RELAZIONE SULL'ELABORATO DEL CORSO DI COMPUTER GRAPHICS AND 3D 2019-2020

Mandelli Lorenzo
matricola 7045200

8 settembre 2020

Sommario

L'obbiettivo dell'elaborato è quello di realizzare un videogioco, mediante il motore grafico Unity, che sia compatibile con l'utilizzo di piattaforme a realtà virtuale quale il dispositivo Oculus Rift. A questo scopo oltre alla modellazione dell'ambiente di gioco e alle sue dinamiche sono stati realizzati degli agenti, che operano da avversari per il giocatore, secondo diversi schemi di progettazione.

1 Introduzione

Il progetto si basa su un sistema di gioco incentrato su *tattical sword play* ovvero combattimenti all'arma bianca in realtà virtuale. Il giocatore utilizzando un controller come spada e l'altro come catalizzatore di magie si trova ad affrontare ondate di nemici dei quali dovrà bloccare gli attacchi e contrattaccare al momento giusto per poterli sconfiggere.
Il fine ultimo del progetto era quello di riprodurre le dinamiche di un combattimento *melee* attraverso il dispositivo Oculus Rift in modo che risulti il più realistico possibile. Il raggiungimento di questo obbiettivo si è scontrato con le limitazioni fisiche che un sistema a realtà virtuale prevede (il limitato campo visivo, la mancanza di un reale feedback sulle collisioni, etc) ed è stato necessario ricorrere a soluzioni che si astraessero dallo scopo principale per favorire l'immersività dell'esperienza. Queste dinamiche su cui si basa il videogioco sono riportate qui in seguito.

2 Sistema di gioco

Il sistema di gioco è schematizzabile nei sistemi di combattimento, di status dei personaggi e di movimento del giocatore.

Figura 1: Esempio di combattimento a distanza ravvicinata in gioco.

2.1 Sistema di combattimento

Come accennato in precedenza il giocatore ha la possibilità di utilizzare un'arma a distanza ravvicinata, quale una spada, e ricorrere a delle magie. In seguito vengono trattate le rispettive dinamiche.

2.1.1 Combattimento ravvicinato

Le meccaniche per la gestione dei combattimenti all'arma bianca sono state riprese dall'articolo *"Sword Mechanics for VR"* [1] in cui l'autore mostra come i limiti fisici dei dispositivi per la realtà virtuale alterino i movimenti eseguiti realmente da quelli che si realizzano invece nell'ambiente di gioco.
In particolare egli evidenzia come, in un videogioco VR basato su combattimenti a distanza ravvicinata, la mancanza di un reale feedback sulla spada non consenta una convincente interazione tra il controller e i nemici: il controller-spada non incontrerà mai alcuna resistenza nella realtà, mentre nell'ambiente simulato andrà a scontrarsi fisicamente con diversi oggetti. Questo fenomeno porta all'alterazione sia dell'impressione del giocatore, che si aspetta un contraccolpo dove invece non avviene, sia delle dinamiche su cui il gioco si basa. Avendo infatti nella simulazione un'arma di peso nullo e di forza infinita, in quanto non può subire resistenze, il giocatore intuitivamente tenderà a muoverla in modo non realistico andando a compromettere in un certo modo la voluta esperienza di gioco.
Nell'articolo si introducono alcune soluzioni per rendere più veritiero il controllo di un oggetto e le sue interazioni con l'ambiente circostante.

L'idea proposta si basa su collegare nell'ambiente di gioco tra il controller e l'oggetto da maneggiare un giunto elastico, realizzabile in Unity attraverso il componente *Configurable Joint*, che simuli un "polso aggiuntivo" caratterizzato da una forza di torsione che ne mantiene l'orientazione e una forza lineare che lo

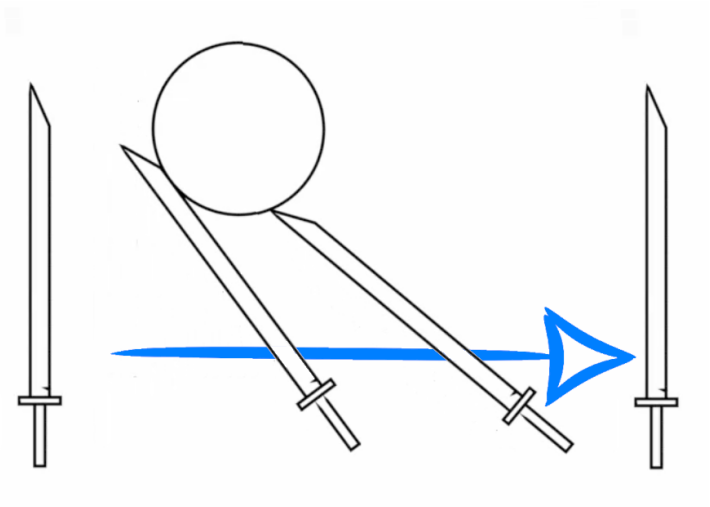


Figura 2: Dinamica di una spada in presenza del giunto elastico: allo spostarsi a destra dell'arma questa impatta con un oggetto di forma sferica inamovibile che ne causa la deflessione per poi tornare nella posizione originaria una volta superato l'ostacolo. Disegno ripreso da [1].

mantenga nella posizione di presa. In questo modo la spada attaccata al giunto potrà interagire con l'ambiente e andrà a esercitare una forza tanto più grande tanto più risulterà deviata dalla sua posizione nominale. Inoltre le impostazioni utilizzate per i valori di forza del giunto rendono la resistenza alla torsione molto superiore a quella lineare facendo in modo che la deflessione angolata di un'arma avversaria risulti più facile rispetto a una deflessione tra oggetti perpendicolari (fenomeno che avviene anche nella realtà). Infine questa soluzione permette di simulare in certo modo la forza che sulle armi viene applicata: se si esercita pressione da un punto vicino all'elsa della propria spada sulla punta della spada avversaria questo movimento risulterà molto semplice a causa della leva favorevole effettuata. Nella situazione opposta invece il giocatore avrà l'impressione di dover applicare un quantitativo maggiore di forza per vincere lo scontro.

Riassumendo il metodo proposto permette che la spada:

1. Non possa attraversare oggetti impenetrabili.
2. Possa fermare altre spade e da esse possa essere fermata indipendentemente da come il controller si stia muovendo.
3. Vada a incoraggiare movimenti realistici quali una deflessione angolata o leve favorevoli.
4. Generi un feedback per il giocatore molto più realistico all'avvenire di collisioni e interazioni con l'ambiente di gioco.

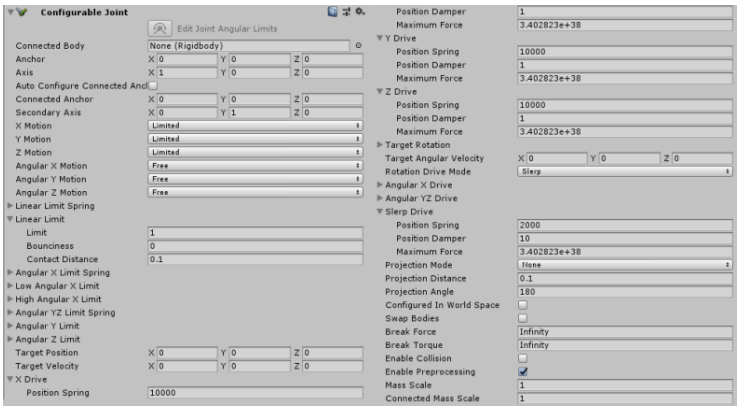


Figura 3: Impostazioni utilizzate per la realizzazione del "giunto elastico".

2.1.1.1 Implementazione

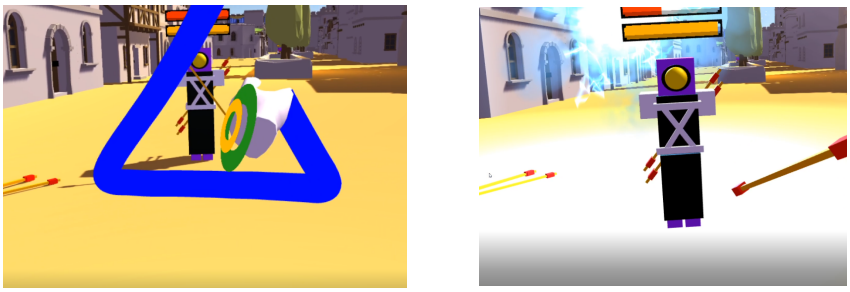
Come gia accenato la meccanica del "polso aggiuntivo" è realizzata in Unity attraverso il componente *Configurable Joint* opportunamente settato come mostrato in figura 3. A differenza di come è stato implementato nell'articolo il giunto non è realizzato al runtime ma è sempre presente negli oggetti afferrabili in gioco. Esso è caratterizzato dal componente *OVRGrabbable* che permette l'azione di *grab* ed è collegato all'oggetto desiderato.

2.1.2 Magie

Il giocatore utilizzando la mano sinistra è in grado di realizzare linee in aria premendo uno specifico tasto del controller. In base alla forma disegnata una differente magia è innescata:

- **Triangolo**: è lanciata una sfera di energia che esplode al primo contatto con un oggetto o dopo aver percorso pochi metri danneggiando ogni nemico nel suo raggio. La magia è realizzata attraverso due animazioni: la prima, rappresentante una nuvola elettrica, è traslata per il tempo in cui deve simulare il suo lancio e la seconda, rappresentante un esplosione, è eseguita non appena la prima risulta terminata. Al iniziare della seconda animazione, dal suo punto centrale, sono rilevati attraverso la funzione *"Physics.OverlapSphere"* quali nemici risultano essere nel raggio d'azione ed ad essi è decrementata la vita e applicata una forza per simulare l'impatto dell'esplosione. In caso la vita di un nemico scenda sotto il valore zero è simulata la sua morte facendo in modo che le parti del suo corpo prima unite insieme si separino e si respingano tra loro facendolo volare in tanti pezzi.

- **Quadrato**: Applica al giocatore una rigenerazione della vita. Essa è realizzata con un'animazione di una durata fissa e da una funzione che



(a) **(b)**

Figura 4: Esempio di utilizzo di una magia in gioco.

opera un incremento della vita di una certa quantita per tutto il tempo che essa rimane attiva.

Il riconoscimento di forme dalle linee realizzate è stato ottenuto attraverso il pacchetto *"OSVR Gesture Framework"* disponibile sull'*Asset Store* [2] modificando le tolleranze dei triangoli e dei quadrati e collegando le classi per la gestione degli effetti delle magie.

Al fine di non rendere la meccanica della magia troppo preponderante nell'esperienza di gioco non è possibile utilizzare due magie contemporaneamente e dopo l'utilizzo di una di esse è necessario attendere alcuni secondi per poterla eseguire nuovamente.

2.2 Sistema di status dei personaggi

Tutti i personaggi, sia il giocatore che gli agenti nemici, dispongono una barra della vita e una della stamina visibili rispettivamente sulla mano sinistra del giocatore e sopra il corpo dei nemici.
La vita è decrementata quando un arma avversaria colpisce il corpo del personaggio. La stamina invece si riduce quando, in possesso di un arma, si eseguono azioni che richiederebbero lavoro, quali accelerazione del corpo della spada, applicazione di una forza di torsione o lineare (ad esempio quando si spinge la punta della spada contro un oggetto) ed è incrementata naturalmente con passare del tempo.

2.2.1 Implementazione

Le barre di vita e stamina sono realizzate attraverso le classi *"HealthScript"* e *"StaminaScript"* comuni a tutti i personaggi. Queste classi espongono i metodi per l'aumento e la riduzione dei rispettivi valori di vita e di stamina realizzati come dei tipi *float*.
Il decremento della vita in seguito a un attacco avversario è realizzato mediante

la classe *"BodyCollisionScript"* che tenendo traccia dei *collider* presenti nel corpo decrementa la vita in base alle velocità riscontrate nelle collisioni con armi avversarie.
Il decremento della stamina invece avviene tramite la classe *"StaminaReduceAction"* che valutando lo scostamento tra la posizione nominale della spada da quella reale e l'accelerazione a cui essa è sottoposta ne riduce opportunamente il valore.

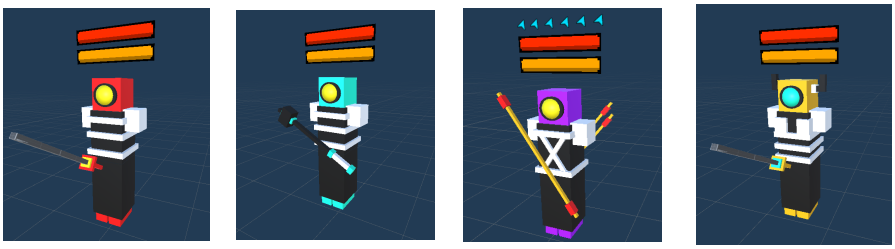
2.3 Sistema di movimento del giocatore

Per ridurre al minimo il fenomeno del *motion sickness* e per le dinamiche di gioco desiderate si è scelto un sistema di movimento basato sull'azione di presa dei controller, il *grab*, utilizzato anche dal videogioco *GORN* [3]. Il giocatore può afferrare l'ambiente che lo circonda e trascinarsi in avanti avvicinando il controller al suo corpo. Questa tipologia di movimento è comparabile al più classico sistema di arrampicata, il *climbing*, ma sul piano orizzontale.

3 Agenti

Per la realizzazione dei nemici sono state scelte due principali modalità: *agenti basati su macchine a stati* e *agenti addestrati* in grado di scelgliere quale azione eseguire in base alle osservazioni ricevute.

- **Macchine a stati**: eseguono un pattern di comportamento prestabilito: di un insieme di animazioni di attacco una viene scelta randomicamente ed eseguita. In seguito all'animazione l'agente aspetta un tempo randomico compreso fra due estremi (nell'ordine dei 1-4 secondi) per dare al giocatore possibilità di contrattaccare.
  Gli agenti basati su macchine a stati si dividono ulteriormente in:
  - **Singoli**: raggiungono il giocatore e lo attaccano singolarmente secondo il loro schema.
  - **In squadra**: sono parte di un una squadra di agenti i cui membri si coordinano per attaccare il loro obbiettivo uno alla volta. La meccania della coordinazione è stata necessariamente introdotta per motivi di *gameplay*, il limitato campo visivo previsto dal visore Oculus e la mancanza di un reale feedback sulle collisioni rendono impossibile al giocatore la gestione di due o più nemici che attaccano all'unisono.
- **Agenti addestrati**: sono stati addestrati attraverso il pacchetto *"ML-Agent"* [4] per la realizzazione di agenti dotati di intelligenza artificiale. Essi sono forniti di animazioni di attacco e di difesa che scelgono di eseguire durante il combattimento in base alle osservazioni sulla posizione della spada dell'avversario e alle sue caratteristiche.



(a) Agente 1 **(b)** Agente 2 **(c)** Agente 3 **(d)** Agente 4

Figura 5: Gli agenti ottenuti tramite macchine a stati.

3.1 Agenti realizzati attraverso macchine a stati

Di questa tipologia ne sono stati realizzati 4 campioni come mostrati in figura 5. Gli agenti 1-2-4 presentano un comportamento simile tra loro: una volta raggiunta la distanza necessaria dal bersaglio eseguono una randomica combinazione di attacchi da un insieme di animazioni possibili. Essi variano tra loro per le animazioni a disposizione e le combinazioni che riescono ad eseguire:

- L'agente 1, armato di spada, possiede 5 animazioni di attacco.
- L'agente 2, dotato di un martello da guerra, può eseguire 3 combinazioni di animazioni offensive.
- L'agente 4, armato di spada e rappresentate una versione più impegnativa dell'agente 1, dispone di un gran numero di animazioni di attacco che esegue ad elevata velocità.

L'agente 3 invece, dotato di lance, è caratterizzato dalla possibilità di attaccare a distanza e possiede un pattern di comportamento più complesso. Egli infatti, all'avvenire dell'animazione di lancio dell'arma, la scollega dal vincolo di seguire le sue mani e le imprime una forza in direzione del bersaglio affinchè non perda quota durante il tragitto. Poco prima di eseguire un successivo lancio una nuova arma è generata fino ad un massimo di 6 ricariche. Una volta terminati i colpi a distanza disponibili, visibili attraverso un contatore posto sopra la barra della vita, un'ultima lancia è generata e l'agente assume un comportamento di attacco all'arma bianca similmente agli agenti 1, 2 e 4 con un suo insieme di animazioni.

3.1.1 Coordinazione di squadra

Gli stessi agenti 1,2 e 3 sono stati modificati per permettere la coordinazioni dei loro attacchi in squadra. Essi fanno riferimento a una comune classe *"SquadManager"* caratterizzata da un intero *id* che sancisce quale agente abbia il permesso di attaccare. A ogni frame infatti gli agenti interrogano la classe se il loro identificativo coincida con tale attributo e nel caso performano l'attacco. Infine al termine di ogni animazione è chiamato un metodo che randomizza il valore *id* per permettere ad altri agenti di agire.

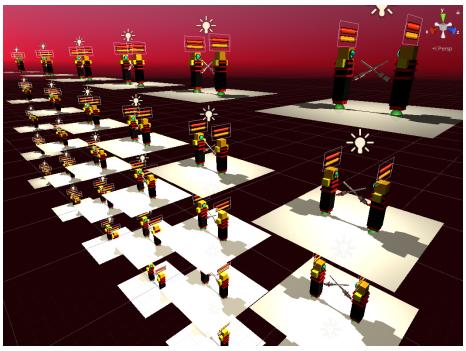


Figura 6: Fase di addestramento di agenti.

3.2 Agenti addestrati

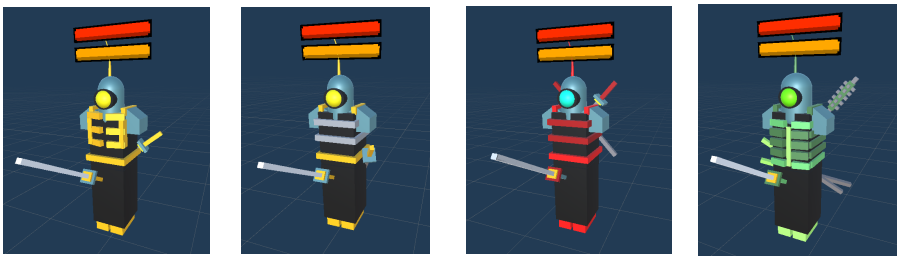
Questa tipologia di agenti è stata ottenuta utilizzando il pacchetto di Unity *"Ml-Agent"* [4] che fornisce un modo per addestrarli a compiere determinati obbiettivi.

3.2.1 Unity Machine Learning Agents

Il pacchetto *"Unity Machine Learning Agents"* (ML-Agents) è un progetto open-source che permette la simulazione e l'addestramento di agenti intelligenti mediante algoritmi di apprendimento per rinforzo, per imitazione, per neuroevoluzione e altri metodi di machine learning attraverso l'uso di chiamate a API Python.
Il pacchetto necessità che gli agenti siano dotati di due componenti: il *"Behaviour Parameters"* che specifica spazio di osservazioni disponibili e di azioni possibili e un componente che erediti da *"Agent"* che invece ne stabilisce il comportamento. In quest'ultimo infatti sono presenti i metodi per l'acquisizione delle osservazioni del mondo di gioco e per l'esecuzione delle azioni che l'agente può compiere. In caso di addestramento tramite rinforzo sono tipicamente incluse le condizioni per assegnare *reward* positivi o negativi all'agente in base al suo comportamento. Una volta terminata la fase di addestramento il modello comportamentale è salvato in memoria in un file di tipo *.NN* per poter essere realmente utilizzato dagli agenti nel videogioco.

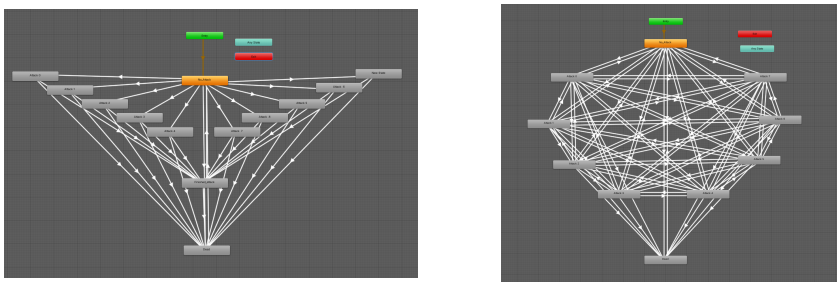
3.2.2 Agenti ottenuti

Utilizzando ML-Agents per l'apprendimento tramite rinforzo sono stati ottenuti gli agenti 5,6,7 e 8 mostrati in figura 7. Essi durante la fase di training avevano il compito di sconfiggere un avversario ottenendo punti quando riuscivano a colpirlo e perdendo punti quando da lui venivano colpiti.
Tutti gli agenti sono caratterizzati da uno spazio di osservazioni composto dalla posizione e dalla velocità del polso e della punta della spada avversaria. Gli agenti 5,6 e 7 sono in grado di scegliere un azione tra 10 animazioni possibili



(a) Agente 5 **(b)** Agente 6 **(c)** Agente 7 **(d)** Agente 8

Figura 7: Gli agenti ottenuti tramite addestramento per mezzo del pacchetto *ML-Agents*.



(a) Agenti 5,6,7 **(b)** Agente 8

Figura 8: I sistemi di animazione relativi agli agenti 5,6,7 e 8. Come si può vedere mentre nella prima figura la scelta dell'animazione unisce univocamente l'inizio e la fine dell'azione, nella seconda figura essa può essere interrotta da altre animazioni.

(5 di attacco, 4 di difesa e 1 di attesa) e differiscono tra loro per la fase di addestramento a cui sono stati sottoposti: l'agente 5 è stato allenato contro l'agente 1, l'agente 6 contro un giocatore reale e infine l'agente 7 contro un agente dello stesso modello.
L'agente 8 invece è stato realizzato secondo uno schema differente: esso possiede 8 animazioni (5 di attacco, 2 di difesa e 1 di attesa) ed è in grado di cambiare animazione durante l'esecuzione della stessa rendendolo in grado di eseguire movimenti più complessi quali finte o deviazioni improvvise.

Riferimenti bibliografici

[1] Evan Fletcher. Sword mechanics for vr. https://evanfletcher42.com/2018/12/29/sword-mechanics-for-vr/.

[2] OSVR Gesture Framework. https://assetstore.unity.com/packages/tools/input-management/osvr-gesture-framework-144686.

[3] GORN. https://store.steampowered.com/app/578620/GORN/.

[4] Ml-Agent. https://github.com/Unity-Technologies/ml-agents.