

# Uno strumento VR basato su Unity per la proposta di itinerari personalizzati

**Relatore:** Prof. Cabitza Federico  
**Co-relatore:** Dott. Sartori Fabio

Relazione della prova finale di:  
Federico Scaramelli  
Matricola 816598

Anno Accademico 2018-2019

*Un ringraziamento speciale va alla mia famiglia, alla mia ragazza, e ai miei amici più cari: è grazie al loro sostegno e incoraggiamento se oggi sono riuscito a raggiungere questo importante traguardo.*

*Dedico i miei sacrifici a mio padre, che sarebbe fiero dei risultati da me ottenuti.*

*Ringrazio inoltre i professori Fabio Sartori e Federico Cabitza, co-relatore e relatore di questa tesi, per avermi dato la possibilità di lavorare a questo progetto molto stimolante sotto l'aspetto tecnico.*

*Grazie di cuore,  
Federico*

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Sedentarietà . . . . .	1
1.1.1	Effetti . . . . .	1
1.1.2	Rischi per la salute . . . . .	1
1.1.3	Statistiche . . . . .	2
1.1.4	Soluzioni al problema . . . . .	2
1.2	MoveUp! . . . . .	4
1.3	Virtual Reality terapeutica . . . . .	5
1.4	MoveUp! VR Edition . . . . .	5
<b>2</b>	<b>Analisi del problema</b>	<b>6</b>
2.1	Problemi principali . . . . .	6
2.1.1	Lo spazio tridimensionale . . . . .	6
2.1.2	La Virtual Reality . . . . .	6
2.1.3	Le mappe . . . . .	8
2.1.4	La visualizzazione panoramica . . . . .	8
2.2	Tecnologie abilitanti . . . . .	8
2.2.1	Android Studio . . . . .	8
2.2.2	Motori grafici tridimensionali dotati di IDE . . . . .	9
2.2.3	GoogleVR Cardboard SDK per Unity . . . . .	11
2.2.4	Mappe integrabili in Unity . . . . .	11
2.3	Funzionalità . . . . .	14
2.3.1	Schermata iniziale . . . . .	14
2.3.2	Street View . . . . .	15
2.3.3	Simulazione . . . . .	17
<b>3</b>	<b>Progettazione e Implementazione</b>	<b>18</b>
3.1	Finite State Machine . . . . .	18
3.1.1	Implementazione della macchina a stati . . . . .	18
3.2	Scripts di calcolo . . . . .	20
3.2.1	Geocoder.cs . . . . .	21
3.2.2	DestinationCalc.cs . . . . .	21
3.2.3	PathTileProvider.cs . . . . .	22
3.2.4	DirectionsCalc.cs . . . . .	23
3.2.5	StreetViewGenerator.cs . . . . .	23
3.2.6	VRManager.cs . . . . .	25
3.3	Scripts per la UI . . . . .	26
3.3.1	SaveStartLocationInput.cs . . . . .	26

3.3.2	SaveInputTime.cs & SaveInputSpeed.cs . . . . .	27
3.3.3	LoadingScreenPanel.cs . . . . .	27
3.3.4	WearVisorPanel.cs . . . . .	27
3.3.5	Scripts dei bottoni . . . . .	27
3.4	Scripts di interazione con la mappa . . . . .	27
3.4.1	CameraHandler.cs e QuadTreeCamMov.cs . . . . .	27
3.4.2	SpawnOnMap.cs . . . . .	29
3.5	Script del player . . . . .	29
3.6	Struttura e funzionamento . . . . .	30
3.6.1	NotStartedState . . . . .	31
3.6.2	StreetViewState . . . . .	31
3.6.3	WalkingState . . . . .	32
3.6.4	Diagrammi UML . . . . .	33
<b>4</b>	<b>Caso di studio</b>	<b>37</b>
4.1	Ambienti urbani . . . . .	37
4.2	Analisi delle performance . . . . .	37
4.2.1	Il Profiler di Unity . . . . .	38
4.2.2	Risultati ottenuti . . . . .	38
4.2.3	Adattamento applicato . . . . .	41
4.2.4	Altri adattamenti . . . . .	42
<b>5</b>	<b>Conclusioni</b>	<b>45</b>
5.1	Risultati . . . . .	45
5.2	Sviluppi futuri . . . . .	45
5.3	Sviluppi alternativi . . . . .	47
<b>Riferimenti bibliografici</b>		<b>49</b>

# Capitolo 1

## Introduzione

In questo capitolo introdurremo il problema che ha portato all’idea dell’applicazione discussa in questa tesi: la *sedentarietà*. Ne vedremo gli effetti sul corpo, i rischi per la salute, le statistiche a riguardo ed alcune possibili soluzioni.

Entreremo poi più nel dettaglio per quanto riguarda le applicazioni sviluppate e discusse in questa tesi, descrivendone l’obiettivo e la loro utilità nel supporto di individui affetti da sedentarietà.

### 1.1 Sedentarietà

In medicina, *stile di vita sedentario* è un termine usato per descrivere uno stile di vita con mancanza di moto ed esercizio fisico, caratterizzato dallo stare seduti per buona parte della giornata.

#### 1.1.1 Effetti

Molte delle malattie del mondo moderno sono direttamente o indirettamente favorite dal ridotto livello di attività fisica tipico di oggi: ipertensione, infarto, ictus, tumori, obesità, diabete e perfino malattie neurodegenerative come Alzheimer e Parkinson sono favorite da uno stile di vita sedentario e contrastate da un regolare piano di attività fisica. Con una cronica sedentarietà il corpo umano avvia un processo di regressione motoria, una progressiva perdita di capacità funzionali che non sono solo a carico dei muscoli, ma anche di cuore, polmoni, vasi sanguigni, nervi e produzione di ormoni e neurotrasmettitori.

Il movimento fisico regolare ristabilisce, tra le altre cose, la corretta produzione di neurotrasmettitori, le sostanze responsabili del nostro benessere psico-emotivo, con diversi effetti sull’organismo. Permette di ristabilire un corretto tono dell’umore e di avere più entusiasmo nei confronti della vita, merito di un giusto equilibrio ormonale.

#### 1.1.2 Rischi per la salute

La sedentarietà è una delle scelte più nocive che si possano prendere per il corpo e per la psiche. Da anni vengono evidenziati gli effetti negativi sulla salute di uno stile di vita sedentario, e il rischio aumenta per una lunga serie di malattie anche gravi: sovrappeso e obesità, diabete e patologie metaboliche, ipertensione e patologie

cardiovascolari come l'infarto, ictus, osteoporosi, artrosi, sarcopenia e sindrome da fragilità dell'anziano, tumori, malattie neurodegenerative, ansia e depressione.

Un'attività fisica adeguata e regolare è un potente fattore preventivo oltre che un'attività capace di rendere la vita più divertente e intensa. [1]

### 1.1.3 Statistiche

Nel mondo sono 1.4 miliardi le persone che non fanno abbastanza attività fisica, mettendo a rischio la loro salute. Le donne sono meno attive rispetto agli uomini e i paesi più ricchi sono quelli in cui la sedentarietà è maggiore. Se la tendenza continuerà nella direzione di crescita / non decrescita di questo valore, l'obiettivo di raggiungere entro il 2025 una riduzione del 10% dell'inattività fisica globale non sarà raggiunto. [2]

I dati riguardanti l'**Italia** ci dicono che il 41.4% della popolazione adulta italiana non fa abbastanza attività fisica. Non siamo tra i paesi con il dato peggiore, ma, come commenta *Fabio Pigozzi*, Rettore e Professore Ordinario di Medicina Interna e specialista in medicina dello sport, Università degli Studi di Roma Foro Italico,

*Gli italiani sono ancora poco educati sull'importanza dell'attività fisica. Molto si sta facendo ma è fondamentale cercare di stimolare il movimento in forma organizzata già da bambini e partire dalla scuola dove purtroppo l'educazione fisica non è sempre offerta in modo adeguato.*

### 1.1.4 Soluzioni al problema

Nel 2010 l'OMS ha pubblicato *Global recommendations on Physical activity for Health* [3], in cui definisce i livelli di attività fisica raccomandata per la salute per tre gruppi d'età:

- **per bambini e ragazzi (5-17 anni)**: almeno 60 minuti al giorno di attività moderata-vigorosa, includendo almeno 3 volte a settimana esercizi per la forza che possono consistere in giochi di movimento o attività sportive;
- **per adulti (18-64 anni)**: almeno 150 minuti alla settimana di attività moderata o 75 di attività vigorosa (o combinazioni equivalenti delle due) in sessioni di almeno 10 minuti per volta, con rafforzamento dei maggiori gruppi muscolari da svolgere almeno 2 volte a settimana;
- **per anziani (dai 65 anni)**: le indicazioni sono le stesse degli adulti, con l'avvertenza di svolgere anche attività orientate all'equilibrio per prevenire le cadute. Chi fosse impossibilitato a seguire in pieno le raccomandazioni deve fare attività fisica almeno 3 volte a settimana e adottare uno stile di vita attivo adeguato alle proprie condizioni;

I livelli raccomandati vanno intesi come un **limite minimo**.

Sempre la OMS, ha introdotto la necessità di adottare politiche internazionali che incentivassero l'attività fisica, tramite il *Piano d'azione globale dell'OMS sull'attività fisica (2018-2030)*, che prende il nome di "*Più persone attive per un mondo più sano*". Lanciato nel Giugno 2018, il Piano raccomanda una serie di 20 azioni politiche con l'obiettivo di creare una società più attiva migliorando gli spazi e

i luoghi per l'attività fisica e aumentando i programmi e le opportunità per persone di tutte le età e condizioni fisiche.

Il vero obiettivo non è quindi convincere le persone a testare la loro attività fisica per un tempo limitato, ma cercare di far diventare questo comportamento benefico un'abitudine, affinché possa prolungarsi a lungo termine.

Maria Brilaki, laureata in ingegneria a Stanford, certificata come personal trainer, autrice di due libri - tra cui *"Surprisingly... unstuck: the power of small healthy habits in a world addicted to instant results"* (*"Il potere delle piccole sane abitudini in un mondo assuefatto ai risultati immediati"*) - e fondatrice del sito *"Fitness Reloaded"* [4] - che ha aiutato un gran numero di individui a migliorare le proprie abitudini - nei suoi lavori suggerisce una serie di pensieri che si sono rivelati efficaci nell'aiutare le persone a non mollare l'esercizio fisico dopo breve tempo, alcuni di questi sono:

- non porsi come obiettivo iniziale un allenamento giornaliero di trenta minuti, che con il tempo può essere difficile da sostenere, ma partire con attività di cinque minuti per abituarsi all'idea dell'allenamento e, solo dopo che questa è ben cementificata, incrementare carico e durata di lavoro;
- non obbligarsi a svolgere esercizio fisico se non lo si vuole fare perché è controproducente, ma iniziare con attività che si trovano piacevoli e non sgridarsi quando non si riesce a soddisfare le proprie aspettative;
- non dare la colpa ad una mancanza di motivazione se non si riesce a mantenere il programma che ci si era imposti, ma trovare quali scuse si usano per giustificarsi (pensare di non aver tempo, sentirsi troppo stanchi per una sessione intera) e affrontarle;
- non pensare che l'esercizio serva solo a perdere peso, ma assimilare gli innumerosi effetti benefici che esso porta;
- non pensare di poter iniziare ad allenarsi solo quando tutti gli altri aspetti della propria vita sono sistemati, ma farlo anche quando non si pensa di poter dare il 100%;
- trovare qualche attività fisica o sportiva che risulti interessante, se si pensa che esercitarsi sia noioso, anche se l'avversione nasce dall'idea di doversi impegnare per un'ora o più a sessione;
- allenarsi in casa o farlo in una palestra vicina al proprio luogo di lavoro quando costituisce un freno il doversi spostare da casa appositamente per allenarsi.

Tutti questi fattori suggeriscono che per invogliare qualcuno ad allenare il proprio corpo è più efficace fargli conoscere i benefici che otterrà per i propri sforzi rispetto a presentargli i problemi ai quali andrà incontro conducendo uno stile di vita sedentario. Un buon metodo per iniziare ad abbandonare la sedentarietà è camminare, ma molto spesso per non farlo si avanzano diverse scuse, come il non sapere dove andare, il tempo da dedicare all'attività, le zone sconosciute, la noia delle aree abitudinariamente frequentate.

L'applicazione oggetto di questa tesi è stata ideata con l'intenzione di sopperire a queste ultime problematiche e aiutare chi ha bisogno di motivazione per iniziare a muoversi.

Il più grande problema della sedentarietà deriva dal fatto che chi ne abusa spesso ha problemi ad uscirne, a causa delle abitudini che si porta dietro da anni, oppure allo stile di vita che la società impone a certi individui, con molte attività da svolgere seduti davanti ad uno schermo. Per questa ragione a volte sono necessarie terapie psicologiche reali per aiutare gli individui a trovare la forza di volontà di fare del movimento con costanza.

## 1.2 MoveUp!

In seguito ad un'idea del Dipartimento di Informatica Sistemistica e Comunicazione dell'Università di Milano - Bicocca, il mio collega *Andrea Infantino* è stato incaricato di sviluppare l'applicazione *MoveUp!*, con lo scopo di dare supporto agli individui affetti da problemi psicofisici legati alla sedentarietà. [5]

Il funzionamento di *MoveUp!* è il seguente: l'utente seleziona un **punto di partenza**, una **velocità** di corsa espressa in *km/h*, una **durata** espressa in *minuti* ed una **direzione** espressa in *gradi da 0° a 360° rispetto al Nord*; l'applicazione calcola quindi una destinazione casuale in funzione dei parametri inseriti; e un **percorso** con partenza nel punto selezionato dall'utente e destinazione nel punto trovato dal software. A questo punto *MoveUp!* è in grado di simulare una corsa con *timing* realistico, sfruttando la successione di immagini *Street View* disponibili lungo il percorso calcolato.

Il presente lavoro di tesi riprende quello del collega, con l'obiettivo di espandere le funzionalità di *MoveUp!* nell'ambito della *Virtual Reality*.



Figura 1.1: Finestra 'Calcola percorso' di MoveUp!



Figura 1.2: Durante la simulazione in MoveUp!

## 1.3 Virtual Reality terapeutica

Negli ultimi anni l'utilizzo di applicazioni in ambito *Virtual Reality* si è fortemente diffuso in diversi contesti: ludico, terapeutico, didattico. Seppur ancora oggetto di studio, diversi articoli scientifici e ricerche si sono concentrati sul fatto che la *Virtual Reality* potesse essere in grado di fornire un supporto a persone affette da gravi problemi psico-fisici, come ad esempio la sindrome Hikikomori, che porta i soggetti affetti a ritirarsi dalla vita sociale, spesso cercando livelli estremi di isolamento e confinamento.

Gli individui colpiti da questo tipo di problematiche, possono, tramite la *Virtual Reality*, ritrovare sicurezza nell'uscire di casa e affrontare il mondo esterno, provando le stesse emozioni che proverebbero varcando la porta della loro casa, ma all'interno di un ambiente protetto, sicuro e virtuale. [6] [7] [8] [9] [10]

Le ultime ricerche scientifiche a riguardo hanno portato alla realizzazione di istituti didattici completamente virtuali, che diano la possibilità, a chi ne ha bisogno, di partecipare con il proprio avatar virtuale alla vita scolastica. In questo modo, i soggetti, oltre a potersi istruire, hanno modo di avviare relazioni sociali, che, seppur virtuali, sono il primo passo verso quelle reali. [11] [12]

Chiaramente le ragioni che possono spingere un soggetto a vivere uno stile di vita sedentario sono diverse e vanno quindi risolte diversamente: per questo è essenziale l'affiancamento della virtual reality ad una terapia psicologica studiata su misura per l'individuo in questione ed il suo specifico problema.

## 1.4 MoveUp! VR Edition

È qui che nasce *MoveUp! VR Edition*, la versione di *MoveUp!* con simulazione in spazio tridimensionale virtuale. *MoveUp! VR Edition* offre, in aggiunta a quelle presenti in *MoveUp!*, le seguenti funzionalità:

- Generazione nello spazio 3D virtuale della porzione del mondo nel quale effettuare la simulazione, il più fedele possibile alla realtà, con edifici e terreno in tre dimensioni;
- Simulazione itinerario all'interno del mondo 3D virtuale, con movimento continuo e possibilità di ruotare il *point of view*;
- Passaggio dalla visualizzazione standard a quella in *Virtual Reality* tramite un bottone presente nelle modalità *Street View* e *Simulazione*;

La funzionalità che permette la simulazione per visualizzazioni successive di immagini statiche *Street View* è stata rimossa: è possibile visualizzare l'immagine panoramica a 360° solamente nel punto di partenza e di arrivo. Tale funzionalità non è stata riprodotta per evitare di utilizzare del tempo sviluppandola nuovamente. È comunque integrabile all'interno del progetto.

# Capitolo 2

## Analisi del problema

In questo capitolo discuteremo dei problemi da risolvere per lo sviluppo di *MoveUp! VR Edition*. Analizzeremo le possibili piattaforme da utilizzare per facilitare lo sviluppo e quali di esse sono state scelte. Infine verranno dettagliatamente spiegate le funzionalità offerte all’utente dall’applicazione.

### 2.1 Problemi principali

#### 2.1.1 Lo spazio tridimensionale

Si vuole realizzare un applicativo Android che lavori nello spazio tridimensionale. I software 3D più diffusi nel mondo sono senz’altro i videogiochi, soprattutto se si parla di applicativi *mobile*. Seppur siano nati prima i software di modellazione rispetto ai videogame tridimensionali, per sviluppare applicativi di questo tipo al giorno d’oggi, la cosa più conveniente da fare è sfruttare i cosiddetti *Game Engine*.

#### Game Engine

Anche detti *Motori grafici*, sono tra i software più complessi al mondo. Motori grafici avanzati come l’Unreal Engine 4 e lo Unity Engine forniscono una suite di strumenti di sviluppo visuali in aggiunta alla componente software riutilizzabile. Questi strumenti vengono forniti generalmente all’interno di un ambiente di sviluppo integrato (IDE). I motori grafici vengono chiamati spesso ”game middleware” perché forniscono una piattaforma flessibile e riutilizzabile che fornisce tutte le funzionalità chiave necessarie per sviluppare un’applicazione ludica riducendo i costi, la complessità e il tempo impiegato; tutti fattori critici e altamente competitivi nell’industria di videogiochi e computer.

Nonostante l’idea di motore grafico rimandi immediatamente al concetto di videogioco, in verità essi vengono utilizzati in molti altri tipi di applicazioni interattive che richiedono grafica in tempo reale, come dimostrazioni commerciali, progettazioni architettoniche, simulazioni e ambienti di modellazione.

#### 2.1.2 La Virtual Reality

La realtà virtuale, per sua stessa definizione, simula la realtà effettiva. L’ avanzamento delle tecnologie informatiche permette di navigare in ambientazioni fotorealistiche in tempo reale, interagendo con gli oggetti presenti in esse.

## Il visore

La periferica alla base di questa tecnologia, che permette a chiunque di usufruire della Virtual Reality, è il visore. Questo dispositivo è un wearable che va indossato sul viso, più comunemente sotto forma di vero e proprio *headset*: va indossato tenendolo sul naso e può mantenersi sulla testa del soggetto con uno specifico supporto dietro la nuca. Inoltre, i visori per la realtà virtuale sono spesso dotati anche di auricolari, per permettere un'esperienza ancora più verosimile, attraverso i suoni dell'ambiente ricreato graficamente.

I visori per la realtà virtuale sono caratterizzati da **due lenti**, o per essere più precisi da due display, che corrispondono a ciascun occhio una volta indossati. Capaci di restituire immagini stereoscopiche, la risoluzione di questi display è altissima, per mantenere fede a rappresentazioni molto dettagliate della realtà virtuale e, in abbinamento a sensori come l'accelerometro e il **giroscopio**, danno l'illusione di muoversi davvero al centro di ciò che si sta guardando. Queste due categorie di sensori permettono di tracciare i movimenti del capo, e di seguirli al fine di mostrare una variazione di immagine corrispondente al movimento.

Esistono alcuni dispositivi che aggiungono anche l'esperienza tattile, attraverso guanti o tute speciali da indossare prima di immergersi nella realtà virtuale.

## Cardboard

Per usufruire della realtà virtuale su applicativi mobile, sono stati progettati dei visori in grado di sfruttare il display dello smartphone. Tali dispositivi sono composti da una struttura *head-wearable*, contenente due lenti dalla lunghezza focale di 45 mm, le quali vengono allineate al display dello smartphone da inserire nel visore, in modo tale che ognuna proietti la porzione di schermo a essa allineata: infatti, ciò che è normalmente rappresentato nel display, viene elaborato in due differenti visualizzazioni della scena, in modo tale da poter restituire a chi lo utilizza un'immagine stereoscopica dell'ambiente virtuale. Questo tipo di visore prende il nome di **Google Cardboard**, le sue specifiche sono state progettate da Google, che ha reso l'elenco di parti, schemi e istruzioni di assemblaggio liberamente disponibili sul loro sito Web, consentendo alle persone di assemblare le Cardboard per conto proprio e interessarsi allo sviluppo di applicativi VR.

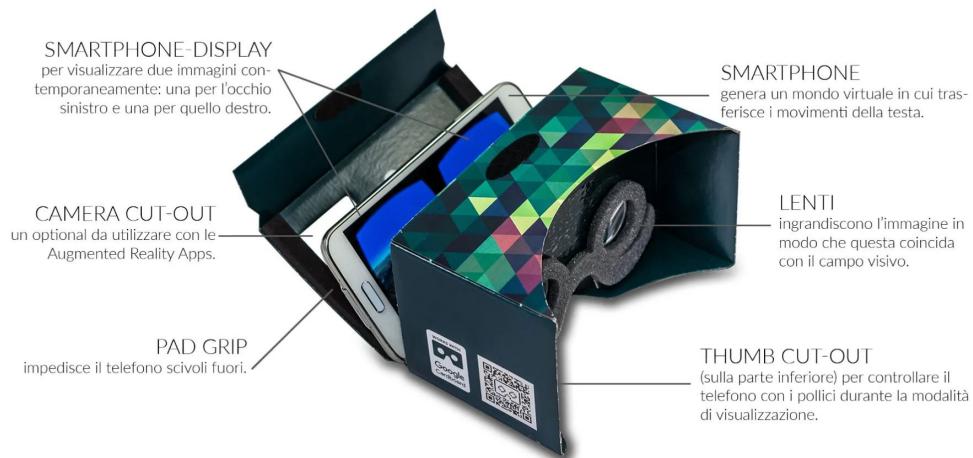


Figura 2.1: Una versione modificata di Google Cardboard

### 2.1.3 Le mappe

Per la realizzazione di *MoveUp! VR Edition* è stato indispensabile l'utilizzo dei dati delle mappe online, le quali contengono informazioni utili come:

- **morfologia delle strade**: necessaria per il calcolo delle indicazioni e la rappresentazione della realtà;
- **altitudine del terreno**: più fedele, simulazioni in aree non-piane risultano così verosimili;
- **morfologia degli edifici**: per generare proceduralmente i modelli tridimensionali in contesto urbano;
- **etichettatura per aree**: ad esempio 'urbana', 'paesaggio', 'acqua' *ecc.*;
- **informazioni sui *Point of Interest* (POI)**: come ristoranti, trasporti pubblici, negozi, *eccetera*;

### 2.1.4 La visualizzazione panoramica

Google offre le funzionalità di Street View per rendere disponibili le immagini panoramiche a 360°. Non è possibile, però, scaricare direttamente il panorama nella sua totalità. Tramite una richiesta Web, Street View Static API è in grado di ritornare una porzione di un'immagine panoramica in base ai seguenti parametri:

- Size: dimensione della porzione di immagine, ad esempio 512x512;
- Location: coordinate nelle quali è stata scattata l'immagine;
- Heading: direzione orizzontale verso la quale si vuole la porzione di immagine;
- Pitch: direzione verticale verso la quale si vuole la porzione di immagine;

Vedremo dettagliatamente nella Sezione 3.2.5 come si è fatto in modo che l'utente finale della nostra applicazione potesse usufruire di immagini panoramiche provenienti da Street View;

## 2.2 Tecnologie abilitanti

### 2.2.1 Android Studio

Android Studio è un ambiente di sviluppo integrato (IDE) per lo sviluppo per la piattaforma Android. Basato sul software di JetBrains IntelliJ IDEA, Android Studio è stato progettato specificamente per lo sviluppo di applicazioni Android, e sostituisce gli Android Development Tools (ADT) di Eclipse, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android.

Seppur le mappe e la realtà virtuale siano facilmente integrabili in un progetto sviluppato su Android Studio, per realizzare applicativi in spazio tridimensionale è richiesta una quantità di tempo e risorse superiore rispetto a quella richiesta da un motore grafico dotato di IDE. Questo è dovuto al fatto che, malgrado siano presenti librerie di grafica tridimensionale per Android Studio, esse non sono affiancate da

un IDE dedicato, il quale offre solitamente allo sviluppatore una serie di strumenti orientati alla realizzazione di applicativi di questo tipo. Andiamo così a ridurre la quantità di tempo necessaria allo sviluppo, rendendo le operazioni di base - come il posizionamento di un oggetto nella scena - semplici, rapide ed intuitive.

Per lo sviluppo di *MoveUp!*, nella sua versione non dotata di funzionalità Virtual Reality, si è utilizzato Android Studio.

### 2.2.2 Motori grafici tridimensionali dotati di IDE

Nel mercato sono presenti moltissimi motori grafici dotati di IDE (*GIDE* - *Game Integrated Developement Kit*). Essendo già a conoscenza del fatto che i due GIDE più utilizzati e potenti sono **Unity** e **Unreal Engine**, e data la mia esperienza pregressa con Unity, la scelta è ricaduta proprio su quest'ultimo.

	<b>Unity</b>	<b>Unreal Engine</b>
<b>Linguaggio</b>	Javascript / C#	C++ / Javascript
<b>Linguaggio visuale</b>	Si	Si, più avanzato
<b>Adatto a piccoli team</b>	Si	No
<b>Fotorealismo</b>	Quasi	Si
<b>Leggero</b>	Si	No
<b>Multipiattaforma</b>	Si	Si

Tabella 2.1: Caratteristiche di Unity e Unreal Engine a confronto.

### Struttura di un progetto in Unity

L'IDE di Unity è formato da una serie di finestre, componibili all'interno di quella principale a proprio piacimento. Le più importanti sono:

- **Project:** contiene un *file explorer* della cartella *Assets* del progetto, all'interno della quale sono organizzate tutte le risorse disponibili per lo sviluppo. Tali risorse sono file contenuti nella memoria di massa del sistema sul quale si sta lavorando.
- **Scene:** permette di visualizzare e interagire con gli oggetti all'interno della scena attualmente in uso. È quindi una rappresentazione dello spazio tridimensionale e di ciò che vi è contenuto al suo interno.
- **Hierarchy:** mostra una lista di tutti i *GameObjects* presenti nella scena attualmente in uso. È possibile raggrupparli per creare famiglie di *GameObjects* con struttura *Parent - Childrens*.
- **Inspector:** mostra i dettagli e le proprietà dell'oggetto selezionato. Può variare molto in base al tipo di oggetto. Nel caso in cui ce ne siano, mostra anche la lista di *Components* assegnati all'oggetto.
- **GameView:** permette di avere un'anteprima direttamente all'interno dello IDE. Evita la necessità di una *Build* ogni volta che si testa il programma.

## Capitolo 2. Analisi del problema

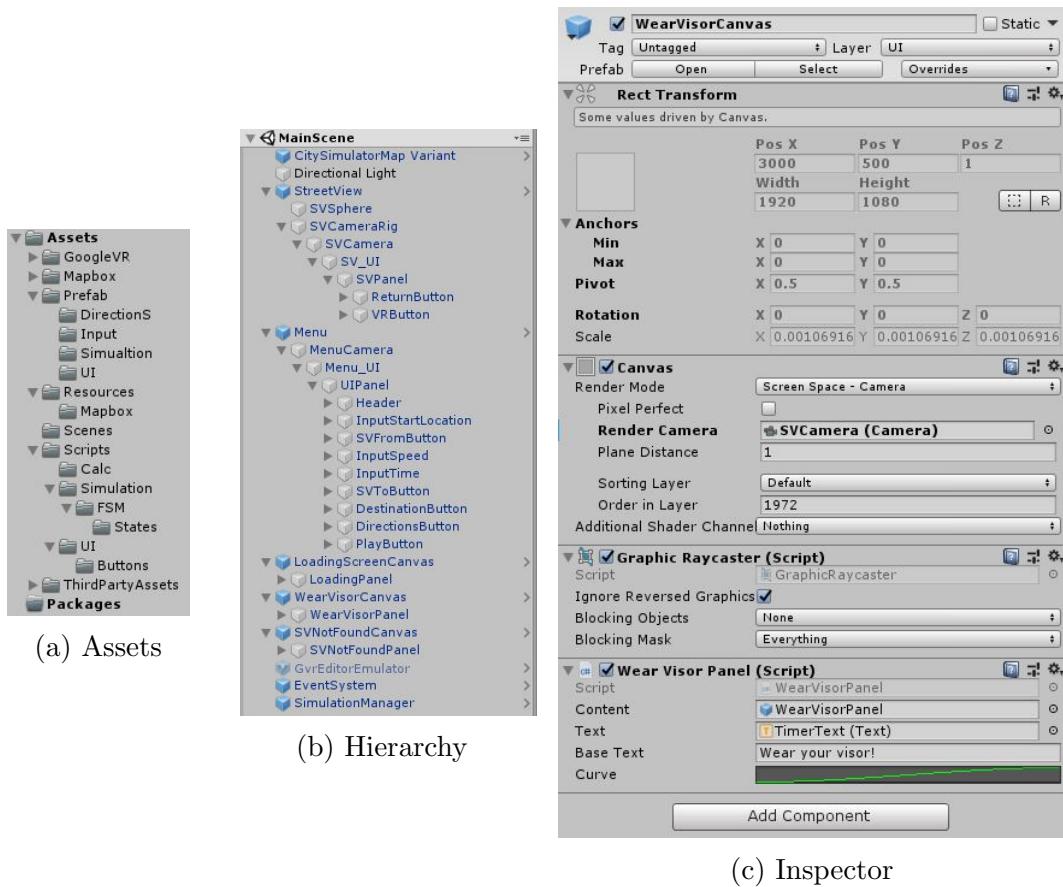


Figura 2.2: Varie parti dell’interfaccia di Unity. Gli screenshot sono tratti dall’applicazione sviluppata.

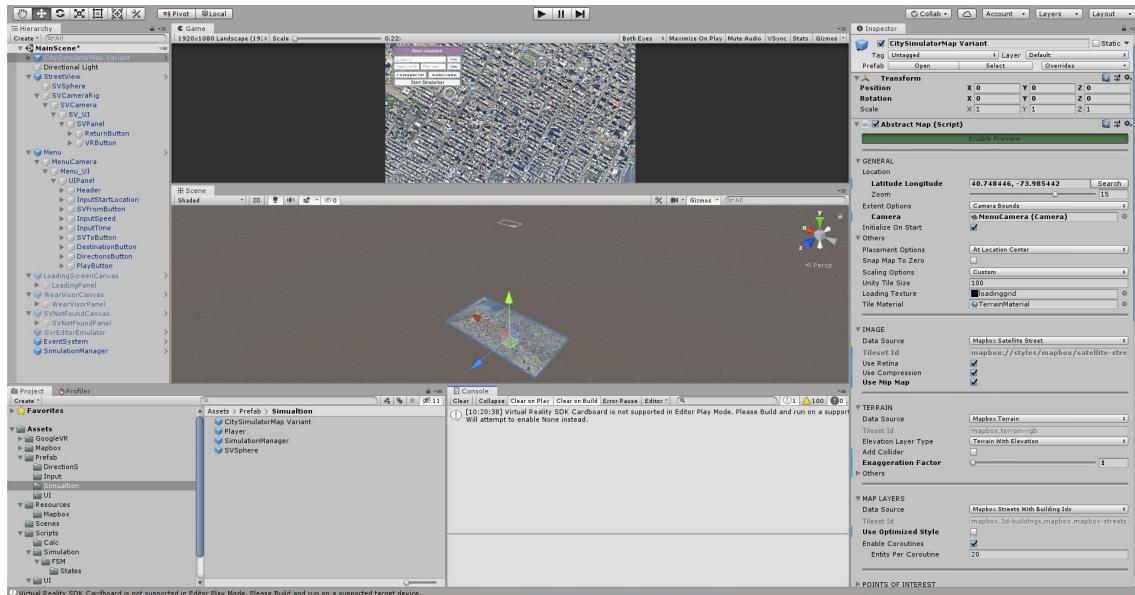


Figura 2.3: IDE di Unity. Il progetto aperto all’interno è quello relativo all’applicazione sviluppata.

Ogni oggetto contenuto in una **Scene** è quindi un **GameObject**. Ad ognuno di essi sono assegnati una serie di **Components**, che ne definiscono il comportamento e l'aspetto. L'unico Components presente in ogni GameObject all'interno della scena è **Transform**, che definisce le informazioni spaziali dell'oggetto come *rotazione*, *posizione* e *scala*. È possibile creare nuovi Components per personalizzare il comportamento degli oggetti facendo uso degli **Scripts**, delle classi C# ereditate dalla classe **MonoBehaviour** delle librerie di Unity. La classe MonoBehaviour contiene una serie di metodi intuitivi che vengono eseguiti automaticamente da Unity, i più utilizzati sono:

- Awake(): eseguito quando l'istanza dello script viene caricata;
- Start(): eseguito al frame precedente la prima esecuzione di Update;
- Update(): eseguito ad ogni frame;
- LateUpdate(): eseguito dopo tutti gli Update, ad ogni frame;
- OnDestroy(): eseguito quando viene invocata la distruzione del GameObject;

### 2.2.3 GoogleVR Cardboard SDK per Unity

Google ha introdotto il suo SDK per sfruttare la realtà virtuale all'interno di applicativi sviluppati con Unity. Per il nostro progetto è sufficiente sfruttare la funzionalità di base della realtà virtuale, ovvero motion tracking e rendering stereoscopico. Non è previsto input tramite il visore e nemmeno l'utilizzo di telecomandi. Per questa ragione, delle tante funzionalità rese disponibili dalla Cardboard SDK, oltre a quella che abilita e disabilita la visualizzazione VR, è bastato sfruttare il GameObject GVREDITOREMULATOR, utile a testare da desktop le funzionalità VR con mouse e tastiera, emulando il movimento della testa.



Figura 2.4: Come appare GvrEditorEmulator nell'Inspector

### 2.2.4 Mappe integrabili in Unity

#### Introduzione - OpenStreetMaps

OpenStreetMap (OSM) è un progetto collaborativo finalizzato a creare mappe del mondo a contenuto libero. Il progetto punta ad una raccolta mondiale di dati geografici, con scopo principale la creazione di mappe e cartografie.

La caratteristica fondamentale dei dati geografici presenti in OSM è che possiedono una licenza libera, la Open Database License: è cioè possibile utilizzarli liberamente per qualsiasi scopo con il solo vincolo di citare la fonte e usare la stessa licenza per eventuali lavori derivati dai dati di OSM. Tutti possono contribuire arricchendo o correggendo i dati.

OSM contiene quindi tutti i dati di cui necessitiamo, ma non dispone di alcuna libreria per elaborare con semplicità tali informazioni. Per ricostruire il mondo nello spazio virtuale, bisognerebbe elaborare tutti i dati in formato XML ottenuti da OSM, ed implementare i metodi che costruiscono gli oggetti 3D veri e propri. Ovviamente, tale lavoro è impegnativo e lungo, si è quindi deciso di affidarsi a librerie sviluppate appositamente per Unity.

### Principali API per mappe

Nella seguente tabella sono mostrate le principali differenze tra le SDK compatibili con Unity per mappe del mondo tenute in considerazione per lo sviluppo del progetto.

	<b>WRLD</b>	<b>Google Maps</b>	<b>Mapbox</b>	<b>Bing Maps</b>
<b>Gratis</b>	limitato	limitato	limitato	si
<b>Scaricabile</b>	si	su richiesta	si	si
<b>Versione stabile</b>	si	si	si	no
<b>Presenza online</b>	scarsa	si	si	no
<b>POI</b>	Yelp	Google Places	proprietario	Yelp
<b>Edifici 3D</b>	si	si, no texture	si	si
<b>Ambiente 3D</b>	si	solo natura	solo natura	si

Tabella 2.2: Principali API per mappe a confronto.

GOOGLE MAPS SDK per Unity è stato scartato in quanto non liberamente scaricabile dal web. Si è tentato di accedervi inviando una richiesta a Google, senza mai ricevere risposta. Cercando online, pare che venga consentito l'accesso solo a grandi team di sviluppo.

WRLD SDK per Unity offre molte funzionalità interessanti, ma, durante i primi test, non mi è piaciuto l'approccio che utilizza.

BING MAPS SDK per Unity permette di ricreare ambienti reali tridimensionali molto accattivanti e realistici, ma ha uno scarsissimo supporto online, ed è attualmente in fase di sviluppo.

MAPBOX SDK per Unity è stata la mia scelta. Dopo averlo testato all'interno dell'IDE, mi è piaciuta la personalizzazione che offre della mappa e l'approccio con cui si interagisce con essa tramite l'interfaccia e il codice. Sono inoltre presenti online molte domande con annesse risposte da parte del team di supporto di MapBox.

### Funzionamento di MapBox SDK per Unity

Mapbox SDK per Unity è un insieme di strumenti per creare applicazioni Unity a partire da dati di mappe reali. È costituito da una solida API per l'interfacciamento con i servizi Web di Mapbox e la conversione delle risorse della mappa in GameObjects, nonché da una solida interfaccia utente all'interno della piattaforma Unity.

Principali caratteristiche:

- **Dati dinamici:** Mentre alcuni plugin Unity per le mappe sono progettati per aiutare gli sviluppatori a creare ambienti di gioco statici a partire da dati cartografici, l'SDK di Mapbox è stato pensato per richiedere e visualizzare i dati cartografici in fase d'esecuzione. Ciò significa che le applicazioni visualizzeranno sempre la versione più recente dei dati spaziali, e richiederanno sempre e solo il sottoinsieme di dati che corrisponde all'area che l'utente sta visualizzando, mantenendo leggeri gli applicativi.
- **Generazione di Mesh:** Mapbox SDK offre la possibilità di generare mesh per creare mappe 2D o 3D, con immagini satellitari, stili di mappe personalizzati, dati del terreno e punti di interesse.
- **Vector Tiles API:** Una Vector Tile è un formato dati leggero per l'archiviazione di dati vettoriali geospaziali, come punti, linee e poligoni. I Vector Tile vengono utilizzati per creare i Vector Tilesets di Mapbox, e sono la fonte dei dati per lo stile della mappa.
- **Raster Tiles API:** I *raster* sono un formato di dati basato su pixel in cui i dati sono memorizzati in una struttura a griglia. Ogni pixel, o unità di informazione, ha le stesse dimensioni e forma, ma varia di valore. Tutte le fotografie digitali sono archiviate in questo formato, comprese le immagini satellitari utilizzate nei Tilesets di Mapbox Satellite. I Raster Tilesets vengono utilizzati da Mapbox per visualizzare le mappe come una griglia di immagini.
- **Geocoding API:** Il FORWARD GEOCODING converte il testo della posizione in coordinate geografiche, trasformando, per esempio, *2 Lincoln Memorial Circle NW* in (-77.050, 38.889).  
Il REVERSE GEOCODING converte le coordinate geografiche in nomi di luoghi, trasformando quindi (-77.050, 38.889) in *2 Lincoln Memorial Circle NW*. Questi nomi di località possono variare in specificità, da singoli indirizzi a stati e paesi che contengono le coordinate indicate.
- **Directions API:** Con l'API delle indicazioni stradali è possibile:
  - Calcolare percorsi di guida, pedonali e ciclabili ottimali utilizzando un percorso consapevole del traffico e degli incidenti;
  - Produrre istruzioni passo-passo;
  - Produrre percorsi con un massimo di 25 coordinate;

Poniamo particolare riguardo per un'ultima caratteristica di MapBox: i **Tile Provider**. In queste classi, ereditate da **AbstractTileProvider**, viene definita la logica da seguire per selezionare le *tile* - ovvero le porzioni di mappa - da caricare. Mapbox mette a disposizione tre Tile Providers:

- **Camera Bounds:** include solo le tile che rientrano nel campo visivo della camera attualmente in uso;
- **Range Around Center:** include tutte le tile poste entro una certa distanza dal punto in cui la mappa è centrata;
- **Range Around Transform:** include tutte le tile poste entro una certa distanza dalla posizione del GameObject scelto;

## 2.3 Funzionalità

*MoveUp! VR Edition* propone un itinerario a partire da un punto prestabilito dall’utente e gli offre la possibilità di simulare il percorso all’interno di un ambiente virtuale tridimensionale. Possiamo considerare l’applicazione come composta da tre schermate, ognuna dotata di diversa visualizzazione:

1. Menù iniziale con visualizzazione *top-down* interattiva sulla mappa;
2. Visualizzazione in modalità *Street View*;
3. Visualizzazione in prima persona durante la simulazione nel mondo virtuale;

### 2.3.1 Schermata iniziale

Avviata l’applicazione, in seguito al caricamento iniziale, troviamo un piccolo menù in alto a sinistra, mostrato a Figura 2.6, contenente i seguenti campi di testo:

- Luogo di partenza
- Velocità di corsa (*km/h*)
- Tempo di corsa (*min*)

ed i seguenti bottoni:

- Bottone *View* associato al punto di partenza, permette di visualizzare l’immagine *Street View* in quelle coordinate;
- Bottone *View* associato al punto di arrivo, come sopra;
- Bottone *Find destination point*, permette di trovare un punto di destinazione in funzione dei parametri inseriti;
- Bottone *Calculate directions*, permette di calcolare e rappresentare il percorso che collega i due punti;
- Bottone *Start simulation*, permette di avviare la simulazione in prima persona lungo il percorso precedentemente calcolato;

Sullo sfondo troviamo una visualizzazione dall’alto (*top-down*) della mappa del mondo, composta da immagini satellitari ed etichette poste su monumenti, strade, quartieri *eccetera*. All’utente è data la possibilità di muoversi liberamente nella mappa (*pan-to-move*), e di sfruttare la funzione di zoom (*pinch-to-zoom*), con aumento incrementale della qualità delle immagini satellitari.



Figura 2.5: Schermata iniziale di MoveUp! VR Edition

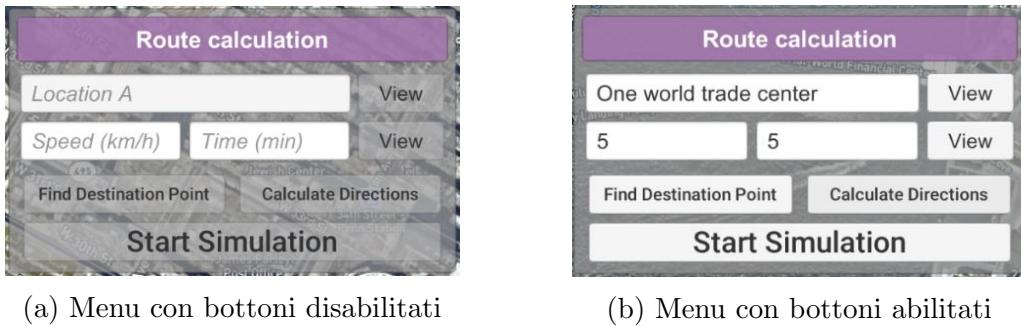


Figura 2.6

Come si può osservare dalla Figura 2.5, i pulsanti sono inizialmente disattivati, e vengono abilitati durante l'utilizzo dell'applicazione, come mostrato da questo diagramma di flusso:

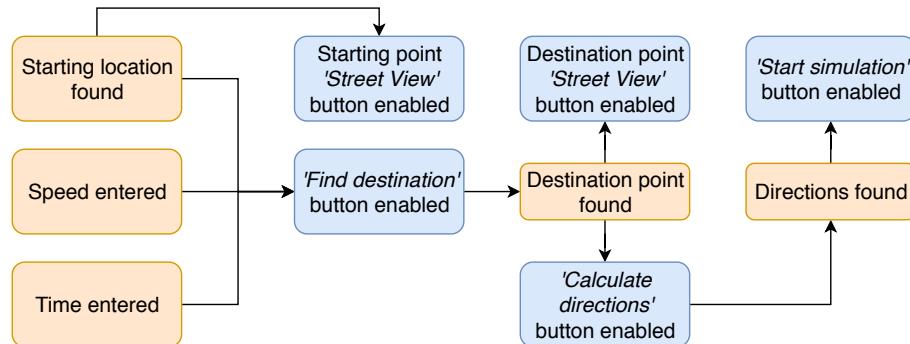


Figura 2.7: Diagramma di flusso dell'attivazione dei pulsanti della UI

### 2.3.2 Street View

Interagendo con il bottone '*View*' posto a fianco del campo *Location A* e con quello posto a fianco dei campi *Speed* e *Time* (Figura 2.6), si ha la possibilità di visualizzare un'immagine panoramica a 360° scattata alle coordinate del punto di partenza e di destinazione rispettivamente. L'immagine panoramica è presa dall'archivio di immagini di *Google Street View*. Dal momento in cui ci si trova nella visualizzazione *Street View*, la rotazione della camera segue quella registrata dal giroscopio del *device* su cui l'applicazione è in esecuzione.



Figura 2.8: Visualizzazione Street View - Central Park, New York

Questa funzionalità è molto importante anche per la buona riuscita della modalità VR. Infatti, in seguito alla pressione del bottone dedicato, l'utente viene invitato ad indossare il visore prima dello scadere del timer presente a schermo, dopo il quale viene abilitata la visualizzazione stereoscopica adatta al visore. Una volta indossato il visore con lo smartphone al suo interno, ci troveremo virtualmente al centro dell'immagine a 360°, e, grazie al giroscopio, l'applicazione sarà in grado di emulare il movimento della nostra testa, rendendo l'esperienza virtuale più realistica.



Figura 2.9: Schermata '*Wear your visor*'

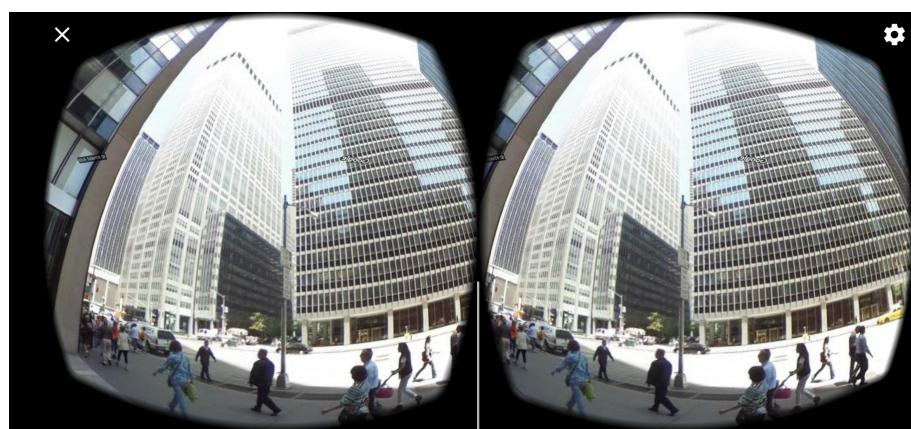


Figura 2.10: Visualizzazione Street View con VR attiva - Manhattan, New York

### 2.3.3 Simulazione

Interagendo con il bottone '*Start simulation*', si passa ad una visualizzazione in prima persona, all'interno di una ricostruzione della realtà limitata alle zone adiacenti al percorso precedentemente calcolato. Una volta caricata tutta la porzione di mappa necessaria, la telecamera comincerà a muoversi automaticamente lungo l'itinerario. Raggiunto il termine della simulazione, viene visualizzata l'immagine panoramica *Street View* del punto di arrivo.



Figura 2.11: Simulazione - World Trade Center, New York

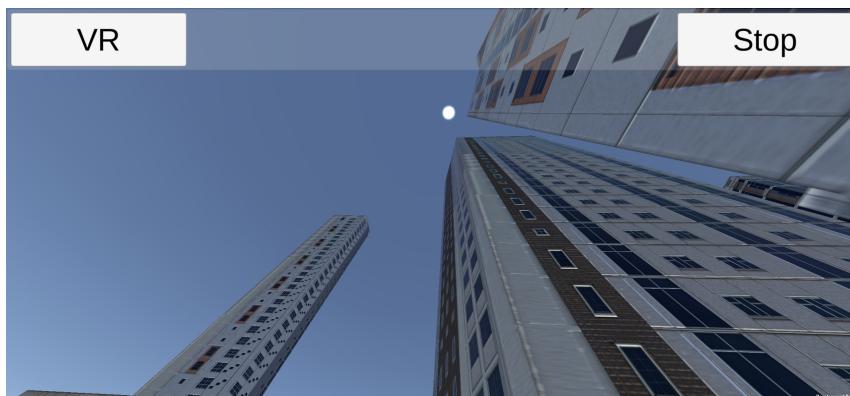


Figura 2.12: Proporzioni in simulazione - World Trade Center, New York

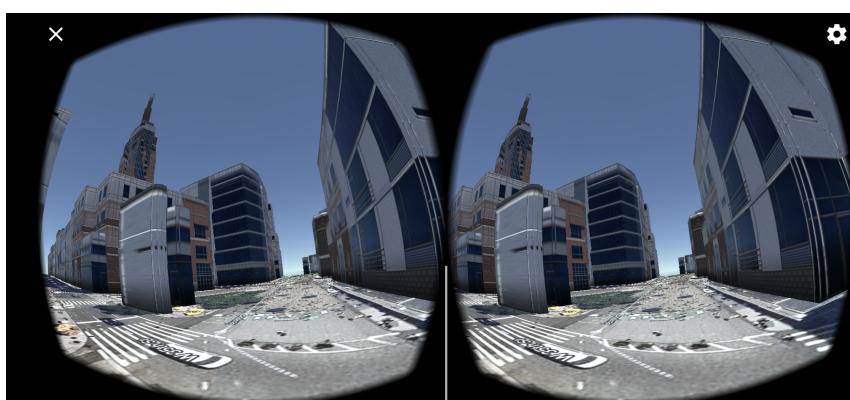


Figura 2.13: Simulazione con VR attiva - Empire State Building, New York

# Capitolo 3

## Progettazione e Implementazione

In questo capitolo descriverò il processo di progettazione e implementazione di *MoveUp! VR Edition*. In seguito alla definizione della struttura a macchina a stati, si entrerà nel dettaglio di ogni script implementato. Con il supporto di diagrammi di flusso e diagrammi UML, si cercherà poi di rappresentare le fasi previste dall'esecuzione di ogni stato e le modalità con cui le varie classi collaborano tra loro.

### 3.1 Finite State Machine

Come descritto alla Sezione 2.2.2, Unity non ha un metodo `main` dal quale controllare l'esecuzione del programma. Ogni oggetto della scena è dotato di script che ereditano i metodi della classe `MonoBehaviour`, eseguiti dalla pipeline di Unity.

Per un miglior controllo e una migliore interazione tra gli oggetti, ho pensato di strutturare l'implementazione come una macchina a stati.

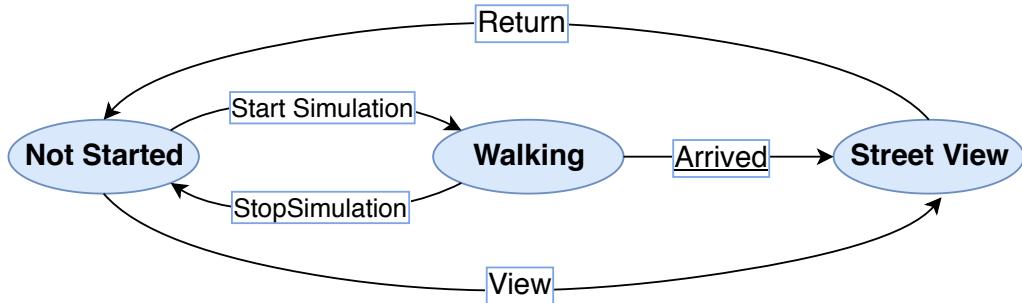


Figura 3.1: Macchina a stati

#### 3.1.1 Implementazione della macchina a stati

Per realizzare la macchina a stati ho implementato inizialmente l'interfaccia `ISimulationState`, dotata dei seguenti metodi:

- `void EnterState()`: metodo di ingresso nello stato;
- `void UpdateState()`: richiamato ad ogni frame;
- `void ToNotStartedState()`: metodo di uscita verso `NotStartedState`;
- `void ToStreetViewState()`: metodo di uscita verso `StreetViewState`;

- `void ToWalkingState()`: metodo di uscita verso `WalkingState`;

Ogni classe che rappresenta uno stato implemeterà questa interfaccia.

Per fare in modo che vengano eseguiti i metodi `EnterState()` e `UpdateState()`, necessitiamo di una classe che li richiami al momento giusto in base a quale stato è attualmente in esecuzione: tale classe è `SimulationStatePattern` ed è figlia di `MonoBehaviour`. Al suo interno troviamo:

- I riferimenti per la macchina a stati, ovvero:
  - Uno ad ogni stato della FSM;
  - Uno allo stato corrente: `ISimulationState currentState`;
- Un riferimento ad ogni `GameObject` al quale è necessario accedere dalle altre classi durante l'esecuzione del programma:
  - Ad ogni elemento della UI;
  - Ad ogni classe che effettua calcoli, come `DirectionsCalc`;
  - Ad ogni oggetto della scena, come il *Player* o le *Camere*;
- I parametri della simulazione, come:
  - `directionsZoom`: lo zoom della mappa nel momento in cui si calcola il percorso;
  - `fovDirections`: parametro *field of view* (campo visivo) della camera nella fase d'esecuzione in cui si visualizza la mappa con percorso calcolato;
  - `alongPathNearTiles`: quanto deve essere vicina una Tile rispetto a quelle già caricate lungo il percorso, per essere anch'essa caricata;
  - `playerOffset`: distanza in altezza del *player* rispetto al terreno;
  - `zoomSpeed`: sensibilità dei comandi che effettuano lo zoom;
- I flags, come:
  - `fromGeocoded`: indica se il punto di partenza è stato identificato come coordinate a partire dalla stringa inserita dall'utente;
  - `loadingSView`: indica se le immagini per la visualizzazione Street View sono in fase di download;
  - `VR_on`: indica se è attiva la visualizzazione stereoscopica per la virtual reality;

Ogni classe che necessita di accedere alle altre, conterrà un riferimento a `SimulationStatePattern`, il quale vi permette l'accesso tramite i suoi riferimenti incapsulati. In questo modo si evita di dover riscrivere, in ogni classe, i riferimenti ad ogni altra a cui essa vuole accedere.

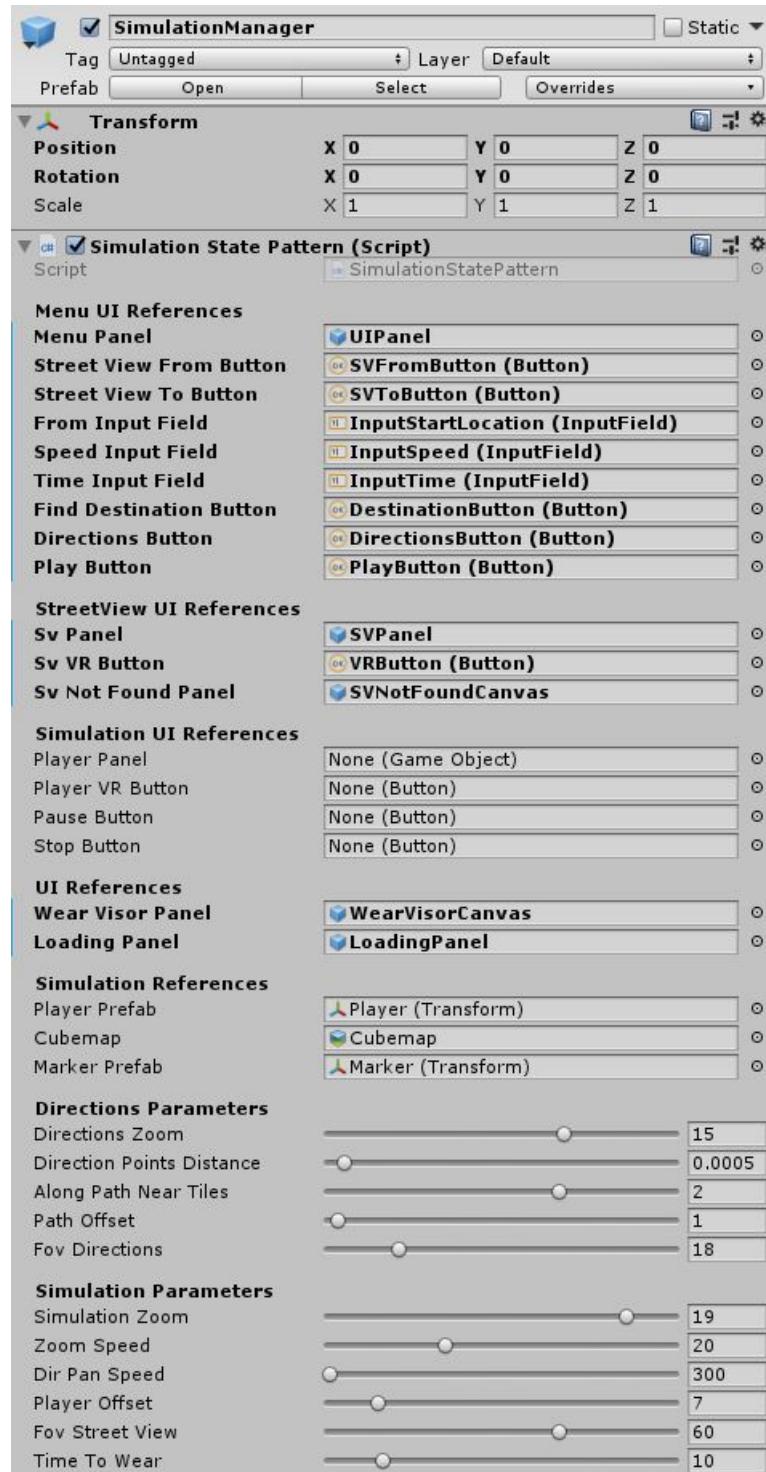


Figura 3.2: Finestra Inspector del GameObject *SimulationStatePattern*. In alto troviamo le references da assegnare manualmente; in basso i parametri.

## 3.2 Scripts di calcolo

Sono script assegnati al GameObject *SimulationManager*, che contiene, oltre a questi script, *SimulationStatePattern.cs*.

### 3.2.1 Geocoder.cs

La classe implementa il metodo al quale viene delegato il compito di gestire la risposta della Geocoder API di Mapbox, come mostrato in seguito. (Figura 3.10) Ricevuta tale risposta contenente le coordinate, esse vengono memorizzate in una variabile leggibile esternamente. Il metodo si occupa anche di ri-centrare la mappa nella posizione di interesse, e di istanziarvi un segnaposto.

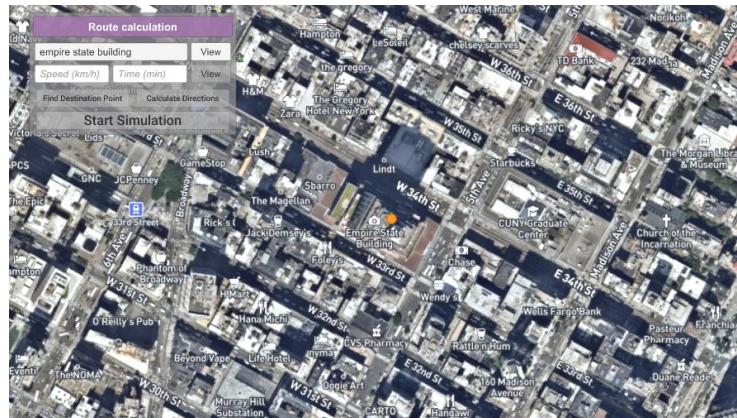


Figura 3.3: 'Empire State Building' geocoded con segnaposto inserito

### 3.2.2 DestinationCalc.cs

Si occupa di trovare il punto di destinazione da proporre all'utente in funzione dei parametri di *velocità* e *tempo*. Per svolgere questo compito sfrutta la classe di Mapbox **CheapRuler.cs**, la quale contiene metodi utili a misurare distanze e fare altri calcoli di questo tipo, senza appesantire troppo l'applicativo. Tale classe è stata implementata con l'intento di essere veloce e leggera, e non troppo precisa.

Una volta calcolata la distanza percorribile sulla base dei parametri, il metodo `ruler.Destination(startLocation, distance, bearing)` ritorna le coordinate di un punto posizionato sulla circonferenza di raggio *distance* con centro in *startLocation* ad un'angolazione rispetto al nord pari a *bearing* - un valore casuale tra 1 e 360. Una volta ottenuta la posizione, essa viene memorizzata in una variabile leggibile esternamente. Successivamente il metodo istanzia il segnaposto e centra la mappa.

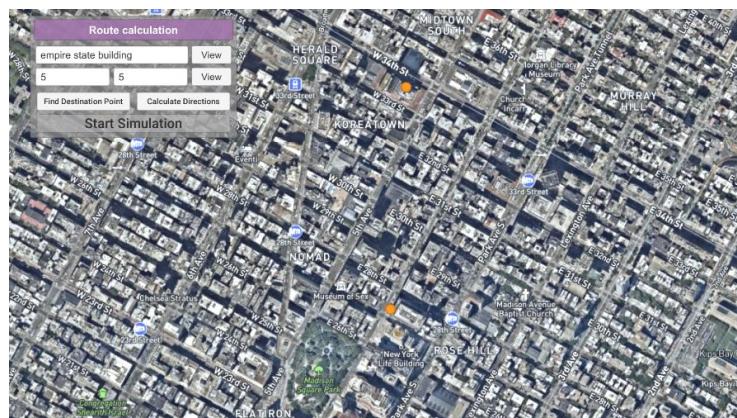


Figura 3.4: Segnaposto inserito nel punto di arrivo proposto

### 3.2.3 PathTileProvider.cs

Durante lo sviluppo ci si è accorti del fatto che i Tile Provider forniti da Mapbox non erano adatti al nostro scopo. La causa è dovuta al dover aggiornare la mappa quando si è già calcolato il percorso, che va quindi ricalcolato e ridisegnato ogni volta che ci si muove all'interno della visuale, causando rallentamenti. Un altro caso problematico si verificava durante l'esecuzione della simulazione, che invece subiva rallentamenti durante il caricamento delle nuove tiles. Si è così deciso di scrivere un Tile Provider personalizzato, che caricaesse una sola volta tutte le Tile necessarie alla simulazione, successivamente al calcolo del percorso: *Along Path*.

`PathTileProvider` è una classe derivata da `AbstractTileProvider`, e in quanto tale implementa i seguenti metodi:

- `OnInitialized()`: non fa altro che inizializzare i valori delle variabili. Viene eseguito nel momento in cui un'istanza della classe viene impostata come Tile Provider della mappa;
- `UpdateTileExtent()`: elimina tutte le tiles caricate precedentemente e invoca il metodo `UpdateTiles()`. Viene invocato ad ogni aggiornamento della mappa;
- `UpdateTiles()`: è la procedura che concretamente seleziona le tiles da caricare. Una volta letti i punti contenuti nella risposta ottenuta dalla Directions API di Mapbox - verosimilmente associati ad ogni svolta da effettuare lungo il percorso - viene calcolata una retta passante per ogni coppia di punti successivi. In seguito, vengono analizzati tutti i punti di ognuna di queste rette, con distanza tra uno e il successivo pari al parametro `DirectionsPointsDistance`. Per ognuno di questi punti, viene richiamato il metodo `addTile(x,y)`;
- `AddTile(x,y)`: calcola l'indirizzo della tile corrispondente al punto in coordinate  $(x,y)$  e, se non già presente al suo interno, la aggiunge alla lista delle tiles attive. Si occupa anche di aggiungere le tiles con una distanza rispetto quelle già caricate minore o uguale al valore del parametro `alongPathNearTiles`. In pratica fa in modo che vengano caricate anche le tiles vicine a quelle che si trovano lungo il percorso.

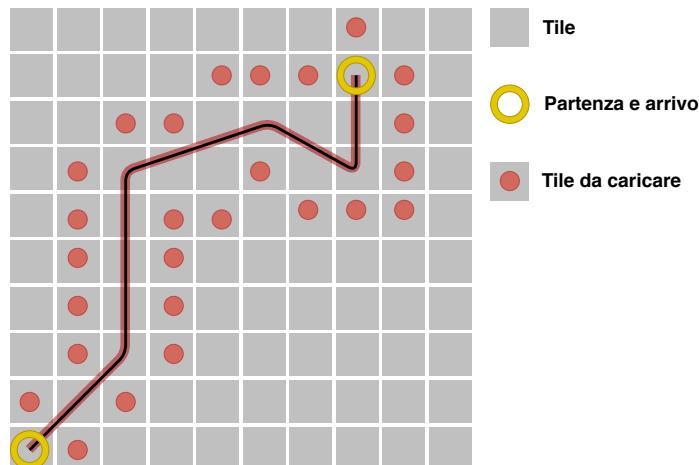


Figura 3.5: Rappresentazione di come vengono selezionate le tiles da caricare. In questo esempio il parametro `alongPathNearTiles` è impostato sul valore 1.

### 3.2.4 DirectionsCalc.cs

Il suo compito è quello di calcolare il percorso tramite la Directions API di Mapbox e di disegnarlo sulla mappa. Contiene il metodo `Query()` per inviare la richiesta alla API, e il metodo `HandleDirectionsResponse` per gestirne la risposta. Una volta ottenuto il percorso, viene impostato *Along Path* come Tile Provider della mappa, che sarà quindi limitata ad un'area attorno a tale percorso.

Successivamente aggiorna la mappa, la quale è programmata per richiamare il metodo `CalculatePointsData()` ogni volta venga aggiornata con *Along Path* impostato. Quest'ultimo metodo si occupa di calcolare l'altezza del terreno in ogni punto del percorso, e di creare proceduralmente un modello tridimensionale da posizionare sulla mappa.

La classe contiene anche il metodo `ReturnToNoDirections()`, per rimuovere un percorso precedentemente calcolato e ritornare nello stato in cui la mappa è totalmente accessibile.

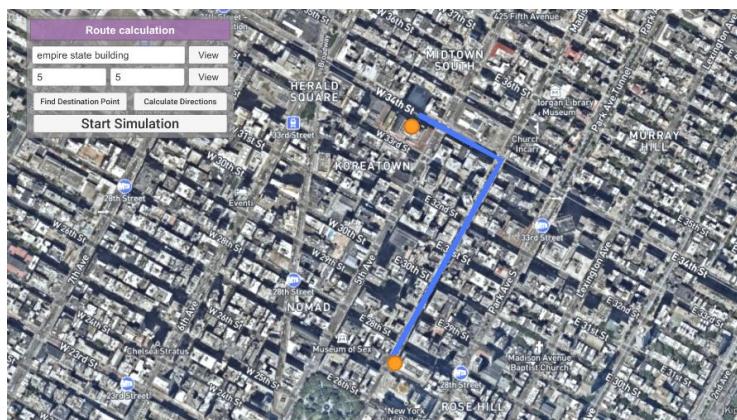
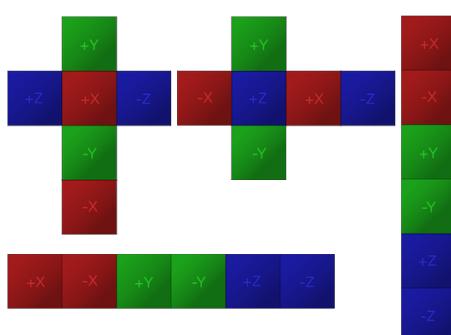


Figura 3.6: Indicazioni calcolate e disegnate sulla mappa

### 3.2.5 StreetViewGenerator.cs

#### Cubemap

Una Cubemap è una raccolta di sei trame quadrate. I sei quadrati formano le facce di un cubo immaginario che circonda un oggetto; ogni faccia rappresenta la vista lungo le direzioni degli assi del mondo (su, giù, sinistra, destra, avanti e indietro).



(a) Struttura di una Cubemap



(b) 'Sorry, we have no imagery here.'

Figura 3.7

Per la generazione e visualizzazione dell'immagine panoramica si è pensato di posizionare una Cubemap attorno ad una sfera, al centro della quale posizionare la camera. Le facce del cubo saranno riempite con sei immagini scaricate dalla Street View Static API di Google tramite richieste HTTP.

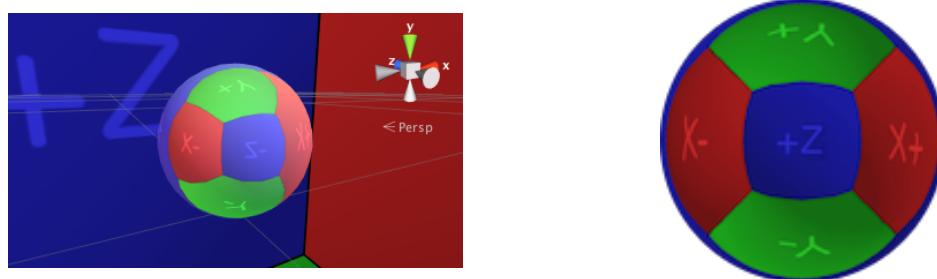


Figura 3.8: Cubemap applicata ad una sfera

In seguito alla chiamata del metodo `DownloadImages(Vector2d _coordinates)`, la classe effettua sei richieste web per ottenere le immagini necessarie. I parametri delle richieste sono memorizzati in array costanti. La procedura è in grado di riavviare il download di una singola immagine nel caso in cui la richiesta dovesse andare in timeout o riscontrare altri tipi di errore.

Una volta ottenute le immagini, tramite il metodo `CheckIfImageExists()` si verifica che la API abbia restituito un'immagine reale e non quella mostrata a Figura 3.7b. Il metodo anche in questo caso sfrutta un array costante in cui sono contenuti i valori dei primi 10 pixel dell'immagine appena citata, da confrontare con i valori ottenuti dall'immagine scaricata. Nel caso in cui dovessero risultare uguali, verrebbe abilitato il GameObject `SVNotFoundCanvas`, pannello della UI con testo '*Street View Images unavailable for these coordinates*'.

### Esempio richiesta Web a Street View API

```
https://maps.googleapis.com/maps/api/streetview?
size=512x512&location=40.7483392,-7.9855392&
heading=180&pitch=0&key=your_personal_API_key
```

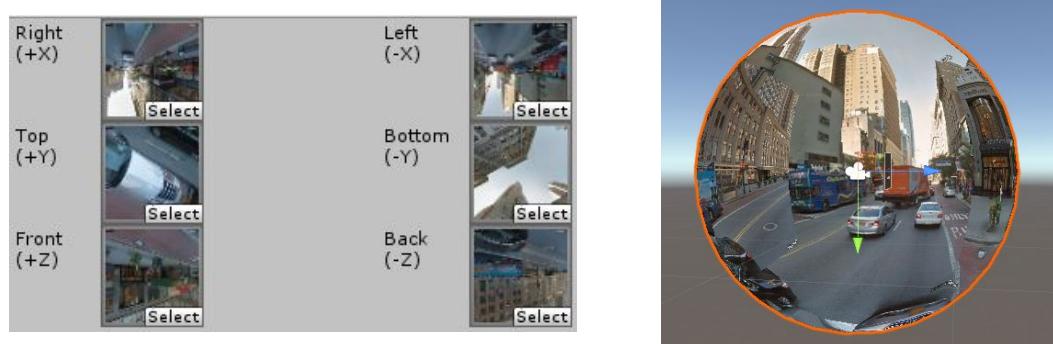


Figura 3.9: Cubemap generata e applicata ad una sfera in Unity

### 3.2.6 VRManager.cs

Gestisce lo switch tra modalità di visualizzazione VR e 2D.

#### Abilitare GVR SDK in Unity

Navigando nei menù di Unity attraverso questo percorso

Edit -> Project Settings -> Player -> XR Settings

accediamo alle impostazioni riguardanti la realtà virtuale. All'interno del menù troviamo il flag *Virtual Reality Supported* - che va selezionato - e, appena sotto, un elenco denominato *Virtual Reality SDKs*. All'interno di questa lista vanno inseriti i SDK per la VR, nel nostro caso ne sono presenti due: *None*, *Cardboard*.

All'avvio dell'applicazione viene invocato il metodo

`XRSettings.LoadDeviceByName("cardboard")`

per selezionare l'SDK specifica del cardboard di GVR; e viene impostato il flag `XRSettings.enabled` sul valore `false`. In questo modo, l'SDK sarà pronto a gestire la VR, che è però momentaneamente disabilitata tramite il flag impostato a false.

#### Funzionamento di VRManager.cs

Contiene i seguenti riferimenti:

- `_gvrEditorEmulator`: permette di emulare la modalità VR da desktop. Con la pressione del tasto `Alt` sulla tastiera si abilita l'inclinazione del capo tramite mouse verso qualunque direzione; con la pressione del tasto `Ctrl`, invece, la rotazione verso destra e sinistra. La classe si occupa anche di gestire la modalità VR su mobile, applicando le variazioni del valore del giroscopio alla rotazione della camera nel mondo virtuale.
- `_graphicRaycaster`: viene utilizzato per il raycast verso un Canvas. Un Canvas è l'area in cui tutti gli elementi dell'interfaccia utente dovrebbero essere contenuti, è un GameObject con un componente Canvas e tutti gli elementi dell'interfaccia utente devono essere figli di tale oggetto. Il Raycaster considera tutti gli elementi grafici all'interno del Canvas e determina se qualcuno di essi è stato colpito.
- `_eventSystem`: è un modo per inviare eventi agli oggetti nell'applicazione in base all'input, che si tratti di tastiera, mouse, tocco o input personalizzato. È costituito da alcune componenti che lavorano insieme per inviare eventi. Senza l'Event System, i componenti dell'interfaccia non reagirebbero agli input.

La classe è dotata dei seguenti metodi per lo switch tra scene in cui si ha visuale libera con possibilità di VR e quelle in cui non la si ha; e per abilitare e disabilitare la visualizzazione VR vera e propria:

- `TurnOnMouseInput()`: non fa altro che impostare il flag `XRSetting.enabled` sul valore `false` e disabilitare il GameObject che contiene la componente `_gvrEditorEmulator`. In questo modo, nella versione desktop non sarà possibile emulare il giroscopio, e nella versione mobile la VR sarà disabilitata.

- `TurnOnHybridInput()`: abilita il `GameObject` che contiene la componente `_gvrEditorEmulator`. In questo modo, nella versione desktop sarà possibile emulare il giroscopio.
- `WaitForVisor()`: disabilita l'input sull'interfaccia utente, e abilita il pannello `WearVisorPanel` mostrato a Figura 2.9. Al termine del timer invoca `TurnOnHybridInput()`.
- `SwitchToVR()`: in seguito alla chiamata a `WaitForVisor()`, il metodo verifica che l'utente non abbia premuto sul bottone '*Don't use VR*' effettuando un controllo sul flag `useVR`. Se così fosse, imposta `XRSettings.enabled` a `true` e disabilita il controllo della camera da parte del GVR SDK tramite l'invocazione di `XRDevice.DisableAutoXRCameraTracking(cam, true)`. Questo perchè si è preferito gestire manualmente la rotazione della camera. Per ultimo, imposta il flag `VR_on` sul valore `true`.

All'interno dei metodi `Update()` dello stato *Street View State* e del *Player*, viene invocato l'ultimo metodo contenuto in `VRManager`:

`TryGetCenterEyeNodeStateRotation(out rot).`

Il metodo serve ad ottenere i valori registrati dal giroscopio del dispositivo di esecuzione e viene invocato ad ogni frame solamente se la piattaforma montata su tale dispositivo è Android. Per la rotazione della camera su desktop si sfruttano le funzionalità di `_gvrEditorEmulator`.

- `TryGetCenterEyeNodeStateRotation(out rot)`: riempie una lista di tipo `XRNodeState` con i valori ottenuti dall'invocazione di `InputTracking.GetNodeStates`. Successivamente, tramite una visita alla lista, ricerca al suo interno un valore di `XRNodeState.nodeType` uguale a `XRNode.CenterEye`, per invocare il metodo in esso contenuto `TryGetRotation(out rot)` che imposta il valore di `rot` uguale al valore ottenuto dal giroscopio del device.

### 3.3 Scripts per la UI

#### 3.3.1 SaveStartLocationInput.cs

Si occupa di fare in modo che l'input inserito dall'utente nel campo del punto di partenza venga letto e inviato alla Geocoder API di Mapbox. È assegnato al `GameObject` del campo di testo. Nello specifico contiene:

- Un `listener` per l'evento `onEndEdit` del campo di testo, che chiama il metodo `HandleUserEndInput`, il quale invia la richiesta alla API;
- Un `listener` per l'evento `onValueChanged` del campo di testo, che, chiamando il metodo `HandleUserInputChanging`, rimette a `false` il flag `fromGeocoded`;
- Un `event` di tipo `Action<ForwardGeocodeResponse>` e di nome `OnGeocoderResponse` che delega un altro metodo per la gestione della risposta della API.

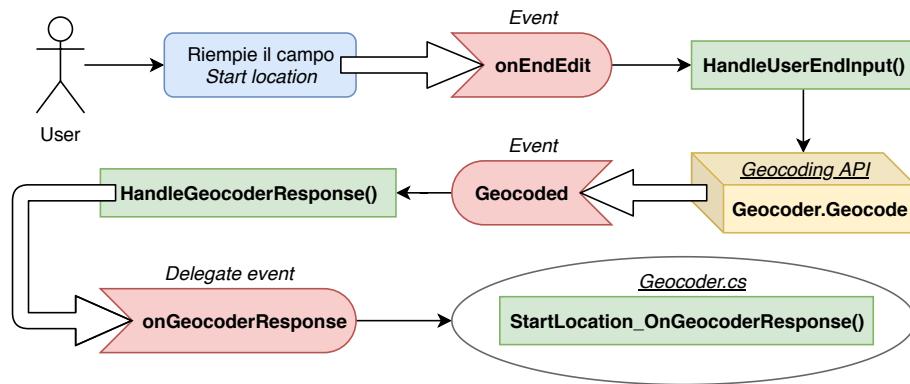


Figura 3.10: Flow Chart della chiamata alla Geocoding API di Mapbox

### 3.3.2 SaveInputTime.cs & SaveInputSpeed.cs

Contengono un metodo che permette di memorizzare gli input utente di *velocità* e *tempo* nella variabile dedicata contenuta in `SimulationStatePattern`. Il metodo viene invocato al verificarsi dell'evento `onEndEdit`.

### 3.3.3 LoadingScreenPanel.cs

Script assegnato al pannello di caricamento. Contiene codice che permette di verificare lo stato di lavoro della mappa, differenziandolo tra *Working* e *Finished*. Nel caso in cui lo stato dovesse passare al valore *Working*, il pannello viene abilitato e quindi mostrato a schermo. Ci sono altri momenti dell'esecuzione in cui il pannello viene abilitato manualmente, ad esempio durante il download delle immagini necessarie alla visualizzazione Street View.

### 3.3.4 WearVisorPanel.cs

Assegnato al pannello che richiede di indossare il visore all'utente, contiene la logica per avviare, resettare, e stoppare il timer.

### 3.3.5 Scripts dei bottoni

Contengono la logica da eseguire al verificarsi del click di ognuno di essi, spesso equivalente ad un cambio di stato o all'invocazione di un metodo esterno. La maggior parte contiene anche, nel metodo `Update()`, i controlli sui flag dai quali dipende l'attivazione dei bottoni.

## 3.4 Scripts di interazione con la mappa

### 3.4.1 CameraHandler.cs e QuadTreeCamMov.cs

Sono gli script che permettono di muovere la visuale posta sopra la mappa; si alternano in base allo stato d'esecuzione in cui ci troviamo. Entrambi gli script funzionano sia con input tramite mouse, sia con input touch.

**QuadTreeCamMov** è attivo nella fase di esecuzione in cui non è ancora stato calcolato il percorso. Tale script viene fornito da Mapbox e permette di muoversi liberamente sulla mappa, con qualità dell'immagine satellitare che cambia dinamicamente in base al valore di zoom attualmente impostato. Il valore di zoom in questo caso è legato alla mappa e non alla camera. Il Tile Provider impostato è *Camera Bounds*, verranno quindi caricate e visualizzate solamente le tiles che rientrano nel campo visivo della camera. Per il movimento della visuale viene calcolato quale sarebbe il nuovo centro della mappa in base all'input ricevuto, che viene aggiornata e centrata in quel punto. È quindi la mappa ad aggiornarsi e non la camera a muoversi.



Figura 3.11: Salto di qualità dell'immagine satellitare

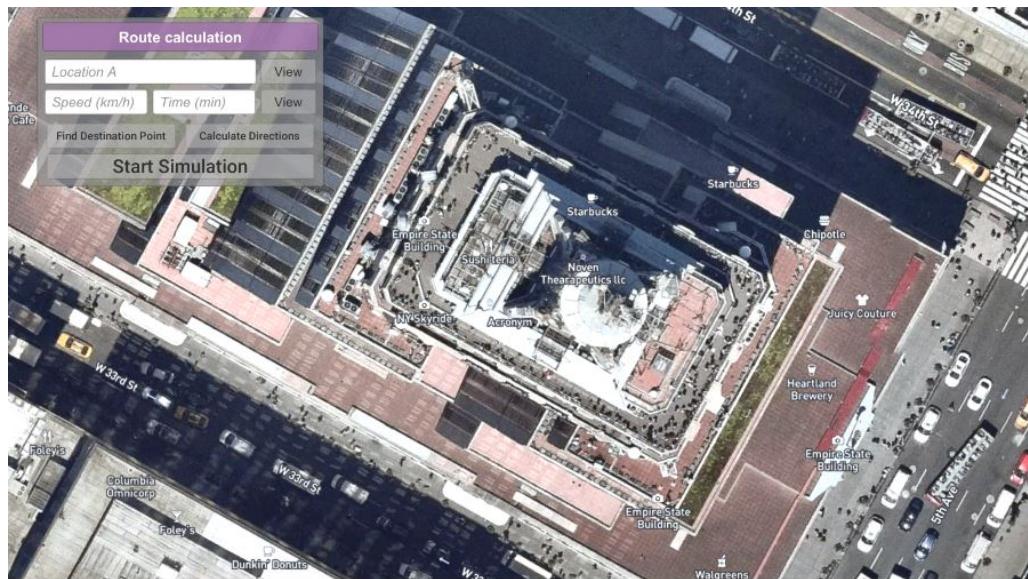
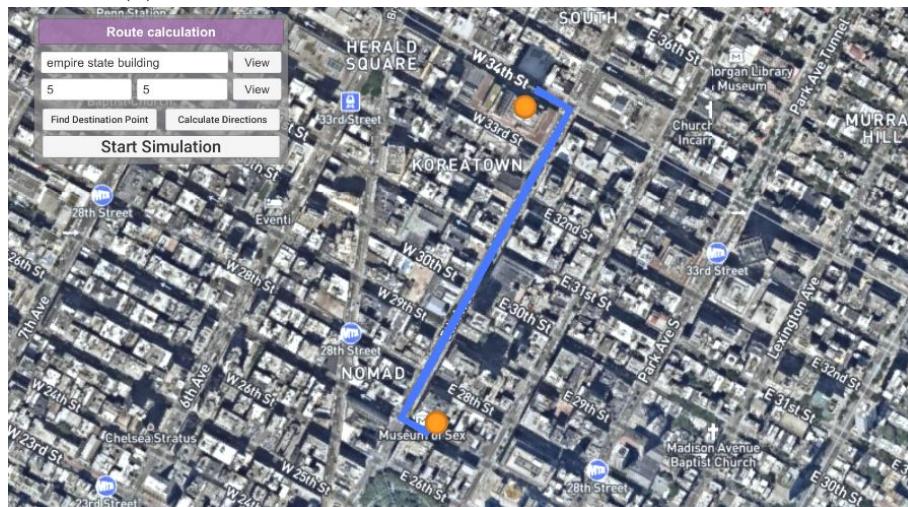


Figura 3.12: Livello di zoom pari a 20, il massimo

**CameraHandler** è attivo nella fase di esecuzione in cui il percorso è stato calcolato ed inserito nella mappa. Il Tile Provider impostato sarà quindi sicuramente *Along Path*, con solamente le tile attorno al percorso caricate e visualizzate. Il movimento della visuale in questo caso è gestito agendo direttamente sulla camera, che viene mossa in base all'input ricevuto. È possibile effettuare lo zoom alterando il valore del parametro `fieldOfView` della camera, che va a ridurre od aumentare la porzione di campo visivo visualizzata.



(a) Zoom minimo della camera con percorso calcolato



(b) Zoom massimo della camera con percorso calcolato

Figura 3.13: La qualità dell'immagine satellitare mostrata nella figura (a) è identica a quella mostrata nella figura (b). È la videocamera a ridurre il suo campo visivo.

### 3.4.2 SpawnOnMap.cs

Anch'esso fornito da Mapbox, permette di istanziare un GameObject in un punto scelto della mappa, e di aggiornarne la posizione in modo da tenerlo centrato in quel punto anche in caso di movimento della visuale o aggiornamento della mappa.

Nel nostro caso è stato utilizzato per posizionare sulla mappa i segnaposto nei punti di partenza e destinazione.

## 3.5 Script del player

### Player.cs

Il GameObject del player è l'unico oggetto attivo che viene istanziato ad un certo punto dell'esecuzione, e non all'avvio. Nello specifico, viene istanziato nel momento in cui la mappa ha terminato di caricare le tiles con livello di zoom indicato dal

parametro `simulationZoom`, in seguito alla pressione del bottone *Start simulation*. Di conseguenza, il metodo `Start()` verrà invocato non appena si sarà avviata la simulazione. L'esecuzione di tale metodo prevede, oltre all'assegnamento delle variabili, l'invocazione di `GetPositions()`, dal quale parte una serie ciclica di invocazioni che permette al player di muoversi all'interno della simulazione. Mostriamo il comportamento della classe sotto forma di flow chart a Figura 3.14.

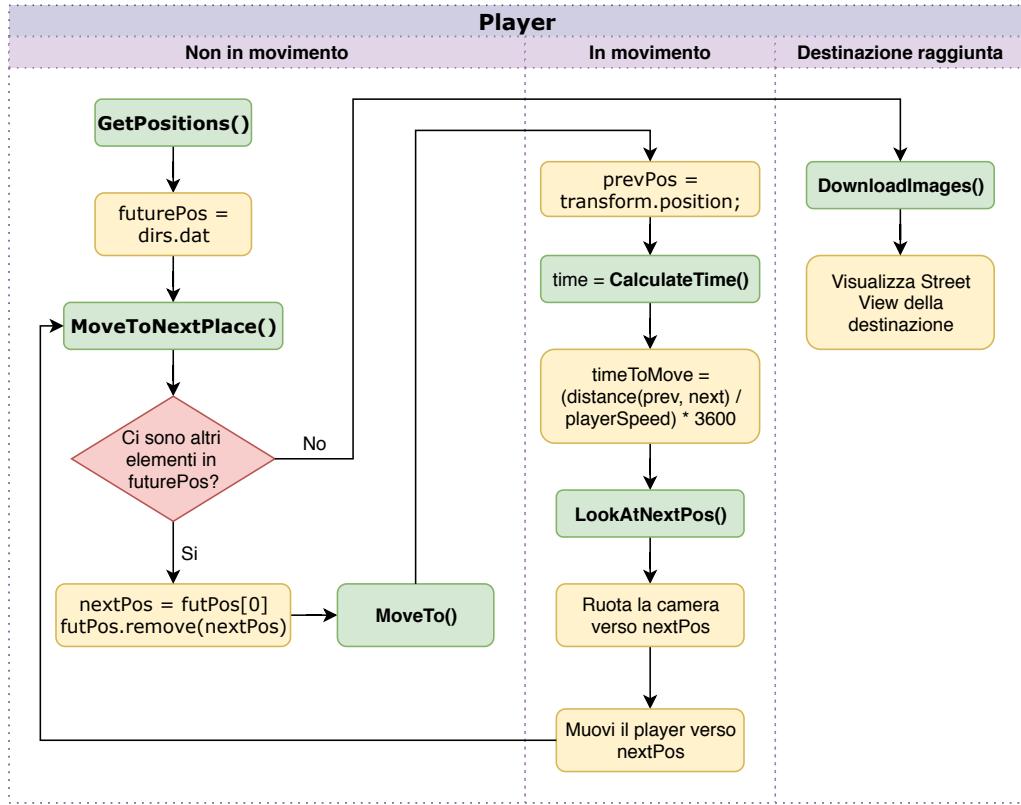


Figura 3.14: Flow Chart d'esecuzione della classe Player.cs

### Update()

La classe `Player.cs` contiene anche il metodo `Update()`, eseguito ad ogni frame durante l'esecuzione. Nel corpo di tale metodo troviamo il codice che permette di ruotare la visuale in base al valore registrato dal giroscopio del dispositivo sul quale si sta eseguendo l'applicativo. Oltre a ciò, è presente il controllo che permette di fermare la simulazione non appena diventa visibile il pannello `WearVisorPanel`, e di riprenderla non appena esso scompare.

## 3.6 Struttura e funzionamento

In questa sezione si cercherà di dare forma a quanto finora descritto, specificando le relazioni tra i vari `GameObject`.

Per facilitare la gestione dell'esecuzione si è realizzata una macchina composta da tre stati, ognuno dei quali prevede l'utilizzo di una camera, una UI e uno script dedicati. Lo switch tra i vari stati è gestito dagli statti stessi, che presentano un metodo `EnterState()` di inizializzazione e dei metodi di uscita contenenti lo switch di stato vero e proprio, oltre alle operazioni di chiusura.

Avviando l'applicazione ci troviamo davanti all'unica scena realizzata, contenente la mappa, gli oggetti utili alla modalità Street View, gli elementi dell'interfaccia grafica e il *SimulationManager*. Quest'ultimo, oltre a tener traccia degli stati e permetterne l'esecuzione, contiene i riferimenti a quasi ogni oggetto e relativi componenti presenti nella scena, quindi le altre classi passeranno attraverso i dati in esso incapsulati per accedere alle informazioni di cui necessitano da parte degli altri moduli.

I GameObjects relativi allo stato *Simulation State* - ovvero la camera dedicata, la relativa interfaccia utente e il player che si muove lungo il percorso - vengono tutti istanziati all'ingresso dello stato stesso, e distrutti all'uscita. Per realizzare questo meccanismo si è creato un *Prefab* - un GameObject *ready-to-use* memorizzato negli Assets - del player, contenente nella sua gerarchia tutti gli elementi citati. Gli elementi dell'interfaccia grafica vengono abilitati e disabilitati per evitare errori da parte dell'utente finale. L'esecuzione è forzata verso una sola direzione.

Entriamo ora nei dettagli dell'esecuzione dei tre stati.

### 3.6.1 NotStartedState

Quando l'esecuzione si trova in questo stato, si ha visuale dall'alto sulla mappa e il menù principale visibile. È possibile scegliere un punto di partenza e calcolare un punto di destinazione in funzione dei parametri inseriti. Alla pressione di uno dei due bottoni *View*, si passa a *StreetViewState*. Alla pressione del bottone *Start Simulation* si passa invece a *WalkingState*.

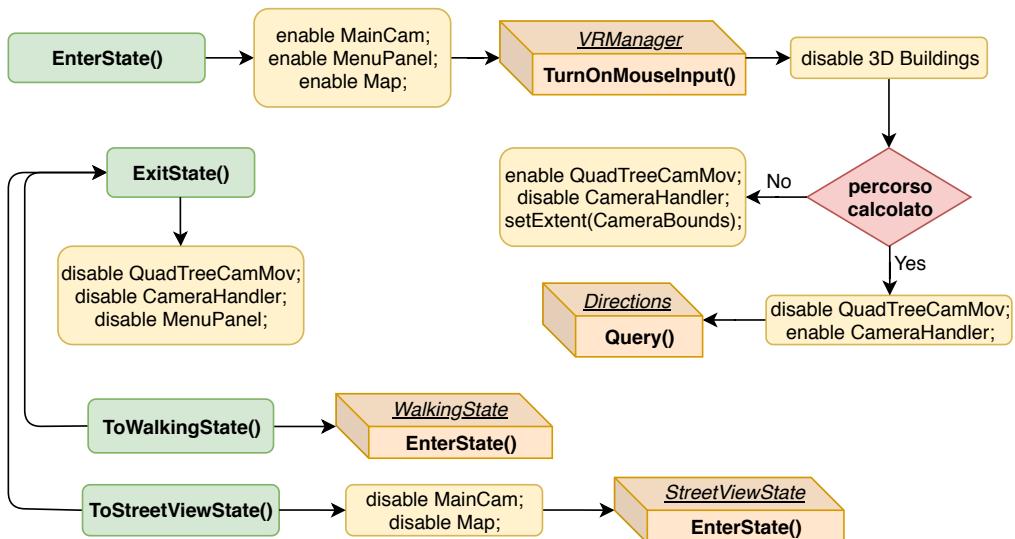


Figura 3.15: Flow chart di NotStartedState

### 3.6.2 StreetViewState

Non appena si effettua richiesta per la visualizzazione Street View di un punto, viene invocato il metodo `StreetViewGenerator.DownloadImages()`, spiegato più dettagliatamente alla Sezione 3.2.5. In seguito alle operazioni effettuate dalla classe appena citata, si entrerà in *StreetViewState*. All'interno di tale stato si visualizzerà la panoramica 360° Street View del luogo scelto, finché non si premerà il bottone *Return*. Si ha inoltre la possibilità di passare alla modalità VR.

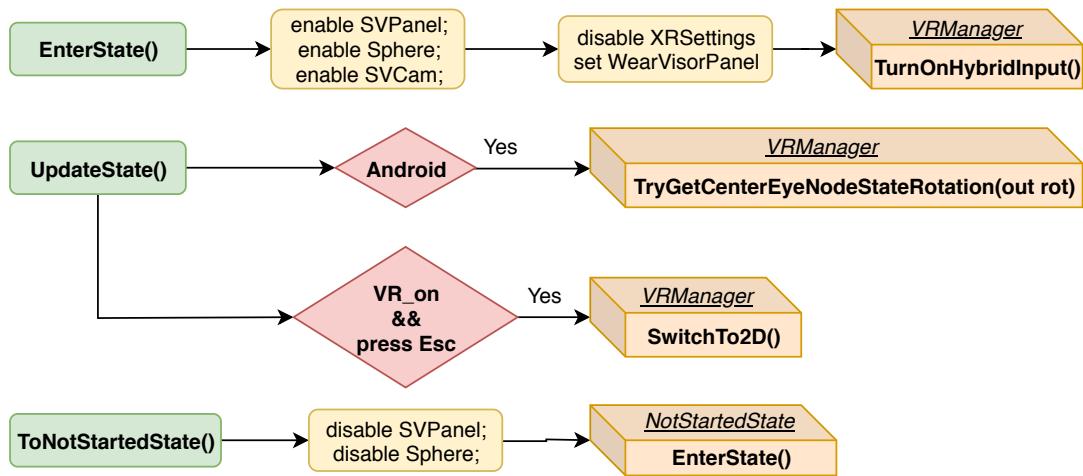


Figura 3.16: Flow chart di StreetViewState

### 3.6.3 WalkingState

Premendo sul bottone `StartSimulation` si entra in `WalkingState`. All'interno di tale stato, bisogna preparare la mappa alla simulazione nel mondo virtuale tridimensionale. Per fare ciò, la prima cosa da fare è impostare lo zoom con il valore del parametro `simulationZoom`, che permette di passare ad un'immagine satellitare di qualità superiore. Successivamente, bisogna ricalcolare il percorso sulla base del nuovo livello di zoom. Non appena il metodo avrà verificato che la mappa ha terminato di caricare, viene abilitata la visualizzazione degli edifici in 3D ed istanziato il GameObject del `Player`.

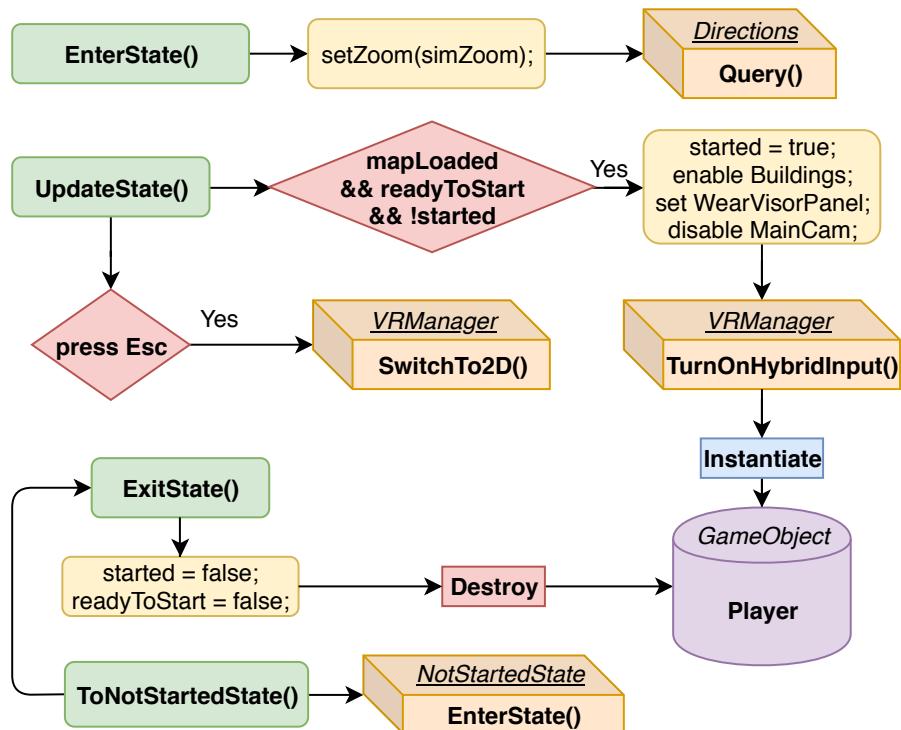


Figura 3.17: Flow chart di WalkingState

### 3.6.4 Diagrammi UML

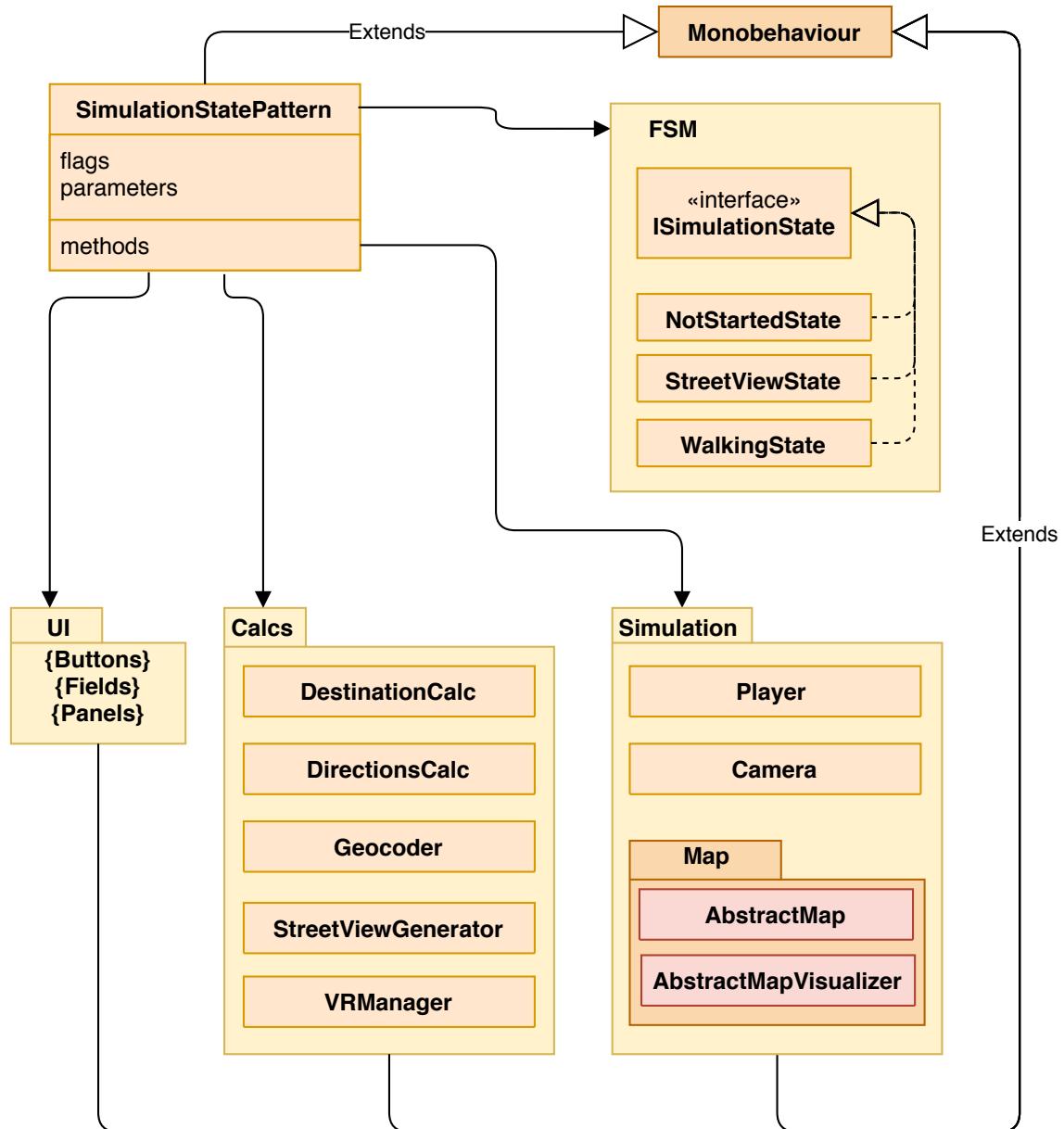
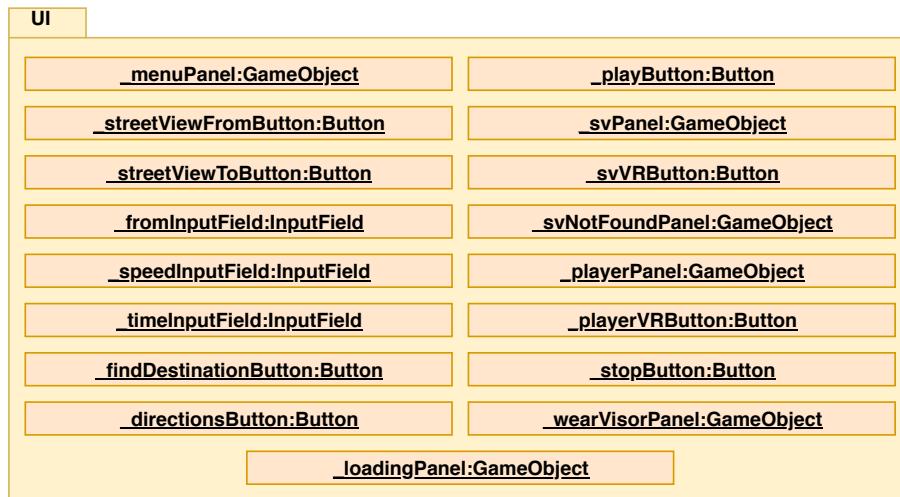
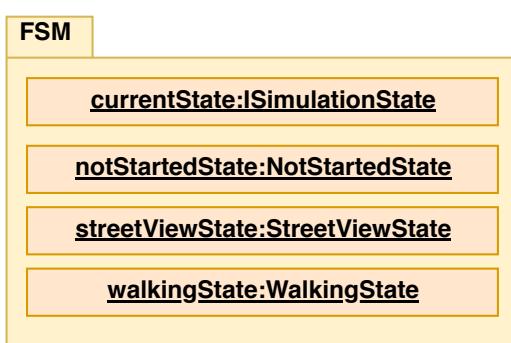


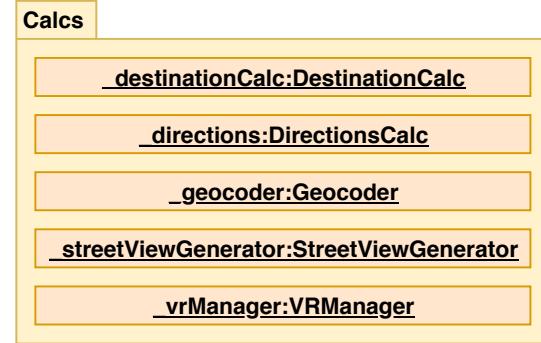
Figura 3.18: UML delle classi generico di MoveUp!VR. **SimulationStatePattern** contiene un riferimento ad un'istanza di ogni classe mostrata.



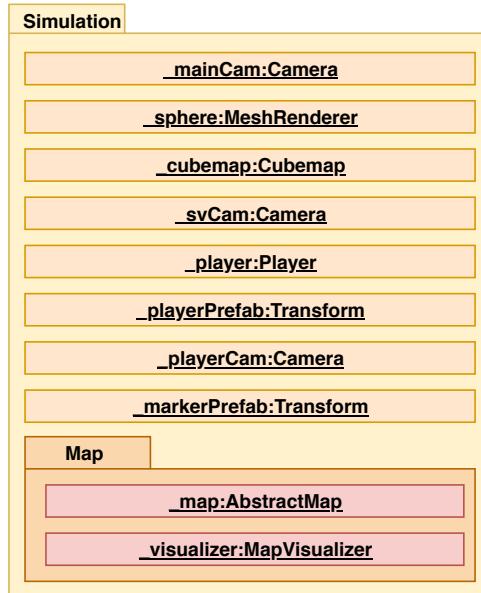
(a) Istanze della UI referenziate



(b) Istanze della FSM referenziate



(c) Istanze di calcolo referenziate



(d) Istanze della simulazione referenziate

Figura 3.19: Tutte le istanze referenziate da SimulationStatePattern

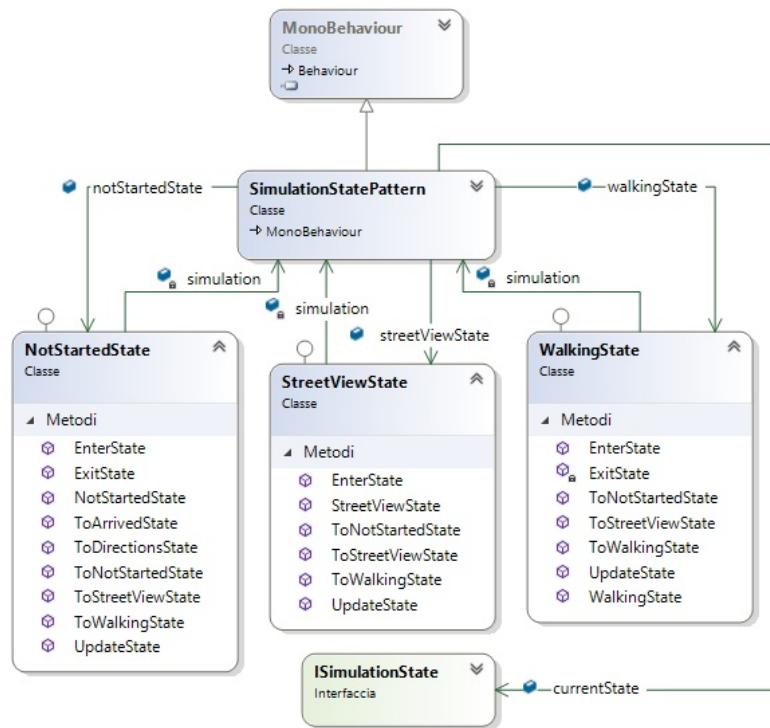


Figura 3.20: Estratto UML della parte dedicata alla FSM

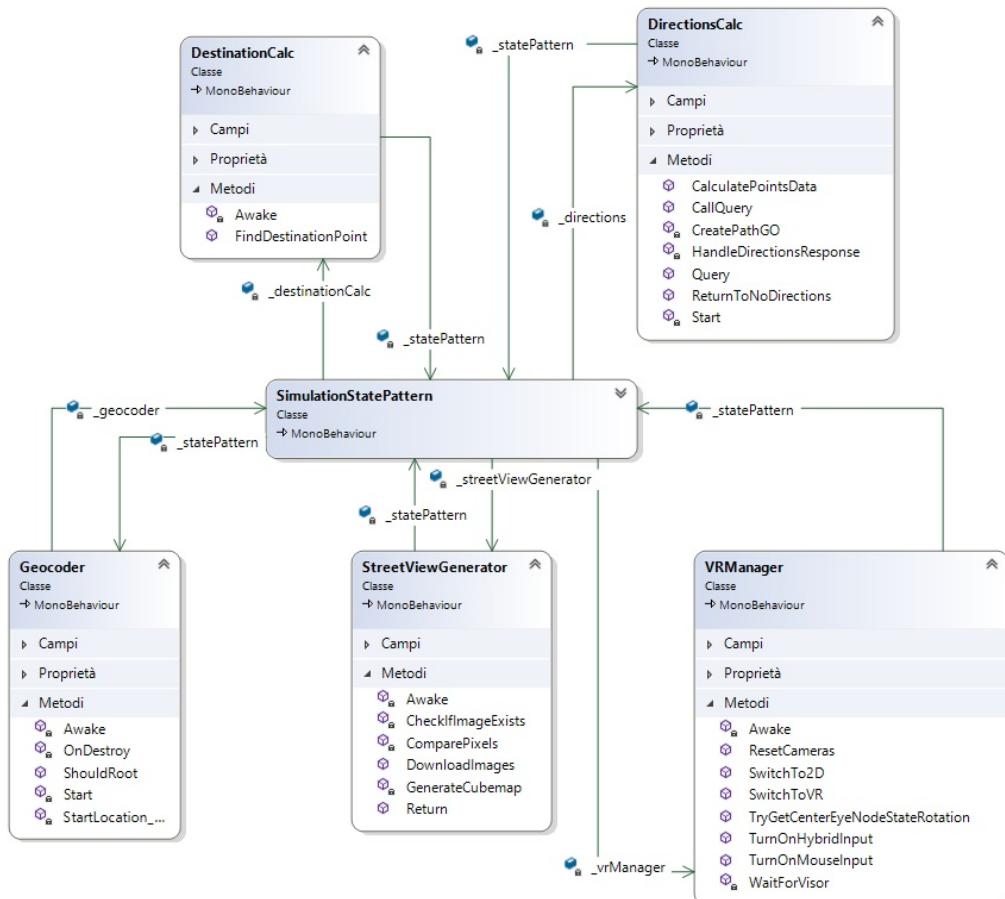


Figura 3.21: Estratto UML della parte dedicata ai calcoli

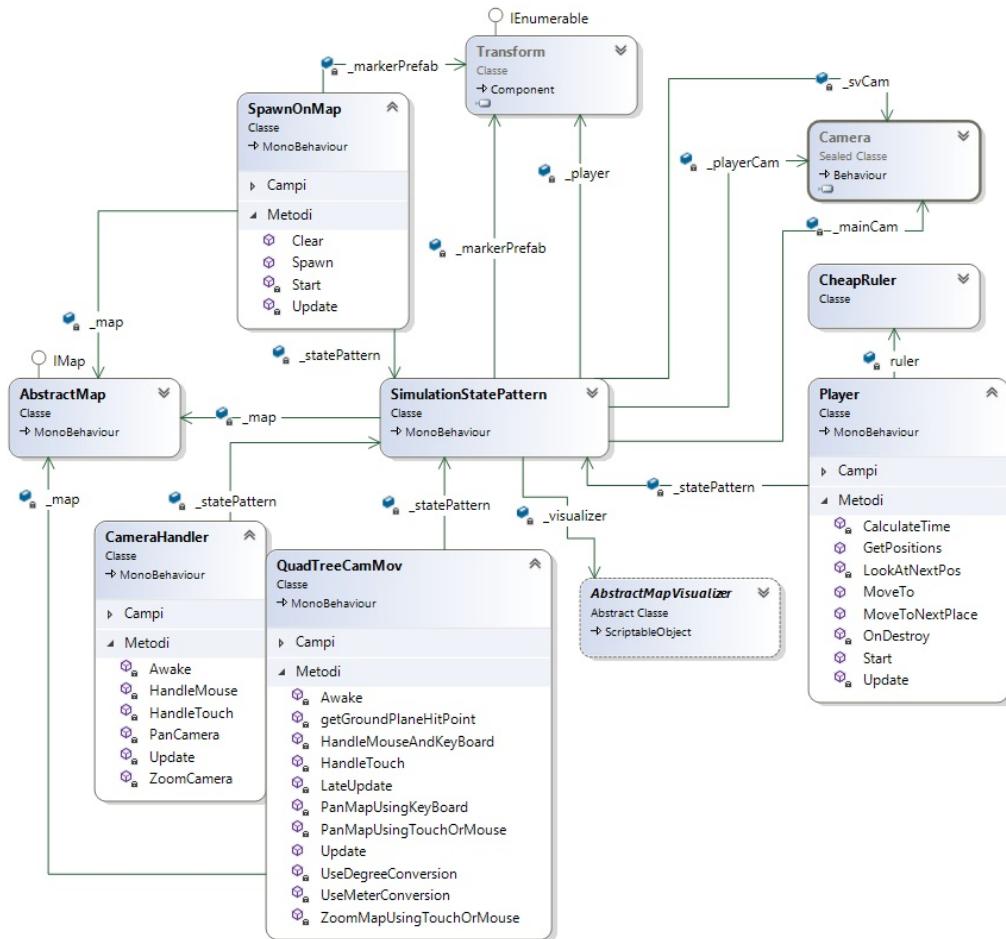


Figura 3.22: Estratto UML della parte dedicata alla simulazione

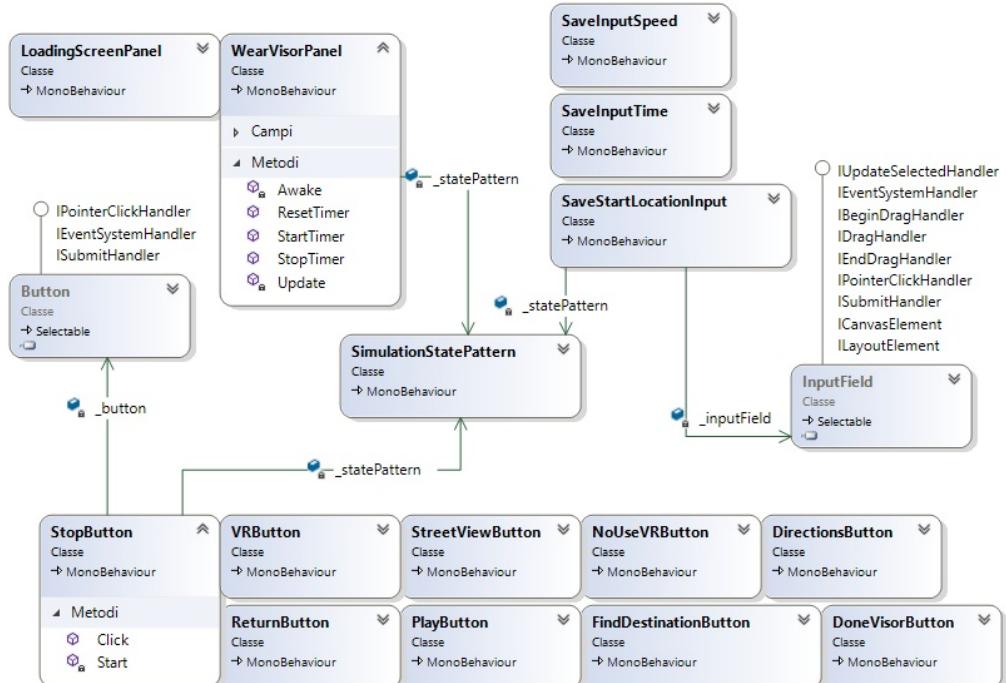


Figura 3.23: Estratto UML della parte dedicata alla UI

# Capitolo 4

## Caso di studio

In questo capitolo verrà presentato il contesto migliore per sfruttare al massimo le funzionalità dell'app. Verranno sottolineati i problemi presenti e i risultati ottenuti sulle performance con il device utilizzato per i test, specificando quali adattamenti sulla qualità sono stati necessari e quali altri migliorerebbero le prestazioni.

### 4.1 Ambienti urbani

Il miglior contesto in cui effettuare simulazioni è in ambienti urbani molto densi di edifici. Questa caratteristica permette di sfruttare a pieno le potenzialità dell'app, che ricordiamo non godere della generazione procedurale degli altri elementi che compongono il panorama, come ad esempio le automobili e gli alberi. È possibile notare nell'esempio a Figura 4.1 che le uniche caratteristiche riprodotte in 3D sono il terreno e i pochi edifici visibili appena al di fuori del Central Park.

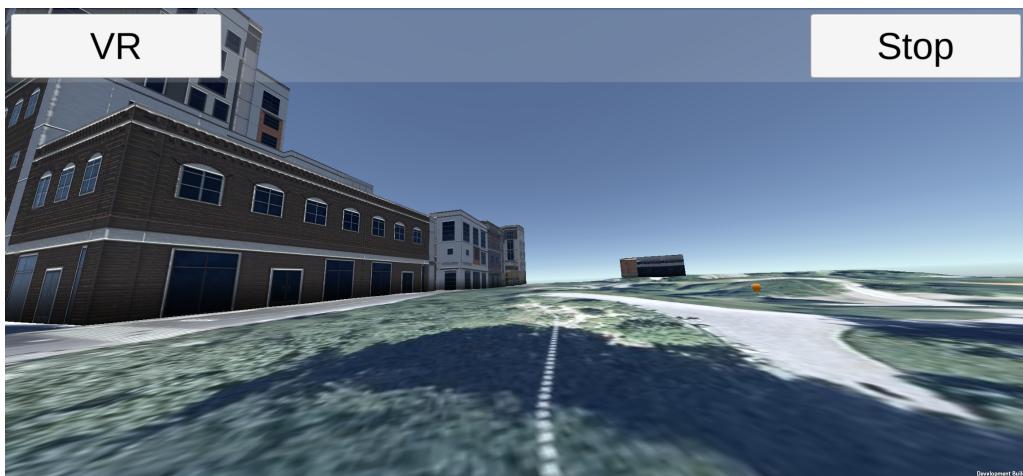


Figura 4.1: Esempio in area non urbana. Central Park - New York

### 4.2 Analisi delle performance

Per il testing dell'applicazione si è utilizzato il seguente dispositivo Android:

Xiaomi - Redmi Note 7

Processore: Octa-core 2GHz

Display: 6.3" - 2340x1080

Memoria RAM: 3GB

Ovviamente, la numerosità degli elementi modellati influisce sulle performance, aumentando il carico computazionale e l'utilizzo di memoria. Infatti, ogni volta che un edificio si trova in una tile da caricare, entrano in gioco tutti gli elementi che lo compongono, ovvero *texture* e *mesh*, che vanno caricate e visualizzate nello spazio virtuale.

Questo fatto porta a non pochi problemi sull'esecuzione *mobile*, causando il *crash* dell'applicazione nel caso in cui si tenti di generare percorsi troppo lunghi e densamente popolati di edifici. Il crash è dovuto all'eccessivo utilizzo di memoria RAM del dispositivo, che supera la quantità disponibile.

### 4.2.1 Il Profiler di Unity

Il Profiler di Unity è uno strumento utile per ottenere informazioni sulle prestazioni relative alla tua applicazione. Puoi collegarlo ai dispositivi della tua rete o ai dispositivi collegati al tuo computer per testare il modo in cui l'applicazione viene eseguita sulla piattaforma di rilascio prevista. Puoi anche eseguirlo nell'editor per ottenere una panoramica dell'allocazione delle risorse mentre stai sviluppando la tua applicazione.

Il Profiler raccoglie e visualizza i dati sulle prestazioni dell'applicazione in aree quali CPU, memoria, renderer e audio. Serve per identificare i punti su cui iterare per il miglioramento delle prestazioni. Visualizza i risultati in una serie di grafici, in modo da poter individuare dove si verificano picchi nelle prestazioni. [13]

#### Memory Profiler

Visualizza i contatori che rappresentano il totale della memoria allocata dalla applicazione. È possibile utilizzarlo per visualizzare informazioni come il numero di oggetti caricati divisi per categoria e la memoria che occupano in totale. Puoi anche vedere il numero di allocazioni del Garbage Collector per ogni frame.

Quando si profila l'applicazione, le informazioni nel modulo di memoria indicano l'utilizzo della memoria nell'Editor. Questi numeri sono generalmente più grandi rispetto a quando sono in esecuzione sulla piattaforma di rilascio, perché l'esecuzione di Unity Editor utilizza oggetti specifici che occupano memoria, e la finestra dell'Editor stesso utilizza memoria aggiuntiva. Il Profiler include nel conteggio anche la memoria che utilizza l'editor poiché Unity non può separarla in modo chiaro dalla memoria utilizzata dalla modalità Play.

Per numeri più precisi sull'utilizzo della memoria, bisogna collegare il Profiler al dispositivo su cui l'app è in esecuzione, tramite il menù dedicato. Ciò consente di vedere l'utilizzo di memoria effettivo sul dispositivo di destinazione. [14]

### 4.2.2 Risultati ottenuti

Non appena si sono verificati i primi crash dell'applicazione, essi hanno continuato a presentarsi nel momento in cui viene caricata la simulazione con gli edifici. Ci si è concentrati sull'analisi dei risultati prodotti dal Profiler per capire da cosa fossero dovuti, e si è notato fin da subito che il dispositivo aveva gran parte della memoria occupata già all'avvio dell'applicazione. I test presentati sono stati effettuati a partire dal World Trade Center di New York con velocità impostata a 10km/h e tempo impostato a 10 minuti.



Figura 4.2: Risultati del profiler all'avvio dell'applicazione su dispositivo mobile. Total System Memory Usage: 2.63GB

Confrontando il risultato con quello ottenuto eseguendo l'applicativo dall'editor di Unity, è stato subito chiaro che purtroppo una porzione significativa della memoria dello smartphone era occupata dal suo stesso sistema.

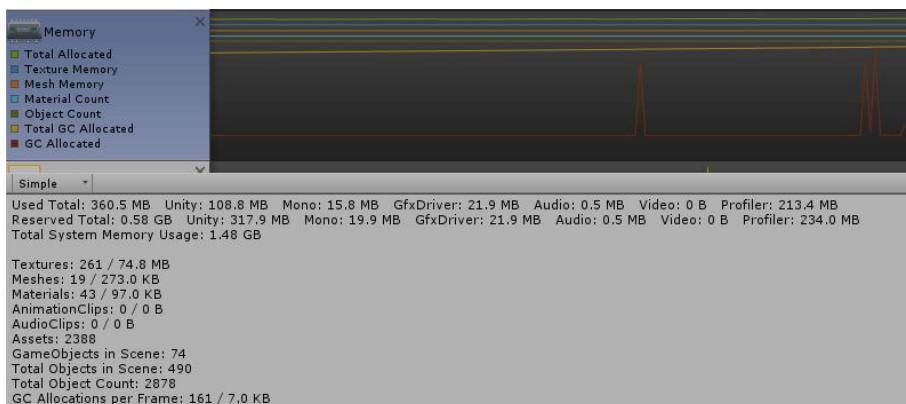


Figura 4.3: Risultati del profiler all'avvio dell'applicazione dentro l'editor di Unity. Total System Memory Usage: 1.48GB

Proseguendo con i test, si è verificata la differenza di utilizzo di memoria tra i momenti prima, durante e dopo il caricamento della simulazione all'interno dell'editor.

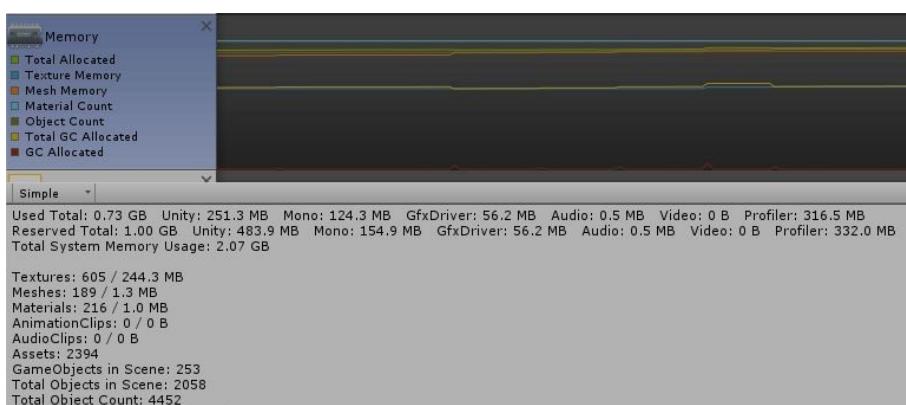


Figura 4.4: Risultati del profiler durante il caricamento della simulazione dentro l'editor di Unity. Total System Memory Usage: 2.07GB

## Capitolo 4. Caso di studio

---

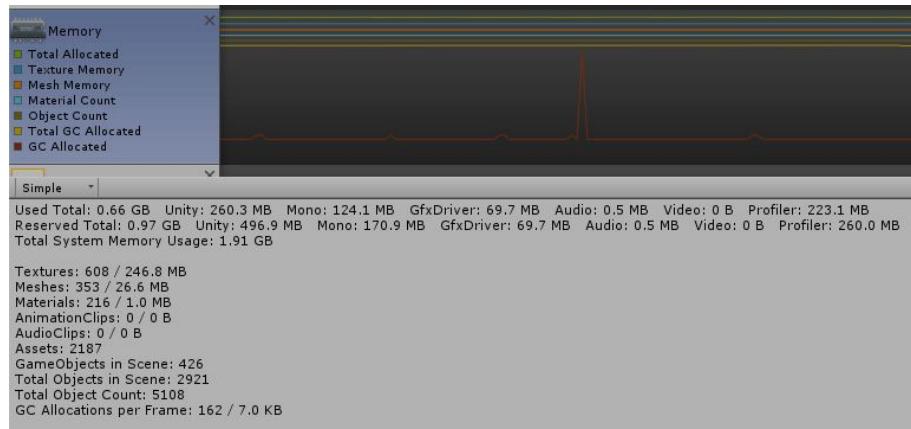


Figura 4.5: Risultati del profiler dopo il caricamento della simulazione dentro l'editor di Unity. Total System Memory Usage: 1.91GB

In questo modo si è venuti a conoscenza della presenza di picchi di utilizzo di memoria durante il caricamento della simulazione.

Effettuando dei nuovi test con velocità pari a 10km/h e tempo pari a 30 minuti, si sono ottenuti i risultati presentati a partire da Figura 4.6.



Figura 4.6: Risultati del profiler durante il caricamento della simulazione dentro l'editor di Unity. Total System Memory Usage: 2.98GB

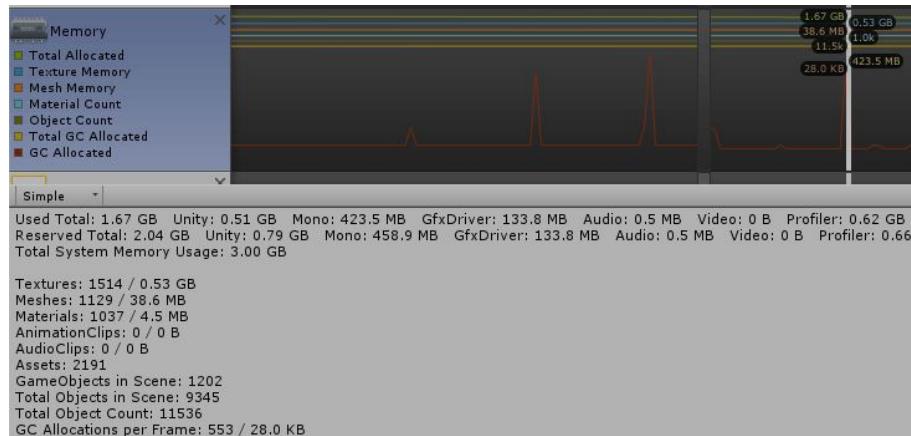


Figura 4.7: Risultati del profiler subito dopo il caricamento della simulazione dentro l'editor di Unity, nel punto di picco. Total System Memory Usage: 3.00GB

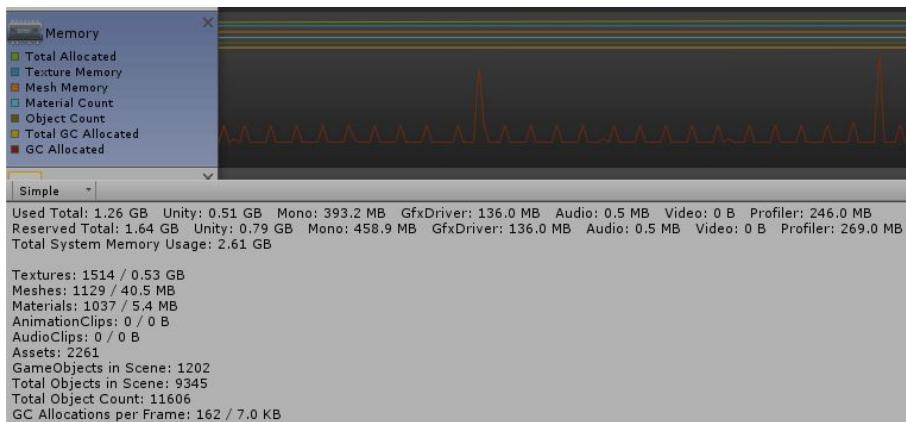


Figura 4.8: Risultati del profiler a simulazione avviata dentro l'editor di Unity. Total System Memory Usage: 2.61GB

Diventa quindi evidente il motivo del crash dell'applicazione eseguita su dispositivo mobile con parametri troppo alti. Se all'interno dell'editor si raggiungono i 3GB di memoria utilizzata partendo da circa 1.5GB, su mobile, dove partivamo con 2.63GB utilizzati, supereremmo sicuramente la memoria massima del sistema.

Purtroppo non si è riusciti a catturare i picchi durante il caricamento delle simulazioni più pesanti su mobile, a causa del fatto che, con l'alto utilizzo di risorse, il profiler non riesce a registrare alcuna informazione fino a quando l'applicativo non torna ad avere un *frame rate* (numero di aggiornamenti al secondo) dal valore accettabile.

Anche testando con velocità impostata a 10km/h e tempo a 20minuti, l'applicativo eseguito su mobile va incontro ad un crash.

### 4.2.3 Adattamento applicato

Premesso ciò, le soluzioni adottabili potrebbero essere due: alleggerire la simulazione oppure strutturare meglio il caricamento in modo da evitare concentrazioni di carico computazionale e di memoria utilizzata. Per arrivare velocemente ad una soluzione accettabile, si è prima tentato di alleggerire la simulazione.

Le misure adottate sono state le seguenti:

- **Modellazione degli edifici:** Mapbox offre diverse modalità per rappresentare gli edifici sulla mappa, sia in 2D che in 3D. Le opzioni per la visualizzazione 3D comprendono la scelta di renderizzare solamente i lati oppure anche i tetti. Inizialmente gli edifici venivano generati nella loro completezza, ma per l'ottimizzazione si è passati a generarne solo i lati.

Un'altra caratteristica importante della modellazione degli edifici riguarda l'impostazione 'Combine Meshes', che si occupa di unire le diverse mesh degli edifici, raggruppandole per tile di appartenenza. In questo modo si avranno meno mesh, seppur più elaborate.

Gli edifici non sono dotati di *collider* per alleggerire ulteriormente e non mettere in gioco elaborazioni del motore fisico di Unity. Inoltre, per il nostro scopo, sarebbe stato inutile.

- **Caratteristiche degli edifici:** Mapbox mette a disposizione anche un menù per aggiungere dei '*Behaviour Modifier*' agli edifici, che altro non sono che script che agiscono sulle loro caratteristiche. Se si vuole agire sui singoli edifici, bisogna selezionare l'opzione 'Buildings With Unique ID'. Nel nostro caso l'opzione è stata disabilitata.
- **Qualità dell'immagine satellitare:** per quanto riguarda la memoria occupata dalle immagini satellitari, si è agito sull'opzione '*Use compression*' offerta da Mapbox.

Con i cambiamenti appena presentati, si è riusciti ad eseguire simulazioni più lunghe, seppur sempre brevi. Testando nuovamente con velocità impostata a 10km/h e tempo a 20minuti, l'applicativo andava ancora incontro a situazioni di crash.

A causa del poco tempo rimanente per lavorare al progetto, si è deciso di mettere da parte l'ottimizzazione per terminare le altre funzionalità che dovevano ancora essere implementate. In conclusione è possibile effettuare simulazione dalla durata minore o uguale a 10 minuti, con velocità minore o uguale a 10km/h.

#### 4.2.4 Altri adattamenti

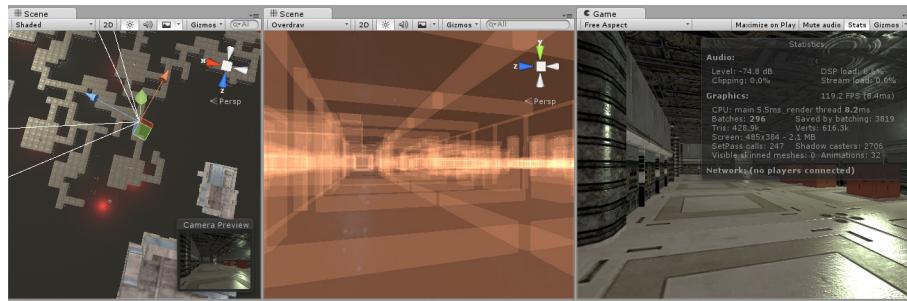
##### AlongPath & NearPlayer & CameraBounds TileProvider

Una soluzione più significativa sarebbe sicuramente quella di scrivere un nuovo Tile Provider, che racchiuda assieme le funzionalità di *Along Path*, *Around Transform* e *Camera Bounds*. In questo modo, in memoria verrebbero caricate solo le porzioni di mappa utili alla visualizzazione in quell'istante, e non utili ad effettuare tutta la simulazione, come succede ora utilizzando solo *Along Path*. Anche lo stesso Along Path si potrebbe migliorare alleggerendo l'algoritmo.

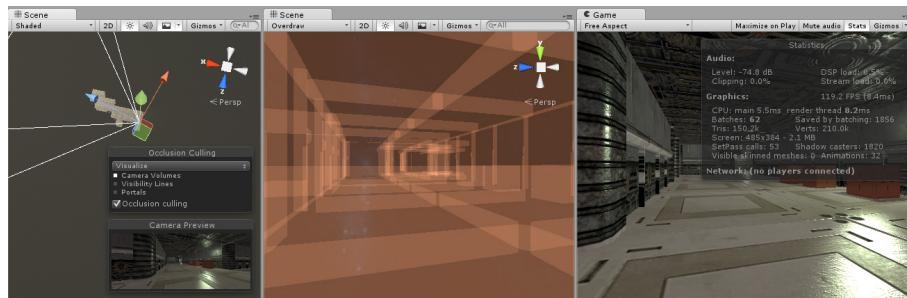
##### Occlusion Culling & Level of Details

Esistono due tecniche che permettono l'ottimizzazione in ambienti vasti e ricchi di entità: Occlusion Culling (OC) e Level of Details (LoD).

La **Occlusion Culling** è una funzione che disabilita il rendering degli oggetti quando non sono attualmente visti dalla telecamera poiché sono oscurati (occlusi) da altri oggetti. Ciò non si verifica automaticamente nella computer grafica 3D poiché la maggior parte delle volte gli oggetti più lontani dalla fotocamera vengono disegnati per primi e gli oggetti più vicini vengono disegnati sopra di essi (questa tecnica è detta *overdraw*).



(a) Overdraw senza OC attivata



(b) Overdraw con OC attivata

Figura 4.9

La **Frustum Culling**, invece, disabilita solo i renderer per gli oggetti che si trovano al di fuori dell'area di visualizzazione della videocamera, ma non disabilita nulla di ciò che è nascosto alla vista per overdraw.



(a) OC disabilitata

(b) Frustum Culling abilitata

Figura 4.10: Differenze tra culling disabilitato e Frustum culling

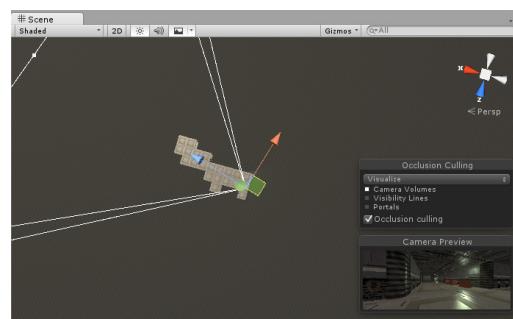


Figura 4.11: Occlusion Culling abilitata completamente

Quando un GameObject nella scena è lontano dalla videocamera, non puoi vedere molti dettagli, rispetto a quando esso si trova vicino alla videocamera. Anche se non è possibile vedere i dettagli di un GameObject lontano, Unity lo renderizza usando lo stesso numero di triangoli a entrambe le distanze.

Per ottimizzare il rendering, è possibile utilizzare la tecnica **LOD (Level Of Detail)**. Essa permette di ridurre il numero di triangoli renderizzati per un GameObject all'aumentare della sua distanza dalla telecamera. Si usano diverse Mesh e, facoltativamente, un Billboard Asset, l'importante è che rappresentino tutti lo stesso GameObject con dettagli decrescenti nella geometria.



(a) Esempio di LOD0 di un GameObject



(b) Esempio di LOD1 di un GameObject

Figura 4.12: Differenza tra due diversi livelli di LOD

Ognuna delle Mesh contiene un componente Renderer Mesh e rappresenta un "livello LOD". Il Billboard Asset ha un componente Renderer Billboard e rappresenta un "livello LOD Billboard"; essi tornano utili a grandi distanze, e per comprenderne al meglio la funzionalità osserviamo la Figura 4.13.

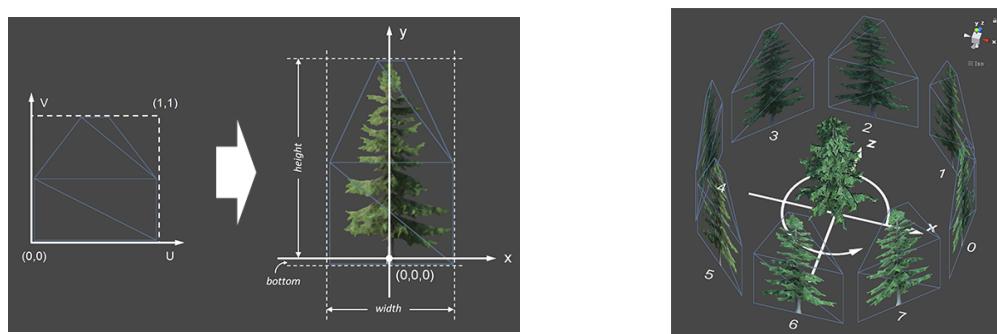


Figura 4.13: Come viene rappresentato un oggetto tramite billboard assets

# Capitolo 5

## Conclusioni

In questo ultimo capitolo vedremo i risultati ottenuti in conclusione del periodo del mio lavoro di stage. Verranno infine presentati i possibili sviluppi futuri e alternativi.

### 5.1 Risultati

Ricapitolando, si è sviluppata un'applicazione in grado di proporre e simulare un percorso calcolato in funzione di alcuni parametri, all'interno del mondo virtuale generato proceduralmente nello spazio tridimensionale. L'utente ha la possibilità di visualizzare il punto di partenza e di arrivo in modalità Street View, e di attivare la modalità Virtual Reality. Il difetto principale del software riguarda il peso computazionale ed in memoria, che porta l'app a situazioni di crash nel caso in cui si cerchi di simulare percorsi troppo lunghi, che includono al loro interno troppe tiles densamente popolate da edifici.

### 5.2 Sviluppi futuri

In questa sezione vedremo quali funzionalità si sarebbero potute aggiungere con più tempo a disposizione.

#### Resa più realistica

Il supporto di un grafico in un progetto di simulazione tridimensionale è indispensabile se si vuole ottenere un effetto il più realistico possibile.

All'interno del nostro progetto la strada viene raffigurata tramite un'immagine satellitare del mondo in quel punto, con relativi difetti, come ad esempio le auto e gli alberi raffigurati appiattiti a terra. Gli edifici sfruttano invece uno dei pacchetti di texture forniti da Mapbox, per la precisione il pacchetto denominato '*realistic*'.

Con l'aiuto di un grafico nel team di sviluppo, si sarebbero potuti ottenere risultati di gran lunga migliori, con texture personalizzate, mesh degli edifici principali costruite manualmente con precisione, ed effetti di luce che aumentino il realismo.



Figura 5.1: Città realistica creata dal team di Mapbox. Per creare questi effetti, hanno aggiunto il supporto alla pipeline di rendering basata sulla fisica di Unity. Grazie ad essa è possibile simulare il modo in cui i materiali reagiscono alla luce nel mondo reale. La nuova pipeline consente di utilizzare *albedo*, *metallic roughness*, *normal*, e *occlusion maps*.

### Punti di interesse

Con le informazioni offerte da Mapbox è possibile implementare l'integrazione dei punti di interesse nel mondo virtuale, visualizzandoli durante la simulazione, ed eventualmente offrendo la possibilità di interagirci. È anche possibile implementare un algoritmo migliore per la proposta del punto di destinazione, includendo nella ricerca la possibilità di passare per punti di interesse di un determinato tipo.

### Generazione procedurale panorama

Con il termine 'panorama' si intendono tutti quegli elementi che contribuiscono a formare l'ambiente nella realtà, come alberi, piante, automobili, panchine, oggetti decorativi, *eccetera*. Alcuni di questi elementi sono generabili proceduralmente in determinate aree della mappa, sfruttando i tag che Mapbox associa ad aree del mondo reale. 'Landscape', ad esempio, è un tag che indica una distesa di erba, in cui è possibile generare alberi.

Con le informazioni sul traffico è invece possibile generare proceduralmente le automobili presenti in strada, con un numero di mezzi proporzionato a quello reale.

Purtroppo Mapbox, rispetto ad altri servizi concorrenti, non ha algoritmi di generazione del panorama automatici.

### Parametri aggiuntivi per la proposta del percorso

Un'altra funzionalità implementabile prevede nuovi parametri per la proposta dello itinerario, come ad esempio:

- più livelli di velocità da alternare;
- numero di pause previste durante la corsa;
- punti attraverso i quali passare necessariamente;

Si potrebbe anche progettare un sistema che non proponga solo un itinerario, ma che, tenendo conto delle esperienze precedenti dell'utente, gli proponga, giorno per giorno, un intero percorso sportivo, con selezione dei parametri automatica in base ai risultati precedentemente raggiunti dall'utente in questione.

### Realtà Aumentata

Sarebbe bello dare all'utente la possibilità di percorrere l'itinerario concretamente in realtà aumentata, una volta simulato in realtà virtuale. Si potrebbero implementare moltissime nuove funzionalità di supporto *real-time* all'attività sportiva, tramite stimoli visivi e audio: un esempio banale potrebbe essere un sistema a punti ottenibili completando degli obiettivi, come percorrere 5km in una sola sessione.



Figura 5.2: Esempio di interfaccia in augmented reality

### Sistema a obiettivi di gioco

Uno stimolo aggiuntivo molto potente potrebbe essere l'inserimento di una serie di obiettivi di gioco che aggiungano all'attività motoria un comparto ludico. Un esempio di gioco che è riuscito a far svolgere attività motorie a molti ragazzi è Pokémon GO, che obbliga i giocatori a ritrovarsi in punti di interesse per completare gli obiettivi del gioco stesso, ovvero catturare Pokémon e conquistare le palestre combattendo contro gli altri giocatori. [7]

## 5.3 Sviluppi alternativi

L'app può essere considerata un punto di partenza per sviluppare software simili in ambiti differenti rispetto quello della sedentarietà e dell'allenamento tramite la corsa all'aria aperta.

### Altre simulazioni VR per la salute

La realtà aumentata potrebbe rappresentare il futuro per venire a capo di problemi psicologici fino ad ora difficili da alleviare. Sono state studiate moltissime forme di fobie nella psicologia umana, e gran parte di esse potrebbero migliorare con una

terapia supportata dalla VR. Un esempio di società che sviluppa simulatori di questo tipo è C2Care [15].

Un’importante applicazione delle simulazioni della realtà si trova nell’ambito della guida di mezzi. Infatti, diversi individui nel mondo presentano problemi nel mettersi alla guida di un’automobile, a causa dell’insicurezza di esserne in grado. Spesso la paura è quella di danneggiare la salute propria e altrui, senza avere la capacità di evitarlo. Altri casi si presentano in seguito ad incidenti stradali.

Una soluzione a questo problema potrebbe essere un percorso psicologico supportato dalla realtà virtuale, che permetta al paziente di testare in prima persona i pericoli della strada all’interno di un mondo virtuale sicuro.

Esistono già diverse ricerche riguardo l’argomento, ma ancora non è presente un simulatore di riferimento, né certezze scientifiche sugli effetti benefici. [16] [17]



Figura 5.3: C2Drive sviluppata da C2Care



Figura 5.4: C2Drive sviluppata da C2Care

# Riferimenti bibliografici

- [1] Filippo Ongaro. *Sedentarietà: effetti negativi sul nostro corpo*. URL: <https://www.filippo-ongaro.com/blog/effetti-negativi-della-sedentarieta-sul-nostro-corpo>.
- [2] Regina Guthold et al. “Worldwide trends in insufficient physical activity from 2001 to 2016: a pooled analysis of 358 population-based surveys with 1·9 million participants”. In: *The Lancet Global Health* (2018). DOI: 10.1016/S2214-109X(18)30357-7.
- [3] World Health Organization. *Global recommendations on physical activity for health*. <https://www.who.int/dietphysicalactivity/publications/9789241599979/en/>. 2010.
- [4] Maria Brilaki. *Fitness Reloaded*. URL: <https://fitnessreloaded.com/>.
- [5] Andrea Infantino. “Visualizzazione interattiva in street view di percorsi su mappe tramite applicazione Android”. Tesi di Laurea triennale. Università degli Studi di Milano - Bicocca, 2019.
- [6] Ruiz JG et al. “Using anthropomorphic avatars resembling sedentary older individuals as models to enhance self-efficacy and adherence to physical activity: psychophysiological correlates.” In: *Studies in Health Technology and Informatics* (2012). URL: <https://europepmc.org/article/med/22357026>.
- [7] Takahiro A. Kato et al. “Can “Pokémon GO” rescue shut-ins (hikikomori) from their isolated world?” In: (2016). DOI: 10.1111/pcn.12481.
- [8] Darren E. R. Warburton. “The Health Benefits of Active Gaming: Separating the Myths from the Virtual Reality”. In: *Current Cardiovascular Risk Reports* (2013). DOI: 10.1007/s12170-013-0322-0.
- [9] Soojeong Yoo and Judy Kay. “VRun: running-in-place virtual reality exergame”. In: *OzCHI ’16: Proceedings of the 28th Australian Conference on Computer-Human Interaction* (2016). DOI: 10.1145/3010915.3010987.
- [10] Konstantinos Kazakos et al. “NEAT-o-Games: novel mobile gaming versus modern sedentary lifestyle”. In: *MobileHCI ’08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services* (2008). DOI: 10.1145/1409240.1409333.
- [11] *ClassVR: Virtual Reality for the classroom*. URL: <https://www.classvr.com/>.
- [12] *JAPAN TO OFFER NEW VIRTUAL HIGH SCHOOL EXPERIENCE!* URL: <https://www.tokyopop.com/blog/japan-to-offer-new-virtual-high-school-experience>.

- [13] Unity manual. *Profiler overview*. URL: <https://docs.unity3d.com/Manual/Profiler.html>.
- [14] Unity manual. *Memory Profiler module*. URL: <https://docs.unity3d.com/Manual/ProfilerMemory.html>.
- [15] *c2care: Virtual reality in the service of health. Applications for the treatment of phobias, addictions, eating disorders, depression and functional disorders*. URL: <https://www.c2.care/en/c2phobia-treating-phobias-in-virtual-reality/>.
- [16] Schoch Stefanie et al. *Driving simulation as virtual reality exposure therapy to rehabilitate patients with driving fear after traffic accidents*. Aug. 2019. URL: [https://www.researchgate.net/publication/335029856\\_DRIVING\\_SIMULATION\\_AS\\_VIRTUAL\\_REALITY\\_EXPOSURE\\_THERAPY\\_TO\\_REHABILITATE\\_PATIENTS\\_WITH\\_DRIVING\\_FEAR\\_AFTER\\_TRAFFIC\\_ACCIDENTS](https://www.researchgate.net/publication/335029856_DRIVING_SIMULATION_AS_VIRTUAL_REALITY_EXPOSURE_THERAPY_TO_REHABILITATE_PATIENTS_WITH_DRIVING_FEAR_AFTER_TRAFFIC_ACCIDENTS).
- [17] Ramona Kenntner-Mabiala. *Treatment of patients with fear of driving following a traffic accident by a virtual reality exposure therapy in a driving simulator - a pilot study funded by the DGUV WPA XVII World Congress of Psychiatry, Berlin*. Sept. 2017. URL: [https://www.researchgate.net/publication/331198676\\_Treatment\\_of\\_patients\\_with\\_fear\\_of\\_driving\\_following\\_a\\_traffic\\_accident\\_by\\_a\\_virtual\\_reality\\_exposure\\_therapy\\_in\\_a\\_driving\\_simulator-a\\_pilot\\_study\\_funded\\_by\\_the\\_DGUV\\_WPA\\_XVII\\_WORLD\\_CONGRESS\\_OF\\_PSYCHIA](https://www.researchgate.net/publication/331198676_Treatment_of_patients_with_fear_of_driving_following_a_traffic_accident_by_a_virtual_reality_exposure_therapy_in_a_driving_simulator-a_pilot_study_funded_by_the_DGUV_WPA_XVII_WORLD_CONGRESS_OF_PSYCHIA).