ESAME C++

MARCO PUNGILLO

INDICAZIONI INIZIALI

Nel progetto sono presenti vari file completamente commentati e organizzati sotto delle region **“DEPRECATEDAndCOMMENTED”.** Questi sono file relativi a feature che ho tentato di implementare, ma che non essendo riuscito a portarle a compimento perfettamente ho preferito rimuovere dal progetto.

Inoltre non sono riuscito ad implementare correttamente il **MovementComponent** comprensivo di tutti quei controlli che lo avrebbero reso efficace, perciò ho preferito partire da un character rispetto ad una Pawn, così da poter sfruttare il CharacterMovementComponent. Ho comunque lasciato non commentato il Componente, perché tanto non viene sfruttato da nessuno.

La restante parte delle funzionalità richieste è stato implementato secondo il testo dell’esame. Ho affrontato i vari argomenti ritenuti importanti da sviluppare in C++, tra i quali:

- **Raycast;**
- **Fisica;**
- **Piattaforme Mobili;**
- **Triggers** (e con sé il richiamo a Delegates);
- **Binding degli Inputs;**
- **Salvataggio** (Attraverso Checkpoints) e **Caricamento** (Attraverso FallingVolume);
- **Interfacce;**

Nelle prossime pagine segue una spiegazione del gioco e delle varie meccaniche implementate (Che nel gioco sono spiegate con dei TextRenderer).

TELECINESI: GAMEPLAY

Il gioco si basa sull'utilizzo della telecinesi per superare delle zone di platform. Nel livello sono presenti diversi elementi con una texture caratteristica, che rispecchia il fatto di poterli controllare con la telecinesi.
Il Player, oltre al movimento e al salto, ha a disposizione diversi usi della telecinesi:

- Premendo il **Tasto Sinistro del Mouse** può effettuare **l'Holding** dell'oggetto. In questo modo l'oggetto continuerà a muoverersi lentamente (lerpando) verso un punto in linea con la telecamera, tale punto è situato alla stessa distanza oggetto-player del momento in cui l'Holding è stato effettuato;

  Quando l'oggetto entra in stato di Holding assume una colorazione relativa ad esso (selezionabile dai details delle blueprint, per ognuna ho scelto la stessa texture con il colore azzurro). Allo stesso modo durante l'holding il cubo che fa da arma al player assumerà la sua colorazione azzurra (anche qui ho dato personalizzabilità attraverso dei materials selezionabili dai details dell'actor).

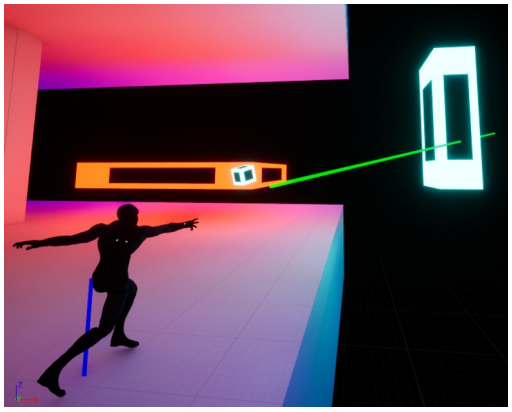


Figura 1: Visuale Out-Game del momento in cui il Player effettua l'Holding

- Premendo il **Tasto Destro del Mouse** il può effettuare il **Release** dell'oggetto. In questo modo l'oggetto rimarrà fermo nell'ultima posizione raggiunta durante l'Holding. Ovviamente è possibile eseguirlo solo durante l'Holding.

  Al rilascio dell'oggetto, il cubo-arma del player riacquisisce la sua normale colorazione verde.

Figura 2: Visuale Out-Game dopo il Release dell'oggetto

- Premendo il **Tasto Q** durante l’Holding, l’oggetto attualmente tenuto con la telecinesi viene avvicinato fino ad una distanza minima.

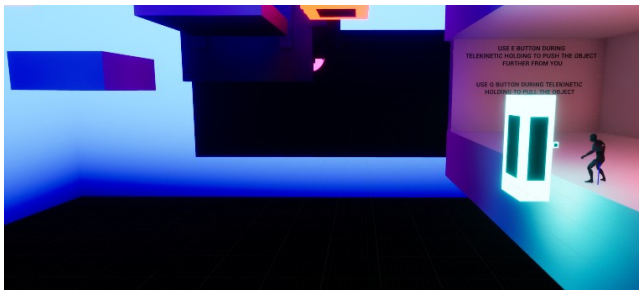


Figura 3: Distanza Minima

- Premendo il **Tasto E** durante l’Holding, invece l’oggetto attualmente tenuto con la telecinesi viene allontanaato fino ad una distanza massima.

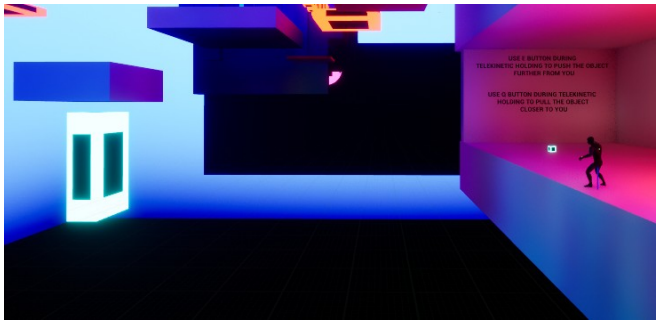


Figura 4: Distanza Massima

- Premendo il **Tasto R** sui coni rosa, particolare tipo di oggetti telecinetici **(riconosciuti attraverso una interfaccia aggiuntiva, implementata in C++)**, è possibile imprimere un impulso su di essi.

  È anche possibile fare questo mentre li si mantiene nello stato di Holding e quindi fermi davanti a sé. In questo caso appena l’Holding verrà rilasciato tutto l’impulso verrà scatenato in una sola volta, lanciando l’oggetto a gran velocità.

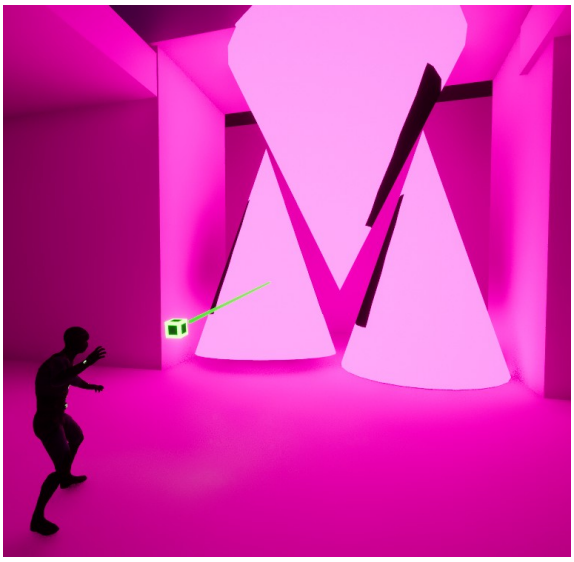


Figura 5: Visuale durante l'esecuzione dell'impulso

  Questa capacità serve poi nel finale per lanciare uno degli oggetti telecinetici contro il pannello finale, così da finire il gioco e ottenere la scritta “VICTORY!”. In questo caso ho eseguito tramite C++ anche il binding al delegate dell’OnComponentBeginOverlap del Trigger.


```
// Bind the overlap events
TriggerVolume->OnComponentBeginOverlap.AddDynamic(this, &AVictoryTrigger::OnOverlapBegin);
```

INPUTS: IMPLEMENTAZIONE

Per implementare gli inputs ho utilizzato il nuovo sistema, dando i riferimenti al mapping context e alle input action al MyExamCharacter. Una volta specificate le Input Action dai details dell'Actor allora le ho collegate alle funzioni nel SetupPlayerInputComponent, assicurandomi che venisse considerato il nuovo sistema di inputs.

```cpp
// Called to bind functionality to input
void AMyExamCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent)
{
    Super::SetupPlayerInputComponent(PlayerInputComponent);

    // Set Up Binding using a safe Cast to Obtain the enhancedInputSystem
    if (UEnhancedInputComponent* EnhancedInputComponent = CastChecked<UEnhancedInputComponent>(PlayerInputComponent))
    {
        EnhancedInputComponent->BindAction(MoveAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::Move);
        EnhancedInputComponent->BindAction(LookAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::Look);
        EnhancedInputComponent->BindAction(JumpAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::StartJump);

        EnhancedInputComponent->BindAction(TelekinesisHoldAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::ApplyTelekineticHold);
        EnhancedInputComponent->BindAction(TelekinesisStopAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::StopTelekineticHold);
        EnhancedInputComponent->BindAction(PushForceAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::ApplyPushForce);
        EnhancedInputComponent->BindAction(PullForceAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::ApplyPullForce);

        EnhancedInputComponent->BindAction(ImpulseAction, ETriggerEvent::Triggered, this, &AMyExamCharacter::ApplyTelekineticImpulse);

    }
}
```

TELECINESI: IMPLEMENTAZIONE

I poteri di Hold e di Impulse si basano sullo sfruttare un Raycast per identificare la presenza di un oggetto telecinetico. Questo Raycast avviene su di un Trace Channel Custom chiamato Telekinesis e sfruttato da C++ come **ECollisionChannel::ECC_GameTraceChannel1**. Una volta identificato l'oggetto su cui interagire:

- L'Hold prende il riferimento all'oggetto e gli imposta una posizione, determinata dal forward della camera del player e dalla distanza che intercorre tra il player e l'oggetto al momento dell'Hold. Cerca poi di mantenere sempre lo stesso offset per la durata dell'Hold.
    - L'avvicinamento e l'allontanamento eseguiti con Q ed E non fanno altro che diminuire o aumentare questo offset fino ad un minimo e un massimo.
- L'Impulse trovato l'oggetto controlla che a questo sia possibile effettuare l'impulso, eseguendo un controllo sull'implementazione dell'Interfaccia **IImpulsable**. Nel caso in cui l'oggetto implementa l'interfaccia, allora abilita la fisica sul suo RootComponent e gli imprime l'impulso.
    - Per estendibilità di comportamento ho aggiunto un metodo di callback OnReceiveImpulse, così da poter prevedere comportamenti diversificati da parte degli oggetti alla ricezione dell'impulso.

SAVE/LOAD: GAMEPLAY E IMPLEMENTAZIONE

Il sistema di salvataggio è, come detto, strutturato attraverso dei checkpoint. Questi salvano la posizione del player all'arrivo, così da poterla ripristinare quando il player fallisce una zona di platform, cadendo così di sotto. Il riconoscimento del fallimento della zona di platform è eseguito attraverso un FallingVolume che fa avvenire il caricamento della posizione dal salvataggio.

Le funzioni di salvataggio e caricamento sono state implementate in C++ attraverso una **BlueprintFunctionLibrary** e poi utilizzate nelle Blueprints.

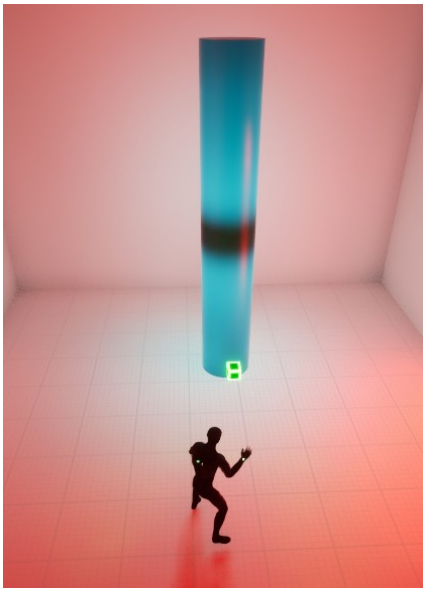
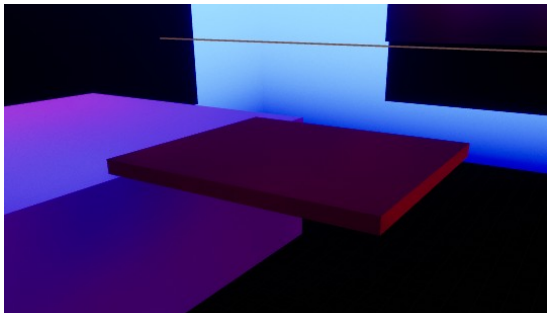


Figura 6: Il Player davanti alla colonna che identifica il Checkpoint

ENVIRONMENT



Le piattaforme mobili sono state implementate dando delle posizioni "di Patrol", un tempo di attesa e un tempo di spostamento.

```cpp
void AExamBaseMovingPlatfrom::MovementBehavior(float DeltaTime)
{
    // Wait Status
    if (bIsWaiting)
    {
        WaitInPositionCounter -= DeltaTime;
        if (WaitInPositionCounter <= 0)
        {
            bIsWaiting = false;
        }
    }
    // Movement Status
    else
    {
        if (((this->GetActorLocation() - Positions[PositionIndex]).SizeSquared()) <= AcceptableRadius * AcceptableRadius)
        {
            // Set waiting
            bIsWaiting = true;
            WaitInPositionCounter = WaitInPositionTime;

            // Resetting Movement
            MovementTimeAccumulator = 0;
            UE_LOG(LogTemp, Warning, TEXT("Destination Reached"));

            if (PositionIndex >= Positions.Num() - 1 || ((PositionIndex <= 0)))
            {
                IndexChangerValue *= -1;
            }
            PositionIndex += IndexChangerValue;
        }
        else
        {
            MovementTimeAccumulator += DeltaTime;
            this->SetActorLocation(FMath::Lerp(Positions[PositionIndex - IndexChangerValue], Positions[PositionIndex], MovementTimeAccumulator / MovementDuration));
        }
    }
}
```

Figura 7: Logica del Movimento delle Piattaforme, eseguita nel Tick