Gameplay Timers

Timer construct for performing delayed or repeated actions

Timers schedule actions to be performed after a delay, or over a period of time. For example, you may want to make the player invulnerable after obtaining a power-up item, then restore vulnerability after 10 seconds. Or you may want to apply damage once per second while the player moves through a room filled with toxic gas. Such actions can be achieved through the use of timers.

Timer Management

Timers are managed in a global **Timer Manager** (of type `FTimerManager`). A global Timer Manager exists on the **Game Instance** Object, as well as on each **World**. Two main functions are used to set up timers with a Timer Manager: `SetTimer`, and `SetTimerForNextTick`, each with several overloads. Each one can be attached to any type of Object or function delegate, and `SetTimer` can be made to repeat at regular intervals, if desired. See the Timer Manager API page for more detail on these functions.

> Timers will be canceled automatically if the Object that they are going to be called on, such as an Actor, is destroyed before the time is up. In this case, the timer handle will become invalid and the function will not be called.

Accessing the Timer Manager can be done through the `AActor` function called `GetWorldTimerManager`, which calls up to the `GetTimerManager` function in `UWorld`. To access the global Timer Manager, use the `UGameInstance` function, `GetTimerManager'. This is also the fallback used if a World doesn't have its own Timer Manager for any reason, and can be used for function calls that aren't relevant to, or should not depend on, the existence of any specific World.

Timers can be used with the standard C++ function pointers `TFunction` Objects or Delegates.

Setting and Clearing Timers

The `SetTimer` functions of `FTimerManager` will set a timer to call a function or delegate after a delay, and can be set to repeat that function call indefinitely. These functions will fill out a **timer handle** (type `FTimerHandle`), which can be used to pause (and resume) the countdown, query or change the amount of time remaining, or even cancel the timer altogether. It is safe to set timers within a function called by a timer, even including reuse of the timer handle that was used to call the function. One use for this might be to delay initialization of one Actor if it depends on another Actor that hasn't spawned yet but is expected to spawn soon; the dependent Actor's initialization function could set a timer to call itself again after a fixed length of time, such as one second. Alternately, the initialization function could be called by a looping timer that clears itself upon success.

Timers can also be set to run on the next frame, rather than with a timed interval. This is accomplished by calling `SetTimerForNextTick`, but note that this function does not fill out a timer handle.

To clear a timer, pass the `FTimerHandle` that was filled out during the `SetTimer` call into the `FTimerManager` function called `ClearTimer`. The timer handle will become invalid at this point, and can be reused to manage a new timer. Calling `SetTimer` with an existing timer handle will clear the timer referenced by that timer handle and replace it with a new one.

Finally, all timers associated with a specific Object can be cleared by calling `ClearAllTimersForObject`.

Example:

```
void AMyActor::BeginPlay()
{
Super::BeginPlay();
// Call RepeatingFunction once per second, starting two seconds from now.
GetWorldTimerManager().SetTimer(MemberTimerHandle, this,
&AMyActor::RepeatingFunction, 1.0f, true, 2.0f);
}

void AMyActor::RepeatingFunction()
{
// Once we've called this function enough times, clear the Timer.
if (--RepeatingCallsRemaining <= 0)
{
GetWorldTimerManager().ClearTimer(MemberTimerHandle);
// MemberTimerHandle can now be reused for any other Timer.
}
// Do something here...
}

```

Copy full snippet

Calling `SetTimer` with a rate less than or equal to zero is identical to calling `ClearTimer`.

Pausing and Resuming Timers

The `FTimerManager` function `PauseTimer` uses a timer handle to pause a running timer. This prevents the timer from executing its function call, but the elapsed and remaining times stay the same while paused. `UnPauseTimer` causes a paused timer to resume running.

Timer Information

In addition to managing timers, timer managers also provide funtions for obtaining information —such as the rate, elapsed time, and remaining time—for a specific timer.

Is Timer Active

The `IsTimerActive` function of `FTimerManager` is used to determine if the specified timer is curently active and not paused.

Example:

```
// Is this weapon waiting to be able to fire again?
GetWorldTimerManager().IsTimerActive(this, &AUTWeapon::RefireCheckTimer);

```

Copy full snippet

Timer Rate

`FTimerManager` has a function called `GetTimerRate` that gets the current rate (time between activations) of a timer from its timer handle. A timer's rate cannot be changed directly, but `SetTimer` can be called with its timer handle to clear it and create a new timer, which could be identical except for the rate. `GetTimerRate` will return a value of `-1` if the timer handle is not valid.

Example:

```
// This weapon's rate of fire changes as it warms up. Is it currently waiting to fire, and if so, how long is the current delay between shots?
GetWorldTimerManager().GetTimerRate(this, &AUTWeapon::RefireCheckTimer);

```

Copy full snippet

Elapsed and Remaining Time

`FTimermanager` provides functionality through `GetTimerElapsed` and `GetTimerRemaining`, returning the elapsed and remaining time, respectively, for the timer associated with the provided timer handle. As with `GetTimerRate`, these functions will return `-1` if the timer handle is invalid.

Example:

```
1 // How long will it be until this weapon is ready to fire again? If the
  answer comes back as -1, it is ready now.
2 GetWorldTimerManager().GetTimerElapsed(this, &AUTWeapon::RefireCheckTimer);
3
```

Copy full snippet

> The sum of the elapsed and remaining time for a timer should be equal to the rate of the timer.

Known Issues

- The code is not thread safe at the moment, and will cause an assert if accessed outside of the game thread.