Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Corso di Laurea in Informatica

Motori di Gioco a Confronto: Analisi Comparativa e Tecnologie Emergenti

Relatore:
Chiar.mo Prof.
Angelo Di Iorio

Presentata da:
Annalisa Poluzzi

Sessione III
Anno Accademico 2023/2024

Alla mia famiglia,
per il loro supporto incondizionato.

Abstract

I motori di gioco rappresentano il cuore dello sviluppo videoludico moderno, fornendo strumenti avanzati per la gestione di grafica, fisica, intelligenza artificiale e audio. Questa tesi si propone di analizzare e confrontare tredici motori di gioco di rilievo esaminandone le caratteristiche tecniche, le funzionalità principali e l'adattabilità a diverse tipologie di videogiochi. L'analisi comparativa si basa su criteri quali modularità, accessibilità, costi e facilità d'uso. Inoltre, viene approfondita l'integrazione di tecnologie emergenti come il Ray Tracing e il Deep Learning Super Sampling (DLSS), che stanno ridefinendo gli standard grafici nei videogiochi moderni. I risultati evidenziano come la scelta del motore dipenda fortemente dal tipo di progetto e dagli obiettivi degli sviluppatori, con alcune soluzioni più adatte a videogiochi ad alto budget e produzione su larga scala (AAA), e altre più accessibili per sviluppatori indipendenti.

Indice

Capitolo 1

Introduzione

I motori di gioco rappresentano uno strumento fondamentale, se non indispensabile, nello sviluppo moderno di videogiochi. Forniscono agli sviluppatori le basi per creare esperienze interattive complesse e immersive, gestendo aspetti cruciali come grafica, fisica, suono e intelligenza artificiale.

Oltre all'utilizzo nel mondo videoludico, fungono da catalizzatori per l'innovazione tecnologica, guidando lo sviluppo di GPU sempre più potenti e di tecnologie avanzate che trovano applicazione in settori come il machine learning e il rendering professionale. Trovano applicazione anche in ambiti di insegnamento ricreativo, come i serious games, che sfruttano meccaniche di gamification per scopi educativi e formativi.[1]

Questa tesi si propone di esplorare il mondo dei motori di gioco, analizzandone le componenti fondamentali e confrontando le soluzioni più diffuse.

Ciascun motore sarà valutato evidenziandone le peculiarità, sia positive che negative, in un'ottica che tiene conto delle esigenze di una vasta gamma di utilizzatori: dai piccoli team senza competenze di programmazione, fino agli sviluppatori esperti che vogliono creare giochi complessi. L'analisi non si limiterà agli aspetti tecnici ma considererà anche quanto questi strumenti siano accessibili a sviluppatori junior o a chi non ha una formazione approfondita nella scrittura di codice.

Un ulteriore obiettivo di questo lavoro è approfondire l'integrazione di funzionalità avanzate come il Ray Tracing in tempo reale e l'uso di tecniche di supercampionamento basate sull'intelligenza artificiale. Tali innovazioni, promosse da tecnologie come RTX e DLSS di NVIDIA, stanno ridefinendo gli standard della resa grafica nei videogiochi. Verrà esaminato come i principali motori implementano e supportano queste tecnologie, valutandone l'impatto nello sviluppo di videogiochi moderni.

Lo scopo di questo lavoro è duplice: da un lato, fornire una guida dettagliata agli sviluppatori per comprendere e sfruttare al meglio i motori di gioco; dall'altro, esplorare il potenziale delle tecnologie emergenti come RTX e DLSS nel ridefinire gli standard della grafica nei videogiochi.

Capitolo 2

Struttura generale dei motori di gioco

Questo capitolo esplora l'architettura tipica dei motori di gioco, descrivendo i principali componenti che li costituiscono, le loro interazioni e come vengono utilizzati nello sviluppo di videogiochi moderni.

Le informazioni presentate in questo capitolo sono basate principalmente sul libro *Game Engine Architecture* di Jason Gregory [2]. Questo testo rappresenta una delle fonti più autorevoli nel campo, fornendo un'analisi dettagliata delle strutture fondamentali che compongono un motore di gioco e delle best practice adottate nell'industria videoludica. Gli argomenti trattati seguono la logica espositiva del libro, adattandone i concetti ai motori di gioco contemporanei analizzati in questa tesi.

2.1 Introduzione all'architettura dei motori di gioco

Un motore grafico, o motore di gioco, è un framework software ed è il nucleo di un videogioco o di qualsiasi altra applicazione con grafica in tempo reale. I videogiochi sono, infatti, sistemi soft real-time. Esso fornisce le tecnologie di base, semplifica lo sviluppo, e spesso permette al gioco di funzionare su piattaforme differenti come le console o sistemi operativi per personal computer. Le funzionalità di base fornite tipicamente da un motore grafico includono un motore di rendering (*renderer*) per grafica 2D e 3D, un motore fisico o rilevatore di collisioni, suono, scripting, animazioni, intelligenza artificiale, networking, e scene-graph[3].

2.1.1 Evoluzione storica

Negli anni '70 e '80, all'inizio dell'industria dei videogiochi, ogni gioco era progettato in modo unico, con codice scritto specificamente per l'hardware su cui doveva funzionare. Questo approccio comportava una forte dipendenza dall'architettura del computer o della console target, rendendo i giochi difficilmente portabili su altre piattaforme. Gli sviluppatori dovevano costruire da zero sistemi per la grafica, la fisica, il suono e il gameplay per ogni progetto, il che rendeva il processo di sviluppo complesso, lungo e costoso. A quel tempo, la mancanza di standardizzazione significava che ogni gioco era un prodotto "a sé stante", ottimizzato esclusivamente per l'hardware specifico per cui era stato creato.

Con l'evoluzione dell'industria, emerse la necessità di strumenti più versatili e modulari per facilitare il lavoro degli sviluppatori. La creazione di motori di gioco riutilizzabili rappresentò un cambiamento radicale. Questi motori fornirono un'astrazione tra il software e l'hardware, permettendo agli sviluppatori di concentrarsi sugli aspetti creativi

piuttosto che sui dettagli tecnici specifici delle piattaforme. Oggi, motori di gioco come Unity e Unreal Engine supportano una vasta gamma di piattaforme, dai computer alle console, fino ai dispositivi mobili e VR. Grazie a questa portabilità, i giochi moderni possono essere sviluppati una sola volta e distribuiti su molteplici sistemi, aumentando la flessibilità e riducendo i costi.

2.2 Componenti principali

I motori di gioco rappresentano strutture software complesse, costruite per gestire in modo efficace le varie componenti fondamentali necessarie allo sviluppo di esperienze interattive. Queste componenti permettono di simulare mondi virtuali realistici o stilizzati, gestendo tutto, dalla grafica tridimensionale ai suoni, dalle interazioni con l'utente alla fisica del mondo simulato. Una delle caratteristiche distintive di un buon motore di gioco è la sua modularità, che consente di combinare o personalizzare i vari sottosistemi a seconda delle esigenze specifiche di un progetto.

L'importanza delle componenti principali di un motore di gioco risiede nella loro cooperazione sinergica. Un motore grafico può mostrare scene visivamente accattivanti, ma senza un sistema di fisica credibile o animazioni fluide, l'immersività ne risente. Analogamente, la gestione audio e l'intelligenza artificiale giocano ruoli cruciali nel creare un ambiente sonoro realistico e avversari o alleati dinamici e reattivi.

Tuttavia, è importante sottolineare che questa sezione offre solo una panoramica generale delle principali componenti dei motori di gioco. Data la vastità e la complessità del tema, ogni sottosistema potrebbe essere oggetto di trattazioni approfondite. L'obiettivo di questa parte della tesi non è esaminare le strutture dei motori in modo completo, bensì offrire un contesto introduttivo utile per comprendere le funzionalità chiave e le loro interazioni di base.

La sezione esplorerà quindi brevemente le componenti fondamentali, partendo dal rendering engine fino agli strumenti e pipeline di sviluppo, evidenziando come ognuna di queste parti si integri nel ciclo di vita del gioco. Questo approccio servirà come base per affrontare in modo mirato l'analisi dei motori specifici trattati successivamente.

2.2.1 Rendering Engine

Il **Rendering Engine** è una delle componenti fondamentali di qualsiasi motore di gioco, responsabile della creazione delle immagini visualizzate su schermo durante l'esecuzione del gioco. Il suo compito principale è trasformare i dati del mondo virtuale, come oggetti 3D, texture, illuminazione e materiali, in frame visibili che vengono renderizzati ad alta velocità per garantire un'esperienza fluida e realistica.

Nel ciclo di vita di un frame, il rendering engine riceve informazioni da altri sottosistemi, come il motore fisico per la posizione degli oggetti, il sistema di animazione per il movimento dei personaggi e il sistema di gestione delle risorse per texture e materiali. Una volta elaborati questi dati, il rendering engine comunica con la GPU tramite API grafiche come OpenGL, DirectX o Vulkan, traducendo le istruzioni del motore in operazioni grafiche a basso livello.

Questo processo è fondamentale per garantire che i giochi siano visivamente accattivanti e allo stesso tempo performanti, mantenendo il frame rate sopra valori critici come 30 o 60 FPS (frame per secondo), a seconda delle piattaforme di destinazione.

Architettura e funzionamento del Rendering Engine Il funzionamento del rendering engine può essere suddiviso in diversi passaggi chiave, ciascuno dei quali corrisponde a una fase della pipeline di rendering grafico. La pipeline parte dall'elaborazione dei vertici degli oggetti 3D fino alla generazione del frame finale che viene mostrato all'utente.

1. Fase di input dei dati: Gli oggetti del mondo virtuale, come personaggi, ambienti e oggetti statici o dinamici, vengono rappresentati attraverso modelli tridimensionali composti da mesh di triangoli (vedi Figura 2.1). Questi insiemi di piccoli triangoli descrivono la forma e la struttura dell'oggetto nello spazio tridimensionale.

   Ogni mesh è definita da vertici, che ne indicano la posizione nello spazio, e da texture, immagini applicate sulla superficie per aggiungere dettagli visivi. I materiali definiscono ulteriori proprietà, come il colore, la rugosità e la capacità di riflettere la luce. Per calcolare correttamente l'illuminazione, vengono utilizzate le normali, vettori associati ai triangoli che indicano la direzione della superficie.

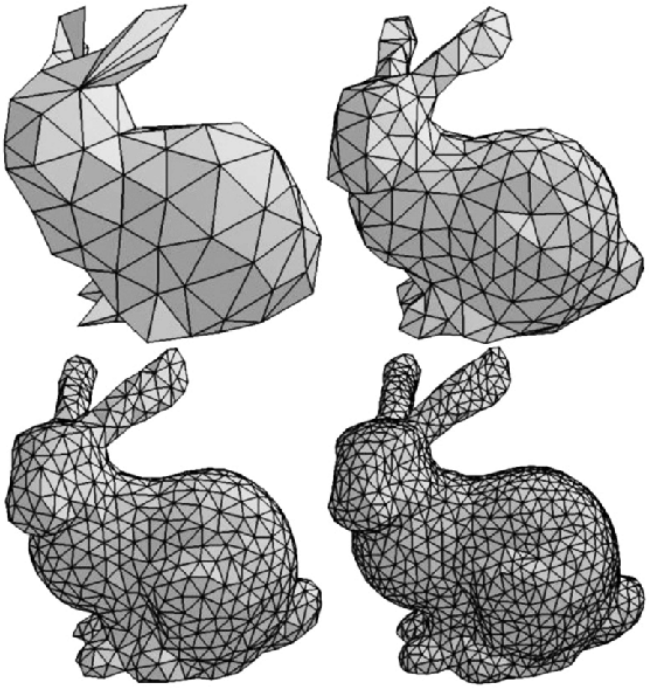

Figura 2.1: Esempio di mesh di triangoli che rappresentano la struttura geometrica di un oggetto tridimensionale.

2. Trasformazione geometrica (Vertex Processing): Nella pipeline grafica, i vertici che definiscono la geometria degli oggetti vengono trasformati attraverso una serie di matrici, che includono:

   - Trasformazioni locali (posizione, rotazione e scala dell'oggetto);
   - Trasformazioni della vista, che posizionano la scena in relazione alla posizione e orientamento della telecamera;
   - Proiezioni prospettiche o ortogonali, per determinare come gli oggetti appariranno sullo schermo.

3. Rasterizzazione: La rasterizzazione è il processo che converte i dati geometrici, rappresentati da primitive come triangoli, in frammenti (unità di calcolo per i pixel). Dopo che i vertici dei triangoli sono stati trasformati e proiettati nello spazio 2D dello schermo, la GPU identifica i pixel che si trovano all'interno dei contorni del triangolo proiettato. Questo avviene tramite una tecnica di scan conversion, dove il triangolo viene attraversato riga per riga (scan-line) per determinare quali pixel devono essere colorati. Ogni frammento, tuttavia, non è ancora il pixel finale: è una rappresentazione temporanea che contiene informazioni come posizione, profondità (valore Z), colore, e coordinate delle texture.

4. Shading dei pixel: Ogni frammento generato dalla rasterizzazione viene passato al **pixel shader** (o fragment shader), un programma eseguito sulla GPU che calcola il colore finale di ciascun pixel. Il pixel shader può applicare una varietà di calcoli e effetti visivi per ottenere un'immagine realistica e coinvolgente. Tra le operazioni principali ci sono applicazione delle texture, calcoli di illuminazione, effetti visivi avanzati come trasparenza o riflessi e correzione del colore e blending.

5. Output al frame buffer: Una volta elaborati, i pixel finali vengono scritti nel frame buffer, la memoria video temporanea che memorizza il frame fino a quando non viene mostrato sul display.

   Per garantire un output fluido e privo di artefatti visivi, i motori di rendering moderni utilizzano il **double buffering**. Nel double buffering, esistono due frame buffer: mentre il primo buffer viene scansionato dall'hardware del display, il motore di rendering aggiorna il secondo buffer con il nuovo frame. Durante l'intervallo di blanking verticale (quando il display sta riposizionando il "pennello" elettronico in alto a sinistra), i due buffer vengono scambiati. Questo processo previene un fenomeno chiamato tearing, dove l'immagine potrebbe essere divisa, mostrando una parte del nuovo frame e una parte del frame precedente.

Tecnologie di Rendering e comunicazione con la GPU Le API grafiche rappresentano il ponte di comunicazione tra il motore di gioco e la GPU. Esse offrono un set di istruzioni specifiche per inviare alla scheda grafica informazioni come dati geometrici, texture, materiali e comandi di rendering, permettendo al motore grafico di sfruttare al meglio la potenza della GPU.

Le API grafiche vengono gestite tramite i driver grafici forniti dai produttori delle schede video (come NVIDIA, AMD o Intel). Quando si installa il driver di una GPU, vengono automaticamente inclusi i componenti necessari per supportare le API più recenti.

Per gli sviluppatori, invece, è necessario accedere ai relativi SDK per lavorare con queste API. Gli SDK forniscono strumenti, librerie e documentazione per integrare correttamente le API nei progetti.

Ecco alcune delle principali:

- OpenGL: Una delle API grafiche più longeve e portabili, utilizzata su PC, console e dispositivi mobili.

  OpenGL non richiede un SDK specifico, ma gli sviluppatori possono utilizzare librerie come GLFW o GLEW per facilitare la creazione di finestre, il rendering e l'accesso alle estensioni di OpenGL. La documentazione ufficiale e risorse utili sono disponibili sul sito ufficiale di Khronos Group (khronos.org/opengl).

- DirectX (Microsoft): Dominante su Windows e Xbox, questa API è sviluppata da Microsoft ed è ottimizzata per offrire alte prestazioni su queste piattaforme.

  Lo sviluppo con DirectX è integrato nativamente in Microsoft Visual Studio, che fornisce l'ambiente necessario per iniziare a lavorare (Visual Studio).

- Vulkan: Progettata per offrire un controllo a basso livello delle risorse hardware, Vulkan è ottimizzata per alte prestazioni e calcoli paralleli. È spesso utilizzata nei giochi ad alte prestazioni e nelle applicazioni che necessitano di un rendering avanzato.

  Gli sviluppatori possono scaricare il Vulkan SDK dal sito ufficiale del progetto (vulkan.org).

Queste API consentono ai motori grafici di eseguire calcoli complessi in modo parallelo, riducendo il carico sulla CPU e migliorando il frame rate.

Illuminazione e Ombre L'illuminazione è un aspetto fondamentale per il realismo grafico nei videogiochi, determinando come la luce interagisce con gli oggetti della scena. Le tecniche di illuminazione possono essere classificate in due categorie principali:

- **Illuminazione statica (Precalcolata):** Calcolata durante la fase di sviluppo e memorizzata sotto forma di *lightmap*, questa tecnica riduce il carico computazionale durante l'esecuzione del gioco. È particolarmente utilizzata in ambienti statici, come i livelli predefiniti nei giochi strategici o platform. Un esempio è *Super Mario 3D World*, dove l'illuminazione è precalcolata per migliorare le prestazioni senza compromettere la qualità visiva.

- **Illuminazione dinamica (In tempo reale):** Permette alla luce di modificarsi e interagire con la scena in base agli eventi e ai movimenti degli oggetti. È essenziale nei giochi open-world con cicli giorno/notte, come *Red Dead Redemption 2* e *The Legend of Zelda: Breath of the Wild*, dove la luce del sole cambia realisticamente nel corso della giornata.

Tecniche principali di illuminazione

- **Modello di illuminazione Phong e Blinn-Phong:** Tecniche tradizionali per simulare la riflessione della luce sulle superfici. Il modello di Phong è usato per calcolare la luce diffusa e speculare su materiali lucidi, mentre Blinn-Phong è un miglioramento che ottimizza i calcoli per prestazioni migliori.

- **Mappe di ombre (Shadow Mapping):** Metodo usato per generare ombre dinamiche. Il motore di rendering proietta un'ombra in base alla posizione della sorgente luminosa, simulando il blocco della luce da parte degli oggetti.

- **Ray Tracing:** Tecnologia avanzata introdotta nei motori più recenti, che permette di calcolare riflessi, rifrazioni e ombre con estrema precisione. Ad esempio, in giochi come *Cyberpunk 2077*, il Ray Tracing viene usato per simulare riflessi realistici su superfici bagnate e vetri. Un approfondimento su questa tecnologia verrà affrontato all'interno del capitolo 5.

Ottimizzazioni: Culling e Level of Detail (LOD) Per garantire alte prestazioni senza sacrificare la qualità visiva, i motori di rendering implementano varie tecniche di ottimizzazione:

- **Frustum Culling:** Questa tecnica esclude dalla pipeline di rendering gli oggetti che si trovano fuori dall'inquadratura della telecamera, riducendo il numero di poligoni da elaborare e migliorando il frame rate.
- **Occlusion Culling:** Evita di renderizzare gli oggetti completamente nascosti dietro altri oggetti.
- **Level of Detail (LOD):** Tecnica che riduce dinamicamente la complessità dei modelli 3D in base alla distanza dalla telecamera. Ad esempio, in *The Witcher 3*, i modelli dei personaggi lontani vengono rappresentati con meno dettagli, mentre quelli vicini sono altamente dettagliati.

Effetti di Post-Processing Il rendering non si conclude con la generazione del frame. Dopo il passaggio attraverso la pipeline principale, vengono applicati effetti di post-processing per migliorare l'aspetto visivo:

- **Anti-Aliasing:** Riduce i bordi seghettati degli oggetti, rendendo le immagini più fluide. Tecnologie moderne includono il *Temporal Anti-Aliasing (TAA)*, utilizzato in *Unreal Engine 5*, e il *Deep Learning Super Sampling (DLSS)* di NVIDIA.
- **Bloom:** Simula l'effetto di diffusione della luce oltre i bordi degli oggetti luminosi.
- **High Dynamic Range (HDR):** Aumenta la gamma dinamica tra aree chiare e scure, migliorando il contrasto e la percezione del colore.

2.2.2 Sistema di fisica

Il sistema di fisica è una delle componenti principali di un motore di gioco, responsabile della simulazione delle interazioni fisiche tra oggetti, come collisioni, gravità, forze e dinamiche dei corpi rigidi. Questa simulazione non è solo un elemento estetico ma contribuisce all'interattività e al realismo dell'esperienza di gioco.

Struttura e funzionamento del sistema di fisica Il sistema di fisica è composto da diverse sottocomponenti, tra cui:

1. Rilevamento delle collisioni: il primo passo consiste nel rilevare quando due o più oggetti entrano in contatto. Questo avviene tramite tecniche di rilevamento delle collisioni come bounding boxes (scatole delimitatrici), bounding spheres (sfere delimitatrici) o algoritmi più complessi basati su strutture dati spaziali, ad esempio KD-trees o BSP trees. Questi metodi permettono di ottimizzare la ricerca degli oggetti in collisione e ridurre il carico computazionale.
2. Simulazione delle dinamiche dei corpi rigidi: gli oggetti rigidi vengono modellati secondo le leggi della meccanica classica di Newton. Ogni corpo rigido ha proprietà come massa, velocità, momento angolare e forza applicata. Durante la simulazione, il motore calcola come questi parametri influenzano il movimento degli oggetti in risposta alle forze esterne, come la gravità o una spinta.

3. Applicazione delle forze: le forze possono essere costanti (come la gravità), dinamiche (generate da un'azione del giocatore) o impulsive (causate da un urto o una collisione). La corretta gestione delle forze è essenziale per simulare interazioni realistiche, come un oggetto che rotola o un personaggio che salta.

4. Risoluzione delle collisioni: quando una collisione viene rilevata, il sistema deve determinare come gli oggetti coinvolti reagiranno. Questo processo tiene conto delle proprietà fisiche degli oggetti, come il coefficiente di restituzione (che definisce quanto rimbalza un oggetto) e l'attrito. La risoluzione delle collisioni può comportare cambiamenti nella velocità, nella direzione e nella deformazione degli oggetti.

5. Simulazione di corpi deformabili (soft body): oltre ai corpi rigidi, molti motori di gioco offrono la possibilità di simulare oggetti che si deformano sotto l'effetto delle forze applicate. Questo include elementi come tessuti, liquidi o materiali morbidi, la cui forma cambia dinamicamente durante il gioco. Questa simulazione aggiunge realismo soprattutto nei giochi di avventura o di simulazione.

Middleware e Framework Utilizzati Molti motori di gioco integrano middleware di fisica avanzati per gestire queste simulazioni. Alcuni dei più popolari includono:

- Havok Physics: Uno dei middleware di fisica più utilizzati, famoso per la sua efficienza e l'ampio supporto multipiattaforma.

- NVIDIA PhysX: Ottimizzato per l'hardware NVIDIA, consente simulazioni di fisica avanzata e realistici effetti di distruzione.

- Bullet Physics: Un framework open-source utilizzato in numerosi giochi per gestire la fisica dei corpi rigidi, le collisioni e i soft body.

Integrazione del Sistema di Fisica nel Motore di Gioco Il sistema di fisica è strettamente legato ad altri sottosistemi del motore di gioco:

- Rendering Engine: Mostra visivamente gli effetti della fisica, come oggetti che si muovono, cadono o rimbalzano.

- Animazione: Le animazioni dei personaggi possono essere influenzate dalla fisica, ad esempio durante il ragdoll effect, dove i personaggi cadono in modo realistico in base alla fisica simulata.

- Sistema di Input: I comandi del giocatore (ad esempio, saltare o correre) sono tradotti in forze applicate dal sistema di fisica.

In sintesi, il sistema di fisica fornisce ai giochi una simulazione realistica delle interazioni nel mondo virtuale. È fondamentale nei generi di gioco che richiedono un'interazione precisa con l'ambiente, come i platform, i giochi di guida e gli FPS.

2.2.3 Animazione

L'animazione è una componente essenziale per la creazione di esperienze visive fluide e realistiche. Essa consente di dare vita a personaggi, oggetti e ambienti, simulando movimenti organici e azioni complesse.

Tipi di animazione nei motori di gioco Le principali tecniche di animazione utilizzate nei videogiochi sono:

- Animazione basata su sprite o texture:

  Questa tecnica, comunemente usata nei giochi 2D, si basa sulla sequenza di immagini statiche predefinite, dette sprite (vedi Figura 2.2), che rappresentano diverse pose o stati del personaggio o oggetto animato. Ogni frame della sequenza è una nuova immagine che viene mostrata in rapida successione per creare l'illusione del movimento.

  A differenza di altre animazioni, dove il movimento è interpolato tra pose chiave, nella sprite animation ogni variazione di movimento richiede un'immagine distinta. Ad esempio, se un personaggio sta correndo, ogni posizione delle gambe e delle braccia deve essere disegnata manualmente. Questo significa che più complesso è il movimento, più immagini sono necessarie.

Figura 2.2: Esempio di personaggio animato tramite sprite.

- Animazione gerarchica (rigid hierarcal animation):

  L'animazione gerarchica è basata su una struttura ad albero in cui ogni parte animata è rappresentata come un nodo della gerarchia. Le trasformazioni come rotazioni, traslazioni e scalature applicate a un nodo superiore (padre) influenzano automaticamente tutti i nodi figli. Questo tipo di animazione è ampiamente utilizzato per oggetti rigidi, come porte, leve, pistoni o meccanismi meccanici.

  Non è ideale per simulare movimenti naturali o organici, poiché i segmenti del corpo si muovono in modo troppo rigido e poco realistico.

- Animazione scheletrica (skeletal animation):

  L'animazione scheletrica è considerata lo standard nei giochi 3D moderni per la gestione dei personaggi e delle creature. Essa si basa su una struttura gerarchica di articolazioni (*joints*, - vedi Figura 2.3), a volte chiamate anche ossa in modo erroneo, in cui ogni articolazione può essere trasformata (ruotata, traslata o scalata) e influenzare la posizione della mesh esterna a essa associata. Ogni vertice della mesh è legato a una o più articolazioni tramite un sistema di pesi, che definisce l'entità dell'influenza di ciascuna articolazione.

  Tuttavia presenta alcune limitazioni, soprattutto nel rappresentare movimenti realistici come quelli del corpo umano. Ad esempio, le articolazioni delle spalle o delle anche, essendo altamente mobili, non si prestano bene a un controllo strettamente gerarchico. Per questo motivo, l'animazione scheletrica viene spesso affiancata da altre tecniche per migliorare la naturalezza dei movimenti.

- Animazione tramite morph targets o blend shapes:

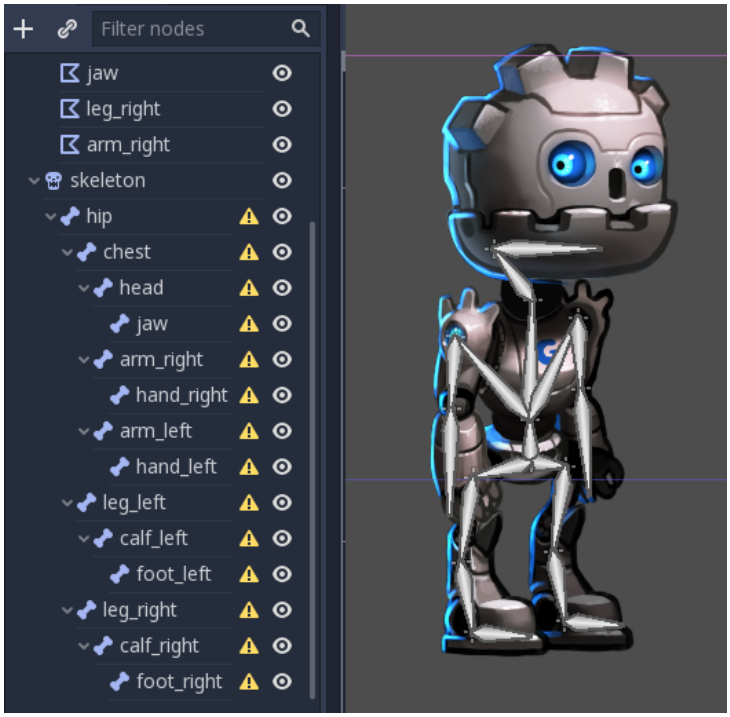

Figura 2.3: Esempio di personaggio costruito tramite uno scheletro.

Questa tecnica si basa sulla manipolazione diretta della mesh tramite una serie di "target" che rappresentano configurazioni diverse della geometria (vedi Figura 2.4). Ogni morph target può essere considerato come una variante deformata della mesh originale e il motore di gioco può interpolare tra diversi target per creare effetti di trasformazione progressivi.

Le blend shapes vengono spesso utilizzate per simulare espressioni facciali, come sorrisi, smorfie o movimenti delle sopracciglia. A differenza dell'animazione scheletrica, questa tecnica è più indicata per cambiamenti sottili o localizzati, come la deformazione di muscoli facciali o il gonfiarsi di un pallone.

Non è ideale per animazioni complesse e articolate, come quelle del corpo intero, poiché richiede un numero elevato di morph targets per rappresentare tutte le possibili variazioni.

Tecniche di ottimizzazione delle animazioni Nel contesto dello sviluppo di videogiochi, le tecniche di ottimizzazione delle animazioni giocano un ruolo cruciale per mantenere prestazioni elevate senza compromettere la qualità visiva.

Una delle tecniche più comuni è la **compressione dei dati** delle animazioni, che riduce la quantità di informazioni memorizzate mantenendo intatta la percezione visiva del movimento. Questo può essere fatto eliminando i keyframe non necessari o rappresentando le trasformazioni scheletriche con formati di dati più efficienti.

Un'altra tecnica fondamentale è il **culling** delle animazioni, in cui il motore di gioco evita di calcolare e aggiornare le animazioni per oggetti che non sono visibili alla telecamera. Questo approccio è particolarmente efficace in grandi ambienti open-world, dove solo una frazione degli oggetti deve essere animata in ogni frame.

Il **Level of Detail (LOD)** delle animazioni permette inoltre di utilizzare versioni semplificate delle stesse animazioni quando il personaggio o l'oggetto si trova lontano

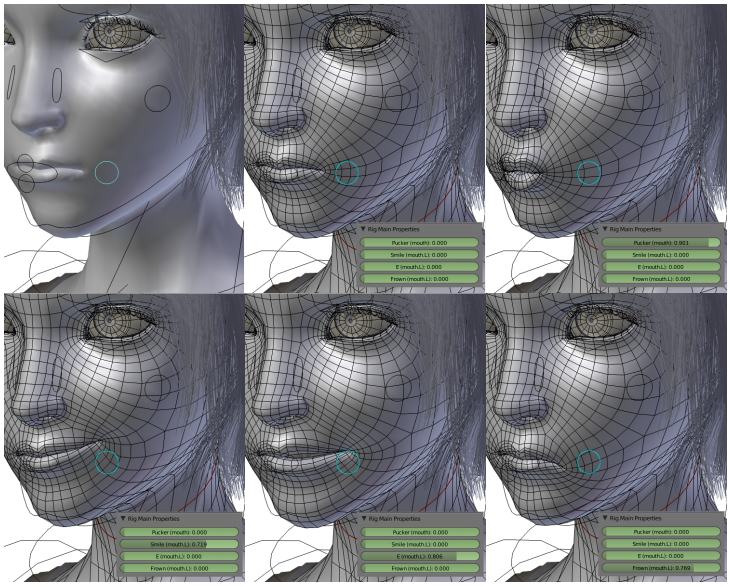

Figura 2.4: Esempio di viso animato tramite morph target.

dalla telecamera. Ad esempio, quando un personaggio è distante, il motore può sostituire i dettagli complessi delle animazioni facciali con movimenti di base.

Infine, il blending ottimizzato consente di ridurre il numero di calcoli combinando pose predefinite solo quando necessario. Invece di interpolare costantemente ogni possibile movimento, il motore seleziona in modo dinamico le animazioni essenziali per la scena.

Queste tecniche, se applicate correttamente, possono migliorare il frame rate globale del gioco e ridurre il consumo di memoria, garantendo che anche su hardware limitato i personaggi si muovano in modo fluido e naturale. Un esempio emblematico è rappresentato da *The Last of Us Part II*, dove la gestione delle animazioni dei personaggi secondari è ottimizzata tramite il culling e il LOD, riservando le risorse principali ai protagonisti principali e alle scene narrative critiche.

2.2.4 Audio

Il sistema audio nei motori di gioco svolge un ruolo cruciale nel creare un'esperienza immersiva, permettendo di gestire suoni ambientali, dialoghi, effetti sonori e colonne sonore in modo sincronizzato con l'azione di gioco. Un'implementazione efficace del comparto audio contribuisce a rafforzare l'atmosfera e l'interattività del gioco.

Ecco le funzioni principali del sistema:

- Effetti sonori sincronizzati: Gli effetti sonori rappresentano elementi fondamentali per segnalare eventi specifici nel gioco, come spari, passi o esplosioni. Questi suoni sono spesso associati a trigger in tempo reale che li attivano in base alle azioni del giocatore o agli eventi di gioco.

- Audio spaziale: Il sistema audio è spesso progettato per simulare la provenienza direzionale dei suoni, aumentando il senso di profondità e realismo. Tecniche di spatial audio, come il surround sound o il 3D audio, permettono ai giocatori di percepire la posizione dei suoni rispetto alla loro posizione nel mondo di gioco.

- Musica dinamica: In molti giochi, la colonna sonora si adatta dinamicamente all'azione. Ad esempio, la musica può intensificarsi durante una battaglia o diventare più calma durante l'esplorazione.
- Gestione dei dialoghi: Nei giochi narrativi, i dialoghi tra personaggi possono essere sincronizzati con le animazioni e gestiti attraverso sistemi di scripting avanzati.

2.2.5 Modularità

I motori di gioco moderni sono costituiti da una serie di sottosistemi specializzati, ciascuno dei quali contribuisce alla gestione di specifici aspetti dell'esperienza interattiva. Una delle caratteristiche fondamentali che garantisce l'efficacia di questi sistemi è la loro modularità, che consente di sviluppare, aggiornare e sostituire i vari componenti senza influenzare l'intera struttura del motore.

Concetto di modularità La modularità si riferisce all'organizzazione a layer del motore di gioco, in cui ogni componente è progettato per essere il più possibile indipendente dagli altri. Questo approccio permette una maggiore flessibilità, facilitando la manutenzione e l'ottimizzazione del codice e la futura espandibilità. Ad esempio, in un gioco che richiede simulazioni fisiche complesse, è possibile sostituire il sistema di fisica di base del motore con un middleware avanzato come Havok o NVIDIA PhysX. In modo analogo, il motore di rendering può essere adattato per supportare tecnologie specifiche, come il ray tracing per riflessi e ombre realistiche.

Implementazione della modularità I principali motori di gioco implementano la modularità tramite una combinazione di:

- Sottosistemi indipendenti: Ogni componente (ad esempio, il motore di rendering, il sistema di animazione o il sistema di input) è sviluppato come un'unità separata che interagisce con gli altri attraverso interfacce ben definite.
- Pipeline di sviluppo flessibile: Il flusso di lavoro è progettato per consentire l'integrazione di moduli aggiuntivi o personalizzati. Questo è particolarmente utile per progetti che richiedono l'uso di tecnologie avanzate o specifiche.
- Plugin e pacchetti: Alcuni motori offrono vasti ecosistemi di plugin che possono essere facilmente integrati per estendere le funzionalità di base senza modificare il codice sorgente del motore.

Vantaggi della modularità

- Adattabilità ai diversi generi di gioco: Un motore modulare può essere utilizzato per sviluppare una vasta gamma di giochi, dai platform in 2D ai giochi open-world complessi in 3D, semplicemente attivando o disattivando i moduli necessari.
- Portabilità su diverse piattaforme: La modularità facilita la creazione di versioni del gioco ottimizzate per specifiche piattaforme hardware (ad esempio, console, PC o dispositivi mobili).
- Riduzione dei costi di sviluppo: Grazie alla possibilità di riutilizzare moduli esistenti, gli sviluppatori possono risparmiare tempo e risorse durante lo sviluppo.

2.2.6 Intelligenza Artificiale

L'intelligenza artificiale (IA) è una delle componenti più importanti nei giochi moderni, poiché consente di creare personaggi non giocanti (NPC) credibili e interazioni dinamiche con l'ambiente e i giocatori. L'IA può determinare come i nemici reagiscono agli attacchi del giocatore, come gli alleati assistono nelle missioni o come le creature virtuali si comportano in un mondo aperto. Oltre agli NPC, può essere applicata a vari aspetti del gioco, come la generazione procedurale di livelli o la simulazione economica.

2.3 Strumenti e pipeline di sviluppo

La creazione di un videogioco moderno richiede l'uso di una pipeline di sviluppo strutturata e di strumenti integrati nei motori di gioco. Questi strumenti facilitano la gestione delle risorse artistiche, la programmazione, il versionamento e il deployment su diverse piattaforme, garantendo un flusso di lavoro collaborativo ed efficiente.

2.3.1 Editor di sviluppo e strumenti integrati

I motori di gioco offrono editor grafici che permettono agli sviluppatori di creare e gestire scene 2D o 3D, posizionare oggetti, applicare texture e definire logiche di gioco. Questi strumenti eliminano la necessità di interventi manuali su file di codice e consentono ai designer di partecipare attivamente al processo di sviluppo.

Esempi di strumenti integrati includono il sistema di visual scripting *Blueprint* di Unreal Engine e l'editor di Unity, che supporta sia la programmazione tramite script in C# sia l'utilizzo di componenti predefiniti (vedi Figura 2.5),

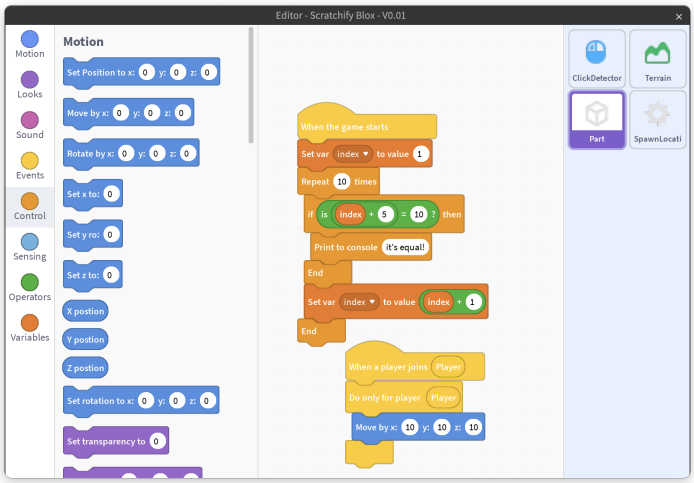

Figura 2.5: Editor grafico semplice di Scratch, con interfaccia drag-and-drop.

2.3.2 Pipeline per la gestione delle risorse

Le risorse artistiche come modelli 3D, texture, audio e animazioni sono una parte fondamentale del gioco. La pipeline di gestione delle risorse prevede i seguenti passaggi:

- **Importazione**: Gli asset creati con strumenti esterni (es. Blender per modelli 3D, Photoshop per texture) vengono importati nel motore di gioco.
- **Ottimizzazione**: Compressione delle texture, creazione di livelli di dettaglio (LOD) per i modelli 3D e riduzione delle dimensioni dei file audio.
- **Conversione**: Le risorse vengono convertite nei formati interni ottimizzati per il motore e per la piattaforma di destinazione.

Questi processi, spesso automatizzati, assicurano che le risorse siano caricate e utilizzate in modo efficiente durante l'esecuzione del gioco.

2.3.3 Versionamento e controllo delle modifiche

Per progetti complessi in cui più sviluppatori lavorano contemporaneamente su codice e risorse artistiche, è essenziale mantenere il controllo sulle modifiche apportate. Molti motori di gioco integrano nativamente strumenti di versionamento per gestire i progetti su larga scala.

Unity offre un'integrazione diretta con Plastic SCM, ottimizzato per gestire grandi asset binari come modelli 3D e texture, oltre al codice. Unreal Engine, invece, supporta nativamente Perforce, un sistema molto utilizzato nei progetti di grandi dimensioni. Anche Git può essere utilizzato in maniera generica, soprattutto per piccoli team.

Questi strumenti permettono di:

- Monitorare le modifiche ai file di progetto e agli asset.
- Gestire conflitti tra diversi membri del team.
- Ripristinare versioni precedenti in caso di errori o problemi.

Il versionamento integrato garantisce la stabilità del progetto e facilita la collaborazione tra programmatori, artisti e designer.

2.3.4 Build e deployment automatico

I motori di gioco moderni dispongono di pipeline automatizzate per la creazione della build finale. Durante questo processo, il motore compila il codice, ottimizza le risorse e genera versioni eseguibili del gioco per diverse piattaforme (PC, console, dispositivi mobili). Ad esempio, Unreal Engine utilizza il sistema di Unreal Build Tool per automatizzare la compilazione, mentre Unity offre il Cloud Build, che consente la creazione di build direttamente nel cloud.

2.3.5 Testing, debugging e profiling

Durante lo sviluppo, il gioco deve essere costantemente testato e ottimizzato. I motori di gioco offrono strumenti per:

- **Debugging grafico**: Individuare problemi con texture, shader o modelli 3D.
- **Profiling delle performance**: Monitorare l'uso della CPU, GPU e memoria, assicurando un frame rate stabile.
- **Test automatici**: Verificare automaticamente la stabilità del gameplay e la correttezza delle funzionalità.

2.4 Conclusione

I motori di gioco sono sistemi complessi, ma il loro funzionamento si basa su una cooperazione ben organizzata tra le diverse componenti, come il rendering, la fisica e l'audio. Grazie alla loro struttura modulare, gli sviluppatori possono modificare e personalizzare facilmente ogni parte del motore per adattarla alle esigenze del progetto. Questo capitolo ha offerto una panoramica generale utile per comprendere come funzionano queste tecnologie, preparando il terreno per un confronto più dettagliato tra i vari motori nei prossimi capitoli.

Capitolo 3

Principali motori di gioco

In questo capitolo verranno presentati i principali motori grafici analizzati in questa tesi.

Ogni motore rappresenta un esempio significativo delle diverse opzioni disponibili sul mercato, che spaziano da soluzioni open-source e gratuite, ideali per sviluppatori indipendenti o piccoli team, fino a piattaforme avanzate utilizzate nelle grandi produzioni AAA.

È importante sottolineare che l'analisi si concentrerà esclusivamente sui motori di gioco accessibili al pubblico, evitando di includere piattaforme proprietarie come **Frostbite** (sviluppato da Electronic Arts) o **Source Engine** (di proprietà di Valve Corporation).

Sebbene questi motori siano noti per il loro impatto nel settore e per essere stati utilizzati in celebri produzioni come *Battlefield* o *Half-Life*, la loro natura chiusa e l'impossibilità di accedervi senza licenze speciali li escludono dal confronto diretto. Questo approccio ci permette di concentrarci su strumenti disponibili a una vasta gamma di sviluppatori, rendendo i risultati dell'analisi più applicabili a contesti pratici e accademici.

Prima di analizzare i singoli motori, verrà presentata una tabella riassuntiva (vedi Figura 3.1) che sintetizza alcune delle loro caratteristiche principali. Tuttavia, questa tabella non è esaustiva e potrebbe contenere discrepanze o semplificazioni, segnalate con il simbolo **(*)**. Ad esempio, alcuni motori di gioco non dispongono nativamente di strumenti per lo sviluppo 3D, ma offrono estensioni o workaround, come proiezioni isometriche o sistemi di simulazione tridimensionale. Sebbene questi espedienti possano permettere una parziale gestione della terza dimensione, essi non rappresentano una soluzione ottimizzata e, di conseguenza, il motore in questione non può essere considerato pienamente adatto allo sviluppo 3D.

Per questo motivo, la tabella deve essere intesa come una panoramica introduttiva, utile per una prima comparazione, mentre le analisi dettagliate nei capitoli successivi approfondiranno gli aspetti più critici e le specificità di ciascun motore.

Motore	2D	3D	VR/AR	Costi	Portabilità	Linguaggio	Facilità d'uso
Unity	Sì	Sì	Sì	Gratuito < 200k$; 2200$/anno	PC, Console, Mobile, VR	C#, Visual Scripting	Media
Unreal Engine	No	Sì	Sì	Gratuito < 1M$; 5% royalties	PC, Console, Mobile, VR	C++, Blueprint	Bassa
Godot	Sì	Sì	Sì	Gratuito	PC, Mobile, Web	GDScript, C#, C++	Alta
CryEngine	No	Sì	Sì	Royalties basate sulle vendite	PC, Console	C++, Lua	Bassa
GameMaker	Sì	No	No	Gratis; 100$/ anno per uso commerciale	PC, Console, Mobile	GML, Drag-and-Drop	Alta
O3DE	No	Sì	Sì	Gratuito	PC, Console, Mobile	C++, Lua	Media
GDevelop	Sì	Circa	No	Gratuito	PC, Web	Visual Scripting	Alta
Construct3	Sì	No	No	Abbonamento richiesto (99$/anno)	PC, Mobile, Web	JavaScript	Alta
Scratch	Sì	No	No	Gratuito	PC, Web	Visual Scripting	Molto alta
Cocos2d	Sì	Sì*	No	Gratuito	PC, Mobile, Web	C++, Lua, Javascript	Media
Defold	Sì	Sì	No	Gratuito	PC, Mobile, Web*	Lua, C++	Media
Stride	No*	Sì	Sì	Gratuito	Windows*	C#	Media
Phaser	Sì	No*	No	Gratuito	PC, Mobile, Web	JavaScript, Typescript	Media

Figura 3.1: Tabella riassuntiva delle caratteristiche principali dei motori di gioco.

3.1 Unity

Unity è uno dei motori di gioco più popolari e versatili sul mercato, conosciuto per la capacità di adattarsi a una vasta gamma di progetti, dai semplici giochi 2D ai complessi ambienti tridimensionali.

Lanciato nel 2005 da Unity Technologies per essere un motore dedicato a Mac OS X, è cresciuto e ha esteso il supporto a svariate piattaforme desktop, mobile, di realtà aumentata e virtuale. Oggi, Unity è una soluzione multipiattaforma in grado di raggiungere quasi tutte le principali piattaforme hardware, inclusi dispositivi emergenti come Apple Vision Pro.

Uno dei punti di forza principali di Unity è la sua flessibilità nello sviluppo sia in 2D che in 3D, rendendolo ideale per una varietà di generi videoludici, come platform, giochi di simulazione, puzzle e persino applicazioni non ludiche. Il motore supporta il visual scripting attraverso strumenti come Bolt, che permette anche agli sviluppatori senza esperienza di programmazione di creare facilmente meccaniche di gioco personalizzate. Inoltre, Unity dispone di una vasta libreria di asset e plugin pronti all'uso, disponibili nel suo Asset Store, che facilita lo sviluppo rapido e riduce i tempi di produzione.

Unity è anche noto per la sua vasta comunità di supporto e per l'ampia disponibilità di risorse online, tra cui tutorial, corsi di formazione e documentazione dettagliata.

La sua gestione delle risorse di progetto, specialmente in applicazioni di grandi dimensioni, può però risultare poco efficiente, causando rallentamenti se non viene ottimizzata correttamente. Inoltre, sebbene sia una soluzione potente per il 3D, non raggiunge ancora il livello di dettaglio e realismo nativo di motori come Unreal Engine, soprattutto in progetti AAA.

In sintesi, Unity è una piattaforma versatile e adattabile, capace di trasformarsi per soddisfare le esigenze di progetti e team di ogni tipo. Grazie al suo equilibrio tra accessibilità, flessibilità e potenza tecnica, rappresenta una scelta solida tanto per sviluppatori indipendenti quanto per produzioni più ambiziose.

Installazione: Unity può essere installato tramite Unity Hub, uno strumento centralizzato che gestisce versioni e progetti. Dopo aver scaricato Unity Hub, basta selezionare la versione desiderata e i moduli opzionali (come Android o iOS) per iniziare lo sviluppo.

3.2 Unreal Engine

Unreal Engine, sviluppato da Epic Games, è uno dei motori di gioco più avanzati e potenti disponibili sul mercato, ampiamente utilizzato per la creazione di giochi AAA e applicazioni interattive di alto livello. Introdotto per la prima volta nel 1998 con il gioco *Unreal*, ha subito continue evoluzioni, rendendolo una scelta ideale per sviluppatori che puntano a risultati visivi spettacolari.

Uno degli aspetti distintivi di Unreal Engine è la sua capacità di gestire grafica ad alta fedeltà grazie a tecnologie avanzate come Nanite, che consente la gestione efficiente di geometrie complesse, e Lumen, un sistema di illuminazione globale dinamica. Queste caratteristiche lo rendono perfetto per la creazione di mondi virtuali dettagliati e immersivi. Inoltre, Unreal Engine offre un sistema di scripting visivo chiamato Blueprint, che

permette anche agli sviluppatori senza conoscenze approfondite di programmazione di creare meccaniche di gioco complesse in modo intuitivo.

Il motore è compatibile con una vasta gamma di piattaforme, inclusi PC, console, dispositivi mobili e VR, garantendo elevata portabilità e flessibilità. Grazie alla sua vasta comunità di sviluppatori e alla documentazione completa, offre supporto continuo e una ricca selezione di risorse.

Unreal Engine è particolarmente indicato per progetti che richiedono un elevato livello di dettaglio grafico, simulazioni realistiche e ambienti open-world. Tuttavia, la sua complessità e la curva di apprendimento piuttosto ripida possono rappresentare una sfida per i team meno esperti. In sintesi, è una scelta eccellente per produzioni ambiziose e per chi desidera sfruttare al massimo le potenzialità tecnologiche offerte dal settore.

Installazione: Unreal Engine può essere scaricato tramite Epic Games Launcher, che consente la gestione delle versioni del motore e l'accesso a una vasta libreria di asset. Dopo aver installato il launcher, è possibile scegliere la versione desiderata e iniziare subito a sviluppare.

3.3 Godot 

Godot è un motore di gioco open-source e gratuito, sviluppato per fornire un ambiente di sviluppo flessibile e accessibile. Introdotto nel 2014, è particolarmente apprezzato dagli sviluppatori indipendenti per la sua leggerezza e semplicità d'uso. Distribuito sotto licenza MIT, permette una completa personalizzazione del codice sorgente, rendendolo adatto a progetti sia di piccola che di media scala.

Uno dei principali punti di forza di Godot è la sua capacità di gestire sia giochi 2D che 3D. Tuttavia, eccelle soprattutto nello sviluppo di giochi 2D, grazie a un sistema di rendering ottimizzato e nativo, che evita molte delle limitazioni tipiche di altri motori che adattano il 3D per creare giochi bidimensionali. Il linguaggio di scripting principale è GDScript, simile a Python, il che lo rende facile da apprendere e da utilizzare anche per chi non ha esperienza avanzata in programmazione. Godot supporta anche C# e VisualScript, offrendo ulteriore flessibilità per gli sviluppatori.

La sua interfaccia utente intuitiva e la disponibilità di numerose risorse online, come tutorial e guide, rendono il motore accessibile a una vasta comunità di utenti. Inoltre, Godot offre strumenti integrati per la creazione di animazioni, la gestione della fisica e il networking, senza richiedere plugin o estensioni esterne.

Nonostante le sue numerose qualità, Godot presenta ancora alcune limitazioni, soprattutto nel rendering 3D avanzato e nella gestione di progetti di grandi dimensioni. Tuttavia, grazie alla sua natura open-source e alla comunità in crescita, il motore continua a migliorare rapidamente. In sintesi, Godot rappresenta una soluzione eccellente per sviluppatori indipendenti e piccoli team alla ricerca di uno strumento flessibile e gratuito.

Installazione: Godot può essere scaricato direttamente dal sito ufficiale godotengine.org. Non richiede installazione: è sufficiente scaricare l'eseguibile per iniziare a sviluppare.

3.4 CryEngine 

CryEngine, sviluppato da Crytek, è stato introdotto nel 2002 con il gioco *Far Cry*. Fin dall'inizio, il motore ha fissato nuovi standard nel settore grazie alle sue capacità grafiche avanzate. Con il rilascio della serie *Crysis*, CryEngine ha consolidato la sua reputazione diventando il benchmark di riferimento per le prestazioni grafiche, al punto da generare il celebre quesito: *"But can it run Crysis?"*.

Le funzionalità avanzate di CryEngine includono la simulazione fisica avanzata tramite CryPhysics, il rendering in tempo reale e il supporto per ambienti open-world dettagliati e complessi. Grazie al suo sistema di illuminazione dinamica e al rendering di materiali altamente realistici, il motore è in grado di offrire esperienze visive immersive e di grande impatto.

Un altro elemento distintivo è la capacità di CryEngine di integrare tecnologie avanzate come il ray tracing e la gestione di ombre e riflessi in tempo reale, mantenendo prestazioni elevate su hardware di fascia alta. Inoltre, gli strumenti per l'intelligenza artificiale e le animazioni integrate permettono agli sviluppatori di gestire complesse interazioni tra personaggi e ambienti.

Nonostante la sua curva di apprendimento ripida e i requisiti hardware elevati, CryEngine rimane una scelta eccellente per progetti AAA che richiedono un alto livello di dettaglio e realismo grafico, rendendolo il motore preferito per giochi ambiziosi e simulazioni visive all'avanguardia.

Installazione: CryEngine può essere scaricato tramite il CryEngine Launcher, che consente di gestire versioni, asset e progetti in modo centralizzato. Dopo aver installato il launcher, è possibile avviare immediatamente lo sviluppo del progetto.

3.5 Open 3D Engine (O3DE) 

Open 3D Engine è un motore di gioco 3D gratuito e open-source sviluppato dalla Open 3D Foundation, parte della Linux Foundation, e distribuito sotto licenza Apache 2.0. Derivato da Amazon Lumberyard, O3DE rappresenta un'evoluzione significativa con miglioramenti che lo rendono adatto a progetti ambiziosi e complessi.

Uno dei principali punti di forza di O3DE è il suo sistema fortemente modulare, che permette agli sviluppatori di selezionare solo i componenti necessari per il proprio progetto, ottimizzando le prestazioni e riducendo il peso complessivo. Il motore offre funzionalità avanzate per il rendering di ambienti open-world, supporto multiplayer e integrazione diretta con i servizi cloud di Amazon Web Services (AWS).

Nonostante il suo grande potenziale, O3DE è ancora relativamente nuovo e presenta una curva di apprendimento elevata, con una comunità in fase di sviluppo che sta crescendo progressivamente.

Installazione: O3DE può essere scaricato dal sito ufficiale o3de.org. La configurazione iniziale permette di personalizzare l'installazione selezionando i moduli e le funzionalità desiderate.

3.6 GameMaker 

GameMaker, sviluppato originariamente da Mark Overmars e ora gestito da YoYo Games, è noto per la sua semplicità e accessibilità. Questo motore si distingue per il suo sistema drag-and-drop intuitivo, che permette anche a chi ha poca o nessuna esperienza di programmazione di creare giochi funzionali in tempi brevi.

Ideale per lo sviluppo di giochi 2D, GameMaker offre un linguaggio di scripting proprietario chiamato GML (GameMaker Language), con la sintassi basata sull'unione dei linguaggi Delphi, Java, Pascal, C e C++, che consente agli utenti più esperti di personalizzare e ottimizzare le meccaniche di gioco. Grazie al suo supporto per piattaforme desktop, mobile e console, è particolarmente apprezzato dagli sviluppatori indipendenti per la prototipazione rapida e la creazione di titoli commerciali.

Installazione: GameMaker può essere scaricato dal sito ufficiale gamemaker.io. L'installazione è semplice e permette di scegliere tra diverse opzioni di licenza, inclusa una versione gratuita per l'uso personale.

3.7 GDevelop 

GDevelop è un motore di gioco open-source pensato per lo sviluppo di giochi 2D e semplici giochi 3D. La sua principale caratteristica è l'accessibilità, grazie a un sistema di sviluppo senza codice basato su eventi visivi, ideale per chi non ha esperienza di programmazione.

Il motore offre un'interfaccia intuitiva e un'ampia libreria di esempi e risorse pronte all'uso, che permettono di creare giochi per piattaforme desktop, mobile e web in modo rapido. GDevelop è particolarmente adatto per progetti indie, prototipi veloci e giochi educativi.

Nonostante le sue limitazioni nelle prestazioni avanzate e nella grafica complessa, è un'ottima opzione per progetti leggeri e per sviluppatori che desiderano iniziare senza una curva di apprendimento impegnativa.

Installazione: GDevelop può essere scaricato gratuitamente dal sito ufficiale gdevelop.io. L'installazione è semplice e permette di iniziare subito la creazione di nuovi progetti.

3.8 Construct3 

Construct3 è un motore di gioco basato su browser progettato per semplificare lo sviluppo di giochi 2D. Grazie alla sua interfaccia intuitiva e al sistema di sviluppo senza codice basato su blocchi visivi, è ampiamente utilizzato da sviluppatori indipendenti ed educatori.

Uno dei principali vantaggi di Construct3 è la sua natura multipiattaforma: i giochi possono essere esportati per dispositivi mobili, desktop e web velocemente. Il motore offre inoltre una vasta gamma di funzionalità integrate, come supporto per fisica, animazioni e gestione degli input, senza la necessità di plug-in aggiuntivi.

Nonostante le sue limitazioni nei progetti complessi e nella gestione di grafica avanzata, Construct 3 è una soluzione ideale per prototipi veloci e giochi 2D leggeri.

Installazione: Construct 3 è accessibile direttamente da browser senza necessità di installazione. Visita construct.net per iniziare subito a creare giochi.

3.9 Scratch 

Scratch è un ambiente di programmazione visiva gratuito progettato per avvicinare i bambini e i principianti al mondo dello sviluppo software e del game design. Sviluppato dal MIT Media Lab, Scratch è noto per il suo approccio intuitivo basato su blocchi visivi, che elimina la necessità di scrivere codice tradizionale.

Uno dei punti di forza di Scratch è la sua interfaccia drag-and-drop, che consente agli utenti di creare rapidamente storie interattive, giochi e animazioni. Il motore supporta una vasta gamma di funzionalità, come suoni, animazioni, interazioni e rilevamento delle collisioni, rendendolo ideale per la prototipazione rapida e per progetti educativi.

Scratch è anche una piattaforma sociale: i progetti possono essere condivisi online sulla comunità ufficiale, dove gli utenti possono collaborare, commentare e riutilizzare i progetti di altri.

Installazione: Scratch è disponibile sia come applicazione online all'indirizzo scratch.mit.edu che come applicazione desktop scaricabile per Windows, macOS e dispositivi mobili.

3.10 Cocos2d-x 

Cocos2d-x è un framework di sviluppo open-source progettato per la creazione di giochi 2D e applicazioni interattive. È l'evoluzione di Cocos2d, originariamente sviluppato in Python, ed è stato riscritto in C++ per offrire migliori prestazioni e supporto multipiattaforma.

Grazie al suo core in C++, Cocos2d-x è altamente efficiente e leggero, rendendolo ideale per dispositivi mobili, browser e piattaforme desktop come Windows, MacOS e Linux. Il framework offre strumenti avanzati per la gestione di grafica, animazioni, suoni e fisica, semplificando il processo di sviluppo mantenendo alte prestazioni.

Cocos2d-x supporta anche scripting in JavaScript e Lua, offrendo agli sviluppatori maggiore flessibilità nella scelta del linguaggio. La community attiva e le numerose risorse online contribuiscono a rendere Cocos2d-x una scelta popolare sia tra sviluppatori indie che professionisti.

Installazione: Cocos2d-x può essere scaricato dal sito ufficiale cocos.com. L'installazione richiede la configurazione di un ambiente di sviluppo completo, con supporto per strumenti come Android Studio e Xcode per lo sviluppo mobile.

3.11 Defold 

Defold è un motore di gioco gratuito e leggero, progettato principalmente per lo sviluppo di giochi 2D, ma con supporto limitato per progetti 3D. Il motore si distingue per la

sua efficienza, offrendo prestazioni elevate anche su dispositivi mobili grazie alla sua architettura ottimizzata.

Un aspetto notevole di Defold è il suo sistema di sviluppo basato su componenti, che permette agli sviluppatori di costruire rapidamente meccaniche di gioco modulari. Inoltre, Defold fornisce un supporto completo per esportazioni su diverse piattaforme, tra cui desktop, mobile e web.

Sebbene la comunità sia più piccola rispetto ad altri motori, la documentazione dettagliata e il supporto attivo degli sviluppatori ne facilitano l'apprendimento e l'utilizzo.

Installazione: Defold può essere scaricato dal sito ufficiale defold.com.

3.12 Stride

Stride, precedentemente noto come Xenko, è un motore di gioco open-source focalizzato sullo sviluppo 3D con supporto per C#. Sviluppato inizialmente da Silicon Studio e ora gestito dalla comunità, si distingue per il suo approccio modulare e flessibile, che permette agli sviluppatori di includere solo le funzionalità necessarie al progetto. Il motore è adatto sia per giochi leggeri che per progetti più complessi, grazie a un'architettura pulita e orientata agli oggetti. Stride supporta rendering avanzato, animazioni, fisica e post-processing, il tutto integrato in un'interfaccia utente intuitiva. Sebbene abbia una community più piccola rispetto ad altri motori, la sua natura open-source consente un'ampia personalizzazione.

Installazione: Stride può essere scaricato dal sito ufficiale stride3d.net. È necessario disporre di un ambiente di sviluppo C# come Visual Studio per iniziare.

3.13 Phaser

Phaser è un framework open-source per lo sviluppo di giochi 2D in HTML5, ideale per giochi destinati al web. È molto apprezzato per la sua semplicità e leggerezza, che lo rendono una scelta popolare tra gli sviluppatori indie e gli educatori.

Phaser include funzionalità per la gestione della fisica, delle animazioni, degli input utente e della grafica, senza richiedere l'installazione di ambienti complessi. Supporta il rendering sia su canvas che su WebGL, garantendo prestazioni ottimali su diverse piattaforme.

Installazione: Phaser può essere scaricato dal sito ufficiale phaser.io o incluso direttamente in progetti web tramite un CDN. Non richiede ambienti di sviluppo specifici: è sufficiente un editor di codice e un browser web per testare i progetti.

Capitolo 4

Analisi comparativa di motori di gioco

In questo capitolo verranno analizzati i principali motori di gioco attraverso una serie di criteri chiave, che permetteranno di evidenziarne i punti di forza e le debolezze in modo strutturato. L'obiettivo è quello di offrire una panoramica chiara e comparativa che possa guidare nella scelta del motore più adatto alle proprie esigenze.

I criteri selezionati derivano da un'analisi della letteratura sui game engine, da best practice adottate nell'industria e da considerazioni legate alle esigenze concrete degli sviluppatori. In particolare, si è tenuto conto di aspetti fondamentali che incidono sulla scelta del motore in base alla tipologia di progetto, alla curva di apprendimento e alle funzionalità offerte.

I criteri di confronto scelti includono:

- **Costi**: sia in termini di licenze che di eventuali royalties.
- **Features**: le funzionalità incluse, come la gestione delle texture, della fisica, dell'audio, e altro ancora, con un'analisi influenzata dal tipo di gioco che si desidera sviluppare (2D, 3D, platform, ecc.).
- **Facilità d'uso e supporto**: valutazione della facilità d'uso del motore di gioco, considerando quanto sia semplice iniziare a utilizzarlo e il tempo necessario per padroneggiarne le funzionalità. Inoltre, si tiene conto della presenza di una comunità attiva, di risorse online per risolvere problemi tecnici e trovare ispirazione, nonché della qualità e completezza della documentazione ufficiale.
- **Portabilità**: la capacità del motore di supportare diverse piattaforme, come PC, console e dispositivi mobili.
- **Linguaggio di programmazione**: i linguaggi supportati e la loro accessibilità per gli sviluppatori.
- **Adattabilità alla tipologia di gioco**: valutazione delle scelte di motori più adatti per ogni macro-categoria di giochi.
- **Flessibilità e modularità**: la possibilità di personalizzare o espandere il motore, ad esempio attraverso plug-in o modifiche al codice sorgente.
- **Accessibilità**: valutazione della presenza di strumenti e funzionalità che permettono di sviluppare giochi accessibili a persone con disabilità.

Questa analisi comparativa fornirà una base solida per comprendere le differenze tra i vari motori e aiutare a scegliere quello più adatto alle specifiche esigenze di sviluppo.

4.1 Costi

Il costo di utilizzo di un motore di gioco è un fattore cruciale nella scelta della piattaforma più adatta. In questa sezione verranno esaminati i modelli di prezzo dei principali motori di gioco:

Motore	Modello di Pagamento	Costo Base	Commissioni	Note
Unity	Gratuito fino a 200K$/anno, poi abbonamento	Gratuito sotto 200K$/anno, poi 2200$/anno per utente	Sopra 1M$, costo basato sull'engagement	Modello di pagamento aggiornato nel 2024; la Runtime Fee è stata cancellata
Unreal Engine	Gratuito fino a 1M$ di entrate, poi royalty	Gratis fino a 1M$	Sopra 1M$, 5% royalties	Accesso al codice sorgente incluso; esenzione royalty per vendite su Epic Games Store
CryEngine	Royalty sui guadagni del gioco	Gratis fino a 5.000$	5% sui ricavi lordi oltre 5.000$/anno	Esenzione royalty per vendite su CRYTEK Marketplace
GameMaker	Licenza fissa o abbonamento per le aziende	99,99$ una tantum per uso pro, 800$ l'anno per aziende	Nessuna commissione sulle entrate	Esportazione su console richiede abbonamento Enterprise (per aziende)
Construct3	Abbonamento	$129,99/anno per uso personale, 429$/anno per aziende	Nessuna commissione sulle entrate	-
Gdevelop	Open-source gratuito con piani premium per servizi aggiuntivi	Gratuito; piani Silver a €4,99/mese, Gold a €9,99/mese e Pro a €30/mese	Nessuna commissione sulle entrate	I piani premium offrono funzionalità aggiuntive come esportazioni giornaliere e rimozione del logo
Godot	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	Supportato da donazioni; nessun costo diretto; licenza MIT
O3DE	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	probabili costi indiretti derivanti da plugin di terze parti per funzionalità avanzate; poche feature native
Scratch	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	-
Cocos2d-x	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	-
Defold	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	-
Stride	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	-
Phaser	Completamente gratuito, open-source	Gratuito	Nessuna commissione sulle entrate	-

Figura 4.1: Tabella riassuntiva dei costi dei motori di gioco.

Dalla tabella si può notare che esistono due principali modelli di pagamento adottati dai motori di gioco:

- Licenza gratuita con royalty o abbonamento dopo una certa soglia di guadagno (es. Unreal Engine, Unity, CryEngine).
- Completamente gratuito o con opzioni premium per funzionalità avanzate (es. Godot, O3DE, Defold, Phaser).

Motori gratuiti e open-source: Motori come Godot, O3DE, Cocos2d-x ed altri sono completamente gratuiti e open-source, il che li rende ideali per sviluppatori indipendenti o aziende che vogliono avere un controllo totale senza vincoli finanziari. Tuttavia, possono avere limitazioni in termini di supporto ufficiale o necessitare di strumenti di terze parti per determinate funzionalità.

Motori con modello freemium o abbonamento: Piattaforme come GDevelop, Construct3 e GameMaker offrono un piano gratuito con funzionalità limitate e richiedono un abbonamento per accedere a funzionalità avanzate o per esportare il gioco senza watermark. Questo modello è conveniente per chi sviluppa giochi senza grandi investimenti iniziali, ma può diventare costoso nel lungo periodo.

Motori con royalty sulle vendite: Unreal Engine e CryEngine adottano un modello di royalty, dove gli sviluppatori possono usare il motore gratuitamente fino a un certo limite di guadagno, dopodiché devono pagare una percentuale sulle vendite. Questo può essere vantaggioso per chi sviluppa giochi di successo, ma meno conveniente per team con margini di profitto ridotti.

La scelta del motore dipende dal budget, dalla scalabilità del progetto e dal modello di monetizzazione desiderato. Per giochi indie o sperimentali, Godot e O3DE offrono la massima libertà economica. Per produzioni AAA con elevate esigenze grafiche, Unreal Engine e CryEngine garantiscono tecnologie avanzate a fronte di royalty. Infine, motori con abbonamento come GameMaker o Construct3 sono adatti a chi cerca un'interfaccia user-friendly con strumenti integrati.

4.2 Features

Le funzionalità offerte dai motori di gioco rappresentano un fattore chiave nella scelta della piattaforma più adatta allo sviluppo. Ogni motore include strumenti specifici per il rendering, la fisica, l'intelligenza artificiale e altre componenti essenziali per la creazione di videogiochi. In questa sezione verranno analizzate e confrontate le principali features dei motori di gioco selezionati.

Categorie di funzionalità Le funzionalità dei motori di gioco possono essere suddivise nelle seguenti categorie principali:

- **Rendering**: Tecnologie di illuminazione, shader, supporto per ray tracing, pipeline grafica.
- **Fisica**: Supporto per motori fisici avanzati, gestione delle collisioni, simulazioni di corpi rigidi e deformabili.
- **Animazione**: Supporto per animazioni scheletriche, cinematica inversa, morph targets.
- **Audio**: Gestione del suono 3D, effetti sonori dinamici, compatibilità con middleware audio.
- **Networking**: Supporto per multiplayer, sincronizzazione in rete, servizi cloud.
- **VR/AR**: Compatibilità con dispositivi di realtà virtuale e aumentata.
- **Modularità**: Possibilità di espandere le funzionalità con plugin e API esterne.

4.2.1 Rendering

Il rendering è una componente fondamentale dei motori di gioco, influenzando direttamente la qualità visiva e le prestazioni del titolo sviluppato.

In questa sezione verranno esaminate le tecniche di rendering implementate dai principali motori di gioco. Alcuni concetti tecnici fondamentali, come le fasi della pipeline di rendering e le tecnologie di base, sono già stati trattati nella sezione 2.2.1. Il ray tracing, menzionato brevemente qui per contestualizzare le tecnologie moderne di rendering, verrà invece analizzato in modo approfondito nel Capitolo 5.

Unreal Engine

Unreal Engine è considerato il numero uno dal punto di vista grafico, noto per il suo avanzato sistema di rendering, che consente di ottenere grafica fotorealistica e prestazioni elevate grazie a tecnologie all'avanguardia. Tra le sue caratteristiche principali troviamo:

Nanite Nanite è il sistema di geometria virtualizzata introdotto in Unreal Engine 5, progettato per gestire dettagli geometrici estremamente complessi in tempo reale senza compromettere le prestazioni.

Il funzionamento generale di Nanite, quando viene attivato, è quello di prendere le mesh già presenti, analizzarle e suddividerle in cluster gerarchici di triangoli (vedi Figura 4.2), comprimerle pesantemente e creare automaticamente un sistema di rappresentazione progressiva della geometria. Questo elimina la necessità di creare manualmente livelli di dettaglio (LOD), poiché il motore carica dinamicamente solo i dettagli visibili all'utente in base alla distanza e all'angolazione di visualizzazione.

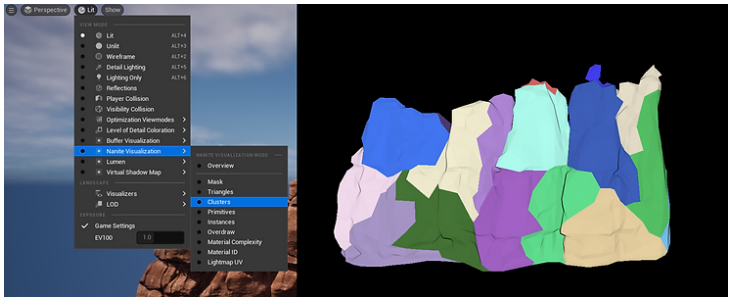

Figura 4.2: Esempio di cluster di triangoli in Nanite

Tuttavia, è importante notare che Nanite potrebbe avere limitazioni con geometrie aggregate, come capelli, fogliame o erba, dove dettagli molto fini possono non essere gestiti in modo ottimale. Inoltre, Nanite è attualmente compatibile solo con mesh statiche e potrebbe non supportare correttamente superfici strettamente sovrapposte.

Nonostante queste limitazioni, Nanite rappresenta un significativo passo avanti nella gestione della complessità geometrica nei motori di gioco, offrendo un equilibrio tra qualità visiva e prestazioni[4][5][6].

Lumen Lumen è il sistema di illuminazione globale dinamica introdotto in Unreal Engine 5, progettato per fornire un'illuminazione indiretta realistica in tempo reale senza la necessità di pre-calcoli delle luci.

Figura 4.3: Esempio di come Lumen migliori il realismo di superfici riflettenti come specchi

Questo sistema simula il comportamento della luce che rimbalza sulle superfici, catturando e riflettendo i colori circostanti, fenomeno noto come *color bleeding*. Lumen supporta infiniti rimbalzi diffusi, essenziali per scene con materiali altamente riflettenti o ambienti con illuminazione complessa. Inoltre, gestisce efficacemente l'illuminazione proveniente da materiali emissivi, permettendo a oggetti come schermi o luci al neon di illuminare dinamicamente l'ambiente circostante senza costi computazionali aggiuntivi.

Lumen utilizza una combinazione di **software ray tracing** e **hardware ray tracing** se disponibile, adattandosi automaticamente alle capacità dell'hardware per garantire la

migliore qualità possibile senza compromettere le prestazioni. Un aspetto chiave di Lumen è la sua integrazione con altre tecnologie di Unreal Engine 5, come Nanite, consentendo di mantenere un'elevata qualità visiva anche in mondi di grandi dimensioni e altamente dettagliati.

Tuttavia, Lumen presenta alcune limitazioni. In particolare, ha difficoltà a gestire superfici altamente riflettenti che non sono perfettamente planari, come quelle curve.

La sua flessibilità rende Lumen particolarmente adatto per giochi e applicazioni interattive che richiedono ambienti dinamici, reattivi e altamente realistici[7][8].

Supporto al Ray Tracing Unreal Engine offre un supporto avanzato per il ray tracing, consentendo di creare scene con illuminazione, ombre e riflessi altamente realistici in tempo reale. Il ray tracing in Unreal Engine è progettato per essere flessibile, permettendo agli sviluppatori di combinare tecniche di rendering tradizionali (rasterizzazione) con il ray tracing. Questo approccio ibrido consente di bilanciare qualità visiva e prestazioni, applicando il ray tracing solo agli elementi che ne traggono maggior beneficio, come riflessi accurati o ombre dettagliate[9][10].

Temporal Super Resolution (TSR) Il Temporal Super Resolution (TSR) è una tecnologia di upscaling temporale integrata in Unreal Engine 5, progettata per migliorare la qualità dell'immagine rendendo possibile il rendering di scene in alta risoluzione, come il 4K, mantenendo elevate le prestazioni. TSR è indipendente dalla piattaforma, il che significa che può essere utilizzato su diverse configurazioni hardware senza richiedere componenti specifici[11].

Figura 4.4: Esempio di frame prima e dopo l'attivazione di TSR

TSR opera raccogliendo informazioni da frame precedenti e combinandole per produrre un'immagine finale più dettagliata. Sfrutta tre principali componenti:

1. Accumulo Temporale (Temporal Accumulation): UE raccoglie informazioni da frame precedenti e utilizza questi dati per ricostruire dettagli mancanti nel frame corrente.

2. Motion Vectors (Vettori di Movimento): Il TSR utilizza vettori di movimento per prevedere la posizione futura degli oggetti. Se un oggetto si sposta, TSR confronta i suoi dettagli nel frame precedente e nel frame attuale, correggendo eventuali distorsioni o artefatti causati dal movimento.

3. Reprojection & Adaptive Feedback: Dopo l'analisi dei frame precedenti, TSR proietta i dettagli delle immagini passate sul frame attuale. Questo permette di migliorare il livello di dettaglio (LOD) e ridurre il cosiddetto ghosting, ovvero la presenza di tracce di movimento indesiderate.

Sistema di Materiali e Shading Il sistema di materiali e shading in Unreal Engine è progettato per creare superfici realistiche e stilizzate in modo intuitivo, grazie a un editor visivo a nodi. Questo permette di combinare texture, colori e proprietà fisiche senza dover scrivere codice shader complesso[12].

Unreal Engine utilizza un approccio **Physically Based Rendering (PBR)**, il che significa che i materiali reagiscono realisticamente alla luce, garantendo un aspetto naturale in ogni condizione di illuminazione. Esistono diversi modelli di shading a seconda del tipo di superficie da rappresentare: dai materiali opachi standard a quelli traslucidi, passando per effetti come verniciature lucide o superfici che diffondono la luce, come la pelle umana.

Con l'introduzione di Unreal Engine 5.2, il Substrate Material Framework ha reso la gestione dei materiali ancora più flessibile, permettendo di combinare più strati per ottenere superfici complesse, come sporco sovrapposto a metallo o verniciature consumate.

Grazie a queste tecnologie, Unreal Engine si posiziona come uno dei motori di rendering più potenti e versatili disponibili, ideale per lo sviluppo di giochi AAA.

Unity

Unity offre un'ampia gamma di opzioni per il rendering sia in 2D che in 3D. Grazie alla sua struttura modulare e alla possibilità di scegliere tra diverse pipeline di rendering, Unity si adatta a una vasta gamma di progetti, dai giochi mobile alle produzioni AAA.

Pipeline di Rendering Unity offre tre principali pipeline di rendering, ciascuna progettata per specifici casi d'uso e livelli di prestazioni. La scelta della pipeline giusta dipende dal tipo di progetto e dal livello di qualità visiva richiesto[13].

- **Built-in Render Pipeline (Pipeline Tradizionale)**: È la pipeline di rendering predefinita di Unity, utilizzata nei progetti più vecchi, limitata rispetto alle altre due pipeline in termini di ottimizzazioni e moderni effetti visivi, ma altamente personalizzabile.

- **Universal Render Pipeline (URP)**: Progettata per essere più efficiente rispetto alla pipeline tradizionale, con supporto per dispositivi mobile, console e PC. Utilizza un sistema basato su scriptable render passes, offrendo un miglior controllo sul rendering. Ottima per giochi 2D, VR e AR, grazie alla sua leggerezza.

- **High Definition Render Pipeline (HDRP)**: Pensata per giochi con grafica avanzata e qualità cinematografica. Supporta illuminazione globale, ray tracing e avanzate tecniche di post-processing. Non compatibile con mobile e VR, dato l'elevato costo computazionale.

Quale Pipeline scegliere?

- **Per progetti già esistenti o con shader personalizzati**, potrebbe essere preferibile mantenere la **Built-in Render Pipeline**.
- **Per giochi mobile, VR o 2D**, la scelta ideale è **Universal Render Pipeline (URP)**.
- **Per giochi realistici e con grafica avanzata**, è consigliato l'uso della **High Definition Render Pipeline (HDRP)**.

Illuminazione Unity offre supporto al ray tracing tramite **DirectX 12**, permettendo di ottenere un elevato livello di realismo nella gestione della luce, delle ombre e dei riflessi.

Tra le principali funzionalità del ray tracing in Unity troviamo riflessi realistici basati sui materiali di riflessione, illuminazione globale con propagazione della luce indiretta, ombre accurate che variano in nitidezza in base alla distanza della sorgente luminosa e occlusione ambientale avanzata.
L'illuminazione in Unity può essere gestita attraverso diverse tecniche, ognuna con vantaggi specifici in termini di realismo e prestazioni:

- **Illuminazione Precalcolata (Light Baking)**: Questo metodo calcola l'illuminazione globale in fase di sviluppo e la memorizza in mappe di luce (lightmaps). È una soluzione altamente ottimizzata, ideale per ambienti statici in cui l'illuminazione non deve cambiare dinamicamente, come nei giochi di strategia o nei simulatori con ambienti predefiniti.
- **Illuminazione Dinamica**: Permette alla luce di interagire in tempo reale con l'ambiente e gli oggetti, adattandosi alle variazioni della scena. Questo metodo è fondamentale per giochi con giorno/notte dinamico, effetti di luci in movimento o ambienti in cui gli oggetti cambiano posizione frequentemente. Tuttavia, è più pesante in termini di prestazioni.
- **Mixed Lighting**: Un compromesso tra le precedenti, in cui alcune componenti della luce vengono precalcolate mentre altre vengono calcolate in tempo reale. Questa tecnica è utile per bilanciare qualità e performance.

Temporal Anti-Aliasing (TAA) e Upscaling Unity include diverse soluzioni per migliorare la qualità visiva senza sacrificare le prestazioni:

- Temporal Anti-Aliasing (TAA): è una tecnica avanzata di riduzione dell'aliasing che utilizza informazioni da più frame precedenti per migliorare la qualità dell'immagine, riducendo aliasing spaziale e flickering su superfici sottili o dettagliate.
- NVIDIA DLSS (Deep Learning Super Sampling): Permette l'upscaling basato su AI (più dettagli nel capitolo 5).
- AMD FSR (FidelityFX Super Resolution): Alternativa a DLSS per hardware non NVIDIA.

Shader e Materiali Unity utilizza il sistema di materiali basato su Shader Graph, che consente di creare shader personalizzati tramite un'interfaccia visuale. Questo permette di generare effetti complessi senza dover scrivere codice shader manualmente. Per gli sviluppatori avanzati, Unity supporta anche HLSL, linguaggio sviluppato da Microsoft per la creazione di shader personalizzati da usare in DirectX.

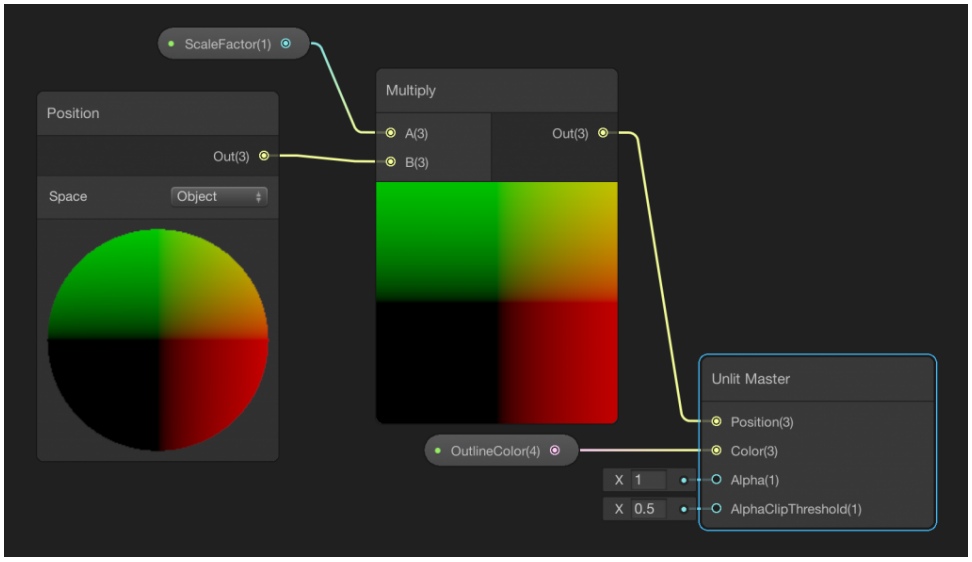

Figura 4.5: Shader Graph in Unity

Ottimizzazioni e Tecniche di Rendering Unity include diverse tecniche di ottimizzazione per garantire un rendering efficiente tra cui occlusion culling, LOD e Dynamic Batching e GPU Instancing che migliora le prestazioni combinando oggetti simili in un unico draw call.

Grazie a queste caratteristiche, Unity rappresenta una scelta versatile per lo sviluppo di giochi e applicazioni interattive, bilanciando qualità visiva e prestazioni.

CryENGINE

CryEngine è noto per il suo motore di rendering avanzato, progettato per offrire grafica fotorealistica con particolare attenzione alla gestione dell'illuminazione globale e degli ambienti open-world.

Sistema di Illuminazione e SVOGI CryEngine utilizza la tecnologia **SVOGI (Sparse Voxel Octree Global Illumination)**, un sistema di illuminazione globale che consente di ottenere rimbalzi di luce in tempo reale senza la necessità di pre-baking. Questo approccio garantisce un'illuminazione dinamica e realistica, rendendo CryEngine particolarmente adatto per giochi con ambientazioni naturali o open-world. Si noti come nella Figura 4.6 la stanza con sistema SVOGI attivato viene illuminata anche in mancanza di una sorgente di luce diretta, per riflessione di luce da altre superfici.

Ray Tracing e Rendering Basato sulla Fisica Il motore supporta il ray tracing in tempo reale, migliorando la qualità di riflessi e ombre senza l'uso di cubemap statiche

Figura 4.6: Stessa scena renderizzata con sistema SVOGI disattivato (sinistra) e attivato (destra).

o tecniche approssimative. Inoltre, utilizza un sistema di Physically Based Rendering, garantendo materiali realistici con interazioni accurate tra luce e superfici.

Uno dei punti di forza di CryEngine è la capacità di renderizzare ambienti naturali con dettagli estremamente elevati. Il motore include un sistema avanzato di gestione della vegetazione, con tecniche di wind simulation, occlusione e LOD dinamici.

Tecniche di Post-Processing CryEngine implementa una serie di effetti avanzati di post-processing, tra cui:

- Depth of Field avanzato, con controllo dinamico della sfocatura in base alla distanza.
- Motion Blur e Temporal Anti-Aliasing (TAA) per una maggiore fluidità dell'immagine.
- HDR per una gestione accurata dei colori e della luminosità in ambienti realistici.

Ottimizzazione e Performance CryEngine è ottimizzato per sfruttare al meglio le GPU moderne, con un sistema di culling avanzato che evita il rendering di oggetti non visibili e tecniche di tiled shading per migliorare l'efficienza dell'illuminazione. Grazie a queste caratteristiche, il motore è in grado di offrire una qualità visiva di alto livello mantenendo un buon compromesso tra prestazioni e fedeltà grafica.

CryEngine continua a essere una scelta eccellente per giochi che richiedono ambienti vasti e dettagliati, garantendo un rendering di altissima qualità con tecnologie di illuminazione e materiali all'avanguardia.

4.2.2 Fisica

La simulazione fisica è uno degli elementi chiave nei motori di gioco moderni, contribuendo in modo significativo al realismo e all'interattività dei mondi virtuali. Come discusso nella sezione 2.2.2, la complessità e la precisione delle simulazioni fisiche possono variare notevolmente tra i diversi motori, influenzando l'esperienza di gioco finale.

Alcuni motori di gioco hanno sviluppato sistemi di fisica proprietari avanzati, offrendo funzionalità dedicate e profondamente integrate nell'ecosistema del motore stesso. Esempi notevoli sono *Chaos Physics* in Unreal Engine e *CryPhysics* in CryEngine, che offrono simulazioni ad alte prestazioni e grande realismo. Altri motori, come Unity e O3DE, si affidano a middleware consolidati come NVIDIA PhysX, sfruttandone la potenza e la flessibilità. Infine, motori più leggeri o specializzati in giochi 2D adottano sistemi di fisica semplificati, adatti a progetti meno complessi ma comunque funzionali.

In questa sezione verranno analizzati i principali sistemi di fisica utilizzati dai motori di gioco presi in esame, suddivisi tra soluzioni proprietarie, middleware e sistemi semplificati.

Unreal Engine - Chaos Physics

Unreal Engine ha sempre posto grande enfasi sulla simulazione fisica per garantire esperienze di gioco immersive e realistiche. Fino alla versione 4.x, UE utilizzava NVIDIA PhysX come sistema di fisica predefinito, sfruttandone le capacità per simulare collisioni, dinamiche dei corpi rigidi, soft body e fluidi. PhysX, largamente adottato nell'industria, garantiva un buon equilibrio tra qualità delle simulazioni e prestazioni, con il vantaggio dell'accelerazione hardware su GPU NVIDIA compatibili. Tuttavia, col passare del tempo, Epic Games ha deciso di sviluppare una soluzione più flessibile, profondamente integrata con il proprio ecosistema.

Con l'arrivo di Unreal Engine 5, Epic ha introdotto *Chaos Physics*, un sistema di fisica proprietario progettato per offrire simulazioni avanzate e scalabili, riducendo la dipendenza da middleware esterni come PhysX. Chaos è pensato per integrarsi perfettamente con le altre tecnologie di Unreal, come Nanite e Lumen, mantenendo elevate prestazioni anche in scenari complessi.

Caratteristiche principali di Chaos Physics:

- Simulazione avanzata di corpi rigidi: Chaos gestisce dinamiche complesse dei corpi rigidi con grande precisione. Supporta collisioni dettagliate, proprietà fisiche personalizzabili come attrito e restituzione e consente la simulazione di oggetti con geometrie complesse.

- Distruzione ambientale: Una delle funzionalità distintive di Chaos è la distruzione ambientale realistica. Grazie alla tecnologia *Geometry Collections*, è possibile creare oggetti che si fratturano e si distruggono in tempo reale in risposta a impatti o forze esterne, arricchendo l'interattività del gioco.

- Ragdoll e fisica dei personaggi: Chaos offre un sistema avanzato per la simulazione di ragdoll, garantendo animazioni fluide e realistiche per i personaggi durante cadute, urti o esplosioni. La transizione tra animazioni predefinite e fisica simulata avviene in modo naturale, migliorando l'immersività.

- Supporto per soft body e fluidi: Sebbene ancora in fase di sviluppo, Chaos include funzionalità per la simulazione di corpi deformabili (soft body) e fluidi, consentendo effetti più realistici in ambiti come l'animazione di tessuti o liquidi.

- Prestazioni e scalabilità: Chaos è progettato per scalare efficacemente su diverse piattaforme, dai PC ad alte prestazioni alle console di nuova generazione. Supporta il multi-threading e l'accelerazione GPU, assicurando simulazioni fisiche complesse senza sacrificare il frame rate.

- Integrazione con Niagara: Il sistema di particelle *Niagara* si integra perfettamente con Chaos, consentendo effetti visivi che reagiscono in tempo reale alle simulazioni fisiche, come polvere generata da crolli o particelle che interagiscono con oggetti distrutti.

In conclusione, Chaos Physics amplia notevolmente le possibilità creative per gli sviluppatori, pur mantenendo elevate prestazioni e scalabilità.

Figura 4.7: Distruzione ambientale realistica di Chaos Physics.

CryEngine - CryPhysics

CryPhysics è il sottosistema fisico integrato nel motore CryEngine, progettato per simulare realisticamente il comportamento degli oggetti all'interno di un ambiente virtuale. Questo sistema gestisce una varietà di entità e geometrie fisiche, permettendo interazioni complesse tra gli oggetti e l'ambiente circostante.

Gestione delle Entità Fisiche

Ogni oggetto fisico registrato in CryPhysics possiede una propria entità e geometria, che possono essere collegate ad altri componenti del motore, come il sistema di entità o le particelle del motore 3D. La creazione di entità fisiche avviene su richiesta esterna al motore fisico, e un riferimento all'entità creata viene mantenuto per future interazioni.

Simulazione Multithread

La simulazione fisica in CryPhysics opera su un thread separato, con la possibilità di utilizzare thread aggiuntivi per migliorare le prestazioni. Durante ogni frame, il sistema richiede un aggiornamento della fisica e riceve notifiche al completamento dei calcoli, assicurando una sincronizzazione efficiente tra la simulazione fisica e gli altri sistemi del motore.

Notifiche ed Eventi Fisici

Il sistema fisico invia notifiche tramite eventi specifici, come *EventPhys*, permettendo agli sviluppatori di registrare listener per intercettare eventi quali collisioni, esplosioni o altre interazioni fisiche rilevanti. Questo meccanismo consente una risposta dinamica agli eventi fisici all'interno del gioco, migliorando l'interattività e il realismo dell'esperienza utente.

CryPhysics offre una simulazione fisica avanzata e integrata che arricchisce l'interattività e il realismo dei giochi sviluppati con questo motore.

Godot - Motore fisico integrato

Un altro esempio di come la fisica venga gestita all'interno dei motori di gioco è Godot; essa è progettata per essere altamente flessibile e ben integrata nel motore, mantenendo al contempo una struttura separata che garantisce efficienza e prestazioni costanti.

Il sistema fisico di Godot è un sottosistema a nodi multithread dedicato che si occupa esclusivamente delle simulazioni fisiche, come la gestione delle collisioni, le forze applicate ai corpi e le interazioni tra gli oggetti nel mondo di gioco. Questo sottosistema lavora in modo indipendente rispetto ad altri componenti del motore, come il rendering o l'audio, assicurando che le simulazioni fisiche siano coerenti e stabili, indipendentemente dalle variazioni di frame rate o dalle attività degli altri sistemi.

Sistema a nodi

La fisica in Godot è suddivisa in due sistemi distinti, uno per la fisica 2D e uno per la fisica 3D, entrambi costruiti su principi simili. Il sistema 2D utilizza nodi come *RigidBody2D*, *KinematicBody2D*, *StaticBody2D* e *Area2D* per creare ambienti interattivi e dinamici. Parallelamente, il sistema 3D offre nodi equivalenti (*RigidBody*, *KinematicBody*, *StaticBody*, *Area*) che permettono di gestire simulazioni fisiche nello spazio tridimensionale. Entrambi i sistemi sono altamente modulari e offrono una vasta gamma di funzionalità per gestire movimenti realistici, collisioni accurate e interazioni complesse.

Sistema multithread

Dal punto di vista delle prestazioni, Godot utilizza un sistema di **multithreading opzionale** per le simulazioni fisiche, che di default vengono eseguite su un thread separato rispetto al thread principale, responsabile di rendering e logica di gioco. Questo consente di distribuire il carico di lavoro in modo efficiente, migliorando le prestazioni e riducendo i rallentamenti dovuti a calcoli fisici complessi. Tuttavia, è possibile disattivare il multithreading e far eseguire la fisica sul thread principale per esigenze di debug o sincronizzazione.

La fisica in Godot opera a un tasso fisso di aggiornamento di **60 Hz**, indipendente dal frame rate del gioco, che può variare in base alle risorse disponibili. Godot distingue tra due processi principali:

- *_process(delta)*: chiamato ogni frame renderizzato (Idle Processing), utile per logiche non legate alla fisica, come l'interfaccia utente.
- *_physics_process(delta)*: eseguito a ogni tick fisico (Physics Processing) con un delta costante (circa 0.01666s a 60 Hz). Tutto il codice che interagisce con la fisica, come forze o collisioni, deve essere gestito qui per garantire simulazioni coerenti e indipendenti dal frame rate.

Questa distinzione permette a Godot di mantenere simulazioni fisiche stabili e deterministiche, migliorando l'esperienza di gioco anche in presenza di frame rate variabili.

Il sistema è inoltre altamente personalizzabile, permettendo agli sviluppatori di interagire direttamente con le simulazioni tramite script (GDScript, C#, C++).

Con Godot 4.x, è possibile integrare motori fisici di terze parti tramite GDExtension, ampliando le funzionalità con soluzioni come Jolt Physics o NVIDIA PhysX.

In sintesi, Godot combina prestazioni e flessibilità, rendendolo adatto a progetti 2D e 3D di varia complessità.

PhysX

NVIDIA PhysX è un motore fisico avanzato sviluppato da NVIDIA che fornisce simulazioni fisiche realistiche in tempo reale per videogiochi e applicazioni interattive. Rilasciato nel 2005, PhysX si è distinto per essere uno dei primi middleware a introdurre il supporto per l'accelerazione hardware su GPU. Prima di questa innovazione, tutti i calcoli fisici venivano gestiti esclusivamente dalla CPU, spesso limitando la complessità delle simulazioni per evitare sovraccarichi. Con PhysX, invece, parte dei calcoli fisici più intensivi è stata trasferita alla GPU, sfruttando la sua capacità di eseguire migliaia di operazioni in parallelo. Questo approccio ha permesso di gestire simulazioni complesse — come collisioni tra centinaia di oggetti, fluidi dinamici, fumo realistico, esplosioni e tessuti — senza gravare eccessivamente sulla CPU.

L'architettura di PhysX è progettata per essere scalabile, adattandosi a diverse piattaforme, dai PC high-end alle console di gioco e ai dispositivi mobili. In ambienti con GPU compatibili, il motore sfrutta i CUDA cores per parallelizzare i calcoli fisici, migliorando significativamente le prestazioni e permettendo simulazioni più complesse e dettagliate rispetto a quelle eseguite solo sulla CPU. Tuttavia, PhysX è progettato anche per funzionare in modalità software-only su dispositivi senza GPU NVIDIA, garantendo una buona compatibilità multipiattaforma.

PhysX gestisce diversi tipi di simulazioni, tra cui:

- Corpi rigidi e morbidi (Rigid Bodies e Soft Bodies), per simulare oggetti solidi e deformabili.
- Sistemi particellari, per effetti come fumo, fuoco e pioggia.
- Fluidi dinamici, per liquidi che rispondono realisticamente a forze esterne e gravità.
- Tessuti e abbigliamento, con simulazioni di stoffe che si muovono e si piegano in modo naturale.

L'adozione di PhysX è stata ampia nell'industria dei videogiochi, venendo integrato in motori come **Unity** e versioni precedenti di **Unreal Engine**, o disponibile come modulo integrabile in **O3DE**. Il supporto per l'accelerazione hardware su GPU ha reso PhysX particolarmente apprezzato nei titoli AAA che richiedono simulazioni fisiche avanzate senza compromettere le prestazioni complessive del gioco.

Motori con fisica semplificata

I motori con fisica semplificata sono progettati per offrire funzionalità base di simulazione fisica, concentrandosi su prestazioni leggere e facilità d'uso piuttosto che su realismo avanzato. Questi motori sono spesso impiegati nello sviluppo di giochi 2D o progetti che non richiedono simulazioni fisiche complesse. In questo contesto, la fisica viene trattata in modo essenziale, privilegiando l'efficienza e la semplicità rispetto alla precisione scientifica.

Le caratteristiche principali dei motori con fisica semplificata sono la gestione basilare delle collisioni, simulazioni leggere (gestione della gravità o rimbalzi) e il mantenimento di un basso utilizzo di risorse, risultando ideali per giochi mobile, browser game e prototipi veloci.

Dei nostri motori analizzati, i motori con fisica semplificata sono **GameMaker**, **Phaser**, **Gdevelop**, **Construct3**, **Scratch**, **Cocos2d-x** e **Defold**.

I sistemi di fisica semplice presentano i loro vantaggi e svantaggi.

I vantaggi possono essere riassunti come performance elevate, ideale per giochi che devono funzionare su dispositivi meno potenti o con requisiti di risorse minimi, facilità d'uso - sono infatti spesso accompagnati da interfacce user-friendly e tool di visual scripting che semplificano l'implementazione della fisica - e rapidità di sviluppo.

Per quanto riguarda gli svantaggi troviamo la limitazione nelle simulazioni, meno realismo e supporto 3D limitato, infatti molti motori con fisica semplificata sono progettati esclusivamente per giochi 2D e offrono supporto minimo o nullo per ambienti tridimensionali.

4.2.3 Animazione

Ogni motore di gioco implementa sistemi di animazione con caratteristiche differenti, offrendo strumenti più o meno avanzati per la gestione del movimento. Alcuni motori sono ottimizzati per l'animazione 3D scheletrica con cinematiche avanzate, mentre altri si concentrano sulle animazioni 2D basate su sprite e interpolazione.

Questa sezione analizza e confronta le principali soluzioni di animazione offerte dai motori di gioco, suddividendo gli engine in base al livello di supporto per diverse tecniche di animazione. La categorizzazione aiuterà a evidenziare le differenze principali tra i motori AAA con avanzate capacità di animazione scheletrica e i motori più leggeri focalizzati su sprite animation per il 2D.

I motori di gioco possono essere suddivisi in tre categorie principali in base alle tecniche di animazione supportate e al livello di complessità della loro implementazione.

Motori con supporto avanzato per animazioni 3D

I motori di gioco più avanzati nell'animazione 3D offrono strumenti sofisticati per la gestione di scheletri, blending tra stati, cinematiche inverse (IK) e motion capture. Queste tecnologie sono essenziali per garantire un movimento realistico e naturale nei personaggi e negli oggetti animati. Tra i motori più completi in questo ambito troviamo Unreal Engine, Unity, CryEngine, Open 3D Engine (O3DE) e Stride, ognuno con soluzioni specifiche che rispondono alle esigenze dei progetti AAA e di alta fedeltà visiva.

Unreal Engine è particolarmente rinomato per il suo potente sistema di animazione, che integra strumenti avanzati per la gestione di animazioni complesse e realistiche. Uno degli strumenti principali è il **Control Rig**[1], che consente agli sviluppatori di creare e modificare rig personalizzati direttamente nell'editor. Grazie a questo sistema, è possibile applicare controlli avanzati alle ossa e alle articolazioni dei personaggi, facilitando sia l'animazione procedurale che l'integrazione con il motion capture.

Un altro elemento chiave è lo **State Machine Editor**[2], che consente di gestire in modo visivo e modulare le transizioni tra le diverse animazioni di un personaggio. Questo editor si basa sul concetto di *finite state machines* (FSM), dove ogni stato rappresenta un'animazione specifica, come camminare, correre o saltare. Il **Graph Editor** consente di definire regole precise per le transizioni tra stati, stabilendo quando e come un'animazione

[1] https://dev.epicgames.com/documentation/en-us/unreal-engine/control-rig-in-unreal-engine

[2] https://dev.epicgames.com/documentation/en-us/unreal-engine/state-machines-in-unreal-engine

deve passare a un'altra in base a variabili e condizioni impostate dallo sviluppatore. Questo sistema permette di creare blending fluidi tra pose diverse, migliorando la naturalezza dei movimenti e garantendo una maggiore reattività alle azioni del giocatore.

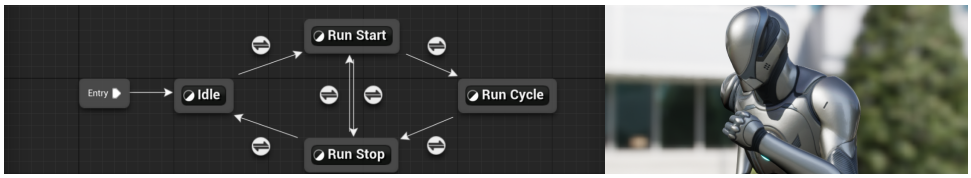

Figura 4.8: Stati e transizioni di una Finite State Machine in Unreal Engine.

Grazie al supporto nativo con **Chaos Physics**, Unreal Engine consente di applicare dinamiche realistiche agli elementi animati, rendendoli reattivi alle forze esterne e migliorando il realismo delle interazioni, e si afferma come uno dei motori più avanzati per l'animazione nei videogiochi.

Unity è dotato di un sistema di animazione flessibile e potente, noto come **Mecanim**[3], che permette di gestire animazioni scheletriche avanzate, blending tra stati e cinematiche inverse (IK). Grazie a Mecanim, gli sviluppatori possono definire sistemi di animazione complessi utilizzando *state machines* per organizzare e controllare le transizioni tra le diverse pose di un personaggio in modo fluido e reattivo.

Un'importante estensione del sistema di animazione è **Animation Rigging**[4], sistema che permette di modificare e influenzare le animazioni dei personaggi in tempo reale, direttamente dentro Unity, senza dover rifare le animazioni da zero in un altro software (come Blender o Maya). Ad esempio, se un personaggio possiede un'animazione di camminata e deve tenere un'arma in mano, anziché generare manualmente animazioni specifiche per ogni posizione dell'arma, è possibile applicare un vincolo alla mano affinché segua automaticamente la posizione dell'oggetto. Questo approccio garantisce maggiore flessibilità e adattabilità, migliorando l'interazione tra le animazioni e l'ambiente di gioco.

Unity offre anche strumenti specifici per la gestione delle scene animate. **Timeline**[5] è un editor visivo che permette di orchestrare sequenze animate in modo intuitivo, risultando particolarmente utile per la creazione di cutscene e sequenze scriptate.

CryEngine implementa un sistema avanzato per la gestione delle animazioni noto come **Mannequin**[6], progettato per offrire un controllo dettagliato sulle sequenze di animazione e sulle transizioni tra stati. Questo strumento consente di combinare animazioni predefinite con un sofisticato sistema di blending, garantendo transizioni fluide tra pose diverse.

Un elemento distintivo di CryEngine è l'integrazione con il motore fisico **CryPhysics**, che permette la simulazione avanzata di *ragdoll* e animazioni basate su eventi fisici. Grazie a questa tecnologia, i personaggi possono reagire in modo realistico agli impatti e alle interazioni ambientali, migliorando l'immersività del gameplay.

Un altro punto di forza di CryEngine è la sua ottimizzazione per ambienti open-world. Il sistema di animazione è progettato per gestire un elevato numero di entità animate con-

[3] https://docs.unity3d.com/Manual/MecanimAnimationSystem.html
[4] https://docs.unity3d.com/Packages/com.unity.animation.rigging@latest
[5] https://docs.unity3d.com/Manual/TimelineSection.html
[6] https://docs.cryengine.com/display/CEPROG/Mannequin+Animation+System

temporaneamente, senza compromettere le prestazioni. Questo lo rende particolarmente adatto per giochi con grandi mappe e NPC dinamici, dove è fondamentale garantire movimenti realistici su vasta scala.

Open 3D Engine (O3DE) si sta affermando come una soluzione altamente modulare per la gestione delle animazioni 3D. Il suo **Animation Editor**[7] offre strumenti avanzati per la creazione e la modifica delle animazioni, supportando sia le tradizionali *state machines* che tecniche più moderne come il **motion matching**. Quest'ultima è una metodologia innovativa basata sull'intelligenza artificiale, che permette di selezionare dinamicamente l'animazione più appropriata in base alla postura e al movimento del personaggio, migliorando la fluidità delle transizioni e il realismo complessivo.

Il motore utilizza **EMotion FX**[8], un sistema di animazione avanzato inizialmente sviluppato per Amazon Lumberyard e successivamente integrato in O3DE. EMotion FX offre un robusto supporto per la gestione di animazioni scheletriche, blending avanzato, inverse kinematics (IK) e transizioni fluide tra stati animati. Questo sistema permette agli sviluppatori di controllare in modo preciso il comportamento dei personaggi e di creare animazioni dinamiche che si adattano all'ambiente di gioco in tempo reale.

Stride offre un sistema di animazione scheletrica e cinematiche con un'interfaccia intuitiva basata su **timeline**[9]. Sebbene meno avanzato rispetto ad altri motori con sistemi potenti di animazione, Stride si distingue per la sua **architettura modulare**, che consente agli sviluppatori di personalizzare le animazioni in base alle esigenze del progetto. Il motore supporta skeletal animation, blending tra pose e keyframe animation, garantendo una gestione fluida delle sequenze animate. Grazie alla sua leggerezza e flessibilità, Stride rappresenta una soluzione adatta a chi necessita di un motore meno complesso ma comunque capace di gestire animazioni articolate.

Questi motori rappresentano le soluzioni più avanzate per l'animazione nei videogiochi, offrendo una combinazione di motion capture, blending, cinematiche inverse e simulazioni fisiche, che rendono possibile la creazione di personaggi animati in modo realistico e coinvolgente.

Motori con supporto misto per animazione 2D e 3D

Alcuni motori di gioco bilanciano il supporto tra **animazione 2D e 3D**, offrendo strumenti versatili per entrambi gli approcci. Sebbene non raggiungano la complessità dei motori AAA per il 3D, garantiscono una gestione efficace delle animazioni scheletriche e delle animazioni basate su sprite.

Godot[10] eccelle nell'animazione 2D grazie al suo **AnimationPlayer**, un sistema potente per interpolazioni e transizioni. Supporta anche animazioni 3D tramite il **AnimationTree**, che permette di creare grafi di stati con blending avanzato e inverse kinematics (IK), sebbene con funzionalità più limitate rispetto a Unreal o Unity.

Cocos2d-x[11] è focalizzato sulle animazioni 2D ma include il supporto per scheletri 3D e keyframe animation. Dispone di un **Skeletal Animation System** che permette

[7] https://docs.o3de.org/docs/user-guide/interactivity/animation/animation-editor/
[8] https://aws.amazon.com/blogs/gametech/1-11/
[9] https://doc.stride3d.net/latest/en/manual/animation/index.html
[10] https://docs.godotengine.org/en/stable/tutorials/animation/index.html
[11] https://docs.cocos.com/creator/3.4/manual/en/animation/

la gestione delle animazioni di personaggi articolati, pur con strumenti meno avanzati rispetto ai motori completamente 3D.

GameMaker utilizza un approccio ibrido, con una gestione avanzata degli sprite e la possibilità di integrare animazioni scheletriche per modelli 2D. Il supporto per il 3D è limitato e generalmente non ottimizzato per giochi complessi.

Defold[12] offre un sistema basato su sprite con interpolazione automatica e keyframe, supportando anche animazioni scheletriche in 2D. Pur avendo strumenti 3D, il focus del motore rimane sulle animazioni bidimensionali.

Questi motori rappresentano una soluzione ideale per chi desidera lavorare sia in 2D che in 3D senza la complessità dei motori di fascia alta, risultando particolarmente adatti per giochi indie e produzioni leggere.

Motori specializzati nell'animazione 2D basata su sprite

Alcuni motori di gioco sono progettati specificamente per lo sviluppo di giochi **2D**, offrendo strumenti ottimizzati per la gestione delle animazioni basate su **sprite sheets** e interpolazione tra frame. Questi engine privilegiano la semplicità e l'efficienza, risultando ideali per produzioni indie e giochi con risorse limitate.

GDevelop[13] e **Construct 3** offrono un sistema di animazione intuitivo basato su **frame-by-frame** e **tweening**, che consente di creare movimenti fluidi senza la necessità di scripting. Entrambi includono strumenti per la gestione delle transizioni tra animazioni e supportano effetti visivi dinamici.

Phaser utilizza un sistema di animazione basato su sprite, con supporto per interpolazione e gestione avanzata dei frame.

Scratch adotta un approccio didattico, semplificando il processo di animazione tramite un'interfaccia visuale a blocchi. Sebbene le sue capacità siano limitate rispetto agli altri motori, è un'ottima scelta per apprendere i concetti fondamentali dell'animazione nei videogiochi.

Questi motori rappresentano la soluzione ideale per chi desidera sviluppare giochi 2D con un workflow rapido e accessibile, senza la complessità aggiuntiva delle animazioni scheletriche o 3D.

4.2.4 Audio

L'implementazione dell'audio varia notevolmente tra i diversi motori di gioco, con alcuni che offrono sistemi audio avanzati e altri che richiedono l'integrazione di middleware dedicati.

Funzionalità avanzate nei motori principali

Unity e Unreal Engine sono i motori più completi in termini di gestione audio. Entrambi supportano l'audio spaziale, l'audio dinamico e l'integrazione con middleware professionali come **FMOD** e **Wwise**, che consentono una gestione più avanzata di effetti sonori, riverberi e mix dinamici.

- **Unity** utilizza il sistema *Audio Mixer*, che permette di creare catene di effetti e mixare l'audio in tempo reale.

[12] https://defold.com/manuals/animation/
[13] https://wiki.gdevelop.io/gdevelop5/objects/sprite/

Motore	Motore Audio Integrato	Supporto Audio 3D	Middleware Esterni
Unity	Audio Mixer	Sì	FMOD, Wwise
Unreal Engine	Unreal Audio Engine	Sì	FMOD, Wwise
Godot	AudioServer	Sì	SoLoud
CryEngine	CryAudio	Sì	FMOD, Wwise
O3DE	Audio System Component	Sì	Wwise
GameMaker	Audio API nativa	No	-
GDevelop	Web Audio API	No	-
Construct 3	Web Audio API	No	-
Cocos2d-x	SimpleAudioEngine	Sì	SoLoud
Defold	OpenAL	No	-
Stride	AudioEngine	Sì	FMOD
Phaser	Web Audio API	No	-

Figura 4.9: Tabella riassuntiva del supporto audio dei vari motori.

- **Unreal Engine** integra un potente motore audio che supporta sintesi audio procedurale, DSP avanzato e spazializzazione realistica.
- **Godot** offre un sistema audio integrato (*AudioServer*) con supporto per effetti DSP e bus audio multipli.
- **CryEngine** utilizza *CryAudio*, un motore audio ottimizzato per ambienti open-world con propagazione del suono realistica.
- **O3DE** supporta nativamente *Wwise*, consentendo un'implementazione flessibile dell'audio nei giochi AAA.

Motori più semplici come **GameMaker**, **GDevelop** e **Construct 3** offrono funzionalità basilari per la riproduzione audio, basandosi su API native o Web Audio API.

Audio spaziale e tecnologie immersive

L'audio spaziale, o **3D audio**, è fondamentale nei giochi in prima persona, negli open-world e nelle esperienze di realtà virtuale (VR). Tecnologie come **HRTF (Head-Related Transfer Function)** e **Dolby Atmos** vengono implementate in motori avanzati per migliorare l'immersività.

- **Unreal Engine** integra il supporto per *HRTF* e piattaforme VR come *Oculus Spatializer*.
- **Unity** supporta il motore *Steam Audio* e il sistema di audio spaziale di Microsoft.
- **CryEngine** utilizza algoritmi avanzati per la propagazione del suono negli ambienti.

Queste tecnologie permettono di riprodurre l'audio in modo realistico, simulando effetti come il rimbalzo del suono sulle superfici e la percezione della profondità sonora.

L'audio nei motori di gioco varia in base alla complessità del motore e al target del gioco. Mentre motori come **Unreal Engine, Unity e CryEngine** offrono strumenti

avanzati per la gestione dell'audio, altri come **GameMaker, Phaser e Construct 3** forniscono soluzioni più semplici, adatte a giochi 2D o browser game.

L'integrazione con middleware professionali come **FMOD e Wwise** rappresenta la scelta ideale per chi desidera un controllo approfondito sul comparto audio. Inoltre, l'evoluzione dell'audio spaziale sta rendendo sempre più immersivi i videogiochi, portando a una maggiore attenzione verso il suono tridimensionale nei titoli moderni.

4.2.5 Networking

Il supporto al networking è un aspetto cruciale per i giochi multiplayer, determinando la capacità del motore di gestire la sincronizzazione tra giocatori, la scalabilità e la facilità di implementazione. In questa sezione analizziamo il networking nei principali motori di gioco rispondendo a quattro domande fondamentali:

- **Quali motori hanno networking integrato?** Alcuni motori offrono un'infrastruttura nativa per il multiplayer, mentre altri necessitano di plugin o servizi esterni.
- **Quanto è scalabile la soluzione?** Valutiamo la capacità del motore di gestire sessioni con molti giocatori e supporto per server dedicati o cloud gaming.
- **Quanto è complesso da implementare?** Analizziamo la difficoltà nell'uso del networking integrato e la necessità di configurazioni avanzate.
- **Esistono plugin o pacchetti esterni che migliorano il networking?** Esaminiamo eventuali strumenti aggiuntivi che possono facilitare o potenziare le funzionalità di rete.

Per semplificare il confronto, i motori sono suddivisi in tre categorie:

- **Soluzioni avanzate**: motori con un networking potente e integrato, adatto a giochi su larga scala.
- **Soluzioni intermedie**: motori con un networking base, che può essere migliorato con plugin esterni.
- **Soluzioni limitate**: motori senza supporto nativo al networking, che richiedono servizi di terze parti.

Soluzioni avanzate

Unreal Engine Unreal Engine offre un supporto multiplayer nativo tra i più avanzati nel settore. Il sistema di replicazione client-server integrato consente di sincronizzare automaticamente gli oggetti di gioco tra i giocatori. Supporta **server dedicati**, garantendo alte prestazioni anche in giochi con molti utenti.

In termini di **scalabilità**, Unreal è progettato per gestire giochi di grandi dimensioni come *Fortnite* e *PUBG*. La sua architettura permette di distribuire il carico tra più server e ridurre il traffico di rete tramite tecniche come il *network relevancy*.

Dal punto di vista della **facilità di implementazione**, Unreal include strumenti per la gestione della rete direttamente nell'editor. Tuttavia, la curva di apprendimento può essere ripida per chi non ha esperienza con il networking.

Sono disponibili anche **plugin** avanzati, tra cui integrazioni con servizi cloud come AWS GameLift e Microsoft Azure, che migliorano ulteriormente la scalabilità del multiplayer[14].

Open 3D Engine (O3DE) O3DE, essendo derivato da Amazon Lumberyard sviluppato da Amazon, offre un'infrastruttura di networking nativa basata su **Amazon Web Services (AWS) GameLift**, che consente la gestione dinamica di server dedicati nel cloud. Il supporto per *client-server* e *peer-to-peer* lo rende una soluzione adatta a giochi multiplayer complessi.

Dal punto di vista della **scalabilità**, O3DE è altamente ottimizzato per il cloud gaming, consentendo di scalare automaticamente i server in base al numero di giocatori.

L'implementazione del networking richiede configurazioni avanzate, soprattutto per la gestione dei servizi cloud, ma la documentazione fornisce supporto dettagliato per l'integrazione.

Essendo open-source, O3DE permette la personalizzazione completa delle sue funzionalità di rete, con possibilità di estenderle tramite moduli personalizzati[15].

Soluzioni intermedie

Unity Unity offre il pacchetto **Netcode for GameObjects** per la gestione del networking, ma non ha un sistema di replicazione avanzato come Unreal. Per migliorare le funzionalità multiplayer, si utilizzano plugin di terze parti come *Mirror* e *Fish-Net*, che offrono una gestione più efficiente della rete.

A livello di **scalabilità**, Unity non è ottimizzato per giochi con migliaia di giocatori simultanei, ma con l'utilizzo dei plugin può essere utilizzato per giochi multiplayer di medio-grandi dimensioni.

L'implementazione del networking è relativamente semplice, soprattutto con l'uso di Mirror, che automatizza molte operazioni. Tuttavia, configurare server dedicati richiede competenze avanzate.

Tra i **plugin** consigliati troviamo *Photon Fusion* per il networking basato su cloud e *PlayFab* per la gestione del matchmaking[16].

Godot Godot include un'API di networking con supporto per TCP, UDP e WebSockets, ma non ha un sistema di replicazione automatica. Gli sviluppatori devono implementare manualmente la sincronizzazione dei dati tra client e server.

In termini di **scalabilità**, Godot è adatto a giochi con pochi giocatori per sessione. Per supportare server dedicati o giochi di grandi dimensioni, si consiglia l'uso di *ENet* o *Nakama*.

Dal punto di vista della **complessità**, il networking in Godot richiede più lavoro rispetto a Unity o Unreal, poiché molte funzioni devono essere gestite manualmente.

Per migliorare le capacità multiplayer, è possibile integrare **plugin** come Nakama, che fornisce matchmaking e gestione utenti[17].

[14] https://dev.epicgames.com/documentation/en-us/unreal-engine/networking-and-multiplayer

[15] https://www.docs.o3de.org/docs/user-guide/networking/

[16] https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@latest

[17] https://docs.godotengine.org/en/stable/tutorials/networking/index.html

Soluzioni limitate

CryEngine CryEngine non ha un'infrastruttura di networking avanzata. Per implementare il multiplayer, gli sviluppatori devono utilizzare librerie esterne come *Photon* o *PlayFab*.

A livello di **scalabilità**, CryEngine non è ottimizzato per MMO o giochi su larga scala senza integrazioni esterne.

L'implementazione del networking è complessa, richiedendo configurazioni personalizzate.

Non esistono molte soluzioni ufficiali per espandere il networking in CryEngine, quindi la scelta dipende dalle librerie di terze parti[18].

GameMaker, GDevelop, Construct 3, Phaser, Scratch Questi motori non hanno alcun supporto nativo per il networking. Il multiplayer deve essere implementato utilizzando API esterne come Firebase o WebSockets.

Non sono scalabili per giochi con molti giocatori e la gestione della rete è interamente manuale.

A livello di **complessità**, il multiplayer deve essere costruito da zero, rendendo difficile l'implementazione per sviluppatori senza esperienza di rete[19].

4.3 Facilità d'uso

Un aspetto cruciale nella scelta di un motore di gioco è la sua facilità d'uso, che influisce direttamente sulla rapidità con cui uno sviluppatore può iniziare a lavorare su un progetto. Un'interfaccia intuitiva, strumenti accessibili, una documentazione ben strutturata e la presenza di un'ampia comunità di supporto possono ridurre significativamente il tempo necessario per familiarizzare con l'ambiente di sviluppo.

Nome Engine	Facilità d'uso	Learning Curve	Community
Scratch	Molto alta	Molto bassa	Molto ampia
Construct3	Alta	Bassa	Ampia
GDevelop	Alta	Bassa	Media
GameMaker	Alta	Bassa	Ampia
Godot	Alta	Bassa	Ampia
Phaser	Media	Media	Piccola
Cocos2d	Media	Media	Media
Defold	Media	Media	Medio-piccola
Stride	Media	Elevata	Molto piccola
Unity	Media	Media	Molto ampia
Open 3D Engine	Media	Elevata	Molto piccola
Unreal Engine	Bassa	Molto elevata	Molto ampia
CryEngine	Bassa	Elevata	Media

Figura 4.10: Tabella riassuntiva Sezione 4.3.

[18] https://docs.cryengine.com/display/CEPROG/Networking
[19] https://help.gdevelop.io/game-logic/networking/

Per analizzare questo aspetto nei principali game engine, ci baseremo su:

- La presenza di strumenti di sviluppo visuali, come sistemi di visual scripting, che permettono di creare logiche di gioco senza necessariamente scrivere codice.
- La presenza e la qualità della documentazione, che influisce sulla velocità con cui si possono risolvere problemi e trovare risorse utili.
- La presenza di una comunità di supporto attiva e/o forum ed altri centri di supporto.

L'analisi si baserà su una combinazione di fonti: recensioni ed esperienze degli utenti, documentazione ufficiale e una valutazione diretta dell'esperienza d'uso dei motori. Verrà mostrata prima una tabella riassuntiva (vedi 4.10) degli engine ordinati per difficoltà d'uso crescente, dopodiché si analizzeranno gli engine uno alla volta.

Scratch

Interfaccia e strumenti: Scratch è un ambiente di programmazione visuale progettato per introdurre i principi della programmazione in modo semplice e intuitivo. L'interfaccia utente è suddivisa in tre aree principali:

Area dello stage: situata in alto a destra, dove gli sprite (personaggi o oggetti) interagiscono e mostrano le loro azioni.

Area dei blocchi: posizionata a sinistra, contiene blocchi di codice suddivisi per categorie (movimento, aspetto, suono, controllo, sensori, operatori, variabili e altro). Gli utenti possono trascinare questi blocchi nell'area di scripting per costruire le logiche del programma.

Area di scripting: situata al centro, è lo spazio in cui si assemblano i blocchi per creare script che definiscono il comportamento degli sprite.

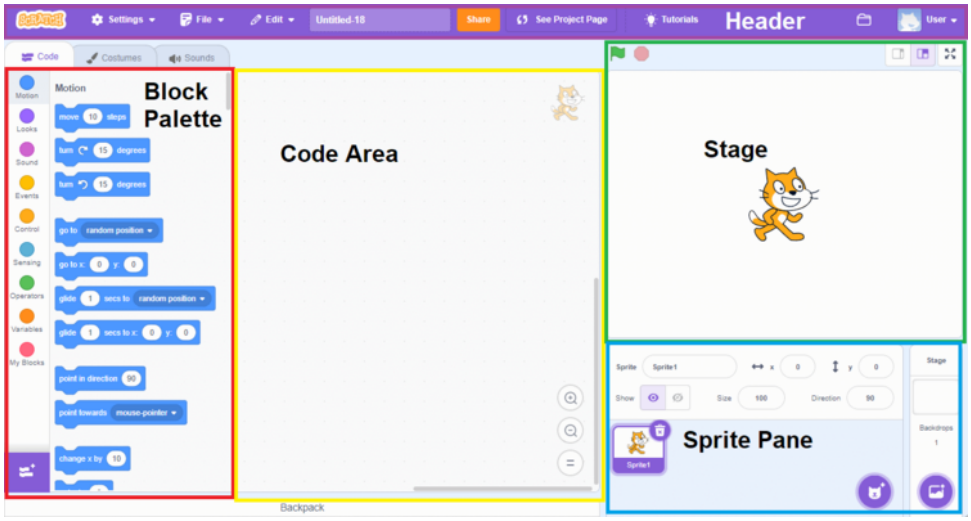

Figura 4.11: Semplice interfaccia Scratch.

L'approccio basato su blocchi elimina la necessità di digitare codice, riducendo gli errori sintattici e rendendo la programmazione accessibile anche ai più giovani. Inoltre, Scratch offre un editor di costumi integrato per disegnare o importare immagini e un editor di suoni per registrare o modificare effetti audio.

Documentazione e comunità: La documentazione ufficiale di Scratch è accessibile dal sito ufficiale Scratch. La comunità online di Scratch è estremamente ampia e attiva, con più di 1,4 milioni di utenti attivi (più di 130 milioni registrati).

Per coloro che desiderano funzionalità aggiuntive, esiste un'estensione del browser chiamata "Scratch Addons" che combina nuove caratteristiche e temi per il sito e l'editor di Scratch in un'unica estensione facile da abilitare. Maggiori informazioni sono disponibili su Scratch Addons.

Facilità d'uso: Uno dei principali punti di forza di Scratch è la sua facilità d'uso **molto alta**. L'interfaccia drag-and-drop e l'assenza di una sintassi complessa permettono anche ai principianti di iniziare a creare progetti interattivi in breve tempo. Questo rende Scratch particolarmente adatto per l'educazione nelle scuole primarie e secondarie, ma anche per chiunque voglia avvicinarsi al mondo della programmazione in modo coinvolgente.

Construct3

Interfaccia e strumenti: Construct 3 è un motore di gioco 2D basato su browser, progettato per offrire un'esperienza di sviluppo intuitiva e accessibile anche a chi non ha conoscenze di programmazione. L'editor è completamente web-based, eliminando la necessità di installazioni locali.

L'interfaccia utente è suddivisa in diverse sezioni principali:

- Layout Editor: permette di costruire visivamente i livelli di gioco, posizionando sprite e oggetti direttamente nella scena.
- Event Sheet Editor: utilizza un sistema di programmazione basato su eventi e condizioni, che consente di definire logiche di gioco senza scrivere codice.
- Properties Panel: mostra le proprietà dell'oggetto selezionato, offrendo opzioni di personalizzazione senza la necessità di script.
- Objects Panel: permette di gestire gli asset del gioco, tra cui sprite, suoni, effetti e oggetti interattivi.

Uno dei punti di forza di Construct 3 è il sistema di eventi che semplifica la creazione di giochi. Tuttavia, per gli sviluppatori più avanzati, è possibile integrare codice JavaScript per funzioni più complesse. Un'altra caratteristica utile è il real-time preview, che consente di testare le modifiche senza dover esportare il gioco.

Documentazione e comunità: Construct dispone di una documentazione ufficiale accessibile dal sito ufficiale Construct 3 Manual.

La community è in crescita, con forum con quasi 800.000 post, un server Discord ed un Asset Store.

Uno degli aspetti più apprezzati dagli utenti è il supporto diretto offerto dagli sviluppatori di Construct 3, che rispondono frequentemente alle richieste nei forum ufficiali. Tuttavia, rispetto a engine più popolati, il numero di tutorial esterni e corsi online è più limitato.

Facilità d'uso: Construct 3 è considerato uno dei motori più accessibili per lo sviluppo di giochi 2D, grazie al suo sistema basato su eventi. Per chi desidera maggiore controllo, la possibilità di integrare JavaScript offre una maggiore flessibilità, ma richiede competenze di programmazione.

Nel complesso, Construct 3 ha una facilità d'uso **molto alta**, ideale per giochi 2D, prototipazione rapida e progetti educativi.

GDevelop

Interfaccia e strumenti: GDevelop offre un'interfaccia utente intuitiva e accessibile, progettata per facilitare lo sviluppo di giochi 2D anche a chi non possiede competenze di programmazione. Il motore utilizza un sistema di sviluppo basato su eventi, l'editor è leggero e ben organizzato, con strumenti integrati per la gestione di sprite, scene e fisica. È disponibile sia come applicazione desktop che come versione web, permettendo lo sviluppo direttamente dal browser. Inoltre, esiste una versione mobile con funzionalità limitate per dispositivi Android e iOS.

Documentazione e comunità: GDevelop dispone di documentazione ufficiale accessibile dal sito GDevelop Wiki. La comunità è attiva e in crescita, con la presenza di un forum e un server Discord. Essendo un progetto open-source, GDevelop conta circa un centinaio di contributor attivi. Tuttavia, rispetto a motori più affermati ed utilizzati, la quantità di risorse e tutorial disponibili è inferiore.

Facilità d'uso: Uno dei principali vantaggi di GDevelop è la sua facilità d'uso **alta**, rendendolo ideale per principianti e per chi desidera prototipare rapidamente idee di gioco. Il sistema basato su eventi elimina la necessità di conoscere linguaggi di programmazione, permettendo a chiunque di creare giochi funzionali. Per gli utenti più avanzati, GDevelop offre la possibilità di integrare codice JavaScript.

GameMaker

Interfaccia e strumenti: GameMaker è noto per la sua interfaccia intuitiva e orientata a facilitare lo sviluppo di giochi 2D. Il motore offre un sistema di sviluppo semplice basato su **drag-and-drop**. Per gli utenti più esperti, è disponibile il **GameMaker Language (GML)**, linguaggio di scripting proprietario che consente di personalizzare meglio il comportamento del gioco.

L'editor di GameMaker è semplice e ben organizzato, con strumenti integrati per la gestione di sprite, livelli, animazioni e logica di gioco.

Documentazione e comunità: GameMaker dispone di un GameMaker Manual ufficiale chiaro e ben organizzato, con all'interno una guida più veloce per cominciare a conoscere l'engine da zero, ma rispetto a motori più diffusi la quantità di risorse disponibili online è comunque inferiore.

Il forum ufficiale di GameMaker conta oltre 100.000 discussioni, con più di 660.000 post, indicando una community coinvolta, ma viene spesso segnalato come l'utilizzo della funzionalità di drag-and-drop per sviluppare renda più difficile mostrare il proprio lavoro e richiedere aiuto.

GameMaker offre anche un marketplace con asset e strumenti creati dalla community.

Facilità d'uso: Uno dei punti di forza di GameMaker è la sua **alta** facilità d'uso, rendendolo uno dei motori più accessibili per chi vuole iniziare a sviluppare videogiochi.

Le limitazioni nel supporto al 3D lo rendono meno adatto a sviluppatori esperti che vogliono lavorare su progetti complessi.

Godot

Interfaccia e strumenti: L'interfaccia di Godot è pulita, modulare e altamente personalizzabile, progettata per essere leggera e reattiva[14]. Il layout è diviso in tre ambienti di lavoro principali: 2D, 3D e script, che permettono agli sviluppatori di passare rapidamente da un tipo di sviluppo all'altro.

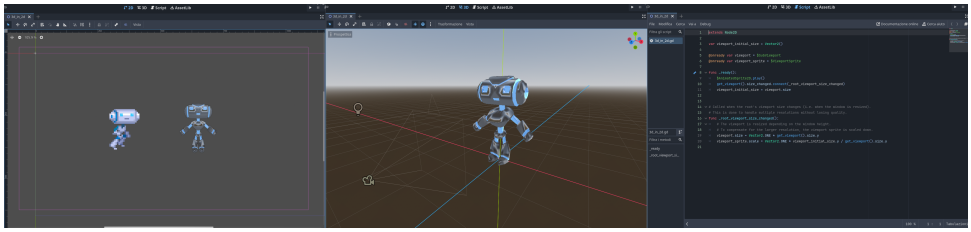

Figura 4.12: Stessa scena di Godot rappresentata nei tre layout di lavoro diversi: 2D, 3D e script.

Uno dei suoi punti di forza è l'organizzazione a scene e nodi: invece di una struttura a oggetti, in Godot ogni elemento del gioco è un nodo, e le scene sono insiemi di nodi gerarchici. Questo approccio rende lo sviluppo molto modulare, ma può risultare inizialmente poco intuitivo per chi proviene da altri motori.

L'editor è generalmente fluido e veloce, senza i lunghi tempi di caricamento di motori più pesanti come Unreal.

È possibile sviluppare tramite GDScript, linguaggio di programmazione simile a Python, C#, sebbene l'integrazione non sia ancora completa come in Unity, e C++, che può essere utilizzato per estendere il motore tramite moduli personalizzati. Per chi preferisce un approccio senza codice, Godot offre un sistema di visual scripting, sebbene sia meno potente e meno utilizzato rispetto a quelli di altri motori. L'editor di Godot include strumenti integrati per debugging, profiling e gestione delle versioni, eliminando la necessità di installare plugin esterni per queste funzionalità. Inoltre, supporta il live reloading, permettendo agli sviluppatori di modificare il codice e vedere i cambiamenti in tempo reale senza dover riavviare il gioco.

Documentazione e comunità: La documentazione ufficiale è disponibile al sito Godot Docs. La community, pur essendo più piccola rispetto a Unity o Unreal, è in rapida crescita con oltre 150.000 utenti sul server Discord ufficiale e un'attiva presenza su Reddit e GitHub. Godot beneficia inoltre del supporto della comunità open-source, con oltre 600 contributori attivi.

Facilità d'uso: Godot è considerato uno dei motori più accessibili per chi inizia a sviluppare giochi, grazie alla sua interfaccia intuitiva e a GDScript. Molti utenti lo scelgono proprio per la sua **medio-alta** facilità d'uso e l'approccio logico alla gestione delle scene e degli oggetti. Tuttavia, chi proviene da Unity o Unreal potrebbe trovare

alcune differenze concettuali inizialmente spiazzanti. Inoltre, sebbene sia ottimo per il 2D, il supporto per il 3D è ancora acerbo, e chi lavora su progetti più complessi potrebbe trovarsi limitato.

Phaser

Interfaccia e strumenti: A differenza di altri motori di gioco, Phaser non fornisce un'interfaccia utente grafica integrata; lo sviluppo avviene principalmente attraverso codice scritto in un editor di testo a scelta dello sviluppatore. Tuttavia, esistono strumenti complementari che facilitano il processo di sviluppo:

- Phaser Editor: un ambiente di sviluppo integrato che offre strumenti visuali per la creazione di scene, gestione delle risorse e altro, semplificando il workflow per gli sviluppatori.

- Tiled Map Editor: un editor esterno per la creazione di tilemaps, facilmente integrabile con Phaser per la progettazione di livelli complessi.

La flessibilità di Phaser consente l'integrazione con vari framework front-end come React, Vue e Svelte, permettendo agli sviluppatori di scegliere l'ecosistema più adatto alle proprie esigenze.

Documentazione e comunità: Phaser dispone di una documentazione ufficiale completa, accessibile dal sito ufficiale Phaser Documentation. La comunità di Phaser è piccola, con meno di 4000 membri sul Forum ufficiale e 10.000 sul server Discord.

Facilità d'uso: Phaser è apprezzato per la sua curva di apprendimento relativamente dolce, soprattutto per gli sviluppatori con una conoscenza di base di JavaScript. Nel complesso, Phaser si prospetta come un motore dalla facilità d'uso **medio-alta**.

Cocos2d-x

Interfaccia e strumenti: Lo sviluppo su Cocos avviene principalmente attraverso codice, utilizzando ambienti di sviluppo come Visual Studio o Xcode. Per facilitare la progettazione, esistono strumenti come Cocos Creator, un IDE integrato che fornisce un'interfaccia visuale per la creazione di scene e l'animazione, semplificando il workflow degli sviluppatori.

Documentazione e comunità: La documentazione ufficiale di Cocos2d-x è disponibile sul sito Cocos2d-x Documentation. La community è abbastanza attiva, con un forum con oltre 230.000 post. Sono disponibili numerosi tutorial e risorse online create dalla community.

Facilità d'uso: Cocos2d-x richiede una buona conoscenza dei linguaggi di programmazione supportati, come C++ o JavaScript, il che potrebbe rappresentare una sfida per i principianti. Tuttavia, l'utilizzo di Cocos Creator e la presenza di una community attiva possono ridurre la complessità iniziale. La facilità d'uso è **media**.

Defold

Interfaccia e strumenti: L'editor di Defold include un editor di scena visiva, un debugger, gestione degli asset e un editor di tilemap, fornendo un ambiente completo per lo sviluppo del gioco. Il motore utilizza Lua per lo scripting, ma consente anche l'uso di estensioni native scritte in C o C++ per funzionalità avanzate. Inoltre, Defold supporta l'integrazione con sistemi di controllo versione come Git direttamente nell'IDE.

Documentazione e comunità: La documentazione ufficiale di Defold è disponibile sul sito Defold Documentation. La comunità di Defold è medio-piccola con circa 30.000 utenti nel forum.

Facilità d'uso: Defold è apprezzato per la sua efficienza e le dimensioni ridotte delle build, rendendolo ideale per giochi web e mobile ad alte prestazioni. L'uso di Lua come linguaggio di scripting facilita l'apprendimento per i nuovi sviluppatori, mentre le estensioni native offrono potenza aggiuntiva per gli utenti esperti. Data l'assenza di un'interfaccia visuale completa, nel complesso la facilità d'uso è **media**.

Stride

Interfaccia e strumenti: Stride utilizza C# per la programmazione e offre un ambiente di sviluppo integrato chiamato Game Studio, che consente di importare asset, creare scene e gestire entità attraverso un sistema di componenti. Stride supporta il Physically Based Rendering e include strumenti come un editor di materiali, un editor di particelle e un sistema di scripting integrato. Inoltre, offre un sistema di prefabbricati e archetipi per facilitare la riutilizzabilità degli oggetti di gioco.

Documentazione e comunità: La documentazione ufficiale di Stride è disponibile sul sito Stride Documentation. La comunità è molto piccola ma attiva su GitHub. Essendo un progetto open-source, Stride beneficia di contributi continui da parte della comunità.

Facilità d'uso: Stride è apprezzato per la sua integrazione con Visual Studio e per l'uso di C#, rendendolo familiare agli sviluppatori con esperienza in questo linguaggio. L'interfaccia del Game Studio è progettata per essere intuitiva, ma la complessità del motore potrebbe rappresentare una sfida per i principianti assoluti con una facilità d'uso complessiva **media**.

Unity

Interfaccia e strumenti: L'opinione della community è che Unity offra un'interfaccia intuitiva e flessibile, adatta sia ai principianti che agli sviluppatori esperti. L'editor è ben organizzato, con strumenti visuali per la gestione delle scene, degli asset e delle animazioni. Il motore supporta la programmazione in C#, ma include anche Bolt, un sistema di visual scripting che permette di creare logiche di gioco senza codice.

Documentazione e comunità: La documentazione di Unity (Unity Documentation è molto ampia e ben strutturata, con guide dettagliate e tutorial ufficiali. La sua community è tra le più grandi nel settore dei game engine, con oltre 1,5 milioni di utenti registrati

solo sul forum ufficiale e un ampio numero di server Discord, subreddit e gruppi Facebook dedicati. Il Learning Hub di Unity offre corsi interattivi, mentre l'Asset Store fornisce risorse aggiuntive, come plugin e strumenti di terze parti, che possono semplificare il lavoro degli sviluppatori.

Facilità d'uso: Unity è un motore dalla facilità d'uso **media**, grazie alla sua interfaccia user-friendly e alla disponibilità di strumenti visuali. I principianti possono iniziare rapidamente grazie alla documentazione ben fatta e ai numerosi tutorial. Tuttavia, la curva di apprendimento diventa più impegnativa data la versatilità dell'engine e dunque la quantità di features disponibili. Nonostante ciò, rispetto ad altri motori, Unity mantiene un buon equilibrio tra accessibilità e potenza, risultando adatto a una vasta gamma di sviluppatori.

O3DE

Interfaccia e strumenti: O3DE offre un'interfaccia modulare e altamente personalizzabile, progettata per fornire un ambiente di sviluppo scalabile. L'editor è più complesso rispetto a Unity o Unreal, ma permette agli sviluppatori di abilitare solo i moduli necessari, ottimizzando le risorse di sistema. L'integrazione con Amazon Web Services (AWS) consente di implementare facilmente funzionalità multiplayer e servizi cloud. Tuttavia, l'interfaccia risulta meno intuitiva rispetto ad altri motori più affermati, richiedendo tempo per essere padroneggiata.

Documentazione e comunità: La documentazione di O3DE, disponibile sul sito ufficiale O3DE Learn e GitHub, è scarsa e a tratti mancante. Essendo un motore relativamente nuovo, la community è molto piccola, con un server Discord di meno di 4000 membri. Questo può rendere più difficile trovare supporto rapido su forum o risorse esterne.

Facilità d'uso: O3DE ha una curva di apprendimento ripida, specialmente per chi proviene da motori più user-friendly e una facilità d'uso **medio-bassa**. La sua architettura modulare offre grande flessibilità, ma richiede agli sviluppatori di configurare manualmente molti aspetti del motore. Questo lo rende più adatto a team con esperienza piuttosto che a sviluppatori alle prime armi.

Unreal Engine

Interfaccia e strumenti: Unreal Engine offre un'interfaccia complessa ma ben strutturata, con strumenti avanzati per il rendering, la fisica e l'animazione. Il sistema di visual scripting Blueprint è uno dei suoi punti di forza, permettendo di creare logiche di gioco senza dover scrivere codice. Tuttavia, il motore richiede hardware potente per funzionare in modo ottimale.

Documentazione e comunità: Unreal offre una delle documentazioni più complete Unreal Engine Docs, arricchita da tutorial video e corsi disponibili su piattaforme come Unreal Academy. La community conta oltre 7.5 milioni di sviluppatori registrati, con forum ufficiali, Discord e una vasta libreria di contenuti su YouTube. Il marketplace di Unreal è una fonte essenziale di asset, spesso rilasciati gratuitamente da Epic Games.

Facilità d'uso: Nel complesso, la facilità d'uso di Unreal Engine è **medio/bassa**: Blueprint, la presenza di corsi e ampia documentazione ed altri strumenti intuitivi lo rendono accessibile ai principianti, ma la curva di apprendimento è molto ripida per chi vuole sfruttare appieno tutte le sue funzionalità. Il passaggio a C++ e all'ottimizzazione avanzata richiede competenze elevate e tempo per padroneggiare il motore.

CryENGINE

Interfaccia e strumenti: CryEngine presenta un'interfaccia complessa, pensata per sviluppatori con esperienza e non molto intuitivo per i principianti. L'editor CryEngine Sandbox offre strumenti avanzati per il rendering fotorealistico, la gestione delle fisiche e l'intelligenza artificiale.

Il motore supporta C++ e C#. Anche se è dotato di un visual scripting system (Schematyc), questo non è così immediato come Blueprint di Unreal o Bolt di Unity. L'editor permette di modificare l'ambiente in tempo reale, ma ha un'interfaccia più tecnica che può allontanare chi è abituato a workflow più accessibili.

Documentazione e comunità: CryEngine dispone di una documentazione ufficiale accessibile dal sito ufficiale CRYENGINE V Manual e dalla sua wiki, tuttavia, gli utenti riportano che la qualità della documentazione sia bassa. Il numero di tutorial e corsi disponibili è inferiore rispetto ad altri engine.

La community è più piccola rispetto a quella degli altri motori, il che può rendere più difficile trovare supporto su forum o gruppi di discussione[15]. Esistono però il CryEngine Discord Server e il forum ufficiale offrono uno spazio per chiedere aiuto[16].

Facilità d'uso: CryEngine ha una curva di apprendimento decisamente ripida, e una facilità d'uso **molto bassa**. Sebbene il motore offra funzionalità avanzate per grafica e fisica, richiede una buona conoscenza di programmazione per sfruttarlo appieno.

In generale, CryEngine è molto potente ma meno accessibile, adatto a chi vuole lavorare su progetti AAA con alta fedeltà grafica. Tuttavia, per gli sviluppatori indipendenti o alle prime armi, è un engine sicuramente difficile da padroneggiare.

4.4 Portabilità

La scelta di un motore di gioco non dipende solo dalle sue funzionalità tecniche o dalla facilità d'uso, ma anche dalla sua capacità di supportare diverse piattaforme di destinazione. La portabilità è un fattore cruciale per gli sviluppatori che desiderano raggiungere un pubblico più ampio, distribuendo i propri giochi su dispositivi mobili, PC, console, web e piattaforme di realtà aumentata.

In questa sezione, analizzeremo i principali motori di gioco confrontando la loro compatibilità con le diverse piattaforme e il loro grado di apertura (open source o proprietario).

La tabella seguente presenta una panoramica delle capacità di porting dei motori di gioco più diffusi, evidenziando quali piattaforme sono supportate nativamente, se il motore è open source e il livello di compatibilità con le tecnologie di Extended Reality. Successivamente verranno descritti in dettaglio i singoli motori, analizzandone i punti di forza e le limitazioni in termini di portabilità.

Motore	Desktop	Mobile	Console	Web	XR	Open Source
Unity	✔	✔	✔	✔	✔	✘
Unreal Engine	✔	✔	✔	✘	✔	✘
Godot	✔	✔	✘	✔	✔	✔
CryEngine	✔	✘	✔	✘	✔	✘
O3DE	✔	✔	✘	✘	✘	✔
GameMaker	✔	✔	✔	✔	✘	✘
Gdevelop	✔	✔	✘	✔	✘	✔
Construct3	✔	✔	✔	✔	✘	✘
Scratch	✘	✘	✘	✔	✘	✔
Cocos2d-x	✔	✔	✘	✔	✘	✔
Defold	✔	✔	✔	✔	✘	✔
Stride	✔	✔	✘	✘	✔	✔
Phaser	✔	✔	✘	✔	✘	✔

Figura 4.13: Tabella riassuntiva del supporto dei vari engine alle piattaforme.

Unity Unity è uno dei motori di gioco più versatili in termini di **portabilità**, supportando una vasta gamma di piattaforme. È compatibile con **desktop** (Windows, macOS, Linux), **dispositivi mobili** (iOS, Android), **console** (PlayStation, Xbox, Nintendo Switch) e il **web** (WebGL). Inoltre, offre supporto avanzato per tecnologie di **Extended Reality (XR)**, permettendo lo sviluppo di esperienze in **realtà virtuale e aumentata** su dispositivi come **Oculus, HoloLens e altri visori VR/AR**[20].

Unreal Engine Unreal Engine è uno dei motori di gioco più avanzati e utilizzati nel settore, particolarmente apprezzato per le sue capacità grafiche e il supporto per progetti **AAA**. Offre un'ampia portabilità, supportando **desktop** (Windows, macOS, Linux), **console** (PlayStation 4, PlayStation 5, Xbox One, Xbox Series X/S, Nintendo Switch), **dispositivi mobili** (iOS, Android) e **web** (WebGL). In ambito **Extended Reality (XR)**, Unreal Engine è compatibile con tecnologie di **realtà virtuale e aumentata**, supportando dispositivi come **Meta Quest, HoloLens, HTC Vive e PlayStation VR**[21].

Godot Godot è un motore di gioco **open source** e multipiattaforma, particolarmente apprezzato per il suo focus su accessibilità e sviluppo indipendente. Supporta la portabilità su **desktop** (Windows, macOS, Linux, BSD), **dispositivi mobili** (Android, iOS), e **web** (HTML5, WebAssembly). In ambito **Extended Reality (XR)**, Godot è compatibile con dispositivi di **realtà virtuale e aumentata** come HTC Vive, Valve Index, Oculus Rift, Oculus Go, Oculus Quest e tutti i visori Windows Mixed Reality[22].

Sebbene Godot non offra supporto ufficiale per le console a causa delle restrizioni delle licenze proprietarie, è possibile distribuire giochi su queste piattaforme tramite soluzioni

[20] https://support.unity.com/hc/en-us/articles/206336795-What-platforms-are-supported-by-Unity
[21] https://www.unrealengine.com/en-US/faq
[22] https://docs.godotengine.org/en/stable/about/faq.html#which-platforms-are-supported-by-godot

di terze parti. Godot è rilasciato sotto licenza **MIT**, il che lo rende completamente **open source**, consentendo agli sviluppatori di modificare e distribuire liberamente il codice sorgente senza restrizioni commerciali.

CryEngine CryEngine supporta nativamente piattaforme **desktop** come Windows e Linux, oltre a console di ultima generazione come **PlayStation 4, PlayStation 5, Xbox One e Xbox Series X/S**. Il supporto per **dispositivi mobili** è attualmente in sviluppo, mentre per il **web** non è disponibile. In ambito **XR**, CryEngine è compatibile con dispositivi di realtà virtuale come **Oculus Rift, PlayStation VR e HTC Vive**, offrendo strumenti avanzati per la creazione di esperienze immersive[23].

Open 3D Engine (O3DE) O3DE supporta **desktop** (Windows, Linux, macOS in sviluppo), mentre il supporto per **dispositivi mobili** (Android, iOS) e **console** è limitato e richiede implementazioni personalizzate. Non offre supporto diretto per il **web**. Per quanto riguarda la **Extended Reality (XR)**, O3DE fornisce compatibilità con OpenXR per lo sviluppo di applicazioni in realtà virtuale e aumentata[24].

Essendo **completamente open source** sotto licenza Apache 2.0, O3DE consente agli sviluppatori di modificarne il codice e personalizzare il motore senza restrizioni commerciali.

GameMaker GameMaker è un motore di gioco noto per la sua facilità d'uso e l'orientamento allo sviluppo **2D**. Supporta la portabilità su **desktop** (Windows, macOS, Ubuntu), **dispositivi mobili** (Android, iOS), **console** (PlayStation 4, PlayStation 5, Xbox One, Xbox Series X/S, Nintendo Switch) e **web** (HTML5) [25].

GDevelop GDevelop supporta la portabilità su **desktop** (Windows, macOS, Linux), **dispositivi mobili** (Android, iOS) e **web** (HTML5). Sebbene il supporto diretto per **console** non sia disponibile, è possibile esportare i giochi in formati compatibili con soluzioni di terze parti.

Essendo rilasciato sotto licenza **MIT**, GDevelop è completamente **open source**[26].

Construct 3 Construct 3 supporta la portabilità su **desktop** (Windows, macOS, Linux via applicazioni basate su browser), **dispositivi mobili** (Android, iOS) e **web** (HTML5). Il supporto per **console** è disponibile tramite esportazione su piattaforme compatibili con soluzioni di terze parti[27].

Scratch Scratch è accessibile principalmente tramite browser web su piattaforme **desktop** (Windows, macOS, Linux) e **dispositivi mobili** (tablet iOS e Android). Esiste anche un'applicazione desktop offline per Windows e macOS; tuttavia, il supporto per Linux è limitato e attualmente non ufficialmente disponibile. Scratch non offre supporto diretto per lo sviluppo su **console** o per applicazioni **web** al di fuori della propria piattaforma.

[23] https://www.cryengine.com/support/view/general#platform-support
[24] https://www.docs.o3de.org/docs/user-guide/platforms/
[25] https://gamemaker.io/en/blog/export-with-gamemaker
[26] https://wiki.gdevelop.io/gdevelop5/publishing/#publish-on-stores-and-other-platforms
[27] https://en.wikipedia.org/wiki/Construct_(game_engine)#Supported_platforms

Rilasciato sotto licenza **GPL-2.0-or-later**, Scratch consente agli utenti di visualizzare e modificare il codice sorgente[28].

Cocos2d-x Cocos2d-x supporta la portabilità su **dispositivi mobili** (iOS 8.0 e versioni successive, Android 3.0 e versioni successive), **desktop** (Windows 7, Windows 8.1, Windows 10, macOS 10.9 e versioni successive) e **web** (HTML5, compatibile con browser come Chrome, Safari e Internet Explorer 9 e versioni successive). Il supporto per **console** non è fornito direttamente, ma può essere implementato con soluzioni di terze parti.

Cocos2d-x è rilasciato sotto licenza **MIT**, rendendolo completamente **open source**[29].

Defold Defold supporta la creazione di giochi per **desktop** (Windows, macOS, Linux), **dispositivi mobili** (iOS, Android), **web** (HTML5) e **console** (Nintendo Switch, PlayStation 4, PlayStation 5). L'accesso alle versioni compatibili con le console richiede l'approvazione come sviluppatore da parte dei rispettivi produttori. Il motore è distribuito con una licenza derivata dall'Apache 2.0, che ne permette l'uso gratuito [30].

Stride Stride supporta la portabilità su **desktop** (Windows 7, 8, 10; Linux), **dispositivi mobili** (Android 2.3 e versioni successive; iOS 8.0 e versioni successive) e **piattaforme universali Windows (UWP)**. Il supporto per **console** non è fornito direttamente e potrebbe richiedere implementazioni personalizzate.

Distribuito sotto licenza **MIT**, Stride è completamente open-source [31].

Phaser I giochi sviluppati con Phaser possono essere eseguiti su **desktop** e **dispositivi mobili** direttamente tramite browser. Inoltre, grazie a strumenti di terze parti, è possibile compilare i giochi in applicazioni native per **iOS**, **Android** e **Steam**, ampliando le piattaforme di distribuzione[32].

È rilasciato sotto licenza **MIT**, è dunque open-source[33].

4.5 Linguaggi di programmazione

La scelta del linguaggio di programmazione è un elemento fondamentale nello sviluppo di un videogioco. Ogni motore di gioco supporta uno o più linguaggi, che possono variare far variare fattori come prestazioni, facilità d'uso e flessibilità. Alcuni motori si basano su linguaggi compilati, che offrono maggiore efficienza e ottimizzazione (ad esempio C++ e C#), mentre altri utilizzano linguaggi interpretati come JavaScript o Lua, che semplificano il processo di sviluppo e testing.

Un ulteriore elemento da considerare è la presenza di strumenti di visual scripting, che permettono di creare logiche di gioco senza scrivere codice tradizionale. Questo approccio è utile per designer e sviluppatori meno esperti, facilitando la prototipazione e lo sviluppo rapido.

[28] https://en.wikipedia.org/wiki/Scratch_(programming_language)
[29] https://www.cocos.com/en/cocos2d-x
[30] https://defold.com/faq/faq/
[31] https://doc.stride3d.net/latest/en/manual/platforms/index.html
[32] https://phaser.io/
[33] https://github.com/photonstorm/phaser

Nella tabella 4.14 vengono confrontati i principali motori di gioco, evidenziando i linguaggi supportati, la loro tipologia (compilato, interpretato o misto) e la disponibilità di strumenti di scripting visuale.

Motore	Linguaggi supportati	Tipologia	Visual Scripting	Note
Unity	C#	Compilato	Bolt (Unity Visual Scripting)	C# principale, ampio supporto
Unreal Engine	C++, Blueprint	Compilato + Visual Scripting	Blueprint	C++ per performance, Blueprint per accessibilità
Godot	GDScript, C#, C++	Interpretato + Compilato	Visual Scripting non più supportato in Godot 4	GDScript simile a Python, più accessibile
CryEngine	C++, Lua	Compilato + Scripting	Schematyc	C++ per sviluppo avanzato, Lua per scripting
O3DE	C++, Lua, Python	Compilato + Scripting	No	Modulare, usa C++ come base
GameMaker	GML	Interpretato	Drag and Drop	GML è un linguaggio ibrido tra scripting e programmazione
GDevelop	JavaScript	Interpretato	Event System	Event System senza necessità di codice
Construct 3	JavaScript	Interpretato	Event System	No-code con possibilità di JS avanzato
Scratch	Scratch Blocks	Visual Scripting	Sì	Totalmente basato su programmazione visuale
Cocos2d-x	C++, Lua, JavaScript	Compilato + Scripting	No	C++ per performance, Lua principale per scripting, supporto JS limitato
Defold	Lua, C++	Scripting + Compilato	No	Lua per scripting, C++ per estensioni native
Stride	C#	Compilato	No	-
Phaser	JavaScript	Interpretato	No	-

Figura 4.14: Tabella riassuntiva dei linguaggi di programmazione usati dai motori.

Dopo aver osservato la tabella, è possibile trarre alcune considerazioni generali sui linguaggi di programmazione utilizzati nei vari motori di gioco:

- **Motori con linguaggi compilati (C++, C#)**: Offrono prestazioni elevate, ma richiedono una conoscenza più avanzata della programmazione. Esempi di questi motori sono **Unreal Engine, Unity** e **CryEngine**.
- **Motori con linguaggi interpretati (Lua, JavaScript, GDScript)**: Sono più accessibili e facili da usare, ma meno ottimizzati per giochi ad alte prestazioni. Alcuni esempi sono **Godot, Defold** e **Phaser**.
- **Motori con supporto a entrambi**: Alcuni motori permettono di scegliere tra un linguaggio compilato per alte performance (C++, C#) o uno interpretato per maggiore velocità di sviluppo (GDScript, Lua). **Godot** e **CryEngine** sono esempi di questa flessibilità.
- **Motori con Visual Scripting**: Strumenti come **Blueprint (Unreal), Bolt (Unity) ed Event System (GDevelop, Construct 3)** consentono di creare la logica di gioco senza scrivere codice, facilitando lo sviluppo anche a chi non ha conoscenze di programmazione.

Conclusioni

Ogni motore di gioco offre un ecosistema di sviluppo con vantaggi e svantaggi legati al linguaggio di programmazione supportato. I linguaggi compilati garantiscono velocità ed efficienza, mentre quelli interpretati rendono più semplice la sperimentazione e la prototipazione. Inoltre, il Visual Scripting gioca un ruolo chiave nell'accessibilità, riducendo la barriera d'ingresso per chi vuole avvicinarsi allo sviluppo di videogiochi senza conoscenze avanzate di programmazione.

4.6 Compatibilità tra motori e generi di gioco

La scelta di un motore di gioco non dipende solo dalle sue caratteristiche tecniche o dalla facilità d'uso, ma anche dal tipo di videogioco che si intende sviluppare. Ogni motore ha punti di forza che lo rendono più adatto a determinati generi: alcuni eccellono nella gestione di ambienti open-world realistici, mentre altri sono ottimizzati per giochi 2D, strategici o multiplayer.

In questa sezione verranno analizzati i principali generi di videogiochi e i motori che meglio si prestano al loro sviluppo. L'obiettivo è fornire una panoramica chiara su quali motori siano più indicati per ciascun tipo di esperienza, evidenziando i loro vantaggi e le eventuali limitazioni.

4.6.1 Action & Shooter (FPS/TPS, Fighting)

I giochi d'azione e sparatutto, sia in prima (FPS, First-Person Shooter) che in terza persona (TPS, Third-Person Shooter), richiedono un motore che garantisca fluidità nel gameplay, una gestione avanzata della fisica e, nel caso del multiplayer, un'infrastruttura di rete efficiente.

First-Person Shooter (FPS) e Third-Person Shooter (TPS) I motori più adatti agli FPS e TPS devono supportare ambienti dettagliati, una gestione avanzata delle animazioni e un sistema di rete efficiente. **Unreal Engine** è la scelta principale per i giochi sparatutto ad alto budget, grazie alla sua capacità di gestire illuminazione avanzata, fisica realistica e un solido sistema di replicazione multiplayer. Alcuni titoli realizzati con Unreal Engine includono *Call of Duty: Warzone* e *Valorant*.

Anche **CryEngine** è una valida alternativa, particolarmente adatto per giochi con un'elevata fedeltà grafica e una fisica avanzata, come dimostrato dalla serie *Crysis*.

Unity, invece, è spesso scelto per giochi FPS/TPS indie o mobile, grazie alla sua flessibilità e leggerezza rispetto a Unreal e CryEngine. Tuttavia, per supportare il multiplayer avanzato, Unity necessita di plugin aggiuntivi.

Fighting Games e Brawler I giochi di combattimento richiedono un sistema di animazioni fluido e preciso, oltre a una gestione efficiente delle hitbox e delle collisioni. **Unreal Engine** è una delle scelte principali per picchiaduro di fascia alta, con titoli come *Tekken 7* e *Mortal Kombat 11*. Anche **Unity** è ampiamente utilizzato per giochi di combattimento indie e mobile, come *Brawlhalla*.

Per giochi di combattimento in 2D, **Godot** offre un motore leggero e altamente personalizzabile, ideale per titoli indie con una forte componente artistica.

4.6.2 Platformer & Metroidvania

I platformer sono giochi basati su meccaniche di salto e movimento preciso, mentre i Metroidvania combinano elementi platform con esplorazione e progressione non lineare.

Platformer 2D e 3D Per lo sviluppo di platformer in 2D, motori come **Godot** e **GameMaker** sono tra le scelte più popolari, grazie alla loro gestione ottimizzata degli sprite e alla facilità di utilizzo. Titoli come *Undertale* sono stati sviluppati con GameMaker, mentre Godot è spesso utilizzato per progetti indie con uno stile artistico peculiare.

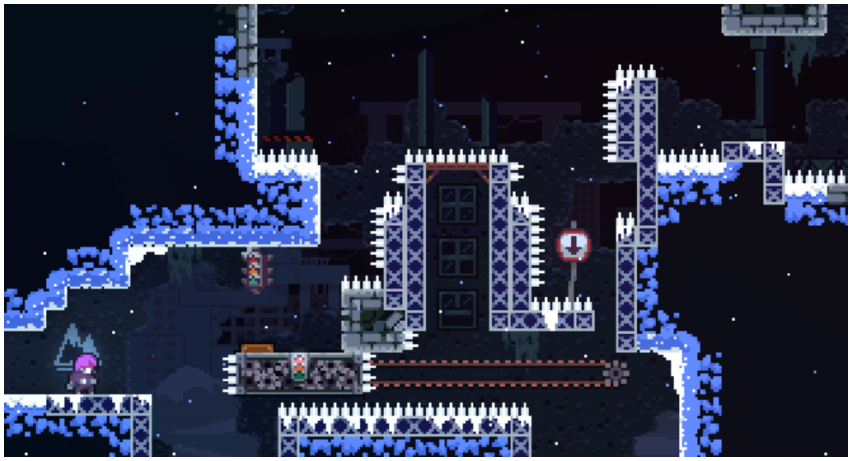

Figura 4.15: Celeste, semplice gioco platform 2D realizzato con GameMaker.

Per platformer in 3D, **Unreal Engine** è una scelta comune per titoli con ambienti dettagliati e meccaniche avanzate.

Metroidvania I giochi Metroidvania, un sottogenere dei platform con elementi di azione e avventura, combinano esplorazione non lineare, backtracking e progressione basata sull'acquisizione di nuove abilità per sbloccare aree precedentemente inaccessibili, un esempio è *Hollow Knight*. Questi titoli richiedono un motore che supporti un'ampia mappa interconnessa e un sistema di gestione delle risorse efficiente. **Unity** è il motore più usato per questo genere, grazie alla possibilità di gestire il world streaming e sistemi di mappa interconnessi. Anche **Godot** è una buona scelta per Metroidvania 2D grazie alla sua leggerezza e facilità di scripting.

4.6.3 Racing & Sports

I giochi di corse e sportivi possono variare da esperienze altamente realistiche a versioni più arcade e accessibili. La scelta del motore dipende dal livello di simulazione richiesto, dalla qualità della fisica e dalla necessità di gestione avanzata delle animazioni e delle interazioni.

Racing Games I giochi di corse si dividono principalmente in due categorie: *simulativi* e *arcade*. Per i titoli simulativi come *Gran Turismo 7*, che richiedono una fisica realistica e un motore grafico avanzato, **Unreal Engine** e **CryEngine** rappresentano le migliori opzioni. Il loro supporto per illuminazione dinamica e dettagli ad alta fedeltà visiva li rende ideali per titoli automobilistici basati su circuiti reali. I giochi di corse arcade

come *Mario Kart 8*, invece, privilegiano la fluidità e la reattività del gameplay. In questo caso, **Unity** è spesso utilizzato grazie alla sua ottimizzazione per il mobile e il supporto multipiattaforma, risultando una scelta ideale per titoli di kart racing o esperienze meno realistiche.

Sports Games Nei giochi sportivi, la gestione dell'animazione e della fisica gioca un ruolo cruciale, così come il supporto per modalità multiplayer. **Unreal Engine** viene comunemente utilizzato nei titoli AAA per la sua capacità di rappresentare modelli dettagliati e simulazioni realistiche - un esempio è *NBA 2K23*, mentre **Unity** è spesso impiegato nei giochi sportivi meno esigenti dal punto di vista grafico o destinati al mercato mobile.

4.6.4 Strategy & Simulation

I giochi strategici e di simulazione si distinguono per la necessità di gestire sistemi complessi, che includono intelligenza artificiale, simulazione di grandi quantità di unità o ambienti dinamici.

Strategy Games I giochi strategici, sia *a turni* che *in tempo reale* (RTS), richiedono motori in grado di gestire molte unità su schermo senza sacrificare le prestazioni. **Unity** è tra le scelte più popolari grazie alla sua capacità di gestire pathfinding avanzato e sistemi di AI per le unità, con celebre esempio *Civilization VI*. Anche **Godot** può essere un'opzione valida per giochi di strategia meno complessi, grazie alla sua efficienza nel rendering 2D e alla sua semplicità di scripting.

Simulation Games I giochi di simulazione comprendono un'ampia gamma di esperienze, dai *city builder* ai *simulatori di vita*. Questi titoli necessitano di motori con supporto per ambienti interattivi, gestione delle risorse e interfacce complesse. **Unity** è ampiamente utilizzato per simulatori gestionali, mentre **O3DE**, grazie alla sua architettura modulare, è particolarmente adatto a simulazioni che necessitano di un'alta scalabilità.

4.6.5 RPG & Open-World Adventures

I giochi di ruolo (*RPG, Role-Playing Games*) e le avventure open-world sono caratterizzati da mondi estesi, interazioni complesse e sistemi di progressione del personaggio. La scelta del motore dipende dalla complessità dell'ambiente di gioco e dalla necessità di rendering avanzato.

Action RPG e Open-World Gli *action RPG* e i titoli open-world richiedono motori capaci di gestire vasti ambienti e un numero elevato di asset senza comprometterne le prestazioni. **Unreal Engine** è ampiamente utilizzato nei giochi di ruolo di fascia alta, grazie al suo supporto per mondi dettagliati e illuminazione realistica. Titoli con grande libertà di esplorazione e interazioni avanzate beneficiano di strumenti dedicati come il *world partitioning* e il rendering ottimizzato.

Figura 4.16: 3D open-world con grafica alta.

Turn-Based RPG e Isometric RPG Per RPG a turni e giochi di ruolo isometrici, la gestione dell'intelligenza artificiale e dell'interfaccia utente è fondamentale. **Unity** è una scelta comune per RPG tattici e giochi con visuale isometrica, grazie alla sua flessibilità e al supporto per asset 2D e 3D. Anche **Godot** si presta bene a RPG meno complessi, in particolare quelli con uno stile grafico più minimalista.

4.6.6 Survival & Sandbox

I giochi di sopravvivenza e sandbox condividono caratteristiche comuni, come la generazione procedurale del mondo e la forte interazione con l'ambiente. Questi titoli necessitano di motori in grado di supportare sistemi dinamici e simulazioni avanzate.

Survival Games I giochi di sopravvivenza, incentrati sulla raccolta di risorse e sulla gestione dell'ambiente, richiedono motori con un buon supporto per la fisica e per il world streaming. **Unreal Engine** è spesso utilizzato nei titoli con ambientazioni dettagliate e una forte componente narrativa, mentre **Unity** è scelto per giochi survival più leggeri o con una grafica stilizzata.

Sandbox Games I giochi sandbox offrono libertà creativa ai giocatori, consentendo loro di modificare il mondo di gioco o di costruire strutture. **Unity** è una scelta popolare per giochi sandbox con un gameplay strutturato, mentre **Godot** è adatto a esperienze sandbox 2D con una gestione personalizzabile della fisica. Per titoli con mondi generativi più complessi, vengono spesso utilizzati motori proprietari, sviluppati su misura per le esigenze specifiche del gioco.

4.6.7 Puzzle & Casual Games

I giochi puzzle e casual sono caratterizzati da meccaniche di gioco semplici e accessibili, spesso con un'interfaccia minimalista e un gameplay immediato. Questi titoli devono essere ottimizzati per il mobile e per il web, richiedendo motori leggeri e flessibili.

Puzzle Games I giochi puzzle possono spaziare da esperienze basate sulla logica a titoli con una forte componente narrativa. I motori più adatti a questo genere sono **Unity** e **Godot**, grazie alla loro capacità di gestire interfacce interattive e animazioni fluide. Per puzzle game 2D con una struttura più semplice, **GameMaker** e **Construct 3** sono alternative leggere e facili da usare.

Casual & Hyper-Casual Games I giochi casual, spesso progettati per dispositivi mobili e con sessioni di gioco brevi, necessitano di motori altamente ottimizzati. **Unity** è il motore più diffuso per il mobile, grazie al supporto per iOS e Android e alla possibilità di integrare rapidamente sistemi di monetizzazione. Per giochi più semplici e basati su interazioni immediate, **GDevelop** e **Phaser** offrono strumenti di sviluppo rapidi e compatibilità con il web.

4.6.8 Horror

I giochi horror pongono l'accento sull'atmosfera e sulla narrazione interattiva. La gestione dell'illuminazione, del suono e delle animazioni è cruciale per immergere il giocatore nell'esperienza.

Horror Games I giochi horror spesso fanno ampio uso di illuminazione dinamica, ombre avanzate e ambienti dettagliati per creare tensione. **Unreal Engine** è una delle scelte più comuni per questo genere, grazie ai suoi strumenti avanzati per il rendering e la gestione della luce, come dimostrano molti titoli di successo. **Unity** è spesso utilizzato per giochi horror indie, grazie alla sua versatilità e alla vasta disponibilità di asset. Per progetti più sperimentali o in 2D, **Godot** offre una buona alternativa con un'architettura leggera e altamente personalizzabile.

4.6.9 Multiplayer & Online Experiences

I giochi multiplayer e online variano da esperienze cooperative a titoli competitivi con una grande infrastruttura di rete. Il motore scelto deve essere in grado di gestire la sincronizzazione tra giocatori, la latenza di rete e, in alcuni casi, il supporto per server dedicati.

Competitive Multiplayer I giochi multiplayer competitivi, come *sparatutto online* o *battle royale*, richiedono un motore con un'ottima gestione della rete e della latenza. **Unreal Engine** è spesso preferito per titoli di fascia alta grazie al suo sistema di replicazione avanzato, che facilita la sincronizzazione tra client e server, un famoso esempio è *Fortnite*. Anche **O3DE**, con il suo supporto per AWS, è un'opzione interessante per giochi con un'infrastruttura cloud scalabile.

Cooperative & MMO I giochi multiplayer cooperativi e gli *MMORPG* necessitano di strumenti che gestiscano sia la rete che il caricamento dinamico di grandi mondi. **Unity** è spesso scelto per esperienze multiplayer più leggere e accessibili, mentre **Unreal Engine** viene impiegato nei MMORPG più complessi. **O3DE**, grazie alla sua modularità e all'integrazione con servizi cloud, rappresenta una soluzione solida per giochi con un elevato numero di giocatori simultanei.

4.6.10 VR, AR & Location-Based Games

I giochi in realtà virtuale (*VR, Virtual Reality*) e realtà aumentata (*AR, Augmented Reality*) sfruttano tecnologie specifiche per creare esperienze immersive. Il motore scelto deve supportare dispositivi VR/AR e garantire prestazioni elevate con una gestione ottimizzata delle risorse.

VR Games Per la realtà virtuale, il rendering deve essere altamente ottimizzato per garantire fluidità e ridurre la latenza, essenziale per un'esperienza immersiva. **Unreal Engine** è uno dei motori più utilizzati in ambito VR, grazie al supporto nativo per dispositivi come Oculus Rift, HTC Vive e PlayStation VR. **Unity** è un'alternativa molto diffusa per esperienze VR meno esigenti in termini di grafica, soprattutto nel mercato indie e mobile.

AR & Location-Based Games I giochi basati sulla realtà aumentata e sulla geolocalizzazione, come quelli mobile che utilizzano ARKit e ARCore, richiedono un motore con un buon supporto per le API di mapping e tracking. **Unity** è la scelta principale per questo genere, essendo ampiamente utilizzato nello sviluppo di giochi AR e location-based come il celebre *Pokémon Go*, grazie alla sua compatibilità con le principali piattaforme mobile. Per applicazioni sperimentali o meno complesse, anche **Godot** sta iniziando a offrire strumenti per AR, sebbene ancora limitati rispetto ai concorrenti.

4.7 Flessibilità e modularità

In questa sezione verranno analizzati i motori che si distinguono per il loro approccio modulare e flessibile. Si partirà da soluzioni altamente modulari come O3DE, Stride e Unity, per poi esaminare motori che offrono un solido supporto per plugin e moduli estendibili, e infine si esploreranno quelli con un approccio più leggero alla modularità, adatti a progetti meno complessi.

4.7.1 O3DE

Open 3D Engine è progettato con un'architettura estremamente modulare, che consente agli sviluppatori di personalizzare e estendere le funzionalità del motore in base alle esigenze specifiche dei loro progetti. Questa modularità è principalmente realizzata attraverso il sistema delle *Gems*.

Le Gemme in O3DE sono pacchetti che contengono codice e/o asset progettati per aggiungere funzionalità o contenuti ai progetti, un esempio è il sistema PhysX di NVidia, che può essere incluso separatamente nel motore sotto forma di Gemma.

Questo sistema permette agli sviluppatori di selezionare solo le componenti necessarie, evitando l'inclusione di elementi superflui e migliorando le performance del motore. Inoltre, gli sviluppatori possono creare Gemme personalizzate per includere collezioni di asset, estendere l'editor o sviluppare logiche di gioco specifiche.

Esistono due tipi principali di Gemme in O3DE:

1. Gemme di **funzionalità**: Aggiungono nuove features o estendono quelle esistenti nel motore.

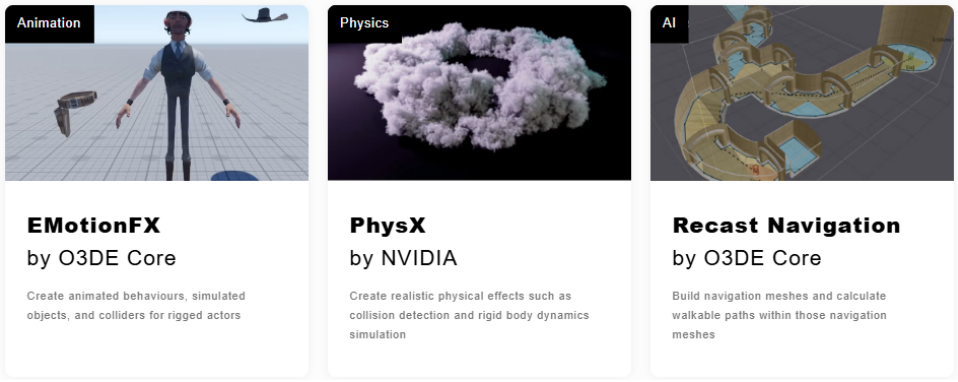

Figura 4.17: Alcune Gemme includibili in O3DE.

2. Gemme di **asset**: Forniscono risorse come modelli 3D, texture o suoni da utilizzare nei progetti.

L'approccio modulare di O3DE offre diversi benefici:

- **Personalizzazione**: Gli sviluppatori possono configurare il motore per soddisfare requisiti specifici, aggiungendo o rimuovendo funzionalità secondo necessità.
- **Espandibilità**: La creazione e l'integrazione di Gemme personalizzate permettono di estendere le capacità del motore senza modificare il codice sorgente principale.
- **Gestione delle dipendenze**: Le Gemme possono dichiarare dipendenze da altre Gemme, facilitando la gestione e l'organizzazione delle componenti del progetto.
- **Miglioramento delle performance:** Poiché O3DE compila solo le Gemme nel progetto, i tempi di build si riducono, migliorando l'efficienza durante lo sviluppo. Anche l'output finale risulta più snello, con eseguibili meno pesanti e più performanti.

Una Gemma tipica include codice sorgente, asset, file di manifest - utile a descrivere la Gemma e le sue dipendenze - e il file di build. Per visualizzarle si può visitare l'Archivio delle Gemme.

Le Gemme possono essere registrate e gestite attraverso l'O3DE Project Manager o tramite interfaccia a riga di comando, offrendo flessibilità nell'organizzazione e nella configurazione dei progetti.

Tuttavia, questo approccio modulare presenta anche alcune note negative. La gestione delle dipendenze tra diverse Gemme può diventare complessa, specialmente nei progetti di grandi dimensioni, e un numero elevato di Gemme attive può introdurre overhead se non gestito correttamente. Inoltre, la qualità delle Gemme sviluppate dalla community può variare, richiedendo attenzione nella selezione di quelle da integrare.

La modularità di O3DE così realizzata fornisce agli sviluppatori un ambiente flessibile e scalabile per creare applicazioni 3D complesse, adattandosi facilmente alle diverse esigenze progettuali.

4.7.2 Stride

Una delle caratteristiche distintive di Stride è il suo sistema di rendering altamente modulare, progettato per offrire flessibilità e personalizzazione agli sviluppatori. Questo sistema è organizzato in diverse fasi chiave:

1. **Collect**: Determina gli oggetti da processare e renderizzare, gestendo la creazione delle viste di rendering e l'aggiornamento dei dati come le matrici di vista e proiezione.

2. **Extract**: Copia i dati dallo stato del gioco agli oggetti di rendering, preparando le informazioni necessarie per le fasi successive.

3. **Prepare**: Prepara le risorse per la GPU ed esegue calcoli complessi, come la computazione dei dati di illuminazione e la compilazione dei buffer costanti.

4. **Draw**: Genera la lista di comandi per la GPU, impostando le texture di rendering e disegnando le combinazioni di stadi di rendering con le viste appropriate.

Questa struttura modulare consente agli sviluppatori di estendere o modificare facilmente la pipeline di rendering in base alle esigenze specifiche del progetto. Ad esempio, è possibile implementare tecniche di rendering avanzate o ottimizzazioni specifiche intervenendo nelle diverse fasi del processo.

A partire dalla versione 3.1, Stride ha introdotto l'utilizzo del formato NuGet per la gestione dei pacchetti, non solo per le librerie di codice ma anche per gli asset del motore. Questo approccio consente agli sviluppatori di creare pacchetti personalizzati contenenti codice e risorse, facilitando la riusabilità e la condivisione tra progetti. L'integrazione con NuGet semplifica l'aggiunta di nuove funzionalità e asset ai progetti, migliorando la gestione delle dipendenze e assicurando la compatibilità tra diverse versioni.

Stride supporta inoltre un sistema di plugin in fase di sviluppo, progettato per estendere le funzionalità del motore senza la necessità di modificare il codice sorgente principale. Questo sistema di estensioni mira a fornire punti di integrazione sia a livello di runtime che nell'editor Game Studio, permettendo l'aggiunta di asset personalizzati, editor specifici e altre estensioni per migliorare l'esperienza di sviluppo.

Un ulteriore punto di forza, che può anche rappresentare una debolezza per alcuni, è l'approccio 'code-only' di Stride. Questo offre maggiore controllo e flessibilità, ma può risultare meno accessibile a chi preferisce strumenti visivi.

In sintesi, l'architettura modulare di Stride, combinata con l'uso di pacchetti NuGet e un sistema di plugin in evoluzione, offre agli sviluppatori un ambiente altamente flessibile e personalizzabile per lo sviluppo di giochi e applicazioni interattive, adattandosi facilmente alle esigenze specifiche di ogni progetto.

4.7.3 Unity

Unity è noto per la sua flessibilità e modularità. Grazie alla sua architettura componibile e a un vasto ecosistema di strumenti, Unity permette agli sviluppatori di adattare il motore alle esigenze specifiche di ogni progetto.

Uno degli elementi principali che contribuiscono alla modularità di Unity è il *Package Manager*, un sistema che consente di gestire e integrare pacchetti ufficiali, di terze parti

o sviluppati internamente. Attraverso questo strumento, gli sviluppatori possono aggiungere facilmente nuove funzionalità al progetto, come sistemi di input avanzati, librerie di fisica, strumenti di animazione o supporto per realtà virtuale e aumentata.

Unity adotta inoltre un'architettura *Entity-Component-System (ECS)*, che separa i dati (entità) dalla logica (componenti) e dal controllo del flusso (sistemi). Questo approccio non solo migliora le performance grazie a un uso efficiente della memoria e del parallelismo, ma offre anche un alto livello di modularità, permettendo di comporre oggetti e comportamenti in modo flessibile e riutilizzabile.

La flessibilità di Unity si estende anche al sistema di rendering e al suo supporto di diverse pipeline di rendering (vedi 4.2.1).

Unity offre anche un solido sistema di *plugin e asset* attraverso il suo *Asset Store*, dove è possibile trovare migliaia di pacchetti, script, modelli 3D, effetti visivi e audio che possono essere integrati direttamente nei progetti. Questo amplia ulteriormente le possibilità di estensione del motore e accelera i tempi di sviluppo.

Infine, la modularità di Unity si riflette anche nelle possibilità di scripting. Il motore supporta il linguaggio C#, ma consente l'integrazione di librerie esterne e l'uso di plugin nativi per estendere le funzionalità del progetto, garantendo agli sviluppatori pieno controllo sulla logica e sulle prestazioni.

In sintesi, la flessibilità e modularità di Unity, supportate da un ecosistema ricco e un'architettura componibile, lo rendono uno dei motori più versatili e adattabili per progetti di qualsiasi scala e complessità.

4.7.4 Motori con supporto per plugin e moduli estendibili

Diversi motori di gioco offrono architetture flessibili che consentono agli sviluppatori di estendere le funzionalità di base tramite plugin e moduli estendibili. Questo approccio facilita l'adattamento del motore alle esigenze specifiche di un progetto, migliorando la scalabilità e la possibilità di personalizzazione senza dover intervenire direttamente sul codice sorgente del motore.

Unreal Engine, CryEngine e Godot rientrano tra i motori che offrono un solido supporto per plugin e moduli estendibili, sebbene con approcci e filosofie leggermente differenti.

Unreal Engine si distingue per il suo sistema avanzato di plugin, che permette agli sviluppatori di estendere il motore attraverso moduli che possono essere abilitati o disabilitati a seconda delle esigenze del progetto. L'accesso completo al codice sorgente consente inoltre personalizzazioni profonde, che spaziano dalla creazione di nuovi sistemi di rendering fino all'integrazione di librerie esterne.

CryEngine, pur condividendo alcune somiglianze con Unreal in termini di estensibilità, presenta un ecosistema di plugin meno ampio e consolidato. Tuttavia, offre comunque la possibilità di integrare moduli personalizzati e di accedere al codice sorgente, permettendo agli sviluppatori di intervenire direttamente sulle funzionalità del motore.

Godot adotta un approccio modulare basato su un sistema a nodi, che consente di costruire scene e logiche di gioco componendo diversi elementi in modo gerarchico. Il motore supporta nativamente plugin sia per l'estensione dell'editor che per l'aggiunta di funzionalità al runtime. Gli sviluppatori possono scrivere plugin utilizzando GDScript, C#, o C++, ampliando così le possibilità di estensione. Inoltre, la natura completamente open-source di Godot facilita la modifica del motore stesso, consentendo agli sviluppatori di personalizzare ogni aspetto in base alle proprie necessità.

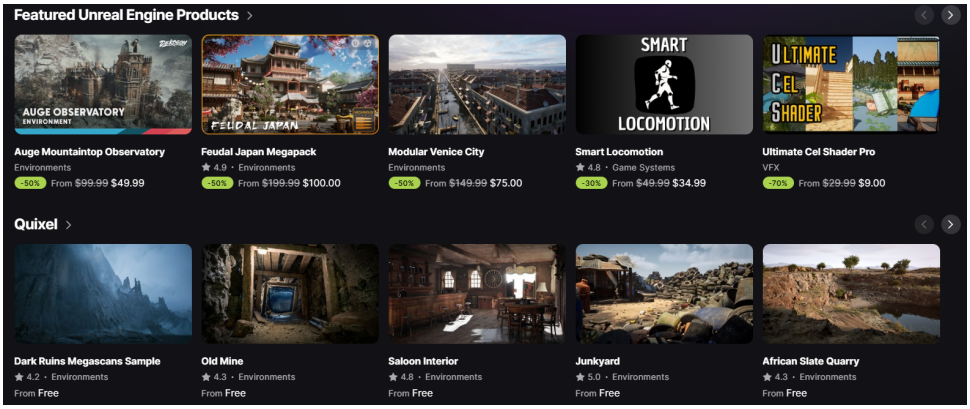

Figura 4.18: Fab, marketplace di proprietà Epic Games.

Un aspetto comune tra questi motori è la presenza di ampie comunità di sviluppo e marketplace dedicati, dove è possibile reperire numerosi plugin e moduli realizzati sia dai team ufficiali che da sviluppatori indipendenti. Questo ecosistema favorisce la condivisione di strumenti, asset e funzionalità, riducendo i tempi di sviluppo e ampliando le potenzialità del motore di base.

4.7.5 Motori leggeri con modularità di base

Alcuni motori di gioco, pur mantenendo un approccio più leggero rispetto alle soluzioni più avanzate, offrono un certo grado di modularità che consente agli sviluppatori di estendere le funzionalità principali con sistemi relativamente semplici. Questi motori sono particolarmente adatti a progetti meno complessi o che richiedono un impatto ridotto sulle risorse hardware.

Defold adotta un'architettura basata su collezioni e factory, che consente la creazione dinamica di oggetti e scene durante il runtime. Il motore supporta l'aggiunta di moduli esterni e asset tramite il proprio asset portal e permette l'integrazione di estensioni scritte in linguaggi nativi per estendere le funzionalità di base.

Cocos2d-x, grazie al suo codice open-source in C++, offre un certo grado di flessibilità nella modifica e nell'estensione delle funzionalità. Sebbene non presenti un sistema di plugin avanzato come altri motori, consente comunque l'integrazione di librerie esterne e l'aggiunta di moduli personalizzati per arricchire il progetto.

Entrambi i motori si distinguono per la loro leggerezza e semplicità, rendendoli ideali per sviluppatori indie o per progetti mobile e web-based. Pur non offrendo le stesse capacità di estensibilità dei motori più avanzati, forniscono comunque strumenti sufficienti per personalizzare e ampliare le funzionalità di base secondo le necessità dei progetti.

4.8 Accessibilità

L'accessibilità nei videogiochi è un aspetto fondamentale per garantire un'esperienza inclusiva a tutti i giocatori, indipendentemente dalle loro capacità fisiche, sensoriali o cognitive. Questa sezione analizza le funzionalità offerte dai motori per supportare lo sviluppo di videogiochi accessibili.

4.8.1 Definizione e importanza dell'accessibilità

L'accessibilità nei videogiochi si riferisce alla progettazione di esperienze interattive che possano essere fruite da un pubblico il più ampio possibile, incluse le persone con disabilità. Un gioco accessibile non solo migliora l'inclusione, ma amplia il bacino di giocatori e rende il medium più accogliente per tutti.

Le principali tipologie di disabilità che influenzano l'accessibilità nei videogiochi sono:

- Disabilità motorie: difficoltà nel muovere le mani, usare controller tradizionali o interagire con input complessi.
- Disabilità visive: cecità, ipovisione, daltonismo e difficoltà nella lettura di testi piccoli o di interfacce con scarso contrasto.
- Disabilità uditive: sordità e ipoacusia, che possono rendere difficile seguire dialoghi e segnali sonori importanti.
- Disabilità cognitive: difficoltà nell'elaborare informazioni complesse, tempi di reazione lenti, problemi di memoria o concentrazione.
- Disabilità comunicative: difficoltà nell'esprimersi verbalmente, come nel caso del mutismo, che possono rendere complessa l'interazione con sistemi che richiedono input vocali o comunicazione parlata.

L'implementazione dell'accessibilità nei motori di gioco permette di adattare l'esperienza alle diverse esigenze dei giocatori. Funzionalità come la personalizzazione dei controlli, l'uso di interfacce leggibili con opzioni per il daltonismo e sottotitoli dettagliati migliorano l'accessibilità per chi ha disabilità motorie, visive o uditive. Inoltre, la possibilità di ridurre stimoli visivi intensi e di regolare la difficoltà rende i giochi più inclusivi anche per chi ha esigenze cognitive specifiche.

Standard e linee guida di riferimento L'accessibilità nei videogiochi si basa su standard e linee guida consolidate, tra cui:

- Game Accessibility Guidelines, linee guida internazionali con 3 livelli di accessibilità progressive.
- Web Content Accessibility Guidelines (WCAG), riferimento per l'accessibilità digitale, utile anche per interfacce di gioco.
- XBox Accessibility Guidelines (XAG), serie di raccomandazioni sviluppate da Microsoft per garantire la compatibilità con controller adattivi e altre tecnologie.

4.8.2 Accessibilità per disabilità motorie

Le disabilità motorie possono limitare la capacità di utilizzare controller tradizionali, mouse o tastiera, rendendo difficile o impossibile interagire con i videogiochi. Per migliorare l'accessibilità, i motori di gioco devono fornire strumenti per la personalizzazione dei controlli, il supporto a dispositivi di input alternativi e la possibilità di semplificare le interazioni.

Supporto per controller adattivi e input alternativi I motori più avanzati permettono di utilizzare hardware alternativo, come l'Xbox Adaptive Controller, sistemi eye-tracking e comandi vocali:

- **Unity** (tramite il New Input System) e **Unreal Engine** offrono supporto nativo per dispositivi adattivi, inclusi controller specializzati.
- **Stride** permette di integrare joystick personalizzati e input alternativi tramite C#.
- **Godot** può essere configurato per supportare API di eye-tracking o controllo vocale, ma richiede più lavoro manuale rispetto ai motori precedenti.

Remappatura dei controlli e interfacce personalizzabili La personalizzazione dell'input è essenziale per adattare i controlli alle esigenze dei giocatori con difficoltà motorie, i motori ad offrire le soluzioni più interessanti sono:

- **Unreal Engine** dispone di un sistema avanzato di Input Mapping, che consente di assegnare più funzioni a un solo tasto o semplificare combinazioni complesse.
- **Construct 3**, grazie alla sua interfaccia a eventi, permette di creare controlli personalizzati senza dover scrivere codice, ideale per giochi che supportano schemi di input alternativi.
- **Defold** include opzioni native per la rimappatura dei tasti senza la necessità di codice avanzato.

Riduzione dell'input richiesto e automazione Alcuni motori permettono di automatizzare azioni complesse, riducendo il numero di input necessari per giocare:

- **Phaser** offre controllo automatizzato del personaggio, utile per ridurre la necessità di input costanti.
- **Cocos2d-x** consente di implementare facilmente funzionalità auto-run e movimenti semplificati, riducendo la necessità di pressione continua dei tasti.

Alcuni giochi hanno sfruttato queste funzionalità per garantire un'esperienza più accessibile, degli esempi sono *Sea of Thieves* (Unity), ottimizzato per funzionare con lo Xbox Adaptive Controller, e *A Short Hike* (Construct 3), che permette ai giocatori di scegliere controlli semplificati, riducendo la necessità di input rapidi o simultanei.

4.8.3 Accessibilità per disabilità visive

Le disabilità visive, che comprendono cecità, ipovisione e daltonismo, possono rendere difficile l'interazione con i videogiochi. Per garantire un'esperienza accessibile, i motori di gioco offrono strumenti come screen reader e filtri per il daltonismo.

Per i giocatori non vedenti o ipovedenti, il supporto agli **screen reader** è essenziale. **Unity** consente l'integrazione con lettori di schermo tramite plugin, mentre **Unreal Engine**, grazie al sistema Slate UI, permette di inviare output testuali ai dispositivi di sintesi vocale.

Per i giocatori con daltonismo, alcuni motori come **Unity** e **Unreal Engine** supportano **shader personalizzati** che modificano la gamma cromatica in base al tipo di

deficit visivo. Anche **Phaser**, grazie alla sua flessibilità nell'uso di filtri grafici, permette di implementare modalità ad alto contrasto.

Molti giochi hanno già implementato queste funzionalità con successo. *The Last of Us Part II* (Unreal Engine) include una modalità ad alto contrasto e una sintesi vocale che descrive menu e interfaccia, *Sea of Thieves* (Unity) offre un'opzione per daltonismo che modifica il colore degli indicatori di gioco, mentre *Celeste* (GameMaker) consente di ridurre gli effetti visivi e migliorare la leggibilità dell'ambiente di gioco.

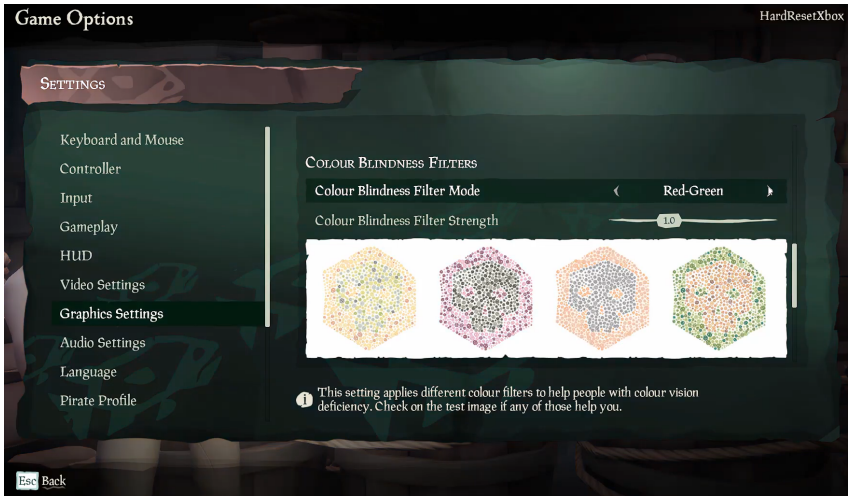

Figura 4.19: Impostazioni per modificare il colore in Sea of Thieves.

4.8.4 Accessibilità per disabilità uditive

Le disabilità uditive, come sordità e ipoacusia, possono limitare l'esperienza di gioco quando elementi cruciali, come dialoghi e segnali sonori, non hanno alternative visive o tattili. Per migliorare l'accessibilità, i motori di gioco devono supportare sottotitoli avanzati, indicatori visivi per i suoni e personalizzazione dell'audio.

I sottotitoli non devono limitarsi a trascrivere i dialoghi, ma includere informazioni essenziali sugli effetti sonori, come la direzione da cui provengono o la loro intensità.

Per chi non può percepire i segnali acustici, gli indicatori visivi per i suoni sono essenziali. **Unreal Engine** offre strumenti per visualizzare suoni come onde grafiche o icone direzionali, mentre **Unity** consente di integrare HUD dinamici che segnalano la fonte dei suoni attraverso effetti visivi. **Defold** e **Phaser**, pur essendo motori più leggeri, permettono di implementare segnali visivi sincronizzati con gli effetti sonori, utili in giochi che devono trasmettere informazioni cruciali senza audio.

4.8.5 Accessibilità per disabilità cognitive

Le disabilità cognitive possono influenzare la capacità di comprendere interfacce complesse, gestire tempi di reazione rapidi o mantenere la concentrazione per lunghi periodi. Per migliorare l'accessibilità, i motori di gioco devono offrire strumenti che permettano la personalizzazione della difficoltà, la semplificazione dell'interfaccia e il controllo del ritmo di gioco.

Molti giochi integrano opzioni per **adattare la difficoltà** ai bisogni dei giocatori, regolando parametri come la velocità di gioco, l'aggressività dell'intelligenza artificiale o

il numero di azioni richieste per proseguire. Unreal Engine e Unity permettono di implementare sistemi di difficoltà dinamica, che modificano il comportamento degli avversari in base alle prestazioni del giocatore. Un'interfaccia chiara e intuitiva è fondamentale per chi ha difficoltà cognitive. Godot e Defold offrono strumenti per ridurre il numero di elementi visivi sullo schermo, semplificare i menu e ingrandire testi e pulsanti per una navigazione più intuitiva. Phaser, utilizzato per giochi web, permette di creare interfacce con interazioni minime, riducendo la necessità di input simultanei e facilitando l'apprendimento delle meccaniche di gioco.

Anche il controllo del ritmo di gioco è essenziale. Alcuni motori permettono di implementare opzioni per disattivare timer o ridurre il numero di azioni richieste per completare una sfida. GameMaker, ad esempio, consente di aggiungere modalità a ritmo più lento, mentre Cocos2d-x offre strumenti per modificare la velocità degli eventi in base alle preferenze del giocatore.

Diversi giochi hanno adottato queste strategie per migliorare l'accessibilità cognitiva.*Animal Crossing: New Horizons* (Unity) è un esempio di gioco progettato con un ritmo rilassato, privo di scadenze pressanti o sfide complesse, rendendolo accessibile a una vasta gamma di giocatori. *Kind Words* (Phaser) utilizza un'interfaccia estremamente semplificata, riducendo al minimo le distrazioni visive e cognitive.

4.8.6 Accessibilità per disabilità comunicative

Le disabilità linguistiche e comunicative, come mutismo, afasia o difficoltà nell'elaborazione del linguaggio scritto e parlato, possono rendere difficile l'interazione con i videogiochi che richiedono input vocali o una comprensione avanzata del testo. Per garantire l'accessibilità, i motori di gioco devono supportare alternative agli input vocali, opzioni di semplificazione del linguaggio e strumenti di traduzione e sintesi testuale.

Molti giochi moderni utilizzano comandi vocali per interagire con il mondo di gioco o con altri giocatori. Per chi non può parlare, è essenziale fornire alternative basate su input testuali o selezioni rapide. Unreal Engine e Unity permettono di sostituire i comandi vocali con menu interattivi o tastiere virtuali. Godot, grazie alla sua natura open-source, può essere configurato per integrare sistemi di input alternativo, mentre GDevelop offre un'interfaccia drag-and-drop per creare comandi basati su icone o pulsanti, riducendo la necessità di digitare testi complessi.

Anche la comprensione del linguaggio può essere un ostacolo. Construct 3 e Phaser permettono di implementare modalità di testo semplificato, che riducono la complessità delle descrizioni e dei dialoghi per renderli più accessibili a chi ha difficoltà linguistiche o cognitive. Defold consente di integrare dizionari personalizzati per adattare il linguaggio ai bisogni del giocatore, mentre Cocos2d-x supporta la sintesi vocale per convertire il testo scritto in parlato, facilitando la fruizione per chi ha difficoltà nella lettura.

Infine, la traduzione e adattamento del testo è un elemento cruciale. Alcuni motori, come Unity e Unreal Engine, permettono l'integrazione di API di traduzione automatica o sottotitoli generati dinamicamente, offrendo un'esperienza più inclusiva per chi ha difficoltà a comprendere il linguaggio del gioco. Godot e Phaser, grazie al loro supporto per script personalizzati, possono essere configurati per offrire dizionari interattivi o alternative visive ai contenuti testuali.

Diversi giochi hanno già implementato queste funzionalità. *The Sims 4* (Unity) utilizza un linguaggio semplificato nei dialoghi, evitando termini complessi mentre *Journey*

(Phaser) elimina del tutto il linguaggio scritto e parlato, basando la comunicazione tra giocatori su simboli e suoni.

4.8.7 Confronto tra i motori di gioco e conclusioni

L'analisi dell'accessibilità nei motori di gioco evidenzia come le diverse piattaforme offrano soluzioni specifiche per supportare i giocatori con disabilità. Tuttavia, il livello di accessibilità varia a seconda delle funzionalità native disponibili, della possibilità di personalizzazione e della facilità di implementazione di strumenti inclusivi.

Unreal Engine e Unity emergono come le soluzioni più complete e versatili, offrendo supporto avanzato per dispositivi adattivi, personalizzazione dei controlli, sottotitoli avanzati, indicatori visivi per i suoni e modifiche dinamiche della difficoltà. Entrambi permettono di integrare API di terze parti, rendendoli adatti a progetti con esigenze di accessibilità complesse. Tuttavia, la loro implementazione può richiedere competenze tecniche avanzate.

Godot e Defold, pur essendo più leggeri, offrono una notevole flessibilità grazie alla loro natura open-source, permettendo di integrare funzionalità di accessibilità personalizzate, anche se spesso richiedono uno sforzo aggiuntivo da parte degli sviluppatori. **Stride** offre un buon livello di personalizzazione dei controlli e il supporto a dispositivi di input alternativi.

Construct 3 e GDevelop, grazie ai loro sistemi di sviluppo senza codice, permettono un'implementazione immediata di interfacce semplificate, comandi alternativi e supporto ai sottotitoli, risultando ideali per progetti con accessibilità di base senza necessità di programmazione avanzata. **Phaser e Cocos2d-x**, invece, si distinguono per la possibilità di integrare indicatori visivi per l'audio e modalità testuali semplificate, pur con limitazioni nei sistemi di personalizzazione avanzata.

In conclusione, la scelta del motore di gioco più adatto all'accessibilità dipende dalle necessità del progetto:

- Per un **supporto completo e avanzato** → **Unreal Engine e Unity** sono le opzioni migliori.
- Per **soluzioni personalizzabili ma più leggere** → **Godot e Defold** offrono una buona flessibilità.
- Per **facilità di implementazione senza codice** → **Construct 3 e GDevelop** risultano le scelte più accessibili.
- Per **giochi 2D e HTML5 con interfacce semplificate** → **Phaser e Cocos2d-x** rappresentano valide alternative.

L'accessibilità nei videogiochi è un elemento sempre più centrale nello sviluppo moderno, e l'integrazione di queste funzionalità non solo rende i giochi più inclusivi, ma amplia anche il pubblico potenziale, garantendo un'esperienza più equa e coinvolgente per tutti.

Capitolo 5

Tecnologie emergenti nei motori di gioco: RTX e DLSS

Negli ultimi anni, il settore dei videogiochi ha visto un'evoluzione significativa nelle tecnologie di rendering, con l'obiettivo di ottenere una qualità visiva sempre più vicina al fotorealismo senza compromettere le prestazioni. L'avanzamento delle schede grafiche e l'introduzione di nuovi algoritmi di rendering hanno permesso agli sviluppatori di superare molte delle limitazioni tecniche del passato, offrendo esperienze sempre più immersive e dettagliate.

Due delle tecnologie più rivoluzionarie in questo contesto sono il **Ray Tracing** in tempo reale, reso possibile dalle GPU della serie RTX di NVIDIA, e il **Deep Learning Super Sampling** (DLSS), una tecnica avanzata di upscaling basata sull'intelligenza artificiale. Queste innovazioni hanno ridefinito il rendering nei videogiochi, consentendo di ottenere immagini di altissima qualità con riflessi, ombre e illuminazione globale molto più realistiche, mantenendo al contempo elevate prestazioni.

L'implementazione di RTX e DLSS nei motori di gioco non è uniforme e varia in base alle architetture grafiche di ciascun engine. Unreal Engine, ad esempio, è stato uno dei primi a integrare il supporto completo per il Ray Tracing e per DLSS, mentre Unity ha adottato un approccio più modulare, con il Ray Tracing accessibile attraverso l'High Definition Render Pipeline. Altri motori, come CryEngine, hanno sviluppato soluzioni alternative come il sistema di illuminazione globale SVOGI, che offre un effetto simile al Ray Tracing senza la necessità di hardware dedicato.

Questo capitolo esplorerà nel dettaglio queste tecnologie, partendo da una spiegazione del funzionamento del Ray Tracing e delle sue applicazioni nei videogiochi, per poi analizzare DLSS e il suo impatto sulle prestazioni. Infine, verrà approfondito il modo in cui RTX e DLSS vengono implementati nei principali motori di gioco e come queste tecnologie influenzano le decisioni degli sviluppatori, sia nel settore AAA che in quello indie.

5.1 RTX: Cos'è e come funziona

Il Ray Tracing è una tecnica di rendering che simula il comportamento fisico della luce per ottenere immagini con un livello di realismo superiore rispetto ai metodi tradizionali basati sulla rasterizzazione. A differenza di quest'ultima, che approssima l'illuminazione tramite tecniche come le shadow maps, il ray tracing calcola esplicitamente la propa-

gazione dei raggi luminosi e la loro interazione con le superfici della scena, producendo ombre, riflessi e illuminazione globale con una precisione molto maggiore.

5.1.1 Differenze tra Rasterizzazione e Ray Tracing

Nella rasterizzazione, l'immagine viene generata convertendo i triangoli della scena in pixel dello schermo. Le tecniche di shading e illuminazione utilizzate in questo metodo sono spesso precalcolate o basate su approssimazioni per simulare riflessi e ombre, il che comporta alcune limitazioni:

- Illuminazione indiretta: la rasterizzazione si basa su lightmaps o screen-space reflections, che non possono simulare accuratamente la propagazione della luce nello spazio tridimensionale.
- Riflessi e trasparenze: gli oggetti riflessi devono essere pre-renderizzati, risultando in artefatti visivi quando sono fuori dallo schermo.
- Ombre dinamiche: tecniche come le shadow maps soffrono di aliasing e richiedono un elevato utilizzo di memoria.

Il ray tracing risolve questi problemi tracciando il percorso dei raggi di luce dalla telecamera attraverso la scena, calcolando in modo accurato la loro interazione con gli oggetti e la loro propagazione nell'ambiente[17].

5.1.2 Funzionamento del Ray Tracing

Il Ray Tracing calcola il colore di ogni pixel seguendo un algoritmo suddiviso in più fasi[18]:

1. Generazione dei raggi primari

Per ogni pixel dell'immagine, un raggio viene lanciato dalla telecamera (o dal punto di vista dell'osservatore) attraverso il piano dello schermo. Questo raggio, chiamato **raggio primario**, viaggia in linea retta e può incontrare diversi oggetti nella scena.

2. Intersezione con le superfici

Si determina quale oggetto viene colpito per primo dal raggio. Per farlo, il motore di rendering utilizza strutture di accelerazione spaziale come le **Bounding Volume Hierarchies** (BVH) (vedi 5.1.4).

- Se il raggio non colpisce alcun oggetto, il pixel assume il colore dello sfondo.
- Se colpisce un oggetto, vengono recuperate le proprietà del materiale della superficie (colore, rugosità, trasparenza, ecc.).

3. Calcolo dell'illuminazione e dei raggi secondari

Una volta che un raggio ha colpito un oggetto, vengono generati nuovi raggi per simulare l'interazione con la luce:

- Raggi di ombra: un raggio viene lanciato verso le sorgenti luminose per determinare se il punto è illuminato o in ombra. Se il raggio viene bloccato da un altro oggetto, il pixel è in ombra.
- Raggi di riflessione: se la superficie è riflettente (es. metallo lucido, specchio), viene generato un raggio riflesso seguendo la **legge della riflessione speculare** ($\theta_i = \theta_r$), dove θ_i è l'angolo di incidenza e θ_r quello di riflessione.
- Raggi di rifrazione: Nei materiali trasparenti come vetro o acqua, il raggio cambia direzione secondo la **legge di Snell**, in base all'indice di rifrazione ($n_1 \sin\theta_1 = n_2 \sin\theta_2$).
- Illuminazione globale: in un sistema fisicamente accurato, la luce rimbalza molte volte prima di raggiungere l'osservatore. Il **Path Tracing**, una tecnica avanzata del Ray Tracing, genera raggi secondari multipli per simulare la propagazione indiretta della luce, accumulando informazioni sull'**illuminazione globale**.

4. Accumulo del colore e compositing finale

Il colore finale di ogni pixel è determinato dall'intensità della luce e dalle proprietà del materiale della superficie, secondo il modello matematico descritto dall'Equazione di Rendering di Kajiya [19].

Questa equazione esprime il comportamento della luce in una scena tridimensionale come un'integrale che descrive la distribuzione dell'illuminazione globale, tenendo conto di riflessioni, rifrazioni e scattering. Il processo viene ripetuto per tutti i pixel dell'immagine, risultando in un rendering fotorealistico con illuminazione accurata, ombre realistiche e riflessi coerenti con il comportamento fisico della luce.

5.1.3 Tipologie di Ray Tracing

Il Ray Tracing non è un'unica tecnica, ma un insieme di varianti ottimizzate per diversi scenari di rendering. Ogni approccio presenta un compromesso tra qualità dell'immagine e costo computazionale. Le principali tecniche di Ray Tracing sono descritte di seguito.

Ray Casting

Il **Ray Casting** rappresenta la forma più semplice di Ray Tracing. Viene utilizzato esclusivamente per determinare la visibilità degli oggetti in una scena, senza calcolare riflessi, ombre o illuminazione globale. Il suo principale utilizzo è nei motori grafici più basilari e nei sistemi di *collision detection*.

Whitted Ray Tracing

Il modello proposto da **Turner Whitted** nel 1980 [17] ha introdotto un'estensione del Ray Tracing che supporta **riflessioni speculari e trasparenze**. Questo metodo prevede il tracciamento di raggi secondari per simulare:

- Riflessi realistici su superfici speculari.
- Rifrazione della luce nei materiali trasparenti secondo la legge di Snell.
- Ombre accurate, determinando se un oggetto blocca un raggio di luce.

Questo approccio permette di ottenere un maggiore livello di realismo rispetto al Ray Casting, ma ha un costo computazionale elevato.

Path Tracing

Il **Path Tracing** è una tecnica basata su **campionamento Monte Carlo** che simula il trasporto della luce attraverso molteplici rimbalzi [19]. A differenza del Whitted Ray Tracing, non si limita a raggi secondari diretti ma genera raggi casuali per calcolare **l'illuminazione globale** in modo fisicamente accurato.

Il Path Tracing segue i principi dell'**Equazione di Rendering di Kajiya**, integrando il contributo della luce in tutta la scena. Sebbene offra risultati altamente fotorealistici, il suo costo computazionale elevato lo rende più adatto a **rendering offline**, come nel cinema e nell'architettura.

Real-Time Ray Tracing

Il **Ray Tracing in tempo reale** è diventato una realtà grazie ai recenti progressi nell'hardware grafico, che hanno permesso di superare le limitazioni computazionali di questa tecnica. Le moderne GPU integrano unità dedicate per accelerare il Ray Tracing, rendendolo utilizzabile nei videogiochi mantenendo framerate accettabili.

Tra le tecnologie che hanno reso possibile il Real-Time Ray Tracing, la più avanzata e diffusa è la serie **RTX** di NVIDIA, che introduce componenti hardware specializzati e un ecosistema software ottimizzato. Accanto a RTX, anche AMD ha sviluppato il proprio supporto con **Radeon Ray Accelerators**, mentre API come DirectX Raytracing e Vulkan Ray Tracing forniscono le basi per l'implementazione di queste tecnologie nei motori di gioco.

RTX rimane il riferimento principale per il Ray Tracing in tempo reale, offrendo un'integrazione più avanzata e strumenti come DLSS, che permettono di migliorare le prestazioni senza sacrificare la qualità visiva.

Queste innovazioni hanno reso possibile l'uso del Ray Tracing in giochi come *Cyberpunk 2077*, *Control* e *Minecraft RTX*, migliorando significativamente la qualità visiva mantenendo framerate giocabili.

5.1.4 RTX di NVIDIA: Ray Tracing in Tempo Reale

Il Ray Tracing è stato per decenni una tecnologia esclusiva del rendering offline, utilizzata nell'industria cinematografica e nella CGI a causa della sua complessità computazionale. Tuttavia, con l'introduzione delle GPU **RTX** di NVIDIA, il Ray Tracing è diventato accessibile anche in tempo reale, grazie a un'architettura hardware specializzata e a ottimizzazioni software avanzate.

RTX non è solo un marchio commerciale, ma un ecosistema di tecnologie progettate per migliorare la qualità del rendering senza sacrificare le prestazioni. Il successo di questa tecnologia si basa su due pilastri principali: **hardware dedicato** e **ottimizzazioni software**.

Miglioramenti hardware: RT Cores e Tensor Cores

Per rendere il Ray Tracing praticabile in tempo reale, le GPU RTX integrano unità di calcolo specifiche che riducono il carico computazionale rispetto alle architetture tradizionali.

RT Cores: Accelerazione del Ray Tracing Gli **RT Cores** (Ray Tracing Cores) sono unità hardware specializzate progettate per gestire il calcolo delle intersezioni raggio-superficie. Questo processo, estremamente oneroso in una GPU standard, viene ottimizzato tramite **Bounding Volume Hierarchies (BVH)**, una struttura dati gerarchica che organizza la geometria della scena per ridurre il numero di calcoli necessari.

I **RT Cores** eseguono in parallelo due operazioni fondamentali:

- **Traversing BVH**: ricerca ottimizzata delle intersezioni tra i raggi e le superfici della scena, accelerando il processo di rendering.
- **Bounding Box Intersection**: calcolo delle collisioni tra i raggi e i volumi della scena, determinando quali oggetti devono essere ulteriormente processati.

Grazie agli RT Cores, le GPU RTX possono eseguire il Ray Tracing in tempo reale senza impattare in modo eccessivo sulle prestazioni, permettendo di ottenere riflessi, ombre e illuminazione globale più realistici.

Tensor Cores: Deep Learning per il Rendering Oltre agli RT Cores, le GPU RTX includono i **Tensor Cores**, unità di calcolo specializzate nel deep learning. Sebbene originariamente progettati per applicazioni di intelligenza artificiale, questi core sono stati adattati per migliorare il Ray Tracing in tempo reale tramite tecniche avanzate come:

- **Denoising AI**: riduzione del rumore generato dal basso numero di raggi tracciati per pixel, migliorando la qualità dell'immagine senza aumentare il costo computazionale.
- **DLSS (Deep Learning Super Sampling)**: upscaling delle immagini a bassa risoluzione utilizzando modelli di intelligenza artificiale, permettendo di ottenere una qualità visiva superiore senza impattare il framerate. Questa tecnologia verrà approfondita nella Sezione 5.2.

L'architettura di una GPU RTX combina quindi **RT Cores, Tensor Cores e unità di shading tradizionali**, ottimizzando il flusso di calcolo per bilanciare qualità visiva e prestazioni.

Miglioramenti software: ottimizzazioni per il Ray Tracing

Oltre alle innovazioni hardware, NVIDIA ha sviluppato una serie di ottimizzazioni software per rendere il Ray Tracing più efficiente nei videogiochi.

Hybrid Rendering: combinazione di rasterizzazione e Ray Tracing Sebbene il Ray Tracing in tempo reale sia oggi possibile, è ancora troppo costoso per essere utilizzato per l'intera pipeline di rendering. Per questo motivo, RTX adotta un approccio ibrido, combinando rasterizzazione e Ray Tracing selettivo, applicandolo solo a effetti visivi specifici come:

- **Riflessi realistici**: il Ray Tracing viene utilizzato solo per le superfici riflettenti, mentre il resto della scena è renderizzato con tecniche tradizionali.
- **Ombre dinamiche**: migliora la qualità delle ombre riducendo gli artefatti tipici delle shadow maps.
- **Illuminazione globale**: alcune implementazioni avanzate utilizzano il Ray Tracing per simulare il comportamento fisico della luce, migliorando l'accuratezza della scena.

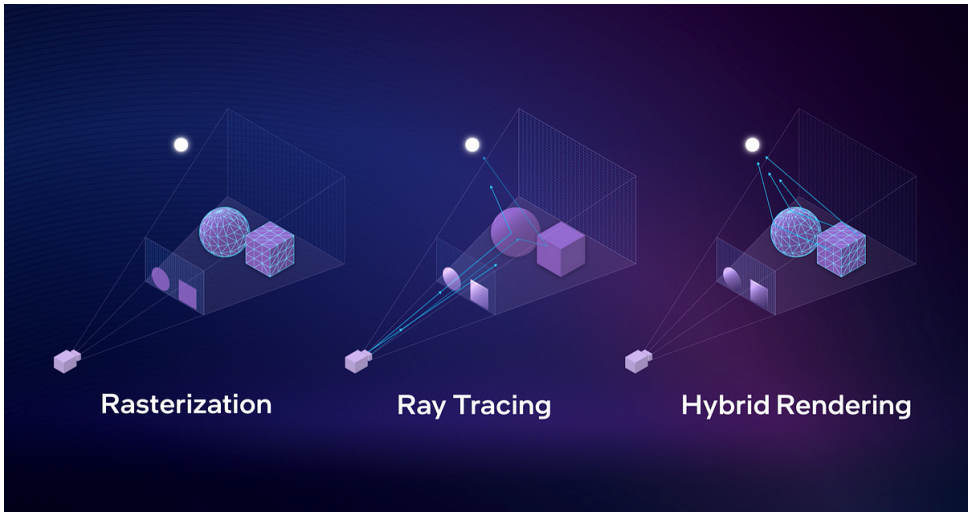

Figura 5.1: Rasterization VS Ray Tracing VS Hybrid Rendering.

Denoising AI e gestione avanzata del rumore Poiché nei giochi il numero di raggi tracciati per pixel è limitato per mantenere prestazioni accettabili, il Ray Tracing in tempo reale genera spesso rumore visibile. NVIDIA ha sviluppato soluzioni basate su AI per mitigare questo problema:

- **Filtri di Denoising**: algoritmi avanzati che riducono il rumore delle immagini senza degradarne la qualità.
- **Super-Risoluzione con AI**: attraverso i Tensor Cores, il Ray Tracing viene combinato con tecniche di deep learning per ottenere un'immagine più pulita e dettagliata.

API per il Ray Tracing: DXR e Vulkan L'integrazione di RTX nei motori di gioco è supportata da API avanzate, che permettono agli sviluppatori di implementare il Ray Tracing in modo efficiente:

- **DirectX Raytracing (DXR)**: API Microsoft per il Ray Tracing, parte di DirectX 12, utilizzata da motori come Unreal Engine.
- **Vulkan Ray Tracing**: alternativa open-source che offre funzionalità simili a DXR, supportata da AMD e utilizzata in giochi come DOOM Eternal.

Queste tecnologie software, combinate con l'hardware RTX, hanno reso il Ray Tracing scalabile e accessibile nei videogiochi moderni. Per più informazioni sulle API DirectX e Vulkan vedere 2.2.1.

RTX ha trasformato il Ray Tracing in tempo reale da un concetto sperimentale a una tecnologia ampiamente adottata nell'industria videoludica. L'integrazione di **RT Cores, Tensor Cores e ottimizzazioni software** ha permesso di bilanciare realismo e prestazioni, portando effetti grafici avanzati nei giochi più recenti. Con il continuo sviluppo dell'architettura RTX e il miglioramento delle API di Ray Tracing, questa tecnologia è destinata a diventare uno standard per il rendering in tempo reale nei videogiochi e oltre.

5.2 DLSS: Cos'è e come funziona

Il *Deep Learning Super Sampling* (DLSS) è una tecnologia sviluppata da NVIDIA che sfrutta l'intelligenza artificiale per migliorare le prestazioni grafiche nei videogiochi senza sacrificare la qualità dell'immagine. DLSS è una componente essenziale della tecnologia RTX, poiché consente l'uso del Ray Tracing in tempo reale mantenendo un framerate accettabile. Senza DLSS, il Ray Tracing sarebbe computazionalmente troppo oneroso, in quanto richiederebbe il rendering di immagini ad altissima risoluzione per ottenere un livello di dettaglio simile.

DLSS affronta questa sfida utilizzando una rete neurale avanzata che esegue un *upscaling* intelligente delle immagini renderizzate a bassa risoluzione, ricostruendo dettagli e migliorando la nitidezza grazie ai Tensor Cores delle GPU NVIDIA RTX. Questo processo permette di ottenere immagini paragonabili a quelle renderizzate nativamente ad alta risoluzione, riducendo significativamente il carico sulla GPU e migliorando le prestazioni complessive del gioco.

La tecnologia DLSS ha subito diverse evoluzioni nel tempo, migliorando la qualità dell'immagine e introducendo nuove funzionalità come il *Frame Generation* con DLSS 3. Inoltre, si pone in diretta competizione con altre soluzioni di *upscaling*, come il FidelityFX Super Resolution (FSR) di AMD e il Temporal Super Resolution (TSR) di Unreal Engine 5, che verranno confrontate nelle sezioni successive.

5.2.1 Funzionamento tecnico del DLSS

Il DLSS si basa su un modello di *Super-Resolution* basato sull'intelligenza artificiale, che sfrutta una rete neurale convoluzionale ricorrente (RCN) per ricostruire dettagli e migliorare la nitidezza dell'immagine. Questo processo viene accelerato dai Tensor Cores, consentendo un'esecuzione in tempo reale senza un impatto significativo sulle prestazioni.

Il ruolo della rete neurale

DLSS utilizza una rete neurale addestrata su immagini ad altissima risoluzione per apprendere a ricostruire i dettagli mancanti nelle immagini a risoluzione inferiore. Il modello di rete è stato sviluppato e ottimizzato da NVIDIA attraverso un processo di deep learning, in cui migliaia di frame di giochi vengono analizzati da supercomputer per migliorare la qualità dell'upscaling.

L'addestramento della rete si concentra su due aspetti principali:

- **Ricostruzione dei dettagli**: la rete è in grado di prevedere e ripristinare dettagli ad alta frequenza, come texture fini e bordi netti.
- **Riduzione degli artefatti visivi**: grazie al feedback temporale tra i frame consecutivi, DLSS riduce il *ghosting*, lo *shimmering* e altri problemi di rendering.

Pipeline del rendering con DLSS

La pipeline di rendering di DLSS segue questi passaggi:

1. Il gioco viene renderizzato inizialmente a una risoluzione inferiore rispetto a quella di output desiderata, riducendo il carico computazionale sulla GPU.
2. I dati del frame vengono inviati alla rete neurale di DLSS, che utilizza i frame precedenti e altre informazioni per migliorare la qualità dell'immagine.
3. La rete neurale genera un frame ad alta risoluzione con dettagli ricostruiti e bordi affinati.
4. Il frame migliorato viene quindi combinato con effetti di post-processing, come l'occlusione ambientale, il motion blur e l'illuminazione globale, prima di essere visualizzato sullo schermo.

L'importanza dei Motion Vectors

Per garantire una qualità dell'immagine elevata e prevenire artefatti, DLSS utilizza i *motion vectors*, ovvero dati che descrivono il movimento degli oggetti tra due frame consecutivi. Questi vettori permettono alla rete neurale di prevedere come gli oggetti si muoveranno nel tempo, evitando distorsioni e mantenendo un'elevata coerenza visiva.

L'utilizzo dei motion vectors permette a DLSS di ottenere:

- Maggiore **stabilità dell'immagine**: evitando che dettagli statici appaiano sfocati o distorti nei frame successivi.
- **Riduzione degli artefatti** di upscaling: prevenendo il *ghosting* e il *flickering* nei movimenti rapidi.
- **Ottimizzazione delle prestazioni**: consentendo al sistema di concentrarsi sulla ricostruzione dei dettagli più rilevanti in ogni frame.

Grazie a questa combinazione di tecniche, DLSS non solo migliora la qualità visiva ma permette anche di ottenere prestazioni significativamente superiori rispetto al rendering nativo ad alta risoluzione, rendendo possibile l'utilizzo del Ray Tracing senza sacrificare il framerate.

5.2.2 Evoluzione delle versioni di DLSS

Nel corso degli anni, la tecnologia del DLSS è stata migliorata attraverso diverse iterazioni, ciascuna con nuove funzionalità e ottimizzazioni per adattarsi meglio alle esigenze dei giochi moderni.

DLSS 1.0: L'introduzione della tecnologia

La prima versione di DLSS, lanciata nel 2018 con le GPU RTX 20-series, utilizzava una rete neurale specifica addestrata su ogni singolo gioco. Il funzionamento di DLSS 1.0 si basava su un modello di super-risoluzione AI che migliorava la qualità dell'immagine a partire da un rendering a risoluzione inferiore. Tuttavia, presentava diverse limitazioni:

- Necessitava di un addestramento specifico per ogni gioco, riducendo la sua applicabilità generale.
- Introduceva artefatti visivi, come sfocature e perdita di dettagli nelle texture.
- Supportava solo alcune risoluzioni e modalità predefinite, limitando la flessibilità dell'implementazione.

A causa di queste problematiche, DLSS 1.0 non ha avuto un'adozione diffusa ed è stato rapidamente sostituito da una versione migliorata.

DLSS 2.0: Upscaling generalizzato e miglioramenti qualitativi

Con DLSS 2.0, rilasciato nel 2020, NVIDIA ha risolto molte delle limitazioni della prima versione, introducendo un modello neurale più avanzato e generalizzato. I principali miglioramenti includono:

- **Modello di super-risoluzione indipendente dal gioco**: DLSS 2.0 non richiede più un addestramento specifico per ogni titolo, rendendolo più flessibile e facilmente implementabile dagli sviluppatori.
- **Maggiore qualità dell'immagine**: Grazie a un nuovo algoritmo basato su *deep learning*, DLSS 2.0 offre immagini più nitide con meno artefatti rispetto alla versione precedente.
- **Diversi preset di qualità**: DLSS 2.0 introduce tre modalità – Qualità, Bilanciata e Prestazioni – consentendo ai giocatori di scegliere tra maggiore nitidezza dell'immagine o incremento del framerate.
- **Supporto ai Motion Vectors**: L'integrazione dei *motion vectors* riduce il ghosting e migliora la stabilità dell'immagine in movimento.

DLSS 3.0: Introduzione del Frame Generation

Con il lancio delle GPU RTX 40-series nel 2022, NVIDIA ha introdotto DLSS 3.0, che include un'importante innovazione: il **Frame Generation**. Oltre a migliorare ulteriormente l'upscaling basato su AI, questa versione introduce:

- **Frame Generation**: Utilizzando gli *Optical Flow Accelerators* delle GPU RTX 40-series, DLSS 3 è in grado di generare interi frame sintetici per incrementare il framerate. Questo consente di ottenere un significativo miglioramento delle prestazioni, ma può introdurre latenza extra.
- **Migliore qualità dell'immagine**: Il modello AI aggiornato migliora la nitidezza e riduce ulteriormente gli artefatti.

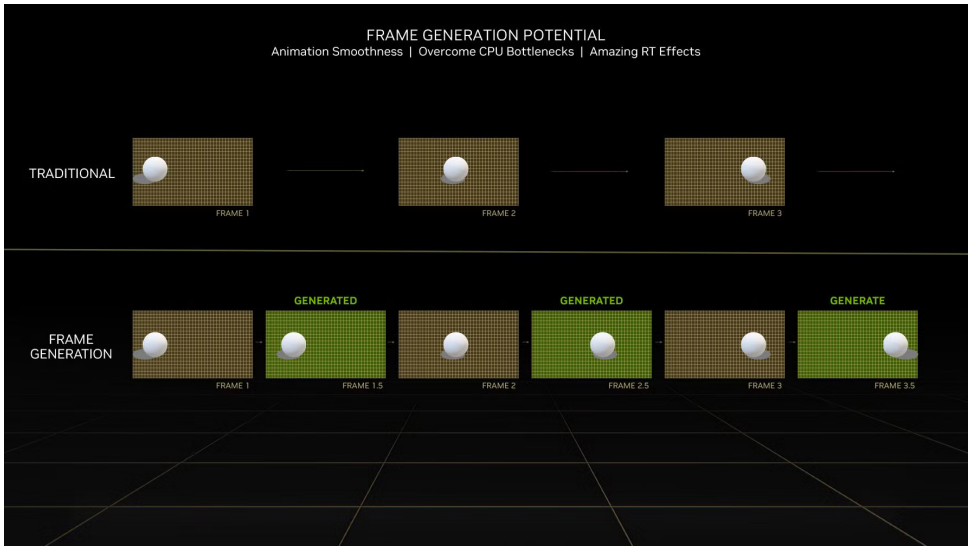

Figura 5.2: Esempio di frame generati tramite Frame Generation.

- **Compatibilità con Reflex**: Per compensare l'aumento della latenza dovuto al Frame Generation, DLSS 3 è integrato con **NVIDIA Reflex**, che ottimizza la catena di rendering per ridurre il ritardo dell'input.

DLSS 3 ha avuto una grande adozione nei giochi moderni, in particolare nei titoli con grafica avanzata e ray tracing intensivo.

DLSS 4.0: Multi-Frame Generation e Vision Transformer

DLSS 4.0, rilasciato con le GPU RTX 50-series, rappresenta un'evoluzione significativa rispetto a DLSS 3, introducendo nuove tecnologie basate su **modelli Transformer** e migliorando la generazione dei frame sintetici. Gli avanzamenti principali includono:

- **Vision Transformer per il Supercampionamento**: DLSS 4.0 sostituisce le reti neurali convoluzionali (CNN) con un modello **Vision Transformer (ViT)**, migliorando la qualità dell'upscaling. Questo nuovo approccio permette una ricostruzione più accurata dei dettagli a lungo raggio, riducendo artefatti come il ghosting e lo shimmering.

- **Multi-Frame Generation**: Evoluzione del *Frame Generation*, questa tecnica permette di generare fino a **tre frame sintetici per ogni frame renderizzato**, anziché uno solo. Questo incrementa notevolmente il framerate percepito, pur mantenendo una fluidità elevata. Tuttavia, questa funzionalità è esclusiva delle GPU RTX 50-series, mentre le RTX 40-series rimangono limitate al rapporto 1:1.

- **Ottimizzazione della latenza con NVIDIA Reflex**: Per mitigare il ritardo introdotto dal *Multi-Frame Generation*, DLSS 4.0 integra una versione migliorata di **NVIDIA Reflex** (2), riducendo il tempo di risposta degli input e migliorando la reattività nei giochi competitivi.

- **Maggiore compatibilità con i giochi e i motori di rendering**: NVIDIA ha semplificato l'integrazione di DLSS 4.0 nei principali motori di gioco, garantendo una più ampia adozione della tecnologia. Al momento del lancio, oltre 75 titoli supportano il Multi-Frame Generation, tra cui *Alan Wake 2* e *Cyberpunk 2077*.

Grazie all'uso dei Transformer e all'introduzione del *Multi-Frame Generation*, DLSS 4.0 non solo migliora significativamente le prestazioni, ma ridefinisce il modo in cui il rendering AI viene implementato nei giochi di nuova generazione.

Evoluzione futura e prospettive

Con l'avanzare delle GPU e delle tecnologie AI, DLSS continuerà a evolversi, mirando a migliorare ulteriormente la qualità dell'immagine e a ridurre l'impatto sulla latenza. Tra le possibili evoluzioni future, si ipotizza l'integrazione con sistemi di rendering neurale avanzati, capaci di generare intere scene con tecniche di deep learning in tempo reale.

L'adozione crescente di DLSS nei giochi e nei motori grafici dimostra l'importanza delle tecnologie di super-risoluzione basate su AI, non solo per migliorare il framerate, ma anche per ridefinire il modo in cui i giochi vengono renderizzati e ottimizzati sulle nuove generazioni di hardware.

5.2.3 DLSS vs Tecniche alternative

Le tecniche di **upscaling** e **super-risoluzione** rappresentano oggi un elemento chiave per garantire prestazioni elevate nei videogiochi senza sacrificare la qualità dell'immagine. L'uso crescente del Ray Tracing ha infatti reso necessario lo sviluppo di soluzioni in grado di compensare il pesante carico computazionale richiesto da questa tecnologia. Tra le più diffuse troviamo il **Deep Learning Super Sampling** (DLSS) di NVIDIA, il **FidelityFX Super Resolution** (FSR) di AMD, il **XeSS** (Xe Super Sampling) di Intel e il **Temporal Super Resolution** (TSR) di Unreal Engine. Sebbene tutte e tre le tecnologie condividano l'obiettivo di migliorare la fluidità dei giochi attraverso il rendering a risoluzioni inferiori e la successiva ricostruzione dell'immagine, differiscono significativamente nell'approccio e nei risultati ottenuti.

DLSS (Deep Learning Super Sampling)

DLSS è la soluzione più avanzata dal punto di vista tecnologico, prevedendo come punti di forza l'uso dei Tensor Cores, un'alta qualità delle immagini dopo la ricostruzione, il Frame Generation e una migliore gestione del movimento tramite motion vectors.

Nonostante le sue qualità, DLSS presenta alcune limitazioni. Innanzitutto, è disponibile esclusivamente sulle GPU NVIDIA RTX, poiché il suo funzionamento si basa su hardware dedicato. Inoltre, la sua integrazione nei videogiochi dipende direttamente dagli sviluppatori, il che significa che non tutti i titoli lo supportano. Questo aspetto, sebbene migliorato con il tempo, rimane un fattore critico rispetto ad alternative più flessibili.

FSR (FidelityFX Super Resolution)

A differenza di DLSS, FSR di AMD adotta un approccio completamente differente. Essendo una tecnologia **open-source** e basata su algoritmi convenzionali, FSR non richiede hardware specializzato, rendendolo compatibile con un'ampia gamma di dispositivi, incluse schede NVIDIA e Intel. Questa universalità lo rende una scelta attraente per gli sviluppatori che desiderano garantire il supporto a più piattaforme senza vincoli legati a uno specifico produttore di GPU.

Dal punto di vista qualitativo, FSR offre buoni risultati, ma si colloca generalmente un gradino sotto DLSS, specialmente nella capacità di ricostruzione dei dettagli più fini.

L'assenza di una rete neurale avanzata lo rende più suscettibile a fenomeni di ghosting e shimmering, particolarmente evidenti a risoluzioni più basse. Inoltre, a differenza di DLSS, non dispone di un sistema di generazione dei frame, il che lo rende meno efficace per migliorare il framerate in scenari in cui la GPU è già al limite delle sue capacità.

Se da un lato FSR offre una maggiore accessibilità e facilità di implementazione, dall'altro non riesce a raggiungere il livello di dettaglio e stabilità che caratterizza DLSS. Per questo motivo, mentre DLSS è spesso preferito nei titoli di fascia alta con grafica avanzata, FSR rappresenta un'ottima opzione per giochi che devono girare su un'ampia gamma di hardware.

Intel XeSS (Xe Super Sampling)

Intel XeSS rappresenta l'alternativa sviluppata da Intel per competere con DLSS e FSR. A differenza di FSR, che utilizza un approccio più tradizionale basato su algoritmi di upscaling spaziale, e di DLSS, che sfrutta i Tensor Cores per l'inferenza di reti neurali, XeSS combina elementi di entrambi i mondi, impiegando machine learning per migliorare la qualità dell'immagine, senza richiedere hardware proprietario per il suo funzionamento.

Uno degli aspetti più interessanti di XeSS è la sua compatibilità con un'ampia gamma di GPU. Pur essendo ottimizzato per le GPU Intel Arc, dove può sfruttare gli XMX (Xe Matrix Extensions) per accelerare l'inferenza, XeSS supporta anche GPU di NVIDIA e AMD, utilizzando invece istruzioni DP4a per il calcolo della super-risoluzione.

Dal punto di vista qualitativo, XeSS offre una ricostruzione dell'immagine superiore a FSR nelle situazioni in cui i dettagli fini sono importanti, grazie all'uso dell'intelligenza artificiale per prevedere i dati mancanti tra i frame. Tuttavia, rispetto a DLSS, soffre ancora di alcune limitazioni nella gestione della stabilità dell'immagine e nella riduzione di artefatti visivi come il ghosting e il flickering. Inoltre, non supporta il Frame Generation.

Un altro vantaggio di XeSS è la sua natura open-source, che consente agli sviluppatori di integrarlo più facilmente nei motori di gioco. Sebbene non sia ancora diffuso quanto DLSS o FSR, il suo supporto multipiattaforma e l'adozione sempre più ampia nei titoli recenti indicano che potrebbe diventare una delle principali tecnologie di super-risoluzione nei prossimi anni.

TSR (Temporal Super Resolution) di Unreal Engine

TSR (vedi 4.2.1), sviluppato da Epic Games e integrato nativamente in Unreal Engine 5, rappresenta un'alternativa solida per i giochi che utilizzano questo motore. Questa tecnologia sfrutta un sistema di **accumulo temporale** per migliorare la qualità visiva, basandosi sui dati dei frame precedenti per ridurre il flickering e migliorare la stabilità dell'immagine. Rispetto a FSR, TSR offre generalmente una qualità superiore, grazie a una gestione più efficace delle informazioni temporali e al miglioramento della coerenza tra i frame ma non supporta il Frame Generation, rendendolo meno efficace rispetto a DLSS nell'aumentare il framerate. Inoltre, rispetto a FSR, TSR richiede maggiori risorse computazionali, il che potrebbe limitarne l'utilizzo su hardware meno potente.

La scelta della tecnologia di super-risoluzione dipende da diversi fattori, tra cui l'hardware disponibile e il motore di gioco utilizzato, con DLSS che resta la soluzione più avanzata dal punto di vista tecnico.

5.3 L'integrazione delle tecnologie RTX e DLSS nei motori di gioco: impatti e considerazioni

5.3.1 Supporto al Ray Tracing RTX nei motori di gioco

L'implementazione del ray tracing RTX varia significativamente nei diversi motori di gioco, a seconda dell'architettura grafica e degli strumenti di rendering offerti. Alcuni motori come Unreal Engine offrono un supporto nativo e avanzato, mentre altri, come Unity, adottano un approccio più modulare attraverso pipeline dedicate. CryEngine, invece, integra tecnologie alternative al ray tracing tradizionale, combinando metodi ibridi per bilanciare qualità visiva e prestazioni.

Figura 5.3: Cyberpunk renderizzato con e senza ray tracing su scheda grafica GeForce RTX 3080.

Unreal Engine: Il più avanzato per RTX

Unreal Engine è uno dei motori con il miglior supporto per il ray tracing in tempo reale, grazie all'integrazione diretta con DirectX Raytracing (DXR) e RTX di NVIDIA. Il motore permette agli sviluppatori di abilitare il ray tracing con pochi passaggi direttamente nell'editor e offre una suite completa di strumenti per la gestione di ombre, riflessi, illuminazione globale e trasparenze basati su RTX.

UE5 utilizza una combinazione di tecnologie avanzate:

- **Lumen**: Sistema di illuminazione globale dinamico che utilizza un'ibridazione tra ray tracing screen-space e ray tracing RTX per garantire un'illuminazione realistica in tempo reale senza necessità di lightmaps precalcolate (Vedere 4.2.1).
- **Ray-Traced Shadows**: Implementazione diretta delle ombre basata su RTX, con supporto per soft shadows e contatto fisico tra ombre e superfici.
- **Ray-Traced Reflections**: Riflessi accurati per superfici speculari e materiali riflettenti, superando i limiti delle reflection probes e delle screen-space reflections.

- **Ray-Traced Global Illumination** (RTGI): Simula la propagazione realistica della luce indiretta, eliminando la necessità di lightmaps pre-renderizzate.
- **Hybrid Ray Tracing**: UE5 combina ray tracing e tecniche raster-based per massimizzare le prestazioni, attivando il ray tracing solo sugli oggetti che ne traggono il massimo beneficio.

Esempi di giochi con RTX in Unreal Engine sono *Cyberpunk 2077* [5.3] e *Fortnite RTX Update*.

Unity: Un approccio modulare con HDRP

Unity supporta il ray tracing RTX tramite l'High Definition Render Pipeline (vedere 4.2.1), che permette l'integrazione di effetti ray-traced su materiali, luci e ombre. A differenza di Unreal Engine, Unity richiede l'attivazione manuale del ray tracing, ed è disponibile solo nei progetti HDRP.

HDRP implementa diversi effetti di ray tracing, già presenti in altri motori, tra cui:

- **Physically Based Rendering (PBR)** per una resa fisicamente accurata dei materiali.
- **Path-Traced Rendering Mode**, una modalità di rendering fisicamente corretto utilizzata per cinematiche e scene statiche.
- **Ray-Traced Global Illumination** e **Ray-Traced Ambient Occlusion**, per una gestione avanzata della luce e delle ombre.

L'integrazione del ray tracing avviene tramite il **Package Manager**, consentendo agli sviluppatori di personalizzare il rendering in base alle necessità del progetto. Tra le demo che hanno mostrato le potenzialità di RTX in Unity troviamo *The Heretic* e *Neon Noir*.

CryEngine: Un approccio ibrido con SVOGI

CryEngine è noto per il suo motore di rendering ottimizzato, che combina diverse tecniche per ottenere un bilanciamento tra qualità visiva e prestazioni. Il motore integra il supporto a RTX, ma utilizza anche tecnologie alternative come il **Sparse Voxel Octree Global Illumination** (SVOGI), che consente di ottenere illuminazione globale in tempo reale con un impatto minore sulle prestazioni rispetto al full ray tracing (Vedere 4.2.1).

Le funzionalità RTX in CryEngine comprendono:

- **Hybrid Ray Tracing**, che combina rasterizzazione e ray tracing per ottimizzare l'efficienza del rendering.
- **Deferred Lighting Pipeline**, che migliora la gestione delle luci dinamiche con tecniche avanzate di illuminazione differita.

Le impostazioni avanzate del ray tracing in CryEngine possono essere regolate tramite la **CVar Console**, offrendo controllo dettagliato sulle prestazioni. Alcuni esempi di applicazione di RTX in CryEngine si trovano in giochi come *Enlisted* e nella tech demo *Neon Noir*.

5.3.2 Implementazione del DLSS nei motori di gioco

L'integrazione del DLSS nei motori di gioco rappresenta una soluzione avanzata per migliorare le prestazioni grafiche mantenendo un'elevata qualità visiva. I principali motori di gioco, come Unreal Engine, Unity e CryEngine, offrono diversi approcci per sfruttare questa tecnologia, garantendo flessibilità agli sviluppatori in base alle esigenze dei loro progetti.

Unreal Engine semplifica notevolmente l'implementazione del DLSS grazie a un plugin ufficiale di NVIDIA, disponibile direttamente all'interno dell'editor. Attivarlo è un processo intuitivo che richiede solo pochi passaggi: una volta abilitato il plugin dalle impostazioni, gli sviluppatori possono selezionare il livello di qualità desiderato, scegliendo tra modalità ottimizzate per la fedeltà visiva o per il massimo incremento delle prestazioni. Questa integrazione nativa permette di sfruttare appieno la potenza delle GPU NVIDIA RTX, combinando DLSS con altre tecnologie avanzate di rendering come Lumen e Nanite, per ottenere immagini di alta qualità senza compromettere il frame rate.

Unity, invece, adotta un approccio più modulare, offrendo il supporto a DLSS attraverso il suo High Definition Render Pipeline. Per implementarlo, gli sviluppatori devono scaricare il pacchetto NVIDIA DLSS tramite il Package Manager e configurarlo nelle impostazioni del progetto. Sebbene il processo richieda un'installazione manuale, Unity garantisce un'elevata personalizzazione, permettendo di adattare l'upscaling in base alle specifiche esigenze del gioco. Questo rende Unity una scelta ideale per progetti che devono bilanciare performance e qualità visiva su una vasta gamma di piattaforme.

CryEngine, noto per il suo rendering di altissima qualità, supporta DLSS ma in modo meno diretto rispetto agli altri motori. L'integrazione di questa tecnologia richiede un intervento più manuale da parte degli sviluppatori, che devono adattare le impostazioni di rendering e modificare gli shader per ottimizzare il processo di upscaling. Nonostante ciò, la flessibilità offerta da CryEngine consente di ottenere risultati eccellenti, specialmente in giochi che puntano al fotorealismo e alla gestione avanzata delle illuminazioni e delle ombre.

In definitiva, mentre Unreal Engine e Unity offrono metodi più diretti e user-friendly per integrare DLSS, CryEngine lascia maggiore libertà agli sviluppatori esperti che desiderano un controllo approfondito sull'ottimizzazione grafica. In ogni caso, la continua evoluzione del DLSS lo rende uno strumento imprescindibile per garantire esperienze di gioco fluide e visivamente accattivanti, riducendo il carico computazionale e permettendo di ottenere prestazioni elevate anche su hardware meno potente.

5.3.3 Considerazioni sull'adozione di RTX e DLSS nei motori di gioco

Nonostante i vantaggi offerti dalle tecnologie RTX e DLSS, la loro adozione nei motori di gioco non è ancora del tutto uniforme. Uno degli ostacoli principali è rappresentato dal costo computazionale del Ray Tracing, che seppur mitigato dal DLSS, può ancora incidere pesantemente sulle prestazioni. Questo porta molti sviluppatori a valutare attentamente se il beneficio visivo giustifichi l'impiego di queste tecnologie, specialmente in generi di

gioco che non fanno un uso intensivo di effetti di illuminazione avanzata, come i giochi strategici o gestionali.

Un altro fattore da considerare è la compatibilità hardware. Mentre DLSS è disponibile esclusivamente sulle schede NVIDIA RTX, alternative come AMD FSR e Intel XeSS offrono soluzioni simili, seppur con approcci differenti. Questa frammentazione nel mercato costringe gli sviluppatori a supportare più tecnologie contemporaneamente, aumentando la complessità del processo di ottimizzazione e sviluppo.

Infine, l'integrazione di RTX e DLSS dipende fortemente dal motore di gioco utilizzato. Unreal Engine offre un'integrazione nativa che semplifica notevolmente il processo, mentre altri motori, come CryEngine, richiedono personalizzazioni più avanzate per sfruttare appieno queste tecnologie. Questo può rappresentare una barriera per i team di sviluppo più piccoli, che potrebbero optare per soluzioni alternative meno complesse da implementare.

5.3.4 Il futuro del rendering nei videogiochi

L'evoluzione del Ray Tracing e delle tecnologie di upscaling basate sull'intelligenza artificiale sta delineando un futuro in cui il rendering nei videogiochi sarà sempre meno dipendente dalla pura potenza hardware. NVIDIA ha introdotto il concetto di Ray Reconstruction con DLSS 3.5, che mira a migliorare ulteriormente la qualità delle scene ray traced senza aumentare il carico computazionale. Questo suggerisce che il Ray Tracing potrebbe diventare uno standard sempre più diffuso nei giochi futuri.

Parallelamente, tecnologie come AMD FSR e Intel XeSS stanno dimostrando che l'upscaling avanzato non è più un'esclusiva delle schede NVIDIA, ma un trend generale nel settore. L'adozione crescente di queste tecniche consentirà di ottenere grafica avanzata anche su hardware meno potente, rendendo le esperienze di gioco di alta qualità accessibili a un pubblico più ampio.

Un altro aspetto chiave è l'integrazione di queste tecnologie sulle console di ultima generazione. Sebbene Xbox Series X e PlayStation 5 supportino il Ray Tracing, il suo utilizzo nei giochi è ancora limitato. Tuttavia, con il miglioramento delle tecniche di upscaling, le console potrebbero adottare più ampiamente queste tecnologie, permettendo una qualità grafica elevata senza compromessi sulle prestazioni.

L'introduzione di RTX e DLSS ha ridefinito il rendering nei giochi moderni, offrendo un compromesso tra qualità visiva e prestazioni. L'adozione di entrambi dipende da diversi fattori, tra cui il tipo di gioco, il motore di sviluppo utilizzato e il target hardware.

Mentre motori come Unreal Engine facilitano l'implementazione di queste tecnologie, altri richiedono maggiore personalizzazione, rendendone l'uso meno immediato.

Guardando al futuro, il rendering in tempo reale continuerà a evolversi grazie all'integrazione di intelligenza artificiale e tecniche di upscaling sempre più sofisticate. Questo permetterà di ottenere esperienze visive di alta qualità su un numero sempre maggiore di dispositivi, riducendo la dipendenza dalla pura potenza computazionale dell'hardware. RTX e DLSS rappresentano dunque un passo fondamentale verso un futuro in cui la qualità grafica nei videogiochi sarà sempre meno vincolata alle limitazioni tecniche tradizionali.

Capitolo 6

Conclusioni

In questa tesi sono stati analizzati tredici motori di gioco, confrontandone le caratteristiche, le prestazioni e l'idoneità per differenti tipologie di sviluppo. Dopo una panoramica sulla loro struttura, sono state esaminate le loro componenti fondamentali, come il rendering engine, la fisica, il sistema di animazione, l'audio e l'intelligenza artificiale. Successivamente, si è proceduto a un confronto approfondito tra i motori più diffusi, valutandone i punti di forza e le limitazioni in termini di costi, accessibilità, modularità e supporto alle tecnologie emergenti. Infine, sono state analizzate le innovazioni recenti, come il Ray Tracing e il DLSS, che stanno ridefinendo gli standard grafici nei videogiochi.

Dall'analisi comparativa, sono emerse alcune tendenze significative nel panorama dei motori di gioco:

- Produzioni AAA vs Indie: Unreal Engine e CryEngine si confermano le scelte migliori per produzioni AAA grazie alle loro capacità avanzate di rendering e simulazione fisica. Unity e Godot, invece, si distinguono per la loro versatilità, risultando particolarmente adatti agli sviluppatori indipendenti.

- Modularità e personalizzazione: Open 3D Engine (O3DE) e Stride emergono come soluzioni altamente modulari, mentre Unity e Unreal Engine vantano un vasto ecosistema di plugin e asset che ne estendono le funzionalità.

- Facilità d'uso: GameMaker, GDevelop e Construct3 si rivelano strumenti ideali per chi si avvicina al game development senza una forte base di programmazione, grazie ai loro strumenti di sviluppo visuale.

- Supporto alle tecnologie emergenti: Il supporto al Ray Tracing e al DLSS è presente principalmente in Unreal Engine, Unity e CryEngine, mentre altri motori come Godot e Phaser si concentrano maggiormente su efficienza e ottimizzazione.

La scelta di un motore di gioco dipende da molteplici fattori, tra cui il budget, il livello di esperienza dello sviluppatore, il target di riferimento e le esigenze specifiche del progetto. L'analisi condotta in questa tesi dimostra che non esiste un motore perfetto per ogni situazione, ma piuttosto una serie di strumenti con punti di forza distinti che possono essere sfruttati in base alle necessità dello sviluppo.

L'industria dei videogiochi è in continua evoluzione, e i motori di gioco continueranno a integrarsi con nuove tecnologie per offrire strumenti sempre più avanzati e accessibili. Comprendere le potenzialità di ciascun motore è essenziale per fare scelte informate e per sfruttare al meglio le risorse disponibili, contribuendo così all'innovazione nel settore videoludico.

Bibliografia e Sitografia

[1] A. Gazis and E. Katsiri, "Serious games in digital gaming: A comprehensive review of applications, game engines and advancements," *WSEAS Transactions on Computer Research*, vol. 11, no. 2, pp. 1–22, 2023. [Online]. Available: https://doi.org/10.37394/232018.2023.11.2

[2] J. Gregory, *Game Engine Architecture*, 3rd ed. CRC Press, 2018.

[3] Wikipedia contributors. (2025) Motore grafico. [Online]. Available: https://it.wikipedia.org/wiki/Motore_grafico

[4] E. Games, "Nanite virtualized geometry in unreal engine - documentation," 2025. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine

[5] ——, "What is nanite? - unreal engine," 2025. [Online]. Available: https://dev.epicgames.com/community/learning/tutorials/LK7/unreal-engine-what-is-nanite

[6] U. University, "Nanite: What you need to know - unreal engine," 2025. [Online]. Available: https://www.unreal-university.blog/nanite-what-you-need-to-know-unreal-engine/

[7] E. Games, "Lumen global illumination and reflections in unreal engine," 2025. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-global-illumination-and-reflections-in-unreal-engine

[8] ——, "Unreal engine 5 goes all-in on dynamic global illumination with lumen," 2025. [Online]. Available: https://www.unrealengine.com/en-US/tech-blog/unreal-engine-5-goes-all-in-on-dynamic-global-illumination-with-lumen

[9] ——, "What is real-time ray tracing? - unreal engine," 2025. [Online]. Available: https://www.unrealengine.com/en-US/explainers/ray-tracing/what-is-real-time-ray-tracing

[10] NVIDIA, "Introducing ray tracing in unreal engine 4," 2025. [Online]. Available: https://developer.nvidia.com/blog/introducing-ray-tracing-in-unreal-engine-4/

[11] E. Games, "Temporal super resolution." [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/temporal-super-resolution-in-unreal-engine

[12] ——, "Materials - unreal engine." [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-materials

[13] U. Technologies, "Render pipelines." [Online]. Available: https://docs.unity3d.com/Manual/render-pipelines.html

[14] G. E. Documentation, *Introduzione a Godot*, 2025. [Online]. Available: https://docs.godotengine.org/it/4.x/getting_started/introduction/introduction_to_godot.html

[15] GamesIndustry.biz, "What is the best game engine: is cryengine right for you?" 2020. [Online]. Available: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-cryengine-the-right-game-engine-for-you

[16] A. Reviews, "Cryengine: recensione software creazione realtà virtuale," 2025. [Online]. Available: https://www.accuratereviews.com/it/software-per-creare-realta-virtuale/cryengine-recensione/

[17] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[18] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *Computer Graphics*, vol. 18, no. 3, pp. 137–145, 1984.

[19] J. T. Kajiya, "The rendering equation," *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4, pp. 143–150, 1986.