



UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria
Corso di Laurea in Ingegneria Informatica

Tesi Di Laurea

Analisi, progettazione e implementazione
di Metarace: un gioco 3D nel Metaverso

Laureando
Emanuele Pietropaolo
Matricola 513489

Relatore
Prof. Franco Milicchio

Anno Accademico 2021/2022

Ringraziamenti

Grazie a tutti

Introduzione

Indice

Introduzione	iii
Indice	iv
1 Background tecnologico	1
1.1 Il Metaverso	1
1.1.1 Storia del metaverso	2
1.1.2 Stato dell'arte - Building the Metaverse	6
1.2 Il motore grafico Unreal Engine 5	9
1.3 Il software di modellazione Blender	13
2 Il core di Metarace	16
2.1 Metarace	16
2.2 La meccanica di base	17
2.2.1 L'algoritmo per il movimento dei cavalli	17
2.3 Animazione dei cavalli	23
2.3.1 Animazione del galoppo creata con Blender	25
2.4 Progettazione del mondo 3D	26
2.4.1 Unreal Engine	26
2.4.2 I modelli 3D	26
2.5 Analisi del software	27
2.5.1 Modello degli eventi	27
3 Fase di Progettazione	28

3.1	Scelte di progetto guidate dalla flessibilità	28
3.2	Architettura Client - Server	28
3.3	Programmazione ad Eventi (EDP)	28
3.4	Model - View - Controller (MVC)	28
4	Fase di Implementazione	29
4.1	Cosa ho fatto con Unreal Engine 5	29
4.2	Cosa ho fatto con Blender	29
4.3	Cosa ho fatto in C++	29
5	Risultati	30
5.1	Demo di gioco	30
5.2	Interazione utente	30
5.3	Design	30
5.4	Esempi	30
Conclusioni e sviluppi futuri		31
Elenco delle figure		32
Elenco degli algoritmi		33
Bibliografia		34

Capitolo 1

Background tecnologico

1.1 Il Metaverso

Il concetto di Metaverso nasce nella letteratura di fantascienza, infatti il termine fu coniato da Neal Stephenson nel suo libro *Snow Crash* del 1992. Il termine descriveva uno spazio tridimensionale dove la realtà si univa con un mondo virtuale costantemente attivo. Nonostante questo concetto non sia recente, saper dare una definizione di cosa è il Metaverso genera molte difficoltà, infatti è una tecnologia che per la maggior parte ancora non esiste e sebbene riusciamo a ragionare su esperienze tecnologiche future siamo ancora lontani dal renderla possibile. Ancora non possiamo sapere quali caratteristiche saranno più importanti e che tipo di dinamiche guideranno la sua formazione, come negli anni '80 era difficile descrivere cosa sarebbe stato internet nel 2022, allo stesso modo è difficile saper descrivere il Metaverso. Ad oggi si può dire che il Metaverso è il nuovo principale obiettivo delle grandi compagnie di tecnologia mondiali, come Facebook - non a caso rinominata *Meta* - e Epic Games - azienda proprietaria del motore grafico Unreal Engine e di Fortnite, il videogioco che ad oggi viene considerato la piattaforma più vicina al Metaverso che sia stata fatta.

Matthew Ball, CEO di Epyllion e scrittore di *The Metaverse and How it Will Revolutionize Everything*, lo definisce in questo modo [1]:

Io definisco il metaverso come un network ampiamente scalabile e interoperabile di mondi virtuali 3D renderizzati in tempo reale che possono essere

vissuti, in modo sincrono e persistente, da un numero infinito di user effettivi, ciascuno con un senso di presenza individuale, supportando al contemporaneo continuo di dati quali cronologia, identità, comunicazione, pagamenti, diritti e oggetti.

Secondo Matthew Ball il Metaverso è quindi una combinazione di molte tecnologie diverse che collaborano per costruire un'esperienza continua e persistente. È l'unione di mondi virtuali, tecnologie - quali visori per la realtà virtuale, dispositivi indossabili e camere a proiezione 3D - e internet. È una nuova era della tecnologia che verrà costruita iteramente e lentamente al di sopra delle infrastrutture e dei protocolli esistenti che verranno migliorati o sostituiti in base alle esigenze. Un ruolo fondamentale lo avranno le piattaforme virtuali, esse infatti daranno effettivamente vita ai mondi virtuali in cui le persone potranno entrare. Alcune di queste hanno scopi puramente di intrattenimento - come Roblox, The Legend of Zelda o Minecraft - altri hanno intenti accademici e professionali - come Osso VR o come i simulatori di volo per l'addestramento di piloti.

1.1.1 Storia del metaverso

Sebbene il termine fu coniato nel 1992, il concetto di Metaverso affonda le radici nella letteratura Cyberpunk. Tale letteratura comprende romanzi come *True Names* di Vernor Vinge del 1981, che descrive quello che può essere considerato il primo esempio di cyber-spazio, e *Neuromancer* di William Gibson nel 1985, che invece descrive un cyber-spazio dalle caratteristiche molto simili al Metaverso che intendiamo oggi.

Il primo dei due libri è particolarmente importante perché è citato come fonte di ispirazione per il primo gioco nel Metaverso: Habitat [11].

1.1.1.1 Habitat - la prima implementazione di Cyber-spazio

Habitat viene definito dai creatori un *ambiente virtuale online multigiocatore* [11], ogni utente utilizzava il proprio Personal Computer come frontend comunicando su un network a commutazione di pacchetto con un sistema back-end centralizzato. La prima versione di Habitat era stata sviluppata per il Commodore 64 nel 1985 e non era possibile avere stanze virtuali popolose né grafica 3D a causa delle limitate risorse che il

modello offriva. Nonostante questo, Habitat rese possibile per la prima volta a persone da tutto il mondo di incontrarsi in uno spazio virtuale. Gli utenti avevano una rappresentazione virtuale di se stessi in terza persona, questa rappresentazione venne chiamata avatar, come nel romanzo *True Names*. L'utente, attraverso il proprio avatar, aveva la possibilità di interagire con oggetti, parlare con altri avatar attraverso una chat che appariva a schermo in stile "word balloon" e muoversi nel mondo di Habitat composto da un grande numero di posizioni che venivano chiamate *Regioni*.



Figura 1.1: Una tipica scena in Habitat.

Habitat fu un importante progetto che fece capire agli sviluppatori le difficoltà nell'approciarsi alla creazione di un ambiente virtuale multigiocatore e lasciarono in eredità un testo con le lezioni principali che impararono. [11]

1.1.1.2 Second Life

Un altro importante esempio di cyber-spazio è Second Life, una piattaforma online lanciata nel 2003 dalla società Linden Lab. Second Life è un ambiente virtuale online multigiocatore 3D e gli utenti sono rappresentati digitalmente attraverso un avatar tridimensionale. Attraverso gli avatar gli utenti hanno la possibilità di socializzare con altri utenti, sia attraverso chat testuale che vocale, esplorare il mondo virtuale, composto da migliaia di regioni chiamate *Sim* e partecipare ad attività di vario genere come concerti, raduni e lezioni e molto altro. La particolarità di questa piattaforma, che la distingue da altri videogiochi 3D, è che il contenuto del mondo di gioco viene interamente creato dai giocatori e non c'è un obiettivo da perseguire né una storia. Inoltre Second Life possiede una sua economia interna e un token virtuale a circuito chiuso chiamato *Linden Dollar L\$*. Questa valuta non ha valore monetario ma può essere

scambiata con Linden Lab per un corrispettivo in dollari. Essa può essere usata per comprare, vendere, affittare o commerciare beni e servizi con altri giocatori all'interno del mondo di gioco.

In Second Life per la prima volta brand e organizzazioni parteciparono alla realizzazione di oggetti ed eventi nel mondo virtuale portando il gioco ad evolversi e distaccarsi dall'essere una pura esperienza d'intrattenimento. Brand come Adidas, Calvin Klein e Lacoste lanciarono linee di vestiti indossabili dagli avatar dei giocatori [8] mentre alcune università hanno usato Second Life con obiettivi educativi e formativi, incluse l'Università di Harvard e di Oxford [13] ma anche alcune italiane come le università di Milano, di Torino, di Salerno e di altre città. [4, 16] Un altro evento importante fu lo sciopero dei lavoratori IBM organizzato su Second Life nel 2007, durante questo sciopero migliaia di avatar contemporaneamente si radunarono per protesta sulla Sim dell'azienda. [16]

1.1.1.3 I visori per la realtà aumentata e virtuale

Un importante contributo alla costruzione del Metaverso lo dobbiamo all'avanzamento tecnologico dei dispositivi per la realtà aumentata (AR) e per la realtà virtuale (VR). Questa tecnologia ha accompagnato il concetto del Metaverso sin dai suoi albori.

I visori per la realtà aumentata e per la realtà virtuale sono un avanzamento di un dispositivo ottico più antico, lo stereoscopio. Inventato nella prima metà dell'800, lo stereoscopio simula la tridimensionalità del mondo reale per come viene percepita dagli occhi umani. Infatti gli occhi umani trasmettono al cervello due immagini della stessa scena con due punti di osservazione leggermente diversi. Il cervello sovrapponendo le due immagini riesce a valutare la distanza degli oggetti: più un oggetto è scostato nelle due immagini più esso viene percepito come vicino, al contrario minore lo scostamento, maggiore è la distanza percepita [17]. Lo stereoscopio permette di far vedere a ciascun occhio una tra due immagini molto simili tra loro, realizzate appositamente per essere percepite dal cervello umano come se fosse un singolo punto di vista reale. Questo conferisce al soggetto la tridimensionalità desiderata.

Il primo a sfruttare questo principio per immergersi in una realtà simulata fu Ivan Sutherland nel 1968. Insieme ai suoi studenti, egli creò il primo sistema di realtà virtuale con visore che permetteva di osservare un ambiente virtuale renderizzato in wire-frame

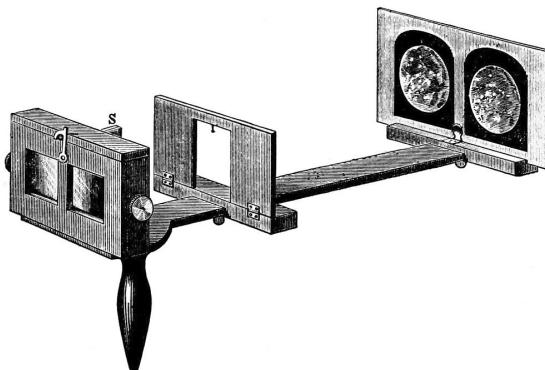


Figura 1.2: Stereoscopio statunitense regolabile.

model (metodo che disegna solo gli spigoli di un oggetto 3d). Il visore era così pesante che doveva essere appeso al soffitto, per questo fu battezzato *La spada di Damocle*.

Dagli anni '70 fino agli anni '90 i dispositivi AR/VR furono designati quasi esclusivamente per scopi medici, simulazioni di volo, progettazione nell'industria automobilistica e addestramento militare [12]. Bisogna aspettare fino al 1991 per assistere al primo visore VR lanciato sul mercato, ossia quando l'azienda Virtuality Group lanciò la serie 1000 dei prodotti Virtuality che girava sul Commodore Amiga 3000. Oltre ai prodotti Virtuality, gli anni '90 videro altre aziende puntare su questa tecnologia, come Sega con l'annuncio del Sega VR e la Nintendo con il lancio del Virtual Boy.

Dopo allora l'interesse verso i visori da parte di grandi aziende è stato modesto fino al 2010, anno in cui venne prodotto il primo prototipo dell'Oculus Rift. Questo visore aveva il tracciamento della posizione, godeva di un angolo di visione di 90° e risolveva i problemi di distorsione che avevano i precedenti visori pre-distorcendo l'immagine renderizzata in tempo reale. Inoltre, nel 2013 l'azienda Valve inventò un display a bassa latenza che permise di creare schermi privi di lag e di motion blur indesiderato (il cosiddetto "*smear-effect*"). Questo in aggiunta all'avanzamento tecnologico di vari componenti sviluppati per smartphone - giroscopi, sensori di movimento, piccoli schermi HD e processori potenti miniaturizzati - diedero nuova linfa a questa tecnologia e vennero lanciati sul mercato molti modelli di visori.

Nel 2014 Google annunciò Cardboard, una piattaforma per la realtà virtuale fruibile con lo smartphone inserito all'interno di un visore. Un simile progetto lo distribuì Sam-

sung nel 2015 con Samsung Gear VR. Entrambi i progetti non offrivano un'esperienza coinvolgente e, nonostante fossero economici, non riscossero il successo desiderato e vennero dismessi. Nel 2016 Valve e l'azienda HTC lanciarono il visore HTC Vive. Questo apparecchio sfruttava una nuova tecnologia chiamata "*room scale*" che permetteva all'utente di muoversi liberamente in una zona di gioco invece di essere vincolato a stare fermo, inoltre i controller erano tracciati grazie a telecamere montate sul visore stesso e grazie ad una serie di sensori che andavano montati nella stanza. Sempre nel 2016 l'azienda Sony lanciò il Playstation VR, visore che montava un pannello OLED a 1080p e aveva un angolo di visuale di 100°. Questo visore non godeva di componenti di rilievo in confronto ai competitor ma si affacciava ad un mercato diverso, quello delle console.

Aggiungere una conclusione!

1.1.2 Stato dell'arte - Building the Metaverse

Oggi parlando di Metaverso si fa riferimento a piattaforme che riescono ad avvicinarsi ma che ancora non sono il Metaverso come definito in precedenza. Piattaforme come Roblox, Fortnite e Horizon Worlds presentano molti elementi del Metaverso, come una consistente rappresentazione 3D degli utenti, come la possibilità per gli utenti di creare i contenuti e come la possibilità di portare queste in una moltitudine di esperienze diverse. Questi "*giochi*" combinano insieme tante tecnologie e provano a produrre un'esperienza che si discosta da tutto ciò che è venuto prima. Ma per avere il Metaverso descritto da Matthew Ball ci vorranno ancora anni di ricerca e sviluppo. Infatti mancano ancora i protocolli, le innovazioni che possono permettere quella visione e soprattutto le tecnologie necessarie.

A livello infrastrutturare le tecnologie per accomodare migliaia o addirittura milioni di persone allo stesso momento in un'esperienza sincrona e persistente semplicemente non esistono. E non solo, internet non è stato progettato per permettere esperienze simili. La maggior parte delle connessioni che avvengono sfruttano il protocollo TCP che garantisce la qualità dei dati ricevuti ma permette solo connessioni singole tra due utenti. In effetti è quello che sperimentiamo quando inviamo una mail o ci connettiamo su Facebook. Avvengono milioni di connessioni singole simultaneamente nel mondo ma è tutt'oggi molto difficile permettere a più di un centinaio di utenti di essere connessi

tra di loro con connessioni full-duplex. Per garantire un'esperienza sincrona, il Metaverso avrebbe invece bisogno proprio di questo tipo di infrastruttura, un sistema simile alle attuali videochiamate o alle connessioni che avvengono all'interno dei videogiochi online ma in scala molto più grande. Le attuali piattaforme virtuali nascondono questa mancanza dividendo il loro mondo virtuale in diverse zone o partite, ognuna connessa con un server e con un numero finito di utenti, e mascherano il passaggio dell'utente da un server ad un altro come un passaggio tra queste divisioni.

Per garantire l'interoperabilità e la sincronia inoltre, il Metaverso avrà bisogno di un largo e complesso insieme di nuovi protocolli e standards. Infatti i sistemi attuali operano su formati simili ma non compatibili tra loro (basti pensare ai numerosi standard per la compressione d'immagini che sono presenti in internet). Questo è mitigato dal fatto che l'attuale esperienza di internet è asincrona, perciò quando un contenuto viene migrato da un sistema ad un altro viene spesso convertito nel formato di destinazione durante la fase non connessa. La difficoltà di questo passaggio è causata dalla naturale resistenza che hanno aziende e organizzazioni diverse a collaborare per creare uno standard condiviso, un esempio di ciò è come Apple sia resistente ad adottare lo standard USB-C sui suoi dispositivi o anche come siano presenti un gran numero di standard per le prese di corrente nel mondo.

Perciò Matthew Ball immagina che il Metaverso arriverà, similmente per come è stato con internet, come un disordinato insieme di processi e sviluppi diversi che gradualmente si uniformerà. La differenza sostanziale è che internet si sviluppò in ambito universitario e aveva come scopo quello di mettere in comunicazione le facoltà sparse nel mondo. Al contrario in questo momento storico è l'industria privata che detiene la consapevolezza del potenziale del Metaverso, le risorse economiche ed ingegneristiche per svilupparlo e tutto l'interesse a far sì che questo divenga realtà.

1.1.2.1 Epic Games

Una delle caratteristiche più importanti che deve avere il Metaverso secondo Ball è quella di essere un ambiente attraente per gli utenti e le aziende. Quello che ad oggi si è avvicinato di più a questo scopo è il gioco Fortnite. Sviluppato e distribuito da Epic Games, questo gioco nel tempo, e grazie alla sua fama, è diventato un occasione

di incontro e di socialità, ma soprattutto si è avvicinato al concetto di Metaverso in quanto il mondo virtuale di gioco è diventato luogo di eventi sociali indipendenti dal gioco stesso quali concerti e luogo di visione di contenuti multimediali a cui gli utenti possono assistere con il proprio avatar. Nella piattaforma, infatti, si sono svolti concerti di artisti famosi come Marshmellow, Ariana Grande e Trevis Scott e sono stati distribuiti i trailer di Star Wars e di Tenet in anteprima.

Fortnite inoltre è una delle poche piattaforme che presenta livelli di interoperabilità così elevati. Infatti, Fortnite si interseca con varie proprietà intellettuali di aziende quali la Marvel, la DC, la Sony, è accessibile attraverso dispositivi concorrenti (iPhone, Android, Playstation, Xbox e computer) e include anche diversi sistemi di identità e pagamento.

Inoltre Epic Games è proprietaria del secondo game engine più utilizzato del settore: Unreal Engine. Questo engine nel tempo è diventato così completo e realistico che ha iniziato a venir usato anche per grosse produzioni hollywoodiane, come *The Mandalorian*, ma anche per progetti di architettura urbana, per progettazione di automobili e per esercitazioni militari simulate dell'esercito degli Stati Uniti e del Regno Unito [1, 2, 10]. Tutto questo rende il loro sistema potenzialmente già compatibile con centinaia di contenuti velocizzando il processo di migrazione di applicativi all'interno del Metaverso.



Figura 1.3: Set di The Mandalorian con scenografia renderizzata con Unreal Engine

1.2 Il motore grafico Unreal Engine 5

Come anticipato nel capitolo 1.1.2.1, Unreal Engine è un motore grafico 3D sviluppato da Epic Games. La prima versione del software fu sviluppata interamente da Tim Sweeney, il fondatore della società, dal suo garage. Tim Sweeney iniziò lo sviluppo nel 1995 e il motore grafico debuttò con un videogioco chiamato *Unreal*, sviluppato da Cliff Bleszinski e James Schmalz, nel 1998. Nonostante né il motore grafico né il gioco fossero completati del tutto, Unreal ottenne il successo desiderato anche grazie a una fedele community che utilizzando l'UnrealScript (il codice proprietario sviluppato da Sweeney per il game engine) espansero il gioco e rese Epic MegaGames (poi rinominata Epic Games) un nome importante nell'industria videoludica. Inoltre il gioco fu anche la vetrina desiderata per l'engine che da quel momento fu utilizzato sotto licenza da altri sviluppatori che portarono Unreal Engine a diventare un motore grafico apprezzato ed accreditato nel settore [14, 15].

Unreal Engine fu lanciato poco prima il passaggio alle schede grafiche dedicate. Questo ha fatto sì che la prima versione si basava totalmente sul software rendering, ossia sfruttava unicamente la CPU. Sweeney sviluppò l'editor di Unreal engine in Visual Basic e prese fortemente ispirazione dall'ambiente di lavoro di quest'ultimo per l'interfaccia utente, soprannominò l'editor UnrealEd. Infatti già dalle prime versioni di Unreal Engine c'erano dei moduli che si potevano trascinare nella scena direttamente, poi una volta nella scena bastava fare un doppio click su di essi per far comparire l'editor di testo e poter scrivere il codice desiderato [9].

Ad oggi il motore grafico ha abbandonato sia UnrealScript che Visual Basic ma le scelte iniziali sono rimaste. Infatti le classi native che troviamo oggi nella libreria Unreal C++ sono le stesse che c'erano in UnrealScript. Sono ancora presenti le classi native principali Actor, Pawn e Character: la classe Actor è la classe base per tutti gli oggetti che vengono disposti o vengono generati nel livello; la classe Pawn è un'espansione della classe Actor per tutti gli oggetti che possono essere governati da giocatori o dall'intelligenza artificiale; la classe Character è un'espansione della classe Pawn che possiede una Mesh, delle Collisions e una logica di movimento incorporata. Alla creazione di una classe l'editor fa scegliere la parent class e in base alla scelta viene generato automaticamente il rispettivo script di base [5].

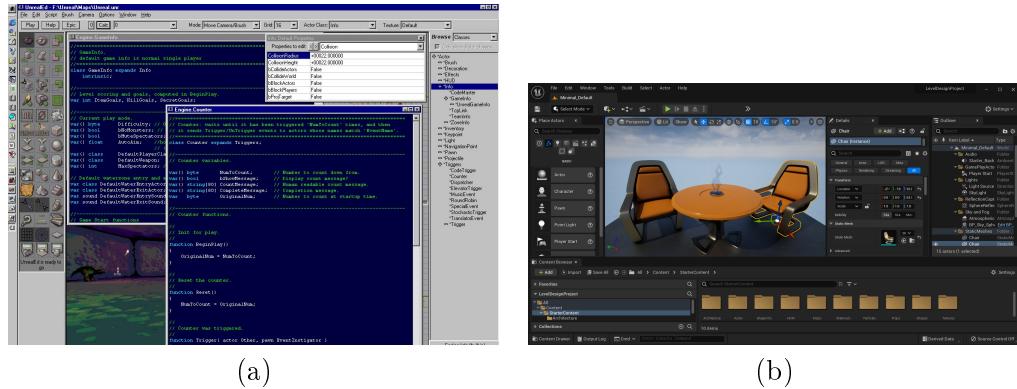


Figura 1.4: Screenshot della prima versione di UnrealEd. (a) Screenshot della versione 5 dell'editor di Unreal Engine. (b)

```

1 #include "GameFramework/Actor.h"
2 #include "MyActor.generated.h"
3
4 UCLASS()
5 class AMyActor : public AActor
6 {
7     GENERATED_BODY()
8
9 public:
10
11     // Sets default values for this actor's properties
12     AMyActor();
13
14     // Called when the game starts or when spawned
15     virtual void BeginPlay() override;
16
17     // Called every frame
18     virtual void Tick(float DeltaSeconds) override;
19
20 };

```

Algoritmo 1.1: File header generato alla creazione di un Actor

È possibile inoltre esporre una variabile dello script all'editor tramite appositi specificatori, espandibili con una lista di proprietà tra cui scegliere:

```

1 UPROPERTY(EditAnywhere, category="VariableCategory")
2 int32 MyVariable;

```

Algoritmo 1.2: Specificatore UPROPERTY per esporre una variabile all'editor

Anche l'impostazione iniziale organizzata in moduli è stata mantetuta. I moduli sono gli oggetti che possono essere trascinati in scena e che hanno funzionalità definite. È possibile espandere la lista di oggetti trascinabili in scena con quelli personalizzati dallo sviluppatore, si troveranno nel Content Browser e non nel pannello di Place Actor.

In Unreal Engine la scena nel quale viene creata l'esperienza di gioco è chiamata Level. Un Level è un ambiente 3D che può essere popolato da oggetti e geometrie. Ogni oggetto che viene posizionato nel Level è un Actor. Più tecnicamente infatti un Actor è la classe che definisce un qualsiasi oggetto a cui è associato un Transform, ossia l'informazione sulla posizione, sulla rotazione e sulla scala [6].

L'interfaccia utente è composta da diverse finestre ed è altamente personalizzabile. Alla prima accensione dell'engine viene proposto il layout di default.

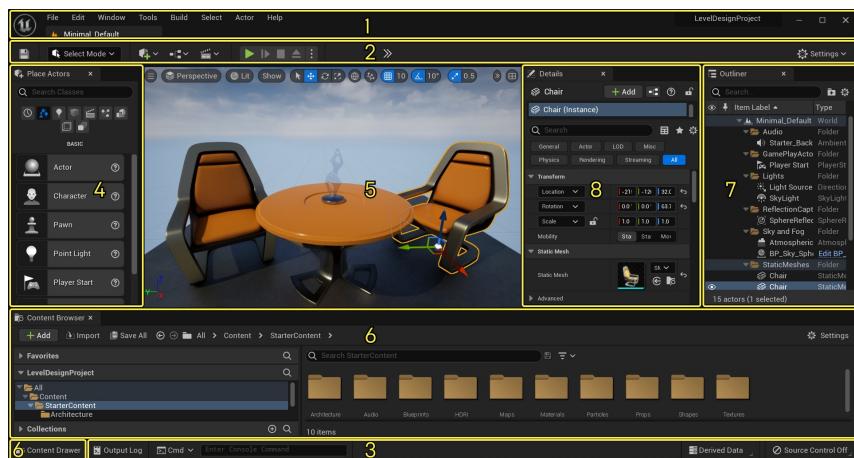


Figura 1.5: Interfaccia di default di Unreal Engine 5

- 1. Tab Bar and Menu Bar:** Il Lever Editor ha delle schede simili a quelle che hanno i browser in alto. Infatti oltre al livello si possono navigare tutti i tipi di oggetti singolarmente e qui possono essere raggruppate tutte le schede.
La Menu Bar offre accesso agli strumenti generali dell'editor quali le impostazioni del progetto e le preferenze.

2. **Toolbar:** Questo pannello mostra un gruppo di comandi, offrendo così rapido accesso agli strumenti e alle funzionalità più utilizzate.
3. **Bottom Toolbar:** Contiene scorciatoie per la Console di comando, per l'Output Log e per la funzionalità di Delivered Data
4. **Place Actor / Modes:** Il Lever Editor può essere messo in diverse modalità attivando specifiche interfacce di editing per particolari tipi di azioni che si vogliono compiere nel livello. Le possibili modalità sono:
 - Select: per selezionare, spostare e aggiungere attori nella scena.
 - Landscape: per la creazione e la modifica di vaste aree di terreno.
 - Foliage: per la creazione e la modifica di un alto numero di istanze di Static Mesh per la popolazione della scena con elementi naturali
 - Mesh Paint: per la pittura di vertici e texture su Static Mesh direttamente nella Viewport.
 - Modeling: per la creazione di mesh poligonali semplici
 - Fracture: per la creazione di oggetti e ambienti distruttibili
 - Brush Editing: per la modifica della geometria delle mesh.
 - Animation: per creare e modificare animazioni.
5. **Viewports:** questo pannello è la finestra verso il mondo che l'utente sta creando. Permette di selezionare, spostare, ruotare e scalare gli oggetti direttamente con il mouse e di muovere il punto di vista all'interno della scena.
6. **Content Browser / Content Drawer:** permette di visualizzare tutti gli asset di gioco e di organizzarli in cartelle
7. **Outliner:** permette di visualizzare e selezionare tutti gli attori presenti nella scena in una vista gerarchica ad albero.
8. **Details:** questo pannello contiene le informazioni, le funzionalità e le utilità degli oggetti che vengono selezionati. Contiene i box per la modifica dei dati del Transform per muovere, ruotare o scalare gli oggetti e mostra tutte le proprietà

editabili in base al tipo di attore che si seleziona. È qui che vengono visualizzate le variabili esposte all'editor dallo script in C++.

Unreal Engine permette di aggiungere funzionalità a classi già presenti del motore tramite le classi Blueprints. Esse permettono di raggruppare componenti di tipologie diverse, chiamati appunto Components, sotto un'unica classe e di espandere le loro funzionalità attraverso la programmazione. In Unreal Engine la programmazione può essere implementata sia attraverso C++ che attraverso la programmazione visiva organizzata a nodi dei Blueprints, chiamata Blueprints Visual Scripting. Il vantaggio della Blueprints Scripting è che è di veloce implementazione soprattutto per i principianti della programmazione. D'altro canto però questa programmazione visiva non da libero accesso al codice di gioco, è meno flessibile e meno efficiente della programmazione C++. Invece, lo svantaggio principale di C++ è che è molto più facile commettere errori che compromettono l'intero engine. Infatti bisogna fare particolare attenzione all'utilizzo dei puntatori, e delle funzioni che li sfruttano, perché l'utilizzo di uno di questi ancora non inizializzato porterà ad una *null pointer exception* e al conseguente crash dell'engine. Inoltre, questo tipo di errori sono spesso di difficile interpretazione.



Figura 1.6: Esempio di classe Blueprint per l'apertura e la chiusura di una porta in gioco

Aggiungere una conclusione!

1.3 Il software di modellazione Blender

Blender è un software per la creazione di contenuti 3D gratuito e *open source*. Supporta l'intera pipeline di strumenti per il 3D - modellazione, rigging, animazioni, creazione e pittura di texture, simulazioni, rendering, compositing e motion tracking. È un

software cross-platform, è disponibile per Linux, Windows e iOS e l'interfaccia utilizza OpenGL. La natura open source del progetto permette al pubblico di fare modifiche al codice sorgente e di creare plug-in per aggiungere features [3].

Blender fu creato da Ton Roosendaal, direttore artistico della casa di animazione olandese NeoGeo e sviluppatore software autodidatta. Il primo file sorgente fu creato nel 1994 e il software doveva essere uno strumento proprietario della casa di animazione. Nel 1998, dopo la chiusura di NeoGeo, Ton Roosendaal fondò l'azienda *Not a Number Technologies (NaN)* per continuare lo sviluppo e Blender fu distribuito come freeware. Nel 2002 la NaN andò in bancarotta e lo sviluppo di Blender fu interrotto. Gli investitori chiedevano un pagamento per la licenza di Blender perciò Roosendaal nello stesso anno creò una fondazione senza scopo di lucro, chiamata Blender Fondation, per raccogliere i fondi necessari. La fondazione lanciò una campagna di crowdfunding che raccolse centodiecimila euro in 7 settimane e nel 2002 Blender fu distribuito come software *open source*. Oggi lo sviluppo di Blender continua grazie alla numerosa community e a 24 dipendenti della Blender Institute.

Dal lancio del software come open source l'interfaccia utente è cambiata e il motore di render si è evoluto ma le caratteristiche principali sono rimaste invariate. Una delle caratteristiche principali di Blender è che è possibile chiamare quasi tutte le funzioni tramite scorciatoie da tastiera. Per questo motivo quasi tutti i tasti della tastiera sono associati ad una o più funzioni. Le funzioni più utilizzate sono associate a tasti che richiamano il nome della funzione che utilizzano, ad esempio il tasto "G" chiama la funzione *Grab* che permette di spostare l'elemento selezionato, il tasto "R" la funzione *Rotate*, il tasto "E" la funzione *Extrude* e così via.

Un'altra caratteristica di Blender è che lo spazio di lavoro è totalmente ad oggetti: l'interfaccia è divisa in finestre che è possibile dividere a loro volta in altre finestre e sottofinestre ognuna delle quali può diventare qualsiasi tipo di vista o immagine che il programma supporta. L'utente può personalizzare la combinazione di finestre e viste per ognuna delle operazioni che preferisce e può posizionare questi layout in schede similmente a come avviene in un browser web. Il programma offre già layout di default per lavorare a compiti diversi.

Blender permette di creare delle Mesh poligonali e di esportarle in formati compa-

tibili con altri programmi 3D, compreso Unreal Engine. Una Mesh poligonale è una struttura dati che rappresenta oggetti nello spazio 3D. In inglese significa maglia perché è strutturata come un reticolo.



Figura 1.7: Una Mesh poligonale in Blender

Capitolo 2

Il core di Metarace

2.1 Metarace

Metarace è un progetto in Unreal Engine che punta a sfruttare i dispositivi AR/VR per far immergere l'utente in un mondo virtuale di competizioni equestri. Si definisce un progetto nel Metaverso perché punta ad avere le caratteristiche che definiscono tutte le piattaforme attualmente vicine a questo concetto: una rappresentazione digitale della persona sotto forma di avatar, un ambiente virtuale in cui è possibile interagire e socializzare con altri utenti, degli oggetti di gioco personalizzabili e portabili al di fuori di Metarace, un'economia interna e una progettazione focalizzata sulla scalabilità per permettere in futuro di aggiornare l'applicativo all'arrivo di nuove tecnologie. In Metarace il giocatore può possedere dei cavalli virtuali e li può iscrivere a delle competizioni in cui competono con i cavalli di altri giocatori. Il giocatore può assistere alle competizioni con un avatar virtuale entrando nel mondo attraverso un visore per la realtà virtuale. In mancanza di questo può comunque assistere alla gara tramite desktop. Il vincitore di ogni gara sarà determinato da una componente intrinseca al cavallo in termini di forza innata e una componente randomica relativa alla singola gara.

L'idea dietro al gioco è quella di far immergere l'utente in un ambiente simile a quello delle competizioni equestri che si svolgono nel mondo reale, dove quindi gli spettatori e i proprietari dei cavalli non prendono parte attiva alla competizione ma vi assistono dagli spalti e dove la vittoria non è mai certa. L'idea di Metarace è di riproporre questa

formula in un ambiente virtuale immersivo, nel Metaverso.

2.2 La meccanica di base

Una partita di Metarace consiste in una gara di cavalli in cui c'è sempre un vincitore, con una piccola probabilità di pareggio. Il risvolto della gara è dato da caratteristiche intrinseche dei cavalli gareggianti e da fattori casuali.

Siccome il giocatore non prende parte attiva alla gara ma vi assiste attraverso il proprio avatar, l'esito della gara è deciso da un server che invia il risultato a ciascun client dei partecipanti. Il client calcola, attraverso un algoritmo, il comportamento dei cavalli in modo che sia coerente con il risvolto della gara inviato dal server. Il server invia al client un insieme di array di tempi. Il client divide il percorso in un numero di step uguale alla cardinalità di questi array e poi fa percorrere ai cavalli questi step sulla base dei tempi stessi. Il client perciò si occupa di renderizzare il mondo di gioco, di far iscrivere l'utente alle competizioni che desidera e a consentire la fruizione della gara. Durante lo sviluppo, è stata trovata subito la necessità di testare come poter tradurre questi dati inviati dal server in movimenti dei gareggianti all'interno del mondo di gioco.

2.2.1 L'algoritmo per il movimento dei cavalli

Lo strumento più adatto tra quelli messi a disposizione da Unreal Engine per implementare questa meccanica è la Spline Component. Una Spline Component è un percorso che lo sviluppatore può definire per utilizzare dati posizionali. Può essere usata per far muovere oggetti o per posizionare una serie di Actor lungo di essa. Questo componente è composto da una serie di nodi ordinati, posizionabili all'interno del mondo di gioco, e da una linea che segue questi nodi in maniera interpolata andando a creare così il percorso. È inoltre possibile far seguire una Spline ad una Mesh (o ad un Actor) impostando un'animazione basata sul tempo. Questo in Unreal è possibile farlo attraverso una Timeline. Una Timeline è uno strumento offerto da Unreal Engine che permette di ottenere dei valori di output nel tempo basati su una Curve impostata inizialmente.

Quello che ho implementato è stato di creare una classe C++ che rappresenta il cavallo e un Blueprint che la estendeva. Inoltre ho creato un altro Blueprint che contiene

tutte le Spline del percorso. Ogni cavallo sarà associato ad una delle Spline e la seguirà secongo l'algoritmo seguente:

```

1 FTimeline RaceTimeline;
2 UPROPERTY()
3 USplineComponent* SplineComponent = nullptr;
4 UPROPERTY()
5 UCurveFloat* CurveFloat;
6

```

Algoritmo 2.1: Sezione del file header (movimento cavallo)

La Timeline si utilizza attraverso una funzione che ne cattura il progresso (legata con una *FOnTimelineFloat*) e grazie ad una *CurveFloat* che ne definisce il comportamento. Per legare un oggetto *FOnTimelineFloat* alla funzione per catturare l'output della Timeline si utilizza la funzione *BindUFunction*. Per associare la Timeline alla *FOnTimelineFloat* e alla *CurveFloat* si utilizza la funzione *AddInterpFloat* chiamata dalla Timeline stessa.

```

1
2 void AHorse::Inizialize()
3 {
4     [...] //creazione della CurveFloat
5
6     FOnTimelineFloat RaceTimelineProgress;
7     RaceTimelineProgress.BindUFunction(
8         this, FName("RaceTimelineProgress"));
9     if(RaceCurveFloat){ //per evitare nullptr exception
10         RaceTimeline.AddInterpFloat(
11             RaceCurveFloat, RaceTimelineProgress);
12     }
13     [...] // Settaggio del RatePlay della Timeline
14 }
15
16

```

Algoritmo 2.2: Inizializzazione della Timeline nel file source (movimento cavallo)

Per far partire la Timeline bisogna chiamare la funzione *Play()*.

```

1 void AHorse::StartTimeline()

```

```

2 {
3     RaceTimeline.Play();
4 }
```

Il valore di output della Timeline (che ho chiamato *Value*) è standardizzato tra 0 e 1. È possibile eseguire un interpolazione tra 0 e la lunghezza della Spline basata su questo valore e ottenere così una posizione lungo la Spline. La posizione trovata viene poi utilizzata dall'Actor che si sposta nella posizione trovata.

```

1 void AHorse::RaceTimelineProgress(float Value)
2 {
3     if (MeshComponent && SplineComponent)
4     {
5         float DistanceInSpline = FMath::Lerp(0,
6             SplineComponent->GetSplineLength(),
7             Value);
8         FTransform TransformAtDistance =
9             SplineComponent->GetTransformAtDistanceAlongSpline(
10                 DistanceInSpline,
11                 ESplineCoordinateSpace::World);
12         SetActorLocationAndRotation(
13             TransformAtDistance.GetLocation(),
14             TransformAtDistance.GetRotation());
15     }
16 }
```

Algoritmo 2.3: Funzione che sfrutta il valore restituito dalla timeline nel tempo (movimento cavallo)

Questa funzione viene chiamata automaticamente dalla Timeline se all'interno della funzione *Tick* le viene passato il valore *DeltaTime*. La funzione *Tick* è chiamata automaticamente da Unreal Engine ad ogni frame e il valore *DeltaTime* è il tempo che è intercorso dall'ultima chiamata della funzione *Tick* (equivalente al tempo intercorso dall'ultimo frame).

```

1 void AHorse::Tick(float DeltaTime)
2 {
3     Super::Tick(DeltaTime);
```

```

4
5     RaceTimeline.TickTimeline(DeltaTime);
6 }
7

```

Algoritmo 2.4: Funzione Tick (movimento cavallo)

Per far muovere la Mesh Component coerentemente all'array di tempi inviato dal server, bisogna creare una *FloatCurve* che si basa su di essi. Questo viene fatto nella funzione *Inizialize*. La classe *UCurveFloat* non offre funzioni che permettono di instanziarne una a tempo di esecuzione, per questo viene utilizzata una *FRichCurve* per la sua creazione runtime. Sia la *FRichCurve* che la *CurveFloat* sono costituite da una serie di punti su un piano cartesiano e hanno entrambe delle regole di interpolazione da un punto ad un altro. Le regole per la definizione dell'interpolazione si possono definire grazie alla struct *ERichCurveInterpMode* che può essere: *RCIM_Linear* (interpolazione lineare), *RCIM_Cubic* (interpolazione cubica) oppure *RCIM_None* (nessuna interpolazione). Si inizia con la definizione di variabili temporanee per lo spazio e il tempo, si definisce inoltre la lunghezza standardizzata di ogni step sulla curva e si calcola il tempo totale che il cavallo impiegherà a percorrere tutto il percorso:

```

1 void AHorse::Inizialize()
2 {
3     float TempS = 0; //Variabile temporanea per lo Spazio
4     float TempT = 0; //Variabile temporanea per il Tempo
5
6     //Lunghezza di ogni step standardizzata
7     float DeltaS = 1 / float(TimeArray.Num());
8
9     for(float Time : TimeArray)
10    {
11        //Tempo complessivo per percorrere tutto il percorso
12        TotalTime += Time;
13    }
14

```

Algoritmo 2.5: Definizione variabili temporanee (movimento cavallo)

Una volta definite queste variabili si inizia la creazione della curva. Il numero dei punti sul piano cartesiano equivale al numero degli elementi dell'array dei tempi più un punto iniziale all'origine del piano cartesiano. Un'istanza della classe *FRichCurve* è costituita da una serie di *FRichCurveKey*, perciò popolo un *TArray* con delle keys di questo tipo.

```
15     RichCurve = new FRichCurve();
16
17     TArray<FRichCurveKey> Keys;
18     FRichCurveKey FirstKey = FRichCurveKey(0, 0, 0, 1,
19                                         ERichCurveInterpMode::RCIM_Cubic);
20     Keys.Add(FirstKey);
21
```

Per la creazione delle keys itero su tutti gli elementi dell'array di tempi e ad ogni iterazione sommo il tempo i con la somma di tutti i tempi precedenti diviso la somma di tutti i tempi in modo da ottenere un valore standardizzato. Ossia ogni key ha come variabile tempo:

$$T_{i=0..n} = \sum_{j=0}^i \frac{t_j}{t_{tot}} \quad (2.1)$$

Ogni key ha come variabile spazio la lunghezza standardizzata del singolo step moltiplicato il numero di iterazione corrente:

$$S_{i=0..n} \equiv \Delta S(i+1) \quad (2.2)$$

Inoltre l'ultima key generata non ha interpolazione per evitare che i cavalli prima dell'arrivo inizino a rallentare

```
22     for (int i = 0; i < TimeArray.Num(); i++)
23     {
24         TempS = DeltaS * (i + 1);
25         TempT += TimeArray[i] / TotalTime;
26         FRichCurveKey Key;
27         if ((i + 1) < TimeArray.Num())
28         {
29             Key = FRichCurveKey(TempT, TempS, 0, 0,
30             ERichCurveInterpMode::RCIM_Cubic);
```

```

31     }
32
33     {
34         Key = FRichCurveKey(TempT, TempS, 1, 0,
35                             ERichCurveInterpMode::RCIM_None);
36     }
37     Keys.Add(Key);
38 }
39

```

Una volta generate le keys le aggiungo alla *FRichCurve*, poi istanzio un nuovo oggetto *UCurveFloat* e setto le sue keys essere quelle della *FRichCurve*.

```

40     RichCurve->SetKeys(Keys);
41     UCurveFloat* RaceCurveFloat = NewObject<UCurveFloat>();
42     RaceCurveFloat->FloatCurve = *RichCurve;
43
44     [...] // Inizializzazione della Timeline
45

```

Infine imposto il RatePlay della Timeline uguale alla frequenza con cui il cavallo si muove lungo il percorso.

```

46     RaceTimeline.SetPlayRate(1 / TotalTime);
47 }
48

```

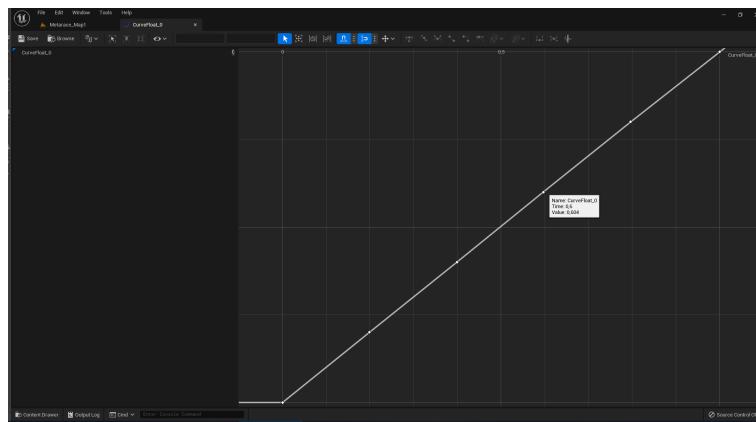


Figura 2.1: *UCurveFloat* generata dall'algoritmo per il movimento dei cavalli

Questo processo genera una curva come quella mostrata in Figura 2.1. Questa curva garantisce un cambiamento di velocità fluido del cavallo durante tutta la gara nonostante, dall’immagine, i cambiamenti di coefficiente angolare non siano apprezzabili.

2.3 Animazione dei cavalli

I cavalli non soltanto si muovono lungo il percorso ma possiedono una serie di animazioni che vengono riprodotte in base allo stato in cui si trovano e alla velocità che possiedono. I cavalli infatti sono delle Mesh associate ad uno scheletro gerarchico di ossa che può essere animato per muovere la Mesh. Questo tipo di Mesh vengono gestite con Unreal Engine come delle Skeletal Mesh. Per la gestione delle animazioni in Unreal Engine si può utilizzare uno strumento chiamato *AnimGraph*. Questo strumento valuta la pose da mostrare per una Skeletal Mesh in ogni frame in base alle animazioni passate come input [7].

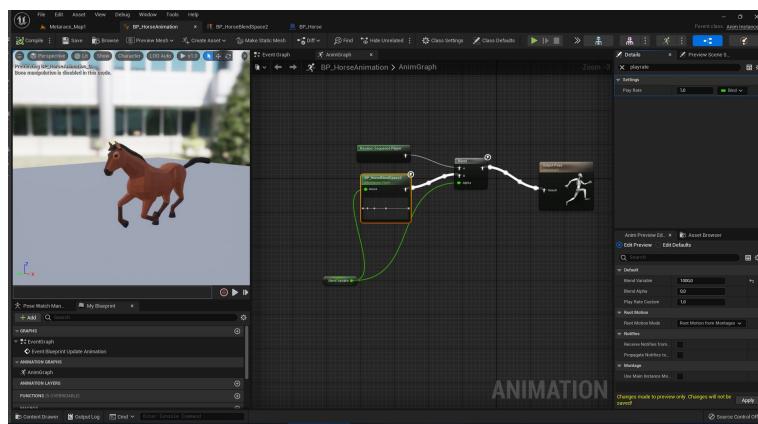


Figura 2.2: *AnimGraph* che gestisce l’animazione del cavallo.

L’*AnimGraph* mostrato in Figura 2.2 utilizza due lettori di animazione: un *BlendSpace2D* e un *Random Sequence Player*.

Il *Random Sequence Player* è un lettore che permette di salvare una lista di animazioni da riprodurre. Queste animazioni vengono riprodotte in maniera casuale una dopo l’altra. Questo nodo è perfetto per riprodurre le animazioni *Idle* del cavallo (le animazioni per quando il cavallo è fermo).

Il *BlendSpace2D* invece è un asset speciale che permette di fare il blend tra più animazioni sulla base di un valore in input. Per sfruttarlo bisogna inserire le animazioni lungo l'asse delle ascisse che il nodo mostra. Posizionando le animazioni in maniera adeguata, ossia per far combaciare il valore di input con la frequenza di riproduzione delle animazioni, si ottiene un blend automatico tra le animazioni legato al valore di input. L'editor aiuta in questo processo segnalando un errore quando l'animazione non combacia con il valore sull'asse. In questo contesto è stato utilizzato come valore di input la velocità del cavallo e questo fa sì che ognuno di questi possiede, in ogni momento, un'animazione coerente con l'andatura che mantiene. Questa tecnica risolve il problema del "pattinamento" - ossia quando un'animazione è più lenta o più veloce di quanto un Actor si muova in scena avendo come effetto quello di sembrare che scivoli (che appunto pattini) sul terreno - nonostante le animazioni date in input non sarebbero adeguate per la velocità sostenuta. Questo perché l'animazione risultante è in realtà un'interpolazione delle animazioni di input in base, appunto, alla velocità. Le animazioni inserite in questo BlendSpace sono 4: una di *Idle*, una di camminata, una per il trotto, una per la corsa leggera e una per il galoppo.

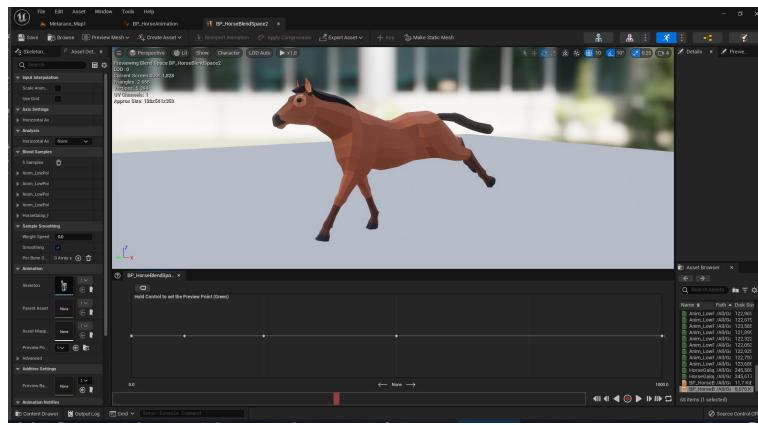


Figura 2.3: *BlendSpace2D* per l'interpolazione delle animazioni sulla base della velocità.

Come si può vedere dalla Figura 2.2, il valore di input per il *BlendSpace* è utilizzato anche come valore di blend tra il *BlendSpace2D* e il *Random Sequence Player*. Questo perché quando il valore di input è nullo vuol dire che si è raggiunti l'andatura in cui è possibile riprodurre una tra le animazioni di *Idle* e perciò ne viene riprodotta una

casualmente.

2.3.1 Animazione del galoppo creata con Blender

Un cavallo non può considerarsi un cavallo da corsa senza un'animazione del galoppo. Per questo motivo ho ideato e creato l'animazione in questione nella suite Blender.

La suite Blender permette di animare qualsiasi tipo di oggetto e qualsiasi tipo di deformazione o traslazione tramite la tecnica di inserimento di *keyframe poses* all'interno di una finestra apposita: l'Action Editor. Letteralmente keyframe pose vuol dire "posa del fotogramma chiave", questo perché questa tecnica prevede di salvare tutti i dati sul Transform (quindi su posizione, rotazione e scala) di uno o più oggetti all'interno di vari frame che riprodotti in sequenza andranno a creare l'effetto di movimento che compone l'animazione. Questa tecnica è particolarmente efficace per le mesh associate ad uno scheletro gerarchico di ossa (chiamato rig). Infatti muovendo una o più ossa si andrà a deformare la mesh senza cambiarne la geometria ma solo cambiandole la posizione, creando una pose, e salvando queste pose nelle keyframe si potrà salvare ed esportare la riproduzione delle keyframe come un'animazione.



Figura 2.4: Vista in Blender della Mesh del cavallo, con il suo rig associato, in una pose tra le 12 dell'animazione del galoppo

Creare animazioni convincenti è un'arte molto complessa che va studiata a fondo. Spesso il modo migliore per crearne una è partire da una reference del mondo reale e così ho fatto. Sono partito da un'immagine che rappresenta lo studio alla base dell'ani-

mazione del galoppo diviso in 12 frame e ho cercato di far combaciare la posizione della mesh con quella che si vede nell’immagine per ogni singolo frame.

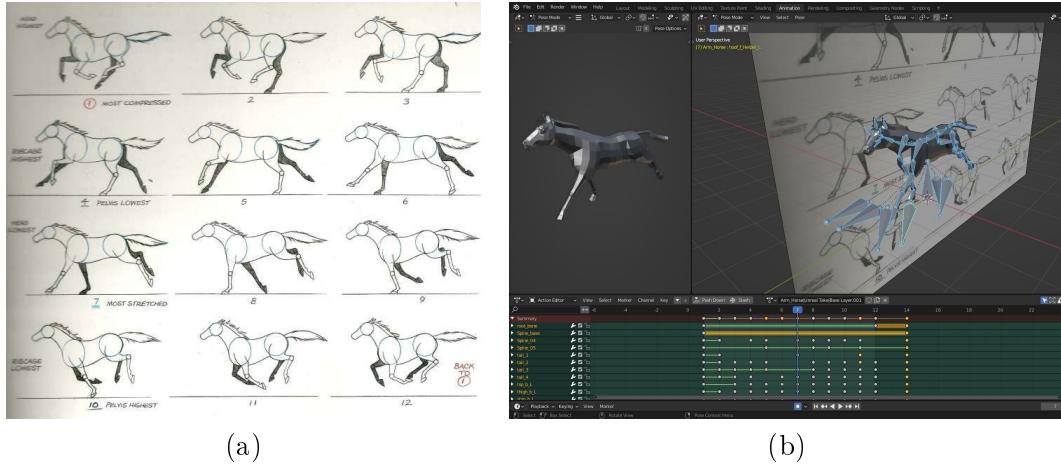


Figura 2.5: 12 frame disegnati di un cavallo al galoppo (a) Schermata in cui è possibile vedere la Pose Mode di Blender, l’Action Editor con le keyframe, la mesh del cavallo in una pose con il suo scheletro e l’immagine utilizzata come reference. (b)

2.4 Progettazione del mondo 3D

2.4.1 Unreal Engine

2.4.2 I modelli 3D

Per creare le mesh per questo progetto è stato scelto il software Blender per via della sua natura multipiattaforma e per la leggerezza di utilizzo. Questo software è stato utilizzato per creare delle mesh interamente e per modificare asset di gioco già esistenti.

Infatti, Blender permette di creare Mesh poligonali a partire da figure elementari, queste figure sono: il piano, il cubo, il cerchio, la UV sfera, la ICO sfera, il cilindro, il cono e l’anello. Inoltre permette di partire anche da due modelli più complessi: la griglia e la scimmietta Suzanne caratteristica di Blender. Questa scimmietta ha fatto la sua comparsa nel 2002, quando l’azienda fondata da Ton Roosendaal, la *Nan*, era ormai destinata alla bancarotta. Gli sviluppatori, in vista dell’inevitabile discontinuità del software, fecero uscire un aggiornamento poco prima della definitiva chiusura che,

come piccolo personale tocco, includeva questo Easter Egg. Venne chiamata Suzanne come la scimmia del film *Jay and Silent Bob... Fermate Hollywood!* Da quel momento Suzanne divenne l'alternativa utilizzabile in Blender come modello di test, oltre ai più comuni Utah Teapot e lo Stanford Bunny, per provare materiali, texture ed altro.

Le mesh sono costituite da tre elementi di base: vertici, facce e spigoli. Blender permette di modificare la mesh agendo direttamente su questi elementi utilizzando la Edit Mode. Le mesh create per Metarace con il software Blender sono state: il terreno di gioco, la griglia di start, gli spalti e altri oggetti di gioco come la stalla per i cavalli, una fontana e un campo ippico per la disciplina degli attacchi.

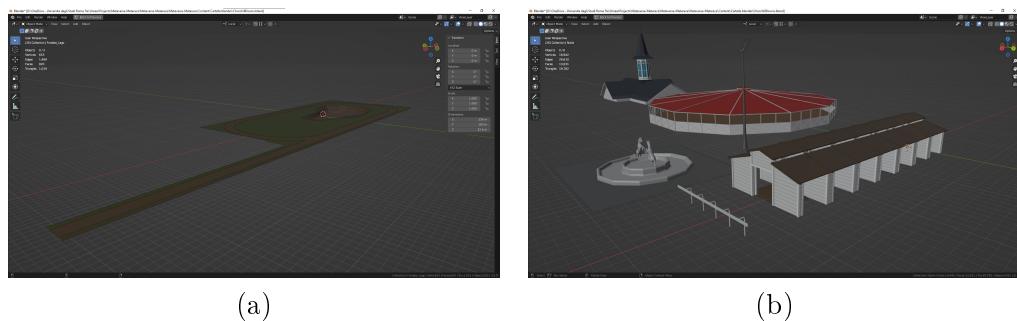


Figura 2.6: Terreno di gioco in Blender (a) Altri modelli in Blender. (b)

Queste mesh sono state create in modo da risultare composte da sole facce triangolari. Infatti un game engine può usare solo delle mesh con questo tipo di facce. È possibile comunque importare su Unreal Engine delle mesh con facce con più lati ma queste verranno automaticamente ricorvertite in triangoli. Perciò per evitare distorsioni indesiderate ed errori nella conversione bisogna utilizzare sole facce triangolari oppure facce con al massimo 4 lati che non presentino angoli maggiori di 180°. Inoltre maggiore sarà il numero di vertici che una mesh possiede maggiore sarà il carico che il motore di gioco dovrà sopportare per la renderizzazione. Per questo motivo le mesh sono state create facendo in modo che il numero dei vertici fosse il più basso possibile.

2.5 Analisi del software

2.5.1 Modello degli eventi

Capitolo 3

Fase di Progettazione

3.1 Scelte di progetto guidate dalla flessibilità

3.2 Architettura Client - Server

3.3 Programmazione ad Eventi (EDP)

3.4 Model - View - Controller (MVC)

Capitolo 4

Fase di Implementazione

4.1 Cosa ho fatto con Unreal Engine 5

4.2 Cosa ho fatto con Blender

4.3 Cosa ho fatto in C++

Capitolo 5

Risultati

5.1 Demo di gioco

5.2 Interazione utente

5.3 Design

5.4 Esempi

Conclusioni e sviluppi futuri

La tesi è finita

Elenco delle figure

1.1	Una tipica scena in Habitat.	3
1.2	Stereoscopio statunitense regolabile.	5
1.3	Set di The Mandalorian con scenografia renderizzata con Unreal Engine . . .	8
1.4	Screenshot della prima versione di UnrealEd. (a) Screenshot della versione 5 dell'editor di Unreal Engine. (b)	10
1.5	Interfaccia di default di Unreal Engine 5	11
1.6	Esempio di classe Blueprint per l'apertura e la chiusura di una porta in gioco	13
1.7	Una Mesh poligonale in Blender	15
2.1	<i>UCurveFloat</i> generata dall'algoritmo per il movimento dei cavalli	22
2.2	<i>AnimGraph</i> che gestisce l'animazione del cavallo.	23
2.3	<i>BlandSpace2D</i> per l'interpolazione delle animazioni sulla base della velocità.	24
2.4	Vista in Blender della Mesh del cavallo, con il suo rig associato, in una pose tra le 12 dell'animazione del galoppo	25
2.5	12 frame disegnati di un cavallo al galoppo (a) Schermata in cui è possibile vedere la Pose Mode di Blender, l>Action Editor con le keyframe, la mesh del cavallo in una pose con il suo scheletro e l'immagine utilizzata come reference. (b)	26
2.6	Terreno di gioco in Blender (a) Altri modelli in Blender. (b)	27

Elenco degli algoritmi

1.1	File header generato alla creazione di un Actor	10
1.2	Specificatore UPROPERTY per esporre una variabile all'editor	10
2.1	Sezione del file header (movimento cavallo)	18
2.2	Inizializzazione della Timeline nel file source (movimento cavallo)	18
2.3	Funzione che sfrutta il valore restituito dalla timeline nel tempo (movimento cavallo)	19
2.4	Funzione Tick (movimento cavallo)	19
2.5	Definizione variabili temporanee (movimento cavallo)	20

Bibliografia

- [1] M. L. Ball. The metaverse explained in 14 minutes. <https://bigthink.com/series/the-big-think-interview/why-the-metaverse-matters/>, 2022.
- [2] M. L. Ball and J. Navok. Epic's flywheel and unreal engine. <https://www.matthewball.vc/all/epicprimer1/#section1>, 2020.
- [3] Blender. The freedom to create. <https://www.blender.org/about/>.
- [4] U. di Torino. Second life e unito. <https://www.unito.it/ateneo/gli-speciali/archivio-degli-speciali/second-life-e-unito>.
- [5] U. E. Documentation. Introduction to c++ programming in ue4. <https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/ProgrammingWithCPP/IntroductionToCPP/>.
- [6] U. E. Documentation. Level editor. <https://docs.unrealengine.com/5.0/en-US/level-editor-in-unreal-engine/>.
- [7] U. Engine. Animgraph documentation. <https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/AnimBlueprints/AnimGraph/>.
- [8] W. S. Life. Fashion in second life. https://wiki.secondlife.com/wiki/Fashion_in_Second_Life.
- [9] D. Lightbown. Classic tools retrospective: Tim sweeney on the first version of the unreal editor. <https://www.gamedeveloper.com/design/>

classic-tools-retrospective-tim-sweeney-on-the-first-version-of-the-unreal-editor
2018.

- [10] S. Lozé. Simcentric scales up tactical military simulation training with unreal engine. <https://www.unrealengine.com/fr/spotlights/simcentric-scales-up-tactical-military-simulation-training-with-unreal-engine>, 2020.
- [11] C. Morningstar and F. R. Farmer. The lessons of lucasfilm's habitat. https://web.stanford.edu/class/history34q/readings/Virtual_Worlds/LucasfilmHabitat.html, 1990.
- [12] U. of Illinois. National center for supercomputing applications: History. <https://web.archive.org/web/20150821054144/http://archive.ncsa.illinois.edu/Cyberia/VETopLevels/VR.History.html>, 2009.
- [13] Q. Parker. A second look at school life. *The Guardian*, 2007.
- [14] C. Plante. Better with age: A history of epic games. <https://www.polygon.com/2012/10/1/3438196/better-with-age-a-history-of-epic-games>, 2012.
- [15] M. Thomsen. History of the unreal engine. <https://web.archive.org/web/20220707005958/https://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine?page=1>, 2010.
- [16] Wikipedia. https://it.wikipedia.org/wiki/Second_Life.
- [17] Wikipedia. Stereoscopia. <https://it.wikipedia.org/wiki/Stereoscopia>.