Gameplay Framework Quick Reference

A quick guide to commonly used gameplay framework classes

The basic gameplay classes include functionality for representing players, allies, and enemies, as well as for controlling these avatars with player input or AI logic. There are also classes for creating heads-up displays and cameras for players. Finally, gameplay classes such as **GameMode**, **GameState**, and **PlayerState** set the rules of the game, and track how the game and the players are progressing.

These classes all create types of Actors, which can either be placed into your level or spawned when needed.

Representing Players, Friends, and Foes in the World

Class	Description
Pawn	A Pawn is an Actor that can be an "agent" within the world. Pawns can be possessed by a Controller, they are set up to easily accept input, and they can do various and sundry other player-like things. Note that a Pawn is not assumed to be humanoid.

Class	Description
Character	A Gameplay Framework is a humanoid-style Pawn. It comes with a CapsuleComponent for collision and a CharacterMovementComponent by default. It can do basic human-like movement, it can replicate movement smoothly across the network, and it has some animation-related functionality.

Controlling Pawns With Player Input or AI Logic

Class	Description
Controller	Controllers is an Actor that is responsible for directing a Pawn. They typically come in 2 flavors, AIController and PlayerController. A controller can "possess" a Pawn to take control of it.
PlayerController	A Player Controllers is the interface between the Pawn and the human player controlling it. The PlayerController essentially represents the human player's will.
AIController	An AIController is a simulated "will" that can control a Pawn.

Displaying Information to Players

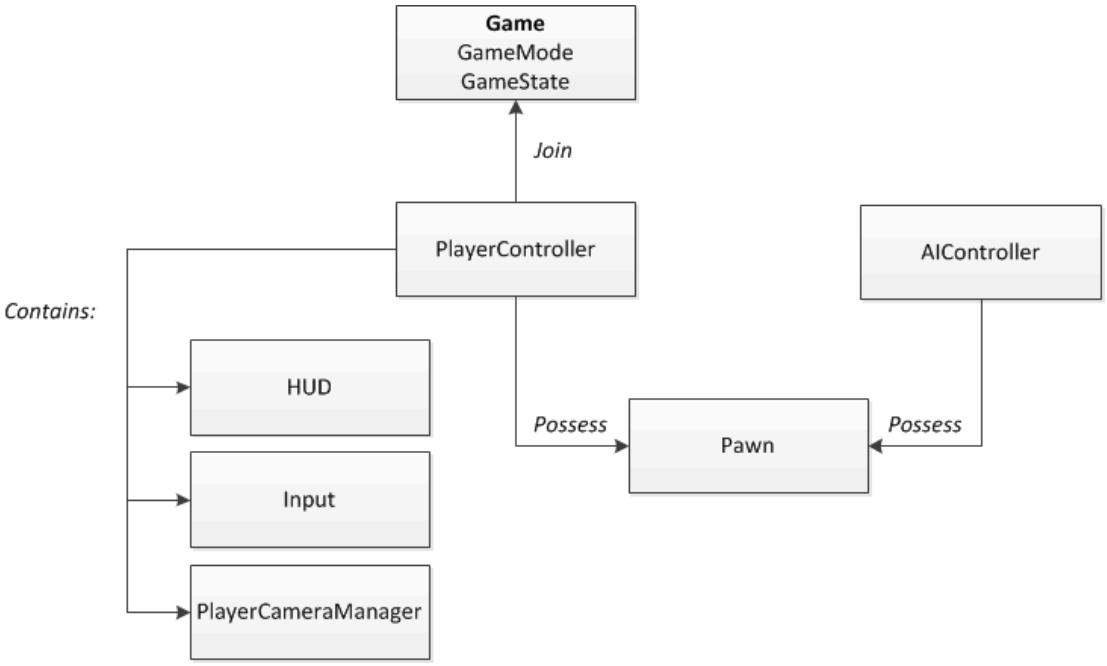
Class	Description
HUD	A HUD is a "heads-up display", or the 2D on-screen display that is common in many games. Think health, ammo, gun reticle, etc. Each PlayerController typically has one of these.
Camera	The PlayerCameraManager is the "eyeball" for a player and manages how it behaves. Each PlayerController typically has one of these as well. For more, see the camera workflow page.

Setting and Tracking the Rules of the Game

Class	Description
GameMode	The concept of a "game" is split into 2 classes. The Game Mode and Game State is the definition of the game, including things like the game rules and win conditions. It only exists on the server. It typically should not have much data that changes during play, and it definitely should not have transient data that clients need to know about.
GameState	The GameState contains the state of the game, which could include things like the list of connected players, the score, where the pieces are in a chess game, or the list of what missions you have completed in an open world game. The GameState exists on the server and all clients and can replicate freely to keep all machines up to date.
PlayerState	A **PlayerState** is the state of a participant in the game, such as a human player or a bot that is simulating a player. Non-player AI that exists as part of the game would not have a PlayerState. Example data that would be appropriate in a PlayerState include player name, score, in-match level for something like a MOBA, or whether the player is currently carrying the flag in a CTF game. PlayerStates for all players exist on all machines (unlike PlayerControllers) and can replicate freely to keep things in sync.

Framework Class Relationships

This flowchart illustrates how these core gameplay classes relate to each other. A game is made up of a GameMode and GameState. Human players joining the game are associated with PlayerControllers. These PlayerControllers allow players to possess pawns in the game so they can have physical representations in the level. PlayerControllers also give players input controls, a heads-up display, or HUD, and a PlayerCameraManager for handling camera views.



For more on gameplay framework classes, see Gameplay Framework.