Politecnico di Torino

POLITECNICO DI TORINO

Corso di Laurea Magistrale
In Ingegneria Informatica

Tesi di Laurea Magistrale

MetaHumans for Unreal (and other platforms)

Relatori
prof. Bottino Andrea
dott. Strada Francesco
firma dei relatori
.....
.....

Candidato
Esaulova Alena
firma del candidato
.....

Anno Academico 2021 - 2022

Sommario

Prefazione

Il mondo tecnologico si sviluppa continuamente. Negli ultimi anni l'umanità ha cercato di ricreare il nostro mondo in un Metaverso utilizzando la modellazione tridimensionale e strumenti di realtà virtuale. Il problema principale è sempre stato come raggiungere un livello alto di fotorealismo per far credere alle persone che ciò che vedono è "vero"? Questo problema può essere risolto in modo relativamente facile quando parliamo degli oggetti statici, basterebbe seguire quattro punti fondamentali: la modellazione precisa, le texture dettagliate, i materiali progettati tenendo conto delle proprietà fisiche dei loro prototipi nel mondo reale, e la scelta giusta di una strategia di illuminazione ambientale. Le stesse regole possono essere applicate anche allo sviluppo dei modelli 3D statici che rappresentano gli esseri umani, ma servirebbe molto lavoro aggiuntivo per costruire un oggetto adatto alla creazione di animazione e utilizzo di sincronizzazione labbiale.

Modelli 3D fotorealistici di esseri umani potrebbero essere utilizzati in una vasta gamma di aree relative alla computer grafica, o nei settori che potrebbero utilizzare computer grafica per scopi educativi. Esempi più comuni sono l'uso di personaggi virtuali nei film, nei videogiochi, in spot pubblicitari e video musicali, e nel dominio dell'intrattenimento in generale. Le persone virtuali possono essere utilizzate nelle applicazioni formative nelle scuole o nelle università nei laboratori virtuali.

Lo scopo di questa tesi è esplorare possibili metodi e strumenti per la creazione di modelli tridimensionali di esseri umani, analizare punti forti e deboli di ogni metodo come, ad esempio, tempo necessario per la creazione di un modello 3D fotorealistico, disponibilità di elementi necessari per la creazione di animazione (uno scheletro, un sistema di miscelazione (Blend System)), la disponibilità e la qualità delle texture, e studiare la possibilità di utilizzare tutte le proprietà di questi modelli nei motori grafici Unity e Unreal Engine.

Inoltre, è stata realizzata una ricerca in forma di sondaggio, il quale scopo era esaminare la percezione di livello di realismo di modelli 3D creati con diversi metodi e strumenti, quali sono i criteri di fotorealismo scelti dalle persone intervistate, quali sono le preferenze di quelle persone per un metodo rispetto ad un altro e che cosa potrebbe essere migliorato in futuro.

Abstract

The technological world develops continuously. In recent years, humanity has tried to recreate our world in a Metaverse using three-dimensional modeling and virtual reality tools. The main problem has always been how to achieve a high level of photorealism to make people believe that what they see is "real"? This problem can be solved relatively easily when we talk about static objects, it would be enough to follow four fundamental points: the precise modeling, the detailed textures, the materials designed taking into account the physical properties of their prototypes in the real world, and the right choice of an ambient lighting strategy. The same rules can also be applied to the development of static 3D models representing humans, but it would take a lot of additional work to build an object suitable for creating animation and using lip sync.

Photorealistic 3D models of humans could be used in a wide range of areas related to computer graphics, or in areas that could use computer graphics for educational purposes. More common examples are the use of virtual characters in films, video games, commercials and music videos, and in the entertainment domain in general. Virtual people can be used in educational applications in schools or universities in virtual laboratories.

The purpose of this thesis is to explore possible methods and tools for the creation of three-dimensional models of human beings, to analyze the strengths and weaknesses of each method such as, for example, the time required for the creation of a photorealistic 3D model, the availability of elements necessary for the creation of animation (a skeleton, a Blend System), the availability and quality of textures, and studying the possibility of using all the properties of these models in the Unity and Unreal Engine graphics engines.

In addition, it was carried out research in the form of a survey, the aim of which was to examine the perception of the level of realism of 3D models created with different methods and tools, what are the photorealism criteria chosen by the interviewees, what are the preferences of those people for one method over another and what could be improved in the future.

1. Metahumans

1.1 Introduzione

Esistono diversi metodi per creare un modello 3D di essere umano, ogni metodo ha i suoi vantaggi e disvantaggi, ogni metodo può essere preferito o meno da diverse persone con diversi livelli di abilità in una determinata area. Ci sono numerose aree dove si utilizzano i modelli 3D creati con un metodo scelto, un progetto può richiedere proprietà specifiche per ogni modello, ad esempio un progetto artistico potrebbe avere bisogno di alto numero di dettagli precisi, non richiederebbe sviluppo di elementi che guidano l'animazione e potrebbe non avere stretti limiti temporali. Dal altro lato, un progetto di videogioco potrebbe avere i tempi limitati, e richiederebbe un modello pronto per essere animato e adatto a un piattaforma specifico con potenza di calcolo limitata.
In seguito saranno analizzati i metodi più popolari della creazione di un modello 3D di essere umano.

1.2. Sviluppo di un modello tridimensionale di essere umano

Creazione di un modello tridimensionale di alta qualità spesso richiede molto tempo e elevate capacità artistiche e tecniche. I modelli più complessi sono dotati anche dei scheletri che permettono di mettere il modello in un grande numero di pose diverse e creare animazioni.
La creazione di un modello di essere umano si inizia da una base-mesh, un modello con un numero ridotto di dettagli, ma che già assomiglia una persona. La base-mesh può essere creata una volta e riutilizzata per tutti modelli dettagliati.[1]

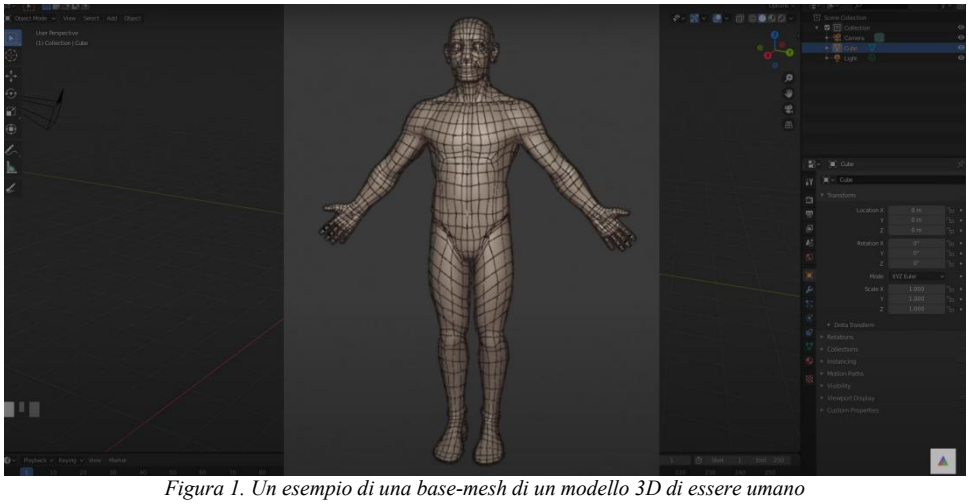

Figura 1. Un esempio di una base-mesh di un modello 3D di essere umano

Prendiamo come un esempio lavori di un artista iraniano Hadi Karimi. L'artista utilizza Zbrush per 3d sculpting, Substance Painter per creare le textures, Xgen core per capelli e Arnold renderer di Maya per rendering della immagine finale.

[1] Sociamix, YouTube, "Modeling a character BaseMesh in Blender (Tutorial)", 20/12/2019

Figura 2. Un modello 3D di Ludwig van Beethoven creato da Hadi Karimi con ZBrush

Il lavoro con un personaggio richiede circa 3-4 giorni per raggiungere la somiglianza con la persona scelta, 2-3 giorni per creare il colore della pelle, 3-4 giorni per disegnare tutti i dettagli della pelle, 1 settimana per la modellazione di capelli, e 1 settimana per vestiti. Quindi in totale è necessario lavorare circa 8-9 ore al giorno durante 3.5 settimane per creare un modello fotorealistico di una persona.[2]

I tempi cosi elevati per la creazione di un personaggio non sono efficienti nel ambiente di lavoro che richiede molti personaggi diversi in breve periodo di tempo. Inoltre, questi modelli non sono ottimizzati per sistemi con hardware diversi e sistemi operativi diversi, e non hanno lo scheletro che permette di creare animazioni.

1.3. Scansione 3d delle persone

I scanner 3D specializzati possono creare modelli 3D di diversi oggetti, animali o anche delle persone in tempi brevi, questi modelli hanno somiglianza elevata con l'oggetto o la persona scansionata.

[2] Hadi Karimi, Portfolio, “Ludwig van Beethoven”, n.d.

Lo scanner ottico raccolta le immagini utilizzando due diverse fotocamere per avere l'effetto stereoscopico. Sull'oggetto vengono apposti dei segni di riferimento sotto forma di adesivi e utilizzando questi e la posizione della testina dello scanner si scattano molti fotogrammi presi intorno all'oggetto e si uniscono ricreando una forma 3D. Lo scanner deve sempre acquisire almeno tre segni di riferimento per fotogramma per unirli. Potrebbero verificarsi problemi nel caso in cui la maggior parte degli indici di riferimento o la geometria si trovano in un sottosquadro o in ombra. È possibile regolare diversi parametri per avere una migliore acquisizione di superfici e marker di riferimento e filtrarli in caso di risultati incerti. Comunemente, più complessa è la geometria, più immagini deve acquisire lo scanner per garantire un calcolo completo e corretto. Altri problemi possono verificarsi nel caso in cui la superficie sia lucida. Ciò può causare il mancato acquisizione dei punti o superfici molto rumorose. Il software solitamente permette di impostare tempi di esposizione differenti per ogni fotogramma. Per ciascuna posizione nel primo tentativo di acquisizione vengono presi due o tre fotogrammi a diverso tempo di esposizione. Se la superficie è sottoesposta (troppo scura) o sovraesposta (linee zebrate) non è possibile acquisire buoni dati e creare la geometria finale con questo fotogramma, che viene escluso causando la geometria incompleta.[3]

Esistono tre tipi di scanner 3D:

- 3D body scanner
- 3D scanner portatile
- Fotocamera di cellulare or fotocamera DSLR

Il 3D body scanner è la tecnologia più comune e veloce per creare un modello 3D di una persona. Si tratta di dispositivi professionali utilizzati principalmente negli studi 3D-figure. La persona da scansionare si trova al centro del body scanner, esso è una cabina circolare in cui sono montate da 60 a 120 telecamere. Tutte queste telecamere sono attivate contemporaneamente e focalizzate sulla persona che sta in piedi. La persona viene fotografata da molte prospettive diverse, tutte le foto vengono elaborate in un modello 3D utilizzando un software di fotogrammetria. Uno dei maggiori vantaggi del 3D body scanner è la sua velocità, le immagine vengono create in una frazione di secondo, questo tipo di scanner 3D è spesso l'unico modo per catturare bambini e animali. Gli svantaggi sono le loro dimensioni e il prezzo, tra 40 e 100 mila euro.

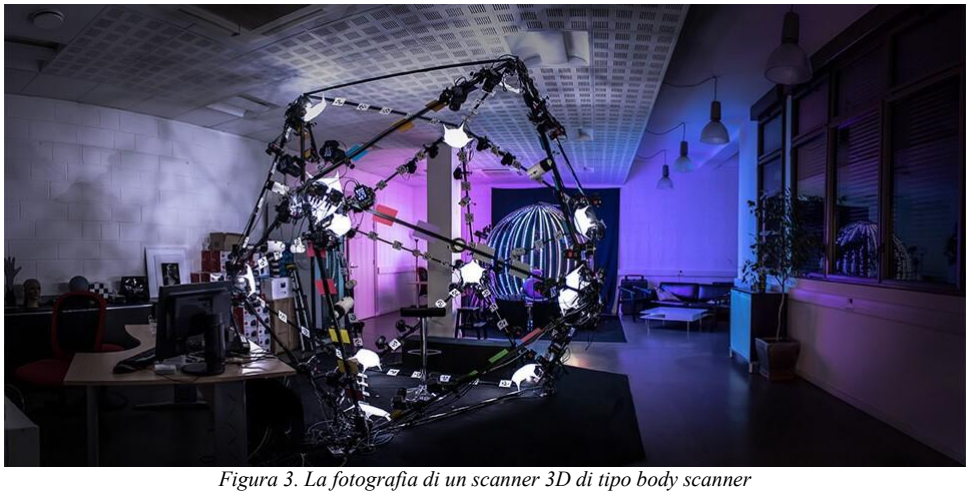

Figura 3. La fotografia di un scanner 3D di tipo body scanner

[3] Zaccardi Giuseppe Massimiliano (2020), "3D Scanner Inspection and Shape Correction of SLS parts through Reverse Engineering compensation", tesi di laurea, Politecnico di Torino, pp. 21-22

Il 3D scanner portatile funziona in modo diverso rispetto a un body scanner 3D. Con uno scanner 3D portatile l'oggetto deve rimanere fermo per circa cinque minuti. L'operatore dello scanner 3D cammina lentamente intorno all'oggetto e lo scansiona da diverse angolazioni. Lo scanner portatile può acquisire solo una sezione molto piccola (circa 20 cm) alla volta, per questo motivo la scansione 3D di oggetti di grandi dimensioni richiede molto più tempo. I vantaggi dello scanner portatile sono mobilità e costo. Si possono scansionare anche gli oggetti di grandi dimensioni, per esempio, un'auto. Uno scanner portatile costa circa 15 mila euro. Il più grande svantaggio dello scanner portatile è il tempo necessario per la scansione 3D, se il soggetto scansionato non rimane fermo, il software dello scanner non è in grado di creare un modello 3D pulito.[4]

Figura 4. La fotografia di un scanner 3D di tipo scanner portatile

La scansione 3D con una fotocamera di cellulare è simile a quella di 3D scanner portatile, il cellulare si sposta attorno all'oggetto, e una volta terminata la scansione, una applicazione raccoglie i dati e li trasforma in un modello 3D. Il vantaggio della scansione 3d tramite cellulare è il costo basso, spesso applicazioni che generano i modelli 3d sono gratuiti o hanno il costo basso. Lo svantaggio più grande di questo modo di scansione 3D è che il software del cellulare ha molti imperfezioni, e il modello generato spesso ha artefatti che devono essere cancellati a mano.

1.4. MetaHuman Creator

MetaHuman Creator è un'applicazione in streaming su cloud che semplifica la creazione di una persona digitale di alta qualità, completamente rigged e pronta per l'animazione in Unreal Engine o Maya. Il modello può essere scaricato tramite Quixel Bridge e importato direttamente nel Unreal Engine o Maya.
La creazione di un modello nel MetaHuman Creator si inizia da uno dei numerosi personaggi già precaricati nel database dell'applicazione. Ogni elemento del viso del personaggio scelto può essere modificato tramite numerosi asset.

[4] Swann Rack, Holocreators, “How to 3D-scan people or animals?”, 12/11/2020

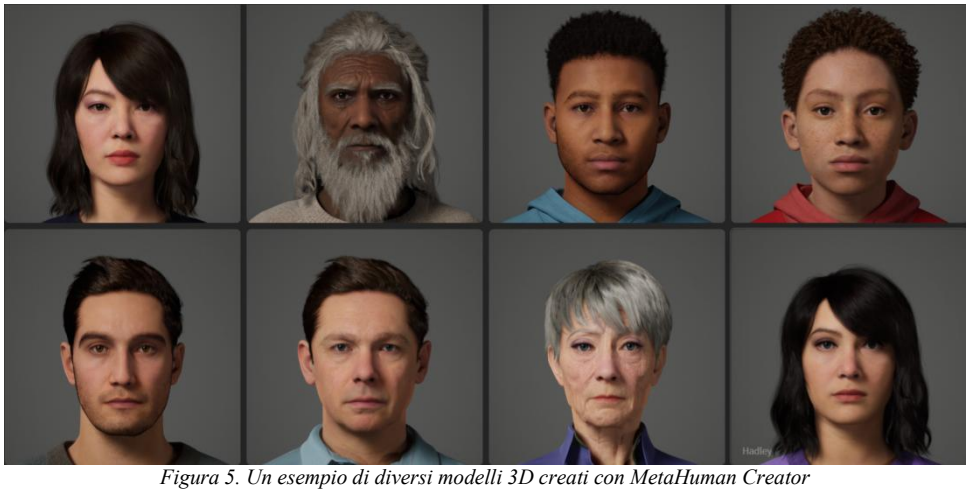

Figura 5. Un esempio di diversi modelli 3D creati con MetaHuman Creator

I dati utilizzati dal MetaHuman Creator derivano da scansioni del mondo reale, le modifiche dell'utente sono vincolate per rientrare nei limiti degli esempi nel suo database, che garantisce che ogni modello modificato rimanga fisicamente plausibile.

Rendering dei MetaHumans può essere eseguito in tempo reale su PC di fascia alta con schede grafiche RTX anche alla massima qualità. Gli asset sono dotati di otto LOD, alcuni dei quali utilizzano hair card, consentendo di ottenere prestazioni in tempo reale su diversi dispositivi, da Android a XSX e PS5.

1.4.1. Struttura dei file di progetto MetaHumans

Un progetto MetaHuman può essere scaricato tramite applicazione Quixel Bridge. Questa applicazione permette l'importazione dei file direttamente in un progetto aperto nel Unreal Engine. Una cartella creata ha un nome "MetaHuman" e contiene altre due cartelle: Common e una cartella che ha il nome di personaggio scaricato.

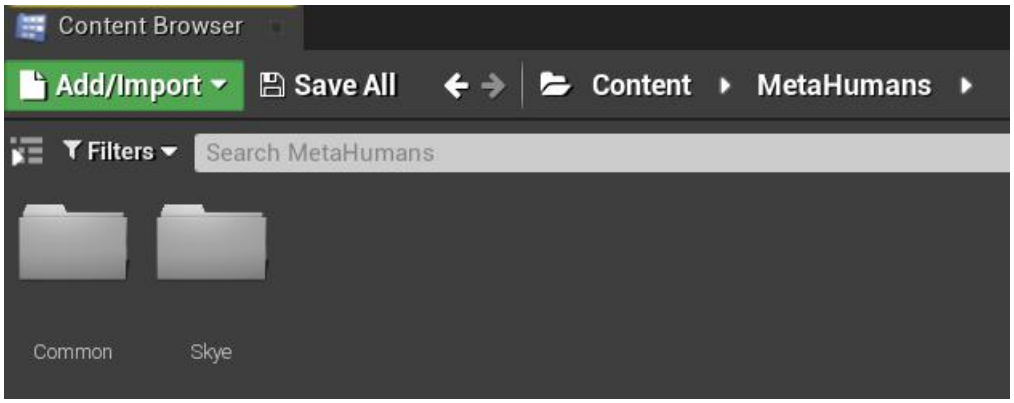

Figura 6. Un esempio delle cartelle create importate in Unreal Engine con MetaHuman

La cartella Common contiene altre 11 cartelle:

- Common: contiene i file che rappresentano i modeli 3D di diversi tipi di corpo maschile e femminile, questi modeli hanno già uno scheletro per creare animazioni, e hanno un certo numero di animazioni preimpostate. Altri due file permettono di impostare le pose e creare le animazioni del personaggio utilizzando il Control rig del corpo nel Sequencer di Unreal Engine. Inoltre la cartella Common contiene altre due cartelle: Mocap e Utilities. La cartella Mocap contiene un modello 3D della testa del personaggio con un certo numero di

animazioni preimpostate. La cartella Utilities contiene un file del simulatore di animazione che rappresenta un grafico ogni nodo (un'ancora) di quale è connesso a diverse parti del scheletro di MetaHuman. Questi nodi definiscono una posizione di un elemento rispetto a una cornice predefinita (Canvas Panel).
- Controls: contiene i file di controllo grafico dell'animazione facciale di MetaHuman.
- Face: contiene diversi componenti del modello 3d della testa di personaggio: Face Animation Blueprint, Archetipo del Viso, file di impostazioni di LOD, scheletro del viso, controllo grafico per animazioni; tre cartelle: ArtistDelights che contiene un file utilizzato per impostare accenti della pelle su un materiale, LookDev che contiene altre sei cartelle contenenti diverse mappe per impostare un materiale, Materials.
- Female: contiene i modelli 3d di diversi tipi di corpo filtrati per altezza (Medium, Short, Tall) e per tipo di corpo (NormalWeight, OverWeight e UnderWeight). Ogni cartella contiene altre due cartelle: Body e Poses. La cartella Body contiene l'Animation Blueprint, la mesh del corpo, lo scheletro, impostazioni del LOD. La cartella Poses contiene una serie di file con impostazioni delle pose predefinite.
- Fonts: contiene i file dello stile di carattere utilizzato per il controllo grafico di animazioni.
- Male: ha contenuti uguali alla cartella Female, ma rappresentanti un corpo maschile.
- MaleHair: contiene texture per impostare un materiale di capelli.
- Materials: contiene i materiali preimpostati di corpo, viso e vestiti.
- Models: contiene le mappe per la creazione di materiali del viso.
- SourceAssets: contiene maschere utilizzati per la creazione di materiali del viso.
- Textures: contiene una default texture.

La cartella che ha il nome di personaggio contiene altre 7 cartelle e un file Blueprint del personaggio:
- Body: contiene una texture del corpo del personaggio.
- Face: contiene le mappe delle textures del viso, una face mesh con tutti i materiali preimpostati, la cartella LookDev che contiene i materiali degli occhi, la cartella Materials che contiene Baked Materials.
- Female: contiene la mesh del corpo del personaggio scaricato.
- FemaleHair: contiene le mesh e i materiali di capelli femminili.
- MaleHair: contiene le mesh e i materiali di capelli maschili. Nel caso di personaggio dimostrato nel esempio, la mesh di capelli è contenuta nella cartella MaleHair, perché è stato utilizzato il stile di capelli maschile.
- Materials: contiene i materiali del corpo, ciglie, denti, capelli e vestiti.
- SourceAssets: contiene le mappe per impostare i materiali di corpo, ciglie, occhi, testa, denti.

Il file Blueprint del personaggio rappresenta il modello completo del personaggio MetaHuman pronto per essere animato e messo nella scena.

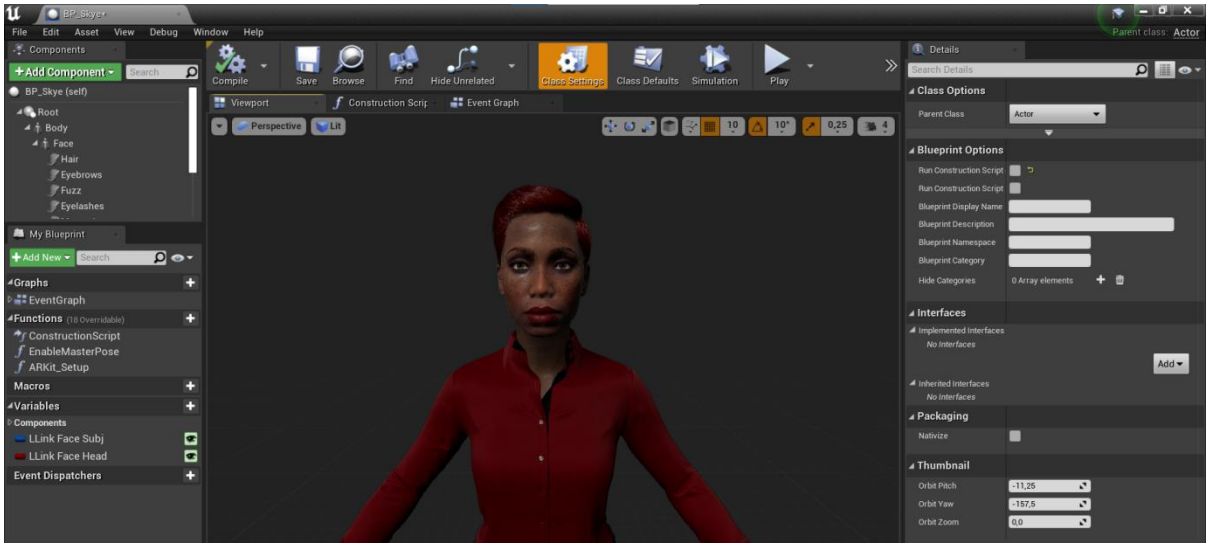

Figura 7. Blueprint di MetaHuman visualizzato nella finestra Viewport

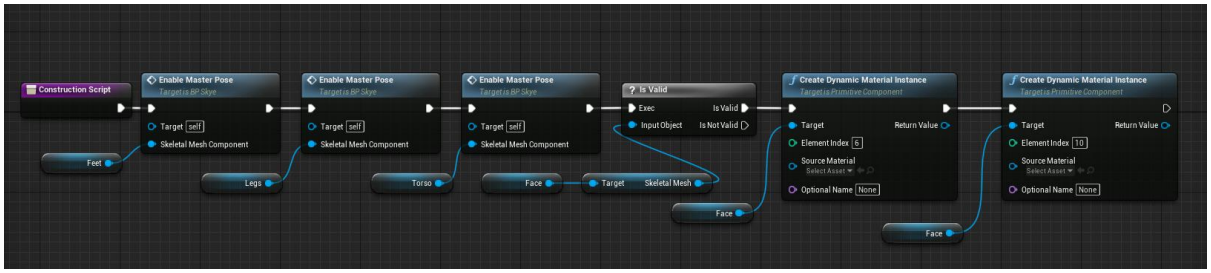

Figura 8. Il grafico contenete i nodi di Blueprint di MetaHuman (Blueprint Construction Script)

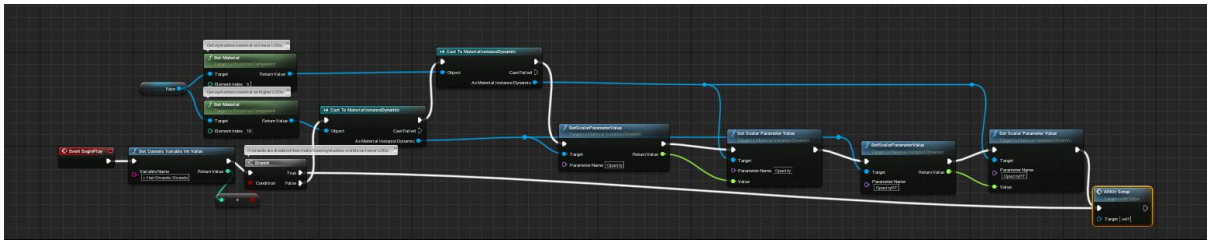

Figura 9. Blueprint Event Graph

1.4.2. MetaHumans Blueprint e suoi componenti

A differenza del personaggio predefinito di UE4 Mannequin, che è composto da un singolo Scheletro e Geometry Mesh con un paio di componenti di gioco specifiche impostati in Blueprint, i MetaHumans sono costruiti da molti componenti diversi.

Skeletal Mesh. I componenti "figlio" del componente Body (Corpo) guidano tutte le animazioni e il movimento di altri componenti del corpo. Il busto, le braccia e le gambe sono collegati a tutti i componenti della Skeletal Mesh rivestita, mentre il viso guida l'animazione della testa per gli espressioni facciali. Groom Assets (Capelli e Barba) ereditano dalla testa per abbinare con precisione il movimento. Il Viso (Face) include anche un TeethShell opzionale per aiutare a migliorare l'ombreggiatura sui denti.

Groom. Una varietà di componenti Groom che definiscono l'aspetto della testa di MetaHuman sono associati al componente Face Skeletal Mesh e includono i capelli, le sopracciglia, la peluria (vellus), le ciglia, i baffi e la barba.

LODSync. Questo componente gestisce il livello di dettaglio (LOD - dall'inglese "Level Of Detail") di MetaHuman in tutti i suoi componenti per garantire che cambino in sincronia tra loro. Questo è un componente importante che garantisce che MetaHumans abbia un aspetto coerente quando cambiano LOD perché alcuni componenti hanno meno LOD di altri. Ad esempio, il componente Face Skeletal Mesh utilizza 8 LODs rispetto ai 4 LODs del corpo.[5]

1.4.3. Control Rigs

I MetaHumans hanno il rig completo e sono pronti per l'uso in Unreal Engine, cioè include il body rig e face rig completo. Ogni MetaHuman Rig è composto da un massimo di cinque singoli Control Rig che lavorano insieme in un approccio a più livelli. Questo approccio consente flessibilità e un controllo più preciso sui MetaHumans e sulle loro animazioni.

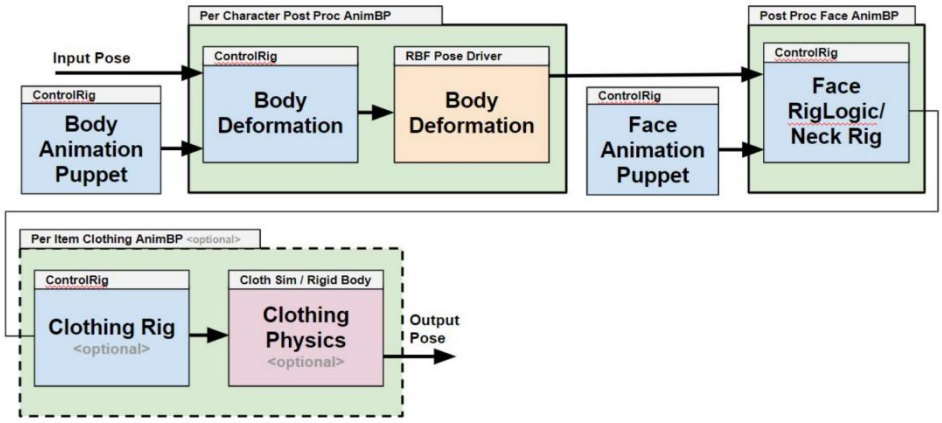

Figura 10. MetaHuman control rigs

MetaHumans Rigs Layer 1: Puppet Rigs

I Puppet Rigs rappresentano il primo strato dei MetaHuman. Questi rig sono i controlli dei movimenti del corpo e del viso utilizzati dal animatore.

MetaHumans Rigs Layer 2: Deformation Rigs

I Deformation Rig costruiscono il secondo strato dei MetaHumans. Qui è dove viene utilizzato un rig di controllo per guidare tutte le articolazioni torcenti e anatomiche. È seguito da dodici nodi di Pose Driver nell'Animation Blueprint, che guida più di cento pose correttive del corpo che aiutano a mantenere il volume e migliorare la deformazione complessiva del MetaHuman.
Il Viso (Face) utilizza il plug-in RigLogic e viene eseguito in un post-process consentendo a tutte le modifiche a cascata di eseguire le curve di espressione RigLogic da diverse interfacce di animazione o LiveLink Face, e deformare l'animazione finale della faccia.

MetaHumans Rig Layer 3: Clothing and Physics

[5] Unreal Engine Documentation, “MetaHumans”, n.d.

Lo strato finale è Clothing and Physics. Questo strato dipende in gran parte dall'abbigliamento utilizzato con MetaHuman, che potrebbe avere il proprio Control Rig.

1.4.4. Hair Grooms

Le funzionalità di Hair Rendering and Simulation di Unreal Engine guidano le simulazioni di capelli basate su ciocche. Componenti come Groom Asset Editor consentono a MetaHumans di gestire le chiocche di capelli e le rappresentazioni geometriche per livelli di dettaglio inferiori. Ogni MetaHuman è composto da un massimo di sei Groom Asset: capelli, baffi, barba, sopracciglia, ciglia e vellus. Ciascuno di questi utilizza un componente Groom nel progetto MetaHuman per descrivere la sua geometria dei capelli (fili (strands), carte o mesh) e le proprietà.
Ogni Groom ha una serie di LOD che vanno dalla qualità cinematografica (LOD 0) a un personaggio che sta lontano dal utente (LOD 7).

Figura 11. Livelli di dettagli di MetaHuman

Un LOD utilizza una particolare rappresentazione geometrica dei capelli, dalle ciocche per alta qualità, alle carte, alle meshes (chiamate anche elmo). Ciascun LOD è rappresentato nei seguenti modi:

- LOD 0-1 utilizza la geometria strand-based su piattaforme di fascia alta (Xbox Series X, PlayStation 5 e PC), ma supporta anche le rappresentazioni delle carte per altre piattaforme.
- LOD 2-4 usa rappresentaizoni basate su carte
- LOD 5-7 usa meshes semplificate per capelli.

Per i peli sottili, come sopracciglia o barbe sottili, i dati del groom vengono applicati direttamente tramite lo shader della testa come ulteriore livello di texture. Ciò consente alcuni risparmi sui costi preservando l'aspetto dei MetaHumans. Tutti i grooms sono attaccati alla testa con un meccanismo di skinning personalizzato, che consente loro di fissare ogni ciocca di capelli e carta per capelli in una posizione precisa sulla testa. Questo meccanismo attualmente richiede che la testa del personaggio abbia la propria opzione di cache della pelle abilitata su tutte le piattaforme (ad eccezione di Mobile, dove i peli sono direttamente attaccati alla deformazione ossea).
Il Groom maschile di MetaHumans si basa sul risolutore della fisica dei capelli per deformare i capelli in base al movimento del personaggio, consentendo ai capelli di scontrarsi con i collisori della testa.

1.4.5. Prestazioni, piattaforme e impostazioni del progetto MetaHumans

I MetaHumans sono sviluppati e progettati per supportare un'ampia gamma di hardware e piattaforme con qualità scalabile, da dispositivi cinematografici di fascia alta a dispositivi mobili di fascia alta. Ciò garantisce che lo stesso MetaHuman possa essere utilizzato su più piattaforme e soddisfare i requisiti di prestazioni per esse mantenendo un alto livello di fedeltà.

Il MetaHuman al LOD 0 ha circa 24000 vertici della testa, 669 blendshapes, 713 joints, 30500 vertici del corpo, 50000 ciocche di capelli con 30000 vertici in totale, e circa 10000 ciocche di peli sul viso con 15000 vertici in totale. Al LOD 1 i numeri scendono intorno a metà del LOD 0, ma la qualità rimane oggettivamente alta.

Figura 12. Visualizzazione di MetaHuman su diverse piattaforme

1.4.6. Miglioramenti futuri per MetaHumans

I MetaHumans sono il futuro della creazione di personaggi per i progetti Unreal Engine utilizzando MetaHuman Creator. Il seguente è un elenco non esaustivo dei miglioramenti apportati a Unreal Engine e, per estensione, a MetaHumans, secondo la documentazione ufficiale di Unreal Engine:

- Vulkan, uno standard industriale multipiattaforma che consente agli sviluppatori di indirizzare un'ampia gamma di dispositivi con la stessa API grafica, non è supportato da MetaHumans attualmente.
- La qualità di MetaHumans su piattaforme iOS, Android e Switch continuerà a migliorarsi. Ciò include le caratteristiche del materiale della pelle, i dettagli della geometria e l'attaccamento dei capelli.
- L'installazione di Pose Driver sul componente Body è costosa in termini di prestazioni. Ci sono diversi ottimizzazioni per questo sistema rilasciate in Unreal Engine 4.26.2 hotfix e 4.27.
- A causa della mancanza di rilegatura e ridimensionamento della simulazione con i LODs, i componenti Groom sono più costosi da usare.
- La simulazione dei capelli è disabilitata su Mac. Questo è un problema specifico della piattaforma che richiede ulteriori sviluppi.[6]

1.5. Daz3D

Daz 3D (o Daz Studio) è un'applicazione di progettazione 3D, essa non è un'applicazione tradizionale come Blender o Maya, Daz 3D è creata per progettazione di personaggi pronti per essere messi in una scena 3D statica o per animazione. L'interfaccia utente dell'applicazione è adatta anche ai principianti, è abbastanza intuitiva, e può essere estesa alla versione avanzata, che ha molti strumenti della modellazione più dettagliata.[7]
Un esempio di personaggio fotorealistico creato con Daz 3D è un cosiddetto "influencer virtuale" Shudu Gram, è stata creata nel 2017 da un fotografo Cameron-James Wilson. Migliaia di persone non avevano capito che non era una persona esistente, ma un personaggio generato sul computer. Questo modello 3D inferiore in fotorealismo ai personaggi creati con MetaHuman Creator. Per esempio, la sua pelle non ha un sottotono, mantiene la stessa tonalità piatta sotto luci diversi, come una figura di cera.

[6] Unreal Engine Documentation, “MetaHumans”, n.d.
[7] Erez Zukerman, PCWorld, “DAZ Studio”, 09/08/2012

Nonostante le sue imperfezioni, Shudu Gram ha acquisito notorietà ed è stata protagonista di alcune campagne pubblicitarie come Fenty Beauty e videoclip come “Frontline”(una canzone della performer etiope-americana Kelela), dove si possono notare imperfezioni nel lipsync e una strana andatura, che ricorda un robot.[8]

Figura 13. Modello 3D Shudu Gram creato da Cameron-James Wilson con Daz3D

1.6. MakeHuman

MakeHuman è uno strumento specializzato nella modellazione 3D dell'umanoide, lo strumento è stato sviluppato alla fine del 1999 con uno script Python per Blender e poi è stato riscritto in un'applicazione C/C++ autonoma. MakeHuman si usa per un'ampia varietà di scopi: dalla produzione di animazioni all'uso di cartoni animati per insegnare lezioni di psicologia sul linguaggio del corpo, alla ricostruzione di prove legali negli studi di criminologia. Tuttavia, l'idea principale è sempre stata quella di utilizzare MakeHuman in pipeline professionali: ottenere oggetto 3D umanoide realistico con textures e rigging, utilizzabile da subito in qualsiasi progetto grafico che deve rappresentare un essere umano.[9]

Lo sviluppo di metaumano deriva da uno studio tecnico e artistico dettagliato delle caratteristiche morfologiche del corpo umano. Tutti modelli creati con MakeHuman utilizzano la stessa base-mesh, con la stessa topologia, e con lo stesso significato morfologico delle facce.[10] MakeHuman è in grado di esportare personaggi con densità variabili di mesh e varie soluzioni di rigging.

[8] Lauren Michele Jackson, The New Yorker, “Shudu Gram is a white man’s digital projection of real-life black womanhood”, 04/05/2018

[9] Manuel Bastioni, Joel Palmius, Jonas Hauquier, 80.lv, “Interview with MakeHuman: The Future of 3D-character creation tools”, 18/02/2015

[10] M.Bastioni, S.Re, S.Misra. Proceedings of the 1st Bangalore Annual Compute Conference, Compute 2008, 2008. "Ideas and methods for modelling 3D human figures: the principal algorithms used by MakeHuman and their implementation in a new approach to parametric modeling".

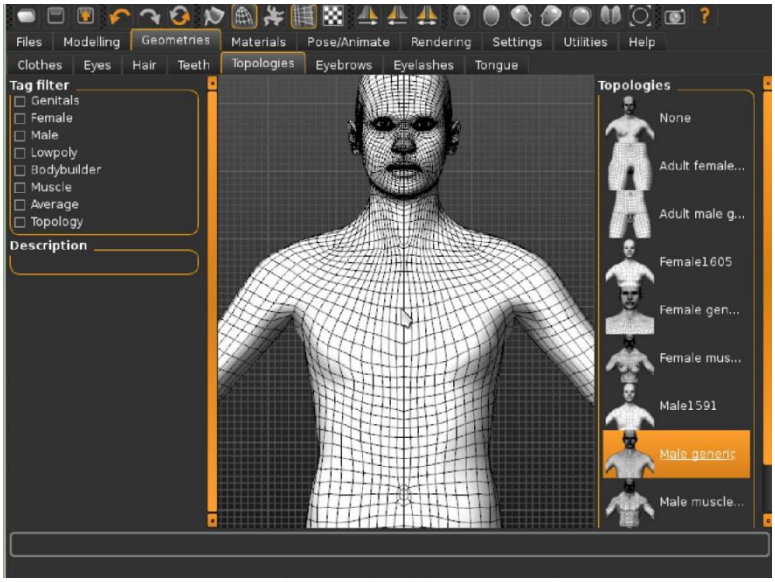

Figura 14. Interfaccia grafica del strumento MakeHuman

L'applicazione funziona attraverso la modellazione parametrica, ci sono un gran numero di cursori che possono essere trascinati per modificare tutti dettagli dell'aspetto del nostro personaggio. Dietro questa semplice interfaccia c'è una tecnologia complessa, basata su un grande database di forme umane (oltre 3000 morphings). Questi sono gestiti da un motore intelligente interno che li combina per adattarsi alle principali caratteristiche biometriche come età, percentuale di grasso, percentuale di muscoli, etc.

Durante lo sviluppo di MakeHuman particolare attenzione è stata dedicata alla creazione dei personaggi di base, detti base-mesh. La topologia della mesh è ottimizzata per i personaggi umani, e particolarmente adatta alle animazioni e allo sculpting con Zbrush o altro software (solo Quads, basso numero di poligoni).[11]

Un esempio di un modello fotorealistico creato con MakeHuman è un lavoro di un artista chiamato ecke101. L'artista ha utilizzato la base-mesh creata in MakeHuman, ha perfezionato il modello con Blender e modificato le textures e materiali:

- Ha alzato leggermente le sopracciglia e ha modificato la texture di sopracciglia per ammorbidire i peli
- Ha fatto la modifica simile anche per ammorbidire ciglia
- Ha creato un target personalizzato per rendere le palpebre più realistiche
- Ha realizzato una mappa di dispersione del sottotono di pelle
- Ha dipinto una mappa bump per i pori della pelle e le labbra
- Ha modificato lo shader dell'occhio in modo che cornea abbia un po' di lucentezza mescolata con trasparenza.[12]

[11] Manuel Bastioni, Joel Palmius, Jonas Hauquier, 80.lv, “Interview with MakeHuman: The Future of 3D-character creation tools”, 18/02/2015

[12] Ecke101, MakeHuman Community, “Trying to make a realistic face”, 02/08/2019

Figura 15. Un modello 3D creato con MakeHuman

2. Strumenti di sviluppo

2.1. Unity

Unity è un motore grafico multipiattaforma sviluppato da Unity Technologies, annunciato e rilasciato per la prima volta nel 2005 come motore grafico esclusivo per Mac OS X. Da allora il motore è stato esteso gradualmente per supportare una varietà di piattaforme desktop, dispositivi mobili, console per videogiochi e strumenti di realtà virtuale. È particolarmente popolare per lo sviluppo di giochi per dispositivi iOS e Android e utilizzato per giochi come Pokémon Go, Monument Valley, Call of Duty, etc. è considerato facile da usare per gli sviluppatori principianti ed è popolare per lo sviluppo di giochi indie (abbreviazione dell'inglese "independent" - un gioco sviluppato da una persona o da un piccolo gruppo di programmatori indipendenti).
L'editor Unity fornisce un ambiente "drag and drop" per la creazione di giochi. È possibile creare un gioco in Unity senza scrivere alcun codice, ma la maggior parte dei progetti richiede programmazione. Gli utenti di Unity possono programmare in C#, JavaScript o Boo, che usa una sintassi simile a Python. L'ambiente di sviluppo funziona su Mono, una versione open source di .NET Framework. Unity stesso è scritto in C++, perché codice come la fisica e l'animazione deve essere eseguito in modo super veloce.
Unity è un sistema di oggetti di gioco basato su componenti, ad ogni oggetto di gioco può essere allegato uno script, che deriva da una certa classe e comportamento del modello. Lo script viene eseguito nel gioco creato dopo essere trascinato su un oggetto di gioco.
Unity offre anche un "Asset Store" che ricorda gli app store su smartphone, solo che contiene componenti e codice utile al posto delle app reali.
David Helgason, uno dei fondatori di Unity, riconosce che il motore è molto indietro sulle console tradizionali, Unity era in ritardo di diversi anni per supportare PS3 e Xbox 360, la maggior parte degli sviluppatori aveva scelto il proprio software per lo sviluppo di giochi su quelle piattaforme. E mentre Unity ha annunciato il supporto per PS4 in collaborazione con Sony, non sarà pronta in tempo per creare giochi di lancio per la console, per questo motivo esistono migliaia di giochi per dispositivi mobili creati con Unity, ma l'elenco dei giochi per console conta solo tra le dozzine.[13]
In luglio 2020 Unity ha annunciato lo strumento di scripting visivo Bolt che consente agli sviluppatori di implementare la logica nei loro progetti senza dover sapere come programmare. Bolt ha grafici visivi basati su nodi che sia i programmatori che i non programmatori possono utilizzare per progettare la logica finale o per creare rapidamente prototipi. Bolt dispone anche di un'API che i programmatori possono utilizzare per attività più avanzate o per creare nodi personalizzati che possono essere utilizzati da altri membri del team.
La capacità di Unity di portare a termine le cose molto rapidamente è un altro punto di forza prezioso: consente un'iterazione molto veloce e può essere estremamente utile quando si fa brainstorming per un nuovo concetto di gioco. Atro vantaggio di Unity è che il gioco creato può essere rapidamente adattato alle piattaforme diverse.
Uno dei svantaggi delle vecchie versioni di Unity è che non erano adatte a grandi progetti. Per esempio, un gioco AAA (Tripla A - è una classificazione informale utilizzata per i videogiochi di medie o grandi dimensioni con budget particolarmente elevato; ad esempio giochi sviluppati da Electronic Arts o Ubisoft Monreal sono di

[13] Joe Brodkin, Dice, "How Unity3D Became a Game-Development Beast", 03/06/2013

categoria AAA) non poteva essere sviluppato con Unity fino alla seconda metà di 2020, invece adesso Unity è diventato uno dei motori grafici preferiti anche tra sviluppattori di videogiochi di grandi dimensioni.[14]

Dal 2017 Unity supporta Vulkan - un'API multipiattaforma a basso costo, si rivolge ad applicazioni grafiche 3D in tempo reale ad alte prestazioni, come videogiochi e media interattivi. In contrasto con le vecchie API OpenGL e Direct3D 11, Vulkan ha lo scopo di offrire prestazioni più elevate e un utilizzo più equilibrato di CPU e GPU. Il supporto di Vulkan aumenta la velocità riducendo al contempo il sovraccarico del driver e il carico di lavoro della CPU, ciò lascia la CPU libera di eseguire calcoli o rendering aggiuntivi e consente di risparmiare sulla durata della batteria per le piattaforme mobili.[15]

In dicembre 2020 Unity ha annunciato Unity Forma, un catalizzatore per la produzione di marketing digitale che consente ai professionisti del marketing di creare e pubblicare facilmente contenuti ed esperienze interattive da dati ingegneristici 3D. Utilizzando Unity Forma, i team di marketing possono ridurre significativamente tempi e costi di produzione e dedicare più tempo al processo creativo per ispirare gli acquirenti. I team di marketing e le loro agenzie creative partner possono mostrare facilmente prodotti con una qualità visiva realistica, in qualsiasi configurazione e in una varietà di formati, inclusi configuratori di prodotti 3D interattivi, immagini e video. I contenuti vengono automaticamente ottimizzati per la piattaforma prevista dal marketer, sia che desiderino trasmettere contenuti in streaming utilizzando la tecnologia cloud o pubblicarli direttamente su dispositivi mobili, web o, eventualmente, dispositivi di realtà aumentata e virtuale (AR e VR).

Caratteristiche principali di Unity Forma:

- Flussi di lavoro senza codice. I professionisti del marketing possono creare configuratori 3D in tempo reale e altri contenuti rapidamente utilizzando un'interfaccia su misura per le loro esigenze. Preparare esperienze interattive, visualizzare in anteprima in tempo reale e pubblicare su più destinazioni, il tutto senza scrivere una sola riga di codice.
- Time to market più rapido (o TTM, indica il periodo di tempo che intercorre tra l'ideazione di un prodotto e la sua effettiva commercializzazione). La preparazione automatizzata dei dati elimina il lavoro manuale dispendioso in termini di tempo per ottenere i dati di prodotto 3D pronti per il marketing. L'API di importazione di Unity Forma può anche connettersi ai dati di origine in modo da poter elaborare rapidamente le modifiche del modello e garantire la correttezza del prodotto.[16]
- Collaborazione senza interruzioni ed economicamente vantaggiosa. Si riducono i costi eliminando il lavoro duplicato.
- Editoria multicanale. Da un'unica scena principale si può pubblicare configuratori su più canali, inclusi dispositivi mobili e browser web tramite streaming cloud o WebGL. La qualità visiva e le prestazioni possono essere ottimizzate automaticamente, garantendo un'esperienza cliente soddisfacente.
- Esperienze di prodotto coinvolgenti [immersive]. Unity Forma consente agli acquirenti di esplorare in modo interattivo i prodotti con una qualità visiva realistica, aumenta il coinvolgimento e la conversione consentendo loro di trovare la loro configurazione preferita.[17]

In 2021 Unity ha presentato la versione 2020.3 che ha allargato il supporto per XR.

---

[14] Marie Dealessandri, gameindustry.biz, "What is the best game engine: is Unity right for you?",16/01/2020

[15] Jeff Grubb, VentureBeat, "Unity 5.6 launches with support for Vulkan graphics, Nintendo Switch, and more", 31/03/2017

[16] Unity Documentation, "Unity Forma", n.d.

[17] Marisa Graves, Richard Barnes, BusinessWire,"Unity Introduces Unity Forma - An Automotive and Retail Solution Tool for the Creation and Delivery of Custom Real-Time 3D Marketing Content", 09/12/2020

XR è un termine generico che include i seguenti tipi di applicazioni: Realtà Virtuale (VR) - l'applicazione simula un ambiente completamente diverso intorno all'utente; Mixed Reality (MR) - l'applicazione combina il proprio ambiente con l'ambiente del mondo reale dell'utente e consente loro di interagire tra di loro; Realtà Aumentata (AR) - l'applicazione sovrappone il contenuto a una visione digitale del mondo reale.
Unity ha sviluppato un nuovo plug-in framework chiamato XR SDK che consente ai provider XR di integrarsi con il motore Unity e sfruttare appieno le sue funzionalità. Questo approccio basato su plug-in migliora la capacità di Unity di correggere rapidamente i bug, distribuire gli aggiornamenti dell'SDK dai partner della piattaforma e di supportare nuovi dispositivi XR e runtime senza dover modificare il motore principale.
Per iniziare lo sviluppo di un'applicazione AR, Unity consiglia di utilizzare AR Foundation che crea l'applicazione per i dispositivi AR portatili e indossabili supportati da Unity. AR Foundation consente di lavorare con piattaforme di realtà aumentata in modo multipiattaforma all'interno di Unity. Questo pacchetto presenta un'interfaccia che gli sviluppatori di Unity possono utilizzare, ma non implementa alcuna funzionalità AR da sola. Per utilizzare AR Foundation su un dispositivo di destinazione, devi anche scaricare e installare pacchetti separati per ciascuna delle piattaforme di destinazione ufficialmente supportate da Unity:

- Plug-in ARCore XR su Android
- Plug-in ARKit XR su iOS
- Plug-in Magic Leap XR su Magic Leap
- Plug-in di Windows XR su HoloLens

AR Foundation supporta le funzionalità come tracciamento del dispositivo, raycast, rilevamento del piano, punti di riferimento, rilevamento della nuvola di punti, gesti, tracciamento del viso, tracciamento delle immagini 2D, tracciamento degli oggetti 3D, environment probes (rileva informazioni sull'illuminazione e sul colore in aree specifiche dell'ambiente), meshing, tracciamento del corpo 2D e 3D body, segmentazione umana, occlusione, tracciamento dei partecipanti.[18]
Per iniziare lo sviluppo della realtà virtuale, Unity consiglia di utilizzare XR Management per caricare e gestire gli SDK della piattaforma di destinazione.

2.1.1. LipSync

LipSync o LipSynch (abbreviazione di sincronizzazione labiale) è un termine tecnico per abbinare i movimenti delle labbra di una persona che parla o canta con la voce cantata o parlata. L'audio per la sincronizzazione labiale viene generato tramite il sistema di amplificazione del suono in un'esibizione dal vivo o tramite televisione, computer, altoparlanti del cinema o altre forme di uscita audio. Nella produzione cinematografica , la sincronizzazione labiale fa spesso parte della fase post-produzione. Doppiaggio dei film o dei videogiochi è un esempio della sincronizzazione labiale.
RogoDigital LipSync è un plug-in per il motore grafico Unity per semplificare il processo di creazione di animazioni facciali di alta qualità e sincronizzazione labiale. Fornisce una finestra dell'editor appositamente creata per sincronizzare fonemi, emozioni e gesti per dialogare in un file audio e un componente per riprodurre questi clip di dialogo su un personaggio utilizzando pose totalmente personalizzabili.

[18] Unity Documentation, “Getting Started with AR development in Unity”, n.d.

L'installazione di LipSync in Unity è piuttosto semplice. Dopo aver scaricato e importato LipSync, si crea la struttura della directory RogoDigital nel progetto corrente. Una volta completata l'importazione, si creano anche altre directory: LipSync Pro e Shared. Dalla versione 1.5, LipSync include una procedura guidata di installazione, è possibile accedere a questa procedura guidata dal menu Window->Rogo Digital->LipSync Pro->AutoSync Setup Wizard.

La procedura guidata rileva automaticamente la versione corretta dell'applicazione di conversione audio SoX in bundle con LipSync Pro. Questo e qualsiasi altro percorso verrà archiviato a livello globale (non specifico del progetto). È consigliato anche di installare il modulo Montreal Forced Aligner per ottenere i migliori risultati da AutoSync e compatibilità con macOS.

Personaggi supportati da LipSync Pro:

- Blend Shape Blend System, per esempio, personaggi creati con Adobe/Mixamo Fuse, Reallusion Character Creator, personaggi personalizzati con blend shapes facciali
- Bones Only Blend System: MakeHuman, personaggi personalizzati con un rig facciale a base di ossa
- Sprite Blend System: personaggi basati su Sprite 2D con più sprite per bocca, occhi, ecc.
- Texture Offset Blend System: personaggi basati su mesh 2D con textures multiple, personaggi 3D con bocche 2D separate
- UMA2 Blend System: personaggi Unity Multipurpose Avatar (UMA) 2

Una volta il personaggio è caricato in Unity, si crea un'istanza di esso nella scena. Il componente LipSync si aggiunge dal menu Add Component sotto Rogo Digital/LipSync Pro. È necessario aggiungere anche un AudioSource per riprodurre i dialoghi. LipSync interagisce con il personaggio scelto attraverso un sistema di miscelazione (Blend System) che dovrebbe essere scelto come il primo passo delle impostazioni del LipSync Component, il menu a tendina mostra tutti i Blend Systems attualmente presentati nel progetto, altri sistemi possono essere scaricati dalla finestra Estensioni.

Figura 16. Interfaccia utente iniziale del componente LipSync Pro in Unity

Il nucleo dell'animazione di LipSync è la posa del fonema (Phoneme Pose). LipSync ha un elenco di fonemi preimpostato, facendo clic su uno di questi si espanderà l'editor di pose (Pose Editor) che consente di personalizzare ciascuna posa una per una.

Figura 17. Impostazioni dei fonemi in LipSync Pro

Quando è finito l'impostazione dei fonemi, il file audio dovrebbe essere sincronizzato con il personaggio. Dalla versione 1.5 di LipSync Pro, AutoSync è stato ampliato in un sistema flessibile e modulare, che può essere utilizzato per automatizzare qualsiasi parte del processo di creazione delle clip. L'uso più comune, tuttavia, è ancora rilevare i fonemi in una clip audio per generare marcatori di fonemi, sono inclusi due metodi per farlo:

- PocketSphinx: questo sistema richiede solo un AudioClip contenete un dialogo, ma fornisce risultati con una qualità estremamente varia ed è compatibile solo con Microsoft Windows. Questo modulo è incluso con il download di LipSync Pro come una impostazione predefinita.
- Montreal Forced Aligner (MFA): questo è un sistema nuovo che utilizza una trascrizione di testo oltre all'AudioClip per fornire risultati di qualità molto superiore rispetto a PocketSphinx ed è compatibile sia con Windows che con macOS. A causa delle sue grandi dimensioni, questo modulo viene scaricato separatamente.

LipSync Pro ha alcuni preset già inclusi in menu, per esempio un preset "Default" viene aggiunto al progetto corrente con il modulo MFA installato. Per usarlo, si apre l'editor di clip (Clip Editor) dal menu Window->Rogo Digital->LipSync Pro, e seleziona un AudioClip da utilizzare. Se disponibile una trascrizione pre-creata per il dialogo, può essere inserita in file .txt nella stessa cartella e con lo stesso nome della clip audio, l'editor di clip la caricherà automaticamente. In caso contrario si può digitare uno nel menu Clip Editor->Edit->Clip Settings nella casella di testo.

Non appena le modifiche vengono salvate, apparirà la barra di avanzamento e dopo alcuni secondi dovrebbero apparire i marcatori di fonemi lungo la timeline.

Esistono due modi per iniziare a riprodurre un'animazione da una LipSyncData clip:

- Play On Awake Setting (riproduci su sveglio): il metodo più semplice è abilitare la casella di controllo "Play on Awake" nell'ispettore LipSync. Quando questa opzione è attiva, all'utente viene fornito un campo oggetto in cui trascinare una clip LipSyncData salvata, e un'opzione predefinita per la lunghezza di ritardo. LipSync riproduce la clip scelta non appena inizia la riproduzione della scena, come una sorgente audio.
- Metodo LipSync.Play: questo è il metodo più comune per riprodurre un'animazione. Si ottiene un riferimento a un componente LipSync, esponendo un riferimento pubblico *LipSync lipSyncCharacter;* o usando *GetComponent.GetComponent<LipSync>();*

Poi si chiama il metodo *.Play(LipSyncData clip, float delay)* su di esso, passando un riferimento a una clip LipSyncData da riprodurre e facoltativamente un ritardo in secondi.
Per accedere all'API LipSync Pro, nella parte superiore del file di script dovrebbe essere aggiunto lo spazio dei nomi: *using RogoDigital.Lipsync;*
Oltre a riprodurre LipSyncData clip contenenti fonemi, emozioni e/o gesti, si può aggiungere manualmente un'emozione personalizzata quando una clip non si riproduce. Questo può essere utile per creare reazioni agli eventi di gioco o per aggiungere più interesse alle animazioni inattive. Viene chiamato il metodo *.SetEmotion(string emotionName, float blendTime)* su qualsiasi componente LipSync passando il nome di qualsiasi emozione definita nelle Impostazioni del progetto. Il personaggio combinerà quindi la sua emozione attuale (o neutra) con l'emozione scelta dal valore blendTime passato. Si può anche chiamare *.ResetEmotion(float blendTime)* per fondersi nuovamente con la faccia neutra del personaggio.
LipSync include un UnityEvent chiamato OnFinishedPlaying che viene chiamato ogni volta che una clip termina, sia in modo naturale sia tramite una chiamata di funzione *.Stop(bool stopAudio)*. Questo può avere ascoltatori aggiunti dall'editor o dal codice.
Il campo *.loop* è un valore booleano che determina se le clip riprodotte ritorneranno all'inizio al loro termine. Se il valore è "true", *onFinishedPlaying* si attiverà ancora alla fine di ogni loop, ma la clip non si fermerà mai da sola. *.Stop(bool stopAudio)* deve essere chiamato manualmente oppure *.loop* deve essere reimpostato su "false".
Il booleano *.keepEmotionWhenFinished* controlla cosa succede alle emozioni alla fine di una clip. Se una clip termina con un'emozione che non si è sfumata e questo valore è "false", l'emozione tornerà comunque a neutra al termine della clip. Se questo valore è "true", tuttavia, l'emozione rimarrà attiva sul personaggio e potrà essere reimpostata in seguito con il metodo *.ResetEmotion(float blendTime)*.

2.1.2. Altri componenti

Eye Controller è un componente separato per aggiungere movimenti oculari automatici e battito di ciglia al personaggio. È progettato per funzionare insieme al componente LipSync condividendo un Blend System, ma può anche essere utilizzato da solo.
Il componente Eye Controller, come il componente LipSync, può essere collegato a qualsiasi GameObject, ma di solito viene posizionato sull'oggetto radice (root object) del personaggio che controlla. Dopo aver aggiunto un nuovo componente Eye Controller, ad esso si assegna un Blend System.

Figura 18. Intefraccia Utente del componente Eye Controller di LipSync Pro

Funzione Blinking (battito di ciglia) aggiunge un sempice effetto di battito casuale al personaggio. Nella modalità classica vengono scelte due BlendShapes, una per ciascun occhio. La modalità Pose-Based li sostituisce con una singola posa in stile LipSync che verrà utilizzata al loro posto, consentendo l'uso di ossa e un numero arbitrario di BlendShapes. Blink Gap controlla la frequenza con cui si verificano i battiti. Il periodo tra i battiti è un numero casuale in secondi. Blink Duration (la durata del battito) è il tempo, in secondi, impiegato da ciascun battito.

Funzione Random Looking (sguardo casuale) viene utilizzata per aggiungere un effetto di sguardo casuale per rendere i personaggi più vivi e interessanti, anche quando non c'è nulla di specifico da guardare. L'unica opzione in questa sezione è Change Direction Gap (periodo di cambio della direzione). Proprio come il Blink Gap, questa è un range che controlla la frequenza con cui gli occhi si girano per guardare in una nuova direzione.

Funzione Look At Target (guarda l'obiettivo) viene utilizzata per far puntare gli occhi in una determinata posizione nel mondo. Ad esempio, può essere utilizzato per consentire il contatto visivo nei giochi VR o negli FPS , o per fare guardare il personaggio di un giocatore agli oggetti con cui interagisce.

Opzione Looking (Shared) è una media tra Random Looking e Look At Target. L'unica opzione disponibile è Eye Turn Speed (la velocità di rotazione degli occhi), valori più alti faranno girare gli occhi più velocemente.

2.1.3. Blend Systems

I Blend Systems sono classi che ereditano dalla classe *RogoDigital.Lipsync.BlendSystem*. Esistono come componenti nascosti su un GameObject a cui è stato aggiunto LipSync o Eye Controller, e vengono gestiti automaticamente. Lo scopo di Blend Systems è quello di consentire a LipSync di essere il più flessibile possibile non facendo effettivamente l'animazione stessa. In termini di Blend Shape Blend System, invece di salvare direttamente i riferimenti a Skinned Mesh Renderer o impostare i valori delle forme di difusione, LipSync passa sempicemente un indice e un valore desiderato al Blend System per gestirlo come vuole.

Una BlendShape è qualsiasi aspetto della rappresentazione del personaggio che Blend System può alterare. Nel Blend Shape Blend System sono Blend Shapes, nello Sprite Blend System sono combinazioni layer/sprite memorizzate nel componente Sprite Manager.
Alcuni Blend Systems sono inclusi immediatamente, altri sono disponibili per il download dalla finestra Estensioni:

- Blend Shape Blend System è il sistema di miscelazione più comune. Usa Blend Shapes su Skinned Mesh Renderer. Richiede un Mesh Renderer principale e supporta anche un numero qualsiasi di quelli aggiuntivi purché abbiano lo stesso set di Blend Shapes;
- Advanced Blend Shape Blend System è una espansione di Blend Shape Blend System, questo sistema consente di utilizzare un numero qualsiasi di mesh completamente non correlate insieme. Mentre il normale sistema di BlendShapes richiede che tutte le mesh abbiano lo stesso set di BlendShapes, nelle stesse posizioni, quello avanzato aggiunge un componente Blend Shape Manager che memorizza i riferimenti a diverse BlendShapes su diverse mesh e le raggruppa insieme.
- Sprite Blend System è il sistema progettato per funzionare con Sprite 2D. Se selezionato, verrà aggiunto anche un componente Sprite Manager al GameObject. Si può usare questo componente per definire uno o più "livelli" (Sprite Renderer) e tutti gli sprite necessari. È quindi possibile selezionare le combinazioni di livello/sprite da utilizzare nelle pose. Ogni livello visualizza lo sprite con il valore più alto di BlendShape e fa swap (scambio) di sprite quando un valore diventa maggiore. Ad esempio, questa funzionalità può creare un livello in primo piano per la bocca e un secondo livello per il resto del viso. Le pose di emozione potrebbero quindi utilizzare il livello di sfondo e le pose di fonema userebbero quello in primo piano, per consentire a entrambi di essere visualizzati in modo indipendente.
- Texture Offset Blend System funziona in modo molto simile allo Sprite Blend System, ma imposta invece Texture, Texture Offset e Texture Scale di una texture su un materiale. Se pianificata correttamente, questa tecnica può essere utilizzata per animazioni facciali in stile anime su un personaggio.
- Bones Only Blend System è una pseudo Blend System, poiché in realtà non fa nulla. Disabilita semplicemente tutte le funzioni correlate alle BlendShapes per rendere l'interfaccia utente più chiara se è necessario utilizzare solo le trasformazioni ossee.[19]

2.1.3. Scripting in Unity

Lo scripting è essenziale in tutte le applicazioni che si sviluppano con Unity. La maggior parte delle applicazioni richiede script per rispondere all'input dal giocatore e organizzare gli eventi nel gameplay per farli accadere quando dovrebbero. Gli script possono essere utilizzati per creare effetti grafici, controllare il comportamento fisico degli oggetti o implementare un sistema AI personalizzato per i personaggi del gioco.
Un ambiente di sviluppo integrato (IDE) è un software che fornisce strumenti e strutture per facilitare lo sviluppo di altri software. Unity supporta i seguenti IDE:

- Visual Studio
- Visual Studio Code

[19] RogoDigital Documentation, “LipSync Pro”, n.d.

- JetBrains Rider

Quando si installa Unity su Windows e MacOS, per impostazione predefinita (default), Unity installa Visual Studio. Si può scegliere di escluderlo quando si selezionano i componenti da scaricare e installare. L'editor di script esterno può essere impostato in Menu: Unity->Preferenze->Strumenti Esterni->Editor di Script Esterno.
È possibile utilizzare un debugger per ispezionare il codice sorgente mentre l'applicazione in esecuzione. Tutti gli editor supportati forniscono le funzionalità di base come punti di interruzione, un singolo passo-passo e ispezione della variabile.
Unity ha l'impostazione dell'ottimizzazione del codice che lavora in due modalità:

- Modalità di debug, che può essere utilizzata per allegare software di debug esterno, ma consente prestazioni C# più lente quando si esegue il progetto in modalità Play nell'editor.
- Release mode, che offre prestazioni C# più veloci quando si esegue il progetto in modalità Play in Editor, ma non è possibile allegare alcun debugger esterno.

Quando si fa clic sul pulsante Debug nella barra di stato, si apre una finestra pop-up che contiene un pulsante con quale si può cambiare modalità. La finestra visualizza anche le informazioni sulla modalità corrente e descrive cosa succede se si cambiano modalità.
Con la crescita del progetto, aumenta il numero di script, classi e metodi, e può diventare difficile garantire che un cambiamento in una parte del codice non influenza negativamente altra parte del codice. Il test automatico aiuta a controllare che tutte le parti del codice funzionino come previsto, risparmia tempo identificando dove e quando si verificano problemi non appena vengono introdotti durante lo sviluppo. Unity Test Framework è uno strumento che consente di testare il codice sia in Edit Mode che in Play Mode, e anche su piattaforme di destinazione come Standalone, Android e iOS.

2.1.3.1. Creazione e utilizzo di script

Il comportamento dei GameObjects è controllato dai componenti che sono attaccati a loro. Sebbene i componenti integrati di Unity possano essere molto versatili, lo sviluppatore deve andare oltre ciò che possono fornire per implementare tutte le funzionalità di gioco. Unity consente di creare i componenti custom usando gli script. Questi consentono di attivare gli eventi di gioco, modificare le proprietà dei componenti nel tempo e rispondere all'ingresso dell'utente in qualsiasi modo. Unity supporta il linguaggio di programmazione C# in modo nativo. Oltre a questo, molti altri linguaggi .NET possono essere utilizzati con Unity se possono compilare una DLL compatibile.
Gli script vengono solitamente creati all'interno di Unity, è possibile creare un nuovo script dal menu Crea del pannello del Progetto o selezionando Asset->Creazione->Crea C# Script dal menu principale. Il nuovo script verrà creato in qualsiasi cartella selezionata nel pannello Progetto.
Quando si fa doppio clic su un asset di script, esso verrà aperto in un editor di testo. I contenuti iniziali del file sembreranno qualcosa del genere:

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figura 19. Un esempio del codice C# iniziale dello script creato in Unity

Uno script si connette con i lavori interni di Unity implementando una classe che deriva dalla classe built-in chiamata MonoBehaviour. Una classe può essere pensata come un tipo di blueprint per la creazione di un nuovo tipo di componente che può essere collegato a GameObject. Ogni volta che si collega un componente di script a un GameObject, esso crea una nuova istanza dell'oggetto definito dal blueprint. Il nome della classe viene prelevato dal nome del file.

Le cose principali da notare, tuttavia, sono le due funzioni definite all'interno della classe.

La funzione di aggiornamento (Update) contiene il codice che gestisce l'aggiornamento di fotogramma per il GameObject. Ciò potrebbe includere movimento, attivare azioni e rispondere all'input dell'utente. Per abilitare la funzione di aggiornamento di eseguire il suo lavoro, è spesso utile essere in grado di impostare le variabili, leggere le preferenze e creare connessioni con altri GameObjects prima che avvenga qualsiasi azione di gioco.

La funzione di avvio (Start) sarà chiamata da Unity prima dell'inizio del gameplay (cioè, prima che la funzione di aggiornamento sia chiamata per la prima volta) ed è un luogo ideale per fare inizializzazioni.

Importante notare, che l'inizializzazione di un oggetto non sia eseguita utilizzando una funzione di costruttore. Questo perché la costruzione di oggetti è gestita dall'editor e non ha luogo all'inizio del gameplay come si potrebbe aspettare. Se si tenta di definire un costruttore per un componente di script, esso interferirà con il normale funzinamento di Unity e può causare gravi problemi con il progetto.

Uno script definisce solo un blueprint per un componente e quindi nessuno dei suoi codici verrà attivato fino a quando un'istanza dello script è collegata a un GameObject. È possibile allegare uno script trascinando lo script asset su uno di GameObjects nel pannello Gerarchia o nell'ispettore del GameObject attualmente selezionato.

2.1.3.2. Prefab

Il sistema Prefab di Unity consente di creare, configurare e archiviare un GameObject, completo di tutti i suoi componenti, valori di proprietà e GameObjects figli, come una risorsa riutilizzabile. Una istanza di un prefabbricato può essere creata utilizzando solamente una riga di codice.

Per creare un'istanza di un prefabbricato in fase di esecuzione, il codice necessita di un riferimento a tale prefabbricato. Questo riferimento può essere creato tramite una variabile pubblica che contiene il riferimento

Prefab. La variabile pubblica nel codice viene visualizzata come campo assegnabile nell'Inspector. È quindi possibile assegnare il prefabbricato effettivo che si desidera utilizzare nell'ispettore.

L'esempio di script seguente ha una singola variabile pubblica, "myPrefab", che è un riferimento a un Prefab. Crea un'istanza di quel Prefab nel metodo Start():

```
using UnityEngine;
public class InstantiationExample : MonoBehaviour
{
    // Reference to the Prefab. Drag a Prefab into this field in the Inspector.
    public GameObject myPrefab;

    // This script will simply instantiate the Prefab when the game starts.
    void Start()
    {
        // Instantiate at position (0, 0, 0) and zero rotation.
        Instantiate(myPrefab, new Vector3(0, 0, 0), Quaternion.identity);
    }
}
```

Figura 20. Un esempio dello script C# che crea un'istanza di Prefab

Un altro esempio dell'utilizzo dei Prefab:

```
using UnityEngine;
public class CircleFormation : MonoBehaviour
{
   // Instantiates prefabs in a circle formation
   public GameObject prefab;
   public int numberOfObjects = 20;
   public float radius = 5f;
   void Start()
   {
       for (int i = 0; i < numberOfObjects; i++)
       {
           float angle = i * Mathf.PI * 2 / numberOfObjects;
           float x = Mathf.Cos(angle) * radius;
           float z = Mathf.Sin(angle) * radius;
           Vector3 pos = transform.position + new Vector3(x, 0, z);
           float angleDegrees = -angle*Mathf.Rad2Deg;
           Quaternion rot = Quaternion.Euler(0, angleDegrees, 0);
           Instantiate(prefab, pos, rot);
       }
   }
}
```

Figura 21. Un esempio dello script C# che crea multiple istanze di Prefab

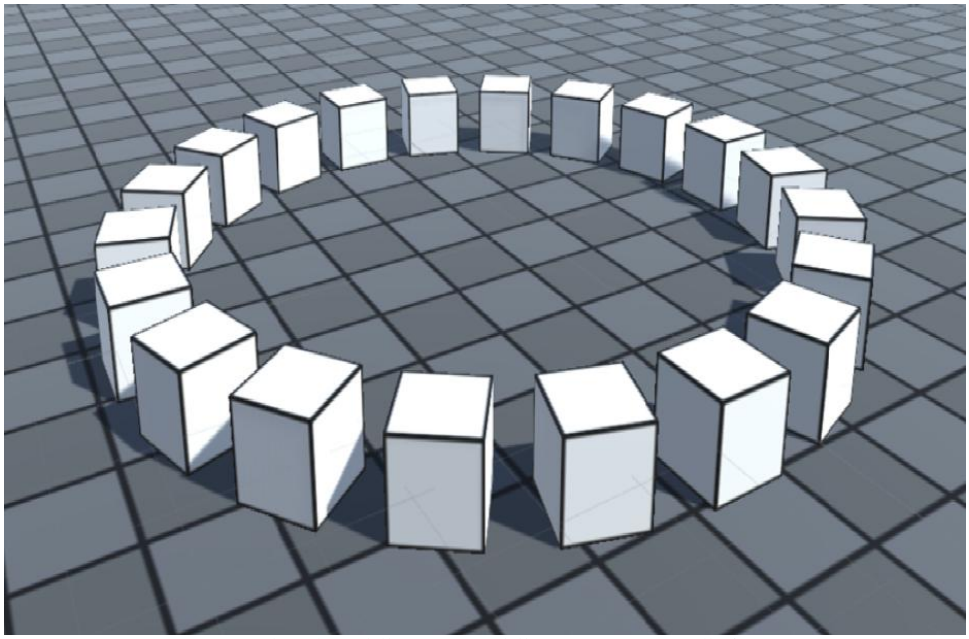

Figura 22. Il risultato dell'esecuzione dello script riportato nella figura 17

2.1.3.3. Eventi

Uno script in Unity non è come l'idea tradizionale del programma in cui il codice viene eseguito continuamente in ciclo finché non completa il suo compito. Al contrario, Unity passa il controllo a uno script in modo intermittente chiamando determinate funzioni dichiarate al suo interno. Al termine dell'esecuzione di una funzione, il controllo viene restituito a Unity. Queste funzioni sono note come funzioni evento poiché vengono attivate da Unity in risposta a eventi che si verificano durante il gioco. Unity utilizza uno schema di denominazione per identificare quale funzione chiamare per un particolare evento.

2.1.3.4. Coroutines

Una coroutine consente di distribuire le attivtà su più frame. In Unity, una coroutine è un metodo che può sospendere l'esecuzione e restituire il controllo a Unity, ma poi continuare da dove era stato interrotto nel frame successivo.

Nella maggior parte delle situazioni, quando si chiama un metodo, viene eseguito fino al completamento e quindi restituisce il controllo al metodo chiamante, oltre a eventuali valori restituiti facoltativi. Ciò significa che qualsiasi azione che avviene all'interno di un metodo deve avvenire all'interno di un singolo aggiornamento del frame.

Nelle situazioni in cui si desidera utilizzare una chiamata al metodo per contenere un'animazione procedurale o una sequenza di eventi nel tempo, è possibile utilizzare una coroutine.

```
IEnumerator Fade()
{
    Color c = renderer.material.color;
    for (float alpha = 1f; alpha >= 0; alpha -= 0.1f)
    {
        c.a = alpha;
        renderer.material.color = c;
        yield return null;
    }
}
```

Figura 23. Un esempio di coroutine

```
void Update()
{
    if (Input.GetKeyDown("f"))
    {
        StartCoroutine(Fade());
    }
}
```

Figura 24. Un esempio del codice utilizzato per impostare una coroutine in esecuzione

2.1.3.5. Classi importanti

- GameObject: rappresenta il tipo di oggetti che possono esistere in una scena.
- MonoBehaviour: la classe base da cui deriva ogni script Unity, by default.
- Object: la classe base per tutti gli oggetti a cui Unity può fare riferimento nell'editor.
- Transform: offre una varietà di modi per lavorare con la posizione, la rotazione e la scala di un GameObject tramite script.
- Vectors: classe per esprimere e manipolare punti 2D, 3D e 4D, linee e direzioni.
- Quaternion: una classe che rappresenta una rotazione assoluta o relativa e fornisce metodi per crearle e manipolarle.
- ScriptableObject: un contenitore di dati che si può utilizzare per salvare grandi quantità di dati.
- Time: questa classe consente di misurare e controllare il tempo e gestire il framerate del progetto.
- Mathf: una raccolta di funzioni matematiche comuni, incluse funzioni trigonometriche, logaritmiche e altre comunemente richieste nei giochi e nello sviluppo di app.
- Random: fornisce metodi semplici per generare vari tipi di valori casuali comunemente richiesti.
- Debug: consente di visualizzare le informazioni nell'editor che possono aiutare a comprendere o indagare su ciò che sta accadendo nel progetto mentre è in esecuzione.
- Gizmos and Handles: consente di disegnare linee e forme nella vista Scena e nella vista Gioco, oltre a maniglie e controlli interattivi.

2.1.4. Architettura di Unity

Il motore Unity è compilato internamente con C/C++ nativo, tuttavia ha un wrapper C# che si usa per interagire con esso. Unity utilizza la piattaforma .NET open source per garantire che le applicazioni realizzate con Unity possano essere eseguite su un'ampia varietà di configurazioni hardware diverse.

Unity ha due backend di scripting, Mono e IL2CPP (Intermediate Language To C++), ognuno dei quali utilizza una tecnica di compilazione diversa:

- Mono utilizza la compilazione JIT (just-in-time) e compila il codice su richiesta in fase di esecuzione.
- IL2CPP utilizza la compilazione anticipata (AOT: ahead-of-time) e compila l'intera applicazione prima che venga eseguita.

Il vantaggio dell'utilizzo di un back-end di scripting basato su JIT è che il tempo di compilazione è in genere molto più veloce di AOT ed è indipendente dalla piattaforma.

L'editor Unity è basato su JIT e utilizza Mono come backend di scripting. Quando si crea un player per l'applicazione, si può scegliere quale backend di scripting usare. Per farlo tramite l'Editor, si può andare su Modifica->Impostazioni proggetto->Player, da qua si apre il pannello Altre impostazioni, si fa clic sul menu a discesa Scripting Backend e si seleziona il backend desiderato.

Quando si compila l'applicazione, Unity esegue la scansione degli assembly compilati (.DLL) per rilevare e rimuovere il codice inutilizzato. Questo processo riduce la dimensione binaria finale della tua build, ma aumenta il tempo di compilazione.

L'eliminazione del codice è disabilitata by default quando si utilizza Mono, ma l'eliminazione del codice non può essere disabilitata per IL2CPP. La rimozione del codice potrebbe essere troppo aggressiva in alcuni casi e potrebbe rimuovere il codice su cui si fa affidamento, specialmente quando si usa la riflessione. È possibile utilizzare attributi Preserve e file link.xml per impedire l'eliminazione di tipi e funzioni specifici.

Unity utilizza il Garbage Collector Boehm sia per il backend Mono che per IL2CPP. Unity usa Incremental mode by default. Si può disabilitare la modalità Incremental per utilizzare la raccolta dei rifiuti "stop the world", sebbene Unity consigli l'uso della modalità Incremental.

In modalità Incremental, il Garbage Collector di Unity viene eseguito solo per un periodo di tempo limitato e non raccoglie necessariamente tutti gli oggetti in un singolo passaggio. Questo distribuisce il tempo necessario per raccogliere oggetti su un numero di fotogrammi e riduce la quantità di balbuzie e picchi di CPU.

Si utilizza l'Unity Profiler per controllare il numero di allocazioni e possibili picchi di CPU nell'applicazione. Si può anche utilizzare l'API GarbageCollector per disabilitare completamente la raccolta dei rifiuti per i giocatori. Quando il garbage collector è disabilitato, lo sviluppatore dovrebbe fare attenzione a evitare di allocare memoria in eccesso.

Unity supporta molte piattaforme e potrebbe utilizzare backend di scripting diversi a seconda della piattaforma. Le librerie di sistema .NET richiedono implementazioni specifiche della piattaforma per funzionare correttamente in alcuni classi. Sebbene Unity faccia del suo meglio per supportare quanto più possibile l'ecosistema .NET, esistono alcune eccezioni a parti delle librerie di sistema .NET che Unity non supporta esplicitamente. Unity non fornisce garanzie sulle prestazioni né sull'allocazione delle librerie di sistema .NET nelle versioni di Unity. Come regola generale, Unity non risolve eventuali regressioni delle prestazioni nelle librerie di sistema .NET.

Unity non suporta la libreria System.Drawing e non è garantito che funzioni su tutte le piattaforme.

Un backend di scripting JIT consente di generare le generazione di codice C#/.NET Intermediate Language (IL) dinamico durante il runtime dell'applicazione, mentre un backend di scripting AOT non supporta la generazione di codice dinamico. Questo è importante da considerare quando si usano librerie di terze parti, perché

potrebbero avere percorsi di codice diversi per JIT e AOT oppure potrebbero usare percorsi del codice che si basano su codice generato dinamicamente.
Mono e IL2CPP memorizzano internamente nella cache tutti gli oggetti di riflessione C# (System.Reflection) e, in base alla progettazione, Unity non li elimina. Il risultato di questo comportamento è che il Garbage Collector esegue continuamente la scansione degli oggetti di riflessione C# memorizzati nella cache durante la durata dell'applicazione, causando un sovraccarico del Garbage Collector non necessario e potenzialmente significativo. Per ridurre al minimo l'overhead del Garbage Collector, è raccomandato evitare metodi come Assembly.GetTypes e Type.GetMethods() nell'applicazione, che creano molti oggetti di riflessione C# in fase di esecuzione. È invece necessario eseguire la scansione degli assembly nell'Editor per i dati richiesti e serializzarli e/o codificarli per utilizzarli in fase di esecuzione.

2.1.4.1. UnityEngine.Object

UnityEngine.Object è un tipo speciale di oggetto C# in Unity, perché è collegato a un oggetto controparte C++ nativo. Ad esempio, quando si usa un componente Camera, Unity non archivia lo stato dell'oggetto nell'oggetto C#, ma piuttosto nella sua controparte C++ nativa.
Quando si utilizza un metodo come Object.Destroy o Object.DestroyImmediate per distruggere un oggetto derivato da UnityEngine.Object, Unity distrugge (scarica) l'oggetto contatore nativo. Non è possibile distruggere l'oggetto C# con una chiamata esplicita, perché il Garbage Collector gestisce la memoria. Quando non sono più presenti riferimenti all'oggetto gestito, il Garbage Collector lo raccoglie e lo distrugge.
Se si accede nuovamente a UnityEngine.Object distrutto, Unity crea l'oggetto controparte nativo per la maggior parte dei tipi. Due eccezioni a questo comportamento di ricreazione sono MonoBehaviour e ScriptableObject: Unity non li ricarica mai una volta che sono stati distrutti.
MonoBehaviour e ScriptableObject sostituiscono gli operatori di uguaglianza (==) e disuguaglianza (!=). Pertanto, se si confronta un MonoBehaviour o ScriptableObject distrutto con null, gli operatori restituiscono true quando l'oggetto gestito esiste ancora e non è stato ancora sottoposto a Garbage Collection.
L'API Unity non è thread-safe e, pertanto, lo sviluppatore non dovrebbe usare attività asincrone e in attesa. Le attività asincrone spesso allocano oggetti quando vengono richiamate, il che potrebbe causare problemi di prestazioni in caso di utilizzo eccessivo. Inoltre, Unity non interrompe automaticamente le attività asincrone eseguite sui thread gestiti quando si esce dalla modalità di riproduzione.
Unity sovrascrive il SyncronizationContext predefinito con un UnitySyncronizationContext personalizzato ed esegue tutte le attività nel thread principale in entrambe le modalità Modifica e Riproduci. Per utilizzare le attività asincrone, è necessario creare e gestire manualmente i propri thread con un TaskFactory, nonché utilizzare il SynchronizationContext predefinito invece della versione Unity. Per ascoltare gli eventi di accesso e uscita dalla modalità di riproduzione per interrompere manualmente le attività, si può utilizzare EditorApplication.playModeStateChanged. Tuttavia, se si adotta questo approccio, la maggior parte delle API di scripting Unity non è disponibile per l'uso perché non si utilizza UnitySyncronizationContext.

2.1.4.2. Ricaricamento del codice nell'editor di Unity

Enter Play Mode configurabile.

La Play Mode è una delle funzionalità principali di Unity. Essa ti consente di eseguire il tuo progetto direttamente all'interno dell'Editor, tramite il pulsante Riproduci nella barra degli strumenti. Quando lo sviluppatore accede alla modalità di riproduzione, il progetto si avvia e viene eseguito come in una build. Qualsiasi modifica apportata nell'editor durante la modalità di riproduzione viene ripristinata quando esci dalla modalità di riproduzione.

Quando si accede alla modalità di riproduzione nell'editor, Unity esegue due azioni significative per garantire che il progetto venga avviato nell'editor allo stesso modo di una build:

- Ripristina lo stato di scripting (questo è anche chiamato "Reload del dominio" o "Domain Reload").
- Ricarica la scena .

Queste due azioni richiedono del tempo per essere eseguite e la quantità di tempo aumenta quando gli script e scene diventano più complessi.

La possibilità di entrare e uscire rapidamente dalla modalità di gioco è un fattore importante durante lo sviluppo del gioco o dell'app. Più velocemente si può entrare e uscire dalla modalità di riproduzione, più velocemente si può apportare e testare le modifiche.

Poiché una rapida velocità di iterazione durante lo sviluppo è importante e poiché il tempo necessario per ripristinare la scena e lo stato di scripting può diventare un ostacolo, Unity offre la possibilità di configurare ciò che accade quando si accede alla modalità di riproduzione, si può disabilitare sia, o entrambe, le azioni "Domain Reload" e "Scene Reload".

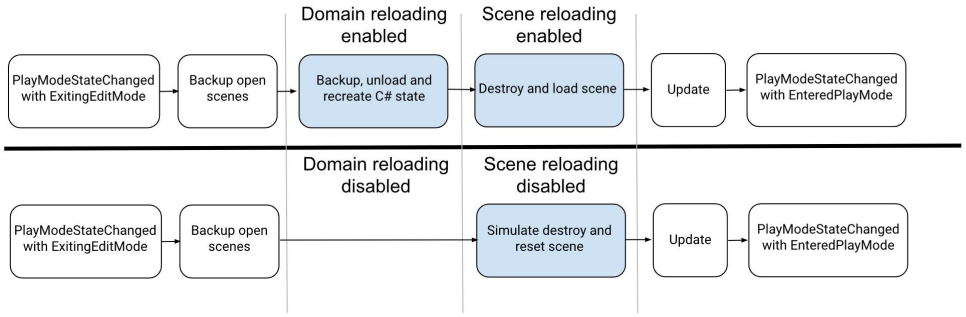

Figura 25. Il diagramma che mostra gli effetti della disabilitazione delle impostazioni "Reload Domain" e "Reload Scene"

Esecuzione dello script all'avvio.

A volte, è utile essere in grado di eseguire del codice dell'editor in un progetto non appena Unity viene avviato senza richiedere un'azione da parte dell'utente. Esso può essere fatto applicando l'attributo InitializeOnLoad a una classe che ha un costruttore statico. Un costruttore statico è una funzione con lo stesso nome della classe, dichiarata statica e senza un tipo o parametri restituiti.

È sempre garantito che un costruttore statico venga chiamato prima che venga utilizzata qualsiasi funzione statica o istanza della classe, ma l'attributo InizializeOnLoad garantisce che venga chiamato all'avvio dell'editor.

2.1.4.3. Serializzazione di script

La serializzazione è il processo automatico di trasformazione delle strutture dati o degli stati degli oggetti in un formato che Unity può archiviare e ricostruire in seguito. Alcune delle funzionalità integrate di Unity utilizzano la serializzazione; funzioni come il salvataggio e il caricamento, la finestra di ispezione, l'istanza e i prefabbricati. Il modo in cui sono organizzati i dati nel progetto Unity influisce sul modo in cui Unity serializza tali dati e può avere un impatto significativo sulle prestazioni del progetto.

Hot reloading è un processo di creazione o modifica degli script menntre l'Editor è aperto e l'applicazione immediata dei comportamenti degli script. Non è necessario riavviare l'applicazione o l'Editor per rendere effettive le modifiche. Quando si modifica e si salva uno script, Unity ricarica tutti i dati dello script attualmente caricati. Prima memorizza tutte le variabili serializzabili in tutti gli script caricati e, dopo aver caricato gli script, li ripristina. Tutti i dati non serializzabili vengono perso dopo un'hot reload.

Unity utilizza la serializzazione per caricare e salvare scene, asset e assetbundle da e verso il disco rigido del computer. Ciò include i dati salvati nei oggetti API di scripting come componenti MonoBehaviour e ScriptableObjects.

Quando si visualizza o si modifica il valore del campo del componente di un GameObject nella finestra Inspector, Unity serializza questi dati e quindi li visualizza nella finestra Inspector. La finestra di ispezione non comunica con l'API Unity Scripting quando visualizza i valori di un campo. Se si utilizzano le proprietà nello script, nessuno dei getter e dei setter di proprietà viene mai chiamato quando si visualizzano o si modificano i valori nelle finestre di ispezione poiché Unity serializza direttamente i campi della finestra di ispezione. Ciò significa che: Mentre i valori di un campo nella finestra Inspector rappresentano le proprietà dello script, le modifiche ai valori nella finestra Inspector non chiamano alcun getter o setter di proprietà nello script.

Le classi personalizzate che non sono derivate da UnityEngine.Object sono serializzate inline in base al valore, in modo simile al modo in cui sono serializzate gli struct. Se è memorizzato un riferimento a un'istanza di una classe personalizzata in diversi campi, questi diventano oggetti separati quando serializzati. Quindi, quando Unity deserializza i campi, questi contengono diversi oggetti distinti con dati identici.

Quando è necessario serializzare un grafico di oggetti complessi con riferimenti, a Unity non deve essere consentito di serializzare automaticamente gli oggetti. Per serializzarli manualmente si utilizza ISerializationCallbackReceiver, ciò impedisce a Unity di creare più oggetti da riferimenti a oggetti.

Unity serializza le classi personalizzate "inline" perché i loro dati diventano parte dei dati di serializzazione completi per MonoBehaviour o ScriptableObject in cui vengono utilizzati.

Unity non supporta il polimorfismo, perché il layout del flusso di dati per un oggetto è noto in anticipo, dipende dai tipi dei campi della classe, piuttosto che da ciò che capita di essere memorizzato all'interno dei campi. Un modo per affrontare questa limitazione è rendersi conto che si applica solo alle classi personalizzate, che vengono serializzate inline. I riferimenti ad altri UnityEngine.Objects vengono serializzati come riferimenti effettivi e, per questi, il polimorfismo funziona effettivamente. Lo sviluppatore dovrebbe creare una classe derivata da ScriptableObject o un'altra classe derivata da MonoBehaviour e fare riferimento a quella. Lo svantaggio è che è necessario archiviare quel MonoBehaviour o scriptable object da qualche parte e che non è possibile serializzarlo inline in modo efficiente.

2.1.5. Visual Scripting con Bolt

Bolt è un asset creato per lo scripting visivo completo in Unity, esso consente ad artisti, designer e programmatori di creare meccaniche di gioco e sistemi interattivi senza scrivere codice. Bolt può essere scaricato da Asset Store ufficiale di Unity.

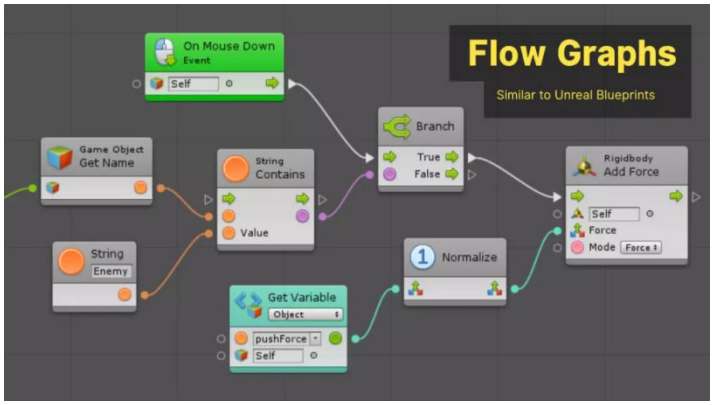

Figura 26. Un esempio di Flow Graph creato con Bolt

Bolt ha 3 finestre principali: Graph, Graph Inspector, Variables. La prima finestra, Graph, è il campo di lavoro in Bolt, qua si crea il grafico che contiene la logica del gioco. Graph Inspector contiene la documentazione di ogni nodo, compreso informazioni sui punti di Input e Output. La terza finestra, Variables, serve per configurare variabili legate all'oggetto.

Bolt lavora con diversi tipi di variabili standard: Float, Integer, Boolean, String, etc., e permette di dichiarare sei tipi di variabili specifici:

- Flow Variables: queste sono variabili usate localmente in un Flow Graph;
- Graph Variables: sono variabili dichiarate localmente in un Flow Graph e non sono visibili all'esterno del grafico in cui sono utilizzate;
- Object Variables: queste variabili appartengono a un oggetto e sono visibili per ogni grafico dell'oggetto stesso;
- Scene Variables: sono variabili a disposizione per ogni oggetto nella scena;
- Application Variables: queste sono variabili che persistono tra le scene, esse verranno "cancellate" solo alla chiusura dell'applicazione;
- Saved Variables: si tratta di variabili che vengono salvate anche dopo la chiusura dell'applicazione. In questo caso si utilizza il sistema di salvataggio di Unity.

Bolt permette di creare due tipi di Grafici: Flow o State. Con il tipo di grafico Flow si possono collegare unità che operano in un certo ordine, è il più comune tipo di grafico e generalmente utilizzato per "programmare" la logica di un oggetto. La finestra di grafico Flow contiene i seguenti elementi:

- Uno spazio di lavoro con una griglia su quale si posizionano e si collegano i blocchi che formano la logica del programma;
- Una barra degli strumenti che contiene una serie di elementi come:
    - Zoom: uno slider che consente di ingrandire o ridurre i nodi dell'area di lavoro;
    - Relations: se abilitato, si può vedere le diverse relazioni tra i pin di ingresso e di uscita dei nodi;

- Values: se abilitato si può visualizzare i valori nei collegamenti tra diversi elementi;
- Dim: se abilitato, tutti elementi fuori il flusso di esecuzione saranno semitrasparenti;
- Carry: se attivato, gli spostamenti di un nodo causeranno gli spostamenti di tutti elementi collegati;
- Align: gli elementi selezionati possono essere allineati nello spazio di lavoro secondo una serie di opzioni;
- Distribute: gli elementi selezionati possono essere distribuiti nell'area di lavoro secondo una serie di opzioni;
- Overview: facendo clic su questo pulsante, si può vedere tutti gli elementi nell'area di lavoro;
- Fullscreen: consente di visualizzare la finestra del grafico nello spazio di Unity, allo stesso modo in cui si otterrebbe con Shift + Spacebar.

I nodi sono gli elementi base di un grafico, sono utilizzati come elementi grafici con ingressi a sinistra ed uscite a destra. I collegamenti indicano l'ordine di esecuzione. Esistono due tipi di collegamenti:

- Controllo: rappresentato da un triangolo bianco che indica la direzione in cui vengono eseguiti i nodi. L'uscita di questo tipo può essere connessa ad un ingresso dello stesso tipo.
- Valore: identificato per tipo consente di inserire valori (parametri) o recuperarli.

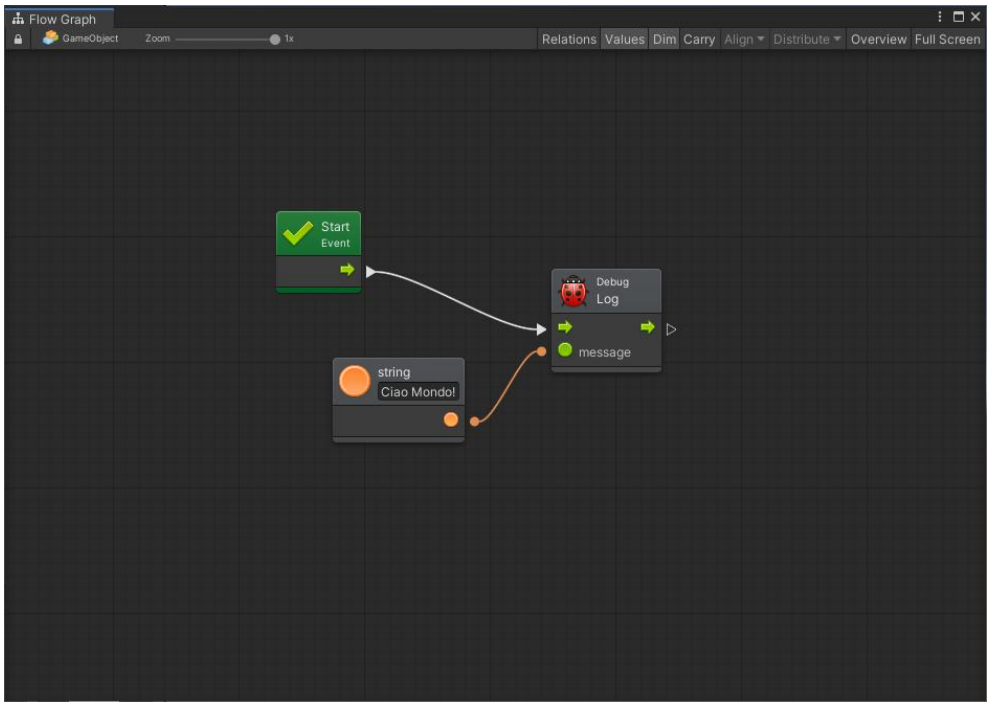

Figura 27. Un esempio di un Flow Graph creato con Bolt

Con lo State Graph si può creare una serie di stati e metterli in relazione tra di loro. Ogni stato dovrebbe essere considerato come una specie di programma in esecuzione. Questo tipo di grafico è il più comunemente usato per creare comportamenti di AI, o logica di alto livello nel progetto.

Uno Stato è un tipo di comportamento che l'oggetto mantiene dal momento di entrata fino al momento di uscita. I stati connessi tra di loro tramite Transizioni, si utilizzano per costruire una sorta di intelligenza artificiale. Esistono tre tipi di stati: Flow, Super e Any States. Quando si esegue uno State Graph, è necessario avere

almeno uno Start State identificato dal colore verde. Se più stati vengono selezionati come Start State, tutti verranno eseguiti in parallelo.

Un grafico Flow State contiene un diagramma di flusso, tutti i nodi all'interno si possono utilizzare esattamente allo stesso modo. Per creare un Flow State, basta cliccare con il pulsante destra nell'area di lavoro e scegliere Create Flow State. Le modifiche possono essere effettuate facendo un doppio click sul grafico. Grafico di tipo Flow State contiene tre tipi di eventi: Update, On Enter State e On Exit State.

Un grafico Super State contiene altri grafici di tipo State, consentendo un'organizzazione gerarchica delle Macchine di Stato. Questo tipo di grafico può essere creato analogamente al grafico Flow State, cioè facendo clic in un'area libera nella zona di lavoro. La struttura del grafico può essere modificando con doppio clic su di esso.

Un grafico Any State consente di creare una transizione da qualsiasi stato a un altro stato specifico.

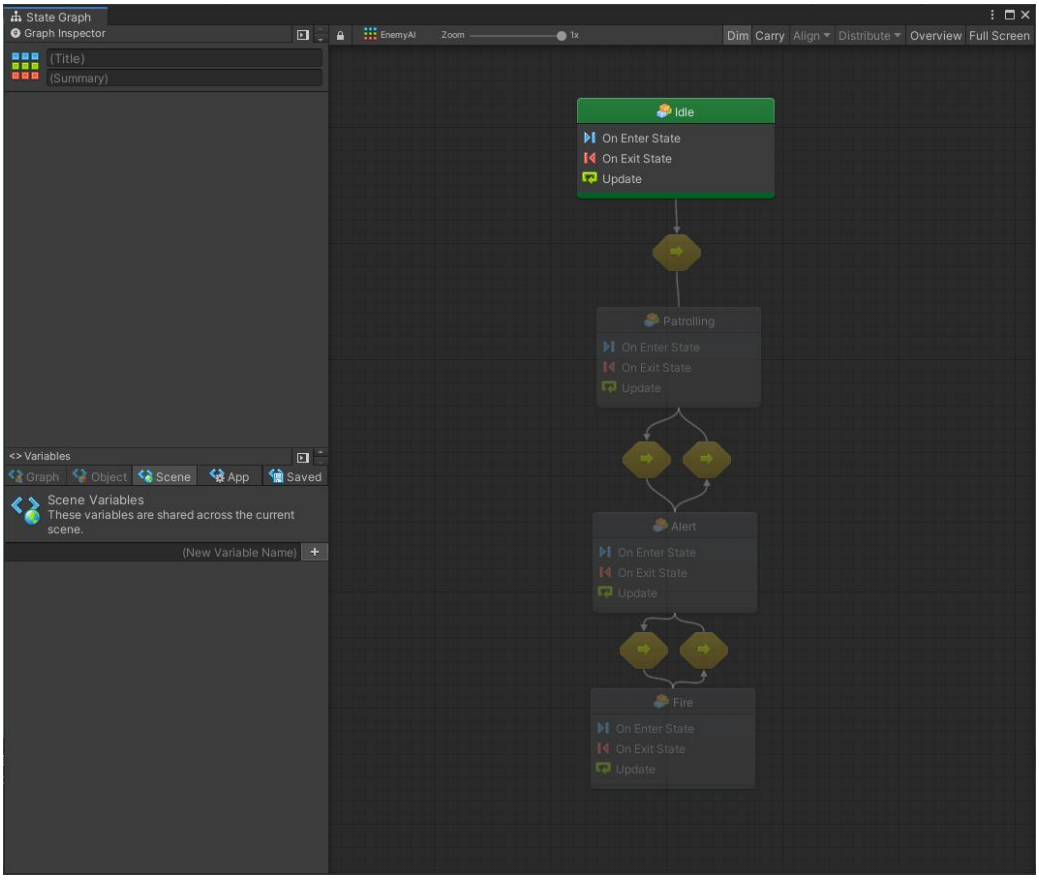

Figura 28. Un esempio di State Graph creato con Bolt

Un Grafico può essere:

- Macro: il grafico è riutilizzabile con diversi componenti del progetto. Poiché la Macro è salvata nel progetto, non ha l'accesso diretto agli oggetti presenti in una scena;
- Embed: il grafico viene innestato direttamente sul componente e non è possibile riutilizzarlo. Se l'oggetto contenete il grafico è presente nella scena, cioè non è istanziato da un prefabbricato, è possibile accedere direttamente a tutti gli elementi che contiene la scena.

2.2. Unreal Engine

Unreal Engine è un motore grafico creato in 1995 da Tim Sweeney, un programmatore di videogiochi e uno dei fondatori di Epic Games. Il motore grafico è nato da una forte concorrenza nello sviluppo dei videogiochi. Tim Sweeney iniziò a sviluppare il motore grafico e due programmatori di videogiochi Cliff Bleszinski e James Schmalz iniziò a progettare un videogioco sparatutto in prima persona per mostrare la potenza del motore. Hanno chiamato il progetto "Unreal".

Sweeney ha iniziato a lavorare al progetto nel suo garage nel 1995, nel 1998 il gioco e il motore erano ancora incompiuti, con i fondi ridotti. I livelli e un'arma sono stati i primi a essere tagliati, ma il linguaggio di programmazione, Unreal Script, insieme a un editor di mappe, UnrealEd, sono stati mantenuti.

Il 22 maggio 1998, Unreal è stato rilasciato. Il plauso della critica, le vendite considerevoli e una comunità leale che ha sfruttato lo script di Unreal, espandendo il gioco tramite Internet, hanno reso Epic un nome istantaneo nella comunità di sviluppo del gioco. Come sperato, il gioco è stato il pezzo forte dell'Unreal Engine. Altri sviluppatori hanno concesso in licenza il potente software per creare i propri giochi, fornendo un flusso di entrate (guadagni) aggiuntivo, che è stato utilizzato per acquisire le aziende piccole.[20]

Tim Sweeney ha iniziato a costruire l'interfaccia utente per Unreal Editor, disponendo l'interfaccia utente in Visual Basic, di tutte le cose. Aveva un'interfaccia della riga di comando in modalità testo per il motore C++ che stava eseguendo il rendering. Successivamente ha scritto l'editor wireframe e da lì è partito.

Tim Sweeney voleva integrare l'editor nel renderer di James Schmaltz e ricevette da James circa 30000 righe di codice assembly, alcune parti delle quali erano così complicate che Tim non voleva nemmeno toccarle con l'editing.

Successivamente avrebbe scritto un piccolo mappatore di texture. Quindi, ha esaminato gli articoli di Michael Abrash sulla mappatura delle texture e ha esaminato alcune cose che Billy Zelsnack ha condiviso.

L'unica vera sfida è stata la creazione dell'albero BSP in tempo reale. L'idea è che puoi riposizionare i pennelli nello spazio 3-D, quindi tutto il lavoro BSP (Binary Space Partitioning) viene aggiornato completamente in tempo reale.

[20] Chris Plante, Polygon, “Better with age: A history of Epic Games”, 01/10/2012

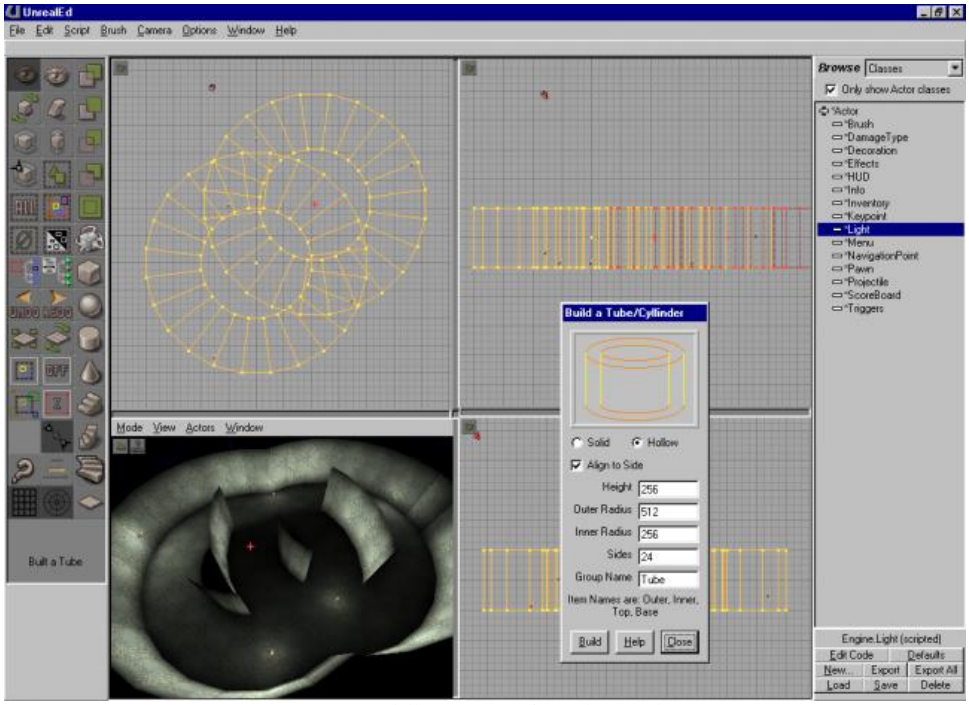

Figura 29. Una delle prime interfacce di Unreal Engine

Una delle basi su quale è stato fondato Unreal Engine è stato il gioco ZZT sviluppato da Tim Sweeney. Il gioco aveva un editor interattivo in tempo reale per creare livelli, si poteva andare avanti e indietro con alcuni tasti, aggiungendo un livello, testandolo e ripetendolo. Quel flusso di lavoro interattivo è stato la chiave. Il gioco è stato ispirato da Turbo Pascal, che è stato il primo set di strumenti utilizzato da Sweeney, era un editor facile da usare per creare codice e compilarlo.

C'è stata un'altra grande ispirazione che ha portato a molti degli elementi di design del motore grafico, ed è stato Visual Basic. L'idea era che l'utente aveva un editor di moduli, si poteva disegnare alcuni elementi e caselle del modulo e poi fare un click su di esso e richiamare il codice. Sweeney ha trasferito tutti questi principi su Unreal: l'utente può rilasciare un oggetto nel livello, fare doppio clic su di esso e si apre l'editor di script. Poi si può andare a digitare uno script ed editarlo. Basterebbero pochi clic del mouse per poter scrivere codice, creare oggetti 3D e fare tutto in modo interattivo in tempo reale.

L'altra cosa che è stata davvero legata allo sviluppo di Unreal è che Epic ha acquistato alcune workstation Silicon Graphics e hanno installato la prima versione di Maya su di esse. All'epoca l'unica cosa davvero interessante che aveva Maya era la modalità 3D interattiva in cui esistevano gli spazi con sfondo blu, rosso e nero, con questi contorni di oggetti e wireframes che erano completamente in real time. Nessuno degli altri programmi aveva realmente raggiunto queste … in quel momento.

Dopo aver terminato Unreal Editor 1, Sweeney ha iniziato fare le ricerche di nuova generazione che generalmente non hanno raggiunto i risultati, mentre Warren Marshall è intervenuto e ha riscritto le parti di Visual Basic dell'Unreal Editor in C++ usando wxWidgets che, al tempo, era la cosa migliore disponibile. Questa era la base del framework dell'interfaccia utente in Unreal Editor 2.

Entro la metà della generazione di Unreal Engine 2, Visual Basic era completamente scomparso dal processo. Il vero problema è stato che, nel tempo, wxWidgets non è migliorato e sono usciti altri toolkit per l'interfaccia utente, quindi i sviluppatori hanno continuato a integrare di nuovi strumenti speciali. All'inizio del ciclo di sviluppo dell'Unreal Engine 4, cinque diversi toolkit dell'interfaccia utente erano già integrate. Nick Atamas ha

risolto questo problema creando un nuovo livello dell'interfaccia utente in C++, chiamata Slate, che non aveva più bisogno di utilizzare tutti i toolkit integrati in passato.[21]

Figura 30. Interfaccia grafica di Unreal Engine 2

Le ultime versioni dell'Unreal Editor hanno molte funzionalità che consentono agli sviluppatori di creare scene fotorealistiche renderizzate in tempo reale, utilizzare gli strumenti di realtà virtuale, animare i personaggi in modo semplice e veloce, ecc.

Figura 31. Interfaccia grafica di Unreal Engine 4

2.2.1. LipSync in Unreal Engine

In Unreal Engine la sincronizzazione labiale può essere creata tramite diversi plug-in che possono essere scaricati dallo store ufficiale o da un sito dei sviluppatori e importato in Unreal Engine. Uno dei esempi è Ynnk Voice Lipsync. Questo plugin utilizza il motore di riconoscimento vocale per generare animazioni di

[21] David Lightbown, Gamasutra, "Classic Tools Retrospective: Tim Sweeney on the first version of the Unreal Editor", 01/09/2018

sincronizzazione labbiale da risorse SoundWave o dati audio PCM. L'animazione viene salvata come curve negli asset di dati e può essere riprodotta in runtime insieme all'audio. A differenza della soluzione utilizzata in Unity, per questo plugin non sono necessari sottotitoli per animare le labbra e l'animazione risultante è molto più realistica. Ynnk Voice Lipsync genera la sincronizzazione labiale in runtime, ma non in tempo reale.[22]
Altro strumento che potrebbe essere utilizzato con Unreal Engine è Oculus Lipsync. Esso analizza il flusso di input audio dall'input del microfono o da un file audio e prevede una serie di valori chiamati visemi, che sono gesti o espressioni delle labbra e del viso che corrispondono a un particolare suono vocale. Oculus Lipsync può essere utilizzato non solo in runtame, ma anche in tempo reale.[23]

2.2.2. Scripting in Unreal Engine

Unreal Engine utilizza il linguaggio di programmazione basato su testo, C++, inoltre, esiste lo scripting visivo chiamato Blueprints che utilizza un'opzione di programmazione più veloce in modalità drag-and-drop. Ogni opzione può svolgere gli stessi compiti, ma una potrebbe essere più adatta dell'altra nelle situazioni diverse.
Uno dei vantaggi dell'utilizzo di C++ per creare meccaniche di gioco è una maggiore flessibilità quando si tratta di personalizzare il funzionamento del gioco. Ciò significa che uno sviluppatore ha l'accesso all'intero codice del gioco piuttosto che a frammenti in Blueprints. Anche le meccaniche C++ vengono eseguite più rapidamente rispetto alle versioni Blueprint e, sebbene ciò potrebbe non essere evidente attraverso i progetti di codifica di giochi per principianti, è importante tenere a mente mentre si continua il percorso di sviluppo del gioco. Un altro vantaggio del C++ è l'organizzazione di meccanismi che utilizzano determinate funzioni come Tick o BeginPlay. In C++, le meccaniche possono essere mantenute organizzate, ma in Blueprints potrebbe essere difficile connettere diverse meccaniche a un singolo nodo Event Tick.
Uno dei maggiori svantaggi di lavorare con C++ è che è più facile commettere errori quando si creano meccaniche di gioco. La maggior parte degli errori sono gli errori minori, ma alcuni possono essere più difficili da rintracciare e, in alcuni casi, potrebbe anche causare il crash di Unreal Engine se l'errore è critico. D'altra parte, è molto difficile mandare in crash Unreal Engine all'interno di Blueprints, e più difficile causare conflitti che interrompono il gioco.
Un vantaggio di Blueprints è che spesso è più veloce creare una nuova meccanica di gioco grazie alla possibilità di sfruttare i nodi precostruiti; la creazione di nodi e la loro connessione richiede spesso meno tempo rispetto alla digitazione del codice e alla sua compilazione. Blueprints hanno anche un vantaggio quando si tratta degli aspetti visivi e 3D delle meccaniche di gioco. È molto più facile creare un collision box e impostare la dimensione corretta all'interno di un blueprint, per questo motivo, quasi tutte le meccaniche basate sulle collisioni dovrebbero essere create all'interno di Blueprints.
Mentre Blueprints può essere più veloce di C++ quando si tratta di creare la maggior parte delle funzionalità, può anche essere più disordinato. La creazione di complesse meccaniche di gioco con Blueprints può creare una struttura enorme di nodi e fili di collegamento. Ciò può rendere più difficile dire dove si trovano alcune parti della meccanica di gioco o risolvere gli errori se la meccanica non funziona correttamente.[24]

[22] Unreal Engine Marketplace, "Ynnk Voice Lipsync", 23/06/2021
[23] Oculus For Developers, "Oculus Lipsync for Unreal Engine", n.d.
[24] iD Tech, "C++ vs. Blueprints: pros and cons, which should be used, and when?", 05/05/2020

2.2.3. Blueprints scripting

Il sistema Blueprint Visual Scripting in Unreal Engine è un sistema di scripting completo di gioco basato sul concetto di utilizzare un'interfaccia basata su nodi per creare elementi di gioco dall'interno di Unreal Editor. Come molti linguaggi di scripting comuni, Blueprints viene utilizzato per definire classi orientati agli oggetti (OO) o oggetti nel motore. Questo sistema è estremamente flessibile e potente in quanto offre ai progettisti la possibilità di utilizzare virtualmente l'intera gamma di concetti e strumenti generalmente disponibili solo per i programmatori. Inoltre, il markup specifico dei Blueprints disponibile nell'implementazione C++ di Unreal Engine consente ai programmatori di creare sistemi di base che possono essere estesi dai progettisti.

Gli eventi sono il punto di partenza dell'esecuzione del grafico Blueprint e possono essere associati a diverse situazioni di gioco. È immediatamente visibile una sezione degli eventi più comunemente utilizzati, visti come nodi di eventi traslucidi.

Vediamo come esempio come fare il controllo delle collisioni tra due oggetti. Dobbiamo selezionare uno dei oggetti, quello per quale creiamo un evento, poi espandere il menu a discesa Add Event for Box, e scegliere Collisione e selezionare Add On Component Begin Overlap node. In questo momento, l'evento OnComponentBeginOverlap verrà eseguito quando qualcosa si sovrappone al Box trigger. Per controllare se l'oggetto, che sovrappone l'oggetto, è il nostro secondo oggetto, per esempio il giocatore, si utilizza il nodo Equal(Object). Il nodo che rappresenta il giocatore si chiama Get Player Pawn. Adesso dobbiamo collegare il nodo On Component Begin Overlap (Box) al nodo Equal(Object) tramite il punto Other Actor, e il nodo Get Player Pawn al Equal(Object). Adesso vogliamo utilizzare un nodo Flow Control, in particolare il nodo Branch, input di quale connettiamo con l'output del nodo On Component Begin Overlap. Il grafico è pronto per impostare un comportamento diverso da eseguire a seconda che l'oggetto sovrapposto sia il player o meno.

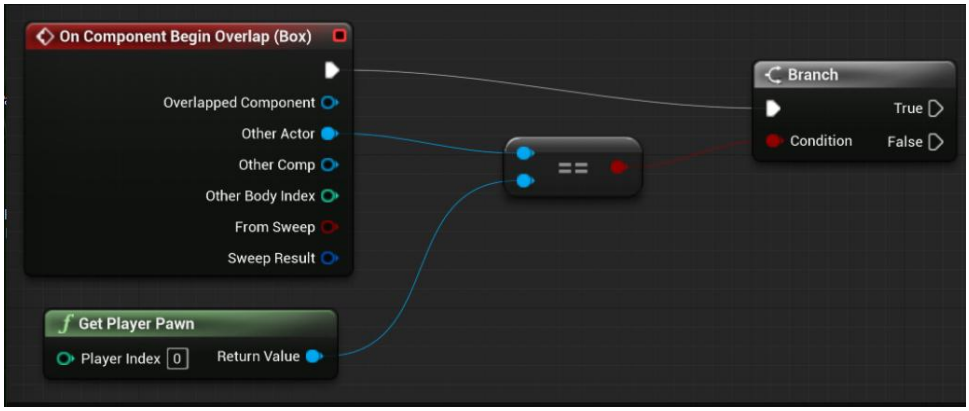

Figura 32. Un esempio di nodi utilizzati nello scripting visivo in Unreal Engine

2.2.3.1. Tipi di Blueprints

Blueprint può essere uno dei diversi tipi che hanno ciascuno il proprio uso specifico, dalla creazione di nuovi tipi degli eventi a livello di script, alla definizione di interfacce o macro da utilizzare da altri Blueprints.

Blueprint Class

Una classe Blueprint, spesso abbreviata in Blueprint, è una risorsa che consente ai creatori di contenuti di aggiungere facilmente funzionalità alle classi già esistenti. Blueprints vengono creati visivamente all'interno di Unreal Editor, invece di digitare il codice, e salvati come risorse in un pacchetto di contenuti. Questi

essenzialmente definiscono una nuova classe o tipo di attore che può quindi essere inserito nelle mappe come istanze che si comportano come qualsiasi altro tipo di attore.

Data-Only Blueprint

Un Data-Only Blueprint è una classe Blueprint che contiene solo il codice (sotot forma di grafici dei nodi), le variabili e i componenti ereditati dal genitore. Questi consentono di modificare le proprietà ereditate, ma non è possibile aggiungere nuovi elementi. Questi sono essenzialmente un sostituto degli archetipi e possono essere utilizzati per consentire ai progettisti di modificare le proprietà o impostare elementi con variazioni. Data-Only Blueprints vengono modificati in un editor di proprietà compatto, ma possono anche essere "convertiti" in blueprint completi semplicemente aggiungendo codice, variabili o componenti utilizzando l'editor completo di blueprint.

Level Blueprint

Un Level Blueprint è un tipo specializzato di blueprint che controlla gli eventi globali sul intero livello. Ogni livello nel progetto ha il proprio Level Blueprint creato per impostazione predefinita che può essere modificato all'interno dell'editor Unreal, tuttavia non è possibile creare nuovi Level Blueprint tramite l'interfaccia dell'editor. Gli eventi relativi al livello nel suo insieme, o istanze specifiche di attori all'interno del livello, vengono utilizzati per attivare sequenze di azioni sotto forma di chiamate alle funzioni o operazioni di controllo del flusso. Level Blueprints forniscono anche un meccanismo di controllo per lo streaming di livello e il Sequencer, nonché per vincolare gli eventi agli attori posizionati all'interno del livello.

Blueprint Interface

Un'interfaccia Blueprint è una raccolta di funzioni senza implementazione, che possono essere aggiunte ad altri Blueprint. Qualsiasi Blueprint a cui è stata aggiunta l'interfaccia è garantito per avere quelle funzioni. Le funzioni dell'interfaccia possono essere dotate di funzionalità in ciascuno dei progetti che l'hanno aggiunta. Questo è essenziale come il concetto di interfaccia nella programmazione generale, che consente a più tipi diversi di oggetti di condividere e accedere tramite un'interfaccia comune. Le interfacce Blueprint possono essere realizzate dai creatori di contenuti tramite l'editor in modo simile ad altri Blueprint, ma presentano alcune limitazioni in quanto non possono:

- Aggiungere nuove variabili;
- Modificare grafici;
- Aggiungere componenti.

Blueprint Macro Library

Una libreria di macro è un contenitore che contiene una raccolta di macro o grafici autonomi che possono essere inseriti come nodi in altri blueprint. Questi possono far risparmiare tempo in quanto possono memorizzare sequenze di nodi di uso comune complete di input e output sia per l'esecuzione che per il trasferimento dei dati. Le macro sono condivise tra tutti i grafici che vi fanno riferimento, ma vengono espanse automaticamente in grafici come se fossero un nodo compresso durante la compilazione. Ciò significa che le librerie di macro non devono essere compilate. Tuttavia, le modifiche a una macro si riflettono solo nei grafici che fanno riferimento a quella macro quando il Blueprint contenente quei grafici viene compilato.

Blueprint Utilities

Un'utilità Blueprint (o Blutility in breve), è un editor-only blueprint che può essere utilizzato per eseguire azioni dell'editor o estendere la funzionalità dell'editor. Questi possono esporre eventi senza parametri come pulsanti

nell'interfaccia utente e hanno la capacità di eseguire qualsiasi funzione esposta ai proggetti e agire sul set corrente di attori selezionati nella finestra.[25]

2.2.4. Programmazione con C++

C++ si utilizza per creare i sistemi di gioco di base su cui i progettisti possono quindi costruire o con cui creare il gameplay personalizzato per un livello o per il gioco. Il programmatore C++ lavora con un editor di testo (come Notepad++) o un IDE (Microdoft Visual Studio o Xcode) e il designer lavora nell'editor di Blueprint all'interno di Unreal Engine. L'API di gioco e le classi framework possono essere usati separatamente, ma mostrano la loro vera potenza se usati insieme per completarsi a vicenda. Significa che il motore funziona meglio quando i programmatori creano blocchi di gioco in C++ e i designer prendono quei blocchi e creano un gameplay interessante.

Come un esempio, prendiamo una classe che può essere estesa tramite Blueprint da un designer o programmatore. In questa classe creeremo alcune proprietà che il progettista può impostare e deriveremo nuovi valori da quella proprietà.

La creazione di una classe si inizia dalla scelta di una Parent Class, per esempio, Actor class. Poi si digita il nome e si sceglie una cartella dove sarà salvata la classe. Dopo aver creato una classe, la procedura guidata di Unreal Engine genererà i file e aprirà l'ambiente di sviluppo in modo che si può iniziare a modificare il codice.

```
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class AMyActor : public AActor
{
    GENERATED_BODY()

public:
    // Sets default values for this actor's properties
    AMyActor();

    // Called every frame
    virtual void Tick( float DeltaSeconds ) override;

protected:
    // Called when the game starts or when spawned
    virtual void BeginPlay() override;
};
```

Figura 33. Un esempio di una classe C++ di Unreal Engine

Le funzioni BeginPlay e Tick sono create automaticamente e sono specificate come "override". BeginPlay è un evento che informa che l'attore è entrato nel gioco in uno stato giocabile. Questo è un buon posto per avviare la logica di gioco per la classe creata. Tick viene chiamata una volta per frame con la quantità di tempo trascorso dall'ultima chiamata passata. Dentro questa funzione si può eseguire qualsiasi logica concorrente. Se queste funzioni non servono, è meglio rimuoverle per risparmiare una piccola quantità di prestazioni.

Una proprietà si esporre all'editor con lo specificatore UPROPERTY. È necessario inserire UPROPERTY(EditAnywhere) sulla riga sopra la dichiarazione di proprietà, come mostrato nella classe seguente:

[25] Unreal Engine Documentation, “Programming with C++”, n.d.

```
UCLASS()
class AMyActor : public AActor
{
    GENERATED_BODY()
public:

    UPROPERTY(EditAnywhere)
    int32 TotalDamage;

    ...
};
```

Figura 34. Un esempio di utilizzo di UPROPERTY

Se si desidera che la proprietà TotalDamage appaia in una sezione con proprietà correlate, è possibile utilizzare la funzione di categorizzazione UPROPERTY(EditAnywhere, Category="Damage"). Le funzioni BlueprintReadOnly, BlueprintReadWrite impostano il modo in cui la proprietà è esposta all'editor.

Per estendere la nostra classe con Blueprint, si seleziona la classe salvata, e si crea una nuova classe predefinita denominata Blueprint.

Durante la creazione di sistemi di gioco, i progettisti dovranno essere in grado di chiamare funzioni create da un programmatore C++. Il programmatore dovra essere in grado di chiamare le funzioni implementate in Blueprints dal codice C++. La macro UFUNCTION() gestisce l'esposizione della funzione C++ al sistema di riflessione. L'opzione BlueprintCallable lo espone alla macchina virtuale Blueprint. Ogni funzione esposta Blueprint richiede una categoria ad essere associata, in modo che il menu contestuale del tasto destro funzioni correttamente.

Classi di gioco: oggetti, attori e componenti

Esistono 4 tipi di classi principali da cui deriva per la maggior parte delle classi di gioco. Sono UObject, AActor, UActorComponent e UStruct. Si possono creare i tipi che non derivano da nessuna di queste classi, ma non partecipano alle funzionalità integrate nel motore. L'uso tipico delle classi create al di fuori della gerarchia UObject sono: integrazione di librerie di terze parti, wrapping di funzionalità specifiche del sistema operativo e così via.

Unreal Objects (UObject)

Il blocco di base nell'Unreal Engine è chiamato UObject. Questa classe, insiame a UClass, fornisce alcuni dei servizi più importanti del motore:

- Riflessione di proprietà e metodi;
- Serializzazione di proprietà;
- Raccolta dei rifiuti;
- Trovare un UObject per nome;
- Valori configurabili per le proprietà;
- Supporto di rete per proprietà e metodi.

Ogni classe che deriva da UObject ha una UClass singleton creata per essa che contiene tutti i metadati sull'istanza della classe. UObject e UClass insieme sono alla base di tutto ciò che fa un oggetto di gioco durante la sua vita. Il modo migliore per pensare alla differenza tra un UClass e un UObject è che UClass descrive come apparirà un'istanza di un UObject, quali proprietà sono disponibili per la serializzazione, il networking e così via. La maggior parte dello sviluppo del gameplay non implica la derivazione diretta da UObject, ma invece da

AActor e UActorComponent. Non è necessario conoscere i dettagli di come funzionano UClass e UObject per scrivere il codice di gioco, ma è bene sapere che questi sistemi esistono.

AActor

Un AActor è un UObject destinato a far parte dell'esperienza del gioco. Gli attori vengono inseriti in un livello da un designer o creati in fase di esecuzione tramite i sistemi di gioco. Tutti gli oggetti che possono essere inseriti in un livello si estendono da questa classe. Gli esempi includono AStaticMeshActor, ACameraActor e APointLight. Gli attori possono essere distrutti in modo esplicito tramite il codice di gioco (C++ o Blueprints) o dal meccanismo di Garbage Collection standard quando il livello proprietario viene scaricato dalla memoria. Gli attori sono responsabili dei comportamenti di alto livello degli oggetti del gioco. AActor è anche il tipo di base che può essere replicato durante il networking. Durante la replica di rete, gli attori possono anche distribuire informazioni per qualsiasi UActorComponent di cui sono proprietari.

Gli attori hanno i propri comportamenti (specializzazione attraverso l'ereditarietà), ma fungono anche da contenitori per una gerarchia di componenti dell'attore (specializzazione attraverso la composizione). Ciò avviene tramite il membro RootComponent dell'attore, che contiene un singolo USceneComponent che, a sua volta, può contenerne molti altri. Prima che un attore possa essere posizionato in un livello, deve contenere almeno un componente della scena, dal quale l'attore trarrà la sua traslazione, rotazione e scala.

Gli attori hanno una serie di eventi che vengono chiamati durante i loro cicli di vita:

- BeginPlay: Chiamato quando l'attore nasce per la prima volta durante il gioco.
- Tick: Chiamato una volta per fotogramma per eseguire il lavoro nel tempo.
- EndPlay: Chiamato quando l'oggetto lascia lo spazio del gioco.

Per gli attori che si trovano in un livello, capire il ciclo di vita è abbastanza facile: gli attori vengono caricati e nascono e alla fine il livello viene scaricato e gli attori vengono distrutti. Generare un attore è un po' più complicato che creare un normale oggetto nel gioco, perché gli attori devono essere registrati con una varietà di sistemi di runtime per soddisfare tutte le loro esigenze. È necessario impostare la posizione iniziale e la rotazione di un attore. La fisica potrebbe aver bisogno di saperlo. Il manager responsabile di dire a un attore di spuntare deve saperlo. E così via. Per questo motivo, è stato creato un metodo dedicato alla generazione di un attore, SpawnActor, un membro di UWorld. Quando l'attore si genera correttamente, il motore chiama il suo metodo BeginPlay, seguito da Tick nel frame successivo.

Una volta che un attore ha vissuto la sua vita, si può chiamare il metodo Destroy, che esegue EndPlay, consentendo allo sviluppatore di eseguire una logica personalizzata prima che l'attore vada alla Garbage Collection. Un'altra opzione per controllare per quanto tempo esiste un attore è usare il membro Lifespan. È possibile impostare una quantità di tempo nel costruttore dell'attore o con altro codice in fase di esecuzione. Una volta scaduto il tempo, l'attore avrà automaticamente chiamato Destroy su di esso.

UActorComponent

Actor Components (classe UActorComponent) hanno i propri comportamenti e sono generalmente responsabili della funzionalità condivisa tra molti tipi di attori, come la fornitura di mesh visive, effetti particellari, prospettive della telecamera e interazioni fisiche. Mentre agli attori vengono spesso assegnati obiettivi di alto livello relativi ai loro ruoli generali nel gioco, i componenti dell'attore di solito svolgono i compiti individuali che supportano quegli obiettivi di livello superiore. I componenti possono anche essere collegati ad altri componenti o possono essere il componente principale di un attore. Un Componente può essere collegato solo a

un Componente o attore padre, ma può avere molti Componenti figli collegati a se stesso. I componenti figli hanno posizione, rotazione e ridimensionamento rispetto al componente o attore padre.

UStruct

Per usare un UStruct è necessario contrassegnare lo struct con USTRUCT(). A differenza di un UObject, le istanze UStruct non vengono raccolte in modo obsoleto. Se uno sviluppatore ne crea istanze dinamiche, deve gestirne il ciclo di vita. Un UStruct dovrebbe essere un tipo di dati semplice con supporto per la riflessione UObject per la modifica all'interno dell'Unreal Editor, la manipolazione del blueprint, la serializzazione, il networking e così via.

2.2.5. Sistema di Riflessione Unreal

Unreal Engine utilizza la propria implementazione della riflessione che abilita funzionalità dinamiche come Garbage Collection, serializzazione, replica di rete e comunicazione Blueprint/C++. Queste funzionalità sono opt-in, il che significa che uno sviluppatore dovrebbe aggiungere il markup corretto ai tipi, altrimenti Unreal le ignorerà e non genererà i dati di riflessione per loro. Ecco una rapida panoramica del markup di base:

- UCLASS() - usato per dire a Unreal di generare dati di riflessione per una classe. La classe deve derivare da UObject.
- USTRUCT() - usato per dire a Unreal di generare dati di riflessione per uno struct.
- GENERATED_BODY() - Unreal Engine lo sostituisce con tutto il codice boilerplate necessario che viene generato per il tipo.
- UPROPERTY() - consente di utilizzare una variabile membro di una UCLASS o una USTRUCT come UPROPERTY. UPROPERTY ha molti usi: può consentire la replica, la serializzazione e l'accesso alla variabile da Blueprints. Sono anche usati da Garbage Collector per tenere traccia di quanti riferimenti ci sono a un Uobject
- UFUNCTION() - consente di utilizzare un metodo di una UCLASS o di una USTRUCT come UFUNCTION. Una UFUNCTION può consentire al metodo di classe di essere chiamato da Blueprints e utilizzato come RPC.

2.2.6. Gestione della memoria

Unreal Engine utilizza il sistema di riflessione per implementare un sistema di Garbage Collection. Con la Garbage Collection uno sviluppatore non deve gestire manualmente l'eliminazione delle istanze UObject, dovrebbe solo mantenere validi riferimenti ad esse. Le classi devono derivare da UObject per essere abilitate per la Garbage Collection.

Nel Garbage Collector c'è un concetto chiamato root set. Il root set è un elenco di oggetti che il collector sa non verranno mai raccolti. Un oggetto non verrà sottoposto a Garbage Collection finché esiste un percorso di riferimenti da un oggetto nel root set all'oggetto in questione. Se non esiste alcun percorso di questo tipo per il root set per un oggetto, esso viene chiamato irraggiungibile è verrà raccolto (eliminato) alla successiva esecuzione del Garbage Collector. Il motore grafico esegue il Garbage Collector a determinati intervalli.

Gli attori di solito non vengono raccolti con Garbage Collector, a parte durante l'arresto di un livello. Una volta generati, lo sviluppatore dovrebbe chiamare manualmente Destroy su di loro per rimuoverli dal livello senza terminare il livello. Gli attori verranno rimossi immediatamente dal gioco e quindi completamente eliminati durante la successiva fase di Garbage Collection.[26]

2.3. Le differenze tra Unity e Unreal Engine

Come abbiamo visto prima, Unity e Unreal Engine sono entrambi motori di gioco molto potenti, sono i componenti chiave per lo sviluppo del gioco. Forniscono un ambiente di sviluppo software che aiuta i designer a creare i loro videogiochi. Entrambe le piattaforme consentono di sviluppare facilmente giochi per PC, console (come Xbox, Wii e PS4) e giochi per dispositivi mobili con sistemi iOS e Android.
La prima differenza fondamentale tra Unity e Unreal Engine è il loro linguaggio di programmazione nativo. Unity usa C# sia nell'editor Unity che nei plug-in aggiuntivi. Unreal Engine utilizza C++, e quando crei il codice di gioco stesso, si utilizza una combinazione di Blueprint (un linguaggio proprietario unico per i prodotti Epic) e C++.

Categoria chiave	Unity	Unreal Engine
Grafica	Rendering fisico, illuminazione globale, luci volumetriche (richiede l'installazione di un plugin), post processing	Rendering fisico, illuminazione globale, luci volumetriche out-of-the-box (pronti all'uso, predefiniti), post processing, editor di materirali
Caratteristiche uniche	Ricco supporto 2D	AI, supporto di rete
Audienza target	Principalmente Indie, Coders, Mobile games	AAA-Game Studios, Indies, Artists
Coding	C#, Prefab, Bolt	C++, Blueprints
Prestazioni	Non scala bene	Ha il supporto per l'esecuzione distribuita (acceleratore Incredibuild)
Facilità d'uso	Nel complesso, gli utenti hanno riscontrato che Unity è leggermente più facile da usare, grazie al suo linguaggio nativo C# che dovrebbe essere relativamente familiare a tutti gli sviluppatori, e al suo layout generale dell'area di lavoro.	
Qualità degli effetti visivi (VFX)		Sebbene entrambe le piattaforme producano VFX di alta qualità, la maggior parte degli utenti ha scoperto che Unreal Engine ha un leggero

[26] Unreal Engine Documentation, “Programming with C++”, n.d.

		vantaggio rispetto a Unity nella qualità dei suoi effetti visivi. Può creare visualizzazioni fotorealistiche che immergono i giocatori e consentono loro di viaggiare liberamente in un nuovo mondo e incorporano risorse di alta qualità da una varietà di fonti.
Rendering		Simile al suo leggero vantaggio con la qualità grafica, la qualità e la velocità di rendering su Unreal Engine superano leggermente Unity in questa categoria. Produce velocità di rendering leader del settore e gli utenti sono costantemente soddisfatti della qualità della grafica renderizzata.
Animazione		Unreal Engine ha potenti capacità di rendering e gli effetti visivi di prim'ordine. È anche il favorito tra gli utenti per la qualità dei suoi strumenti di animazione e per i rendering delle animazioni.
Collaborazione di squadra	Unity detiene una quota di mercato maggiore del 48% rispetto al 13% di Unreal Engine. Ciò significa che più persone usano Unity rispetto a Unreal Engine, il che significa che è più facile trovare più collaboratori di Unity rispetto a quelli di Unreal Engine.	
Scripting (entrambe)	Entrambe le piattaforme scripting che ti script per il tuo gioco Quando si tratta di piattaforme ti forniranno Necessarie per scrivere senza interruzioni.	hanno ottimi strumenti di permetteranno di creare dall'inizio alla fine. scripting, entrambe le tutte le funzionalità il tuo gioco in modo rapido e
Qualità del supporto (entrambe)	Entrambe le piattaforme ore su 24, 7 giorni su 7, i problemi che potresti Indipendentemente dalla	Hanno servizi di supporto 24 che possono aiutarti con tutti incontrare. piattaforma che scegli, avrai

	un team che ti supporta continuino a funzionare	per assicurarti che le cose senza intoppi.
G2 Rating (entrambe)	G2, una delle fonti più affidabili per le recensioni di software, ha valutato sia Unity che Unreal Engine come un solido 4.5/5 stelle, evidenziando ulteriormente quanto queste piattaforme siano testa a testa quando si tratta di popolarità e qualità tra utenti.	
Capterra Rating		Capterra ha assegnato all'Unreal Engine 4.8/5 stelle, a cui Unity è arrivato poco al di sotto con 4.7/5 stelle.
Prezzo	Unity ha una versione gratuita, ma per sbloccare tutte le funzionalità, l'utente deve eseguire l'aggiornamento alla versione Pro, disponibile per una quota di abbonamento mensile di 150 dollari al mese.	Unreal Engine ha una modello di prezzo unico. Il software stesso è gratuito, ma dopo aver rilasciato il gioco, Unreal Engine ha diritto a una commissione di royalty del 5% su tutte le vendite di giochi dopo i primi 3000 dollari per prodotto.

Per chi è più adatto Unity?

Unity è un'ottima piattaforma per i designer di giochi indie. Con il lunguaggio C# nativo e un'ampia comunità di altri sviluppatori e designer, Unity è un'ottima piattaforma per i designer indipendenti che vogliono iniziare a creare subito e non vogliono dover alla piattaforma una royalty dai loro giochi sul back-end.

Unity è più adatto allo sviluppo mobile, i giochi creati sono leggeri, l'app Unity può essere incorporata in un'altra app. Unity supporta ogni possibile metodo di monetizzazione mobile (ads, acquisto in-app). Unity fornisce l'ampio supporto per i giochi 2D: illuminazione 2D, strumenti di animazione, sprites, tilemaps.[27]

Per chi è più adatto Unreal Engine?

Unreal Engine è incentrato sulla grafica ottimizzata e sulle velocità di rendering fulminee, che lo rendono perfetto per gli sviluppatori indipendenti che desiderano quella qualità extra-fine sui loro giochi e non si preoccupano di dover pagare le royalty sul back-end.[28]

Unreal Engine è più facile da iniziare per i non programmatori che vogliono avere possibilità di creare uno script di qualsiasi cosa, perché ha lo strumento di scripting visivo out-of-the-box, che è stato creato per supportare sviluppatori indie, insieme con tutti strumenti che lavorano out-of-box. Il rendering in Unreal si concentra sul rendering fisico e sulla telecamera di prospettiva. Esistono molti strumenti out-of-the-box per lo

[27] Unreal Community, “Unreal vs Unity”, n.d.

[28] Evelyn Trainor-Fogleman, Evercast, “Unity vs Unreal Engine: Game engine comparison guide for 2021”, n.d

sviluppo dei personaggi, animazioni scheletriche, non limitate solo a umanoidi, rendering di capelli, pellicce e shader degli occhi.

Unreal Engine è più adatto allo sviluppo di giochi di combattimento / azione grazie al sistema di animazione scheletrico esteso. Un nuovo sistema di Control Rig consente la creazione dell'animazione procedurale, keyframe animation attraverso il sequencer e bake del risultato nell'asset di animazione.

3. Analisi della percezione umana di modelli 3D realizzati con vari metodi

Nell'ambito di questa tesi, è stato condotto uno studio il cui scopo era analizzare quanto sono fotorealistici i modelli 3D di esseri umani creati utilizzando diversi metodi, quali sono i criteri di fotorealismo scelti dalle persone intervistate e che cosa potrebbe essere migliorato in futuro.

E' stato intervistato un gruppo di persone tra 16 e 60 anni, 100% di loro non hanno nessuna esperienza in una sfera inerente la computer grafica. 50% delle persone intervistate erano gli uomini, e 50% - le donne. 53% delle persone hanno età tra 16 e 30 anni, 26% - tra 31 e 45, e 21% tra 46 e 60 anni.

Le prime quattro domande sono state fatte in seguente modo: al rispondente è stata mostrata una serie di immagine, a lui o a lei è stato chiesto scegliere quelle immagini che secondo lui o lei sembrano le fotografie delle persone vere. Le immagini che rappresentano i render di modelli 3D sono state create con diversi software: MetaHuman Creator, Daz3D, ZBrush. Le foto delle persone vere sono state scelte da 37,29% di rispondenti; 29,91% hanno scambiato per una foto i modelli 3D create con Daz3D; 13,10% hanno detto che le foto delle persone vere sono le immagini che rapresentano i modelli 3D creati con ZBrush; e 19,70% hanno detto che i modelli 3D creati con MetaHuman Creator sono le foto delle persone vere.

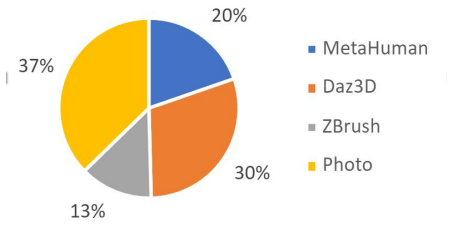

Figura 35. La percentuale delle immagini scelte come le foto delle persone vere

Le prossime due domande dimostravano agli intervistati solamente le immagini che rappresentano i modelli 3D. E' stato chiesto quanto fotorealistiche sono le immagini dimostrate secondo a intervistato o intervistata e perché. La valutazione media ha ottenuto all'incirca lo stesso punteggio su una scala da uno (poco fotorealistico) a cinque (molto fotorealistico) per tutti i metodi con quale possono essere creati modelli 3D di esseri umani. Secondo gli intervistati, le immagini migliori possono essere ottenuti utilizzando ZBrush (3,56), secondo strumento migliore potrebbe essere MetaHuman Creator (3,45), scansione 3D dimostra risultati simili a MetaHuman Creator (3,42), e Daz3D è stato scelto come quello che permette di creare immagini meno fotorealistiche rispetto a tutti altri strumenti considerati (3,05).

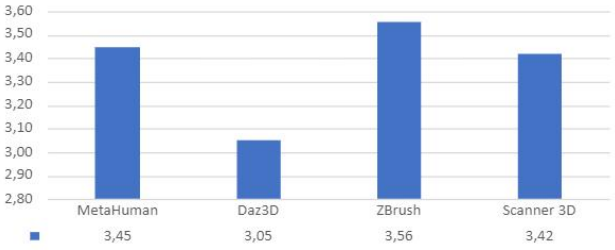

Figura 36. Quanto fotorealistiche l'immagini create con diversi metodi

Le proprietà principali che hanno influenzato l'opinione delle persone intervistate sono gli impostazioni della luce e effetti di riflessione di essa sulla pelle dei personaggi, la texture della pelle, presenza di dettagli come rughe, lentiggini e piccole imperfezioni. Un altro dettaglio che influenza molto sulla percezione del fotorealismo di personaggi è lo sguardo che, in molti casi, potrebbe essere più naturale.

Nella prossima sezione agli intervistati è stato chiesto di guardare tre video. Il primo video dimostra l'animazione del viso e lipsync di un personaggio creato con MetaHuman Creator. La valutazione media di fotorealismo del personaggio è stata 3,84 su una scala da 1 a 5, dove 1 significa che il personaggio è poco fotorealistico, non ammette dubbi che è stato creato con un computer, e 5 significa che il personaggio è molto fotorealistico, sembra una persona vera. In seguito, è stato chiesto di valutare diversi proprietà di un modello 3D che potrebbero influenzare o meno la percezione di fotorealismo. I risultati hanno dimostrato che tre proprietà che influenzano maggiormente la percezione sono colore e dettagli della pelle (3,87), struttura dei capelli (3,87) e colore e dettagli degli occhi (3,87). Un'altra proprietà importante è l'illuminazione ambientale (3,26) che è stata già mentionata prima. L'illuminazione è un parametro importante su ogni scena tredimensionale che include personaggi umanoidi o meno. Quattro proprietà specificate prima fanno credere alle persone che l'immaggine dimostrata rappresenta un essere umano vero. Ultime due proprietà prese in considerazione sono espressioni facciali (2,97) e movimenti della testa (2,66) che nel caso del primo video dimostrato influenzano negativamente la percezione di fotorealismo. Secondo le persone intervistate, si potrebbe migliorare i movimenti degli occhi e della testa del personaggio rendendoli più fluidi e più coerenti tra di loro, le espressioni facciali dovrebbero essere più morbidi per sembrare più realistici. Si potrebbe lavorare ancora con le texture della pelle aggiungendo più dettagli, imperfezioni e rughe.

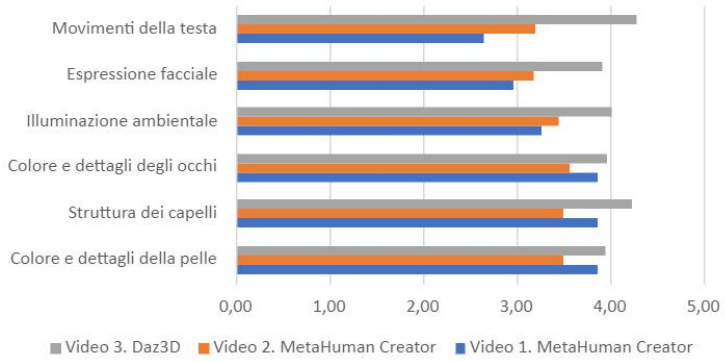

Figura 37. I dettagli che influenzano la percezione di fotorealismo

L'impressione personale di ogni intervistato ha rilevato che nonostante alcune imperfezioni il personaggio è piaciuto molto a 52,63% delle persone. Secondo l'opinione di altri 23,68% il personaggio non è abbastanza fotorealistico, ma è piaciuto a loro. 13,16% non hanno espresso nessuna preferenza rimanendo neutrali. A 5,26% delle persone non è piaciuto personaggio perché li spaventa. In questo caso possiamo osservare l'effetto chiamato "Uncanny valley" o "zona perturbante". Questo termine è stato utilizzato in robotica per descrivere situazioni quando le persone avevano la sensazione di familiarità e di piacevolezza per un robot antropomorfi che aumenta al crescere della somiglianza di questi robot con un essere umano, ma solo fino ad un certo punto

in cui elevati livelli di realismo producono un brusco calo delle reazioni emotive positive, le persone iniziano sentire disgusto e paura verso questi robot.[29]

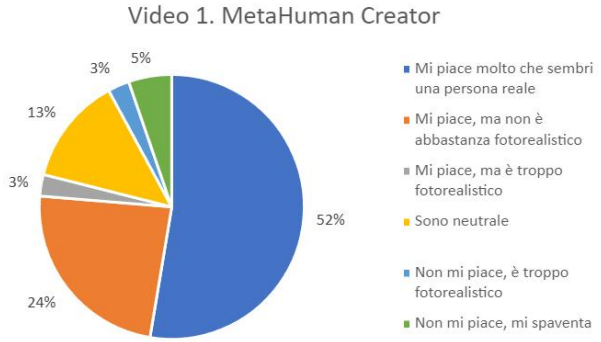

Figura 38. L'impressione degli intervistati riguardo ai modelli visti nel video 1

Il secondo video rappresenta un preview di un videogioco dove sono stati usati modelli creati con MetaHuman Creator. Questo video dimostra diversi animazioni del viso e del corpo, utilizzo di lipsync, utilizzo di diversi impostazioni dell'illuminazione ambientale. La valutazione media di fotorealismo del personaggio è stata 3,87 su una scala da 1 a 5, dove 1 significa che il personaggio è poco fotorealistico, non ammette dubbi che è stato creato con un computer, e 5 significa che il personaggio è molto fotorealistico, sembra una persona vera. In seguito, è stato chiesto di valutare diversi proprietà di un modello 3D che potrebbero influenzare o meno la percezione di fotorealismo. I risultati hanno dimostrato che in questo caso la proprietà che influenza la percezione più positivamente è stata colore e detagli degli occhi (3,58), altre due proprietà importanti sono state colore e dettagli della pelle (3,50) e Struttura dei capelli (3,50). Le proprietà, che secondo gli intervistati, sono meno importanti sono illuminazione ambientale (3,45), espressioni facciali (3,18) e movimenti della testa (3,21). L'impressione personale di ogni intervistato ha rilevato che i personaggi presentati nel video sono piaciuti molto a 55,26% dei rispondenti, non sono abbastanza realistici secondo altri 18,42% delle persone intervistate, e 18,42% delle persone hanno scelto di restare neutrali. 7,89% delle persone hanno detto che i personaggi sono troppo fotorealistici, secondo loro in computer grafica dovrebbe rimanere una certa percentuale di imperfezione per poter distinguere le immagini creati con computer da quelle vere, perché altrimenti non possiamo esseri sicuri di tutto ciò che vediamo. Le immagini e video troppo fotorealistici potrebbero causare situazioni pericolosi, quando le informazioni false sono presentate come verità.

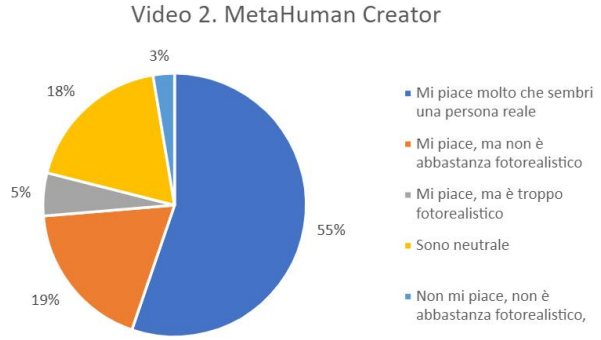

Figura 39. L'impressione degli intervistati riguardo ai modelli visti nel video 2

[29] Wikipedia. L'enciclopedia libera, "Uncanny valley", n.d., <https://it.wikipedia.org/wiki/Uncanny_valley>

Il terzo video rappresenta uno spot pubblicitario koreano realizzato utilizzando i modelli creati con Daz3D. In questo video sono utilizzati poco le espressioni facciali, non è utilizzato lipsync e tutte le riprese sono fatte da mezza busto o da lontano. La valutazione media di fotorealismo del personaggio è stata 4,47 su una scala da 1 a 5, dove 1 significa che il personaggio è poco fotorealistico, non ammette dubbi che è stato creato con un computer, e 5 significa che il personaggio è molto fotorealistico, sembra una persona vera. In seguito, è stato chiesto di valutare diversi proprietà di un modello 3D che potrebbero influenzare o meno la percezione di fotorealismo. I risultati hanno dimostrato che in questo caso la proprietà che influenza la percezione più positivamente è stata movimenti della testa e del corpo (4,29), altre proprietà hanno ricevuto le seguenti valutazioni: struttura dei capelli (4,24), illuminazione ambientale (4,03), colore e dettagli degli occhi (3,97), colore e dettagli della pelle (3,95) ed espressioni facciali (3,92). Il video è piaciuto a 71,05% delle persone intervistate, altri 10,53% hanno preferito rimanere neutrali, a 7,89% non è piaciuto il personaggio, perché non è abbastanza fotorealistico, 5,26% hanno detto che il personaggio è piaciuto a loro, ma non e abbastanza fotorealistico, e altri 5,26% hanno detto che il personaggio è troppo fotorealistico, ma è piaciuto a loro.

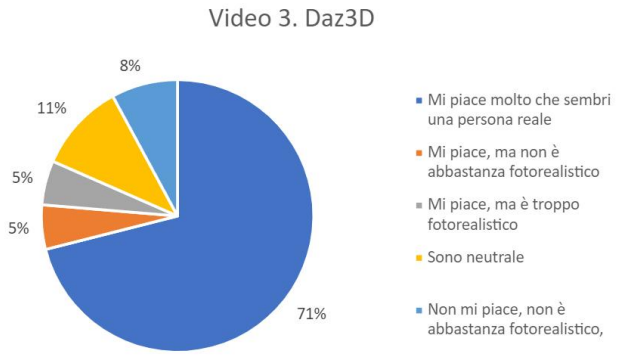

Figura 40. L'impressione degli intervistati riguardo ai modelli visti nel video 3

L'ultima sezione del sondaggio è stata dedicata al confronto di diversi metodi utilizzati per creare i modelli 3D antropomorfi. 52,63% delle persone hanno scelto ZBrush perché i modelli creati con questo software hanno un livello di dettaglio superiore a quello di altri modelli. 21,05% hanno scelto MetaHuman Creator perché le texture della pelle e degli occhi lavorano meglio con l'illuminazione ambientale rispetto a altri modelli. 10,53% hanno scelto Daz3D perché l'animazione del personaggio creato con questo software lo fa sembrare quasi una persona vera. 15,79% hanno dato la preferenza a modelli creati utilizzando lo scanner 3D, perché essi hanno l'espressione facciale più naturale.

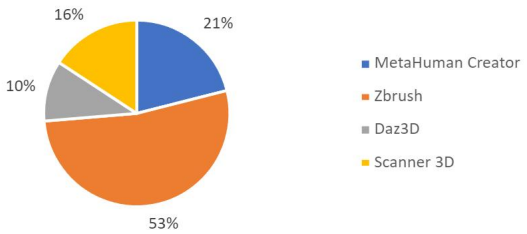

Figura 41. Le preferenze delle persone intervistate per ogni metodo della creazione di modelli 3D analizzato

Conclusioni

Il confronto tra diversi metodi utilizzati per creare modelli 3D antropomorfi ha rilevato benefici e svantaggi di ogni metodo rispetto agli altri, così come le preferenze di un numero di persone intervistate per un metodo rispetto ad un altro. Sono stati analizzati i seguenti metodi per la realizzazione di modelli 3D che rappresentano gli esseri umani: sviluppo di un modello tridimensionale partendo da zero o da una base mesh utilizzando un programma di modellazione 3D, texturizzazione e animazione come Autodesk Maya, ZBrush o Blender; scansione 3D delle persone vere; utilizzo di un'applicazione cloud-based MetaHuman Creator; utilizzo di un editor di modelli tridimensionali Daz3D e altri programmi. Il sondaggio ha mostrato che la maggioranza degli intervistati preferiva modelli 3D creati con ZBrush, questi modelli hanno un elevato numero di dettagli, che permettono raggiungere la massima somiglianza con una persona vera. Sviluppo di un modello 3D con un programma di modellazione 3D consente modificare ogni dettaglio del modello senza alcuni vincoli, ma d'altra parte i tempi necessari per la creazione di un modello di alta qualità sono molto elevati, possono raggiungere settimane o addirittura mesi. Uno scanner 3D permette creare un modello tridimensionale in pochi minuti generando una mesh dalle fotogramme riprese con numerose telecamere, un modello creato in questo modo ha notevole assomiglianza con una persona scanarizzata, ma per utilizzare esso nei ambienti che richiedono sviluppo di animazioni o adattamento di diversi LOD, necessitano ulteriori lavori con la mesh. Gli editor di modelli tridimensionali come Daz3D permettono creare un modello umanoide in pochi minuti, questi modelli possono essere importati in altri software 3D, sono già dotati di un scheletro necessario per lo sviluppo di animazioni. Uno svantaggio di questo metodo consiste nel fatto che per ottenere fotorealismo è necessario modificare le texture del modello che potrebbe essere un'operazione non banale per le persone che non hanno conoscenze artistiche.

Lo strumento MetaHuman Creator riunisce vantaggi di tutti metodi specificati sopra in un'unica soluzione. Utilizzando la cloud-based app qualsiasi persona potrebbe creare un modello 3D di essere umano con le texture di alta qualità, uno sceletro per lo sviluppo di animazioni e impostazioni di diversi LOD per utilizzo del modello nei diversi ambienti. Questo modello può essere importato in Autodesk Maya e in Unreal Engine per utilizzare tutti gli strumenti disponibili. Esso potrebbe essere un vantaggio per le persone che sono abituati lavorare con questi due software, invece per altri potrebbe creare difficoltà il fatto che la struttura complessa del modello MetaHuman è contenuta nei file *.UASSET di Unreal Editor e non può essere esportata pienamente in un clic. Sarebbe necessario studiare la struttura di Unreal Blueprint e altri file per poter riscriverli utilizzando linguaggi supportati da altri software.

Oltre ad adattamento dei modelli 3D a diversi motori grafici e software di modellazione 3D, sarebbero necessari ulteriori miglioramenti delle prestazioni dei movimenti degli occhi e della testa del personaggio, rendendoli più fluidi e più coerenti tra di loro. Le espressioni facciali di un personaggio virtuale dovrebbero essere più morbide per sembrare più realistiche. Si potrebbe lavorare ancora con le texture della pelle aggiungendo più dettagli, imperfezioni e rughe.

Da altra parte, in computer grafica dovrebbe rimanere una certa percentuale di imperfezione per poter distinguere le immagini creati con computer da quelle vere, perché altrimenti non possiamo essere sicuri di tutto ciò che vediamo. Le immagini e video troppo fotorealistici potrebbero causare situazioni pericolosi, quando le informazioni false sono presentate come verità.

Bibliografia e Sitografia

[1] Unreal Engine Documentation, “MetaHumans”, n.d., <https://docs.unrealengine.com/4.27/en-US/Resources/Showcases/MetaHumans/>

[2] sociamix, YouTube, “Modeling a character BaseMesh in Blender (Tutorial)”, 20/12/2019, <https://www.youtube.com/watch?v=WlaMfIgS2ns&ab_channel=sociamix>

[3] Hadi Karimi, Portfolio, “Ludwig van Beethoven”, n.d., <https://hadikarimi.com/portfolio/ludwig-van-beethoven>

[4] Swann Rack, Holocreators, “How to 3D-scan people or animals?”, 12/11/2022, <https://holocreators.com/blog/how-to-3d-scan-people-and-animals/>

[5] Zaccardi Giuseppe Massimiliano (2020), "3D Scanner Inspection and Shape Correction of SLS parts through Reverse Engineering compensation", tesi di laurea, Politecnico di Torino, pp. 21-22

[6] Erez Zukerman, PCWorld, “DAZ Studio”, 09/08/2012, <https://www.pcworld.com/article/460388/daz_studio.html>

[7] Lauren Michele Jackson, The New Yorker, “Shudu Gram is a white man’s digital projection of real-life black womanhood”, 04/05/2018, <https://www.newyorker.com/culture/culture-desk/shudu-gram-is-a-white-mans-digital-projection-of-real-life-black-womanhood>

[8] M.Bastioni, S.Re, S.Misra. Proceedings of the 1st Bangalore Annual Compute Conference, Compute 2008, 2008. "Ideas and methods for modelling 3D human figures: the principal algorithms used by MakeHuman and their implementation in a new approach to parametric modeling".

[9] Manuel Bastioni, Joel Palmius, Jonas Hauquier, 80.lv, “Interview with MakeHuman: The Future of 3D-character creation tools”, 18/02/2015, <https://80.lv/articles/makehuman-interview/>

[10] eche101, MakeHuman Community, “Trying to make a realistic face”, 02/08/2019, <http://www.makehumancommunity.org/forum/viewtopic.php?f=15&t=17969>

[11] Joe Brodkin, Dice, “How Unity3D Became a Game-Development Beast”, 03/06/2013, <https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/>

[12] Marie Dealessandri, gameindustry.biz, “What is the best game engine: is Unity right for you?”, 16/01/2020, <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you>

[13] Jeff Grubb, VentureBeat, “Unity 5.6 launches with support for Vulkan graphics, Nintendo Switch, and more”, 31/03/2017, <https://venturebeat.com/2017/03/31/unity-5-6-launches-with-support-for-vulkan-graphics-nintendo-switch-and-more/>

[14] Marisa Graves, Richard Barnes, BusinessWire, “Unity Introduces Unity Forma - An Automotive and Retail Solution Tool for the Creation and Delivery of Custom Real-Time 3D Marketing Content”, 09/12/2020, <https://www.businesswire.com/news/home/20201209005406/en/Unity-Introduces-Unity-Forma---An-Automotive-and-Retail-Solution-Tool-for-the-Creation-and-Delivery-of-Custom-Real-Time-3D-Marketing-Content>

[15] Unity Documentation, “Unity Forma”, n.d., <https://unity.com/products/unity-forma>

[16] Unity Documentation, “Getting Started with AR development in Unity”, n.d., <https://docs.unity3d.com/Manual/AROverview.html>

[17] RogoDigital Documentation, “LipSync Pro”, n.d., <https://rogodigital.gitbook.io/lipsync-pro/>
[18] Chris Plante, Polygon, “Better with age: A history of Epic Games”, 01/10/2012, <https://www.polygon.com/2012/10/1/3438196/better-with-age-a-history-of-epic-games>
[19] David Lightbown, Gamasutra, “Classic Tools Retrospective: Tim Sweeney on the first version of the Unreal Editor”, 01/09/2018, <https://web.archive.org/web/20180823012812/https://www.gamasutra.com/blogs/DavidLightbown/20180109/309414/Classic_Tools_Retrospective_Tim_Sweeney_on_the_first_version_of_the_Unreal_Editor.php>
[20] Evelyn Trainor-Fogleman, Evercast, “Unity vs Unreal Engine: Game engine comparison guide for 2021”, n.d., <https://www.evercast.us/blog/unity-vs-unreal-engine>
[21] Unreal Community, “Unreal vs Unity”, n.d., <https://unrealcommunity.wiki/differences-between-unity-and-unreal-b2c4rqwm>
[22] iD Tech, “C++ vs. Blueprints: pros and cons, which should be used, and when?”, 05/05/2020, <https://www.idtech.com/blog/c-vs-blueprints-differences>
[23] Unreal Engine Documentation, “Blueprint Overview”, n.d., <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/Overview/>
[24] Unreal Engine Documentation, “Programming with C++”, n.d., <https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/>
[25] Wikipedia. L’enciclopedia libera, “Uncanny valley”, n.d., <https://it.wikipedia.org/wiki/Uncanny_valley>
[26] Unreal Engine Marketplace, “Ynnk Voice Lipsync”, 23/06/2021, <https://www.unrealengine.com/marketplace/en-US/product/ynnk-voice-lipsync>
[27] Oculus For Developers, “Oculus Lipsync for Unreal Engine”, n.d., <https://developer.oculus.com/documentation/unreal/audio-ovrlipsync-unreal/?locale=it_IT>