Introduction

(NB : J'espère ce tutoriel vous sera utile et qu'il sera compréhensible par tous. Si vous avez des questions n'hésitez pas à m'envoyer un message sur Facebook « Antoine Gargasson » ou par mail antoine.gargasson@gmail.com)

(NB2 : Ce tuto est réalisé sur la version 4.8.3 de UE4)

Dans ce premier tutoriel, nous allons réaliser un petit jeu basique en C++.
L'objectif de ce tuto est que vous sachiez comment créer un déplacement, gérer des statistiques de joueurs, créer un tir, effectuer du partage de variables et gérer des scores de joueurs. Tout ceci évidemment en **multijoueur** !

Je vous encourage vivement à lire le tuto 0 sur le multijoueur car il contient tous les termes et les bases concernant le multijoueur sur UE4 ainsi que les tips et conseils sur UE4.

Avant de vous lancer dans un jeu multijoueur, je vous recommande d'avoir déjà pratiqué du blueprint ou du C++ sur des jeux solo ou même d'avoir lu des tuto sur le solo.

Préparation

Tout d'abord, commencez par **créer** un **projet vide sans** le **Starter Content**. Comme à notre habitude, nous allons tout faire de 0.

J'ai nommé mon projet **TutoMultijoueur**.

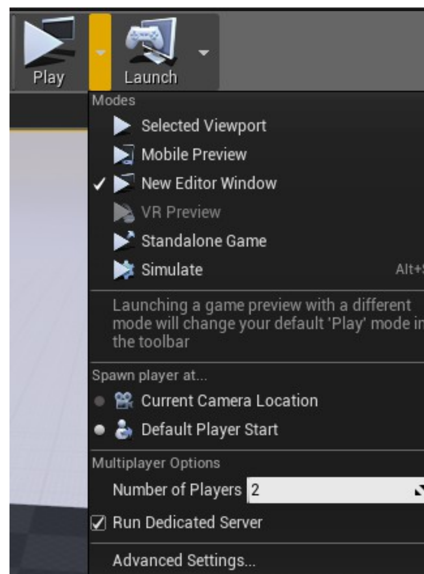
J'ai également **créé** le dossier **AddingContent** dans le **Content** afin de repérer tout ce que créer moi même et que ça soit plus facile à retrouver.

Dans le dossier **AddingContent**, créez **2** sous **dossiers Blueprint** et **UMG**.

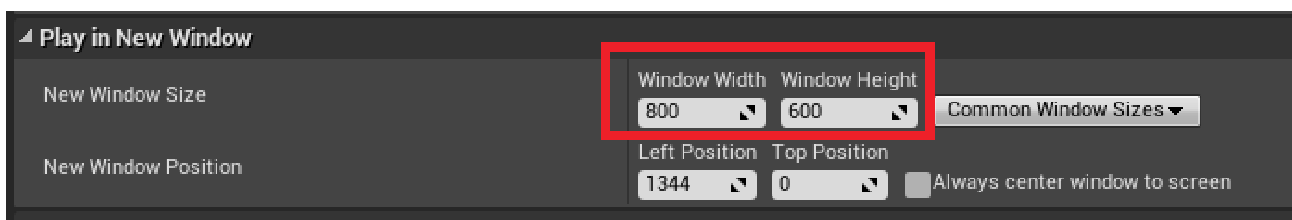
Avant de commencer le vif du sujet, nous allons régler encore le multijoueur.

En **cliquant** sur la petite **flèche** à côté du bouton **Play**, vous allez dérouler les **options** de **lecture**.

Remplissez les comme suit :



Dans **Advanced Settings**, Vous allez modifier la **taille** de vos **fenêtres** :



Si vous lancez maintenant vous devriez avoir 2 fenêtres pas trop grosses où vous pouvez voir tout ce qu'il se passe sur chacune d'elles. En vous déplaçant vous constaterez que vous pouvez bouger les deux caméras.

Pour continuer, **créez** une nouvelle **classe** C++ (File->New c++ class) de type **Character** et nommez-là **TutoPlayer**.

Créez-en une nouvelle de **type PlayerState** et nommez là **TutoPlayerState**. Créez-en une dernière de **type GameMode** et nommez là **TutoGameMode**.

Téléchargez ensuite les objets à ce lien :

https://drive.google.com/folderview?id=0B7q9BoXd72XfSzZaRUMzSGRHN1U&usp=sharing

Vous trouverez 2 modèles ainsi que des animations de première personne.

Ouvrez votre **visual studio** et **trouvez** votre **fichier header** de projet "NomDeProjet.h". Pour moi ça sera " TutoMultijoueurC.h".

Mettez vos **header** et **rajoutez** les **include** du **Network** pour obtenir ceci :

TutoMultijoueurC.h

```
#include "Engine.h"
#include "TutoPlayer.h"
#include "TutoGameMode.h"
#include "UnrealNetwork.h"
#include "TutoPlayerState.h"
#include "EngineUtils.h"
```

Vous pouvez maintenant passer à la suite :)

Le personnage

Je vais commencer par faire tout le personnage, vous n'aurez plus besoin de revenir dessus ensuite ;)

Dans **l'éditeur**, **Rajoutez** les **bind** suivants (**Edit->Project Settings**) :

Action(Fire, LeftMouseButton)
Action(Jump, SpaceBar)
Action(ShowScore, Tab)

Axis(Forward, Z, 1.0)
Axis(Forward, S, -1.0)
Axis(Right, D, 1.0)
Axis(Right, Q, -1.0)
Axis(LookUp, MouseY, -1.0)
Axis(LookAround, MouseX, 1.0)

Un fois ces bind créés, vous pouvez **retourner** dans **visual studio**.

Ouvrez votre **TutoPlayer.h** :

De base vous avez **4 fonctions** dans un **character** :
- Le constructeur : **ATutoPlayer**()
- Le début de jeu : **BeginPlay**()
- La boucle de jeu : **Tick**(...)
- Le bind des touches : **SetupPlayerInputComponent(…)**

Toutes ces **fonctions** doivent être **public**.

Ici vous pouvez rajouter les deux fonctions de déplacement :

TutoPlayer.h
```
// Handles the forward axis
void Forward(float Value);

// Handles the right axis
void Right(float Value);
```

Créez un deuxième constructeur pour le TutoJoueur. Ce constructeur utilise l'initialisateur d'objet :

TutoPlayer.h
```
// Constructor for AFPSCharacter
ATutoPlayer(const FObjectInitializer& ObjectInitializer);
```

On va pouvoir rajouter les **fonctions** de **saut** :

TutoPlayer.h
```
//sets jump flag when key is pressed
UFUNCTION()
void OnStartJump();
//clears jump flag when key is released
UFUNCTION()
void OnStopJump();
```

On va également créer **deux fonctions** qui vont servir à **montrer** les **scores** :

TutoPlayer.h
```
// Show scores
void ShowScores();

// Stop Show scores
void StopShowScores();
```

Passons maintenant à la partie réseau ! :D

Quand vous voulez **appeler** une **fonction** sur le **serveur**, vous devez faire une **fonction** sur le **client** qui **appellera** la **fonction** sur le **serveur**. La **fonction** sur le **serveur** nécessite une **implémentation** et une **validation** sous cette forme :

Fonction_Implementation et **Fonction_Validate**.

Commençons par le **tir**. Il sera effectué sur le **côté serveur** car c'est le **serveur** qui **sait** exactement **où** sont les **joueurs**.

Créez une fonction sur le client et une fonction sur le serveur. On a donc une chose comme ceci :

TutoPlayer.h
```
// Handles the Fire on the client
void Fire();

// Handles the Fire on the server
UFUNCTION(Reliable, Server, WithValidation)
void ServerFire();
virtual void ServerFire_Implementation();
virtual bool ServerFire_Validate();
```

La fonction **ServerFire**() est donc **envoyée** sur le **serveur** avec **validation** et sera forcément **exécutée** (**reliable**).

De même pour un **affichage** quand on **touche** un **joueur** :

TutoPlayer.h

```
void MulticastDebug();

// Handles the Fire on the server
UFUNCTION(Reliable, NetMulticast, WithValidation)
void ServerMulticastDebug();
virtual void ServerMulticastDebug_Implementation();
virtual bool ServerMulticastDebug_Validate();
```

Même fonctionnement que pour le **ServerFire** sauf que ici on va **envoyer** à **tous** les **clients l'implémentation** de la **fonction**.

On va également créer une **fonction** qui **reçois** les **tirs** et qui **envoi** une **notification** à tous les **joueurs** :

TutoPlayer.h

```
// Handles the Fire on the server
void TakeHit(int32 Damage, ATutoPlayer* Causer);

// Handles the Fire on the server (server side)
UFUNCTION(Reliable, NetMulticast, WithValidation)
void ServerTakeHit(int32 Damage, ATutoPlayer* Causer);
virtual void ServerTakeHit_Implementation(int32 Damage, ATutoPlayer*
Causer);
virtual bool ServerTakeHit_Validate(int32 Damage, ATutoPlayer* Causer);
```

On fini avec une **fonction multicast** pour **reset** les **stats** du joueur quand il **meurt** :

TutoPlayer.h

```
// Reset Stats of the player
void ResetStats();

// Reset Stats of the player (server side)
UFUNCTION(Reliable, NetMulticast, WithValidation)
void ServerResetStats();
virtual void ServerResetStats_Implementation();
virtual bool ServerResetStats_Validate();
```

Pour finir avec les **fonctions**, créez une **fonction** qui sera un **événement écrivable** en **blueprint** et qui servira à **rafraîchir l'affichage** du **HUD** :

TutoPlayer.h

```
// Refresh the HUD of scores
UFUNCTION(BlueprintImplementableEvent)
void RefreshScores();
```

Nous allons à présent passer sur les **propriétés** du **joueur**, j'espère que vous suivez toujours ;)

Nous allons créer **plusieurs composants** : une **caméra** pour la **1ere personne** ainsi que un **skeletal mesh** pour la **1ere personne**.
Puis nous allons créer **deux propriétés** utiles : la **longueur** du **tir** ainsi qu'une variable qui nous dira si on **affiche** les **scores** ou non.
Pour finir nous ajouterons **deux variables répliquées** qui seront utiles pour le tir : la **vie** et les **dommages**.

TutoPlayer.h

```
// First person camera
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Camera)
UCameraComponent* FirstPersonCameraComponent;

// Pawn mesh: 1st person view (arms; seen only by self)
UPROPERTY(VisibleDefaultsOnly, Category = Mesh)
USkeletalMeshComponent* FirstPersonMesh;

// Distance for the hit when fire
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = Camera)
float FireLength;

// Showing score
UPROPERTY(BlueprintReadOnly, Category = Scores)
bool ShowingScore;

// Health of the player
UPROPERTY(Replicated)
int32 Health;

// Damage of the player
UPROPERTY(Replicated)
int32 Damage;
```

Nous allons pouvoir passer au fichier **TutoPlayer.cpp**. C'est la plus **grosse partie**.
Accrochez vous, vous en verrez le bout rapidement ;) Je vais essayer de vous donner un **maximum d'explications** !

Commençons par le **constructeur sans paramètres**. Il sert à **initialiser** les **variables**.

TutoPlayer.cpp

```
// Sets default values
ATutoPlayer::ATutoPlayer()
{
	// Set this character to call Tick() every frame.
	PrimaryActorTick.bCanEverTick = true;

	// Set this actor to be replicate.
	bReplicates = true;

	FireLength = 1000;
	Health = 100;
	Damage = 10;
}
```

Ici on dit au **joueur** qu'il doit se **répliquer** avec la variable **bReplicated** et on **initialise** les **variables**.

Passons au **constructeur** avec **paramètre**. Il va servir à **initialiser** les **composants**.

TutoPlayer.cpp

```
ATutoPlayer::ATutoPlayer(const FObjectInitializer& ObjectInitializer)
	: Super(ObjectInitializer)
{
	// Create a CameraComponent
	FirstPersonCameraComponent =
ObjectInitializer.CreateDefaultSubobject<UCameraComponent>(this,
TEXT("FirstPersonCamera"));
	FirstPersonCameraComponent->AttachParent = GetCapsuleComponent();
	// Position the camera a bit above the eyes
	FirstPersonCameraComponent->RelativeLocation = FVector(0, 0,
BaseEyeHeight);
	// Allow the pawn to control rotation.
	FirstPersonCameraComponent->bUsePawnControlRotation = true;

	// Create a mesh '1st person' view (when controlling this pawn)
	FirstPersonMesh =
ObjectInitializer.CreateDefaultSubobject<USkeletalMeshComponent>(this,
TEXT("FirstPersonMesh"));
	// only the owning player will see this mesh
	FirstPersonMesh->SetOnlyOwnerSee(true);
	FirstPersonMesh->AttachParent = FirstPersonCameraComponent;
	FirstPersonMesh->bCastDynamicShadow = false;
	FirstPersonMesh->CastShadow = false;

	// everyone but the owner can see the regular body mesh
	GetMesh()->SetOwnerNoSee(true);

}
```

Ici on **créé** une **caméra** et on **l'appelle FirstPersonCamera**.
On **l'attache** ensuite à la **capsule component** qui est le **composant principale** du **joueur**.
On **positionne** la **caméra** en **hauteur** à la **BaseEyeHeight** qui est une variable utile car elle **localise l'œil** du **modèle** à quelque chose près.
Pour finir avec la **caméra**, on lui dit qu'elle doit **utiliser** la **rotation** du **joueur**. C'est cette rotation qui va tourner la caméra sans bouger le pawn.

Ensuite on **créé** le **skeletal mesh** que l'on **nomme FirstPersonMesh**. Il faudra **l'affecter** dans le **blueprint**.
On dit ensuite qu'il faut que **seul** le **joueur** qui **possède** ce **character voit** le **FirstPersonMesh**.
On **attache** le **skeletal mesh** sur la **caméra**. Ainsi il **suivra** la **rotation** de la **caméra**.
On **retire** les **ombres** sur le **mesh** de **1ere personne** pour qu'elles ne soit pas visible.

Pour finir on dit au **Mesh** de **3eme personne** qu'il ne doit **pas** être **visible** par le **possesseur** du **character**.

C'est tout pour le constructeur et la création de composants. :)

Pour répliquer les variables, il ne suffit pas de le spécifier à la déclaration dans le .h, il faut aussi spécifier dans le .cpp qu'elle sont répliquées :

TutoPlayer.cpp

```
void ATutoPlayer::GetLifetimeReplicatedProps(TArray< FLifetimeProperty >
& OutLifetimeProps) const
{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);


	DOREPLIFETIME(ATutoPlayer, Health);
	DOREPLIFETIME(ATutoPlayer, Damage);
}
```

Vous l'aurez remarqué, cette **fonction n'apparaît pas** dans le **.h**, c'est normal, c'est une **fonction** qui est **par défaut** dans le **Character**. On dit à UE4 que les **variables Health** et **Damage** sont **répliquées** en **permanence**.

Le **BeginPlay** et la **fonction Tick** ne sont **pas** à **modifier**.

Vous pouvez ajouter comme moi un message affiché au début du jeu afin de voir si votre joueur est bien créé :

TutoPlayer.cpp

```
// Called when the game starts or when spawned
void ATutoPlayer::BeginPlay()
{
	Super::BeginPlay();

	if (GEngine)
	{
		GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Yellow,
TEXT("TEST MULTIPLAYER"));
	}
}
// Called every frame
void ATutoPlayer::Tick( float DeltaTime )
{
	Super::Tick( DeltaTime );

}
```

Passons maintenant à la **fonction** de **bind** de **touches** :

TutoPlayer.cpp

```
// Called to bind functionality to input
void ATutoPlayer::SetupPlayerInputComponent(class UInputComponent*
InputComponent)
{
	Super::SetupPlayerInputComponent(InputComponent);

	InputComponent->BindAxis("Forward", this, &ATutoPlayer::Forward);
	InputComponent->BindAxis("Right", this, &ATutoPlayer::Right);
	InputComponent->BindAxis("LookUp", this,
&ATutoPlayer::AddControllerPitchInput);
	InputComponent->BindAxis("LookAround", this,
&ATutoPlayer::AddControllerYawInput);

	InputComponent->BindAction("Fire", IE_Pressed, this,
&ATutoPlayer::Fire);
	InputComponent->BindAction("Jump", IE_Pressed, this,
&ATutoPlayer::OnStartJump);
	InputComponent->BindAction("Jump", IE_Released, this,
&ATutoPlayer::OnStopJump);
	InputComponent->BindAction("ShowScore", IE_Pressed, this,
&ATutoPlayer::ShowScores);
	InputComponent->BindAction("ShowScore", IE_Released, this,
&ATutoPlayer::StopShowScores);

}
```

Si vous avez déjà fait un peu de C++ sur UE4, c'est un classique. On **assigne** aux **bind** créés sur UE4 des **fonctions**.
N'oubliez pas **d'utiliser** le **Super** avant de faire quoi que ce soit !

Passons aux deux fonctions de déplacement :

TutoPlayer.cpp

```
void ATutoPlayer::Forward(float Value)
{
	if (Controller != NULL && Value != 0.0f)
	{
		FRotator Rotation = Controller->GetControlRotation();
		if (GetCharacterMovement()->IsMovingOnGround() ||
GetCharacterMovement()->IsFalling())
		{
			Rotation.Pitch = 0.0f;
		}
		const FVector Direction =
FRotationMatrix(Rotation).GetScaledAxis(EAxis::X);
		AddMovementInput(Direction, Value);
	}
}

void ATutoPlayer::Right(float Value)
{
	if (Controller != NULL && Value != 0.0f)
	{
		FRotator Rotation = Controller->GetControlRotation();
		const FVector Direction =
FRotationMatrix(Rotation).GetScaledAxis(EAxis::Y);
		AddMovementInput(Direction, Value);
	}
}
```

On vérifie sur les **deux fonctions** s'il **existe** un **contrôleur** et si on **appuie** sur la **touche** correspondante.
On **récupère** ensuite la **rotation** du **joueur**. On **créé** une **matrice** de **rotation** grâce à cette **rotation** et on **récupère** un **axe particulier** (**X** pour aller vers **l'avant** et **Y** pour aller vers la **droite**).
On fini en **ajoutant** le **mouvement** dans la **direction** souhaitée avec la bonne **valeur**.

Pour la fonction **forward**, on **vérifie** également si on est en **train** de **tomber** ou **d'avancer** et on **bloque** la **rotation**.

Pour sauter, utiliser les fonctions suivantes :

TutoPlayer.cpp

```
void ATutoPlayer::OnStartJump()
{
	bPressedJump = true;
}
void ATutoPlayer::OnStopJump()
{
	bPressedJump = false;
}
```

Le **character inclus** déjà une **variable** de **saut**. On **réutilise** donc cette **variable** qui va nous **faciliter** la **vie** ;)

On va finir dans le facile avec les **fonctions** qui **affichent** le **score** :

TutoPlayer.cpp

```
void ATutoPlayer::ShowScores()
{
	ShowingScore = true;
	RefreshScores();
}


void ATutoPlayer::StopShowScores()
{
	ShowingScore = false;
}
```

Commençons les fonctions multijoueurs par le **reset** des **stats** du joueur quand il **meurt** :

TutoPlayer.cpp

```
void ATutoPlayer::ResetStats()
{
	if (Role == ROLE_Authority)
	{
		Health = 100;
	}
	else if (Role < ROLE_Authority)
	{
		ServerResetStats();
	}
}

void ATutoPlayer::ServerResetStats_Implementation()
{
	ResetStats();
}

bool ATutoPlayer::ServerResetStats_Validate()
{
	return true;
}
```

Dans la fonction **ResetStats**, on **regarde** si on **est** le **serveur**, si c'est le cas on **remonte** la **vie** à **100** et si on est **pas** le **serveur**, on **appelle** la **fonction ServerResetStats**.
Le **serveur** va donc **appeler** la **fonction ResetStat** et **changera** donc la **vie** du joueur.
Pour la **validation**, faites un **return true** à **chaque** fois.

Voici comment se passe le tir côté seveur :

TutoPlayer.cpp

```
void ATutoPlayer::Fire()
{
	if (Role < ROLE_Authority)
	{
		ServerFire();
	}
}

bool ATutoPlayer::ServerFire_Validate()
{
	return true;
}

void ATutoPlayer::ServerFire_Implementation()
{
	//location the PC is focused on
	const FVector Start = FirstPersonCameraComponent->GetComponentLocation();

	//1000 units in facing direction of PC (1000 units in front of the camera)
	const FVector End = Start + (FirstPersonCameraComponent->GetForwardVector() * FireLength);

	FHitResult HitInfo;
	FCollisionQueryParams QParams;
	ECollisionChannel Channel = ECollisionChannel::ECC_Visibility;
	FCollisionQueryParams OParams = FCollisionQueryParams::DefaultQueryParam;

	if (GetWorld()->LineTraceSingleByChannel(HitInfo, Start, End, ECollisionChannel::ECC_Visibility))
	{
		auto MyPC = Cast<ATutoPlayer>(HitInfo.GetActor());
		if (MyPC) {
			GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Blue, TEXT("" + PlayerState->PlayerName + " hits " + MyPC->PlayerState->PlayerName));
			MulticastDebug();
			MyPC->TakeHit(Damage, this);
		}
	}
}
```

Tout d'abord on **vérifie** dans le **Fire** si on **est** le **serveur**. Si c'est pas le cas, on **demande** au **serveur** de **lancer l'implémentation** du **Fire**.
On **récupère** la **position** de la **caméra** dans le **monde**. On utilise la position de la caméra et on lui **ajoute 1000 unités** vers **l'avant**. Ça sera la **fin** de la **ligne** de **tir**.
On **initialise** des **paramètres** de **réceptions** de **collisions**, des **paramètres** sur les **objets** à **toucher**.
On **tente** de **lancer** une **ligne** de **tir** en utilisant les paramètres créés. S'il le **tir réussi**, on **caste l'objet** touché en **TutoPlayer** pour s'assurer qu'on **touche** un **joueur**.

Si c'est bien un joueur qu'on **touche** on **affiche** un **message**, on **utilise** la **méthode d'envoi** de message à tous le joueurs (**MulticastDebug**) et on **envoi** le **hit** au **joueur** concerné.

Concernant le message à tous les joueurs, on effectue les **actions suivantes** :

TutoPlayer.cpp

```
void ATutoPlayer::MulticastDebug()
{
	if (Role == ROLE_Authority)
	{
		if (GEngine)
		{
			GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Blue, 
TEXT("Hit SomeOne !"));
		}
	}
}

void ATutoPlayer::ServerMulticastDebug_Implementation()
{
	MulticastDebug();
}

bool ATutoPlayer::ServerMulticastDebug_Validate()
{
	return true;
}
```

On **vérifie** si on **est** le **serveur**, si c'est le cas, on **envoi** un **message** à **tous** les **joueurs**.
Dans **l'implémentation**, on **rappelle** la **fonction**.

Il reste à écrire la fonction de **prise** de **dommages** :

On **vérifie** au début, si on **est client**, on **appelle** la **fonction** sur le **serveur**.
Si on **est serveur**, on **affiche** un **message** et on **vérifie** si le **joueur** n'a **plus** de **vie**.
S'il n'a **plus** de **vie**, on **ajoute** un **kill** sur le **PlayerState** du **joueur** qui **envoi** les **dommages** et on lui **ajoute** du **score**.
On **ajoute** une **death** sur le **PlayerState** du **joueur** qui **reçois**. On fini par **écrire** un **message**.
Pour finir on **récupère** tous les **PlayerStart** sur la **carte**.
On en **choisi** un au **hasard**, on **change** la **position** et la **rotation** du **joueur**.
Pour finir on **reset** les **stats** du **joueur mort**.

TutoPlayer.cpp

```
void ATutoPlayer::TakeHit(int32 Damage, ATutoPlayer* Causer)
{
	Health -= Damage;

	if (Role < ROLE_Authority)
	{
		ServerTakeHit(Damage, Causer);
	}
	else if (Role == ROLE_Authority)
	{
		if (GEngine)
		{
			GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Blue, 
TEXT("" + PlayerState->PlayerName + " life : " + 
FString::FromInt(Health)));
		}
		if (Health <= 0)
		{
			auto MyPS = Cast<ATutoPlayerState>(Causer-
>PlayerState);
			if (MyPS)
			{
				MyPS->TheKills += 1;
				MyPS->TheScore += 100;
			}
			auto ThePS = Cast<ATutoPlayerState>(PlayerState);
			if (ThePS)
			{
				ThePS->TheDeath += 1;
			}
			if (GEngine)
			{
				GEngine->AddOnScreenDebugMessage(-1, 5.f, 
FColor::Blue, TEXT("Scores Updated !"));
			}

			TArray<APlayerStart*> Tab;
			//find all PlayerStart
			for (TActorIterator<APlayerStart> ActorItr(GetWorld()); 
ActorItr; ++ActorItr)
			{
				Tab.Add(*ActorItr);
			}
			int32 random = FMath::RandRange(0, Tab.Num()-1);
			SetActorLocation(Tab[random]->GetActorLocation());
			SetActorRotation(Tab[random]->GetActorRotation());
			ResetStats();
		}
	}
}
```

```
void ATutoPlayer::ServerTakeHit_Implementation(int32 Damage,
ATutoPlayer* Causer)
{
	TakeHit(Damage, Causer);
}

bool ATutoPlayer::ServerTakeHit_Validate(int32 Damage, ATutoPlayer*
Causer)
{
	return true;
}
```

C'est tout pour le **TutoPlayer** :)
N'hésitez pas à bien analyser le TutoPlayer. C'est la plus grosse classe et c'est ici qu'il va se passer beaucoup de choses. N'hésitez pas à poser des questions et à vous documenter sur des méthodes et variables.

Nous allons pouvoir **passer** sur le **TutoGameMode.h** .

Il vous suffit de **rajouter** le **constructeur** avec **paramètre** :

TutoGameMode.h

```
ATutoGameMode(const FObjectInitializer& ObjectInitializer);
```

On va **affecter** le **Pawn** par **défaut** dans le **TutoGameMode.cpp** :

TutoGameMode.cpp

```
ATutoGameMode::ATutoGameMode(const FObjectInitializer&
ObjectInitializer) :
Super(ObjectInitializer)
{
	DefaultPawnClass = ATutoPlayer::StaticClass();
}
```

C'est tout pour le **TutoGameMode**.
On a juste **remplacé** le **Pawn** par **défaut**, ce qui va nous permettre de **tester** et de **jouer** avec le bon **Character**.

Enchaînons avec le **PlayerState** :

Le **PlayerState** est un **objet** qui est **créé** pour **chaque joueurs** et qui est **répliqué** sur **tous** les **clients**.

Commençons par le **.h** :

On va **déclarer** des **variables** qui seront **répliquées**. Sur le **PlayerState**, vous devez **stocker** des **variables** qui **resteront** les **même** quand votre **TutoPlayer mourra**.
Nous allons donc stocker ses **kill**, ses **death**, son **score** et son **nom**.

TutoPlayerState.h

```
virtual void PostInitializeComponents();

UPROPERTY(BlueprintReadOnly, Replicated)
int32 TheDeath;

UPROPERTY(BlueprintReadOnly, Replicated)
int32 TheKills;

UPROPERTY(BlueprintReadOnly, Replicated)
int32 TheScore;

UPROPERTY(BlueprintReadOnly, Replicated)
FString TheName;
```

Comme vous pouvez le voir, j'ai également **rajouter** la **fonction PostInitializeComponents**() car il n'y a **pas** de **constructeur** sur le **PlayerState**.

Dans le **.cpp** :
On va **initialiser** les **variables** et dire à UE4 de **répliquer** les **variables**.

TutoPlayerState.cpp

```
void ATutoPlayerState::PostInitializeComponents()
{
	Super::PostInitializeComponents();

	TheDeath = 0;
	TheKills = 0;
	TheScore = 0;
	TheName = "Player";
}
```

Il reste ensuite à répliquer les variables :

TutoPlayerState.cpp

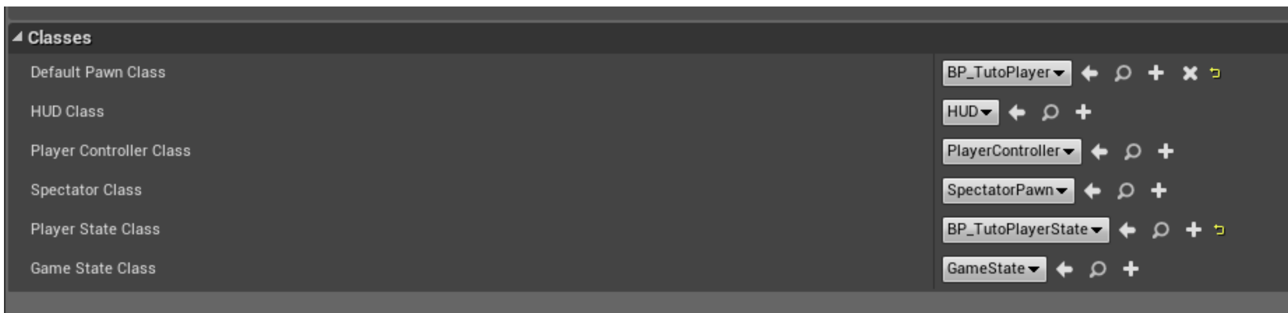
```
void ATutoPlayerState::GetLifetimeReplicatedProps(TArray<
FLifetimeProperty > & OutLifetimeProps) const
{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);

	DOREPLIFETIME(ATutoPlayerState, TheDeath);
	DOREPLIFETIME(ATutoPlayerState, TheKills);
	DOREPLIFETIME(ATutoPlayerState, TheScore);
	DOREPLIFETIME(ATutoPlayerState, TheName);
}
```

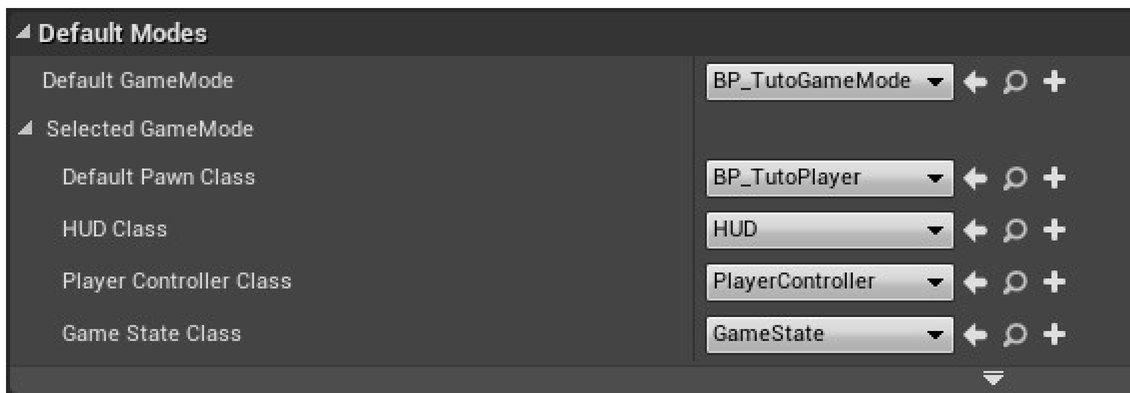
Vous avez donc **fini** de **coder** votre **jeu** !
Le **reste** se passera dans **l'éditeur** !
Vous pouvez **compiler** et **sauvegarder**. Si vous avez des **erreurs**, ne paniquez pas et **regardez** bien **l'erreur**. Si vous avez correctement recopié, essayer de **comprendre l'erreur** et dans le dernier des cas, vous pouvez **poser** vos **question** sur **Facebook** et/ou m'envoyer un **mail**.

De **retour** dans **l'éditeur**, Créez un **dossier** de **blueprint**.
Clique droit et créez un **nouveau blueprint** de **type TutoPlayer**. **Nommez** le **BP_TutoPlayer**. Créez également **deux** autres **blueprints**. Le premier sera de **type TutoGameMode** et il s'appellera **BP_TutoGameMode**. Le second sera de **type TutoPlayerState** et se **nommera BP_TutoPlayerState**.

Ouvrez le **BP_TutoGameMode** et **changez** le **pawn** par **défaut** en **BP_TutoPlayer** ainsi que le **Player State Class** en **BP_TutoPlayerState**.



Dans les Project **Settings** (Edit->Project Settings), **changez** le **game mode** par **défaut** par le vôtre (**BP_TutoGameMode**). **Vérifiez** bien que votre **Pawn Class** est la **bonne**.

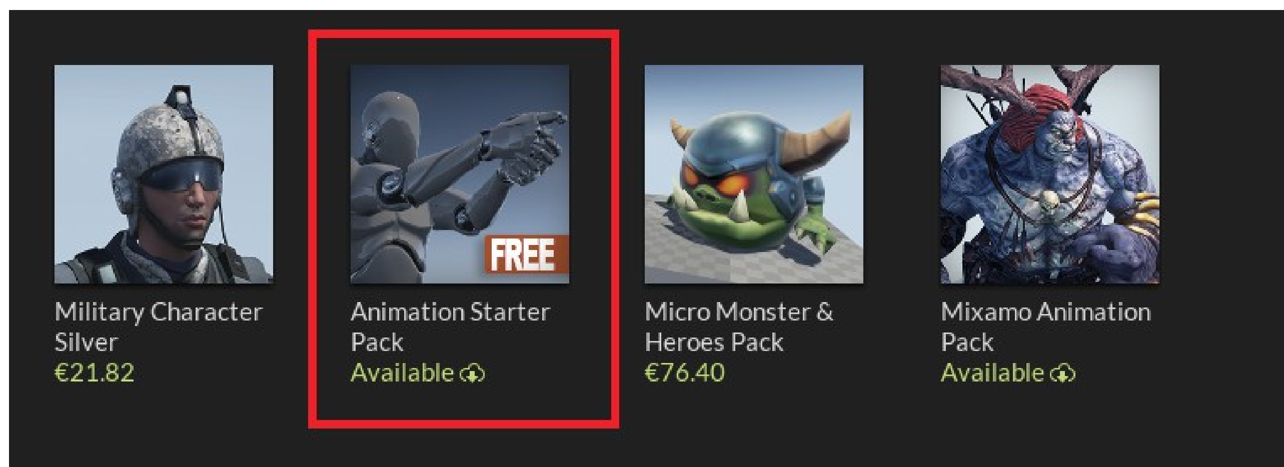


Créez vous un **dossier d'import**.
Dedans, **importez** le **GenericMale** pour les **tests.**
Importez le **HeroFPP** pour la **1ere personne**.
Importez les **animations** et **n'oubliez pas** de mettre en **skeleton** le **HeroFPP**.

Ouvrez votre **BP_TutoPlayer**. Il est possible que vous n'ayez **pas accès** à tous le **blueprint**, dans ce cas là **cliquez** sur **Open Full Blueprint Editor**.

Dans le **viewport**, **ajoutez** le **HeroFPP** sur le **FirstPersonMesh**. Nous allons faire l'animation pour la 1ere personne un peu plus bas.

Dans le **Market Place**, **retrouvez l'AnimStaterPack**. **Téléchargez** le et **importez** le dans votre **projet**.



C'est un **kit** qui vous sera très **utile** pour vos **futurs projets** ! Il **contient** un **personnage** de **base** et **beaucoup d'animations !**

De retour dans votre **BP_TutoPlayer**, mettez en **Mesh** le **SK_Mannequin** et en **Anim Blueprint Generated Class** le **UE4ASP_HeroTPP_AnimBlueprint**.
Positionnez le **Mesh** en 0, 0, -88 , Votre **CameraComponent** en 50, 0, 70 et votre **FirstPersonMesh** en 0, 0 -150 .

N'oubliez pas de **tourner** votre **Mesh** de **-90** pour qu'il soit dans le même **sens** que la **flèche bleue**.

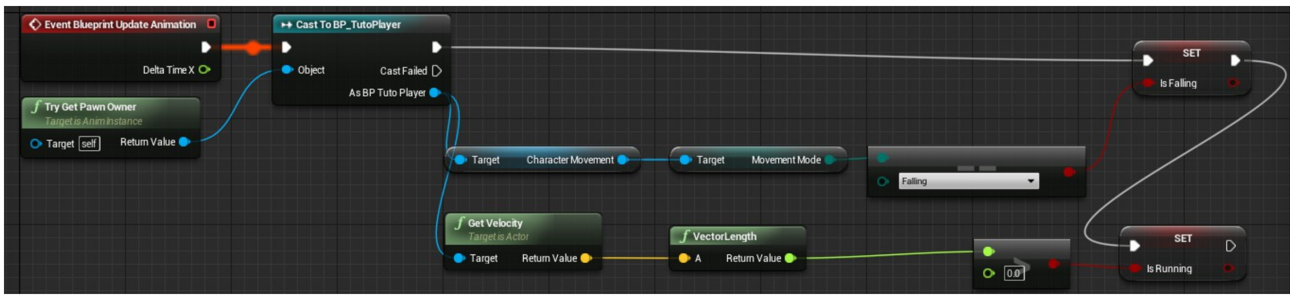
On va maintenant **créer** un **anim blueprint** avant de passer sur le HUD.

Créez un **dossier Animation**. Dedans, Clique **droit** , **animation**, **animation blueprint**. **Sélectionnez** le **HeroFPP_Skeleton** et faites **OK**. **Nommez** le **BP_AnimBlueprint**.

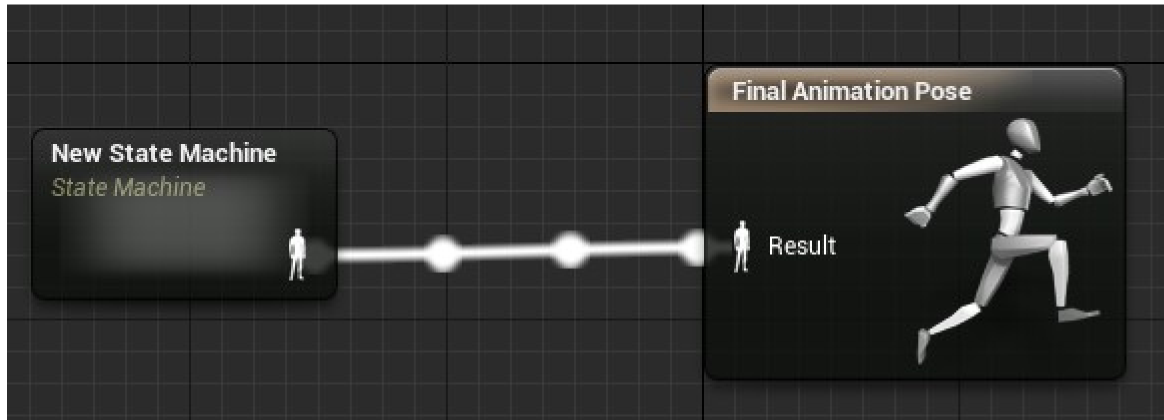
Ouvrez votre **anim blueprint** et **créez deux variables** de **type booléennes bIsFalling** et **bIsRunning**.

Dans l'event graph, on va gérer le cas où le joueur cours et quand il tombe.

Vous pouvez donc rajouter ce qui suit dans votre Event graph :



Dans **l'anim graph**, **créez** un **new state machine** (clique droit → new State Machine) :

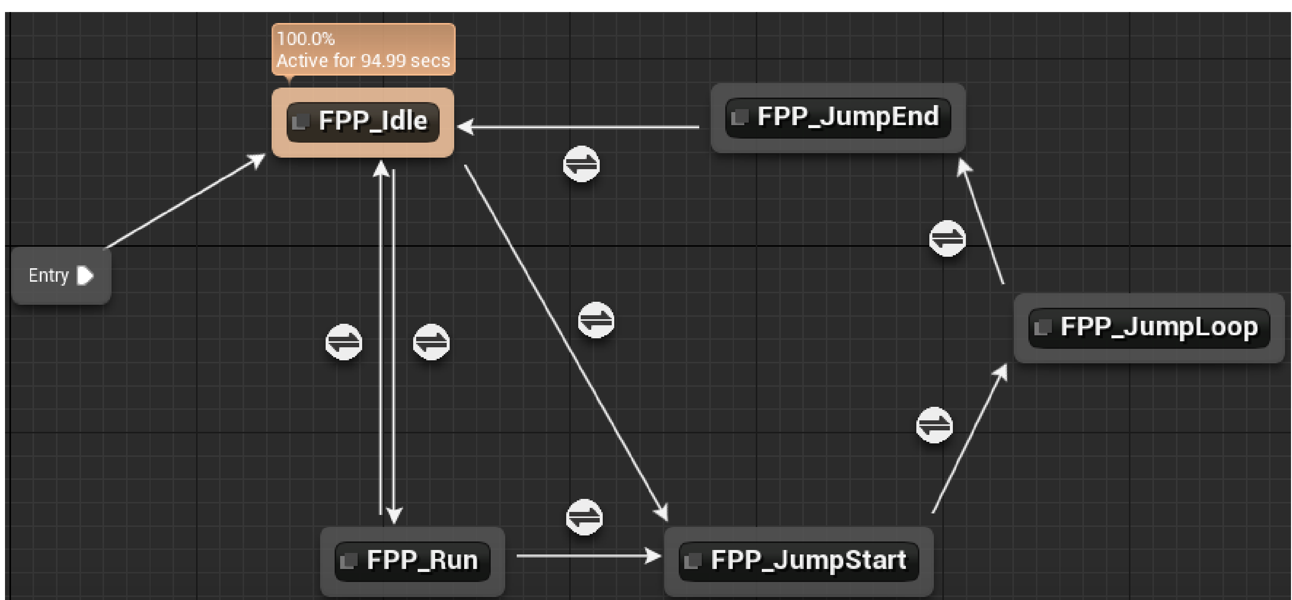


Ouvrez votre **new State Machine**.

Cliquez sur **l'Asset Browser**. Vous devriez **voir** vos **animations** : Idle, JumpEnd, JumpLoop, JumpStart et Run.

Faites un **drag&drop** de **chacune** de vos **animations** sur le graphique.

Nous allons à présent créer des **liens** entre chaque **animations** pour **créer** des **transitions**. Créez des liens pour obtenir le schéma suivant :

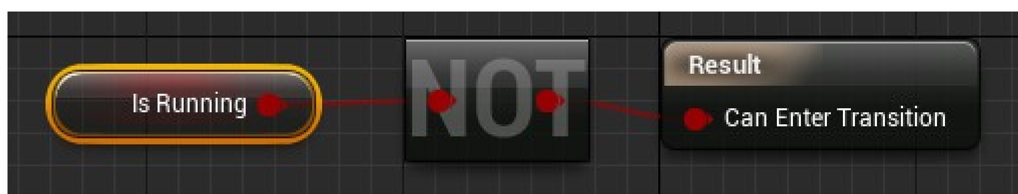


Pour les liens entre les animations, il vous suffit de double cliquer sur un symbole de transition.

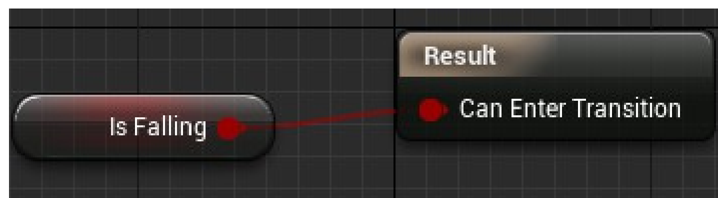
Mettez les informations suivantes sur vos transitions :

Idle → Run :

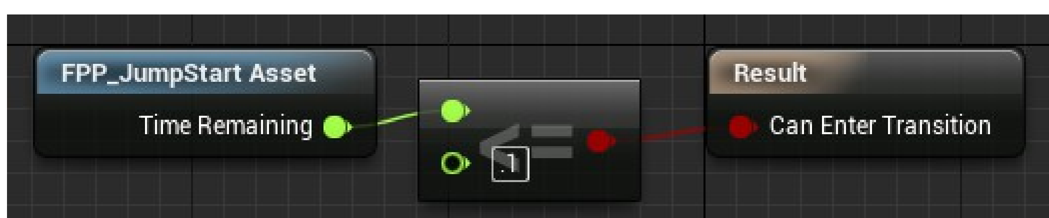
Run → Idle :



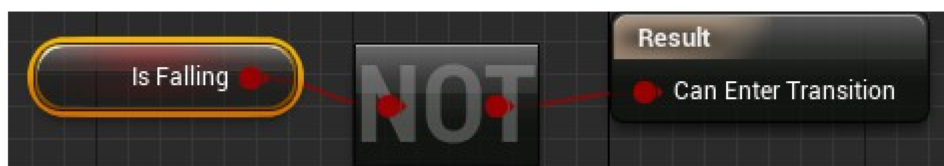
Run → JumpStart et Idle → JumpStart :



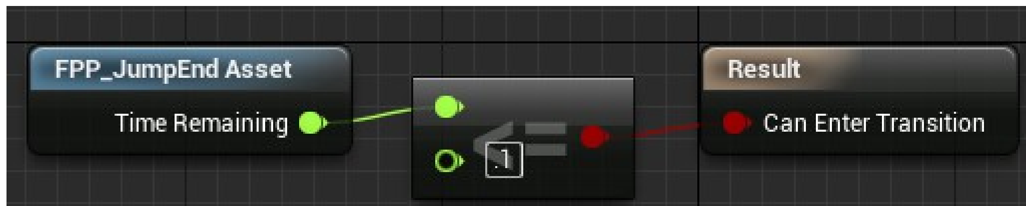
JumpStart → JumpLoop :



JumpLoop → JumpEnd :



JumpEnd → Idle :



Votre **Animblueprint** est **prêt** ! :)

Retournez dans votre **BP_TutoJoueur** et **sélectionnez** Votre **FirstPersonMesh**.
Dans **Anim blueprint generated class**, mettez votre **AnimBlueprint**.

Il nous reste à **gérer** le **HUD** !

Pour commencer, **Créez** un nouveau **Widget blueprint** (User Interface → Widget Blueprint). **Nommez** le **HUD** et **ouvrez** le.

Laissez le **Canvas Panel** et **ajoutez** une **image**. **Sélectionnez** là, mettez **l'anchors** au **milieu** de l'écran. En **position X** on aura **-5** et en **Y -5**. En **Size X** on aura **10** et en **Y 10** également.
En **Color and Opacity** mettez du **rouge**.

Créez deux autres **Widget Blueprint**, **PlayerInfos** et **ScoreHUD**.

Ouvrez **PlayerInfos** et **supprimez** le **Canvas Panel**. Mettez une **horizontal box** avec **4 text** à **l'intérieur**. Mettez leur **tailles** en **Fill** avec un **alignement horizontal** et **vertical** au **milieu**.

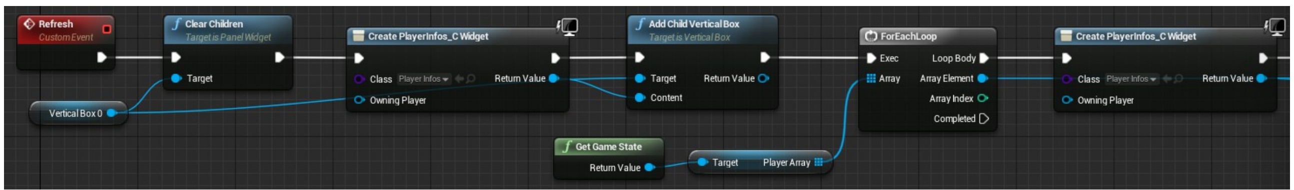
Changez les **textes** par **défaut** ainsi que leur nom en **Name**, **Kills**, **Death** et **Score**. Mettez **tous** ces **text** en **IsVariable**.

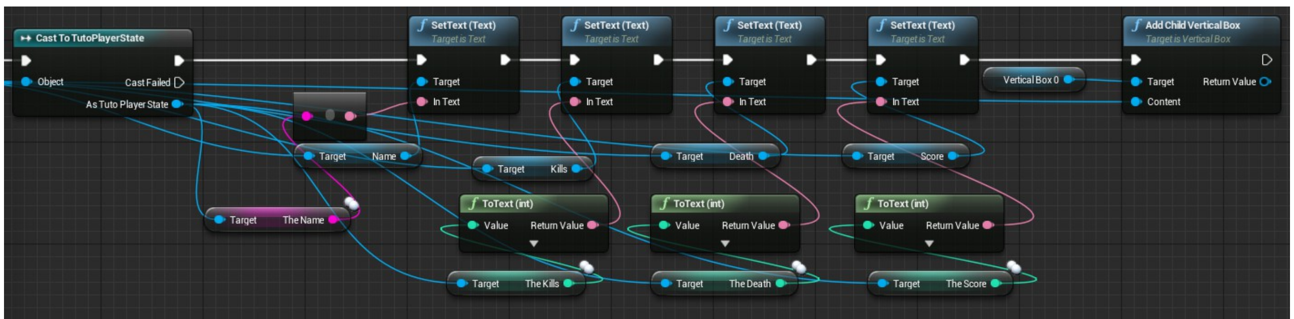
Ouvrez **ScoreHUD**. **Supprimez** le **CanvasPanel** et **ajoutez** une **border**. **Dedans**, **ajoutez** une **vertical box**. **Dans** cette **vertical box ajoutez** un **Playerinfos** que vous trouverez dans **User Created**. Mettez votre **Vertical Box** en **IsVariable**.

Passez dans le **Graph**. **Créez** un **custom Event** et **nommez** le **Refesh**. Commencez par **Récupérer** votre **VerticalBox** et Faites un **ClearChildren dessus**. **Ajoutez** un **CreateWidget** avec comme **classe PlayerInfos**, celui-là sera les libellés du score. Faites un **AddChild** sur la **VerticalBox** avec ce Widget. Ensuite On **récupère** le **Game State** et dessus on **récupère** la variable **PlayerArray**. Un **forEach** sur le **PlayerArray** nous permet de récupérer **tous** les **PlayerStates** de la partie. Dans le loop body on **créer** un **widget** de **classe PlayerInfos**. On **cast** ensuite **l'Array Element** de la loop en **TutoPlayerState**. Vous allez pouvoir récupérer les variables ainsi. Sur le **Widget**, **récupérez tous** les **Text** (Name, Kills, Death et Score) et faites un **SetText** sur **chacun**. Grâce au cast, vous allez pouvoir

mettre vos **variable** au bon **endroit**, **TheName** sur **Name**, **TheKills** sur **Kills**, etc.
On fini en **ajoutant** le **widget** à la **verticalBox**.

Ça donne ça :

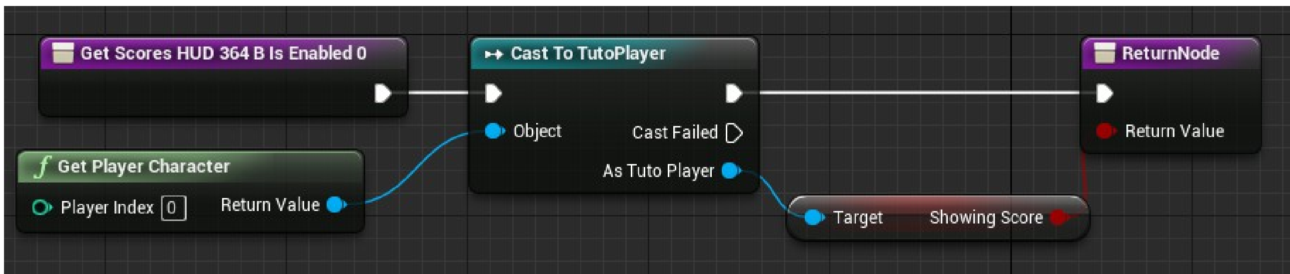




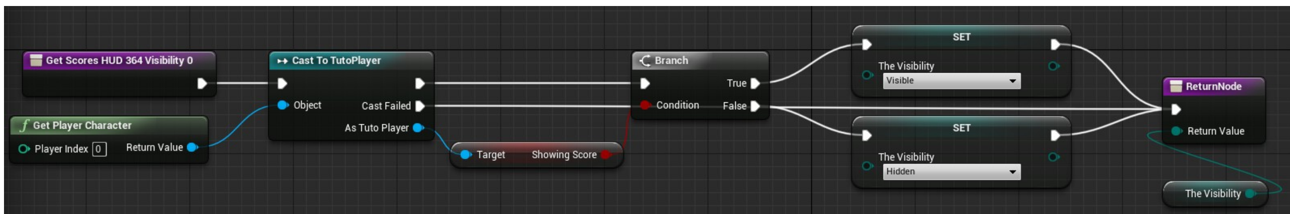
Il ne vous reste plus qu'à **ajouter** un **ScoreHUD** à vôtre **HUD**.

On va binder la visibilité et l'enabled du ScoreHUD sur le widget HUD.

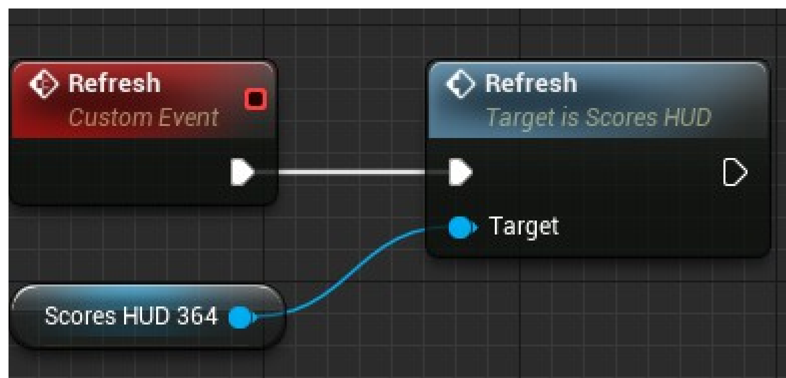
Pour le IsEnabled, c'est plutôt classique :



Pour la visibilité, **créez** une **variable locale** de **type ESlateVisibility** et mettez là par **défaut** à **Hidden**.
Ajoutez ceci :

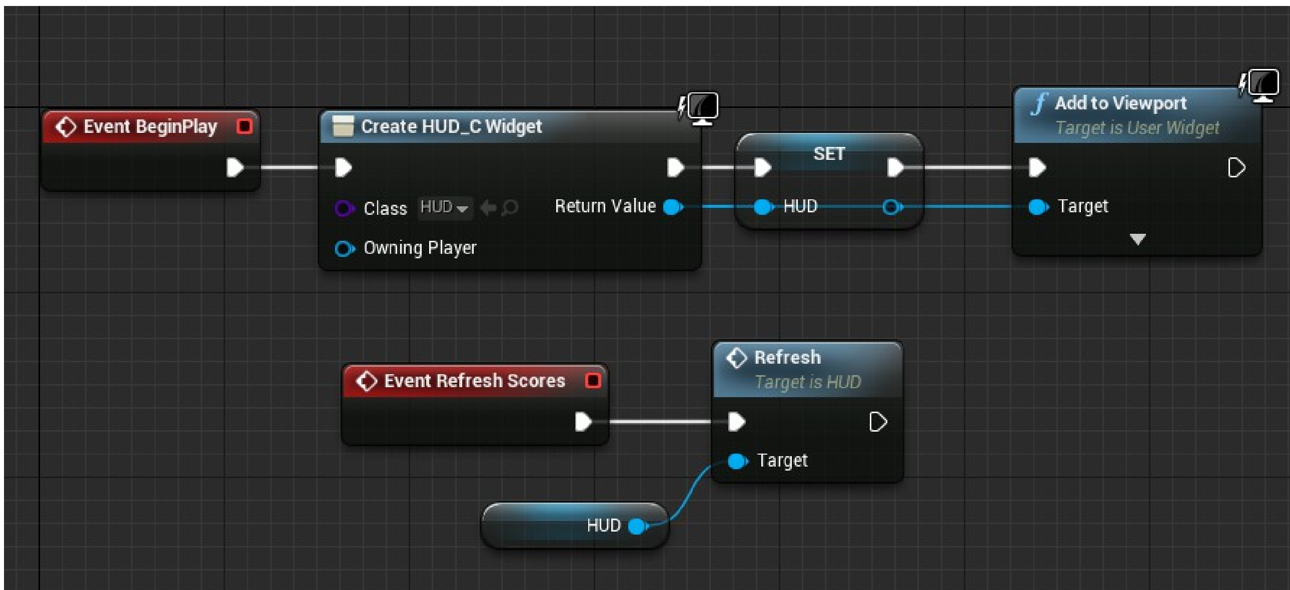


Dans **l'event Graph**, **Créez** un **custom event** et **nommez** le **Refresh**. Il va **appeler** le **custom event refresh** du **ScoreHUD**.



Pour finir, **retournez** dans le **BP_TutoPlayer**. **Ajoutez** une **variable HUD** de **type HUD**.

On va **créer** le **HUD**, **l'attribuer** et enfin on va **binder l'event RefreshScore** qu'on avait créé en C++ :



Sauvegardez tout, **compilez** les **blueprint** qui ne le sont pas. Si vous n'avez pas d'erreurs, vous pouvez **tester** votre projet à **2** joueurs ou **plus** en **Dedicated Server**.
En appuyant sur Tab vous devriez avoir les scores qui s'affichent et si vous tuez quelqu'un, vous devriez gagner un kill et du score et lui devrait gagner une death. Vous pouvez plus facilement viser grâce au curseur rouge au milieu de votre écran.

Vous pouvez à présent ajouter d'autres choses et système tel qu'un nom personnalisable, un niveau, un salon pour tester en local avec d'autres personnes, des animations, etc.

Pour toute éventuelles questions, remarques, détails ou demandes, n'hésitez pas à prendre contact avec moi ! (Cf : début du tuto)