Unreal Engine 4

Network Replication

Reference documentation for Udemy course
“Unreal Engine 4 Mastery: Create Multiplayer Games”
by Tom Looman

BeginPlay and some notes

Runs once for server and once for client.

Use PlayerStart entity with AutoReceiveInput to Player 0 or Player 1 for setting the players spawn position and pawn posses logic.

GameMode

Does not exist on a client to be fetched and used, it’s only instance is running on the server.

Can’t replicate to clients so it’s a bad place to try and add replication logic.

```
AFPSGameMode* gm = Cast<AFPSGameMode>(GetWorld()->GetAuthGameMode());
if (gm) {  // would be nullptr (false) on client side
      gm->CompleteMission(myPawn, true);
}
```

GameState

Child component of GameMode

Can be invoked on client side.

```
AFPSGameState* gs = Cast<AFPSGameState>(GetWorld()->GetGameState());
```

Or from GameMode once we set GameStateClass = AFPSGameState::StaticClass() in the constructor.

```
AFPSGameState* gs = GetGameState<AFPSGameState>();
```

Can be used to call Multicast delegate functions.

Header:

```
UFUNCTION(NetMulticast, Reliable)
void MulticastOnMissionComplete(APawn* InstigatorPawn, bool bMissionSuccess);
```

Code:

```
void AFPSGameState::MulticastOnMissionComplete_Implementation(APawn* InstigatorPawn, bool
bMissionSuccess) {
    for (FConstPawnIterator it = GetWorld()->GetPawnIterator(); it; it++) {
        APawn* pawn = it->Get();
        if (pawn && pawn->IsLocallyControlled())
            pawn->DisableInput(nullptr);
    }
}
```

Variable Replication

Header:

```
UPROPERTY(Replicated, EditAnywhere, BlueprintReadWrite, Category =  "Gameplay")
bool bIsCarryingObjective;
```

Code:

```
#include "Net/UnrealNetwork.h"

AFPSCharacter::AFPSCharacter()
{
    SetReplicates(true);
}

void AFPSCharacter::GetLifetimeReplicatedProps(TArray<FLifetimeProperty> & OutLifetimeProps)
const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(AFPSCharacter, bIsCarryingObjective);
    // make server replicate to owner client only
    //DOREPLIFETIME_CONDITION(AFPSCharacter, bIsCarryingObjective, COND_OwnerOnly);
}
```

Variable Replication
Continued

Here is a quick glance at the list of conditions currently supported:

- **COND_InitialOnly** - This property will only attempt to send on the initial bunch
- **COND_OwnerOnly** - This property will only send to the actor's owner
- **COND_SkipOwner** - This property send to every connection EXCEPT the owner
- **COND_SimulatedOnly** - This property will only send to simulated actors
- **COND_AutonomousOnly** - This property will only send to autonomous actors
- **COND_SimulatedOrPhysics-** This property will send to simulated OR bRepPhysics actors
- **COND_InitialOrOwner** - This property will send on the initial packet, or to the actors owner
- **COND_Custom** - This property has no particular condition, but wants the ability to toggle on/off via SetCustomIsActiveOverride

https://www.unrealengine.com/en-US/blog/network-tips-and-tricks

Variable ReplicateUsing

Header:

```
UPROPERTY(ReplicatedUsing = OnRep_GuardState)
EAIState GuardState;

void AAIGuard::OnRep_GuardState();

// optional, raise event in blueprints
UFUNCTION(BlueprintImplementableEvent, Category = "AI")
void OnStateChanged(EAIState newState);
```

Code:

```
#include "Net/UnrealNetwork.h"

void AAIGuard::OnRep_GuardState() {
    // call said event in blueprint
    OnStateChanged(GuardState);
}
void AAIGuard::GetLifetimeReplicatedProps(TArray<FLifetimeProperty> & OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(AAIGuard, GuardState);
```

Run Code on Server Only

```
void AObjectiveActor::NotifyActorBeginOverlap(AActor * OtherActor)
{
    Super::NotifyActorBeginOverlap(OtherActor);

    // Client - [ROLE_SimulatedProxy | ROLE_AutonomousProxy]
    // Server - [ROLE_Authority]
    // Role - this entity, RemoteRole - the other entity
    if (Role == ROLE_Authority) { // if we are the server entity
        auto otherChar = Cast<AFPSCharacter>(OtherActor);
        if (otherChar) {
            otherChar->bIsCarryingObjective = true;
            Destroy();
        }
    }
}
```

https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/Roles/

Global Fetch Methods

Called from GameMode to get a reference list of a certain type of actors:

```
TSubclassOf<AActor> SpectatingViewpointClass;  // in header

TArray<AActor*> returnedActors;
UGameplayStatics::GetAllActorsOfClass(this, SpectatingViewpointClass, returnedActors);
```

To get all player controllers we call off of GetWorld, GetPlayerControllerIterator:

```
for (auto it = GetWorld()-> GetPlayerControllerIterator(); it; it++) {
  APlayerController* pc = it->Get();
  pc->SetViewTargetWithBlend(newViewTarget, 2.5f,
EViewTargetBlendFunction::VTBlend_Cubic);
}
```

Logging

```
UE_LOG(LogTemp, Warning, TEXT( "Your message" ));

UE_LOG(LogTemp, Warning, TEXT( "UMyClass %s entering FireWeapon()" ),
      *GetNameSafe(this));
```

https://wiki.unrealengine.com/Logs,_Printing_Messages_To_Yourself_During_Runtime#Quick_Usage

Useful resources

UE4 C++ coding standard
https://docs.unrealengine.com/latest/INT/Programming/Development/CodingStandard/