Indice

Introduzione

Nel presente elaborato è descritta la progettazione e lo sviluppo dell'applicazione "Fantasy World Adventures", un gioco realizzato all'interno del progetto 3D4Amb. Tale progetto, condotto dall'Università degli Studi di Bergamo sotto la direzione del Professor Angelo Gargantini, si occupa di fornire degli strumenti alternativi alla diagnosi e alla cura dell'ambliopia, una patologia dell'apparato visivo che colpisce soggetti in età pediatrica.
Nel primo capitolo, è presentata nel dettaglio tale patologia, viene descritto il funzionamento dell'apparato visivo e definito il problema dell'ambliopia. Dopo aver definito le cause e le tipologie in cui viene classificata tale malattia, sono mostrate le cure tradizionali e i nuovi trattamenti di cura utilizzati.
In seguito è presentato il progetto 3D4Amb: gli obiettivi su cui esso si basa, una descrizione di come sono utilizzati i principi della tecnologia 3D e le tecniche utilizzate per la realizzazione di trattamenti di cura dell'ambliopia.
Il terzo capitolo mostra invece come è stata realizzata l'applicazione oggetto della presente tesi, in particolar modo spiega il suo funzionamento e gli strumenti utilizzati per la sua creazione. Viene inoltre descritto il processo di sviluppo utilizzato, i requisiti funzionali e non funzionali del gioco e i principi su cui si è basata la sua progettazione.
La parte più importante e cardine di questa relazione è descritta nel capitolo 4, in cui viene dettagliata l'implementazione di Fantasy World Adventures. Dopo un'esposizione dei metodi implementati e degli oggetti realizzati per la formazione del gioco, è presentato l'utilizzo della libreria 3D4Amb-Lib. Il seguente progetto ha infatti permesso la partecipazione al Beta Test della libreria 3D4Amb-Lib, una libreria creata nell'ambito del progetto 3D4Amb allo scopo di uniformare la gestione del trattamento di cura dell'ambliopia all'interno delle diverse applicazioni create. Al termine del quarto capitolo è spiegato come le funzionalità offerte da tale libreria sono state utilizzate nel gioco realizzato, i cambiamenti effettuati e le funzioni integrative implementate.
In conclusione, sono presentate le osservazioni riguardanti il lavoro svolto e i possibili miglioramenti dell'applicazione stessa.

Capitolo 1

Ambliopia

1.1 Cosa è l'ambliopia

L'ambliopia, conosciuta comunemente con il nome di "occhio pigro", è un'alterazione dello sviluppo della funzione visiva di uno (ambliopia monolaterale) o di entrambi gli occhi (ambliopia bilaterale), che si verifica durante il periodo di sviluppo del sistema visivo. In queste situazioni viene quindi determinata un'anomala interazione binoculare o una mancata formazione di immagini nitide sulla retina. La principale conseguenza è un deficit dell'acutezza visiva[1], non riconoscibile dai genitori senza una visita oculistica specializzata.
Si considera ambliope un occhio che abbia almeno una differenza di visione di 3/10 rispetto all'altro, oppure un visus inferiore ai 3/10.
Tale patologia è presente in circa il 3% di tutta la popolazione e il 4-5% dei bambini. Una diagnosi e una terapia precoce possono, nella maggioranza dei casi, curarla e prevenirne i disturbi permanenti in età adulta.
Il problema dell'ambliopia è quindi strettamente correlato con il corretto sviluppo della visione binoculare.

1.2 La visione binoculare

La capacità cerebrale di utilizzare le immagini fornite da entrambi gli occhi, per produrne una unica di grado superiore, è uno dei più alti livelli di sviluppo raggiunti dagli esseri viventi. Tale capacità prende il nome di binocularità.
La visione binoculare si basa sull'interazione tra il cervello e l'occhio per analizzare ed elaborare le informazioni visive. La luce entra nell'occhio, dove la retina traduce l'im-

[1] Acutezza visiva: o acuità visiva o visus è una delle abilità visive principali del sistema visivo ed è definita come la capacità dell'occhio di risolvere e percepire dettagli di un oggetto; essa dipende direttamente dalla nitidezza dell'immagine proiettata sulla retina.

magine in segnali nervosi che vengono inviati, grazie alle vie ottiche, al cervello. Quest'ultimo integra e interpreta gli stimoli visivi provenienti da ciascun occhio in un'immagine tridimensionale. La visione binoculare è un fenomeno molto complesso in cui interagiscono varie componenti: l'adeguato sviluppo delle strutture neuro-anatomiche, la buona capacità visiva degli occhi e una matura esperienza visiva.

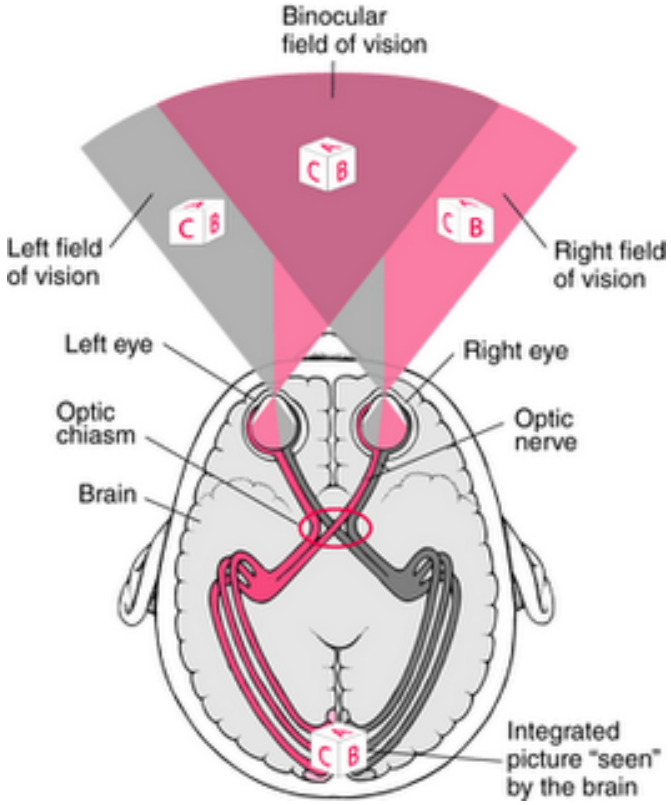

Figura 1.1: *Funzionamento della visione binoculare nell'uomo*

Per questi motivi la binocularità non è presente alla nascita, ma viene acquisita durante l'età del primo sviluppo, cioè nella fase da 0 a 2 anni; tuttavia essa si perfeziona in una fase successiva che si prolunga fino agli 8 anni. Dopo questa età tutto ciò che non si è sviluppato o non è stato acquisito è perso. Ciò può accadere quando il patrimonio genetico del bambino ha dei deficit strutturali che non consentono l'adeguato sviluppo degli strumenti neuro-sensoriali, oppure quando la presenza di una precoce ed elevata ametropia[2] non consente lo sviluppo visivo.
Gli aspetti di sviluppo della visione binoculare possono essere classificati secondo il modello proposto da Claud Worth[3] nel 1915.

[2]Ametropia: anomalia dell'occhio, per cui il suo secondo fuoco non cade sulla retina, ma al di là (ipermetropia o presbiopia) o al di qua (miopia).

[3]Claud Worth: oculista britannico inventore del Test delle 4 luci di Worth, per mezzo del quale viene valutato il grado di visione binoculare del paziente

Esso riconosce tre fasi, che nella pratica clinica vengono denominati i tre gradi della binocularità:

1. La percezione simultanea: le immagini colte da ogni singolo occhio in un dato istante vengono percepite dal cervello simultaneamente, ma in punti diversi dello spazio;

2. La fusione: le immagini sono viste dagli occhi simultaneamente e nella stessa direzione. In questa fase gli occhi hanno imparato ad orientarsi con precisione su un oggetto visivo. In particolar modo l'immagine fissata si proietta sulle retine degli occhi in due punti corrispondenti: le fovee[4]. Il cervello è quindi in grado di integrare i segnali provenienti dai due occhi in una percezione singola;

3. La stereopsi: quando i due occhi fissano uno stesso oggetto, inviano al cervello segnali provenienti da due immagini retiniche leggermente differenti. Occhio destro e occhio sinistro sono infatti collocati in due punti diversi rispetto al naso e quindi rispetto al punto di fissazione. Il cervello è in grado di ottenere da questa disparità importanti informazioni, in particolare consente la percezione della tridimensionalità degli oggetti e la loro localizzazione nello spazio, cioè la percezione delle distanze.

Tuttavia, se un deficit colpisce uno degli occhi durante la crescita, la qualità dei segnali trasmessi al cervello diventa perturbata e questo, a sua volta, influenza l'interpretazione delle immagini. Ciò significa che il bambino può vedere meno chiaramente da un occhio e tende ad affidare la propria visione all'altro.
Questo è il tipico problema dell'ambliopia: le strutture dell'occhio ambliope appaiono sane e funzionali, ma non sono utilizzate correttamente, in quanto il cervello favorisce l'altro occhio (detto dominante). Come risultato, il cervello fa sempre più affidamento sulla parte dominante ed inizia ad ignorare i segnali ricevuti dall'occhio ambliope.

1.3 Tipologie e Cause

L'ambliopia può essere classificata in due tipologie principali: funzionale e organica. Si definisce ambliopia funzionale, la patologia tale per cui le strutture dell'occhio appaiono sane e funzionali. In queste situazioni l'anomalia è legata a un non corretto sviluppo visivo e neuronale. Come visto in precedenza quindi, il cervello "disattiva" le immagini che arrivano da un occhio perché non riesce a combinarle con quelle provenienti dall'altro occhio: il cervello rinuncia all'effetto di tridimensionalità per avere una buona vista bidimensionale.

[4]Fovea: regione centrale della retina di massima acutezza visiva

L'ambliopia di tipo funzionale può a sua volta essere suddivisa, in base alla causa generativa, in:

- Ambliopia strabica: lo strabismo è causato da uno squilibrio muscolare che impedisce l'allineamento coordinato dei bulbi oculari: un occhio guarda dritto davanti a sé, mentre l'altro guarda a sinistra, a destra, in alto o in basso. In tal caso i segnali provenienti dall'occhio strabico saranno soppressi dal cervello e quindi esso sarà l'occhio ambliope;

- Ambliopia da errori di rifrazione: gli errori di rifrazione sono causati da alterazioni strutturali dell'occhio, il quale non focalizza l'immagine in modo corretto. Questa forma di occhio pigro è il risultato di una differenza significativa di rifrazione tra i due occhi, cioè la visione di un occhio è molto differente dall'altro (anisometropia), questo si verifica a causa della miopia[5], dell'ipermetropia[6] o di un' imperfezione sulla superficie dell'occhio (astigmatismo[7]). Questa tipologia può comportare inoltre l'insorgenza di strabismo perchè l'immagine dell'occhio col difetto maggiore viene soppressa a livello cerebrale, e si perde così la visione binoculare.

Invece viene definita ambliopia organica, anche detta ex anopsia (da deprivazione), la patologia legata ad un'alterazione delle vie ottiche, quindi ad una lesione del globo oculare o delle vie visive cerebrali. In tale tipologia, che è molto meno frequente rispetto a quella di tipo funzionale, l'ambliopia è causata dalla presenza di un "ostacolo" che impedisce la formazione dell'immagine sulla retina di un occhio.
Le cause principali di questo tipo di ambliopia sono:

- Cataratta congenita, opacizzazione del cristallino dell'occhio presente fin dalla nascita;

- Palpebra cadente (ptosi), anomalo abbassamento della palpebra superiore o inferiore;

- Glaucoma, malattia oculare che colpisce il nervo ottico, causando un continuo aumento della pressione all'interno dell'occhio;

- Patologie oculari, come un'ulcera o cicatrice corneale, cioè una lesione della cornea che ne può provocare opacità.

[5]Miopia: difetto di rifrazione per cui l'immagine di oggetti lontani si forma nell'occhio davanti alla retina rendendo la loro visione indistinta, mentre la visione degli stessi a breve distanza resta chiara e distinta.

[6]Ipermetropia: anomalia dell'accomodazione dell'occhio per cui si ha un difetto di convergenza dei raggi luminosi provenienti da oggetti distanti che si focalizzano in una zona dietro la retina, con formazione su di essa di un'immagine sfocata.

[7]Astigmatismo: difetto della vista per cui l'immagine di un punto appare più o meno allungata.

1.4 Diagnosi

L'ambliopia non è sempre evidente, molti casi di occhio pigro vengono diagnosticati durante visite oculistiche di routine, prima che i genitori si rendano conto della presenza di questo problema. È tuttavia necessario che la diagnosi avvenga il prima possibile per permettere una migliore correzione del difetto. Per tale motivo i bambini di età compresa tra i 3 e i 5 anni dovrebbero sottoporsi ad un esame della vista completo prima di iniziare la scuola e ad ulteriori controlli almeno ogni due anni; di solito, l'ambliopia viene diagnosticata intorno a quattro anni.
Nonostante i bambini piccoli non realizzino di avere problemi visivi, è possibile individuare alcuni segnali che possono essere causati da tale patologia: movimento involontario di un occhio verso l'interno o verso l'esterno, bassa sensibilità al contrasto, bassa sensibilità al movimento, scarsa percezione della profondità; per esempio i bambini con un occhio pigro, di solito, hanno problemi a giudicare con precisione la distanza tra sé e gli oggetti, questo può rendere difficili alcune attività come prendere una palla.

1.5 Cure tradizionali

La cura dell'ambliopia consiste principalmente in due fasi: la rimozione della causa scatenante e l'esecuzione di trattamenti anti-ambliopici, attraverso i quali il bambino viene incoraggiato ad utilizzare l'occhio ambliope allo scopo di consentire un corretto sviluppo della visione.
La correzione del difetto visivo è quindi la fase iniziale del trattamento di tale patologia. Il suo svolgimento varia in base alla sua tipologia e quindi al difetto visivo da correggere. In particolar modo, in caso di ambliopia anisometrica sarà previsto l'utilizzo di lenti a contatto o più frequentemente occhiali; anche per la cura dello strabismo sono utilizzati occhiali e lenti a contatto, ma in alcuni casi è necessaria la sua correzione per mezzo di un'operazione chirurgica. Invece, nel caso di ambliopia organica verrà effettuata un'operazione chirurgica al fine di correggere la cataratta congenita o la ptosi.
In seguito alla risoluzione del problema causa di ambliopia, verranno iniziati dei trattamenti volti alla correzione di tale patologia.
Tra i principali trattamenti ortottici anti-ambliopici sono presenti:

- Occlusione con un cerotto (patching): consiste nella copertura dell'occhio dominante. Il trattamento migliore è l'occlusione totale dell'occhio dominante (in almeno il 60%-80% delle ore di veglia). L'occlusione parziale (solo qualche ora al giorno o per qualche giorno alla settimana) viene attuata o per coloro che non

tollerano quella totale o in coloro che necessitano di una occlusione come terapia di mantenimento. Questa terapia prevede il posizionamento di un cerotto opaco, con un bordo adesivo, direttamente sulla pelle sopra l'occhio dominante, costringendo il bambino ad utilizzare l'altro. Il processo di recupero può richiedere diverso tempo, in base alla gravità del problema. Alcuni oculisti ritengono inoltre che lo svolgimento di particolari attività (lettura, visione di un programma televisivo, colorare ecc.) durante il tempo in cui il paziente mantiene il cerotto, possa essere più stimolante per il cervello ed agevolare un recupero più rapido;

Figura 1.2: *Bambina a cui è stato applicato il trattamento di occlusione*

- Penalizzazione ottica, attuata con filtri di Bangerter (lenti con gradi diversi di opacizzazione, a seconda della penalizzazione che si vuole attuare) o con lenti più forti o più deboli poste davanti all'occhio sano per costringere quello malato a lavorare; è prescritta per coloro che necessitano di una occlusione "morbida" o come terapia di mantenimento;

Figura 1.3: *Esempio di occhiali con Filtro di Bangerter*

- Penalizzazione farmacologica: collirio a base di atropina instillato nell'occhio sano per escluderlo dal processo di visione e costringere quello malato a lavo-

rare. L'atropina è un farmaco anticolinergico[8]; strutturalmente è un alcaloide[9] naturale estratto dalle foglie di alcune Solanacee (Atropa belladonna). L'atropina blocca le risposte alla stimolazione colinergica del muscolo dell'iride e del muscolo ciliare; il farmaco è in grado di dilatare la pupilla (midriasi) e di paralizzare l'accomodazione visiva (cicloplegia). Per la cura dell'ambliopia viene somministrata una goccia di atropina in collirio al giorno o due volte alla settimana, questa può infatti offuscare temporaneamente la vista nell'occhio più forte. Il trattamento con atropina stimola indirettamente la vista nell'occhio più debole ed aiuta la parte del cervello che gestisce la visione a svilupparsi in modo più completo;

Figura 1.4: *Bambino a cui viene somministrata atropina in collirio*

- Settorizzazione: copertura di parte del campo visivo dell'occhio sano con pellicole adesive traslucide sugli occhiali.

1.6 Cure innovative

Alcune nuove tecniche si basano sull'utilizzo del Filtro di Gabor[10] e sulla neurostimolazione visiva: i primi utilizzi sono avvenuti con il trattamento CAM-Cambridge

[8]Colinergico: fa riferimento a particolari composti che stimolano il sistema nervoso parasimpatico, il quale è una parte del sistema nervoso autonomo. Il sistema nervoso autonomo, conosciuto anche come sistema nervoso vegetativo o viscerale, è quell'insieme di cellule e fibre che innervano gli organi interni e le ghiandole, controllando quelle funzioni che generalmente sono al di fuori del controllo volontario.

[9]Alcaloide: si intende una sostanza organica di origine vegetale avente gruppi amminici tali da impartire alla struttura un carattere basico, e dotata di grandi effetti farmacologici in relazione all'assunzione di piccole dosi di sostanza (per es. caffeina, morfina, stricnina).

[10]Filtro di Gabor: filtro lineare ottenuto mediante la modulazione di una funzione sinusoidale con una funzione gaussiana di standard deviation σ. Con tale tecnica vengono quindi prodotti diversi schemi visivi.

Stimulator e il trattamento Flicker, recentemente è stato sviluppato anche il software RevitalVision.
Cambridge Stimulator si basa sullo stimolo delle cellule visive corticali, ponendo in lenta rotazione superfici con barre bianche e nere alternate con differenti frequenze spaziali ed elevato contrasto.
Pattern Flicker, invece, si basa su uno stimolo strutturato idoneo per tutte le cellule delle vie ottiche e della corteccia visiva. Tale stimolo è effettuato per mezzo di bande alternanti rosse (tale luce monocromatica rossa consente una stimolazione selettiva fovea-maculare) e nere su sfondo nero (per distogliere da stimoli indesiderati dell'ambiente) con inclinazione, forma e movimento vari.

Figura 1.5: *Pattern Flicker*

In RevitalVision sono previste invece sedute di allenamento: una serie di esercizi visivi e di immagini ripetute stimola il cervello a migliorare l'elaborazione dell'immagine ricevuta dalla retina. Tali sessioni sono inoltre costantemente aggiornate in base ai risultati ottenuti e alle caratteristiche del singolo utilizzatore.

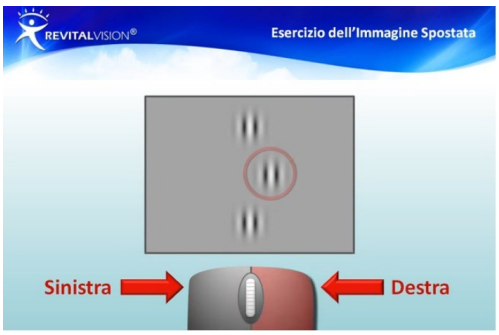

Figura 1.6: *Esempio di esercizio del programma RevitalVision*

Queste tecnologie pur diverse tra loro si basano sull'invio di stimoli luminosi di vario tipo sulla retina dell'occhio ambliope, forzandolo a trasmettere l'impulso luminoso al

cervello, e riattivando così i canali "impigriti" dall'ambliopia. L'efficacia di queste metodologie è ancora oggi molto dibattuta.
Un altro esempio di cure innovative riguarda l'utilizzo di farmaci neuro-protettivi (come la colina[11]) come sostegno alla terapia occlusiva: studi recenti indicano che in questo modo viene potenziato l'effetto della terapia occlusiva e viene più facilmente stabilizzato il miglioramento della funzione visiva. Per mezzo di tali tecnologie stanno quindi aumentando le possibilità di miglioramenti dell'ambliopia anche in età adulta, o comunque dopo i 9 anni di età, età che rappresenta il termine della cosiddetta "età plastica", in cui il bambino sviluppa tutte le sue funzioni organiche.

[11] Colina: spesso detta anche vitamina J, è una molecola simile alle vitamine del gruppo B che viene sintetizzata dal fegato e che interviene come coenzima in numerose reazioni metaboliche. Può essere particolarmente importante per il buon funzionamento del sistema nervoso.

Capitolo 2

Progetto 3D4Amb

Il progetto 3D4Amb mira a sviluppare un sistema basato sul 3D per il trattamento dell'ambliopia nei bambini piccoli. Si basa sull'utilizzo delle tecnologie 3D per offrire una visione binoculare, cioè per mostrare immagini differenti all'occhio ambliope rispetto a quelle trasmesse all'occhio sano, consentendo quindi lo sviluppo dell'occhio ambliope. Esso consente una semplice diagnosi dell'ambliopia e un suo trattamento per mezzo di giochi di intrattenimento. Il progetto si basa sull'utilizzo di diverse tecnologie 3D per la creazione di queste applicazioni, tra cui 3D active shutter technology, gli occhiali anaglifici e i visori di realtà virtuale.

2.1 Obiettivi

L'obiettivo principale di tale progetto è quello di individuare un nuovo ed innovativo trattamento volto alla cura dell'ambliopia per mezzo di un insieme di applicazioni diverse aventi come principio portante l'accessibilità. Questo comporta la necessità che il trattamento sia economico: è infatti necessario che sia accessibile a tutte le famiglie, quindi il suo utilizzo si basa su tecnologie a basso costo facilmente reperibili nei negozi al dettaglio o in Internet (per esempio smartphone o computer, occhiali 3D o anaglifici o VR cardbord e un joystick o mouse).
Un'altra caratteristica importante del progetto è la possibilità di un suo utilizzo domestico, le applicazioni realizzate sono infatti utilizzate direttamente a casa senza la necessità di recarsi in strutture ospedaliere o in ambulatori specializzati.
Il progetto 3D4Amb si basa inoltre sulla realizzazione di applicazioni per mezzo di software open source allo scopo di consentire un loro successivo sviluppo e miglioramento. Molto importante inoltre è la semplicità d'uso delle applicazioni allo scopo di garantire un utilizzo autonomo da parte dei bambini, limitando l'intervento di un adulto all'iniziale installazione.

L'obiettivo principale tuttavia si basa sull'offerta di un'alternativa divertente ai metodi tradizionali per la cura dell'ambliopia, che sono tendenzialmente difficilmente sopportabili da parte dei pazienti.

2.2 Tecnologia 3D e visione binoculare

Il trattamento dell'ambliopia realizzato dal progetto 3D4Amb pur essendo un'alternativa ai metodi di cura classici, mantiene lo stesso principio di funzionamento. Esso come detto in precedenza utilizza le tecnologie 3D che si basano sui principi della realtà virtuale[1]. Tuttavia, le caratteristiche di tale tecnologia sono sfruttate in un'ottica differente; il loro utilizzo non ha infatti lo scopo di consentire l'interazione in un mondo, una realtà completamente diversa, come avviene tipicamente nelle applicazioni basati sulla realtà virtuale e quindi sulla visione tridimensionale, ma il suo obiettivo è consentire una visione binoculare, questo viene permesso sfruttando il modo in cui i visori sovrappongono le immagini. L'utilizzo classico dei sistemi 3D infatti permette di offrire agli occhi due diverse immagini della stessa scena aventi un leggero offset tra gli angoli di visione, che corrisponde ai diversi punti di vista che hanno l'occhio destro e sinistro. Questo metodo produce un'illusione di profondità della scena che è la base della realtà virtuale.
Nel nostro progetto, invece, è stata sfruttata solo la capacità dei sistemi 3D di inviare due immagini diverse agli occhi mentre non è stata sviluppata la ricreazione di una realtà virtuale che non rientra nei nostri obiettivi. Il principio base del nostro sistema è la trasmissione all'occhio ambliope e a quello sano di due immagini correlate, ma differenti. In particolar modo, avverrà la trasmissione all'occhio sano di un'immagine, in cui vengono nascosti dettagli e particolari importanti per il corretto svolgimento del gioco, e a cui è applicato una sorta di filtro scuro per ridurre la visione nitida dell'immagine stessa (viene creata un'occlusione parziale). Al contrario l'occhio ambliope riceverà l'immagine originale, nitida e comprendente tutti i dettagli e particolari. In tal modo il bambino sarà spronato ad utilizzare e sviluppare l'occhio ambliope da cui potrà ricevere l'immagine completa e cioè tutte le informazioni.
Nella Figura 2.1 è quindi possibile vedere il funzionamento delle tecnologie 3D ed in particolar modo come avviene il processo di trasmissione di immagini differenti e il loro assemblaggio da parte del cervello. Il contenuto da mostrare al paziente (scena di gioco o immagine) è infatti separato dal software 3D4Amb in due parti, una verrà tra-

[1]La realtà virtuale, per sua stessa definizione, è una tecnologia volta a simulare la realtà effettiva. Essa è realizzata per mezzo delle tecnologie informatiche e permette di navigare in ambientazioni fotorealistiche in tempo reale, interagendo con gli oggetti presenti in esse. Questa tecnologia si è evoluta fino allo sviluppo della Realtà Virtuale Immersiva o RVI che si basa sulla creazione di un sistema totalmente immerso in cui tutti i sensi umani possono essere utilizzati.

smessa all'occhio destro (in questo esempio è l'occhio ambliope) ed una sarà trasmessa all'occhio sinistro (quello sano nell'esempio). L'occhio ambliope sarà maggiormente stimolato a lavorare, ma l'occhio sano non è completamente escluso dal processo di visione (non c'è occlusione totale). Il cervello del paziente dovrà quindi unire le due immagini per vedere l'immagine completa.

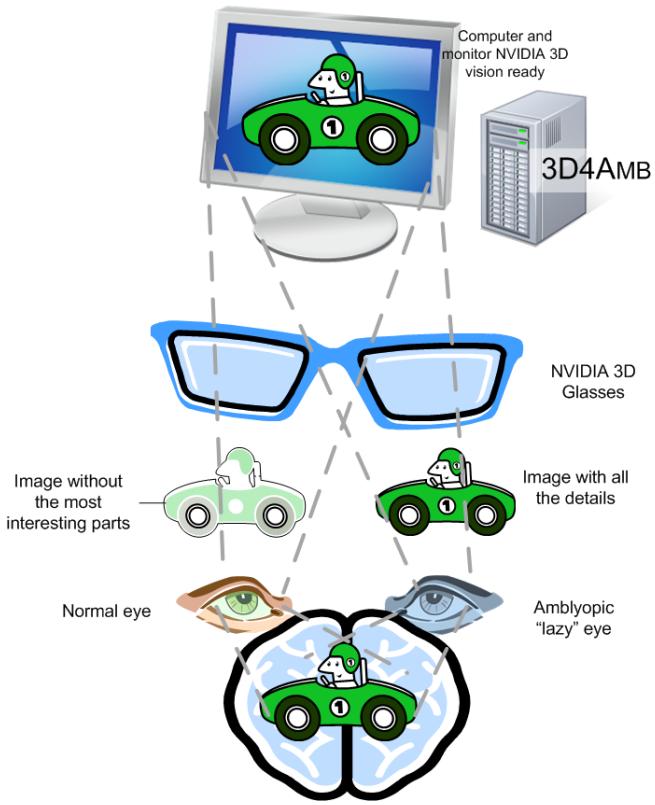

Figura 2.1: *Funzionamento del processo 3D per l'ambliopia*

2.3 Tecniche di progetto

Nel progetto 3D4Amb la realizzazione del sistema di trasmissione di immagini diverse agli occhi per mezzo della tecnologia 3D, viene realizzato utilizzando tre tecniche principali: tecnica con occhiali 3D, tecnica anaglifica e tecnica per visori di realtà virtuale.

2.3.1 Tecnica con occhiali 3D

Tale tecnica si basa sull'utilizzo di appositi occhiali 3D a lenti non colorate, come per esempio gli occhiali 3D di Nvidia. Per mezzo di questi occhiali, si possono guardare immagini specifiche o video che sono stati precedentemente filtrati dal sistema di

Nvidia 3D, in base alla scelta di quale immagine inviare ad un occhio rispetto all'altro (cioè in base a quale dei due occhi è quello sano). In questo modo si può degradare l'immagine ricevuta dall'occhio sano per costringere il cervello del paziente ad utilizzare le immagini e quindi le informazioni ricevute dall'occhio ambliope. L'esempio del suo funzionamento è mostrato nella Figura 2.1.

Figura 2.2: *Occhiali NVIDIA 3D*

2.3.2 Tecnica anaglifica

Gli anaglifi sono immagini stereoscopiche o stereogrammi che se osservate attraverso appositi occhiali, dotati di due filtri di colore complementare l'uno rispetto all'altro, forniscono un' illusione di tridimensionalità. Il principio di funzionamento dell'anaglifo tradizionale riguarda la creazione di due immagini parallele che vengono sovrapposte. La discriminazione delle due immagini che sono destinate separatamente ai due occhi avviene per mezzo di filtraggio cromatico: nelle due immagini gli oggetti e le parti importanti saranno colorate per mezzo di due colori complementari (rosso/verde, blu/giallo, o più comunemente rosso/ciano) e avverrà poi la loro sovrapposizione che permetterà al cervello di interpretare le differenze visive e di determinare le distanze tra gli oggetti. Ogni occhio vede solo la parte che gli compete e l'unione delle informazioni consente quindi di avere una visione tridimensionale. Il progetto 3D4Amb sfrutta, per la cura dell'ambliopia, la possibilità di mostrare agli occhi immagini diverse per mezzo della scelta di posizionamento dei colori nell'immagine. Per poter osservare un'immagine anaglifica, è necessario l'utilizzo di appositi occhiali filtrati con i medesimi colori complementari presenti: in tal modo l'occhio che utilizza la lente rossa vedrà le parti rosse dell'immagine come "chiare/bianche", mentre le componenti in ciano saranno viste come "scure/nere". Viceversa, l'occhio che utilizza il filtro ciano (la lente) vedrà le componenti rosse dell'immagine come "scure/nere", mentre quelle blu saranno viste come "bianche/chiare". Le parti bianche, nere o grigie

dell'immagine saranno invece percepite da entrambi gli occhi. Ogni lente assorbirà/filtrerà quindi le parti dell'immagine del proprio colore mostrando invece quelle del colore complementare, che sarà comunque soggetto al filtraggio e per tale motivo apparirà nero. In base quindi alla scelta dei colori complementari distribuiti negli oggetti dell'immagine è possibile nascondere diversi dettagli all'occhio sano, in modo da potenziare l'apparato oculare di quello ambliope. L'utilizzo di tali applicazioni è possibile per mezzo del solo acquisto di occhialini anaglifici.

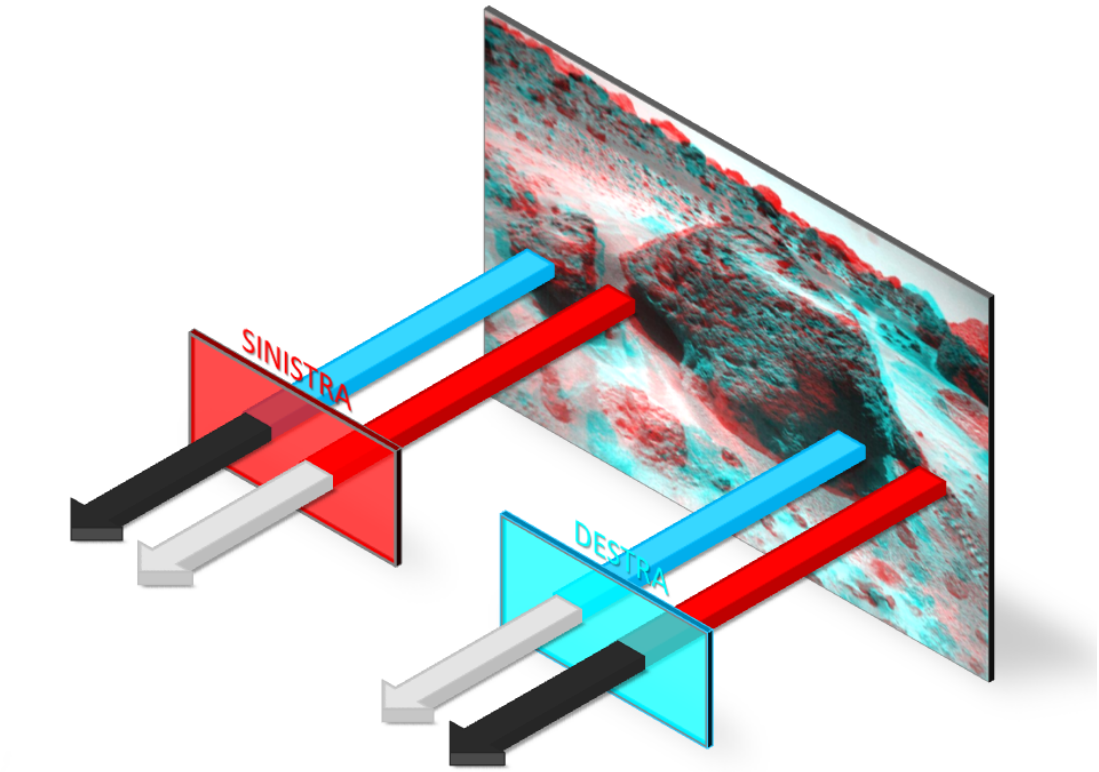

Figura 2.3: *Funzionamento della visione di un'immagine anaglifica*

2.3.3 Tecnica per visori di realtà virtuale

Questa tecnica si basa sull'utilizzo dei visori per la realtà virtuale per mostrare un'applicazione, in esecuzione su uno smartphone, e consentire la sua visione in 3D. Sullo schermo del cellulare, per permettere la visione tridimensionale e binoculare, sarà presente la stessa scena duplicata, una per ogni occhio. Il progetto 3D4Amb sfrutta questa duplicazione della scena, che è alla base dell'utilizzo di questi visori, per modificare una delle due immagini duplicate allo scopo di consentire la penalizzazione dell'occhio sano e quindi uno sviluppo di quello ambliope. Come detto in precedenza, la scena mostrata all'occhio sano sarà quindi privata di dettagli importanti e sarà meno

nitida rispetto a quella mostrata all'occhio ambliope; in tal modo quest'ultimo sarà spronato a svilupparsi.

Figura 2.4: *Esempi di VR*

Capitolo 3

Fantasy World Adventures: Modello e Progettazione

3.1 Informazioni generali

Figura 3.1: *Livelli di gioco di "Fantasy World Adventures"*

Il gioco “Fantasy World Adventures” oggetto della seguente tesi, è un platform game (gioco a piattaforme), cioè un videogioco d’azione in cui l’obiettivo principale è l’attraversamento di un insieme di livelli costituiti da piattaforme, disposte su piani diversi su cui il giocatore/avatar può saltare, salire e scendere. In particolar modo il gioco è costituito da otto livelli, che si differenziano per background, nemici e oggetti presenti, ma in cui la dinamica è la stessa per consentire un utilizzo semplice e intuitivo da parte dei bambini. Il gioco si basa sul comando, da parte del giocatore, di un avatar che può essere scelto tra due versioni: maschile e femminile. Precisamente, il

giocatore può far correre l'avatar avanti o indietro nella scena (la possibilità di tornare indietro non è infinita, ma è consentita solo per un tratto limitato che dipende dalla posizione in cui si trova l'avatar in ogni preciso momento), saltare in qualsiasi posizione si trovi (non sono definiti dei limiti ai salti consecutivi per garantire maggior libertà e divertimento, ma è solo definito un limite di altezza raggiungibile che varia in base al livello stesso). L'obiettivo del gioco è il raggiungimento della fine di ogni livello entro un tempo predefinito e senza perdere tutte le vite a disposizione. Nella scena di gioco sono infatti presenti dei nemici in movimento che il giocatore deve evitare, il contatto con un qualsiasi nemico comporta infatti la perdita di una vita. Il giocatore può inoltre sconfiggere questi nemici saltando sopra di essi, questo gli farà guadagnare dei punti. In base al livello saranno presenti diverse tipologie di nemici con diverse azioni di attacco, precisamente sono presenti: cactus e ragni nel primo livello, troll e orchi nel secondo livello, maghi e draghi nel terzo livello, zombies nel quarto livello, guerrieri nel quinto livello, dinosauri nel sesto livello, yeti e vichinghi nel settimo livello e alieni nell'ottavo livello. I nemici scelti sono tutti personaggi della fantasia o delle storie raccontate ai bambini e per questo a loro più famigliari. Oltre ai nemici sono presenti degli ostacoli che è necessario evitare e che variano in base alle diverse scene quali per esempio sabbie mobili, buchi neri, lava... La scelta di questi è stata fatta in base all'ambientazione utilizzata e ai personaggi/nemici da essa popolata.

Nel corso del gioco, in ogni livello, il giocatore può raccogliere delle monete e degli oggetti che gli permetteranno di guadagnare punti. Sono inoltre presenti degli aiuti extra quali vite aggiuntive, immunità temporanee contro i nemici e gli ostacoli e tempo aggiuntivo. In particolar modo è possibile accumulare al massimo tre vite, nel caso in cui l'avatar abbia già tre vite e ne raccoglie una successiva essa incrementa solamente i punti. Il gioco è stato realizzato consentendo di riprovare ogni livello un numero infinito di volte fino alla sua vittoria senza perdere i risultati ottenuti nei livelli precedenti. Al raggiungimento dell'ultimo livello verrà mostrato il risultato complessivo, dato dalla somma dei punti dei vari livelli, che sarà comparato con i risultati precedentemente raggiunti dall'attuale giocatore. Quando il giocatore interrompe una partita essa sarà salvata in automatico (il salvataggio viene effettuato solo dopo aver vinto almeno un livello), questo permette al giocatore di scegliere, successivamente, se continuare la partita non completata o se iniziarne una nuova.

3.2 Modello di sviluppo e analisi dei requisiti

La realizzazione del seguente gioco è avvenuta attraverso un processo di sviluppo iterativo, a cui si è affiancata una fase di integrazione di componenti preesistenti; il

processo di sviluppo utilizzato può essere rappresentato come segue:

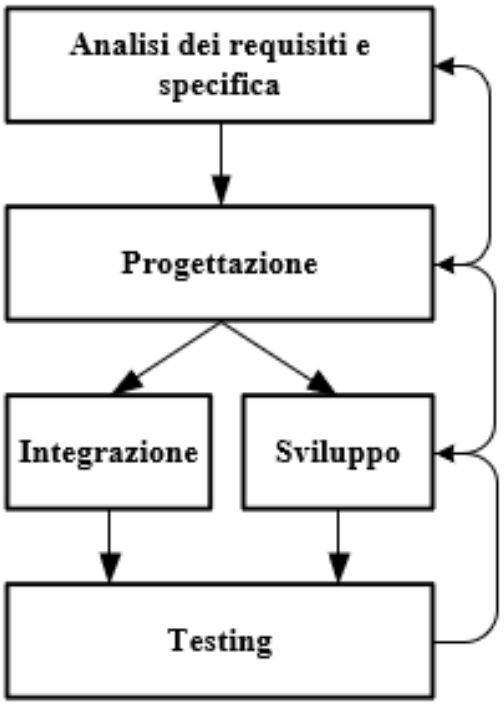

Figura 3.2: *Modello di sviluppo utilizzato per la realizzazione di "Fantasy World Adventures"*

In seguito ad un'analisi dei requisiti e delle specifiche, sono stati definiti gli obiettivi del gioco e le funzionalità principali che lo caratterizzano. È stata eseguita la progettazione del gioco stesso, determinando come realizzare le funzionalità richieste e permettendo in tal modo un successivo sviluppo più semplice e lineare.

Successivamente allo sviluppo delle diverse funzionalità è stata effettuata la fase di test, per mezzo degli strumenti descritti nel paragrafo successivo, durante la quale è stato possibile individuare eventuali modifiche da apportare. Inoltre gli incontri con il docente relatore hanno portato alla definizione di nuove funzionalità da inserire e per tale motivo il processo di sviluppo è stato rieseguito in modo iterativo.

La realizzazione della funzionalità di penalizzazione e di gestione del menu è stata invece effettuata integrando l'applicazione realizzata con la libreria 3D4Amb-Lib preesistente.

La fase di analisi dei requisiti dell'applicazione ha quindi portato alla definizione dei seguenti requisiti funzionali:

- Configurazione dell'occhio da penalizzare (occhio sano);

- Configurazione del grado e della tipologia di penalizzazione: l'utente deve poter impostare la percentuale minima e massima di trasparenza degli oggetti e dell'oscuramento della scena;
- Gestione di più utenti: l'utente ha la possibilità di creare un nuovo giocatore, cambiare il giocatore attuale ed eventualmente eliminarlo;
- Gestione della velocità di gioco: l'utente può scegliere la velocità che utilizzerà il suo avatar nel gioco, la scelta può avvenire da un minimo a un massimo prestabilito;
- Opzione di continuare una partita: l'utente ha la possibilità di scegliere se continuare la partita iniziata e non terminata o iniziarne una nuova;
- Pausa: l'utente ha la possibilità di mettere in pausa il gioco e riprenderlo successivamente;
- Interfaccia utente: l'utente ha la possibilità di vedere durante l'esecuzione della partita i punti acquisiti durante quel livello e il tempo rimanente per il suo completamento;
- Salvataggio di tutte le impostazioni di ogni utente: le impostazioni scelte dall'utente sono salvate in locale sul dispositivo in utilizzo e caricate in base alla selezione dello specifico giocatore;
- Salvataggio dei risultati al termine di ogni partita: l'utente ha a disposizione tutti i risultati delle partite completate con l'indicazione del giorno in cui li ha eseguiti.

I requisiti non funzionali dell'applicazione rispecchiano invece quelli del progetto 3D4Amb di cui fa parte, essi sono riassunti in:

- Usabilità:
  - Semplicità di utilizzo: il gioco deve essere strutturato in modo semplice e il suo uso deve essere intuitivo in modo da consentire all'utente medio (bambini di 5-6 anni) di poter giocare autonomamente;
  - Divertente: il gioco deve invogliare il bambino al suo utilizzo permettendo quindi la corretta esecuzione del trattamento;
  - Economicità: il gioco deve essere accessibile a tutti, il costo per il suo utilizzo si limita all'acquisto di un visore di realtà virtuale e di un remote controller (oltre alla presenza di un cellulare).

- Portabilità: il gioco deve essere compatibile con tutte le versioni Android attualmente utilizzate;
- Efficienza: il gioco deve garantire buone prestazioni su smartphone di fascia medio-bassa.

3.3 Strumenti utilizzati

3.3.1 Software

Per la realizzazione di "Fantasy World Adventures" sono stati utilizzati i seguenti strumenti software che hanno permesso la scrittura degli scripts, la modifica delle immagini e la creazione effettiva delle scene del gioco.

Unity

Unity è un ambiente di sviluppo grafico gratuito che permette la realizzazione di giochi e applicazioni 3D o 2D. Tale ambiente è disponibile sia per Windows che per macOS, inoltre è caratterizzato da una grande versatilità in quanto i giochi e i contenuti prodotti possono essere eseguiti su pc con diversi sistemi operativi (Windows, Mac, Linux), ma anche su altri dispositivi (smartphone con qualunque OS, Xbox, Playstation...). Questo permette una grande portabilità dell'applicazione che può essere diffusa su diverse piattaforme, senza la necessità di una nuova realizzazione.
Un altro vantaggio di questo motore grafico è la possibilità di agire sugli elementi grafici che caratterizzano il gioco, definiti nell'ambiente di sviluppo con il termine di "GameObject", non solo attraverso scripts di codice, ma anche per mezzo della gestione di alcune proprietà e impostazioni offerte all'interno dei pannelli dell'editor. Unity permette la scrittura degli scripts per mezzo di due linguaggi di programmazione: JavaScript e C#.
JavaScript è un linguaggio di scripting comunemente utilizzato nella programmazione Web lato client, per la creazione di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati dall'utente sulla pagina web in uso. C# invece è un linguaggio di programmazione orientato agli oggetti sviluppato da Microsoft all'interno dell'iniziativa .NET. La sintassi e struttura del C# prendono spunto da vari linguaggi nati precedentemente, in particolare Delphi, C++, Java e Visual Basic. Nella realizzazione del gioco "Fantasy World Adventures" è stato tuttavia utilizzato solamente il linguaggio C#.
In altre parole, Unity è un insieme di strumenti completo per la creazione di videogiochi e altri progetti interattivi, semplificando il processo di sviluppo e rendendolo più

veloce. Molti sviluppatori di giochi indipendenti ("indie games") utilizzano Unity per risparmiare e distribuire i propri prodotti sul maggior numero di mercati possibile. Il team Rovio (i creatori di Angry Birds) ha utilizzato Unity per il gioco Bad Piggies, e molti altri giochi molto noti sono stati creati con questo tool.

Visual Studio

Microsoft Visual Studio è un ambiente di sviluppo integrato (Integrated development environment o IDE) sviluppato da Microsoft. Esso supporta diversi tipi di linguaggi, quali C, C++, C#, F#, Visual Basic .Net, Html e JavaScript, e permette la realizzazione di applicazioni, siti web, applicazioni web e servizi web. Un altro vantaggio di questo ambiente è il fatto che è multiplatform e quindi permette la realizzazione di programmi per server, workstation, PC, smartphone e per i browser. Visual Studio integra la tecnologia IntelliSense la quale permette di correggere eventuali errori sintattici (ed alcuni logici) senza compilare l'applicazione, possiede un debugger interno per il rilevamento e la correzione degli errori logici nel codice in runtime e fornisce diversi strumenti per l'analisi prestazionale. Tutte queste caratteristiche lo rendono un ottimo strumento per affiancare Unity nella realizzazione di giochi e applicazioni. In particolar modo mentre in Unity è possibile definire gli oggetti che fanno parte del gioco ("GameObjects") e modificare le loro proprietà; Visual Studio permette l'implementazione, attraverso specifici scripts, dei comportamenti di ogni oggetto, per esempio le reazioni ad alcuni eventi. In conclusione, Unity definisce i GameObjects e le loro proprietà, mentre VisualStudio determina il loro Behaviour. In assenza di Visual Studio, Unity consente la modifica del codice per mezzo di MonoDevelop, ma questo ambiente di sviluppo non offre tutte le funzionalità consentite da VisualStudio, che permette uno sviluppo più semplice, individuando e quindi richiedendo la correzione di eventuali errori in tempo reale; per tale motivo nella realizzazione del gioco è stato preferito il suo utilizzo.

Gimp

GIMP (GNU Image Manipulation Program) è un software libero multipiattaforma per l'elaborazione digitale delle immagini. Fra i vari usi possibili vi sono fotoritocco, fotomontaggio, conversioni tra molteplici formati di file, animazioni. GIMP è compatibile anche con il formato proprietario PSD di Adobe Photoshop, concorrente molto simile a livello di funzionalità. Tale software è stato utilizzato nella realizzazione del gioco, oggetto di questa tesi, per la creazione e modifica dei background delle scene

dei diversi livelli, oltre che per la modifica e ridimensione di alcuni sprites[1]. Gli sprites utilizzati per la realizzazione del gioco sono stati nella maggior parte dei casi scaricati da appositi siti web che si occupano della realizzazione della grafica per giochi interattivi e che offrono tali contenuti gratuitamente.

3.3.2 Hardware

I componenti hardware sono stati utilizzati principalmente nella fase di testing dell'applicazione. Infatti, durante il suo sviluppo il controllo della correttezza dei diversi metodi e delle funzionalità implementate è stato gestito per mezzo dell'editor di Unity, che consente di simulare il gioco durante la sua creazione.

Visori

Figura 3.3: *VR-Shark M3 - Google Cardboard*

Per consentire la trasmissione di immagini diverse agli occhi e quindi permettere la visione binoculare, oltre alla tecnologia 3D, è stato utilizzato un visore per la realtà virtuale. Sono presenti diverse tipologie di visori: alcuni sono molto economici e quindi adatti a tutte le esigenze, mentre altri più costosi consentono visualizzazioni migliori e un utilizzo più comodo. Ogni visore si basa sulla presenza di un vano regolabile, in cui andrà inserito il proprio smartphone, e due lenti. Il trattamento su cui si basa il progetto, è effettuato per mezzo di uno smartphone, su cui è installata l'applicazione stessa. Tale applicazione prevede la separazione dello schermo in due parti, entrambe rappresentanti la stessa immagine, ma leggermente modificata. Ogni lente mostra solo una

[1] Sprite: in grafica informatica lo sprite è una figura bidimensionale che può essere spostata rispetto allo sfondo.

parte dello schermo e trasmette all'occhio un'immagine specifica; i due occhi avranno due visualizzazioni diverse, consentendo la penalizzazione di un occhio (quello sano) e ottenendo quindi il risultato desiderato.

Remote Controller

Figura 3.4: *Remote Controller*

Per poter interagire con il gioco "Fantasy World Adventures", e in generale con tutti i giochi e le applicazioni che sono realizzate per l'utilizzo di VR all'interno del progetto 3D4Amb, è necessario l'utilizzo di un remote controller.
Essendo infatti lo smartphone all'interno del vano del visore non sarà possibile l'utilizzo del touchscreen, tale problema è semplicemente risolto per mezzo di questo strumento. Il remote controller è un telecomando esterno che interagisce con lo smartphone e quindi con il gioco in esecuzione per mezzo della tecnologia Bluetooth. In modo analogo ai visori, anche i remote controller possono essere classificati in diverse categorie in base al loro prezzo e alle loro prestazioni. Proprio questa varietà di offerta è molto importante per la realizzazione del progetto perché permette l'utilizzo di tale tecnologia con una spesa minima e una semplicità di utilizzo.

3.4 Progettazione

La fase di progettazione riguarda la definizione della struttura del software che consenta di soddisfare le specifiche funzionali e non funzionali; quindi è necessario determinare come tali funzioni debbano essere allocate tra i diversi componenti da cui è composta l'applicazione. Gli scripts per la realizzazione delle funzionalità richieste, sono stati quindi creati nel rispetto delle regole di progettazione:

- Minimizzazione dell'accoppiamento: ridurre al minimo l'interazione tra le diverse componenti. Nello sviluppo del gioco non è stato possibile garantire il massimo grado di disaccoppiamento a causa della realizzazione del gioco in

scene diverse; questo ha reso necessario l'utilizzo dell'accoppiamento di tipo common, cioè i diversi moduli interagiscono per mezzo di scambio di flag e dati;

- Massimizzazione della coesione: è stato garantito un alto grado di coesione, ogni script racchiude infatti le funzionalità e i dati di uno specifico tema applicativo o al più esegue un'unica funzione applicativa. Questo consente una semplice comprensione del sistema e un suo possibile riuso;

- Information hiding: i moduli e quindi i diversi scripts nascondono i dettagli interni agli altri scripts, mantenendo visibili solo le proprietà per mezzo delle quali i diversi componenti interagiscono;

- Riuso dei componenti preesistenti: nella gestione di alcune funzionalità, in particolar modo la penalizzazione e la gestione del menu, è stata utilizzata la libreria 3D4Amb-Lib. Inoltre per la creazione dei GameObjects sono stati usati componenti preesistenti reperibili dall'AssetStore[2] o da altri siti web specializzati in grafica.

[2]AssetStore: negozio online di Unity, dove è possibile comprare dei gameobjects, assets già completi; sono inoltre disponibili alcuni modelli gratuiti.

Capitolo 4

Fantasy World Adventures: Implementazione

Lo sviluppo di Fantasy World Adventures è stato effettuato in due fasi principali: un'iniziale implementazione del sistema di gioco ed una fase di integrazione e miglioramento. Nella prima fase di sviluppo è avvenuta la creazione delle scene principali di gioco, in cui sono stati inseriti gli oggetti più importanti e implementati gli scripts per consentire il funzionamento del gioco vero e proprio. Solo successivamente, sono stati inseriti tutti gli effetti grafici e sonori, inoltre sono state integrate le funzionalità offerte dalla libreria 3D4Amb-Lib e aggiunte ulteriori funzionalità e migliorie. In questo paragrafo, sono presentate le principali scene visualizzabili dal giocatore e gli scripts associati agli oggetti, in esse contenuti, che consentono la corretta esecuzione del gioco.

4.1 Le scene

La prima fase di implementazione del gioco è stata la creazione delle scene, cioè le schermate che il giocatore potrà visualizzare durante il suo utilizzo. Inizialmente sono stati posti solamente gli aspetti di base e solo in un momento successivo sono stati integrati dettagli e ulteriori particolari. Le principali scene su cui si basa il gioco sono:

- Menu principale: in questa scena è possibile visualizzare i diversi pannelli di configurazione, un pannello per mezzo del quale avere informazioni sull'utilizzo dell'applicazione (help) e sarà inoltre possibile iniziare una partita;
- Livelli: è stata creata una scena per ogni livello disponibile, in tali scene sono presenti tutti gli oggetti che il giocatore visualizzerà nel corso della partita;

- Scena LoadNext: scena visualizzata alla vittoria di un livello che consente di proseguire la partita;
- Scena LoadRetry: scena visualizzata alla perdita di un livello che consente di riprovare il livello appena perso.

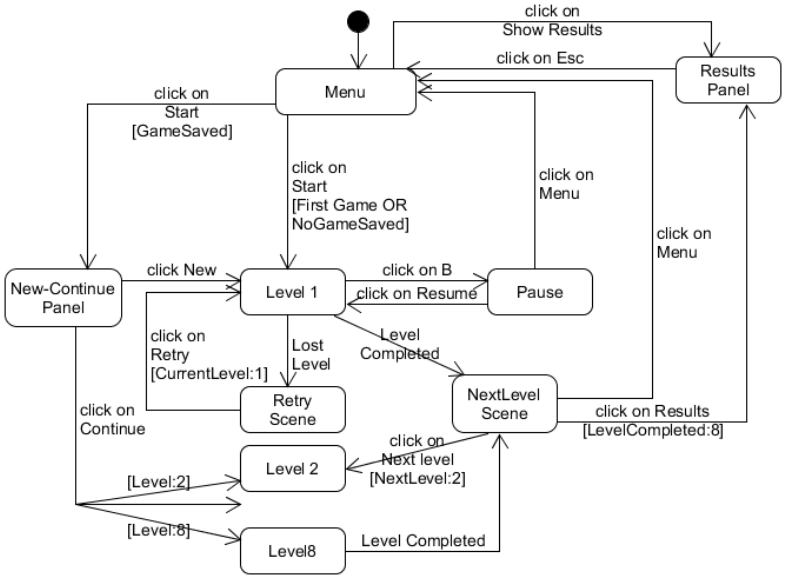

Figura 4.1: *Macchina a stati che mostra il funzionamento del gioco: il caricamento dei diversi livelli e delle varie scene di gioco*

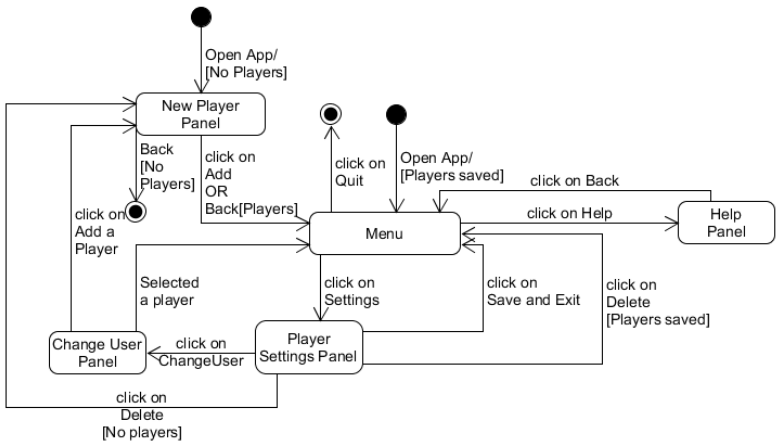

Figura 4.2: *Macchina a stati che mostra il funzionamento del menu principale*

Ogni scena è quindi popolata da GameObjects a cui possono essere associati diversi scripts che regolano il loro comportamento all'interno del gioco o consentono l'interazione con l'utente. Questi scripts contengono sia metodi realizzati ad hoc per la

gestione del gioco, sia metodi standard di Unity opportunamente integrati ed utilizzati per una corretta esecuzione del gioco.

4.2 Metodi Standard di Unity

In Unity tutti gli scripts derivano da una classe base detta MonoBehaviour. Sono inoltre disponibili dei metodi la cui esecuzione avviene secondo un ordine predefinito e che permettono quindi di applicare delle assegnazioni ed eseguire dei metodi secondo un ordine specifico o ciclicamente. Questi metodi, che possono essere utilizzati in ogni script creato, sono:

- Update: esso è invocato ad ogni frame temporale, se il MonoBehaviour dell'oggetto a cui è associato è abilitato. È la funzione che viene maggiormente utilizzata per l'implementazione di uno script di gioco proprio perché permette di verificare periodicamente delle condizioni/stati ed eseguire ciclicamente delle azioni;

- Start: tale metodo è invocato solamente nel frame temporale in cui uno script viene abilitato, esso viene eseguito prima di un qualsiasi metodo di Update. È quindi invocato una sola volta nel periodo di "vita" dello script;

- Awake: tale metodo è invocato quando lo script di ogni oggetto è caricato. Tale funzione è invocata su tutti gli oggetti prima dell'esecuzione di una qualsiasi funzione di Start. Essa permette di inizializzare una qualsiasi variabile o stato del gioco prima che il gioco abbia inizio. Viene invocata solo una volta nel corso della "vita" dello script, inoltre dato che viene invocata dopo la creazione di tutti gli oggetti della scena in esecuzione può agire e fare riferimento ad uno qualsiasi di essi. L'ordine di esecuzione delle diverse funzioni Awake che sono associate ai diversi GameObjects avviene in modo casuale, tuttavia è possibile impostare manualmente tale ordine, qualora i diversi oggetti siano tra loro collegati, modificando le ProjectSettings di Unity (questo meccanismo è stato effettuato per gestire l'ordine con cui sono inizializzate le variabili per la gestione delle preferenze del giocatore e della penalizzazione gestite nella libreria 3D4amb-Lib di cui si tratterà nel paragrafo 4.9 Libreria 3D4Amb-Lib a pagina 52). Tuttavia, se gli oggetti sono istanziati durante la scena di gioco, la funzione di Awake sarà invocata dopo le funzioni di Start.

Questi metodi sono molto utilizzati nella realizzazione dei diversi scripts per regolare il comportamento degli oggetti presenti nelle scene di gioco.

4.3 Camere

Lo scopo del gioco è, come detto in precedenza, la realizzazione di un trattamento per la cura dell'ambliopia. È quindi necessario gestire la visione binoculare, cioè mostrare ad ogni occhio la stessa scena, per farlo sono possibili due metodologie differenti:

- Duplicazione della scena: questo comporta la duplicazione di ogni singolo oggetto e componente della scena stessa;
- Utilizzo di due camere: sono presenti due camere diverse orientate nello stesso modo e quindi aventi la stessa visuale della scena che è mostrata solo su una parte dello schermo del cellulare.

Il gioco è stato realizzato utilizzando il secondo metodo, esso infatti permette una gestione più semplice della realizzazione del gioco stesso, senza la necessità di duplicare tutti gli oggetti presenti ed evitando uno spreco di risorse. Inoltre, tale metodo consente la separazione della fase di creazione della scena di gioco dalla fase di gestione della visualizzazione binoculare. La visione binoculare è stata gestita creando due camere, una per la visualizzazione da parte dell'occhio sinistro ed una per quella da parte dell'occhio destro; la penalizzazione invece è stata effettuata per mezzo di due scripts che modificano opportunamente gli oggetti visualizzati da una delle due camere, in base all'occhio impostato dal giocatore come sano. Le camere hanno la stessa inquadratura della scena, ma la mostrano solo in una parte dello schermo. In entrambe le camere sono state, infatti, impostate le dimensioni della finestra di visualizzazione pari esattamente alla metà dello schermo e rispettivamente la metà a sinistra per la camera destinata alla visualizzazione dell'occhio sinistro e quella a destra per quella associata all'occhio destro.

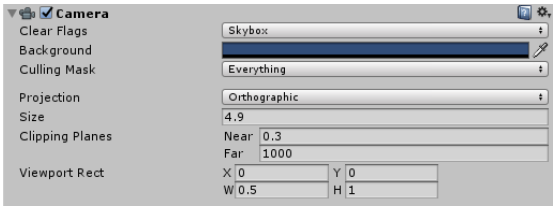

Figura 4.3: *Esempio di impostazioni della Camera Sinistra.*

Gli scripts per la gestione della penalizzazione, che sono associati alle rispettive camere, fanno riferimento alla libreria 3D4Amb-Lib e per tale motivo sono trattati nel paragrafo 4.9 Libreria 3D4Amb-Lib a pagina 52.
Invece lo spostamento delle camere in funzione del movimento del player è stato gestito per mezzo dello script CameraFollow:

```
1  public class CameraFollow : MonoBehaviour {
2
3    private GameObject player;
4    private float xmin;
5    private float xmax;
6    private float ymin;
7    private float ymax;
8
9    void Start () {
10     if(ActiveCharacter.index==0)
11       player = GameObject.FindGameObjectWithTag("PlayerBoy");
12     else
13       player = GameObject.FindGameObjectWithTag("PlayerGirl");
14   }
15
16   void LateUpdate () {
17     float x=Mathf.Clamp(player.transform.position.x, xmin, xmax);
18     float y=Mathf.Clamp(player.transform.position.y, ymin, ymax);
19
20     if ((gameObject.transform.position.x)< x)
21       gameObject.transform.position = new Vector3(x, y,
             gameObject.transform.position.z);
22     else
23       gameObject.transform.position = new
             Vector3(gameObject.transform.localPosition.x, y,
             gameObject.transform.position.z);
24   }
25 }
```

In questo script viene inizialmente determinato, all'interno del metodo standard di Unity Start(), quale avatar è attivo (è infatti possibile scegliere tra due diversi avatar) e quindi quale dovrà essere seguito dalla camera. Il secondo metodo implementato in questa classe è un altro metodo standard di Unity, LateUpdate(), esso è simile ad Update(), in quanto anch'esso è invocato ad ogni frame temporale; tuttavia, è eseguito in seguito all'esecuzione di tutte le funzioni Update; questo consente di definire un ordine di esecuzione tra i diversi metodi. In tal modo è possibile consentire lo spostamento della camera solo in base al movimento dell'avatar, che viene definito dal giocatore e gestito per mezzo dello script PlayerController (la trattazione di tale script è fatta nel paragrafo 4.4.2Player Controller a pagina 38). La funzionalità base dello script CameraFollow è quindi verificare se il giocatore è in movimento, e cioè se la sua posizione è variata, al verificarsi di tale situazione la camera si sposta conseguentemente. Tuttavia, tale spostamento si verifica solo se l'avatar del giocatore si sposta verso destra, al contrario se il suo spostamento avviene verso sinistra la camera rimane ferma: il giocatore

può tornare indietro nella scena al massimo fino ai limiti della scena mostrata dalla camera stessa (lines 20-23 script precedente). In tal modo l'avatar del player mentre prosegue la sua movimentazione verso destra sarà sempre al centro della camera consentendo al giocatore di avere una corretta visuale del tratto successivo della scena.

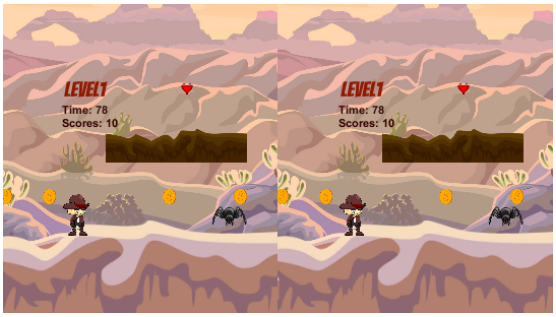

Figura 4.4: *Esempio di sdoppiamento dell'immagine di gioco per mezzo di due camere.*

4.4 Player Scripts

Uno dei componenti più importanti all'interno delle scene di gioco è il player, cioè l'avatar che attraverso i comandi impartiti dal giocatore deve attraversare l'ambientazione di gioco e interagire con gli oggetti che la compongono. Questo elemento è gestito attraverso tre scripts principali: ActiveCharacter, PlayerController e LimitXFollower.

4.4.1 Active Character

Come detto in precedenza ogni giocatore ha la possibilità di scegliere tra due avatar differenti. Questa scelta è effettuata alla creazione di un nuovo giocatore, tale argomento è trattato nel paragrafo 4.9Libreria 3D4Amb-Lib a pagina 52.
In ogni scena di gioco verrà, quindi, visualizzato l'avatar scelto dal giocatore che sta effettuando la partita. Per attivare l'avatar corretto, in ogni livello è stato creato un GameObject, "Character_List", contenente i due GameObjects che rappresentano gli avatar tra cui si può scegliere. Ogni avatar è caratterizzato dal suo sprite[1], dagli scripts e dalle proprietà necessarie per la sua corretta interazione con il gioco. In particolar modo l'attivazione all'interno della scena dell'avatar, scelto dal giocatore, è permessa attraverso lo script ActiveCharacter, legato al GameObject "Character_List":

[1]Sprite: un'immagine, generalmente bidimensionale (2D), che fa parte di una scena più grande (lo "sfondo") e che può essere spostata in maniera indipendente rispetto ad essa; può essere sia statica che dinamica.

```
1  public class ActiveCharacter : MonoBehaviour {
2    private GameObject[] characterList;
3    public static int index;
4    PrefManager pm;
5
6    private void Awake(){
7      pm = GameObject.Find("PrefManager").GetComponent<PrefManager>();
8      index = pm.actualPlayer.IdAvatar;
9    }
10
11   private void Start(){
12     characterList = new GameObject[transform.childCount];
13
14     for (int i = 0; i < transform.childCount; i++)
15       characterList[i] = transform.GetChild(i).gameObject;
16
17     foreach (GameObject go in characterList)
18       go.SetActive(false);
19
20     characterList[index].SetActive(true);
21   }
22 }
```

Unity consente di gestire l'insieme di GameObjects contenuti in "Character_List" come una vera e propria lista di GameObjects. Per l'attivazione dell'avatar è necessario innanzitutto determinare quale avatar è stato scelto dal giocatore; questo è effettuato all'interno del metodo Awake(). In esso viene effettuata la ricerca del GameObject "PrefManager" (line 7) al quale è associato uno script che si occupa della gestione delle preferenze impostate dal giocatore. Tale script fa parte della libreria 3D4Amb per tale ragione verrà trattato nel dettaglio nel paragrafo 4.9Libreria 3D4Amb-Lib a pagina 52. Per mezzo del GameObject "PrefManager" e in particolar modo attraverso il suo attributo "actualPlayer" è possibile determinare l'avatar associato al giocatore, tale attributo fornisce infatti l'indice identificativo dell'avatar scelto. Nel metodo Start() viene inizialmente creata una lista di GameObjects avente dimensione pari agli oggetti che sono contenuti nel "Character_List" (tale operazione è eseguita per mezzo di transform.childCount, esso è un metodo della classe Transform[2]). Con il ciclo for successivo viene quindi semplicemente riempita la lista di GameObjects appena creata con i GameObjects contenuti in "Character_List" e cioè con gli objects associati agli avatar; questo è effettuato per mezzo dei metodi della classe Trasform (lines 14-15). In seguito, con il ciclo foreach, sono disattivati tutti gli oggetti della lista appena creata.

[2]Transform: classe di Unity utilizzata per memorizzare e manipolare la posizione, rotazione e scala di un oggetto. Tutti gli oggetti hanno un attributo Transform.

In Unity è infatti possibile attivare e disattivare i vari oggetti nella scena: nel momento in cui un oggetto è disattivato oltre a non essere visibile non avrà alcun ruolo nella scena, ciò implica che non saranno eseguiti i metodi degli script ad esso associati. Infine, viene riattivato il GameObject, l'avatar, la cui posizione nella lista corrisponde all'indice dell'avatar scelto dal giocatore. I GameObjects corrispondenti agli avatar sono stati infatti inseriti nel "Character_List" nello stesso ordine utilizzato per l'assegnazione degli indici agli avatar, allo scopo di garantire una semplice e corretta selezione dell'avatar nella scena.

4.4.2 Player Controller

Lo script cardine del corretto funzionamento del gioco è "PlayerController". Esso è legato in ogni scena ad entrambi gli avatar. Tale script, come il precedente, è caratterizzato da un metodo Start() con cui vengono inizializzate le variabili caratterizzanti il livello in esecuzione: in ogni livello il giocatore ha una sola vita iniziale, la velocità di movimento dell'avatar varia in funzione della scelta effettuata dal giocatore (line 7 dello script seguente), quest'ultima influenza anche il tempo a disposizione del giocatore per completare il livello (line 8 dello script seguente). Ho effettuato tale scelta in quanto l'impostazione di un tempo massimo fisso, cioè indipendente dalla velocità scelta dal giocatore, complica notevolmente la gestione delle situazioni limite. Infatti, calcolando tale tempo in base alla minima velocità consentita si otterrebbe una riduzione della difficoltà del gioco qualora la velocità impostata dal giocatore fosse massima, al contrario se il tempo fosse impostato in funzione della massima velocità consentita, risulterebbe troppo difficile o quasi impossibile terminare il livello scegliendo la velocità minima, inoltre anche l'utilizzo di un tempo medio non è risultato equo e stimolante per il giocatore.
La seconda parte del metodo si basa sull'attivazione della grafica delle vite disponibili (grafica in alto alla schermata di gioco), tale procedura avviene con lo stesso meccanismo usato nello script ActiveCharacter. È necessario gestire separatamente la grafica visualizzata dalle due camere, infatti, per la visualizzazione degli oggetti grafici sono state utilizzate le Canvas[3]. Ogni Canvas è associata ad una delle due camere presenti nella scena, per tale motivo è necessaria la replicazione dei contenuti della scena (le interfacce), che ognuna delle due Canvas mostra separatamente attraverso una specifica camera. Un'alternativa all'utilizzo delle Canvas poteva essere sfruttare il fatto che GameObjects standard non necessitano di essere duplicati, perché non sono associati

[3]Canvas: è l'area che contiene tutti gli elementi UI. Canvas è un GameObject avente un componente interno di tipo canvas e tutti gli elementi UI devono essere child di tale GameObject per essere visualizzati dalla camera a cui è associata la Canvas; ogni Canvas è infatti associata ad una specifica camera della scena.

ad una camera specifica, ma sono nella visuale complessiva della scena. Tuttavia, ho scelto di utilizzare le Canvas nella gestione della grafica UI perché esse permettono una gestione più comoda, anche se in parte più onerosa, delle parti testuali e della loro modifica da script; quest'ultima funzionalità è stata utilizzata per mostrare lo scorrere del tempo durante l'esecuzione del gioco e l'incremento del punteggio.

```
1  private void Start() {
2
3    sessionmanager
         =GameObject.Find("SessionManager").GetComponent<SessionManager>();
4    prefmanager =
         GameObject.Find("PrefManager").GetComponent<PrefManager>();
5
6    lives = 1;
7    playerSpeed = playerSpeed *
         prefmanager.actualPlayerSettings.speed;
8    timeLeft = 60 + (1-prefmanager.actualPlayerSettings.speed)*60;
9
10   foreach (GameObject go in livesList1)
11     go.SetActive(false);
12
13   if (livesList1[0])
14     livesList1[0].SetActive(true);
15
16   foreach (GameObject go in livesList2)
17     go.SetActive(false);
18
19   if (livesList2[0])
20     livesList2[0].SetActive(true);
21 }
```

La gestione della movimentazione del giocatore e del suo stato è gestita invece per mezzo del metodo Update(). Dato che la sua esecuzione è ciclica, in esso sono eseguite tutte quelle azioni che devono essere aggiornate periodicamente; principalmente viene gestita la movimentazione del player e la sua interazione con i nemici. Viene inoltre decrementato il tempo, questo avviene con la seguente assegnazione:

```
timeLeft = timeLeft - Time.deltaTime;
```

Sono aggiornate le variabili utilizzate per mostrare la grafica (time e scores) e sono gestite le dinamiche di specifici oggetti grafici che non sono fissi, ma che sono attivati solo per un periodo di tempo al verificarsi di un preciso evento e al suo termine sono nuovamente disattivati. Durante il gioco, infatti, in caso di perdita di una vita viene mostrato un "cuore spezzato", in modo analogo la sconfitta di un nemico mostra l'ac-

quisizione di +100 punti, mentre la raccolta dell'oggetto "polvere magica" modifica il colore del player; tutte queste funzioni sono svolte nel metodo Update().

Come detto in precedenza periodicamente viene gestita la movimentazione del player, questo viene effettuato mediante l'invocazione del metodo PlayerMove().

```
1  void PlayerMove() {
2
3    moveX = Input.GetAxis("Horizontal");
4
5    if (moveX < 0.0f)
6      GetComponent<SpriteRenderer>().flipX = true;
7    else if (moveX > 0.0f)
8      GetComponent<SpriteRenderer>().flipX = false;
9
10   gameObject.GetComponent<Rigidbody2D>().velocity = new
       Vector2(moveX * playerSpeed,
       gameObject.GetComponent<Rigidbody2D>().velocity.y);
11
12   if (Input.GetButtonDown("Jump"))
13     gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up *
         playerJumpPower);
14
15   if (moveX != 0 && !(Input.GetButtonDown("Jump")))
16     GetComponent<Animator>().SetBool("isRunning", true);
17   else if (Input.GetButtonDown("Jump"))
18     GetComponent<Animator>().SetBool("isJumping", true);
19   else {
20     GetComponent<Animator>().SetBool("isRunning", false);
21     GetComponent<Animator>().SetBool("isJumping", false);
22   }
23 }
```

Con tale metodo viene quindi gestito lo spostamento del player in funzione dell'input dato dal giocatore, inoltre in base alla direzione dello spostamento viene effettuato il flip dello sprite per fare in modo che il player si direzioni in modo opportuno seguendo gli input forniti (lines 5-8). Nelle righe 12-13 dello script precedente è invece possibile visualizzare la gestione dell'input imposto dal giocatore per saltare.

All'interno di tale metodo avviene inoltre l'opportuno settaggio delle variabili che invocano le diverse animazioni. Le animazioni sono state gestite per mezzo dell'animator controller, un'interfaccia di Unity che permette il controllo delle animazioni di un GameObject. Tali animazioni sono create a partire dall'utilizzo di più sprites, immagini bidimensionali unite nello stesso oggetto, che alternandosi danno l'idea del movimento dell'oggetto a cui si riferiscono. Tale meccanismo è stato sfruttato sia per

la creazione delle animazioni del player che per gli enemies e per tutti gli altri oggetti in "movimento" nelle diverse scene. In particolar modo, per il player (sia per l'avatar in versione femminile che maschile) sono state create cinque animazioni: Idle(), l'animazione iniziale in cui il player è fermo e non compie alcuna azione, Run(), l'animazione che viene impostata quando il giocatore per mezzo del joystick si sposta a destra o sinistra, Jump(), eseguita quando il giocatore preme il tasto A del joystick, Die() animazione attivata nel momento in cui il giocatore ha perso tutte le vite a disposizione e Slide() eseguita alla vittoria di ogni livello. La loro gestione avviene attraverso l'animator controller che si basa su una macchina a stati in cui le condizioni per il passaggio da uno stato all'altro sono delle variabili opportunamente settate all'interno dello script PlayerController(), allo scopo di invocare ogni animazione al momento opportuno. Nello script precedente per esempio si può vedere come viene effettuato il settaggio a true della variabile isRunning (line 16) nel momento in cui il giocatore dà un input con il joystick, questo consentirà all'animator controller di attivare l'animazione corrispondente alla corsa dell'avatar.

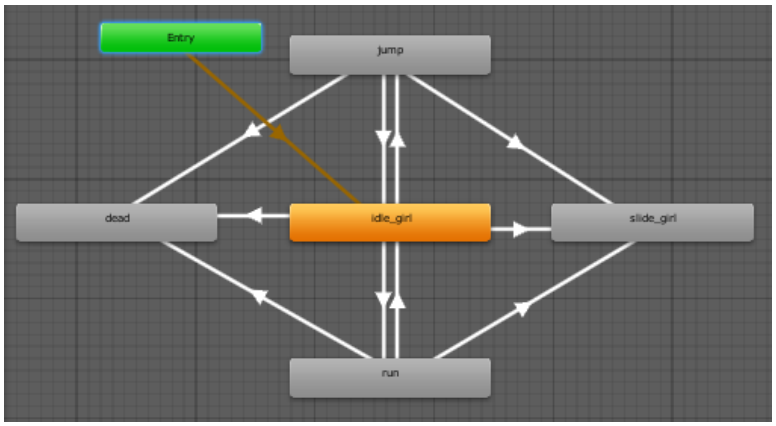

Figura 4.5: *Macchina a stati di Player*

Un'altra funzionalità importante è l'interazione del player con gli enemies, essa viene effettuata attraverso il metodo PlayerRaycast():

```
1 void PlayerRaycast() {
2
3   RaycastHit2D raydown = Physics2D.Raycast(transform.position,
        Vector2.down);
4
5   if (raydown.collider != null && raydown.collider.tag == "enemy"
        && raydown.distance < distanceToBottomOfPlayer) {
6     GetComponent<AudioSource>().volume = 0.5f;
7     GetComponent<AudioSource>().PlayOneShot(EnemyDeath);
8     gameObject.GetComponent<Rigidbody2D>().AddForce(Vector2.up *
          800);
```

```
9
10      if
            (raydown.collider.gameObject.GetComponent<EnemyGroundController>())
11          raydown.collider.gameObject.GetComponent<EnemyGroundController>()
12          .enabled = false;
13      else
14          raydown.collider.gameObject.GetComponent<EnemyAnimated>().enabled
                = false;
15      enemy_death(raydown.collider.gameObject);
16    }
17  }
```

Il controllo della sconfitta dei nemici, che si verifica quando il player salta su di essi, è stato gestito per mezzo del metodo Raycast utilizzabile da tutti gli Objects appartenenti alla classe Phisycs2d (classe predefinita di Unity). Tale metodo è una sorta di sensore lidar, emettitore laser; viene infatti "lanciato" un raggio da un punto specifico verso una determinata direzione all'interno della scena. Ogni oggetto che entra in contatto con questo raggio viene intercettato e identificato. Tale metodo ritorna quindi un oggetto RaycastHit, che è un riferimento all'oggetto intercettato, se invece non è stato intercettato alcun oggetto ritornerà null. Per rilevare se il player ha colpito un nemico viene quindi calcolato ad ogni ciclo di frame il raggio direzionato verso il basso a partire dalla posizione in cui si trova il player in ogni istante (line 3 dello script precedente), successivamente viene quindi fatta una valutazione in base alle informazioni ritornate dal metodo Raycast. In particolar modo, viene verificato che sia stato intercettato un oggetto (il riferimento al collider è presente), se tale oggetto è un enemy (gli oggetti in Unity possono infatti essere identificati per mezzo di un Tag) ed infine se la distanza è minore rispetto ad una determinata soglia, che in seguito a diverse simulazioni è stata individuata come quella più adatta alla gestione di tale funzionalità. In caso quindi tutte le condizioni precedentemente elencate si verificassero significa che il player ha effettivamente colpito un enemy e viene quindi effettuata la sua eliminazione (nello script tale parte viene effettuata sulla base di un precedente controllo sul Tag specifico di enemy in quanto possono essere presenti due diverse tipologie, trattate nel paragrafo 4.5 Enemy Scripts a pagina 45). La gestione della sconfitta del nemico, attivata dopo essere stato colpito, è effettuata per mezzo del metodo enemy_death() :

```
1 void enemy_death(GameObject en) {
2   playerScore += 100;
3   count_enemies++;
4
5   puntiC1.GetComponent<RectTransform>().position = new
        Vector2(transform.position.x-2, transform.position.y);
6   puntiC1.GetComponent<Text>().enabled = true;
```

```
7     puntiC2.GetComponent<RectTransform>().position = new
          Vector2(transform.position.x-2, transform.position.y);
8     puntiC2.GetComponent<Text>().enabled = true;
9
10    mostra_punti = true;
11    Destroy(en);
12  }
```

In questo metodo vengono incrementati i punti ottenuti dal player, viene posizionata la grafica che mostra l'acquisizione dei punti (la "notifica" +100) esattamente accanto al player (lines 5-8), per consentire al giocatore di visualizzare ed essere informato dell'acquisizione di ulteriori punti senza dover controllare ogni volta il punteggio totale che compare in alto nella schermata. Viene inoltre settata la variabile mostra_punti per mezzo della quale viene gestita la successiva disattivazione di questa grafica al passare di pochi secondi. Infine, avviene la distruzione effettiva dell'enemy.

Nello script PlayerController() vengono infine gestite le funzionalità di acquisizione delle monete, degli aiuti extra, delle vite aggiuntive, del contatto con un ostacolo oltre che del raggiungimento della fine del livello per mezzo del metodo OnTriggerEnter2D che informa che il player è entrato in contatto con un collider di tipo trigger. Tale metodo sarà invocato ad ogni collisione da parte del player con questi specifici GameObjects. Questo metodo ha come argomento il collider che lo ha invocato e per mezzo quindi di una serie di if viene determinato a quale categoria (Tag) appartiene tale oggetto al fine di eseguire le azioni opportune. Ogni giocatore durante la partita può potenzialmente avere fino a tre vite a disposizione, per tale motivo, nel momento in cui esso entra in contatto con un nemico o un ostacolo questo scontro non causa direttamente la perdita del livello, ma viene invocato il metodo Die() a cui è rimandato il controllo delle vite a disposizione del player. Qualora il player abbia più di una vita a disposizione, viene decrementata una vita e gestita la relativa animazione grafica, inoltre viene verificato se il player è sotto l'effetto della "magic_dust" e quindi immune da eventuali attacchi che comportano la perdita di vite. Al contrario, nella situazione peggiore in cui il player abbia solo una vita disponibile e non sia sotto l'effetto di magia, oppure se è stato terminato il tempo a disposizione, il livello è perso e viene quindi caricata la scena RetryLevel. Quando invece il giocatore ha raggiunto il limite di fine livello, realizzato per mezzo di un collider, viene invocato il metodo LoadLevel(). Con questo metodo sono calcolati i punti acquisiti nel livello attuale; essi sono determinati dalla somma di tutti i punti derivanti dalle scores, dagli oggetti raccolti e dalla sconfitta di enemies (visibili in alto a sinistra durante lo svolgimento del gioco), a cui si aggiungono punti supplementari calcolati in base al tempo rimanente:

```
playerScore = playerScore + (int)(timeLeft * 10);
```

In questo metodo avviene inoltre il salvataggio del livello attuale, il gioco permette infatti di continuare l'ultima partita che è stata iniziata, ma non completata o di iniziarne una nuova. Tale salvataggio è effettuato per mezzo del metodo AddLResForActualPlayer() (esso è stato da me creato e inserito nella classe PrefManager della libreria 3D4Amb-Lib), di cui verrà parlato nei paragrafi successivi. Il salvataggio del livello è eseguito solamente se non si è giunti all'ultimo livello, in tal caso verrà invece salvata l'intera partita.

4.4.3 Limit_x_Follower

Un altro script correlato al movimento del player è Limit_x_follower. Tale script al contrario di Player Controller si occupa di impedire al giocatore alcuni spostamenti. Principalmente, il suo scopo è quello di evitare che il giocatore possa tornare indietro nella scena; il gioco, come già anticipato, è infatti impostato in modo tale che nel momento in cui il giocatore avanza nella scena non possa tornare nel tratto superato: viene consentito solo un certo limite di azione in "retromarcia". Questo meccanismo è stato implementato in modo similare alla gestione delle camere. La principale differenza riguarda il fatto che l'oggetto, che delimita il movimento del giocatore, rimane sempre ad una specifica distanza sull'asse delle x rispetto alla posizione del giocatore, mentre sull'asse delle y esso segue in ogni istante il giocatore. Anche in questo caso come per le camere lo spostamento dell'oggetto limitante avviene solo quando il player si sposta verso destra (cioè verso valori superiore di x), questo allo scopo di garantire il bloccaggio del giocatore stesso.

```
1  void LateUpdate() {
2
3    float x = Mathf.Clamp(player.transform.position.x, xmin, xmax)-3;
4    float y = player.transform.position.y + 8;
5
6    if((gameObject.transform.position.x) < x)
7      gameObject.transform.position = new Vector3(x,y,
           gameObject.transform.position.z);
8    else
9      gameObject.transform.position = new
           Vector3(gameObject.transform.position.x, y,
           gameObject.transform.position.z);
10 }
```

Per realizzare effettivamente questo meccanismo è stato creato un GameObject avente la struttura di semplice collider e lo scopo di bloccare il Player.
Mentre il player che entra in contatto con questo collider è solamente bloccato, i nemici

che vengono a contatto con esso sono invece distrutti (tale operazione è stata fatta con il metodo offerto da Unity Destroy(GameObject)). Qualora infatti questo meccanismo non fosse implementato tutti questi oggetti verrebbero trascinati dal collider lungo tutta la scena di gioco.

4.5 Enemy Scripts

I nemici sono componenti molto importanti per la realizzazione di un'applicazione più avvincente che invogli l'utente al suo utilizzo. Nel gioco sono presenti due tipologie di nemici; la loro principale differenza riguarda il fatto che i primi non sono dotati di animazioni e in caso di attacco provocano solo la perdita della vita o la perdita del livello del giocatore, al contrario i secondi sono provvisti di animazioni e quindi oltre ad avere lo stesso comportamento dei precedenti viene mostrato un attacco vero e proprio che varia in funzione del nemico che lo effettua. Ad essi sono quindi associati due diversi scripts, che consentono il loro movimento e l'interazione con gli altri oggetti della scena, in particolar modo con il player: EnemyGroundController ed EnemyAnimated. Le principali funzionalità che sono gestite dallo script EnemyGroundController sono: la movimentazione dei nemici, che possono muoversi solo all'interno di uno specifico range dipendente dalla loro posizione iniziale, e il rilevamento del player che deve essere attaccato. La gestione di queste due funzionalità sono state inserite nel metodo Update() di questo script, in quanto è necessario che tali controlli vengano eseguiti periodicamente. In particolar modo la gestione della movimentazione è stata effettuata nel seguente modo:

```
1 transform.position = Vector2.MoveTowards(transform.position,
      target, EnemySpeed * Time.deltaTime);
2
3 if (transform.position.x == max)
4   Flip();
5 else if (transform.position.x == min)
6   Flip();
```

È stato quindi utilizzato il metodo offerto da Unity, MoveTowards: esso consente di spostare un oggetto fino al raggiungimento di un determinato target prestabilito. La movimentazione effettuata inoltre non supera mai una massima velocità, impostata in termini di distanza infinitesima percorsa:

$$EnemySpeed * Time.deltaTime.$$

Allo scopo di consentire il movimento continuo dei nemici tra due target specifici è sta-

to creato il metodo Flip(): esso si occupa di variare opportunamente il target argomento della funzione MoveForward.

```
protected void Flip() {

  if (target.x==max) {
    if (gameObject.GetComponent<SpriteRenderer>())
      gameObject.GetComponent<SpriteRenderer>().flipX = true;
    target.x = min;
  }
  else {
    if (gameObject.GetComponent<SpriteRenderer>())
      gameObject.GetComponent<SpriteRenderer>().flipX = false;
    target.x = max;
  }
}
```

Per esempio, al raggiungimento del target massimo, cioè della massima posizione sull'asse x che può raggiungere l'oggetto enemy, sarà assegnato alla variabile target il valore minimo, cioè la minima posizione raggiungibile dall'enemy, e viceversa. Oltre alla modifica del target in tale metodo viene inoltre capovolto lo sprite, immagine bidimensionale associata al GameObject, allo scopo di mostrare l'effettiva movimentazione del nemico verso la direzione corretta. Tale metodo è invocato anche in caso di collisione con un oggetto diverso dal player (in tal caso verrà effettuato l'attacco) e dal collider limite (cioè dall'oggetto che limita i movimenti del player, in tal caso l'oggetto enemy sarà distrutto).

Un'altra funzione essenziale svolta dai nemici è il rilevamento del player. Quando il player è troppo vicino ad un nemico quest'ultimo procede con l'azione di attacco. Per gestire tale funzione è stato utilizzato un meccanismo analogo a quello eseguito dal player per sconfiggere i nemici: è stato quindi utilizzato il metodo Raycast() che permette, come detto in precedenza, di intercettare e identificare ogni GameObject che si trovi sulla traiettoria di una sorta di raggio laser.

```
RaycastHit2D hitsx=Physics2D.Raycast(transform.position, new
    Vector2(1,0));
RaycastHit2D hitdx=Physics2D.Raycast(transform.position, new
    Vector2(-1, 0));

if (hitsx.distance < dist) {

  if (hitsx.collider!=null&&(hitsx.collider.gameObject.tag ==
      "PlayerBoy" || hitsx.collider.gameObject.tag == "PlayerGirl"))
      {
```

```
7
8       if((hitsx.collider.gameObject.transform.position.x>
            gameObject.transform.position.x))
9         gameObject.GetComponent<SpriteRenderer>().flipX = false;
10
11       gameObject.GetComponent<Rigidbody2D>().constraints =
             RigidbodyConstraints2D.FreezeAll;
12      target.x = transform.position.x;
13
14      PlayerController.enemyattack = true;
15      PlayerController.enemy = gameObject;
16    }
17  }
```

In particolar modo, ad ogni frame temporale vengono aggiornate due variabili che identificano due raggi "trasmessi", uno direzionato verso sinistra ed uno verso destra, per consentire al nemico di rilevare il player sia davanti che dietro di lui (lines 1-2). Qualora quindi si verifichi che il raggio sinistro (per il raggio destro il procedimento è analogo) ha intercettato un oggetto ad una determinata distanza (tale limite, impostato nella variabile dist, è stato determinato dopo una serie di test sul gioco stesso), si procederà a verificare se l'oggetto intercettato è il player. In questo ultimo caso il nemico verrà direzionato verso il player (questa operazione è eseguita solo se la direzione del nemico in quel preciso istante è opposta a quella del player: line 8), il nemico verrà inoltre bloccato nella posizione in cui si trova (per mezzo del comando RigidbodyConstraints.FreezeAll) e verrà modificato il target, allo scopo di fermare il nemico in quel punto. Vengono infine settate le variabili dello script PlayerController per informare il player dell'attacco; sarà infatti tale script, come è stato illustrato nel sottoparagrafo 4.4.2 Player Controller a pagina 38, che gestirà la perdita di una vita del player o dell'intero livello. I nemici a cui è associato questo script non eseguono alcuna azione nel momento in cui si verifica lo stato di attacco e questa è proprio la differenza con lo script EnemyAnimated. In sostanza, nel primo caso i nemici non hanno animazioni, mentre nel secondo ne sono provvisti. Lo script EnemyAnimated, che è associato ai nemici provvisti di animazioni, è una sottoclasse di EnemyGroundController. Essa eredita tutti i suoi metodi e proprio per tale motivo questi ultimi sono stati definiti come protected, consentendo quindi il loro accesso e utilizzo da parte di tutte le sottoclassi di EnemyGroundController. Gli unici metodi che non sono ereditati sono quelli standard di Unity, cioè Start() e Update(), questo perché sono propri della specifica classe. Questi ultimi sono analoghi a quelli presenti in EnemyGroundController e precedentemente descritti, l'unica differenza riguarda il settaggio delle variabili relative alle animazioni. Come per il player anche per gli enemies, la gestione delle anima-

zioni è effettuata per mezzo dell'animator controller. Gli enemies sono caratterizzati solamente da due tipologie di animazioni: Walk(), si tratta dell'animazione iniziale in cui i nemici camminano all'interno di un range predefinito e Attack(), eseguita solo al contatto con il player, tale attacco varia in funzione del nemico che lo attua.

4.6 Pause Manager

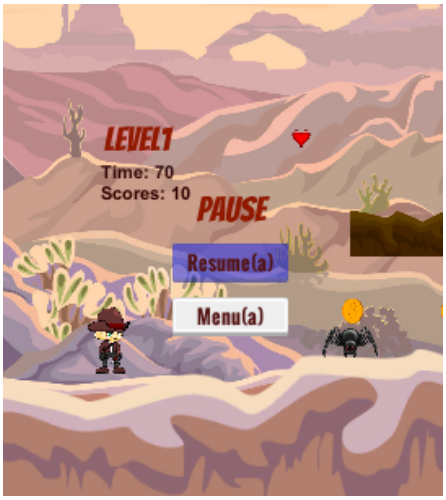

Figura 4.6: *Esempio di schermata di pausa*

Durante l'esecuzione del gioco, il giocatore può sospendere momentaneamente la partita, metterla in pausa. Questa funzione è stata gestita attraverso la creazione di un GameObject PauseManager che fa riferimento al rispettivo script PauseManager. La procedura di pausa del gioco verrà effettuata all'avvenimento di una richiesta di sospensione, effettuata dal giocatore attraverso il remote controller.

```
if (Input.GetButtonDown("Submit") && !paused)
  paused = true;
```

Al verificarsi di tale evento verrà quindi sospeso lo scorrere del tempo (per mezzo di Time.timeScale[4]), verrà interrotta la musica eseguita durante il gioco e verrà mostrata la schermata di pausa, come in Figura 4.6.

```
1 if (paused) {
2   Time.timeScale = 0;
3   canvas1.SetActive(true);
4   canvas2.SetActive(true);
```

[4] Time.timeScale: determina la velocità con cui scorre il tempo, in particolar modo se esso è impostato a 1.0 il tempo scorre alla stessa velocità della realtà. Valori inferiori determinano lo scorrere del tempo più lentamente e viceversa valori superiori riguardano lo scorrere rapido del tempo. Se timeScale è posto a 0 è come se il gioco fosse messo in pausa.

```
5     audioBack.GetComponent<AudioSource>().Pause();
6     audioGirl.GetComponent<AudioSource>().mute = true;
7     audioBoy.GetComponent<AudioSource>().mute = true;
8   }
```

L'uscita dalla modalità di pausa viene impartita dal giocatore: dallo stato di pausa è infatti possibile continuare il gioco o tornare al Menu iniziale (tale schermata sarà presentata successivamente). La scelta di una delle due opzioni è fatta per mezzo dello spostamento in verticale del controller del joystick. Tuttavia, la gestione della pausa non dipende solo dall'input impartito dall'utente, ma anche da una variabile di appoggio: getInput. L'utilizzo di tale variabile è necessario per impedire l'alternarsi continuo delle due opzioni (menu e resume). Questo problema dipende dal fatto che il comando impartito per mezzo del joystick non è eseguito in modo diretto, ma è presente un periodo di transizione (non è cioè presente un passaggio diretto tra lo stato 1 e lo stato 0). Sulla base dei comandi verticali impartiti con il joystick, verrà quindi variata la variabile buttonSelected che permette la modifica dei colori dei pulsanti presenti nella scena e utilizzati per la selezione dell'opzione scelta (lines 8-18 script successivo). La variabile getInput sarà settata a false solo nel momento in cui l'input dell'asse verticale del joystick sarà a zero, cioè solo al termine dell'intera transizione di un comando verticale impartito dall'utente (lines 21-22).

```
1  if (Input.GetAxisRaw("Vertical") != 0 && paused && getInput==false){
2       if (buttonSelected == 0)
3         buttonSelected = 1;
4       else
5         buttonSelected = 0;
6
7       getInput = true;
8       switch (buttonSelected) {
9         case 1:
10        menu_button.GetComponent<SpriteRenderer>().color = new
              Color(15 / 255f, 33 / 255f, 1f, 130 / 255f);
11        resume_button.GetComponent<SpriteRenderer>().color =
              Color.white;
12        break;
13
14        case 0:
15        menu_button.GetComponent<SpriteRenderer>().color =
              Color.white;
16        resume_button.GetComponent<SpriteRenderer>().color = new
              Color(15 / 255f, 33 / 255f, 1f, 130 / 255f);
17        break;
18      }
```

```
19  }
20
21  if (Input.GetAxisRaw("Vertical") == 0)
22    getInput = false;
```

Scegliendo, dalla finestra di pausa, Resume saranno riattivate tutte le funzioni precedentemente sospese e verrà disattivata la schermata di pausa. Principalmente sono quindi eseguiti tali comandi:

```
Time.timeScale = 1;
audioBack.GetComponent<AudioSource>().Play();
```

Al contrario con l'opzione Menu, si tornerà al menu principale e i risultati ottenuti durante quello specifico livello saranno persi.

4.7 Load Next e Retry Level Scripts

Il gioco è inoltre provvisto di apposite scene in cui viene mostrata la vittoria o la perdita di un livello. Alla vittoria di ogni livello, prima del caricamento del livello successivo, viene caricata la scena LoadNext in cui sono mostrati i punti acquisiti nel livello appena terminato e il giocatore può scegliere se continuare il gioco o tornare al menu. I risultati ottenuti fino a quel momento sono già stati salvati nel dispositivo (come detto in precedenza, il salvataggio avviene nello script PlayerController(), vedi sottoparagrafo 4.4.2 Player Controller a pagina 38).

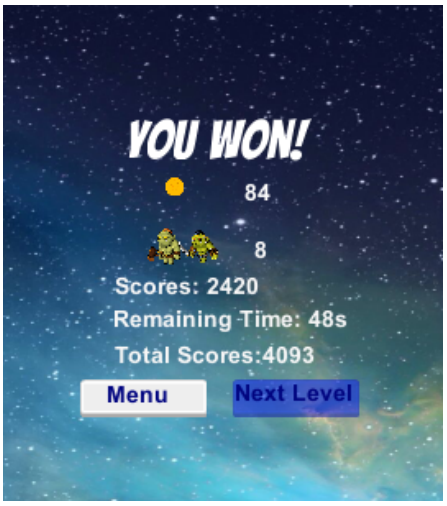

Figura 4.7: *Esempio di scena "LoadNext"*

In questa scena è possibile visualizzare i valori relativi alle scores, al tempo rimanente e ai punti totali acquisiti durante tutti i livelli vinti; essi sono definiti per mezzo dello script PlayerController() durante la fase di salvataggio del livello. Invece, la visua-

lizzazione dell'immagine dei nemici, sconfitti durante il livello, viene gestita con lo stesso meccanismo di ActiveCharacter, vedi sottoparagrafo 4.4.1Active Character a pagina 36. In particolar modo, tra i diversi GameObjects della scena è presente una lista contenente tutti i nemici che dovranno essere visualizzati. A questa lista è associato lo script EnemyView per mezzo del quale viene effettuata la visualizzazione vera e propria: l'attivazione dei due oggetti corrispondenti ai nemici del livello appena vinto.
In caso di sconfitta, invece, si verrà indirizzati alla scena GameOver in cui sarà possibile scegliere se tornare al Menu principale, riprovare il livello appena perso o chiudere l'applicazione.

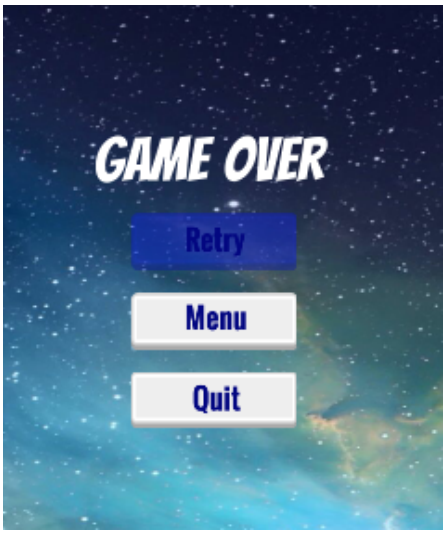

Figura 4.8: *Esempio di "LoadRetry" scene*

4.8 Effetti Sonori

Per consentire un'esperienza di gioco più piacevole e coinvolgente sono stati inseriti degli effetti sonori. Per farlo è stato associato il componente AudioSource() ai GameObjects corrispondenti ai players. Questo componente consente l'attivazione di tutti gli effetti sonori relativi all'acquisizione delle monete, degli aiuti extra, delle vite aggiuntive, alla perdita di una vita, alla perdita del livello e al termine del livello; cioè a tutte le azioni che interessano il player stesso. L'esecuzione di ogni effetto sonoro non avviene in modo continuo, ma solo una volta ad ogni sua invocazione:

```
GetComponent<AudioSource>().PlayOneShot(ItemCollect);
```

Per gestire invece la musica di background è stato realizzato un GameObject specifico contenente un componente AudioSource, a cui è associato l'audio clip da eseguire. In

questo caso la musica verrà ripetuta ciclicamente, la variabile Loop di questo oggetto è stata infatti impostata a true. Il controllo della corretta esecuzione della musica è garantita per mezzo dello script AudioBackground; esso consente l'avvio dell'audio clip al caricamento di ogni scena e della sua interruzione in caso di richiesta di pausa o al raggiungimento della fine del livello.

4.9 Libreria 3D4Amb-Lib

Per stimolare l'utilizzo dell'occhio ambliope, è necessario impedire una visione completa della scena da parte dell'occhio sano, questo viene consentito modificando la visuale da parte della camera associata a questo occhio. Tale funzionalità che è la base dell'utilizzo del gioco per il trattamento dell'ambliopia è stata eseguita attraverso la libreria 3D4Amb-Lib.
3D4Amb-Lib è stata sviluppata da Paolo Cattaneo, laureando magistrale, con l'obiettivo di creare una libreria che permetta di adattare in modo più semplice i giochi, creati per mezzo del motore Unity, per poterli utilizzare come cura dell'ambliopia.
Oltre a tutte le funzionalità che gestiscono la penalizzazione dell'occhio sano, nella libreria sono offerte anche funzioni per la gestione del menu e il salvataggio dei risultati ottenuti durante una partita di gioco, garantendo quindi un utile supporto alla creazione di un gioco completo.

4.9.1 Beta Test

Lo sviluppo del gioco Fantasy World Adventures è avvenuto durante le prime releases di questa libreria, per tale motivo è stato possibile integrarla durante lo sviluppo del gioco e confrontarsi con lo sviluppatore, che è stato informato di bug riscontrati e di possibili miglioramenti. Lo sviluppo del gioco è quindi servito anche come Beta Testing della presente libreria, consentendo il suo miglioramento e l'individuazione di alcune anomalie, visibili solo in seguito ad un suo utilizzo da parte di persone estranee al suo sviluppo.
L'integrazione della libreria è avvenuta a gioco già in gran parte ultimato, per tale motivo alcune funzionalità non sono state sfruttate completamente. Per esempio, lo sdoppiamento dell'immagine per mezzo di due camere era già stato realizzato e per tale motivo è stato effettuato solo un confronto tra i parametri utilizzati nella libreria e quelli da me impostati, adattando solamente eventuali discordanze. All'inizio l'utilizzo di tale libreria si è rivelato leggermente complicato, infatti, essa utilizza numerose classi aventi una gerarchia abbastanza complessa che comporta una difficile comprensione di alcuni attributi; è risultato quindi di fondamentale importanza un'iniziale "studio"

di 3D4Amb-Lib. Tuttavia, dopo questa analisi è stato possibile comprendere il suo funzionamento e adattarla meglio al gioco già realizzato. Utili a questo scopo sono stati anche il manuale e i video guida, forniti dallo sviluppatore, che hanno permesso un utilizzo più semplice della libreria.

Il risultato del Beta Test effettuato è stato nel complesso positivo, 3D4Amb-Lib ha infatti permesso una gestione rapida della penalizzazione; inoltre è stata un utile supporto alla gestione del menu di gioco e delle funzionalità di salvataggio. Il principale aspetto negativo di tale libreria è tuttavia la logica di funzionamento contorta, non immediatamente comprensibile e adattabile ad un gioco già terminato. Per consentire infatti un corretto utilizzo di tutte le funzionalità offerte dalla libreria, senza perdere il lavoro precedentemente fatto, è stata necessaria una fusione quasi completa tra il gioco ed essa. Per tale motivo, un possibile miglioramento della libreria potrebbe essere un manuale più approfondito di utilizzo che consenta una sua comprensione più rapida, evitando di doverla analizzare gerarchicamente.

In conclusione, nonostante alcuni possibili miglioramenti 3D4Amb-Lib è molto importante perché consente di uniformare la gestione della penalizzazione nei diversi giochi. Tale funzionalità è il fulcro per il trattamento dell'ambliopia su cui si basa il progetto 3D4Amb. La libreria permetterà quindi di modificare il meccanismo di cura di tale patologia, in seguito alla scoperta di trattamenti più efficaci, senza la necessità di comprendere il funzionamento di ogni singolo gioco.

4.9.2 Impostazioni utilizzate

Come detto in precedenza la parte fondamentale di tale libreria è la penalizzazione dell'occhio sano. Per realizzare tale funzione la libreria prevede la presenza di due camere, offerte come "prefab[5]". Tuttavia, avendo già realizzato la gestione di due camere differenti, prima dell'integrazione della seguente libreria, non ho utilizzato questi oggetti. Ho invece usufruito degli scripts per mezzo dei quali è realizzata la penalizzazione. Lo script più importante è CameraPenalizer, esso permette di individuare tutti i GameObjects che devono essere penalizzati: aumenta la loro trasparenza nella camera associata all'occhio sano, mentre nella camera associata all'occhio ambliope sono mantenuti opachi e quindi sono visti normalmente. L'individuazione automatica dei GameObjects da penalizzare è realizzata attraverso una lista di tags dei GameObjects oggetto di penalizzazione: ogni GameObject per poter essere penalizzato deve quindi essere associato ad uno di questi tags.

[5]Prefab: particolari GameObjects realizzati in Unity, già impostati (dallo sviluppatore) e pronti per l'utilizzo.

In questo gioco ho scelto di penalizzare tutti i nemici e gli oggetti raccoglibili, mantenendo invece sempre opachi lo sfondo e le piattaforme; sono stati quindi opportunamente aggiornati i tags della lista con quelli da me creati. L'altro script fondamentale per la realizzazione della penalizzazione è DrawEyePatch. Esso si occupa di oscurare la parte di schermo associata all'occhio sano, creando una sorta di occlusione; il suo livello di intensità può essere impostato dal giocatore. La gestione effettiva della penalizzazione è tuttavia consentita per mezzo dello script PenaltyManager che insieme a PenaltyTrigger, si occupa di definire la politica di gestione della penalizzazione: specifica la metodologia utilizzata per incrementare la sua intensità e le difficoltà iniziali e finali. 3D4Amb-Lib consente di gestire la penalizzazione per mezzo di PenaltyManager con una progressione di difficoltà infinita o fissata. Essendo il gioco suddiviso in livelli e avendo offerto al giocatore la possibilità di stabilire i limiti iniziali e finali di penalizzazione è stata ritenuta più adeguata una progressione fissata. La gestione di tale progressione poteva essere svolta manualmente, sulla base di impostazioni scelte in precedenza dallo sviluppatore, o BySteps cioè tale incremento avviene in un insieme di passi. L'opzione ritenuta più adatta è la progressione BySteps; sono stati definiti inoltre sette steps, infatti, nel primo livello non avverrà un incremento della penalizzazione, ma verrà mantenuta quella iniziale impostata dal giocatore. L'incremento della penalizzazione è eseguito per mezzo dello script PenaltyTrigger, precedentemente citato. È possibile effettuare tale incremento in base allo scorrere del tempo o al verificarsi di un evento. Nel gioco in questione la penalizzazione è stata gestita in base al verificarsi di un evento e principalmente è stato realizzato un evento Custom: l'incremento della penalizzazione avverrà nel momento di passaggio al livello successivo. La penalizzazione rimane invece costante durante l'esecuzione di un livello di gioco, per evitare un incremento continuo che potrebbe non consentire l'efficacia del trattamento. La progressione della penalizzazione, che avverrà a partire dal secondo livello, è eseguita per mezzo del metodo IncreasePenaltyNow(int X), che esegue l'aumento della penalizzazione un numero X di volte, dove X è stato impostato esattamente pari a:

$$X = NumeroLivelloCorrente - 1.$$

Ad ogni invocazione di questo metodo le impostazioni di penalizzazioni attuali sono modificate con i seguenti valori:

$$PatchingAttuale = PatchingAttuale + \frac{PatchingFinale - PatchingIniziale}{Steps}$$

$$TraspAttuale = TraspAttuale + \frac{TrasparenzaFinale - TrasparenzaIniziale}{Steps}$$

È quindi evidente come la penalizzazione, pur avvenendo in modo automatico, dipenda dalle impostazioni scelte dal giocatore stesso. Sono infatti gestite tutte le preferenze impostate dal giocatore, che riguardano la scelta dell'occhio da penalizzare, il livello iniziale e finale di trasparenza e di patching, per mezzo dello script PrefManager. Esso insieme a PenaltyManager permette la corretta gestione dei diversi giocatori memorizzando le preferenze impostate e le partite vinte da ognuno di loro. Nel gioco è inoltre molto utilizzato lo script SessionManager, che si occupa di gestire e quindi memorizzare temporaneamente i risultati della sessione corrente.

Figura 4.9: *Esempio di utilizzo di penalizzazione dell'occhio sinistro in due scene diverse della stessa partita. Le impostazioni scelte nel gioco sono: parametri di Eye-Patch e Transparency iniziali pari al 20% e parametri finali al 100%. Nella prima figura essendo la scena di Level3 sono quindi al 42,86%, mentre nel Level6 sono al 77,14%.*

4.9.3 Adattamenti

L'utilizzo quindi della libreria 3D4Amb-Lib ha comportato una sua diretta fusione con gli scripts già realizzati. Tuttavia, alcune sue caratteristiche non si adattavano alla logica del gioco, questo ha reso necessario apportare alcune modifiche.
La principale modifica effettuata riguarda l'oggetto SessionManager, che consente la gestione delle informazioni e dei risultati della sessione di gioco. Esso non è stato progettato per la gestione di livelli, associati a scene diverse, in cui è previsto il salvataggio intermedio delle partite. Per tale motivo è stato necessario impostare tale oggetto come DontDestroyOnLoad, similmente a quanto fatto per l'oggetto PrefManager; in tal

modo questi oggetti, a seguito del caricamento delle diverse scene non sono eliminati, ma permangono durante tutta l'esecuzione dell'applicazione. Questa loro caratteristica comporta la necessità che essi siano presenti solo in una scena: la prima. In tal modo l'intera partita è considerata come una singola sessione di gioco, indipendentemente dal passaggio tra i diversi livelli e schermate.

Il testing del gioco ha consentito di individuare un malfunzionamento nell'utilizzo di DontDestroyOnLoad. Infatti, ogni volta che veniva ricaricata la scena menu (scena in origine contenente tali oggetti), essi venivano ricreati; questo pur permettendo l'esecuzione del gioco senza errori comportava la modifica del giocatore attuale, cioè venivano sempre caricate le informazioni relative all'ultimo giocatore creato.

Per superare tale inconveniente è stata introdotta una scena iniziale "Intro" il cui scopo è la creazione dei GameObjects PrefManager e SessionManager; essi saranno mantenuti durante i passaggi tra le diverse scene, e non verranno duplicati in quanto la scena iniziale non sarà più caricata. Questo aggiustamento ha comportato, inoltre, la modifica dell'istante di invocazione di alcune funzionalità per garantire il corretto funzionamento del gioco. Questa soluzione ha permesso inoltre una corretta gestione dell'eliminazione dei giocatori; funzionalità introdotta nella libreria per consentire al giocatore di eliminare i players creati e non utilizzati.

4.9.4 Funzionalità integrative

Oltre alla pura gestione della penalizzazione, la libreria consente la realizzazione e gestione di un menu principale. Tale menu comprende le funzionalità per l'inizio del gioco, l'impostazione delle preferenze, la consultazione di un help, che introduce al gioco e alla scelta delle impostazioni, ed infine la visualizzazione dei risultati ottenuti dall'attuale giocatore (nella scritta di benvenuto viene indicato il giocatore a cui ci si sta riferendo).

Figura 4.10: *Menu Principale del gioco*

La principale modifica che ho apportato alla configurazione del menu riguarda la funzionalità Start, attraverso cui si può iniziare il gioco. La libreria prevede infatti che, in seguito alla scelta di tale opzione, venga mostrata una scena con cui si invita il giocatore ad indossare un VR e alla pressione di un qualsiasi pulsante viene caricato direttamente il gioco.

Figura 4.11: *Schermata di invito ad indossare il VR*

Per rendere tuttavia più funzionale il gioco e consentire al giocatore di interrompere una partita e continuarla in un momento futuro, ho dovuto apportare delle modifiche a tale meccanismo.
Quando, per il giocatore corrente, è stata salvata una partita iniziata in un momento passato e non ancora terminata, il giocatore potrà scegliere se continuare la partita precedente o iniziarne una nuova. Tale scelta è consentita in seguito alla pressione del pulsante Start. Inoltre, qualora il giocatore scegliesse di iniziare una nuova partita, la perdita dei risultati ottenuti nella partita precedente avverrà solo al completamento di almeno un livello. Se invece venisse scelto di continuare la partita salvata, verrà caricato il primo livello non ancora vinto e saranno ripristinate le impostazioni di penalizzazione settate durante quella partita.
Anche in questo caso prima del caricamento della scena di gioco da continuare, verrà visualizzato un messaggio con cui si esorta l'utente ad indossare il visore: da questo momento in poi infatti lo schermo sarà diviso in due inquadrature. In assenza di partite salvate per il giocatore corrente (questa situazione può verificarsi nel caso si tratti della prima partita del giocatore o al termine del gioco stesso), verrà invece avviato il gioco con la procedura standard implementata dalla libreria.
Il salvataggio intermedio del gioco è gestito in modo automatico e viene eseguito al superamento di ogni livello. Questa funzione è stata aggiunta alla libreria attraverso

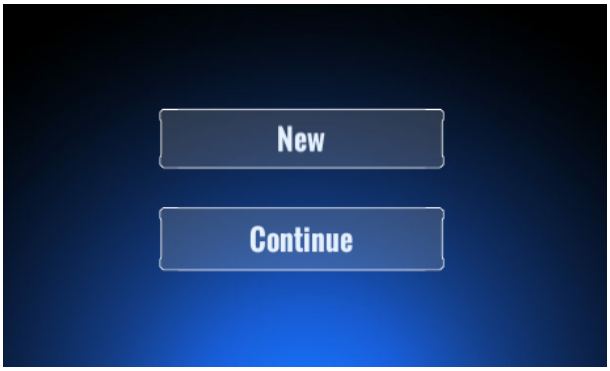

Figura 4.12: *Pannello NewContinue, visualizzabile alla pressione del button Start in presenza di una partita non terminata.*

il metodo AddLResForActualPlayer(SessionResult sr), realizzato con la stessa logica utilizzata per il salvataggio delle partite per mantenere una certa coerenza nella libreria stessa e nei dati salvati. Per poter continuare correttamente una partita è necessario conoscere la penalizzazione impostata e la velocità scelta dal giocatore, per tale motivo è stato necessario creare un nuovo costruttore della classe SessionResult, che si occupa appunto di definire le caratteristiche di una sessione di gioco.

Il salvataggio del livello attuale è stato effettuato, come si può vedere dal codice seguente, per mezzo della classe JsonHelper: essa consente di lavorare con dati in formato JSON, JavaScript Object Notation. In particolar modo per mezzo del metodo ToJson viene generata una rappresentazione in tale formato dell'oggetto in argomento, al contrario per mezzo di FromJson, utilizzato nei metodi con cui si caricano le impostazioni del giocatore, viene creato un oggetto partendo dalla rappresentazione in Json. Nonostante l'utilizzo di un formato adatto allo scambio in remoto, i dati sono salvati in locale. Il salvataggio delle preferenze e di tutti i dati da conservare durante le diverse sessioni di gioco è gestito per mezzo della classe PlayerPrefs. Essa associa ad una chiave identificativa, che nel nostro caso è il semplice nome della preferenza da salvare seguito dal nome del player, un determinato valore, cioè l'oggetto json contenente le informazioni.

```
1 public void AddLResForActualPlayer(SessionResult sr){
2
3   SessionResult[] to;
4   string key_sesres_playername = key_LevResults + "_" +
        actualPlayer.PlayerName;
5   to = new SessionResult[1];
6   to[0] = sr;
7   PenaltyInfo pistart = LoadPlayerPIstart();
8   to[0].eyeP_Start = pistart.PenaltyEyePatch;
9   to[0].trans_Start = pistart.PenaltyTransparency;
```

```
10    PenaltyInfo piend = LoadPlayerPIend();
11    to[0].eyeP_End = piend.PenaltyEyePatch;
12    to[0].trans_End = piend.PenaltyTransparency;
13    string sres = JsonHelper.ToJson(to);
14    PlayerPrefs.SetString(key_sesres_playername, sres);
15
16  }
```

Un'altra integrazione che ho apportato alla libreria è stata la funzionalità di eliminazione di un giocatore. Tale opzione è stata inserita nella lista delle preferenze del giocatore corrente; in questo menu è possibile cambiare giocatore (questa scelta apre un pannello in cui si può scegliere tra i giocatori esistenti o creare un nuovo utente), scegliere le proprie preferenze di penalizzazione e la velocità con cui deve correre il player nel gioco, salvare le impostazioni modificate ed infine cancellare il giocatore corrente.

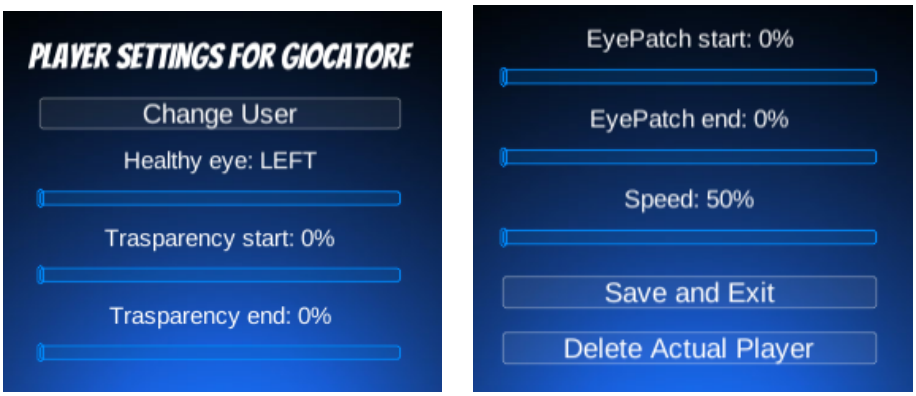

Figura 4.13: *Pannello in cui si possono modificare la velocità e le impostazioni di trattamento dell'ambliopia: l'occhio sano, la trasparenza e l'occlusione. Ci sono inoltre le opzioni per cambiare, salvare ed eliminare un giocatore.*

L'eliminazione di un giocatore è realizzata per mezzo del metodo RemoveActualPlayer() in tre fasi: individuazione del giocatore da eliminare, eliminazione del giocatore, caricamento di un altro giocatore.
L'individuazione del giocatore da eliminare è stata svolta attraverso un ciclo for, con il quale è stato possibile individuare nella lista dei giocatori salvati l'indice di quello da eliminare. È stato quindi necessario creare una nuova lista di giocatori, escludendo il giocatore appena individuato, per mezzo di due cicli for: uno per inserire nella nuova lista di giocatori tutti gli utenti che precedono in posizione il giocatore da eliminare ed uno per l'inserimento dei giocatori che seguono quest'ultimo. Tale lista è quindi memorizzata utilizzando i metodi della classe PlayerPrefs.
Dopo aver creato una nuova lista di giocatori in cui non compare il giocatore da eliminare, è necessario eliminare tutte le informazioni ad esso associate: i risultati ottenuti

nelle diverse partite, le preferenze impostate, un'eventuale partita non conclusa (se presente). Quest'ultima eliminazione è stata svolta per mezzo di un nuovo metodo introdotto nella libreria: RemoveLResforActualPlayer(), il cui comportamento è similare a quello già presente di rimozione delle sessioni complete.
Infine, avviene il caricamento di un nuovo giocatore: in seguito all'eliminazione del giocatore corrente viene ricaricato il menu e tutte le impostazioni che sono presenti fanno riferimento al giocatore corrente, che corrisponde all'ultimo giocatore inserito nella lista dei giocatori appena creata. Tuttavia, se il giocatore appena eliminato fosse l'ultimo e quindi la lista creata fosse vuota, viene informato lo script DeletePlayer(), che ha invocato questo metodo di eliminazione, settando la variabile noplayer a true. Tale script invece di attivare il pannello del menu, come avverrebbe in caso contrario, apre il pannello per la creazione di un nuovo giocatore. Per poter continuare il gioco, l'utente è quindi obbligato ad inserire un giocatore.
Tale pannello sarà inoltre visualizzato anche al primo utilizzo dell'applicazione; in questa situazione, infatti, non sono ancora presenti dei giocatori e per poter procedere nel gioco è necessario crearne uno.

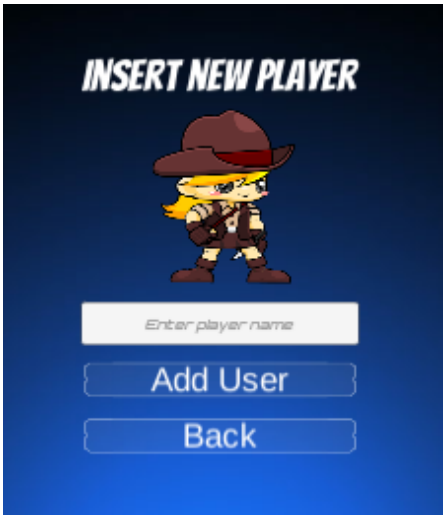

Figura 4.14: *Pannello per inserire un nuovo giocatore*

Conclusione

Il lavoro oggetto della presente tesi aveva come obiettivo la realizzazione di una nuova applicazione per il trattamento dell'ambliopia all'interno del progetto 3D4Amb. In seguito, all'esecuzione di opportuni test e prove, effettuati anche da parte degli altri tesisti che cooperano al progetto 3D4Amb, è possibile affermare che il gioco realizzato rispecchia gli obiettivi prefissati.

La sua semplicità di utilizzo lo rende adatto ai bambini; la grafica e gli effetti sonori tipici dei cartoni animati permettono, inoltre, un'esperienza di gioco più completa e divertente anche per i più piccoli. Il gioco è accessibile a tutti: per poterlo utilizzare è sufficiente avere uno smartphone, un VR e un joystick, strumenti facilmente reperibili sul mercato a modici prezzi.

Il trattamento dell'ambliopia personalizzabile permette una cura più efficace: ogni utente può impostare la modalità di gioco in base alle terapie imposte dal medico. Questa flessibilità dell'applicazione la rende ottimale per la cura di questa patologia, in cui il trattamento varia in funzione della gravità della malattia e delle caratteristiche del paziente stesso. Un altro vantaggio importante riguarda la possibilità di gestire più utenti, questo permette l'utilizzo del gioco da parte di più bambini per mezzo dello stesso dispositivo. L'applicazione è quindi funzionale anche per famiglie in cui più fratelli sono affetti da tale patologia; questa situazione anche se non troppo comune è comunque possibile in quanto l'ambliopia è una malattia ereditaria. Il salvataggio dei risultati, oltre ad essere utile per rendere il gioco più competitivo e quindi invogliare l'utente a cercare di superare continuamente il proprio record, permette di mostrare i miglioramenti conseguiti evidenziando anche la difficoltà di gioco impostata.

Oltre al soddisfacimento di tutti i requisiti funzionali richiesti, Fantasy World Adventures presenta le caratteristiche principali dei giochi moderni: consente infatti di riprendere una partita non terminata, di mettere in pausa il gioco, di variare la difficoltà del gioco non solo in termini di trattamento dell'ambliopia, ma anche in termini ludici, variando cioè la velocità di azione del giocatore. L'applicazione è inoltre portabile, essa è stata realizzata per Android, ma l'utilizzo di Unity, un software multipiattaforma, la rende distribuibile per altri sistemi, come per esempio iOS, per i quali tuttavia non è

ancora stato effettuato il testing.
Il gioco ha inoltre fornito un contributo importante nella fase di testing della libreria 3D4Amb-Lib. I miglioramenti apportati potranno infatti permettere un suo semplice utilizzo da parte di futuri partecipanti al progetto 3D4Amb.
Come tutte le applicazioni, anche Fantasy World Adventures è migliorabile; i possibili sviluppi futuri possono infatti essere:

- Incrementare il numero di livelli per rendere l'esperienza di gioco sempre più avvincente;
- Migliorare la grafica e le animazioni presenti nel gioco, oltre che aggiungere nuove caratteristiche, azioni eseguibili da parte del giocatore per rendere il gioco più completo;
- Salvare i risultati raggiunti non solo in locale, ma anche su server, consentendo un controllo diretto anche da parte del medico;
- Testing dell'applicazione anche su sistemi operativi differenti.

In conclusione, Fantasy World Adventures nonostante le possibili migliorie è adatto al trattamento dell'ambliopia e quindi potrebbe contribuire a facilitare la cura di questa patologia nei bambini che difficilmente sopportano i metodi di occlusione tradizionali.

Sitografia

[1] Ministero della Salute: http://www.salute.gov.it/

[2] Humanitas - Research Hospital:
http://www.humanitas.it/malattie/ambliopia-occhio-pigro

[3] SOI - Società Oftalmologica Italiana:
http://www.soiweb.com/patologie_visive.php

[4] Francesco Vargellini, optometrista e docente: http://www.vargellini.it/zaccagnini/download/privatisti%201&2/dispense%20esterne%20IBZ/5.%20LA%20VISIONE%20BINOCULARE.pdf

[5] RevitalVision: http://www.revitalvision.it/

[6] RieducazioneVisiva:
http://www.rieducazionevisiva.it/metodo/visione-binoculare/

[7] Pharmamedix: http://www.pharmamedix.com/

[8] 3D4Amb: https://3d4amb.unibg.it/

[9] Unity: https://docs.unity3d.com/

[10] GameArt2D: https://www.gameart2d.com/freebies.html

[11] CraftPix: https://craftpix.net/

[12] Unity-AssetStore: https://assetstore.unity.com/

[13] OpenGameArt: https://opengameart.org/