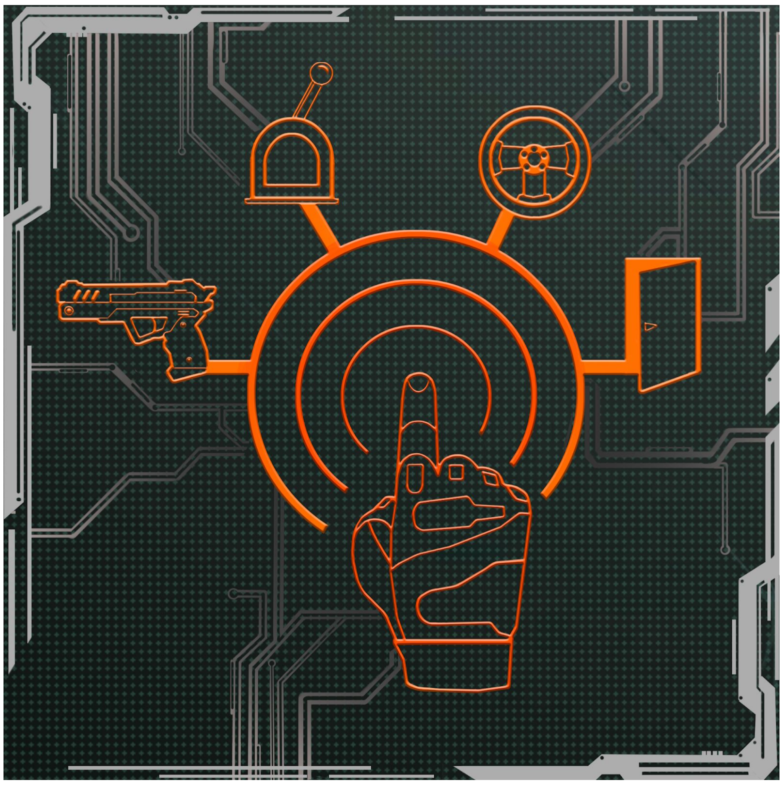
Character Interaction

Documentation

(Unreal Engine 4 asset)

https://www.unrealengine.com/marketplace/character-interaction



Video overview of the project
Video of the first update (IK, First Person Camera support, etc.)
Video of the second update (Extended system for movement)
Video of the second update (Complete revision of the project)

Documentation written by the project author ZzGERTzZ.

Manual version : 3.0
Last update : 12/07/2018

1. Introduction:

Project **Character Interaction** is created to provide developers with a template for games of different genres. The template includes a system for character movement, character interaction with different objects, use of various firearms, picking up and crafting different items, use of means of transportation, making of different enemy AI, etc.

The package includes the following necessary elements of gameplay:

Basic movement with procedural bending to utilize fewer animations while feeling the character's response to the change of direction during movement. Also, there is a complete implementation of weapon management that comprises aiming, reloading, shooting in any direction, equipping, and changing weapons. There are the jumping system, movement while crouching, movement while aiming, etc.

Interaction with objects. These can be either various switches, levers, buttons, valves or any other interactive object types limited only by the imagination of the developer. There are 10 templates for such objects in the project (it can be extended in updates). These are different buttons, furniture, levers, a climbing wall, etc. Learn more about them from the video overview of the project in the title page.

Picking up items. Picking up items is an intrinsic part of the genre. The autodetection of the character location and selection of a proper animation are implemented in the class. Therefore, all that is required is to specify the item name, its model, and text. Also picking up of custom items is possible. In the demo level, there is an example of a handgun that must be picked up with a custom animation. In the demo level "ZombieCity", there is a bulletproof vest that increases character's health or gas mask that allows to traverse rooms with gas.

Inventory. There is an inventory class in the project that allows you to create various quest items and transfer resources. It also supports different ammo types in the inventory to use with different firearms and so on.

Firearm. A fully functional weapon class includes settings to create different firearms. It includes ballistics (one may either simulate flying projectiles or use ray-tracing systems), reloading, bullet recoil and spread, physical momentums, firearm shells and clips, animations of both weapon and a character who possesses this weapon… There is a list of settings in the video. The settings were greatly extended in the 3rd update.

Storage of weapon. This class contains a system to create and store various weapon. It allows one to create a new weapon and add it into the game with two clicks.

Weapon manager. There is a weapon manager where both firearm and melee weapons available are visually displayed. You can equip a weapon on the character or disarm him, or throw the weapon away and pick it up again.

AI. Basic Artificial Intelligence is capable of hearing, seeing, following, and attacking the player with both melee weapons and firearms. There are several sample enemies like a soldier, zombie (slow), killer zombie (fast). The characters are fully animated, set up and have all necessary sounds. Also, as an example how to work with AI, each class has its unique traits, e.g the soldier class can use a helmet, the zombie class has character look randomizer, etc.

Transportation. There is a class for creating various transportation means in the projects. The class supports interaction with the character, as well as a vast amount of modules such as the decor module (car repainting with the help of aerography), nitro (boost), fuel consumption, impulse based defense module, weapon system and so on. There are two types of cars in the project: basic one and combat one. Cars are fully animated, set up and have all necessary sounds.

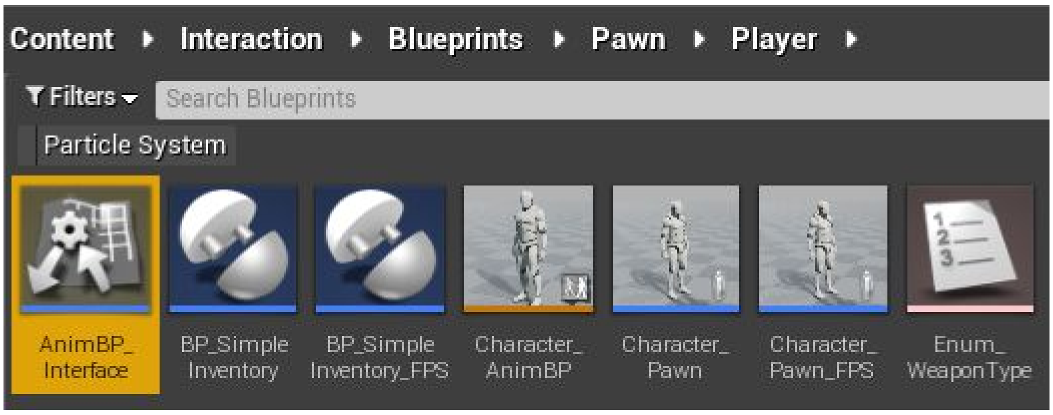
Close quarter combat. There is a basic support of close quarter combat. Functionality is implemented for the character to fight depending on presence or absence of a weapon in his hands. For instance, the character can hit with a buttstock. Also, it is possible to add various melee weapons. By default, we add a knife as an example. There is an execution system for enemies: takedowns, which includes the functionality to set up animations, conditions for use, sound support, effect, and so on.

The package also contains various sound samples, weapon models, fully physical target for shooting, basic inventory, HUD for the character, HUD for the car, character models with additional body kits (a bulletproof vest, gas mask, handgun holster, mechanical holster for a two-handed weapon), a soldier model and a helmet for him, 3 zombie characters split for randomization, 3 cyber-zombie characters split for randomization. There are demo levels for different classes including completely playable zombie town.

There are many comments within the code of the project, as well as interactive tutorials. Both First Person Camera support and Third Person Camera support are present.
I hope this starter package will be useful for game developers and it will make the initial stages of development easier.

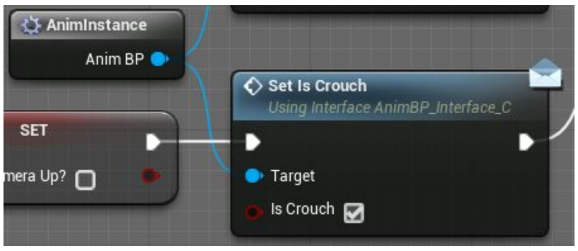
2. Basic movement:

The character movement is based on the basic class for moving from "*Epic games*" **character movement**. This is why adjust all the parameters of movement in this component. The character can move in any direction, move while crouching, do jumping, and move while aiming with a weapon in his hands, use melee, use cars and so on. The Pawn animation is passed to anim blueprint via interfaces.



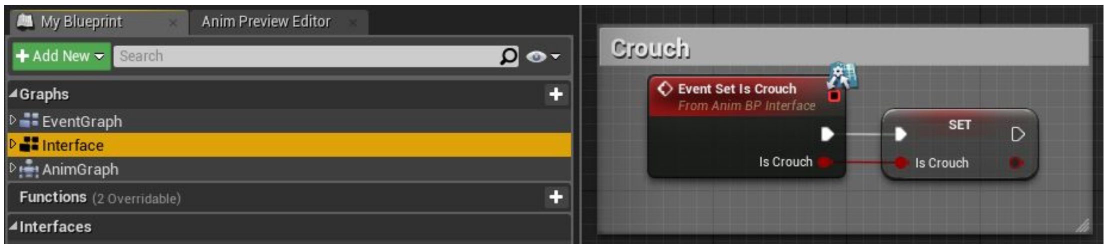
It is done in order to avoid binding the character's pawn to a particular anim blueprint. In future, this should simplify integration of other characters, which have their own skeleton.

After the integration of the character, you can use the direct reference to anim blueprint via “cast to” or by adding functions to the interface “AnimBP_Interface”.

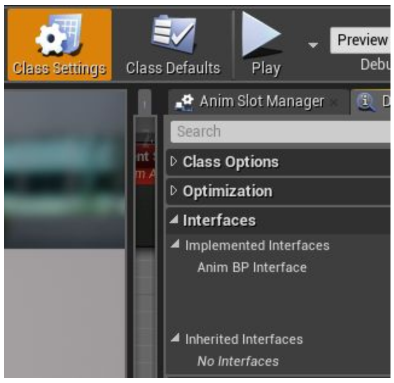


Let’s consider crouching action to illustrate this with an example. Several calculations are being executed once you press the button for crouching (can the character crouch in this moment?). If the character starts crouching, we call the interface message “Set Is crouch” (it is a function, which we have created in AnimBP_Interface beforehand) to call the necessary event in AnimBlueprint, which activates the character animation and sets it in the state of crouching.

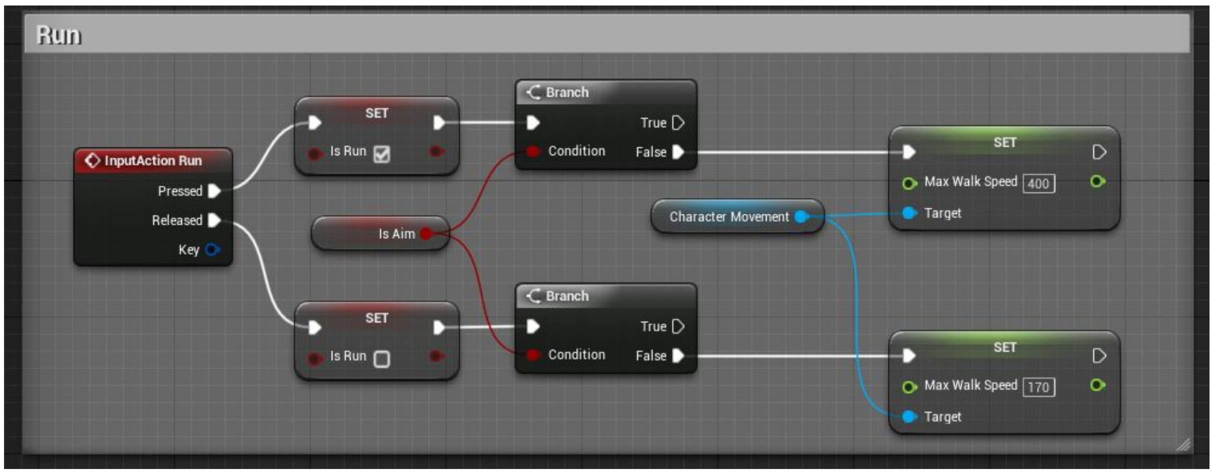
All events of interface AnimBP_Interface in anim blueprint are located here:



You must initialize the interface in “Class settings” before you can call its events.



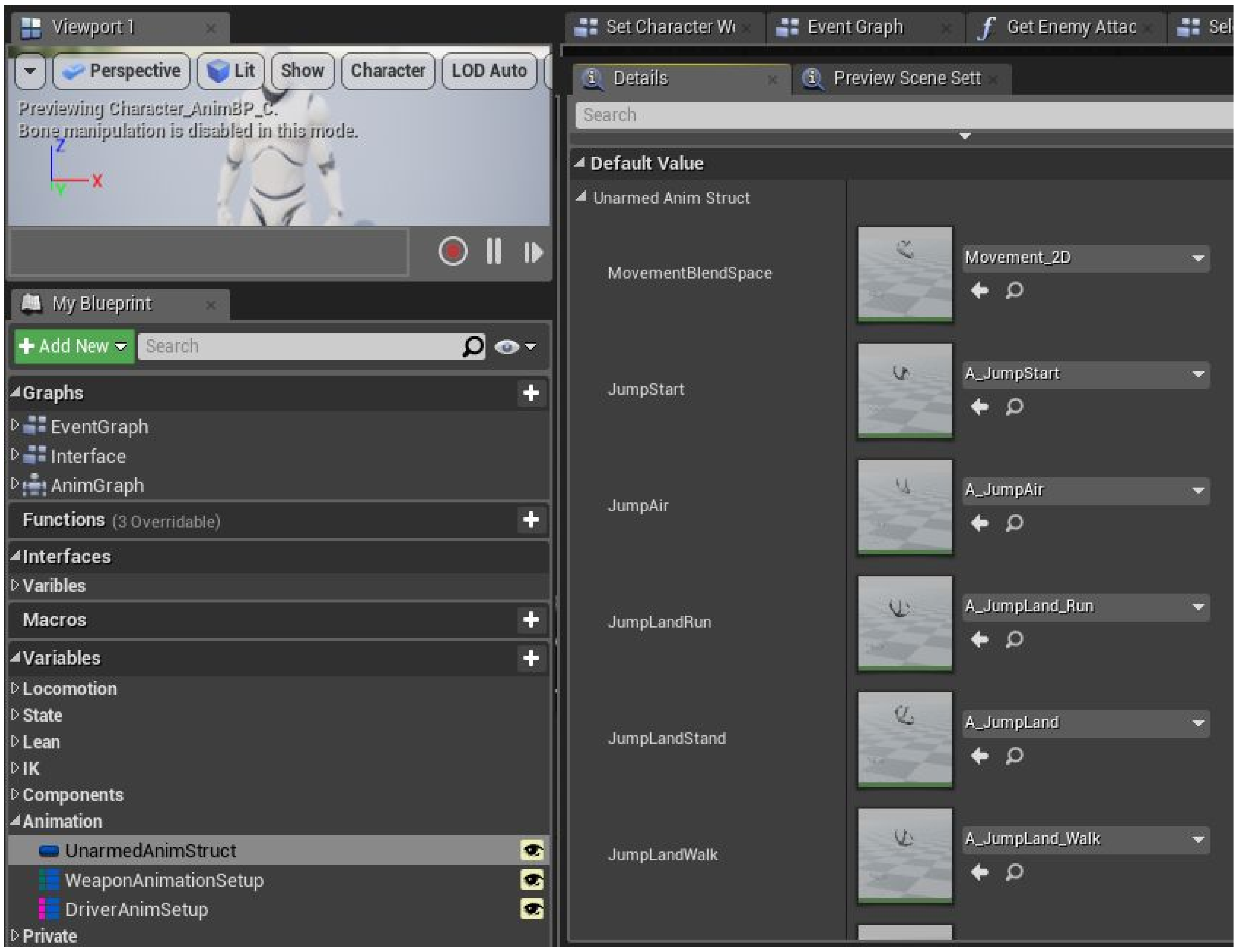
As far as the logic of character movement is concerned, it is based on dynamic changes of parameters of the class component “character movement” from “Epic Games”. For example, when running, the top speed is changed by pressing the button for acceleration (SHIFT by default).



The character animation in anim blueprint is modified based on the character speed. This is how the character animation and movement are interconnected within the package “character interaction”.

Animation settings

Structures for animation have been added into the project to add or replace animations of a character more conveniently. In order to change animation, open "**Character_AnimBP**" and find tab "**Animation**" in the list of variables.



Default Value
Unarmed Anim Struct

MovementBlendSpace	Movement_2D
JumpStart	A_JumpStart
JumpAir	A_JumpAir
JumpLandRun	A_JumpLand_Run
JumpLandStand	A_JumpLand
JumpLandWalk	A_JumpLand_Walk

By selecting different structures, you can replace the current animation of a character, add new animation types, e.g. behavior with a weapon or different transportation means.

3. Interactions with objects:

The class of interaction with objects is used for visualization of character interaction with the surroundings. The class is developed in such a way that developers can implement object interactions of different complexity. Initially, there are 10 types of objects and animations: various levers, buttons, furniture, valve, and etc. Watch the demo here video.

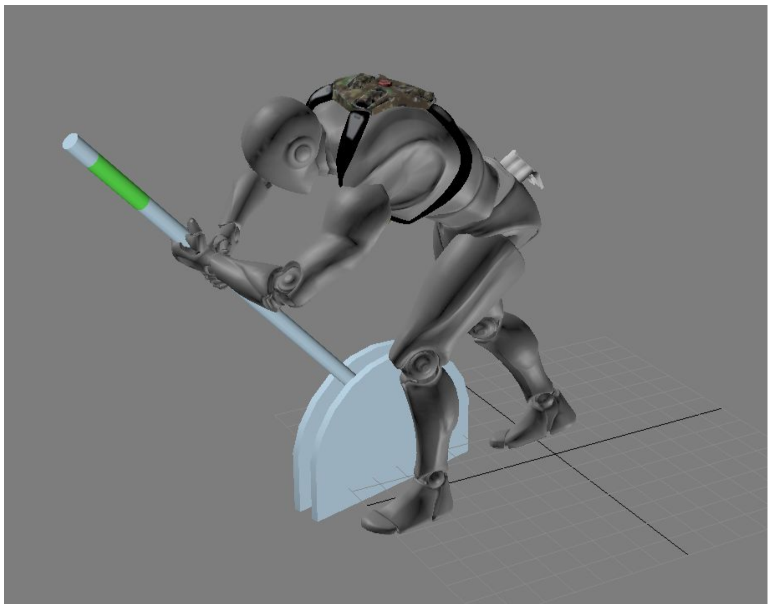
How it works:
The interaction system works according to the following scheme: there is a trigger in the actor "BP_InteractionObject" that is activated when the character enters it and presses the key for the interaction ("E" by default) to start the interaction. Once the key is pressed, a calculation is run in the Pawn to determine whether the character can interact with the object in this moment (is not he in the shooting state? is not he interacting with another object already in this moment? and so on and so forth). In case the character is allowed to interact with the object, a logic is executed in BP_InteractionObject, which chooses proper animations, smoothly adjusts the character's position, takes into account the object type… and later, based on set timer, calls an event dispatcher of the result according to which you choose different events (opening of the door, turning on electricity and so forth).

How to create a custom object for interaction:

Let’s consider the first example in the demo with a lever which rolls a retractable bridge.

***Animation**:*

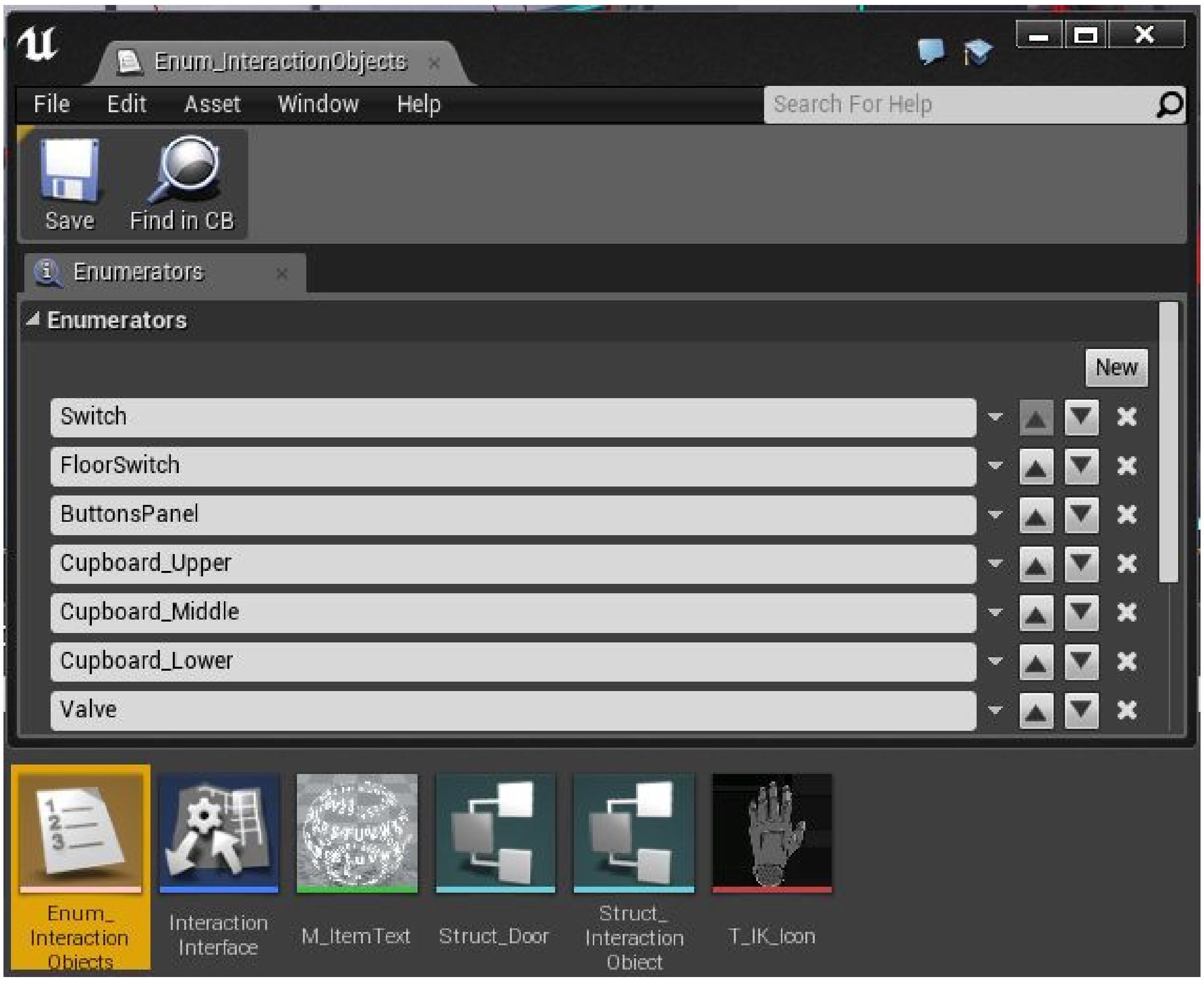
First of all, we need required animations for both the character and the object, which is the lever in this case. To automatically set the correct positions of the character and the object, make the animations relative to zero coordinates. In other words, animate the object of interaction relative to the character as shown in figure.



Adding a new object:

After creating the animation and importing it into Unreal Engine 4, let’s add our new object into the list of objects for multi use (if you intend to use the object one time only and only in one level, it is sufficient to add it into the array in BP_InteractionObject on the scene, but I would recommend to add it into the list as it is done with all objects by the default).

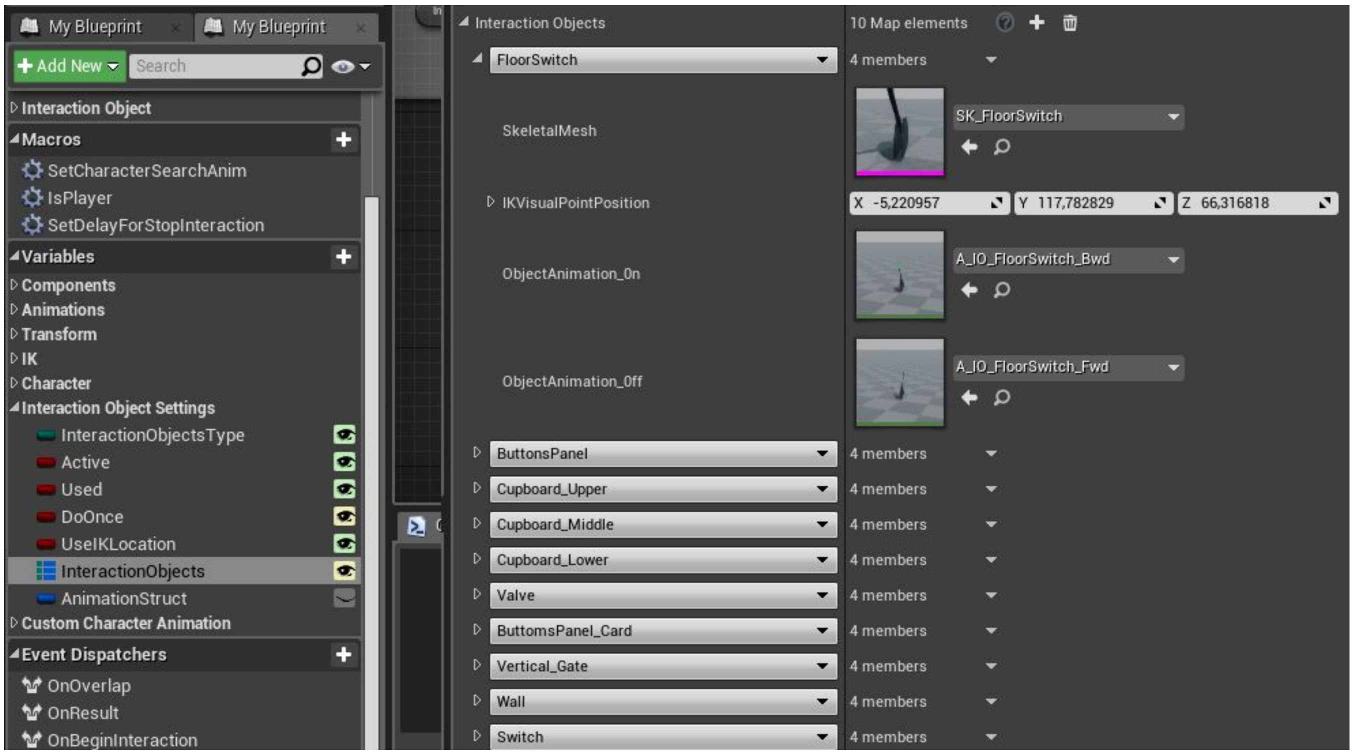
The object must be included into enum (set of objects according to which settings are set for a particular object) Enum_InteractionObjects



Switch
FloorSwitch
ButtonsPanel
Cupboard_Upper
Cupboard_Middle
Cupboard_Lower
Valve

Let's name our new object "FloorSwitch" (it can be any arbitrary name).

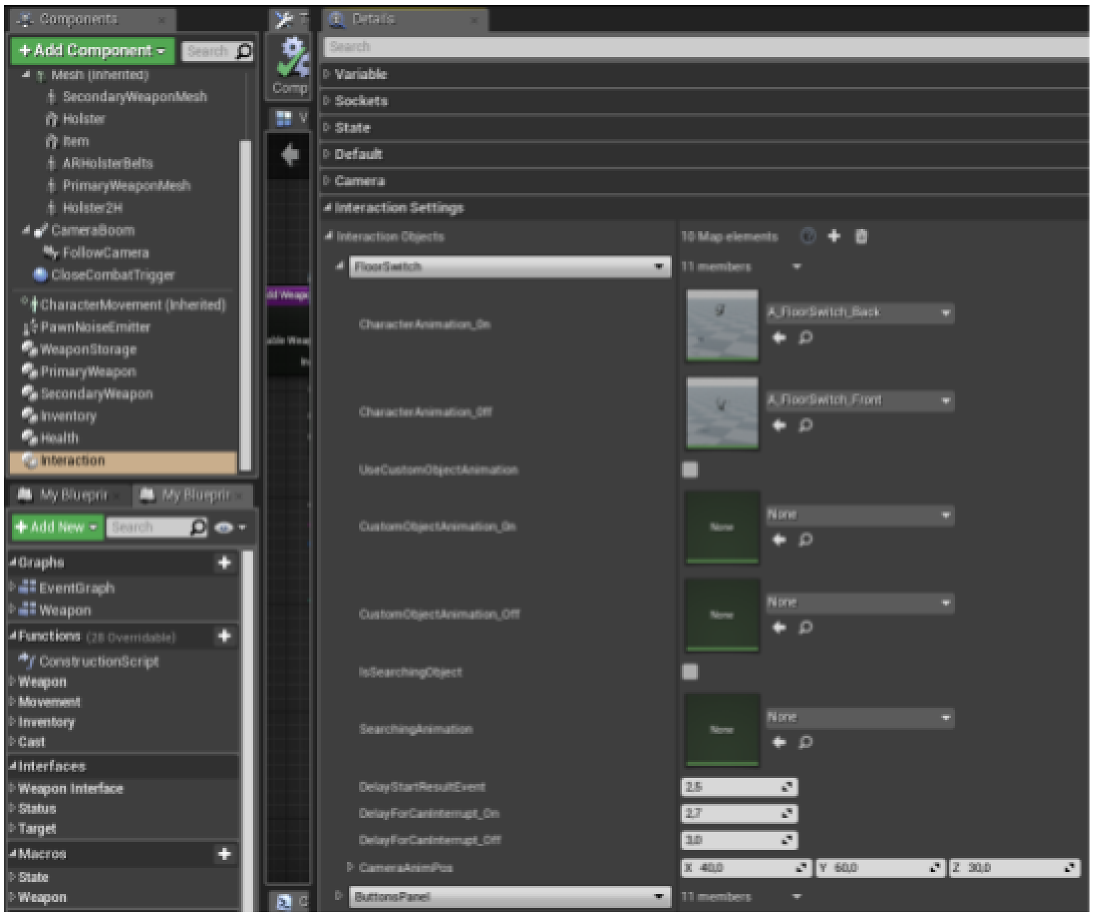
Customization of the new object:
The next step is to customize our new object. For that, open BP_InteractionObject in the content browser. Add settings for our new objects into array **Interaction Objects Array**:


My Blueprint
My Blueprint
+ Add New
Search
Interaction Object
Macros
SetCharacterSearchAnim
IsPlayer
SetDelayForStopInteraction
Variables
Components
Animations
Transform
IK
Character
Interaction Object Settings
InteractionObjectsType
Active
Used
DoOnce
UseIKLocation
InteractionObjects
AnimationStruct
Custom Character Animation
Event Dispatchers
OnOverlap
OnResult
OnBeginInteraction

Interaction Objects	10 Map elements
FloorSwitch	4 members
SkeletalMesh	SK_FloorSwitch
IKVisualPointPosition	X -5,220957 Y 117,782829 Z 66,316818
ObjectAnimation_On	A_IO_FloorSwitch_Bwd
ObjectAnimation_Off	A_IO_FloorSwitch_Fwd
ButtonsPanel	4 members
Cupboard_Upper	4 members
Cupboard_Middle	4 members
Cupboard_Lower	4 members
Valve	4 members
ButtomsPanel_Card	4 members
Vertical_Gate	4 members
Wall	4 members
Switch	4 members

Description of all settings:
Skeletal_Mesh is the model of the interactive object
IKVisualPointPosition is the default location of the hand icon.
CharacterAnimation_0n is a character animation for activating the object.
ObjectAnimation_0ff is an animation for deactivating the object.

Our new object is ready. Now specify settings for interaction between the character and the object. For that, proceed to character's pawn "Character_Pawn". Here we consider the parent class, while you may have your own pawn as a child class. In this case, use it. In the character's pawn class, select component "Interaction" and find settings "Interaction Settings" on the right. Next, open the list "Interaction Objects" and, likewise, add the new object there like you did in in the IO class.



Interaction Objects — 10 Map elements

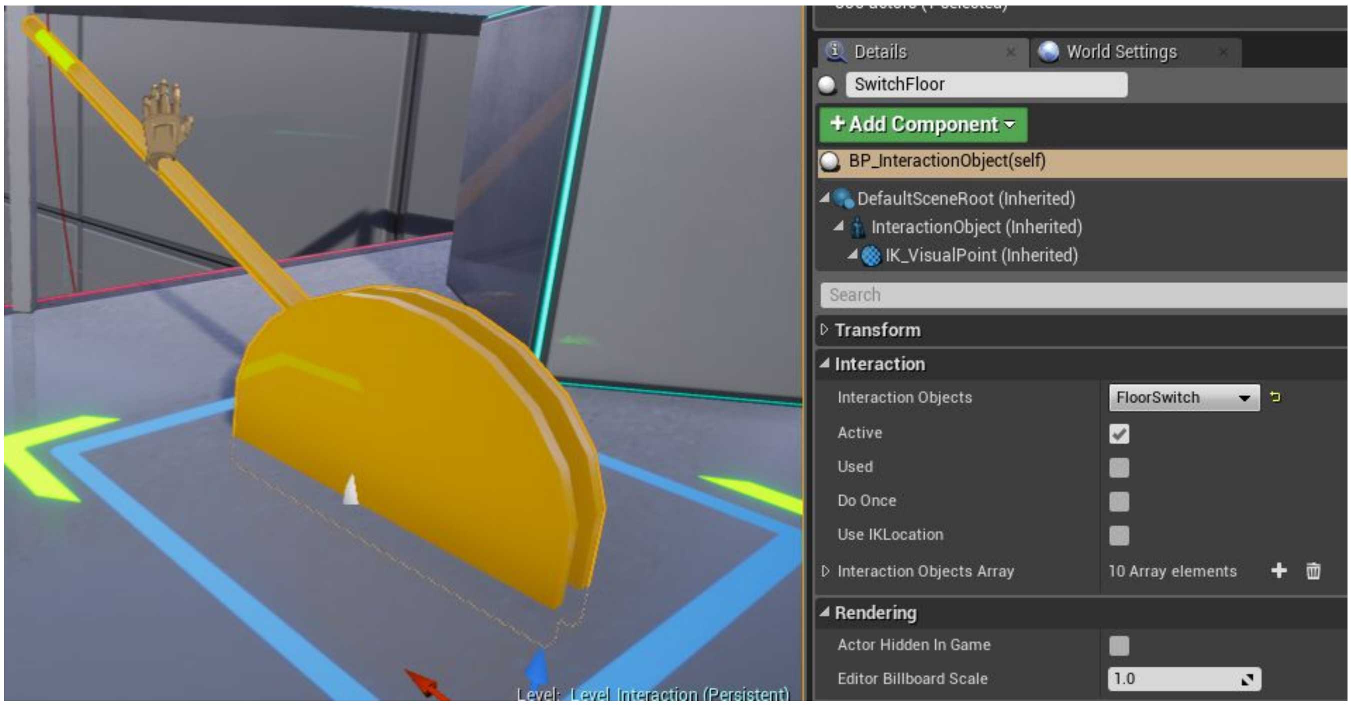
FloorSwitch — 11 members

CharacterAnimation_On	A_FloorSwitch_Back
CharacterAnimation_Off	A_FloorSwitch_Front
UseCustomObjectAnimation	☐
CustomObjectAnimation_On	None
CustomObjectAnimation_Off	None
IsSearchingObject	☐
SearchingAnimation	None
DelayStartResultEvent	2,5
DelayForCanInterrupt_On	2,7
DelayForCanInterrupt_Off	3,0
CameraAnimPos	X 40,0 Y 60,0 Z 30,0

ButtonsPanel — 11 members

Description of all settings:
CharacterAnimation_0n is a character animation for activating the object.
CharacterAnimation_0ff is a character animation for deactivating the object.
UseCustomObjectAnimation. Whether to use specified below animation for the object. It can be useful when, for example, your pawn is not a human character and it has its own animations for interaction with the object. Another case is when a human character has a different way to interact with current object.
CustomObjectAnimation_0n is a new animation for deactivating the object (it is used only when the parameter *UseCustomObjectAnimation* is on).
CustomObjectAnimation_0ff is a new animation for activating the object (it is used only when the parameter *UseCustomObjectAnimation* is on).
IsSearchingObject. Whether the object can be rummaged through (not needed for the leverage).
SearchingAnimation is an animation which the character is using after activating the object (used during the search). Not needed in this case.
DelayStartResultEvent is a delay after which the dispatcher call is triggered (which triggers the animation of the bridge in its turn) at 2.5 seconds after the beginning of the animation.
DelayForCanInterrupt_On is a delay after which the character can interrupt the interaction with the object when activating it.
DelayForCanInterrupt_Off is a delay after which the character can interrupt the interaction with the object when deactivating it.
IKVisualPointPosition is the default location of the hand icon.
CameraAnimVector is the location of the character camera during the interaction.
This is all, our new object has been added together with all the settings for interaction with it.

Application of object in the scene:
The first thing is to place **BP_InteractionObject** in the scene by dragging it from the content browser into the scene window. Choose our new object “FloorSwitch” from tab interaction in actor properties.



SwitchFloor

BP_InteractionObject(self)

DefaultSceneRoot (Inherited)
InteractionObject (Inherited)
IK_VisualPoint (Inherited)

Transform

Interaction

Interaction Objects	FloorSwitch
Active	✓
Used	
Do Once	
Use IKLocation	
Interaction Objects Array	10 Array elements

Rendering

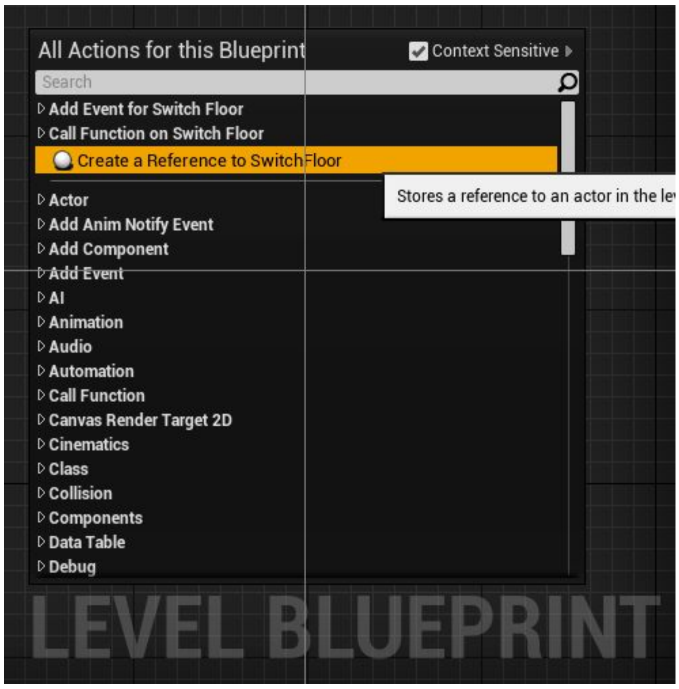
Actor Hidden In Game	
Editor Billboard Scale	1.0

Properties of Interaction*:*
Active. Whether the object is active and whether the character can use it in this moment. The property can be changed dynamically, for instance, when a certain action is required for the object activation (search for the keyword or access card in the example from the demo level).
Used. Whether the object is active in the beginning. It switches on/off the used property of the object.
Do once. Single usage of the object.
Use IKLocation. Use IK placements of hands on the objects.
Interaction Objects Array contains an array of default settings which you choose for objects, which are assigned to the current object in the current level only.

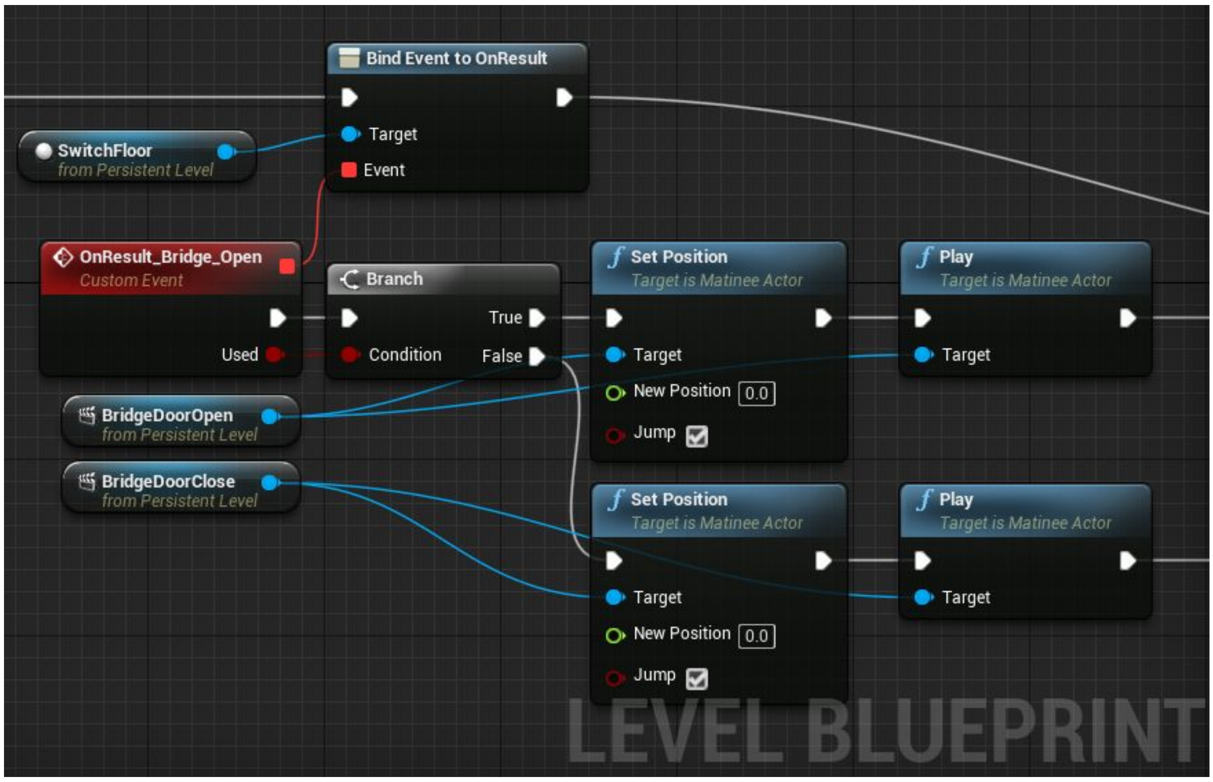
Usage of object in the scene:

The next step is the binding of the bridge animation (or any other of your actions) to the interaction with the lever.

Select the object in the scene and go to the level blueprint. There, add a reference to our object in the scene by clicking the right mouse button.



Then bind the required event to the dispatcher (in this case, animation of the bridge)



That is all. Now it is possible to control the bridge with the lever.

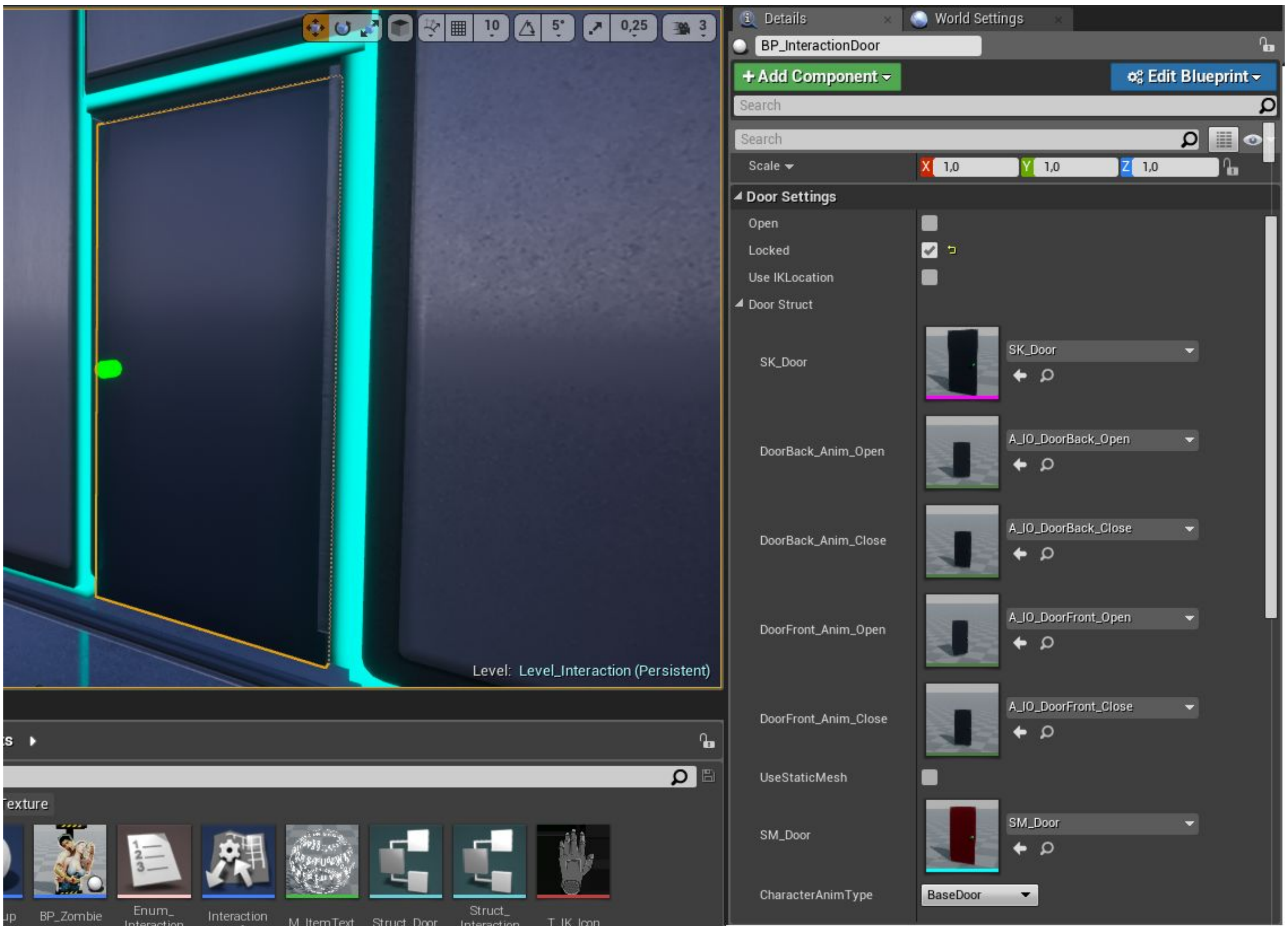
1. Doors :

The door class is intended for quick and convenient placement of doors in your level. The door automatically determines on which side the character is and activates required logic.

How it works:
The interaction system with doors works according to the following scheme: in actor “BP_InteractionDoor” there are triggers, so when the player intersects them, he can start the interaction with the door by pressing the interaction key (“E” by default). Once the key is pressed, the calculations are run in the Pawn to determine whether the character is allowed to interact with the door in this moment (is not he shooting? is not he interacting with the object already in the moment? and so forth). In case the character is allowed to interact with the door, BP_InteractionDoor executes a logic, which sets required animation, smoothly adjusts the character locations, takes into account on which side relative to the door the character is… and later, the character opens the door (if it is not locked).

How to place and customize the door:

First step is to place **BP_InteractionDoor** into the scene by dragging it from the content browser into the scene window. Let’s discuss the door properties in tab Door Setting of the actor.



BP_InteractionDoor

Scale X 1,0 Y 1,0 Z 1,0

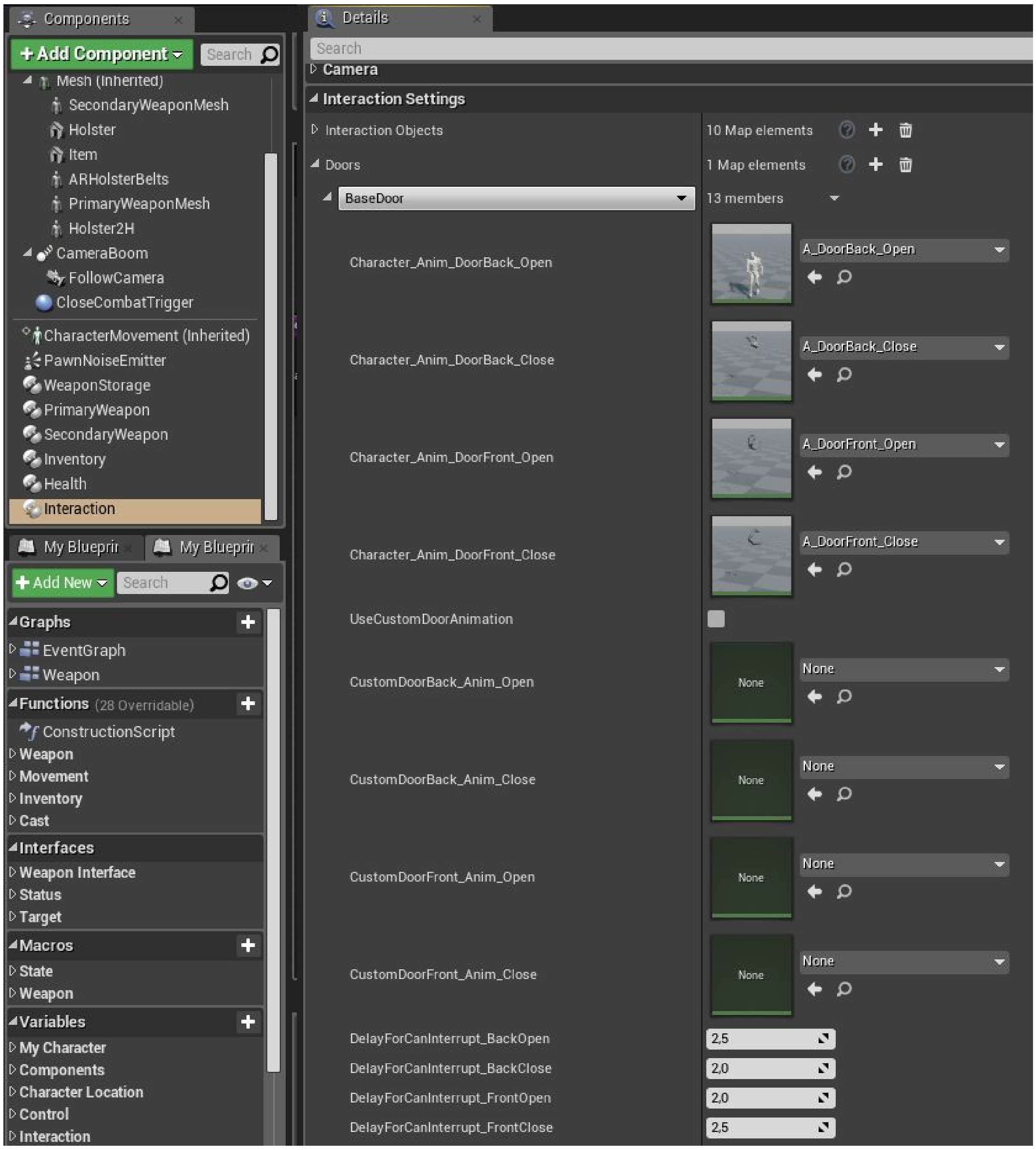
Door Settings

Open	☐
Locked	✓
Use IKLocation	☐
Door Struct	
SK_Door	SK_Door
DoorBack_Anim_Open	A_IO_DoorBack_Open
DoorBack_Anim_Close	A_IO_DoorBack_Close
DoorFront_Anim_Open	A_IO_DoorFront_Open
DoorFront_Anim_Close	A_IO_DoorFront_Close
UseStaticMesh	☐
SM_Door	SM_Door
CharacterAnimType	BaseDoor

***Properties of Door Setting**:*
Open. Whether the door is open initially.
Locked. Whether the door is locked (during the interaction, instead of door opening, the dispatcher *OnDoorLocked* will be called. You can implement the logic to open the door with this dispatcher. In the demo level, the logic is to find the key).
Use IKLocation. Whether to use IK placements for hands on the object.
Interaction Objects Array contains an array of default settings which you choose for objects, which are assigned to the current object in the current level only.

Structure DoorStruct:
SK_Door sets a skeletal mesh for the door. If you wish to use a door with a skeletal mesh, then you need to bind the door mesh to the current skeleton of the door (vertex skinning), so that default animations can be used.
Door_anim... are parameters that point to animations of the door. You can choose them at any moment. For example, in the demo level *ZombieShootingRange*, the door is kicked down and it uses different animation.
Character_anim… are parameters that point to animations of the character by the analogy with the door.
UseStaticMesh. Whether a static mesh should be used for the door.
SM_Door sets a static mesh for the door.
CharacterAnimType is a type of interaction for the character. You can set several interaction types, for instance, knock off the door or open it slowly, etc. Also, instead of making several interaction types, you can simply specify character's animation with the help of the block of settings "Custom Character Animation". This can be useful when specific interaction is needed with this particular door. The settings are also in the same door class.

Character's settings to interact with the door:
Proceed to the player's pawn "Character_Pawn". Here we consider the parent class, while you may have your own pawn as a child class. In this case, use it. In the character's pawn class, select component "Interaction" and find settings "Interaction Settings" on the right. Next, open the list "Doors", where interaction types are. By default, there is just one "BaseDoor".


Components

+ Add Component

- Mesh (Inherited)
  - SecondaryWeaponMesh
  - Holster
  - Item
  - ARHolsterBelts
  - PrimaryWeaponMesh
  - Holster2H
- CameraBoom
  - FollowCamera
- CloseCombatTrigger
- CharacterMovement (Inherited)
- PawnNoiseEmitter
- WeaponStorage
- PrimaryWeapon
- SecondaryWeapon
- Inventory
- Health
- Interaction

My Blueprint

+ Add New

- Graphs
  - EventGraph
  - Weapon
- Functions (28 Overridable)
  - ConstructionScript
  - Weapon
  - Movement
  - Inventory
  - Cast
- Interfaces
  - Weapon Interface
  - Status
  - Target
- Macros
  - State
  - Weapon
- Variables
  - My Character
  - Components
  - Character Location
  - Control
  - Interaction

Details

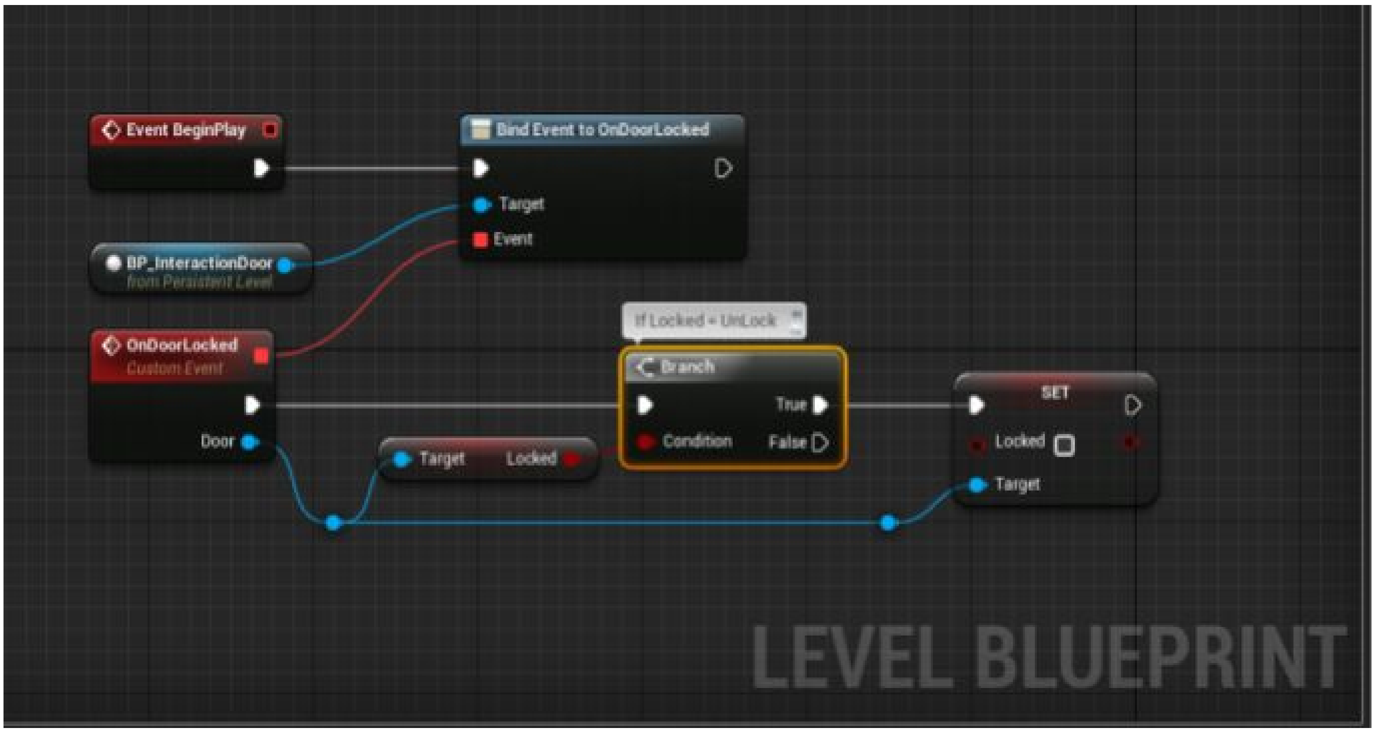
Camera

Interaction Settings

Interaction Objects	10 Map elements
Doors	1 Map elements
BaseDoor	13 members
Character_Anim_DoorBack_Open	A_DoorBack_Open
Character_Anim_DoorBack_Close	A_DoorBack_Close
Character_Anim_DoorFront_Open	A_DoorFront_Open
Character_Anim_DoorFront_Close	A_DoorFront_Close
UseCustomDoorAnimation	☐
CustomDoorBack_Anim_Open	None
CustomDoorBack_Anim_Close	None
CustomDoorFront_Anim_Open	None
CustomDoorFront_Anim_Close	None
DelayForCanInterrupt_BackOpen	2,5
DelayForCanInterrupt_BackClose	2,0
DelayForCanInterrupt_FrontOpen	2,0
DelayForCanInterrupt_FrontClose	2,5

Door_anim... are parameters that point to animations of the door.
UseCustomDoorAnimation. Whether to use specified below animation for this type of doors. It can be useful when, for example, your pawn is not a human character and it has its own animations for interaction with the door. Another case is when a human character has a different way to interact with current door.
DelayForCanInterrupt.. are time-delay after which the character animation can be interrupted.

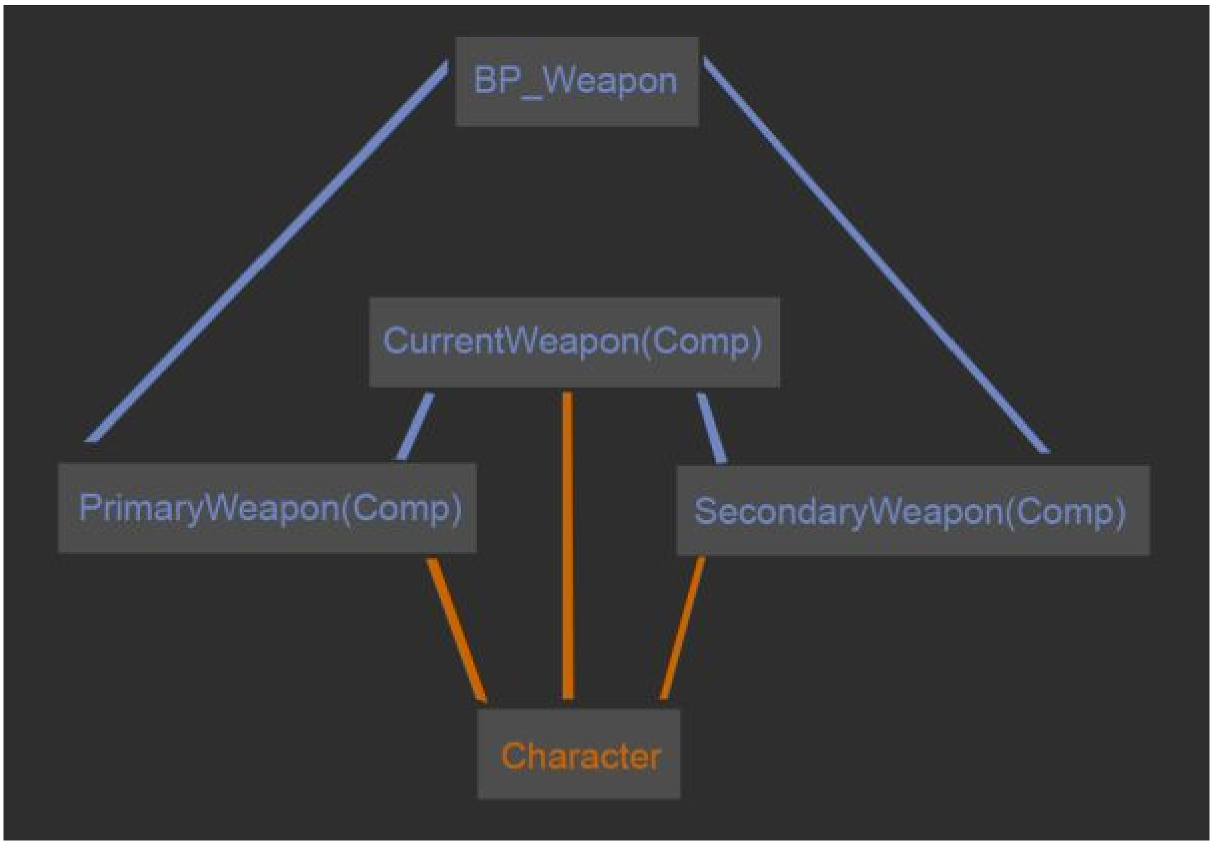
That is all, the new door is ready to be used. The door calls two event dispatchers, which you can use to add extra logic for the moment the door is being opened, *OnDoorOpen,* and if the door is being closed, *OnDoorLocked,* respectively. You can call these event dispatchers similarly to how it is described in section ***Usage of the object in the scene.***



Firearms

The weapon class in character interaction allows you to implement various firearms. The weapon class is shown as a character component. The character has two weapon components by default: primary one and secondary one.

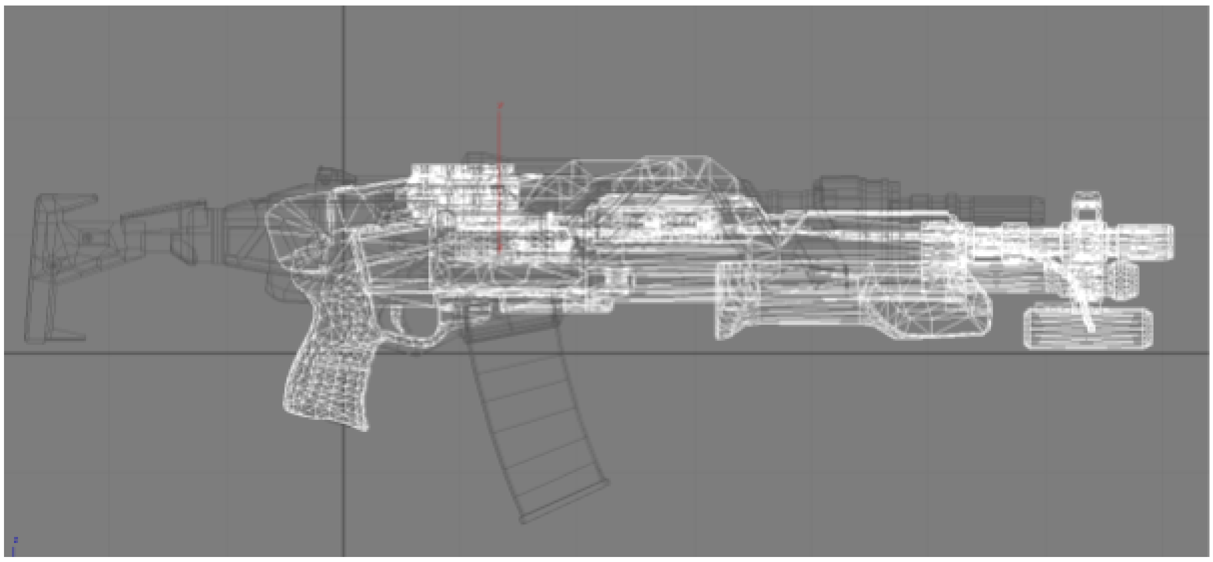
We implemented the following scheme of interaction between the weapon and the character:



In other words, the character has 3 weapon components “**BP_Weapon**”: primary, secondary, and current one.
The weapon class itself is called “**BP_Weapon**”. This class contains all logic of firearms. However for a more convenient way to create and store firearms, we have devised the class “**WeaponStorage**”. This class deals with the structures of weapon component and stores them based on indexes. By doing so, it replaces the structure of mention above components for the use of weapon. This is why to switch between weapons or to change them, we have created vidget “**Hud_WeaponManager**”, which creates a feedback between the component “**WeaponStorage**” and the pawn (character).

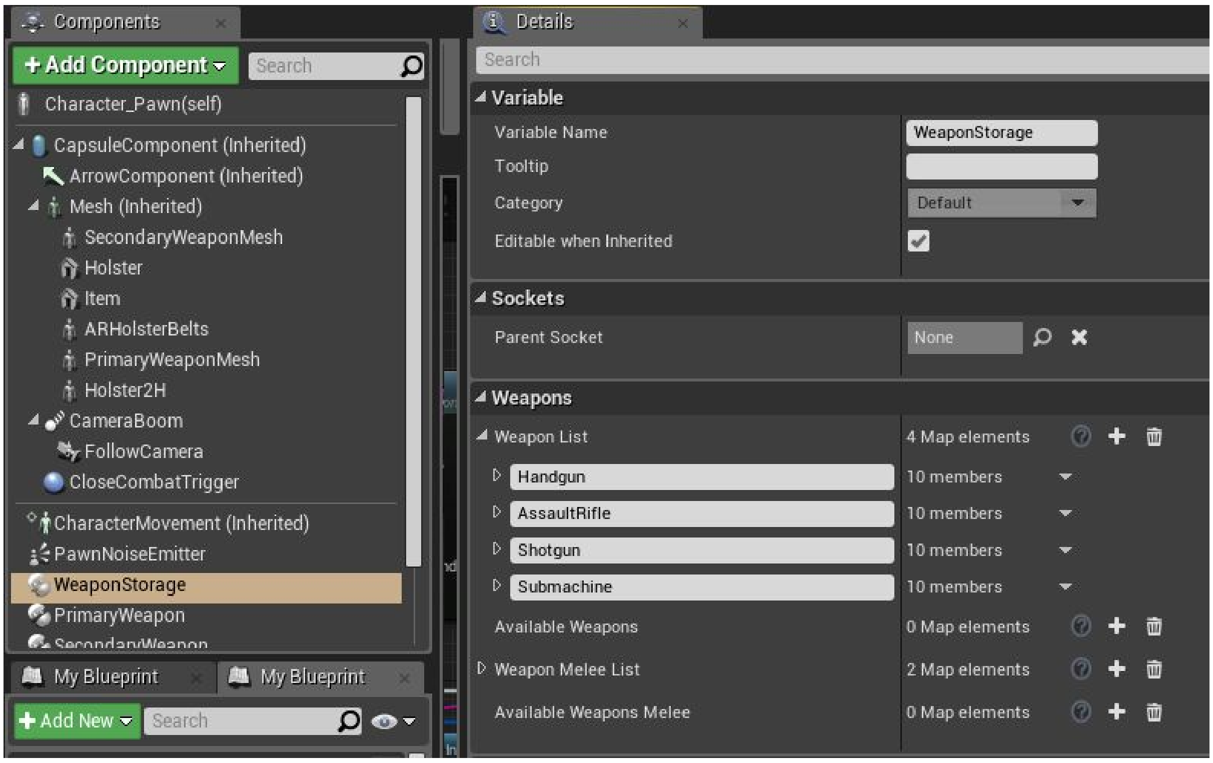
Adding a new firearm

Here we will consider how to add a new firearm in details by using shotgun as an example. The first step is to add all the content of the shotgun.
At first, create the shotgun model in such a way, that its zero coordinates are in the center of the shotgun handle as shown in figure below, so you will not need to realign them later. Create a skeleton for the weapon if it is needed (for locks, forearm, etc). If you do not use animations on the weapon, then assign the model as a skeleton mesh during import. A weapon in Character Interaction must be a skeletal mesh.


A shotgun is available in the project (update 3)

Once the weapon model is created, proceed with making necessary sockets:
Socket_Muzzle is a location to simulate the firearm muzzle.
Socket_Shell is a location of ejection of a bullet shell.

Socket_Clip is a location of clip ejection (not used in the example with the shotgun, since there is no clip).
Also, if needed, add sounds for the shooting and reloading in the project.
After all the shotgun related content is added in the project, proceed to the component "**WeaponStorage**" in order to set the weapon up and add it into the game. For that, open the pawn "**Character_Pawn**" and select the component **WeaponStorage**. On the right you will see its settings.



Variable

Variable Name	WeaponStorage
Tooltip	
Category	Default
Editable when Inherited	✓

Sockets

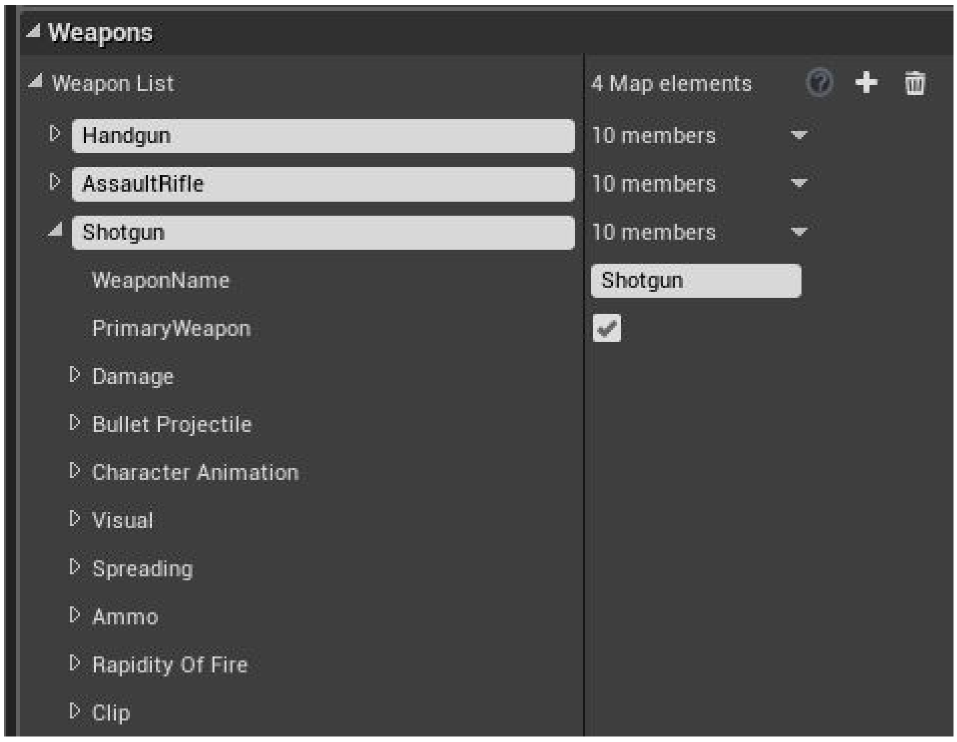
Parent Socket	None

Weapons

Weapon List	4 Map elements
Handgun	10 members
AssaultRifle	10 members
Shotgun	10 members
Submachine	10 members
Available Weapons	0 Map elements
Weapon Melee List	2 Map elements
Available Weapons Melee	0 Map elements

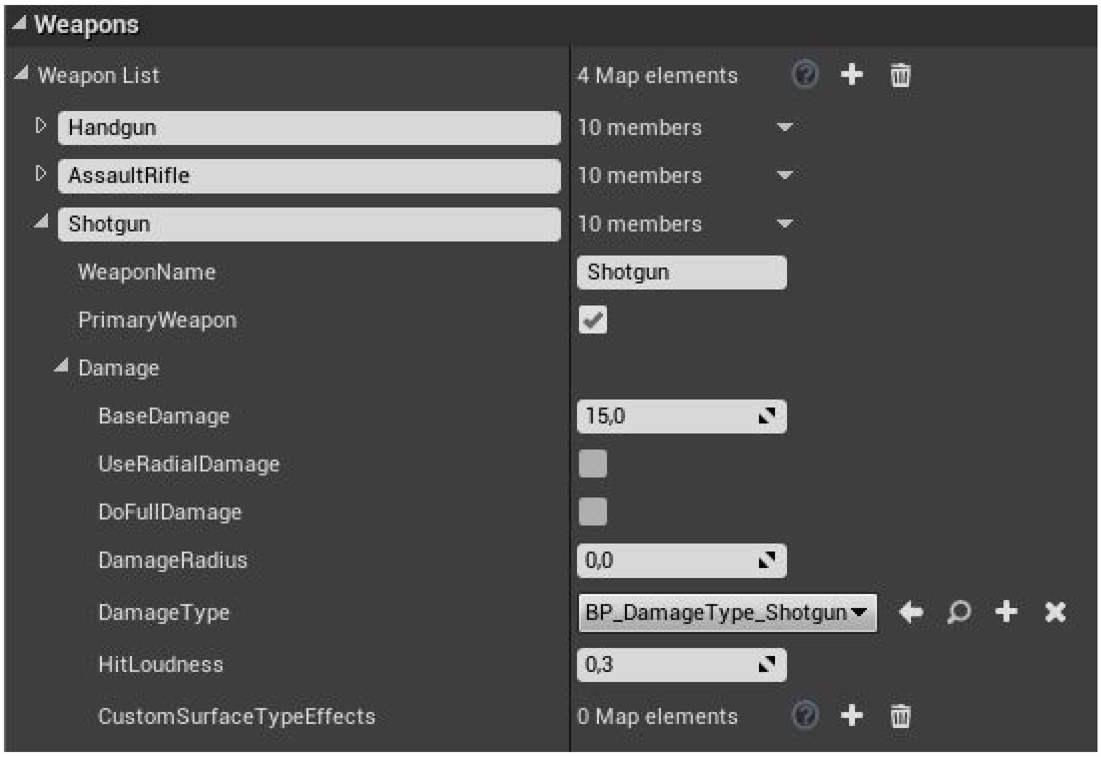
Weapon list is a list for all weapon in the project. Add here all the weapon that the current character can use. You may set its onw list for each character. If a weapon can be used by all characters, I would recommend to add it in the parent class "**Character_Pawn**". Then, in future, all child classes will know about this added weapon.

Add our new firearm shotgun via clicking on +, which is opposite WeaponList tab.
Give a name to the weapon. In this example, it is called "**shotgun**". Let's set settings of our weapon step by step.


Weapons
Weapon List — 4 Map elements
Handgun — 10 members
AssaultRifle — 10 members
Shotgun — 10 members
WeaponName — Shotgun
PrimaryWeapon — ✓
Damage
Bullet Projectile
Character Animation
Visual
Spreading
Ammo
Rapidity Of Fire
Clip

WeaponName is a technical name of the weapon. Use the same name you use to name the new index in the array. In this example, it is shotgun.
PrimaryWeapon. Whether the weapon is primary or not. This parameter determines two weapon types option a weapon can be attributed to: either primary (two-handed) or secondary (one-handed). In our case, the weapon is primary, so checkbox PrimaryWeapon.

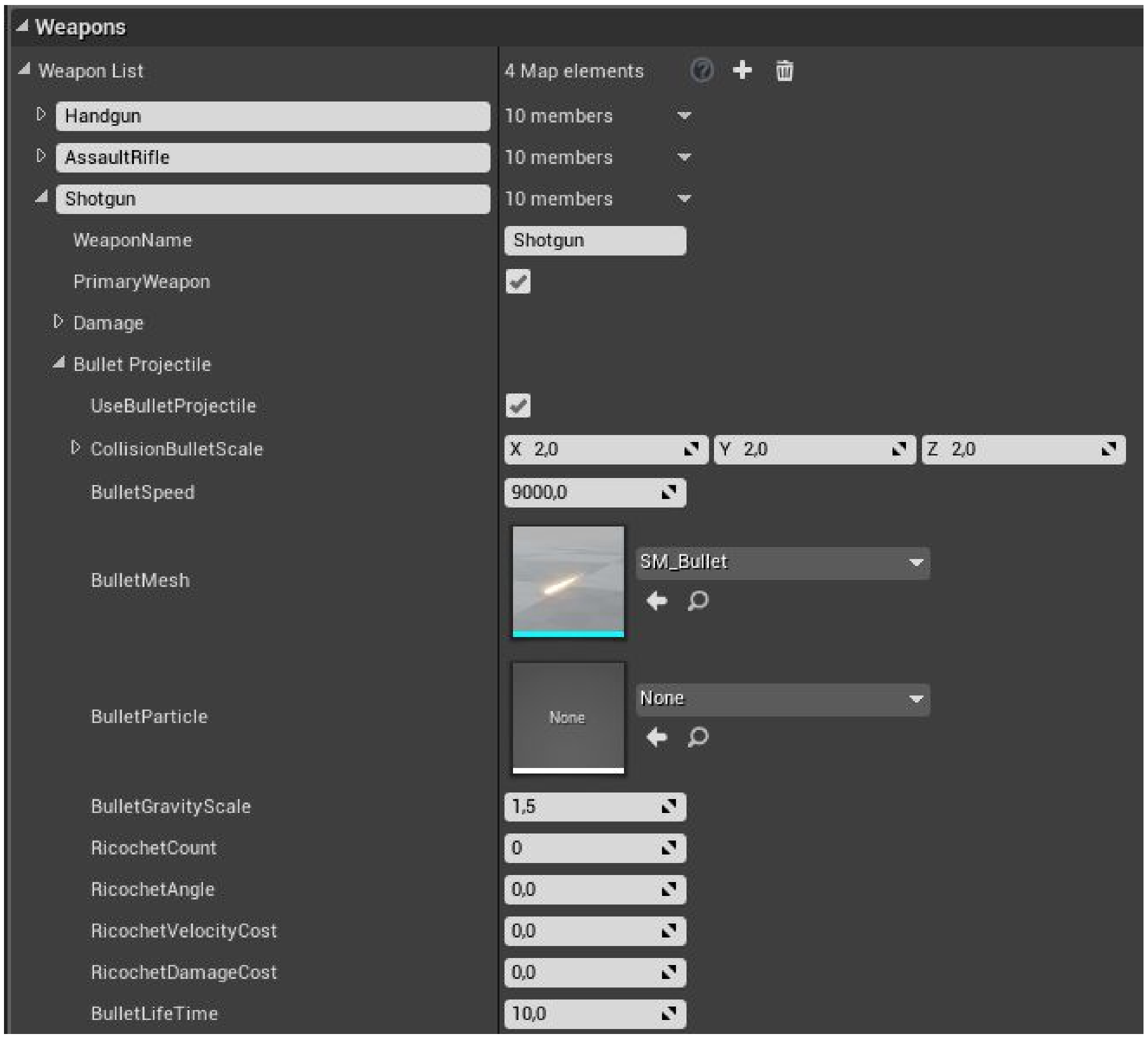
Damage settings:


Weapons
Weapon List — 4 Map elements
Handgun — 10 members
AssaultRifle — 10 members
Shotgun — 10 members
WeaponName — Shotgun
PrimaryWeapon — ✓
Damage
BaseDamage — 15,0
UseRadialDamage — ☐
DoFullDamage — ☐
DamageRadius — 0,0
DamageType — BP_DamageType_Shotgun
HitLoudness — 0,3
CustomSurfaceTypeEffects — 0 Map elements

BaseDamage is the base damage value from the weapon when it hits. Since we consider the shotgun, so multiple projectiles will fly, we key in a small damage value.
UseRadialDamage. Whether the projectiles will do area of effect type of damage. For instance, an explosion from a rocket.
DoFullDamage is related to the area damage. When the parameter is enabled, the

damage will be applied fully. When it is disabled, the inflicted damage value will drop depending on the distance the projectile hit location.
DamageRadius is related to the area damage. The value is in cm.
DamageType is an option for damage type. This class sets up impulse and damage values for destructible objects. [Click here for more details]
HitLoudness - is the loudness of the projectile hit. It does not influence the actual sound volume. It is a technical parameter for AI.
CustomSurfaceTypeEffects. Specify here additional effects upon hit or replace default effects for a particular weapon type. For example, a rocket would require an explosion, a decal for the blast crater, and a corresponding sound. Here you can set up the effect for any surface type present in the project ([SurfaceType]).

Projectile settings:

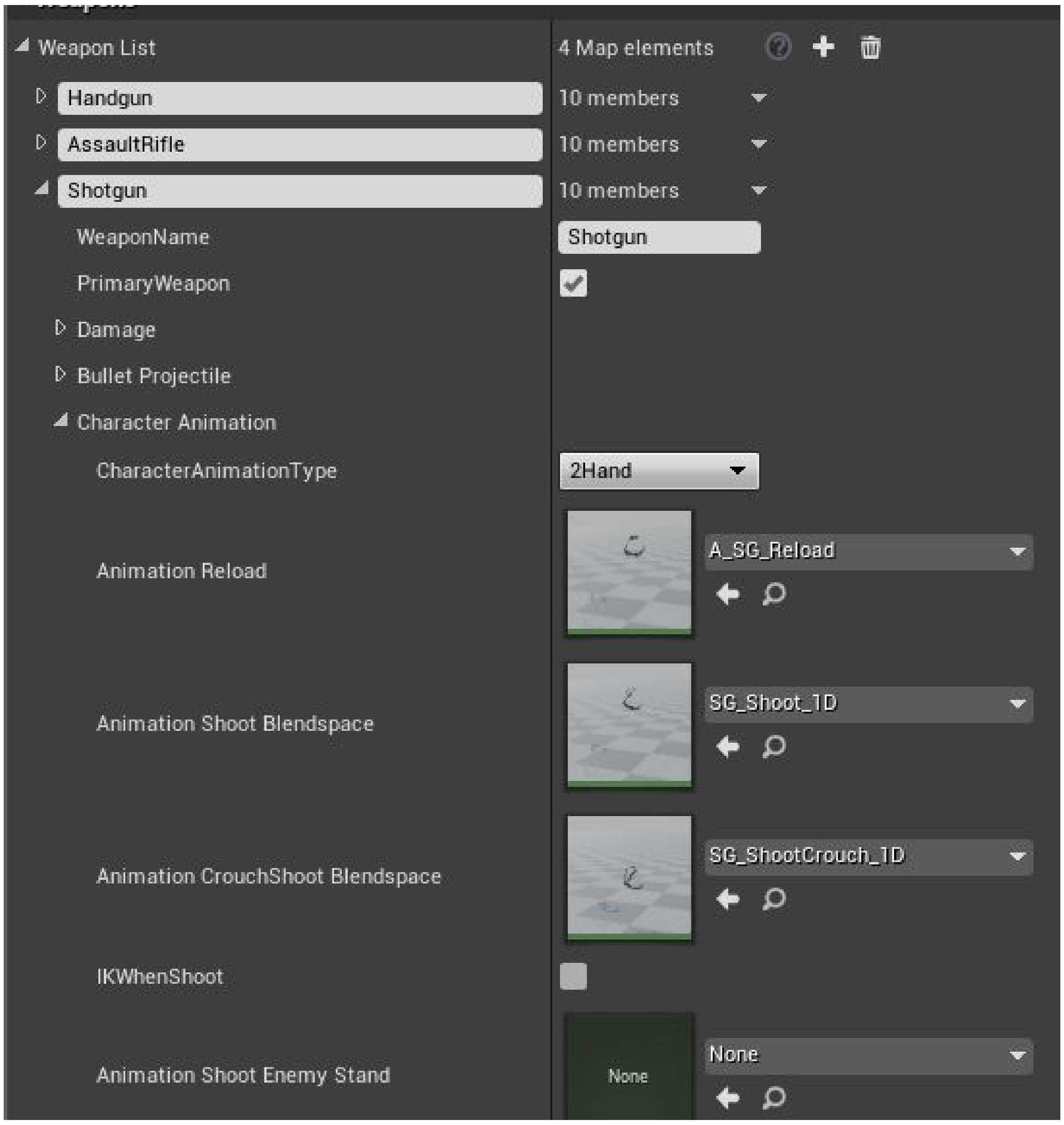


Weapons	
Weapon List	4 Map elements
Handgun	10 members
AssaultRifle	10 members
Shotgun	10 members
WeaponName	Shotgun
PrimaryWeapon	✓
Damage	
Bullet Projectile	
UseBulletProjectile	✓
CollisionBulletScale	X 2,0 Y 2,0 Z 2,0
BulletSpeed	9000,0
BulletMesh	SM_Bullet
BulletParticle	None
BulletGravityScale	1,5
RicochetCount	0
RicochetAngle	0,0
RicochetVelocityCost	0,0
RicochetDamageCost	0,0
BulletLifeTime	10,0

UseBulletProjectile is a type of the projectile. When the parameter is enabled, the projectile will have a visual representation and follow a predetermined path. When disabled, the projectile will hit the target momentarily.
CollisionBulletScale is a multiplier of the projectile collider (size of the projectile)
BulletSpeed is a velocity value of the projectile
BulletMesh is a 3d model of the projectile.
BulletGravityScale is a gravity multiplier for the projectile. The bigger the value, the fast the projectile will fall.
RicochetCount is how many type the projectile can ricochet from a surface.
RicochetAngle is an angle value at which the ricochet is possible.
RicochetVelocityCost is how many percentages of the velocity value the projectile loses

upon a ricochet. The range is between 0 and 1.
RicochetDamageCost is how many percentages of the damage value the projectile loses upon a ricochet. The range is between 0 and 1.
BulletLifeTime is a maximum lifetime of the projectile in seconds (for optimization needs).

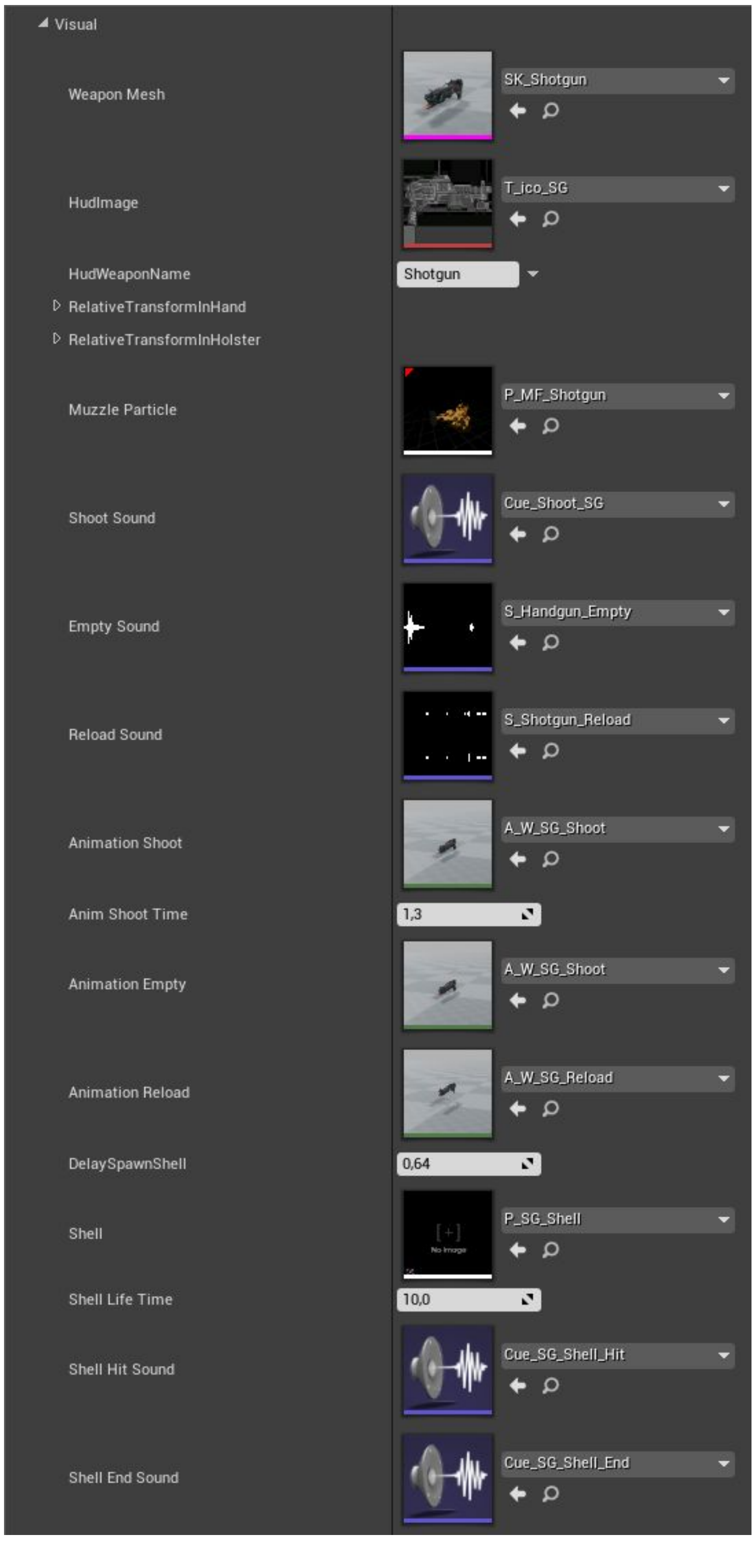
Animation settings for a character with a firearm:



Weapon List	4 Map elements
Handgun	10 members
AssaultRifle	10 members
Shotgun	10 members
WeaponName	Shotgun
PrimaryWeapon	✓
Damage	
Bullet Projectile	
Character Animation	
CharacterAnimationType	2Hand
Animation Reload	A_SG_Reload
Animation Shoot Blendspace	SG_Shoot_1D
Animation CrouchShoot Blendspace	SG_ShootCrouch_1D
IKWhenShoot	
Animation Shoot Enemy Stand	None

CharacterAnimationType is an animation type for the character with the current weapon. 2H is when the character uses the firearm as a two-handed weapon, 1H is when the character uses the firearm as a one-handed weapon.
AnimationReload is a reload animation for the character with the current weapon.
AnimationShootBlendpace is a shooting animation for the character with the current weapon.
IKWhenShoot. Whether to use IK for the left hand when shooting. Since we have the shotgun and a movable pump, we enable this parameter.
AnimationShootEnemyStand is a shooting animation when standing for the AI.

Visual settings for firearm:

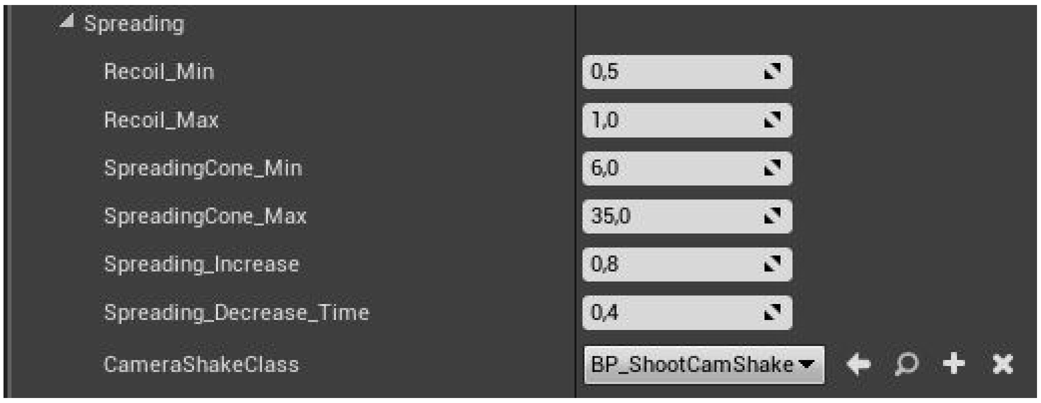


Visual	
Weapon Mesh	SK_Shotgun
HudImage	T_ico_SG
HudWeaponName	Shotgun
RelativeTransformInHand	
RelativeTransformInHolster	
Muzzle Particle	P_MF_Shotgun
Shoot Sound	Cue_Shoot_SG
Empty Sound	S_Handgun_Empty
Reload Sound	S_Shotgun_Reload
Animation Shoot	A_W_SG_Shoot
Anim Shoot Time	1,3
Animation Empty	A_W_SG_Shoot
Animation Reload	A_W_SG_Reload
DelaySpawnShell	0,64
Shell	P_SG_Shell
Shell Life Time	10,0
Shell Hit Sound	Cue_SG_Shell_Hit
Shell End Sound	Cue_SG_Shell_End

WeaponMesh is a 3d model of the firearm.
HudImage is an icon of the weapon for HUD.
HudWeaponName is a name of the weapon for HUD.
RelativeTransformHand are relative coordinates of firearm location with respect to the character's hand.
RelativeTransformHolster are relative coordinates of firearm location with respect to the weapon holster.
Muzzle Particle is the firearm firing effect.
Shoot Sound is a sound of the firearm firing.
Empty Sound is a sound of the firearm trigger clicking when the firearm is out of ammo.
Reload Sound is a sound of the firearm reloading.
Animation Shoot is an animation of the firearm firing.

Animation Empty is an animation of the firearm out of ammo.
Animation Reload is an animation of the firearm reloading.
DelaySpawnShell is a delay of the shell ejection. Since the shotgun shell does not appear instantly, we specify some delay value.
Shell is for the shell ejection effect
Shell Life Time is a life-time of the shell.
ShellHitSound is a sound of the shell hit.
ShellEndSound is a sound of the last hit of the shall.

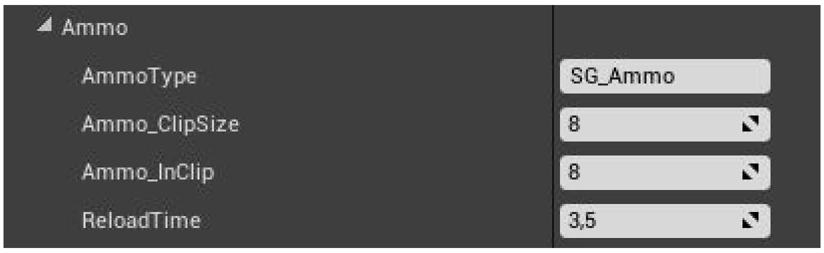
Spread settings of firearm:


Spreading

Recoil_Min	0,5
Recoil_Max	1,0
SpreadingCone_Min	6,0
SpreadingCone_Max	35,0
Spreading_Increase	0,8
Spreading_Decrease_Time	0,4
CameraShakeClass	BP_ShootCamShake

Recoil_Min is the minimum (initial at the first shot) multiplier of recoil (camera shaking).
Recoil_Max is the maximum multiplier of recoil (camera shaking). The higher is the value, the more pronounced is the recoil effect during continuous shooting.
SpreadingCone_Min is the minimum (initial at the first shot) cone of bullet spread. It determines maximum accuracy of the weapon. The lower the value, the more accurate is the weapon.
SpreadingCone_Max is the maximum cone of bullet spread. The higher is the value, the higher is the bullet spread of the weapon during continuous shooting.
Spreading_Increase sets the bullet spread increase rate with each shot fired. The higher is the value, the faster the weapon reaches the maximum bullet spread condition.
Spreading_Decrease_Time sets the bullet spread decrease rate per second. In other words, how fast the weapon restores in accuracy.
CameraShakeClass sets a class for camera shaking. You may use the default one and increase or decrease the shaking with parameters Recoil_Min and Recoil_Max.

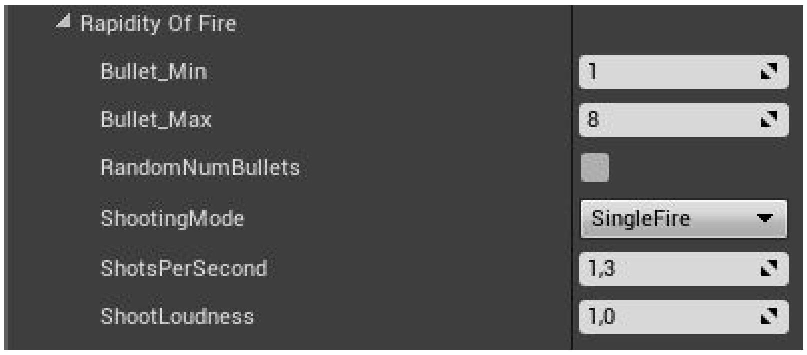
Ammo settings:


Ammo

AmmoType	SG_Ammo
Ammo_ClipSize	8
Ammo_InClip	8
ReloadTime	3,5

AmmoType is a bullet type. More details in the description below. In short, what bullet type the current weapon uses.
Ammo_ClipSize is the maximum clip size for the weapon.
Ammo_InClip sets the initial amount of bullets in the clip.
ReloadTime is the reload time (specified for the animation).

Rate of fire settings:

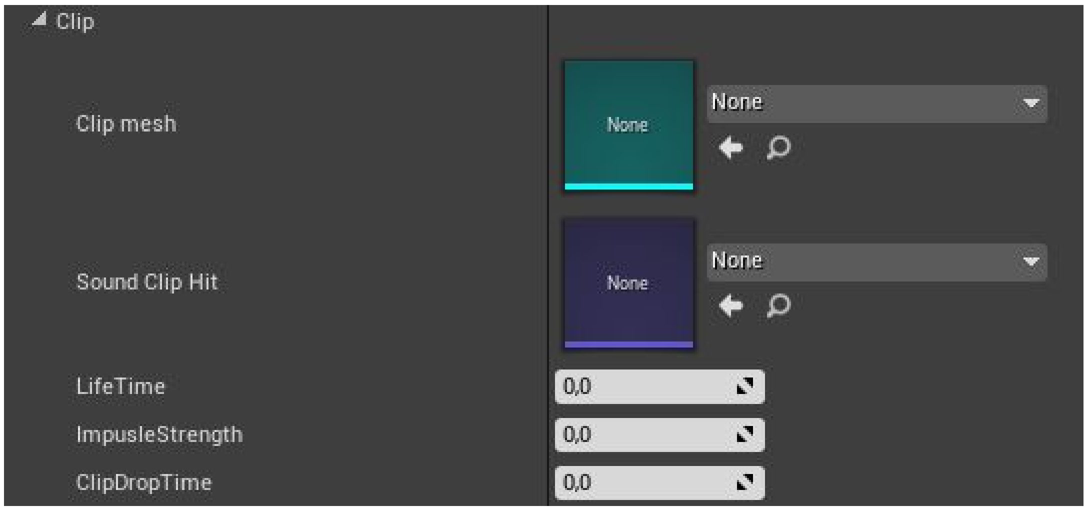


Rapidity Of Fire

Bullet_Min	1
Bullet_Max	8
RandomNumBullets	☐
ShootingMode	SingleFire
ShotsPerSecond	1,3
ShootLoudness	1,0

Bullet_Min is the minimum amount of projectiles in a shot.
Bullet_Max is the maximum amount of projectiles in a shot.
RandomNumBullets controls random number of projectiles fired from the value Bullet_Min to the set value for *Bullet_Max.*
ShootingMode sets weapon fire mode. *Single Fire* is a single round per trigger pull fire mode. *AutoFire* is automatic fire.
ShotsPerSecond is the maximum amount of shots per second.
ShootLoudness is the loudness of the gunshot. It does not influence the actual sound volume. This is a technical parameter for the AI. As an example, we added a firearm with a suppressor, when the gunshot loudness is minimum and the enemy does not react.

Clip ejection settings:

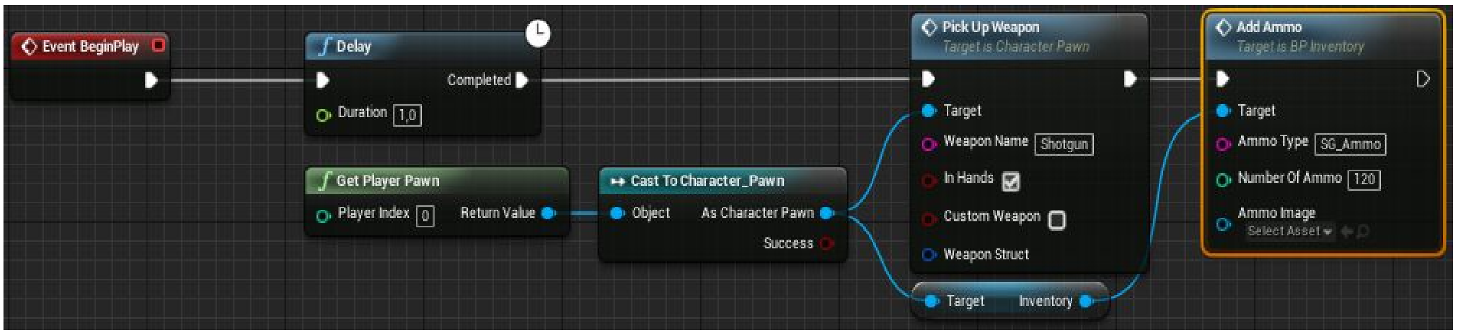


Clip

Clip mesh	None
Sound Clip Hit	None
LifeTime	0,0
ImpusleStrength	0,0
ClipDropTime	0,0

ClipMesh is a 3d model for the clip.
Sound Clip Hit is a sound of the ejected clip hit.
LifeTime is a life time of the clip.
ImpusleStrength is an impulse force driving the ejection of the clip (the direction is set with the socket Socket_Clip in the weapon).
ClipDropTime is a time-delay before the clip ejection (specified for the animation).

This is it, our weapon has been added into the game and has its own settings. In order to test the weapon by adding it to the character, use following nodes from the example for the level blueprint. Note, if you add the weapon as soon as the game starts (BeginPlay), then add a time-delay for the initialization of the system.



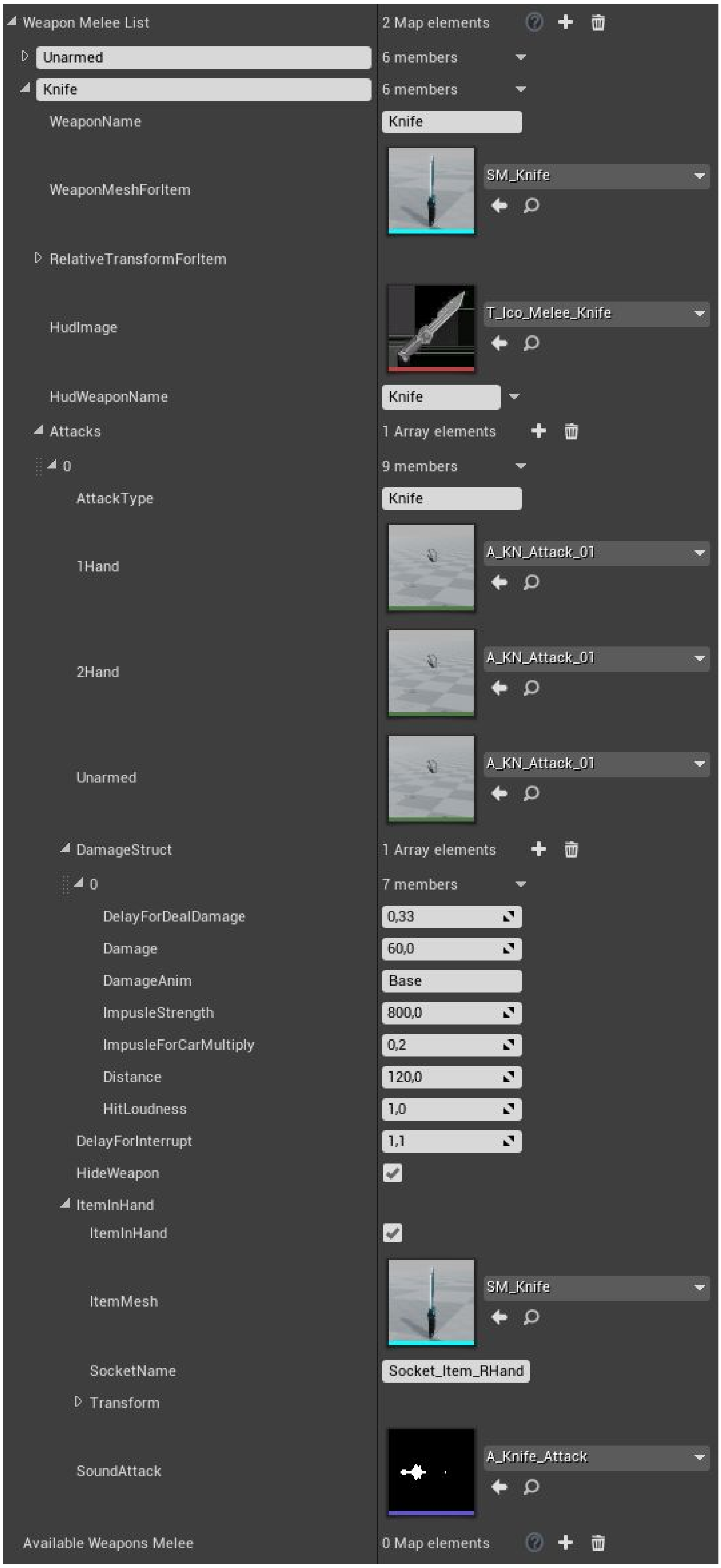
Duration 1,0
Player Index 0
Weapon Name Shotgun
In Hands ✓
Custom Weapon ☐
Ammo Type SG_Ammo
Number Of Ammo 120

A little further below, we will consider how to add a weapon as an item and how to to switch it with the help of “**WeaponManager**”.

Melee Weapon:

By default, the melee weapon is used by the character when he does not aim, which allows one to combine close quarter combat with ranged weapon. The melee weapons work just like firearms. To add a melee weapon, open the pawn “**Character_Pawn**” and select the component **WeaponStorage.** You will see its settings on the right. Proceed to the list of melee weapons “**Weapon Melee List**”. The weapon type “**Unarmed**” is the basic weapon which is added by default and used when you are unarmed with a melee weapon or when you take a weapon of the character.
To add a melee weapon, click + button which is opposite WeaponMeleeList.

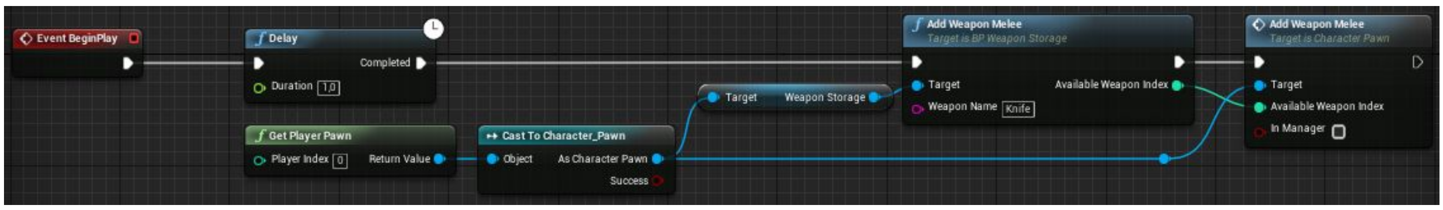
Give a name to the new weapon. Let’s consider the weapon settings:



Weapon Melee List	2 Map elements
Unarmed	6 members
Knife	6 members
WeaponName	Knife
WeaponMeshForItem	SM_Knife
RelativeTransformForItem	
HudImage	T_Ico_Melee_Knife
HudWeaponName	Knife
Attacks	1 Array elements
0	9 members
AttackType	Knife
1Hand	A_KN_Attack_01
2Hand	A_KN_Attack_01
Unarmed	A_KN_Attack_01
DamageStruct	1 Array elements
0	7 members
DelayForDealDamage	0,33
Damage	60,0
DamageAnim	Base
ImpusleStrength	800,0
ImpusleForCarMultiply	0,2
Distance	120,0
HitLoudness	1,0
DelayForInterrupt	1,1
HideWeapon	✓
ItemInHand	
ItemInHand	✓
ItemMesh	SM_Knife
SocketName	Socket_Item_RHand
Transform	
SoundAttack	A_Knife_Attack
Available Weapons Melee	0 Map elements

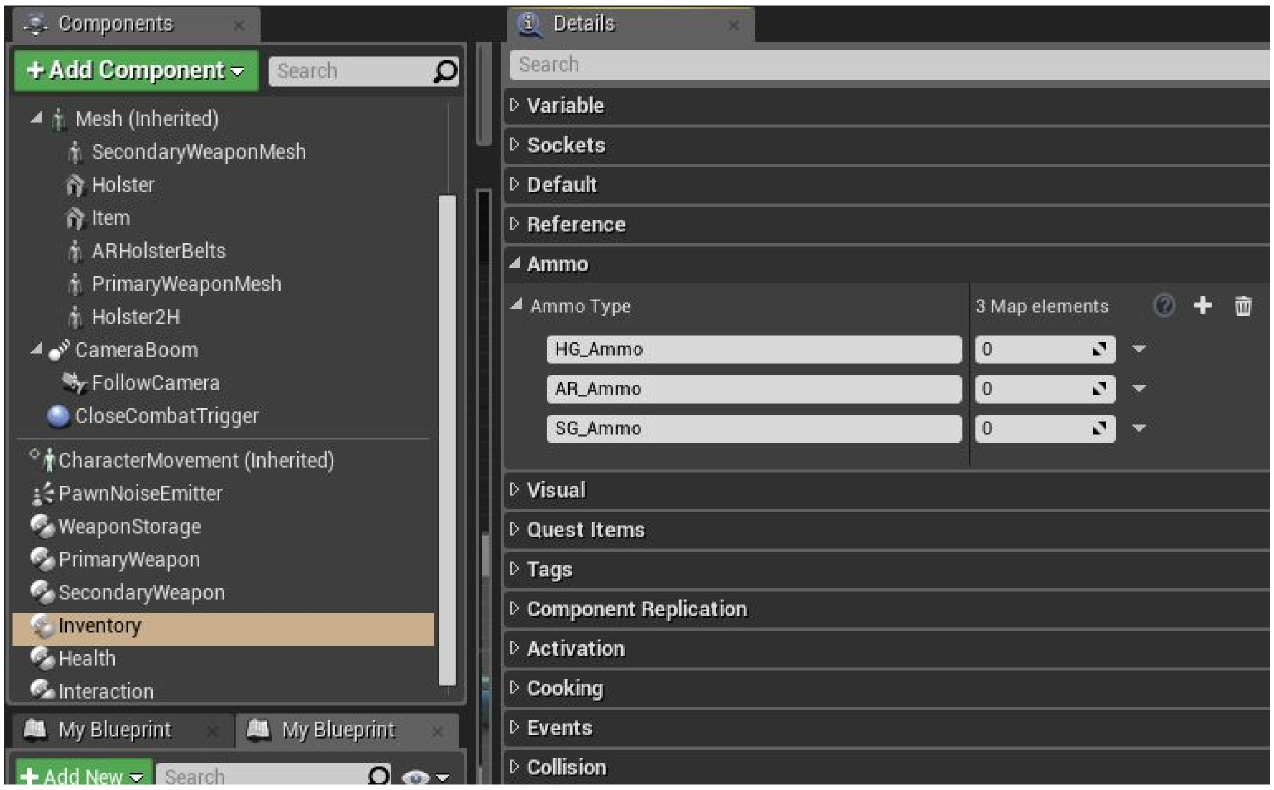
WeaponName is a technical name of the weapon. Use the same name you use to name the new index in the array. In this example, it is Knife.
WeaponMeshForItem is a 3d model of the weapon an item. It is used when you drop the weapon on the ground with the help of WeaponManager
RelativeTransformForItem is a relative location of the item.
HudImage is an icon for the weapon.
HudWeaponName is a name of the weapon for HUD.
Attacks is a list of attack types with the weapon. It is expanded by clicking upon + which is across Attacks.
1Hand is an animation of the strike when a one-handed weapon is used.
2Hand is an animation of the strike when a two-handed weapon is used.
Unarmed is an animation of the strike when no weapon is used.
*DamageStruc*t - Cis a structure which contains damage parameter. It is possible to assign several damage values for a single strike for a combo, for instance.
DelayForDealDamage is a time-delay in seconds before inflicting damage. Note, if you have several damage values, then the following delay value calculated relative to the previous one.
Damage is a weapon damage value.
DamageAnim is a name for the enemy animation when it gets hit. For instance, a character can fall or fly in a particular direction. In this case, set this animation for the enemy and give it a name. Each enemy can have its own reaction for this strike.
ImpulseStrength is an impulse value for the strike on physical objects.
ImpulseForCarMultiply is a percentage of base impulse applied to the car. The value is in range between 0 and 1.
Distance is a distance in cm at which the damage will be applied.
HitLoudness is the loudness of the hit for the AI.
DelayForInterrupt is a time-delay after which an attack can be interrupted.
HideWeapon. Whether to hide firearm when attacking.
ItemInHand is a structure of parameters to display an item in hand.
ItemMesh is a 3d model of the weapon.
SocketName is a socket to which the weapon will be attached. By default, there are two sockets for the left and the right hand:Socket_Item_LHand and Socket_Item_RHand respectively.
Transform is a weapon location relative to the socket.
SoundAttack is a sound of the attack.

This is it, our weapon has been added into the game and has its own settings. In order to test the weapon by adding it to the character, use following nodes from the example for the level blueprint. Note, if you add the weapon as soon as the game starts (BeginPlay), then add a time-delay for the initialization of the system.



Inventory and picking up items

Inventory is for storing various items which the character picks up. It is presented as the component “**BP_Inventory**” and it contains the logic for different items. By default, the inventory class supports the following types of items:
Weapon is a firearm. When one picks up such items, the inventory executes the logic for adding a weapon in “**WeaponStorage**” for the pawn.
WeaponMelee is a melee weapon. When one picks up such items, the inventory executes the logic for adding a weapon in “**WeaponStorage**” for the pawn.
Ammo is an ammunition. Firearms use ammo right from the inventory. For each weapon type, a particular ammo type to use must be specified. By default, there are 3 ammo types used in the inventory:
HG_Ammo is ammo for a handgun and semi-automatic handgun.
AR_Ammo is ammo for an assault rifle.
SG_Ammo is ammo for a shotgun.
You can add any amount of ammo types. For that, select the component “Inventory” in the character’s pawn and proceed to the corresponding tab settings “Ammo”.

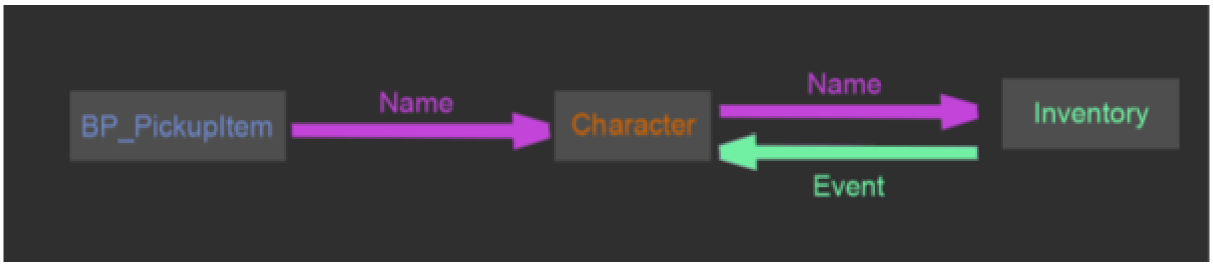


Ammo Type — 3 Map elements

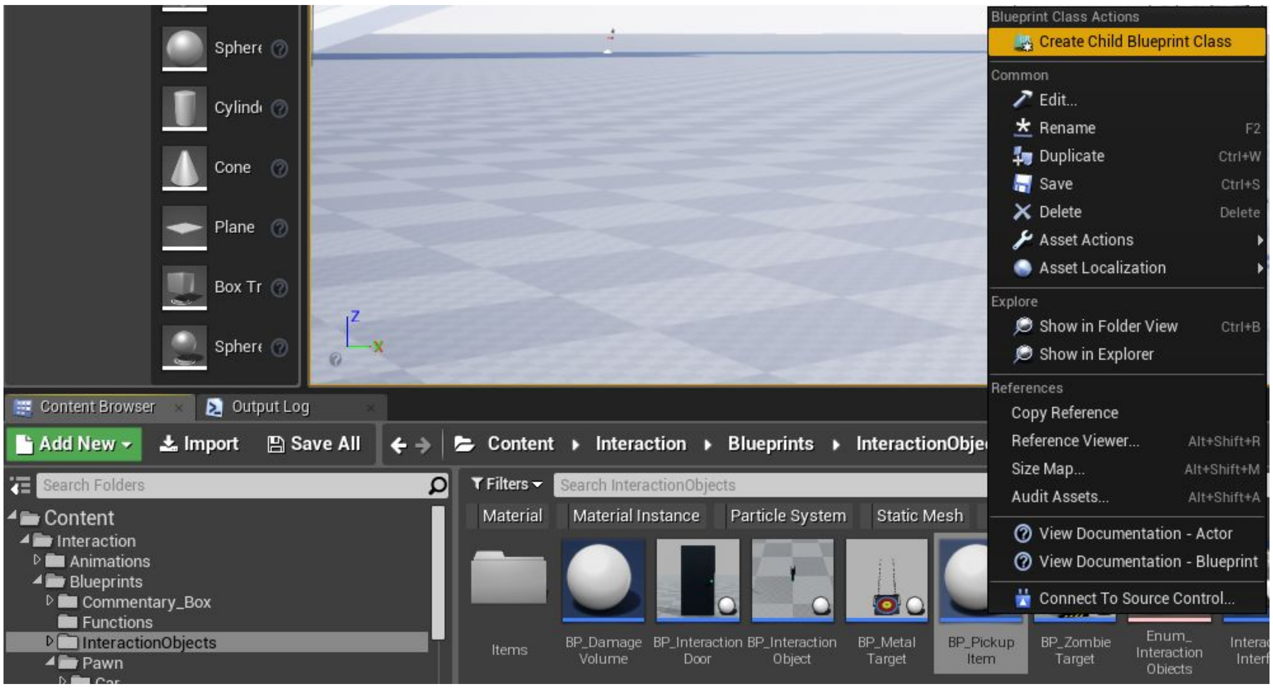
HG_Ammo	0
AR_Ammo	0
SG_Ammo	0

And the last type of items in the inventory is:
QuestItem. This item can be anything, e.g. either a resource for crafting of anything or door key, etc.

Let’s consider all item types in practice.
The class for picking up items **BP_PickupItem** is for interaction between the character and items. It works like this:

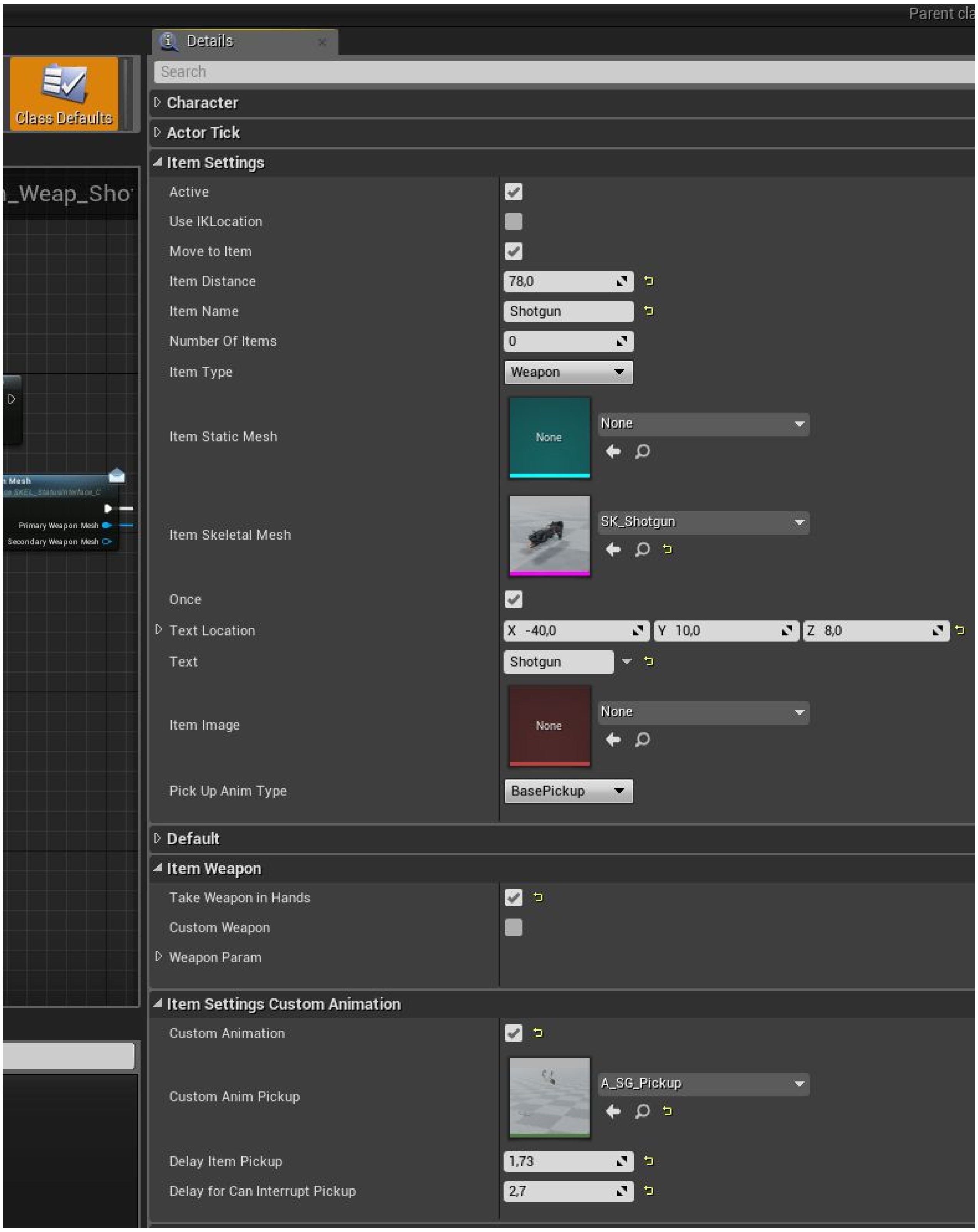


Let’s consider an item for a firearm. You can directly place the item in the scene and set it up there (since the functionality of the item to be picked up allows one to completely create a weapon without the “WeaponStorage” component), but, in this case, you will need to either copy the item or use it just once. I recommend to create a child item class and set it up for multi use, since it will be easier to add it into the scene via blueprint.
For that, click RMB on **BP_PickupItem** and choose “**Create Child Blueprint Class**” next.



In the project, there are already templates available for child classes of Items in the project (weapons, ammo, gas canister, and so on...)

Open the newly created item class and find “**Class Defaults**”, and from there proceed to the settings block “ItemSettings”. We will use a shotgun as an example for the item.


Details

Character

Actor Tick

Item Settings

Active	✓
Use IKLocation	
Move to Item	✓
Item Distance	78,0
Item Name	Shotgun
Number Of Items	0
Item Type	Weapon
Item Static Mesh	None
Item Skeletal Mesh	SK_Shotgun
Once	✓
Text Location	X -40,0 Y 10,0 Z 8,0
Text	Shotgun
Item Image	None
Pick Up Anim Type	BasePickup

Default

Item Weapon

Take Weapon in Hands	✓
Custom Weapon	
Weapon Param	

Item Settings Custom Animation

Custom Animation	✓
Custom Anim Pickup	A_SG_Pickup
Delay Item Pickup	1,73
Delay for Can Interrupt Pickup	2,7

Active. Whether the item is active (whether it can be picked up). Since we need to pick up the shotgun, we leave the property active (true). This functionality can be useful when there is no need to pick up the item, for instance, when the item is behind a glass cabinet. While the cabinet is locked, the parameter Active must be false.
Use IKLocation. Whether to use IK placements for hands on the item.
Move To Item. Whether the character will be moving towards the item. Since we will pick up the shotgun from a table, we leave the property active (true).
Item Distance is for guidance of the animation. The picking up of a two-handed weapon was animated under the condition that the character was 78 cm away from the item.
Number Of Items is the amount of items. There is no need to specify it in this example, since we only have a single weapon. However, if you create an item for the ammo, then you need to specify its amount.

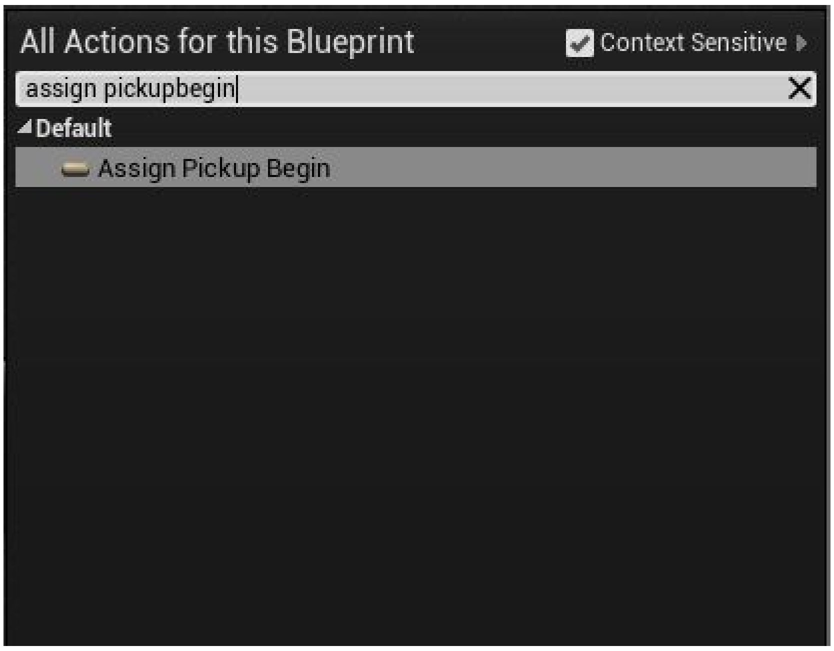
ItemType is an item type in the inventory. Since our item is a weapon, we choose “Weapon” type.
ItemStaticMesh sets the visual representation of the item as a static mesh. Since our shotgun is a skeletal mesh, we leave the property empty.
ItemSkeletalMesh sets the visual representation of the item as a skeletal mesh. Our shotgun is a skeletal mesh, so choose the shotgun model.
Once. Whether the item is picked up one time only. It is so in our case. For example, this might be useful for ammo crates, because one can pick up ammo clips from it several times.
Text Location is a local coordinates for placement of a pop-up text.
Text is a message displayed as a pop-up text for an item when the character approaches it.
ItemImage is an icon for the item when it is picked up and passed to the inventory. By default, it is used for the ammo.
PickUpAnimType is an animation type for the character when he picks up the item. The animation for picking up items can be set up in the component “Interaction” in the pawn.

Settings stack “Item Weapon”
With the help of these settings, you can choose whether the weapon will be held in hands (*TakeWeaponInHands*) or it will be immediately moved in WeaponManager. Since we are going to use special animation to pick up the shotgun from the table, we specify that, yes, the character will hold the weapon in his hands.
The parameter “*CustomWeapon*” allows one to create a unique weapon from the item without adding it to the list “Weapon List” in “WeaponStorage” component. Below is the structure for weapon settings. The settings are described above.

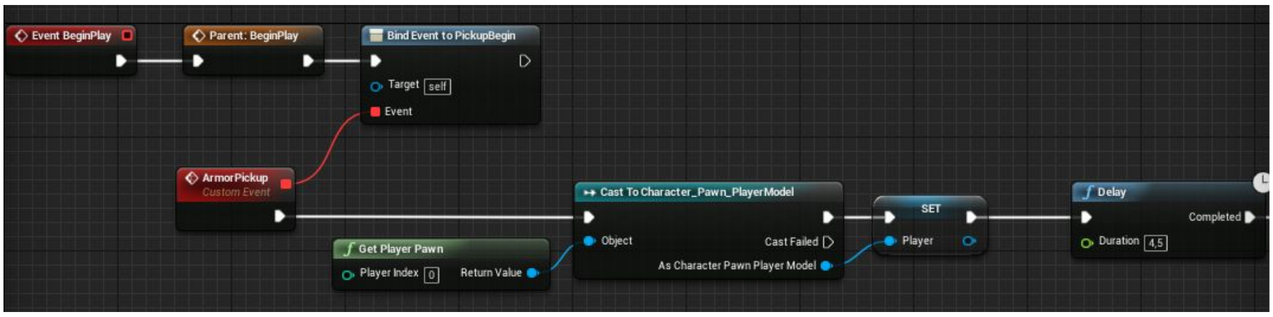
Settings stack “Item Settings Custom Animation”
These settings are to replace animation for basic item pick up. Since we have an animation to pick up the shotgun from the table, we use it.
Delay Item Pickup determines the time delay between the moment an item pick up starts and the moment the logic is executed in the inventory (the shotgun appears in hands).
Delay for Can interrupt Pickup determines the amount of time since an item pick up has begun, within which it is possible to interrupt the interaction.

Now our item-shotgun is ready, just place it in the scene and pick it up with the character.

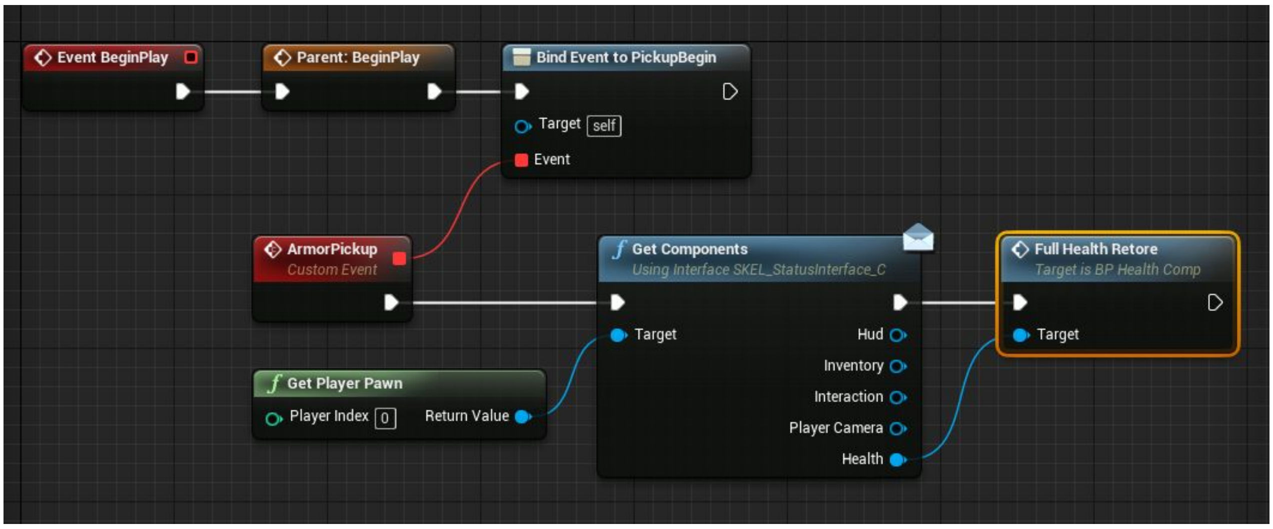
Items with additional logic.
Likewise, you can create completely unique items. In the project, there are examples of such items: a bulletproof vest and a gas mask.
As it is described above, create a new item and specify necessary settings. Then, move to the event graph and attach for the event “BeginPlay” a dispatcher “PickupBegin”.



Later, this event will be triggered in the moment when the item pick up just starts. For the bulletproof vest, these are various effects and character’s health buff.



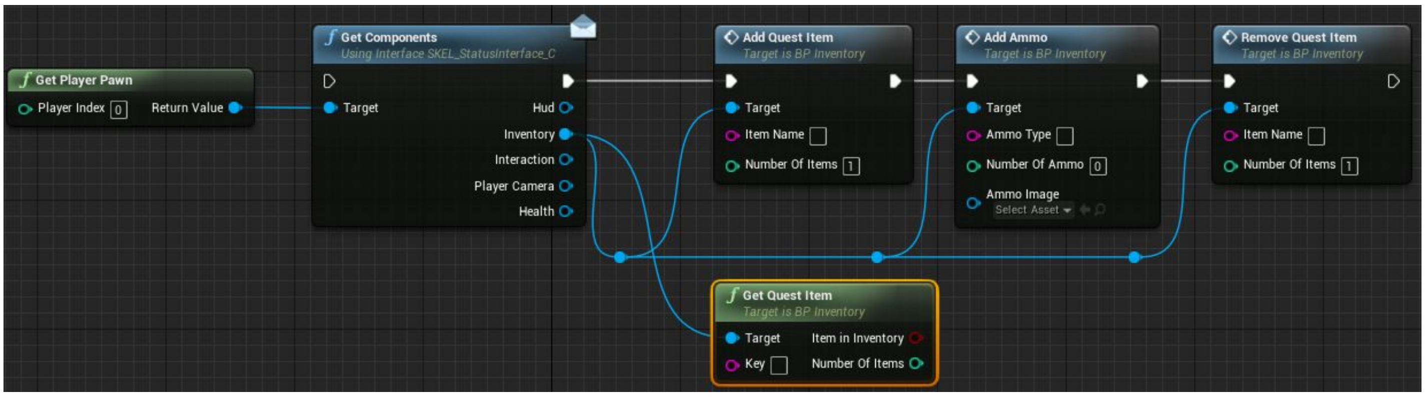
You can create other items, for instance, a med-kit would look like this:



The item references the health component of the character and it triggers an event of full health restoration, The interface “Get Components” is created for a quick access to character’s components.

Additional events in the inventory

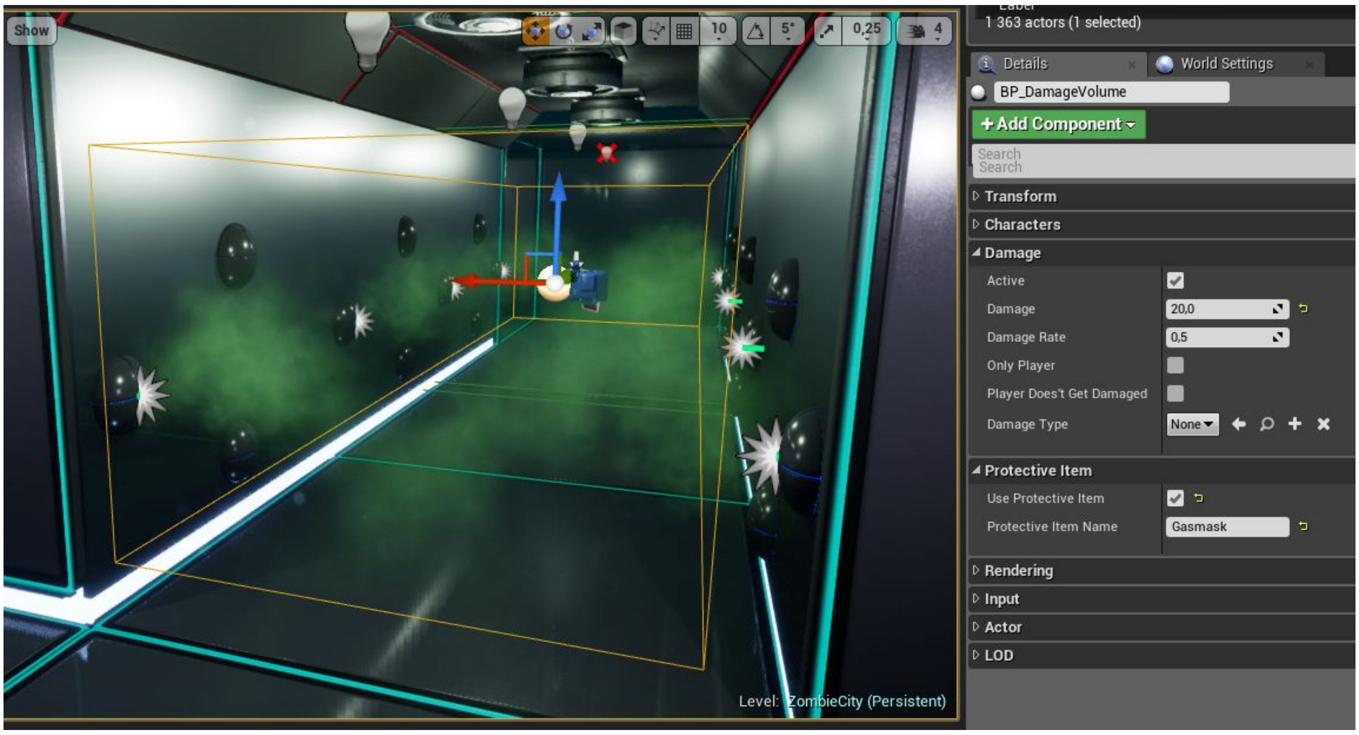
There are several functions available by default to work with the inventory. If you need to execute logic not from the pawn, you can access the inventory and other components via the interface “Get Components”.



Add Quest Item adds a quest item in the inventory. Specify name and amount.
Get Quest Item is a request about the availability of the quest item in the inventory. For instance, if there is a key in the inventory, you will be able to open a door or use it elsewhere.
Add Ammo adds specified ammo type.
Remove Quest Item removes an item or its certain amount from the inventory.

Danger zones

In the project, there is a class to assist you in creating danger zones for the characters. For example, these are gas chambers, fire regions and so on.
Let’s consider an example of gas corridor from the level “ZombieCity”, where the danger zone deals damage to the character if he or she does not have a gas mask.



BP_DamageVolume

Damage
Active ✓
Damage 20,0
Damage Rate 0,5
Only Player
Player Does't Get Damaged
Damage Type None

Protective Item
Use Protective Item ✓
Protective Item Name Gasmask

To create a danger zone, simply drag class **BP_DamageVolume** into the scene and adjust its bounds to necessary size.
Settings:
Active. Whether the danger zone is active. It will deal damage if it is active.
Damage - Cis the amount of damage that will be applied to the character with a certain

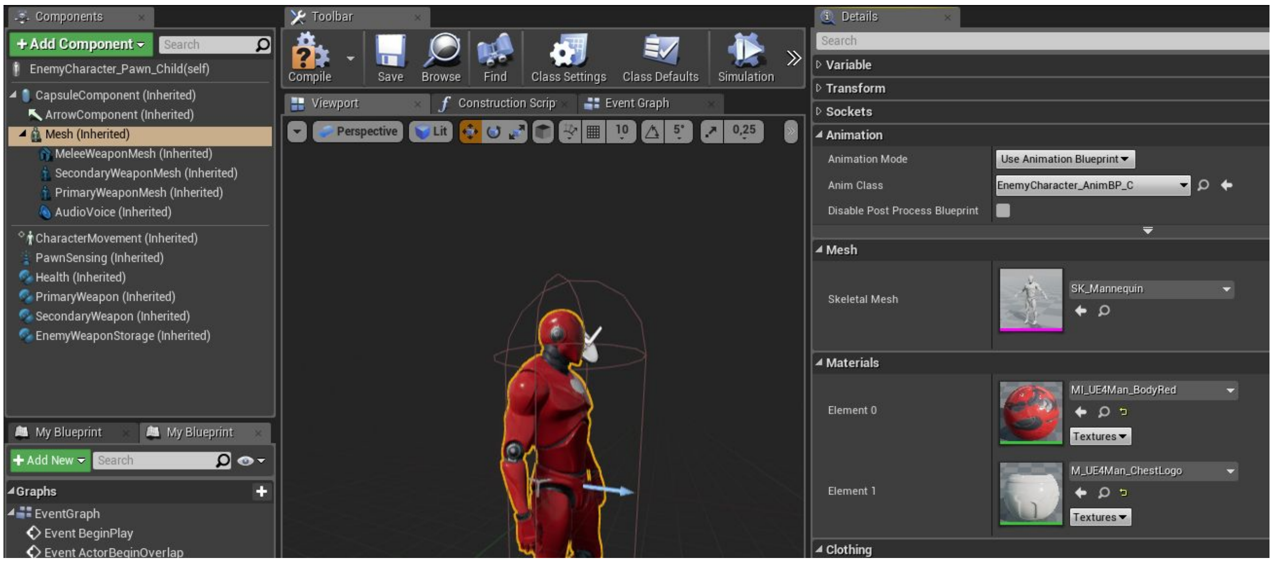
time interval.
DamageRate is a time interval in seconds, after which the damage will be dealt.
OnlyPlayer. When the damage will be applied to the character only.
PlayerDoes'tGetDamage.The damage will be inflicted upon everyone except the player.
DamageType is a type of damage. Based on this type, you can make additional logic, for instance, you can create damage type "Fire" and trigger the on-fire effect for the characters.

Settings "Protective Item"
UseProtectiveItem. Whether using a protective item from this danger zone.
ProtectiveItemName is a name of the protective item. The danger zone will look specifically for an item with this name in the inventory of the character.

Artificial Intelligence - Enemies

The capabilities of the Artificial Intelligence (AI) have been greatly expanded in the 3rd update. There is the base parent enemy class "**EnemyCharacter_Pawn**" in the project . All other child classes inherit from this class as types of enemies. By default, there are 3 types of enemies available in the project: a soldier, a zombie, and a cyber-zombie.

Let's consider how to set up AI for a new enemy. Make a child class from the class "**EnemyCharacter_Pawn"** and open it. At first, we will determine the visual representation. Select **Mesh** component and choose a character's model. Also, you can add additional meshes or other components, for example, enemies type "Zombie" have additional meshes, which split a character into 3 parts. The parts are picked up randomly, so the zombies do not look the same.



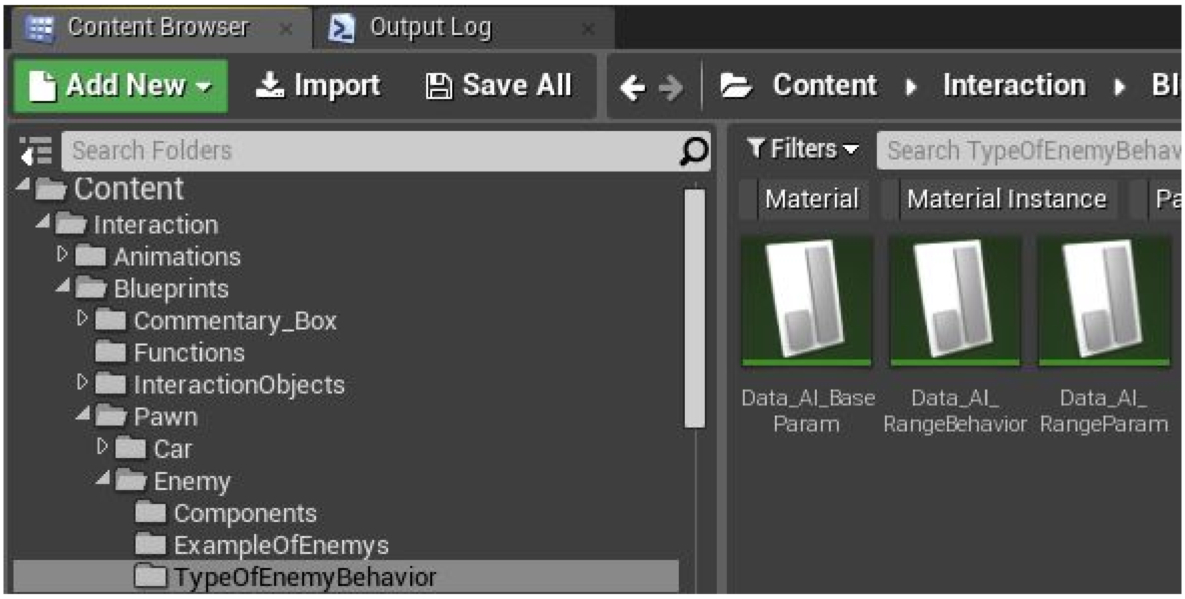
Animation Mode: Use Animation Blueprint
Anim Class: EnemyCharacter_AnimBP_C
Disable Post Process Blueprint
Skeletal Mesh: SK_Mannequin
Element 0: MI_UE4Man_BodyRed
Element 1: M_UE4Man_ChestLogo

Next, we set up animation for our new enemy. For that, open "**EnemyCharacter_AnimBP**" and find tab "**Animation**" in the list of variables.
You can replace the current animation of the character by selecting different structures of animation.

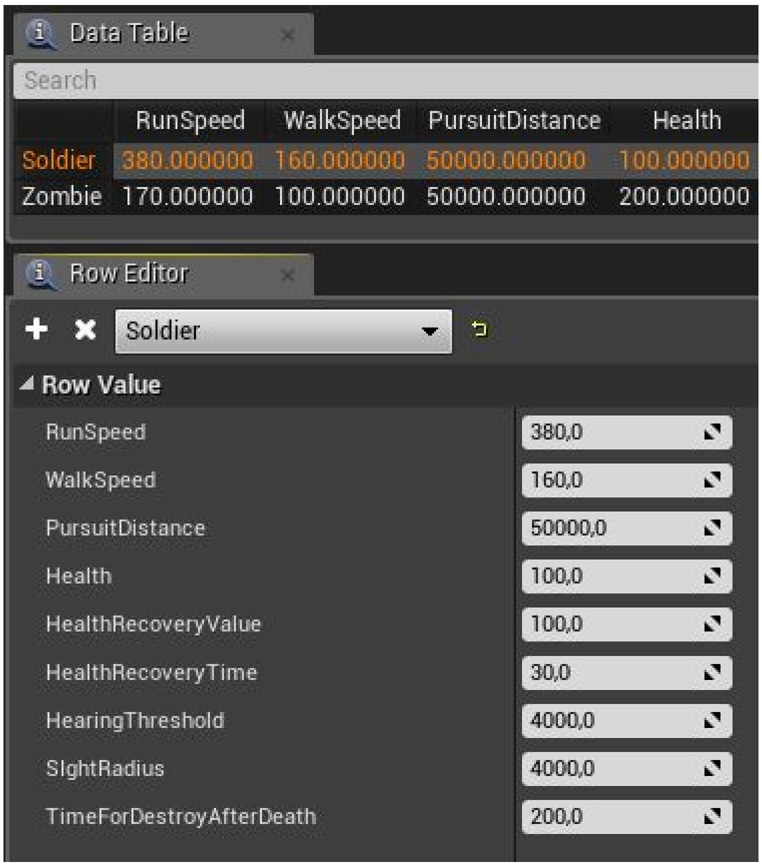
Make all necessary changes of the enemy looks and proceed to its settings. First of all, configure its basic parameters.

Enemy parameters

Data tables are used in the project for the convenience of filling in and storing data regarding characters behavior in combat. They are located as shown in figure below:



Expand "**Data_AI_BaseParam**"



	RunSpeed	WalkSpeed	PursuitDistance	Health
Soldier	380.000000	160.000000	50000.000000	100.000000
Zombie	170.000000	100.000000	50000.000000	200.000000

Soldier

RunSpeed	380,0
WalkSpeed	160,0
PursuitDistance	50000,0
Health	100,0
HealthRecoveryValue	100,0
HealthRecoveryTime	30,0
HearingThreshold	4000,0
SIghtRadius	4000,0
TimeForDestroyAfterDeath	200,0

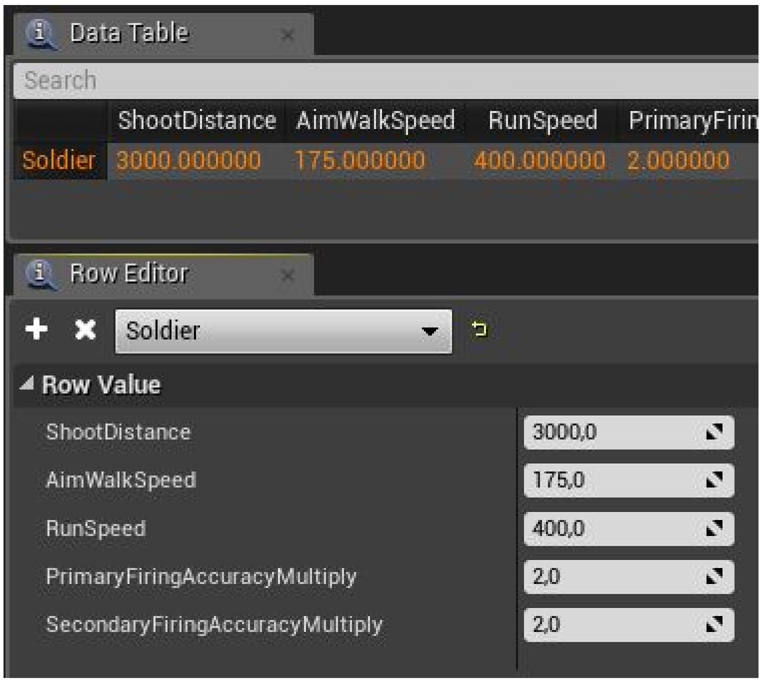
To create a new version of base parameter settings, click upon + and key-in name of parameters for our new enemy.

RunSpeed is a running speed of the character.
WalkSpeed is a walking speed of the character.
PursuitDistance is a distance within which the character will continue pursuing the target during combat.
Health is a base health value for the character.
HealthRecoveryValue is an amount of health the character can restore. Specify 0 if the

character needs to fully restore his health.
HealthRecoveryTime is a time interval after which the character's health gets restored after taking damage. Specify 0 if the character does not need to restore his health.
HearingTheshold is a hearing distance for the character.
SightRadius is a seeing distance for the character.
TimeForDestroyAfterDeath is a time-interval after which the character will be removed after death (for optimization). If you specify 0 then the character will not be deleted.

After we are done with the base character's settings, we proceed to next settings. If your character can use ranged weapons (which include all sort of throwing, shooting and such), then you need to specify behavior settings for long range combat.

Open table "**Data_AI_RangeParam**", add new settings, and give them a name (for convenience, it is better to use the same name you used in base settings).



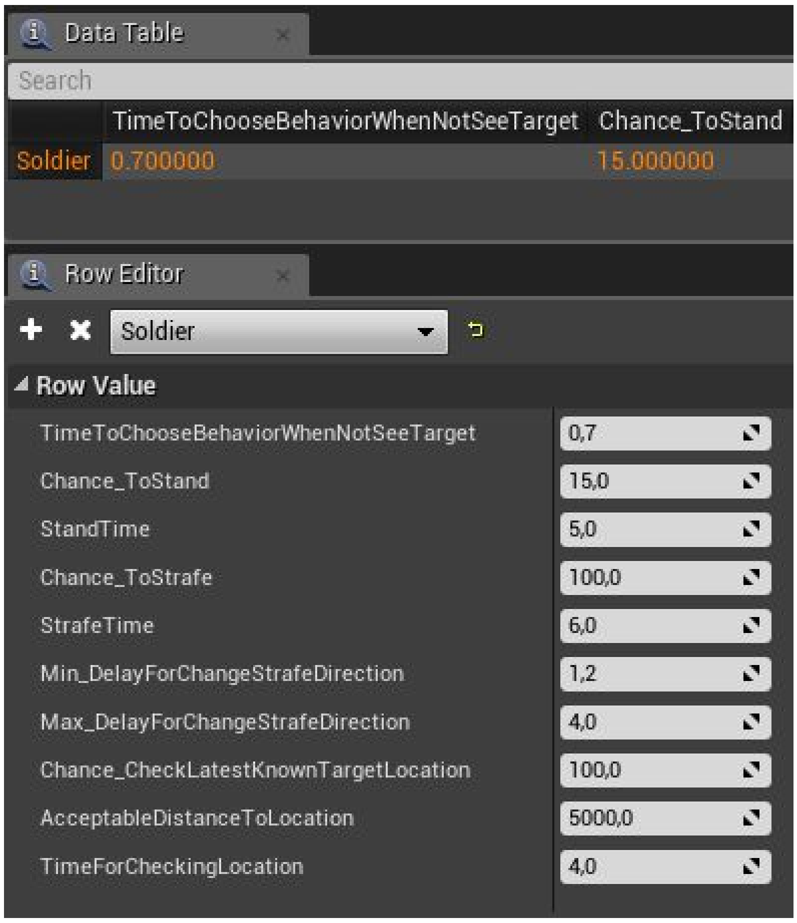
	ShootDistance	AimWalkSpeed	RunSpeed	PrimaryFirin
Soldier	3000.000000	175.000000	400.000000	2.000000

ShootDistance	3000,0
AimWalkSpeed	175,0
RunSpeed	400,0
PrimaryFiringAccuracyMultiply	2,0
SecondaryFiringAccuracyMultiply	2,0

ShootDistance is a distance to the target at which the enemy can engage in long range combat.
AimWalkSpeed is a character's walking speed during long range combat.
RunSpeed is a character's running speed during combat state.
PrimaryFiringAccuracyMultiply is an accuracy multiplier for the primary weapon based on its base settings.
SecondaryFiringAccuracyMultiply is an accuracy multiplier for the secondary weapon based on its base settings.

After you are done with base settings for long range combat, let's proceed to adjust behavior settings for long range combat.

Expand table "**Data_AI_RangeBehavior**", add new settings, and give it a name. This table contains settings for enemy behavior during long range combat.

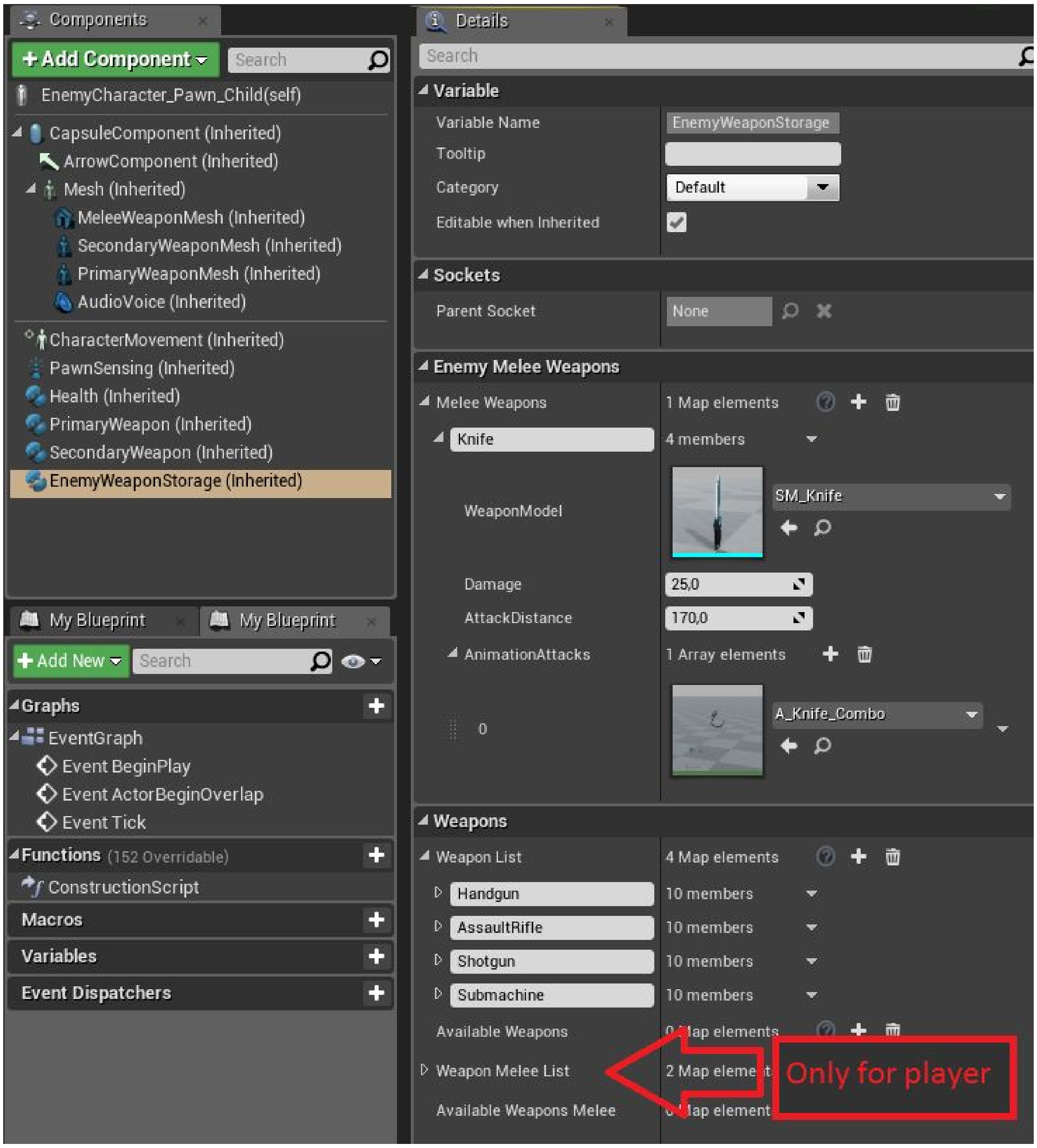


	TimeToChooseBehaviorWhenNotSeeTarget	Chance_ToStand
Soldier	0.700000	15.000000

TimeToChooseBehaviorWhenNotSeeTarget	0,7
Chance_ToStand	15,0
StandTime	5,0
Chance_ToStrafe	100,0
StrafeTime	6,0
Min_DelayForChangeStrafeDirection	1,2
Max_DelayForChangeStrafeDirection	4,0
Chance_CheckLatestKnownTargetLocation	100,0
AcceptableDistanceToLocation	5000,0
TimeForCheckingLocation	4,0

TimeToChooseBegaviorWhenNotSeeTarget is a time after which the character resumes patrolling or begins searching for target location last known to him, if the target is not in sight.
Chance_ToStand is a probability to fight without changing the position (15%).
StandTime is an amount of time the character will remain in the same position if it is *Chance_ToStand*.
Chance_ToStrafe is a probability for the character to move sideways or strafe when shooting.
StrafeTime is a time duration of moving sideways.
Min_DelayForChangeStrafeDirection is a minimum time-interval, after which the character changes the direction of his or her sideway movements.
Max_DelayForChangeStrafeDirection is a maximum time-interval, after which the character changes the direction of his or her sideway movements.
Chance_CheckLatestKnownTargetLocation is a probability for the character to search the last known target location if the target is not in sight.
AcceptableDistanceToLocation is a distance at which the character will go check the last known location of the target.
TimeForCheckingLocation is a time duration of the search for the target at its last known location.

The next step is to arm our new enemy. These can be both weapons for ranged combat and melee combat. Open our newly created enemy and proceed to component “**EnemyWeaponStorage**”.



Components

+ Add Component

EnemyCharacter_Pawn_Child(self)
- CapsuleComponent (Inherited)
  - ArrowComponent (Inherited)
  - Mesh (Inherited)
    - MeleeWeaponMesh (Inherited)
    - SecondaryWeaponMesh (Inherited)
    - PrimaryWeaponMesh (Inherited)
    - AudioVoice (Inherited)
- CharacterMovement (Inherited)
- PawnSensing (Inherited)
- Health (Inherited)
- PrimaryWeapon (Inherited)
- SecondaryWeapon (Inherited)
- EnemyWeaponStorage (Inherited)

My Blueprint

- Graphs
  - EventGraph
    - Event BeginPlay
    - Event ActorBeginOverlap
    - Event Tick
- Functions (152 Overridable)
  - ConstructionScript
- Macros
- Variables
- Event Dispatchers

Details

Variable

Variable Name	EnemyWeaponStorage
Tooltip	
Category	Default
Editable when Inherited	✓

Sockets

Parent Socket	None

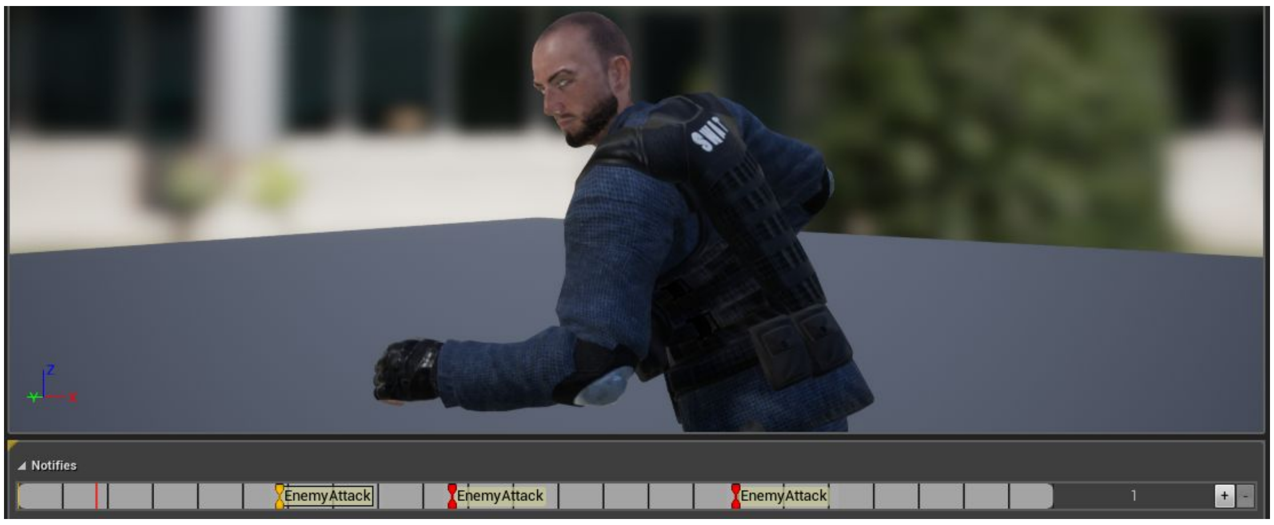
Enemy Melee Weapons

Melee Weapons	1 Map elements
Knife	4 members
WeaponModel	SM_Knife
Damage	25,0
AttackDistance	170,0
AnimationAttacks	1 Array elements
0	A_Knife_Combo

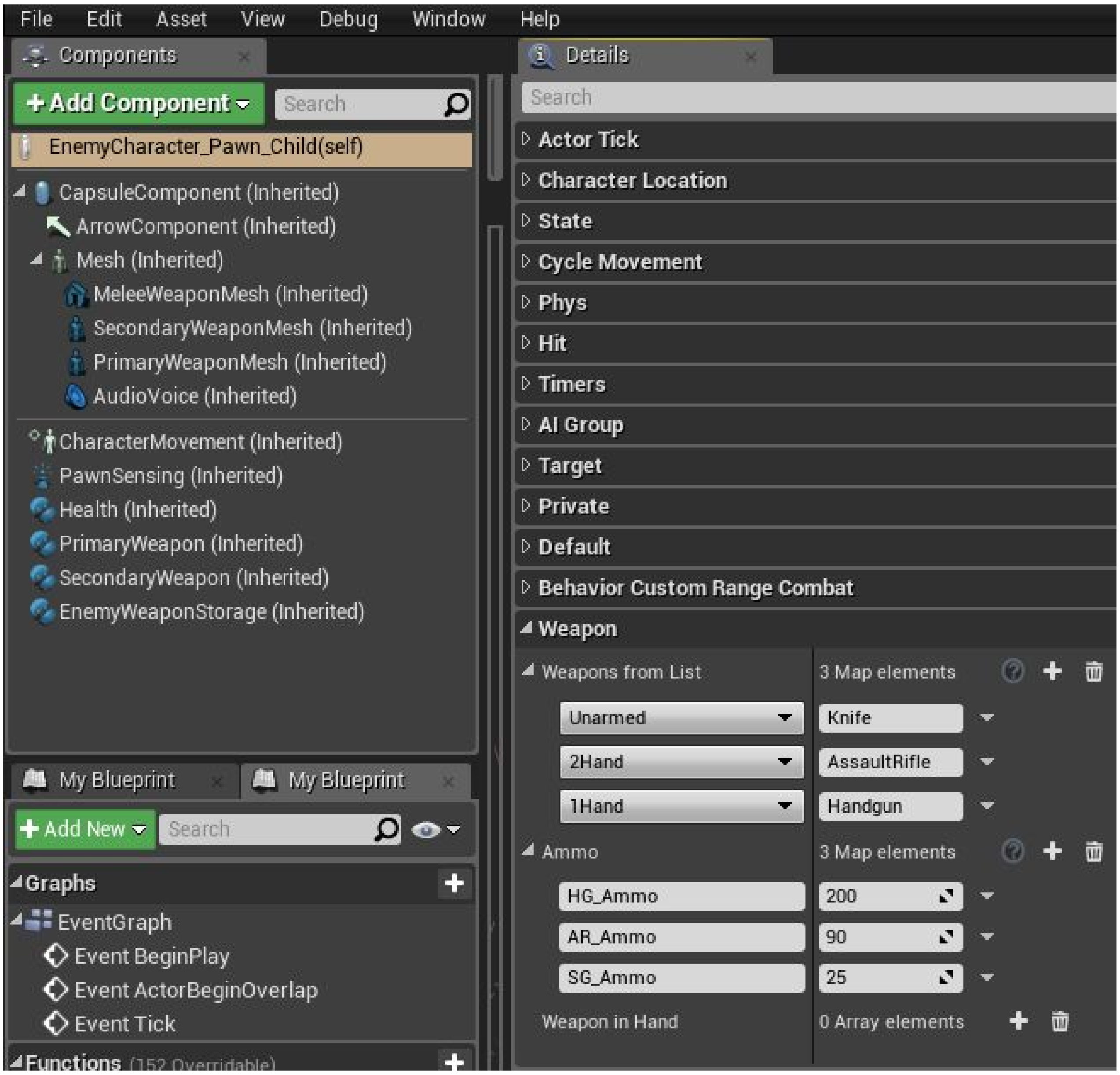
Weapons

Weapon List	4 Map elements
Handgun	10 members
AssaultRifle	10 members
Shotgun	10 members
Submachine	10 members
Available Weapons	0 Map elements
Weapon Melee List	2 Map elements
Available Weapons Melee	0 Map elements

Weapon is added to enemies based on the same principles it is added to the player, except a melee weapon. The moments of applying damage are specified in the animation of the enemy's attack. Note, here you add **all the weapons**, that can be used by this enemy.



With the help of notify “**EnemyAttack**”.
After you have added all possible weapons to the enemy, choose a weapon he will appear within the level. You can always change these parameters when a enemy spawns or select it in the scene and move to settings in tab “**Details**”. To choose a default weapon in this class, select the root component in the list of components and, on the right, find category “**Weapon**”.



Weapons from List	3 Map elements
Unarmed	Knife
2Hand	AssaultRifle
1Hand	Handgun

Ammo	3 Map elements
HG_Ammo	200
AR_Ammo	90
SG_Ammo	25

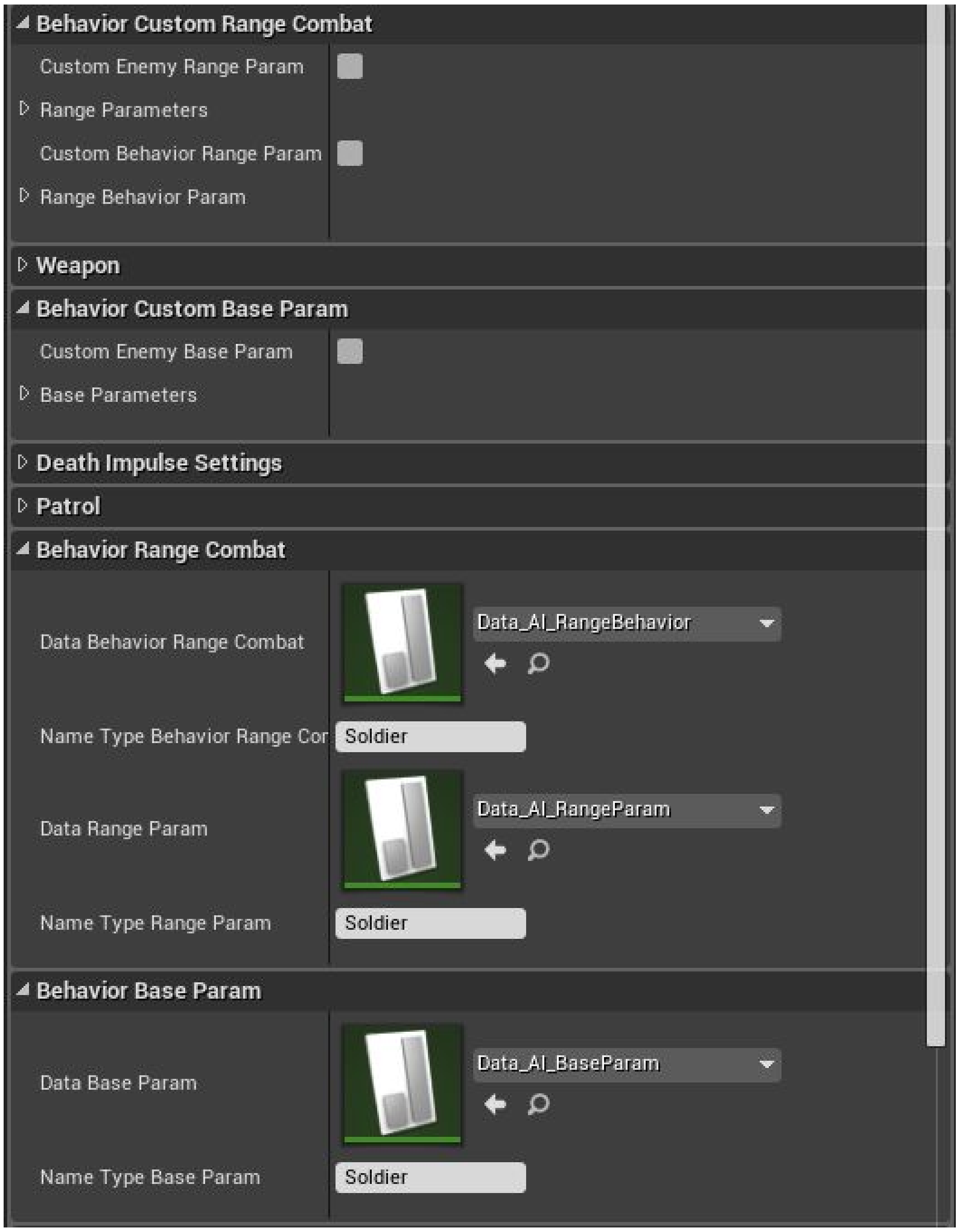
Weapon in Hand — 0 Array elements

By default, in the project, enemies can use two ranged weapons and a melee weapon. In this category, specify a default weapon for this enemy as well as its ammo type. As soon as the enemy is out of ammo, he will switch to melee combat.

Additional parameters of an enemy

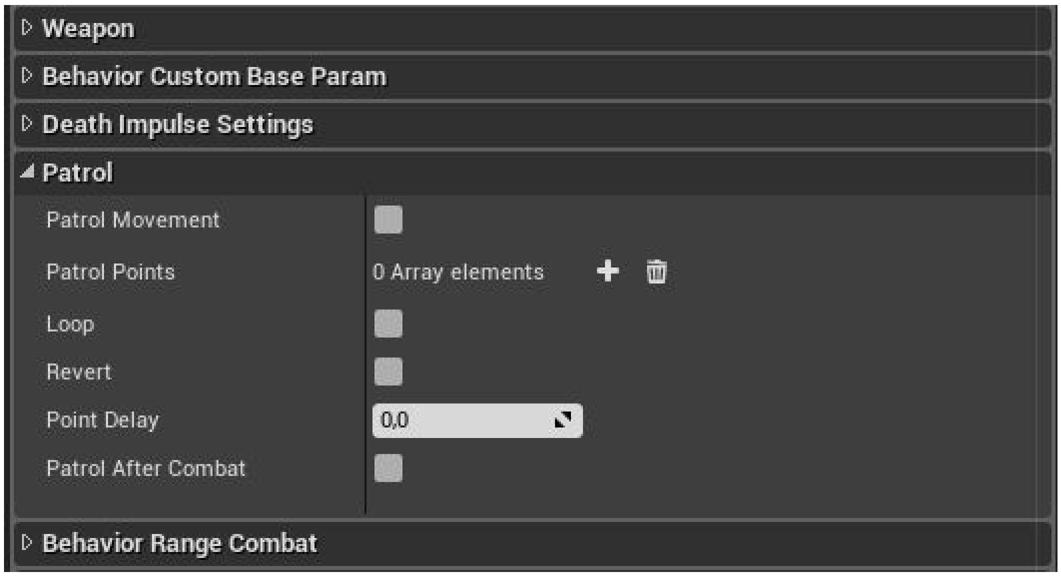
Also, let’s consider additional enemy parameters you can configure by default.

Behavior categories:


Behavior Custom Range Combat
Custom Enemy Range Param ☐
Range Parameters
Custom Behavior Range Param ☐
Range Behavior Param
Weapon
Behavior Custom Base Param
Custom Enemy Base Param ☐
Base Parameters
Death Impulse Settings
Patrol
Behavior Range Combat
Data Behavior Range Combat: Data_AI_RangeBehavior
Name Type Behavior Range Cor: Soldier
Data Range Param: Data_AI_RangeParam
Name Type Range Param: Soldier
Behavior Base Param
Data Base Param: Data_AI_BaseParam
Name Type Base Param: Soldier

In these categories, settings names from data tables must be specified (you have specified the settings themselves above). Also, in categories marked “**Custom**”, you can specify the settings directly without the use of tables.

Patrolling category:



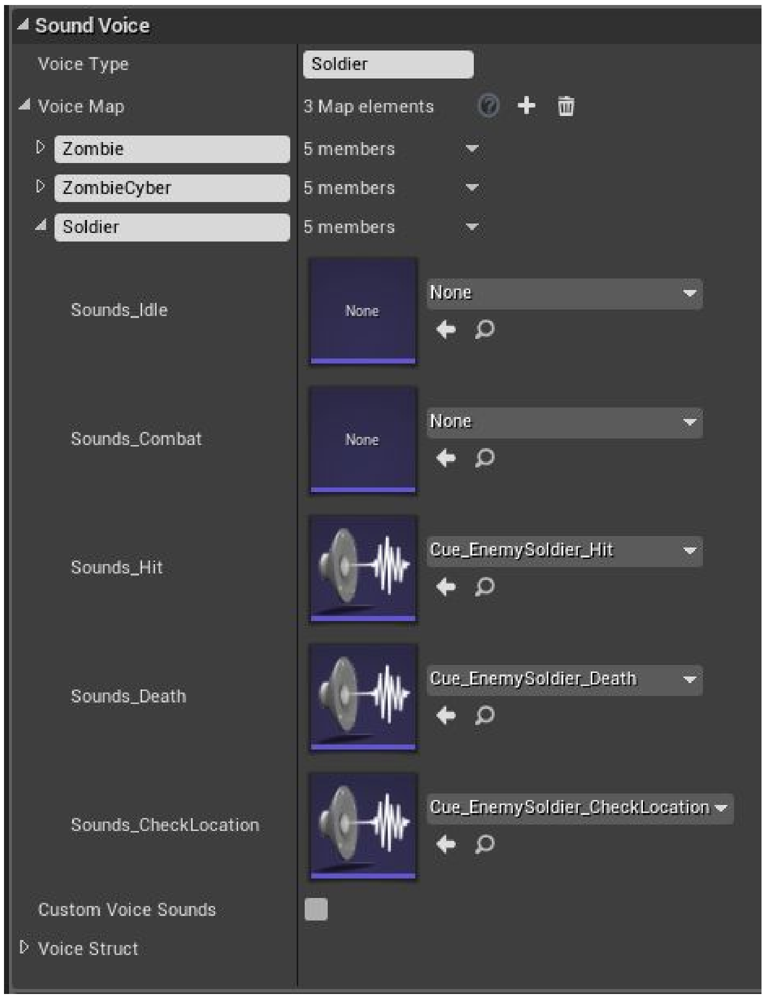
Weapon
Behavior Custom Base Param
Death Impulse Settings
Patrol

Patrol Movement	☐
Patrol Points	0 Array elements
Loop	☐
Revert	☐
Point Delay	0,0
Patrol After Combat	☐

Behavior Range Combat

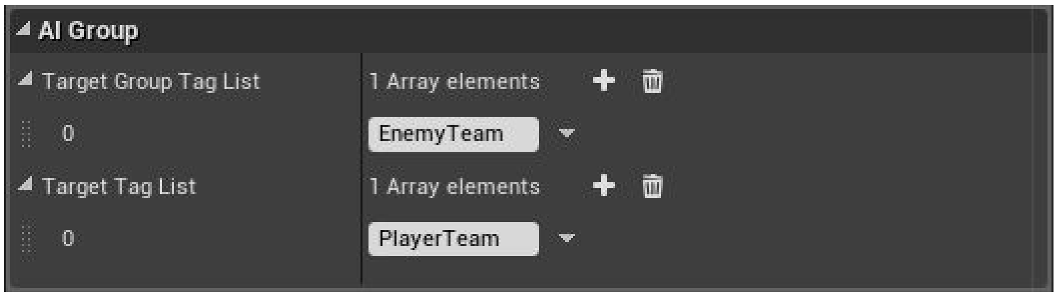
Patrol Movement. Whether the character will do patrolling of the area initially.
Patrol Points are key points for patrolling route.
Loop. Whether the character goes on patrolling upon reaching the last point.
Revert. If the parameter is enabled, then, when the enemy resumes patrolling, he returns in the first point of the route through points he has already passed.
PointDelay determines the amount of time it takes for the character to reach a route point.
PatrolAfterCombat. Whether the character resumes patrolling as soon as he is out of combat.

Voice settings:


Sound Voice
Voice Type: Soldier
Voice Map: 3 Map elements
Zombie: 5 members
ZombieCyber: 5 members
Soldier: 5 members
Sounds_Idle: None
Sounds_Combat: None
Sounds_Hit: Cue_EnemySoldier_Hit
Sounds_Death: Cue_EnemySoldier_Death
Sounds_CheckLocation: Cue_EnemySoldier_CheckLocation
Custom Voice Sounds
Voice Struct

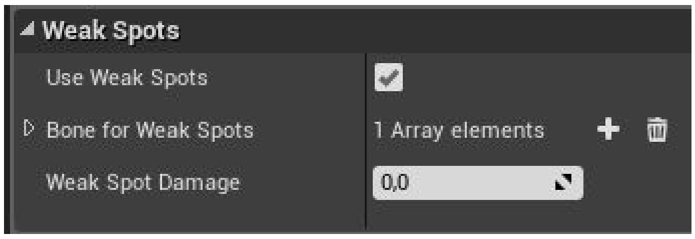
In this category, you can set up the sound of the enemy's voice, i.e. growling when in idle state like a zombie, sound response to taking damage, dying sound, and sounds of searching the location for the target. You can create voice types right away. For instance, Soldier would suffice to all people, so simply specify Soldier in the parameter *VoiceType*.

AI Group category


AI Group
Target Group Tag List: 1 Array elements
0: EnemyTeam
Target Tag List: 1 Array elements
0: PlayerTeam

Enemies can have both several types of enemies and allies. In this category, specify to which group the enemy is related and what groups are hostile to him.

Weak Spots category

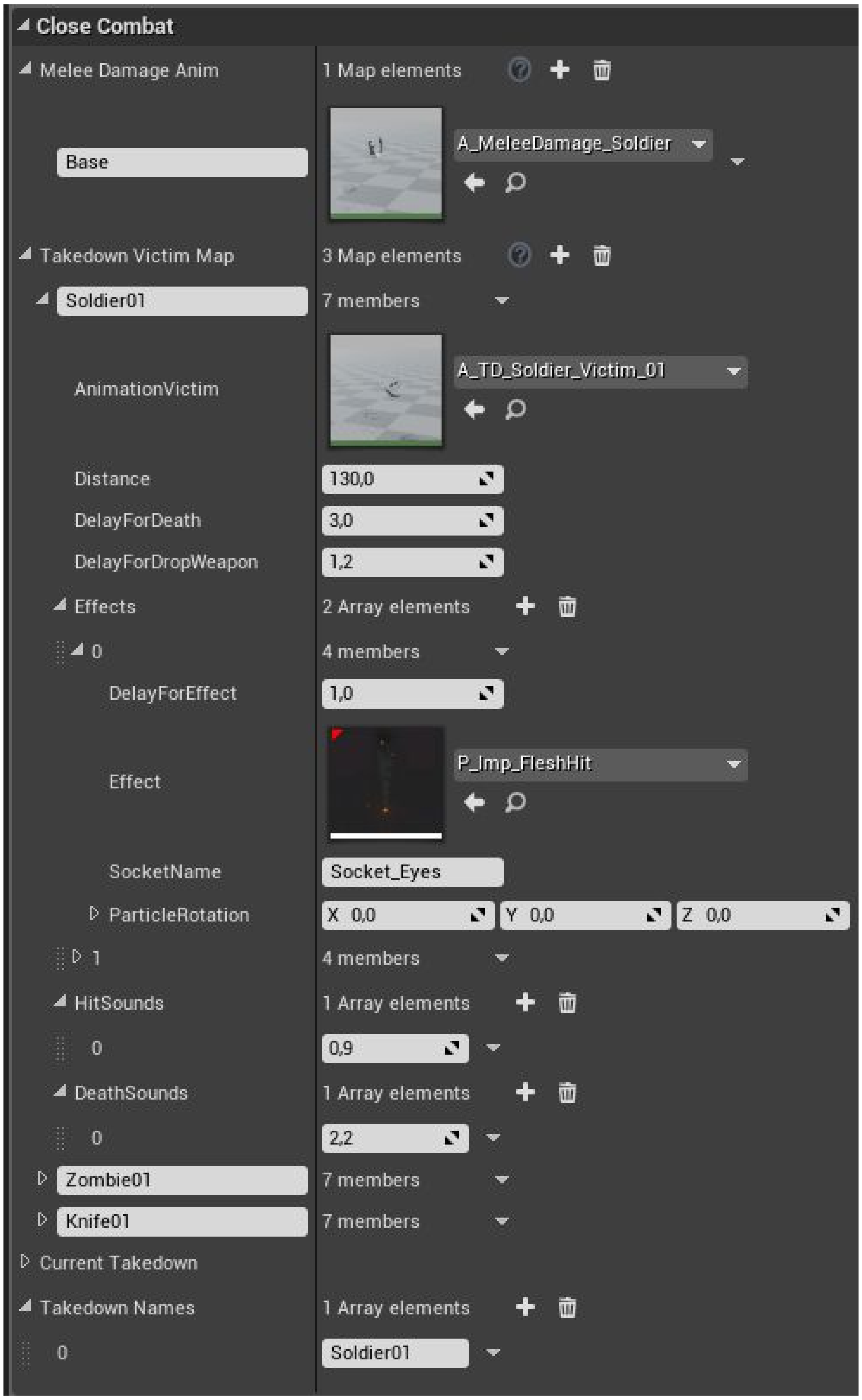

Weak Spots
Use Weak Spots: ✓
Bone for Weak Spots: 1 Array elements
Weak Spot Damage: 0,0

Specify here weak spots on the enemy which increase damage dealt when he is getting hit there. By default, it is needed for headshots. Also, you can use this functionality to create, for instance, bosses with weak spots.
UseWeakSpots. Whether the character has weak spots.
Bone for Weap Spots is a list of bones which are weak spots for the enemy.
Weak Spot Damage is an amount of inflicted damage when weak spots are hit. When its value is 0, it subtracts all enemy's health value, which results in death.

Close quarter combat with enemies

Enemies can receive damage from melee weapons. Also, the player can take down or execute an enemy with synchronized animations with the help of "**Takedown**" functionality.

Expand category "**CloseCombat**":



Close Combat

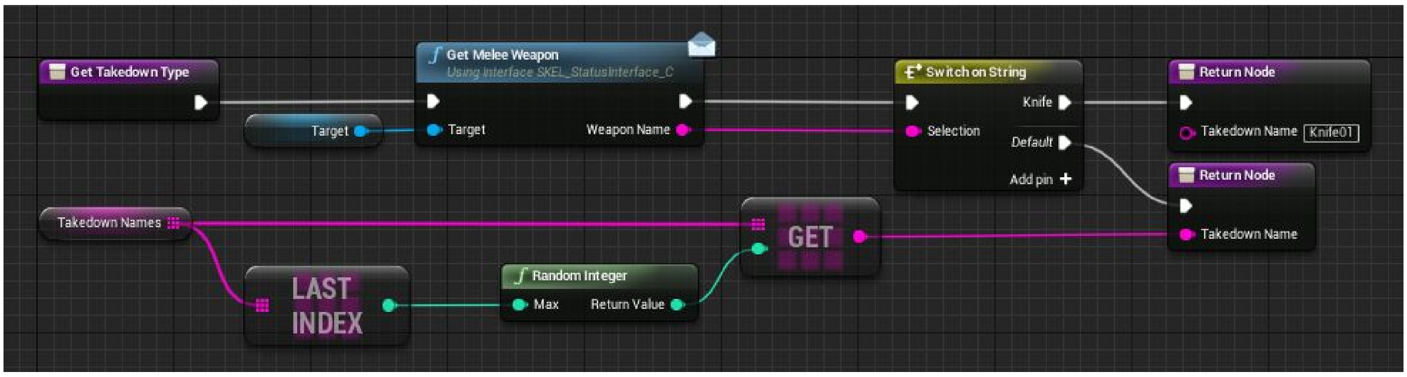
Melee Damage Anim	1 Map elements
Base	A_MeleeDamage_Soldier
Takedown Victim Map	3 Map elements
Soldier01	7 members
AnimationVictim	A_TD_Soldier_Victim_01
Distance	130,0
DelayForDeath	3,0
DelayForDropWeapon	1,2
Effects	2 Array elements
0	4 members
DelayForEffect	1,0
Effect	P_Imp_FleshHit
SocketName	Socket_Eyes
ParticleRotation	X 0,0 Y 0,0 Z 0,0
1	4 members
HitSounds	1 Array elements
0	0,9
DeathSounds	1 Array elements
0	2,2
Zombie01	7 members
Knife01	7 members
Current Takedown	
Takedown Names	1 Array elements
0	Soldier01

Melee Damage Anim contains a set of animations for receiving damage. By default, there is only one animation added for each enemy type. The type of animation when striking is specified in player’s melee weapon. Specify name, type, and animation itself.

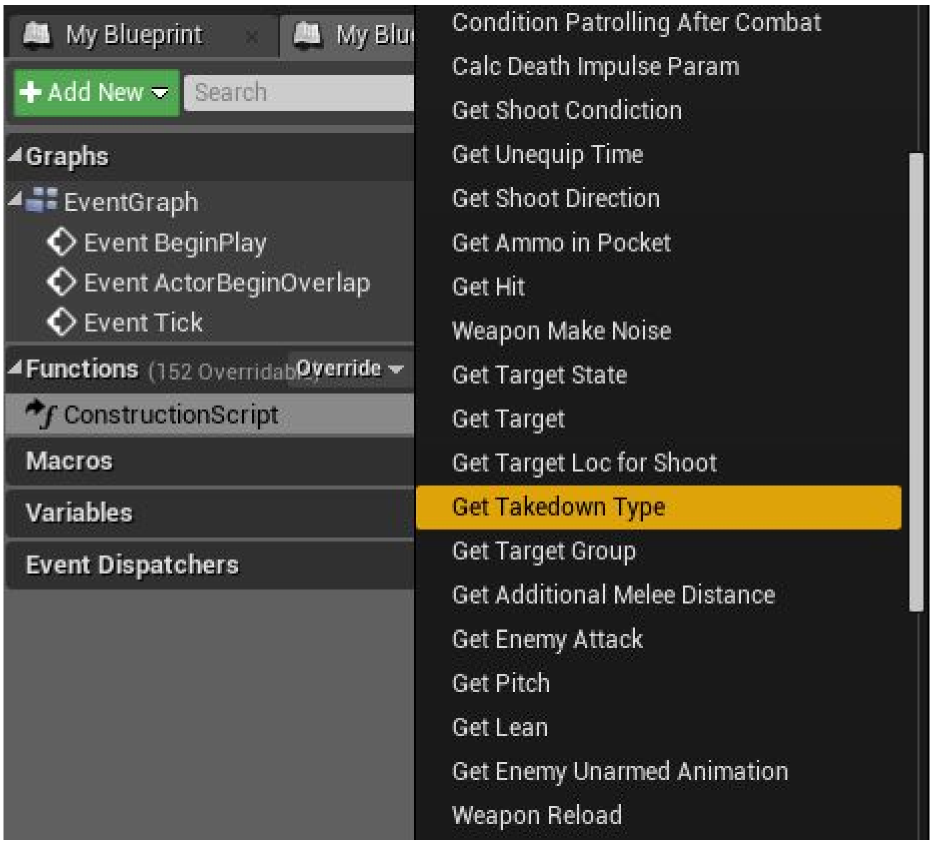
Takedown Victim Map are settings for enemy takedown (Takedown). In this group, specify takedowns available for this enemy type.
AnimationVictim is a target (enemy) animation during the takedown.
Distance is a relative distance between characters (the attacker and the target) during the takedown.
DelayForDeath is a time-delay after which the target dies.
DelayForDropWeapon is a time-delay after which the target drops its weapon.

Effects contains effects (particles) which will be played during the takedown. You can specified unlimited number of effects.
DelayForEffect is a time-delay after which the effect starts playing.
Effect is a type of the effect (particle).
SocketName is a location of the effect.
ParticleRotation is a rotation of the effect.
HitSounds is an array for playable voice sounds when receiving damage. The sounds themselves are to be specified in category "**SoundVoice**".
DeathSounds is an array of playable dying sounds. The sounds themselves are to be specified in category "**SoundVoice**".
.
Takedown Names are supported by this enemy takedowns specified above.

This is how takedowns are set up for enemies. By default, the takedown condition is when one gets right near the enemy. The conditions are set in function "**GetTakedownType**" and here is how it looks.

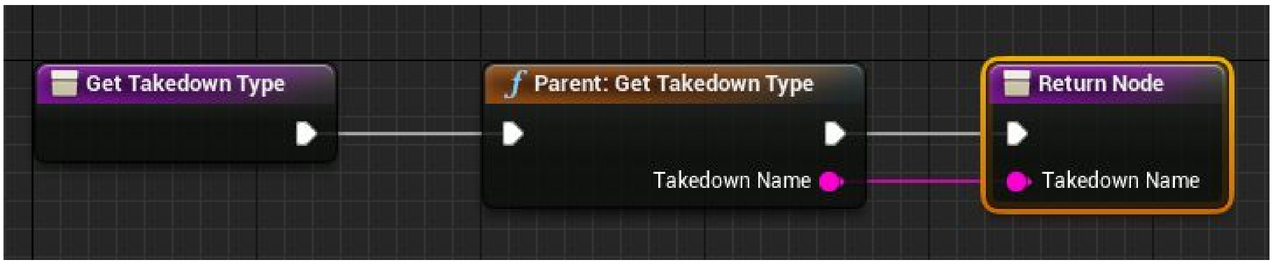


Any takedown is picked from the list "*Takedown Names",* but, if the target is equipped with a knife, then a takedown with a knife is selected. In order to change the conditions for the current player, rewrite the function in the following way:

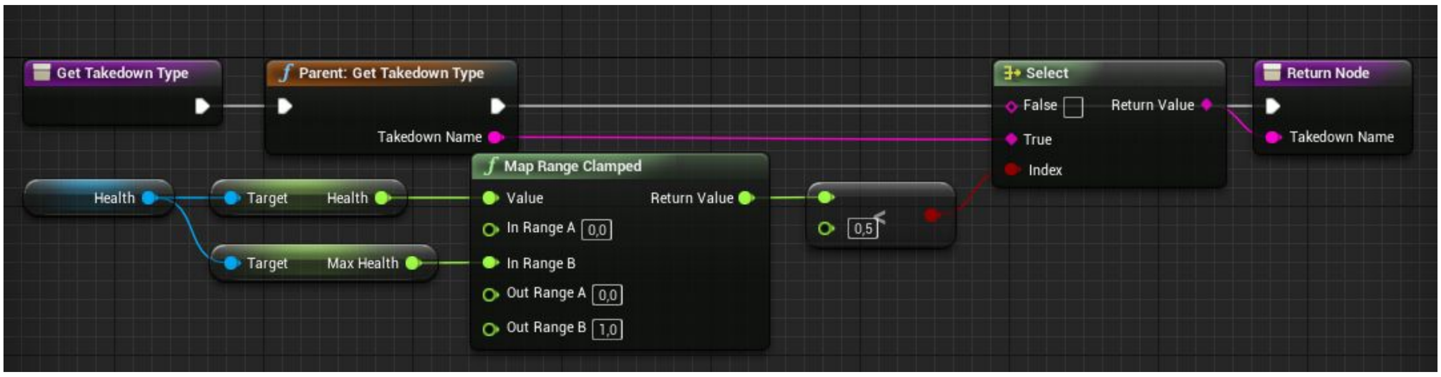


Press the button "Override" in the list of functions and find function"*Get Takedown Type*" there.

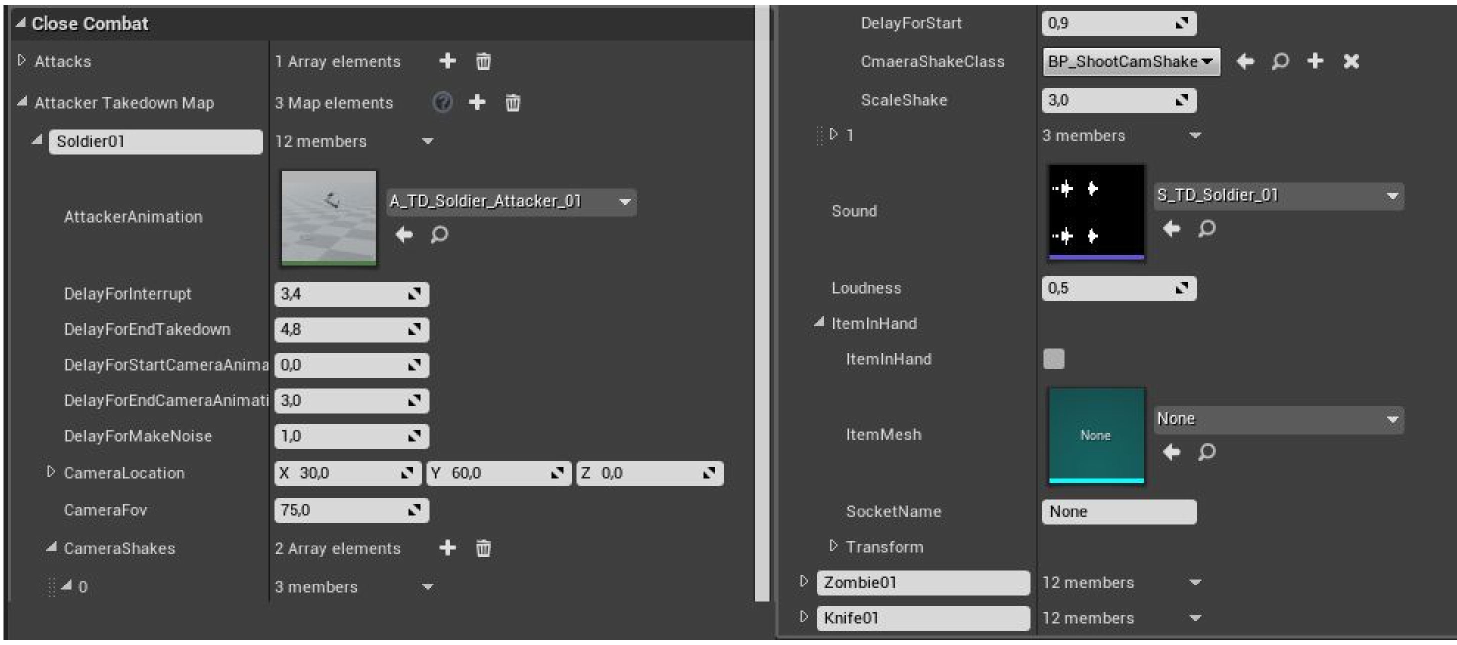
Open the function and this is what you will see:



Node “*Parent:Get Takedown Type*” accounts for basic conditions (described above) set by the parent class. You can add any conditions of your own. For instance, to execute an enemy only when he has less than 50% health remaining.



Leave the return variable “*Takedown Name*” of the function empty and then the enemy will be impossible to kill.
It concerns only settings for the enemy. Now we configure takedown settings for the player. For that, proceed to the player’s pawn and select component “Interaction”, then find category “CloseCombat”.



Close Combat

Attacks	1 Array elements
Attacker Takedown Map	3 Map elements
Soldier01	12 members
AttackerAnimation	A_TD_Soldier_Attacker_01
DelayForInterrupt	3,4
DelayForEndTakedown	4,8
DelayForStartCameraAnima	0,0
DelayForEndCameraAnimati	3,0
DelayForMakeNoise	1,0
CameraLocation	X 30,0 Y 60,0 Z 0,0
CameraFov	75,0
CameraShakes	2 Array elements
0	3 members
DelayForStart	0,9
CmaeraShakeClass	BP_ShootCamShake
ScaleShake	3,0
1	3 members
Sound	S_TD_Soldier_01
Loudness	0,5
ItemInHand	
ItemInHand	☐
ItemMesh	None
SocketName	None
Transform	
Zombie01	12 members
Knife01	12 members

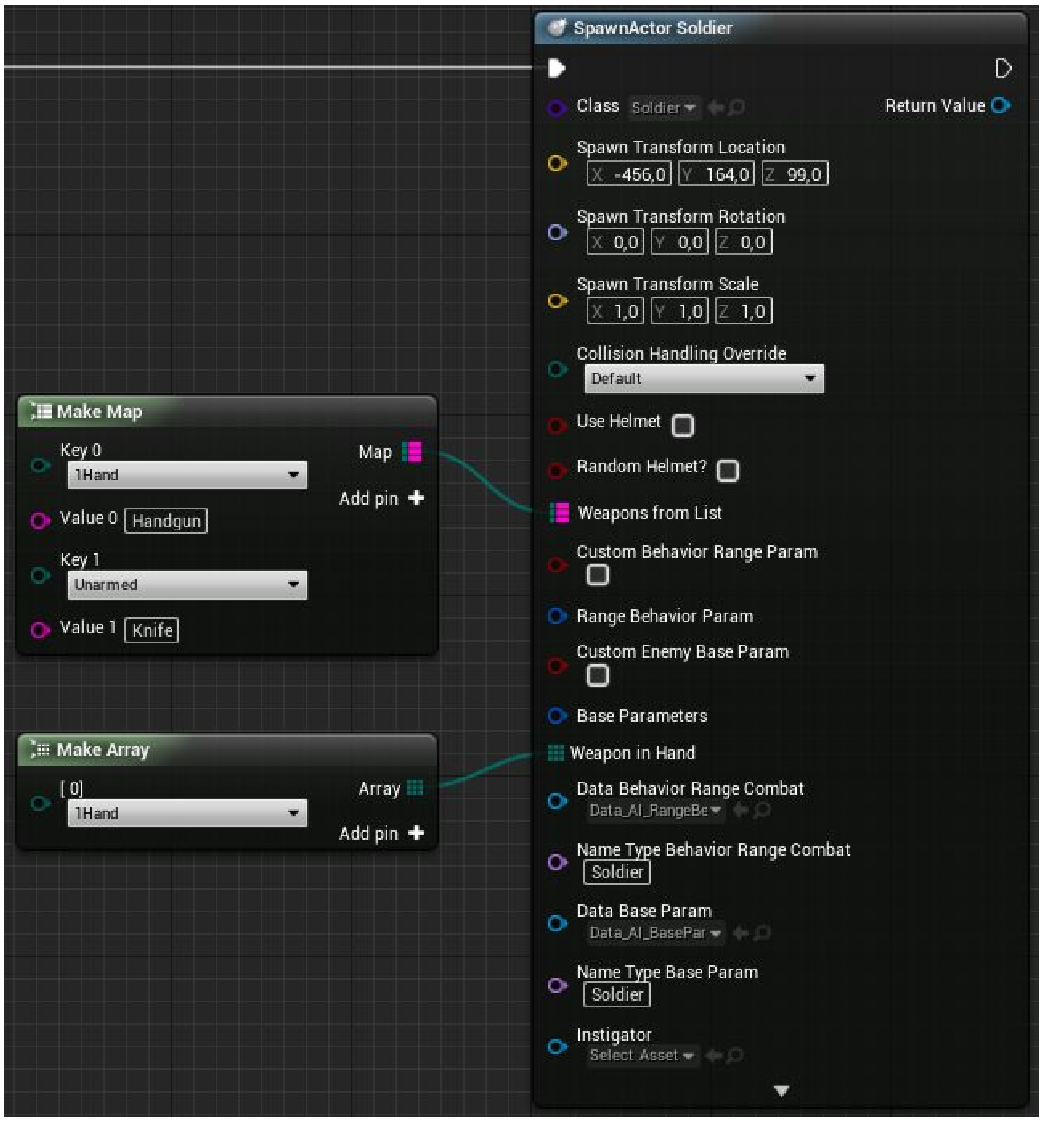
AttackerTakedownMap. By analogy with the target, this array contains all settings of the attacker for takedowns.
You must use the same name for attacker’s takedown as you used in the settings for the target.
AttackerAnimation is an attacker’s takedown animation.
DelayForInterrupt is a time-delay after which the takedown can be interrupted.
DelayForEndTakedown is an end time of the takedown.
DelayForStartCameraAnimation is when the camera animation starts.
DelayForStartCameraAnimation is when the camera animation is completed.

DelayForMakeNoise is a time moment when the takedown creates noise for the AI (nearby AI agents can hear it).
CameraLocation is a camera location for animation. During the takedown, camera will be moved there. These are sockets socketOffset in springarm.
CameraFov is a Field of view of the camera during animation.
CameraShakes is a shaking effect of the camera when hits land. You can specify any amount of such shackings.
DelayForStart is a time duration of camera shaking (each following camera shaking will use the delay relative to the previous one).
CameraShakeClass is a selection of camera shaking class.
ScaleShake is a multiplier the amplitude of camera shaking.
Sound is a sound of the takedown.
Loudness is a loudness value to be heard by the AI.
ItemInHand are settings for an item in hand (analogous settings for picking up items have been described above).

This is how you can add different number of various takedowns including unique ones.

How to add enemies into the scene via Blueprint

The project can also take care of spawning enemies in the scene with the help of node "**Spawn Actor From Class**":



SpawnActor Soldier
Class Soldier
Return Value
Spawn Transform Location X -456,0 Y 164,0 Z 99,0
Spawn Transform Rotation X 0,0 Y 0,0 Z 0,0
Spawn Transform Scale X 1,0 Y 1,0 Z 1,0
Collision Handling Override Default
Use Helmet
Random Helmet?
Weapons from List
Custom Behavior Range Param
Range Behavior Param
Custom Enemy Base Param
Base Parameters
Weapon in Hand
Data Behavior Range Combat Data_AI_RangeBe
Name Type Behavior Range Combat Soldier
Data Base Param Data_AI_BasePar
Name Type Base Param Soldier
Instigator Select Asset

Make Map
Key 0 1Hand
Value 0 Handgun
Key 1 Unarmed
Value 1 Knife
Map
Add pin

Make Array
[0] 1Hand
Array
Add pin

Choose your enemy in parameter "class" and additional parameters of the enemy will appear. Specify the location where the enemy should appear. Make sure to specify what weapon he will use and what weapon is in his hands initially.

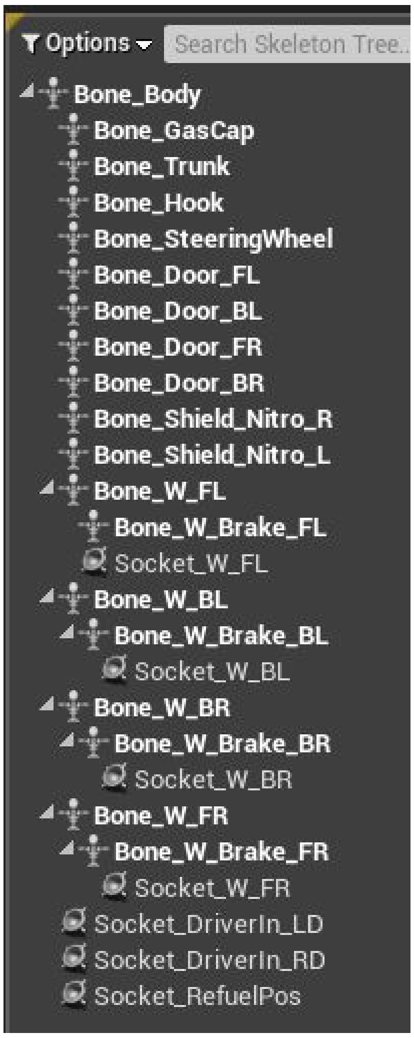
To add additional parameters, we recommend to use a child class of parent class "**EnemyCharacter_Pawn**". For example, a helmet is implemented for the enemy soldier to save him from a single headshot (*UseHelmet RandomHelmet?*). Randomized look is implemented for enemy type "Zombie".

Cars

There is a support for interaction with transportation means in the project.
Transportation means are implemented based on class "WheeledVehicle".
Transportation means contain different modules and can be fine-tuned depending on the gameplay.

How to add a new car

For the start, create a new car as it is done here "Vehicle Art Setup", but give bones names as they are shown in figure:

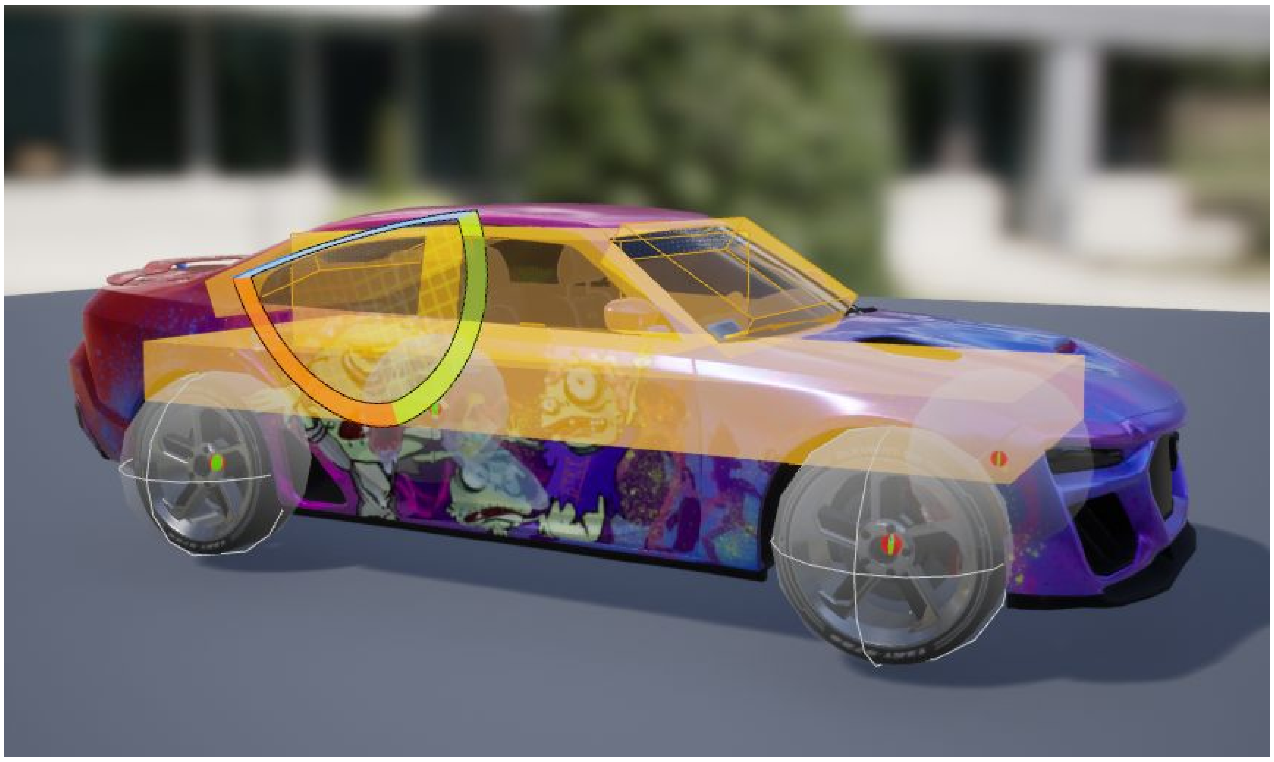


```
Bone_Body
  Bone_GasCap
  Bone_Trunk
  Bone_Hook
  Bone_SteeringWheel
  Bone_Door_FL
  Bone_Door_BL
  Bone_Door_FR
  Bone_Door_BR
  Bone_Shield_Nitro_R
  Bone_Shield_Nitro_L
  Bone_W_FL
    Bone_W_Brake_FL
    Socket_W_FL
  Bone_W_BL
    Bone_W_Brake_BL
      Socket_W_BL
  Bone_W_BR
    Bone_W_Brake_BR
      Socket_W_BR
  Bone_W_FR
    Bone_W_Brake_FR
      Socket_W_FR
  Socket_DriverIn_LD
  Socket_DriverIn_RD
  Socket_RefuelPos
```

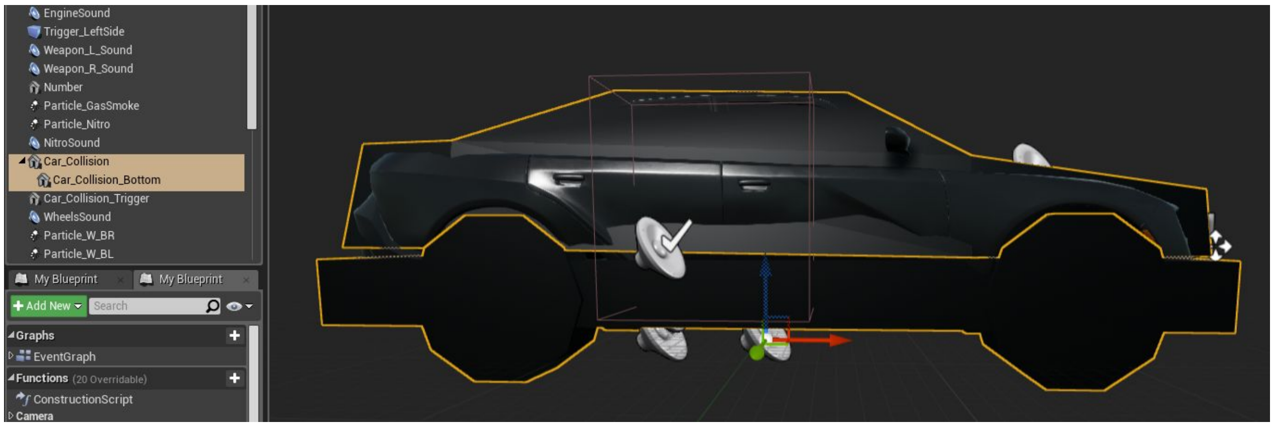
We recommend you to export the "SyferJet" car and use bones with needed orientations by positioning them in locations for wheels and other elements of your car.
These are bones that must have these names:
Bone_Body is a name of the body frame of the car.
*Bone_W_* are names of car's wheels.
Bone_W_Brake. Use the same name for brake pads (if there are any).
It does not matter how you name the rest of bones.

You must add these sockets:
Socket_W_.. are sockets for wheels.
Socket_DriverIn_LD is a place from which the character begins to open the left door.
Socket_DriverIn_RD is a place from which the character begins to open the right door.
Socket_RefuelPos is a place from where the character will refuel the car.

The next step is to add colliders in the physical asset (phat) for the car. Colliders must be rather simple in analogy with the existing car:



Also add 2 static meshes for collisions with the top and with the bottom as it is done in the existing car in the project:

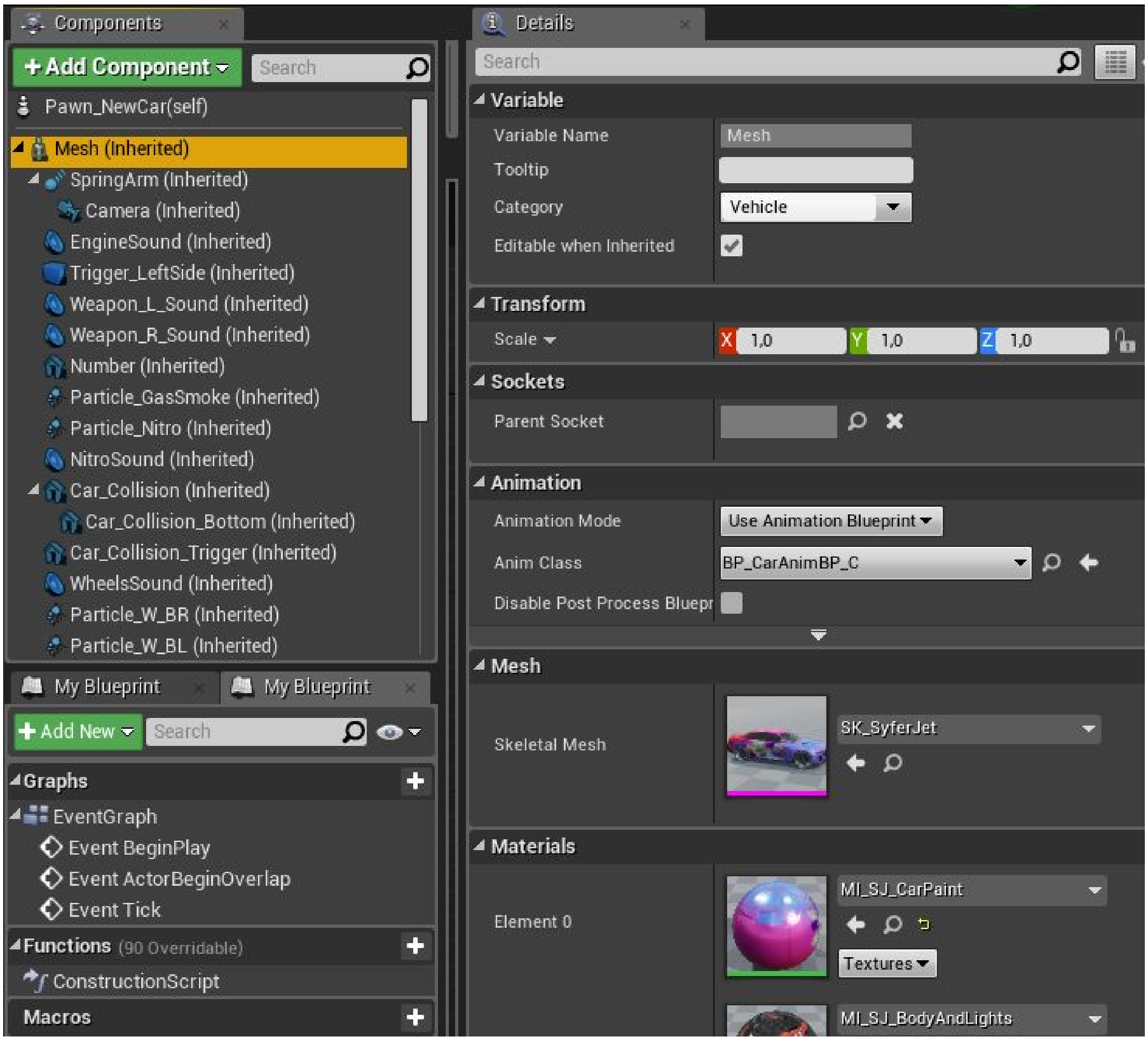


This is needed to make sure the car can slip through small bumps and curbs. The bottom collider is activated when the character steps out of the car.

Next, let’s add a new car and consider all possible settings. There are two transportation means implemented in the project: basic one and combat one. A combat vehicle is a child of the base car class and has advanced settings.

Create a child class of combat car class (**Pawn_Car_Combat**), so we can discuss all available settings by default. If you do not need the car to engage in combat, just use class “**Pawn_Car**” as a parent class. Give the child class name (in the example, it is

Pawn_NewCar) and open it. The first thing to do is to add your 3d model of the car by selecting **mesh** component:



Pawn_NewCar(self)
Mesh (Inherited)
SpringArm (Inherited)
Camera (Inherited)
EngineSound (Inherited)
Trigger_LeftSide (Inherited)
Weapon_L_Sound (Inherited)
Weapon_R_Sound (Inherited)
Number (Inherited)
Particle_GasSmoke (Inherited)
Particle_Nitro (Inherited)
NitroSound (Inherited)
Car_Collision (Inherited)
Car_Collision_Bottom (Inherited)
Car_Collision_Trigger (Inherited)
WheelsSound (Inherited)
Particle_W_BR (Inherited)
Particle_W_BL (Inherited)

Graphs
EventGraph
Event BeginPlay
Event ActorBeginOverlap
Event Tick
Functions (90 Overridable)
ConstructionScript
Macros

Variable

Variable Name	Mesh
Tooltip	
Category	Vehicle
Editable when Inherited	✓

Transform

Scale	X 1,0 Y 1,0 Z 1,0

Sockets

Parent Socket	

Animation

Animation Mode	Use Animation Blueprint
Anim Class	BP_CarAnimBP_C
Disable Post Process Bluepr	

Mesh

Skeletal Mesh	SK_SyferJet

Materials

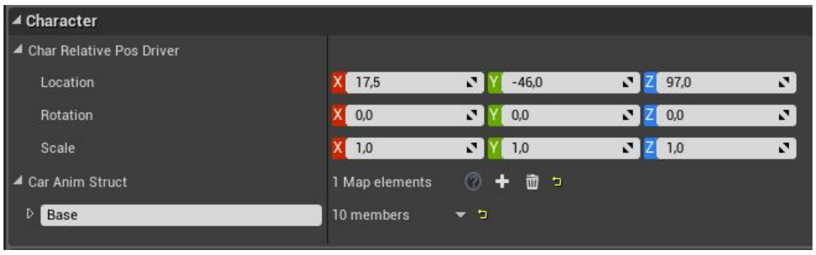
Element 0	MI_SJ_CarPaint
	Textures
	MI_SJ_BodyAndLights

Also, you need to retarget "**BP_CarAnimBP"** for the skeleton of your car and specify it in "**AnimClass**". Note, without **AnimBP** the car will not be animated and work correctly.

Car settings.

Technical parameters of the car can be specified in component "VehicleMovement". You may learn about the parameters here: [Vehicle]()

Open your new car and move to basic settings tab:



Character
Char Relative Pos Driver

	X	Y	Z
Location	17,5	-46,0	97,0
Rotation	0,0	0,0	0,0
Scale	1,0	1,0	1,0

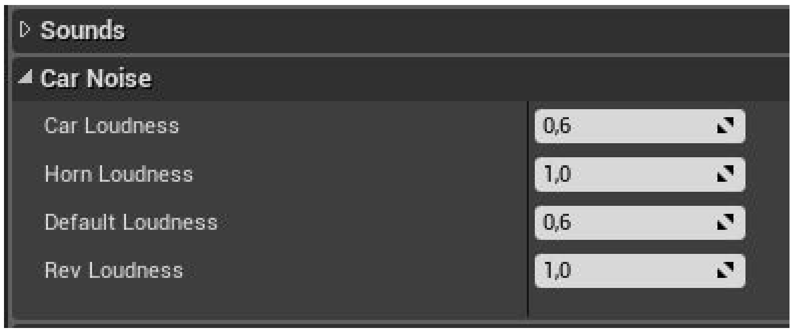
Car Anim Struct — 1 Map elements
Base — 10 members

At first, set up parameters for interaction with the car.

Char Relative Pos Driver is a driver's location relative to the car (driver seat)

Car Anim Struct is a data block where all animation of the car for the interaction with the character is specified.

The next step is to specify all sounds for the car as well as the noise caused by it:

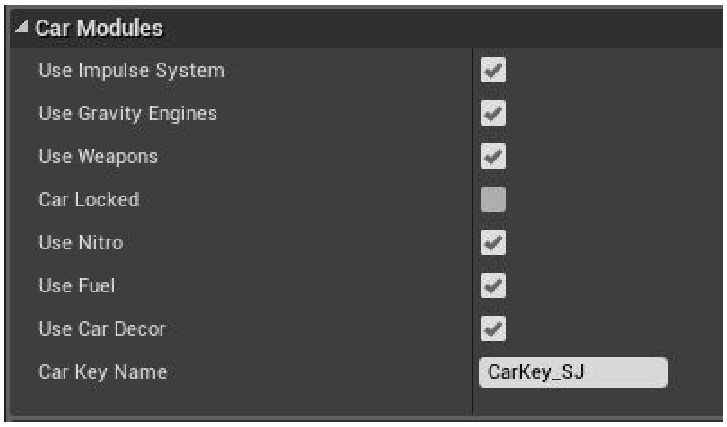


Car Noise

Car Loudness	0,6
Horn Loudness	1,0
Default Loudness	0,6
Rev Loudness	1,0

Car modules

Let’s discuss all available modules for the car. For that, expand the category “Car Modules”:

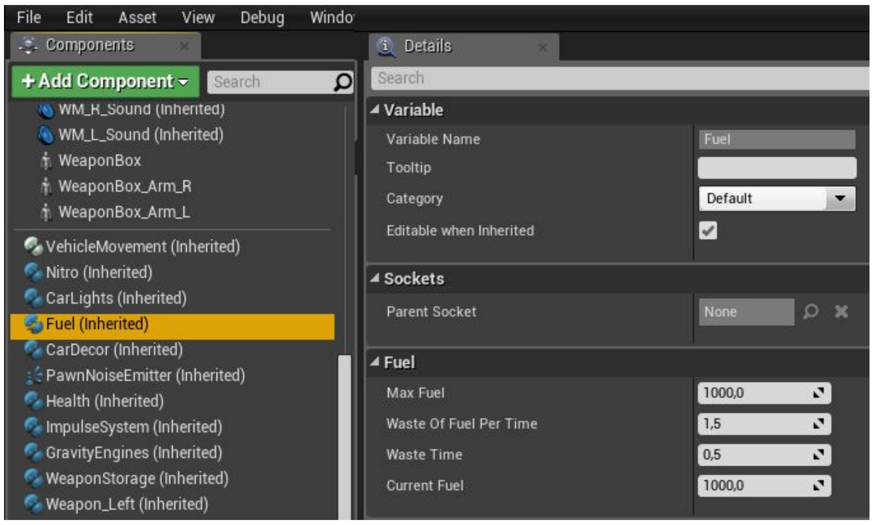


Car Modules

Use Impulse System	✓
Use Gravity Engines	✓
Use Weapons	✓
Car Locked	☐
Use Nitro	✓
Use Fuel	✓
Use Car Decor	✓
Car Key Name	CarKey_SJ

For the start, let us consider just basic modules:
CarLocked is whether the car will be locked. This parameter can be dynamically modified.
Car Key Name is a key that can open this car. Just create an item (pickup item) with the specified name, and, if it is in the inventory, the car can be unlocked.

UseFuel.If the module is enabled, the car will consume fuel. When the fuel tank is empty, the car will stall. The fuel settings are in component “Fuel”.



WM_R_Sound (Inherited)
WM_L_Sound (Inherited)
WeaponBox
WeaponBox_Arm_R
WeaponBox_Arm_L
VehicleMovement (Inherited)
Nitro (Inherited)
CarLights (Inherited)
Fuel (Inherited)
CarDecor (Inherited)
PawnNoiseEmitter (Inherited)
Health (Inherited)
ImpulseSystem (Inherited)
GravityEngines (Inherited)
WeaponStorage (Inherited)
Weapon_Left (Inherited)

Variable

Variable Name	Fuel
Tooltip	
Category	Default
Editable when Inherited	✓

Sockets

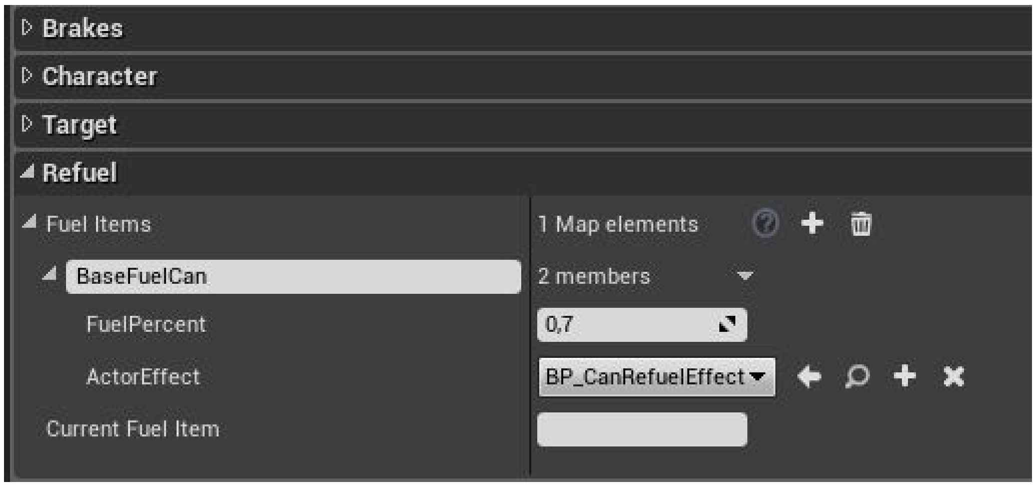
Parent Socket	None

Fuel

Max Fuel	1000,0
Waste Of Fuel Per Time	1,5
Waste Time	0,5
Current Fuel	1000,0

Max Fuel is the maximum amount of fuel.
Waste of Duel Per Time is the fuel consumption rate.
Waste Time is a time interval after which the fuel is consumed.
Current Fuel is the current amount of fuel in the tank.

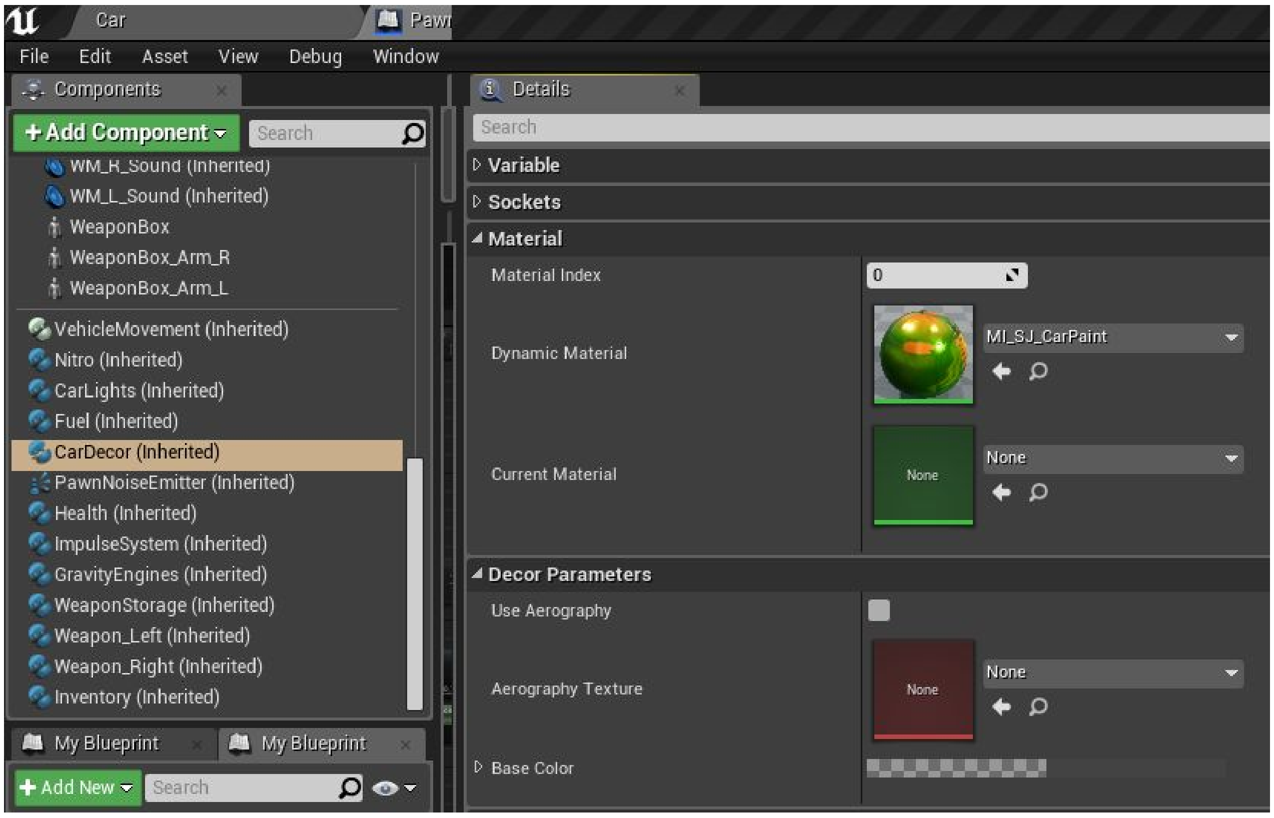
Also, you can create various items to refuel your car. They are specified in basic settings in category "Refuel":


Brakes
Character
Target
Refuel

Fuel Items	1 Map elements
BaseFuelCan	2 members
FuelPercent	0,7
ActorEffect	BP_CanRefuelEffect
Current Fuel Item	

Fuel Items are items that can refuel this car. The item name must coincide with the name of the item in the inventory.
FuelPercent is an amount in percent of fuel this items restores in the fuel tank.
ActorEffect is a class that contains various effects for refueling. By default, it has an animated fuel canister with sounds and physics.

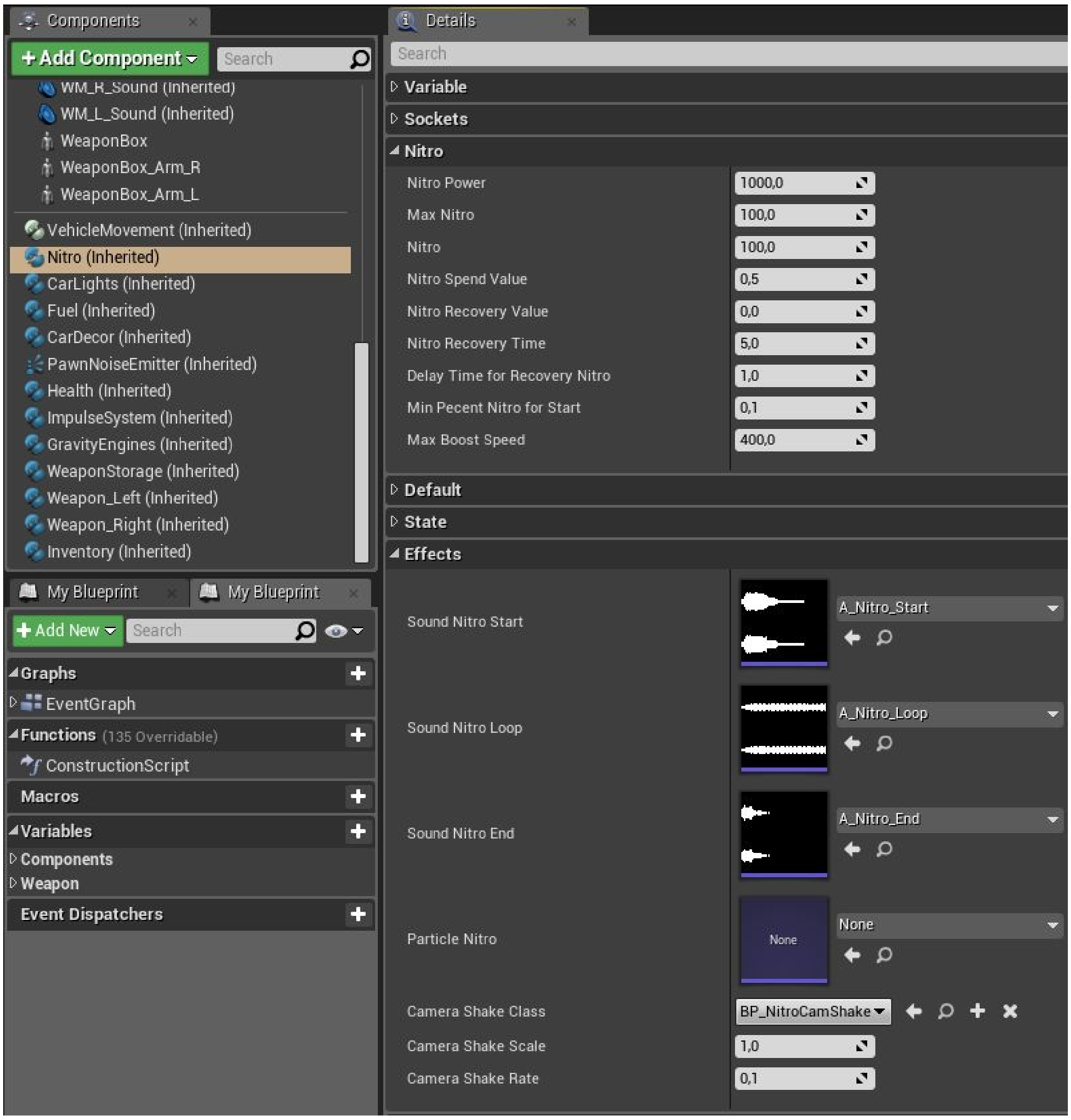
Use Car Decor is a module for car's appearance. It comprises settings for aerography and car's paint color. The settings are specified in component "CarDecor":



Material Index	0
Dynamic Material	MI_SJ_CarPaint
Current Material	None
Use Aerography	
Aerography Texture	None
Base Color	

Material Index is a material index on your car responsible for the car's paint color.
Dynamic Material is a material that contains the paint of the car.
Use Aerography. Whether to use aerography.
Aerography Texture is a texture for the aeropraphy. By default, several textures have been added for the aerography as examples.
Base Color is a color of the car. It is used when the aerography parameter is disabled.

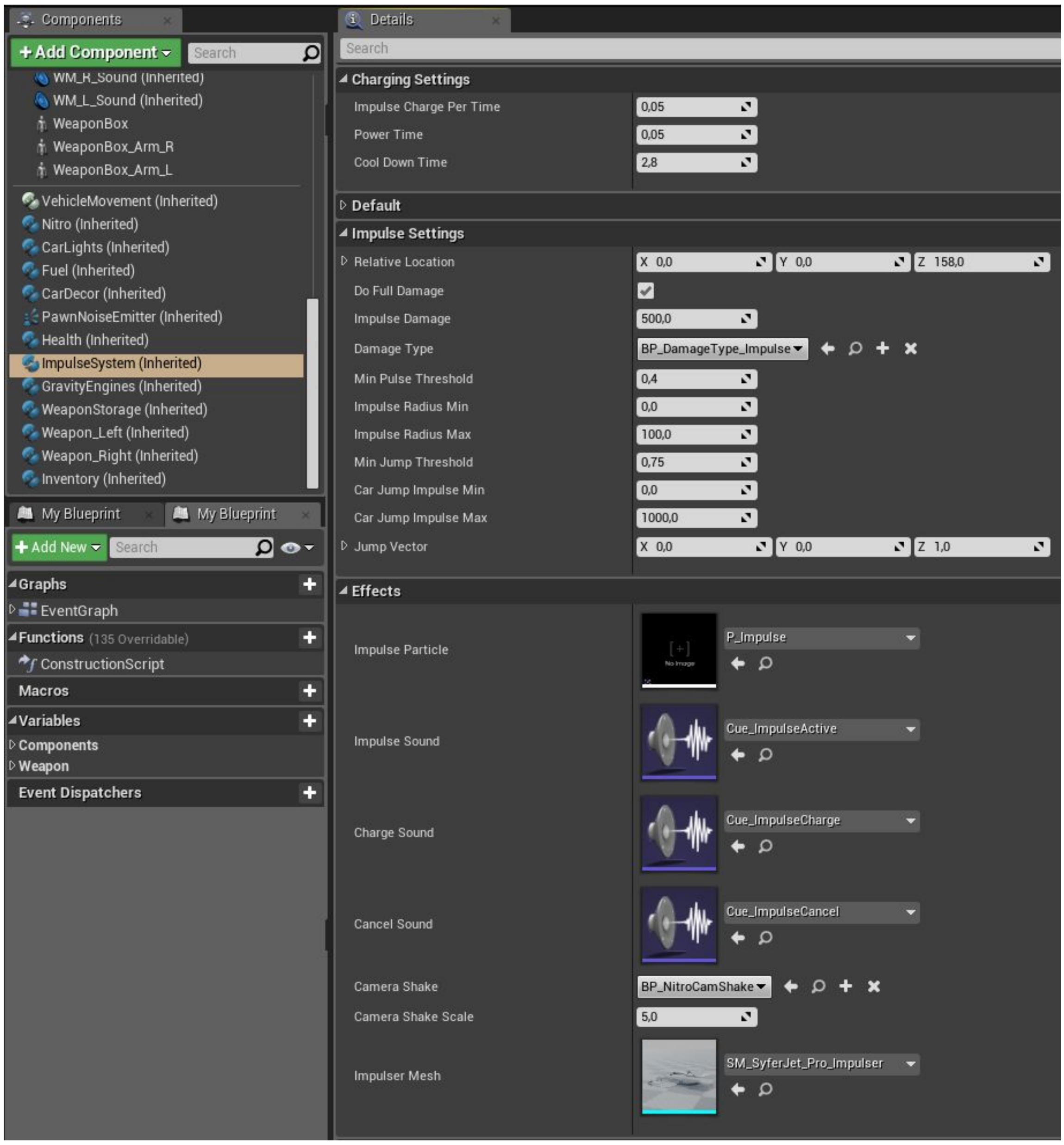
Use Nitro. This module is for boost of the car. All settings are in component “**Nitro**”:



Nitro Power	1000,0
Max Nitro	100,0
Nitro	100,0
Nitro Spend Value	0,5
Nitro Recovery Value	0,0
Nitro Recovery Time	5,0
Delay Time for Recovery Nitro	1,0
Min Pecent Nitro for Start	0,1
Max Boost Speed	400,0
Sound Nitro Start	A_Nitro_Start
Sound Nitro Loop	A_Nitro_Loop
Sound Nitro End	A_Nitro_End
Particle Nitro	None
Camera Shake Class	BP_NitroCamShake
Camera Shake Scale	1,0
Camera Shake Rate	0,1

Nitro Power is a boost power.
Max Nitro is the maximum amount of nitro.
Nitro is the current amount of nitro.
Nitro Spend Value is the nitro consumption rate every 0.05 second.
Nitro Recovery Value is an amount of nitro that gets replenished (when 0, it replenishes the nitro fully)
Nitro Recovery Time is a replenish time for nitro.
Delay Time For Recovery Nitro is a time delay before the replenish begins.
Min Percent Nitro for Start is the minimum amount of nitro needed in order for the boost to work.
Max Boost Speed is the maximum boost speed in km/h.
All effects for Nitro boost are specified in category **Effects**.

Use Impulse System is a combat module responsible for the impulse system. It can be tuned in component “**ImpulseSystem**”:



WM_R_Sound (Inherited)
WM_L_Sound (Inherited)
WeaponBox
WeaponBox_Arm_R
WeaponBox_Arm_L
VehicleMovement (Inherited)
Nitro (Inherited)
CarLights (Inherited)
Fuel (Inherited)
CarDecor (Inherited)
PawnNoiseEmitter (Inherited)
Health (Inherited)
ImpulseSystem (Inherited)
GravityEngines (Inherited)
WeaponStorage (Inherited)
Weapon_Left (Inherited)
Weapon_Right (Inherited)
Inventory (Inherited)

Charging Settings

Impulse Charge Per Time	0,05
Power Time	0,05
Cool Down Time	2,8

Default

Impulse Settings

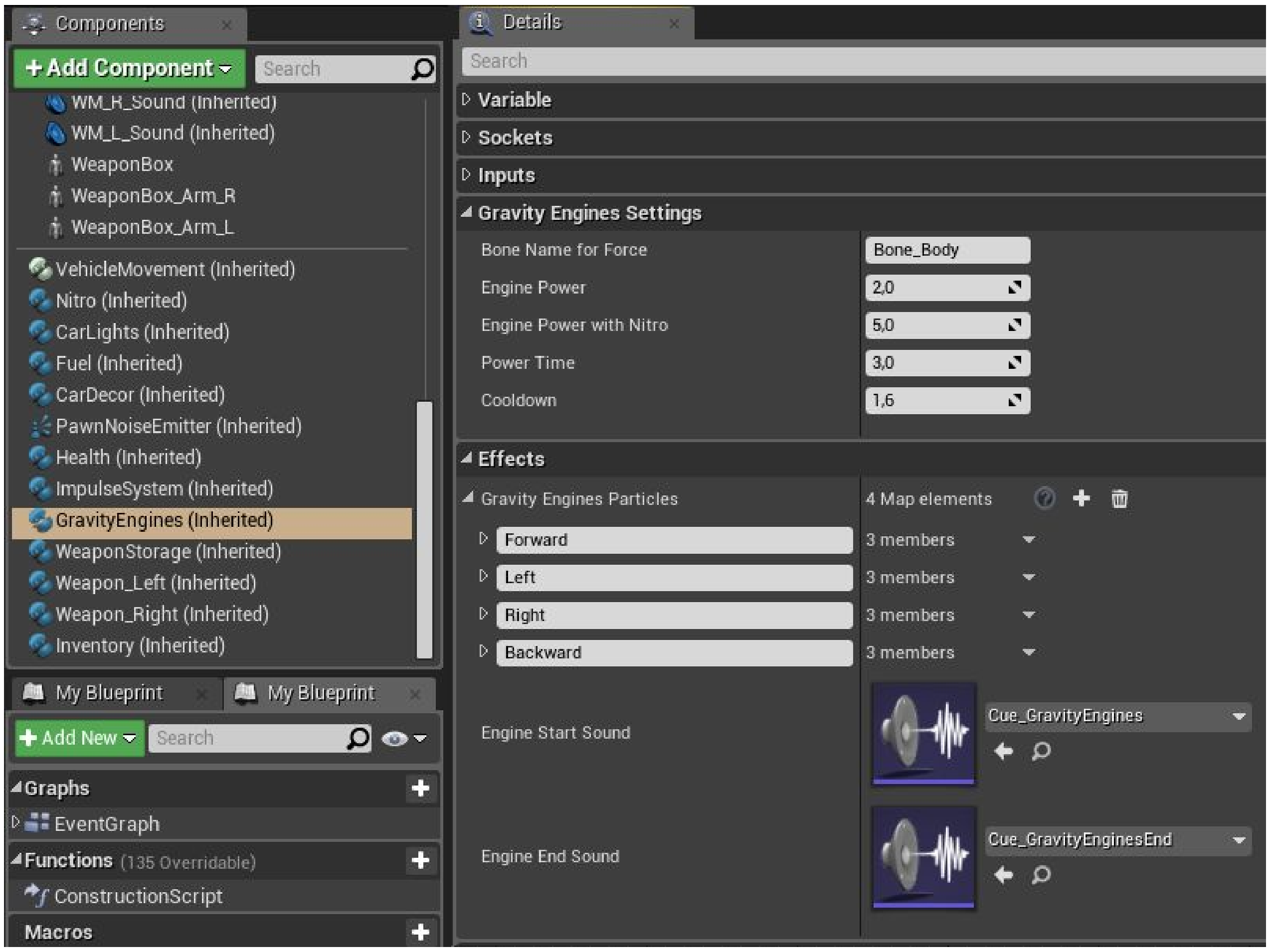
Relative Location	X 0,0 Y 0,0 Z 158,0
Do Full Damage	✓
Impulse Damage	500,0
Damage Type	BP_DamageType_Impulse
Min Pulse Threshold	0,4
Impulse Radius Min	0,0
Impulse Radius Max	100,0
Min Jump Threshold	0,75
Car Jump Impulse Min	0,0
Car Jump Impulse Max	1000,0
Jump Vector	X 0,0 Y 0,0 Z 1,0

Effects

Impulse Particle	P_Impulse
Impulse Sound	Cue_ImpulseActive
Charge Sound	Cue_ImpulseCharge
Cancel Sound	Cue_ImpulseCancel
Camera Shake	BP_NitroCamShake
Camera Shake Scale	5,0
Impulser Mesh	SM_SyferJet_Pro_Impulser

Charging Settings (Settings for the recharging of the impulse):
Impulse Charge Per Time is an amount of charge per an interval of time.
Power Time is a time-interval after which the impulse gets recharged.
Cool Down Time is a cooldown time for the system.
Impulse Settings:
Relative Location is a location for the impulse relative to the car.
Do Full Damage whether to deal damage fully within the radius of effect.
Impulse Damage is the damage value caused by the impulse.
Damage Type is a type of damage (in the class, the force of physical impulse is specified as well).
Min Pulse Threshold is an amount of charge in percent needed for the impulse to work.
Impulse Radius min is the minimum radius of the impulse relative to the charge.
Impulse Radius Max is the maximum radius of the impulse relative to the charge.
Min Jump Threshold is an amount of charge in percent needed for jump to work.
Car Jump Impulse Min is the minimum jump strength relative to the charge.
Car Jump Impulse Max is the maximum jump strength relative to the charge.
Jump Vector is a direction of the jump.
In category “Effects”, all effects for the impulse system including its look can be specified.

Use Gravity Engines is a module for gravitational drivers needed to rotate the car in air. By default, these drivers are activated by pressing the hand break in air. The rotation can be amplified by activating nitro boost. All settings are in component "**Gravity Engines**":



Gravity Engines Settings

Bone Name for Force	Bone_Body
Engine Power	2,0
Engine Power with Nitro	5,0
Power Time	3,0
Cooldown	1,6

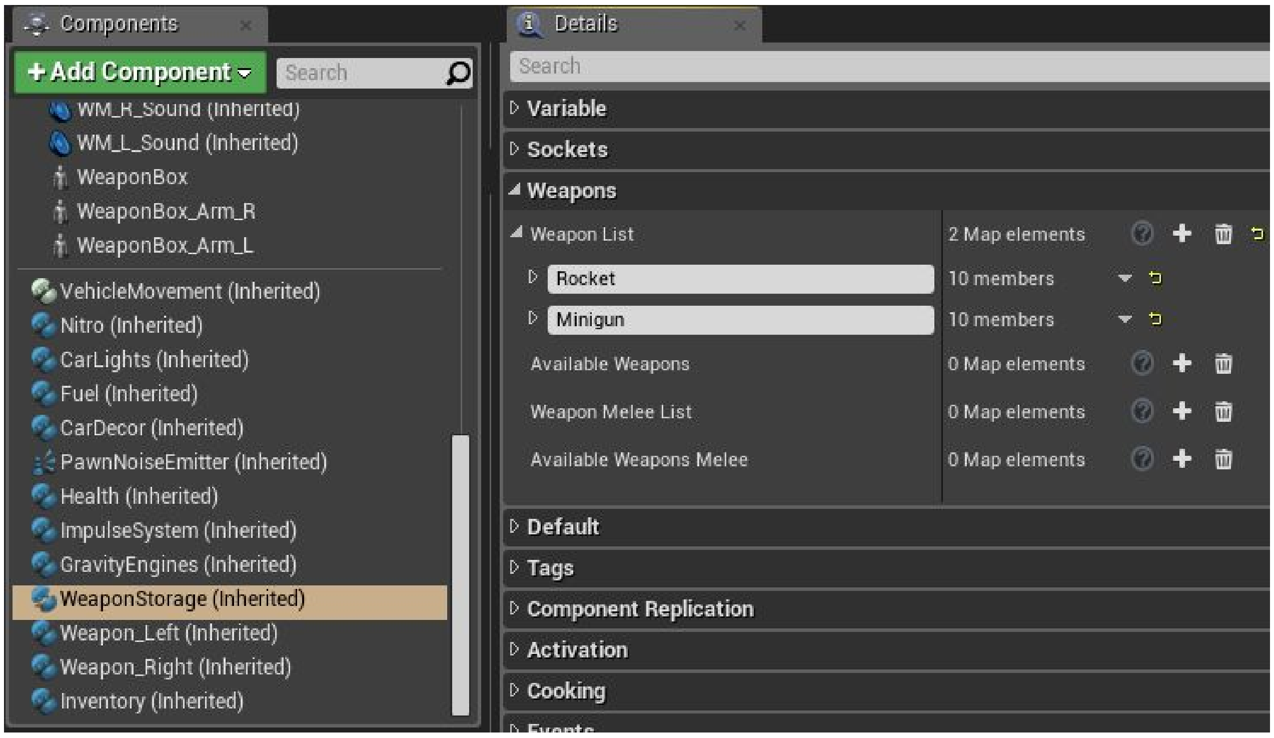
Effects

Forward	3 members
Left	3 members
Right	3 members
Backward	3 members
Engine Start Sound	Cue_GravityEngines
Engine End Sound	Cue_GravityEnginesEnd

<u>Gravity Engines Settings</u> (Basic settings for drivers):
Bone Name for Force is a name of the bone which will be affected by the drivers.
Engine Power is a power of the drivers.
Engine Power with Nitro is a power of drivers when nitro is used.
Power Time is a working time of drivers before they overheat.
Cooldown is a cooldown time.

Drivers' particles, their locations, and their sounds are specified in category "*Effects*".

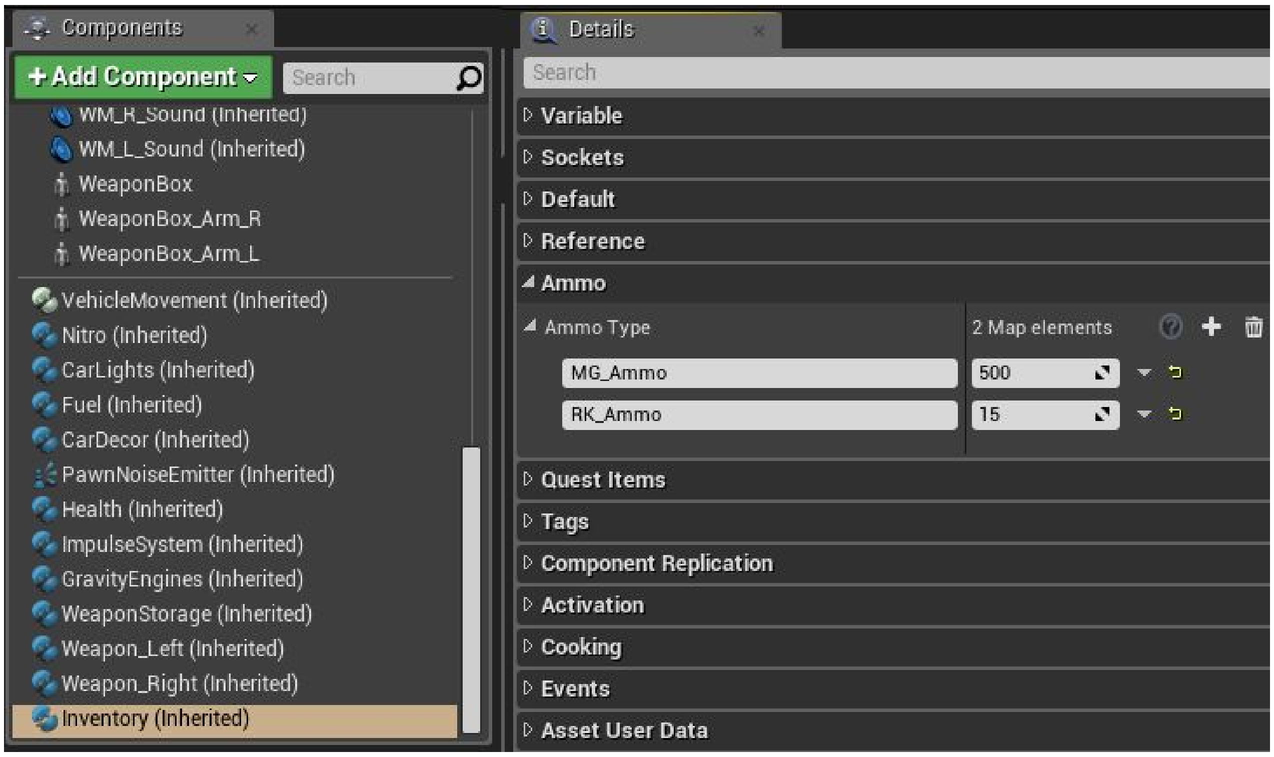
Use Weapons is a combat module for the weaponry. By default, combat vehicles can use two weapons at the same time: the left one and the right one. The weapon is based on the same logic and character's weapon and it uses same components: "Weapon Storage", "Weapon", "Inventory" (for ammo).
First step is to move to component "Weapon Storage" and set up all available weapon for this transportation mean. By default, there are two weapon types available: a mini-gun and a rocket-launcher.



```
WM_R_Sound (Inherited)
WM_L_Sound (Inherited)
WeaponBox
WeaponBox_Arm_R
WeaponBox_Arm_L
VehicleMovement (Inherited)
Nitro (Inherited)
CarLights (Inherited)
Fuel (Inherited)
CarDecor (Inherited)
PawnNoiseEmitter (Inherited)
Health (Inherited)
ImpulseSystem (Inherited)
GravityEngines (Inherited)
WeaponStorage (Inherited)
Weapon_Left (Inherited)
Weapon_Right (Inherited)
Inventory (Inherited)
```

```
Variable
Sockets
Weapons
  Weapon List                 2 Map elements
    Rocket                    10 members
    Minigun                   10 members
  Available Weapons           0 Map elements
  Weapon Melee List           0 Map elements
  Available Weapons Melee     0 Map elements
Default
Tags
Component Replication
Activation
Cooking
Events
```

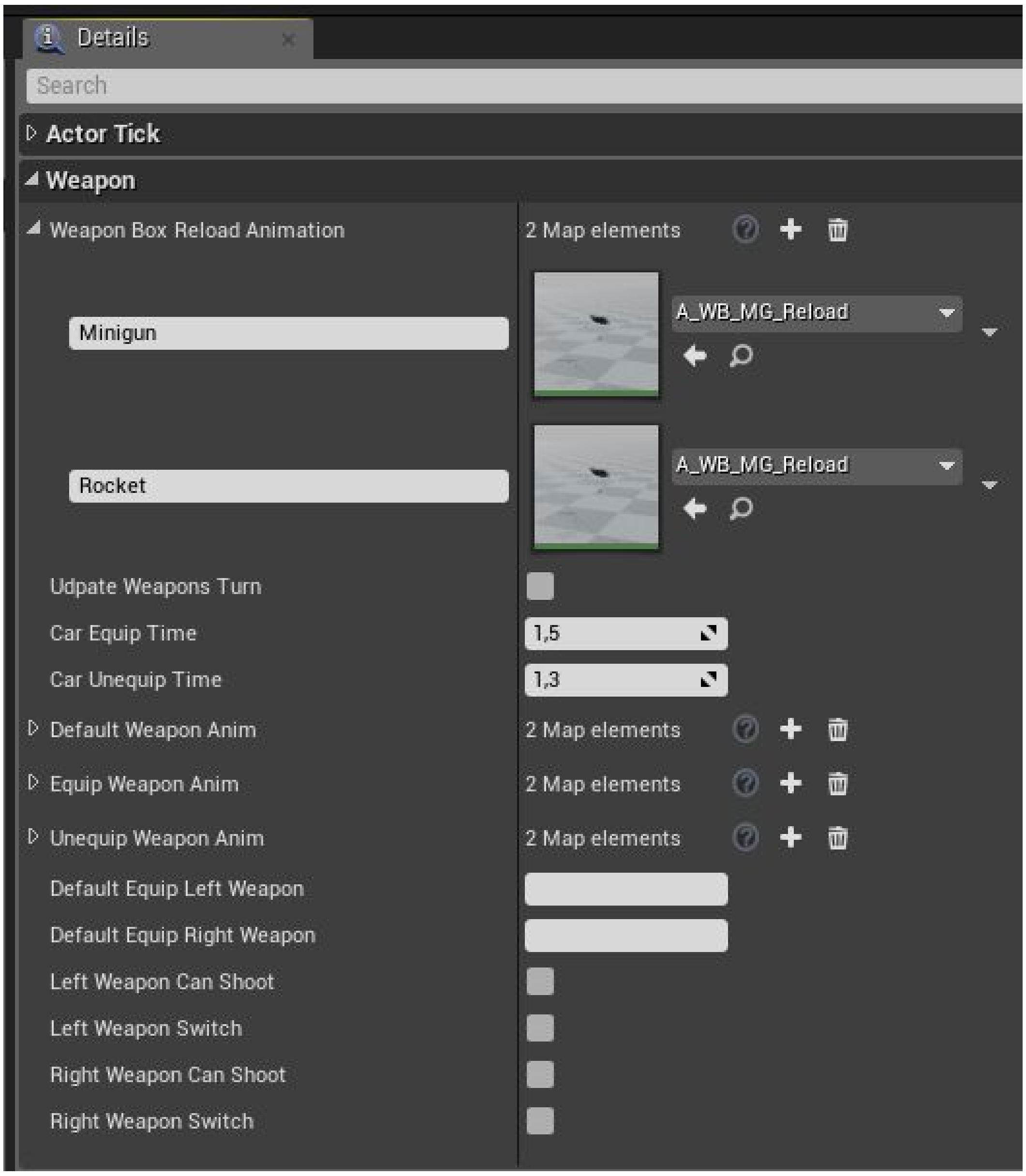
After the weapons are added, add necessary ammo types in component Inventory:



```
WM_R_Sound (Inherited)
WM_L_Sound (Inherited)
WeaponBox
WeaponBox_Arm_R
WeaponBox_Arm_L
VehicleMovement (Inherited)
Nitro (Inherited)
CarLights (Inherited)
Fuel (Inherited)
CarDecor (Inherited)
PawnNoiseEmitter (Inherited)
Health (Inherited)
ImpulseSystem (Inherited)
GravityEngines (Inherited)
WeaponStorage (Inherited)
Weapon_Left (Inherited)
Weapon_Right (Inherited)
Inventory (Inherited)
```

```
Variable
Sockets
Default
Reference
Ammo
  Ammo Type                   2 Map elements
    MG_Ammo                   500
    RK_Ammo                   15
Quest Items
Tags
Component Replication
Activation
Cooking
Events
Asset User Data
```

The following step is to place created weapons on the car. This can be done by specifying default names of weaponry in category Weapon in parameters *Default Equip Left Weapon* and *Default Equip Right Weapon.* Additional animations can be specified in this category as well, for instance, a reloading animation for a flexible ammo belt and such:


Details

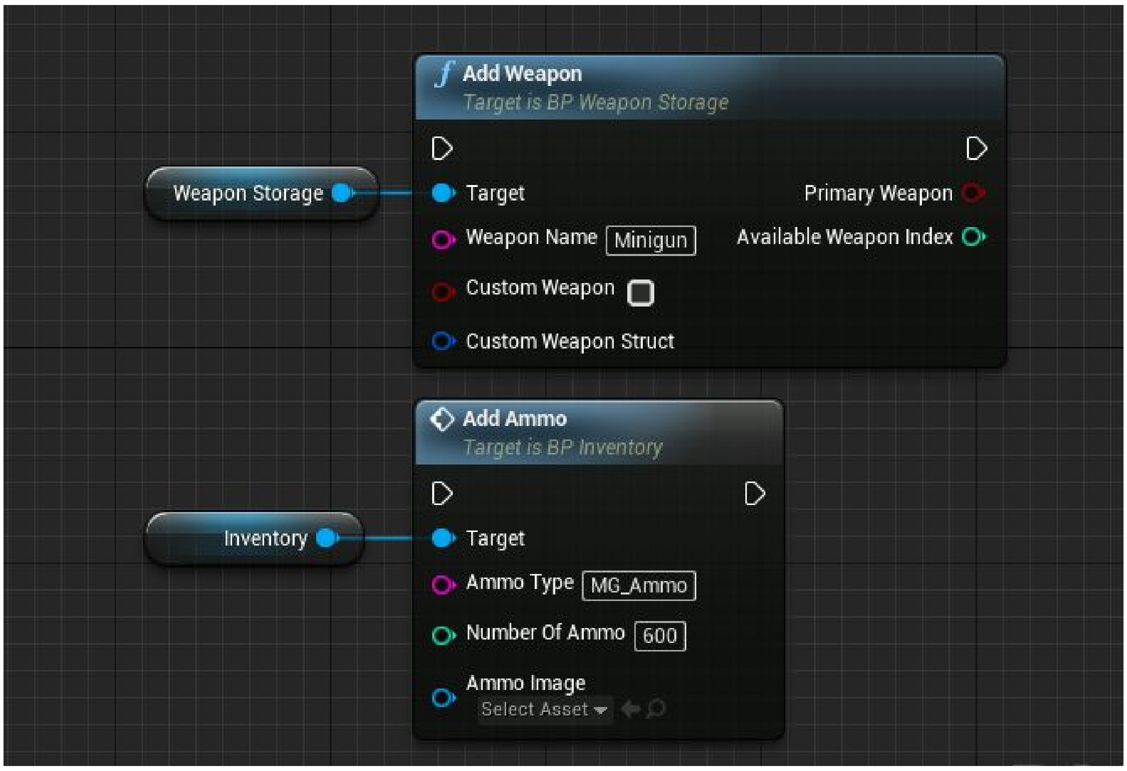
Search

Actor Tick

Weapon

Weapon Box Reload Animation	2 Map elements
Minigun	A_WB_MG_Reload
Rocket	A_WB_MG_Reload
Udpate Weapons Turn	☐
Car Equip Time	1,5
Car Unequip Time	1,3
Default Weapon Anim	2 Map elements
Equip Weapon Anim	2 Map elements
Unequip Weapon Anim	2 Map elements
Default Equip Left Weapon	
Default Equip Right Weapon	
Left Weapon Can Shoot	☐
Left Weapon Switch	☐
Right Weapon Can Shoot	☐
Right Weapon Switch	☐

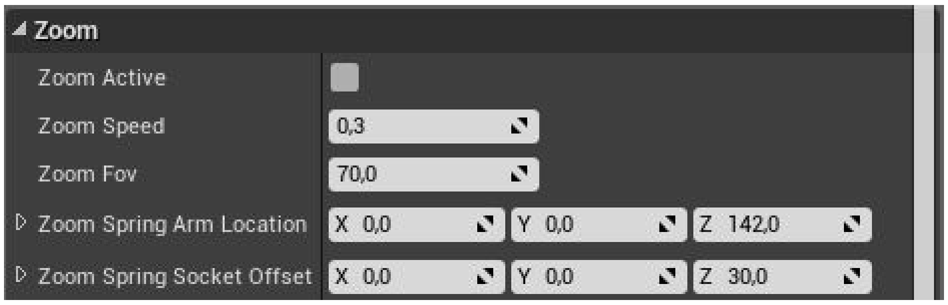
This is the second way to add weapons and ammo. Use following nodes to add weapons and ammo:


Add Weapon
Target is BP Weapon Storage
Weapon Storage → Target
Weapon Name: Minigun
Custom Weapon: ☐
Custom Weapon Struct
Primary Weapon
Available Weapon Index

Add Ammo
Target is BP Inventory
Inventory → Target
Ammo Type: MG_Ammo
Number Of Ammo: 600
Ammo Image: Select Asset

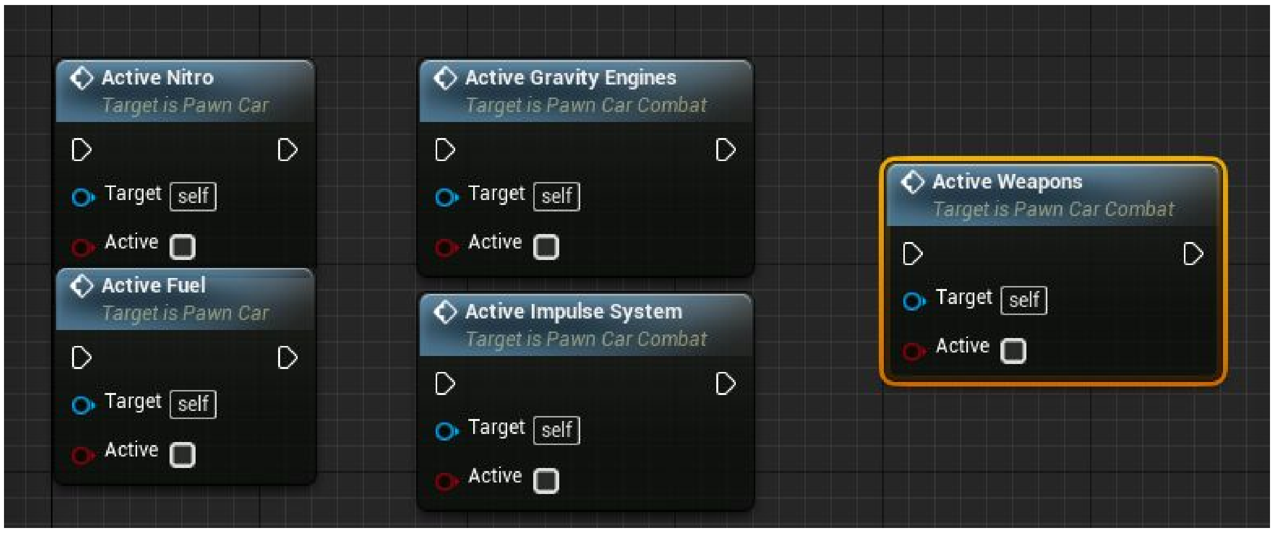
Note, the car uses all added weapons and switches between them upon pressing buttons 1 and 2. In other words, if you wish to add two mini-guns on the car, you need to add two weapons with the name Minigun. By default, in order to drop a weapon, press and hold 1 or 2.

Combat vehicles support parameters for zoom (Zooming). It is activated when the weapon is put forward.
Move to basic settings in category Zoom:


Zoom
Zoom Active
Zoom Speed 0,3
Zoom Fov 70,0
Zoom Spring Arm Location X 0,0 Y 0,0 Z 142,0
Zoom Spring Socket Offset X 0,0 Y 0,0 Z 30,0

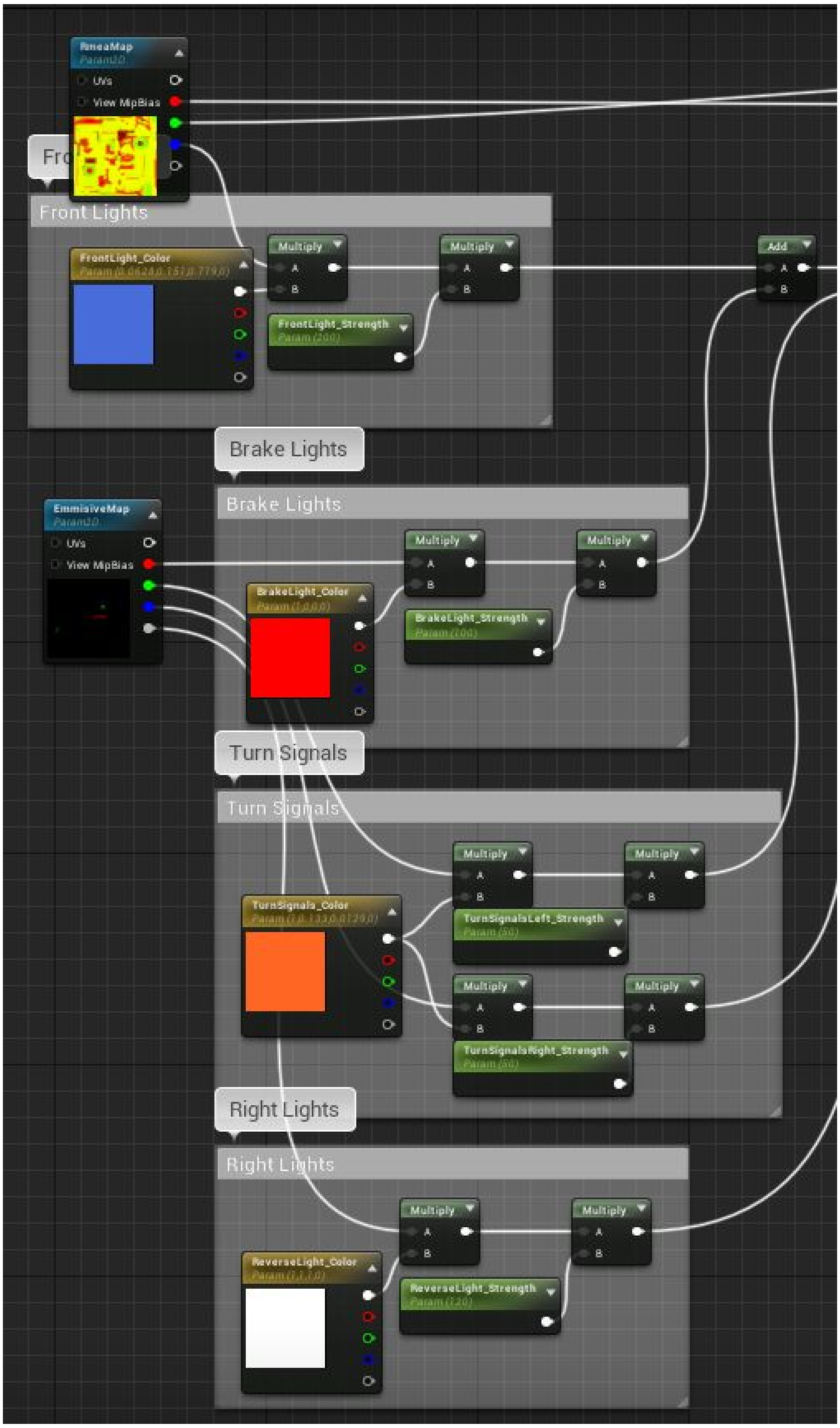
Zoom Speed is a zoom speed in seconds.
Zoom Fov is a field-of-view when zooming in.
Zoom Spring Arm Location is a location of the spring arm when zooming in.
Zoom Spring Socket Offset is a camera position relative to the spring arm when zooming in.

You can activate/deactivate modules in real time with the help of the function in category “CarModules”:



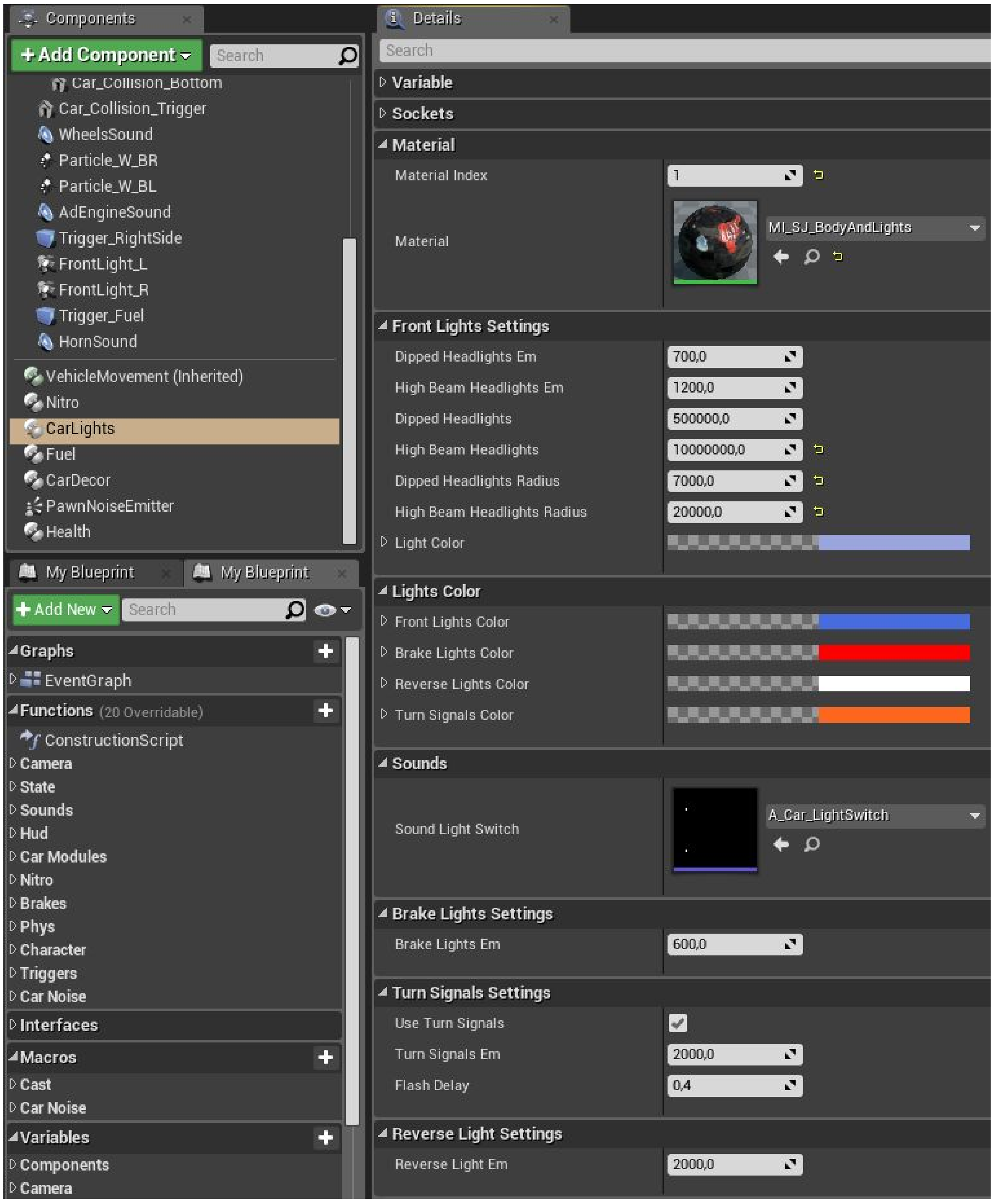
Component **CarLights.** The component is needed to set up car headlights and signals (reverse gear, turn signals, brake). The first thing is to prepare textures (masks) for your car. The masks for headlights must be in the following textures and channels:
1. The mask for the front headlights must be in the blue channel of the material mask (RMEA).
A separate texture is devised for other signals. (By default, it is T_Corpus_Emissive_M).
2. The mask for the break signal must be in the red channel.
3. The mask for the left turn signal must be in the green channel.
4. The mask for the right turn signal must be in the blue channel.
5. The mask for the reverse gear signal must be in the alpha channel.

You can open material “**M_SJ_Body**” and check all the channels there:



The front headlights of the car support 3 regimes of lighting:
Turned off
Low beam (dipped headlights)
High beam

Proceed to the car signal settings. Select component “Car lights”.



Components

+Add Component

Car_Collision_Bottom
Car_Collision_Trigger
WheelsSound
Particle_W_BR
Particle_W_BL
AdEngineSound
Trigger_RightSide
FrontLight_L
FrontLight_R
Trigger_Fuel
HornSound
VehicleMovement (Inherited)
Nitro
CarLights
Fuel
CarDecor
PawnNoiseEmitter
Health

My Blueprint

+Add New

Graphs
EventGraph
Functions (20 Overridable)
ConstructionScript
Camera
State
Sounds
Hud
Car Modules
Nitro
Brakes
Phys
Character
Triggers
Car Noise
Interfaces
Macros
Cast
Car Noise
Variables
Components
Camera

Details

Variable

Sockets

Material

Material Index	1
Material	MI_SJ_BodyAndLights

Front Lights Settings

Dipped Headlights Em	700,0
High Beam Headlights Em	1200,0
Dipped Headlights	500000,0
High Beam Headlights	10000000,0
Dipped Headlights Radius	7000,0
High Beam Headlights Radius	20000,0
Light Color	

Lights Color

Front Lights Color	🟦
Brake Lights Color	🟥
Reverse Lights Color	⬜
Turn Signals Color	🟧

Sounds

Sound Light Switch	A_Car_LightSwitch

Brake Lights Settings

Brake Lights Em	600,0

Turn Signals Settings

Use Turn Signals	✓
Turn Signals Em	2000,0
Flash Delay	0,4

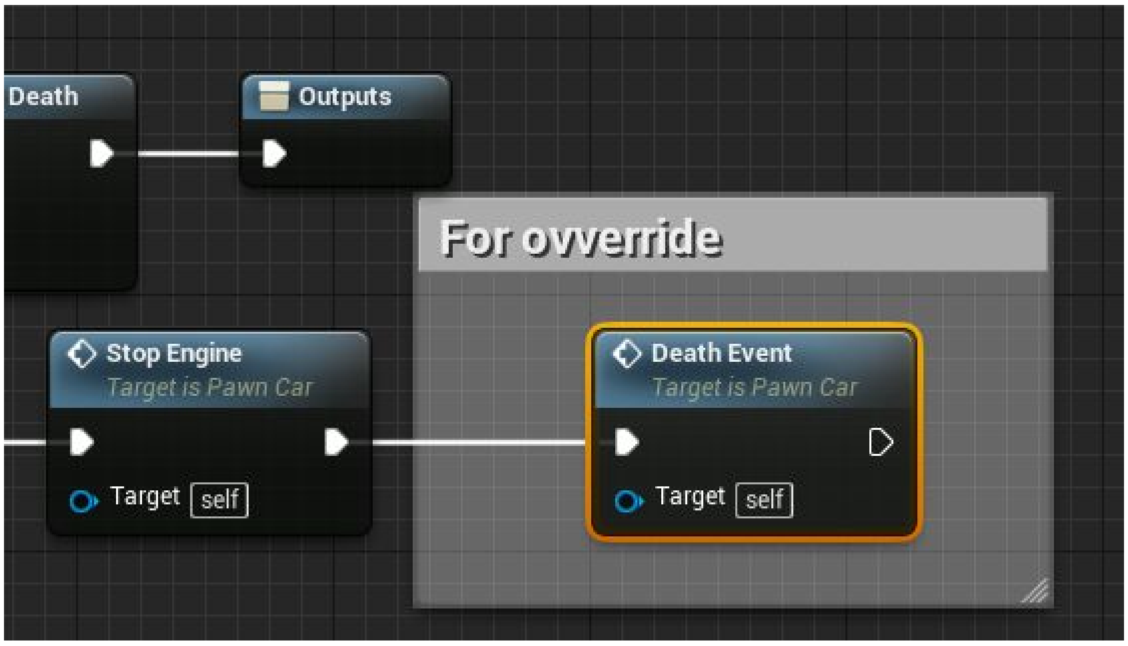
Reverse Light Settings

Reverse Light Em	2000,0

Choose your material in category “**Material**” and its reference index to models.
Front headlights are adjusted in category “**Front Lights Settings**”.
Dipped Headlights Em is the glowing strength of dipped headlights.
High Beam Headlights Em is the glowing strength of high beam headlights.
Dipped Headlights is the brightness of dipped headlights.
High Beam Headlights is the brightness of high beam headlights.
Dipped Headlights Radius is the light range for dipped headlights.
High Beam Headlights Radius is the light range for high mean headlights.
Light Color is a color of headlights lighting.

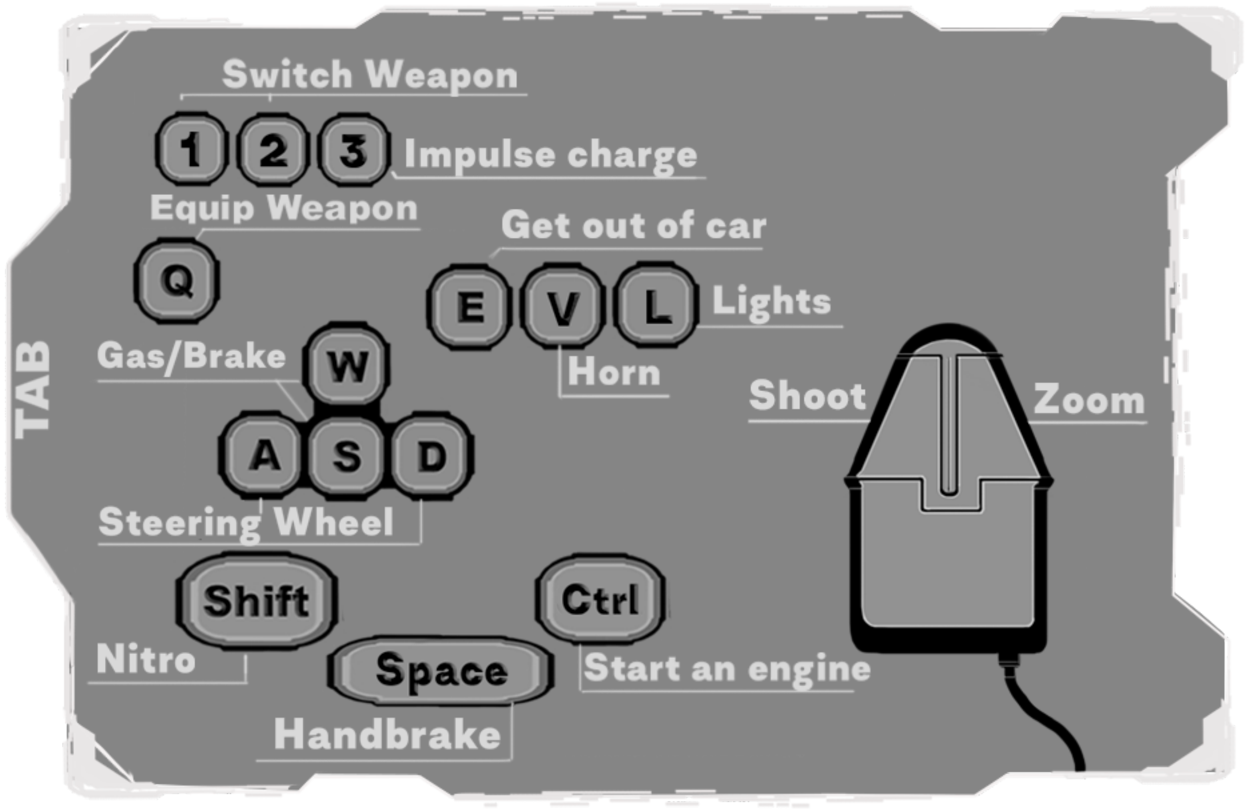
The category “**Lights Color**” is to adjust the lighting of other signals.
The category “**Sounds**” is to specify the sound to switch between headlights regimes.
The category “**Brake Lights Em**” is to adjust the lighting strength of braking signal.
The category “**Turn Signals Settings**” is to adjust signals for turn lights.
The category “**ReverseLightSett**” is to adjust the lighting strength of reverse gear.

Additionally, the car contains "**Health**" component by default. When the health is depleted, the car stalls. A special function "**DeathEvent**" has been added to be overridden. When you override the function, you can add any death events for the car, for instance, an explosion.



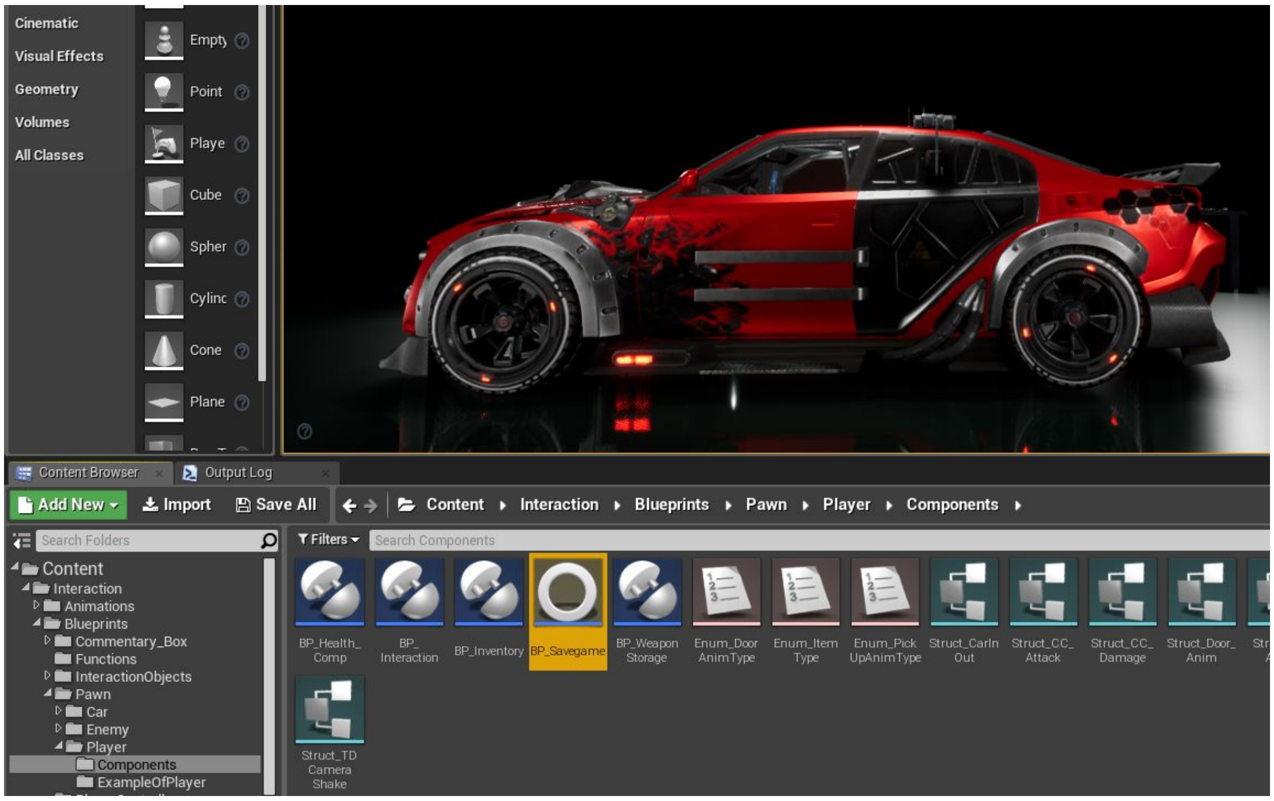
An ignition key button (ctrl) has been envisaged in the car. This can be useful when one needs to save the gasoline or in order to make sure the car does not attract enemies with noise.

Basic car controls:

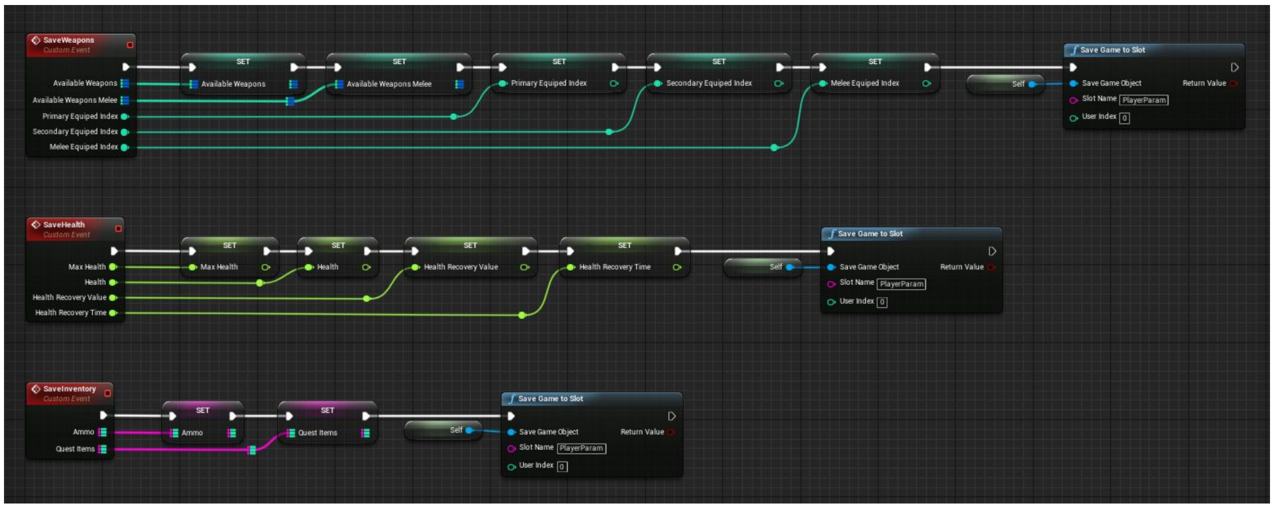


Save and load the game.

Since there were many questions about the saving of the character. The project adds a class of saving the game, which preserves the weapon, inventory and health of the character.

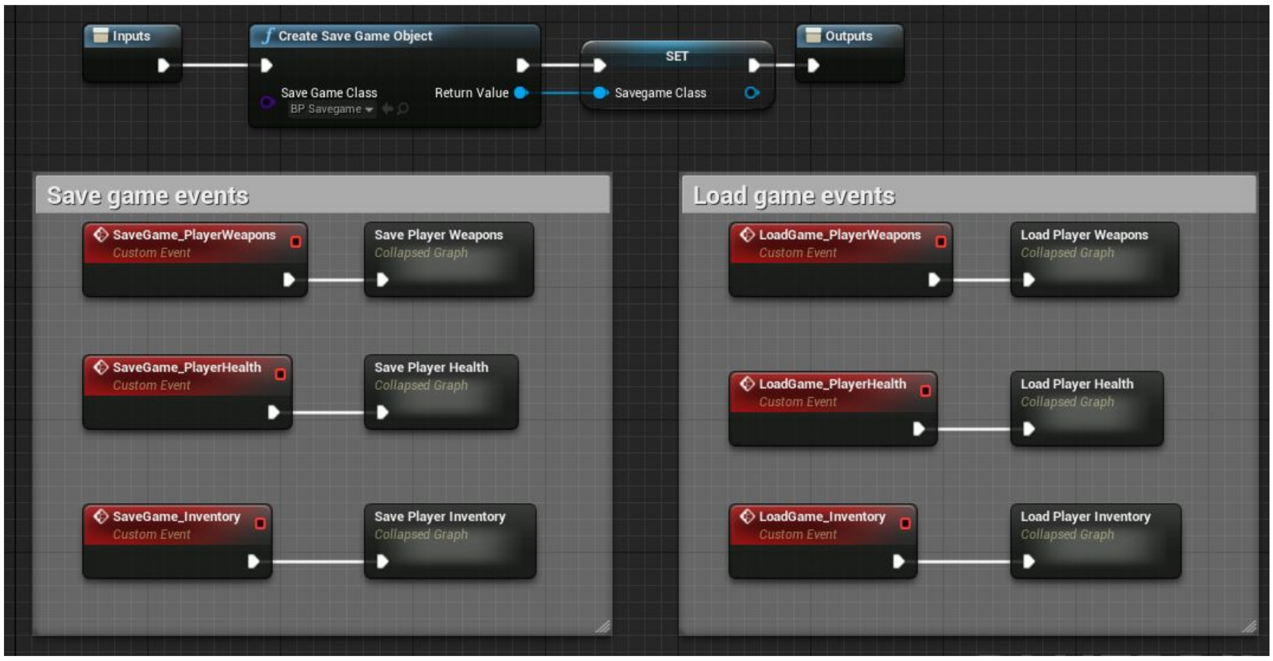


This class is used to save variables in more detail here - SaveGame.
This class uses the save slot "**PlayerParam**", in this slot the player's parameters are saved.



In the player's pawn, there are events in the "**SaveGameComp**" block. The save events refer to the class "**BP_Savegame**" and save the current player parameters. The load events receive the saved data from the "BP_Savegame" class and apply them to the character. On the example of weapons - all weapons are reset and all stored weapons and data are added to "**BP_Savegame**".

For the test, 6 and 7 are added in this block: 6 - saving, 7 - loading.



With the help of this class, you can expand the saving system for example by preserving the position of the player, the surrounding enemies, etc. To make it easier to navigate around the code, there are a lot of comments.

Online Support Channel - Discord channel.
Thank you for attention.
With best regard Andrew (ZzGERTzZ).