Gameplay Ability System High-Level Overview 2.0

1. GameplayCue

Encapsulates Cosmetic effects.

2. Gameplay Effects

Gameplay Effects are the Gameplay Ability System's way to change Attributes.

2.1. Direct changes to the Attribute's base value, like taking away health points from an Actor that has received damage.

2.2. Temporary changes (often called "buffs" or "debuffs"), like granting a boost to movement speed for a few seconds.

2.3. Persistent changes that are applied over time, like regenerating a certain amount of magic points per second over a period of several seconds (or indefinitely).

3. Gameplay Attributes

A Gameplay Attribute contains any numerical measurement of an Actor's current state that can be described by a single floating-point value, such as health points, physical strength, movement speed, resistance to magic, and so on. Attributes are declared as UProperties of FGameplayAttribute type within Attribute Sets, which both contain Attributes and supervise any attempts to modify them. they are modified by Gameplay Effects.

3.1. Inherit public UAttributeSet

3.1.1. FGameplayAttributeData Health;

Example of defining a Gameplay Attribute

3.1.2. Once you have created an AttributeSet,

you must register it with the AbilitySystemComponent.

4. **Revoking Abilities**

Override UGameplayAbility class to make a GameplayAbility. "Abilities define custom gameplay logic that can be activated by players or external game logic."

4.1. Execution lifecycle

A Gameplay Ability's basic execution lifecycle, after being granted to an Actor's AbilitySystemComponent, looks like: see Subtopics

4.1.1. CanActivateAbility

lets the caller know whether or not an Ability is available for execution without attempting to execute it.

4.1.2. CallActivateAbility

Executes the game code with the Ability, but does not check to see if the Ability should be available, This function is usually called in cases where some logic is needed between the CanActivateAbility check and the execution of the Ability. The main code that users need to override with their Ability's custom functionality is either the C++ function called "ActivateAbility", or the Blueprint Event called Activate Ability. The CommitAbility function, if called from within Activate, will apply the cost of executing the ability, such as by subtracting resources from GameplayAttributes (E.g. Magic, Points, Stamina) and applying cooldowns. CancelAbility provides a mechanism to cancel the Ability, although the Ability's CanBeCanceled function can reject the request.

4.1.3. CommitAbility

The CommitAbility function, if called from within Activate, will apply the cost of executing the ability, such as by subtracting resources from GameplayAttributes (E.g. Magic, Points, Stamina) and applying cooldowns.

4.1.4. EndAbility

EndAbility shuts the Ability down when it is finished executing. If the Ability was canceled, the UGameplayAbility class will handle this automatically as part of the cancelation process, but in all other cases, the developer must call the C++ function or add the node into the Ability's Blueprint graph. Failing to end the ability properly will result in the Gameplay Ability System believing that the Ability is still running

4.1.5. TryActivateAbility

TryActivateAbility is the typical way to execute Abilities. This function calls CanActivateAbility to determine whether or not the Ability can run immediately, followed by CallActivateAbility if it can.

5. **Granting access to Ability**

This is required before revoking abilities

5.1. Ability System Component

5.1.1. GiveAbility

Specifies the Ability to add with an FGameplayAbilitySpec, and returns an FGameplayAbilitySpecHandle

5.1.2. GiveAbilityAndActivateOnce

Specifies the Ability to add with an FGameplayAbilitySpec, and returns an FGameplayAbilitySpecHandle. The Ability must be instanced and able to run on the server. After attempting to run the Ability on the server, a FGameplayAbilitySpecHandle will be returned. If the Ability did not meet the required criteria, or if it could not execute, the return value will be invalid and the Ability System Component will not be granted the Ability.

5.1.3. ClearAbility

Removes the specified Ability from the Ability System Component

5.1.4. SetRemoveAbilityOnEnd

Removes the specified Ability from the Ability System Component when that ability is finished executing. If the Ability is not executing, it will be removed immediately. If the Ability is executing, its input will be cleared immediately so that the player cannot reactivate or interact with it any further.

5.1.5. ClearAllAbilities

Removes all Abilities from the Ability System Component. This function is the only one that does not require an FGameplayAbilitySpecHandle.

5.1.6. Using the FGameplayAbilitySpecHandle that was returned when the Ability was granted

6. **Triggering with Gameplay Events**

Gameplay Events are data structures that can be passed around to trigger Gameplay Abilities directly, sending a data payload for context, without going through the normal channels. The usual way to do this is by calling Send Gameplay Event To Actor and providing an Actor that implements the IAbilitySystemInterface interface and the contextual information that Gameplay Events require, but it is also possible to call Handle Gameplay Event directly on an Ability System Component. Because this isn't the normal path to calling Gameplay Abilities, context information that the Ability may need will be passed in through the FGameplayEventData data structure. The polymorphic ContextHandle field will provide support for additional information as needed.

7. **Some GameplayAbility properties**

7.1. Replication

The Ability's Gameplay Ability Replication Policy can be set to "Yes" or "No", and this will control whether the Ability replicates instances of itself, makes state updates, or sends Gameplay Events across the network. For multiplayer games with Abilities that do replicate, there are a few options for how replication is handled, known as the Gameplay Net Execution Policy:

7.1.1. Gameplay Net Execution Policy

- Local Predicted

  Run on client immediately, but the server will have the final word, and can override the client in terms of what Ability's actual impact was. The client is effectively asking permission from the server to execute the Ability, but also proceeding locally as if the server is expected to agree with the client's view of the outcome. Because the client is locally predicting the behavior of the Ability, it will feel perfectly smooth and lag-free as long as the server does not contradict the client's prediction.

- Local Only

  The client simply runs the Ability locally. There is no replication to the server. Anything the client affects with this Ability will still follow normal replication protocol, including potentially receiving corrections from the server.

- Server Initiated

  Abilities that initiate on the server will propagate to clients. These are often more accurate, from the client's perspective, to what is actually happening on the server, but the client using the Ability will observe a delay due to the lack of local prediction. While this delay should be very short, some types of Abilities, especially rapidly-performed actions in pressured situations, will not feel as smooth as they would in Local Predicted mode.

- Server Only

Server Only Abilities will run on the server, and will not replicate to clients. Any variables these Abilities change will replicate as they normally do, meaning that the Ability can affect server-authoritative data, which will then propagate to the clients. In this way, the Ability can still have effects that clients observe, even though the Ability itself will only run on the server.

7.2. Instancing Policy

When a Gameplay Ability executes, a new Object (of the Ability's type) will usually spawn to track the Ability in progress. Since Abilities can execute very frequently in some cases, such as battles between a hundred or more players and AI characters in Battle Royale, MOBA, MMO, or RTS games, there may be cases where rapid creation of Abiltiy Objects can negatively impact performance. To address this, Abilities have a choice of three different instantiantion policies, offering tradeoffs between efficiency and functionality.

7.2.1. Non-Instanced

This is the most efficient instancing policy in all categories. The Ability will not spawn any Object when it runs, and will instead use the Class Default Object. You can create Blueprint classes of a non-instanced Ability, but only to change the default values of exposed Properties. Must be written in C++ Must not change member variables Must not bind Delegates Can not replicate variables or handle RPCs Only use on Abilities that require no internal variable storage (although setting Attributes on the user of the Ability is possible) and don't need to replicate any data. It is especially well-suited to Abilities that run frequently and are used by many characters, such as the basic attack used by units in a large-scale RTS or MOBA title.

7.2.2. Instanced per Actor

Each Actor will spawn one instance of this Ability when the Ability is first executed, and future executions will reuse it. Per-Actor is ideal for replication, as the Ability has a replicated Object that can handle variable changes and RPCs, but does not waste network bandwidth and CPU time spawning a new Object every time it runs. In larger-scale situations, this policy performs well, since large numbers of Actors using the Ability (for example, in a big battle) will only spawn Objects on their first Ability use.

7.2.3. Instanced per Execution

A copy of the Ability's Object will spawn each time the Ability runs.

7.3. Cancel Abilities Matching Tag Query

7.4. Each Ability possesses a set of Tags

The "if". They work like if-statement conditions. Each Ability possesses a set of Tags that identify and categorize it in ways that can affect its behavior, as well as Gameplay Tag Containers and Gameplay Tag Queries to support these interactions with other Abilities.

7.4.1. Ability Tags

The Gameplay Tags this ability has

7.4.2. Cancel Abilities With Tag

Cancels any already-executing Ability with Tags matching the list provided while this Ability is executing.

7.4.3. Block Abilities With Tag

Prevents execution of any other Ability with a matching Tag while this Ability is executing

7.4.4. Activation Owned Tags

While this Ability is executing, the owner of the Ability will be granted this set of Tags

7.4.5. Activation Required Tags

The Ability can only be activated if the activating Actor or Component has all of these Tags.

7.4.6. Activation Blocked Tags

The Ability can only be activated if the activating Actor or Component does not have any of these Tags.

7.4.7. Source Required Tags

GameplayAbility can only be activated if the source has all of these GameplayTags.

7.4.8. Source Blocked Tags

GameplayAbility will not be activated if the source has any of these GameplayTags

7.4.9. Target Required Tags

The Ability can only be activated if the targeted Actor or Component has all of these Tags.

7.4.10. Target Blocked Tags

The Ability can only be activated if the targeted Actor or Component does not have any of these Tags.

7.5. Triggering a GameplayEffect when Ability is commited

7.5.1. "Cost Gameplay Effect Class"

Represents the cost

7.5.2. "Cooldown Gameplay Effect Class"

Represents the cooldown

7.6. Ability triggers array

7.6.1. Trigger Tag

7.6.2. Trigger Source

- Gameplay Event
- Owned Tag Added
- Owned Tag Present