameter in Chaos Destruction Data
Add Velocity [From Point]	Controls the initial velocity of the mesh.	Velocity Speed: 25.0f
Drag	Air resistance.	Drag: 1.0f
Mesh Rotation Force	Controls the rotation force of the meshes.	Rotation: (-10,-10,-10)-(10,10,10)

Collision Enabled Niagara Particle Effect

Niagara Particle Effect – NS_Bathtub

Overview of Effect

The NS_Bathtub Niagara is an essential particle effect to enhancing the core gameplay experience in Blockout Shooter. Its primary function is to dye the colorable blocks with a more dynamic and visually engaging approach. When players use their weapons, the bubbles in NS_Bathtub effect will start to splat, creating a visually captivating spectacle. The bouncing bubbles enhanced the gameplay experience by providing dynamic visual feedback which attracts player characters to dye more colorable blocks on the map.

Effect Description

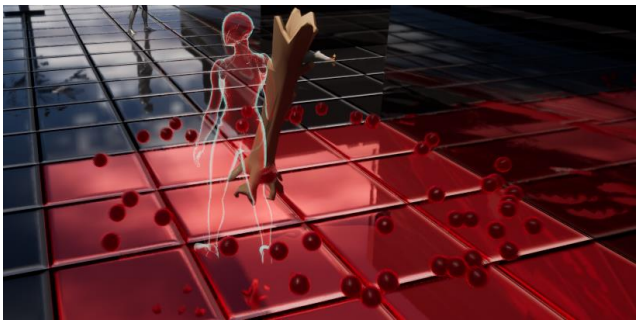
When a player uses their weapon, the NS_Bathtub effect is triggered. It generates a burst of bubbles from a specific center point of an object. The bubbles within the effect are collision-enabled, meaning they interact with the actors in the map. When these bubbles hit the ground, wall, or any surface, they bounce off. When they overlap with colorable blocks, they will create shape collisions to dynamically change the color of the blocks to match the player's team color. This innovative approach turns the NS_Bathtub effect into a critical gameplay element, and added more strategies for the player (for example, the player can shoot a wall to create bubbles to dye the blocks that near their opponent).

Inspiration / Reference Images:

The inspiration for the NS_Bathtub effect from Blockout Shooter is from the weapon bathtub in Splatoon. Where players can splat bubbles out to color the arena. Using similar effects aiming to capture the same level of visual appeal and makes more sense for the main topic of Blockout Shooter (which is coloring the blocks) the same time.

(Retrieved from: my own recordings)

In-Engine Screenshots:



The dye effect appears when a player uses their club.

Properties and Values

Property	Description of Purpose	Value
Add velocity [in cone]	Add a velocity to the mesh in cone.	Velocity Speed: 300.0f
Collusion	Cast rays to calculate its collusion in the world.	Bounce: 1.2

C++ Interaction Description

The NS_Bathtub effect is equipped with a collision module, enabling the bubbles it generates to interact with objects in the game world. It also holds a reference to the player character who spawned it through the "ClassToTell" object parameter. This reference establishes a connection between the particle effect and the player character. The "Export Particle Data to Blueprint" module is set to export data under the condition of a collision event, ensuring that relevant information is passed to the Blueprint only if particle collided. The Callback Handler Parameter for the NS_Bathtub effect is set as "ClassToTell" object, which is the player character who spawned this effect.

The player character class in Blockout Shooter is a child class of "INiagaraParticleCallbackHandler." This means that it has the ability to receive particle data from the NS_Bathtub effect. When any particles from any effect collide with any object, the player character's "ReceiveParticleData_Implementation" function is triggered. Inside this function, for each particle collision event, the player character generates a small FCollisionShape sphere with a radius of 10 cm. This sphere is positioned at the location of the bubble particle's collision. If the sphere collides with a colorable block in the game world, the player character calls the "UpdateTeamNumber" function for that specific block.

This comprehensive process ensures that the NS_Bathtub effect, the player character, and the game environment are seamlessly interconnected. It allows for dynamic interactions and strategic color control within the Blockout Shooter game, enhancing the gameplay experience.

Shader Effects

Shader Effect 1 – M_Bubble

Overview of Effect

The M_Bubble shader effect is a visually charming material in Blockout Shooter. When a player character obtains an invulnerability potion, the M_Bubble material will apply on their appearance, creating a unique, hologram-like aesthetic. This effect impacts gameplay by signifying the player's enhanced state, making it easier for both the player and the opponent to identify invulnerable characters during intense battles.

Effect Description

The M_Bubble shader material seems like an air bladder at first glance, featuring a transparent and reflective surface. This material is designed to convey the player character's invulnerable status.

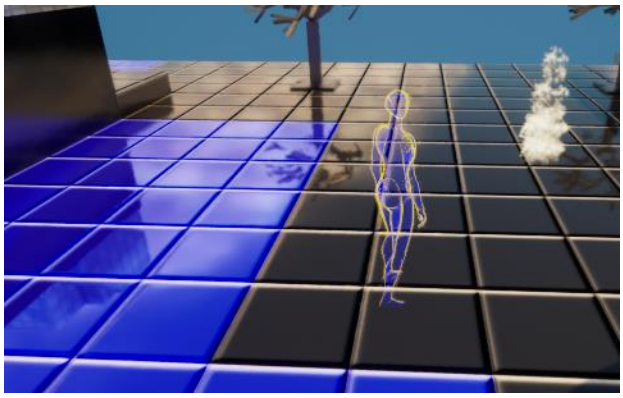
M_Bubble is designed to reflect two distinct colours. The central portion of the material reflects a base colour, while the outer edges reflect a secondary colour. Both colours are dynamically controlled by parameters, ensuring that they align with the player character's team affiliation. In C++ code of BlockoutShootherCharacter, the base colour parameter of the M_Bubble material is set to match the player's team colour, providing a clear visual representation of their alliance. The secondary colour parameter, on the other hand, is derived from the reverse colour of the player's team.

Inspiration / Reference Images:

Such transparent material is widely used across games when characters in game are invincible, invisible, have special vision, etc. For example, in Hogwarts Legacy, the character turns transparent when using Disillusionment spell.

(Retrieved from: my own recordings)

In-Engine Screenshots:

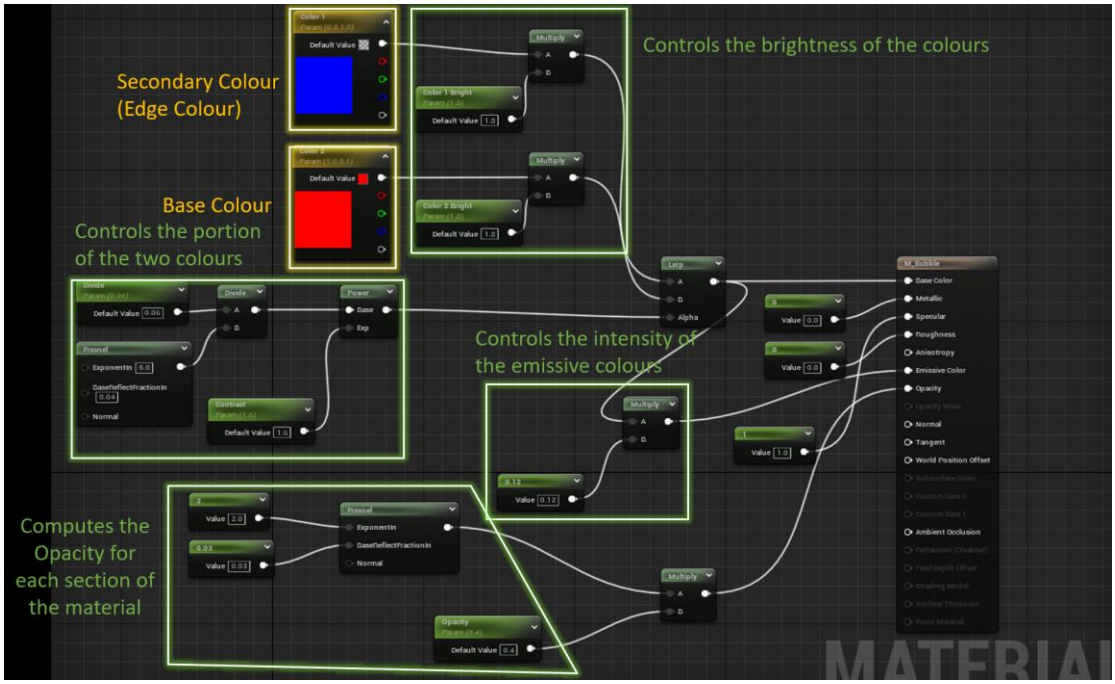


The blue team player claimed a potion and is invulnerable now.

Properties and Values

Property	Description of Purpose	Value
Color 1	Used to control the edge color of shader dynamically	Float4 (0,0,1,0)
Color 2	Used to control the base color of shader dynamically	Float4 (1,0,0,0)
Color 1 Bright	Used to control the brightness of the edge color of shader dynamically	Float (1.8)
Color 2 Bright	Used to control the brightness of the base color of shader dynamically	Float (1.8)
Opacity	Addition overall opacity for the shader	Float (0.4)
Divide	Used to control the alpha portion of the base color	Float (0.06)
Contrast	Used to control the contrast of the base color	Float (1.6)

Node Graph



Shader Effect 2 - M_Plate

Overview of Effect

The M_Plate shader effect is mainly designed for pressure plates within the gameplay environment. When a pressure plate is activated, the M_Plate material will be light up. It serves as an indicator, both visually and functionally, by conveying whether the pressure plate is active or not. This effect aids players in understanding the status of pressure plates.

Effect Description

M_Plate has a colour parameter that determines the material's main colour hue. This parameter enables customization of the material's colour to different components (not in current stage of development). There's another parameter named 'z' represents the object or component's location on the Z-axis. The M_Plate material responds to changes in this parameter by dynamically adjusting its brightness. When the 'z' value is lower, indicating that the pressure plate is pressed or activated, the material becomes brighter. This visual change makes it clear to players that the pressure plate is currently in use. Conversely, when the pressure plate is released, allowing the 'z' value to gradually increase back, the material smoothly transitions back to its normal colour. This visual feedback conveys the status of the pressure plate, adding a dynamic element to the game environment.

Inspiration / Reference Images:

Pressure plate with colour changing is common in video games. Take Mario 3D world as example, Mario used the rock monster to activate the plate. Use such material will make player understand that the component is "push-able".

(Retrieved from: https://www.youtube.com/watch?v=hOB8bYcV1ik)

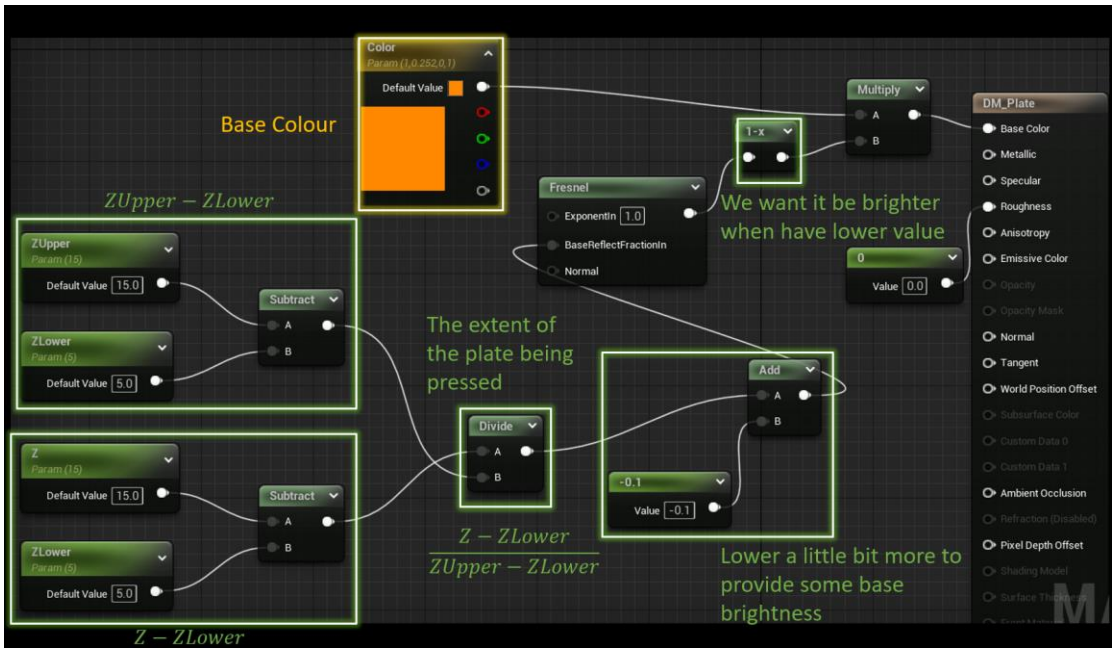
In-Engine Screenshots:

The plate has different brightness in different states.

Properties and Values

Property	Description of Purpose	Value
Color	Used to control colour of shader dynamically	Float4 (0.45, 0.12, 0.75, 1)
Z	Used to pass the z component location of the actor	Float (15.0)
ZUpper	Used to pass the maximum reachable z component location	Float (15.0)
ZLower	Used to pass the minimum reachable z component location	Float (5.0)

Node Graph



Post Processing Effects

Local Post Processing Effect – MP_Death

Overview of Effect

The MP_Death post-processing effect in Blockout Shooter will activate when a player's followed camera hits the PostProcessVolume5 on the bottom of the map. This effect can also be triggered when a player character dies, as it's controlled by a PostProcessComponent in C++. The primary purpose of this effect is to convey the player's "death" or defeat within the game environment.

Effect Description

The effect desaturates the screen, which emphasizes the dark and defeated mood following the character death. However, the colorable blocks on the player's screen remain unaffected. This intentional design choice ensures that the status of colorable blocks remains clearly visible to players during their respawn time, which allows players to strategize and make informed decisions. A diamond-shaped gradient is superimposed onto the screen, creating a vignette effect that darkens the edges of the viewport. This serves to darken the screen further.

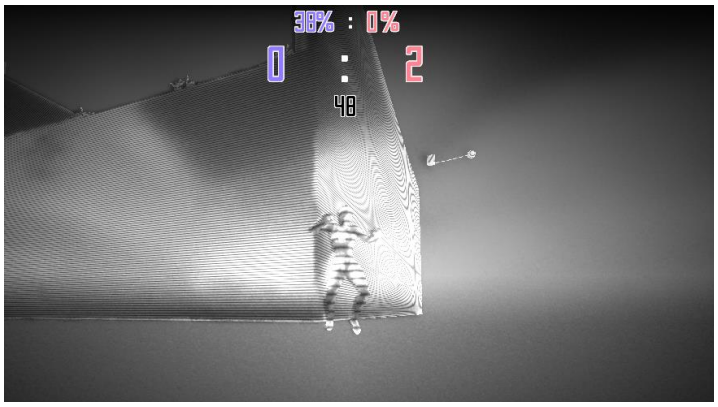
Additionally, The MP_Death effect includes an old-TV effect, achieved by using the "MF_HologramBand" material function. This component adds a touch of distortion and vintage aesthetics to the screen.

Inspiration / Reference Images:

The inspiration for the MP_Death post-processing effect is drawn from the failure screen in Grand Theft Auto V. It contains a desaturated look with a focus on the central of the screen to highlight the player is "waste".

(Retrieved from: https://www.youtube.com/watch?v=KOeJAoSyVWA)

In-Engine Screenshots:

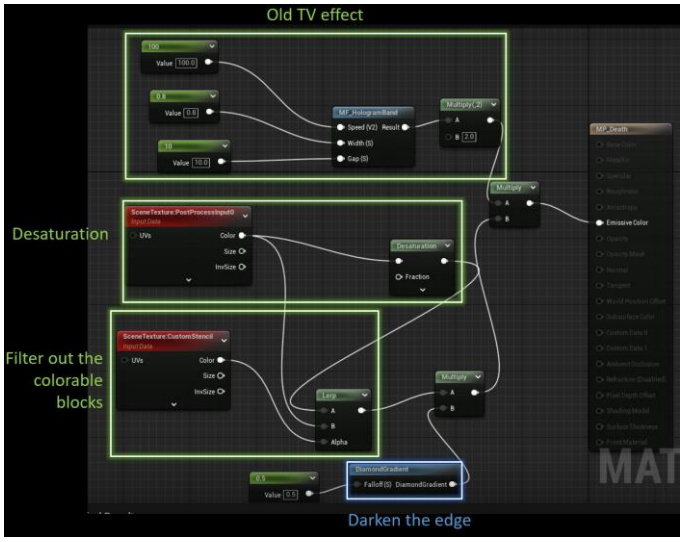


The player fell out of the world.

Properties and Values

Property	Description of Purpose	Value
MF_HologramBand	Generate continuous moving bands with width and gap.	Speed: 100 Width: 0.8 Gap: 10
DiamondGradient	Generate a diamond gradient on the screen.	Falloff: 0.5
SceneTexture:PostProcessInput0	The player screen.	PostProcessInput0
SceneTexture:CustomStencil	Render of actors on the screen that have a value of 1 in custom stencil.	CustomStencil

Node Graph



Global Post Processing Effect

Overview of Effect

The MP_Danger post-processing effect in the game is crucial in enhancing the player's sense of urgency and peril (as there's no health bar ui). It activates when a player's character's hitpoints drop below 50%, providing constant feedback about their fragile state. This effect is visible across the entire game environment, providing constant feedback to the player about their in-game condition, cueing player to get heal as soon as possible.

Effect Description

The MP_Danger effect remains active as long as the player character's hit points are below 50%. Once activated, the player's screen is immersed in a striking purple hue. This coloration serves as a strong visual cue to convey the player's precarious state. It radiates from the edges of the screen and keep breathing (expanding and contracting) over time, further reinforcing the notion of danger, and urging players to take immediate action. This purple hue emphasizes the edges of the screen while leaving the centre unaffected by the purple colour to prevent the player be distracted too much.

Additionally, A subtle heat wave distortion continues to be present, creating a visual illusion of rising temperature or stress. This effect adds to the overall sense of urgency and vulnerability.

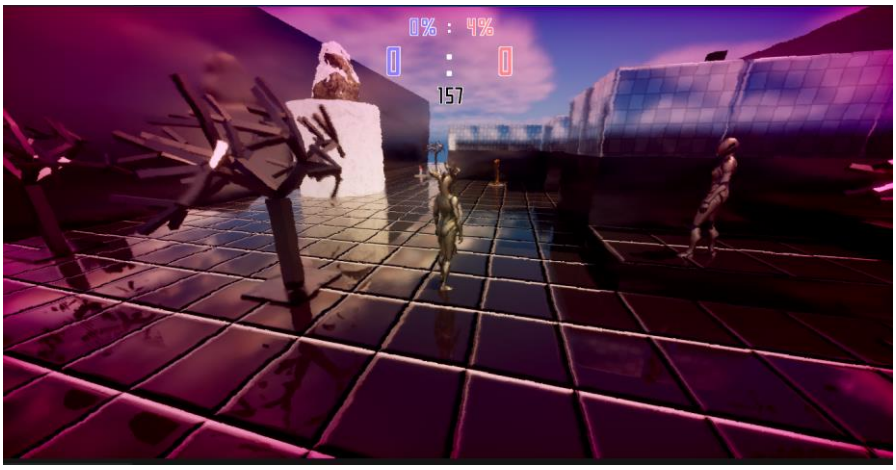
Once the character's hit points reached 50% or more (e.g., healed by green mushroom), the effect dissipates, indicating the player is temporary out of danger.

Inspiration / Reference Images:

Danger effects are very common technique in video games which use extreme colours on the edge of the screen to alert players that the characters are in danger. Such effect is too common that even the driving navigation application Amap have used it as well when the user is speeding!

(Retrieved from: also my own recordings)

In-Engine Screenshots:

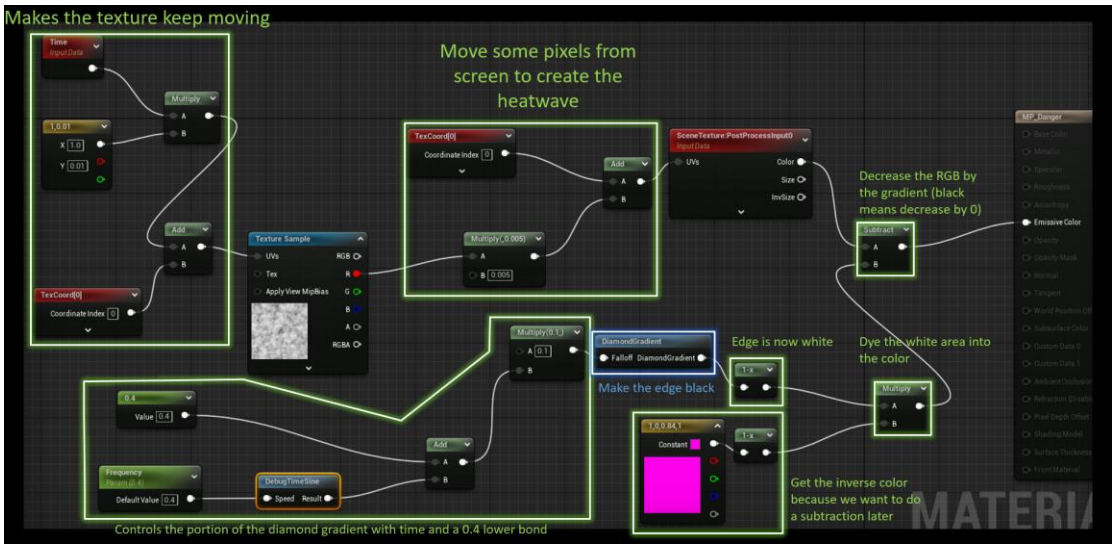


The player character hit themselves which puts they in danger.

Properties and Values

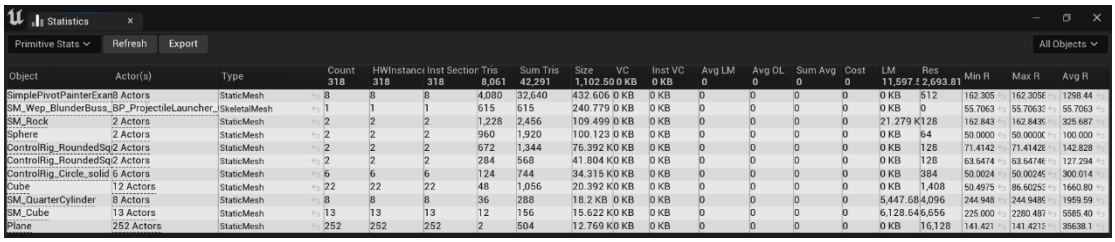
Property	Description of Purpose	Value
Time	The game time, which is a value that constantly increases.	Time
SceneTexture:PostProcessInput0	The player screen.	PostProcessInput0
Texture Sample	A picture that is randomly distributed in color which can create effects that looks like random.	T_Perlin_Noise_M
DebugTimeSine	A sine wave (sin(time)), can be used to make things grow and shrink smoothly and continuously.	Frequency: 0.4
Colour	The colour that appear on the edge of the screen.	Float4 (1,0,0.48,1)

Node Graph



Optimisation

Statistics Auditor Report



Object	Actor(s)	Type	Count	HWInstanc	Inst Sectior	Tris	Sum Tris	Size	VC	Inst VC	Avg LM	Avg OL	Sum Avg	Cost	LM	Res	Min R	Max R	Avg R
			318	318	318	8,061	42,291	1,102.50	0 KB	0 KB	0	0	0	0	11,597.5	2,693.81			
SimplePivotPainterExan	8 Actors	StaticMesh	8	8	8	4,080	32,640	432.606	0 KB	0 KB	0	0	0	0	0 KB	512	162.305	162.3058	1298.44
SM_Wep_BlunderBuss_	BP_ProjectileLauncher_	SkeletalMesh	1	1	1	615	615	240.779	0 KB	0 KB	0	0	0	0	0 KB	0	55.7063	55.7063	55.7063
SM_Rock	2 Actors	StaticMesh	2	2	2	1,228	2,456	109.499	0 KB	0 KB	0	0	0	0	21.279 K	128	162.843	162.843	325.687
Sphere	2 Actors	StaticMesh	2	2	2	960	1,920	100.123	0 KB	0 KB	0	0	0	0	0 KB	64	50.0000	50.0000	100.000
ControlRig_RoundedSq	2 Actors	StaticMesh	2	2	2	672	1,344	76.392 K	0 KB	0 KB	0	0	0	0	0 KB	128	71.4142	71.4142	142.828
ControlRig_RoundedSq	2 Actors	StaticMesh	2	2	2	284	568	41.804 K	0 KB	0 KB	0	0	0	0	0 KB	128	63.6474	63.6474	127.294
ControlRig_Circle_solid	6 Actors	StaticMesh	6	6	6	124	744	34.315 K	0 KB	0 KB	0	0	0	0	0 KB	384	50.0024	50.0024	300.014
Cube	12 Actors	StaticMesh	22	22	22	48	1,056	20.392 K	0 KB	0 KB	0	0	0	0	0 KB	1,408	50.4975	86.6025	1660.80
SM_QuarterCylinder	8 Actors	StaticMesh	8	8	8	36	288	18.2 KB	0 KB	0 KB	0	0	0	0	5,447.68	4,096	244.948	244.948	1959.59
SM_Cube	13 Actors	StaticMesh	13	13	13	12	156	15.622 K	0 KB	0 KB	0	0	0	0	6,128.64	6,656	225.000	2280.48	5585.40
Plane	252 Actors	StaticMesh	252	252	252	2	504	12.769 K	0 KB	0 KB	0	0	0	0	0 KB	16,128	141.421	141.421	35638.1

Analysis

The colorable blocks are the main component within the game. In order to fill the ground with this actor to achieve the desire gameplay, I have used a total number of 252 planes in the game to create the colorable block actors for the ground. It is fortunate that each plane mesh contains 2 Tris, resulting in a total of 504 Tris. Given the low impact on performance, there seems to be no immediate need for optimization at this stage.

However, the table reveals that the "SimplePivotPainterExample" mesh, which is used as the "Tree" actor's mesh in the game, has a significant number of Tris (4080). This might because that this mesh is dynamic. Since there are currently 8 trees on the map to be rendered, the sum of Tris adds up to 32,640. This suggests that optimizing or replacing the mesh for the trees might be very beneficial for performance.

Additionally, the report identifies two instances of "SM_Rock" which serve as decorations in the game world, with a combined sum of 2456 Tris. Given that these rocks are primarily decorative and do not significantly impact gameplay, it may be worthwhile to consider their removal to further optimize performance.

Solution

A potential solution is to find or create a less complex mesh for the trees while retaining the desired visual quality, which would reduce the overall Tris count and improve performance without sacrificing the visual of the game.

Since the decorative rocks do not play a critical role in gameplay, I will consider removing them to reduce the Tris count and improve overall performance.

Another viable solution is to use the level streaming / seamless loading. The game can unload these meshes when player is far away from them and load them only if they are getting closer to them.

GPU Profiler Report

The most time-consuming aspect identified in the GPU Profiler Report is the "DiffuseIndirectAndAO", which takes for 3.10ms+1.87ms of GPU processing time, followed by "PostProcessing" at 1.79ms+0.82ms and "LumenSceneLightning" at 1.33+1.22ms. These figures suggest that the game is currently relatively well-optimized.

If further optimize performance is required, I can reduce the intensity or the numebr of post-processing effects, which can potentially lower the GPU processing time; however, post-processing plays a significant role in enhancing the game's visual quality, this will decrease the quality of visual aspect in the game.

Shadow casting can be disabled, as shadows does not play a critical role in the gameplay or visuals. By doing so, the processing required for dynamic shadows will be omit, freeing up GPU resources.

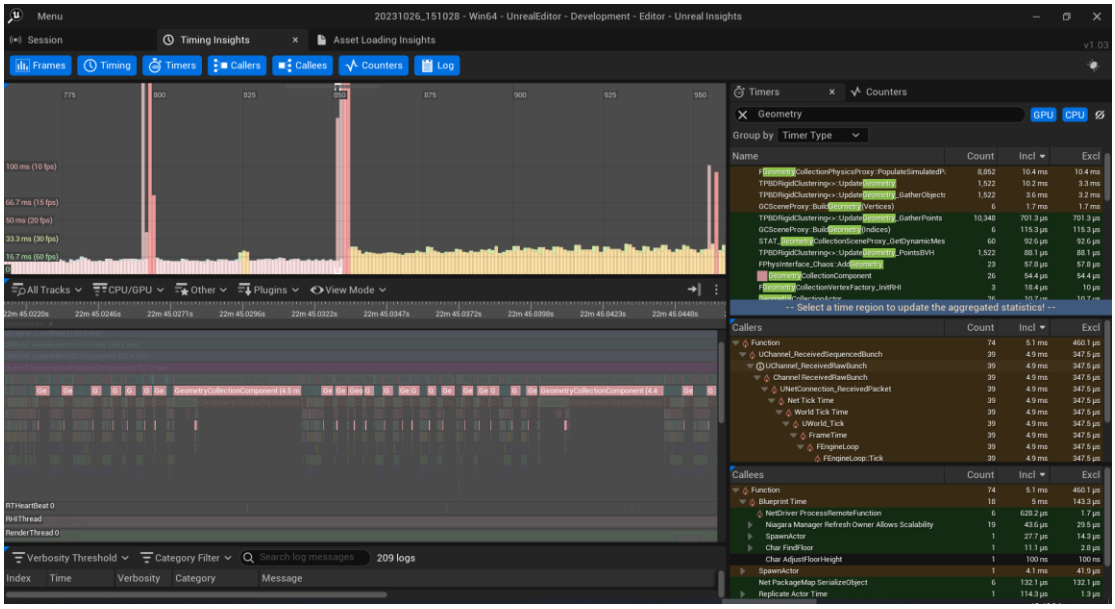
Unreal Insights Report

Timing Sections Report

In this insight analysis, the specific scenario is a player character uses a rocket launcher to interact with fragile obstacles twice and eventually suicides by using the rocket launcher near a wall. These actions trigger various in-game events and showcase performance metrics at different stages of gameplay.

The recorded insights highlight that the game's performance is not entirely stable, with significant peaks at different points in the gameplay sequence. These peaks are observed at the game's initiation, when the player character interacts with and breaks the fragile obstacles, and finally, when the game ends. This indicates potential areas where performance can be improved.

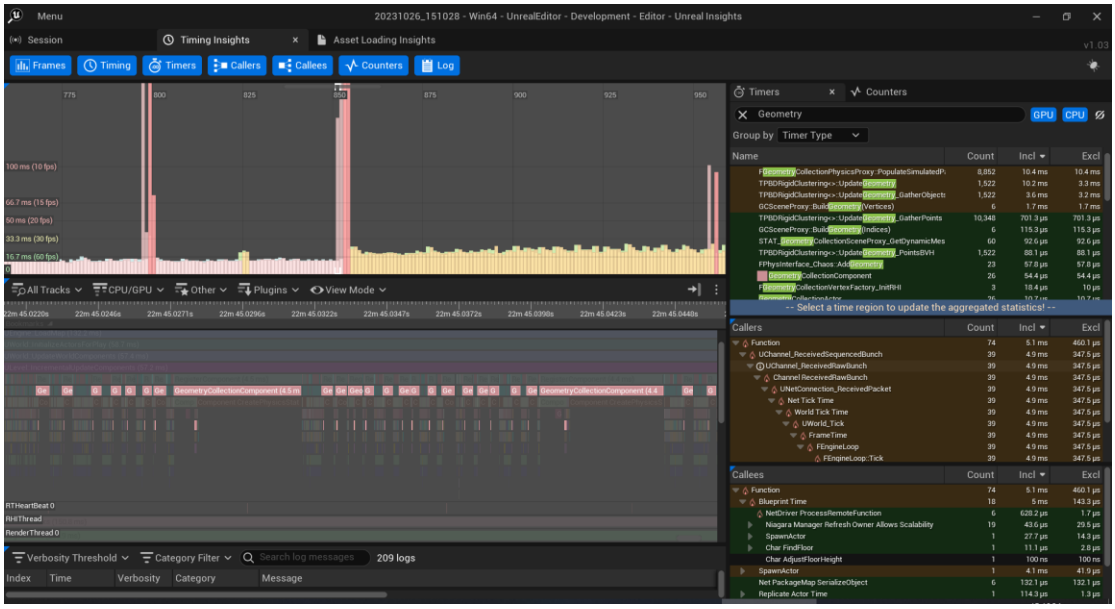
Timings Section 1



When the player character performs a rocket launcher action, there are five instances of SpawnActor, which 5ms in total. This is already efficient in current stage of development. However, it's important to note that player may be able to increase the number of rockets fired per shot after they collects blue mushrooms, the performance might degrade.

To improve these values in the future, we should consider simplifying the structure or mesh of the rocket. This optimization can help maintain stable performance even when multiple rockets are being spawned simultaneously due to power-ups. Another approach is to lower the upper limits of the multiplier attribute of the player, which limit the number of rockets spawned. By doing so, we can ensure smooth gameplay experiences for players, particularly when player take a lot of blue mushrooms. This optimization will contribute to the overall stability which will enhancing the player experience.

Timings Section 2



The timing report shows that a large number of GeometryCollectionComponent instances (count: 78) are consuming a substantial amount of processing time, approximately 18.1 milliseconds in total. The extensive computational demands are primarily due to the need to calculate the movement and physics for numerous fragments simultaneously.

A direct approach is to reconsider the placement of fragile obstacles in the game world. Currently the obstacles are placing next to each other in order to completely blocked the high platform. By not positioning them too close to each other, the computational load required to calculate the physics and movement of numerous fragments can reduce dramatically at once.

Alternatively, increasing the fragment size for the fragile obstacles geometry collection can also be an effective solution. Fewer, larger fragments will require less computational resources, enhancing performance and maintaining a smooth gameplay experience.