UNREAL ENGINE

LECTURE 4

UE4 Gameplay Framework Classes

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Identify the Gameplay Framework classes
- Present the Game Mode class and its default classes
- Explain the Player Controller and Player State classes
- Present the Pawn class and some subclasses
- Introduce the Game Instance and Game Session classes

Outcomes

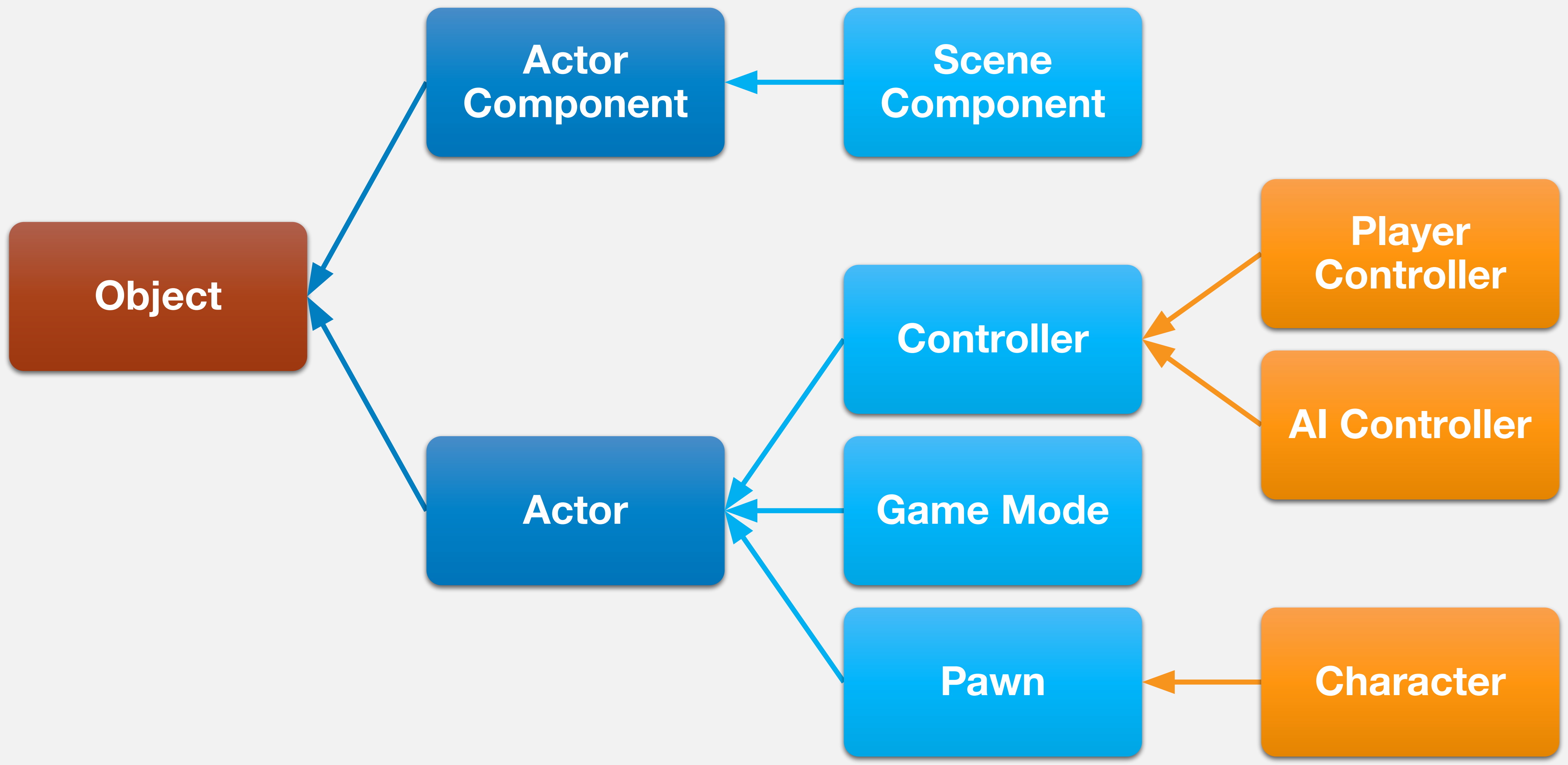
By the end of this lecture you will be able to

- Identify the common classes of the Gameplay Framework
- Understand the relationship between the Game Mode and Game State
- Understand the relationship between the Player Controller and Player State
- Know how to possess a Pawn using a Controller
- Draw simple information using the HUD class

COMMON CLASSES

When creating a Blueprint, you will be presented with a list of classes commonly used as the Blueprint parent class. These **common classes** are part of the **Gameplay Framework** and are used to represent players, characters, controllers, and game rules.

The image on the right shows the hierarchy of the common classes. The arrow indicates that the class on the right is inheriting from the class on the left. There is a base class in Unreal called the **Object** class.

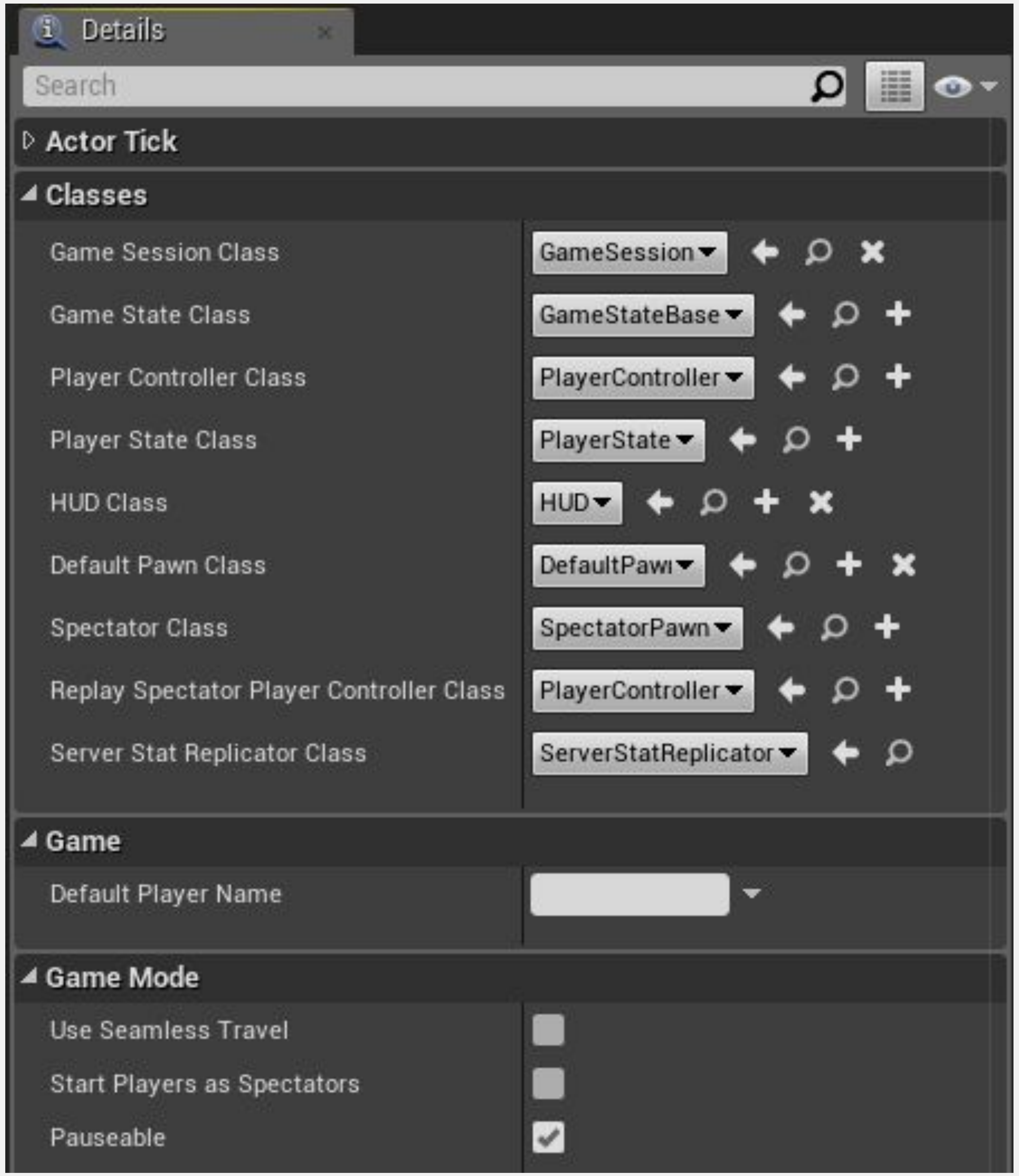


GAME MODE

The **Game Mode** class is used to define the rules of the game. The Game Mode also specifies the default classes that will be used for the creation of **Pawn**, **Player Controller**, **Game State**, **HUD**, and other classes, as seen in the image on the right.

Each Level can have a different Game Mode. If a Game Mode is not specified for the Level, it will use the Game Mode that has been set for the project.

In a multiplayer game, the Game Mode exists only on the server and is not replicated to the clients.



Details

Search

Actor Tick

Classes

Game Session Class	GameSession
Game State Class	GameStateBase
Player Controller Class	PlayerController
Player State Class	PlayerState
HUD Class	HUD
Default Pawn Class	DefaultPawn
Spectator Class	SpectatorPawn
Replay Spectator Player Controller Class	PlayerController
Server Stat Replicator Class	ServerStatReplicator

Game

Default Player Name	

Game Mode

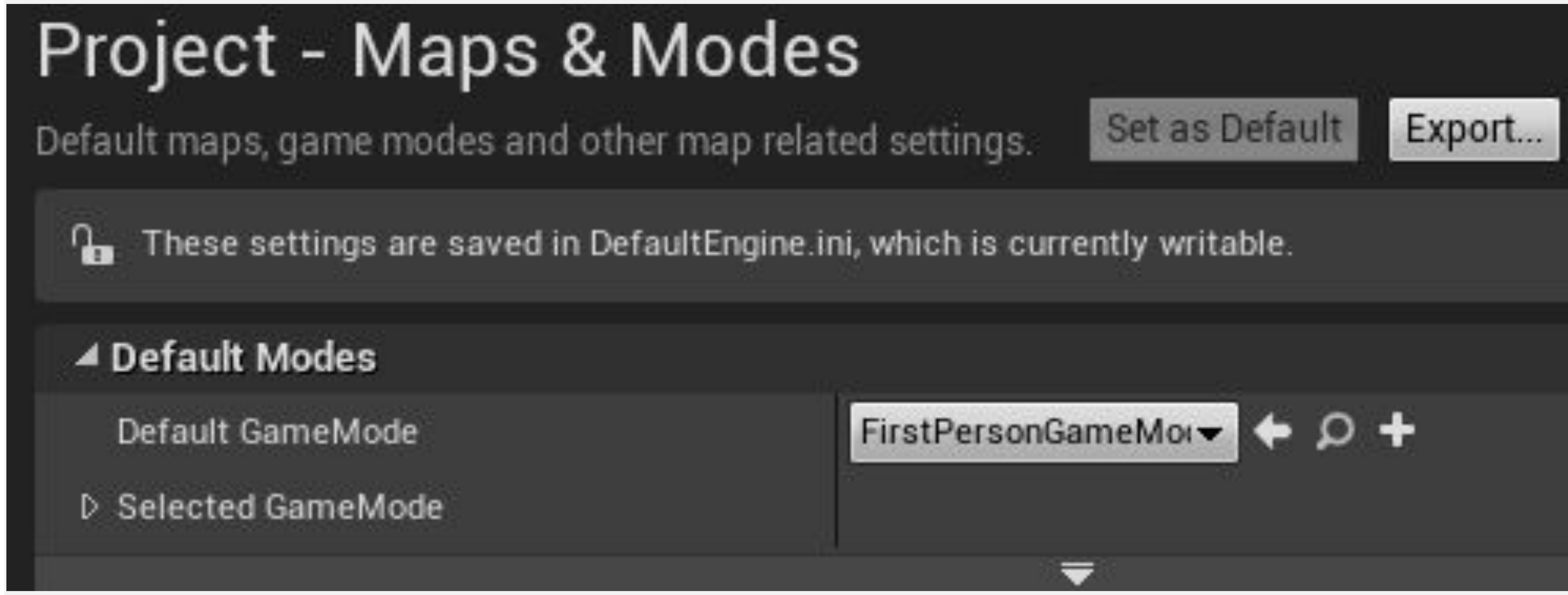
Use Seamless Travel	☐
Start Players as Spectators	☐
Pauseable	✓

ASSIGNING A GAME MODE

To specify the default Game Mode of a project, go to **Edit > Project Settings...** in the **Level Editor** and in the **Project** category select the **Maps & Modes** option. Choose the Game Mode in the **Default GameMode** property's drop-down, as seen in the top image on the right.

To specify the Game Mode of a Level, click the **Settings** button in the **Level Editor** and choose the **World Settings** option. Choose the Game Mode in the **GameMode Override** property's drop-down, as seen in the bottom image.

The Game Mode of a Level overrides the default Game Mode of the project.

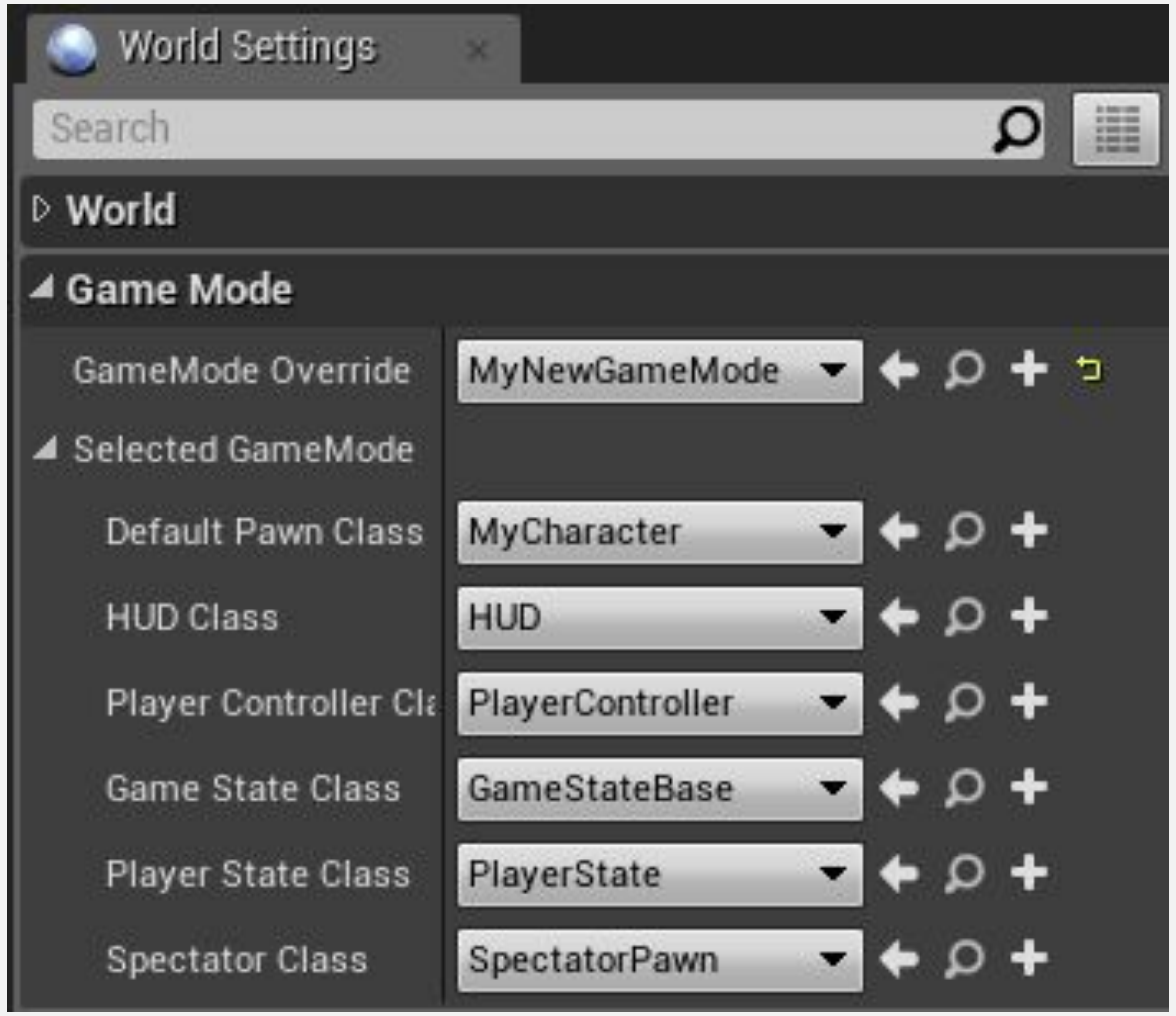

Project - Maps & Modes

Default maps, game modes and other map related settings.

These settings are saved in DefaultEngine.ini, which is currently writable.

Default Modes

Default GameMode	FirstPersonGameMo
Selected GameMode	



Game Mode

GameMode Override	MyNewGameMode
Selected GameMode	
Default Pawn Class	MyCharacter
HUD Class	HUD
Player Controller Cla	PlayerController
Game State Class	GameStateBase
Player State Class	PlayerState
Spectator Class	SpectatorPawn

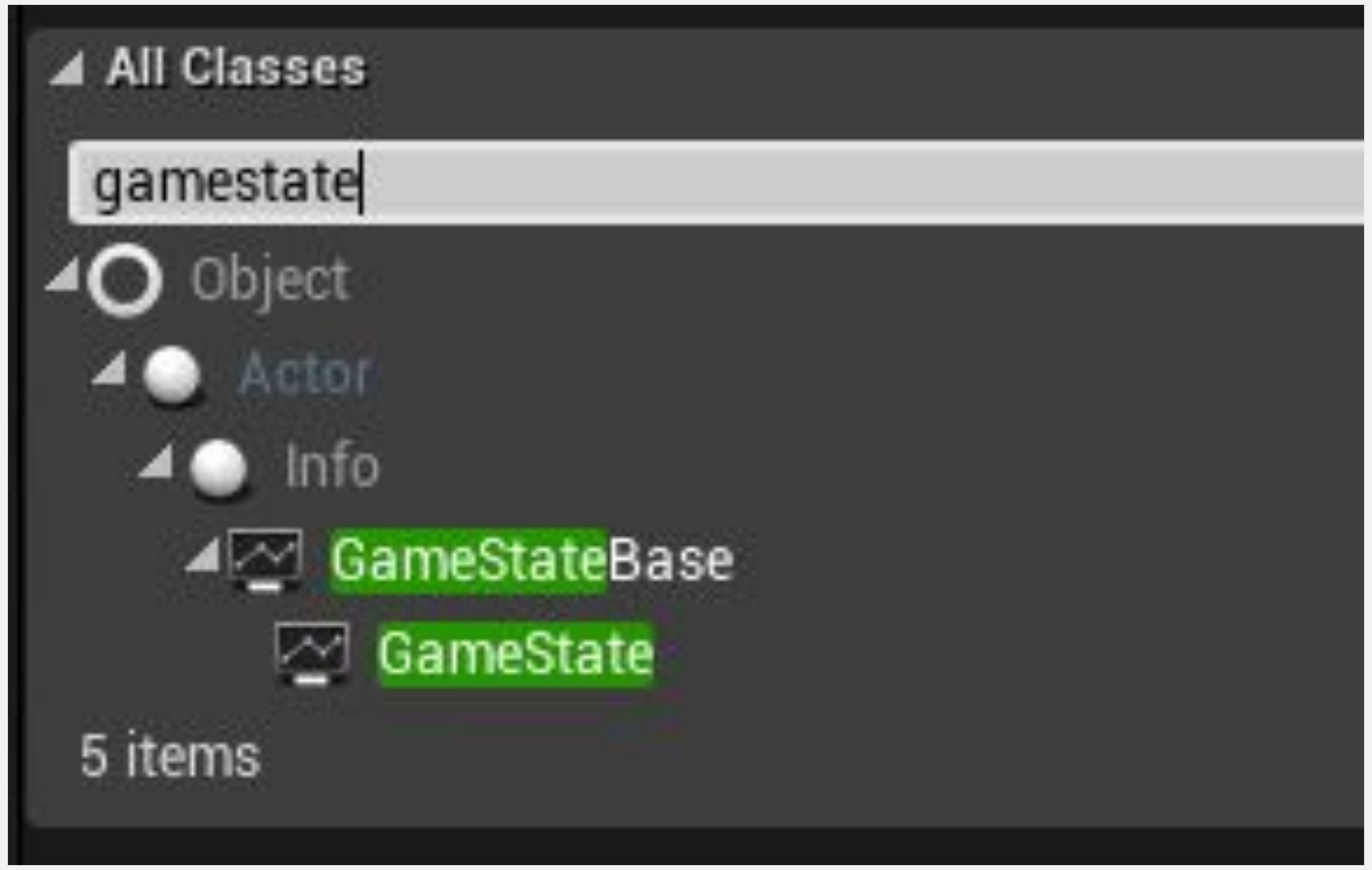
GAME STATE

The **Game State** class is used to keep track of the variables that represent the current state of the game, and it is shared with all clients in a multiplayer game.

The basic idea is that a Game Mode defines the rules on the server, whereas the Game State manages the information that changes in the game and needs to be sent to the clients.

To use a new Blueprint based on the Game State class, you must assign your custom Game State class to the **Game State Class** parameter of the Game Mode.

The Game State class extends the Game State Base class and adds some multiplayer functionality.

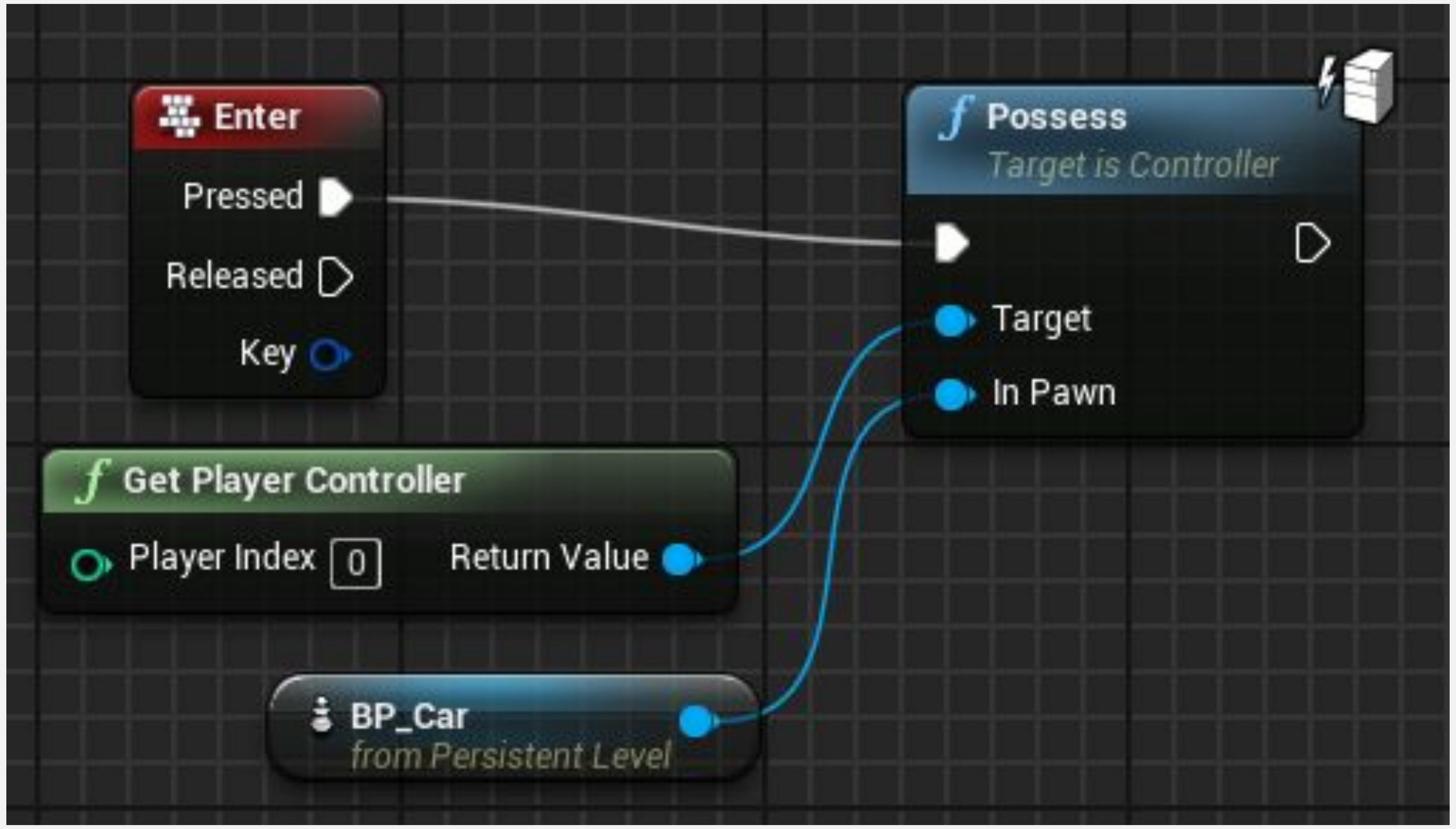


PLAYER CONTROLLER

The **Controller** class has two main subclasses. The **Player Controller** class is used by human players, and the **AI Controller** class uses artificial intelligence to control the **Pawn**.

Pawn and **Character** classes only receive input events if they are being possessed by a Player Controller.

The Pawn class being possessed by a Player Controller can be changed in-game. The image on the right is from a Level Blueprint and shows the use of the **Possess** function. In this example, a **Player Controller** will possess the **BP_Car** Pawn Actor in the Level when the **Enter** key is pressed.

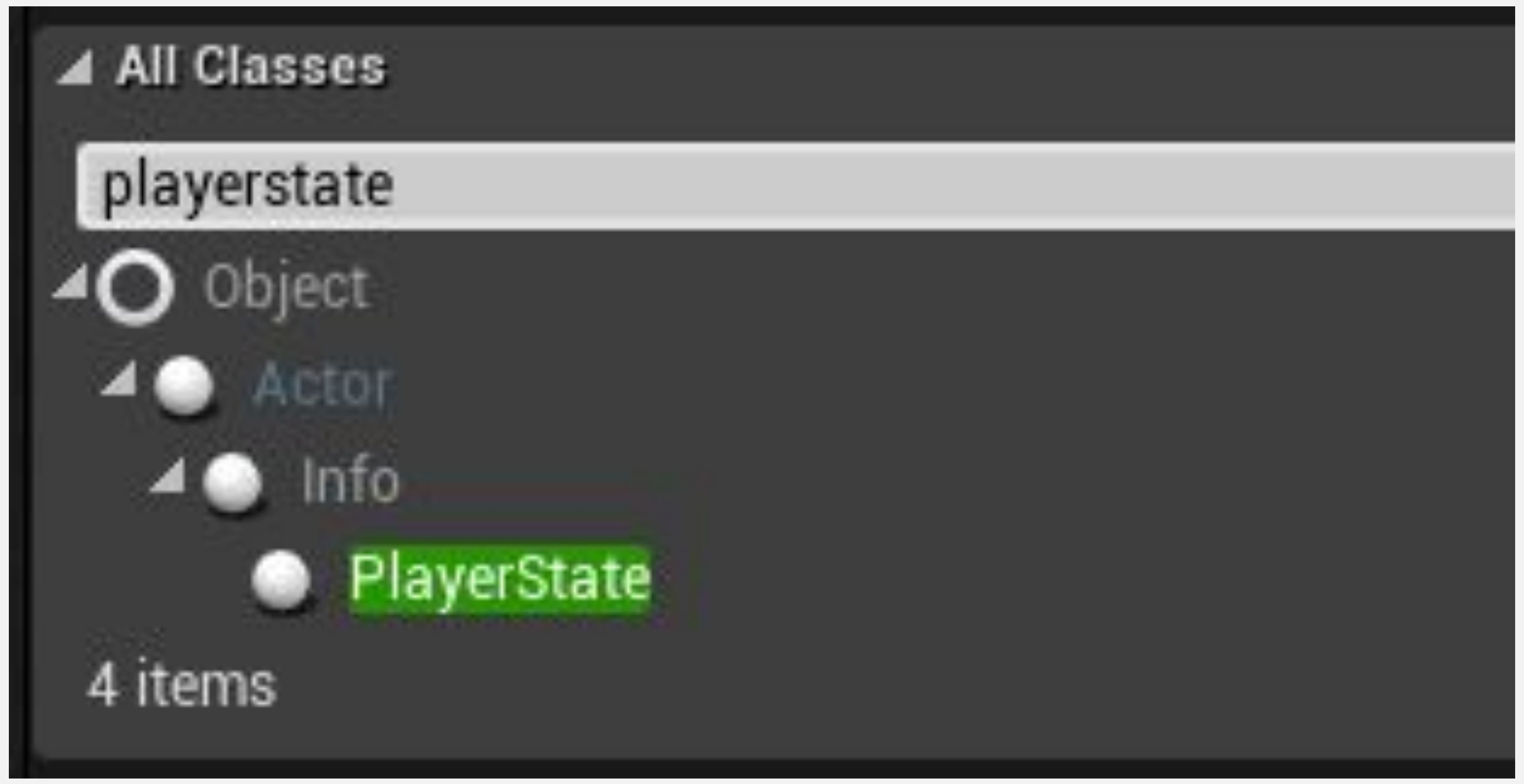


PLAYER STATE

The **Player State** class is used to keep track of the information of a specific player that needs to be shared with the others clients in a multiplayer game.

The Player Controller exists only on the clients, while the Player State is replicated to all clients from the server.

To use a new Blueprint based on the Player State class, you must set it in the **Player State Class** parameter of the Game Mode.



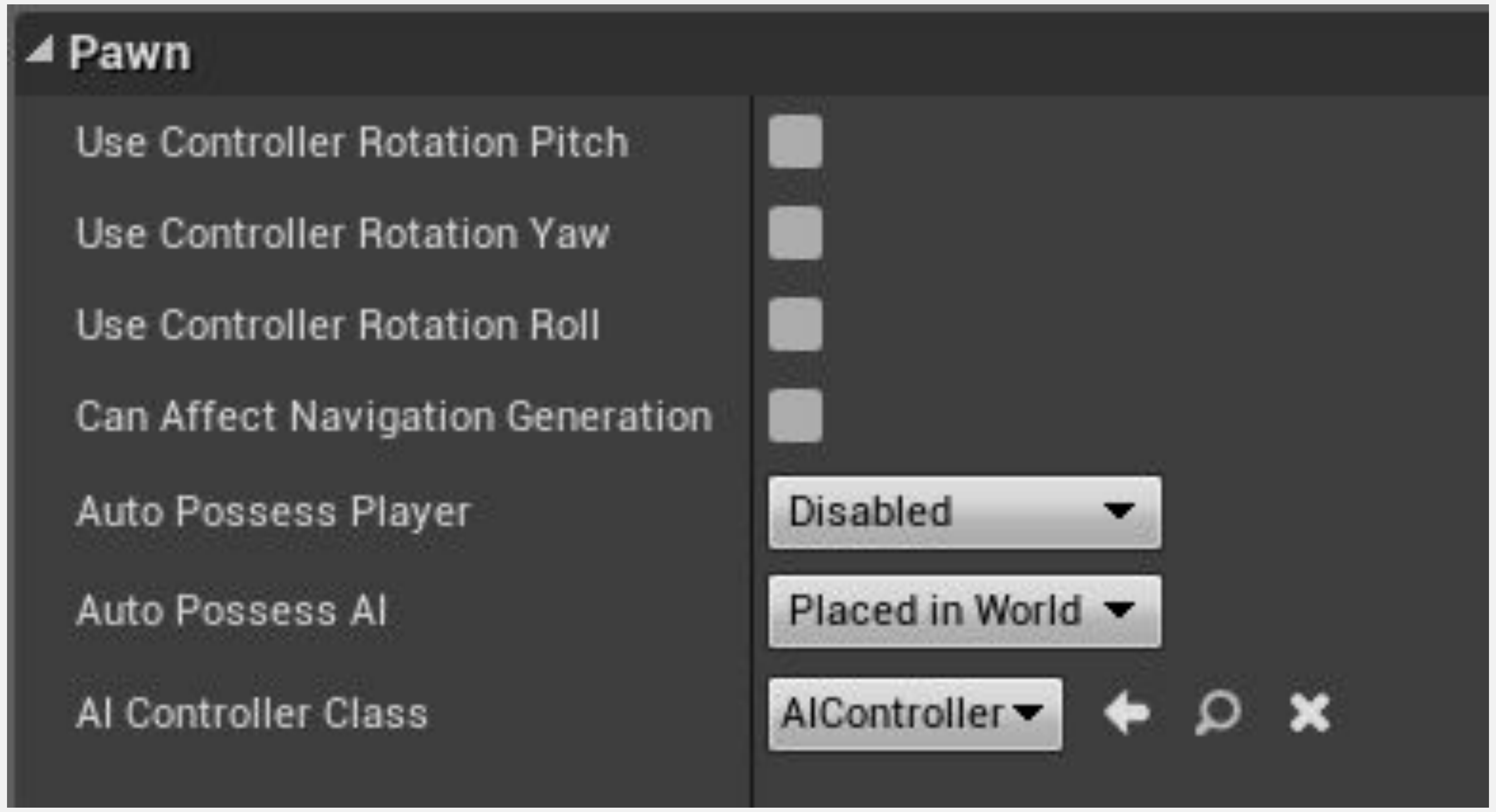
PAWN

Pawns are **Actors** that can be controlled (possessed) by **Controllers** (Player or AI).

The Pawn class represents the physical body, and the Controller class represents the brain.

The image on the right shows some of the parameters that are inherited from the Pawn class. The Pawn class can use the rotation values of the Controller that is possessing it.

Other attributes indicate how the Pawn is possessed by a Controller.



Pawn

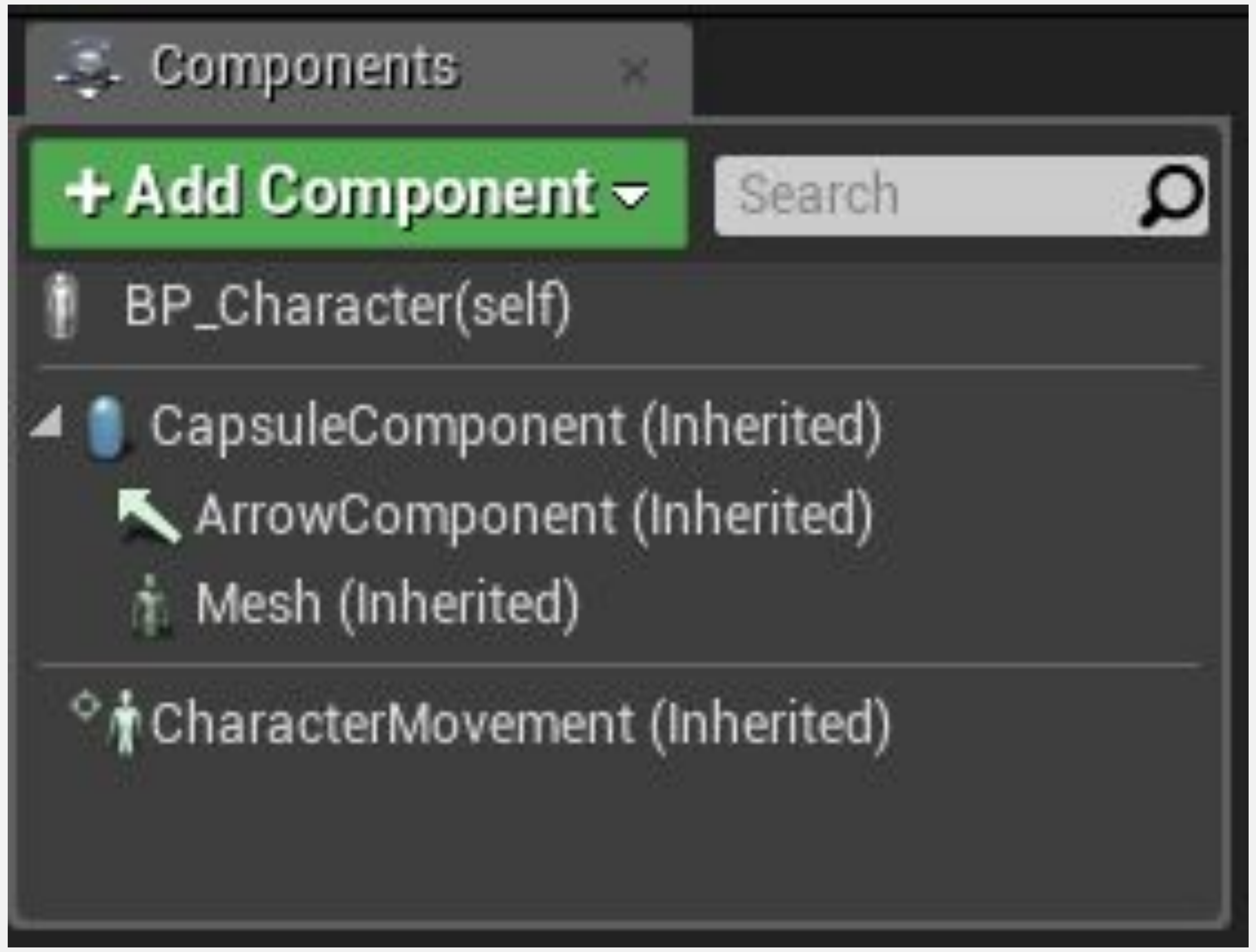
Use Controller Rotation Pitch	☐
Use Controller Rotation Yaw	☐
Use Controller Rotation Roll	☐
Can Affect Navigation Generation	☐
Auto Possess Player	Disabled
Auto Possess AI	Placed in World
AI Controller Class	AIController

PAWN: CHARACTER

The **Character** class is a subclass of the **Pawn** class that is made to represent bipedal characters that can walk, run, jump, swim, and fly. This class already has a set of components to assist in this goal.

The image on the right shows the components present in the Character class.

- The **CapsuleComponent** is used for collision testing.
- The **ArrowComponent** indicates the current direction of the character.
- The **Mesh** component is the Skeletal Mesh that visually represents the character. The animation of the Mesh component is controlled by an **animation Blueprint**.
- The **CharacterMovement** component is used to define various types of character movement such as walking, running, jumping, swimming, and flying.

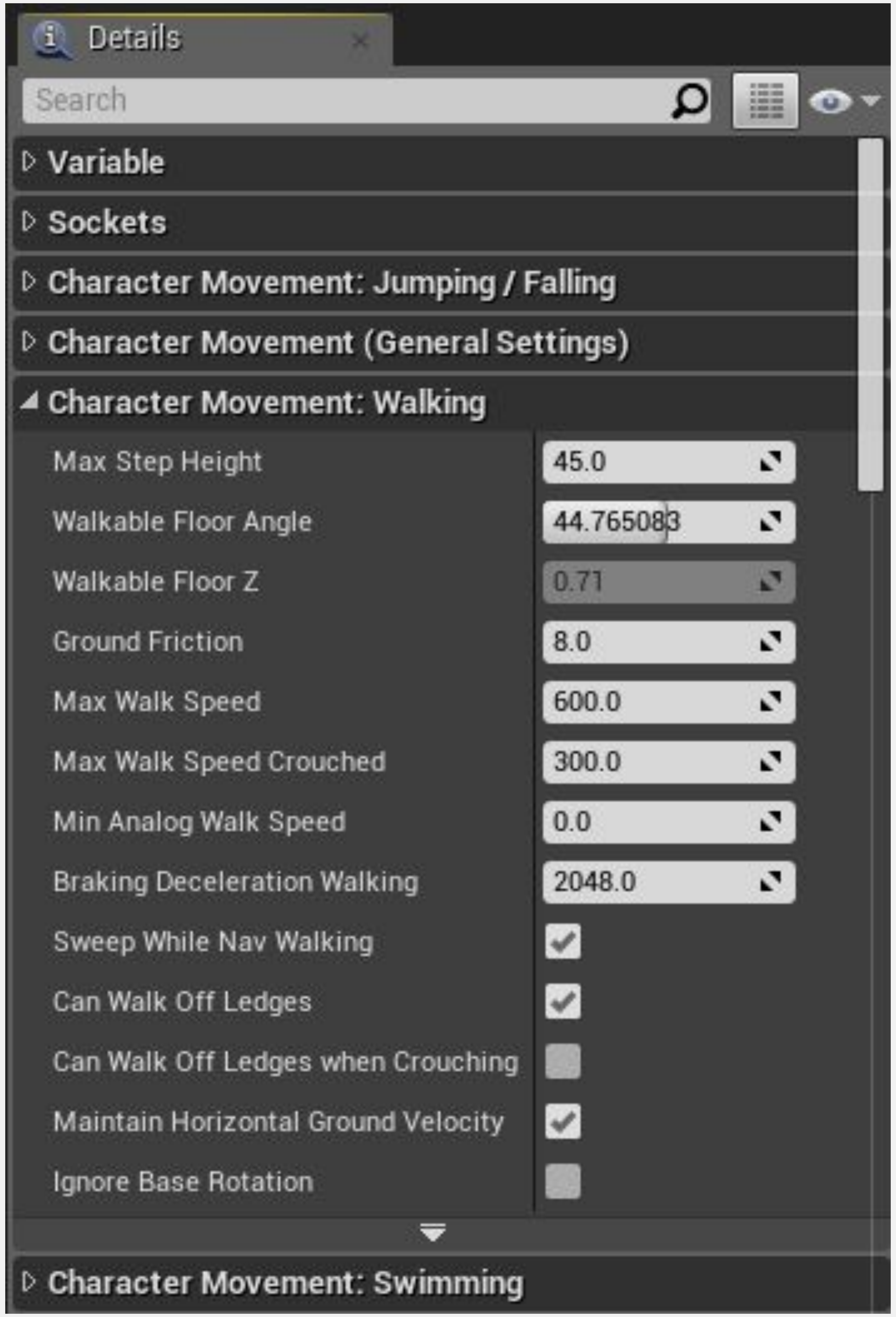


BP_Character(self)
CapsuleComponent (Inherited)
ArrowComponent (Inherited)
Mesh (Inherited)
CharacterMovement (Inherited)

CHARACTER MOVEMENT

The **CharacterMovement** component is written in C++ and handles movement as well as replication and prediction in multiplayer games.

There are a lot of properties for various types of movements that can be adjusted on the component.



Details
Search
Variable
Sockets
Character Movement: Jumping / Falling
Character Movement (General Settings)
Character Movement: Walking
Character Movement: Swimming

Max Step Height	45.0
Walkable Floor Angle	44.765083
Walkable Floor Z	0.71
Ground Friction	8.0
Max Walk Speed	600.0
Max Walk Speed Crouched	300.0
Min Analog Walk Speed	0.0
Braking Deceleration Walking	2048.0
Sweep While Nav Walking	✓
Can Walk Off Ledges	✓
Can Walk Off Ledges when Crouching	
Maintain Horizontal Ground Velocity	✓
Ignore Base Rotation	

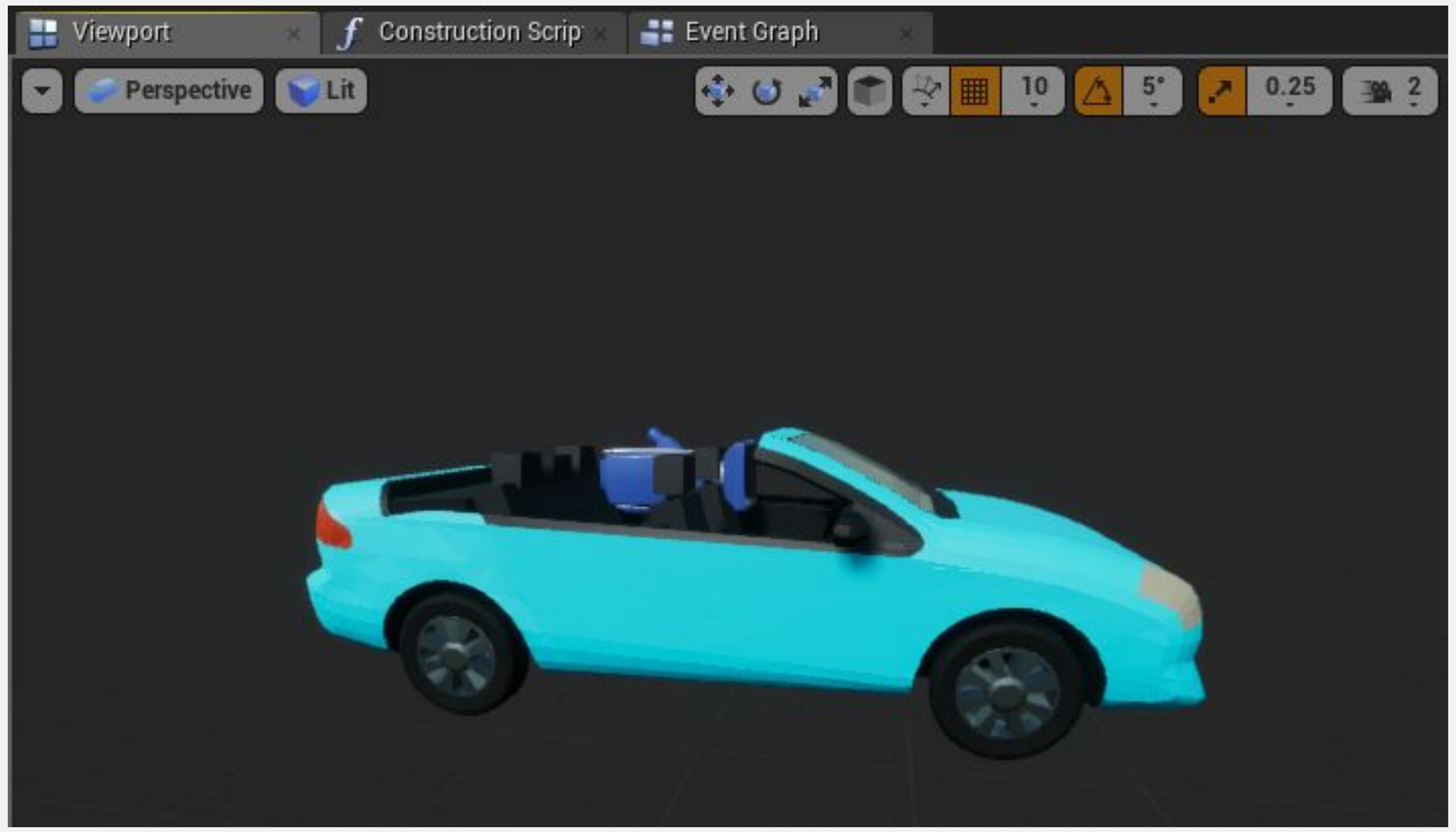
PAWN: VEHICLE

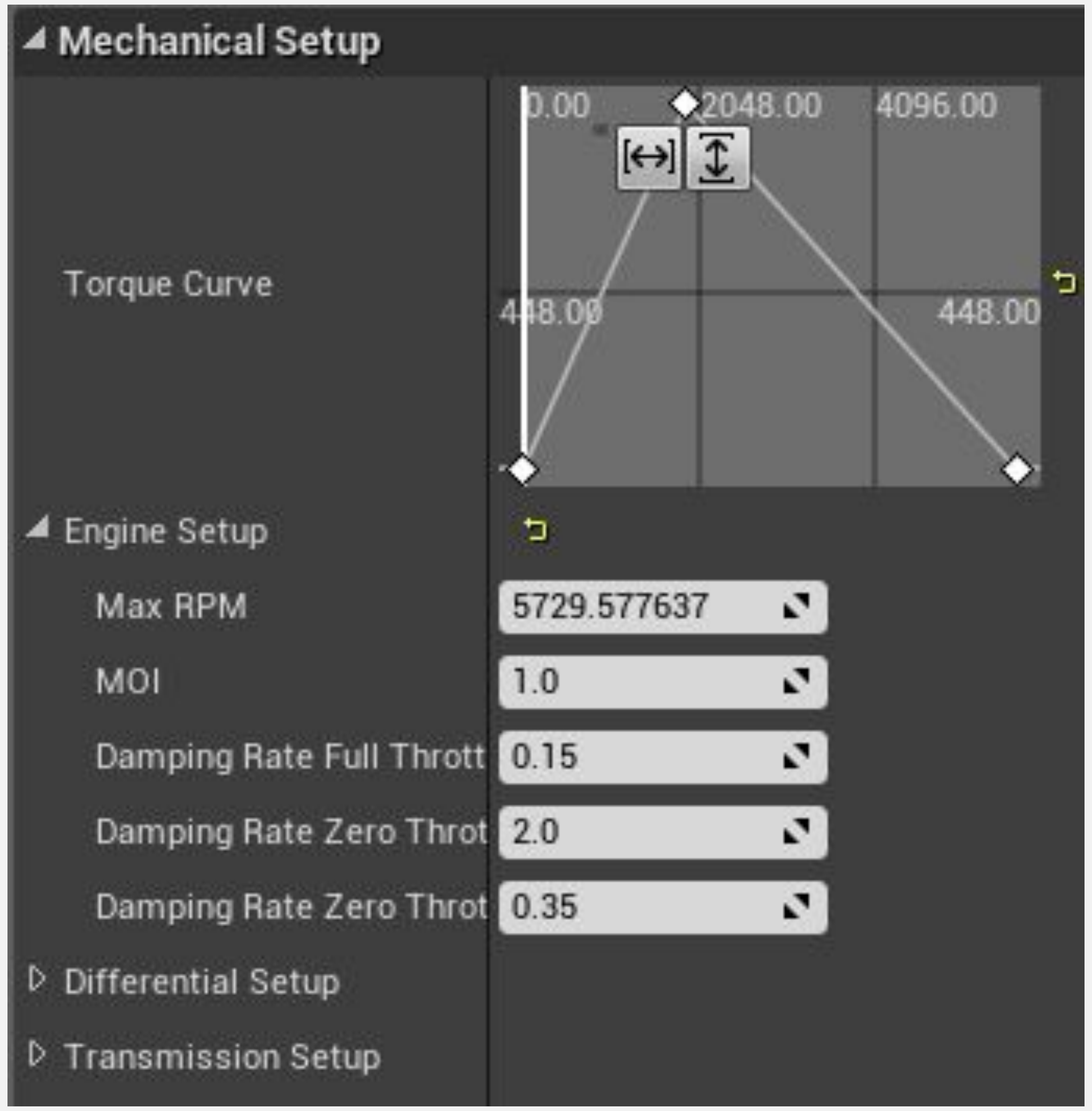
The **Wheeled Vehicle** class is another example of a subclass of the Pawn class.

It contains a **VehicleMovement** component that is written in C++ and has code to simulate the behavior of a vehicle. It has many parameters used to define the movement and physics of a vehicle. It can also handle replication and prediction in multiplayer games.

The top image on the right is from the Vehicle template.

The image at the bottom shows some properties of a VehicleMovement component.





Mechanical Setup

Torque Curve

Engine Setup

Max RPM	5729.577637
MOI	1.0
Damping Rate Full Thrott	0.15
Damping Rate Zero Throt	2.0
Damping Rate Zero Throt	0.35

Differential Setup

Transmission Setup

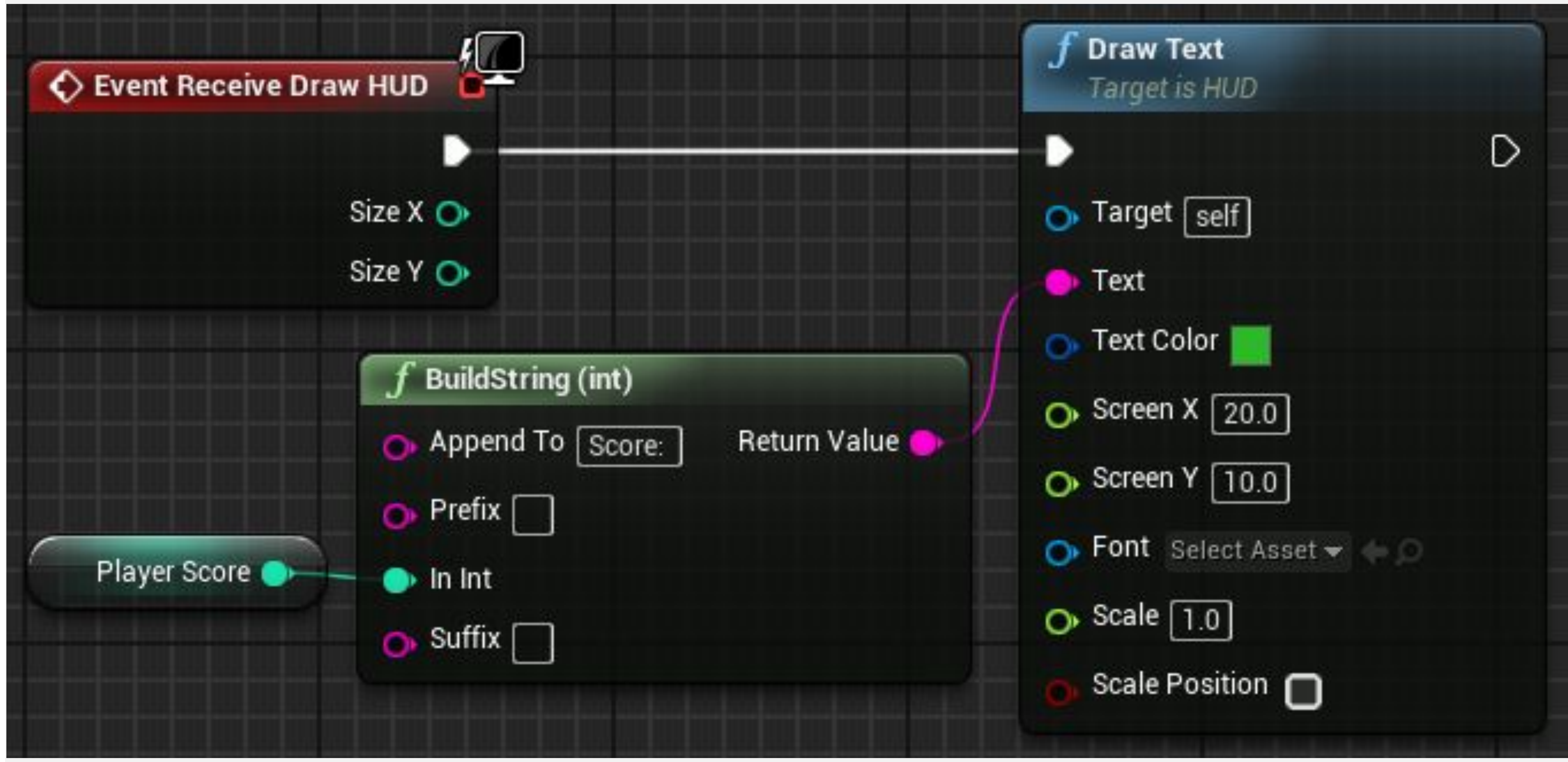
HUD

The **HUD** class is used to create a heads-up display (HUD), which is a kind of on-screen display that provides quick access to important information. The HUD is used in games to display various information to the players, such as their score, time, energy, and so on.

In Unreal Engine, the HUD class is a base class that contains a **Canvas**, which is an object on which primitives, such as text and textures, can be drawn.

The HUD class contains an event named “**Receive Draw HUD**”, which is used to draw the primitives each frame.

The HUD class exists only on each client and does not replicate.



GAME INSTANCE

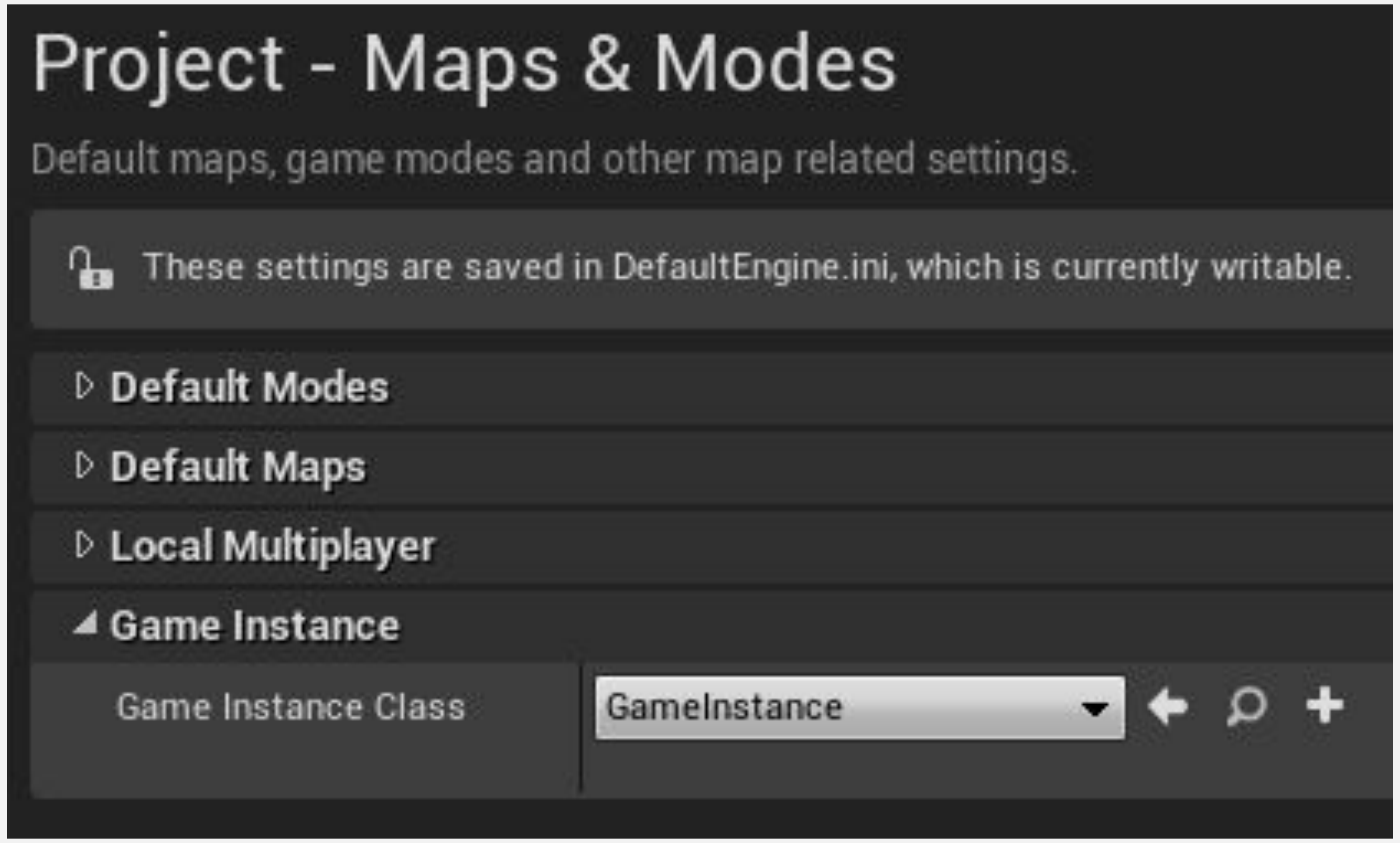
An instance of the **Game Instance** class is created at the beginning of the game and is only removed when the game is closed.

All Actors and other objects in a Level are completely destroyed and respawned each time a Level is loaded.

The Game Instance class and the data it contains persist between Levels.

The Game Instance exists only on each client and does not replicate.

To assign the **Game Instance** class for use in your game, modify the project's settings by going to **Edit > Project Settings > Maps & Modes**.



Project - Maps & Modes

Default maps, game modes and other map related settings.

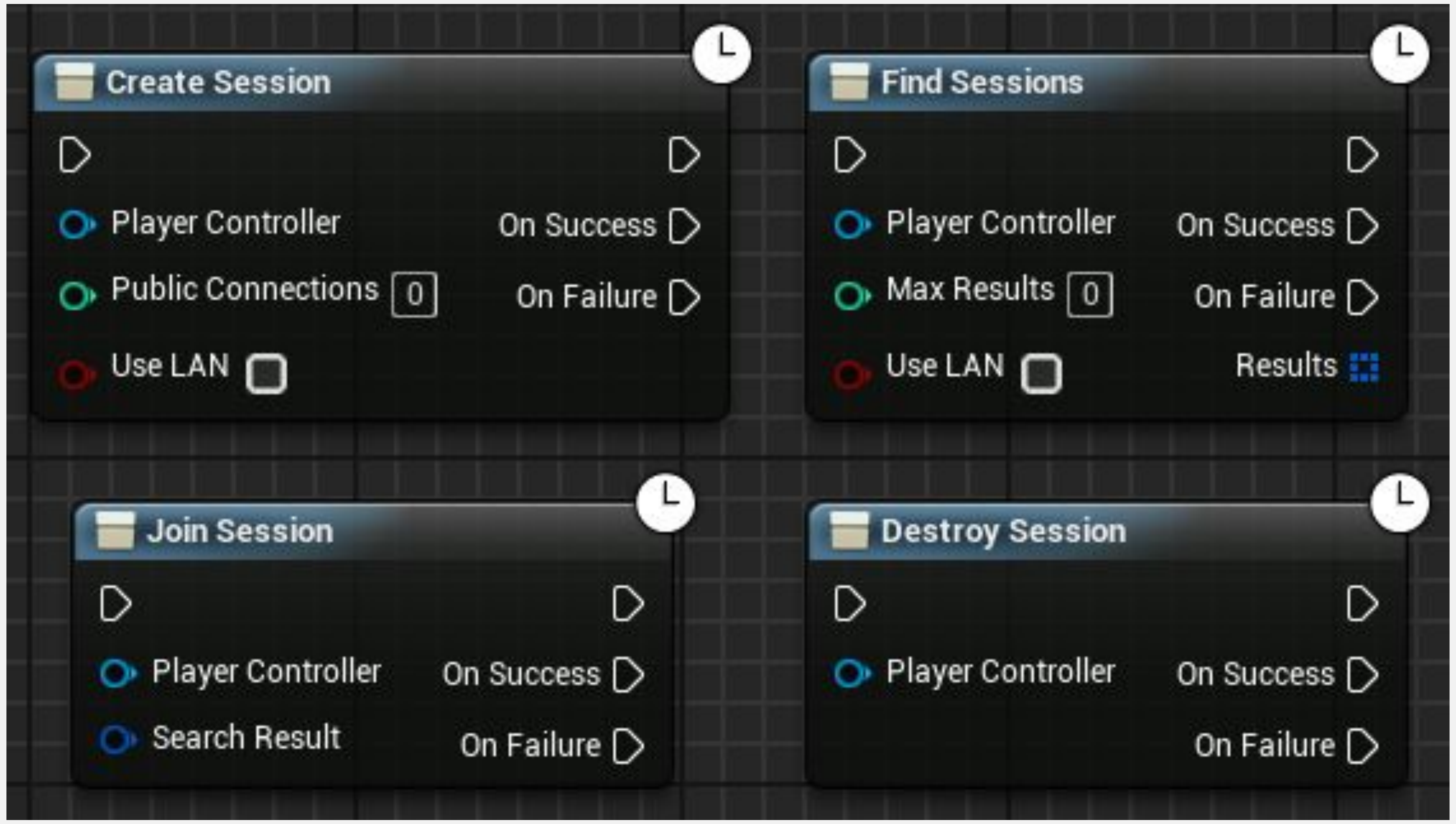
These settings are saved in DefaultEngine.ini, which is currently writable.

Game Instance Class: GameInstance

GAME SESSION

Simply stated, a **Game Session** is an interface for a game running on a server that clients can use to discover and join the game.

The image on the right shows some Blueprint functions that can be used to manage a session.



SUMMARY

This lecture presented the Gameplay Framework classes and the relationship between some common classes.

It showed how to possess a Pawn using a Controller and how to draw simple information using the HUD class.