ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

TITOLO DELL'ELABORATO

ARisiko - Una soluzione da tavolo in realtà aumentata

Elaborato in

Computer Graphics

Relatore	Presentata da
Damiana Lazzaro	Francesco Agnoletti

SESSIONE UNICA

Anno Accademico 2018/2019

Dedico questa tesi ai miei genitori

che mi sostengono sempre

da sempre

INDICE

Introduzione

Questa tesi si pone come obiettivo la realizzazione di un sistema di gioco che migliori la praticità dei giochi da tavolo, tramite l'utilizzo delle funzionalità di Realtà Aumentata. I sistemi di Realtà Aumentata, sviluppati nel corso degli anni, hanno dimostrato come questa tecnologia, in alcune sue applicazioni, possa semplificare ed arricchire situazioni della vita reale, talvolta riuscendo ad evitare l'effetto di estraniazione dalla realtà stessa, che spesso caratterizza le applicazioni multimediali. Questa sua caratteristica rende la Realtà Aumentata una candidata ideale per la realizzazione di sistemi virtuali per il gioco da tavolo, che spesso sono concepiti sia per offrire un grado di divertimento e coinvolgimento ai giocatori, che per creare un'opportunità di socializzazione. L'idea del progetto ARisiko, perciò, nasce e si sviluppa tramite questa tecnologia, tenendo sempre presente la necessità di semplificare e migliorare la fruizione del gioco da tavolo classico, senza però ignorare o ridurre la convivialità che lo caratterizzano.

Per lo sviluppo di ARisiko sono stati utilizzati il software Blender, per la modellazione, ed il software Unity, per la gestione, tramite il plugin Vuforia, della Realtà aumentata, e per la creazione delle funzionalità di gioco.

Questa tesi è così strutturata:

- il primo capitolo riassume alcune delle tecnologie e dei sistemi che nel corso del tempo hanno portato alla diffusione della realtà aumentata nei suoi diversi ambiti di applicazione;
- il secondo capitolo descrive le motivazioni che hanno portato alla nascita del progetto ARisiko ed espone le tecnologie utilizzate per la sua realizzazione;
- il terzo capitolo descrive dettagliatamente il processo di modellazione e di sviluppo che è stato svolto per la realizzazione del progetto ARisiko;
- il quarto capitolo mostra il funzionamento del progetto ARisiko, visto dall'utente finale.

Capitolo 1 - Realtà aumentata

Introduzione

In questo capitolo saranno esposti i principali sistemi che hanno contribuito alla diffusione delle tecnologie per la Realtà Aumentata.

Per Realtà Aumentata si intende un'esperienza virtuale, in cui gli oggetti del mondo reale vengono arricchiti da informazioni acquisite e generate da un computer e restituite all'utente attraverso diverse modalità sensoriali[1].

1.1 Storia

La prima menzione storica di un'idea che comprende i concetti della Realtà Aumentata risale al 1901, quando Lyman Frank Baum ipotizza, in "The Master Key", un paio di occhiali che consentano di sovrapporre dati alla vita reale, chiamati "Character Marker"[2]; in questo specifico caso gli occhiali aiutano il protagonista a riconoscere quale sia la natura dei personaggi incontrati.
La prima applicazione di Realtà Aumentata nasce, invece, nel 1962 per mano di Morton L. Heilig, che registra il brevetto di una macchina chiamata Sensorama[3], il cui scopo era quello di immergere l'utente come protagonista di una simulazione tramite stimoli visivi, uditivi, tattili ed olfattivi[4].
Il termine "Realtà Aumentata" viene coniato solo nel 1990, quando Thomas Caudel, su richiesta di Boeing[5], per rimpiazzare i diagrammi ed i dispositivi di marcatura allora utilizzati per guidare i lavoratori nella costruzione, progetta insieme al suo collega David Mizell, HUDSET[6], un Head-Up Display indossabile che indica visivamente agli addetti ai lavori il corretto posizionamento degli oggetti richiesti. Pochi anni dopo, nel 1992, Louis Rosenberg realizza il primo sistema di Realtà Aumentata immersivo ed interattivo. Questo sistema, chiamato Virtual Fixtures, consisteva di un visore con due lenti ed un esoscheletro che l'utente doveva indossare[1]. Il sistema permetteva di creare un'esperienza immersiva perchè,

riuscendo a rielaborare le braccia dell'utente, le mostrava all'utente modificate a seconda delle preferenze d'uso. Il sistema permetteva inoltre di aggiungere informazioni e barriere simulate in sovraimpressione, aiutando l'utente nelle sue reali mansioni.

Fino al 1998 la Realtà Aumentata è stata effettivamente utilizzata solo per sistemi di tipo industriale, ma in quest'anno Sportsvision ha trasmesso la prima partita NFL arricchita dal sistema "1st & Ten"[7]. Questo sistema permetteva di inserire elementi grafici sul campo di gioco, fornendo agli spettatori riferimenti chiari ed utili per capire l'andamento delle azioni di gioco, come ad esempio la "Linea di scrimmage" oppure la "Linea del primo down". Questa è stata la prima applicazione di Realtà Aumentata di tipo non industriale ed orientata al grande pubblico.

Questo tipo di applicazione della Realtà Aumentata tramite l'elaborazione delle immagini e dei video viene accolta positivamente, infatti nel 1999 Hirokazo Kato presenta al SIGGRAPH, una conferenza annuale sulla Computer Grafica, ARToolkit[8], una libreria sviluppata per HITLab che permette di creare applicazioni in Realtà Aumentata per Windows DirectShow Video. Nel 2001 ne viene rilasciato il codice sorgente, rendendo ARToolkit open-source. ARToolkit diviene in seguito un punto di riferimento per la realizzazione di applicazioni di Realtà Aumentata, aggiungendo nel corso degli anni il supporto per lo sviluppo su più piattaforme (Linux, macOS) e la portabilità a diversi linguaggi di programmazione e piattaforme di sviluppo. In questi anni in ARToolkit vengono integrate anche alcune importanti funzionalità utili allo sviluppo di applicazioni di Realtà Aumentata, tra cui figurano il tracciamento degli oggetti ed il relativo posizionamento della camera ed il riconoscimento e tracciamento di piani naturali, cercando di eliminare la necessità di utilizzare marcatori per riconoscere l'ambiente su cui porre gli elementi virtuali.

La diffusione degli smartphones, la loro integrazione di diversi sensori ottici e fisici, e soprattutto la presenza di un Marketplace dedicato ha spostato l'attenzione degli sviluppatori di sistemi in Realtà Aumentata verso questo tipo di dispositivi ed utenti.

Nel 2008 viene infatti rilasciata la prima versione di Wikitude AR Travel Guide[1], la prima applicazione mobile di Realtà Aumentata basata sulla geolocalizzazione. Wikitude AR Travel Guide utilizzava i sensori GPS e WiFi per conoscere la posizione dell'utente e la bussola oppure l'accelerometro per riconoscerne l'orientamento. Una

volta rilevata la posizione e l'orientamento tracciava sullo schermo informazioni utili riguardanti gli elementi inquadrati.

Nel 2013 viene rilasciato da Niantic, Ingress, un gioco di Realtà Aumentata non basato sull'elaborazione delle immagini, bensì solo sulla posizione degli utenti nel mondo reale[1]. Lo scopo dei giocatori di Ingress, divisi in due fazioni, era infatti quello di esplorare e conquistare i punti di interesse della propria città, visitandoli fisicamente e svolgendo azioni di gioco da distanza ravvicinata, spesso collaborando con altri giocatori.

Il concetto alla base di Ingress viene poi riutilizzato e rielaborato da Niantic stessa nello sviluppo di Pokémon Go, probabilmente il più popolare e famoso gioco sviluppato in Realtà Aumentata, rilasciato nel 2016[1].

Per lo sviluppo di Pokémon Go, Niantic ha, infatti, riutilizzato i dati acquisiti per Ingress riguardanti i punti di interesse di tutto il globo, mantenendo un sistema di gioco simile a quello di Ingress, ma questa volta arricchendolo con azioni basate sull'elaborazione delle immagini.

Con la diffusione degli smartphone anche i grandi social network hanno iniziato a concentrarsi sulla creazione di applicazioni ottimizzate per l'utilizzo su dispositivi mobile; allo stesso modo, sono nati social network ed applicazioni volte principalmente, se non esclusivamente, ad essere utilizzate tramite questo tipo di dispositivi.

Nel 2011 viene pubblicata la prima versione di Snapchat, un'applicazione di messaggistica istantanea, la cui principale peculiarità è quella di poter limitare la visualizzazione nel tempo dei dati inviati e di poterli distruggere automaticamente alla loro visualizzazione. Tuttavia Snapchat merita una menzione in quanto un'altra sua peculiarità è quello di poter scattare foto e registrare video applicandogli in sovraimpressione filtri ed animazioni in modo accurato, tramite l'elaborazione delle immagini ed il riconoscimento del volto o di altri elementi inquadrati. Tale funzionalità verrà poi ripresa e sviluppata in varie forme in molte altre applicazioni, come ad esempio Instagram ed anche alcune applicazioni native di controllo della fotocamera, sviluppate dalle stesse case produttrici di smartphone.

Realtà Aumentata in Futuro

I software e gli strumenti utilizzati nello sviluppo di applicazioni di Realtà Aumentata si svilupperanno nei prossimi anni cercando di semplificare e migliorare la percezione reale degli elementi che vengono aggiunti virtualmente alla scena, cercando perciò di acquisire più elementi possibile dall'immagine ricevuta e cercando di replicarne l'ambiente per simularne i comportamenti sugli oggetti virtuali e di migliorarne la renderizzazione per renderli fotorealistici, aumentando così il grado di immersione ed integrazione dei sistemi di Realtà Aumentata.
Parallelamente, verranno diffusi ed utilizzati metodi utili ad incrementare l'interattività delle applicazioni, ad esempio utilizzando il riconoscimento dei movimenti per eseguire azioni specifiche.

1.2 Campi di applicazione

Dal 1960 la Realtà Aumentata ha suscitato sempre più interesse nelle aziende. Fino agli anni '90 i sistemi sviluppati avevano uno scopo dimostrativo delle potenzialità della Realtà Aumentata stessa. Molti dei sistemi sviluppati in questi anni avevano come obiettivo l'arricchimento della scena inquadrata con informazioni utili all'utente; un esempio importante di questo utilizzo è il lavoro di ricerca svolto da Gavan Lintern per dimostrare l'utilità di un HUD(Head Up Display) nello studio delle tecniche di volo[1].

Industriale

Durante gli anni '90 iniziano ad apparire i primi strumenti basati sulla Realtà Aumentata volti ad aiutare gli operatori nello svolgimento delle proprie mansioni industriali.
Virtual Fixtures e HUDSET ne sono un ottimo esempio:

-Virtual Fixtures in termini di sicurezza, in quanto la possibilità di aggiungere barriere visive alla scena inquadrata permetteva di riconoscere in tempo reale le zone di lavoro su cui non bisognava mettere mano;
-HUDSET invece aiutava l'utente nella propria mansione perchè dava un'indicazione precisa riguardo al posizionamento corretto degli oggetti con cui si stava lavorando in fase di assemblaggio o disassemblaggio.

Il concetto alla base di HUDSET viene ripreso più volte negli anni successivi, fino ad arrivare, nel 2013, a MARTA[9], un'app sviluppata da Volkswagen e Metaio che consente ai meccanici di riconoscere i componenti richiesti e la loro posizione precisa; MARTA fornisce inoltre tutte le informazioni necessarie al montaggio ed allo smontaggio dei componenti, come ad esempio gli strumenti da utilizzare ed i passi da effettuare nello smontaggio degli stessi.

Informazione

L'utilizzo a scopo informativo dei sistemi di Realtà Aumentata è molto diffuso: nel corso degli anni sono stati infatti sviluppati moltissimi strumenti a supporto dell'utente ai fini di fornirgli in modo semplice ed immediato le informazioni necessarie. Un esempio lampante di questo ambito è l'Head-Up Display ad oggi disponibile in molti modelli di automobili. Questo sistema consiste nella proiezione sul parabrezza delle informazioni utili alla guida del proprio mezzo, come ad esempio la velocità attuale oppure le indicazioni del navigatore.
Questo particolare sistema di navigazione in Realtà Aumentata è recentemente stato reso disponibile su Google Maps per gli smartphone Android e iOS con il nome Live View, in modo da poterla utilizzare anche negli spostamenti a piedi; in questo particolare caso, oltre ad avere in sovraimpressione le indicazioni riguardanti il proprio percorso, l'app segnalerà gli ostacoli e gli elementi a cui fare attenzione durante il proprio percorso, come ad esempio le auto parcheggiate oppure la segnaletica stradale.
A questo sistema di informazione durante i propri movimenti a piedi si affiancano altri sistemi di navigazione che indicano in che direzione si trovano i punti di interesse più vicini, come ad esempio i monumenti o le fermate dei trasporti pubblici.

Educazione

In ambito educativo, sono presenti alcune applicazioni per smartphone che hanno l'obiettivo di aiutare nella comprensione e nell'applicazione dei concetti. Un esempio è Star Walk, che aiuta a trovare le costellazioni fornendo indicazioni precise sulla loro posizione in base alla posizione ed all'orientamento dell'utente, fornendo anche consigli utili per riuscire a trovarle anche senza aiuti esterni.
Altri esempi importanti in questo ambito sono:
-Anatomy 4D, che consente, inquadrando un marker apposito, di vedere la posizione ed il funzionamento degli organi e dei sistemi del corpo.
-Elements 4D, che, utilizzando diversi marcatori, permette di combinare virtualmente uno o più elementi chimici e visualizzarne le informazioni principali per capire e vedere che tipo di reazione chimica avviene tra di loro.
Altri sistemi educativi di Realtà Aumentata sono utilizzati prevalentemente nei siti archeologici per dare una rappresentazione visiva degli stessi nel corso della loro storia. Questi sistemi si basano sulla ricostruzione dei siti in 3D direttamente nella scena inquadrata, applicando un modello virtuale costruito appositamente e mantenendone la prospettiva.

Simulazione ed anteprima

Moltissime aziende stanno sviluppando sistemi di Realtà Aumentata per permettere ai clienti di provare gli oggetti prima di acquistarli, simulandone la presenza nel mondo reale, così da poterne verificare, ad esempio, l'occupazione dello spazio in un ambiente, come consente di fare l'App IKEA Place[10], oppure applicando più in profondità questo concetto si può arrivare ad applicare ad una stanza la simulazione di un intero progetto di arredo come permesso da Homestyler[11].
Questo metodo di visualizzazione in anteprima di un oggetto virtuale è stato esteso a molti altri ambiti. Infatti è possibile tramite le App di Realtà Aumentata, spesso fornite dagli stessi produttori dei prodotti, simulare l'utilizzo di molti oggetti di uso comune, come ad esempio vestiti o altri prodotti di abbigliamento, gioielli ed anche la

visualizzazione di trucchi, tatuaggi ed acconciature applicate direttamente sul proprio corpo.

Intrattenimento

Nell'ambito dell'intrattenimento la Realtà Aumentata è molto diffusa, infatti sono sempre più presenti App per Android e iOS che sfruttano la fotocamera e consentono di visualizzare e personalizzare oggetti e personaggi in 3D, oppure di interagire con gli stessi, fino ad arrivare a veri e propri giochi, alcuni basati sulle azioni dell'utente nella realtà, come ad esempio Pokémon Go oppure Zombies Go; altri invece basati sulla vista tridimensionale di oggetti interattivi costruiti su un piano inquadrato oppure basati sul riconoscimento di un marker.

1.3 Tecnologie Hardware e Software

1.3.1 Hardware

A partire dagli anni '60 sono state sviluppate molte tecnologie hardware che permettessero di sovrapporre visivamente elementi virtuali alla realtà percepita dagli utenti.
Questi sistemi possono essere basati sulla sovrapposizione di elementi alla realtà vista dall'utente, talvolta senza effettuare un'elaborazione dell'immagine; altri sistemi invece si basano sull'elaborazione dell'immagine inquadrata, restituendo all'utente il risultato dell'elaborazione.

Sovrapposizione

Tra i sistemi hardware basati sulla sovrapposizione delle immagini troviamo l'Head-Mounted Display(HMD) oppure l'Head-Up display(HUD).
La sovraimpressione in questi sistemi è realizzata proiettando gli elementi virtuali richiesti su una superficie trasparente posizionata davanti agli occhi degli utenti. I sistemi più avanzati si basano su una camera che viene utilizzata per elaborare la corretta posizione degli elementi virtuali da proiettare, oppure per offrire funzionalità legate al riconoscimento dell'immagine, come nel caso della traduzione istantanea.

Il primo Head-Mounted Display è stato realizzato da Ivan Sutherland nel 1968[1], ma il concetto alla base di questo dispositivo è in continuo sviluppo.
Negli ultimi anni, infatti, sono stati sviluppati e commercializzati dispositivi HMD di diversi tipi, anche per l'utilizzo quotidiano.
Un esempio importante da questo punto di vista è quello dei Google Glass.
I Google Glass[12] erano infatti pensati anche per l'utilizzo quotidiano; permettevano infatti, tramite l'integrazione con i servizi Google e con app di terze parti, di ricevere indicazioni sulla navigazione, notifiche riguardanti i propri social network oppure le chiamate o le email ricevute, fino a consentire una traduzione istantanea del testo che si stava visualizzando.
Un altro esempio di HMD è Microsoft Hololens[13]. Hololens permette di visualizzare, in modo similare a Google Glass, elementi in sovraimpressione alla realtà inquadrata, aggiungendo però un'importante funzionalità. Hololens permette infatti di riconoscere i movimenti della testa dell'utente, mettendo così a disposizione un ambiente virtuale a 360 gradi e tridimensionale; e consente di rilevare la posizione delle mani, mettendo così a disposizione un sistema di movimenti(o Air Gestures) che permette di interagire con gli elementi virtuali proiettati davanti ai propri occhi.

Elaborazione

I sistemi di elaborazione di Realtà Aumentata si basano invece sull'acquisizione delle immagini, non necessariamente in tempo reale e non necessariamente tramite una camera, sulla successiva rielaborazione per applicarne gli elementi virtuali richiesti, ed infine sulla restituzione rielaborata all'utente.
Questi sistemi necessitano perciò di un sistema di acquisizione dell'immagine, come ad esempio una camera, e di un display che consenta all'utente di visualizzare l'immagine rielaborata.
La nascita del primo sistema di questo tipo risale al 1993, quando Mike Abernathy ha segnalato l'utilizzo di un sistema che sovrappone le traiettorie dei satelliti alle immagini generate da un telescopio[1].
Questo sistema di rielaborazione delle immagini in tempo reale è adesso ampiamente diffuso grazie alla sempre crescente potenza di calcolo degli

smartphone ed alla loro completezza per quanto riguarda i requisiti fondamentali all'utilizzo della Realtà Aumentata.

1.3.2 Software

Nello sviluppo di un sistema di Realtà Aumentata è importante l'utilizzo di software che consentano di elaborare e manipolare le immagini ed i flussi di dati video.

OpenCV

Una delle librerie più diffuse per l'elaborazione e la manipolazione delle immagini è OpenCV[14], una libreria open-source, sviluppata a partire dal 1999 e disponibile in una moltitudine di linguaggi di programmazione, che al giorno d'oggi integra funzionalità che consentono, ad esempio, di riconoscere pattern nelle immagini, oppure movimenti particolari in un video e di manipolarle per produrre immagini modificate o arricchite da elementi aggiuntivi.

ARCore

ARCore[15] è un Software Development Kit(SDK) sviluppato da Google e disponibile per lo sviluppo di applicazioni Android ed iOS e supportato da Unity ed Unreal Engine.
ARCore offre funzionalità utili alla realizzazione di applicazioni di Realtà Aumentata, mettendo a disposizione degli sviluppatori funzionalità avanzate come il riconoscimento delle superfici e delle sue reali dimensioni, oppure la possibilità di stimare le condizioni di illuminazione della scena inquadrata.

ARKit

ARKit[16] è l'SDK fornito da Apple per lo sviluppo di applicazioni in Realtà Aumentata per iOS e macOS.
Permette di catturare i movimenti del corpo e di utilizzarli in modo interattivo nelle App di Realtà Aumentata; inoltre ha funzionalità di profondità, permettendo così agli elementi virtuali di interagire con gli elementi reali in base alla loro posizione reale.

Capitolo 2 - Descrizione del progetto e tecnologie utilizzate

Questo capitolo descrive gli obiettivi e le funzionalità del progetto ARisiko, nonchè le tecnologie e le metodologie utilizzate per la sua realizzazione.

ARisiko

ARisiko è un'applicazione il cui obiettivo è quello di simulare, tramite la realtà aumentata, il gioco da tavolo Risiko![17], permettendone così la giocabilità in modo più comodo ed immediato.

Risiko!

Risiko! è un gioco da tavolo multigiocatore pubblicato per la prima volta nel 1968 da Giochiclub.
Risiko! è un gioco strategico a turni, l'interazione tra i giocatori avviene pertanto solo durante il proprio turno, fatta eccezione quando vengono interpellati a difendersi dagli altri giocatori.
Si gioca su una plancia in cui è raffigurato il mappamondo, suddiviso in 42 territori, che sono raggruppati in 6 continenti.
All'inizio della partita i territori vengono equamente spartiti tra i giocatori, ognuno dei quali riceverà anche un obiettivo, che può variare dalla conquista di un determinato numero di territori o continenti, all'eliminazione di uno o più giocatori dalla partita, che i giocatori dovranno mantenere segreto.
Ogni giocatore ha poi un numero determinato di armate che dovrà piazzare a suo piacimento in ognuno dei territori che gli sono stati assegnati.
Una volta terminata la fase di popolazione dei territori inizia il gioco vero e proprio.
A rotazione, ogni giocatore svolgerà il suo turno di gioco, in cui potrà, in quest'ordine:

- piazzare nuove armate nei territori che possiede;

- attaccare territori confinanti con quelli da lui posseduti;
- spostare le truppe da un territorio in suo possesso ad un altro contiguo.

L'azione di attacco può essere effettuata solo da territori in cui sono presenti almeno due armate, così come l'azione di spostamento di armate da un territorio ad un altro. L'attacco può avvenire, se il territorio lo permette, utilizzando fino a tre armate contemporaneamente, la riuscita dell'attacco viene decisa dal singolo punteggio del lancio da uno a tre dadi(in base alle armate utilizzate) da parte del giocatore attaccante; nel frattempo, il giocatore difensore, proprietario del territorio sotto attacco, potrà lanciare fino a tre dadi(in base alle armate presenti sul proprio territorio) per cercare di difendersi dall'attacco. I singoli risultati vengono perciò confrontati fra di loro in ordine decrescente per decidere quale giocatore ha vinto lo scontro. (Possono essere presenti vittorie parziali, ad esempio entrambi i giocatori potrebbero perdere una sola armata). Quando il territorio difensore resta senza armate viene conquistato dal giocatore attaccante, che dovrà spostare una o più armate dal territorio attaccante al territorio appena conquistato.

Ad ogni turno, se un giocatore conquista un nuovo territorio, riceve una "Carta Territorio" che può essere di diversi tipi. Una volta ricevute tre carte, le potrà usare per poter piazzare più armate all'inizio del proprio turno. Nel gioco ogni Carta Territorio rappresenta un Territorio esistente e nel caso un giocatore utilizzi una Carta Territorio che rappresenta un Territorio da lui posseduto, avrà un ulteriore vantaggio e potrà piazzare ulteriori armate.
La partita termina quando uno dei giocatori porta a termine il proprio obiettivo, conquistando perciò il numero o il tipo di Territori richiesti, oppure eliminando le armate avversarie a lui assegnate.

Perchè ARisiko

L'obiettivo di ARisiko è quello di permettere di fruire del gioco da tavolo Risiko! in modo più semplice e meno dispendioso in termini di spazio e di tempo.
ARisiko dà perciò la possibilità di visualizzare la plancia di gioco come elemento virtuale, inquadrando un marker posizionato al centro del proprio tavolo; consente di interagire con gli elementi visualizzati sullo schermo del proprio smartphone e di svolgere così le azioni di gioco, come ad esempio il posizionamento delle armate durante il proprio turno e le azioni di attacco e conquista dei territori avversari.
ARisiko consente, inoltre, di tenere traccia dei dati di gioco, come ad esempio la quantità di armate presenti sui territori, facilitando la gestione degli elementi di gioco da parte dei giocatori.
Eliminando la necessità di utilizzare gli oggetti fisici del gioco nella loro interezza, si ha così la possibilità di giocare semplicemente tramite il proprio smartphone ed un marker, riducendo così le dimensioni degli oggetti da utilizzare a quelle del singolo marker e del proprio smartphone ed ottimizzando i tempi di inizializzazione della partita
L'utilizzo di un marker aiuta inoltre a mantenere l'aspetto sociale del gioco da tavolo, in quanto consente di condividere la scena di gioco; essendo inoltre Risiko! un gioco a turni, è possibile partecipare attivamente alla partita solo durante il proprio turno oppure quando si è chiamati a difendere un proprio territorio. In questo modo i giocatori possono partecipare alla partita senza dover concentrare tutta l'attenzione su quello che succede nel gioco, lasciando così la possibilità di socializzare con gli altri giocatori presenti.

2.2 Tecnologie utilizzate

Il progetto ARisiko è stato realizzato utilizzando il motore grafico Unity[18], le funzionalità ed il comportamento sono stati sviluppati in C#, le funzionalità di riconoscimento del marker e di relativo posizionamento della camera sono integrate nel plugin per Unity del Framework Vuforia[19].
La modellazione 3D è stata realizzata invece tramite il software Blender[20].

Unity

Unity è un motore grafico multi-piattaforma sviluppato da Unity Technologies in C# e C++. Rilasciato inizialmente nel 2005 esclusivamente per Mac OS X, Unity nel corso degli anni è stato migliorato, integrando sempre maggiori funzionalità ed estendendo il supporto ai più diffusi sistemi operativi; attualmente infatti Unity è disponibile sia per i sistemi operativi UNIX-Like che per i sistemi Windows; Unity permette inoltre di produrre applicazioni sia per i maggiori sistemi operativi per Desktop Computer, sia per i maggiori sistemi operativi per dispositivi mobile, cioè per Smartphone e Tablet Android e iOS; Unity permette inoltre di sviluppare applicazioni 3D per il web, grazie all'utilizzo dapprima di un plugin per web browser chiamato Unity Web Player, poi soppiantato dal supporto a WebGL.

La presenza inoltre di un marketplace dedicato ha permesso ad enti di terze parti di mettere a disposizione plugin ed SDK per lo sviluppo di applicazioni per la propria piattaforma utilizzando Unity. Molti di questi plugin ed SDK sono poi stati integrati nel motore grafico, portando Unity a supportare ufficialmente oltre 25 piattaforme diverse, tra cui si trovano i maggiori sistemi di intrattenimento domestico, come ad esempio Sony PS4 o XBOX ONE, le più diffuse console portatili, come ad esempio Nintendo Switch, ed anche i più famosi sistemi di Realtà Virtuale, come Oculus Rift e PlayStation VR.

Attualmente sul marketplace di Unity si possono trovare anche SDK per lo sviluppo di applicazioni di Realtà Aumentata, tra cui Apple ARKit, Google ARCore e Vuforia.

Unity offre funzionalità complete di modellazione ed animazione di oggetti e scene 2D e 3D, mette a disposizione funzionalità di rete e funzionalità di Scripting utilizzando il linguaggio C#.

Utilizza una pipeline di rendering che fornisce il supporto per tecniche di renderizzazione come l'occlusione ambientale, le ombre dinamiche e la mappatura dei riflessi. È inoltre attualmente in fase di sviluppo il supporto al Ray-Tracing in tempo reale, integrato nella High Definition Rendering Pipeline[18].

Vuforia

Vuforia è un SDK sviluppato inizialmente da Qualcomm ed acquisito nel 2015 da PTC. È attualmente disponibile per lo sviluppo su Android, iOS, Universal Windows Platform e Unity e consente di creare applicazioni di Realtà Aumentata per i dispositivi Android, iOS ed UWP, tra cui merita menzione Hololens.

Vuforia utilizza funzioni di Computer Vision per consentire il riconoscimento di superfici piane, di immagini specifiche e di oggetti tridimensionali.

La possibilità di utilizzare immagini specifiche, da utilizzare come marker, consente a Vuforia di piazzare oggetti virtuali nella scena inquadrata e di cambiarne l'orientamento, in base alla prospettiva rilevata, su 6 gradi di libertà, dando così una corrispondenza reale agli utenti, facendo sembrare l'oggetto inquadrato virtuale parte della realtà inquadrata. Questo grado di immersione è ulteriormente incrementato dalla capacità di Vuforia di riconoscere i marker anche parzialmente, dando così la possibilità di aggiungere profondità agli oggetti virtuali. Inoltre, la sovrapposizione di oggetti reali ai marker utilizzati, consente a Vuforia di rilevare i movimenti che avvengono rispetto alla scena virtuale, dando la possibilità così di agire sulla scena virtuale utilizzando oggetti reali[21].

Blender

Blender è un software di modellazione ed animazione 3D open-source, pubblicato e distribuito sotto licenza General Public License(GPL) da Blender Foundation nel 1994.

È disponibile per i sistemi operativi Windows, macOS, Linux e FreeBSD ed offre una moltitudine di funzionalità avanzate per semplificare e perfezionare la realizzazione di modelli 3D, animazioni o rendering 3D fotorealistici e simulazioni fisiche.

Blender permette di gestire una buona quantità di primitive geometriche diverse, tra cui le maglie poligonali, le curve di Bézier o NURBS e di scolpirli utilizzando diverse modalità; mette a disposizione strumenti per simulare le proprietà ed i movimenti di oggetti deformabili, tra cui la rilevazione delle collisioni e la simulazione

fluido-dinamica; permette di realizzare effetti particellari e consente di utilizzare scripts in Python ed altri sistemi integrati, come ad esempio i Modificatori, per automatizzare la creazione dei modelli e delle animazioni oppure per implementare logiche di gioco o strumenti personalizzati[22].

Capitolo 3 - Implementazione del progetto

Questo capitolo descrive dettagliatamente il processo di sviluppo che ha portato alla realizzazione del progetto ARisiko e di tutti i suoi componenti.

3.1 Modellazione 3D

La realizzazione dei modelli è stata effettuata utilizzando il software Blender 2.81.
Il modello 3D “Tank” è una rappresentazione 3D di un carro armato; è costituito da un oggetto chiamato “TankBase”, che ne rappresenta il telaio; a questo oggetto sono poi stati collegati gli elementi mobili del carro armato.

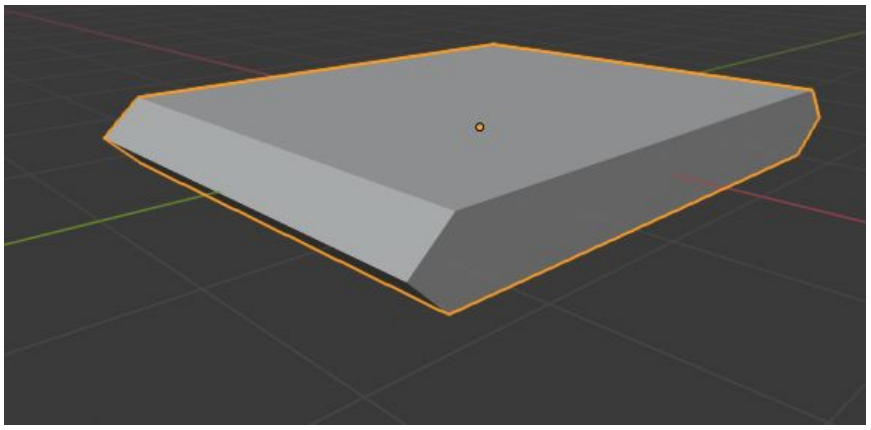

Figura 1: Componente “TankBase”

Il telaio del carro è stato realizzato partendo da un parallelepipedo al quale sono stati suddivisi e poi uniti centralmente gli archi nella parte frontale e posteriore del carro, in modo da conferirgli l’aspetto concavo che contraddistingue i veicoli di questo tipo.
L’oggetto TankBase rappresenta il centro di gravità dell’oggetto 3D “Tank”.
Sopra all’oggetto “TankBase” è stato posizionato l’oggetto “TankCabin”, che rappresenta la cabina di controllo del carro armato.

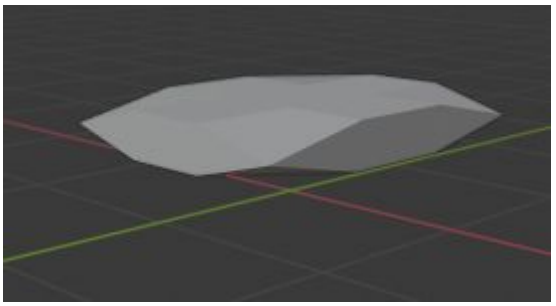

Figura 2: Componente “TankCabin”

L’oggetto “TankCabin” è collocato appena sopra l’oggetto “TankBase” ed è stato realizzato anch’esso a partire da un parallelepipedo. In questo caso sono stati suddivisi in tre parti gli archi che lo compongono e sono state unite verticalmente le coppie di vertici che ne rappresentano gli spigoli inferiori e superiori.
L’oggetto “TankCabin” possiede un Pivot Point posizionato nel suo centro di gravità che gli consente di ruotare orizzontalmente intorno all’asse Z.

All’oggetto “TankCabin” è stato collegato l’oggetto “Cannon”, che rappresenta il cannone del carro armato.

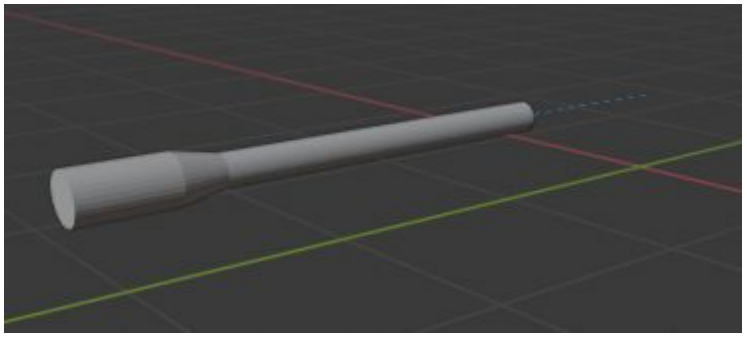

Figura 3: Componente “Cannon”

Il componente “Cannon” è stato modellato a partire da un cilindro, che è stato scalato lungo l’asse Y e successivamente suddiviso in 3 parti:

- la prima parte è rimasta immutata e rappresenta la canna del cannone; copre il 60% della lunghezza complessiva del componente;
- la seconda parte è stata scalata lungo il suo diametro per simulare lo “strozzatore”, elemento tipico dei cannoni installati sui carri armati, e rappresenta il 20% delle dimensioni complessive dell’elemento “Cannon”;

- la terza parte rappresenta il 10% delle dimensioni dell'elemento "Cannon", è costituito dalla sezione di un cono e fa da collegamento per gli altri due elementi.

Il componente "Cannon" è stato collegato al componente "TankCabin" tramite un vincolo di tipo "Child-Of". In questo modo, le trasformazioni di rotazione applicate all'elemento "TankCabin" avranno effetto anche sull'elemento "Cannon", che avrà come riferimento lo stesso Pivot Point dell'oggetto "TankCabin".

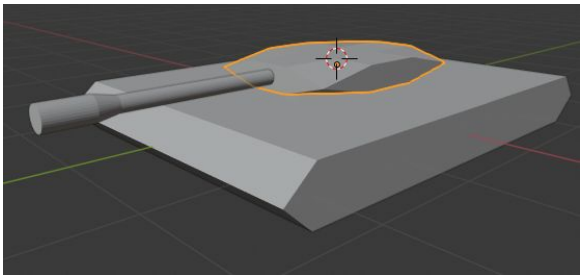

Figura 4.1: Posizione predefinita

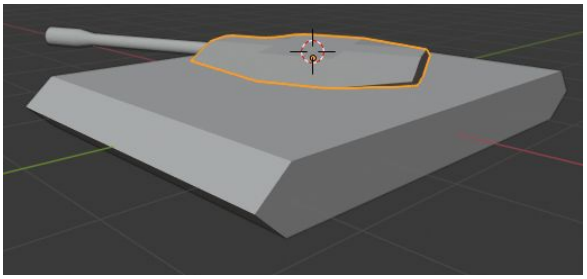

Figura 4.1: Rotazione di "TankCabin"

Il modello "Tank" è poi costituito da una collezione di 16 cilindri chiamati "Tyres", quattro dei quali sono posizionati agli angoli del carro ed hanno un diametro leggermente minore rispetto agli altri.

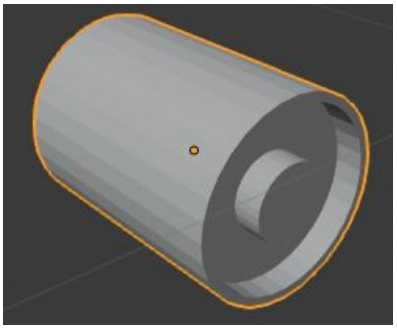

Figura 5: Elemento "Tyre"

L'elemento "Tyre" è stato creato partendo da un cilindro, nella cui facciata rivolta verso l'esterno del carro è stata poi inserita una facciata leggermente rimpicciolita che poi è stata estrusa verso l'interno del cilindro, creando così il contorno della ruota. Questa facciata è stata poi ulteriormente ridimensionata fino a raggiungere le dimensioni del cilindro interno che rappresenta il mozzo della ruota, ed è stata

estrusa verso l'esterno del cilindro, portando così al risultato finale con il mozzo in bella vista.
L'elemento "Tyre" è stato poi ripetutamente duplicato e posizionato lungo l'asse Y fino al raggiungimento del risultato desiderato.

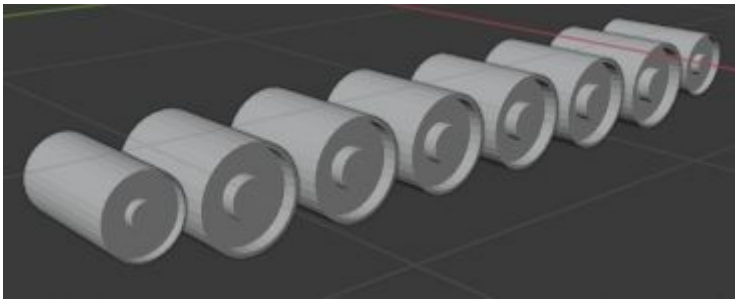

Figura 6: Collezione di elementi "Tyres"

Infine, il carro è stato dotato di cingoli, che sono costituiti da una collezione di elementi "TankTrack".

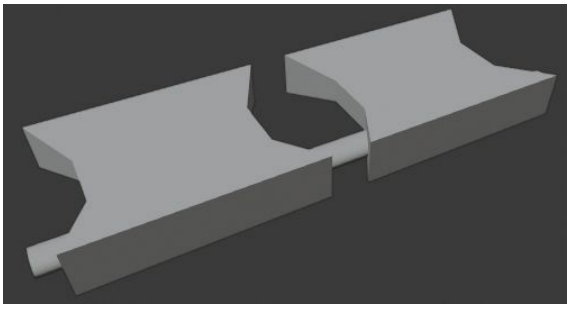

Figura 7.1: Elemento "TankTrack"

L'elemento "TankTrack" è costituito da tre elementi, un parallelepipedo, i cui lati esterni sono stati suddivisi in tre parti e smussati al centro, che poi è stato duplicato, ed un cilindro che li collega. I vertici di questi tre elementi di base che lo compongono sono poi stati uniti per formare un elemento singolo.
Una volta creato l'elemento di base per la creazione dei cingoli, è stata predisposta una curva di Bézier, il cui percorso segue il perimetro degli elementi "Tyres" precedentemente posizionati.

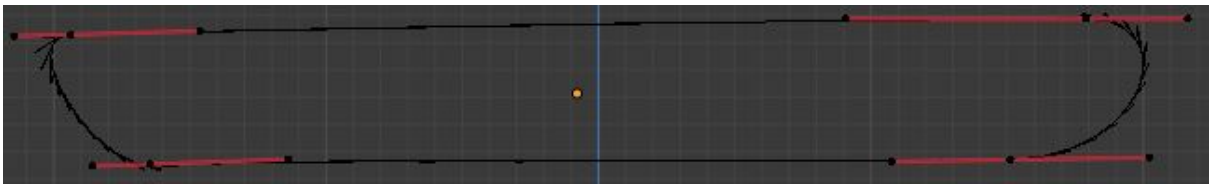

Figura 7.2: Curva di Bézier con i relativi punti di controllo

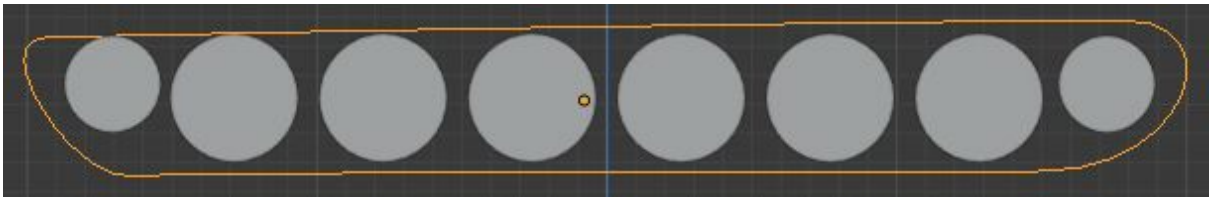

Figura 7.3: Curva di Bézier posizionata attorno agli elementi “Tyres”

Successivamente è stata costituita la collezione “TankTrack”, formata dalla Curva di Bézier e dall’elemento “TankTrack”, a cui sono stati applicati due modificatori:

- un modificatore Array di tipo “Fixed Count”, che ha consentito di creare molteplici copie dell’elemento “TankTrack” e di posizionarle automaticamente lungo l’asse Y con un valore offset relativo di 0.7, nonchè di unirle facendo coincidere l’ultima copia dell’elemento con quello originario;
- un modificatore Curva, che ha consentito al modificatore Array di posizionare le copie lungo la curva di Bézier precedentemente costruita.

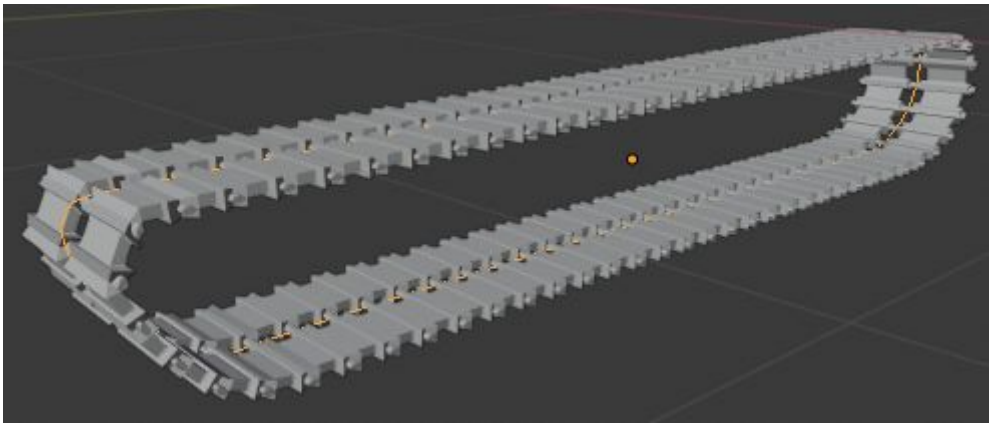

Figura 7.4: La collezione “TankTrack” completa con la curva di Bézier(Fig. 7.2) evidenziata

Il modello 3D completo del carro è stato poi esportato in formato .fbx per poter essere utilizzato con Unity.

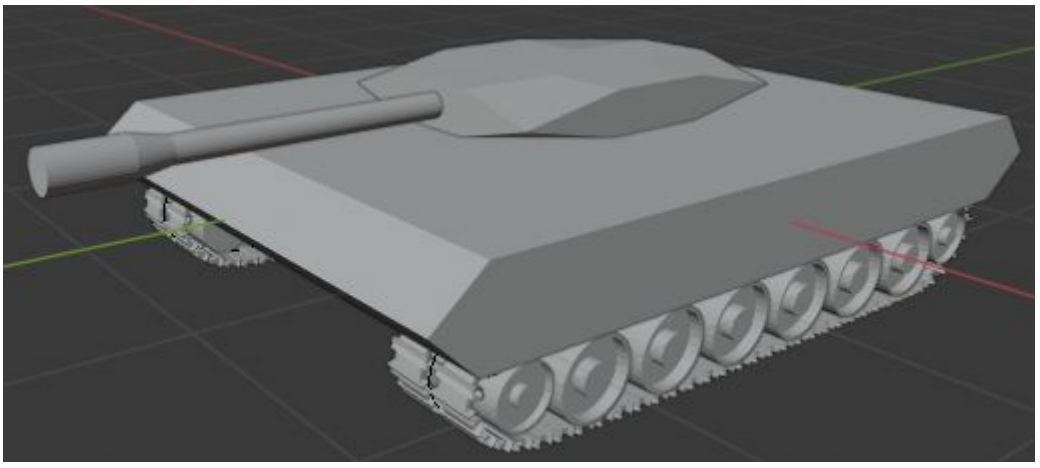

Figura 7.5: Il modello “Tank” completo

3.2 Motore di gioco

Per la realizzazione delle funzionalità di ARisiko è stato utilizzato il motore grafico Unity nella versione 2019.3.0f5, servendosi del plugin Vuforia nella versione 8.5.9 per la realizzazione delle tecniche di Realtà Aumentata.

3.2.1 Vuforia

Le funzionalità di Computer Vision integrate in Vuforia hanno consentito il riconoscimento di un marker ed il posizionamento degli elementi virtuali relativamente alla posizione della camera rispetto alla scena inquadrata.
Il plugin Vuforia fornisce, per il raggiungimento di tale scopo, un oggetto ARCamera che integra uno Script C# chiamato “Vuforia Behaviour”.
Questo Script contiene due campi, chiamati “World Center” e “World Center Mode”.
Il campo “World Center” consente di scegliere l’oggetto della scena che si intende utilizzare come centro del mondo, pertanto quell’oggetto, nella propria scena virtuale, sarà posizionato alle coordinate (0, 0, 0).

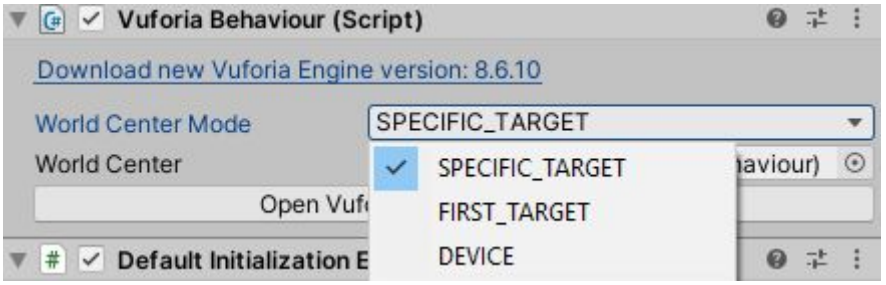

Figura 8.1: Script Vuforia Behaviour

Il campo “World Center Mode”, invece, permette di decidere in che modo la propria scena virtuale si sposterà nell’ambiente inquadrato. Quest’opzione può essere impostata in tre modalità differenti:

- Modalità SPECIFIC_TARGET; in questo caso la scena virtuale viene posizionata relativamente al marker inquadrato, l’oggetto virtuale inserito nel campo “World Center” resta perciò sempre al centro del marker, variando così la prospettiva da cui lo si può vedere muovendosi attorno al marker stesso.
- Modalità FIRST_TARGET; in questo caso è possibile utilizzare più marker, e la scena virtuale si comporterà in modo simile a quello della modalità SPECIFIC_TARGET, ma resterà ancorata al primo marker riconosciuto.
- Modalità DEVICE; in questa modalità la scena virtuale viene renderizzata attorno al proprio dispositivo, la Camera rimane perciò ancorata alla posizione (0, 0, 0) del mondo, in fase di sviluppo bisogna perciò tenere conto di queste caratteristiche nel posizionamento degli oggetti.

La modalità scelta per l’ancoraggio degli oggetti virtuali del progetto ARisiko è la modalità SPECIFIC_TARGET, in quanto è stata ritenuta la più appropriata per l’utilizzo di un marker singolo che potesse essere rilevato da angolazioni diverse.

Nel campo “World Center” è stato invece inserito l’oggetto “ImageTarget”.

L’oggetto “ImageTarget” contiene un componente Script che fa parte dell’SDK Vuforia, chiamato “Image Target Behaviour”, che consente di scegliere una o più immagini da utilizzare come Marker, nonchè il percorso in cui le immagini sono salvate. L’immagine scelta come Marker per il progetto ARisiko è l’immagine “Astronaut_scaled”, che fa parte dei marker predefiniti forniti da Vuforia.

Figura 8.2: Immagine “Astronaut_scaled”

Una volta aggiunto creato il riconoscimento del marker ed impostato come centro del mondo, l’oggetto “ImageTarget” è stato utilizzato come oggetto padre di tutta la scena di gioco; tutti gli oggetti della scena, pertanto, sono stati inseriti come figli dell’elemento “ImageTarget”, consentendo in questo modo di posizionarli nella scena relativamente al centro del mondo renderizzato.

3.2.2 Unity

3.2.2.1 Struttura della scena di gioco

La scena di gioco “GameScene” è composta da quattro elementi principali:

- “ARCamera”, che è la camera di gioco che gestisce il riconoscimento del marker ed il suo posizionamento prospettico;
- “ImageTarget”, che è l’oggetto che indica all’oggetto ARCamera quali immagini riconoscere ed in questo caso rappresenta il centro del mondo virtuale;
- “Directional Light”, è una sorgente luminosa direzionale piazzata perpendicolarmente ad “ImageTarget” e fornisce un’illuminazione di base alla scena ai fini della renderizzazione;
- “StateManagerOBJ”, è un oggetto virtuale che ha come unico componente lo Script C# “StateManager”.

Figura 9.1: Gerarchia della scena di gioco

L'oggetto "ImageTarget" contiene tutti gli elementi che saranno visualizzati a schermo. L'oggetto "Plane" è un piano posizionato al centro dell'elemento "ImageTarget" ed appena sopra, così da evitare sovrapposizioni quando viene inquadrato il Marker. Esso è composto da un componente Mesh Renderer a cui è stata applicata una texture, chiamata "RisikoMap" che rappresenta la plancia di gioco di ARisiko.

Figura 9.2: Texture RisikoMap

La collezione “Territories” contiene invece 41 oggetti figli, rappresentati graficamente da cubi, che rappresentano i singoli territori in cui è suddivisa la plancia di gioco e raggruppati, per comodità di sviluppo, in sei collezioni che rappresentano i continenti di cui fanno parte.

Figura 9.3: La plancia di ARisiko con i cubi posizionati

Infine, ogni oggetto territorio ha come figlio un oggetto “Tank”, che è costituito dal modello 3D omonimo precedentemente modellato tramite Blender.

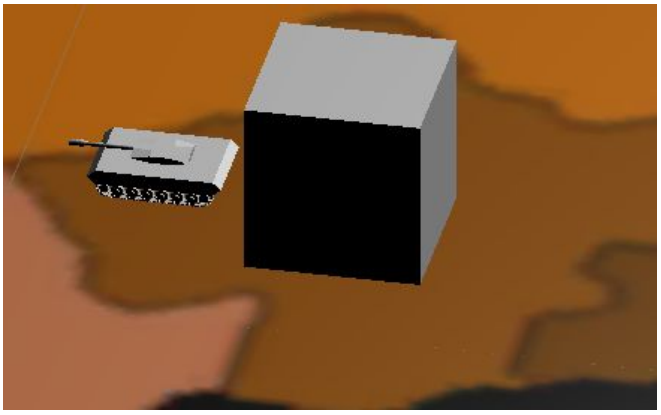

Figura 9.4: Territorio con oggetto figlio “Tank”

3.2.2.2 Implementazione delle funzionalità di gioco

Player.cs

Le informazioni di base dei giocatori vengono memorizzate in istanze della classe “Player”. Questa classe è composta da due campi:

- **string** playerName, che viene utilizzata per memorizzare il nome del giocatore;
- **Color** playerColor, che viene utilizzata per evidenziare i Territori appartenenti al giocatore.

```
public class Player : MonoBehaviour
{
   private string playerName;
   public string PlayerName { get => playerName; set => playerName = value; }

   private Color playerColor;
   public Color PlayerColor { get => playerColor; set => playerColor = value; }

   public Player(string name, Color color)
   {
          this.playerName = name;
          this.playerColor = color;
   }
}
```

Classe “Player.cs”

Implementazione del Grafo

La connessione tra i territori è stata costruita implementando un Grafo Pesato Non orientato tramite due classi: “Territory.cs” e “Neighbor.cs”.

Territory.cs

La classe “Territory.cs” rappresenta il singolo nodo del Grafo ed è stata inserita all’interno della scena di gioco come componente di ogni oggetto “Territorio”.

```
public class Territory : MonoBehaviour {
    //Current territory's proprietary
    private Player proprietary;
    public Player Proprietary { get => proprietary; set => proprietary = value; }

    //The continent which this territory is part of
    private string continent = "No";
    public string Continent { get => continent; set => continent = value; }

    //The name of the territory object
    private string territoryname;
    public string Territoryname { get => territoryname; set => territoryname = value; }

    //Keeps count of the tanks active in the territory
    private int tankCount = 0;
    public int TankCount { get => tankCount; set => tankCount = value; }

    public List<Neighbor> neighbors;
    public Territory() {
            TankCount = 3;
            neighbors = new List<Neighbor>();
    }
    // Start is called before the first frame update
    void Start() {
            new Territory();
            this.name = transform.name;
            this.continent = transform.parent.name;
    }
}
```

Classe “Territory.cs”

La classe “Territory.cs” è utilizzata come componente di ogni oggetto Territorio e viene utilizzata per mantenerne lo stato, essa è infatti costituita da:

- **string** territoryname: in questo campo viene salvato il nome dell’oggetto di cui fa parte;
- **string** continent: in questo campo viene salvato il continente di cui il territorio è figlio;
- **Player** proprietary: contiene un riferimento all’oggetto Player proprietario del territorio;
- **int** tankCount: tiene traccia del numero di armate presenti sul territorio.

La classe Territory ha inoltre una Lista di oggetti di tipo Neighbor, chiamata “neighbors”, che contiene i riferimenti ai territori ad esso collegati.
La classe Territory possiede un costruttore che ne inizializza la quantità di armate con valore “3” e crea una lista di oggetti Neighbor vuota.
È presente inoltre la funzione Start(), ereditata dalla classe MonoBehaviour di UnityEngine, che viene chiamata una sola volta prima dell’elaborazione del primo frame di gioco [23]. All’interno di questa funzione viene istanziato l’oggetto Territory tramite il costruttore e ne vengono settati i campi “territoryname” e “continent”. È stata scelta questa soluzione in quanto i valori richiesti vengono letti dall’oggetto della scena di cui Territory fa parte, perciò la lettura deve avvenire dopo aver istanziato tutti gli oggetti.

Neighbor.cs

La classe “Neighbor.cs” rappresenta il singolo arco che costituisce il grafo e viene istanziata in ogni oggetto “Territory” all’interno della Lista chiamata “neighbors”. Essa permette di mantenere in memoria i collegamenti che intercorrono tra i territori e le loro caratteristiche.
La classe Neighbor.cs è costituita da:

- **Territory** source: questo campo contiene il riferimento all’oggetto Territory da cui parte l’arco;
- **Territory** dest: questo campo contiene il riferimento all’oggetto Territory destinazione dell’arco;
- **string** EdgeType: questo campo consente di pesare il grafo, dando la possibilità di assegnare valori distinti ad ogni arco.

È stata scelto di pesare il grafo per poter distinguere i tipi di collegamento tra i territori, che possono essere collegati sia per via terrestre che per via marina; in questo modo si è reso possibile diversificare i modelli e le animazioni visualizzate ad ogni attacco per rendere la scena più realistica.

```
public class Neighbor : MonoBehaviour {
    private string EdgeType;
    private Territory source;
    private Territory dest;

    public Neighbor(Territory source, Territory t, string s) {
            this.source = source;
            this.dest = t;
            EdgeType = s;
    }
    public Territory getSource() {
            return source;
    }
    public Territory getDest() {
            return this.dest;
    }
    public void setDest(Territory t) {
            this.dest = t;
    }
    public string getEdgeType() {
            return EdgeType;
    }
    public void setEdgeType(string s) {
            EdgeType = s;
    }
}
```

Classe “Neighbor.cs”

StateManager.cs

La classe StateManager gestisce i collegamenti tra i territori e gli eventi che avvengono all’interno del gioco.

StateManager contiene due oggetti fondamentali:

- **List<Player>** players: contiene la lista dei giocatori partecipanti.
- **List<Territory>** territories: contiene la lista dei Territori presenti nella scena; tutti gli oggetti di gioco di tipo “Territory” vengono aggiunti a questa collezione utilizzando il metodo FindObjectsOfType<T>() fornito da UnityEngine[24].

StateManager contiene inoltre una struttura dati apposita, che viene utilizzata per inizializzare il grafo che collega tutti i Territori:

List<KeyValuePair<string source, List<KeyValuePair<string dest, string edgetype>>>>

La struttura dati di base è costituita da una coppia chiave-valore di tipo string - string (KeyValuePair<string, string>), in cui la chiave indica il nome del Territory destinazione ed il valore indica il tipo di arco. Tutti i Territori destinazione di un determinato Territorio pertanto sono inseriti in una lista composta da questo tipo di elementi(List<KeyValuePair<string, string>>).
Ognuna di queste liste è stata poi associata ad una chiave stringa che contiene il nome del Territory sorgente, creando un oggetto di tipo
KeyValuePair<string, List<KeyValuePair<string, string>>>.
Il campo “neighbors” del Territorio sorgente verrà popolato con la lista corrispondente.

```
new KeyValuePair<string, List<KeyValuePair<string, string>>>(
        "TERRITORY",
        new List<KeyValuePair<string, string>>(){
                new KeyValuePair<string, string>("NEIGHBOR", "EdgeType"),
                new KeyValuePair<string, string>("NEIGHBOR", "EdgeType"),
        })
```

Il template utilizzato per l’inserimento di nuovi archi all’interno della lista per la popolazione del grafo.

Infine, queste coppie chiave-valore sono state inserite in una lista da iterare all’avvio del gioco per generare il grafo, chiamata neighbors.

```
foreach(KeyValuePair<string, List<KeyValuePair<string, string>>> t in neighbors) {
  foreach (KeyValuePair<string, string> neigh in t.Value)
        Territories.Find(x => x.name == t.Key)
                .neighbors
                   .Add(new Neighbor(
                        Territories.Find(x => x.name == t.Key).GetComponent<Territory>() ,
                        Territories.Find(x => x.name == neigh.Key) ,
                        neigh.Value));
}
```

La funzione che genera il grafo che collega tutti i territori

Il primo ciclo Foreach scorre la lista dei Territori sorgente, all’interno di questo ciclo viene eseguito un altro ciclo Foreach che scorre la lista dei Territori destinazione. Il

primo metodo Find scorre la lista di Territori e restituisce il territorio con il nome corrispondente al Territorio a cui si vogliono aggiungere gli archi; accedendo alla lista di Neighbors di questo Territorio vengono aggiunti gli archi, istanziando nuovi oggetti di tipo Neighbors.

3.2.2.3 Inizializzazione del gioco

Gli oggetti e le variabili che costituiscono il grafo alla base di ARisiko vengono inizializzati all'interno del metodo "Start()" della classe "StateManager.cs".
Vengono perciò istanziati gli oggetti Player ed aggiunti alla collezione "players"(in questo caso 2).

```
players = new List<Player>();
        players.Add(new Player("AI", Color.black));
        players.Add(new Player("Player", Color.magenta));
```

Poi viene istanziata e popolata la collezione "territories", aggiungendole tutti gli oggetti di tipo Territory.
La collezione Territory viene poi mescolata utilizzando un metodo estensione di "StateManager.cs", chiamato "MyShuffle", che utilizza i metodi della classe Random[25] per generare valori casuali:

```
static class MyShuffle
{
   private static System.Random rng = new System.Random();

   public static void Shuffle<T>(this IList<T> list)
   {
        int n = list.Count;
        while (n > 1)
        {
             n--;
             int k = rng.Next(n + 1);
             T value = list[k];
             list[k] = list[n];
             list[n] = value;
        }
   }
}
```

Metodo estensione "MyShuffle"

La collezione “territories” viene poi iterata ed ogni Territorio viene assegnato ciclicamente ad ogni giocatore, impostando il campo “proprietary” del territorio con il riferimento all’oggetto Player ed il colore dell’oggetto di gioco con il colore che contraddistingue il giocatore, memorizzato anch’esso all’interno dell’oggetto Player.

3.2.2.4 Interazione

L’interazione di ARisiko è stata gestita tramite Input Touch all’interno del metodo “FixedUpdate()”[26] di “StateManager.cs”. FixedUpdate è un metodo ereditato da MonoBehaviour che viene chiamata in modo sincronizzato ad ogni aggiornamento del motore fisico di Unity.

La gestione dell’Input è stata realizzata tramite la classe “Input” integrata in UnityEngine.

Quando viene rilevato un tocco, si istanzia un “Ray”, un oggetto parte di UnityEngine che genera una linea definita da un punto di origine e da una direzione; l’oggetto “Ray” viene generato utilizzando la funzione “ScreenPointToRay()”[27], che riceve in input la posizione del tocco e restituisce un oggetto Ray il cui punto di origine è la posizione della Camera e la direzione è definita dalla posizione del tocco.

```
ray = Camera.main.ScreenPointToRay(Input.GetTouch(0).position);
```

L’oggetto ray viene poi usato come parametro del metodo “Raycast”, che restituisce un valore booleano “true” se l’oggetto “Ray” interseca un Collider.

```
Physics.Raycast(ray, out hit)
```

Il metodo Raycast contiene un parametro di tipo “out” che restituisce un valore di tipo RaycastHit alla variabile passata come parametro, l’oggetto RaycastHit utilizzato conterrà pertanto le informazioni relative al Collider intersecato.

L’oggetto “hit” viene perciò analizzato e, nel caso il Collider sia parte di un oggetto “Territory”, significa che è stato selezionato un Territorio, altrimenti sarà invocato il metodo “ResetClick” della classe “ClickEvent.cs”, passandole come parametro la lista dei Territori del gioco. ResetClick imposterà tutti i colori dei territori con il colore del corrispondente proprietario.

```
public void ResetClick(List<Territory> t)    {
        foreach(Territory tr in t) {
                tr.GetComponent<Renderer>().material.color = tr.Proprietary.PlayerColor;
        }
}
```

Classe “ClickEvent.cs” - Metodo “ResetClick”

La sequenza di attacco si svolge in due fasi:

- Alla prima selezione di un Territorio, viene invocato il metodo “TerritoryClicked()”, che prende in input un parametro Territory t. Il territorio selezionato verrà passato alla funzione “TerritoryClicked”, che lo evidenzierà con il colore bianco e ne itererà la lista di Territori vicini (neighbors); i Territori presenti in questa lista saranno evidenziati, se appartenenti ad un giocatore diverso rispetto al Territorio selezionato, con il colore giallo, nel caso l’arco sia di tipo “Land”, altrimenti con il colore blu.
  Il Territorio appena selezionato sarà infine salvato all’interno di StateManager in una variabile globale chiamata “selected”.

  ```
  public void TerritoryClicked(Territory t) {
    t.GetComponent<Renderer>().material.color = Color.white;
    foreach (Neighbor n in t.neighbors) {
      if(n.getSource().Proprietary.PlayerName != n.getDest().Proprietary.PlayerName) {
          if (n.getEdgeType() == "Land") {
            n.getDest().GetComponent<Renderer>().material.color = Color.yellow;
          } else {
            n.getDest().GetComponent<Renderer>().material.color = Color.blue;
          }
      }
    }
  }
  ```

  Classe: “ClickEvent.cs” - Metodo “TerritoryClicked”

- Se la variabile “selected” contiene il riferimento ad un Territory, sarà eseguito un controllo sul Territorio appena selezionato, questo controllo servirà a garantire che il Territorio sia collegato da un arco al Territorio precedentemente selezionato e che il proprietario dei due Territori non

coincida. Nel caso questo controllo vada a buon fine, sarà avviata la sequenza di attacco, invocando il metodo “TerritoryAttacked” della classe “ClickEvent.cs”. Il metodo “TerritoryAttacked” prende in input un Territorio attacker ed un Territorio defender; di questi Territori ricava poi i riferimenti agli oggetti figli “Tank”. La cabina dell’oggetto “Tank” figlio del Territorio attaccante viene poi ruotata, tramite la funzione “LookAt()”[28] di UnityEngine, in direzione del Territorio attaccato. Il Territorio attaccante conquisterà poi il Territorio difensore, di cui verrà modificato il giocatore proprietario e sarà aggiornato il colore che ne contraddistingue la proprietà, sia dell’oggetto Territorio, che dei suoi oggetti figli “Tank”. Al termine dell’attacco, sarà invocato il metodo “ResetClick”, che reimposterà i colori degli oggetti Territory della scena con i colori assegnati ai rispettivi giocatori proprietari e sarà impostata a “null” la variabile “selected”, permettendo così di ripetere la sequenza di attacco successiva.

```
public void TerritoryAttacked(Territory attacker, Territory defender) {
        Transform tank = attacker.transform.Find("Tank");
        if (tank != null)
        tank.FindChild("TankCabin").LookAt(defender.transform.position);
        //End rotation
        defender.Proprietary = attacker.Proprietary;
        Transform dtank = defender.transform.Find("Tank");
        if (dtank != null) {
                Renderer[] renderers = dtank.GetComponentsInChildren<Renderer>();
                foreach (Renderer r in renderers) {
                        r.material.color = defender.Proprietary.PlayerColor;
                }
        }
  }
```

Classe “ClickEvent.cs” - Metodo “TerritoryAttacked”

Capitolo 4 - Il progetto ARisiko

Il progetto ARisiko è stato sviluppato con l'obiettivo di essere utilizzato principalmente tramite il proprio smartphone. A tal proposito sono state fatte scelte riguardanti le dimensioni del marker e degli oggetti di scena, per garantire un buon grado di comodità nel suo utilizzo. Tuttavia, considerando il rateo d'aspetto che caratterizza la mappa di ARisiko, si consiglia di utilizzare l'applicazione in modalità "Panorama".

ARisiko è stato installato, grazie al plugin integrato in Unity, che ne ha generato il file di installazione(.apk), su uno smartphone Android con un display la cui diagonale è di 5.9'' ed il rateo d'aspetto è di 21:9. Di seguito sono mostrate le immagini di gioco generate da questo dispositivo.

Figura 10.1: Il marker posizionato al centro del tavolo

Il marker è stato stampato su un foglio di dimensioni A5(15cm x 21cm), mentre il tavolo su cui è posizionato è grande 72cm x 136cm.

Figura 10.2: La plancia di gioco posizionata sul marker(Inquadratura in piedi)

La figura 10.2 mostra le dimensioni della plancia di gioco virtuale rispetto al tavolo.

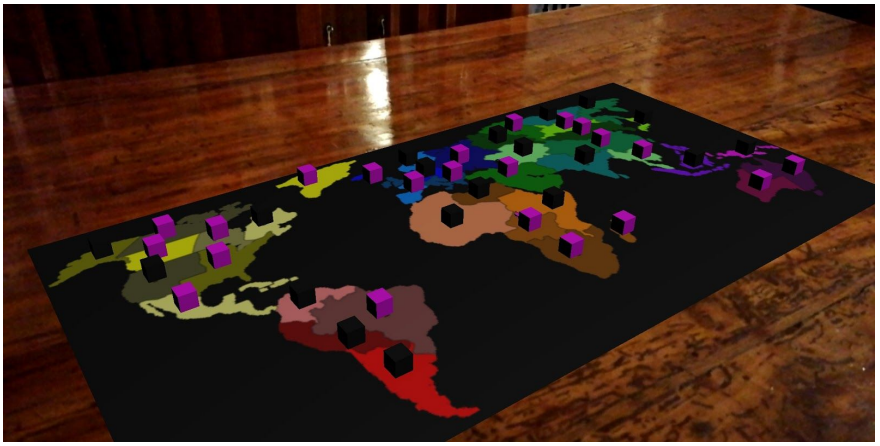

Figura 10.3: La plancia di gioco posizionata sul marker(vista da seduti)

La figura 10.3 riporta la vista della plancia di gioco virtuale dalla postazione di un giocatore.

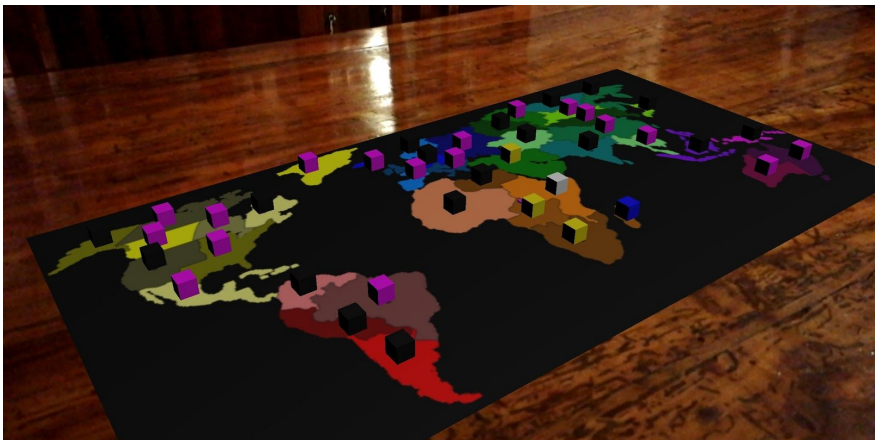

Figura 10.4: Territorio selezionato

La figura 10.4 mostra la situazione in una fase di attacco: il territorio colorato di bianco è di proprietà del giocatore nero ed è stato selezionato. I territori che può attaccare sono pertanto colorati di giallo o di blu, in base al tipo di attacco possibile.

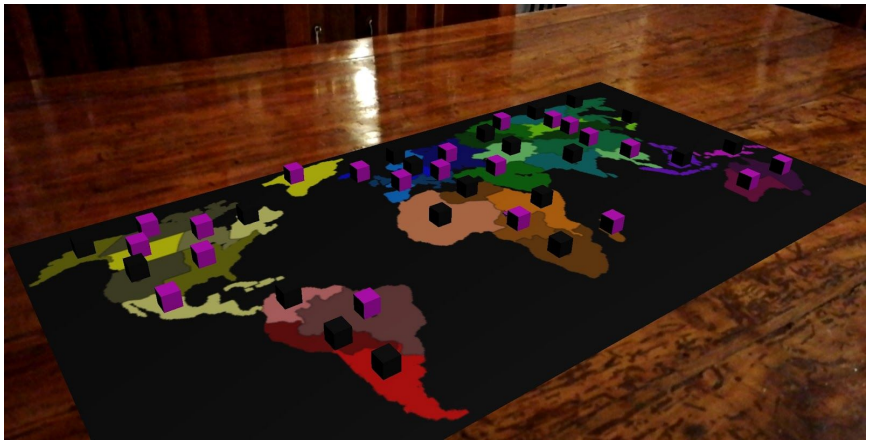

Figura 10.5: Fase di attacco terminata

Il territorio che rappresenta il Sud-Africa è stato conquistato dal giocatore nero.

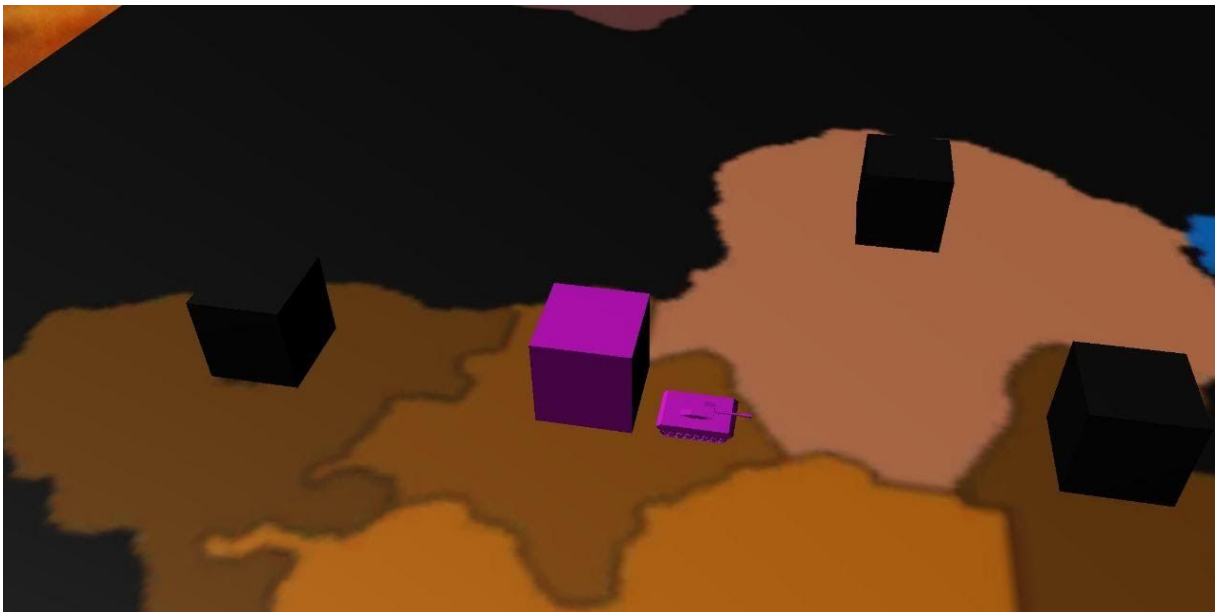

Figura 10.6: Un carro nella posizione predefinita

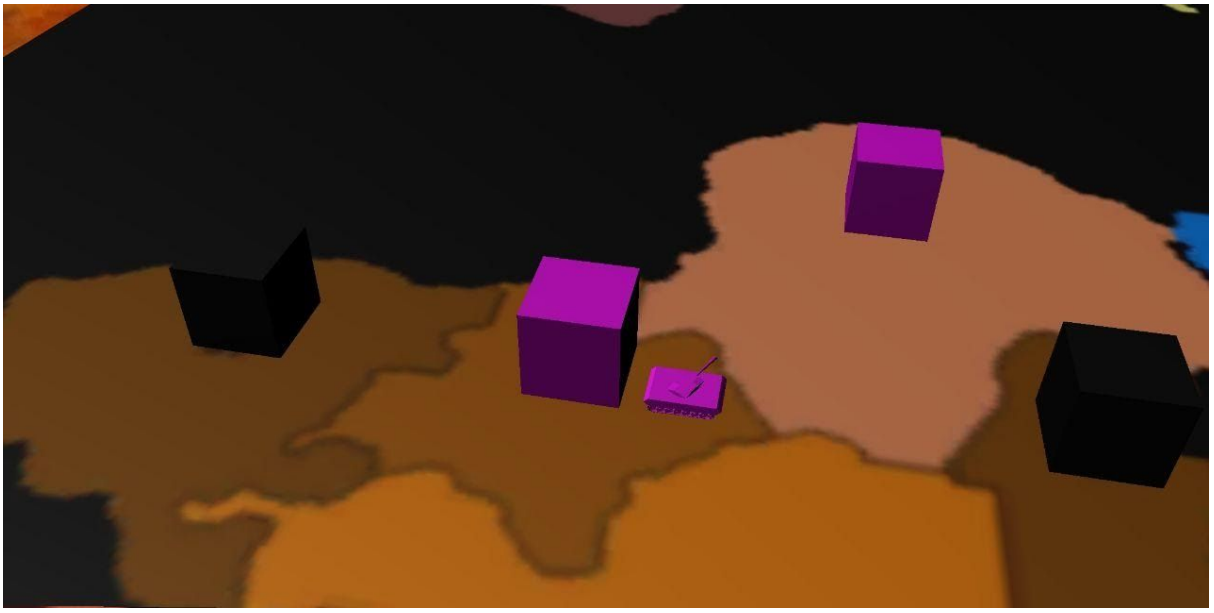

Figura 10.7: Il carro dopo la sequenza di attacco

Tra la figura 10.6 e la figura 10.7 il territorio del giocatore rosa ha attaccato il territorio del giocatore nero, conquistandolo. Il carro armato, che prima si trovava nella posizione predefinita, ora ha il cannone puntato verso il territorio attaccato.

Sviluppi futuri

L’implementazione attuale del progetto ARisiko possiede un grado di scomodità dato dalla necessità di essere utilizzato su un unico dispositivo. In questo caso perciò ogni giocatore dovrà portare a termine il proprio turno e passare il dispositivo al giocatore successivo, togliendo partecipazione alla partita.

L’implementazione di un sistema multigiocatore consentirebbe, invece, di partecipare alle azioni di gioco anche al di fuori del proprio turno, rendendolo più comodo ed immersivo.

Sitografia

[1] https://en.wikipedia.org/wiki/Augmented_reality
[2] https://www.inverse.com/article/18146-l-frank-baum-the-master-key-augmented-reality-futurism
[3] http://uschefnerarchive.com/mortonheilig/
[4] https://www.tomshw.it/altro/storia-della-realta-virtuale/
[5] http://www.idemployee.id.tue.nl/g.w.m.rauterberg/presentations/hci-history/sld096.htm
[6]https://www.researchgate.net/publication/3510119_Augmented_reality_An_application_of_heads-up_display_technology_to_manual_manufacturing_processes
[7] https://learn.g2.com/history-of-augmented-reality
[8] http://www.hitl.washington.edu/artoolkit/documentation/history.htm
[9] https://www.volkswagenag.com/en/group/research/virtual-technologies.html
[10] https://www.ikea.com/ch/it/customer-service/mobile-apps/ikea-place-app-pub0bab12b1
[11] https://www.homestyler.com/
[12] https://en.wikipedia.org/wiki/Google_Glass
[13] https://en.wikipedia.org/wiki/Microsoft_HoloLens
[14] https://opencv.org/
[15] https://developers.google.com/ar
[16] https://developer.apple.com/augmented-reality/
[17] https://it.wikipedia.org/wiki/RisiKo
[18] https://en.wikipedia.org/wiki/Unity_(game_engine)
[19] https://en.wikipedia.org/wiki/Vuforia_Augmented_Reality_SDK
[20] https://en.wikipedia.org/wiki/Blender_(software)
[21] https://engine.vuforia.com/content/vuforia/en/features.html
[22] https://www.blender.org/about/
[23] https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html
[24] https://docs.unity3d.com/ScriptReference/Object.FindObjectsOfType.html
[25] https://docs.unity3d.com/ScriptReference/Random.html
[26] https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html
[27] https://docs.unity3d.com/ScriptReference/Camera.ScreenPointToRay.html
[28] https://docs.unity3d.com/ScriptReference/Transform.LookAt.html

Ringraziamenti

Ringrazio i miei genitori, che in tutti questi anni hanno sempre creduto in me e mi hanno sempre motivato ad arrivare in fondo, senza mai cambiare idea ed aiutandomi in ogni mia scelta.

Ringrazio la professoressa Damiana Lazzaro, che ha deciso di accompagnarmi verso il coronamento di questo percorso, con grande disponibilità e fornendomi un sostegno fondamentale per la mia riuscita.

Ringrazio Claudia, che è sempre al mio fianco, e in questi anni mi ha sempre convinto a dare il meglio di me, e ha sempre saputo come sostenermi ed aiutarmi quando ne avevo bisogno.

Ringrazio infine i miei amici, che mi hanno aiutato in questi anni ad affrontare questo percorso, a superare i momenti bui, e a festeggiare quelli più luminosi.

Grazie