Sommario

Capitolo 1: Introduzione

Il meccanismo della vista è uno fra i più affascinanti e complessi del corpo umano: la luce penetra nell'occhio attraverso la pupilla, e per mezzo del cristallino e della cornea viene messa a fuoco sulla retina. Successivamente le immagini acquisite vengono convertite in impulsi elettrici, e attraverso i nervi ottici vengono trasportate dapprima ai corpi genicolati laterali, e da qui alla corteccia cerebrale. È un sistema concepito efficientemente, ma che non sempre funziona in modo totalmente corretto; non è raro dunque imbattersi in uno dei comuni disturbi visivi. Uno di questi, l'ambliopia, è oggetto di questa tesi; è una condizione patologica caratteristica dell'età pediatrica, ed è caratterizzata da una diminuzione più o meno accentuata della capacità visiva di uno o di entrambi gli occhi. L'università degli studi di Bergamo, attraverso il progetto 3D4Amb, lavora da diversi anni su metodi innovativi di diagnosi e trattamento dell'ambliopia, in particolare grazie all'utilizzo di software e di occhiali e visori 3D. L'obiettivo di questa tesi è proprio lo sviluppo di una nuova applicazione, chiamata 3D4Amb Team Racing, che verrà inserita nel progetto 3D4Amb, e sarà disponibile in futuro per il trattamento della patologia. L'applicativo sostanzialmente consiste in un gioco di guida, in cui il compito dell'utente è quello di completare le varie gare prima del termine del tempo messo a disposizione. Basato sui risultati ottenuti nelle più recenti ricerche scientifiche, l'applicazione sfrutta il meccanismo della penalizzazione ottica per stimolare il cervello ad utilizzare le immagini provenienti dall'occhio ambliope, favorendo dunque il recupero della funzionalità visiva.

Nel capitolo successivo verranno illustrate le caratteristiche dell'ambliopia e le terapie attualmente a disposizione per la sua cura. Si descriveranno poi alcuni degli studi scientifici che negli ultimi anni hanno permesso di comprendere meglio la malattia, e una serie di trattamenti innovativi basati proprio sui risultati di tali ricerche.

Nel terzo capitolo della tesi verrà approfondito il progetto 3D4Amb, la sua filosofia, e le tecnologie di cui ha fatto uso nel corso del tempo.

Nel capitolo quattro verrà finalmente spiegato come è stato pensato, progettato, sviluppato e infine testato il gioco 3D4Amb Team Racing; ci si concentrerà soprattutto sugli aspetti più critici della sua realizzazione, sulle difficoltà riscontrate, e sull'implementazione delle funzionalità più importanti. Verranno inoltre descritti i

dispositivi hardware richiesti per l'utilizzo corretto dell'applicazione, e i software di cui si è fatto uso durante lo sviluppo.
Nell'ultimo capitolo si cercherà di fornire delle indicazioni su come migliorare e far evolvere il software, con la speranza che in futuro nuovi tesisti possano continuarne lo sviluppo, offrendo un prodotto sempre migliore per il trattamento dell'ambliopia.

Capitolo 2: L'ambliopia

All'interno di questo capitolo verrà innanzitutto descritto il funzionamento di base del sistema visivo; questo permetterà di introdurre l'argomento principale della tesi, ovvero l'ambliopia. Verranno illustrate le caratteristiche principali della malattia, i suoi sintomi, le cause e le classificazioni. Verranno poi esposti i trattamenti attualmente disponibili per la sua cura e, dopo un piccolo approfondimento sulle ricerche scientifiche portate avanti negli ultimi anni, si illustreranno alcune delle terapie più recenti e innovative.

2.1 Funzionamento del sistema visivo

In un sistema visivo perfettamente funzionante il cervello utilizza le immagini provenienti da entrambi gli occhi; in particolare viene utilizzata maggiormente l'immagine proveniente dall'occhio dominante, mentre l'altra è di supporto, completa la visione e permette la stereoscopia (ossia la capacità di vedere la profondità di quello che ci circonda). L'effetto stereoscopico si genera grazie ad un'importante caratteristica del sistema visivo umano: la visione binoculare. Questa consiste nella partecipazione di entrambi gli occhi alla visione dello stesso oggetto o della stessa area di campo, e si attiva quando la parte del singolo campo visivo monoculare dell'occhio destro può andarsi a sommare alla stessa parte visibile anche dal singolo campo visivo monoculare dell'occhio sinistro, formando appunto una immagine binoculare. La visione binoculare è permessa dalla stereopsi, ovvero da quella capacità percettiva che consente di unire le immagini provenienti dai due occhi e che, considerando la loro diversa posizione nello spazio, permette di generare la visione tridimensionale.

2.2 Caratteristiche dell'ambliopia

L'ambliopia è una diminuzione dell'acutezza visiva, ovvero la capacità dell'occhio di risolvere e percepire dettagli fini di un oggetto, e colpisce il 4-5% dei bambini; nella maggior parte dei casi è monolaterale, cioè colpisce un occhio solo, ma può anche essere bilaterale. Si considera ambliope un occhio che abbia almeno una differenza di 3/10 rispetto all'altro, oppure un'acutezza visiva inferiore ai 3/10. Il termine ambliopia deriva

dal greco, più precisamente da “ops” (che significa “visione”) e “amblyos” (che significa “ottusa”, “pigra”), ed effettivamente il suo nome comune è “occhio pigro”. L'età in cui i bambini sono più sensibili alla patologia coincide coi primi 2 o 3 anni di vita, nel periodo di massima plasticità cerebrale, e questa sensibilità diminuisce gradualmente fino a quando il bambino raggiunge l'età di 8 o 9 anni, quando la maturità visiva può dirsi ormai completata. L’ambliopia è causata da una trasmissione alterata del segnale nervoso fra occhio ambliope e cervello, quest’ultimo di conseguenza inizia ad utilizzare sempre di meno tale occhio, fino a causare la perdita della visione binoculare e della percezione di profondità. Una diagnosi e una terapia precoce possono, nella maggioranza dei casi, curare la patologia e prevenirne i disturbi permanenti in età adulta. Il problema è che non sempre però questa si manifesta in modo evidente, e difficilmente i bambini sono in grado di identificare ed esternare il proprio disagio, dunque per i genitori non è facile accorgersi della sua presenza. Per questo motivo si raccomanda di effettuare la prima visita oculistica ai bambini, anche in assenza di sintomi, entro i 3-4 anni di età, e di ripetere ulteriori controlli della vista perlomeno ogni due anni.

2.3 Cause e classificazioni

Utilizzando diversi criteri l’ambliopia può essere classificata in svariati modi. Se ad esempio si va a considerare l’integrità strutturale dell’occhio possiamo distinguere la patologia in funzionale od organica: nel primo caso si ha che le strutture dell’occhio risultano essere sane e funzionali, nel secondo caso invece la malattia è legata ad un’alterazione delle vie ottiche (ovvero una lesione del globo oculare o delle vie visive cerebrali). Un’altra utile distinzione che si può fare è quella fra ambliopia reversibile e irreversibile, ed è legata alla capacità del sistema visivo di recuperare dalle conseguenze neurofisiologiche di impulsi visivi anomali nella prima infanzia; il recupero può dipendere da diversi fattori: dallo stadio di maturità del sistema visivo in cui inizia a manifestarsi il disturbo, dalla durata dello stesso, e dall’età in cui si inizia la terapia.
Andando ad approfondire le cause di patologia funzionale possiamo parlare di ambliopia strabica o di ambliopia anisometropica: nel primo caso, come si può facilmente intuire, la malattia è causata dallo strabismo; questo è dovuto ad uno squilibrio muscolare che impedisce l’allineamento coordinato dei bulbi oculari. Nel secondo caso la patologia è

causata da una forte anisometropia, ovvero quella condizione per cui gli occhi hanno una rifrazione diversa, e dunque sulle retine vanno a formarsi delle immagini con diversa sfocatura. Le patologie che causano questo tipo di ambliopia sono: la miopia (le immagini si focalizzano in un punto anteriore alla retina), l'ipermetropia (le immagini si focalizzano in un punto posteriore alla retina) e l'astigmatismo (condizione in cui non si ha la presenza di un singolo punto focale).

Andando poi ad approfondire le cause di patologia organica si può individuare l'ambliopia da deprivazione e quella relativa: la prima è causata dalla riduzione o mancanza della stimolazione retinica per alterazioni organiche, quali ptosi palpebrale (abbassamento della palpebra che può coprire la pupilla), opacità di cornea e cristallino, cataratta congenita, terapie occlusive prolungate; questa categoria comprende anche l'ambliopia ametropica e quella da nistagmo. In caso di ambliopia relativa è infine riconoscibile un danno anatomico (come ad esempio una deformità del nervo ottico), ma il livello di acuità visiva è sproporzionato a quest'ultimo; in questa condizione il meccanismo patogenetico è probabilmente da ricondurre ad un certo grado di difformità fra le immagini percepite dai due occhi, che induce il soggetto a privilegiare l'occhio indenne rispetto all'altro.

2.4 Segni e sintomi

Per capire se un bambino ha un occhio pigro è talvolta necessario interpretare alcuni segnali:

- si avvicina eccessivamente al foglio quando disegna o legge
- apre e chiude le palpebre per guardare
- si sfrega continuamente gli occhi
- piega la testa da entrambi i lati frequentemente

Altri sintomi dell'ambliopia possono comprendere:

- movimento involontario di un occhio verso l'interno o verso l'esterno
- bassa sensibilità al contrasto
- bassa sensibilità al movimento
- scarsa percezione della profondità

In molti casi la patologia può essere sospettata in quanto sono evidenti alcuni problemi come lo strabismo, la cataratta congenita e la ptosi delle palpebre, ma si può confermare sempre e solo attraverso una visita oculistica.

2.5 Trattamenti classici

La prima fase della terapia consiste nel correggere il problema di fondo, ovvero ciò che causa l'ambliopia; questo avviene solitamente mediante occhiali o lenti corneali (soprattutto in presenza di anisometropie e vizi di rifrazione), ma in alcuni casi è necessario un intervento chirurgico (se ad esempio la malattia è conseguenza di ptosi, cataratta congenita o strabismo). Successivamente il bambino viene incoraggiato ad usare di nuovo l'occhio più debole, in modo da ripristinare il corretto sviluppo della vista; le tecniche classiche si basano sostanzialmente sull'inibizione dell'occhio sano o sulla stimolazione diretta dell'occhio ambliope, e possono essere così raggruppate:

- Inibizione occhio sano
    - Occlusione con un cerotto
    - Atropinizzazione
    - Penalizzazione ottica
    - Annebbiamento calibrato
- Stimolazione occhio ambliope
    - Stimolazione mediante CAM e GPG
    - Stimolazione mediante Pattern Flicker MF17

2.5.1 Occlusione con un cerotto (patching)

Questa terapia prevede l'applicazione di un cerotto opaco sull'occhio dominante, che costringe il bambino ad utilizzare l'occhio ambliope. A seconda dei casi può essere applicato direttamente sulla pelle o sulla lente degli occhiali. È previsto il suo utilizzo per 3-6 ore al giorno, e per un periodo che può durare da 6 mesi a 2 anni. Il trattamento raramente fallisce, e per questo continua ad essere il più diffuso per la cura della patologia.

Figura 2.1: Trattamento tramite patching

2.5.2 Somministrazione di atropina

Una valida alternativa al bendaggio consiste nel somministrare atropina all'occhio dominante, creando un effetto di sfocatura temporaneo che obbliga il bambino ad utilizzare l'occhio pigro. L'impego dell'atropina permette al bambino di evitare l'imbarazzo e il disagio sociale dato dall'utilizzo del bendaggio, ma di contro può avere alcuni effetti collaterali, quali irritazioni o dolore agli occhi, arrossamento della pelle, tachicardia, fotofobia (sensibilità alla luce) e mal di testa. Per quanto riguarda l'efficacia dei due metodi, è stato dimostrato che non ci sono differenze significative fra il trattamento con atropina rispetto a quello con bendaggio [1].

2.5.3 Penalizzazione ottica

Si raggruppano sotto sotto questa denominazione una serie di trattamenti che sfruttano l'effetto di sfocatura delle immagini nell'occhio fissante ottenuto mediante lenti opportune, in combinazione con l'instillazione di atropina. Possono essere effettuate penalizzazioni per lontano, per vicino, totale, doppia e leggera. La penalizzazione, nelle sue varie modalità, trova impiego soprattutto nei bambini più grandi, anche in età scolare.

2.5.4 Annebbiamento calibrato

Utilizza filtri da applicare alle lente anteposta all'occhio dominante. Tali filtri consistono in una sottile pellicola di plastica autoadesiva traslucida e sono graduati in modo da ridurre l'acutezza visiva ad un valore noto.

2.5.5 CAM e GPG

Sono apparecchiature che vennero sviluppate negli anni ’70-‘80, si basano sulla stimolazione luminosa strutturata dell’occhio ambliope. Ebbero una discreta diffusione, che si interruppe però pochi anni dopo, soprattutto perché la metodica d’uso escludeva completamente l’impiego contemporaneo delle terapie classiche, come l’occlusione. Il loro obiettivo è quello di stimolare le cellule visive corticali, ponendo in lenta rotazione superfici con barre bianche e nere alternate di differenti frequenze spaziali e di diverso contrasto. Nel sistema CAM l’esercizio consiste nel far eseguire al bambino dei disegni su uno schermo trasparente dietro il quale ruota un disco scelto fra sei con barre di differenti frequenze spaziali e di elevato contrasto. Il GPG è una versione modificata del CAM, in cui i dischi rotanti sono sostituiti da uno schermo catodico.

2.5.6 Pattern Flicker MF 17

Lo strumento Pattern Flicker MF17 è stato introdotto recentemente, ed è attualmente utilizzato presso numerosi centri ortottici e di riabilitazione visiva. Si basa sul principio che una luce intermittente con frequenza temporale di poco inferiore alla frequenza critica di fusione, ovvero quella frequenza alla quale una luce alternata viene percepita come continuamente accesa, è in grado di stimolare numerosi elementi della macula, ovvero la zona centrale della retina; l’effetto è particolarmente evidente usando una luce rossa, a cui la macula è particolarmente sensibile. L’unica grossa controindicazione nell’utilizzo del sistema è che la luce lampeggiante stimolante può scatenare una crisi epilettica (solo nei pazienti con epilessia).

Figura 2.2: Pattern Flicker MF 17

2.6 Ricerca e trattamenti innovativi

Negli ultimi anni la ricerca ha fatto grandi passi in avanti nella comprensione degli automatismi che si attivano a livello cerebrale in presenza di ambliopia; parallelamente il mercato ha cominciato a proporre dispositivi tecnologici estremamente potenti a prezzi contenuti (pc, smartphone, tablet, visori per la realtà virtuale). La concomitanza di questi fattori ha permesso lo sviluppo di nuovi sistemi per il trattamento della malattia, alcuni dei quali già diffusi a livello commerciale.

2.6.1 La ricerca scientifica

Una delle scoperte più interessanti riguarda l'evoluzione di trattamenti per gli adulti: le tecniche classiche di occlusione o di atropinizzazione dell'occhio sano infatti sono applicabili solo fino al termine dello sviluppo della vista, quindi fino a circa 8-10 anni, dopodiché perdono di efficacia. Si è dimostrato però che stimolando gli occhi con segnali diversi (dicoptici) il cervello torna ad utilizzare i dati provenienti dall'occhio ambliope [2]; questo tipo di trattamento permette di ridurre il fenomeno della soppressione, ovvero il meccanismo chiave che porta il cervello a non utilizzare più i segnali acquisiti dall'occhio ambliope, e fornisce dunque una speranza di recupero anche agli adulti. Uno studio condotto nel 2013 alla McGill University di Montréal, in Canada, ha confermato la validità del metodo [3]: 18 adulti hanno giocato per due settimane ad una versione particolare del gioco Tetris, sviluppata con l'intento di mandare diversi segnali visivi ai due occhi; 9 pazienti si sono allenati al videogioco utilizzando solo l'occhio più debole, con l'altro bendato, mentre gli altri 9 hanno usato entrambi gli occhi. Dopo il periodo di test il secondo gruppo ha mostrato un netto miglioramento della visione dall'occhio pigro, così come una maggiore percezione della profondità; i risultati sono stati circa 4 volte migliori rispetto a quelli ottenuti dal primo gruppo.

È importante sottolineare che il progetto sviluppato in questa tesi si basa proprio sulle conclusioni di questi studi.

Nel frattempo sono stati sperimentati altri approcci, come dimostra uno studio preliminare eseguito nel 2016 al MIT (Massachusetts Institute of Technology) di Boston: in questo caso si è affrontato il problema come si fa generalmente con uno smartphone o un pc che funziona male, ovvero si è tentato di riavviare il sistema. Lavorando su cuccioli di topo

con un difetto paragonabile all'ambliopia umana, i ricercatori hanno infatti disattivato temporaneamente la retina di entrambi gli occhi utilizzando un anestetico, dimostrando che una volta svanito l'effetto del farmaco gli animali recuperano la normale visione anche nell'occhio pigro. È stato inoltre constatato che la correzione del difetto visivo è stata mantenuta dai topi fino all'età adulta. Chiaramente il lavoro da fare è ancora tanto prima di poter avviare una sperimentazione umana, ma i risultati conseguiti lasciano pensare che anche questo approccio possa essere valido.

2.6.2 Nuovi trattamenti

Negli ultimi anni si è assistito alla nascita di nuovi sistemi per il trattamento dell'ambliopia, alcuni frutto delle più recenti scoperte scientifiche, altri sviluppati sulla base delle conoscenze e degli strumenti del passato. L'elemento che accumuna i nuovi metodi è l'utilizzo della più recente tecnologia, ovvero pc, smartphone e visori per la realtà virtuale; ciò permette a bambini e adulti di portare avanti la propria terapia riabilitativa comodamente da casa.

2.6.2.1 RevitalVision

La tecnica RevitalVision si basa sulla stimolazione visiva. Consiste in un software per pc che, sfruttando il filtro di Gabor, permette di stimolare le connessioni cerebrali a livello corticale, portando ad un miglioramento nella sensibilità al contrasto e nell'acutezza visiva. Il filtro di Gabor, detto anche cerotto Gabor, è un filtro lineare che ha trovato nell'oftalmologia un importante campo di applicazione, dopo che alcuni studi sulla neurostimolazione hanno dimostrato i suoi effetti benefici proprio sui pazienti affetti da ambliopia. Esso consiste in un banco di filtri di diversa scalatura e orientazione, che vengono esposti in successione e su due diversi schermi all'utente. Lo scopo è quello di allenare l'occhio a diverse frequenze spaziali dei cerotti Gabor, a una loro diversificazione spaziale nello schermo, al livello di contrasto e alla durata dell'esposizione. Questa tecnica, che prende il nome di "mascheramento laterale", migliora i difetti della vista perché sensibilizza il rapporto fra immagine visiva captata dall'occhio e il segnale neurale che arriva alla corteccia visiva.

Il sistema è clinicamente e scientificamente provato, e negli Stati Uniti ha ottenuto l'approvazione della FDA (Food and Drug Administration); il miglioramento medio che

si ottiene sui pazienti è di circa due linee sulla tavola ETDRS, che rappresenta lo standard per la misura dell'acutezza visiva, e del 100% della sensibilità al contrasto.

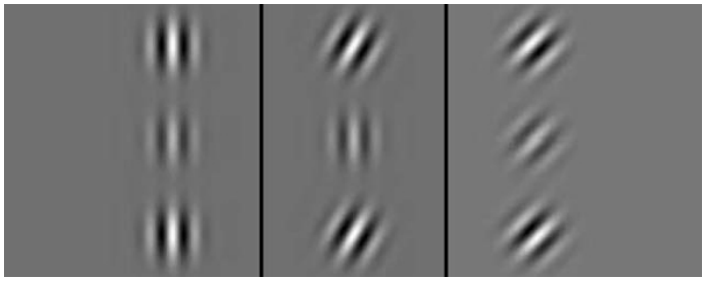

Figura 2.3: I Gabor Patches sul quale si basa il programma RevitalVision

2.6.2.2 Dig Rush

Dig Rush è un videogioco per tablet sviluppato da Amblyotech Inc., una startup statunitense, in collaborazione con Ubisoft, azienda francese leader nello sviluppo e nella produzione di videogiochi, specificatamente per il trattamento dell'ambliopia. Si basa sulle innovazioni brevettate dai ricercatori della McGill University di Montréal al termine dello studio descritto nel paragrafo *La ricerca scientifica*, e successivamente concesse in licenza alla Amblyotech. Mettendo in pratica gli stessi principi il gioco sfrutta entrambi gli occhi a livello binoculare per addestrare il cervello a migliorare l'acutezza visiva del paziente, utilizzando livelli di contrasto di rosso e blu differenti, che possono essere percepiti tramite l'uso di occhiali stereoscopici; il medico può regolare le impostazioni di gioco in base alla debolezza dell'occhio del paziente, consentendo a entrambi gli occhi di usufruire dell'esperienza di gioco. L'applicazione si trova attualmente in una fase avanzata di sperimentazione, Amblyotech attraverso i propri studi clinici la sta testando su più di 500 pazienti, e per i prossimi due anni sono già in programma altre ricerche. Nel frattempo anche all'università di Auckland è in corso un trial per la valutazione del trattamento dell'ambliopia attraverso il gioco Dig Rush [4]: lo studio prevede la somministrazione del trattamento a 108 pazienti dai 7 anni in su, ed è esteso su tre diversi continenti.

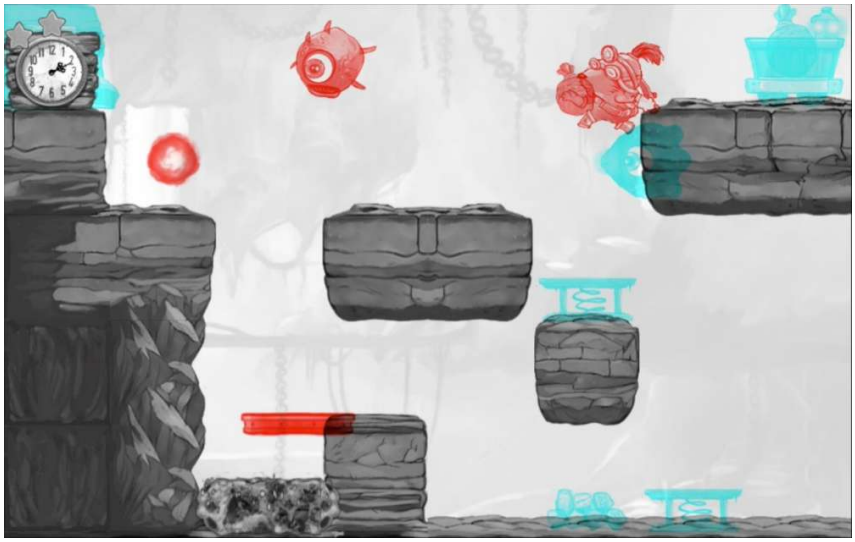

Figura 2.4: Una schermata del gioco Dig Rush

2.6.2.3 Vivid Vision

La proposta Vivid Vision è già presente sul mercato, ed è il sistema che più si avvicina al lavoro portato avanti in questa tesi. Per questo motivo vale la pena descriverlo in modo dettagliato. Esso si basa sull'utilizzo di un visore per la realtà virtuale collegato ad un pc (nella versione Clinical) o ad uno smartphone (nella versione Home), sul quale vengono messi in esecuzione test o giochi; anche in questo caso si favorisce la visione binoculare, ma le immagini presentate agli occhi differiscono di qualche caratteristica o di qualche dettaglio, in modo da stimolare il recupero dell'occhio ambliope.

Figura 2.5: Il sistema Vivid Vision Home

Il sistema Vivid Vision è stato sviluppato seguendo questi sei principi:

- Impegno: è stato dimostrato che l'attenzione visiva aumenta il rapporto segnale-rumore delle rappresentazioni visive all'interno della corteccia cerebrale [5] [6] [7], e influisce dunque sui meccanismi di apprendimento percettivo. Dal momento che tali meccanismi possono migliorare l'ambliopia negli adulti [8] si richiede ai pazienti impegno, e la massima attenzione agli stimoli visivi
- Easy-to-hard: il livello di difficoltà degli esercizi deve essere sufficientemente alto da impegnare il paziente, ma non tanto da renderli ineseguibili
- Input bilanciato: i segnali inviati ai due occhi devono essere tali da potenziare le capacità percettive dell'occhio più debole; questo avviene bilanciando il loro contrasto e/o la loro luminanza (quest'ultima è una grandezza definita come il rapporto tra l'intensità luminosa emessa da una sorgente nella direzione dell'osservatore e l'area apparente della superficie emittente, così come vista dall'osservatore)
- Immagini retiniche corrispondenti: è necessario favorire la fusione sensoriale, cioè quel fenomeno per cui si uniscono in una singola impressione visiva le due immagini di un oggetto che si formano su parti corrispondenti delle retine; ciò avviene facendo in modo che le immagini mandate ai due occhi abbiano una buona corrispondenza binoculare
- Uso della visione periferica, dal momento che l'attenzione sulla periferia influenza la rilevazione degli oggetti da parte dell'occhio [9]
- Integrazione visuo-motoria: dato che la visione binoculare guida il comportamento manuale [10] è utile integrare agli esercizi visivi delle attività di raggiungimento e presa di obiettivi, da compiere in contemporanea

I parametri su cui il medico può agire durante l'esecuzione dei giochi sono:

- Luminanza interoculare/rapporto di contrasto, per ridurre la soppressione interoculare
- Sfocatura
- Offset delle immagini, per la compensazione del disallineamento binoculare
- Dimensione degli oggetti, che può essere aumentata per migliorare la visibilità nell'occhio ambliope

Il sistema Vivid Vision, disponibile attualmente in più di 100 cliniche oculistiche nel nord America, è stato analizzato in due studi preliminari: nel primo [11] 14 pazienti hanno svolto 15 sessioni di gioco della durata di 30-60 minuti in 3 settimane, e fra i risultati è stata rilevata una riduzione del fenomeno di soppressione. Nel secondo [12] 17 pazienti adulti con ambliopia anisometropica hanno svolto 8 sessioni di gioco da 40 minuti l'una, 2 a settimana per 4 settimane; in questo caso sono stati rilevati miglioramenti significativi nell'acutezza visiva e nella stereoacuità (ovvero la misura della stereopsi). Affinché questi risultati positivi si possano confermare sono necessari ulteriori studi, più strutturati e con un maggior numero di pazienti.

2.6.2.4 Binovision

Binovision è la proposta israeliana per la cura dell'ambliopia. È un sistema realizzato da Medisim, azienda fondata nel 1995 che sviluppa, produce e commercializza dispositivi medici per uso domestico e professionale, con brevetti registrati in tutto il mondo. Il dispositivo consiste in un paio di occhiali high-tech collegabile a qualunque sorgente video in streaming (come pc, console di gioco o smartphone), che trasforma cartoni animati, videogiochi, e qualunque altro contenuto in un metodo di cura [13]. Il visore invia il video inalterato all'occhio ambliope, mentre all'occhio sano ne viene trasmessa una versione oscurata.

Figura 2.6: Il sistema Binovision

Anche in questo caso dunque avviene una stimolazione del cervello, progettata per attivarne la plasticità, basata sulla penalizzazione delle immagini mostrate all'occhio dominante. Uno studio preliminare condotto su 15 bambini, a cui è stato chiesto di utilizzare il sistema per 60 minuti al giorno, 6 giorni alla settimana per 3 mesi, ha mostrato come il trattamento sia più efficace rispetto a quello classico con patch o colliri.

Capitolo 3: Il progetto 3D4amb

All'interno di questo capitolo verrà descritto il progetto universitario 3D4Amb, la sua filosofia e i suoi concetti base, e verranno illustrate le tecnologie di cui il progetto ha fatto uso nel corso degli anni per la visualizzazione di immagini 3D.

3.1 Concetti base e requisiti fondamentali

3D4amb è il progetto creato dall'Università degli studi di Bergamo in collaborazione con il centro di ipovisione e di riabilitazione visiva degli Ospedali Riuniti di Bergamo, con lo scopo di sviluppare un sistema basato sul 3D per la diagnosi e il trattamento dell'ambliopia nei bambini piccoli. L'idea alla base è di stimolare il cervello a riattivare l'utilizzo dell'occhio ambliope mediante la presentazione di stimoli dicoptici, concetto che come abbiamo visto in precedenza è stato analizzato in diversi studi e ed è già presente in alcuni prodotti commerciali. I requisiti fondamentali del progetto 3D4amb sono:

- Economicità: il sistema deve avere un basso costo, deve essere economicamente sostenibile per una famiglia; per questo motivo non si prevede l'utilizzo di dispositivi custom, ma si deve basare sull'uso di tecnologia standard, facilmente accessibile
- Facile da usare: gli utenti devono essere in grado di usare il sistema, non è richiesta loro alcuna particolare abilità o livello di istruzione; gli stessi bambini devono poterlo gestire autonomamente, limitando dunque l'intervento degli adulti. Per rendere più facile l'interazione è possibile integrare l'utilizzo di dispositivi come joystick, telecomandi o mouse
- Essere adatto all'uso domestico: deve essere possibile trattare i pazienti a casa, in modo da poter evitare frequenti e lunghe sedute di lavoro in ospedale, e permettendo dunque una maggiore flessibilità nei tempi e negli orari di terapia; il sistema deve essere in grado di registrare i tempi effettivi di utilizzo
- Facilmente estendibile: deve essere possibile sviluppare facilmente nuove applicazioni da aggiungere al sistema; per questo motivo è suggerito l'utilizzo di librerie software aperte e standard

3.2 Tecnologia utilizzate per la visualizzazione del 3D

Il programma nel corso degli anni ha fatto uso di diversi dispositivi per la visualizzazione del 3D; sono stati utilizzati:

- Occhiali Active Shutter: sono occhiali che si collegano direttamente alla tv o al pc, in modo da potersi sincronizzare con lo schermo, e che per funzionare necessitano di batterie o di alimentatore Usb. Lo schermo su cui è attivo il video mostra fotogrammi in sequenza, alternativamente per lente destra e per lente sinistra, e mentre questo accade la lente che non riceve l'immagine viene oscurata. Questa successione avviene così velocemente (ad una frequenza di almeno 60 Hz) che il cervello interpreta i fotogrammi come se provenissero da un'unica immagine stereoscopica, e dunque le fonde in una sola rappresentazione visiva
- Occhiali anaglifici: permettono di visualizzare correttamente gli anaglifi, ovvero immagini stereoscopiche ottenute dalla sovrapposizione di due fotogrammi di colori complementari, leggermente scostate fra loro, che mostrano lo stesso soggetto. Questi occhiali sono costituiti da due lenti colorate (solitamente sono una rossa e una color ciano), ognuna delle quali permette di vedere la parte dell'immagine con filtraggio ad essa complementare. Il loro punto di forza è il bassissimo costo, mentre il loro principale difetto consiste nella scarsa fedeltà cromatica offerta, soprattutto in presenza di colori saturi
- Visori per la realtà virtuale (detti anche visori VR): rappresentano la tecnologia di più recente introduzione per la visione 3D, e grazie alle loro illimitate possibilità d'impiego e al loro costo contenuto sono diventati i dispositivi di maggiore interesse per lo sviluppo di nuove applicazioni per il progetto 3D4Amb. Vista la loro importanza all'interno del progetto, e considerando che il gioco 3D4Amb Team Racing è stato progettato per funzionare proprio con tali dispositivi, i visori 3D verranno successivamente descritti nel dettaglio

Capitolo 4: 3D4Amb Team Racing

In questo ampio capitolo verrà descritto come l'applicazione oggetto di questa tesi è stata pensata, progettata e realizzata. Nella prima parte verrà spiegato come è nata l'idea del gioco, e quali sono le sue funzionalità di base. Attraverso un piccolo approfondimento verranno poi descritti i dispositivi hardware richiesti per l'utilizzo corretto del programma, e gli strumenti software di cui si è fatto uso durante la sua produzione. Ci si concentrerà successivamente sulle varie fasi che hanno portato alla realizzazione del gioco: elencazione di requisiti e casi d'uso, definizione delle regole di progettazione, codifica e testing. All'interno di questa sezione verrà illustrato dettagliatamente come sono stati gestiti gli aspetti più critici e delicati del software, e come sono state implementate sue le principali funzionalità.

4.1 Introduzione

Il gioco 3D4Amb Team Racing è stato sviluppato per il trattamento dell'ambliopia monolaterale e verrà inserito nel progetto 3D4Amb, in modo da arricchire la collezione di software che già comprende. Attraverso questo sarà possibile sperimentare gli effetti di penalizzazioni combinate sull'occhio dominante, e verranno messi a disposizione per futuri lavori una serie di script per la gestione dei dispositivi hardware e per il salvataggio e caricamento dei dati di gioco. Si è cercato di realizzare un prodotto il più possibile professionale, in modo da soddisfare le esigenze videoludiche dei bambini di oggi, abituati ormai a giochi per console e a film d'animazione di altissimo livello tecnico.
L'applicazione è ispirata al famoso gioco per Playstation 1 Crash Team Racing.

Figura 4.1: Una schermata del gioco Crash Team Racing

Lo scopo principale di quel gioco, nella modalità “Avventura”, era quello di vincere una serie di gare di corsa su kart contro sette avversari bot, ovvero personaggi controllati dal gioco stesso, in modo da sbloccare i livelli successivi. Al termine di una sequenza di gare bisognava affrontare una sfida particolare contro uno dei cosiddetti boss, personaggi speciali che comparivano solo in questo tipo di livelli, e solo battendolo si poteva accedere alla successiva serie di gare. I vari percorsi erano spesso molto diversi l’uno dall’altro, si passava da spiagge tropicali a piattaforme sopraelevate, da paesaggi innevati a città antiche, e in ognuno di essi erano presenti delle trappole, come piante carnivore o getti infuocati, che cercavano di rallentare il kart del giocatore. I personaggi attivi durante le corse potevano inoltre colpire e distruggere delle casse di legno, per ottenere e potenziare armi e trabocchetti da usare contro gli avversari. La scelta del percorso che si intendeva avviare avveniva attraverso una mappa base, in cui erano presenti delle piattaforme circolari, ognuna delle quali associata ad una gara. L’utente era completamente libero di muoversi all’interno delle mappe base, e per avviare una gara non doveva fare altro che posizionarsi col suo kart su una di queste piattaforme. Per l’applicazione 3D4Amb Team Racing l’idea iniziale era molto ambiziosa, si voleva ricalcare la stessa modalità di gioco nel minimo dettaglio, ma ben presto ci si è resi conto della dimensione effettiva di un

progetto del genere. Si è deciso dunque successivamente di ridurre la complessità dell'applicazione, rimuovendo i bot e modificando per forza di cose l'obiettivo di gioco.

4.2 Funzionalità di base

Quello che alla fine si è deciso di sviluppare è un gioco di corse, costituito da una serie di percorsi e da una mappa base, in cui grazie ad alcune piattaforme circolari simili a quelle presenti nel gioco Crash Team Racing è possibile selezionare e avviare le diverse gare. L'obiettivo è quello di completare i vari percorsi entro un tempo limite, definito in base alla tipologia e alla lunghezza del tracciato. Lungo il tragitto l'utente può imbattersi in casse di legno che, in base al colore (verde o rosso), se colpite, aggiungono o rimuovono secondi al tempo ancora a diposizione per il completamento del percorso. Al primo avvio dell'applicazione solo la prima gara è disponibile, mentre le altre sono bloccate; è solo completando i vari livelli disponibili che si sbloccano i successivi. Il gioco termina quando tutti le gare vengono completate entro il tempo limite.
Questa breve descrizione ha introdotto come funziona sostanzialmente il gioco nel suo aspetto più videoludico, nel capitolo *Requisiti e specifiche* tutte le funzionalità richieste saranno illustrate nel dettaglio.
Prima di proseguire è bene però ricordare il motivo per cui questa applicazione è stata creata, ovvero il trattamento dell'ambliopia monolaterale, e fornire una piccola panoramica sul modo con cui questo avviene. Il lavoro si è basato sullo studio avvenuto alla McGill University di Montréal nel 2013, già descritto nel capitolo *La ricerca scientifica*, i cui principi già vengono sfruttati nei prodotti Vivid Vision e Binovision. Il software 3D4Amb Team Racing è stato sviluppato per smartphone Android, e per essere utilizzato ha bisogno di un visore mobile per realtà virtuale (i vari tipi di visore verranno analizzati in seguito, nel capitolo *Hardware utilizzato*). Le immagini che compaiono sulle lenti di quest'ultimo sono leggermente diverse fra loro, dal momento che quella mostrata all'occhio dominante subisce un processo di penalizzazione, mentre quella mostrata all'occhio ambliope è sempre priva di modifiche; tutto questo ha lo scopo di stimolare il cervello ad utilizzare proprio tale occhio. In base alla configurazione impostata la penalizzazione può consistere in un oscuramento dell'intero fotogramma, oppure in un effetto di trasparenza assegnato ad oggetti rilevanti presenti nel gioco (come le casse di

legno), oppure in una combinazione dei due meccanismi appena descritti. L'applicazione permette la libera configurazione dei parametri di penalizzazione.

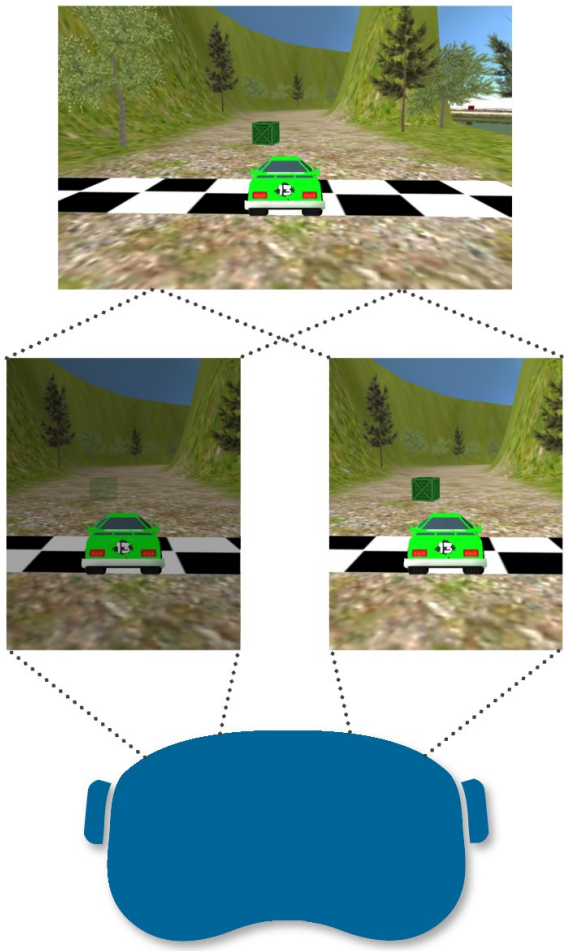

Figura 4.2: Funzionamento penalizzazione con visore 3D

Per quanto riguarda l'interazione dell'utente col gioco sono previste due modalità:

- Direttamente via tocco dello schermo, quando sono visualizzati i menu iniziali
- Attraverso un telecomando Bluetooth, quando è in corso la fase attiva di gioco, e dunque lo smartphone è già posizionato nel visore VR

Il software permette la libera configurazione dei tasti di input, in modo che qualunque telecomando Bluetooth possa essere utilizzato.

Un'ultima importante funzionalità di base che vale la pena citare riguarda il salvataggio dei dati di gioco: l'applicazione è stata sviluppata in modo tale da poter salvare/caricare dati sia su smartphone che su un server online (previa registrazione e login all'avvio della

partita). È bene sottolineare che durante il lavoro fatto per la tesi ci si è concentrati unicamente sul lato client dell'applicazione, la parte server (base dati, web server) sarà messa a disposizione, già pronta, dall'università.

4.3 Hardware utilizzato

Ora che sono state illustrate le caratteristiche principali dell'applicazione sviluppata, e che è stato chiarito quali dispositivi hardware vengono utilizzati, è bene fare un piccolo approfondimento proprio su questi ultimi, in modo da capire meglio come funzionano, e il perché sono stati scelti.

4.3.1 Visori VR

I visori per la realtà virtuale sono fra i dispositivi tecnologici più in voga in questi ultimi anni. Permettono all'utente di essere trasportato in una realtà virtuale parallela, che lo assorbe completamente, estraniandolo dal mondo reale; il concetto è molto diverso da quello di realtà aumentata, che indica la condizione in cui l'utente potenzia la propria percezione del mondo reale grazie a una serie di contenuti digitali. La realtà virtuale ormai è impiegata in una quantità infinita di campi: dai videogiochi al cinema, dall'archeologia alla medicina, dal turismo al settore militare; la rapida diffusione dei visori dipende in buona parte da questa molteplicità d'impiego. In realtà oltre ai visori esistono numerosi dispositivi che permettono l'immersione in mondi virtuali: auricolari, guanti, cybertute, sensori ad infrarossi ed altri ancora; il motivo per cui questi non sono così diffusi è probabilmente legato ad un concetto molto semplice: è la vista il senso più sviluppato negli esseri umani, e dunque è quello che permette di cogliere e vivere meglio l'esperienza virtuale. Il funzionamento base di un visore è molto semplice: sulle due lenti vengono proiettate due immagini distinte relative ad una singola rappresentazione grafica, leggermente diverse fra loro per simulare l'effettiva distanza che esiste fra i due occhi. Le lenti concentrano la luce in modo tale da far apparire i fotogrammi infinitamente lontani, con lo scopo di far credere all'utente di trovarsi in uno spazio ampio e profondo; diversi visori utilizzano lenti di Fresnel, capaci di generare lo stesso effetto possibile con grandi e pesanti lenti sferiche utilizzando meno materiale e combinando sottili sezioni circolari. Queste lenti, inoltre, ingrandiscono il display del visore in modo che non si

notino i bordi dello schermo e facendo sì che l'immagine riempia l'intero campo visivo dell'utilizzatore. Avendo a disposizione immagini a fuoco e leggermente sfalsate fra loro il cervello è in grado di fonderle, attivando dunque la visione binoculare e la percezione della tridimensionalità.
Parlando di visori VR bisogna segnalare l'esistenza di due grosse categorie: i visori tethered e i visori mobili.

4.3.1.1 Visori tethered

Sono i più performanti, hanno uno schermo integrato e normalmente per l'utilizzo si connettono al pc o ad altri sistemi via cavo; inoltre vengono utilizzati spesso insieme ad altri accessori, come controller o telecamere, che permettono di potenziare ulteriormente l'esperienza virtuale. Possiedono una serie di sensori per l'head tracking, ossia il tracciamento delle rotazioni e delle traslazioni della testa, mentre si intravedono già per il prossimo futuro i primi visori con eye tracking, il sistema di tracciamento degli occhi. Viste le loro caratteristiche hanno costi relativamente elevati. Alcuni fra i migliori visori tethered sono:

- Htc Vive: monta uno schermo OLED da 2160x1200 pixel con refresh rate a 90Hz, e utilizza un sistema chiamato "Lighthouse positional tracking system" per il tracciamento della posizione di uno o più utenti. Il kit si completa con un set di controller. Il suo costo è di circa 500 euro
- Playstation VR: integra un sistema di 9 led che, sincronizzandosi con la Playstation Camera in dotazione, offrono un tracciamento a 360°. È dotato di un display OLED da 5,7 pollici con risoluzione Full Hd (1920x1080), con refresh rate pari a 120 Hz. Il visore si completa con un accelerometro e un giroscopio triassiali e un sistema audio 3D, che aumenta il senso di immersione nella realtà virtuale. Il suo costo è di circa 400 euro
- Oculus Rift: è dotato di un display OLED da 2160x1200 pixel con refresh rate pari a 90 Hz, e utilizza un sistema di sensori laser a infrarossi per tracciare il movimento della testa, chiamato "Constellation". L'aspetto forse più interessante del visore è il tempo di lag dell'head tracking, che è stato ridotto fino a 30 ms (con tempo di lag si intende quello che trascorre da quando un movimento reale compiuto con la testa viene rappresentato sul display del visore). Il kit si completa

con un set di due controller, chiamato “Oculus Touch”. Il suo costo è di circa 450 euro

4.3.1.2 Visori mobili

All’interno di questa tipologia ricadono tutti quei visori che non si collegano ad un pc o ad una console di gioco, ma che utilizzano come sorgente dati un più comodo smartphone, che deve essere posizionato proprio all’interno del visore stesso, davanti alle lenti.

Figura 4.3: Un visore mobile: Samsung Gear VR

Conseguenza più importante di tale accorpamento si traduce nella mancanza di un display interno al visore, dato che la sua funzione è assolta direttamente dallo schermo dello smartphone. Questo può essere da una parte un grande vantaggio, dal momento che non sono più necessari cavi per la trasmissione dei segnali video, e ciò rende i visori davvero mobili, ma dall’altra parte bisogna sottolineare come l’esperienza virtuale perda tanto in qualità. I motivi sono molteplici:

- La frequenza di aggiornamento del display degli smartphone è solitamente di 60 Hz, contro i 90-120 Hz dei visori tethered
- La risoluzione standard degli smartphone odierni è di 1280x720, molto più bassa dunque rispetto a quella dei visori tethered
- I sistemi di head tracking nei visori mobili, quando presenti, sono generalmente meno precisi e meno veloci
- I visori mobili non montano sistemi audio avanzati quanto quelli presenti nei visori tethered

- Fondamentalmente non esistono accessori per potenziare il coinvolgimento nell'esperienza virtuale, si possono eventualmente utilizzare solo controller Bluetooth o gli auricolari dello smartphone

Questa lunga lista di differenze si ripercuote ovviamente anche sui prezzi: i visori mobili costano sostanzialmente dai 20 ai 130 euro, molto meno rispetto a quelli tethered.
A questo punto è facile comprendere i motivi del perché si sia deciso di sviluppare per il progetto 3D4Amb un'applicazione per visori mobili: essi soddisfano due requisiti fondamentali del progetto, hanno costo contenuto (contrariamente a quelli tethered) e sono adatti all'uso domestico; inoltre, grazie al fatto che non necessitano di pc o console per funzionare, essi non richiedono una postazione di lavoro dedicata, ma possono essere utilizzati comodamente sul divano, a letto o in qualunque altro luogo.

4.3.2 Smartphone

Gli smartphone, telefoni cellulari con capacità di calcolo, di memoria, di connessione dati molto più avanzate rispetto ai normali telefoni cellulari, negli ultimi anni si sono diffusi in un modo quasi incontrollato: basti pensare che ad inizio 2016 nel mondo ne erano presenti ben 3,4 miliardi [14], e i numeri sono in continua crescita. Negli ultimi anni sono stati resi estremamente performanti, grazie all'integrazione di processori sempre più evoluti e di memorie sempre più capienti; ciò li ha resi disponibili all'utilizzo di applicazioni di alto livello, sia computazionale che grafico. Come è già stato indicato il gioco 3D4Amb Team Racing è stato sviluppato per smartphone Android. La scelta è ricaduta su questo sistema operativo considerando la sua diffusione: nel primo quadrimestre del 2017 l'85% degli smartphone del mondo montava Android, seguito a molta distanza da iOS, presente solamente sul 14,7% dei dispositivi [15]. Oltre alla diffusione dei sistemi operativi è stato tenuto in considerazione anche il prezzo degli smartphone attuali: quelli che montano iOS, ovvero gli iPhone, hanno un prezzo minimo di circa 400 euro (il modello in questione è iPhone SE), mentre quelli che montano Android hanno un prezzo base di circa 50-60 euro (basti pensare a marchi poco blasonati come Wiko, Brondi, Mediacom, Neffos e altri); già con 100-150 euro è possibile assicurarsi un dispositivo sufficientemente performante, in grado di far funzionare in modo fluido il gioco 3D4Amb Team Racing. La scelta dunque ha tenuto in forte considerazione il requisito dell'economicità, fondamentale nel progetto 3D4Amb.

4.3.3 Telecomando Bluetooth

L'ultimo dispositivo richiesto per utilizzare l'applicazione oggetto della tesi è un telecomando o un joystick Bluetooth, necessario dal momento in cui si inserisce lo smartphone nel visore VR. In realtà potrebbero essere utilizzati altri tipi di periferiche, come tastiere wireless/Bluetooth o joystick collegabili via Usb, ma per problemi di praticità o di intralcio (non tutti i visori VR permettono il collegamento di un cavo Usb allo smartphone) se ne sconsiglia l'uso. Generalmente i controller per smartphone montano la versione Bluetooth 3.0 e sono in classe 2, il che significa che funzionano in un raggio d'azione di circa 10 metri; perché possano essere rilevati come joystick è necessario che supportino il profilo HID (Human Interface Device). Il prezzo di questi dispositivi è solitamente basso, una buona parte di essi costa meno di 20 euro. È bene sottolineare che solo i controller con almeno 6 tasti possono essere utilizzati per interagire con l'applicazione 4D4Amb Team Racing.

Figura 4.4: Un modello di telecomando Bluetooth

4.4 Software utilizzato

Durante la creazione del gioco sono stati impiegati diversi software, ognuno dei quali ha permesso di gestire o supportare precisi aspetti della sua realizzazione. L'unica richiesta del relatore della tesi è stata quella di utilizzare il software Unity come strumento di sviluppo principale.

4.4.1 Unity

Unity è un ambiente di sviluppo disponibile per Windows e MacOS, utilizzato per la creazione di videogiochi o di altri contenuti interattivi. La sua prima versione è stata rilasciata nel 2005, e da quel momento è diventato sempre più popolare, soprattutto fra gli sviluppatori di applicazioni mobili. I suoi punti di forza sono:

- la semplicità di utilizzo, che consente la creazione di videogiochi e animazioni 3D anche a chi non è esperto delle relative basi teoriche
- la molteplicità d'impiego, dal momento che Unity permette di creare giochi di qualunque genere; i suoi principali concorrenti, come UDK (Unreal Development Kit) e Cry Engine, spesso sono orientati verso un genere specifico
- la compilazione multipiattaforma: un gioco sviluppato in Unity può essere compilato per essere eseguito su un elevato numero di piattaforme: smartphone, tablet, visori VR, pc, console di gioco e Smart Tv
- il suo costo, che considerando le sue potenzialità è relativamente basso; Unity esiste nella versione Pro (che costa circa 100 euro mensili, ed è per professionisti che richiedono il massimo livello di flessibilità), nella versione Plus (che costa circa 30 euro mensili, ed è rivolta a chi sviluppa piccole applicazioni destinate al mercato) e nella versione Personal (completamente gratuita, che è indirizzata a studenti e appassionati)

La realizzazione di progetti in Unity avviene principalmente attraverso le scene e gli script: le prime possono essere considerate come i contenitori dei vari livelli di gioco, mentre i secondi sono quegli oggetti che contengono le varie logiche dell'applicazione, scritte sotto forma di codice; i linguaggi supportati da Unity sono C# e JavaScript.

Dal punto di vista dell'utilizzo il software si presenta come un IDE (Integrated Development Environment) contenente la gerarchia degli oggetti presenti in scena, un editor grafico, un visualizzatore delle proprietà e un'anteprima dal vivo del gioco; come motori di rendering utilizza Direct3D, OpenGL, OpenGL SE, DirectX e altre API proprietarie. Un ultimo aspetto che vale la pena sottolineare è l'elevato livello di integrazione di Unity con diversi software di modellazione 3D, come 3D Studio Max, Maya e Blender. Per lo sviluppo dell'applicazione è stata utilizzata la versione 2017.2.0f3 del programma, in licenza Personal.

4.4.2 Visual studio

Visual Studio è un ambiente di sviluppo integrato nato nel 2002, ed è stato utilizzato durante la realizzazione dell'applicazione 3D4Amb Team Racing per la scrittura degli script di Unity. È stato sviluppato da Microsoft per la creazione di software per pc, siti, servizi web e applicazioni mobili, e permette l'utilizzo di ben 36 diversi tipi di linguaggio, come C, C++, C#, JavaScript, F#, Visual Basic .Net e Html. In particolare per la scrittura degli script di gioco è stato scelto il linguaggio C# (per un semplice interesse personale, la scelta di Javascript sarebbe stata altrettanto corretta).

Visual Studio include una serie di strumenti, fra cui:

- un editor di codice che facilita il refactoring e supporta IntelliSense, un valido componente che ne permette il completamento automatico e consente la rapida documentazione di variabili, funzioni e metodi
- un debugger interno per il rilevamento e la correzione degli errori logici nel codice in runtime
- diverse finestre designer, per la costruzione della grafica di applicazioni e siti web, ma anche per la generazione di database e classi

Un ultimo aspetto interessante di Visual Studio riguarda la possibilità di installare diversi plugin che permettono l'aggiunta di nuove funzionalità, come il supporto al controllo del codice sorgente o la decompilazione di librerie.

La versione più recente di Visual Studio è distribuita nelle licenze:

- Community, gratuita e rivolta a studenti e sviluppatori singoli e open-source
- Professional, destinata a piccoli team per progetti di medie dimensioni e al costo di circa 450 euro annui
- Enterprise, rivolta a team numerosi e pensata per progetti complessi e di grandi dimensioni, che costa circa 2500 euro all'anno

Per lo sviluppo dell'applicazione 3D4Amb Team Racing è stata utilizzata la versione 2015 in licenza Community.

4.4.3 Gimp

Gimp è un software gratuito per l'elaborazione digitale delle immagini, disponibile per Linux, MacOS e Windows; in particolare è incluso in molte distribuzioni Linux come

editor di immagini standard. La sua prima versione risale al 1998, ed attualmente è sviluppato principalmente da volontari. Può essere utilizzato in molteplici ambiti, a partire da semplici operazioni pittoriche, fino ad arrivare a complesse elaborazioni fotografiche, passando per la grafica web e la conversione di formati; nello sviluppo dell'applicazione 3D4Amb Team Racing è stato utilizzato (nella versione 2.8.22) specificamente per la creazione e la modifica delle texture e dell'icona del gioco.
GIMP permette l'importazione e l'esportazione delle immagini in un gran numero di formati di file differenti, fra cui il formato nativo di Adobe Photoshop (PSD).
Gli strumenti di manipolazione di GIMP possono essere raggiunti attraverso barre degli strumenti, menu o finestre di dialogo, che a loro volta possono essere raggruppate in pannelli. Possiede circa 150 effetti standard e filtri, ed è facilmente estendibile grazie ad un potente sistema di plug-in; offre inoltre delle capacità di scripting tramite diversi linguaggi, come Perl e Python, che permettono di automatizzare ed eseguire interattivamente lunghe sequenze di azioni.
Per le sue caratteristiche avanzate GIMP viene spesso descritto come una valida alternativa a prodotti commerciali come Photoshop.

4.4.4 Maya

Maya è un software di computer grafica utilizzato per l'animazione, la modellazione, la simulazione e il rendering 3D, che per le sue funzionalità trova principalmente impiego nello sviluppo di videogiochi, film e progetti architettonici. Grazie all'alta qualità dei suoi strumenti Maya è uno dei software più apprezzati del settore; le sue principali caratteristiche sono:

- Creazione di simulazioni accurate di fluidi, corpi rigidi e morbidi
- Creazione di effetti atmosferici, come nebbia e foschia
- Creazione di pellicce, capelli, lana ed erba realistici
- Strumenti per il Motion Graphics
- Animazioni 2D e 3D basate su script e fotogrammi chiave
- Strumenti per il Character Animation
- Modellazione mesh poligonale
- Modellazione di superfici
- Strumenti per la gestione delle coordinate UV

- Produzione di immagini grazie a motori di rendering integrati ed esterni
- Complessi effetti di ombreggiatura

Occorre sottolineare poi che Maya offre una grande liberà di personalizzazione dell'interfaccia grafica, ed è facilmente estendibile grazie a diversi plugin, che possono essere scritti in Python, MEL (Maya Embedded Language) e C++.
Nella sua versione più completa Maya ha un costo annuo di circa 2000 euro, ed è disponibile per Windows, MacOS e Linux. Durante lo sviluppo di 3D4Amb Team Racing è stato utilizzato (nella versione 2013) per la modifica poligonale di alcune mesh (ad esempio la macchina controllata dall'utente), per la creazione di alcuni oggetti 3D (come gli edifici presenti nel secondo livello di gioco) e delle relative coordinate UV, e per la creazione dell'icona dell'applicazione.

4.4.5 Uniform Server

Uniform Server è un pacchetto gratuito che include tutto il necessario per utilizzare il Web Server Apache, i linguaggi di scripting PHP e Perl e il motore di database MySQL. La sua caratteristica più interessante è quella di essere totalmente portable, ovvero di non richiedere l'installazione, il che l'ha reso estremamente utile per il testing del software 3D4Amb Team Racing. Lo sviluppo del servizio web e del database che l'applicazione utilizzerà per il salvataggio e il caricamento dati infatti non è ancora stato completato, dunque durante il testing è nata la necessità di appoggiarsi ad un applicativo temporaneo. Grazie a Uniform Server e ad alcune pagine PHP create ad hoc è stato possibile verificare il funzionamento delle routine di salvataggio e caricamento dati su server.
È stato utilizzato nella versione 13.3.2, che include le release 2.4.25 di Apache, 5.6.35 di MySQL, 10.0.29 di Mariadb e 4.6.6 di phpMyAdmin. Ultime caratteristiche del software che vale la pena menzionare sono l'alto livello di sicurezza, impostato di default su tutte le sotto-applicazioni utilizzate, la facilità nella modifica e nella gestione della configurazione del server, e la valida funzionalità di log.

4.4.6 Bluestacks

Bluestack è un emulatore gratuito di app Android per MacOS e Windows. Le sue caratteristiche principali sono l'ambiente completamente personalizzabile, il supporto per

molteplici configurazioni di sistema operativo e l'integrazione con Google Play (il portale di Android che permette il download delle applicazioni). La funzionalità che l'ha reso indispensabile nello sviluppo dell'applicazione sta nella possibilità di impostare liberamente la risoluzione del dispositivo virtuale; ciò ha permesso di testare la corretta visualizzazione del gioco su schermi con diverse dimensioni e differenti aspect ratio. Durante il testing dell'applicazione è stata utilizzata la versione 3.54.65.1755 del software.

4.5 Modello di sviluppo

Ora che è chiaro il contesto di utilizzo dell'applicazione e che sono stati presentati gli hardware e i software coinvolti, è possibile cominciare a descrivere come operativamente il gioco è stato realizzato. La prima cosa che occorre illustrare è il modello di sviluppo utilizzato: fondamentalmente l'idea è stata quella di seguire il classico modello a cascata, e di fare affidamento su diversi prototipi per definire la progettazione di alcuni componenti.

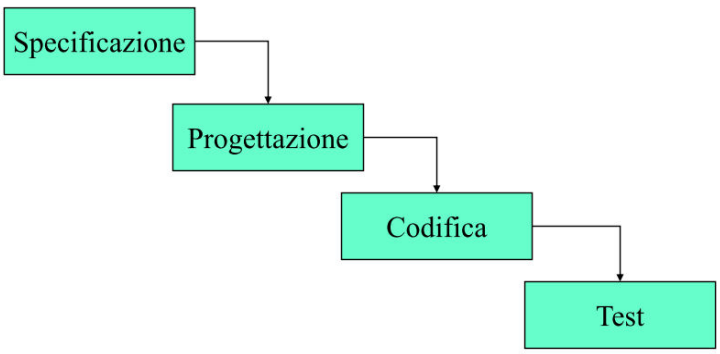

Figura 4.5: Modello di sviluppo a cascata

La scelta di questo tipo di modello è stata abbastanza obbligata da precisi fattori:

- già all'inizio del progetto si sapeva che, a causa di motivi personali, il tempo a disposizione per lo sviluppo dell'applicazione sarebbe stato poco, e sarebbe stato ancora meno il tempo a disposizione per gli incontri con il professore relatore; dal momento che le interazioni fra le persone e la collaborazione col cliente sono due aspetti fondamentali dei modelli di sviluppo agili, ci si è resi subito conto che

affrontare il progetto con questo approccio e in queste condizioni sarebbe stato impossibile

- sempre a causa del poco tempo a disposizione e dell'impossibilità di incontri frequenti con il professore relatore si è dovuto scartare anche un modello di sviluppo iterativo
- essendo il gioco un progetto di tesi, non sono mai stati previsti rilasci successivi del software, dunque anche un modello di sviluppo incrementale si è rivelato fin da subito inadatto

È dunque chiaro il perché all'avvio del progetto il modello a cascata sia sembrato l'unica scelta realmente a disposizione, è l'unico modello che permette tempi veloci di sviluppo e scarsa interazione fra le persone coinvolte. Sono però anche ben noti i suoi difetti classici: necessita di requisiti chiari e definiti, e non è in grado dunque di adattarsi a specifiche incerte o a possibili evoluzioni del progetto. In effetti questi suoi problemi si sono manifestati anche durante la realizzazione di 3D4Amb Team Racing: al termine dello sviluppo il gioco è stato presentato al professore relatore, e nell'occasione è stata richiesta la modifica di alcune specifiche. A questo punto si è passati dunque ad un diverso modello di sviluppo, più simile a quello iterativo: si è cominciato a trattare l'applicazione fino a quel momento realizzata come un prototipo, ed ogni richiesta di modifica è coincisa con una nuova fase di progettazione, sviluppo e testing. Al termine si è dunque ottenuta un'applicazione progettualmente valida, che ha soddisfatto i requisiti richiesti.

4.6 Requisiti e specifiche

All'avvio del progetto, per qualche settimana, c'è stata un'intensa comunicazione con il professore relatore, sia attraverso incontri che mediante scambio di e-mail, per la definizione dei requisiti dell'applicazione. Dal momento che si era già deciso di svilupparla seguendo un modello a cascata si è cercato di analizzare ogni singolo aspetto del gioco e di chiarire qualunque dubbio, in modo da evitare sorprese spiacevoli durante le fasi di sviluppo successive. Va detto che i requisiti fondamentali richiesti sono stati davvero pochi, mi è stata concessa una grandissima libertà d'azione.

Come già indicato nel capitolo precedente, nuovi requisiti sono stati richiesti in una fase già molto avanzata del processo di sviluppo.

4.6.1 Requisiti funzionali

Vengono di seguito descritti i requisiti funzionali dell'applicazione sotto forma di elenco strutturato.

RF01	Avvio del gioco
L'utente deve aver la possibilità di avviare il gioco sia in modalità online, passando per una fase di login, sia in modalità offline.	

RF02	Funzione di login
La funzione di login deve avvenire tramite l'inserimento di e-mail e password; in caso di login fallito l'applicazione deve visualizzarne i motivi. Deve essere inoltre presente un'opzione che permetta all'utente di salvare su dispositivo i dati di login; ad opzione abilitata i dati devono essere automaticamente caricati ad ogni successivo avvio dell'applicazione.	

RF03	Registrazione account e recupero password
L'utente deve poter accedere alle pagine web per la registrazione di un nuovo account e per il recupero della password di un account esistente.	

RF04	Configurazione occhio pigro
L'utente deve poter impostare l'occhio pigro, l'informazione deve essere salvata su dispositivo e caricata ad ogni avvio dell'applicazione.	

RF05	Configurazione e applicazione penalizzazione (trasparenza oggetti)
Il primo metodo di penalizzazione consiste nel rendere più trasparenti all'occhio sano le casse di legno e le trappole presenti lungo i percorsi di gioco. Deve essere possibile impostare due valori percentuali compresi fra 0 e 100 (numeri interi), e durante l'esecuzione delle gare la trasparenza degli oggetti deve passare dal valore minimo a quello massimo, in modo inversamente proporzionale al tempo ancora a disposizione	

per il completamento del livello. I valori di trasparenza impostati devono essere salvati su dispositivo e caricati ad ogni avvio dell'applicazione.

RF06	Configurazione e applicazione penalizzazione (oscuramento scena)
Il secondo metodo di penalizzazione consiste nel rendere più scuro l'intero fotogramma di gioco che viene mostrato all'occhio sano. Deve essere possibile impostare due valori percentuali compresi fra 0 e 100 (numeri interi), e durante l'esecuzione delle gare l'oscuramento dei fotogrammi deve passare dal valore minimo a quello massimo, in modo inversamente proporzionale al tempo ancora a disposizione per il completamento del livello. I valori di oscuramento impostati devono essere salvati su dispositivo e caricati ad ogni avvio dell'applicazione.	

RF07	Configurazione telecomando Bluetooth
Deve essere possibile associare tutti i comandi di gioco ai tasti del telecomando Bluetooth. La configurazione impostata deve essere salvata sul dispositivo, e caricata ad ogni avvio dell'applicazione.	

RF08	Funzionamento mappa base
Deve essere presente una mappa base liberamente esplorabile, contenente una serie di piattaforme circolari attraverso le quali l'utente può avviare le varie gare.	

RF09	Abilitazione livelli di gioco
Al primo avvio del gioco solo la piattaforma associata al primo livello deve essere sbloccata, e solo completando quest'ultimo si sblocca la seconda piattaforma; la stessa logica vale per le gare successive. Il livello di gioco raggiunto deve essere salvato su dispositivo in caso di partita offline e su server in caso di partita online, e deve essere caricato (secondo lo stesso criterio) ad ogni avvio dell'applicazione.	

RF10	Esecuzione gara
Per completare il livello con successo l'utente deve riuscire a completare 3 giri completi del percorso entro un tempo limite. Se lungo il tragitto la macchina colpisce le casse di legno verdi il tempo a disposizione deve aumentare di qualche secondo (bonus), se	

invece colpisce le casse di legno rosse il tempo a disposizione deve diminuire di qualche secondo (malus). I tempi bonus e malus devono essere gli stessi in ogni livello di gioco.

RF11	Riavvio gara
L'utente deve poter riavviare il livello di gioco in corso.	

RF12	Visualizzazione informazioni durante la gara
Durante l'esecuzione di una gara l'utente deve poter visualizzare il tempo ancora a disposizione e il numero del giro in corso.	

RF13	Salvataggio dati di gara
In caso di partita online al termine di una gara devono essere salvate su server tutte le informazioni utili relative alla gara stessa.	

RF14	Limiti percorso di gara
Sia nella mappa base che nelle mappe di gara devono essere presenti dei limiti fisici che limitino l'area percorribile dalla macchina.	

4.6.2 Requisiti non funzionali

Vengono di seguito descritti i requisiti non funzionali dell'applicazione sotto forma di elenco strutturato; il prefisso del codice indica la tipologia dei vari requisiti (RUS-usabilità, REF-efficienza, RFU-funzionalità, RPO-portabilità, RAF-affidabilità, RMA-manutenibilità).

RUS01	Flusso lineare di gioco
Il flusso di controllo dell'applicazione deve essere semplice e lineare, in modo che qualunque utente (indicativamente dai 6-7 anni) sia in grado di giocare autonomamente.	

RUS02	Longevità del gioco

L’applicazione deve stimolare e far divertire l’utente, ovvero il bambino che la utilizza, in modo che non si stanchi di giocarci. Più il trattamento è svolto volentieri, più è proficuo.

RUS03	Lingue del gioco
I testi del gioco devono essere in italiano se la lingua impostata sul dispositivo è l’italiano, in inglese in tutti gli altri casi.	

RUS04	Interazione con l’applicazione
Nelle fasi di avvio e di configurazione del gioco l’interazione deve avvenire solo mediante la funzione touch dello smartphone, in tutte le altre fasi l’interazione deve avvenire unicamente attraverso il telecomando Bluetooth.	

REF01	Prestazioni dell’applicazione
Il gioco deve poter essere eseguito su smartphone di bassa-media fascia; si richiede l’aggiornamento della scena ad una frequenza di almeno 25 fps.	

RFU01	Sicurezza trasmissione password account su server
Si richiede che la password di accesso all’account venga trasmessa al server solo in seguito ad un’operazione di hashing.	

RPO01	Compatibilità versioni Android
L’applicazione deve essere compatibile con tutte le versioni di Android a partire dalla 4.4 (KitKat); questo perché attualmente solo circa il 6% dei dispositivi Android monta una versione precedente [16].	

RPO02	Compatibilità risoluzioni smartphone
L’applicazione deve essere in grado di adattarsi a schermi di diverse dimensioni, anche con differenti aspect ratio.	

RPO03	Compatibilità con telecomandi Bluetooth

L'applicazione deve essere compatibile con tutti i telecomandi Bluetooth che supportino il profilo HID, e su cui siano montati almeno 6 tasti.

RAF01	Livello di maturità
Non si esclude la possibilità che il gioco in futuro possa essere disponibile su Play Store, e che dunque possa essere utilizzato da un elevato numero di utenti; per questo motivo è richiesto fin da subito un buon livello di maturità.	

RMA01	Livello di analizzabilità e modificabilità
È possibile che in futuro l'applicazione possa essere ripresa da qualche altro tesista, sia per la correzione di eventuali bug che per lo sviluppo di nuove funzionalità; per questo motivo il codice deve essere chiaro e ben documentato.	

Leggendo la lista di requisiti è facile porsi alcune domande come: perché non si è fatto riferimento alla figura del medico? Quale è l'utilità del salvare i dati delle gare? Quali sono i vantaggi nell'effettuare il login? Sono domande chiaramente lecite, che trovano un'unica risposta: il sistema non è completo. Il lavoro della tesi è stato unicamente lo sviluppo del gioco 3D4Amb Team Racing, ovvero la parte del sistema utilizzata dal paziente; il passo successivo deve essere quello di sviluppare una nuova applicazione, in questo caso utilizzata da un medico, che permetta di impostare per i propri pazienti i corretti parametri della terapia, di visionare il lavoro svolto e i progressi fatti. Solo in questo modo si avrà a disposizione uno strumento completo e realmente utilizzabile.

4.6.3 Casi d'uso

Dopo aver illustrato i requisiti dell'applicazione è bene visualizzare i principali casi d'uso; anche in questo caso viene tenuto in considerazione il solo lato client dell'applicazione.

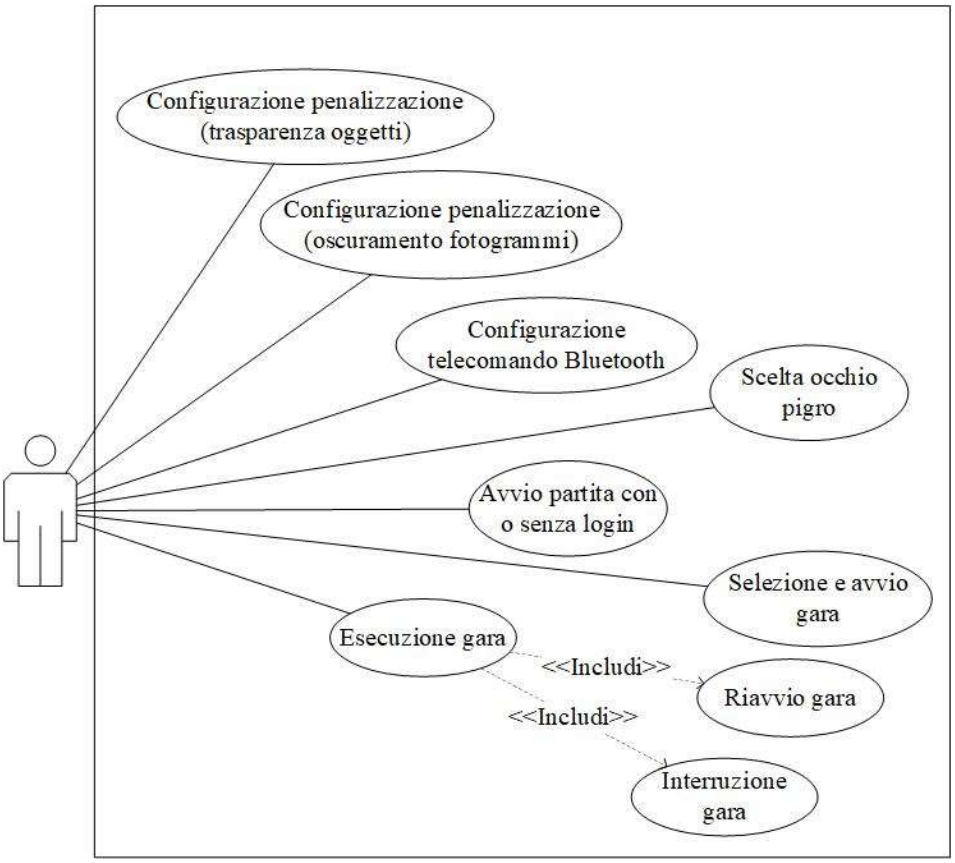

Figura 4.6: Diagramma UML dei principali casi d'uso

CU01	Configurazione penalizzazione (trasparenza oggetti)
1- Dalla schermata iniziale l'utente accede alle opzioni 2- Seleziona il tasto per accedere alla configurazione della trasparenza degli oggetti 3- A schermo vengono visualizzati i valori di trasparenza minimo e massimo correnti 4- Mediante gli appositi tasti l'utente definisce i nuovi valori di trasparenza 5- Cliccando sull'apposito tasto si chiude il pannello di configurazione	

CU02	Configurazione penalizzazione (oscuramento fotogrammi)
1- Dalla schermata iniziale l'utente accede alle opzioni 2- Seleziona il tasto per accedere alla configurazione di oscuramento dei fotogrammi 3- A schermo vengono visualizzati i valori di oscuramento minimo e massimo correnti 4- Mediante gli appositi tasti l'utente definisce i nuovi valori di oscuramento 5- Cliccando sull'apposito tasto si chiude il pannello di configurazione	

CU03	Configurazione telecomando Bluetooth
1- Dalla schermata iniziale l'utente accede alle opzioni 2- Clicca il tasto per accedere alla configurazione dei comandi di gioco 3- Seleziona il tasto relativo al comando che intende configurare 4- A schermo vengono visualizzate le informazioni relative al tasto del telecomando attualmente associato al comando 5- L'utente clicca sul telecomando il tasto che vuole associare al comando di gioco 6- La nuova configurazione del comando viene salvata su dispositivo 7- Automaticamente si chiude il pannello di configurazione del comando Estensioni: 5a- L'utente clicca sul tasto a schermo per annullare l'operazione; si ritorna al passo 7 dello scenario principale	

CU04	Scelta occhio pigro
1- Dalla schermata iniziale l'utente accede alle opzioni 2- Seleziona il tasto per accedere alla schermata di scelta dell'occhio pigro 3- Viene evidenziato il tasto relativo all'occhio attualmente impostato come pigro 4- L'utente seleziona il tasto relativo all'occhio che si vuole impostare come pigro 5- Viene salvato su dispositivo il nuovo valore 6- L'utente clicca sul tasto per tornare alla schermata precedente 7- Viene visualizzata la schermata iniziale delle opzioni Estensioni: 4a- L'utente clicca sul tasto per tornare alla schermata precedente; si ritorna al passo 7 dello scenario principale	

CU05	Avvio partita con o senza login
1- All'avvio del gioco compare la schermata iniziale 2- L'utente inserisce e-mail e password per il login 3- Clicca sul tasto per avviare la partita con login 4- Il sistema riconosce le credenziali di accesso 5- Viene avviata la partita Estensioni:	

2a- I dati di login vengono caricati automaticamente dal dispositivo, e mostrati a schermo; si ritorna al passo 3 dello scenario principale 2b- L'utente clicca sul tasto per avviare la partita senza login; si ritorna al passo 5 dello scenario principale 4b- Il sistema non riconosce le credenziali d'accesso, viene visualizzato un messaggio d'errore; si ritorna al passo 2 dello scenario principale

CU06	Selezione e avvio gara
1- L'utente manovra la macchina all'interno della mappa base 2- Individua la piattaforma circolare associata alla gara che desidera avviare 3- Con la macchina si posiziona sulla piattaforma circolare scelta 4- Dopo una piccola pausa la gara inizia	

CU07	Esecuzione gara
1- Si apre la schermata di gara 2- Dopo un countdown iniziale l'utente assume il controllo della macchina, e percorre 3 giri del tracciato prima del termine del tempo a disposizione 4- La partita è online, i risultati della gara vengono salvati su server 5- Compare un messaggio che indica che il livello è stato completato 6- Viene chiusa la schermata di gara, e si apre la scena contenente la mappa base Estensioni: 2a- Durante il countdown o l'esecuzione della gara l'utente decide di riavviare la gara; si ritorna al passo 2 dello scenario principale 2a- Durante il countdown o l'esecuzione della gara l'utente decide di interrompere la gara; si ritorna al passo 6 dello scenario principale 4a- La partita è offline, i risultati della gara non vengono salvati su server; si ritorna al passo 5 dello scenario principale	

CU08	Riavvio gara
1- L'utente preme sul telecomando Bluetooth il tasto di pausa 2- Utilizza i tasti direzione per raggiungere il tasto di riavvio gara 3- Preme il tasto di conferma	

CU09	Interruzione gara
1- L'utente preme sul telecomando Bluetooth il tasto di pausa 2- Utilizza i tasti direzione per raggiungere il tasto di interruzione gara 3- Preme il tasto di conferma	

4.7 Regole di progettazione

Durante la progettazione dell'applicazione si è cercato di rispettare i principi studiati durante il corso di Ingegneria del software:

- coesione: in ogni script si è cercato di inserire dati e operazioni propri di una specifica area funzionale, in modo da garantire il riuso e rendere più semplice l'architettura del sistema; nei casi in cui non si è seguito questo criterio i motivi sono da ricondurre soprattutto all'ambiente di sviluppo Unity: la sua logica a scene ha infatti posto alcuni vincoli alla libertà di progettazione del software
- accoppiamento: per ridurre la complessità del sistema si è cercato di minimizzare l'accoppiamento fra i vari script, in particolare per quanto riguarda lo scambio dei dati; bisogna ammettere però che come grado di accoppiamento non è stata fatta la scelta tecnicamente migliore. Si è deciso infatti di adottare il grado Content, il che significa che i vari script hanno la possibilità di modificare i dati interni ad altri script; anche in questo caso la scelta è stata in parte dettata dalla logica a scene di Unity. L'alternativa migliore sarebbe stata probabilmente utilizzare il grado di accoppiamento Stamp, inserendo cioè sia tipi di dati che dati stessi all'interno di script comuni; l'opzione è stata scartata unicamente per favorire la coesione modulare
- information hiding: un punto su cui si è prestata la massima attenzione è quello relativo alla visibilità dei dati; ogni script infatti espone all'esterno unicamente le informazioni e le funzioni necessarie a garantire la corretta interconnessione con altri script
- riuso: alcuni script sono stati scritti in modo tale da poter essere riutilizzati in caso di futuri sviluppi del progetto; ad esempio lo script che gestisce l'esecuzione della

gara (che verrà descritto nel dettaglio successivamente) può essere riutilizzato, senza modifiche, per gestire eventuali nuovi livelli di gioco

- utilizzo componenti COTS (Commercial Off-The-Shell): utilizzare risorse e strumenti già presenti sul mercato può essere una valida scelta in molti tipi di progetti; nel caso di 3D4Amb Team Racing si è deciso di utilizzare diverse risorse multimediali (come modelli 3D e textures) e strumenti (come quello utilizzato per definire i limiti dei tracciati) già esistenti e pronti all'uso; questo ha permesso di potersi concentrare maggiormente sulle parti più critiche dell'applicazione. La fonte principale da cui è stata scaricata buona parte delle risorse esterne è l'Asset Store, il negozio online di Unity

L'architettura del sistema viene descritta dall'immagine seguente; ricordo che la parte server non è oggetto di questa tesi, dunque è possibile che nel diagramma non sia rappresentata in modo totalmente corretto:

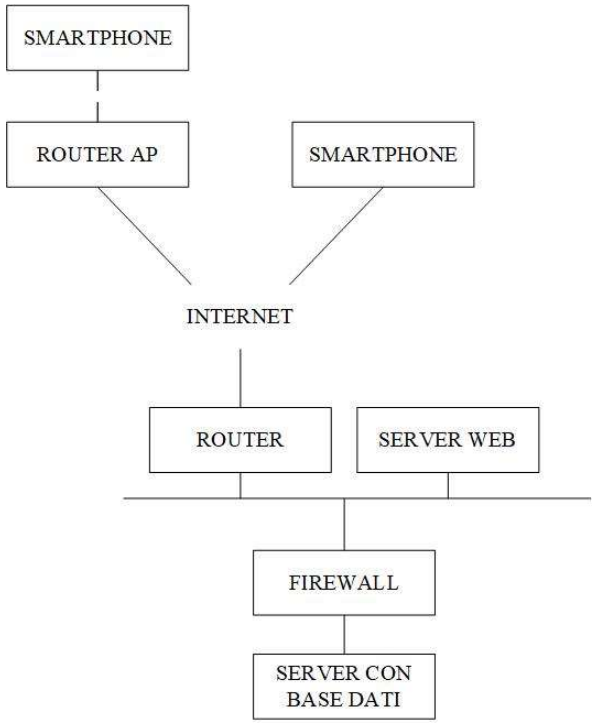

Figura 4.7: Architettura del sistema

4.8 Implementazione

4.8.1 Struttura del progetto

Come è già stato anticipato, la logica di Unity prevede l'utilizzo di scene. Ognuna di esse sostanzialmente rappresenta un ambiente di gioco, dal momento che al suo interno vengono inseriti gli oggetti 3D che l'applicazione visualizza a schermo quando la scena stessa è attiva. Oltre ai modelli 3D questa può contenere luci, effetti particellari, camere, e molti altri tipi di elementi, tutti estremamente personalizzabili; in Unity questi oggetti vengono chiamati GameObjects.

Il gioco è attualmente composto da cinque scene:

- Home: la scena lanciata all'avvio del software, contiene le schermate di login, di personalizzazione delle penalizzazioni, di selezione dell'occhio pigro e di configurazione del telecomando Bluetooth
- MappaBase: è la scena che viene avviata all'inizio della partita, contiene le piattaforme circolari che consentono di far partire le varie gare
- MappaGara0, MappaGara1 e MappaGara2: sono le scene relative ai tre livelli di gioco

Come è facile intuire, per aggiungere un'ulteriore percorso di gara all'applicazione è sufficiente creare una nuova scena (dal probabile nome di MappaGara3), inserire al suo interno tutti i GameObject necessari per il funzionamento del livello e inserire nella scena MappaBase la piattaforma per il suo avvio; per quanto riguarda il codice le modifiche sono minime, il lavoro maggiore consiste nella realizzazione degli script specifici per il nuovo livello.

Un altro aspetto delle logiche di Unity riguarda l'archiviazione delle risorse dell'applicazione: tutti i modelli 3D, gli script, le texture, le scene, e qualunque altro tipo di risorsa devono essere contenuti all'interno di una cartella del progetto chiamata Assets. La collocazione delle varie risorse è stata attuata in primo luogo utilizzando la tipologia come criterio di base, dunque all'interno della cartella Assets sono state create le sottocartelle:

- Editor: contiene quegli script che vengono eseguiti in fase di editor del gioco, non in fase di runtime

- Font: contiene il font utilizzato per disegnare numeri e stringhe sullo schermo
- Materials: contiene i materiali, ovvero quelle risorse che vengono “spalmate” sui modelli 3D per definire il loro colore, i riflessi e la trasparenza
- Modelli: contiene i modelli 3D utilizzati dall’applicazione, sono generalmente file con estensione FBX o OBJ
- Polydraw: contiene i file necessari per eseguire Polydraw, uno strumento utilizzato per limitare l’area in cui la macchina controllata dall’utente può muoversi. La scelta di mettere questa cartella direttamente in Asset è stata obbligata: infatti Polydraw, per funzionare correttamente, richiede di essere copiato proprio in questo percorso
- Prefab: i prefab in Unity sono sostanzialmente dei GameObject composti, costituiti a loro volta da diversi GameObject e componenti, e vengono messi a disposizione dall’ambiente di sviluppo per evitare di dover ricostruire da zero ogni volta oggetti più o meno complessi. Un esempio di Prefab è la macchina controllata dall’utente, che è formata da mesh, materiali, script e altri componenti utilizzati per gestire la fisica del veicolo
- Scene: la cartella contiene le scene utilizzate dall’applicazione e le relative lightmap (se ne discuterà dettagliatamente in seguito)
- Script: include tutti gli script che vengono eseguiti in runtime, ossia quelli che contengono le logiche dell’applicazione
- Shaders: gli shaders sono fondamentalmente dei programmi eseguiti sulla GPU, che descrivono come poligoni e immagini debbano essere processati prima di essere visualizzati a schermo; nel progetto sono stati ampliamente utilizzati per rendere più veloce e leggera l’applicazione
- Standard assets: è la cartella contenente le risorse standard di Unity
- Terrain: i terrain sono particolari oggetti che Unity utilizza per salvare diverse informazioni relative ai terreni, come le pendenze, la presenza di alberi, ecc. Nel software i terrain sono stati utilizzati per la realizzazione della mappa base e dei primi due livelli di gioco
- Texture: contiene le immagini utilizzate dall’applicazione
- Tools: cartella che contiene tutti quegli strumenti esterni che, al pari di Polydraw, sono stati estremamente utili per la realizzazione del gioco

Il secondo criterio scelto per la collocazione delle risorse riguarda l'appartenenza alle varie scene; in buona parte delle cartelle appena descritte infatti troviamo le ulteriori sottocartelle:

- Scena Home, Scena mappa base, Scena mappa gara 0, Scena mappa gara 1, Scena mappa gara 2: contengono quelle risorse utilizzate unicamente durante l'esecuzione della rispettiva scena
- Comuni: contengono quelle risorse utilizzate in almeno due scene del gioco

4.8.2 Schermate e flusso di gioco

Dopo aver introdotto la struttura del progetto è bene illustrare il flusso di gioco, che è costituito dalle schermate dell'applicazione e dalle modalità di passaggio fra esse. Un modo intuitivo per rappresentarlo è il diagramma di macchina a stati UML:

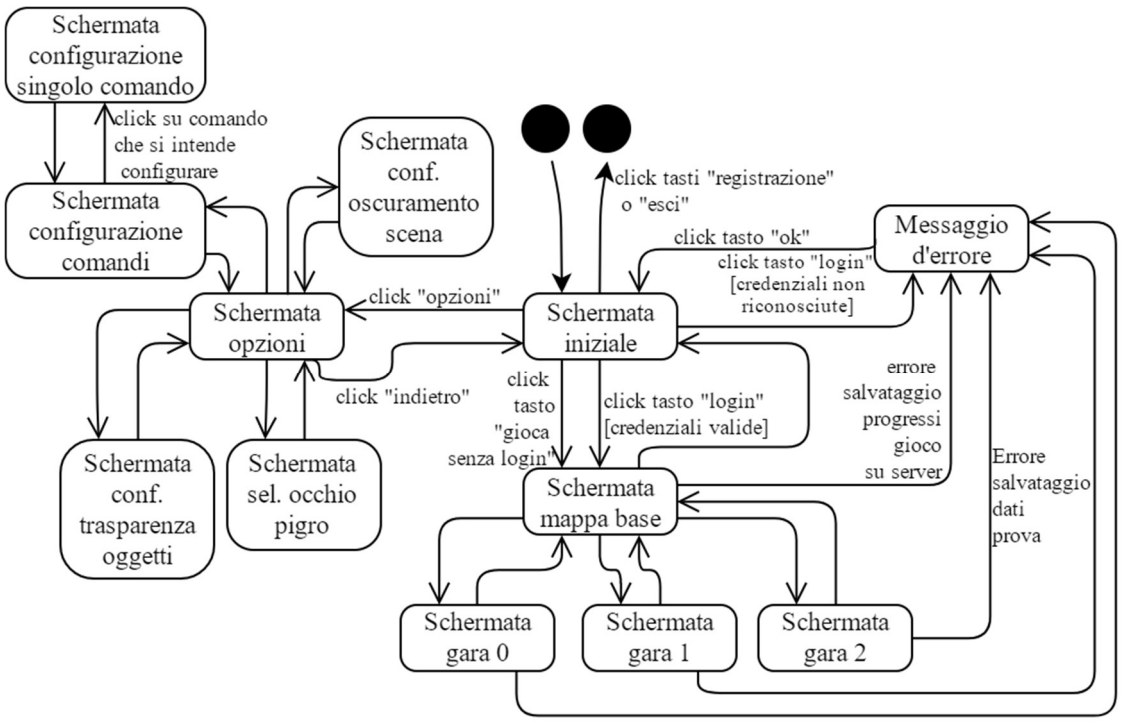

Figura 4.8: Diagramma di macchina a stati raffigurante il flusso delle schermate di gioco

Il flusso può essere anche descritto non utilizzando le schermate ma le scene del gioco; in tal caso risulta essere molto più compatto:

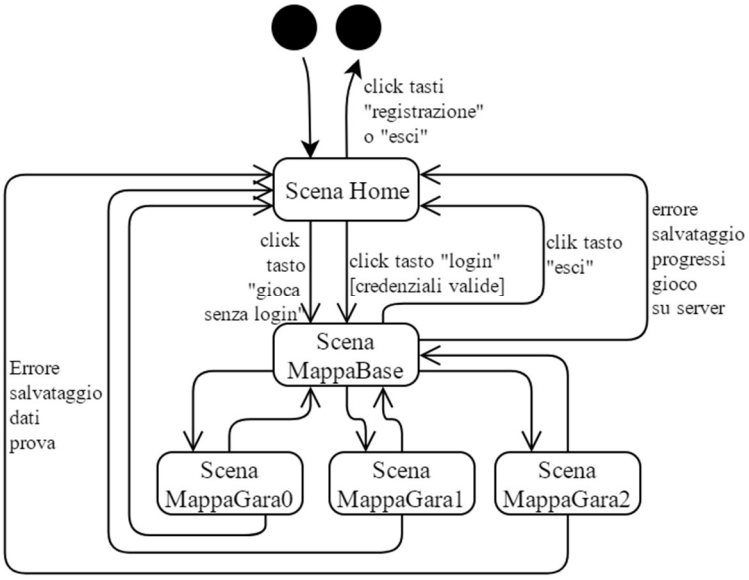

Figura 4.9: Diagramma di macchina a stati raffigurante il flusso delle scene di gioco

4.8.3 Realizzazione scene

Prima di addentrarsi nella spiegazione del codice del gioco, ossia dei suoi script, vale la pena soffermarsi sulle modalità di creazione delle scene, le regole base che sono state seguite e le difficoltà riscontrate.

4.8.3.1 Filosofia

La cosa che sempre si è cercato di tenere a mente durante la realizzazione del software è il target di utilizzo: saranno principalmente i bambini a giocare a 3D4Amb Team Racing. Nello sviluppo di ogni scena si è dunque fatta molta attenzione all'impatto visivo, ai contenuti multimediali, ai colori e all'illuminazione. La scena Home è stata costruita utilizzando colori scuri e scarsa illuminazione, in modo da conferirle un aspetto accattivante. Nelle scene contenenti la mappa base e i percorsi di gara si è voluto invece utilizzare colori vivaci e una forte intensità luminosa, per dar loro quell'aspetto allegro e poco serio che richiama molto il gioco che ha ispirato il progetto, Crash Team Racing.

4.8.3.2 Struttura delle scene

Come si è anticipato in precedenza ogni scena è costituita da diversi GameObject, che definiscono la grafica e il funzionamento della stessa. Home è quella dal punto di vista visivo più semplice: contiene la macchina del gioco e il relativo pavimento su cui è

appoggiata; la macchina è in continua e lenta rotazione, e passando fra le varie schermate la camera si sposta in diversi punti nello spazio.

Figura 4.10: La scena Home

I GameObject presenti sono:

- Camera: l'oggetto che permette di visualizzare a schermo il contenuto della scena, e si comporta come una vera e propria videocamera; fra i suoi componenti si può individuare lo script per il suo controllo
- Macchina: è il modello 3D relativo alla macchina controllata dall'utente
- Luce: permette l'illuminazione della scena
- Terreno: è il piano su cui poggia la macchina
- GestoreInput: l'oggetto permette di interfacciarsi con lo script di gestione dell'input, è necessario per il corretto funzionamento della configurazione del telecomando Bluetooth
- Menu: gestisce via script i vari menu attivabili nella scena (configurazione penalizzazione, comandi di gioco, occhio pigro, ecc)
- Scena: contiene gli script per la gestione della scena stessa
- SaveLoadServer: permette di interfacciarsi con lo script per il salvataggio/caricamento dati da server, è indispensabile durante la fase di login
- SaveLoadLocale: permette di interfacciarsi con lo script per il salvataggio/caricamento dati da device, è necessario per caricare diversi dati di gioco, fra cui la configurazione dei tasti e i livelli di penalizzazione impostati

MappaBase è molto ispirata alla vera mappa base di Crash Team Racing. Consiste in un'isola sabbiosa in cui sono presenti palme, una struttura rocciosa, un forziere e un molo, oltre alle tre piattaforme circolari che permettono l'avvio delle gare. Accanto ad ognuna di esse sono stati posizionati degli oggetti apparentemente fuori luogo, ma che in realtà danno la possibilità all'utente di capire a quale gara è associata la piattaforma stessa.

Figura 4.11: La scena MappaBase

Anche in questa scena sono presenti i GameObject Luce, GestoreInput, Scena, SaveLoadServer, SaveLoadLocale, che hanno funzioni similari a quelle descritte in precedenza. Altri GameObject, come Pallone, AutoPolizia, Forziere o Nave, sono semplici modelli 3D di decoro, per cui non sono necessarie descrizioni. Altri GameObject ancora, invece, meritano di essere approfonditi:

- CameraSx, CameraDx: rispetto alla scena precedente, in cui l'immagine catturata dalla camera andava ad occupare l'intero schermo dello smartphone, in questo caso si ha che la scena viene disegnata due volte sullo schermo, una sul lato destro e una su quello sinistro, perché è progettata per essere visualizzata tramite visore 3D. Questo significa che sono necessarie due camere
- Macchina: anche questa è molto diversa rispetto alla scena precedente; non è infatti un semplice modello 3D, ma è un Prefab costituito da una serie di

componenti (mesh, collider, rigid body e script) che danno la possibilità di pilotare l'auto stessa all'interno della mappa

- Gui: ha una funzione simile al GameObject Menu descritto nella scena precedente: permette infatti di gestire la visualizzazione del menu di pausa
- LimitiMappa: il GameObject ha lo scopo di limitare il movimento dell'auto all'interno della mappa, non consente cioè di avventurarsi in mare
- Skybox: lo Skybox ha lo scopo di simulare graficamente l'effetto del cielo; senza di esso infatti si vedrebbe oltre il mare un infinito muro nero
- Terreno: rispetto alla scena precedente la macchina non è appoggiata su un semplice piano, ma su un oggetto Terrain; questo ha permesso di modificare pendenze e dislivelli del terreno, e dunque di creare il piacevole effetto dell'acqua che lambisce la sabbia

MappaGara0, MappaGara1 e MappaGara2 sono le scene associate ai tre livelli di gioco. La prima consiste in un tracciato terroso che passa all'interno di un bosco di montagna, attraversato a sua volta da un corso d'acqua. Lungo il tragitto l'utente si trova ad affrontare due ostacoli inaspettati: il primo è un masso che rotola giù da una montagna, il secondo è un albero che ad un certo punto cade, invadendo la strada.

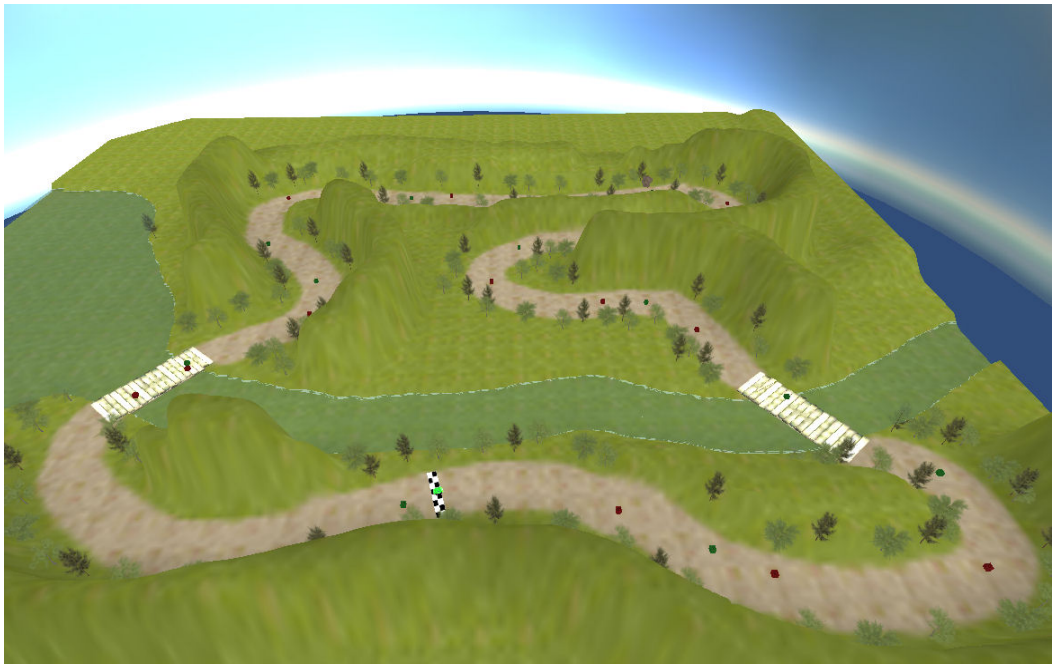

Figura 4.12: La scena MappaGara0

MappaGara1 contiene invece un ambiente cittadino: l'utente deve pilotare la macchina su una carreggiata asfaltata che passa attraverso case, palazzi e macchine parcheggiate, ma anche in questo caso è presente una piccola area verde, costituita da un parco e da un laghetto. In questo livello l'utente deve stare attento a due veicoli in movimento, che sfrecciano lungo il percorso in direzione opposta alla propria, e a un furgone fermo sulla carreggiata.

Figura 4.13: La scena MappaGara1

L'ultima scena, MappaGara2, è ambientata in uno stadio di calcio; il percorso si sviluppa sul campo di gioco, fra birilli e palloni. In quest'ultimo caso l'utente deve stare attento sia ai palloni vaganti che agli attrezzi da allenamento disseminati lungo il percorso.

Figura 4.14: La scena MappaGara2

In tutte e tre le mappe sono attive le configurazioni di penalizzazione, che agiscono sul fotogramma mostrato all'occhio sano:

- L'immagine subisce un generale oscuramento
- Le casse di legno, gli ostacoli e le trappole presenti lungo i tre tracciati vengono resi più trasparenti

È bene precisare che durante l'esecuzione delle gare a schermo vengono mostrati sia il tempo ancora disponibile per il completamento del percorso, sia il numero del giro in corso, come richiesto dai requisiti funzionali dell'applicazione.

Così come MappaBase anche queste scene contengono i GameObject CameraDx, CameraSx, Luce, Macchina, GestoreInput, SaveLoadServer, LimitiMappa, Skybox e Scena, e tutti mantengono le stesse funzioni. Altri GameObject che meritano di essere descritti sono:

- EsplosioneVerde: permette di generare un effetto particellare somigliante ad un'esplosione verde nel momento in cui qualunque cassa di legno verde viene colpita dalla macchina
- EsplosioneRossa: è praticamente identico al GameObject precedente, con le sole differenze che genera un effetto particellare rosso e si attiva colpendo le casse di legno rosse

- Gui: anche in queste scene permette di gestire il menu di pausa, ma oltre a questo si occupa di visualizzare sulle due immagini mostrate all'utente il tempo ancora a disposizione per completare la gara e il numero del giro in corso
- BoxBonusMalus: GameObject che contiene tutte le casse, sia rosse che verdi, presenti lungo il tracciato
- TriggerContaGiri: il GameObject è composto da diversi collider, ovvero componenti in grado di intercettare le collisioni, ed ha il compito di monitorare lo stato di avanzamento della macchina lungo i tre giri di gara
- TriggerGiro1, TriggerGiro2, TriggerGiro3: come già scritto in precedenza nelle mappe sono presenti delle trappole dinamiche che scattano solo in determinati momenti; questi tre GameObject sono composti da componenti collider, e servono come interruttori per l'attivazione di tali trappole

4.8.3.3 Il problema delle prestazioni

Uno dei più grossi problemi con cui ci si è dovuti scontrare durante la realizzazione del programma riguarda le prestazioni: al termine dei primi tentativi di sviluppo dei vari livelli, infatti, ci si è resi conto che il gioco, fluido e veloce su pc, aveva un frame rate bassissimo su smartphone. Non ci si aspettava chiaramente un livello di prestazioni identico, ma neanche un divario di questa entità. Facendo una serie di prove si è verificato che il problema non riguardava il codice degli script, già sufficientemente ottimizzato, ma diverse impostazioni nell'editor delle scene, ed è stato molto attenuato agendo su questi aspetti:

- Camera culling: qualunque motore grafico 3D, renderizzando una scena, prima di tutto definisce come devono essere disegnati i modelli tridimensionali che si trovano nel campo di visione della camera, dopodiché li mette in ordine di profondità, in modo da determinare quali deve effettivamente disegnare e quali no, perché coperti da altri oggetti. La prima parte di questa procedura può essere molto lenta, dipende direttamente dalla quantità di modelli presenti in scena. Nello sviluppo del gioco sono stati creati diversi prototipi per testare le alternative a disposizione per eliminare il problema. Si è provata prima di tutto la funzionalità Occlusion Culling di Unity: questa consiste nella creazione di una matrice di dati, generata grazie ad una telecamera virtuale, che permette di definire prima dell'avvio dell'applicazione quali modelli e superfici devono essere tenute in

considerazione durante il rendering delle scene, in base alla posizione e all'orientamento della camera attiva.

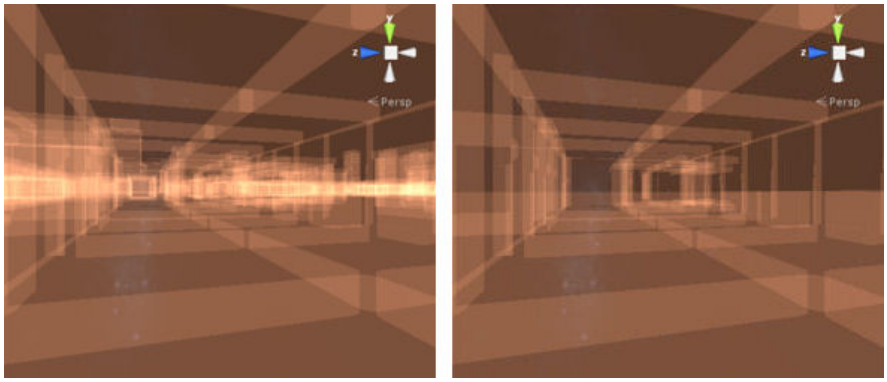

Figura 4.15: Una scena con Occlusion Culling disattivato (a sinistra) e attivato (a destra)

L'alternativa è stata bocciata perché, pur introducendo un miglioramento delle performance, in alcuni casi l'algoritmo di Occlusion Culling non riusciva a calcolare perfettamente i dati della matrice. Come secondo tentativo si è provato a restringere l'angolo di apertura delle camere, in modo da far diminuire il numero di oggetti da loro rilevabili. Anche in questo caso c'è stato un incremento delle prestazioni, ma il risultato grafico ne ha risentito parecchio: riducendo il campo di vista delle camere infatti si è ristretta molto la loro visuale, portando ad un generale peggioramento dell'esperienza di gioco. L'ultima alternativa provata è il tool Per-Layer Camera Culling, prelevato dall'Asset Store. Il suo funzionamento è estremamente semplice: le logiche di Unity prevedono che tutti i GameObject presenti nelle scene possano essere associati ad un Layer, e la lista di questi ultimi è liberamente modificabile dall'utente. Per-Layer Camera non fa altro che associare ad ogni Layer una distanza massima di visuale, oltre il quale gli oggetti 3D non vengono tenuti in considerazione durante il rendering. Sono stati creati dunque Layer con nomi intuitivi come "Da renderizzare lontani" o "Non renderizzati", e gli sono stati associati i vari oggetti presenti sulle scene del gioco; infine per ogni Layer e per ogni camera sono state definite adeguate distanze visive. Il risultato è stato ottimo; si è pertanto deciso di confermare l'utilizzo del tool

- Luci: inizialmente nel progetto si è fatto ampio uso di luci dinamiche, che vengono calcolate in tempo reale mentre il gioco è in esecuzione, ma presto ci si è resi conto della pesantezza computazionale di questo approccio. Si è deciso dunque di passare all'impiego di luci statiche, chiamate baked, che permettono di eliminare totalmente il carico di lavoro in real-time sia per CPU che per GPU. Questa tecnica si serve di un calcolo degli effetti dell'illuminazione che viene eseguito prima dell'avvio del gioco, e che tiene conto delle geometrie e delle luci presenti in scena. Può essere applicata sia alle luci singole che all'illuminazione globale. Il risultato di questo calcolo è una serie di lightmap, ovvero di texture map che contengono gli effetti della luce sugli oggetti della scena. Le lightmap si sommano alle texture diffuse moltiplicandone i valori RGB, e definendo il colore finale degli oggetti. Questo approccio è stato applicato a tutte le luci dell'applicazione
- Ombre: un'altra grande causa di frame rate basso su smartphone è l'applicazione delle ombre su oggetti e superfici. Si è deciso dunque durante lo sviluppo di eliminare qualunque tipo di ombra dal gioco
- Modelli 3D: in qualunque applicazione 3D il risultato visivo dipende in buona parte dai modelli 3D impiegati, e in generale vale la regola che più questi sono dettagliati, più l'effetto grafico è fotorealistico e coinvolgente. Ad un alto livello di dettaglio corrisponde però sempre una maggiore richiesta computazionale, in genere difficile da soddisfare su smartphone. Nella realizzazione di 3D4Amb Team Racing si è dunque scelto di non utilizzare modelli 3D molto definiti, rinunciando ad uno stile di rendering realistico, ma di sfruttare modelli 3D poco dettagliati (i cosiddetti low poly) per dare maggiore fluidità al gioco. La scena che ha avuto maggiore giovamento da questa decisione è MappaGara1, quella ambientata in città; bisogna infatti ammettere che tentando di utilizzare modelli di media complessità il livello su smartphone funzionava ad un frame rate così basso da risultare quasi impossibile da giocare
- Terrain: come già descritto in precedenza i terrain sono stati utilizzati per la creazione di alcune mappe di gioco. In realtà sarebbe stato meglio farne a meno. Sono infatti oggetti densi di informazioni: contengono i dati relativi alle pendenze e ai dislivelli, le modalità di applicazione delle varie texture usate per "pitturare" il terreno, diversi livelli di dettaglio, i dati relativi al posizionamento di alberi e

fogliame, ecc; sono dunque piuttosto pesanti da gestire. Purtroppo, in assenza di reali alternative si è stati costretti ad utilizzarli. Molto probabilmente sarebbe stato meglio creare i terreni direttamente con Maya, esportarli come mesh ed utilizzarli in Unity come semplici modelli 3D; ciò avrebbe sicuramente contribuito a rendere il gioco più performante

- Skybox: in Unity lo Skybox è l'oggetto messo a disposizione per riempire tutto lo spazio che rimane vuoto all'interno dei fotogrammi di gioco, perché nessun modello 3D e nessun componente della GUI vanno ad occuparlo. Solitamente rappresenta il cielo (da lì il nome), ma talvolta contiene elementi di sfondo come montagne, skyline, ecc. È costituito da 6 immagini in alta qualità che vanno a comporre un'unica texture, e dà la sensazione di trovarsi veramente nel paesaggio che si sta esplorando. Il suo problema è che non è un oggetto sufficientemente performante da essere suggerito per lo sviluppo di applicazioni mobili. Per questo motivo da Asset Store è stato scaricato un tool, SkySphere, che permette di ricreare l'effetto di uno skybox senza richiedere la stessa quantità di risorse. SkySphere infatti metta a disposizione una semplice semisfera, su cui è possibile applicare una delle texture integrate nello strumento. La semisfera deve essere poi posizionata e ridimensionata in modo da racchiudere l'intera mappa della scena. L'effetto finale è risultato essere molto piacevole e verosimile, tanto da far decidere di utilizzare SkySphere sia nella scena MappaBase che nelle tre scene associate ai percorsi di gara

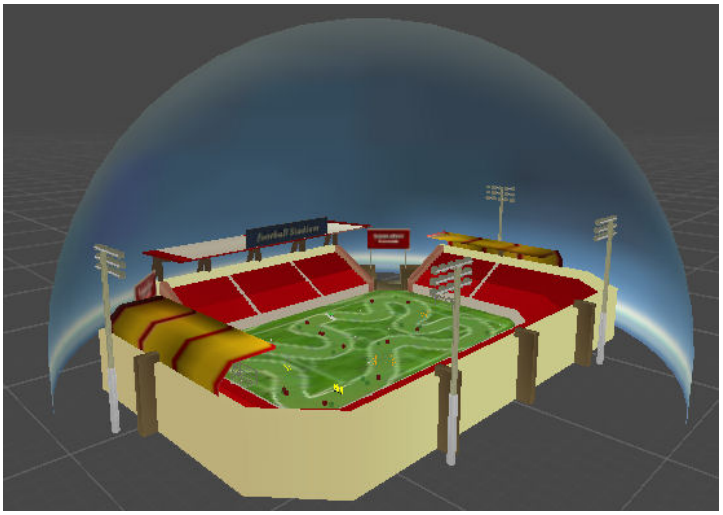

Figura 4.16: Utilizzo del tool SkySphere su una scena del gioco

4.8.3.4 Limitazione dell'area esplorabile

Uno degli aspetti che coinvolge diversi generi di videogioco è la limitazione dell'area in cui il giocatore può muoversi. Basti pensare ad esempio ai giochi d'avventura o agli sparatutto: a livello di progettazione è previsto che l'utente abbia una certa libertà d'azione, ma in alcuni punti il suo movimento deve essere bloccato, ad esempio quando si trova ai limiti della mappa di gioco. Anche in 3D4Amb Team Racing ci si è trovati a dover ragionare su questo discorso, e a valutare le alternative di implementazione dei limiti. La scelta del loro posizionamento è stata abbastanza semplice e ovvia:

- Nella mappa base la macchina non deve avere la possibilità di finire in mare, il suo movimento deve essere limitato alla terraferma
- Nelle mappe di gara la macchina non deve poter uscire dalla strada. Questo per due motivi: prima di tutto per non dare la possibilità all'utente di ridurre la lunghezza del tracciato, in secondo luogo per evitare di dover gestire una serie di situazioni un po' complicate. Basti pensare al primo livello di gioco: se la macchina dell'utente fosse libera di finire nel corso d'acqua sarebbe necessario implementare un sistema per il suo recupero e riposizionamento sulla strada

È stata molto facile anche la scelta degli oggetti da utilizzare per la rilevazione delle collisioni della macchina con i limiti. Unity infatti dispone di componenti molto validi, i MeshCollider, che permettono di intercettare le collisioni del veicolo con tutti i GameObject contenenti mesh. L'unico vero problema dunque è stato capire come effettivamente creare le mesh da associare ai limiti. Probabilmente il modo migliore sarebbe stato modellarle con Maya basandosi sulle mappe di gara, ma non essendo il sottoscritto un esperto di modellazione 3D si è dovuto scartare questo approccio. La scelta è dunque ricaduta sull'utilizzo di un tool esistente e scaricabile dall'Asset Store: Polydraw. Lo strumento permette infatti di disegnare modelli 3D direttamente da Unity, in modo molto semplice e intuitivo. Oltre a questo mette poi a disposizione una serie di opzioni per la configurazione delle collisioni e per la gestione delle geometrie create.

4.8.4 Panoramica sugli script

Dopo aver spiegato come sono state create le scene di gioco da editor di Unity è bene entrare nel cuore dell'applicazione, introducendo gli script. Questi sono file scritti in C# che contengono tutte le logiche del software, e che ne permettono il corretto

funzionamento. Come già visto in precedenza alcuni script vengono attivati trasversalmente in diverse scene, mentre altri vengono utilizzati durante il ciclo di vita di una sola scena. Tutti gli script sono sostanzialmente delle classi, che derivano dalla classe MonoBehaviour, cioè quella base di Unity. Il codice al loro interno è strutturato in Region, in modo da essere ordinato e ben leggibile; i Region più comuni sono i seguenti: "Costanti", "Enumerativi", "Variabili", "Procedure di Unity", "Procedure e funzioni custom". All'interno del Region "Procedure di Unity" si trovano tutti quei metodi propri di Unity, e che vengono lanciati autonomamente dallo stesso motore grafico:

- Start(): viene lanciato all'inizio del ciclo di vita dello script, non appena viene abilitato
- Update(): viene lanciato prima del rendering di ogni singolo frame di gioco
- FixedUpdate(): viene lanciato con una frequenza costante definita da progetto; solitamente è più alta rispetto alla frequenza di aggiornamento dell'immagine di gioco, dunque il metodo viene spesso utilizzato per gestire in modo preciso trasformazioni di oggetti che avvengono ad alta velocità, o simulazioni fisiche. In 3D4Amb Team Racing il metodo viene avviato ogni 0.02 secondi
- OnGUI(): viene lanciato sia per il rendering che per la gestione di tutti gli eventi GUI; questo significa che può essere richiamato diverse volte per ogni frame

All'interno del Region "Procedure e funzioni custom" si trovano invece tutti quei metodi e quelle funzioni non appartenenti a Unity, ma definiti dal sottoscritto.

È importante osservare che gli script e i relativi membri possono essere utilizzati da altri script solo se dichiarati come Static, oppure se associati come componenti a GameObject di scena (in questo modo le classi vengono istanziate); questo spiega la presenza nelle scene di oggetti come SaveLoadServer, Gui, GestoreInput, ecc. Nella realizzazione dell'applicazione in generale si è seguita la regola di associare sempre gli script a GameObject, e di definire come Static solo le variabili per cui è richiesto il mantenimento dei dati nei passaggi da una scena all'altra.

Di seguito sono elencati tutti gli script scritti per il gioco 3D4Amb Team Racing, suddivisi per cartella di appartenenza all'interno del progetto:

4.8.4.1 Comuni

- ControlloCameraMacchina: gestisce il posizionamento e l'orientamento delle camere durante l'esecuzione delle scene MappaBase, MappaGara0, MappaGara1 e MappaGara2
- ControlloMacchina: all'interno delle stesse scene si occupa di gestire tutte le dinamiche relative al controllo della macchina, utilizzando i dati di guida forniti in input
- Costanti: contiene una serie di costanti utilizzate da diversi script
- DisegnaTexCardboard: è lo script che disegna sul fotogramma completo di gioco la maschera per separare le immagini destinate ad occhio destro e sinistro, in modo da rendere l'applicazione utilizzabile con visore mobile
- FunzioniMapping: attualmente contiene una sola funzione di mapping condivisa; partendo dal presupposto che ad ogni valore di un certo tipo di dato possono essere associati uno o più valori di un altro tipo di dato, le funzioni di mapping si occupano di effettuare proprio questo lavoro di mappaggio
- GestioneCollisioneBox: lo script, attivo nelle scene di gara, gestisce le collisioni con le casse di legno e le relative esplosioni particellari; genera inoltre un evento ogni volta che avviene la collisione con una cassa
- GestionePenalizzazione: all'interno delle scene di gara si occupa di applicare la penalizzazione sull'immagine da mostrare all'occhio sano, sia tramite trasparenza degli oggetti che tramite oscuramento dei fotogrammi
- GestioneScenaGara: è probabilmente lo script più importante dell'intera applicazione, si occupa di gestire l'esecuzione dei vari livelli di gioco; tramite handlers gestisce diversi eventi generati da altri script, come la collisione con le casse di legno o il completamento del giro del tracciato in corso, mentre applicando una logica a stati finiti garantisce il corretto ciclo di vita della scena
- GestRisoluzioneGioco: partendo dalla definizione dello schermo dello smartphone imposta la risoluzione di gioco più adatta; mediante funzioni pubbliche espone inoltre tale dato all'esterno, in modo da essere utilizzato dagli script che gestiscono la Gui
- LetturaInput: è lo script che amministra tutti gli input di gioco. Si occupa di intercettare la pressione dei tasti di joystick, telecomandi e tastiere, e di tradurli in

effettivi input utilizzabili dall'applicazione. Permette inoltre la lettura e la scrittura delle configurazioni associate ai vari comandi di gioco

- InterfacciaInputMacchina: ha il semplice scopo di inoltrare i dati di input ottenuti dallo script LetturaInput verso lo script ControlloMacchina
- RilevatoreAvanzamentoGara: lo script, attivo nelle scene di gara, sfrutta alcuni trigger per tenere monitorato l'avanzamento della macchina lungo il percorso; espone inoltre il dato all'esterno dello script tramite eventi
- SaveLoadLocale: permette il salvataggio e il caricamento su device della configurazione del telecomando Bluetooth, delle credenziali di accesso utilizzate durante il login, del massimo livello di gioco raggiunto, e delle configurazioni di penalizzazione impostate
- SaveLoadServer: lo script si occupa di tutti gli aspetti che riguardano l'interazione dell'applicazione con il server; coordina dunque la funzione di login, il salvataggio dei dati di gara, il salvataggio e caricamento dei progressi di gioco

4.8.4.2 Scena Home

- ControlloCameraScenaHome: lo script gestisce la camera durante l'esecuzione della scena; per il corretto disegno dei menu, infatti, è necessario che la camera si sposti in diversi punti, in modo che la macchina in rotazione vada ad occupare diverse parti del fotogramma di gioco
- GestioneMenuScenaHome: coordina il disegno delle varie schermate presenti nella scena e l'interazione con esse; attraverso la generazione di alcuni eventi fornisce allo script GestioneScenaHome le richieste di avvio partita (con o senza login), le modifiche di occhio pigro e impostazioni di penalizzazione e i cambiamenti nella configurazione del telecomando Bluetooth
- GestioneScenaHome: così come lo script GestioneScenaGara rappresenta il centro di controllo delle scene di gara, allo stesso modo questo script rappresenta il cardine del funzionamento della scena Home. All'avvio del gioco lancia il caricamento dei dati salvati su device, e attraverso la gestione di eventi generati da altri script si occupa di avviare la partita (con o senza login) e di salvare su device tutte le informazioni relative alle configurazioni di tasti e penalizzazioni
- RotazioneMacchinaScenaHome: lo script semplicemente controlla la rotazione della macchina durante l'esecuzione della scena

4.8.4.3 Scena mappa base

- RilevazioneInizioGara: Utilizzando dei trigger presenti in scena permette di identificare quando la macchina si trova sopra una delle piattaforme circolari di avvio gara, e genera un evento contenente la richiesta di lancio della relativa scena
- GestioneScenaMappaBase: è l'ultimo script vitale dell'applicazione, insieme a GestioneScenaHome e GestioneScenaGara; gestisce infatti il funzionamento della scena MappaBase sfruttando una logica a stati finiti, mentre tramite handler dell'evento generato dallo script RilevazioneInizioGara si occupa di avviare i vari livelli di gioco. Un'altra funzione fondamentale che svolge è quella di avviare il salvataggio dei progressi compiuti su device o server al termine di una gara

4.8.4.4 Scena mappa gara 0

- GestTrappoleDinamicheGara0: lo script coordina il ciclo di vita delle trappole dinamiche presenti nella scena MappaGara0, ovvero il masso che cade da una montagna durante il primo giro e l'albero che cade all'inizio del terzo giro

4.8.4.5 Scena mappa gara 1

- GestTrappoleDinamicheGara1: lo script ha la stessa funzione di quello appena descritto, con la sostanziale differenza che è applicato alla scena MappaGara1; in questo caso le trappole dinamiche consistono in due automobili che compaiono sul tracciato, rispettivamente al primo e al terzo giro, e che si muovono in direzione opposta rispetto a quella controllata dall'utente

4.8.4.6 Scena mappa gara 2

- CadutaOmini1: nei primi due livelli di gara le trappole dinamiche si attivano applicando forze direttamente ai GameObject interessati. Nel terzo livello invece una delle trappole non viene attivata direttamente: una forza viene applicata ad un pallone, che a usa volta colpisce una barriera costituita da sagome rigide, che infine cade sul tracciato di gara. Questo script ha il compito di gestire la collisione del pallone con l'attrezzo da allenamento, facendolo cadere
- GestTrappoleDinamicheGara2: ha la stessa funzione degli script GestTrappoleDinamicheGara0 e GestTrappoleDinamicheGara1 descritti in precedenza, ma si applica alla scena MappaGara2; in questo caso le trappole

dinamiche consistono in un pallone che durante il primo giro del livello inizia a rotolare verso la macchina, e in un attrezzo da allenamento che durante il terzo giro, come appena descritto, viene colpito da un pallone e cade sul percorso di gara

4.8.5 Ordine di avvio degli script

Uno degli aspetti più delicati a cui si è dovuta prestare la massima attenzione durante l'implementazione del gioco è l'ordine di avvio degli script. Se in una scena sono presenti due GameObject infatti, e ognuno di essi ha associato uno script che contiene il metodo Start(), all'avvio della scena Unity esegue i due metodi in modo totalmente casuale, causando potenzialmente diversi bug. Vediamo un esempio pratico: nella scena Home dell'applicazione è presente il GameObject Menu, e fra i suoi componenti troviamo lo script GestioneMenuScenaHome. Oltre a questo è presente il GameObject Scena, che contiene fra i suoi componenti lo script GestRisoluzioneGioco.
All'interno del metodo Start() del primo script è possibile individuare le seguenti righe di codice:

```
int screenWidth = GestRisoluzioneGioco.getLarghezzaRis();
int screenHeight = GestRisoluzioneGioco.getAltezzaRis();
```

All'interno del metodo Start() del secondo script invece troviamo il seguente snippet:

```
//definisco la lista di risoluzioni supportate dal gioco
List<clsDatiRisoluzione> listaRisoluzioniGioco;
listaRisoluzioniGioco = new List<clsDatiRisoluzione>();
listaRisoluzioniGioco.Add(new clsDatiRisoluzione(16f / 9f, 853, 480)); //1.77
listaRisoluzioniGioco.Add(new clsDatiRisoluzione(5f / 3f, 800, 480)); //1.66
listaRisoluzioniGioco.Add(new clsDatiRisoluzione(16f / 10f, 768, 480)); //1.6
listaRisoluzioniGioco.Add(new clsDatiRisoluzione(3f / 2f, 720, 480)); //1.5
//individuo partendo dalla risoluzione dello schermo del device, quale risolu-
zione di gioco è maggiormente adattabile
…
…
…
//imposto per il gioco la risoluzione più adatta
Screen.SetResolution(listaRisoluzioniGioco[indice].width, listaRisoluzioni-
Gioco[indice].height, true);
larghezzaRis = listaRisoluzioniGioco[indice].width;
altezzaRis = listaRisoluzioniGioco[indice].height;
```

Le istruzioni del primo blocco hanno lo scopo di recuperare la risoluzione del gioco in modo da poter disegnare successivamente i menu a schermo in modo corretto, mentre quelle del secondo blocco hanno lo scopo di identificare e impostare la corretta risoluzione di gioco fra quelle a disposizione. Chiaramente se viene prima eseguito il

secondo blocco di codice, e poi il primo, tutto funziona in modo corretto, ma in caso contrario nelle variabili screenWidth e screenHeight verrebbero salvati valori 0, causando poi diversi malfunzionamenti durante il disegno dei menu. Ecco dunque perché è estremamente importante l'ordine di esecuzione degli script. Unity mette a disposizione una finestra di editor, denominata "Script Execution Order", con la quale è possibile definire l'ordine di avvio dei vari script, per evitare problemi come quello appena descritto.

Per definire il corretto ordine è stato necessario verificare le dipendenze fra i vari script, che ricordo essere sostanzialmente delle classi; sono state tenute in considerazione solo le dipendenze relative all'uso dei dati, non quelle relative ai loro tipi. Questa analisi ha permesso di definire il seguente diagramma delle classi UML:

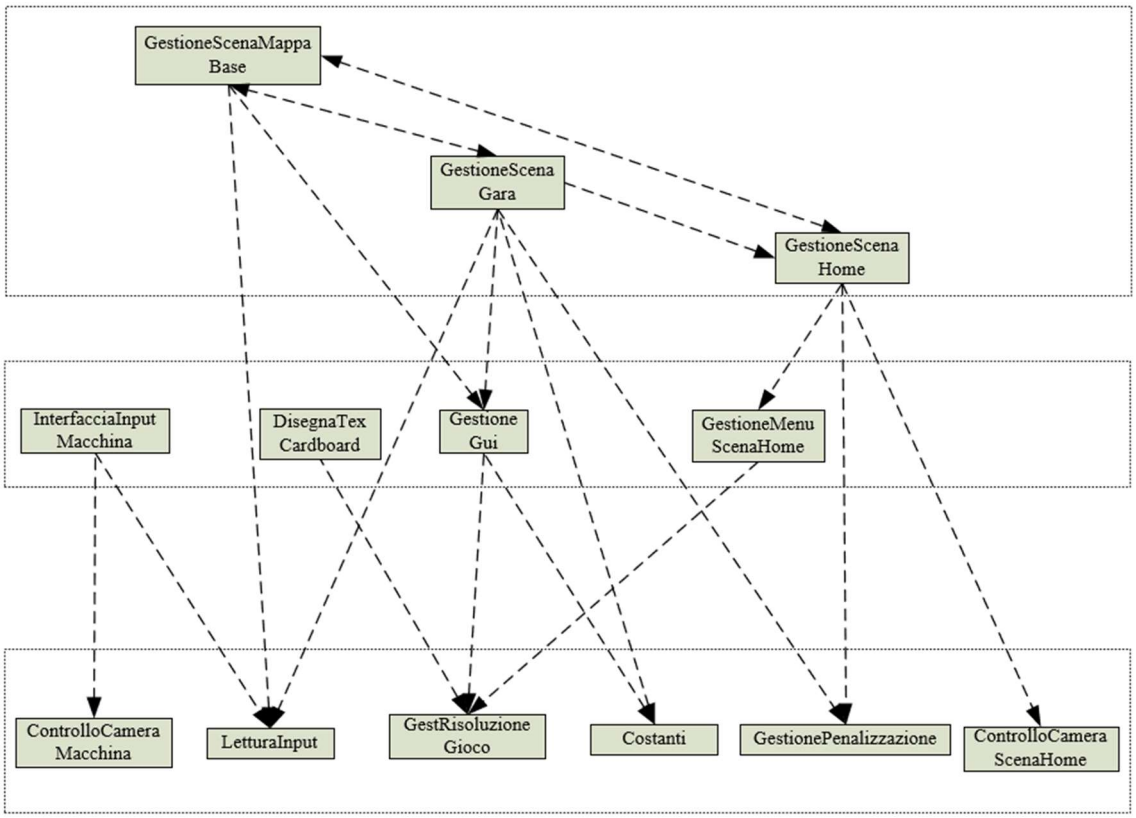

Figura 4.17: Diagramma UML rappresentante le dipendenze di dati fra le classi

Il diagramma mostra come sostanzialmente esistano tre livelli di dipendenza. Del primo fanno parte gli script basilari del gioco, quelli che gestiscono la logica delle scene. È bene sottolineare questi non sono mai in esecuzione concorrente, dunque nonostante ci sia fra

loro uno scambio di dati, questo non può mai causare malfunzionamenti. Negli altri due livelli si può invece notare come non ci sia alcuna dipendenza fra gli script appartenenti alla stessa fascia, dunque anche in questo caso l'ordine di avvio non è importante. Occorre precisare che un'intera categoria di script non è stata inserita nel diagramma, ovvero quella contenente tutte le classi prive di dipendenze di dati.
L'ordine di esecuzione degli script nel progetto 3D4Amb Team Racing è stato dunque definito in questo modo: prima di tutto vengono avviati quelli privi di dipendenze, poi quelli di terzo livello, quelli di secondo, e infine quelli presenti nella prima fascia.

4.8.6 Implementazione funzionalità

Per entrare ancora più a fondo nei dettagli dell'applicazione è utile illustrare come sono state implementate alcune funzionalità del gioco.

4.8.6.1 Controllo della camera

Può sembrare un aspetto secondario, ma in realtà il controllo della camera si è rivelato essere un punto piuttosto critico nello sviluppo del software. La camera doveva essere infatti gestita in modo tale da trovarsi sempre dietro la macchina, ad una distanza tale da permettere all'utente di esaminare il tracciato sia nel tratto ravvicinato che in lontananza. Allo stesso modo però le modifiche nella traslazione e nell'orientamento della camera dovevano essere abbastanza smussate in modo da evitare all'utente variazioni nelle inquadrature troppo brusche. Per soddisfare entrambi i requisiti si è deciso di sfruttare un filtro in media mobile, per la definizione sia della posizione della camera che del suo orientamento.
Prima di tutto, nel metodo Start() dello script ControlloCameraMacchina, sono stati definiti i valori preferiti per la distanza della camera dalla macchina, e del punto rispetto alla macchina stessa verso cui è richiesto che la camera inquadri:

```
//voglio che si cerchi di mantenere la telecamera 3.5 metri sopra e 9 metri
dietro la macchina
distPosCamera = new Vector3(0f, 3.5f, -9f);
//voglio che il target della camera sia esattamente 10 metri di fronte alla
macchina
distTargetCamera = new Vector3(0f, 0f, 10f);
```

Allo stesso modo sono state inizializzate le liste dati utilizzate per l'applicazione del filtro in media mobile:

```
listaTargetCamera = new List<Vector3>();
listaPosCamera = new List<Vector3>();
```

All'interno del metodo FixedUpdate() dello stesso script è stato poi implementato il controllo vero e proprio della telecamera:

```
//aggiungo in lista l'ultima posizione desiderata per la camera e l'ultimo target
desiderato
listaPosCamera.Add(riferimentoCamera.transform.TransformPoint(distPosCamera));
listaTargetCamera.Add(riferimentoCamera.transform.TransformPoint(distTargetCamera
));
//faccio in modo che nelle liste non ci siano mai più elementi di quelli
richiesti
while (listaPosCamera.Count > finestraMediaMobile)
{listaPosCamera.RemoveAt(0);
 listaTargetCamera.RemoveAt(0);
}

//posiziono la camera solo se ho a disposizione almeno un dato in lista
if (listaPosCamera.Count > 0)
{Vector3 targetCamera, posCamera;
 targetCamera.x = 0; targetCamera.y = 0; targetCamera.z = 0;
 posCamera.x = 0; posCamera.y = 0; posCamera.z = 0;
 int elementiInLista = listaPosCamera.Count;
 for (byte i = 0; i < elementiInLista; i++)
  {targetCamera.x = targetCamera.x + listaTargetCamera[i].x;
   targetCamera.y = targetCamera.y + listaTargetCamera[i].y;
   targetCamera.z = targetCamera.z + listaTargetCamera[i].z;
   posCamera.x = posCamera.x + listaPosCamera[i].x;
   posCamera.y = posCamera.y + listaPosCamera[i].y;
   posCamera.z = posCamera.z + listaPosCamera[i].z;
  }
 targetCamera.x = targetCamera.x / elementiInLista;
 targetCamera.y = targetCamera.y / elementiInLista;
 targetCamera.z = targetCamera.z / elementiInLista;
 posCamera.x = posCamera.x / elementiInLista;
 posCamera.y = posCamera.y / elementiInLista;
 posCamera.z = posCamera.z / elementiInLista;
 gameObject.transform.position = posCamera;
 gameObject.transform.LookAt(targetCamera);
}
```

È bene sottolineare che RiferimentoCamera punta al GameObject della macchina. L'unico vero problema riscontrato durante lo sviluppo di questa funzionalità è stato uno spiacevole effetto di jitter (tremolio) della camera, che compariva quando la macchina era quasi ferma. Questo è stato risolto introducendo una nuova dimensione alla finestra di media mobile, più grande rispetto a quella di default, attiva solo a macchina quasi ferma.

```
//conversione velocità da m/s in km/h
float velAssoluta = Mathf.Abs(rbRiferimento.velocity.magnitude * 3.6f);
int finestraMediaMobile;
if (velAssoluta < SOGLIA_VEL_CAMBIO_FINESTRA_MEDIA_MOBILE) //1 km/h
    finestraMediaMobile = FINESTRA_MEDIA_MOBILE_MAX; //30
else
    finestraMediaMobile = FINESTRA_MEDIA_MOBILE_MIN; //20
```

È bene puntualizzare che il controllo della camera è stato inserito nel metodo FixedUpdate(), e non nel metodo Update(), in modo da essere gestito con una temporizzazione più precisa, e dunque per avere un risultato grafico più stabile.

4.8.6.2 Controllo della macchina

La funzionalità di controllo della macchina è stata probabilmente la più difficoltosa da implementare dell'intero progetto, soprattutto a causa delle scarse conoscenze del sottoscritto sull'argomento. È stato necessario infatti raccogliere informazioni sul funzionamento reale delle automobili, riguardanti sia aspetti meccanici che dinamici, e si è dovuto poi sviluppare diverse logiche per rendere più piacevole e verosimile la guida della macchina. Tutto ciò che riguarda questa funzionalità è stato implementato nello script ControlloMacchina.

Fondamentalmente la guida si basa sui WheelCollider, cioè quei collider di Unity creati appositamente per realizzare veicoli di terra. Hanno un sistema di gestione delle collisioni integrato, e controllano autonomamente la fisica degli pneumatici. Ogni ruota nel software non è altro che un GameObject, contenente il modello 3D della ruota stessa, e un WheelCollider:

```
[System.Serializable]
public class Ruota : System.Object
 {public WheelCollider collider;
  public GameObject gameObject;
 }
```

La guida dell'auto consiste fondamentalmente nell'applicazione di forze sull'asse trasversale dei WheelCollider, in modo da far muovere il veicolo in avanti e indietro, e sulla loro rotazione sull'asse verticale, in modo da far sterzare la macchina.

L'automobile dal punto di vista fisico è composta semplicemente dalle quattro ruote e da un RigidBody relativo al corpo macchina:

```
public Ruota davantiSx, davantiDx, dietroSx, dietroDx;
public Rigidbody rigidBodyMacchina;
```

A livello di codice la prima operazione eseguita sul mezzo, nel metodo Start(), è stata abbassarne il centro di massa:

```
Vector3 centroDiMassa = rigidBodyMacchina.centerOfMass;
centroDiMassa.y -= 0.5f;
rigidBodyMacchina.centerOfMass = centroDiMassa;
```

Questo perché sulle automobili reali il centro di massa non si trova mai esattamente nel punto centrale del corpo macchina, ma un po' più in basso e più avanti, vicino al motore. Nel progetto è stato sufficiente abbassarlo di mezzo metro per garantire maggiore stabilità alla vettura.

L'accelerazione e la frenata sono stati implementati agendo sulle variabili MotorTorque e BrakeTorque dei WheelCollider associati alle ruote anteriori del veicolo; si è infatti optato per la realizzazione di un mezzo a trazione anteriore. Un aspetto interessante riguarda l'utilizzo del tasto di gioco Frena/Retro. Lo stesso tasto infatti gestisce due funzioni di controllo diverse:

- La frenata, nel caso in cui il mezzo si stia muovendo al di sopra di una certa velocità
- L'attivazione della retromarcia, nel caso in cui il mezzo si fermo o quasi

Per gestire la doppia situazione è stato implementato il seguente segmento di codice, che sfrutta l'orientamento della macchina e la sua velocità corrente per determinare se è richiesta l'attivazione della retromarcia o della frenata:

```
if (inputAcceleraFrenaRetro < 0)
 {//frenata/retro
  Vector3 velNorm, orientNorm;
  float sum;
  //calcolo i vettori normalizzata di orientamento e velocità della macchina e di
velocità (normalizzati)
  velNorm = Vector3.Normalize(rigidBodyMacchina.velocity);
  orientNorm = Vector3.Normalize(rigidBodyMacchina.transform.forward);
  sum = (velNorm + orientNorm).magnitude;
  //ci si aspetta che se l'auto si sta muovendo in avanti, la somma dei due
vettori avrà modulo pari a ~2 (i vettori si sommano), mentre se l'auto si sta
muovendo indietro, la somma dei due vettori avrà modulo pari a ~0 (i vettori si
annullano). Uso 1 come valore di soglia per distinguere i due casi. Utilizzo
anche una soglia di velocità (SOGLIA_VEL_ATTIVAZIONE_RETRO), sotto il quale
considero sempre la macchina in fase di retromarcia
  if (sum < 1 || velAttuale< SOGLIA_VEL_ATTIVAZIONE_RETRO)
   {//sto andando in retromarcia, o mi sto muovendo in avanti a velocità
bassissima, quindi devo attivare la retro
    if (inputAcceleraFrenaRetro > 0)
     {davantiSx.collider.motorTorque = -inputAcceleraFrenaRetro * coppiaMaxVel;
      davantiDx.collider.motorTorque = -inputAcceleraFrenaRetro * coppiaMaxVel;
     }
    else
     {davantiSx.collider.motorTorque = inputAcceleraFrenaRetro * coppiaMaxVel;
```

```
      davantiDx.collider.motorTorque = inputAcceleraFrenaRetro * coppiaMaxVel;
     }
    davantiDx.collider.brakeTorque = 0;
    davantiSx.collider.brakeTorque = 0;
   }
  else
   {//mi sto muovendo in avanti e sto frenando
    davantiSx.collider.brakeTorque = -inputAcceleraFrenaRetro * coppiaMaxVel;
    davantiDx.collider.brakeTorque = -inputAcceleraFrenaRetro * coppiaMaxVel;
   }
  }
```

InpupAcceleraFreno è un valore compreso fra -1 e 1 rilevato dalla classe LetturaInput, mentre coppiaMaxVel è un valore costante che indica la coppia massima applicata alle ruote. La sterzata della macchina avviene in modo molto più semplice, modificando il valore della variabile SteerAngle dei WheelCollider:

```
davantiSx.collider.steerAngle = maxSteeringAngle * inputSterza;
davantiDx.collider.steerAngle = maxSteeringAngle * inputSterza;
```

MaxSteerAngle non è altro che il massimo angolo di rotazione raggiungibile dalle ruote, e inputSterza è un altro valore compreso fra -1 e 1 rilevato dalla classe LetturaInput.

Una cosa che occorre sottolineare è che le rotazioni dei WheelCollider non si riflettono sulle mesh rappresentanti le ruote; per questo motivo è stato creato il metodo VisualizeWheel, che viene lanciato al termine del metodo Update():

```
private void VisualizeWheel(Ruota wheel)
    {Quaternion rot;
     Vector3 pos;
     wheel.collider.GetWorldPose(out pos, out rot);
     wheel.gameObject.transform.position = pos;
     wheel.gameObject.transform.rotation = rot;
    }
```

Un'ultima caratteristica che è stato necessario implementare per il controllo della macchina è la barra antirollio. Senza di essa infatti succedeva che in seguito ad una curva, percorsa anche a velocità modesta, la macchina perdesse aderenza sulla ruota interna, e spesso si ribaltava. La barra antirollio è un organo meccanico che mette in connessione le sospensioni delle ruote di uno stesso asse, che permette di trasferire parte della forza dalla sospensione non in compressione a quella in esercizio, garantendo maggiore stabilità.

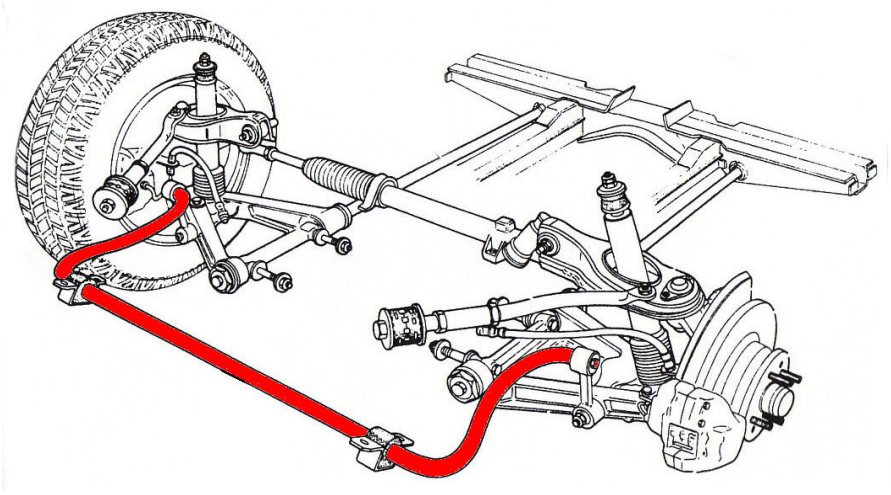

Figura 4.18: Struttura tipica della barra antirollio

Bisogna ammettere che la sua implementazione è stata recuperata direttamente dal sito di Unity; è stata poi inserita nel metodo FixedUpdate(), in modo da essere attivata con un'alta frequenza.

4.8.6.3 Applicazione maschera Cardboard

L'applicazione della maschera che permette di separare le immagini da mostrare ad occhio destro e ad occhio sinistro, e dunque di utilizzare l'applicazione con visore mobile, è estremamente semplice. Ad una delle due camere di scena, in particolare a quella che genera il frame mostrato all'occhio sinistro, viene applicato lo script DisegnaTexCardboard, i cui metodi Start() e OnGUI() sono di seguito descritti:

```
void Start()
{rectSchermo = new Rect(0, 0, GestRisoluzioneGioco.getLarghezzaRis(),
GestRisoluzioneGioco.getAltezzaRis());}

void OnGUI()
{GUI.depth = depthGui;
 GUI.DrawTexture(rectSchermo, texCardboard);}
```

Come si può vedere la funzionalità viene implementata con sole tre righe di codice:

- All'avvio dello script si definisce l'area che deve essere occupata dalla texture; questa deve ricoprire l'intero schermo dello smartphone, e dunque è pari alla risoluzione del gioco
- La modifica diretta della variabile GUI.depth nel metodo OnGUI() permette di assicurarsi che la texture venga disegnata in primo piano, davanti a qualunque altro elemento della GUI

- Infine tramite metodo GUI.DrawTexture la texture viene fisicamente applicata al fotogramma completo di gioco

È bene sottolineare il motivo per cui lo script viene applicato ad una sola delle camere di scena: l'istruzione GUI.DrawTexture non agisce a livello di frame generato dalla camera, ma a livello di intera immagine di gioco, dunque applicando lo stesso script ad entrambe le camere il risultato non cambierebbe, semplicemente si applicherebbe due volte la stessa maschera, con l'unico risultato di ridurre le performance dell'applicazione.

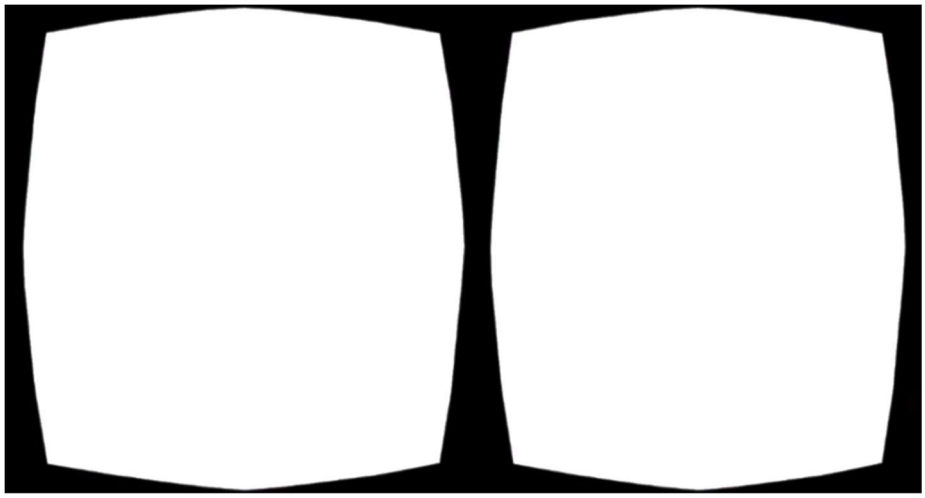

Figura 4.19: La maschera applicata ai fotogrammi di gioco

4.8.6.4 Salvataggio e caricamento dati su device

Per il salvataggio e il caricamento dei dati, implementato nello script SaveLoadLocale, si è fatto ampio uso della classe PlayerPrefs del namespace UnityEngine. Prima di tutto si è definito per ogni informazione da salvare/caricare un nome univoco, identificato da stringa:

```
const string stPercMaxTrasparenza = "PercMaxTrasparenza";
const string stPercMinOscuramentoScena = "PercMinOscuramentoScena ";
const string stLivelloRaggiunto = "LivelloRaggiunto";
const string stOcchioPigro = "OcchioPigro";
…
…
```

È stato poi creato un enumerativo, contenente tutti i dati che il gioco ha bisogno di salvare e caricare su device:

```
public enum enumDatiLocali
    {inputSu,
     inputGiù,
     …
     percMinOscuramentoScena,
     percMaxOscuramentoScena
    };
```

È bene evidenziare che non è presente una relazione univoca tra le stringhe e l'enumerativo, dal momento che ad ogni enumerativo possono corrispondere una o più stringhe; questo perché per ogni dato può essere necessario salvare/caricare diverse informazioni. Per fare un esempio, all'enumerativo enumDatiLocali.inputGiù sono stati associati i campi su device "InputSuConfigurato", "InputSuTipo, "InputSuNomeAsse", "InputSuValAsse", "InputSuValButton", perché quando si va a salvare la configurazione del tasto sono diverse le informazioni di cui si deve tener traccia.

Lo script mette a disposizione un metodo per il salvataggio e una funzione per il caricamento.

Il primo è così strutturato:

```
public void salvaDatiLocali(enumDatiLocali[] dati, object[] val)
  {for (int i = 0; i < dati.Length; i++)
    {if (dati[i] == enumDatiLocali.inputSu)
       salvaInput(val[i], stInputSuConfigurato, stInputSuTipo, stInputSuNomeAsse,
       stInputSuValAsse, stInputSuValButton);
     if (dati[i] == enumDatiLocali.inputGiù)
       salvaInput(val[i], stInputGiùConfigurato, stInputGiùTipo,
       stInputGiùNomeAsse, stInputGiùValAsse, stInputGiùValButton);
     …
     if (dati[i] == enumDatiLocali.livelloRaggiunto)
       PlayerPrefs.SetInt(stLivelloRaggiunto, (int)val[i]);
     if (dati[i] == enumDatiLocali.occhioPigro)
       PlayerPrefs.SetInt(stOcchioPigro,
FunzioniMapping.enumDxSx2byte((GestionePenalizzazione.enumDxSx)val[i]));
   }
  }
```

Come si può vedere il metodo è strutturato in modo tale da accettare un numero variabile di dati da salvare. Questi vengono poi elaborati con modalità diverse:

- Alcuni vengono salvati in modo quasi diretto, è solo necessario un piccolo cast dei dati (vedi ad esempio come viene gestito l'enumerativo enumDatiLocali.livelloRaggiunto)
- Altri richiedono un mapping con altri tipi di dati prima di essere effettivamente salvati (vedi la gestione dell'enumerativo enumDatiLocali.occhioPigro)
- Altri ancora, ovvero le configurazioni dei comandi di gioco, come si è visto non sono dati atomici, ma sono costituiti da diverse informazioni, dunque il

salvataggio su device è delegato ad un ulteriore metodo, chiamato salvaInput (che comunque a sua volta sfrutta sempre la classe PlayerPrefs)

La funzione per il caricamento dei dati da device segue gli stessi principi, ed è così strutturata:

```
public object[] caricaDatiLocali(enumDatiLocali[] dati)
 {object[] datiRitorno = new object[dati.Length];
  for (int i = 0; i < dati.Length; i++)
    {if (dati[i] == enumDatiLocali.inputSu)
       caricaInput(out datiRitorno[i], stInputSuConfigurato, stInputSuTipo,
       stInputSuNomeAsse, stInputSuValAsse, stInputSuValButton);
     if (dati[i] == enumDatiLocali.inputGiù)
       caricaInput(out datiRitorno[i], stInputGiùConfigurato, stInputGiùTipo,
       stInputGiùNomeAsse, stInputGiùValAsse, stInputGiùValButton);
     …
     if (dati[i] == enumDatiLocali.livelloRaggiunto)
       datiRitorno[i] = PlayerPrefs.GetInt(stLivelloRaggiunto, -1);
     if (dati[i] == enumDatiLocali.occhioPigro)
       datiRitorno[i] = int2enumDxSx2(PlayerPrefs.GetInt(stOcchioPigro, 1));
    }
  return datiRitorno;
 }
```

4.8.6.5 Salvataggio e caricamento dati su server

In caso di partita online avvengono diverse interazioni fra l'applicazione e il server:

- All'avvio del gioco avviene la fase di login
- Dopo questa fase viene caricato l'ultimo livello raggiunto dall'utente
- Al termine di un livello di gara vengono salvati i dati della prova
- Se si completa l'ultimo livello a disposizione viene salvato il progresso di gioco

Tutte queste interazioni avvengono in modo quasi identico, l'unica differenza sostanziale è che le prime due utilizzano delle richieste GET per trasmettere i dati, mentre le altre li incapsulano in stringhe Json (JavaScript Object Notation). GET è una richiesta Http che prevede che le informazioni vengano aggiunte nella parte finale dell'Url, mentre Json è un semplice formato per lo scambio di dati, totalmente indipendente dal linguaggio di programmazione in uso.

Per capire meglio come le interazioni con il server avvengono viene di seguito mostrata la funzione salvaDatiProva, presente nello script SaveLoadServer:

```
public enumRisComunicazioneServer salvaDatiProva(clsDatiProva datiProva, out bool
esitoSalvataggio)
    {//definizione variabili
     enumRisComunicazioneServer esitoCom;
     bool sospendiRichiesta;
     Stopwatch swTimeout;
     string stJson;
```

```
string url;
Dictionary<string, string> headers;
bool esito = false;
//inizializzazione variabili
sospendiRichiesta = false;
swTimeout = new Stopwatch();
headers = new Dictionary<string, string>();
//preparazione dati da mandare a server via json
stJson = JsonUtility.ToJson(datiProva);
url = "http://localhost:82/3d4amb/salvaDatiProva.php";
headers.Add("Content-Type", "application/json");
byte[] body = Encoding.UTF8.GetBytes(stJson);
//invio dati a server
swTimeout.Start();
WWW www = new WWW(url, body, headers);
//resto in attesa della risposta per TOT secondi, dopodiché interrompo la
  richiesta
while (!www.isDone && !sospendiRichiesta)
  {if (swTimeout.Elapsed.TotalSeconds >= SEC_ATTESA_RISPOSTA)
    sospendiRichiesta = true;
  }
swTimeout.Stop();
//rilevazione esito comunicazione
if (sospendiRichiesta)
  esitoCom = enumRisComunicazioneServer.timeoutServer;
else
  {if (www.error == null)
    {//1: salvataggio andato a buon fine
     if (www.text == "1")
      {esito = true;
       esitoCom = enumRisComunicazioneServer.comunicazioneOk;
      }
     else
      esitoCom = enumRisComunicazioneServer.malfunzionamentoServer;
    }
    else
     esitoCom = enumRisComunicazioneServer.serverNonRaggiungibile;
  }
www.Dispose();
www = null;
esitoSalvataggio = esito;
return esitoCom;
}
```

Come si può notare la funzione restituisce una variabile di tipo EnumRisComunicazioneServer, cioè un enumerativo che può assumere i valori:

- ComunicazioneOk: quando non vengono rilevati problemi di comunicazione
- ServerNonRaggiungibile: quando durante la comunicazione viene generato un errore
- TimeoutServer: quando dopo un'attesa di 10 secondi il server non ha ancora risposto alla richiesta
- MalfunzionamentoServer: quando il server risponde all'invocazione con dei dati che l'applicazione non è in grado di interpretare

Attraverso un parametro out la funzione restituisce anche il risultato della richiesta, che in questo caso è l'esito del salvataggio; l'altro parametro, datiProva, contiene tutti i dati della gara appena eseguita, ed è di tipo clsDatiProva, una classe serializzabile.
Come si può vedere nella prima parte della funzione viene creata la stringa Json contenente le informazioni presenti in datiProva, dopodiché la stringa viene convertita in vettore di bytes e inglobata nell'oggetto UnityEngine.WWW. Successivamente il software rimane in attesa per 10 secondi della risposta dal server (10 è il contenuto della costante SEC_ATTESA_RISPOSTA); in base all'arrivo o meno della risposta e al suo contenuto vengono generati i valori di output della funzione.
Come già anticipato in precedenza la parte server del sistema non è stata ancora realizzata, per questo motivo nell'Url è presente la dicitura "localhost"; durante lo sviluppo infatti si è simulato in locale il funzionamento del server web.

4.8.6.6 Menu iniziali

Come abbiamo già visto durante l'esecuzione della scena Home vengono visualizzate le varie schermate relative ai menu di gioco iniziali. La logica della scena è affidata allo script GestioneScenaHome, mentre della gestione della Gui si occupa lo script GestioneMenuScenaHome. In quest'ultimo si fa uso della variabile pubblica menuAttivo, che è di tipo enumQualeMenu.

```
public enum enumQualeMenu
    {login,
     loginFallito,
     errComunicazioneServer,
     opzioni,
     setComandi,
     setTasto,
     configOcchio,
     configPenalizzazione
    };
```

Come si può vedere l'enumerativo indica tutte le schermate che possono comparire durante l'esecuzione della scena; all'interno metodo OnGUI() lo script gestisce la variabile come da seguente snippet:

```
switch (menuAttivo)
   {case enumQualeMenu.login:
     disegnaMenuLogin();
     break;
    …
    case enumQualeMenu.opzioni:
     disegnaMenuOpzioni();
     break;
    …
```

```
    case enumQualeMenu.configPenalizzazione:
     disegnaMenuConfigPenalizzazione();
     break;
   }
```

Questo significa che ogni schermata è disegnata in modo indipendente dalle altre, minimizzando la complessità del componente, e dando la possibilità di inserire facilmente nuove schermate o di eliminare quelle esistenti.

Il disegno dei vari menu avviene sfruttando la classe GUILayout del namespace UnityEngine; come esempio di seguito è esposto il metodo disegnaMenuOpzioni:

```
private void disegnaMenuOpzioni()
 {//in base al numero di elementi e alla dimensione del menu definisco le
  //nuove regole di stile
  configuraStileMenu(5, 1, areaMenuOpzioni);

  GUILayout.BeginArea(areaMenuOpzioni);
  if (GUILayout.Button(stImpostaComandi, stileButton))
    menuAttivo = enumQualeMenu.setComandi;
  if (GUILayout.Button(stImpostaOcchioPigro, stileButton))
    menuAttivo = enumQualeMenu.configOcchio;
  if (GUILayout.Button(stConfiguraTrasparenza, stileButton))
   {parPenalizzazioneInEsame = enumParametroPenalizzazione.trasparenzaOggetti;
    menuAttivo = enumQualeMenu.configPenalizzazione;
   }
  if (GUILayout.Button(stConfiguraOscuramento, stileButton))
   {parPenalizzazioneInEsame = enumParametroPenalizzazione.oscuramentoScena;
    menuAttivo = enumQualeMenu.configPenalizzazione;
   }
  if (GUILayout.Button(stIndietro, stileButton))
    menuAttivo = enumQualeMenu.login;
  GUILayout.EndArea();
 }
```

Il metodo ConfiguraStileMenu permette di definire la dimensione e i margini dei vari oggetti grafici, in questo caso tutti tasti, a partire dall'area di visualizzazione del menu (areaMenuOpzioni) e dal numero elementi per riga (1) e colonna (5). L'interazione con i tasti genera poi la modifica di alcune variabili di controllo, come la stessa menuAttivo.

In altri casi, come nella schermata in cui è possibile selezionare l'occhio pigro, l'interazione con i tasti genera degli eventi gestiti dallo script GestioneScenaHome:

```
if (GUILayout.Button(stDestro, stileBtnOcchioDx))
   {occhioDaPenalizzare = GestionePenalizzazione.enumDxSx.Sx;
    evtSelezioneOcchioPigro();
   }
```

Con evtSelezioneOcchioPigro dichiarato come:

```
public event HandlerSelOcchioPigro evtSelezioneOcchioPigro;
public delegate void HandlerSelOcchioPigro();
```

Nel modulo Start() dello script GestioneScenaHome è presente la registrazione del gestore dell'evento:

```
mainMenu.evtSelezioneOcchioPigro += new
GestioneMenuScenaHome.HandlerSelOcchioPigro(setOcchioDaPenalizzare);
```

Con la relativa procedura di gestione:

```
private void setOcchioDaPenalizzare()
 {GestionePenalizzazione.occhioDaPenalizzare = mainMenu.occhioDaPenalizzare;
  GestionePenalizzazione.enumDxSx occhioPigro;
  if (GestionePenalizzazione.occhioDaPenalizzare ==
       GestionePenalizzazione.enumDxSx.Dx)
     occhioPigro = GestionePenalizzazione.enumDxSx.Sx;
  else
     occhioPigro = GestionePenalizzazione.enumDxSx.Dx;
  saveLoadLocale.salvaDatiLocali(new SaveLoadLocale.enumDatiLocali[] {
   SaveLoadLocale.enumDatiLocali.occhioPigro }, new object[] { occhioPigro });
 }
```

4.8.6.7 Gestione input

La gestione dei comandi di input nel progetto è interamente gestita dallo script LetturaInput. È previsto l'impiego di 8 input di gioco, definiti dall'enumerativo enumTasti:

- Comandi Su, Giù e Ok (funzionanti solo durante la visualizzazione dei menu di pausa)
- Comando Pausa (attivo sia durante la fase attiva di gioco che durante la visualizzazione dei menu di pausa)
- Comandi SterzaDx, SterzaSx, Accelera, FrenaRetro (utilizzabili solo durante la fase attiva di gioco)

Ad ognuno di essi è associata una configurazione, definita mediante la struttura ConfInput:

```
public struct confInput
  {public bool configurato;
   public enumTipoInput tipo;
   public string nomeAsse;
   public int valAsse;
   public KeyCode valButton;
  };
```

Il campo Configurato indica se per il comando è stata impostata una configurazione dall'utente, mentre Tipo indica la tipologia di input utilizzata, e può assumere valore enumTipoInput.Tasto o enumTipoPunto.Asse. Questa distinzione viene fatta perché solitamente i bottoni dei joystick vengono rilevati da Unity come tasti, mentre le frecce direzionali vengono rilevate come assi. Ad ogni tasto Unity associa un codice, che viene salvato nel campo ValButton, mentre la pressione di un asse viene identificata dal suo

nome e dal suo valore, che può essere uguale a -1 o +1. I campi NomeAsse e ValAsse della struttura contengono questi ultimi due dati.

All'interno del metodo Update() lo script verifica l'attivazione dei vari comandi, e genera dei dati di input disponibili al resto del programma.

```
public float inputAcceleraFrenaRetro, inputSterza;
public bool inputSu, inputGiù, inputPausa, inputOk;
```

I primi due sono gli ingressi utilizzati per il controllo della macchina, e hanno sempre valori decimali compresi fra -1 e +1; il fatto che non siano variabili intere permette di far raggiungere loro i valori limite in modo graduale, garantendo accelerazioni, frenate e sterzate più morbide e verosimili. I successivi quattro sono gli ingressi utilizzati per gestire il menu di pausa, ed essendo semplici interruttori on-off sono contenuti in variabili booleane.

Di seguito come esempio è riportato uno snippet che indica come la variabile inputAcceleraFrenaRetro viene modificata in seguito all'attivazione del comando frenaRetro:

```
if (getInputSettato(confTastoFrenaRetro))
    {if (inputAcceleraFrenaRetro <= 0)
      {inputAcceleraFrenaRetro = inputAcceleraFrenaRetro - velPedaleRetro *
       Time.deltaTime;
       if (inputAcceleraFrenaRetro < -1)
         inputAcceleraFrenaRetro = -1;
      }
     else
      {inputAcceleraFrenaRetro = inputAcceleraFrenaRetro - velPedaleFreno *
       Time.deltaTime;
       if (inputAcceleraFrenaRetro < 0)
          inputAcceleraFrenaRetro = 0;
      }
    }
```

Come si può notare il valore di inputAcceleraFrenaRetro dipende sia dalla quantità di tempo nella quale il tasto frenaRetro viene premuto, sia dalla velocità di variazione impostata. In questo caso specifico velPedaleRetro è pari a 10 valori al secondo, il che significa che il passaggio da valore +1 a valore 0 della variabile di input richiede un decimo di secondo.

La funzione getInputSettato intercetta la pressione dei pulsanti e degli assi del telecomando Bluetooth tramite queste semplici istruzioni:

```
private bool getInputSettato(confInput input)
  {if (input.tipo == enumTipoInput.tasto)
    {if (Input.GetKey(input.valButton))
        return true;
     else
```

```
            return false;
        }
    else
        {if (Input.GetAxisRaw(input.nomeAsse) == input.valAsse)
            return true;
         else
            return false;
        }
    }
```

4.8.6.8 Gestione livelli di gioco

L'esecuzione delle gare viene coordinata dallo script GestioneScenaGara, che a sua volta fa riferimento ad altri script per la gestione della penalizzazione, delle collisioni con le casse di legno, del menu di pausa, ecc.

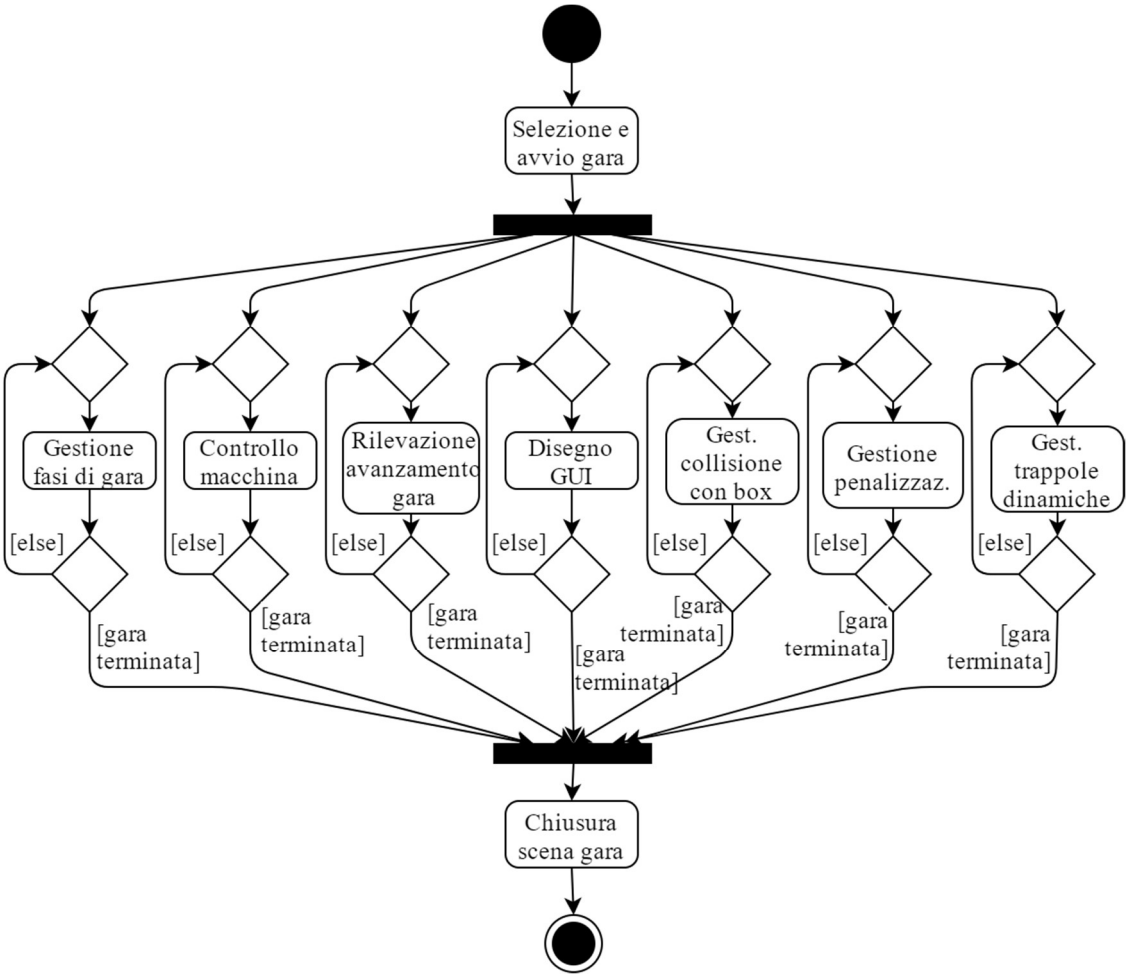

Figura 4.20: Diagramma di attività UML che indica alcuni degli script parallelamente attivi durante l'esecuzione delle gare

Anche in questo caso si segue la logica di una macchina a stati. Il ciclo di vita della scena attiva viene suddiviso in fasi, che possono assumere i seguenti valori:

```
private enum enumFaseScena
    {attesaInizioGara,
     countdown,
     garaInCorso,
     attesaVisualRisultato,
     visualRisultato
    };
```

La prima fase è quella in cui la macchina viene mostrata sulla griglia di partenza, la gara non è ancora iniziata, e l'utente non ha il controllo sul mezzo. La seconda fase è quella di countdown, che viene visualizzato a schermo e ha una durata di tre secondi. Nella fase GaraInCorso avviene l'esecuzione della gara vera e propria. L'enumerativo attesaVisualRisultato viene utilizzato nel momento in cui la gara termina, o perché il tempo a disposizione è terminato, o perché la macchina ha completato i tre giri richiesti, e consiste in una piccola pausa a cui segue l'attivazione della fase finale, visualRisultato, ovvero quella in cui compare a schermo un messaggio indicante l'esito di gara. La variabile impiegata come contenitore della fase in corso si chiama faseScena.

Il menu di pausa può essere visualizzato durante una delle prime tre fasi dell'elenco, e la sua comparsa è gestita per mezzo di una variabile booleana, chiamata inPausa. Il controllo dell'intera scena avviene all'interno del metodo Update():

```
void Update()
  {//gestione fasi scena
   switch (faseScena)
     {case enumFaseScena.attesaInizioGara:
         …
      case enumFaseScena.countdown:
         …
      case enumFaseScena.garaInCorso:
         //calcolo secondi disponibili per completamento percorso
         …
         //aggiornamento GUI con secondi rimanenti
         …
         //aggiorno la penalizzazione
         …
         //scrittura in GUI in caso di collisioni bonus e malus
         …
      case enumFaseScena.attesaVisualRisultato:
         …
      case enumFaseScena.visualRisultato:
         //visualizzazione in GUI esito di gara
         …
         //salvataggiodati gara su server, in caso di partita online
         …
     }
   //gestione input pausa
```

```
    if (faseScena != enumFaseScena.attesaVisualRisultato && faseScena !=
    enumFaseScena.visualRisultato)
      {if (letturaInput.inputPausa)
         {…
         }
       else
         {…
         }
       //gestione menu di pausa
       if (inPausa)
         {…
         }
      }
  }
```

Altri aspetti di gara che non vengono gestiti all'interno del metodo vengono controllati tramite eventi; ci si riferisce alle rilevazioni di avanzamenti nel percorso e di collisione con le casse di legno. In questi casi nel metodo Start() dello script vengono registrati i gestori degli eventi:

```
rilevatoreAvanzamentoGara.evtGiroCompletato += new RilevatoreAvanzamentoGara.Hand-
                                               lerGiroCompletato(gestioneGiroCompletato);
gestioneCollisioneBox.evtBoxColpito += new GestioneCollisioneBox.HandlerBoxCol-
                                       pito(gestioneBoxColpito);
rilevatoreAvanzamentoGara.evtPercGiroCompletata += new RilevatoreAvanzamento-
                         Gara.HandlerPercGiroCompletata(gestionePercGiroCompletata);
```

Gli eventi vengono generati a loro volta dagli script RilevatoreAvanzamentoGara e GestioneCollisioneBox.

Entrando nel dettaglio di come vengono gestite le casse di legno bisogna precisare che ad ogni giro di gara vengono visualizzate dieci diverse casse per colore; questo permette di mantenere il livello di difficoltà, e dunque la concentrazione del giocatore, costanti per tutta la durata del livello.

All'interno del metodo Start() dello script GestioneScenaGara viene definito quali box devono essere sbloccati all'inizio di ogni nuovo giro:

```
objDaSbloccarePerGiro = new List<GameObject[]>();
GameObject[] primoGiro = new GameObject[] { GameObject.Find("BoxBonusGiro1"),
          GameObject.Find("BoxMalusGiro1"), GameObject.Find("TriggerGiro1") };
GameObject[] secondoGiro = new GameObject[] { GameObject.Find("BoxBonusGiro2"),
          GameObject.Find("BoxMalusGiro2"), GameObject.Find("TriggerGiro2") };
GameObject[] terzoGiro = new GameObject[] { GameObject.Find("BoxBonusGiro3"),
          GameObject.Find("BoxMalusGiro3"), GameObject.Find("TriggerGiro3") };
objDaSbloccarePerGiro.Add(primoGiro);
objDaSbloccarePerGiro.Add(secondoGiro);
objDaSbloccarePerGiro.Add(terzoGiro);
```

La procedura di effettiva abilitazione/disabilitazione delle casse avviene al termine della fase di countdown, e all'interno del metodo gestioneGiroCompletato:

```
private void bloccaSbloccaBoxGiro(byte giro)
  {bool sblocca;
   for (byte i = 0; i < objDaSbloccarePerGiro.Count; i++)
    {sblocca = (giro - 1 == i);
     for (byte i2 = 0; i2 < objDaSbloccarePerGiro[i].Length; i2++)
      {objDaSbloccarePerGiro[i][i2].SetActive(sblocca);
      }
    }
  }
```

La collisione con i box viene intercettata all'interno dello script GestioneCollisioneBox, mediante il metodo di Unity OnTriggerEnter, che viene lanciato ogni volta che la mesh della macchina interseca un trigger:

```
private void OnTriggerEnter(Collider other)
  {bool colpitoBonus = colliderBoxBonus.Contains(other);
   bool colpitoMalus = colliderBoxMalus.Contains(other);
   if (colpitoBonus || colpitoMalus)
    {//trovo tutti i GameObject associati al collider
      GameObject box = other.gameObject;
      GameObject boxSx = other.transform.GetChild(0).gameObject;
      GameObject boxDx = other.transform.GetChild(1).gameObject;
      //nascondo subito i box renderizzati
      boxSx.SetActive(false);
      boxDx.SetActive(false);
      //posiziono l'esplosione al centro del box (devo fare una piccola
        correzione manuale)
      Vector3 posEsplosione = box.transform.position;
      posEsplosione.x -= 1;
      posEsplosione.y += 1;
      posEsplosione.z += 1;
      ParticleSystem esplosione;
      if (colpitoBonus)
          esplosione = esplosioneVerde;
      else
          esplosione = esplosioneRossa;
      esplosione.transform.position = posEsplosione;
      //avvio l'esplosione
      esplosione.Clear();
      esplosione.Play();
      //distruggo tutti i GameObject associati al box colpito
      GameObject.Destroy(boxSx);
      GameObject.Destroy(boxDx);
      GameObject.Destroy(box);
      //genero l'evento di box colpito
      evtBoxColpito(colpitoBonus);
    }
  }
```

Come si vede dal segmento di codice la logica è molto semplice: prima di tutto viene verificato se è stata colpita una cassa rossa o verde (ColliderBoxBonus e ColliderBoxMalus contengono i box attivi nel giro corrente), dopodiché questa viene eliminata dalla scena, facendo comparire al suo posto l'effetto particellare di esplosione.

Al termine della sua animazione tutti i GameObject associati alla cassa vengono distrutti, e viene generato l'evento di collisione rilevata.

Un'ultima funzionalità che vale la pena illustrare è quella della penalizzazione dell'occhio sano. La trasparenza degli oggetti è stata gestita creando per ogni oggetto grafico logico due GameObject contenenti il corrispondente modello 3D; attraverso la configurazione delle camere di scena il primo GameObject viene visualizzato solo all'occhio destro, mentre il secondo solo all'occhio sinistro. Con la scena configurata in questo modo la penalizzazione avviene semplicemente modificando il canale alpha del materiale applicato al GameObject che si intende rendere più o meno trasparente.

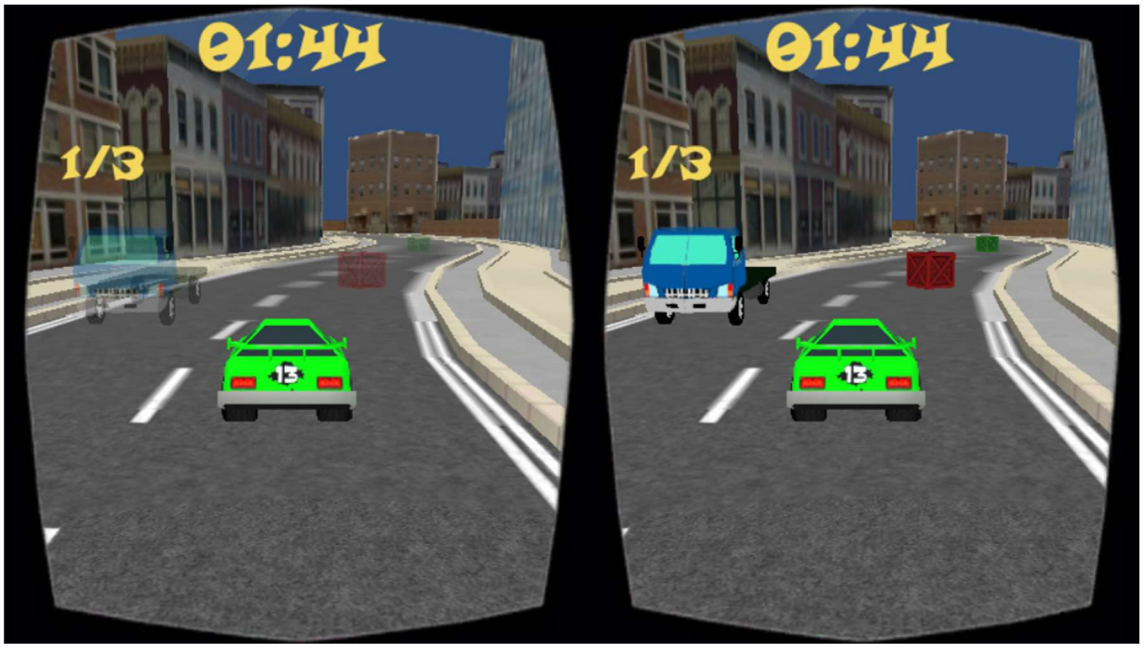

Figura 4.21: Esempio di penalizzazione mediante trasparenza degli oggetti

Molto simile è la gestione della penalizzazione via oscuramento dell'intera scena: in questo caso si è deciso di porre un piccolo rettangolo nero proprio di fronte ad ogni camera, e di controllare la luminosità dell'immagine di gioco attraverso il canale alpha del suo materiale.

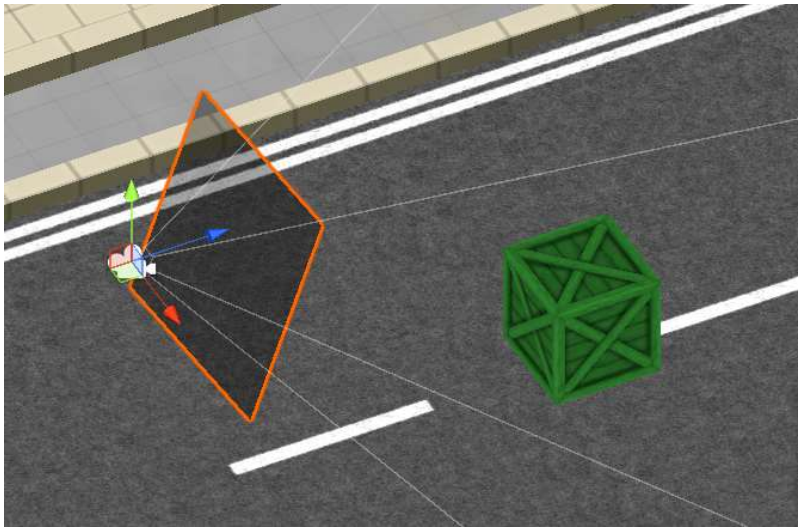

Figura 4.22: Logica di oscuramento dell'intera scena di gioco

Ovviamente al rettangolo che si trova di fronte alla camera impiegata per generare l'immagine visualizzata all'occhio ambliope è applicato un materiale con canale alpha sempre uguale a 0.

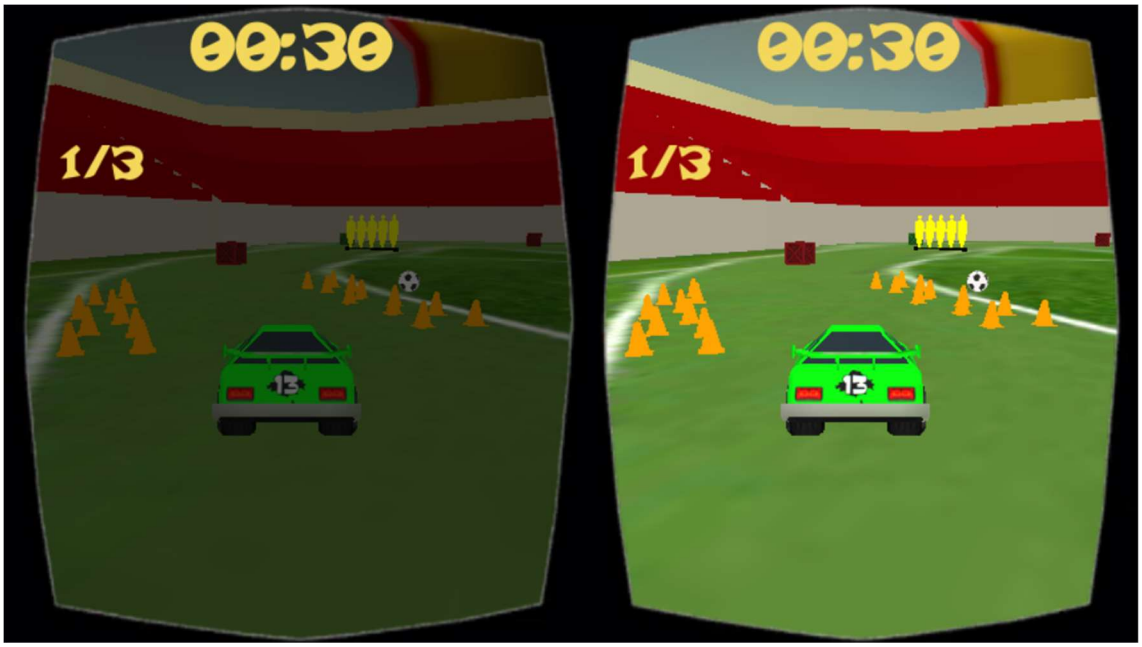

Figura 4.23: Esempio di penalizzazione mediante oscuramento del fotogramma di gioco

Per non degradare le prestazioni dell'applicazione tutte le modifiche dei materiali avvengono ad una frequenza molto bassa, 0.5Hz; questo non peggiora l'esperienza di

gioco, dal momento che comunque all'interno di 2 secondi il valore del canale alpha non può andare a modificarsi di una quantità significativa.

L'aggiornamento dei dati di penalizzazione avviene all'interno del metodo Update() dello script GestioneScenaGara:

```
if (swAggiornaPenalizzazione.Elapsed.TotalSeconds >= SEC_AGGIORNAMENTO_PENALIZZA-
ZIONE || !swAggiornaPenalizzazione.IsRunning)
   {//il livello di penalizzazione deve essere inversamente proporzionale al
    tempo disponibile per concludere il percorso
    gestionePenalizzazione.livelloTraspObjCorrente = GestionePenalizzazione.li-
     velloMaxTrasparenzaObj - (float)secondiRimanentiFineGara / (float)tempoDi-
     sponibileGara * (GestionePenalizzazione.livelloMaxTrasparenzaObj - Gestio-
     nePenalizzazione.livelloMinTrasparenzaObj);
    gestionePenalizzazione.livelloOscurScenaCorrente = GestionePenalizzazione.li-
     velloMaxOscurScena - (float)secondiRimanentiFineGara / (float)tempoDisponi-
     bileGara *(GestionePenalizzazione.livelloMaxOscurScena -GestionePenalizza-
     zione.livelloMinOscurScena);
    swAggiornaPenalizzazione.Reset();
    swAggiornaPenalizzazione.Start();
   }
```

La temporizzazione avviene mediante lo stopwatch swAggiornaPenalizzazione e la costante SEC_AGGIORNAMENTO_PENALIZZAZIONE (che vale 2 secondi); come si vede dal segmento di codice i due valori di penalizzazione vengono calcolati in modo da essere inversamente proporzionali al tempo ancora disponibile per il completamento del percorso.

L'applicazione dei livelli di trasparenza e oscuramento calcolati avviene all'interno dello script GestionePenalizzazione, nel metodo Update(): prima di tutto si calcola il valore da applicare al canale alpha dei materiali visualizzati all'occhio sano, verificando che resti sempre compreso fra 0 e 1. Successivamente si verifica che tale valore sia diverso da quello correntemente impostato; se la condizione è rispettata, la lista di materiali in esame viene recuperata, e ad ogni suo elemento viene applicato il valore calcolato. Ad esempio, la penalizzazione tramite trasparenza degli oggetti di gioco avviene in questo modo:

```
//calcolo il livello di alpha richiesto
livelloRichiestoAlpha = 1 - livelloTraspObjCorrente / 100;

//mi assicuro che sia sempre compreso fra 0 e 1
if (livelloRichiestoAlpha > 1)
  livelloRichiestoAlpha = 1;
if (livelloRichiestoAlpha < 0)
  livelloRichiestoAlpha = 0;

//se i due livelli sono diversi modifico l'alpha applicato ai materiali
if (livelloAttualeAlpha != livelloRichiestoAlpha && !vettMatNullo)
 {Material[] matVariazioneAlpha;
```

```
  if (occhioDaPenalizzare == enumDxSx.Dx)
    matVariazioneAlpha = matVisibiliDx;
  else
    matVariazioneAlpha = matVisibiliSx;
  for (byte i = 0; i < matVariazioneAlpha.Length; i++)
   {Color col = matVariazioneAlpha[i].color;
    col.a = livelloRichiestoAlpha;
    matVariazioneAlpha[i].SetColor("_Color", col);
   }
 }
```

La penalizzazione tramite oscuramento dell'intera scena è pressoché identica.

4.9 Testing e debug

La qualità di un prodotto software si può ottenere attraverso due approcci complementari:

- Definendo e utilizzando un processo di qualità durante la sua realizzazione
- Eseguendo al termine del suo sviluppo un insieme di attività di controllo

Per quanto riguarda il gioco 3D4Amb Team Racing si è deciso di raggiungere l'obiettivo della qualità soprattutto tramite il primo approccio: è stato infatti dedicato molto tempo all'analisi, alla progettazione e allo sviluppo, mentre per quanto riguarda il testing bisogna ammettere che è stato svolto in maniera sbrigativa e poco strutturata. Questo non significa che l'applicazione è instabile, significa solamente che il processo non è stato ben organizzato: non è stato pianificato un piano di test, né si è tenuta traccia dei casi di test effettivamente eseguiti.

Prima di tutto occorre sottolineare che sono stati fatti solo test black box, il che significa che le varie funzionalità sono state verificate considerando il software come una scatola nera, ignorando dunque la sua struttura interna. Alcuni aspetti più critici, come il salvataggio e il caricamento dei dati su device e server, sono stati provati a fondo, simulando anche malfunzionamenti e utilizzi di valori errati e limite. Altre funzionalità meno importanti invece, come il controllo dei menu di pausa, sono state provate in modo più blando, il più delle volte semplicemente giocando con l'applicazione.

È bene sottolineare che dopo una serie di test preliminari è stata svolta una fase di alpha-testing, ovvero il gioco è stato utilizzato per diverse ore dal sottoscritto, cercando di simulare il più possibile i comportamenti degli utenti finali.

Al termine di questa c'è stata una brevissima sessione di beta-testing, nella quale un utente realmente affetto da ambliopia monolaterale ha avuto modo di provare l'applicazione.

I risultati sono stati piuttosto confortanti: non si sono mai presentati crash del software, né malfunzionamenti significativi; anche a livello di performance è stata rilevata una buona fluidità di gioco anche su smartphone con potenza computazionale ridotta.
Gli unici malfunzionamenti che si sono presentati sono sostanzialmente due:

- è stato accertato che in alcuni casi la macchina riesce ad oltrepassare i limiti impostati sul percorso, anche a velocità ridotta. Questo probabilmente è causato da un qualche bug dello strumento Polydraw, che a quanto pare genera meshCollider non perfetti. Il problema non è stato attualmente risolto, anche se la soluzione è a portata di mano. Sarebbe sufficiente infatti modellare le mesh associate ai limiti di gioco con Maya o software simile, e importarle nel progetto, andando a sostituire quelle generate da Polydraw. Questa attività non è stata svolta in fase di debug unicamente a causa delle scarse conoscenze di modellazione 3D del sottoscritto
- l'altro problema rilevato durante il testing riguarda l'utilizzo di smartphone con schermi con risoluzione diversa rispetto a quella definita durante lo sviluppo dell'applicazione. Un errore che inizialmente era stato compiuto era infatti l'aver progettato il gioco solo per una risoluzione, 854x480, lasciando a Unity il compito di "spalmare" ogni fotogramma sulla reale dimensione dello schermo in uso. Un effetto indesiderato di questa logica si era presentato ad esempio lanciando l'applicazione su uno smartphone con schermo di dimensioni 1280x720; si era notato infatti che i testi che compaiono nei pulsanti dei menu avevano una dimensione minima, e dunque erano quasi illeggibili. Il problema è stato risolto prima del rilascio del gioco introducendo nel progetto lo script GestRisoluzioneGioco. Al suo interno state definite quattro risoluzioni base (854x480, 800x480, 768x480 e 720x480) relative a quattro diversi aspect ratio (1.78, 1.67, 1.6, 1.5); al suo avvio lo script applica poi al gioco quella risoluzione il cui aspect ratio si avvicina di più a quello dello schermo in uso. Grazie al software Bluestacks è stato possibile verificare che grazie al nuovo script il malfunzionamento è stato completamente eliminato

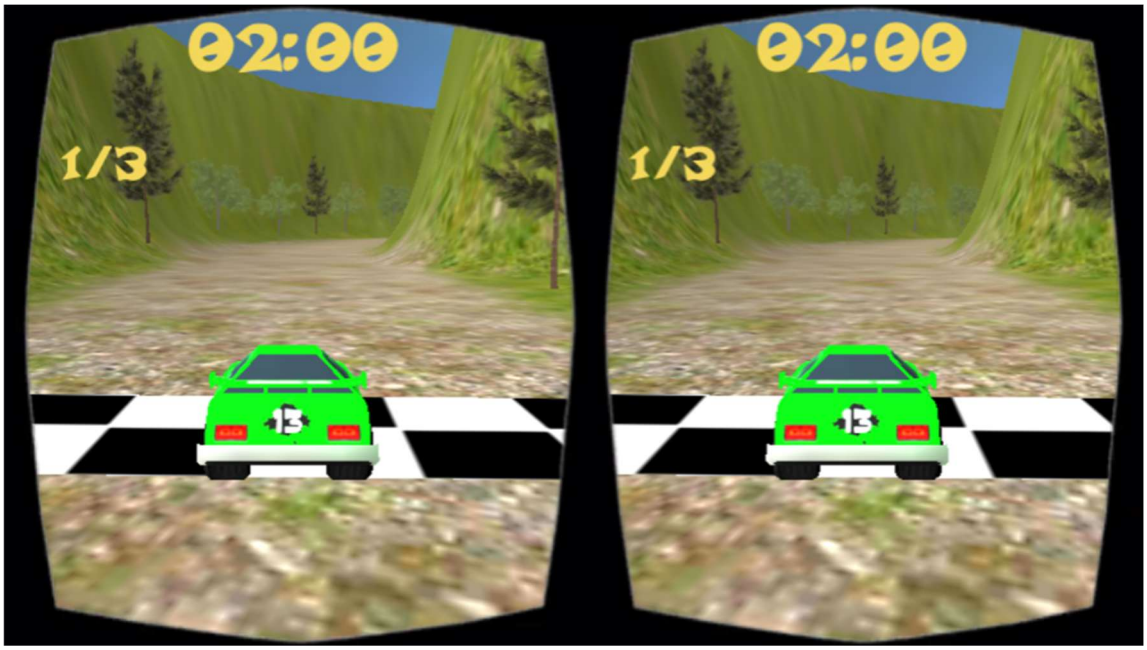

Figura 4.24: Immagine di gioco su schermo con risoluzione 480x854

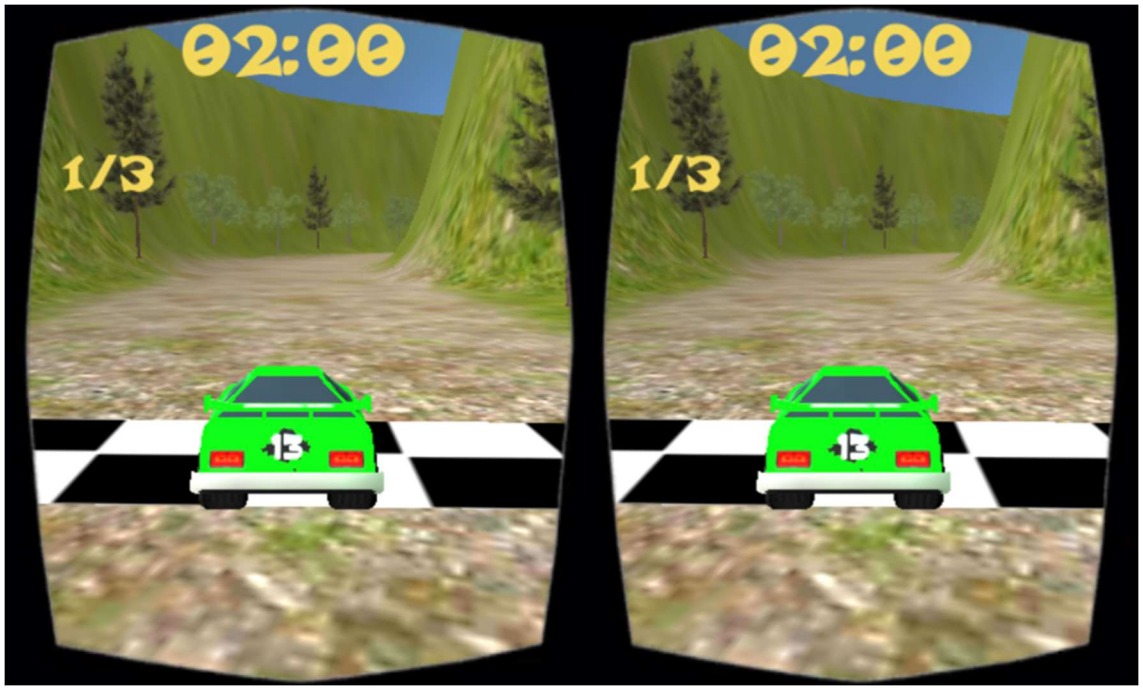

Figura 4.25: Immagine di gioco su schermo con risoluzione 1280x768

Capitolo 5: Conclusioni e sviluppi futuri

L'obiettivo di questo lavoro di tesi era quello di fornire al progetto 3D4Amb una nuova applicazione per il trattamento dell'ambliopia monolaterale. Quello che ci si augura è di essere riusciti a creare uno strumento realmente utile, che possa in futuro contribuire al miglioramento delle condizioni di vita dei pazienti affetti da tale patologia. A tal proposito sarebbe utile rendere disponibile l'applicazione ad uno o più centri ospedalieri, in modo che i pazienti possano testarla e fornire il proprio feedback. Nel frattempo sarebbe bene anche verificare le sue reali potenzialità, magari mediante un piccolo trial clinico. Per quanto riguarda i suoi sviluppi futuri le direzioni in cui potersi muovere sono diverse:

- dal punto di vista medicale si può pensare di potenziare l'applicazione eliminando dal sistema il telecomando Bluetooth, e introducendo un sistema di rilevazione delle mani come Leap Motion. L'idea sarebbe quella di far posizionare all'utente le mani come se stesse impugnando un volante, e di controllare la macchina muovendole come per svoltare a destra e a sinistra. Tutto ciò a vantaggio di un'integrazione visuo-motoria, già presente in sistemi come Vivid Vision

Figura 5.1: Rilevazione movimento delle mani via sistema Leap Motion

- dal punto di vista dell'evoluzione del gioco si potrebbero andare ad inserire nuovi livelli, per aumentarne il livello di longevità; relativamente a questo discorso è

bene ricordare che il software è stato progettato in modo tale da rendere facile l'aggiunta di nuove mappe di gara

- un'ultima direzione può essere quella del consolidamento di ciò che è stato realizzato finora: si potrebbe ottimizzare il gioco, rendendolo sempre più performante, oppure correggere i piccoli bug presenti, o infine renderlo visivamente più gradevole, ad esempio implementando delle ombre statiche ai vari GameObject presenti nelle scene

Si spera che altri studenti in futuro possano portare avanti questo lavoro, in modo da fornire un sistema per il trattamento dell'ambliopia sempre più valido.

Bibliografia

[1] Repka , Kraker, Holmes, Summers, Glaser, Barhardt e Tien, «Atropine vs patching for treatment of moderate amblyopia: follow-up at 15 years of age of a randomized clinical trial,» 2014.

[2] Hess, Mansouri e Thompson, «A new binocular approach to the treatment of amblyopia in adults well beyond the critical period of visual development,» 2010.

[3] Li, Thompson, Deng, Chan, Yu e Hess, «Dichoptic training enables the adult amblyopic brain to learn,» 2013.

[4] Guo, Babu, Black, Bobier, Lam, Dai, Gao, Hess, Jenkins, Jiang, Kowal, Parag, South, Staffieri, Walker, Wadham e Thompson, «Binocular treatment of amblyopia using videogames (BRAVO): study protocol for a randomised controlled trial,» 2016.

[5] Ferrera, «Effects of Attention in Visual Cortex: Linking Single Neuron Physiology to Visual Detection and Discrimination,» 2016.

[6] Martinez-Trujillo e Treue, «Feature-based attention increases the selectivity of population responses in primate visual cortex,» 2004.

[7] Silver, Ress e Heeger, «Neural correlates of sustained spatial attention in human early visual cortex,» 2007.

[8] Tsirlin, Colpa, Goltz e Wong, «Behavioral training as new treatment for adult amblyopia: a meta-analysis and systematic reviewmeta-analysis of behavioral training for amblyopia,» 2015.

[9] Posner, «Orienting of attention,» 1980.

[10] Melmoth e Grant, «Advantages of binocular vision for the control of reaching and grasping,» 2006.

[11] Aderman, Deiner, Gupta, Blaha e Levin, «Dichoptic virtual reality therapy for amblyopia in adults,» 2015.

[12] Žiak, Holm, Halička, Mojžiš e Piñero, «Amblyopia treatment of adults with dichoptic training using the virtual reality oculus rift head mounted display: preliminary results,» 2017.

[13] Medisim Ltd, «Services,» [Online]. Available: http://www.bino-vision.com/services.html.

[14] «Ericsson Mobility Report,» 2016.

[15] IDC Corporate USA, «Smartphone OS Market Share, 2017 Q1,» [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os.

[16] Google LLC, «Android - Dashboards,» [Online]. Available: https://developer.android.com/about/dashboards/index.html.