Università Politecnica delle Marche

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sviluppo e progettazione di una soluzione multipiattaforma in realtà aumentata e virtuale facilmente adattabile a diversi ambienti

-

Development and design of a multi-platform solution in augmented and virtual reality easily adaptable to different environments

Relatore:

Prof. Aldo Franco Dragoni

Correlatore:

Ing. Ivano Corradetti

Tesi di Laurea di:

Angelo Serafini

Anno Accademico 2021/2022

Sommario

INTRODUZIONE

Alla base di questo progetto vi sono lo studio e la realizzazione tramite game engine di soluzioni in realtà aumentata e virtuale godibili per l'utente finale su una qualsiasi piattaforma che utilizzi uno dei sistemi operativi più diffusi, integrando quindi queste due tecnologie, AR e VR, per ottenere un prodotto in grado di valorizzare qualsiasi ambiente reale, costruendo quindi un software in grado di essere facilmente adattato ad una qualsivoglia ambientazione integrata in futuro. In questa tesi sono state implementate tre piazze storiche del territorio marchigiano.

Il software nasce da una collaborazione con l'azienda Beesoft[1], tramite il consulente Ivano Corradetti[2], correlatore di questa tesi.

La tesi tratta della realizzazione di un'applicazione multi-piattaforma che integra la Realtà Aumentata (AR), godibile sul luogo, alla sua ricostruzione virtuale in ambiente 3D, liberamente esplorabile tramite applicazione desktop o web. Gli elementi che vanno ad "aumentare" la realtà vengono definiti e posizionati nell'ambiente virtuale e le eventuali modifiche apportate ad essi si rispecchiano sull'applicazione AR quando si va ad inquadrare con la telecamera del dispositivo il corrispettivo ambiente reale.

La tesi è divisa in cinque capitoli, preceduti da un'introduzione. Il primo capitolo descrive brevemente il concetto di Realtà Aumentata, una tecnologia in rapida evoluzione che solo negli ultimi anni è arrivata all'utente comune, anche grazie al miglioramento dei dispositivi mobili alla portata di tutti. Il secondo capitolo tratta dei principali strumenti utilizzati: l'ambiente di sviluppo Unity, soffermandosi brevemente sui plugin e i servizi ad esso integrati, e Vuforia Engine. Il terzo capitolo elabora i requisiti del progetto proposto dall'azienda, dettagliando quindi gli obiettivi del progetto. Il quarto capitolo tratta della realizzazione vera e propria del progetto, descrivendone nel dettaglio le componenti implementate. Il quinto

[1] https://www.beesoft.it/
[2] https://www.ivanocorradetti.com/

ed ultimo capitolo contiene una breve conclusione del trattato e tratta alcuni eventuali sviluppi futuri, non attualmente realizzabili per limitatezza tecnologica.

1.0 LA REALTÀ AUMENTATA

La Realtà Aumentata è un'esperienza interattiva di un ambiente reale dove gli oggetti che risiedono nel mondo reale sono "aumentati" da informazioni percettive generate da computer, a volte attraverso diverse modalità sensoriali, come visiva, uditiva, tattile e olfattiva. L'informazione sensoriale sovrapposta può essere costruttiva, se si aggiunge all'ambiente naturale, o distruttiva, se maschera l'ambiente naturale, ed è inserita nel mondo fisico facendo in modo che sia percepita come un aspetto naturale dell'ambiente reale. Quindi, la Realtà Aumentata altera la percezione dell'utilizzatore dell'ambiente reale, mentre la Realtà Virtuale rimpiazza completamente il mondo circostante per l'utente con un ambiente simulato.

Il merito principale della Realtà Aumentata è che porta componenti del mondo digitale nella percezione del mondo reale di una persona, e non come una semplice rappresentazione di dati, ma come integrazione di sensazioni percepite come parti naturali dell'ambiente. Il primo sistema di Realtà Aumentata funzionante che ha prodotto esperienze di realtà mischiate per gli utenti risale ai primi anni '90, sviluppato dall'aeronautica militare Americana. Le prime esperienze commerciali di Realtà Aumentata sono state per la maggior parte nei settori di intrattenimento e videoludici, ma ora anche altri settori stanno sviluppando soluzioni di Realtà Aumentata per

esempio in ambito educativo, ottenendo informazioni scannerizzando o vedendo un'immagine tramite un dispositivo mobile, o anche nel campo delle costruzioni dove i lavoratori indossano elmetti che gli permettono di visualizzare informazioni sul cantiere in realtà aumentata.

In questo progetto la Realtà Aumentata è utilizzata per migliorare l'ambiente reale osservato per offrire un'esperienza migliore all'utente finale, sovrapponendo immagini virtuali al mondo circostante tramite la camera di un dispositivo mobile.

2.0 STRUMENTI

2.1 Ambiente di sviluppo Unity

Per lo sviluppo del progetto si è scelto di lavorare su un Game Engine, in particolare Unity, essendo esso l'ambiente di sviluppo più adeguato per poter realizzare un'applicazione che sposa sia VR che AR.

Un Game Engine è un IDE disegnato per permettere la costruzione di videogiochi o altre applicazioni con grafica in tempo reale. Le funzionalità principali tipiche di un game engine includono un motore per il rendering di componenti grafiche bidimensionali e tridimensionali, un motore per la fisica e la gestione delle collisioni, strumenti per la gestione di suoni, scripting, animazioni, intelligenza artificiale, networking, streaming, gestione della memoria, threading, localizzazione e divisione in scene del progetto.

L'ambiente di sviluppo Unity è disponibile sia su Microsoft Windows sia su macOS, e può produrre applicazioni compatibili con iOS, Android, Tizen, Windows, Universal Windows Platform, Mac, Linux, WebGL, PlayStation 4, PlayStation Vita, Xbox One, Wii U, 3DS, Oculus Rift, Google Cardboard, SteamVR, PlayStation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV, Samsung Smart TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple's ARKit, Google's ARCore, e Vuforia.

Unity offre agli utenti la possibilità di creare giochi sia in 2D che in 3D, e utilizza un'interfaccia di scripting in C# sia per l'editor di Unity sotto la forma di plugins e sia per i giochi stessi. Essendo il più diffuso game engine ha un vasto supporto per i plugins, che permettono di integrare molte librerie middleware, software che fungono da intermediari fra strutture e programmi informatici, permettendo l'utilizzo di librerie come ad esempio delle librerie per lo sviluppo in Realtà Aumentata come in questo caso Vuforia.

I seguenti paragrafi sono destinati a descrivere le principali componenti aggiuntive integrate all'ambiente Unity, utilizzate per realizzare il progetto.

2.1.1 Newtonsoft Json

Json.NET è un popolare framework JSON ad alte prestazioni per .NET, ed anche il framework più popolare dell'ecosistema .NET in generale.

Le principali funzioni che il package offre sono:

- serializzazione JSON flessibile per la conversione tra oggetti .NET e JSON;
- LINQ (Language-Integrated Query) per scrivere e leggere manualmente JSON;
- JSON in formato indentato per una facile lettura e compresione;
- conversione di JSON in e da XML.

Questa libreria viene utilizzata nel progetto ai fini di effettuare una *serializzazione* degli stati degli oggetti 3D destinati ad aumentare la realtà nel lato AR del programma.

La *serializzazione* è il processo di traduzione di una struttura dati o dello stato di un oggetto in un formato che può essere salvato o trasmesso per poi essere ricostruito in un secondo momento, anche in un ambiente diverso. La risultante serie di bit può quindi essere letta, a seconda del formato di serializzazione, per creare un clone semanticamente identico all'oggetto originale. Tuttavia, la serializzazione di un oggetto, in un contesto di programmazione ad oggetti, non permette di salvare e replicare i metodi ad esso collegati.

Questo processo, all'interno dell'applicazione, permette la comunicazione tra ambiente VR e AR, applicando la serializzazione sia a informazioni di stato del programma, come quali scenari sono associati a quali date, sia ad informazioni legate allo stato degli oggetti, come posizione e scalatura.

Le informazioni serializzate e poi convertite in un JSON facilmente comprensibile e modificabile anche esternamente all'ambiente di sviluppo vengono poi salvate utilizzando il PlayFabSDK, descritto in seguito in questo capitolo.

2.1.2 PlayFabSDK

PlayFabSDK è la libreria che permette il salvataggio in tempo reale su un server dei dati ottenuti tramite la serializzazione descritta nel capitolo precedente.

Azure PlayFab è un servizio Microsoft che offre una piattaforma backend completa per la gestione e l'analisi di game services in tempo reale.

I principali servizi offerti da PlayFab che sono stati integrati nell'applicazione sono:

- il salvataggio di dati su Cloud, sia come singole variabili e sia come JSON contenenti tutte le informazioni che costituiscono lo stato di un oggetto al momento del salvataggio;
- eventuale gestione di account, offrendo la possibilità di collegarlo a servizi esistenti come Google o Windows;
- gestione di permessi di caricamento di dati salvati in base alla piattaforma in utilizzo o all'account in uso.

Per permettere la comunicazione dell'ambiente di sviluppo Unity con i server PlayFab è necessario, inoltre, estendere l'interfaccia dell'ambiente con l'estensione PlayFab per consentire a quest'ultimo la gestione dei servizi da utilizzare nel progetto, tramite il login con l'account da sviluppatore e l'inserimento dei dati relativi all'applicazione in uso.

2.1.3 Oculus XR Plugin e XR Management Support

L'Oculus XR Plugin è necessario per permettere di sviluppare un'applicazione compatibile con visori per realtà virtuale o mista.

Il plugin si divide in due sottosistemi principali:

- Display subsystem: fornisce supporto per il rendering stereo al XR Plugin e offre supporto per i pricipali sistemi Windows, Rift e Rift S

tramite DirectX 11, e Android, per Quest e Quest 2 tramite OpenGL ES 3.0;

- Input subsystem: fornisce supporto per i sistemi di input aptici e per i controller collegati ai visori.

A questo plugin è stato integrato, seppur non obbligatorio per il funzionamento base, l'XR Management Support plugin.

Questo plugin permette di interagire in modo più intuitivo e versatile con tutte le funzionalità del plugin principale, offrendo due tipi di servizi fondamentali:

- Opzioni a runtime: permette di configurare molte opzioni durante l'esecuzione come la modalità di rendering, di depth buffer sharing, di gestione della latenza e l'integrazione della Dashboard Oculus;
- Gestione del ciclo di vita del programma: implementa il sistema di gestione del ciclo vitale, ovvero le funzioni di inizializzazione, spegnimento, pausa e ripresa.

2.1.4 OBJImporter/FBXExporter

Runtime OBJ Importer e FBXExporter sono le due librerie utilizzate nel progetto per gestire l'inserimento e l'estrazione sotto particolari condizioni degli asset 3D, preservandone materiali e texture.

Come suggerisce il nome, la particolarità del Runtime OBJ Importer è appunto la capacità di inserire un nuovo oggetto tridimensionale durante l'esecuzione stessa del programma, cosa altrimenti non contemplata dall'editor Unity. Per poter permettere al programma di caricare materiali e texture insieme al modello base 3D è sufficiente che siano tutti e tre nella stessa cartella di origine e con lo stesso nome. Il plugin inoltre permette il caricamento dei file da stream oltre che da cartella locale.

FBX Exporter permette di esportare modelli 3D offrendo un workflow fluido tra Editor Unity e i principali programmi di modellazione 3D come Autodesk® Maya®.

La principale forza di questo plugin è la possibilità di esportare geometrie, animazioni, luci e camere come FBX per trasferire l'interezza del contesto del singolo modello 3D ad un programma esterno, con il quale la libreria comunica, ricordandosi la destinazione dei file e quali di essi è necessario poi importare di nuovo nell'Editor. All'interno del progetto questo plugin è stato utilizzato principalmente per permettere la creazione di Model Target, concetto spiegato successivamente all'interno della tesi, a partire dai modelli 3D di edifici o di dettagli di essi.

2.1.5 VuforiaEngineAR

VuforiaEngineAR è il plugin che permette di integrare Vuforia Engine SDK all'editor Unity.

Questo plugin inserisce nell'editor le componenti necessarie per poter sviluppare il lato di Realtà Aumentata dell'applicazione, ovvero la Camera AR e tutti i tipi di Target supportati da Vuforia.

La Camera AR va a sostituire la normale Camera all'interno della scena, andando quindi a rappresentare effettivamente il punto di vista dell'utente, preservando comunque anche tutte le proprietà e caratteristiche di una Camera standard. Consente il riconoscimento degli elementi inquadrati nel mondo reale, i Target, sui quali agire per aumentarli, per poi procedere a fissare e manenerne il tracciamento per poter permettere agli elementi di realtà aumentata di restare ancorati al proprio Target reale, a prescindere dai movimenti effettuati dal dispositivo.

Per poter iniziare ad utilizzare queste componenti, oltre alla normale installazione, è necessario inserire una chiave da sviluppatore Vuforia, ottenibile sul sito ufficiale, all'interno delle impostazioni presenti sulla Camera AR.

Per definire dei Target di qualsivoglia tipo occorre prima inizializzarli sul sito di Vuforia, nella sezione Developer e nel progetto collegato alla chiave in uso. Una volta inizializzati è sufficiente effettuare il download in formato Unity package per poterli aggiungere effettivamente al progetto.

Infine, occorre definire la scala delle componenti del progetto in base alla scala dei Target inizializzati sul sito Vuforia, in quanto modificare la scala dei Target all'interno dell'editor Unity causa problemi di prestazioni e di perdita del tracciamento.

2.1.6 AR Foundation

AR Foundation è un framework appositamente costruito per lo sviluppo di applicazioni in realtà aumentata che permette agevolmente di effettuare il deploy di un'applicazione in un contesto multi-platform.

Il pacchetto include funzionalità fondamentali da:

- ARKit: piattaforma di sviluppo Apple per applicazioni di realtà aumentata per dispositivi mobili iOS;
- ARCore: piattaforma di sviluppo Google per applicazioni di realtà aumentata per dispositivi mobili Android;
- Magic Leap: visori improntati alla realtà mista;
- HoloLens: visori Microsoft improntati alla realtà mista.

Inoltre, il pacchetto contiene funzionalità uniche per Unity per permettere lo sviluppo di applicazioni solide su ogni piattaforma attraverso un workflow unificato.

AR Foundation può essere utilizzato in concomitanza con Vuforia Engine, permettendo a quest'ultimo di sfruttarne alcune caratteristiche. Tuttavia se integrate in modo errato è possibile riscontrare problemi di qualità e latenza, in quanto entrambi i plugin cercherebbero di accedere contemporaneamente allo stesso sistema di coordinate per l'ancoraggio e il tracciamento in tempo reale.

2.2 Vuforia

Vuforia Engine è un software development kit (SDK) per realtà aumentata mirato principalmente a dispositivi mobili che permette la creazione di applicazioni di realtà aumentata e mista. Utilizza tecnologie di computer vision per riconoscere e tracciare immagini planari e oggetti 3D in tempo reale. Questa capacità di registrare immagini permette agli sviluppatori di posizionare e orientare oggetti virtuali, come modelli 3D e altri tipi di media, in relazione con oggetti appartenenti al mondo reale quando questi sono inquadrati dalla camera di un dispositivo mobile. L'oggetto virtuale quindi traccia posizione e orientamento dell'immagine in tempo reale, in modo tale che la prospettiva dell'utente corrisponda con la prospettiva del

bersaglio, dando così l'impressione che l'oggetto virtuale faccia effettivamente parte della scena nel mondo reale.

Vuforia offre API in C++, Java, Objective-C++, in linguaggi .NET tramite l'estensione per Unity game engine. In questo modo l'SDK supporta sia sviluppo in ambiente nativo per iOS, Android, e UWP, sia permette lo sviluppo delle applicazioni AR in Unity, consentendo così una più facile architettura multi-piattaforma.

Le principali funzioni dell'SDK sono:

- AR recognition: la capacità di riconoscere tramite la camera del dispositivo scelto l'ambiente circostante, utilizzando angoli, rilievi, piani e oggetti immobili per valutare distanze e mantenere stabilità, o dei marker, come immagini bersaglio o oggetti 3D bersaglio, consentendo l'avvio in automatico della funzione che, una volta riconosciuto il bersaglio voluto, genera l'evento desiderato in Realtà Aumentata, come ad esempio fare apparire un'automobile virtuale sopra il bersaglio.
- AR tracking: la capacità di tracciare l'oggetto generato in Realtà Aumentata, dando la possibilità di muovere il dispositivo o il bersaglio generante senza però che scompaia l'elemento virtuale, facendogli mantenere la sua posizione relativa che occuperebbe nello spazio reale, dando quindi l'impressione che l'oggetto sia effettivamente parte dell'ambiente in quanto ci si può avvicinare, allontanare o girare intorno a differenza di un normale oggetto rappresentato su schermo, il quale anche muovendo il dispositivo viene rappresentato sempre alla stessa distanza.
- Content rendering: la capacità di generare gli oggetti virtuali tridimensionali di qualità aggiornandoli in tempo reale.

Il tracciamento dei bersagli in Vuforia può essere diviso in due categorie principali: il **tracciamento di immagini** ed il **tracciamento di oggetti**.

2.2.1 Tracciamento di immagini

A sua volta il tracciamento di immagini si distingue in **Image Targets**, **Multi Targets**, **Cylinder Targets** e **VuMarks**.

Gli **Image Targets** sono bersagli costituiti da un'immagine piana, come una foto o il lato di un pacco. È il tipo di bersaglio più semplice e al contempo tra i più potenti, offrendo molta versatilità e buona precisione. L'Engine riconosce e traccia l'immagine facendo una comparazione tra i dettagli naturali estratti dalla camera del dispositivo con il bersaglio conosciuto presente nel database. A questo punto Vuforia è in grado di tracciare l'immagine reale e aumentarla in modo che le componenti virtuali possano seguire fluidamente i movimenti e le pose dell'immagine reale. Per poter creare un Image Target è necessaria come base una qualsiasi immagine con larghezza minima superiore a 320 pixels, un peso non superiore a 2.25MB e del numero più alto possibile di features individuabili ponendo l'immagine in scala di grigi. Questa immagine va successivamente caricata nel *Vuforia Target Manager*, presente nell'area personale del sito Vuforia, per processamento e valutazione. Il Target Manager produce quindi l'immagine processata e rappresenta, sia come dato sia in modo visivo, le features estratte da essa. Inoltre, Vuforia offre una semplice valutazione da una a cinque stelle sulla qualità prevista di riconoscimento e tracciamento, basandosi sulla quantità di features estratte (**Fig.1**). Una volta conclusa l'elaborazione è possibile scaricare il database contenente i propri bersagli direttamente in formato integrabile automaticamente da Unity.

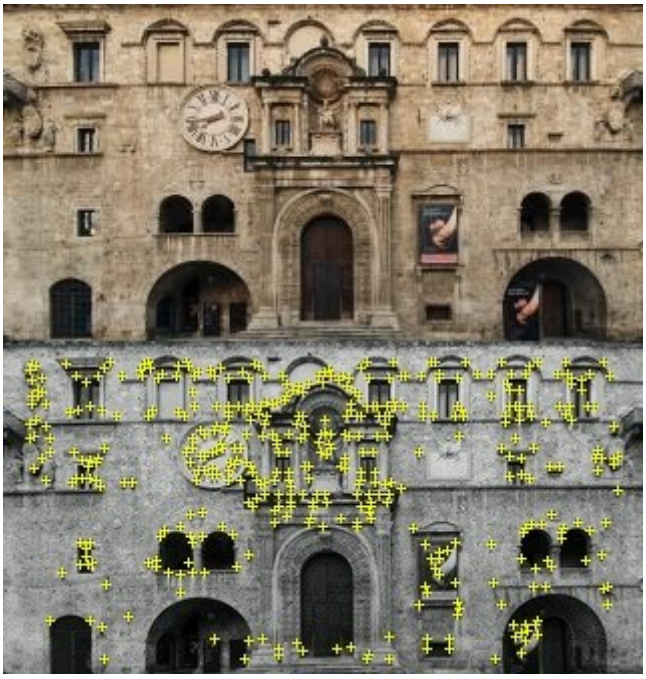

Fig.1, features estratte da una foto del fronte di Palazzo dei Capitani

I **Multi Target** sono collezioni di molteplici Image Targets combinati tra loro per formare una forma geometrica definita come un cubo, permettendo quindi riconoscimento e tracciamento per ogni faccia della forma geometrica. I Multi Targets agiscono in maniera tale che ogni faccia di esso venga tracciata nello stesso momento in quanto condividono una posa predefinita relativa all'origine del Multi Target. Questo consente il tracciamento dell'intero Multi Target ogni volta che ognuno dei bersagli che lo costituiscono viene riconosciuto, cosicchè il Multi Target possa agire da riferimento singolo per ognuno dei sui bersagli figlio.

I **Cylinder Targets** sono utilizzati per il riconoscimento e il tracciamento di immagini impacchettate in una forma cilindrica o conica. Vuforia traccia i lati e le basi piane dei bersagli cilindrici. Questi targets sono principalmente utilizzati per riconoscere prodotti comuni commerciali come lattine e bottiglie, ma ogni oggetto che ricordi queste forme può essere utilizzato in maniera affidabile come bersaglio. Sono costruiti in modo analogo ai Multi

Target, e di conseguenza condividono i prerequisiti degli Image Target per la scelta delle immagini che li costituiscono.

L'ultimo tipo di bersaglio per il tracciamento delle immagini è il **VuMark**, un tipo di codice a barre liberamente personalizzabile nella forma. I VuMark costituiscono una soluzione universale per consentire un'esperienza di realtà aumentata su qualsiasi oggetto. È un metodo adatto ad immagazzinare dati come URL e codici e contemporaneamente fornire un bersaglio affidabile per il tracciamento AR, cosa non possibile con un normale codice a barre a matrice. Lo stesso design di VuMark può essere utilizzato con un range di identificatori unici, ottendendo quindi un unico logo bersaglio per molti oggetti offrendo comunque informazioni differenti per ognuno di essi. Anche la creazione dei VuMark si svolge tramite Vuforia Target Manager, utilizzando come input dei file SVG.

2.2.2 Tracciamento di oggetti

Il tracciamento di oggetti reali si distingue in **Model Targets**, **Area Targets** e **Ground Plane**.

I **Model Targets** permettono alle applicazioni costruite con Vuforia di riconoscere e tracciare particolari oggetti nel mondo reale basandosi sulla forma dell'oggetto. Per poter creare un Model Target di un particolare oggetto è necessario come base un modello tridimensionale dell'oggetto, come un modello CAD o una scansione 3D dell'oggetto. Questa base deve necessariamente essere geometricamente rigida, ovvero non deformabile o malleabile, e deve avere superfici stabili in quanto elementi come specchi o vetri trasparenti non sono supportati.

Il tracciamento dei Model Target è pensato assumendo che l'oggetto tracciato rimanga immobile dopo essere stato riconosciuto. L'utente può

muoversi liberamente attorno all'oggetto, ma idealmente l'oggetto in sé dovrebbe restare immobile.

Gli oggetti colorati o con dei motivi sulla superfice sono più semplici da tracciare rispetto a oggetti composti da un colore unico e uniforme, qualche varianza sulla superficie è necessaria per distinguere gli oggetti.

La complessità geometrica è la chiave per distinguere un oggetto dalle altre forme presenti nell'ambiente, forme semplici come cubi o sfere sono facilmente confondibili con gli altri oggetti presenti nell'area dell'utente. Anche oggetti perfettamente simmetrici possono causare problemi in quanto il tracker potrebbe non essere in grado di distinguerne i lati.

Inoltre, i modelli degli oggetti devono avere la stessa esatta forma e scala delle loro controparti reali. Gli oggetti con una scalatura errata possono essere distinti correttamente, ma il loro tracciamento non sarebbe affidabile. La teconologia Model Target ha una tolleranza del 10% di deviazione di scalatura tra modello e oggetto reale, oltre la quale si presenta un deterioramento delle prestazioni.

Per poter creare un Model Target è necessario l'utilizzo di un programma chiamato Model Target Generator, abbreviato con MTG. L'applicazione converte un modello 3D esistente in un database Vuforia utilizzabile per riconoscimento e tracciamento, notificando eventualmente se il modello di partenza è inutilizzabile come target nella condizione attuale. Il MTG permette di definire una o più Viste Guida per ogni oggetto e consente di unire più Model Targets differenti, ognuno con le proprie Viste Guida Avanzate in un unico database. Una Vista Guida è composta dalla coppia formata dal modello 3D e da una specifica angolazione e distanza da esso, come visibile in **Fig.2**. Molteplici Viste Guida di angolazioni principali, dalle quali si suppone che gli utenti inquadrino l'oggetto, aiutano ad addestrare i Model Target Database con meccanismi di deep learning per permettere un riconoscimento facilitato dai punti di origine delle Viste Guida anche alterando l'angolazione.

Fig.2, esempio di una Vista Guida per il Palazzo dei Capitani

Gli **Area Target** utilizzano l'ambiente stesso come bersaglio, tracciando ed aumentanto l'area circostante. Utilizzando quindi una scansione 3D ad alta fedeltà è possibile adottare gli elementi stazionari dell'ambiente come ancore per il riconscimento e quindi tracciamento. Tuttavia, al momento di scrittura di questa tesi, l'Area Target risulta una tecnologia molto recente e di conseguenza ancora limitata, sia per i proibitivi requisiti di scansione dell'ambiente che richiedono strumentazioni professionali e non commodity, sia per le limitazioni esistenti negli ambiti di applicabilità, dato che al momento funziona in modo affidabile solo in ambienti di dimensioni moderate e non all'aperto, nei quali sono presenti numerosi elementi costanti tra gli utilizzi.

Il **Ground Plane** permette di ancorare gli oggetti virtuali a superfici orizzontali come tavoli o pavimenti. A differenza di tutti gli altri tipi di

riconoscimento visti in precedenza, il Ground Plane è quindi l'unico che non ha bisogno di un bersaglio predefinito a monte dato che ogni superficie sufficentemente riconoscibile, ovvero non troppo riflettente o trasparente, rappresenta un ancoraggio possibile. Grazie all'integrazione di Vuforia con *AR Foundation*, introdotto nel capitolo sui plugin di Unity Engine, è possibile anche utilizzare superfici verticali come punti di ancoraggio.

2.2.3 Device Pose Observer

Il tracciamento delle informazioni su posizione e orientazione del dispositivo in utilizzo rispetto al mondo reale è eseguito dal Device Pose Observer. È computato utilizzando i frame della camera dell'ambiente e le misurazioni ottenute dai sensori.

Il tracciamento utilizza *Vuforia Fusion* per poter funzionare, la quale è la componente che permette di individuare e "fondere" le informazioni ottenibili dal device in utilizzo al Vuforia Engine. Queste informazioni comprendono sia le elaborazioni ottenute tramite componenti hardware come camere, sensori e chipsets, sia le funzionalità dell'ambiente di sviluppo AR nativo del dispositivo, come ad esempio ARKit per iOS, risolvendo quindi una problematica dello sviluppo in realtà aumentata data dalla frammentazione dei dati.

Il *Device Pose Observer* permette quindi il tracciamento esteso di tutti i tipi di bersagli visti e del ground plane. Per tracciamento esteso si intende la capacità del dispositivo di ricordare dove sono ancorati gli oggetti virtuali nel mondo reale anche se si smette di inquadrarli, in modo tale da fornire più realismo all'ancoraggio e dare così la possibilità all'utente di girare liberamente per l'ambiente mantenendo tutti gli elementi virtuali ancorati correttamente. A differenza di altri Observers che tracciano immagini e oggetti, vengono quindi tracciati i dettagli visivi inquadrati dell'ambiente circostante e, se presente nel dispositivo, viene utilizzato anche il sensore di

tracciamento inerziale per determinare la posa del dispositivo con sei gradi di libertà.

Inoltre, il *Device Pose Observer* entra in gioco quando si presentano situazioni che interrompono il tracciamento AR per cause esterne, come ad esempio il presentarsi di una chiamata in entrata durante l'utilizzo dell'applicazione che porta alla messa in pausa della stessa. Per riprendere il tracciamento il programma può riposizionarsi correttamente e in modo automatico se il tempo trascorso e il movimento effettuato durante la pausa non sono eccessivi.

3.0 ANALISI DEI REQUISITI

3.1 Raccolta dei requisiti

L'analisi dei requisiti è una attività preliminare allo sviluppo di un software, il cui scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire, ovvero i requisiti che devono essere soddisfatti dal software sviluppato per soddisfare le esigenze del committente.

L'obiettivo del progetto è la realizzazione di un software multipiattaforma, con versioni desktop, web, android e iOS, che permetta di valorizzare ambienti reali utilizzando in parallelo tecnologie di realtà aumentata e di ricostruzione virtuale dell'ambiente, rimanendo facilmente riadattabile per un qualsiasi numero di possibili ambienti reali diversi.

Il primo ambiente utilizzato in questa tesi è stato Piazza del Popolo di Ascoli Piceno, al quale sono seguite porzioni dei centri storici di Grottammare ed Offida.

Piazza del Popolo prevede sei scenari differenti, ognuno dei quali veste la piazza con gli scudi raffiguranti i simboli dei sei Sestieri di Ascoli Piceno,

Il progetto quindi è composto da due sottoprogetti comunicanti tra loro:

- la realizzazione di un'applicazione che permetta di visitare e modificare tutte le diverse ambientazioni implementate in ambiente virtuale e quindi renderle liberamente esplorabili dall'utente desktop o web come in un classico videogioco, offrendo inoltre le possibilità di alternare diversi scenari per ogni ambientazione e, per ognuno di essi, di modificare, entro limiti imposti e tramite un'interfaccia utente alla portata di chiunque senza competenze specifiche, posizione e orientazione degli oggetti virtuali aggiunti all'ambientazione e le date assegnate ai diversi scenari che definiscono qual è lo scenario attivo predefinito all'accensione dell'applicazione in base alla data odierna;

- la realizzazione di un'applicazione mobile in realtà aumentata che permette di visualizzare, inquadrando con il dispositivo l'ambiente reale corrispettivo, gli oggetti che compongono ogni scenario in modo parallelo a quelli visualizzabili nella versione virtuale, ovvero utilizzando ad ogni accensione le ultime impostazioni di date e posizioni definite nella versione desktop/web.

3.2 Requisiti funzionali

Di seguito quindi i principali requisiti funzionali individuati per la componente del progetto di esplorazione digitale:

- L'utente deve essere in grado di muoversi e girarsi liberamente nell'ambiente;
- L'utente deve essere vincolato a terra all'altezza media di una persona reale rispetto all'ambiente;
- L'ambiente deve prevedere la collisione con l'utente per evitare, ad esempio, che quest'ultimo passi attraverso dei muri;
- L'utente non deve essere in grado di fuoriscire dall'area prevista;
- L'utente deve essere il grado di alternare gli scenari in tempo reale;
- L'applicazione deve essere in grado di salvare le modifiche effettuate dall'utente;
- L'applicazione, al lancio, deve aggiornarsi automaticamente con i cambiamenti più recenti effettuati dall'utente;
- L'applicazione, al lancio, deve caricare lo scenario relativo alla data riconosciuta nel dispositivo in uso;
- L'utente deve essere in grado di modificare le date associate agli scenari;

- L’applicazione deve gestire e segnalare errori riscontrati in seguito al cambio di una data, come ad esempio formattazione incorretta o data già in utilizzo da un altro scenario;
- L’utente deve essere in grado di modificare, entro limiti preimpostati, quali degli elementi aggiuntivi all’ambiente andranno a comporre lo scenario;
- L’utente deve essere in grado di modificare, entro limiti preimpostati, la posizione degli elementi aggiuntivi all’ambiente che compongono lo scenario;
- L’utente deve essere in grado di interagire tramite click con gli elementi aggiuntivi all’ambiente che compongono lo scenario;
- L’applicazione deve essere in grado di alternare diversi ambienti con i relativi scenari in runtime;
- L’applicazione deve essere strutturata in modo tale da poter funzionare come applicazione web;

Di seguito, invece, i principali requisiti funzionali individuati per la componente del progetto di realtà aumentata:

- L’applicazione deve essere in grado di riconoscere elementi predefiniti dell’ambiente reale ed applicare su di essi gli elementi virtuali dello scenario corrente;
- L’applicazione deve essere in grado di tracciare il movimento dell’utente per mantenere gli elementi virtuali saldamente ancorati a quelli reali;
- L’applicazione, al lancio, deve aggiornarsi automaticamente con i cambiamenti più recenti effettuati dall’utente nella versione desktop per rimanere sempre sincronizzata con quest’ultima;
- L’applicazione, al lancio, deve caricare lo scenario relativo alla data riconosciuta nel dispositivo in uso;
- L’applicazione deve ancorare gli elementi virtuali al mondo reale nelle ultime posizioni salvate dall’utente nell’applicazione desktop;

- L’applicazione deve essere in grado di funzionare anche su dispositivi non high-end;
- L’applicazione deve essere in grado di cambiare lo scenario corrente a runtime senza perdere il tracciamento attuale, andando quindi a cambiare gli elementi che vengono visualizzati senza la necessità di riconoscere nuovamente l’ambiente;
- L’utente deve essere in grado di interagire con gli oggetti che compongono gli scenari tramite tocco sullo schermo o prossimità:
- L’applicazione deve essere strutturata in modo tale da essere facilmente trasportabile su piattaforme diverse, quindi facendo attenzione in fase di realizzazione a non utilizzare componenti che potrebbero entrare in conflitto con apparecchi o sistemi operativi differenti.

È requisito funzionale fondamentale che racchiude entrambe le applicazioni:

- Il progetto deve essere strutturato in modo tale da essere facilmente riutilizzabile ed adattabile con qualsiasi altra ambientazione si volesse in futuro implementare.

3.3 Requisiti non funzionali

I principali requisiti non funzionali individuati in fase di raccolta ed analisi dei requisiti sono:

- L'applicazione deve essere user-friendly e quanto più semplice possibile da utilizzare per un utente estraneo al progetto;
- L'applicazione AR mobile deve mantenere un peso il più possibile contenuto, sia dal punto di vista di spazio di archiviazione richiesto sia dal punto di vista di sforzo elaborativo per il dispositivo;
- Il progetto nel suo intero deve consentire facile manutenibilità;
- Il progetto deve poter permettere scalabilità futura senza che l'aggiunta di nuove possibili funzionalità non richieda stravolgimento di codice ma solo aggiunte di nuove porzioni.

4.0 PROGETTAZIONE

4.1 Premessa

Una volta definiti i requisiti funzionali e non, si procede alla progettazione del software, strutturandola in modo tale da soddisfare ognuno dei requisiti elencati nel capitolo precedente.

Il primo passo è la scelta degli strumenti da utilizzare, descritti nel capitolo *2.0 STRUMENTI* di questa tesi, e i migliori strumenti per realizzare questa applicazione sono, per esperienza personale, *Unity Engine* e *Vuforia Engine*.

Prima di procedere a descrivere come sono state realizzate le diverse parti del progetto è necessario, per consentire una maggiore comprensione di questo capitolo della tesi, introdurre qualche concetto ricorrente utilizzato tipico dell'ambiente di sviluppo Unity.

Scena: La scene sono dove si lavora con i contenuti in Unity, sono degli assets che contengono la totalità o una parte di un gioco o di un'applicazione. Per esempio, si può costruire un gioco semplice in un'unica scena, mentre per un gioco più complesso si potrebbe scegliere di utilizzare una scena per ogni livello, ognuna contenente i propri elementi, come ambiente, personaggi, eventi e interfacce. Un progetto può contenere un numero arbitrario di scene e, al momento della creazione di un nuovo progetto 3D, Unity si apre in una scena contenente soltanto una fonte di luce ed una camera.

GameObject: Il GameObject è il concetto più importante nell'Editor Unity. Ogni singolo oggetto in una scena è un GameObject, siano essi personaggi, effetti ambientali, oggetti o interfacce. Un GameObject di per sé non assolve nulla, però funge da contenitore per delle *Componenti* che implementano funzionalità. Per rendere quindi un GameObject una luce o un albero è necessario aggiungere delle componenti ad esso, a seconda quindi del risultato che si vuole creare se ne aggiungeranno diverse combinazioni. Unity offre molte componenti primitive precostruite, per esempio, una fonte di luce puntiforme semplice è creata aggiungendo la componente Light ad

un GameObject. Per ottenere oggetti con funzionalità necessarie per una specifica applicazione è necessario dunque creare le proprie componenti, in questo caso si parla di *Script* realizzati dal creatore del progetto in C#. Un GameObject senza nessuna componente ha comunque alcuni attributi basici come Trasform, contenente informazioni su posizione, rotazione e scala.

Prefab: Il sistema di Prefab in Unity permette di configurare ed immagazzinare un GameObject completo di tutte le sue componenti, di tutti i valori dei suoi attributi e di tutti i suoi GameObject figli come un asset riutilizzabile. Questo asset agisce come un template dal quale è possibile creare nuove istanze del Prefab in una scena. Questo sistema è quindi indicato quando si vuole riutilizzare un GameObject con una configurazione specifica più volte, ed è meglio del creare semplici duplicati perché andando ad agire sul Prefab si possono modificare tutte le sue istanze, ottenendo quindi sincronia tra le copie. Tuttavia, questa proprietà del Prefab non vuol dire che tutte le istanze debbano essere necessariamente identiche, è possibile effettuare l'override di una specifica istanza di un Prefab se serve che essa differisca in qualche modo dal template.

4.2 L'utente nell'ambiente virtuale

Una volta creata la prima scena è stato inserito in essa il modello 3D di Piazza del Popolo, fedelmente realizzato dallo studio del Dott. Arch. Umberto Alesi. Il modello è stato strutturato come GameObject gerarchico, ovvero un GameObject padre che contiene tutta la piazza, con ad esempio un figlio Palazzo dei Capitani, che a sua volta ha un figlio che rappresenta l'arcata d'ingresso che infine ha come figli tutte le sue componenti al dettaglio, come visibile in parte in **Fig.4** più avanti in questo capitolo.

Per dare una maggiore verosimiglianza alla scena sono state inserite due fonti di luce in essa, una direzionale dall'alto leggermente inclinata per

simulare l'illuminazione naturale del sole, ed un'altra molto più lieve onnipresente ed uniforme nella scena per garantire che anche le parti in ombra siano sufficientemente visibili per coglierne i dettagli come visibile in **Fig 3**.

Fig 3, elemento in zona in ombra con luce soffusa per i dettagli

Una volta rifinito il modello nello spazio 3D della scena è stato creato ed associato a tutte le parti del GameObject Piazza un layer, chiamato Ground. Un layer è una definizione associabile ad uno o più GameObject utile per definire collisioni selettive, richiamare gli oggetti, definire cosa viene renderizzato da una Camera o anche in che modo delle fonti di luce interagiscono con diversi Layer.

La piazza è stata quindi usata per costruire un Prefab, in modo tale da poter riutilizzare questa in più scene diverse, e comunque poterla modificare una volta sola per tutte le scene.

L'ultimo passo per concludere la stesura della piazza è la creazione dei "confini invisibili" che servono ad evitare che l'utente possa uscire dai confini della piazza o che possa arrivare in angoli non dettagliati pensati per essere visti a distanza per rafforzare l'immersione dell'utente. Questi confini sono realizzati tramite la creazione di GameObject contenenti solo la componente Collider, ovvero la componente che rappresenta la dimensione dello spazio di collisione di un oggetto, e quindi senza nessuna componente visibile. A meno che non sia esplicitamente impostato in modo contrario, un oggetto con un Collider non può passare attraverso un altro oggetto con un Collider.

Definita l'ambientazione in modo basilare è il momento di creare l'oggetto Utente, che funge appunto da personaggio controllato dall'utente con una prospettiva in prima persona. All'interno dell'utente è stato creato quindi un oggetto Corpo, per il quale come oggetto base è stato utilizzato un cilindro, le cui dimensioni sono state alterate per simulare l'altezza di una persona media. Il secondo oggetto figlio all'interno di Utente è l'oggetto Testa, un GameObject vuoto il quale Transform è utilizzato come riferimento per la prospettiva visiva in prima persona all'altezza di una testa.

A questo punto è stato creato e definito il GameObject Camera contenente due oggetti figli, la Main Camera normale utilizzata per definire la prospettiva di chi utilizza il programma desktop, e la ARCamera che che definisce appunto prospettiva, proprietà e impostazioni della camera per la realtà aumentata mobile. Queste due diverse Camera sono mutualmente esclusive e solo una di esse alla volta sarà attiva per ogni dato momento di funzionamento del programma.

Per collegare quindi Camera con la Testa di Utente ho scritto e aggiunto, a Camera come componente, un breve script che aggiorna continuamente la

posizione di Camera con la posizione di un qualsiasi GameObject scelto nell'Editor, in questo caso quindi Testa.

Ora che la telecamera si muove con il personaggio è necessario scrivere lo script, da utilizzare come componente di Utente, per consentire a quest'ultimo di muoversi. Lo script in questione, chiamato Player Movement, definisce come il Transform di Utente reagisce all'inserimento di input da parte dell'utente, in questo caso ho scelto WASD per il movimento e il movimento del mouse tenendo premuto il tasto destro per la rotazione della telecamera, definendo in base all'orientazione della camera cos'è avanti, destra, sinistra e indietro, ignorando l'inclinazione della telecamera secondo l'asse verticale. Lo script permette di definire dall'editor stesso velocità del movimento, accelerazione, decelerazione, massima angolazione della telecamera guardando verso l'alto o il basso, il tutto per rendere il movimento più realistico possibile. Inoltre questo script rende gli oggetti di cui è componente soggetti ad una forza costante che simula quella di gravità, in modo tale che il personaggio rimanga ancorato in ogni situazione al pavimento, per consentire un buon movimento su salite e discese o scalinate.

Player Movement infine limita il movimento soltanto su oggetti facenti parte del layer Ground in modo tale che Utente si può muovere solo e soltanto sugli oggetti intesi per questo scopo, ed in nessun modo può finire così a camminare ad esempio sugli oggetti aggiunti per "vestire" un'ambientazione.

Per strutturare meglio questa parte del progetto il Prefab della piazza, le luci e l'insieme dei muri Collider di confinamento sono racchiusi nel GameObject *PiazzaVR*, e quest'ultimo insieme ad Utente sono figli di *ScenaVR*, come visibile parzialmente in **Fig 4**.

Fig.4, vista gerarchica di ScenaVR (Prefab in blu)

4.3 Object Manager

Essendo il GameObject Camera, introdotto nel capitolo precedente, sempre presente e sempre attivo a prescindere da dispositivo in uso, scenario, programma VR o AR, è stato scelto come oggetto al quale dare le componenti centrali del progetto, contenenti metodi pubblici, riferimenti e la possibilità di impostare direttamente da Editor i vari collegamenti ad oggetti ed elementi dell'interfaccia (descritta nei capitoli successivi), come visibile in **Fig.5**.

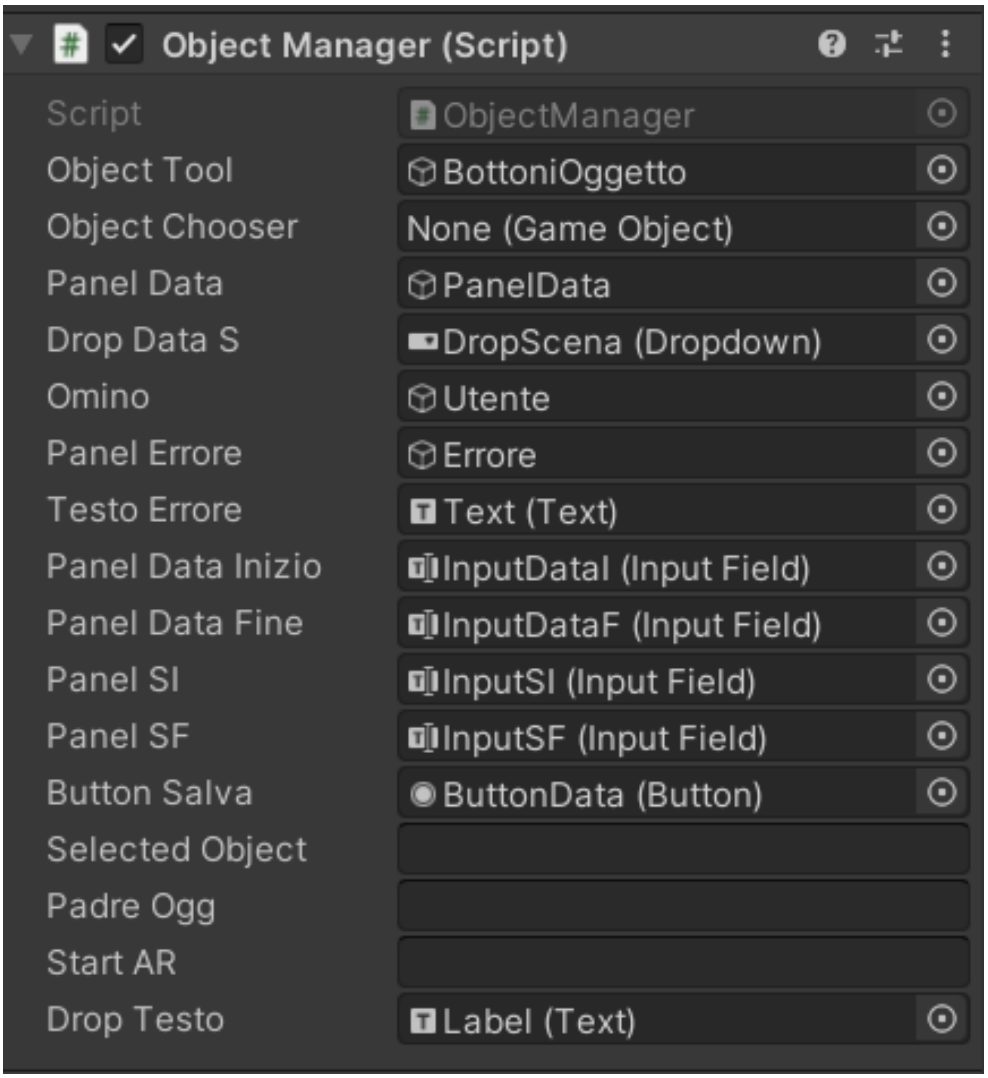

Fig.5, elementi pubblici modificabili direttamente da Editor

Alcuni degli elementi di *Object Manager* visibili in **Fig.5** sono volutamente lasciati vuoti nell'Editor e verranno automaticamente riempiti ed aggiornati a runtime, questi elementi sono molto utili per facilitare il debugging in fase di test.

Le principali funzioni, approfondite nei capitoli seguenti, contenute in Object Manager sono:

- Abilitazione del menù, solo mentre un oggetto di uno scenario è selezionato, per la rotazione degli oggetti con relative funzioni per gestire quattro gradi di rotazione associati ai relativi pulsanti;
- Gestione di tutte le comunicazioni necessarie tra stati, oggetti selezionati ed interfaccia, ad esempio una di queste funzioni scrive nell'interfaccia le date associate allo scenario corrente prendendola dal GameObject che rappresenta lo scenario, mentre un'altra modifica gli attributi rappresentanti le date sul GameObject scenario in seguito ai cambiamenti effettuati dall'utente sull'interfaccia;
- Gestione dei messaggi da mostrare a schermo in base all'evento, come messaggi di errore o di conferma;
- Definire gli elementi visibili in base allo scenario selezionato, questa funzione, visibile in **Fig.7**, è spiegata nel dettaglio nel capitolo seguente;
- Gestione del cambiamento di scena;
- Trasformazione degli oggetti che compongo lo scenario che arricchisce l'ambiente in oggetti di Realtà Aumentata, facendo in modo che nel mondo reale appaiano nella stessa posizione impostata in ambiente virtuale.

4.4 Gli scenari

Una volta impostata la Piazza è il momento di definire i primi sei scenari concordati per essa. Questi sei scenari rappresentano i Sestieri della Quintana di Ascoli e per ognuno di essi la Piazza sarà vestita con scudi rappresentanti il simbolo del sestiere in questione.

Per ogni scudo è stato costruito un Prefab contenente forma, materiale, mesh, luci indipendenti e una componente script chiamata Augment Click.

Le luci indipendenti sono necessarie per visualizzare ben illuminati gli scudi anche come elementi di realtà aumentata durante generazione e tracciamento con dispositivo mobile, essendo le luci ambientali definite nel contesto VR e quindi disabilitate nel runtime AR.

Lo script Augment Click permette al GameObject di essere riconosciuto tramite click con il mouse o tap sul touchscreen, dotando l'oggetto di un Collider che reagisce, tramite raycasting, agli input provenienti dalla Camera in uso. Gestendo tramite raycasting l'interazione è possibile bloccarla, quando necessario, facendo uso di elementi d'interfaccia interposti tra la Camera e il GameObject. Questo script quindi dona un metodo OnClick all'oggetto che a sua volta invoca una qualsiasi funzione definita dall'Editor, in questo caso specifico gli scudi invocano una funzione definita nell'Object Manager che fa apparire sullo schermo informazioni sul luogo su cui è posto lo scudo cliccato. Questo metodo inoltre definisce sull'Object Manager l'ultimo oggetto selezionato, permettendo quindi di modificarne la rotazione. Lo script infine abilita la possibilità di muovere l'oggetto tramite drag del mouse entro un raggio limite definito a priori.

Per permettere il cambiamento in tempo reale dello scudo visualizzato con il cambiamento della scena tramite interfaccia, spiegata nel capitolo seguente, ho scelto di utilizzare i Tag. I Tag sono dei Layer semplificati con meno proprietà, utilizzati unicamente per poter chiamare insiemi di oggetti che condividono un Tag con una sola chiamata.

I Prefab dei singoli scudi hanno una Tag chiamata Augment, nella scena essi sono inseriti ognuno all'interno di un GameObject avente come Tag il nome dello scenario corrispondente, e tutti e sei questi GameObject sono inseriti dentro un GameObject padre che permette quindi di muovere tutti gli scudi delle scene quando si prova a muoverne uno, essendo il programma strutturato in modo tale da vedere sempre e soltanto gli scudi relativi allo scenario corrente. Questa gerarchia per gli elementi di arricchimento

dell'ambiente è visibile in **Fig.6**, dove sono osservabili due allocamenti di scudi posti sulla facciata del palazzo Meletti, *Scudi0* e *Scudi1*, uno con la gerarchia estesa e uno con la gerarchia collassata.

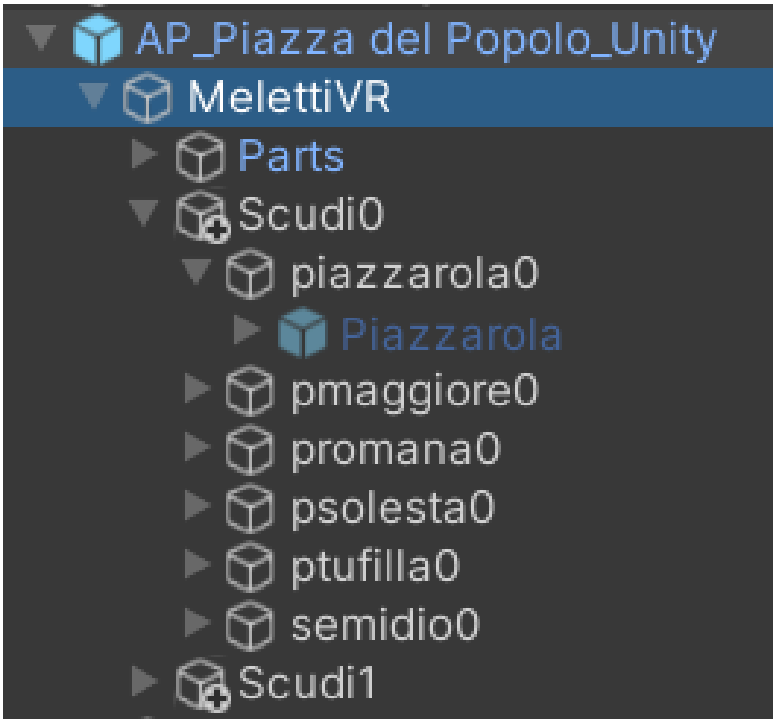

Fig.6, gerarchia degli elementi che compongono il palazzo Meletti

Per poter definire ed interagire con gli scenari sono stati creati sei GameObject, uno per sestiere, messi tutti come figli di un GameObject contenitore chiamato Scene. Ognuno dei GameObject ha il nome del sestiere, come Tag *Scena* e come componente lo script *Data Scena*.

Questo script definisce quattro attributi pubblici visibili e modificabili nell'Editor, due attributi contenenti data d'inizio e data di fine scenario e due attributi per definire e controllare lo stato dello scenario in fase di

debugging. Lo script inoltre contiene la funzione *CheckData* che, al lancio del programma e direttamente dopo il caricamento dei dati salvati (descritto in seguito nella tesi), estrae e riformatta nella stessa formattazione utilizzata nel programma la data del sistema in uso e controlla se essa è compresa all'interno del range di date definito per il GameObject, e in caso affermativo procede ad impostare nel menù a tendina degli scenari sull'interfaccia, richiamabile tramite metodi dell'Object Manager, lo scenario corrispondente e quindi chiamando la funzione legata alla scelta dello scenario. Questa funzione, chiamata *MostraS* e definita nello script Object Manager come mostrato in **Fig.7**, gestisce il cambio degli oggetti visibili al cambiare dello scenario.

```
public void MostraS()
{
    Debug.Log(dropDataS.options[dropDataS.value].text);
    GameObject[] aug = GameObject.FindGameObjectsWithTag("Augment");
    foreach (GameObject go in aug)
    {
        go.SetActive(false);
    }
    if (dropDataS.options[dropDataS.value].text != "Nessuno")
    {
        buttonSalva.interactable = true;
        panelSI.interactable = true;
        panelSF.interactable = true;
        GameObject[] oggetti = GameObject.FindGameObjectsWithTag(dropDataS.options[dropDataS.value].text);
        foreach (GameObject go in oggetti)
        {
            go.transform.GetChild(0).gameObject.SetActive(true);
        }
        GameObject scena = GameObject.Find(dropDataS.options[dropDataS.value].text);
        panelSI.text = scena.GetComponent<DataScena>().DataInizio;
        panelSF.text = scena.GetComponent<DataScena>().DataFine;
    }
    else
    {
        buttonSalva.interactable = false;
        panelSI.interactable = false;
        panelSF.interactable = false;
    }
}
```

Fig.7, funzione MostraS

Questa funzione è chiamata quando viene scelto uno scenario diverso da quello corrente, dopo aver mostrato in finestra di debug il nome dell'opzione scelta nel menù a tendina *dropDataS*, per prima cosa cerca e disabilita tutti gli oggetti con Tag "Augment" per togliere tutti gli oggetti dello scenario corrente prima di caricarne degli altri. Successivamente avviene il controllo se lo scenario selezionato è diverso da "Nessuno", lo scenario vuoto senza nessun oggetto di arricchimento dell'ambiente, nel qual caso si procede ad abilitare gli elementi di interfaccia per cambiare date e per salvare i cambiamenti, a ricercare tutti i GameObject con Tag corrispondente al nome della scelta effettuata tramite menù a tendina (ad esempio "piazzarola0" in **Fig.6**) per poi abilitarne il GameObject figlio. Successivamente si cerca il GameObject con lo stesso nome dello scenario

scelto e si impostano le date visibili e modificabili sull'interfaccia come le stesse date presenti negli attributi del GameObject trovato. In caso venga selezionato lo scenario "Nessuno" vengono disabilitati anche gli elementi dell'interfaccia relativi all'impostazione e salvataggio delle date.

La funzione *MostraS* è alla base della scelta organizzativa mostrata in **Fig.6**, in quanto è necessario disabilitare gli elementi degli scenari diversi da quello corrente ma è anche necessario poterli ricercare per riattivarli. La creazione quindi degli oggetti padre, con Tag uguale al nome dello scenario, dei Prefab da posizionare nella piazza è dovuta al fatto che non è possibile cercare tramite Tag gli oggetti disabilitati ma utilizzando questo sistema a doppio Tag è possibile usare quello dei figli per disabilitare e quello dei padri per abilitare.

4.5 Interfaccia utente

L'interfaccia grafica è ciò che permette la mutua interazione tra l'utente ed il programma, tramite rappresentazioni grafiche che permettono di eseguire comandi invece di dover interagire con un'interfaccia a riga di comando.

Il GameObject basilare per poter iniziare a definire un'interfaccia è il Canvas, ovvero una tela. Come il nome suggerisce è l'oggetto padre che definisce l'area dentro al quale è possibile "disegnare" elementi di interfaccia grafica. Nel progetto il Canvas è tracciato dinamicamente sull'intero schermo dell'utente ed esso e tutti i suoi oggetti figli sono dotati di un Graphic Raycaster. Grazie a questa componente è possibile trattare gli

elementi di interfaccia, normalmente 2D e superimposti in primo piano alla telecamera, come oggetti fisici nella scena, bloccando quindi le interazioni dell'utente con i GameObject dietro di essi. Il Canvas, visibile in **Fig.8** è strutturato gerarchicamente dove l'elemento figlio più in basso è l'elemento più in primo piano sullo schermo.

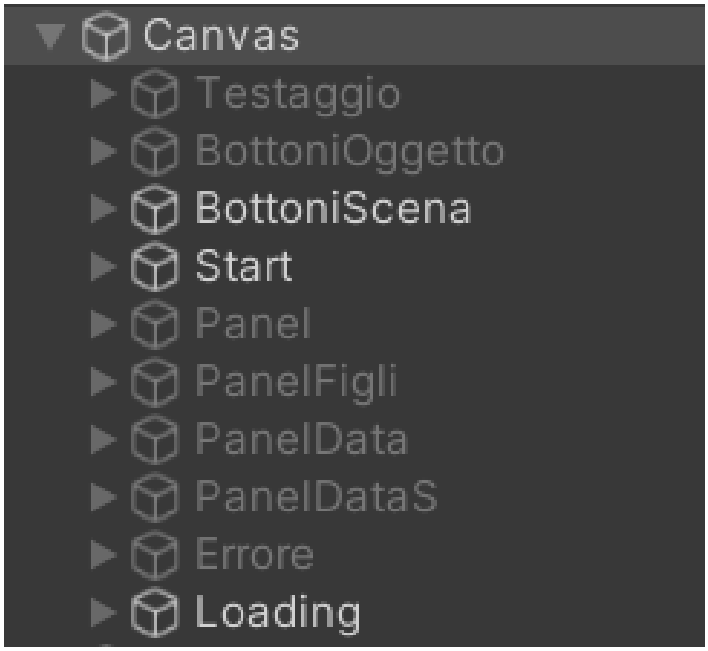

Fig.8, gerarchia dell'interfaccia all'istante di inizio runtime

Dalla **Fig.8** è possibile notare che soltanto alcuni elementi sono attivi dal primo istante di runtime del programma e come *Loading*, l'elemento più in basso nella vista gerarchica, sia l'elemento in primo piano della scena per svolgere la funzione, come il nome suggerisce, di schermata di caricamento per coprire e bloccare l'interazione con gli oggetti della scena prima che essi vengano correttamente impostati dalla fase di caricamento dati descritta nel capitolo successivo della presente.

I diversi pannelli, tra cui *Errore* e *Testaggio*, e tutti gli elementi, interattivi o meno, sono i GameObject che rappresentano tutti gli oggetti d'interfaccia che devono essere attivati soltanto a determinate condizioni. Si può

comunicare con questi elementi grazie alle definizioni pubbliche di essi effettuate tramite *Object Manager* come mostrato in **Fig.5**, in quanto, come già accennato, non è possibile altrimenti chiamare tramite nome o Tag un elemento disabilitato e per ottimizzare le prestazioni di un programma, in modo particolare se è un progetto scalabile con possibile aggiunta continua di elementi, è sempre meglio disabilitare una componente non necessaria al momento piuttosto che renderla semplicemente invisibile all'utente, specialmente in contesto mobile dove le prestazioni dei dispositivi sono molto varie e a volte davvero limitate.

Il pannello *BottoniScena*, visibile parzialmente in **Fig.9**, costituisce l'insieme degli elementi grafici dell'interfaccia sempre visibili dall'utente:

- Menù a tendina *DropScena* contenente un numero di opzioni generato dinamicamente pari a 1 + il numero degli Scenari, dove l'1 è *Nessuno*, ovvero l'opzione per non avere nessuno scenario attivo al momento, e le altre opzioni sono determinate dai GameObject aventi la Tag *Scena*, introdotti nel capitolo precedente. Questo approccio di generazione dinamica degli elementi mira a ottenere una più semplice e intuitiva manutenabilità e adattabilità a nuove ambientazioni che necessitano un numero diverso di scenari. *DropScena* effettua invoca la funzione *MostraS*, in **Fig.7**, a ogni cambiamento rispetto alla selezione corrente e al lancio del programma è impostato di base su *Nessuno*. Durante la fase di caricamento degli attributi salvati in precedenza, effettuata immediatamente dopo il lancio del programma, viene effettuata la scelta dell'opzione di *DropScena* con lo stesso nome dello scenario attivo durante la data di esecuzione, andando così effettivamente ad utilizzare il metodo associato alla selezione di un elemento di *DropScena*;
- Due coppie casella di testo-intestazione che definiscono la data di inizio e la data di fine dello scenario selezionato al momento su *DropScena*. Queste due caselle di testo sono inizializzate a ogni

invocazione di *MostraS* con gli ultimi dati correttamente salvati dall'utente, e sono liberamente modificabili dall'utente per poter modificare le date degli scenari.

- Un pulsante di salvataggio che invoca il metodo *PannelloDataAggiornaS* dell'Object Manager. Questo metodo per prima cosa controlla che l'utente abbia inserito correttamente nel formato "dd/MM/yy" delle date nelle due caselle di testo ed in caso contrario apre la schermata di errore relativa che alla chiusura si occupa di resettare i contenuti delle caselle di testo, per poi verificare che la data di inizio sia effettivamente precedente alla data di fine con relativo errore qualora fossero in ordine errato. In caso la data abbia una formattazione corretta si procede a verificare, per ogni altro elemento di *DropScena*, che la nuova data non si intersechi con una data già esistente. La sovrapposizione delle date è verificata controllando che preso A come intervallo di date nuovo per lo scenario corrente e B come intervallo di date già esistente, allora se *(InizioA < FineB) and (FineA > InizioB)* sono entrambe vere per un qualsiasi altro scenario B=/=A allora si è verificata una sovrapposizione di date e quindi verrà visualizzato a schermo il relativo messaggio di errore. A questo punto se la data è sia formattata correttamente e sia in un frangente non occupato, si procede ad aggiornare gli attributi delle date del GameObject relativo allo scenario corrente e ad invocare i metodi necessari al salvataggio online tramite *PlayfabManager*, descritto nel capitolo successivo;
- Un pulsante per ogni altra ambientazione differente, ad esempio nel caso si fosse nell'ambientazione Piazza del Popolo di Ascoli saranno visibili due pulsanti, uno per Grottammare e uno per Offida, le altre due ambientazioni al momento disponibili. Questi pulsanti si occupano di effettuare il cambiamento scena, chiudendo tutti i processi relativi alla scena corrente e caricando a partire dalla fase di *Loading* la nuova scena desiderata.

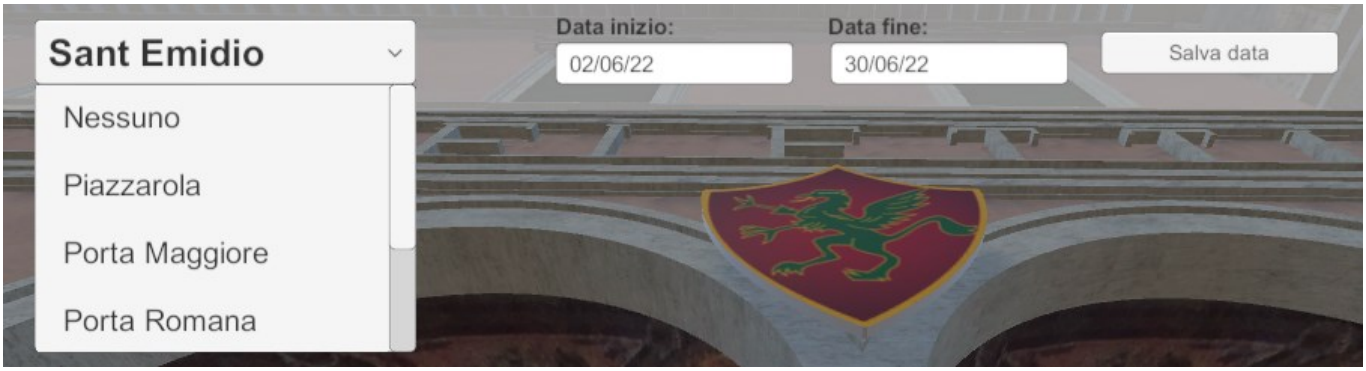

Fig.9, vista parziale di *BottoniScena* con *DropScena* esteso.

L'ultimo elemento di interfaccia di cui parlare è *Start*. Questo GameObject contiene riferimenti agli oggetti *ScenaAR* e *ScenaVR*, quest'ultimo visibile in **Fig.4**, e gestisce la scelta al lancio del programma se eseguirlo in versione virtuale o in versione di realtà aumentata. Questa scelta può essere sia effettuata automaticamente, obbligata per alcune piattaforme come quella Web, e sia lasciando la scelta all'utente. Nel caso la scelta venga effettuata automaticamente l'elemento *Start* non sarà mai visibile dall'utente ma verrà disabilitato insieme alla schermata di *Loading*, nel caso venga lasciata la scelta all'utente la schermata presenterà due pulsanti denominati AR e VR. Nel caso si scelga, automaticamente o no, VR allora verranno conservati abilitati *ScenaVR*, quindi tutti gli elementi dell'ambiente e l'*Utente*, e verrà attivato *Main Camera*, la camera che rappresenta la prospettiva del personaggio descritta nei capitoli precedenti. Nel caso si scelga, invece, AR verrà disabilitato il personaggio, verrà abilitato *CameraAR*, il GameObject che contiene la camera di Vuforia Engine, e verrà effettuata l'inizializzazione e la costruzione di *ScenaAR*, descritta nel dettaglio in un capitolo successivo.

4.6 Salvataggio e Caricamento dei dati

Il salvataggio ed il caricamento dei dati avviene sfruttando i servizi offerti da Microsoft Azure Playfab, brevemente introdotto nel capitolo 2.1.2 della presente, che oltre ad offrire gli strumenti necessari per creare le componenti utili al salvataggio e al caricamento, mette a disposizione numerosi strumenti di monitoraggio, come visibile in **Fig.10**.

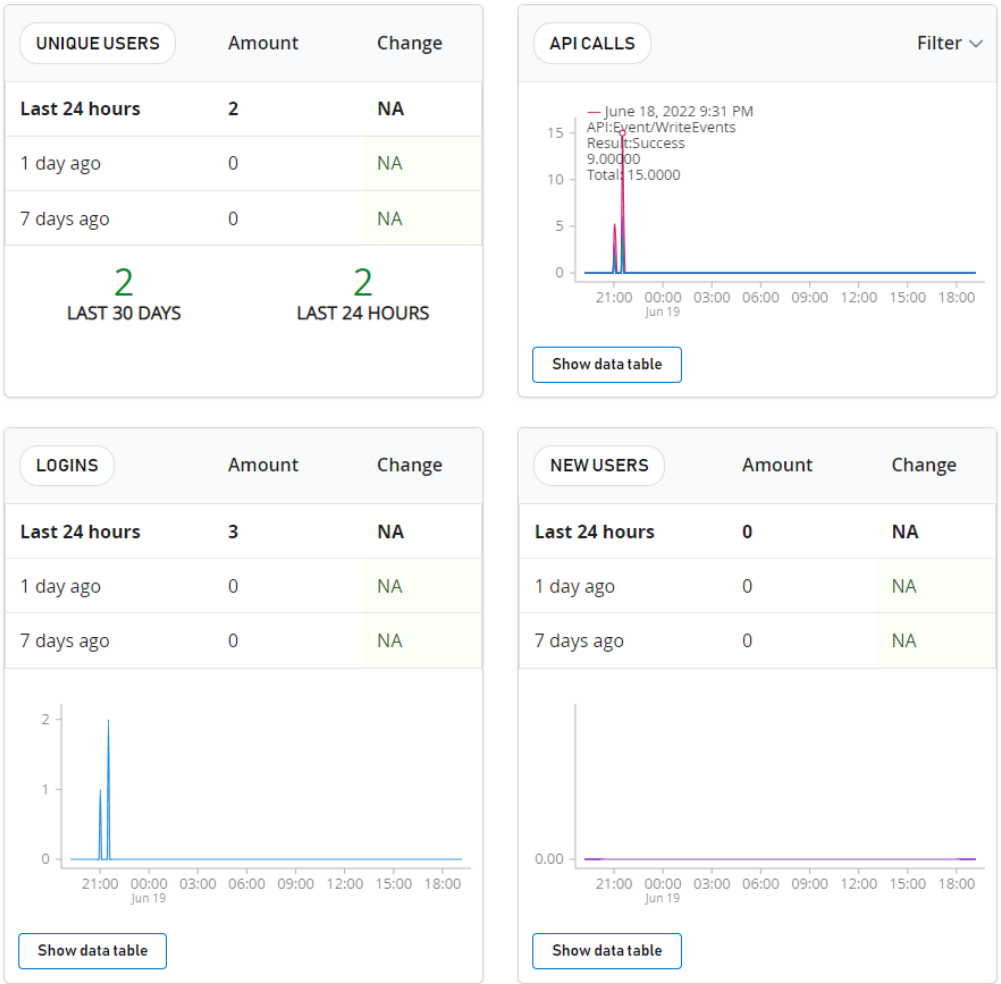

Fig.10, vista di alcuni dei servizi di monitoraggio Azure PlayFab

In modo analogo ad *Object Manager*, descritto nel capitolo 4.3 della tesi, è stata creata una componente script con attributi pubblici chiamata *PlayfabManager* assegnata al GameObject *Camera.* I suoi attributi pubblici, il quale stato è reso sempre visibile sull'Editor durante il runtime per facilitare il debugging, sono *Loggato*, che rappresenta lo stato di login, *Scena Corrente*, che indica lo scenario attivo attualmente nella scena, e *Loading*, che indica lo stato dell'omonimo pannello.

PlayfabManager si occupa di tutte le funzioni riguardanti login, salvataggio e caricamento di dati sul server e gestisce gli eventi, direttamente o tramite invocazioni di metodi pubblici, che accadono al successo o al fallimento di ognuna di queste operazioni.

Al lancio del programma viene invocata la funzione *Login*, che come suggerisce il nome si occupa di mandare la richiesta al server per effettuare un accesso, questa funzione di default effettua l'accesso con un account predefinito che condivide tutti i salvataggi effettuati durante l'utilizzo del software, ma è anche implementata, e lasciata come commento all'interno della funzione, la possibilità di effettuare un accesso unico in base ad un nuovo account o al dispositivo se si vogliono utilizzare salvataggi diversi per utenti diversi. In caso la richiesta fallisca verrà mostrato un messaggio di errore, in caso di successo si invocherà la funzione *CaricaDati*.

CaricaDati è la funzione che si occupa di richiedere al server i dati salvati dall'utente che ha effettuato il login. In caso di richiesta fallita si avrà un messaggio di errore, in caso di successo viene invocata la funzione *OnDataRecieved*.

OnDataRecieved in caso i dati ricevuti con successo siano non nulli, evento possibile ad esempio se è stato effettuato il login con nuovo utente, ricerca ogni GameObject avente Tag "Augment" e per ognuno di essi controlla se esistono dati salvati posizionali o di data. Il controllo di questi dati viene effettuato verificando l'esistenza di un dato salvato avente come nome lo stesso nome del GameObject con l'aggiunta di "posx", per i dati posizionali,

o con l'aggiunta di "dataI", per le date, visibili per maggiore chiarezza in **Fig.11**.

In caso esistano dati posizionali riguardanti l'oggetto vengono creati due Vector3, vettori da tre elementi utilizzati per definire un punto nello spazio, nel primo vengono caricati i tre valori float *posx*, *posy*, e *posz* relativi alle tre coordinate spaziali per identificare la posizione, mentre nel secondo vengono caricati in modo analogo *rotx*, *roty* e *rotz* per l'orientazione. Questi due vettori vengono quindi utilizzati per modificare i parametri Trasform di posizione e rotazione dell'oggetto.

Per quanto riguarda la data, i dati relativi a Inizio e Fine presenti sul server vanno a sostituire quelli presenti di default nell'oggetto e, successivamente, per ogni oggetto viene invocata la funzione *CheckData*, definita nella componente *DataScena* e descritta nel capitolo 4.4 della tesi. Infine, *OnDataRecieved* disabilita il pannello *Loading* rendendo quindi così visibile l'ambiente virtuale, nel caso VR, o la camera del dispositivo, nel caso AR.

In caso si decida che il programma effettui automaticamente la scelta se entrare in modalità VR o in modalità AR al lancio, *OnDataRecieved* si occupa di invocare il metodo corrispondente dalle componenti di *Start*, descritto nel capitolo 4.5.

Key	Value
PiazzaroladataF	30/10/22
Piazzaroladatal	01/10/22
Scudi0posx	-0,08245888
Scudi0posy	0,3844416
Scudi0posz	-0,01679992
Scudi0rotx	0
Scudi0roty	40,42469
Scudi0rotz	0

Fig.11, vista con attributi modificabili sul sito PlayFab dei dati salvati

La funzione di *PlayfabManager* che si occupa del salvataggio dei dati è *SalvaDati*. Questa funzione è invocata da *PannelloDataAggiornaS*, descritta nel capitolo 4.5, che a sua volta è chiamata alla pressione del pulsante di salvataggio nell'interfaccia utente. *SalvaDati* effettua un procedimento inverso al caricamento effettuato con *OnDataRecieved*, andando ad inviare al server i valori degli attributi posizionali, rotazionali e di data, e a salvarli con un nome formato dall'unione del nome del GameObject e un descrittore, come ad esempio "PiazzaroladataF", visibile in **Fig.11**, contiene il valore dell'attributo *Data Fine* del GameObject *Piazzarola*.

4.7 La realtà aumentata

Nel progetto sono stati utilizzati i Model Target e gli Image Target, descritti nel capitolo 2.2.1 e 2.2.2 della presente.

Per poter utilizzare un qualsiasi Target è necessario utilizzare un database Unity costruito nella sezione Developer dal sito di Vuforia.

Il GameObject principale per le funzionalità AR è *CameraAR*. Questa componente deve essere l'unica camera attiva durante l'esecuzione e svolge tutte le normali funzioni della camera della scena oltre a contenere le componenti di Vuforia Engine che permettono alla camera di non renderizzare i Target, ovvero di non mostrare all'utente le immagini e i modelli 3D che fungono da bersagli di ricerca anche se sono attivi nella scena digitale, e di riconoscere e tracciare quest'ultimi nel caso si riconoscano un numero sufficiente di features nell'ambiente reale.

Per poter conservare scala e posizione degli oggetti della scena, oltre a garantire migliori prestazioni di riconoscimento delle features reali, è necessario che sia i modelli 3D dell'ambiente virtuale e sia i GameObject Target abbiano la stessa scala dell'ambiente reale, la stessa posizione rispetto agli altri elementi che compongono la scena e, ovviamente, siano il più fedele possibile alla loro controparte reale. In **Fig.12** si può vedere la sovrapposizione di un GameObject ImageTarget realizzato utilizzando foto della facciata con il modello 3D del Palazzo dei Capitani.

Fig.12, sovrapposizione ImageTarget – Modello 3D

CameraAR di default non gestisce le interazioni tramite click o tocco sul touchscreen, per questo ho aggiunto ad esso come componente lo script *OnTouchDown*. Questo script si occupa di effettuare un raycast al tocco/click dell'utente, il quale permette l'interazione con i GameObject in caso impatti con un Collider.

Effettuando test dal vivo nell'area ho verificato che in base alla parte inquadrata e alla posizione dell'utente rispetto agli oggetti, un tipo di Target ha delle prestazioni nel riconoscimento migliori dell'altro. Quindi per alcuni elementi sono presenti sia l'ImageTarget che il Model Target dello stesso

oggetto, con una funzione che in caso venga riconosciuto uno venga interrotta la ricerca dell'altro.

Affinchè un GameObject possa essere utilizzato come oggetto di realtà aumentata da far apparire al riconoscimento di un elemento dell'ambiente reale, è necessario che esso sia figlio del GameObject Target nella gerarchia della scena. Essendo il sistema di coordinate di un GameObject relativo alla posizione del padre e non alla scena in assoluto è necessario per poter gestire la posizione degli oggetti tramite serializzazione dei suoi attributi, che essi siano posizionati nella scena virtuale al momento del caricamento dei dati dal server in quanto, se aprendo il programma in modalità AR gli Augment venissero caricati direttamente posizionati come figli dei Target, si avrebbe inconsistenza rispetto ai cambiamenti salvati dall'utente in modalità VR.

È stato quindi implementato un sistema che prima carica gli oggetti in ambiente virtuale e poi ne cambia il padre nella gerarchia. Per gestire questo sistema i GameObject che devono fungere da bersaglio e da riferimento nel mondo reale hanno Tag "Padre", lo script componente *ObjectHighlighter*, tutte le componenti figlio del Prefab raggruppate in un GameObject *Parts*, e il suffisso -VR alla fine del nome, come visibile in **Fig.6**. Dall'immagine si può notare *MelettiVR*, il GameObject contenente tutte le parti del palazzo Meletti, che ha come figli gli oggetti da mostrare in realtà aumentata e *Parts*, il GameObject padre di tutte le parti del palazzo. I GameObject dei Target hanno lo stesso nome del Padre corrispettivo ma senza il suffisso -VR.

All'inizializzazione del programma in modalità AR viene effettuata la ricerca di tutti i GameObject con Tag "Padre" e per ognuno di essi viene invocato il metodo *SpostaFigli* definito in *ObjectHighlighter*. *SpostaFigli* controlla se il GameObject ha dei figli, nel qual caso procede ad individuare, e a salvare tramite *Object Manager*, il GameObject avente lo stesso nome fatta eccezione per i due caratteri finali (VR). A questo punto viene effettuato un ciclo iterativo che, per ognuno dei figli non chiamato

“Parts” , cambia il padre nella gerarchia con il GameObject salvato in precedenza.

Questo procedimento semplifica notevolmente sia l’aggiunta di nuovi oggetti che la modifica degli attuali anche a livello di editor, permettendo di fissare gli elementi degli scenari soltanto nella parte contenuta in *ScenaVR*, senza dover quindi fissare a mano la controparte AR ed eliminando del tutto la componente di errore umano nella realizzazione del parallelismo.

Per minimizzare il peso in versione mobile è quindi possibile limitare gli elementi di *ScenaVR* buildati nel programma ai soli elementi necessari a fissare gli Augment, riducendo quindi notevolmente il peso dell’applicazione.

5.0 CONCLUSIONI

L'obiettivo di questo lavoro è stato quello di realizzare e di descrivere nel modo più esauriente possibile la progettazione di un software per la fruizione di contenuti in realtà virtuale ed in realtà aumentata, partendo da dei requisiti fissati dall'azienda che mi ha proposto questo progetto.

L'idea di integrare in modo dinamico l'ambiente reale con la realtà aumentata ed un ambiente completamente digitale che ricostruisce quello reale è stata, a mio parere, molto interessante sia dal punto di vista innovativo che dal punto di vista realizzativo, essendo una sfida concreta in quanto non esistono, perlomeno disponibili pubblicamente, progetti simili dai quali attingere idee per definire la struttura del software.

In particolare il progetto è stato strutturato dall'inizio focalizzandomi principalmente sui requisiti di adattabilità e portabilità. L'adattabilità in particolare è il fulcro centrale di tutta la composizione gerarchica dei GameObject e della scelta di utilizzare l'*Object Manager* per gestire quelli che sono principalmente riferimenti e invocazioni sempre mirate ad oggetti generalizzati, classi costruttrici o ad oggetti con l'unica funzione di essere gerarchicamente Padri. Non posso fornire un indice obiettivo per descrivere la qualità dell'adattabilità, tuttavia posso con certezza dire che il requisito è stato soddisfatto, in quanto in principio il programma è stato costruito avendo disponibile soltanto l'ambientazione Piazza del Popolo di Ascoli e solo in un secondo momento, dopo aver già ultimato la struttura finale del software, mi sono state fornite le altre due ambientazioni per Offida e Grottammare e implementarle si è rivelato un processo fluido e rapido.

La portabilità ha portato vincoli di Engine ed in misura minore di struttura. Per l'engine bisogna utilizzare elementi elaborativi, come gli shader, le librerie, come ES OpenGL 3, la gestione della definizione dello spazio dei colori gamma o lineare, ed impostazioni di editor compatibili con tutti i dispositivi. Il vincolo strutturale è dato dalla portabilità web in quanto WebGL non supporta Vuforia Engine ed è quindi necessario che ogni elemento Vuforia sia facilmente escludibile dalla build e senza ripercussioni

sulla parte virtuale. Infine, per il porting per Oculus è necessario definire una nuova Camera specifica ed una differente componente di gestione di movimento e di input.

Il soddisfacimento dei requisiti non funzionali è stato quindi conseguenza diretta della strutturazione focalizzata sull'adattabilità, permettendo di ottenere un software dalla facile manutenzione e scalabilità.

5.1 Sviluppi futuri

Due possibili sviluppi futuri per ampliare il programma sono emersi continuando ad interfacciarmi con l'azienda ospite durante il tirocinio, e durante la finalizzazione del progetto sono stati tentati approcci ed effettuate ricerche sull'effettiva realizzabilità di essi: l'inserimento durante l'esecuzione del programma di nuovi modelli 3D da parte dell'utente come parte degli scenari e la definizione, sempre a runtime, di nuovi Target direttamente dall'utente nell'ambiente reale simulato.

Entrambi queste richieste non sono al momento attuabili per limitazioni tecnologiche in quanto, per la prima richiesta, seppur sia possibile caricare modelli 3D con tutte le loro componenti a runtime, non è possibile poi serializzare e quindi conservare tra diverse esecuzioni del programma, indifferentemente se in contesto VR o AR, questi nuovi GameObject.

Per quanto riguarda la seconda richiesta al momento Vuforia Engine richiede che i Target vengano elaborati sulla propria piattaforma separata da Unity Engine e al momento non è possibile effettuare questo processo né da Editor e tantomeno a runtime.

RINGRAZIAMENTI

Finalmente sono giunto al capitolo più semplice della tesi, un capitolo non dedicato a me ma a tutti quelli che hanno fatto sì che io sia qui a scrivere le parole finali di un testo che rappresenta il coronamento di un percorso di studi.

Per primi vorrei ringraziare in particolare il mio relatore, il professor A. F. Dragoni, il mio correlatore, l'Ing. Corradetti, e Beesoft per avermi proposto questo progetto e indirizzato durante tutto il suo svolgimento.

Un sentito ringraziamento anche a Paolo, Selene e Nicola sempre presenti nelle riunioni dell'AIRTLab e pronti a dare consigli e opinioni.

Ovviamente un ringraziamento alla mia famiglia, mio fratello, mia sorella ed in particolare mia madre, che mi hanno continuato a sostenere per tutta la durata del mio percorso di studi di "5 anni".

Ultimi solo di ordine, vanno ringraziati tutti gli amici di sempre, che nonostante la distanza non sono mai stati davvero distanti, gli amici incontrati all'università, che hanno reso indubbiamente più belli questi anni, e gli amici sempre presenti online, senza i quali sicuramente mi sarei laureato prima.

Un ringraziamento in particolare anche a Nespola per il montaggio del video di presentazione, a Jeorger per l'aiuto in fase di revisione e a Matgre, Giammix e Pinello per aver contribuito alla fase di test su campo.

Grazie a tutti,

Angelo "Elindor" Serafini