APooledCharacter

APooledCharacter is mainly responsible for handling the pooling logic for the characters that use it and properly handle instances that aren't pooled.

Inherits from class: ACharacter

Implements interfaces: IPoolable, IEventListener, ISaveable

Important Variables to Know

This isn't *all* of the variables that are present but are some that are very important to the APooledCharacter's functionality:

int64 SoulID: A unique identifier for an NPC. This number is used to find the correct data associated with that NPC, so if you give the APooledCharacter a SoulID number, you will always get that "soul" (if it exists). Think of it like the equivalent of an American Social Security Number.

TEnumAsByte<EVE_ObjectType> objectType: The type of object that the NPC is. When using a pooled instance of APooledCharacter, objectType helps determine which definition database to reference. This works in tandem with the DefinitionID variable to get the correct "species" of the APooledCharacter.

int32 DefinitionID: The order that the DefinitionData is in its corresponding database. It refers to general information about an NPC type, but not to a specific instance of that type (that would be SoulID). Think about this as the "species" that the APooledCharacter will become. Keep in mind that DefinitionData assets from different databases can have the same value, so it is important to use the objectType variable along with this so that the correct database is referenced.

bool bIsUsed: Is this instance of APooledCharacter currently in use? Unused instances can be given the data needed to become an NPC while not overriding an instance that's already in use. Used APooledCharacters are added to a list of objects whose data will be saved by the Saving Subsystem.

bool bUsingPooledMethod: Determines whether or not this instance of APooledCharacter is actually part of a pool. Instances that are in a pool are re-used so that we don't have to constantly create and destroy APooledCharacter instances, which can boggle down performance if left unchecked. Non-pooled instances are still monitored for saving and function normally; the only difference is that they are destroyed when they are no longer in use.

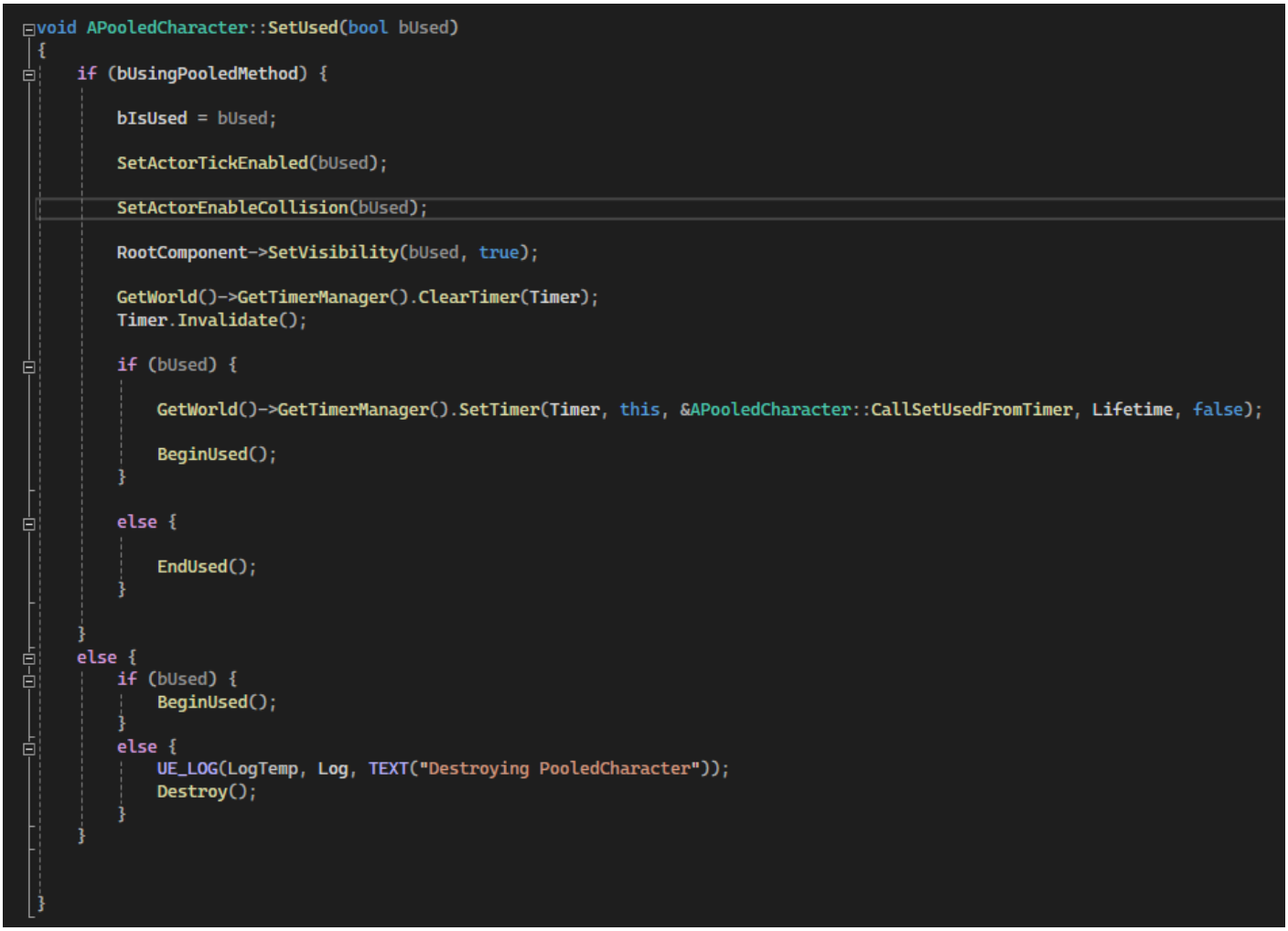
bool saveableInstance: Can this instance of APooledCharacter be saved? This is used to exclude any ambient APooledCharacters that do not need to be saved.

void Initialise(int64 characterSoul, int32 DefinitionDataID, EVE_ObjectType NPCType)

Initialise is a Blueprint Native function that can be called and defined by blueprint children. At the APooledCharacter level, the initialize function is a lightweight function; all it does is sets the value of the APooledCharacter's soulID, objectType, and DefinitionDataID (as seen in Initialise_Implementation). The child blueprints need to use the NPCType variable to find the correct definition database that the DefinitionDataID references. When overriding this function in blueprint, ensure that it has a call to its parent function so that the variables are properly set.

void SetUsed(bool bUsed)

SetUsed tells the APooledCharacter instance whether or not it is being used. If the instance is part of a pool, it enables/disables the ActorTick, collision, and visibility, then either begins or ends using the APooledCharacter (more on that later). If the APooledCharacter is NOT part of a pool, then it either begins using the instance or destroys it.



```
void APooledCharacter::SetUsed(bool bUsed)
{
    if (bUsingPooledMethod) {

        bIsUsed = bUsed;

        SetActorTickEnabled(bUsed);

        SetActorEnableCollision(bUsed);

        RootComponent->SetVisibility(bUsed, true);

        GetWorld()->GetTimerManager().ClearTimer(Timer);
        Timer.Invalidate();

        if (bUsed) {

            GetWorld()->GetTimerManager().SetTimer(Timer, this, &APooledCharacter::CallSetUsedFromTimer, Lifetime, false);

            BeginUsed();
        }

        else {

            EndUsed();
        }

    }
    else {
        if (bUsed) {
            BeginUsed();
        }
        else {
            UE_LOG(LogTemp, Log, TEXT("Destroying PooledCharacter"));
            Destroy();
        }
    }


}
```

void BeginUsed() and void EndUsed()

BeginUsed() registers the APooledCharacter instance to all subsystems that need to know what objects are being used. An example of this would be the SavingSubsystem, which needs a reference to the object so that it gets the data that needs to be saved.

EndUsed() does the opposite; it delists the APooledCharacter instance from all the subsystems that may call to it. Afterwards, it either clears the data from pooled instances via ClearDataInBP() if it's a pooled instance or outright destorys it if it isn't pooled.

void ClearDataInBP()

ClearDataInBP() is a Blueprint Implementable Event that clears all data that the pooled APooledCharacter instance is storing. Since there are variables at the blueprint level that need to be reset, children of APooledCharacter make the logic while the parent only calls the function. Keep in mind that ClearDataInBP() cannot be called by blueprint, only give it implementation.

void SaveData() and TInstancedStruct<FNPCSoul> MakeDataForManagerSubsystem()

SaveData() is an interface function from the ISaveable interface. It is used by the SavingSubsystem to make all saveable objects start its saving logic. APooledCharacter uses it to make the struct that holds the data that needs to be saved and sends it to the SavingSubsystem for it to be saved.

MakeDataForManagerSubsystem() is the Blueprint Implemented (and Blueprint Callable) event that makes the struct that is saved by the SavingSubsystem. The blueprint children make the struct type that they need to save, then wrap it in an Instanced Struct so that different struct types can be saved under the same variable.