

Diploma Accademico in Teoria e tecniche della comunicazione visiva multimediale

Anno Accademico 2024/2025

Flusso dei resti

Guida all’ecosistema audiovisivo interattivo

ACCADEMIA DI BELLE ARTI MACERATA

Giacomo Cappella

Flusso dei resti

Software utilizzati:
Unreal Engine, Ableton Live, oscHook v2,
Blender, Reality Scan

Progetto grafico: Giacomo Cappella

Font utilizzato: Montserrat, Anek Bangla

Stampato a Dicembre 2025

© 2025

ACCADEMIA
DI BELLE ARTI
MACERATA

Flusso dei resti

Guida all'ecosistema audiovisivo interattivo

Diplomando: Giacomo Cappella

Diploma Accademico in
Teoria e tecniche della comunicazione
visiva multimediale

Anno Accademico 2024/2025

Relatore: Federico Cingolani
Correlatore: Francesco Mariano

Abstract e Concept

"**Flusso dei resti**" non è solo un esercizio tecnico, ma un'installazione audiovisiva interattiva che indaga la relazione ibrida tra ambiente naturale, percezione umana e mediazione tecnologica. Il progetto nasce dalla digitalizzazione di un paesaggio reale — le formazioni geologiche delle Lame Rosse — catturato tramite fotogrammetria mobile. Nel passaggio dal fisico al digitale, la materia perde la sua solidità per tradursi in un ambiente virtuale instabile: una "**rovina digitale**" composta da nuvole di punti e frammenti sonori.
L'opera mira a una riflessione critica su come la tecnologia filtri, deformi e riconfiguri costantemente la nostra realtà, traducendo il mondo in un flusso perpetuo di dati.

Riferimenti Teorici

La struttura concettuale dell'opera poggia su due pilastri fondamentali:

- **L'Opera Aperta** (Umberto Eco): L'installazione rifiuta una narrazione lineare e chiusa. È il fruitore, attraverso la sua interazione, a completare l'opera, rendendo ogni esperienza unica e irrepetibile.
- **Rete di Attanti** (Bruno Latour): L'opera mette in scena una rete paritaria dove attori umani (il fruitore) e non-umani (il software, i sensori, l'algoritmo) collaborano alla generazione dell'evento.

L'Esperienza e l'Interazione

Allestita in una sala buia per favorire l'immersione, l'installazione utilizza lo smartphone come protesi percettiva. Attraverso l'inclinazione e la rotazione del dispositivo, l'utente naviga l'ambiente virtuale, manipolando in tempo reale sia la materia visiva che il paesaggio sonoro. Questo meccanismo genera uno "**straniamento percettivo**": il gesto fisico sullo schermo si traduce in una reazione complessa e granulare nell'ambiente audiovisivo.

Questo manuale documenta l'intero processo produttivo necessario alla realizzazione di un sistema interattivo complesso come "**Flusso dei resti**".

Questo documento esplora l'interconnessione tra diverse tecnologie per creare un flusso di lavoro (workflow) coeso.

Nelle pagine seguenti verranno analizzati:

- **Acquisizione e preparazione**:
Dalla fotogrammetria in Blender alla creazione di nuvole di punti che ricalcano i modelli.

- **Visualizzazione in Real-Time**:
L'utilizzo di Unreal Engine 5 e del sistema particellare Niagara per gestire l'estetica della "rovina digitale".

- **Protocollo di comunicazione**:
L'implementazione del protocollo OSC (Open Sound Control) per trasformare lo smartphone in un controller gestuale a bassa latenza.

- **Interattività sonora**:
Il ponte dati verso Ableton Live per la sintesi granulare e la spazializzazione audio reattiva.

Questa guida è concepita per fornire una visione d'insieme, illustrando come singole tecnologie possano dialogare per creare un'esperienza artistica organica ed interconnessa.

Flusso di lavoro

1. Preparazione degli asset e creazione del sistema Niagara di base
2. Creazione del materiale dinamico e interattività di base
3. Creazione dell'attore giocabile e della modalità di gioco
4. Configurazione e test del ricevitore OSC
5. Implementazione del controllo "Modalità Ruota"
6. Implementazione della "Modalità Osservazione"
7. Generazione di un effetto di prossimità tra il giocatore e la nuvola di punti
8. Itegrazione della Dissolvenza per Prossimità sulla Static Mesh
9. Implementazione della caduta e respawn con Landscape Visibility Mask
10. Aggiunta del Client OSC per il Forwarding dei Dati ad Ableton Live
11. Interazione audio attraverso la prossimità
12. Implementazione dell'Avanzamento di Livello Condizionale tramite caduta
13. EXTRA - Implementazione di un Ricevitore OSC Universale con Inoltro Dati Unificato

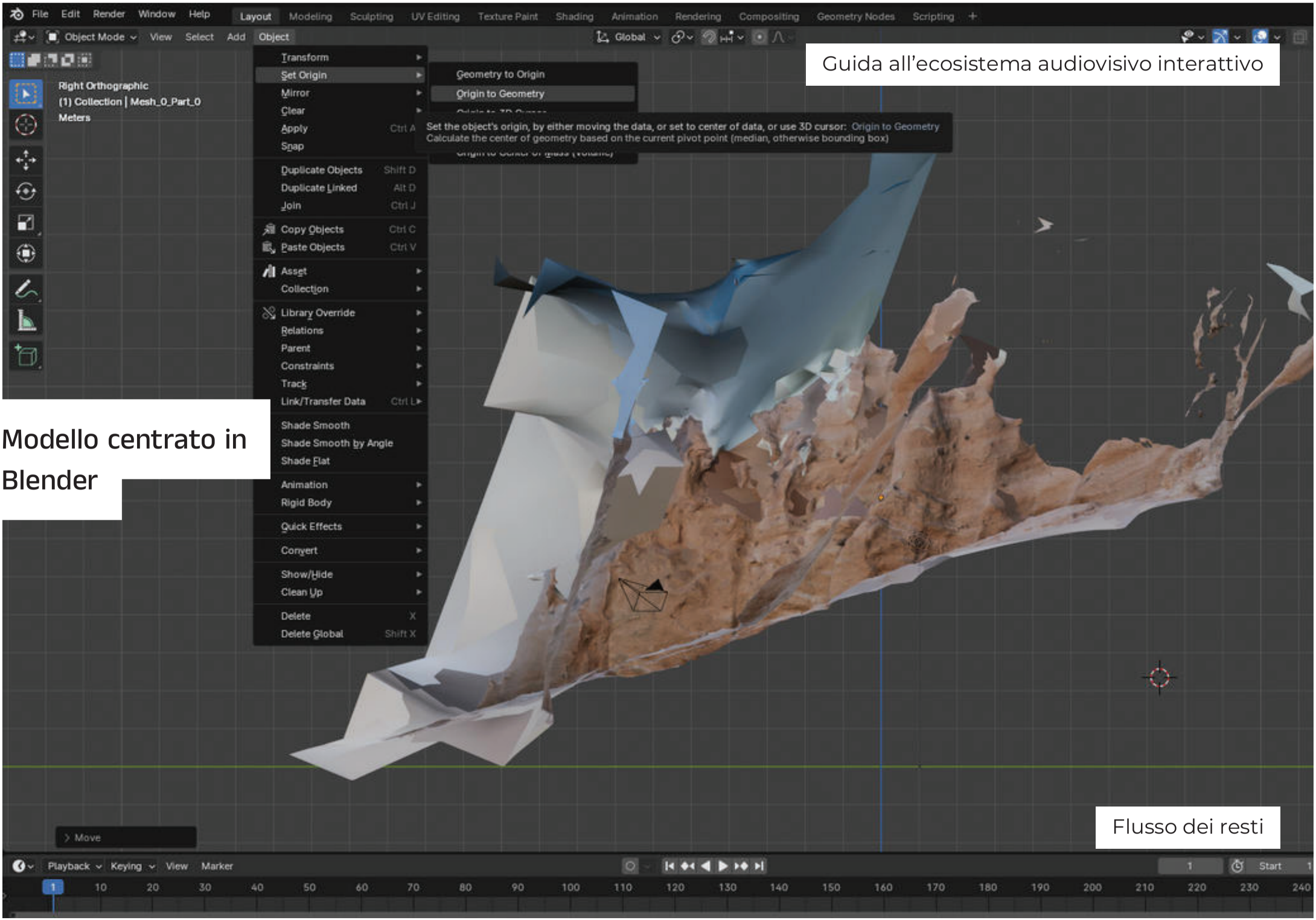


Vista del Livello 4
Lame Rosse

Preparazione degli asset e creazione del sistema Niagara di base

Questo capitolo guida attraverso i passaggi iniziali ed essenziali per creare un effetto visivo in Niagara. L'obiettivo di questa prima fase è preparare correttamente i nostri modelli 3D e materiali, poi usarli per generare un semplice sistema di particelle che nascono sulla superficie del nostro modello.

1. Preparazione della Mesh in Blender
2. Importazione degli asset in Unreal Engine
3. Creazione del sistema Niagara di base


Guida all’ecosistema audiovisivo interattivo
Modello centrato in Blender
Flusso dei resti
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
Object Mode View Select Add Object
Global
Right Orthographic
(1) Collection | Mesh_0_Part_0
Meters
Transform
Set Origin
Mirror
Clear
Apply Ctrl A
Snap
Duplicate Objects Shift D
Duplicate Linked Alt D
Join Ctrl J
Copy Objects Ctrl C
Paste Objects Ctrl V
Asset
Collection
Library Override
Relations
Parent
Constraints
Track
Link/Transfer Data Ctrl L
Shade Smooth
Shade Smooth by Angle
Shade Flat
Animation
Rigid Body
Quick Effects
Convert
Show/Hide
Clean Up
Delete X
Delete Global Shift X
Geometry to Origin
Origin to Geometry
Set the object's origin, by either moving the data, or set to center of data, or use 3D cursor: Origin to Geometry
Calculate the center of geometry based on the current pivot point (median, otherwise bounding box)
Move
Playback Keying View Marker
Start
1 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240

Preparazione della Mesh in Blender

Prima ancora di aprire Unreal Engine, è fondamentale assicurarsi che il nostro asset 3D sia preparato correttamente. Un modello con un punto di origine (o "pivot") errato causerà problemi di posizionamento, rotazione e scala una volta importato.

1. Aprire il File in Blender: Avviare Blender e importare o aprire il file della mesh (es. GLB, FBX, OBJ).

2. Selezionare l'Oggetto: Nella viewport, selezionare la mesh con cui si vuole lavorare.

3. Centrare l'Origine sulla Geometria: Il "punto di origine" è il punto nello spazio attorno al quale l'oggetto ruota e scala, rappresentato da un piccolo punto arancione. Per un corretto funzionamento in Unreal, questo punto dovrebbe trovarsi al centro logico dell'oggetto o alla sua base.

    - Con l'oggetto selezionato, navigare nel menu in alto: Object -> Set Origin -> Origin to Geometry. Questa operazione calcola il centro volumetrico della mesh e vi sposta l'origine.

4. Centrare l'Oggetto nel Mondo: Per una facile gestione in Unreal, è buona norma che l'oggetto stesso sia posizionato all'origine del mondo di Blender (coordinate 0, 0, 0).

    - Con l'oggetto selezionato, premere Alt + G per resettare la sua posizione (Location) a zero.

    - Correggere la rotazione degli assi X, Y, Z, se necessario.

5. Esportare la Mesh: Una volta preparata, esportare la mesh in un formato compatibile con Unreal Engine, come .glb o .fbx.

Importazione degli asset in Unreal Engine

Ora che la mesh è pronta, possiamo importarla nel nostro progetto Unreal.

1. Trascinare il File nel Content Browser: Aprire il Content Browser nella cartella desiderata e trascinare il file della mesh (es.modello.glb) direttamente al suo interno.

2. Confermare le Opzioni di Importazione: Apparirà una finestra di dialogo "GLB Import Options". Per questo scopo, le impostazioni di default sono generalmente sufficienti. Assicurarsi che le seguenti opzioni siano attive:

    - Import Mesh: Per importare la geometria del modello.

    - Import Materials and Textures: Per creare automaticamente i materiali e le texture associati al modello.

3. Verificare gli Asset Creati: Dopo l'importazione, Unreal Engine avrà creato diversi nuovi asset nella cartella:
    - Una Static Mesh: L'asset della geometria 3D.
    - Uno o più Material: Gli asset che definiscono l'aspetto della superficie della mesh.
    - Una o più Texture: I file immagine utilizzati dai materiali (es. colore, rugosità, etc.).

Creazione del sistema Niagara di base

Con i nostri asset pronti nel progetto, possiamo iniziare a costruire l'effetto di particelle.

1. Creare un Nuovo Sistema Niagara:
    - Nel Content Browser, cliccare con il tasto destro del mouse.
    - Dal menu contestuale, navigare su FX -> Niagara System.
    - Scegliere l'opzione "New system from a selected emitter" o "New system from a template". Selezionare un preset semplice come "Simple Sprite Burst" o "Fountain". Questo ci fornirà un punto di partenza funzionante.
    - Nominare il nuovo sistema in modo descrittivo, ad esempio NS_PointCloud.
2. Configurare la Posizione di Nascita delle Particelle (Spawn Location):
    - Aprire il sistema Niagara appena creato (NS_PointCloud) con un doppio clic.
    - All'interno dell'emettitore, trovare il gruppo di moduli chiamato Particle Spawn.
    - Cliccare sul pulsante + all'interno di questo gruppo e aggiungere il modulo Static Mesh Location. Se era già presente un modulo di posizione (come Sphere Location), è possibile sostituirlo o disabilitarlo.
    - Selezionare il nuovo modulo Static Mesh Location.
    - Nel pannello "Selection" (o "Details") a destra, individuare la proprietà Default Mesh.

- Cliccare sul menu a tendina e assegnare la Static Mesh che abbiamo importato in precedenza. La stessa mesh apparirà anche nel campo Preview Mesh.

- Risultato: Nel viewport di anteprima di Niagara, si vedrà che le particelle non nascono più da un punto o da una sfera, ma vengono generate direttamente sulla superficie del nostro modello 3D.

3. Assegnare un Materiale di Base alle Particelle:

    - Sempre nell'emettitore, scorrere fino al gruppo di moduli chiamato Render.

    - Selezionare il modulo Renderer presente (es. Sprite Renderer).

    - Nel pannello "Selection", individuare la proprietà Material.

    - Cliccare sul menu a tendina e assegnare il Material che è stato creato automaticamente durante l'importazione della nostra mesh.

    - Risultato: Le particelle ora non sono più semplici quadrati bianchi, ma utilizzeranno il materiale del nostro modello. Tuttavia, non avranno ancora il colore corretto in base alla loro posizione.

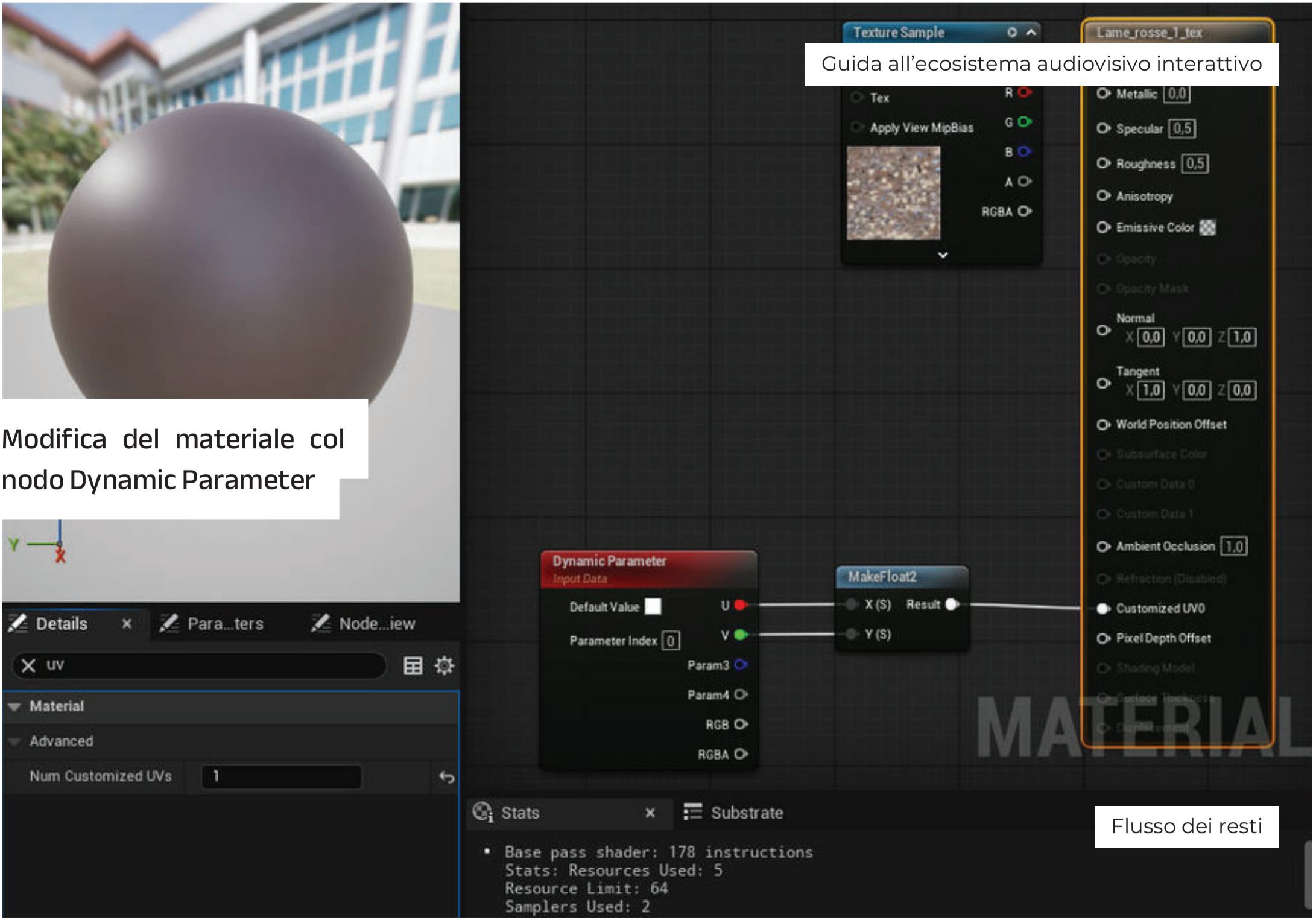
Procediamo con la fase successiva, concentrandoci sulla creazione del materiale dinamico e sul collegamento con Niagara per campionare il colore, esattamente come abbiamo specificato.

Questa facciata è stata lasciata intenzionalmente vuota.

Creazione del materiale dinamico e interattività di base

Con il nostro sistema Niagara che emette particelle sulla mesh, il passo successivo è renderlo visivamente più integrato e prepararlo per l'interazione. Faremo in modo che ogni particella "legga" il colore della texture direttamente sotto di essa e aggiungeremo una forza di base che potremo controllare in seguito.

1. Creazione e modifica del materiale dinamico
2. Collegamento e configurazione in Niagara
3. Aggiungere una forza di base controllabile


Modifica del materiale col nodo Dynamic Parameter
Texture Sample
Tex
Apply View MipBias
R
G
B
A
RGBA
Lame_rosse_1_tex
Metallic 0,0
Specular 0,5
Roughness 0,5
Anisotropy
Emissive Color
Normal
Tangent
World Position Offset
Ambient Occlusion 1,0
Customized UV0
Pixel Depth Offset
Dynamic Parameter
Input Data
Default Value
Parameter Index 0
U
V
Param3
Param4
RGB
RGBA
MakeFloat2
X (S)
Y (S)
Result
Details
Para...ters
Node...iew
UV
Material
Advanced
Num Customized UVs 1
Stats
Substrate
Base pass shader: 178 instructions
Stats: Resources Used: 5
Resource Limit: 64
Samplers Used: 2
MATERIAL
Flusso dei resti

Creazione e modifica del materiale dinamico

L'obiettivo è modificare il materiale delle particelle affinché le sue coordinate UV (che determinano quale parte della texture visualizzare) possano essere controllate direttamente da Niagara.

1. Aprire il Materiale:
    - Nel Content Browser, individuare il Material che è stato creato durante l'importazione della mesh. Se per qualche motivo non fosse stato creato, è possibile crearlo manualmente cliccando con il tasto destro sulla Texture principale e selezionando "Create Material".
    - Aprire il materiale con un doppio clic per accedere al Material Editor.
2. Aggiungere il Nodo Particle Dynamic Material Parameter:
    - All'interno del grafico del materiale, cliccare con il tasto destro in uno spazio vuoto e cercare il nodo Particle Dynamic Material Parameter. Aggiungerlo al grafico. Questo nodo speciale funge da ponte, permettendo a Niagara di inviare dati numerici direttamente al materiale ad ogni frame.
3. Configurare il Parametro Dinamico:
    - Il nodo Particle Dynamic Material Parameter ha quattro uscite Float. Per controllare le coordinate UV, che sono un valore a due dimensioni (U e V), quindi si deve rinominare i primi due valori con U e V.
    - Selezionare il nodo Particle Dynamic Material Parameter appena creato.
    - Nel pannello "Details" a sinistra, sotto la sezione "Material Expression Dynamic Parameter", individuare l'array Parameter Names.
4. Collegare il parametro alle coordinate UV:
    - Trascina un filo dalla prima uscita (indice 0) del Particle Dynamic Material Parameter e cerca il nodo MakeFloat2. Collega la prima uscita all'input A del MakeFloat2.
    - Trascina un altro filo dalla seconda uscita (indice 1) e collegalo all'input B del MakeFloat2.

- Ora, bisogna creare un input personale nel nodo principale con tasto destro e cercando “”Custom Parameter”. L'output del nodo MakeFloat2 deve essere collegato al suo input UVs.

- Risultato: Abbiamo appena detto al materiale: "Invece di usare le coordinate UV di default, prendi due valori numerici che ti arriveranno da Niagara attraverso il canale CustomizedUV e usali come coordinate U e V per campionare questa texture".

- Salvare e chiudere il Material Editor.

Collegamento e configurazione in Niagara

Ora torniamo al nostro sistema Niagara per inviare i dati necessari al materiale che abbiamo appena modificato.

1. Assegnare il Materiale al Renderer:

    - Aprire il sistema Niagara (NS_PointCloud).

    - Nel gruppo Render, selezionare lo Sprite Renderer.

    - Nella proprietà Material, assicurarsi che sia assegnato il materiale che abbiamo appena modificato. Potrebbe essere necessario utilizzare la sezione Override Materials se si sta usando un Mesh Renderer.

2. Aggiungere il Modulo Dynamic Material Parameters:

    - Nel gruppo Particle Spawn, cliccare sul + per aggiungere un nuovo modulo.

    - Cercare e aggiungere Dynamic Material Parameters.

    - Se il materiale è stato configurato correttamente, in questo nuovo modulo dovrebbero apparire automaticamente i nomi dei due valori, U e V.

    - Da queste voci aprire la tendina tramite la freccia a destra e selezionare “Make Float from Vector 2D”.

3. Passare i dati UV campionati:

    - Il valore di Vector2D deve provenire dal modulo che sa dove si trova ogni particella sulla mesh.

    - Cliccare sulla freccia a discesa accanto a Vector2D e navigare nel menu: Link Inputs -> Particles -> SampledUV.

- Questo crea un collegamento diretto: il modulo Static Mesh Location calcola la coordinata UV sulla superficie della mesh per ogni particella che genera, e questo valore SampledUV viene ora inviato al materiale attraverso il canale che abbiamo creato. Le particelle dovrebbero ora colorarsi correttamente, rispecchiando la texture della mesh sottostante.

Aggiungere una forza di base controllabile

Per preparare l'interazione, aggiungiamo una forza che potremo modificare in seguito.

1. Aggiungere il Modulo Wind Force:
    - Nel gruppo Particle Update (poiché la forza deve agire costantemente sulle particelle, non solo alla loro nascita), aggiungere il modulo Wind Force.
    - Impostare la Wind Speed a un valore basso (es. 5.0) per un leggero movimento di base.
2. Creare un parametro utente per il controllo:
    - Vogliamo poter controllare l'intensità del vento dall'esterno del sistema Niagara (ad esempio, tramite un Blueprint).
    - Individuare la proprietà Wind Speed Scale. Cliccare sulla freccia a discesa accanto ad essa e selezionare Create New User Parameter.
    - Apparirà una finestra per creare una nuova variabile. Nominarla in modo chiaro, ad esempio wind force (usare lettere minuscole e spazi è una convenzione comune per i parametri utente). Lasciare il tipo come Float.
    - Risultato: Ora abbiamo una variabile globale "wind force" nel nostro sistema Niagara che agisce come un moltiplicatore per la forza del vento. Modificando questa singola variabile, possiamo aumentare o diminuire l'effetto del vento su tutte le particelle.

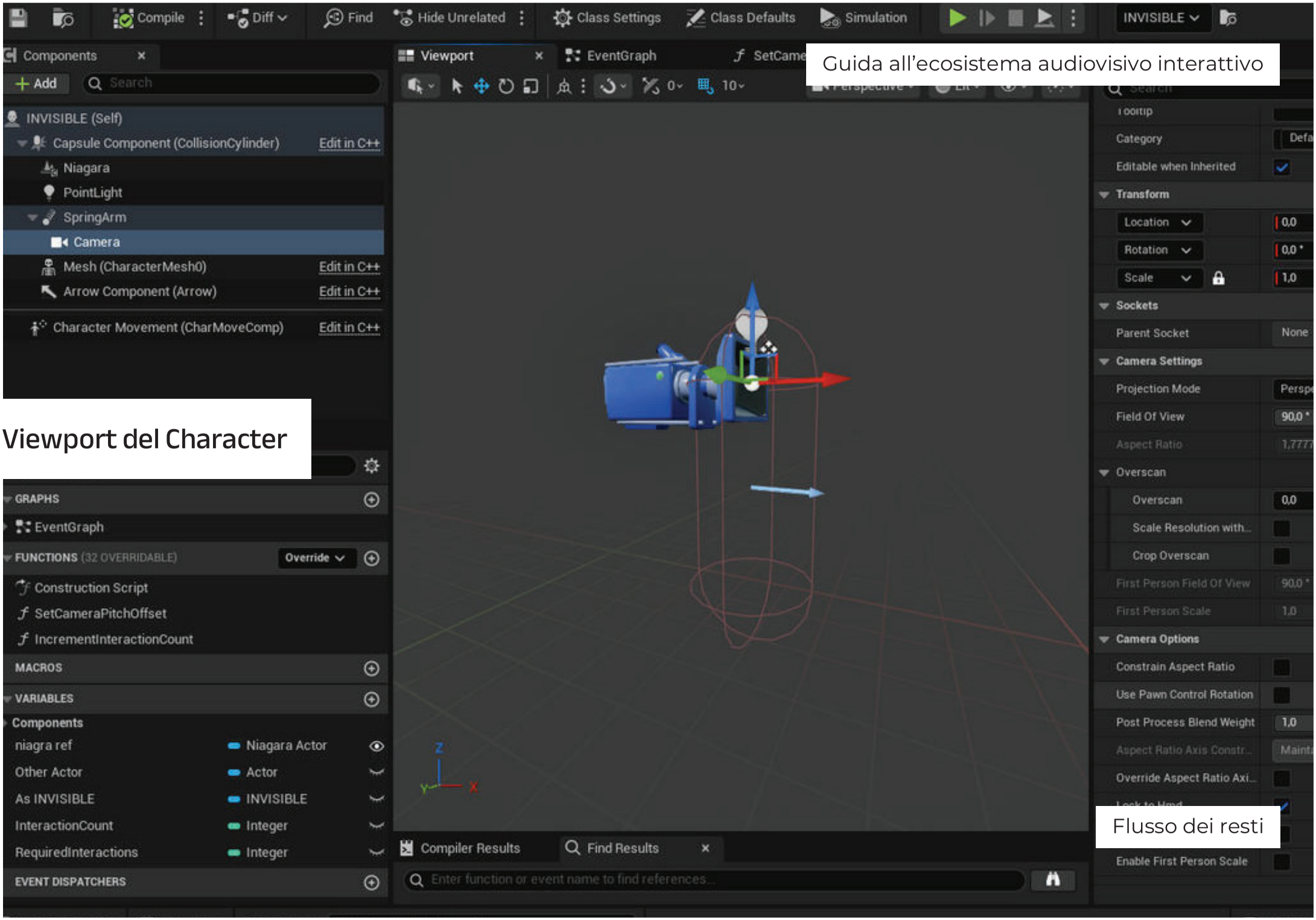
A questo punto, il nostro sistema Niagara non solo è visivamente corretto, ma è anche pronto per l'interazione, con un "gancio" (wind force) che possiamo manipolare tramite Blueprint per creare effetti dinamici.

Questa opera è stata lasciata intenzionalmente aperta.

Creazione dell'attore giocabile e della modalità di gioco

Questo capitolo descrive la procedura fondamentale per creare un personaggio giocabile in prima persona, invisibile, e per configurare il mondo di gioco in modo che questo personaggio venga automaticamente utilizzato all'avvio.

1. Creazione del Blueprint del personaggio (BP_INVISIBILE)
2. Creazione delle Regole del Gioco (Game Mode)
3. Applicazione delle Regole al Livello
4. Verifica del sistema


Compile
Diff
Find
Hide Unrelated
Class Settings
Class Defaults
Simulation
INVISIBLE
Components
Add
Search
INVISIBLE (Self)
Capsule Component (CollisionCylinder)
Edit in C++
Niagara
PointLight
SpringArm
Camera
Mesh (CharacterMesh0)
Edit in C++
Arrow Component (Arrow)
Edit in C++
Character Movement (CharMoveComp)
Edit in C++
Viewport
EventGraph
Guida all'ecosistema audiovisivo interattivo
Viewport del Character
GRAPHS
EventGraph
FUNCTIONS (32 OVERRIDABLE)
Override
Construction Script
SetCameraPitchOffset
IncrementInteractionCount
MACROS
VARIABLES
Components
niagra ref
Niagara Actor
Other Actor
Actor
As INVISIBLE
INVISIBLE
InteractionCount
Integer
RequiredInteractions
Integer
EVENT DISPATCHERS
Compiler Results
Find Results
Enter function or event name to find references...
Tooltip
Category
Editable when Inherited
Transform
Location
Rotation
Scale
Sockets
Parent Socket
None
Camera Settings
Projection Mode
Field Of View
Aspect Ratio
Overscan
Overscan
Scale Resolution with...
Crop Overscan
First Person Field Of View
First Person Scale
Camera Options
Constrain Aspect Ratio
Use Pawn Control Rotation
Post Process Blend Weight
Override Aspect Ratio Axi...
Enable First Person Scale
Flusso dei resti

Il nostro primo passo non è creare un personaggio visibile, ma un "contenitore" invisibile per il giocatore. Questo approccio è ideale per i progetti in prima persona dove il corpo del giocatore non deve essere visibile, permettendoci di concentrarci unicamente sulla telecamera, la collisione e il movimento. Creeremo questo contenitore utilizzando una classe specializzata di Unreal Engine: il Character.

Creazione del Blueprint del personaggio (BP_INVISIBILE)

Il Blueprint è il cuore del nostro personaggio, dove assembleremo i suoi componenti e, in seguito, definiremo il suo comportamento.

1. Creazione del File:
    - Nel Content Browser, cliccare con il tasto destro (o usare il pulsante Add New).
    - Selezionare Blueprint Class.
    - Apparirà la finestra "Pick Parent Class". Scegliere la classe Character. Questa classe è fondamentale perché fornisce nativamente un CharacterMovementComponent, un componente pre-configurato per gestire la fisica del movimento umanoide (camminare, saltare, ecc.).
    - Nominare il nuovo Blueprint BP_INVISIBILE.
2. Configurazione dei Componenti:
    - Aprire BP_INVISIBILE con un doppio clic. Si aprirà l'editor del Blueprint.
    - Nel pannello "Components" (in alto a sinistra), noteremo diversi componenti già presenti (Inherited), forniti dalla classe Character:
        - CapsuleComponent: La forma di collisione del nostro personaggio. È un cilindro arrotondato che definisce i confini fisici del giocatore nel mondo.
        - ArrowComponent: Indica la direzione "in avanti" del personaggio.
        - Mesh: Normalmente conterrebbe il modello 3D del personaggio. Per il nostro scopo, lo lasceremo vuoto o, se presente un mesh di default, lo selezioneremo e nel pannello "Details" disattiveremo la sua visibilità (Rendering -> Visible).

        - CharacterMovement: Il motore fisico del personaggio. Gestisce la velocità, l'accelerazione, la gravità, ecc. Lo configureremo in seguito.

3. Aggiunta della visuale in Prima Persona:
    - Per "vedere" dal punto di vista del personaggio, dobbiamo aggiungere una telecamera.
    - Nel pannello "Components", cliccare su + Add Component e cercare SpringArm. Questo componente agisce come un braccio invisibile che tiene la telecamera, utile per prevenire che la telecamera entri nei muri.
    - Assicurarsi che lo SpringArm sia "figlio" del CapsuleComponent trascinandolo sopra di esso. Posizionare lo SpringArm all'altezza desiderata degli occhi all'interno della capsula.
    - Ora, con lo SpringArm selezionato, cliccare su + Add Component di nuovo e cercare Camera. La telecamera verrà automaticamente attaccata alla fine dello SpringArm.
    - Nel pannello "Details" della Camera, è consigliabile attivare l'opzione Use Pawn Control Rotation. Questo "scollega" la rotazione verticale della telecamera da quella del corpo, permettendo al giocatore di guardare su e giù liberamente.

Creazione delle Regole del Gioco (Game Mode)

Ora che abbiamo un personaggio, dobbiamo dire a Unreal Engine di usarlo ogni volta che il gioco inizia. Questo si fa attraverso un Game Mode.

1. Creazione del Game Mode Base:
    - Nel Content Browser, cliccare con il tasto destro e selezionare Blueprint Class.
    - Nella finestra "Pick Parent Class", selezionare Game Mode Base.
    - Nominare il nuovo Blueprint GM_Base (o un nome simile).
2. Assegnazione del Personaggio di Default:
    - Aprire GM_Base. L'editor è molto più semplice di quello del personaggio.
    - Nel pannello "Details" a destra, cercare la sezione Classes.
    - La prima opzione è Default Pawn Class. "Pawn" (pedina) è il termine generico per un attore controllato dal giocatore.

    - Cliccare sul menu a tendina e selezionare il nostro BP_INVISIBILE.
    - Compilare e salvare.

Applicazione delle Regole al Livello

Abbiamo creato le regole, ora dobbiamo applicarle al nostro livello di gioco.

1. Impostazioni del Mondo (World Settings):
    - Nell'editor principale, se il pannello "World Settings" non è visibile, aprirlo dal menu Window -> World Settings.
    - Nel pannello "World Settings", cercare la sezione Game Mode.
    - Sotto Game Mode Override, cliccare sul menu a tendina e selezionare il nostro GM_Base.
2. Posizionamento del Punto di Partenza:
    - Per dire al Game Mode dove far apparire il personaggio, dobbiamo usare un Player Start.
    - Nel pannello "Place Actors" (o cercando nella libreria), trovare l'attore Player Start e trascinarlo nel livello. Posizionarlo leggermente sopra il pavimento nel punto in cui si desidera che il gioco inizi.

Verifica del sistema

A questo punto, la configurazione di base è completa.

- Non è necessario trascinare BP_INVISIBILE nel livello manualmente.
- Quando si preme il pulsante "Play", il motore eseguirà la seguente sequenza:
    1. Il livello carica e attiva il GM_Base come regola.
    2. Il GM_Base controlla la sua Default Pawn Class, che è BP_INVISIBILE.
    3. Il motore crea un'istanza di BP_INVISIBILE nella posizione del Player Start.
    4. Il controllo viene assegnato al giocatore, che ora vede il mondo attraverso la telecamera del suo attore invisibile.

Il nostro contenitore è pronto. Ora abbiamo una base solida su cui costruire sistemi di controllo più complessi, come quello via OSC.

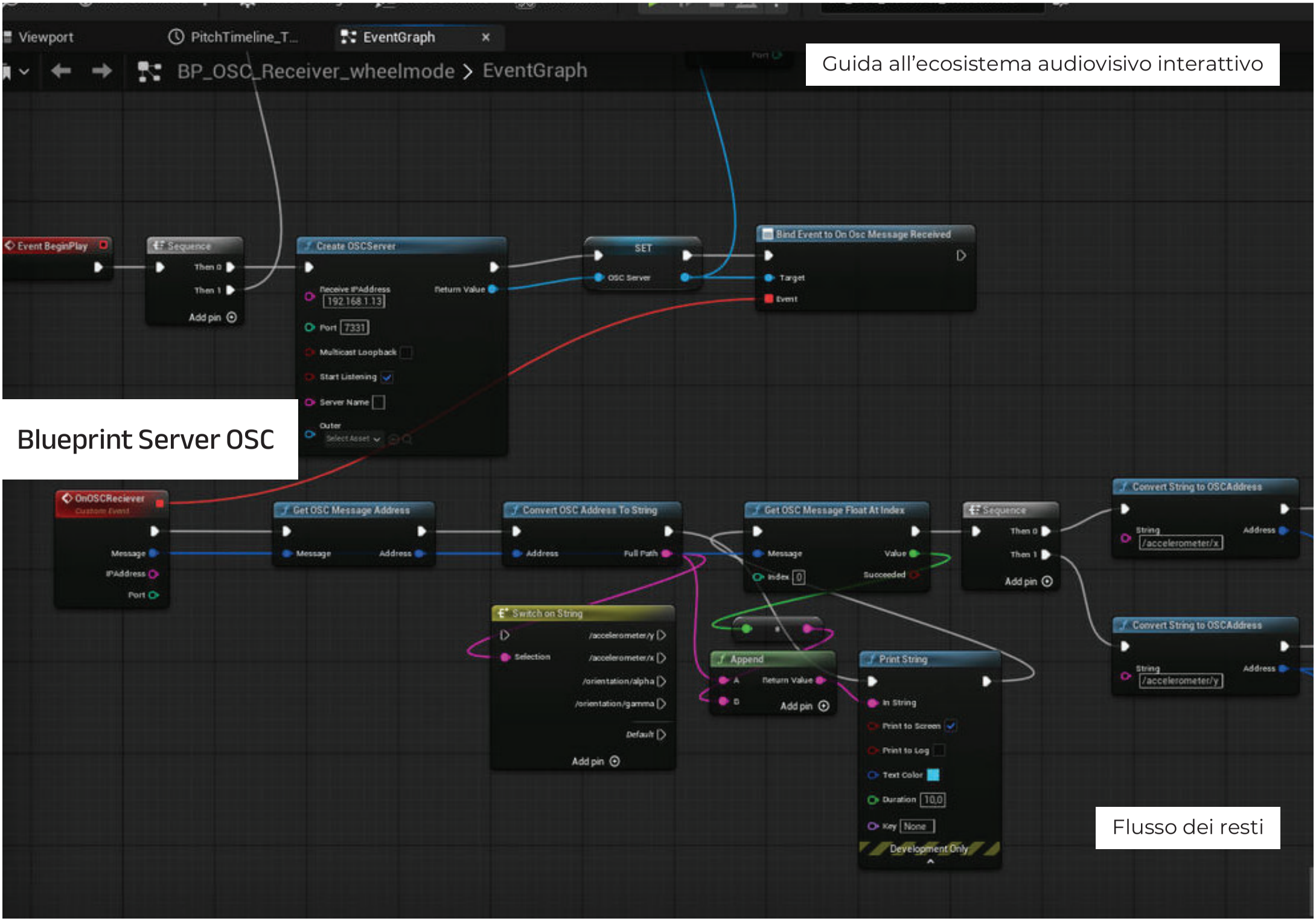
Questo sensore è stato lasciato intenzionalmente agire.

Configurazione e test del ricevitore OSC

Questo capitolo illustra la procedura tecnica per creare e configurare il "cervello" del nostro sistema: un attore dedicato (BP_OSC_Receiver) il cui unico scopo è stabilire una comunicazione di rete via OSC, ricevere messaggi e verificarne la corretta ricezione.

1. Prerequisiti: Indirizzo IP e Firewall
2. Creazione del Blueprint Ricevitore (BP_OSC_Receiver)
3. Logica di inizializzazione e debug (Event BeginPlay)
4. Logica di test e visualizzazione a schermo
5. Finalizzazione e test

Guida all’ecosistema audiovisivo interattivo



Blueprint Server OSC

Flusso dei resti

Prerequisiti: Indirizzo IP e Firewall

Prima di scrivere una sola riga di codice, è fondamentale preparare l'ambiente di rete. Un server OSC è un'applicazione di rete e come tale è soggetto alle regole del sistema operativo.

1. Individuazione dell'Indirizzo IP Locale:
    - Il server OSC deve "legarsi" a un indirizzo IP specifico del computer su cui gira Unreal Engine. È necessario l'indirizzo IPv4 della macchina all'interno della rete locale (LAN).
    - Procedura (Windows): Aprire il Prompt dei Comandi (cercare cmd nel menu Start) e digitare il comando ipconfig. Premere Invio. Tra i risultati, individuare la scheda di rete attiva (es. "Scheda LAN wireless Wi-Fi") e annotare l'indirizzo riportato alla voce "Indirizzo IPv4". Solitamente ha un formato simile a 192.168.1.XX.
    - Questo indirizzo sarà utilizzato sia dal server in Unreal sia dall'applicazione mittente sul dispositivo mobile per sapere dove inviare i dati.
2. Configurazione del Firewall:
    - I sistemi operativi moderni, per sicurezza, bloccano le connessioni di rete in entrata non richieste. La prima volta che si avvia un'applicazione che apre una porta di rete (come il nostro server OSC), il Firewall di Windows (o equivalente) chiederà l'autorizzazione.
    - È obbligatorio concedere l'accesso. Nella finestra di dialogo che apparirà al primo "Play", assicurarsi di spuntare la casella relativa alle "Reti private" e cliccare su "Consenti accesso". Ignorare questo passaggio impedirà la ricezione di qualsiasi dato OSC.

Creazione del Blueprint Ricevitore (BP_OSC_Receiver)

Questo attore non avrà una rappresentazione visiva, sarà puramente logico.

1. Creazione del File:
    - Nel Content Browser, cliccare con il tasto destro e selezionare Blueprint Class.
    - Come classe base ("Parent Class"), scegliere Actor. Questa è la classe più generica per un oggetto che può essere posizionato nel mondo.
    - Nominare il Blueprint BP_OSC_Receiver.

Logica di inizializzazione e debug (Event BeginPlay)

Tutta la configurazione del server avviene all'inizio del gioco. Useremo l'Event BeginPlay per questo.

1. Creazione del Server OSC:
    - Aprire BP_OSC_Receiver e andare nell'Event Graph.
    - Trascinare un filo dal nodo Event BeginPlay e cercare Create OSCServer.
    - Configurazione dei parametri:
        - Receive IP Address: Inserire l'indirizzo IPv4 annotato in precedenza (es. "192.168.1.102"). È importante inserirlo come una stringa, tra virgolette.
        - Port: Inserire un numero di porta non utilizzato, tipicamente superiore a 1024. Un valore comune e facile da ricordare è 8000.
        - Start Listening: Lasciare la spunta attiva per avviare immediatamente il server.
    - Memorizzazione del riferimento: Il server appena creato deve essere accessibile in seguito. Cliccare con il tasto destro sul pin di output Return Value del nodo Create OSCServer e selezionare "Promote to variable". Nominare la nuova variabile OSC_Server. Questo ci permette di interagire con il server in altri punti del codice.
2. Associazione di un evento generico (Bind Event to On OSC Message Received):
    - Per il nostro test iniziale, non ci interessa filtrare i messaggi per indirizzo. Vogliamo catturare qualsiasi messaggio OSC in arrivo per verificare che la connessione funzioni.
    - Trascinare un filo dal pin di esecuzione del Create OSCServer (o dalla variabile OSC_Server appena creata).
    - Cercare il nodo Bind Event to On OSC Message Received.
    - Questo nodo lega un evento custom a ogni singolo messaggio ricevuto dal server, indipendentemente dal suo indirizzo.
3. Creazione dell'evento ricevitore (OnOSCReceived):
    - Trascinare un filo dal pin di output rosso (Event) del nodo Bind... e selezionare Add Custom Event.

- Nominare questo nuovo evento in modo descrittivo, ad esempio OnOSCReceived. Questo evento si attiverà automaticamente ogni volta che un messaggio OSC colpirà la porta 8000.

Logica di test e visualizzazione a schermo

Ora implementiamo la logica all'interno dell'evento OnOSCReceived_Debug per visualizzare a schermo le informazioni ricevute, confermando il successo della comunicazione.

1. Estrazione dei dati dal messaggio:
    - L'evento OnOSCReceived_Debug ha un pin di output chiamato Message. Questo pin contiene tutte le informazioni del pacchetto OSC.
    - Trascinare un filo da Message e cercare Get OSC Message Address. Questo estrae l'indirizzo del messaggio (es. /accelerometer/x).
    - Dall'output di Get OSC Message Address, cercare Convert OSC Address to String per ottenere una stringa di testo standard.
    - Trascinare un altro filo da Message e cercare Get OSC Message Float at Index. Lasciare l'indice a 0. Questo estrae il primo valore numerico (Float) contenuto nel messaggio.
2. Visualizzazione combinata con Append e Print String:
    - Vogliamo stampare una singola riga di testo che contenga sia l'indirizzo che il valore, per un debug completo.
    - Cercare il nodo Append (nella categoria String). Questo nodo concatena più stringhe in una sola. Cliccare su Add pin per avere un totale di 3-4 input.
        - A: Collegare l'output di Convert OSC Address to String.
        - B: Scrivere a mano un separatore per la leggibilità, ad esempio : .
        - C: Collegare l'output di Get OSC Message Float at Index. Il motore convertirà automaticamente il Float in una Stringa.
    - Infine, collegare l'output Return Value del nodo Append all'input In String di un nodo Print String. Impostare la durata (Duration) a 0.0 o 1.0 per evitare di riempire lo schermo.

Finalizzazione e test

1. Posizionamento nel Livello: Trascinare un'istanza del BP_OSC_Receiver dal Content Browser in un punto qualsiasi del livello di gioco. Se non è presente nel livello, il suo codice non verrà mai eseguito.

2. Avvio del Test:

    - Configurare l'app mittente sul dispositivo mobile per inviare i dati all'indirizzo IP e alla porta 8000 del computer.

    - Premere "Play" nell'editor di Unreal Engine.

    - Muovere il dispositivo mobile.

    - Se la configurazione è corretta, nell'angolo in alto a sinistra dello schermo di gioco appariranno in tempo reale le stringhe di debug, ad esempio:

        - /accelerometer/x: 0.542

        - /accelerometer/y: -0.122

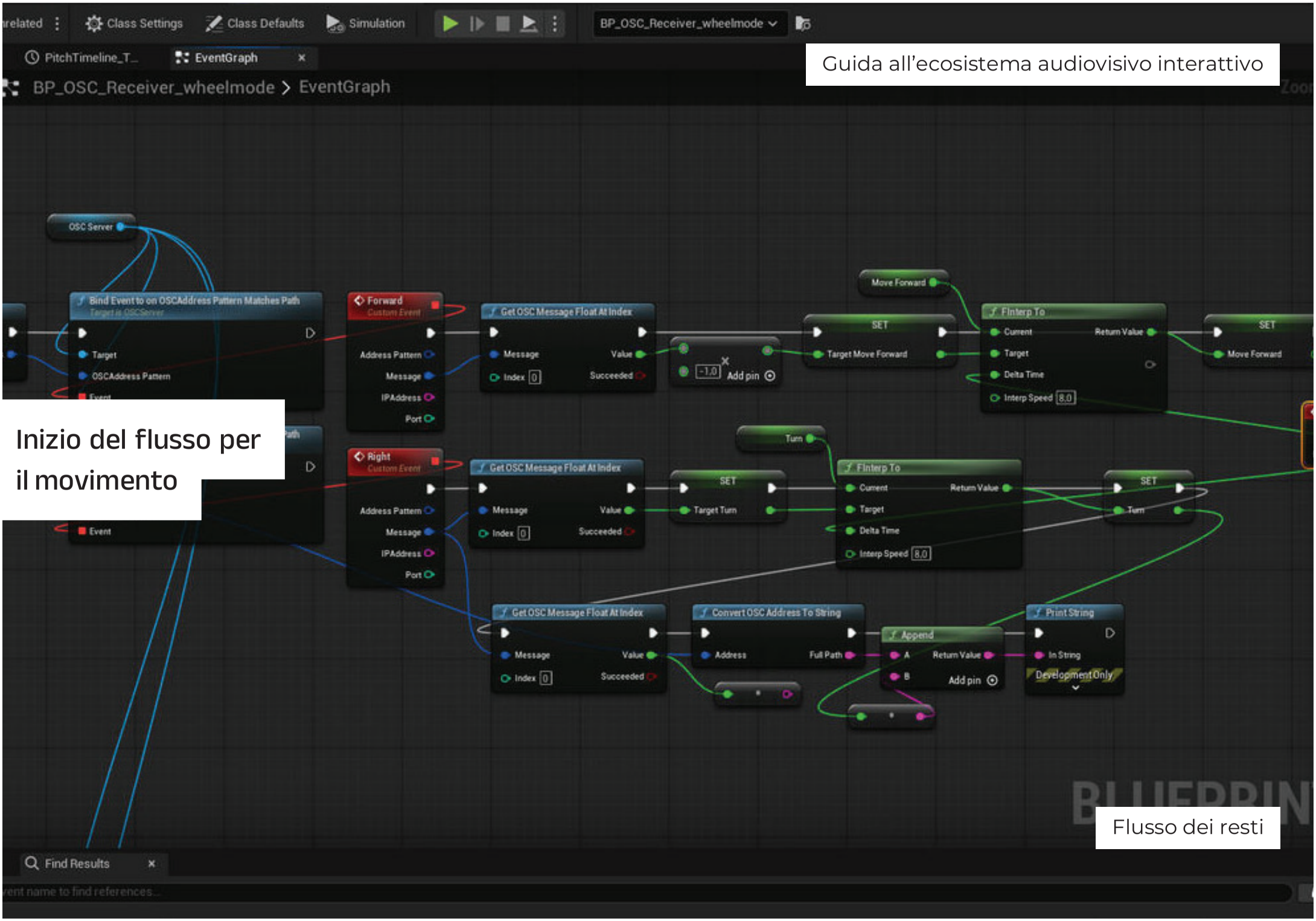
Questa conferma visiva è la prova inconfutabile che la comunicazione di rete è stabilita e funzionante. Da questa base solida, si può procedere a filtrare i messaggi e ad applicare la logica di gioco.

Implementazione del controllo "Modalità Ruota"

Questo capitolo descrive l'implementazione del sistema di controllo principale. L'obiettivo è creare un controllo intuitivo dove un asse del sensore gestisce la "sterzata" (rotazione) del personaggio, mentre un altro asse gestisce l'accelerazione in avanti e indietro rispetto alla nuova direzione.

1. Preparazione del personaggio per il controllo diretto
2. Filtraggio e assegnazione dei segnali OSC
3. Smussatura e applicazione del movimento

Guida all’ecosistema audiovisivo interattivo



Inizio del flusso per il movimento

Flusso dei resti

Preparazione del personaggio per il controllo diretto

Prima di inviare comandi, è necessario configurare il Blueprint del personaggio (BP_INVISIBILE) affinché accetti input di rotazione diretti, ignorando i suoi automatismi.

1. Apertura del Blueprint: Navigare e aprire BP_INVISIBILE.
2. Selezione del CharacterMovementComponent: Nel pannello "Components", selezionare CharacterMovement.
3. Disattivazione della rotazione automatica: Nel pannello "Details", cercare l'opzione booleana Orient Rotation to Movement e assicurarsi che sia disattivata (senza spunta). Questo impedisce al personaggio di girarsi automaticamente nella direzione del movimento, dandoci pieno controllo manuale.
4. Attivazione del controllo da controller: Selezionare il componente radice del Blueprint (INVISIBILE(self)). Nel pannello "Details", sotto la sezione "Pawn", assicurarsi che l'opzione Use Controller Rotation Yaw sia attivata. Questo comanda al personaggio di applicare fedelmente qualsiasi input di rotazione orizzontale (Yaw) che riceve.

Con queste impostazioni, il nostro "corpo" è pronto a ricevere ordini di sterzata e accelerazione separati.

Filtraggio e assegnazione dei segnali OSC (BP_OSC_Receiver)

La logica di ricezione deve ora filtrare i messaggi OSC in base al loro indirizzo e assegnarli a eventi specifici.

1. Logica in Event BeginPlay: Dopo il nodo Create OSCServer, inizia la logica di associazione (binding).
2. Preparazione dell'indirizzo: Per ogni indirizzo OSC che ci interessa (es. /accelerometer/x, /accelerometer/y), si utilizza il nodo Convert String to OSCAddress. Si inserisce l'indirizzo testuale (es. "/accelerometer/x") nel pin di input String.
3. Associazione dell'evento: L'output OSCAddress del nodo precedente viene collegato al pin di input OSC Address Pattern del nodo Bind Event to on OSCAddress Pattern Matches Path.
4. Creazione dell'evento Custom: Dal pin rosso Event del nodo Bind..., si trascina un filo per creare un Custom Event. Questo evento viene nominato in modo descrittivo in base alla sua funzione. Per questo sistema:
    - /accelerometer/x viene associato a un evento custom Forward_Input.

    - /accelerometer/y viene associato a un evento custom Turn_Input.

5. Estrazione dei dati: All'interno di ogni evento custom (es. Forward_Input), il pin Message viene usato per estrarre il valore numerico. Si collega al nodo Get OSC Message Float at Index (con indice 0) per leggere il primo (e unico) valore float del messaggio.

6. Salvataggio negli "Obiettivi": Il valore float estratto viene salvato in una variabile Target per la successiva smussatura.

    - Forward_Input imposta la variabile TargetMoveForwardValue.

    - Turn_Input imposta la variabile TargetTurnValue.

    - Nota: In questa implementazione, si assume che i valori inviati dall'app siano già in un range adeguato (es. -1 a 1), rendendo il nodo Map Range Clamped opzionale.

Smussatura e applicazione del movimento (Event Tick del BP_OSC_Receiver)

L'Event Tick è il motore che, ad ogni frame, legge gli "obiettivi", calcola i valori smussati e li applica al personaggio.

1. Interpolazione del movimento:

    - Per garantire un'accelerazione fluida, si utilizza un nodo FInterp To.

    - Il pin Delta Seconds dell'Event Tick viene collegato all'input Delta Time.

    - Current è collegato a Get MoveForwardValue.

    - Target è collegato a Get TargetMoveForwardValue.

    - Il risultato (Return Value) viene usato per aggiornare (Set) la variabile MoveForwardValue, chiudendo il ciclo di feedback.

2. Riferimento al personaggio:

    - Si utilizza il nodo Get Player Character, il cui output viene collegato all'input Object di un nodo Cast To BP_INVISIBILE. Questo conferma che il giocatore sta controllando il nostro personaggio e ci fornisce un riferimento diretto ad esso.

3. Sequenza di azioni:

- Dall'output di esecuzione del Cast, si collega un nodo Sequence per gestire la rotazione e il movimento come due operazioni indipendenti.

4. Logica di movimento (Catena Then 0):
    - Il flusso di esecuzione prosegue verso un Branch per la logica della Dead Zone (ignora MoveForwardValue se il suo valore assoluto è troppo piccolo).
    - Se il Branch è True:
        - Si crea un nodo Add Movement Input. Il suo Target è il riferimento As INVISIBILE dal Cast.
        - Il suo Scale Value è la variabile smussata MoveForwardValue.
        - Per la direzione, si usa il nodo Get Actor Forward Vector (il cui Target è anch'esso il riferimento As INVISIBILE), collegandolo all'input World Direction. Questo assicura che il movimento sia sempre relativo alla direzione attuale del personaggio.
5. Logica di rotazione (Catena Then 1):
    - Il flusso di esecuzione prosegue verso il nodo Add Controller Yaw Input. Il suo Target è il riferimento As INVISIBILE.
    - Il suo Val è collegato direttamente alla variabile TargetTurnValue (o a una versione smussata TurnValue se si desidera anche una sterzata fluida).
    - Opzionale: Per regolare la reattività, il valore TurnValue può essere moltiplicato per una variabile Float (es. TurnSensitivity) prima di essere passato al nodo di input.

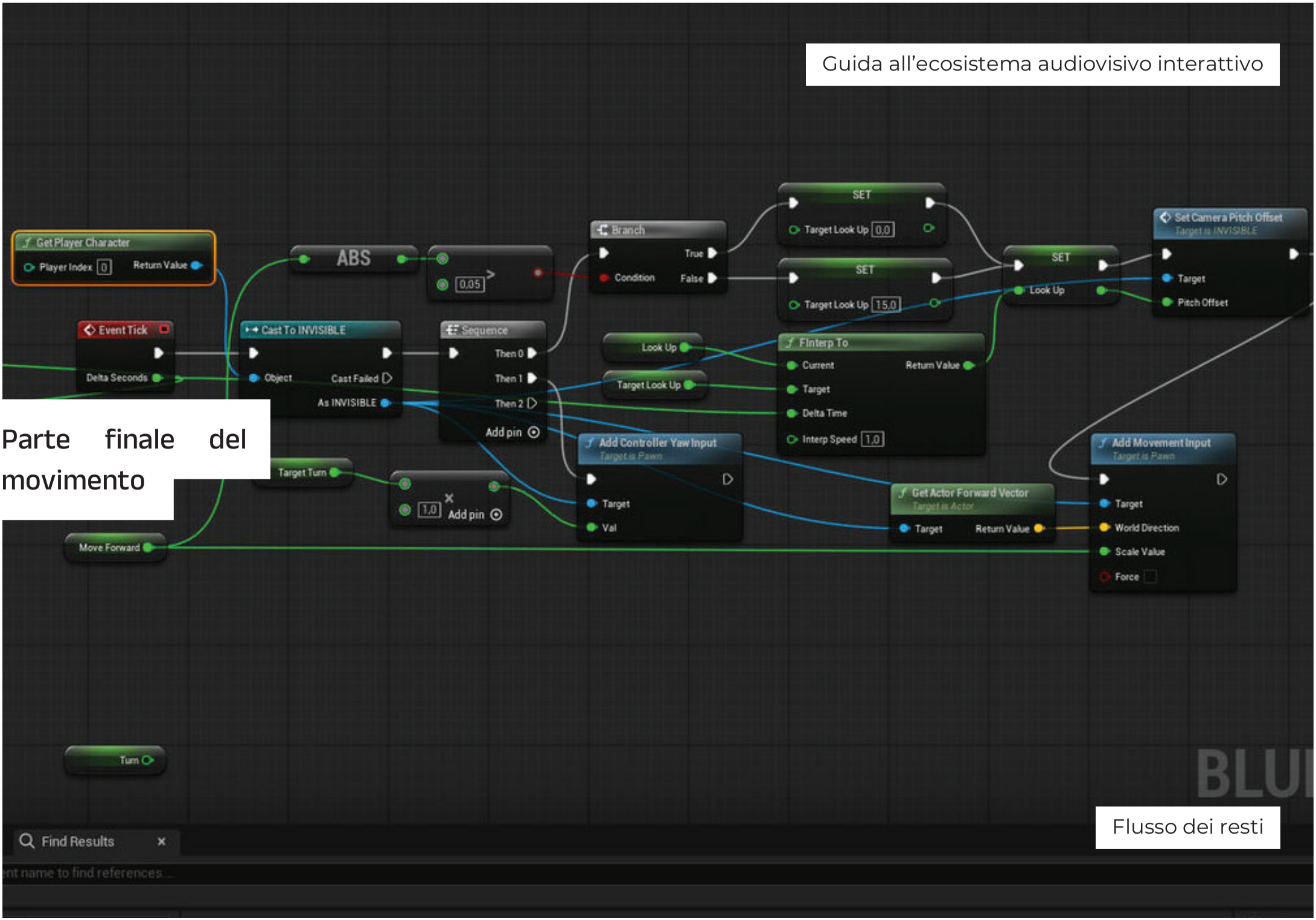
Il risultato è un sistema di controllo dove il giocatore "sterza" il proprio punto di vista con un asse e "accelera" in quella direzione con l'altro, con un movimento e un'accelerazione fluidi e gestiti da una fisica credibile.

Questo dato è stato lasciato intenzionalmente incompleto.

Implementazione della "Modalità Osservazione"

Questo capitolo descrive la creazione di una meccanica di gioco contestuale: una "modalità osservazione" che si attiva automaticamente quando il giocatore si ferma. L'obiettivo è far inclinare dolcemente la visuale del giocatore verso l'alto per osservare gli effetti particellari, e farla tornare alla posizione normale quando il movimento riprende.

1. Concetto architettonico: separazione dei compiti
2. Creazione della funzione
3. Implementazione della Logica di Controllo



Parte finale del movimento

Flusso dei resti

Concetto architettonico: separazione dei compiti

Per garantire un sistema stabile e senza conflitti, la logica viene divisa tra i nostri due Blueprint principali:

1. BP_OSC_Receiver (Il Cervello): Sarà responsabile di decidere quando attivare la modalità osservazione (cioè, quando il giocatore è fermo) e di calcolare il valore di inclinazione (pitch) desiderato in modo fluido e graduale.
2. BP_INVISIBILE (Il Corpo): Sarà responsabile di eseguire meccanicamente l'ordine di inclinazione. Conterrà una funzione dedicata che riceve un valore numerico e lo applica direttamente alla rotazione della sua telecamera.

Creazione della funzione (BP_INVISIBILE)

Il primo passo è dotare il nostro personaggio della capacità meccanica di inclinare la sua visuale su comando.

1. Creazione della Funzione:
    - Aprire il Blueprint BP_INVISIBILE.
    - Nel pannello "My Blueprint", creare una nuova Funzione (+ Function) e nominarla SetCameraPitchOffset.
2. Definizione dell'input:
    - Selezionare il nodo di inizio della funzione. Nel pannello "Details" a destra, aggiungere un nuovo parametro di Input.
    - Nominare l'input PitchOffset e impostare il suo tipo di dato su Float. Questa funzione ora si aspetta di ricevere un singolo valore numerico.
3. Implementazione della logica:
    - All'interno del grafico della funzione, trascinare un riferimento al componente SpringArm.
    - Dal riferimento allo SpringArm, creare il nodo Set Relative Rotation. Questo nodo imposta la rotazione di un componente rispetto al suo genitore, perfetto per inclinare la telecamera senza ruotare l'intero personaggio.
    - Per l'input New Rotation, creare un nodo Make Rotator.

    - Collegare il pin di input della funzione, PitchOffset, all'input Y (Pitch) del Make Rotator. Gli input X (Roll) e Z (Yaw) vengono lasciati a 0.

    - Compilare e salvare il Blueprint. BP_INVISIBILE è ora in grado di eseguire l'azione "imposta l'inclinazione della telecamera a un valore specifico".

Implementazione della Logica di Controllo (BP_OSC_Receiver)

Ora, nel "Cervello", implementiamo la logica che decide quando e come chiamare la funzione appena creata.

Creazione delle Variabili di Stato: Aprire BP_OSC_Receiver. Nel pannello "My Blueprint", creare due nuove variabili di tipo Float:

- CurrentPitchOffset: Memorizzerà il valore di inclinazione attuale, calcolato ad ogni frame. Valore di default: 0.0.

- TargetPitchOffset: Memorizzerà il valore di inclinazione che vogliamo raggiungere (0 o -15). Valore di default: 0.0.

Logica nell'Event Tick: tutta la logica verrà eseguita all'interno dell'Event Tick per garantire transizioni fluide.

1. Determinazione dell'obiettivo:

    - Individuare il Branch della Dead Zone per il movimento in avanti.

    - Dal pin di esecuzione True (il giocatore si sta muovendo), creare un nodo Set TargetPitchOffset e impostare il suo valore a 0.0.

    - Dal pin di esecuzione False (il giocatore è fermo), creare un altro nodo Set TargetPitchOffset e impostare il suo valore a -15.0.

2. Interpolazione del valore:

    - Dopo i due percorsi del Branch inserire due di interpolazione.

    - Creare un nodo FInterp To:

        - Delta Time: Collegare il Delta Seconds dell'Event Tick.

        - Interp Speed: Impostare una velocità di transizione bassa per un effetto morbido (es. 2.0).

        - Current: Collegare un Get CurrentPitchOffset.
        - Target: Collegare un Get TargetPitchOffset.
    - Il Return Value del FInterp To viene usato per aggiornare (Set) la variabile CurrentPitchOffset, chiudendo il ciclo di feedback.

3. Chiamata alla funzione:
    - Dopo l'aggiornamento di CurrentPitchOffset, è il momento di inviare l'ordine al personaggio.
    - Prendere il riferimento As BP_INVISIBILE dal Cast.
    - Da questo riferimento, chiamare la funzione SetCameraPitchOffset.
    - Collegare la variabile aggiornata CurrentPitchOffset all'input PitchOffset della funzione.

Ora quando il giocatore si muove, il TargetPitchOffset viene costantemente impostato a 0. FInterp To spinge gradualmente CurrentPitchOffset verso 0, e questo valore viene inviato alla funzione in BP_INVISIBILE, che raddrizza la telecamera. Appena il giocatore si ferma, il TargetPitchOffset scatta a -15. FInterp To inizia a spingere CurrentPitchOffset verso -15, e la funzione SetCameraPitchOffset riceve valori progressivamente più negativi, inclinando dolcemente la visuale verso l'alto fino a stabilizzarsi.

Questa scansione è stata lasciata intenzionalmente deformata.

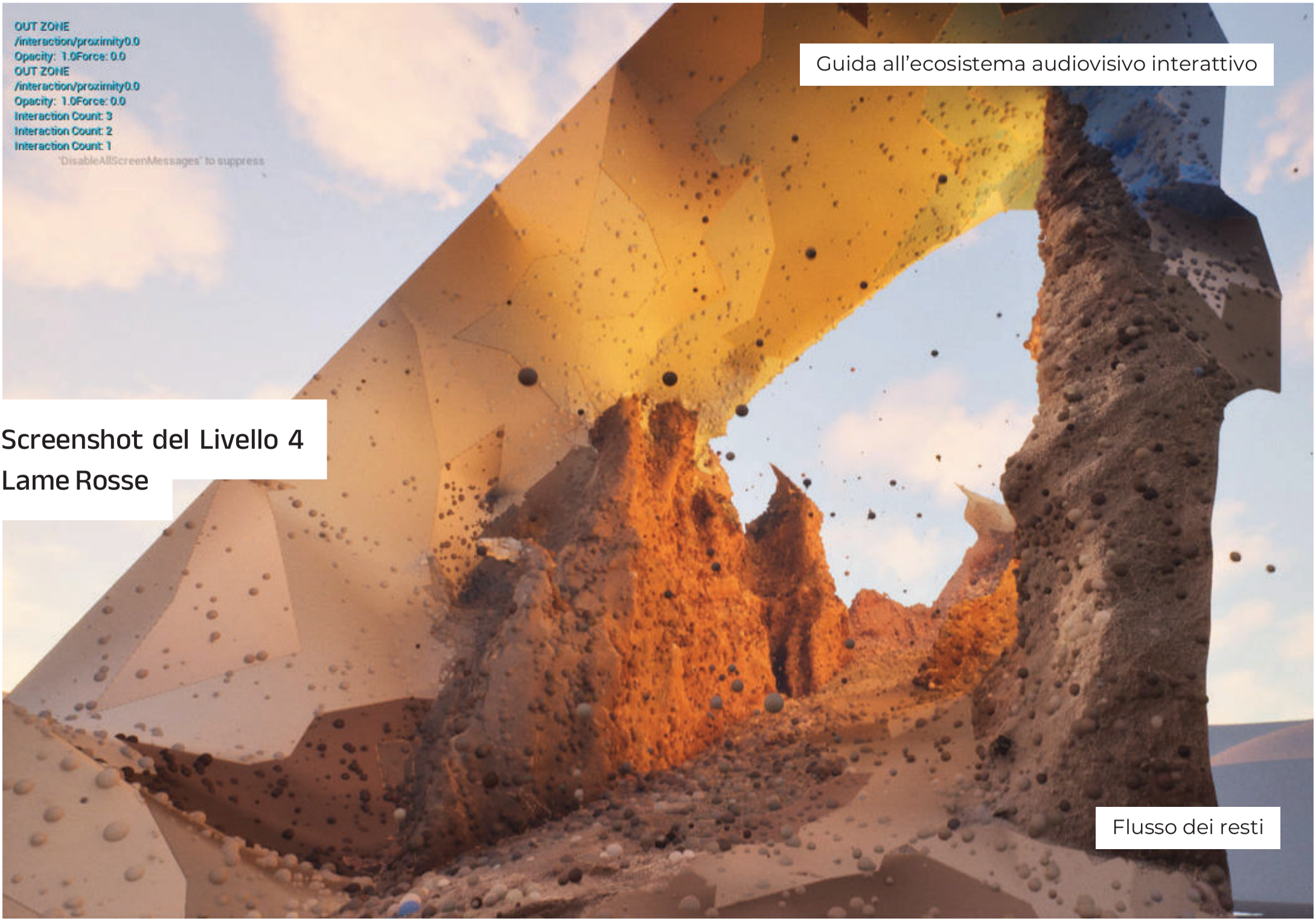

FALSE ZONE
/interaction/proximity0.0
Opacity: 1.0Force: 0.0
FALSE ZONE
/interaction/proximity0.0
Opacity: 1.0Force: 0.0
'DisableAllScreenMessages' to suppress
Guida all’ecosistema audiovisivo interattivo
Screenshot del Livello 2
Sentiero
Flusso dei resti



Screenshot del Livello 3
Parete di roccia



Screenshot del Livello 5 Tunnel

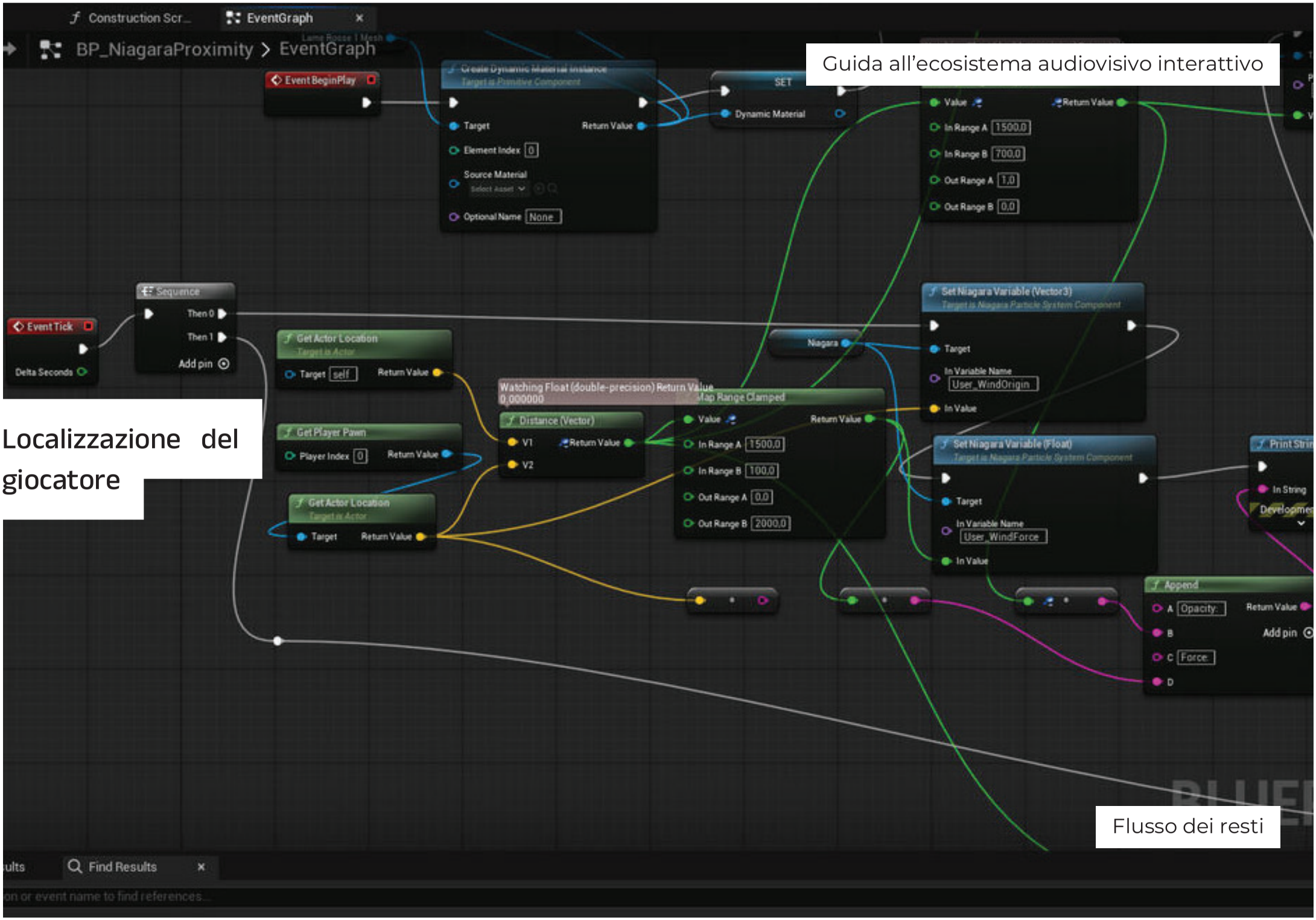


Screenshot del Livello 4
Lame Rosse

Generazione di un effetto di prossimità tra il giocatore e la nuvola di punti

Questo capitolo descrive la procedura per l'implementazione di un sistema reattivo in cui le particelle vengono dinamicamente influenzate dalla prossimità dell'attore controllato. La forza applicata si origina dalla posizione di quest'ultimo e la sua intensità è modulata in funzione della distanza. La metodologia proposta centralizza la logica di controllo all'interno di un singolo Blueprint per facilitare la costruzione e la calibrazione del sistema.

1. Modifica dell'Emitter Niagara di Base
2. Configurazione della Dinamica delle Particelle
3. Sviluppo del Blueprint Actor di Test
4. Logica di Prossimità nel Blueprint
5. Test e Calibrazione


Construction Scr_
EventGraph
BP_NiagaraProximity > EventGraph
Guida all’ecosistema audiovisivo interattivo
Event BeginPlay
Create Dynamic Material Instance
Target is Primitive Component
Target
Return Value
Element Index 0
Source Material
Optional Name None
SET
Dynamic Material
Value
Return Value
In Range A 1500,0
In Range B 700,0
Out Range A 1,0
Out Range B 0,0
Sequence
Then 0
Then 1
Add pin
Event Tick
Delta Seconds
Get Actor Location
Target is Actor
Target self
Return Value
Niagara
Set Niagara Variable (Vector3)
Target is Niagara Particle System Component
Target
In Variable Name User_WindOrigin
In Value
Watching Float (double-precision) Return Value
0,000000
Map Range Clamped
Value
Return Value
In Range A 1500,0
In Range B 100,0
Out Range A 0,0
Out Range B 2000,0
Get Player Pawn
Player Index 0
Return Value
Distance (Vector)
V1
V2
Return Value
Get Actor Location
Target is Actor
Target
Return Value
Set Niagara Variable (Float)
Target is Niagara Particle System Component
Target
In Variable Name User_WindForce
In Value
Print String
In String
Append
A Opacity:
B
C Force:
D
Return Value
Add pin
Localizzazione del giocatore
Flusso dei resti
Find Results
on or event name to find references

Modifica dell'Emitter Niagara di Base

Per permettere l'interfacciamento con il sistema di scripting Blueprint, è necessaria la definizione di due parametri nel pannello User Exposed Parameters. Eventuali parametri preesistenti vengono rimossi per garantire una configurazione pulita.

- User.WindOrigin (Tipo: Vector): Questo parametro è destinato a ricevere le coordinate spaziali del giocatore.
- User.WindForce (Tipo: Float): Questo parametto riceverà il valore calcolato per l'intensità della forza..

Configurazione della Dinamica delle Particelle (Point Force)

Il modulo principale della dinamica è un Point Force, i cui parametri Force Origin e Force Strength vengono trascinati, rispettivamente, ai parametri utente User.WindOrigin e User.WindForce. Il raggio di influenza della forza, Force Falloff Distance, viene impostato su un valore numerico fisso (es., 300.0).

Sviluppo del Blueprint Actor di Test

Il contenitore per l'effetto e la sua logica è un Blueprint Class di tipo Actor, nominato BP_InteractiveObject. All'interno di questo Blueprint vengono aggiunti i componenti necessari: un Static Mesh Component con la mesh della fotogrammetria e un Niagara Particle System Component, a cui viene assegnato l'asset NS_PointCloud. Entrambi possono essere visualizzati tramite il viewport e devono avere la stessa posizione e scala.

Logica di Prossimità nel Blueprint

La logica di controllo, eseguita a ogni frame, è all'interno dell'Event Graph di BP_InteractiveObject. A partire dal nodo evento Event Tick, il flusso di dati è il seguente:

1. Acquisizione delle Posizioni: Vengono ottenute le coordinate spaziali del giocatore (tramite Get Player Pawn e Get Actor Location) e dell'oggetto stesso (tramite Get Actor Location con self come target).
2. Calcolo della Distanza: La distanza tra le due posizioni viene calcolata per mezzo di un nodo Distance.
3. Mappatura Distanza-Forza: Questa distanza alimenta un nodo Map Range Clamped, che funge da nucleo della logica di controllo. Questo nodo converte il range di distanza in un range di forza. I parametri sono configurati in questo modo:
    - In Range A: 1500.0 (Distanza massima per l'attivazione).

    - In Range B: 100.0 (Distanza per l'intensità massima).
    - Out Range A: 0.0 (Intensità della forza alla distanza massima).
    - Out Range B: 2000.0 (Intensità della forza alla distanza minima).
4. Trasmissione dei Dati a Niagara: I valori calcolati vengono infine trasmessi al sistema Niagara. Un nodo Set Niagara Variable (Vector) aggiorna il parametro User.WindOrigin con la posizione del giocatore. Contemporaneamente, un nodo Set Niagara Variable (Float) aggiorna User.WindForce con il valore di forza risultante dalla mappatura del Map Range Clamped. Entrambi i nodi hanno come Target il Niagara Particle System Component.

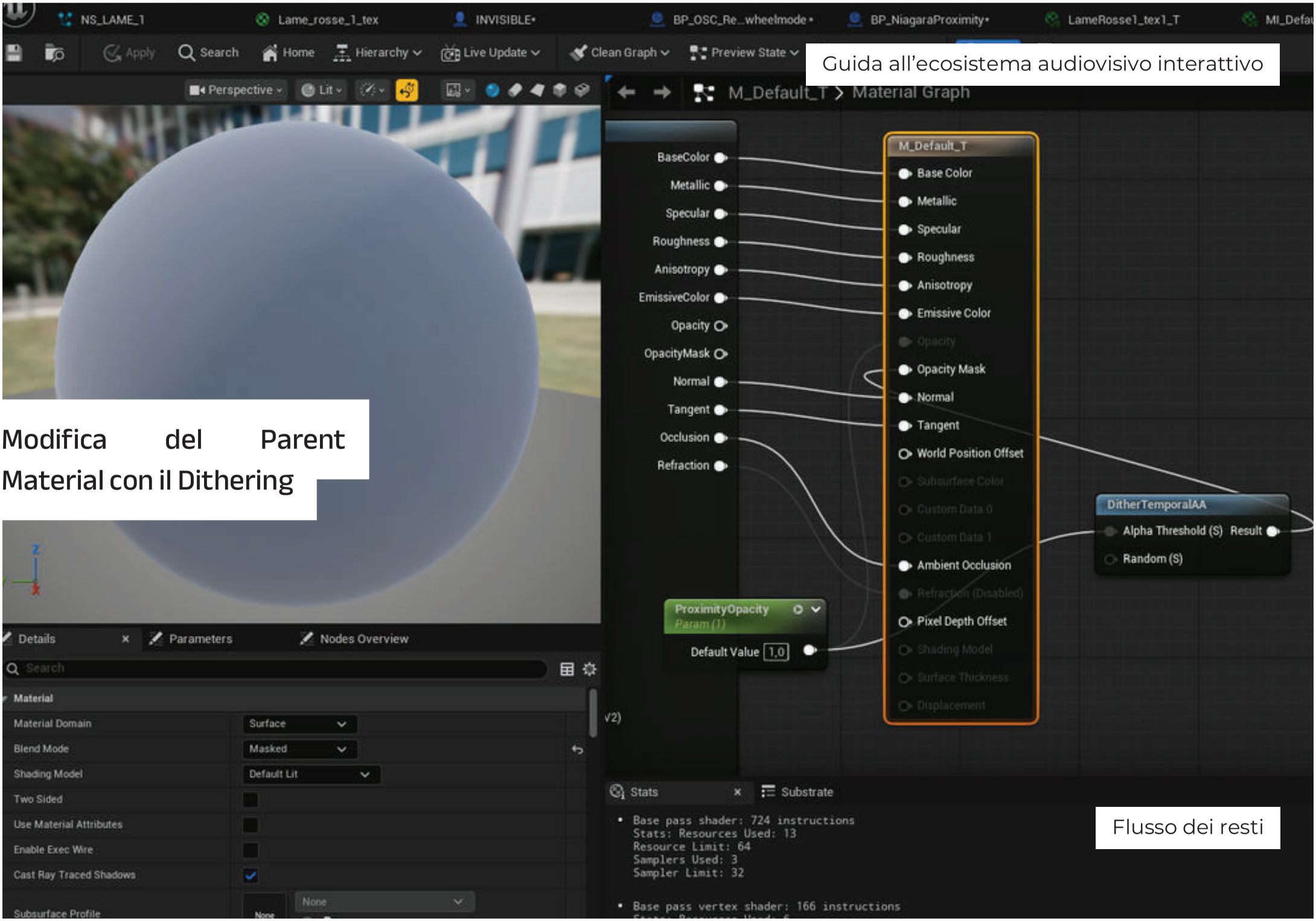
Test e Calibrazione

La fase finale consiste nel posizionare un'istanza del BP_InteractiveObject nel livello e avviare la simulazione. Durante l'esecuzione le particelle reagiranno alla vicinanza del giocatore. La calibrazione dell'effetto si ottiene modificando i quattro parametri di input del nodo Map Range Clamped all'interno del BP_InteractiveObject, permettendo di controllare il raggio di interazione e l'intensità della forza applicata.

Integrazione della Dissolvenza per Prossimità sulla Static Mesh

In aggiunta all'effetto particellare, viene implementato un sistema di dissolvenza sulla Static Mesh solida. L'obiettivo è rendere la mesh visibile a distanza e farla svanire gradualmente all'avvicinarsi del giocatore, creando un'interazione fluida e prevenendo l'occlusione della visuale.Per preservare la qualità fotorealistica dell'illuminazione del materiale originale, si adotta una tecnica di Temporal Dithering, che simula la trasparenza attraverso il Blend Mode: Masked anziché Translucent.

1. Adattamento del Materiale per il Dithering
2. Logica di Controllo nel Blueprint
3. Creazione del sistema Niagara di base
4. Test e Risultato Finale


Guida all’ecosistema audiovisivo interattivo
Modifica del Parent Material con il Dithering
Flusso dei resti
M_Default_T > Material Graph
BaseColor
Metallic
Specular
Roughness
Anisotropy
EmissiveColor
Opacity
OpacityMask
Normal
Tangent
Occlusion
Refraction
Base Color
Metallic
Specular
Roughness
Anisotropy
Emissive Color
Opacity Mask
Normal
Tangent
World Position Offset
Ambient Occlusion
Pixel Depth Offset
DitherTemporalAA
Alpha Threshold (S)
Result
Random (S)
ProximityOpacity
Param (1)
Default Value
1,0
Details
Parameters
Nodes Overview
Material
Material Domain
Surface
Blend Mode
Masked
Shading Model
Default Lit
Two Sided
Use Material Attributes
Enable Exec Wire
Cast Ray Traced Shadows
Subsurface Profile
Stats
Substrate
Base pass shader: 724 instructions
Stats: Resources Used: 13
Resource Limit: 64
Samplers Used: 3
Sampler Limit: 32
Base pass vertex shader: 166 instructions

Adattamento del Materiale per il Dithering

Il processo richiede la modifica del materiale genitore (Parent Material) per introdurre la logica di dissolvenza in modo non distruttivo.

1. Localizzazione e Duplicazione del Parent Material: Si identifica il Parent Material del Material Instance utilizzato dalla mesh (e.g., M_Default). Per preservare l'integrità degli asset originali, questo materiale viene duplicato e rinominato, ad esempio, in M_Default_DitherFade.

2. Configurazione del Nuovo Parent Material: Si apre l'editor del materiale M_Default_DitherFade.

    - Nel pannello Details del nodo principale, si verifica che il Blend Mode sia impostato su Masked. Questo Blend Mode permette di mantenere il modello di illuminazione di alta qualità tipico degli oggetti opachi.

    - Si crea un nodo Scalar Parameter. Lo si nomina in modo univoco, "ProximityOpacity", e si imposta il suo Default Value a 1.0. Questo parametro fungerà da controllo per la dissolvenza.

    - Viene introdotto un nodo DitherTemporalAA. Questo nodo genera un pattern di dithering ad alta frequenza, che risulta stabile con l'anti-aliasing temporale (TAA), creando un'illusione di trasparenza morbida.

    - L'output del Scalar Parameter "ProximityOpacity" viene collegato all'input Alpha del nodo DitherTemporalAA.

    - L'output del nodo DitherTemporalAA viene infine collegato all'input Opacity Mask del nodo principale del materiale.

    - Si salva il materiale modificato.

3. Creazione e Applicazione della Nuova Material Instance: Si crea una nuova catena di Material Instance basata sul Parent Material M_Default_DitherFade, replicando la gerarchia originale se necessario. L'istanza finale (e.g., MI_Photogrammetry_InteractiveFade) viene quindi applicata alla Static Mesh Component all'interno del Blueprint BP_InteractiveObject.

Logica di Controllo nel Blueprint

La logica nel Blueprint gestisce il calcolo della prossimità e l'invio del valore di opacità al materiale.

1. Creazione dell'Istanza Dinamica del Materiale: Nell'evento Event Begin Play del BP_InteractiveObject, viene utilizzato un nodo Create

Dynamic Material Instance sul Static Mesh Component. L'istanza risultante viene salvata in una variabile (DynamicMaterial) per un accesso efficiente durante l'esecuzione.

2. Aggiornamento della Logica nell'Event Tick:
    - Viene riutilizzato il calcolo della Distance tra il giocatore e l'oggetto.
    - Si impiega un nodo Map Range Clamped per mappare il range di distanza a un range di opacità.
        - In Range A: 800.0 (Distanza massima, opacità piena).
        - In Range B: 300.0 (Distanza minima, opacità zero).
        - Out Range A: 1.0 (Valore di opacità quando il giocatore è lontano).
        - Out Range B: 0.0 (Valore di opacità quando il giocatore è vicino).
    - Il valore di opacità calcolato viene trasmesso al materiale tramite un nodo Set Scalar Parameter Value, il cui Target è la variabile DynamicMaterial. Il Parameter Name deve corrispondere esattamente a quello definito nel materiale: "ProximityOpacity".

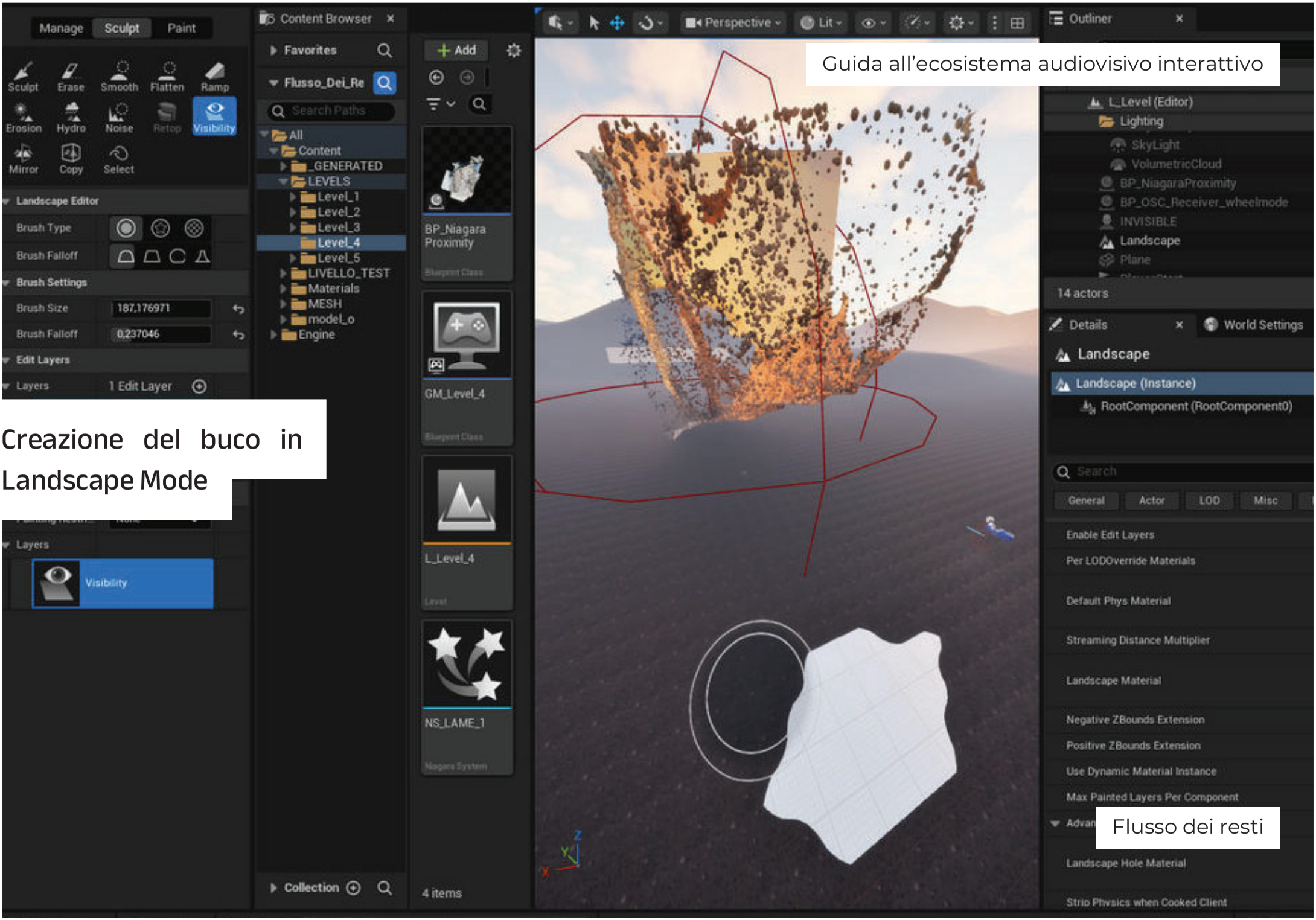
Test e Risultato Finale

Avviando la simulazione, il comportamento atteso è il seguente: la Static Mesh mantiene la sua resa fotorealistica a distanza. Avvicinandosi, il pattern di dithering controllato dal Scalar Parameter "ProximityOpacity" inizierà a "erodere" la superficie della mesh in modo uniforme e morbido, simulando una dissolvenza fino alla semi-invisibilità, mentre l'effetto particellare si attiva in modo complementare. Questo approccio garantisce sia l'interattività desiderata sia la massima fedeltà visiva.

Implementazione della caduta e respawn con Landscape Visibility Mask

Una meccanica fondamentale per la robustezza di qualsiasi gioco è la gestione della "morte" del giocatore per caduta fuori dal mondo di gioco. Questo capitolo descrive come creare un'area "bucata" nel livello utilizzando la tecnica standard della Visibility Mask e come implementare un sistema di respawn automatico.

1. Preparazione del materiale del terreno
2. Creazione di un buco nel terreno
3. Architettura del sistema di Respawn
4. Implementazione della Rilevazione della Caduta
5. Implementazione della Gestione del Respawn


Guida all’ecosistema audiovisivo interattivo
Creazione del buco in Landscape Mode
Flusso dei resti
Manage
Sculpt
Paint
Content Browser
Outliner
Perspective
Lit
Details
World Settings
Landscape
Visibility
14 actors
4 items

Preparazione del materiale del terreno

Per poter "cancellare" parti del terreno, il suo materiale deve essere configurato per supportare la trasparenza basata su una maschera speciale.

1. Aprire il Materiale del Landscape:
    - Individuare e aprire il Material utilizzato dall'attore Landscape nel livello.
2. Impostare la Modalità Masked:
    - Selezionare il nodo principale del materiale (il nodo finale "Material Attributes" o "Output").
    - Nel pannello "Details" a sinistra, sotto la sezione "Material", cambiare il Blend Mode da Opaque a Masked.
    - L'opacità mascherata (Masked) è una forma di trasparenza efficiente che definisce ogni pixel come completamente visibile o completamente invisibile, ideale per creare buchi con bordi netti.
3. Aggiungere il Nodo Landscape Visibility Mask:
    - Cliccare con il tasto destro in un punto vuoto del grafico del materiale e cercare il nodo LandscapeVisibilityMask.

Questo nodo speciale funge da ponte tra gli strumenti di scultura del Landscape e il materiale.

- Collegare il suo unico pin di output all'input Opacity Mask del nodo principale del materiale.
- Salvare e chiudere il materiale.

Creazione di un buco nel terreno

Ora che il materiale è pronto, possiamo usare gli strumenti del Landscape per "dipingere" l'invisibilità.

1. Selezione e Modalità Landscape:
    - Nell'editor principale, selezionare l'attore Landscape.
    - Passare alla modalità Landscape (scorciatoia da tastiera: Shift+2).
2. Utilizzo dello Strumento Visibility:

    - Nel pannello degli strumenti a sinistra, selezionare la tab Sculpt.
    - Tra gli strumenti disponibili, scegliere lo strumento Visibility.
    - Nella viewport, il cursore si trasformerà in un pennello.
    - Per creare un buco, assicurarsi che il pennello sia impostato per "dipingere" (il primo riquadro, completamente nero, nella palette dello strumento).
    - Cliccare e trascinare sul Landscape. Le sezioni del terreno su cui si dipinge diventeranno istantaneamente invisibili, creando un buco attraverso cui il giocatore può cadere. Per ripristinare la visibilità, usare il secondo riquadro (completamente bianco).

Architettura del sistema di Respawn

Il sistema è diviso logicamente in due parti per una migliore organizzazione del codice: la rilevazione e la gestione.

- BP_INVISIBILE (Il Personaggio): Ha la responsabilità di rilevare la propria condizione. Controlla costantemente la sua posizione e, se cade al di sotto di una certa soglia, notifica al sistema di gioco che è "morto".
- GM_Base (Il Game Mode): Ha la responsabilità di gestire l'evento. Riceve la notifica dal personaggio e orchestra la sequenza di azioni necessarie per far respawnare il giocatore: distruggere il vecchio corpo, trovare un punto di partenza e creare un nuovo corpo.

Implementazione della Rilevazione della Caduta (BP_INVISIBILE)

Questa logica viene eseguita ad ogni frame per garantire una rilevazione immediata.

1. Controllo Continuo nell'Event Tick:
    - Aprire il Blueprint BP_INVISIBILE e navigare nell'Event Graph.
    - Dall'evento Event Tick, si ottiene la posizione attuale del personaggio con il nodo Get Actor Location.
    - Si usa un nodo Break Vector per isolare la componente Z (l'altitudine).
2. Definizione della "Soglia di Morte":
    - La coordinata Z viene confrontata con un valore soglia predefinito (es. -2000.0) tramite un nodo < (Less).

- L'esito di questo confronto (Vero/Falso) viene passato a un Branch. La logica prosegue solo se il giocatore si trova al di sotto della soglia.

3. Notifica Unica al Game Mode:
    - Per evitare che la logica di respawn venga chiamata centinaia di volte, si inserisce un nodo DoOnce subito dopo il pin True del Branch.
    - Dal pin Completed del DoOnce, si ottiene un riferimento al Game Mode (Get Game Mode), lo si converte al nostro tipo specifico (Cast To GM_Base), e si chiama la funzione RespawnPlayer.
    - Alla funzione RespawnPlayer viene passato un riferimento al Player Controller (ottenuto tramite Get Player Controller), in modo che il Game Mode sappia chi far respawnare.

Implementazione della Gestione del Respawn (GM_Base)

Questa logica centralizzata gestisce il ciclo di vita del personaggio.

1. Creazione della Funzione RespawnPlayer:
    - Aprire il Blueprint GM_Base.
    - Creare una nuova Funzione chiamata RespawnPlayer con un Input di tipo Player Controller.
2. Sequenza di Azioni all'interno della Funzione:
    - Distruzione del Vecchio Personaggio: Dal riferimento PlayerController, si ottiene il Pawn controllato (Get Controlled Pawn) e lo si elimina con Destroy Actor.
    - Ricerca del Punto di Partenza: Si utilizza la funzione interna del Game Mode, Find Player Start, per localizzare un attore Player Start valido.
    - Ottenimento della Posizione di Spawn: Dal riferimento al Player Start, si ottiene la sua posizione e rotazione con Get Actor Transform.
    - Creazione del Nuovo Personaggio: Si usa Spawn Actor from Class, specificando BP_INVISIBILE come Class e la Transform del Player Start come Spawn Transform.

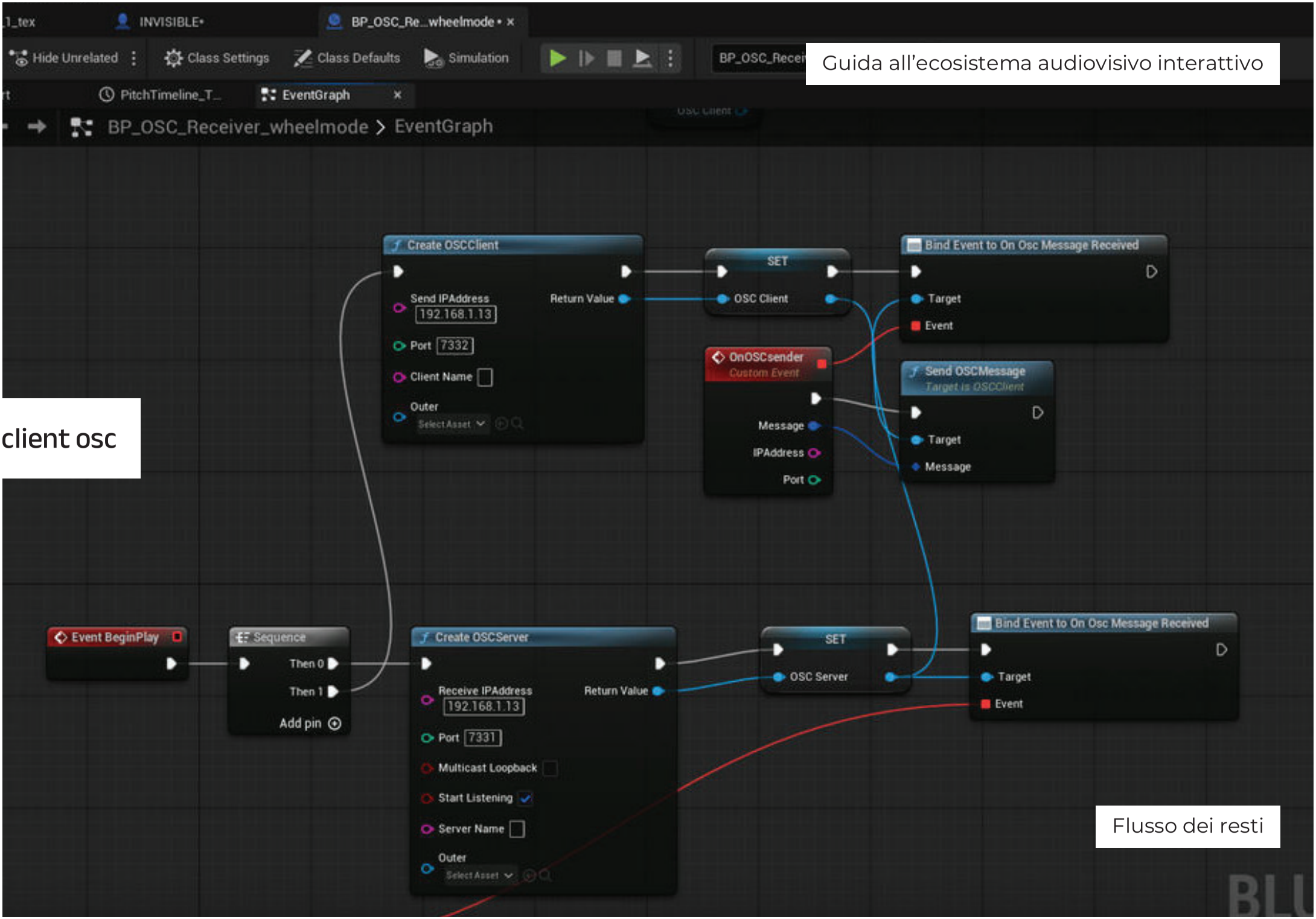
- Assegnazione del Controllo: Si usa il nodo Possess. Il Target è il PlayerController, e il In Pawn è il nuovo BP_INVISIBILE appena creato.

Questo ciclo completo assicura che, ogni volta che un giocatore cade attraverso un buco creato con la Visibility Mask, venga rimosso e ricreato in modo pulito in una posizione sicura.

Aggiunta del Client OSC per il Forwarding dei Dati ad Ableton Live

Per integrare il controllo del personaggio in Unreal Engine con la manipolazione di parametri in un software audio esterno come Ableton Live, è necessario implementare un sistema di "forwarding" o "duplicazione" del segnale. Questo capitolo descrive la procedura per configurare un client OSC all'interno dello stesso Blueprint che funge da server. L'obiettivo è fare in modo che BP_OSC_Receiver non solo riceva e interpreti i dati dal dispositivo mobile per controllare il gioco, ma li ritrasmetta simultaneamente a un'altra porta di rete, sulla quale Ableton Live sarà in ascolto.

1. Architettura del Sistema di Forwarding
2. Modifica nel Blueprint BP_OSC_Receiver
3. Integrazione della Logica di Invio
4. Configurazione in Ableton Live
5. Verifica del Sistema



client osc

Flusso dei resti

Architettura del Sistema di Forwarding

Il BP_OSC_Receiver_wheelmode assumerà un doppio ruolo:

1. Server OSC: Continuerà ad ascoltare su una porta specifica (es. 8000) per ricevere i dati inviati dall'applicazione oschookv2.

2. Client OSC: Si connetterà a un indirizzo IP e a una porta differente (es. 9000), inviando una copia esatta di ogni messaggio ricevuto.

Questo approccio centralizza tutta la logica di comunicazione in un unico attore, semplificando il debug e la gestione del flusso dei dati:

- Dispositivo Mobile (oschookv2) -> IP Locale:8000 (Server OSC in Unreal) -> Logica di Gioco

- Logica di Gioco -> IP Locale:9000 (Client OSC in Unreal) -> Ableton Live (Live Grabber)

Modifica nel Blueprint BP_OSC_Receiver

La logica di creazione del client e di invio dei messaggi viene aggiunta al Blueprint esistente.

1. Creazione e Configurazione del Client OSC

La configurazione del client deve avvenire all'avvio del gioco, insieme a quella del server. La procedura si svolge nell'Event Graph, all'interno della sequenza di esecuzione dell'evento Event BeginPlay.

1. Aprire il Blueprint: Navigare fino a BP_OSC_Receiver_wheelmode e aprirlo.

2. Aggiungere il Nodo Create OSCClient: Subito dopo il nodo Create OSCServer e la sua memorizzazione nella variabile OSC_Server, si aggiunge il nodo Create OSCClient.

3. Configurare i Parametri del Client:

    - Target IP Address: Inserire lo stesso indirizzo IP locale utilizzato per il server (es. "192.168.1.16"). Poiché Ableton Live è in esecuzione sulla stessa macchina, il client invierà i dati a sé stesso.

    - Port: Inserire un numero di porta diverso da quello del server. Se il server ascolta sulla porta 8000, una scelta comune per il client è 9000. È fondamentale che questa porta non sia già in uso da altre applicazioni.

4. Memorizzare il Riferimento al Client: Come per il server, è essenziale conservare un riferimento al client appena creato. Cliccare con il

tasto destro sul pin di output Return Value del nodo Create OSCClient e selezionare "Promote to variable". Nominare la nuova variabile OSC_Client. Questo permette di accedere al client in altri punti del codice per inviare messaggi.

Integrazione della Logica di Invio

Invece di creare un secondo evento di "bind" (che potrebbe creare ridondanza), si modifica l'evento custom già esistente che gestisce la ricezione dei messaggi. Questo evento (precedentemente nominato OnOSCReceived o simile) diventa il punto centrale da cui si diramano sia la logica di gioco sia quella di forwarding.

1. Individuare l'Evento di Ricezione: Trovare l'evento custom che viene attivato dal Bind Event to On OSC Message Received del server. Questo evento ha un pin di output Message che contiene i dati OSC in arrivo.

2. Organizzare il Flusso con un Nodo Sequence: Per mantenere il codice pulito, subito dopo l'inizio dell'evento di ricezione, si può inserire un nodo Sequence. Questo nodo esegue una serie di azioni in ordine.

    - Il pin Then 0 sarà collegato a tutta la logica preesistente che controlla il personaggio (il filtraggio degli indirizzi, la smussatura dei valori, i nodi Add Movement Input, ecc.).

    - Il pin Then 1 verrà utilizzato per la nuova logica di invio.

3. Aggiungere il Nodo Send OSCMessage:

    - Trascinare un filo di esecuzione dal pin Then 1 del nodo Sequence e creare un nodo Send OSCMessage.

    - Target: Trascinare una reference Get della variabile OSC_Client e collegarla all'input Target del nodo Send OSCMessage. Questo specifica che il messaggio deve essere inviato tramite il client che abbiamo configurato.

    - Message: Collegare direttamente il pin di output Message dell'evento di ricezione all'input Message del nodo Send OSCMessage. Questo passaggio è cruciale: garantisce che il messaggio inviato ad Ableton sia una copia esatta e non modificata di quello ricevuto dal telefono.

Configurazione in Ableton Live

Affinché la comunicazione abbia successo, è necessario configurare lo strumento Live Grabber (o qualsiasi altro dispositivo M4L compatibile con OSC) in Ableton Live perché ascolti sulla porta corretta.

1. Aprire le preferenze di Live Grabber in Ableton.
2. Se necessario impostare l'indirizzo IP sull'IP locale della macchina (lo stesso usato in Unreal).
3. Impostare la Porta di Ascolto (Listen Port) sul valore scelto per il client OSC in Unreal (es. 9000).

Verifica del Sistema

A questo punto, l'implementazione è completa. Per verificare il corretto funzionamento del sistema:

1. Avviare la simulazione in Unreal Engine premendo "Play".
2. Muovere il dispositivo mobile.
3. Risultato Atteso: Si dovrebbero osservare due comportamenti simultanei:
    - Il personaggio BP_INVISIBILE si muove nel livello di gioco, rispondendo ai comandi come configurato nei capitoli precedenti.
    - I parametri mappati in Ableton Live tramite Live Grabber si aggiornano in tempo reale, rispecchiando i dati dei sensori inviati dal dispositivo.

Questa architettura assicura un ponte di comunicazione robusto ed efficiente tra il mondo interattivo di Unreal Engine e l'ambiente di produzione audio di Ableton Live, aprendo la possibilità a complesse interazioni audiovisive controllate da un unico input.

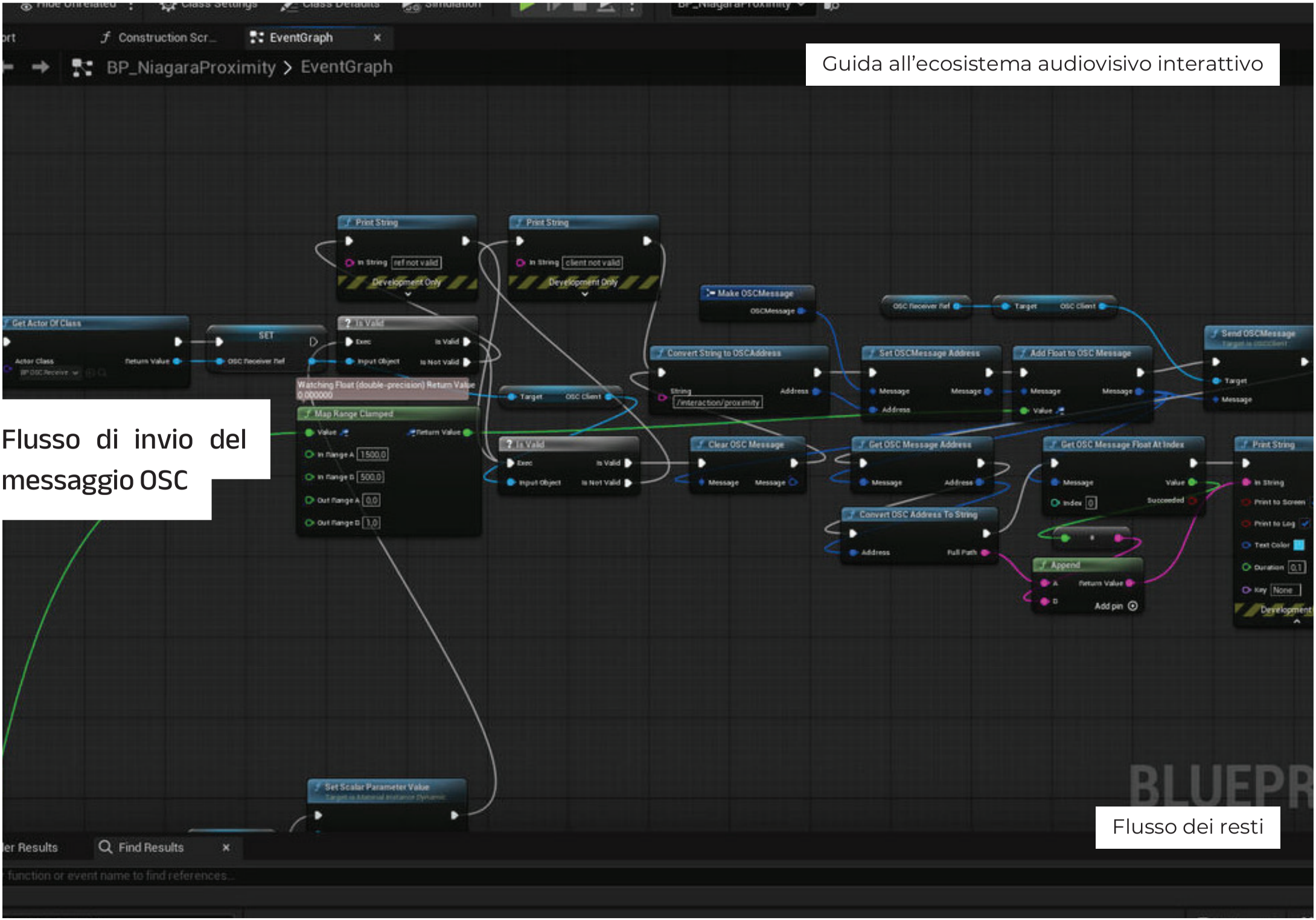
Questa registrazione è stata lasciata intenzionalmente granulare.

Interazione audio attraverso la prossimità

Questo capitolo descrive l'architettura per inviare un segnale di controllo OSC che parte da Unreal Engine e arriva ad Ableton Live. La metodologia si basa sull'utilizzo dell' Event Tick per un aggiornamento di dati float in tempo reale e implementa una pulizia del segnale per prevenire l'accumulazione di dati indesiderati nel messaggio OSC. Questo approccio garantisce che ogni messaggio inviato sia pulito e contenga solo i dati pertinenti a quel preciso istante, risolvendo i problemi di "valori bloccati" osservati in Ableton Live.

1. Preparazione dei Riferimenti in Event BeginPlay
2. Implementazione della Logica di Esecuzione in Event Tick
3. Debug e Verifica
4. Configurazione in Ableton Live e Risultato Finale

Guida all’ecosistema audiovisivo interattivo



Flusso di invio del messaggio OSC

Flusso dei resti

Preparazione dei Riferimenti in Event BeginPlay

L'efficienza del sistema si basa sull'ottenere un riferimento stabile al gestore OSC una sola volta, all'avvio del gioco.

1. Variabile di Riferimento: Nel Blueprint BP_InteractiveObject, è necessaria un'unica variabile, OSCReceiver_Ref, di tipo BP_OSC_Receiver (Object Reference).

2. Popolamento del Riferimento: Nell'evento Event BeginPlay, il flusso di esecuzione parte e popola la variabile OSCReceiver_Ref tramite un nodo Get Actor Of Class, che ha come target il nostro Blueprint BP_OSC_Receiver.

Implementazione della Logica di Esecuzione in Event Tick

L'intera logica di calcolo, pulizia, costruzione e invio del messaggio è contenuta in un'unica catena di esecuzione che parte dalla logica di prossimità già esistente nell' Event Tick.

1. Calcolo del Valore di Prossimità: Il flusso parte dal valore Distance già calcolato nel Blueprint. Questo valore viene normalizzato in un range da 0.0 a 1.0 tramite un nodo Map Range Clamped.

2. Validazione Robusta dei Riferimenti:

    - Il flusso di esecuzione prosegue verso un primo nodo IsValid. Come Input Object, si collega la variabile OSCReceiver_Ref. Un Print String sul pin Is Not Valid può aiutare a diagnosticare problemi di riferimento.

    - Dal pin di esecuzione Is Valid, si prosegue verso un secondo nodo IsValid. Come Input Object, si collega un nodo Get OSC_Client, il cui Target è la variabile OSCReceiver_Ref. Anche qui, un Print String su Is Not Valid è utile per il debug.

3. Costruzione del Messaggio con Ciclo di Pulizia (Logica Chiave):

    - Pulizia del Messaggio Precedente: Dal pin Is Valid dell'ultimo controllo, il primo passo è creare un nodo Clear OSCMessage. Questo nodo è fondamentale per "azzerare" il contenuto del messaggio prima di ripopolarlo.

    - Creazione e Popolamento: Si procede con la catena di costruzione:

        1. Un nodo Convert String to OSCAddress, inserendo l'indirizzo desiderato (es. /interaction/proximity).

        2. Si crea un Make OSCMessage.

3. Si aggiunge un Set OSCMessage Address, il cui input Message proviene dal Make OSCMessage e il cui input Address proviene dal Convert String to OSCAddress.

4. Un nodo Add Float To OSCMessage. Il suo input Message deve provenire dall'output Message del nodo precedente (Set OSCMessage Address). Il suo input Value riceve il valore normalizzato dal Map Range Clamped.

- Invio del Messaggio: Il messaggio finale, completo di indirizzo e valore, che esce dal nodo Add Float, viene passato all'input Message di un nodo Send OSCMessage. Il Target di questo nodo è il riferimento a Get OSC_Client.

4. Chiusura del Feedback Loop: Per completare il ciclo di pulizia, l'output Message del nodo Add Float To OSCMessage (il messaggio appena costruito e inviato) viene collegato all'input Target del nodo Clear OSCMessage dell'inizio della catena. Questo assicura che il messaggio generato in questo frame sia quello che verrà svuotato all'inizio del frame successivo.

Debug e Verifica

Per verificare che il messaggio venga costruito correttamente prima dell'invio, si implementa una catena di debug completa.

- L'output Message del nodo Add Float è il punto di partenza per l'ispezione.

- Da questo messaggio, si estraggono l'indirizzo e il valore:

    1. Si usano i nodi Get OSCMessage Address e Convert OSC Address to String.

    2. Si usa il nodo Get OSCMessage Float at Index.

- I due risultati (la stringa dell'indirizzo e il valore float) vengono combinati tramite un nodo Append e visualizzati con un Print String. Se nel log vediamo l'indirizzo corretto e i valori che cambiano dinamicamente con il movimento del giocatore, la meccanica funziona correttamente.

Configurazione in Ableton Live e Risultato Finale

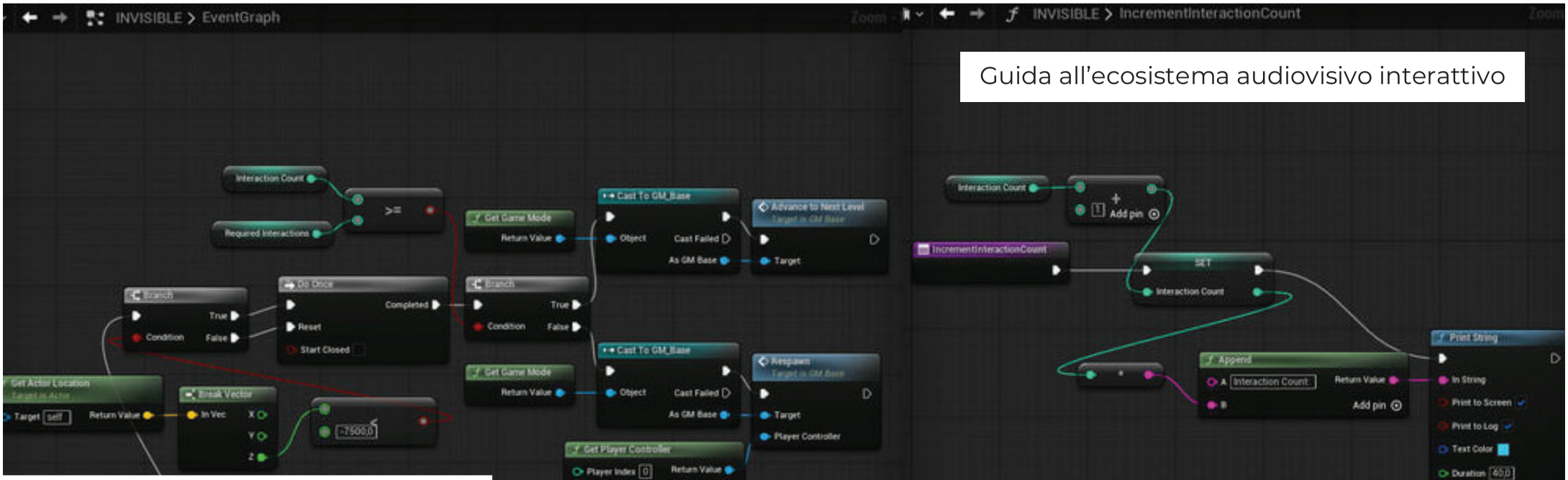
Il flusso di dati OSC, ora pulito e corretto, verrà ricevuto da Live Grabber in Ableton Live. Da lì, tramite Param Grabber, è possibile mappare il segnale a qualunque parametro di un dispositivo, di un effetto o di una Macro. L'interazione risulterà fluida e il parametro in Ableton si aggiornerà in tempo reale, riflettendo la prossimità del giocatore nel mondo di Unreal Engine.

Implementazione dell'Avanzamento di Livello Condizionale tramite caduta

Questo capitolo descrive come trasformare la meccanica di respawn in un sistema di progressione condizionale. La caduta del giocatore attraverso un buco nel terreno diventa il trigger per avanzare al livello successivo, ma solo dopo che è stata soddisfatta una condizione di interazione con l'ambiente. Questo sistema lega la meccanica di esplorazione alla progressione e si interfaccia con Ableton Live per orchestrare un cambio di soundscape dinamico.

1. Implementazione del Contatore di Interazioni
2. Modifica della Logica di Caduta in BP_INVISIBILE
3. Gestione dell'Avanzamento nel Game Mode
4. Configurazione Finale

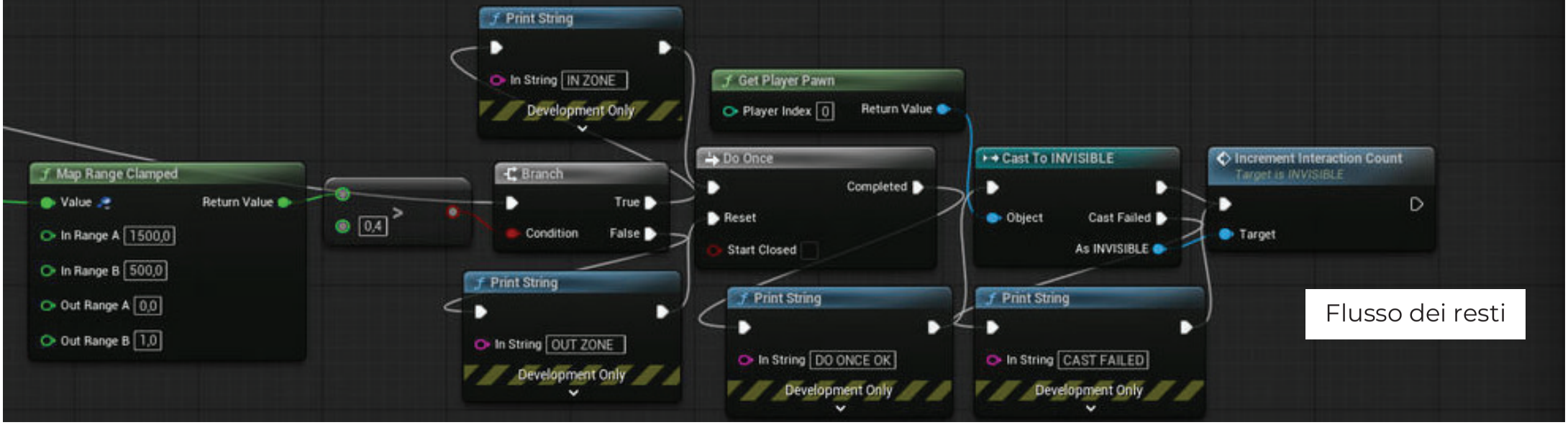
Guida all’ecosistema audiovisivo interattivo



funzione condizionale per prossimità e gestione caduta e avanzamento

BLUEPRINT

BLUEPRINT



Flusso dei resti

L'architettura logica è divisa tra BP_INVISIBILE (che rileva la condizione), BP_InteractiveObject (che registra l'interazione) e GM_Base con le sue classi figlio (che gestiscono la conseguenza: respawn o cambio di livello).

Implementazione del Contatore di Interazioni

Il gioco deve memorizzare il numero di interazioni del giocatore. Questo stato viene salvato nel personaggio giocabile.

1. Creare le Variabili in BP_INVISIBILE:
    - Aprire il Blueprint BP_INVISIBILE.
    - Creare due variabili di tipo Integer:
        1. InteractionCount: Memorizza il numero di interazioni. Valore di default: 0.
        2. RequiredInteractions: La soglia da raggiungere. Valore di default: 3.
2. Creare la Funzione di Incremento in BP_INVISIBILE:
    - Creare una nuova funzione e nominarla IncrementInteractionCount.
    - All'interno, implementare la logica Get InteractionCount -> + 1 -> Set InteractionCount.
3. Implementare il Rilevamento in BP_InteractiveObject:
    - Aprire l'attore interattivo. Nell' Event Tick, dopo il calcolo del valore di prossimità (0-1), si implementa la logica per chiamare la funzione di conteggio. Per pulizia, si consiglia di usare un nodo Sequence dopo l' Event Tick per separare questa logica da quella dell'invio OSC per la prossimità.
    - Logica di Conteggio:
        1. Si usa un nodo > (Greater) per verificare se il valore di prossimità ha superato una soglia (es. 0.5).
        2. L'output viene collegato a un Branch.
        3. Il pin True del Branch si collega a un DoOnce. Dal suo pin Completed, si ottiene un riferimento al giocatore (Get Player Pawn), si esegue un Cast To BP_INVISIBILE e si chiama la funzione IncrementInteractionCount.

4. Il pin False del Branch (che si attiva quando il giocatore si allontana) deve essere collegato al pin Reset del DoOnce. Questo riarma il sistema, permettendo di contare interazioni multiple.

Modifica della Logica di Caduta in BP_INVISIBILE

La logica di caduta esistente viene modificata per includere il controllo della condizione.

1. Implementare la Decisione Condizionale:
    - Nell' Event Tick di BP_INVISIBILE, dopo il DoOnce che rileva la caduta, si inserisce un Branch.
    - La condizione del Branch è il risultato del confronto Get InteractionCount >= Get RequiredInteractions.
    - Pin False: Se la condizione non è soddisfatta, si esegue la logica di respawn esistente, chiamando la funzione RespawnPlayer sul Game Mode (Get Game Mode -> Cast To GM_Base -> RespawnPlayer).
    - Pin True: Se la condizione è soddisfatta, si chiama una nuova funzione sul Game Mode, AdvanceToNextLevel.

Gestione dell'Avanzamento nel Game Mode

La logica per cambiare livello e suono risiede nel sistema di Game Mode gerarchico.

1. Preparazione di GM_Base (Classe Genitore):
    - Aprire GM_Base. Assicurarsi che contenga le seguenti variabili:
        - NextLevelName (Tipo: Name)
        - NextLevelIndexOSC (Tipo: Float)
    - Creare la funzione AdvanceToNextLevel.
2. Implementazione della Funzione AdvanceToNextLevel in GM_Base:
    - Caricare il Nuovo Livello: All'inizio della funzione per creare due comandi paralleli si crea il nodo Sequence, nella prima uscita si chiama Open Level (by Name), collegando la variabile NextLevelName.
    - Ottenere Riferimenti e Validare: Nella seconda uscita del Sequence, si usa Get Actor Of Class per trovare

BP_OSC_Receiver e si eseguono i controlli IsValid sia sul riferimento che sul suo componente OSC_Client per prevenire errori.

- Calcolo del Valore OSC Normalizzato: Per mappare correttamente l'indice del livello al Chain Selector di Ableton (che ha un range 0-127), il valore da inviare viene calcolato con la seguente formula: Valore da Inviare = Numero del Livello Desiderato / 127.0.
  - Nel Blueprint, questo si traduce in: Get NextLevelIndexOSC -> / (Divide) -> 127.0. È fondamentale usare un Float (127.0) per garantire una divisione corretta.
- Inviare il Messaggio OSC: Si usa la catena di costruzione del messaggio (Clear, Make, etc.) per inviare un messaggio all'indirizzo /game/loadLevel, usando il valore normalizzato appena calcolato come payload.

3. Configurazione delle Classi Figlio del Game Mode:
   - Per ogni livello, si crea una classe figlio di GM_Base (es. GM_Level_01, GM_Level_02).
   - In ogni classe figlio, si aprono i Class Defaults e si impostano i valori delle variabili ereditate:
     - In GM_Level_01: NextLevelName = L_Level_02, NextLevelIndexOSC = 2.0.
     - In GM_Level_02: NextLevelName = L_Level_03, NextLevelIndexOSC = 3.0.

Configurazione Finale

1. Assegnare i Game Mode: In ogni livello (L_Level_01, L_Level_02, etc.), aprire i World Settings e impostare il Game Mode Override sulla classe figlio corrispondente (GM_Level_01, GM_Level_02, etc.).
2. Configurare Ableton Live: Preparare un Instrument Rack con più catene. Mappare il parametro Chain Selector a un Param Grabber che riceve i dati dall'indirizzo /game/loadLevel. Live Grabber interpreterà correttamente il valore normalizzato (0.0-1.0) e lo scalerà sul range 0-127 del Chain Selector.

Con questa configurazione, il ciclo di gioco è completo: l'interazione del giocatore sblocca la progressione, e la caduta diventa un portale che trasforma sia lo spazio visivo che il paesaggio sonoro in un'esperienza audiovisiva coesa.

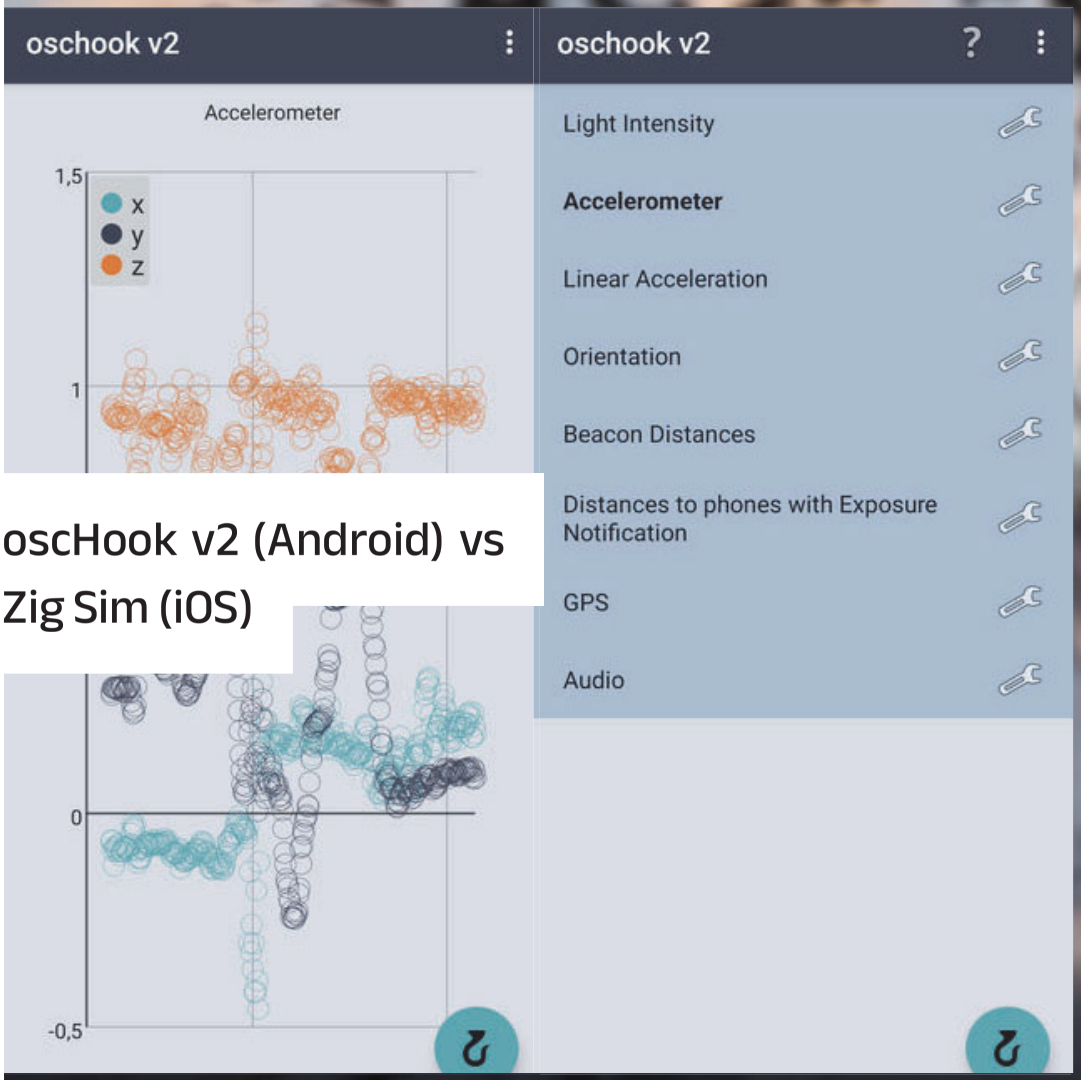
Questa rovina è stata lasciata intenzionalmente virtuale.

EXTRA - Implementazione di un Ricevitore OSC Universale con Inoltro Dati Unificato

Questo capitolo descrive come trasformare il BP_OSC_Receiver in un "hub di traduzione" centrale, capace di ricevere e interpretare messaggi da diverse applicazioni OSC (come OSChook e Zig Sim), standardizzare i dati in un formato interno, controllare il personaggio giocabile e inoltrare (forward) un flusso di dati pulito e coerente ad Ableton Live.

1. Architettura e Preparazione degli Eventi Standard
2. Implementazione dei Gestori (Handler) di Ricezione
3. Logica di Traduzione dei Dati
4. Implementazione dell'Inoltro (Forwarding) Unificato
5. Configurazione in Ableton Live e Risultato Finale

Guida all’ecosistema audiovisivo interattivo



oscHook v2 (Android) vs Zig Sim (iOS)



Flusso dei resti

L'architettura si basa sul principio di disaccoppiamento: i gestori (handler) specifici per ogni app traducono i messaggi grezzi in un formato interno comune. Tutta la logica di gioco e di interazione audio risponderà solo a questo formato, rendendo il sistema modulare, scalabile e agnostico rispetto alla fonte dei dati.

Architettura e Preparazione degli Eventi Standard

Invece di avere un unico evento di ricezione, creeremo dei "punti di ingresso" specifici per ogni formato di messaggio, e dei "punti di uscita" standardizzati per le azioni di gioco.

1. Eventi Interni Standardizzati: Nel BP_OSC_Receiver, creiamo due nuovi eventi custom che rappresentano le azioni astratte del giocatore. Questi eventi saranno il cuore del nostro sistema unificato.
    - Process_Forward_Input: Con un pin di input di tipo Float chiamato Value.
    - Process_Turn_Input: Con un pin di input di tipo Float chiamato Value.
2. Spostamento della Logica di Gioco: Tutta la logica che precedentemente gestiva il movimento (impostare TargetMoveForwardValue e TargetTurnValue) viene ora collegata a questi due eventi. Process_Forward_Input imposterà la variabile di movimento, e Process_Turn_Input imposterà quella di sterzata.

Implementazione dei Gestori (Handler) di Ricezione

Nell'evento Event BeginPlay di BP_OSC_Receiver, usiamo il nodo Bind Event to on OSCAddress Pattern Matches Path per creare dei "listener" dedicati per ogni app.

1. Handler per OSChook (Android):
    - Si crea un nodo Bind Event... per l'indirizzo /accelerometer/x e lo si lega a un nuovo evento custom chiamato OnOSChook_Forward.
    - Si crea un secondo Bind Event... per l'indirizzo /accelerometer/y e lo si lega a un evento custom OnOSChook_Turn.
2. Handler per Zig Sim (iOS):
    - Si crea un terzo Bind Event... per l'indirizzo /zigsim/1/gravity (o l'indirizzo specifico del dispositivo). Lo si lega a un evento custom OnZigSim_Gravity.

Logica di Traduzione dei Dati

Ora implementiamo la logica di "traduzione" all'interno di ogni evento handler.

1. Logica per OnOSChook_Forward e OnOSChook_Turn:
    - Nell'evento OnOSChook_Forward, si usa Get OSC Message Float at Index (con indice 0) per estrarre il valore. Questo valore viene poi usato per chiamare il nostro evento standard: Process_Forward_Input.
    - Si ripete lo stesso per OnOSChook_Turn, che chiamerà Process_Turn_Input.
2. Logica per OnZigSim_Gravity (Separare i Float):
    - Questo handler riceve un singolo messaggio contenente tre float. Dal pin Message, si usano tre nodi Get OSC Message Float at Index per estrarre i valori X (Index 0), Y (Index 1) e Z (Index 2).
    - Il valore X (Index 0) viene usato per chiamare Process_Forward_Input.
    - Il valore Y (Index 1) viene usato per chiamare Process_Turn_Input.

Implementazione dell'Inoltro (Forwarding) Unificato

Invece di inoltrare la copia del messaggio grezzo, integriamo l'invio ad Ableton direttamente nei nostri eventi standard. In questo modo, Ableton riceverà sempre dati con indirizzi coerenti.

1. Sganciare la Vecchia Logica: Assicurarsi che nessun nodo Send OSCMessage verso Ableton sia più attivato da eventi di ricezione generici.
2. Modificare l'Evento Process_Forward_Input:
    - Subito dopo il nodo che imposta la variabile TargetMoveForwardValue, si aggiunge la logica di invio OSC.
    - Si usano i controlli IsValid sul riferimento al OSC_Client.
    - Si costruisce un nuovo messaggio (Clear, Make, etc.) con un indirizzo standardizzato, ad esempio /control/forward.
    - Come Value per il nodo Add Float, si usa direttamente il pin di input Value dell'evento Process_Forward_Input.

    - Si invia il messaggio tramite il OSC_Client.

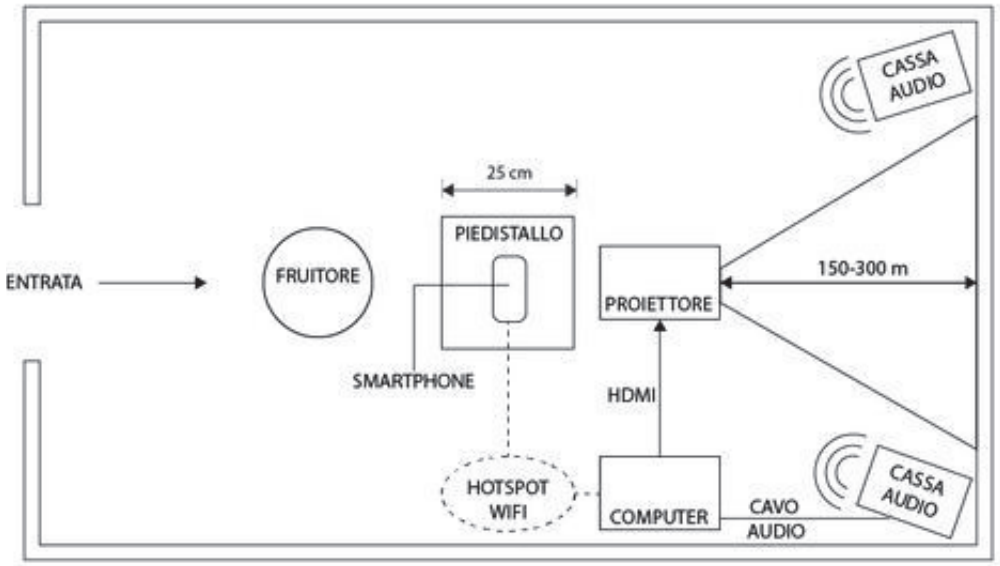
3. Modificare l'Evento Process_Turn_Input:
    - Si ripete esattamente la stessa procedura, ma si usa un indirizzo diverso, ad esempio /control/turn, e si prende il valore Value dall'input di questo evento.

Configurazione in Ableton Live e Risultato Finale

La configurazione in Ableton Live diventa ora semplice e a prova di futuro.

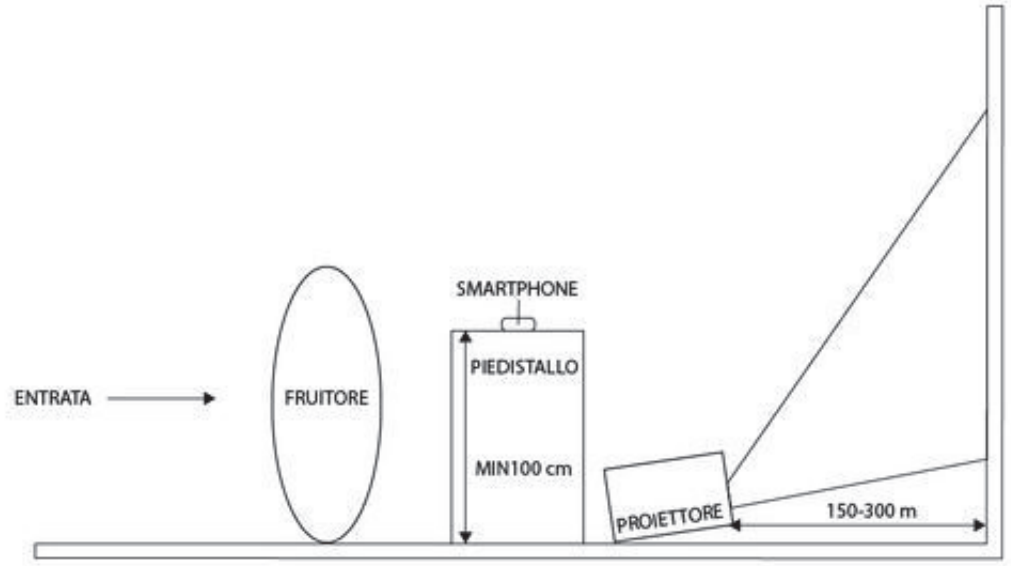
- Configurazione: In Live Grabber, si devono ascoltare solo i due indirizzi standardizzati: /control/forward e /control/turn.
- Risultato: Indipendentemente dal fatto che un utente stia usando un iPhone con Zig Sim o un Android con OSChook, il personaggio nel gioco si muoverà correttamente e Ableton Live riceverà lo stesso, identico flusso di dati OSC. Questo disaccoppiamento totale rende il sistema robusto, facile da manutenere e pronto per supportare nuove app in futuro semplicemente aggiungendo un nuovo handler di traduzione.

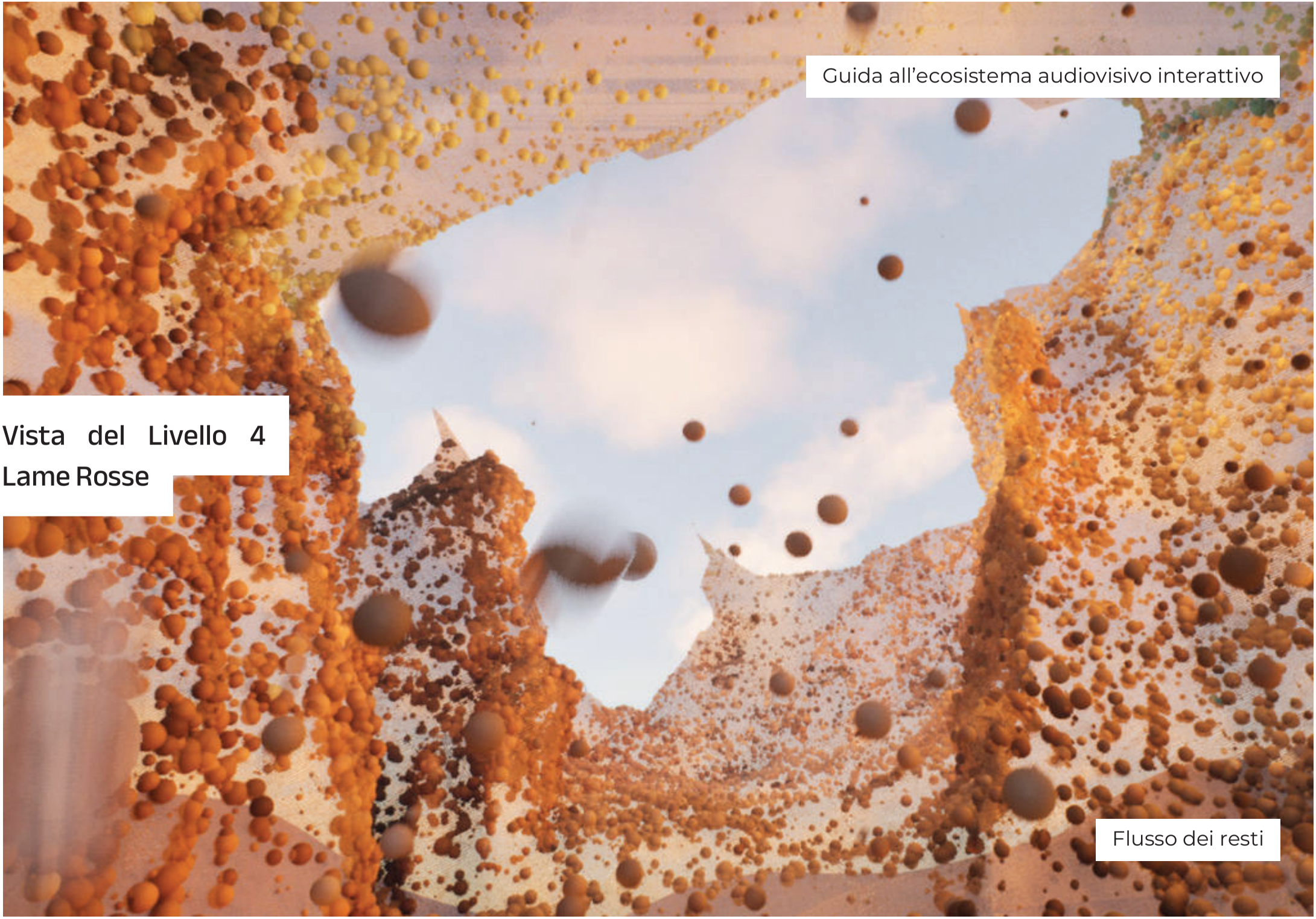
Vista dall’alto



Schemi di allestimento per una stanza piccola

Vista laterale




Guida all’ecosistema audiovisivo interattivo
Vista del Livello 4
Lame Rosse
Flusso dei resti

TESI SCRITTA



ESTRATTO VIDEO



Per ulteriori informazioni sul
progetto, collaborazioni o richieste.

Giacomo Cappella
Email: gcappe@gmail.com
Social: @ggggcap

ACCADEMIA
DI BELLE ARTI
MACERATA

Flusso dei resti
Guida all'ecosistema audiovisivo interattivo

ACCADEMIA
DI BELLE ARTI
MACERATA

Diplomando: Giacomo Cappella

Diploma Accademico in
Teoria e tecniche della comunicazione
visiva multimediale

Relatore: Federico Cingolani
Correlatore: Francesco Mariano

© 2025