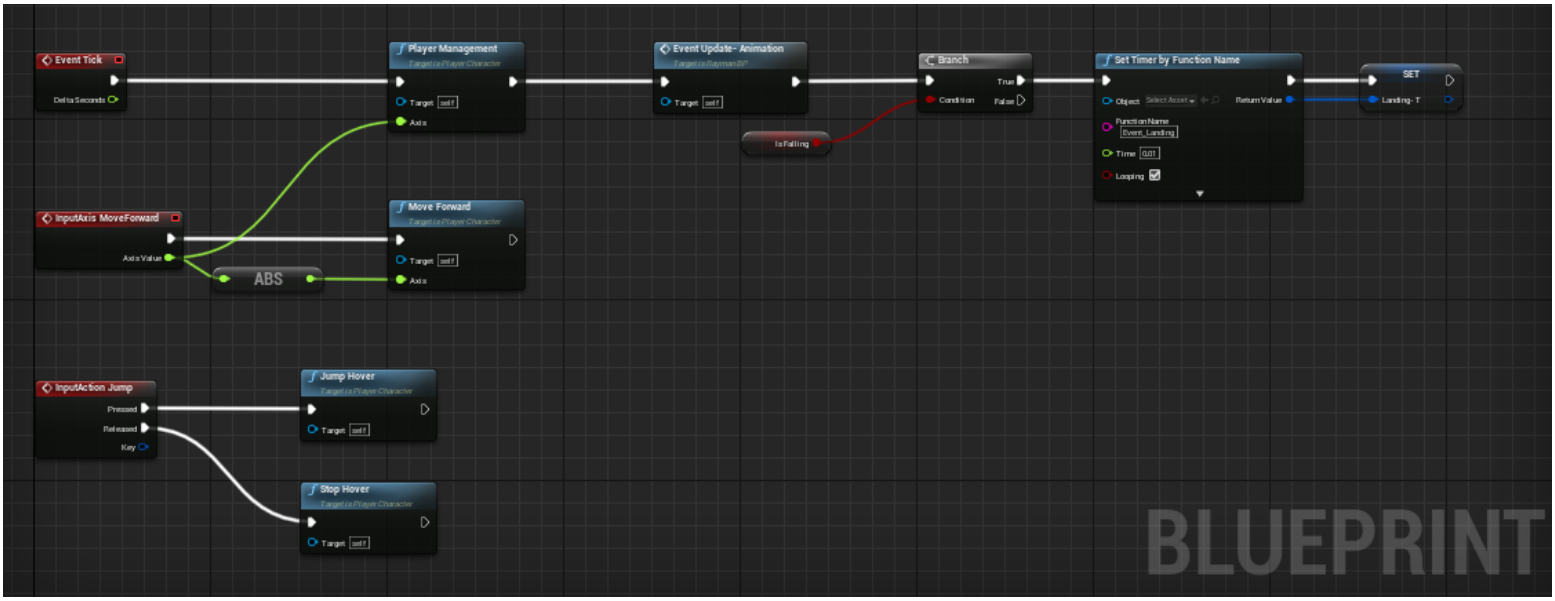
Projet Rayman 2021:

Notre projet consiste à programmer un jeu en 2D simple avec des niveaux, un scénario dans le même style de Rayman, a vrai dire ce sera un Rayman :)

- L'aspect graphique sera créé avec les sprites permettant de créer les animations du personnage et paysage du jeu, les collisions etc

Exemple d'un rendu du jeu !

- Le code pur sera en C++ et BluePrint. Le code C++ sera composé des principales fonctions (avancer, reculer, s'accroupir, léviter etc). Avec BluePrint on relie simplement des éléments et on les fait interagir entre eux grâce au Visual Scripting. Cela évité de surcoder



```cpp
void APlayer_Character::PlayerManagement(float Axis)
{
    // À chaque frame récupère la velocité du personnage
    Velocity_X = fabsf(FloatingPawn->Velocity.X);
    Velocity_Z = fabsf(FloatingPawn->Velocity.Z);

    // Création d'une 'LineTrace' pour savoir s'il on touche le sol
    FVector Start = GetActorLocation() + GetActorUpVector();
    FVector End = GetActorLocation() + (GetActorUpVector() * -75.f);

    GetWorld()->LineTraceSingleByChannel(OutHit, Start, End, ECC_Visibility);
    DrawDebugLine(GetWorld(), Start, End, FColor::Yellow, false, 5.f, 0.f, 0.5f);

    // Appelle de la fonction YawRotation
    YawRotation(Axis);

    // Si la  'LineTrace' touche ne touche rien alors on applique la gravité
    if (!OutHit.bBlockingHit && bEnableGravity)
    {
        FloatingPawn->Velocity.Z = GravityForce * GetWorld()->GetDeltaSeconds();
        bIsFalling = true;
        WalkingAngle = 0.f;

        // Reinitialise la Rotation Du Personnage
        SetActorRotation(UKismetMathLibrary::RInterpTo(GetActorRotation(), FRotator(0.f, GetActorRotation().Yaw, 0.f), GetWorld()->GetDeltaSeconds(), 5));

    }

    else
    {
        bIsFalling = false;

        // Si l'on touche le sol, on aligne le personnage au sol
        if (GetVelocity().X != 0.f && bCanRotate)
        {
            // On effectue un produit scalaire pour trouver l'angle du sol
            WalkingAngle = FVector::DotProduct(OutHit.Normal, FVector(0.f, 0.f, 1.f));
            // On convertit l'angle en radian, puis en degree
            WalkingAngle = acosf(WalkingAngle) * (180 / PI);
            // On effectue un produit vectoriel pour savoir si l'angle et positif où négatif
            WalkingAngle *= UKismetMathLibrary::SignOfFloat(FVector::CrossProduct((GetActorRightVector()), OutHit.Normal).GetSafeNormal(0.f).Z);
            // Interpolise cette valeur pour un rendu plus doux
            WalkingAngle = UKismetMathLibrary::FInterpTo(GetActorRotation().Pitch, WalkingAngle, GetWorld()->GetDeltaSeconds(), 15.f);
            // Donne cette nouvelle valueur comme rotation au personnage
            SetActorRotation(FRotator(WalkingAngle, GetActorRotation().Yaw, 0.f));
        }

    }
}
```

Exemple de Fonctions, codé en Blueprint en haut, et en C++ en bas

Les fonctions de notre projet seront toutes commentés, pour facilité leur lectures
Les entrées de notre programme seront les touches de la personne y jouant et elles vont retourner

Jusqu'à maintenant, nous avons les fonctions:

- PlayerManagement()→ Cette fonction et celle sur la capture, elle nous permet de gérer la base du personnage chaque frame, c'est-à-dire par exemple de savoir s'il l'on est dans l'air et non sur le sol, ou d'aligner le personnage avec la surface du sol
- d'aligner le personnage sur le sol, ou de
- MoveFoward(float Axis)→ Cette fonction va nous permettre d'avancer dans le monde
- Jump()→ Comme son nom l'indique, nous pourrons sauter grâce à elle
- Hover()→ Grâce à elle nous pourrons planer sur de grande distance tant que nous ne relâchons pas la toucher "Planer"
- Crouch()→ Cette fonction va nous permettre de nous accroupir et de ramper