Exercice3 : Réseau

Sélectionner branche 'exercice3' comme point de départ.

Comparer avec branche 'master' pour la version complétée.

Noter le mauvais comportement de l'instance client et le bon comportement de l'instance listen-server

Adapter UPotatoPlantingComponent au réseau

- Remarquer que le client est incapable de planter une potato alors que le listen-server peut le faire
- Rendre le component répliqué

```
UPotatoPlantingComponent()
{
    // Invoquer SetIsReplicatedByDefault(true) pour que
        // les instances soient répliquées
}
```

- Créer un RPC serveur pour planter une potato

```
// En-tête
UFUNCTION(Server, Reliable)
void Server_PlantPotato();

// Définition
void Server_PlantPotato_Implementation()
{
    // Invoquer UPotatoPlantingComponent::Authority_PlantPotato()
}
```

- Rebinder l'input joueur sur le RPC

```
void OnSetupPlayerInput(UInputComponent* inputComponent)
{
    // Binder "Fire" sur UPotatoPlantingComponent::Server_PlantPotato au ...
        // lieu de UPotatoPlantingComponent::Authority_PlantPotato
}
```

- Remarquer que le client peut planter des potatoes

Adapter UPotatoPickUpComponent au réseau

- Remarquer que le client est incapable de ramasser une potato alors que le listen-server peut le faire
- Rendre le component répliqué

```
UPotatoPickUpComponent()
{
    /// Invoquer SetIsReplicatedByDefault(true) pour que
        // les instances soient répliquées
}
```

- Créer un RPC serveur pour déposer une potato

```
// En-tête
UFUNCTION(Server, Reliable)
void Server_DropPotato()

// Définition
void Server_DropPotato_Implementation()
{
    // Invoquer UPotatoPickUpComponent::Authority_DropPotato()
}
```

- Rebinder l'input du joueur sur le RPC

```
void OnSetupPlayerInput(UInputComponent* inputComponent)
{
    // Binder "Release" sur UPotatoPickUpComponent::Server_DropPotato) au ...
        // lieu de UPotatoPickUpComponent::Authority_DropPotato
}
```

- Tester le client. Remarquer l'important problème de mouvement lorsqu'une potato est ramassée.
- Répliquer la potato tenue
    - Anoter l'attribut Replicated à HeldPotato

```
UPROPERTY(Transient, Replicated)
APotato* _heldPotato = nullptr;
```

    - Ajouter l'attribut HeldPotato à la liste de réplication

```
void GetLifetimeReplicatedProps(TArray< FLifetimeProperty >&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(UPotatoPickUpComponent, _heldPotato);
}
```

- Réagir côté client à la réplication en ajoutant ReplicatedUsing

```
UPROPERTY(Transient, Replicated, ReplicatedUsing = OnRep_HeldPotato)
APotato* _heldPotato = nullptr;
```

- Définir OnRep_HeldPotato

```
void OnRep_HeldPotato(APotato* old)
{
    // Invoquer OnUpdate_HeldPotato(old) pour informer la classe ..
        // du changement de valeur
}
```

- Remarquer que le client peut ramasser et déposer des potatoes

Adapter UPotatoEatingComponent au réseau

- Remarquer que le client est incapable de manger une potato alors que le listen-server peut le faire
- Rendre le component répliqué

```
UPotatoEatingComponent()
{
   // Invoquer SetIsReplicatedByDefault(true) pour que
        // les instances soient répliquées
}
```

- Créer un RPC pour manger la potato tenue

```
// En-tête
UFUNCTION(Server, Reliable)
void Server_EatHeldPotato();

// Définition
```

```
void Server_EatPotato_Implementation(APotato* potato)
{
   // Invoquer UPotatoEatingComponent::Authority_EatPotato(potato)
}
```

- Rebinder l'input du joueur sur le RPC

```
void OnSetupPlayerInput(UInputComponent* inputComponent)
{
    // Binder "Fire" sur UPotatoEatingComponent::Server_EatHeldPotato) au ...
        // lieu de UPotatoPickUpComponent::Authority_EatHeldPotato
}
```

- Répliquer CaloriesEaten
  - Anoter CaloriesEaten par Replicated

```
UPROPERTY(Transient, Replicated)
float _caloriesEaten;
```

  - Ajouter CaloriesEaten à la liste de réplication

```
void GetLifetimeReplicatedProps(TArray< FLifetimeProperty >&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(UPotatoEatingComponent, _caloriesEaten);
}
```

- Remarquer que le client peut manger des potatoes

Adapter APotatoPlayerState au réseau

- Remarquer que le client ne connait pas son propre rôle
- Répliquer CurrentRole
  - Anoter CaloriesEaten par Replicated

```
UPROPERTY(Transient, Replicated)
float _currentRole;
```

  - Ajouter CurrentRole à la liste de réplication

```
void GetLifetimeReplicatedProps(TArray< FLifetimeProperty >&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(APotatoPlayerState, _currentRole);
}
```

Adapter APotatoPlayerController au réseau

- Remarquer que le client est incpable pas changer de rôle ou quitter
- Créer un RPC pour changer de rôle

```
// En-tête
UFUNCTION(Server, Reliable)
void Server_ChangeRole();

// Définition
void Server_ChangeRole_Implementation()
{
    // Invoquer APotatoPlayerController::Authority_ChangeRole()
}
```

- Créer un RPC pour quitter la partie

```
// En-tête
UFUNCTION(Server, Reliable)
void Server_QuitGame();

// Définition
void Server_QuitGame_Implementation()
{
    // Invoquer APotatoPlayerController::Authority_QuitGame()
}
```

- Rebinder les inputs aux RPCs

```
void SetupInputComponent()
{
    // Binder "Switch" sur APotatoPlayerController::Server_ChangeRole)
    // Binder "Quit" sur APotatoPlayerController::Server_QuitGame)
}
```

- Remarquer que le client peut maintenant changer de rôle et quitter

Adapter APotatoGameState au réseau

- Noter que le client n'est pas informé de la fin de la partie
- Répliquer GameEnded
    - Anoter GameEnded par Replicated

```
UPROPERTY(Transient, Replicated)
bool _gameEnded = false;
```

    - Ajouter GameEnded à la liste de réplication

```
void GetLifetimeReplicatedProps(TArray< FLifetimeProperty >&
OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps(OutLifetimeProps);
    DOREPLIFETIME(APotatoGameState, _gameEnded);
}
```

- Remarquer que le client est maintenant informé de la fin de la partie