ALMA MATER STUDIORUM · UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA - SCUOLA DI SCIENZE

Corso di Laurea in Scienze e Tecnologie Informatiche

Un'applicazione di realtà virtuale mediante tecnologia immersiva Oculus

Relazione finale in

Metodi numerici per la grafica

Relatore

Dott.ssa Damiana Lazzaro

Presentata da

Marika Mandolini

III SESSIONE

Anno Accademico 2013-2014

"La fantasia è più importante della conoscenza,

perché la conoscenza ha i suoi limiti"

- A. Einstein

Introduzione

L'evolversi della tecnologia ha introdotto l'uso di nuovi strumenti che permettono, all'utente che li utilizza, una vera e propria immersione in una realtà virtuale realizzata al calcolatore.

Si parla di nuove tecnologie, quindi, dispositivi che sono in grado di immergerci totalmente in una realtà virtuale. Tali dispositivi comprendono caschi con visori stereoscopici, speciali guanti per interagire manualmente con le componenti tridimensionali e rilevatori di movimento come dispositivi di motion tracking ed head tracking.

Nella tesi in questione si vuole illustrare l'impiego del visore "Oculus DK2" per la realizzazione di un'applicazione grafica interattiva. Una volta indossato il visore, l'utente della suddetta applicazione si troverà all'interno di un museo nel quale potrà muoversi, guardarsi attorno e interagire con i vari oggetti in scena.

L'intero elaborato è stato svolto all'interno della struttura aziendale "Studio Rossetti", che ha contribuito nella realizzazione del progetto fornendo l'idea e i materiali necessari per lo sviluppo.

L'elaborato è stato sviluppato su Unity 3D, un software nato per lo sviluppo di videogiochi 2D/3D. Quest'ultimo essendo un software multipiattaforma è supportato dai vari sistemi operativi: Microsoft, Mac, Linux, iOS, Android, ecc.

La tesi è strutturata in quattro capitoli:

- Il primo capitolo parla della realtà virtuale e fornisce una descrizione generale del progetto partendo dall'idea;
- Nel secondo capitolo sono descritte tutte le tecnologie utilizzate per lo sviluppo dell'elaborato;

- Il terzo capitolo è dedicato alla spiegazione dei vari metodi che permettono il funzionamento dell'applicazione;
- Nell'ultimo capitolo vengono mostrati degli screenshoot dell'elaborato in esecuzione per capire al meglio il suo utilizzo e sono accennati alcuni dei possibili sviluppi futuri.

Indice

Introduzione I

Indice III

Lista delle Figure VII

1. Realtà virtuale e introduzione all'applicazione 1

1.1 Realtà virtuale 1

1.1.1 Alcune definizioni 2

1.1.2 I campi di utilizzo della realtà virtuale 3

1.1.3 Dispositivi che ingannano i cinque sensi 3

1.1.4 Altri dispositivi 5

1.1.5 Realtà virtuale su mobile 6

1.2 Introduzione all'applicazione 8

1.2.1 Descrizione generale dell'applicazione 8

1.2.2 Oggetti in scena 10

1.2.3 Ambiente grafico 11

1.2.4 Tecnologie impiegate per lo sviluppo 12

2. Unity3D e OculusDK2 15

2.1 Unity3D 15

2.1.1 Le peculiarità di Unity3D 15
2.1.2 L'interfaccia di Unity3D 16
2.1.3 Il progetto di Unity3D 19
2.1.4 Lo scripting di Unity3D 20
2.1.5 Le animazioni 23
2.1.6 I suoni 25
2.1.7 Il nuovo sistema di interfacce 25
2.1.8 Assets e Assets Store 28
2.1.9 Il Package OVR 29
2.2 OculusDK2 32
2.2.1 Caratteristiche tecniche 32
2.2.2 Come funziona OculusDK2 33

3. Implementazione del progetto 35
3.1 Osservatore e Oggetto interattivo 35
3.2 Le variabili esposte 38
3.3 L'operazione di intersezione "RayCast" 38
3.4 Livelli in scena 41
3.5 Lo script "OggettoInquadrato.cs" 43
3.5.1 Struttura Generale 43
3.5.2 Intersezione con il layer "OggettoInterattivo" 46
3.5.3 Intersezione con il layer "Exit" 51
3.5.4 Intersezione con il layer "Percorso" 54
3.5.5 Inersezione con il layer "Animation" 56

3.5.6 Intersezione con il resto della scena 58

3.5.7 Attivazione e disattivazione delle pedane di movimento 58

3.5.8 Animazioni di caricamento 59

3.6 Lo script "OpenLabel.cs" 61

3.6.1 Assegnazione dello script "OpenLabel.cs" 61

3.6.2 Struttura generale 62

3.7 Lo script "MovePlayer.cs" 68

3.7.1 Struttura generale 68

4. Risultato finale e sviluppi futuri 71

4.1 Applicazione in esecuzione 71

4.2 Possibilità di utilizzo 80

4.3 Sviluppi Futuri 80

Ringraziamenti 83

Bibliografia e Sitografia 87

Lista delle figure

1.1 Visori per realtà virtuale 4

1.2 Esoscheletri robotici 4

1.3 Pedana Virtux Omni 5

1.4 Leap Motion 5

1.5 Project Morpheus 6

1.6 Google Cardboard 6

1.7 Dive 7

1.8 Samsung GearVR 7

1.9 Esempio dell'applicazione in esecuzione 9

1.10 Oggetto con descrizione 10

1.11 Pedana di movimento 10

1.12 Pulsanti di interfaccia 11

2.1 Interfaccia Grafica di Unity3D 16

2.2 Esempio di compoment nell'area Inspector 19

2.3 Esempio di uno script vuoto con le funzioni Start() e Update() 21

2.4 Il metodo SendMessage() 22

2.5 Animazione composta da una traslazione e una rotazione 23

2.6 Component Animator 24

2.7 State Machine 24

2.8 Component AudioSource 25

2.9 Esempio del nuovo sistema di interfaccia 25

2.10 Asset Store 28

2.11 PackageOVR 29

2.12 OVRPlayerController in scena 30

2.13 OculusDK2 32

2.14 Le due immagini relative ai due schermi 34

3.1 La compoment script "OggettoInquadrato" 36

3.2 La component script "OpenLabel" 36

3.3 La component script "MovePlayer" 37

3.4 Variabili esposte 38

3.5 Raycast: esito positivo 39

3.6 Raycas : esito negativo 40

3.7 Esempio assegnazione layer 42

3.8 Animazione sul mirino 46

3.9 Icone delle animazioni 59

3.10Animazione di "Loading" 59

3.11 State Machine relativa al mirino 60

3.12 La component "OpenLabel.cs" 61

4.1 Ambiente virtuale realizzato 72

4.2 Selezione pedana di movimento 73

4.3 Selezione di un oggetto con descrizione 74

4.4 Attivazione di un oggetto 75

4.5 Selezione del pulsante di chiusura 76

4.6 Selezione del pulante di animazione 79

4.7 Esempio di Shader di Outline 80

Capitolo 1

Realtà virtuale e introduzione applicazione

Il primo capitolo tratta della realtà virtuale (VR, Virtual Reality) e di come essa si inserisce all'interno del progetto realizzato in questa tesi. Viene introdotta l'applicazione descrivendone funzionamento e l'ambiente grafico realizzato.

1.1 Realtà virtuale

Sempre più spesso si sente palare di realtà virtuale e quasi tutte le più grandi aziende del settore elettronico/informatico si stanno attrezzando per supportare questo nuovo tipo di tecnologia, ma cos'è la virtual reality?

1.1.1 Alcune definizioni

Dare una definizione unica e precisa di realtà virtuale non è affatto semplice in quanto è un termine aperto a diverse interpretazioni, sia scientifiche che tecnologiche.
Per questo è opportuno riportare alcune definizioni di realtà virtuale date da esperti del settore.
Lo scienziato informatico Frederick Phillips Brooks, Jr. definisce l'esperienza di "realtà virtuale" come un'esperienza in cui l'utente è effettivamente immerso in un mondo virtuale , e questo implica un controllo dinamico del punto di vista da parte dell'utente [1].
Il Professor Grigore C. Burdea ed il ricercatore Philippe Coffiet scrivono nel loro libro che la "realtà virtuale" è una simulazione in cui la grafica 3D viene utilizzata per creare un mondo realistico, quindi un mondo sintetico non statico che risponde all'input dell'utente (es: gesto, comando vocale, ecc.) [2].
Secondo la società IBM la "realtà virtuale" è un'interfaccia uomo-computer che consente all'utente di sperimentare un ambiente di sintesi interattivo e tridimensionale. Questo mondo artificiale contiene oggetti e suoni atti a simulare il loro corrispettivo nel mondo reale. L'utente che può agire sull'ambiente virtuale circostante sprofonda in un ambiente realistico dal quale nasce l'esperienza di immersione [3].
Lo scrittore Nicholas Negroponte sostiene che l'idea che sta alla base della "realtà virtuale" è dare la sensazione di essere "sul posto" presentando alla vista tutto ciò che si vedrebbe realmente, e modificando la scena istantaneamente quando cambia il punto di vista[4].

1.1.2 I campi di utilizzo della realtà virtuale

Sebbene di primo impatto si potrebbe pensare che la realtà virtuale possa essere oggetto solamente di videogiochi, allora ci si dovrebbe ricordare che quest'ultima viene utilizzata in diversi ambiti:

- Architettonico: si utilizza la realtà virtuale per mostrare al cliente il progetto finale di un edificio.
- Medico: in questo caso la realtà virtuale viene sfruttata per effettuare simulazioni chirurgiche.
- Aereo: si utilizza la realtà virtuale per fare delle simulazioni di volo a scopo istruttivo.
- Psicologico: ci si avvale della realtà virtuale per la riabilitazione di persone con deficit cognitivi.
- Turistico/Culturale: la realtà virtuale viene sfruttata per effettuare visite interattive a siti archeologici o beni culturali.

Il campo di sviluppo dell'applicazione trattata in questa tesi si orienta verso il turistico/culturale in quanto si ambienta all'interno di una stanza di un museo.

1.1.3 Dispositivi che ingannano i cinque sensi

Per godere della realtà virtuale una persona necessita di alcuni strumenti che permettono di ingannare i sensi umani.

La vista è sicuramente il primo dei sensi che deve essere raggirato, in quanto l'utente deve poter osservare un luogo differente da quello in cui si trova realmente.

Per fare ciò sono stati creati dei visori, cioè dei particolari display che possono essere indossati come se fossero una maschera da sub.

Fig 1.1 - Visori per realtà virtuale

Questi dispositivi sono dotati di lenti speciali che permettono una visione corretta di un display anche se posizionato ad una distanza di appena tre centimetri dagli occhi.
Nella tesi in questione è stato utilizzato il più comune dei visori, quello prodotto dall'azienda americana Oculus VR [5].
Un'altro senso che dovrebbe essere ingannato è l'udito. Questo può essere fatto facilmente tramite un paio di comuni cuffie che permettono di isolare i suoni provenienti dal mondo reale e allo stesso tempo di riprodurre suoni scaturiti dal mondo virtuale.
Per simulare un terzo senso, il tatto, sono stati creati dei "guanti robotici", più precisamente "esoscheletri robotici" che danno la possibilità di sentire quello che sta succedendo sulla propria pelle. Questi guanti permettono quindi di capire se nella realtà riprodotta si sta sfiorando una superficie liquida o solida. Gli esoscheletri in questione sono i "Dexmo" ideati dalla compagnia cinese Dexta Robotics [6]. Nell'applicazione trattata in questa tesi non è prevista l'illusione del tatto, pertanto non è stato utilizzato nessun tipo di esoscheletro robotico.

Fig 1.2 - Esoscheletri robotici

Gli strumenti che sono in grado di raggirare i restanti due sensi, quindi gusto e olfatto, sono ancora in fase di studio e sviluppo.

1.1.4 Altri dispositivi

Oltre ai dispositivi sopra citati ne esistono altri che non ingannano i cinque sensi umani, ma se utilizzati per l'esperienza virtuale consentono di renderla ancor più realistica. Introduciamo alcuni dei dispositivi che permettono di rendere il movimento all'interno del mondo sintetico il più naturale possibile.

La pedana Virtuix Omni [7] è un tapis roulant passivo che permette di spostarsi all'interno del modo virtuale semplicemente camminando, quindi senza joypad o tastiera.

Fig 1.3 - Pedana Virtux Omni

Il Leap Motion [8] è un dispositivo che tramite un sistema a raggi infrarossi consente di leggere il movimento delle mani. Questo strumento se utilizzato con Oculus DK2 [9] consente di tener traccia del movimento delle mani all'interno della realtà virtuale.

Il Leap Motion a differenza del Virtux Omni è stato oggetto di studio per questa tesi.

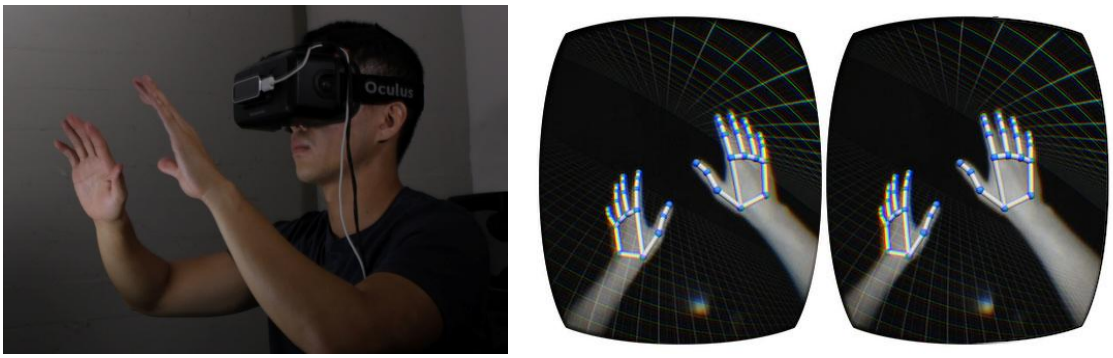


Fig 1.4 - Leap Motion

1.1.5 Realtà virtuale su Mobile

Sebbene si è parlato di Oculus DK2 come strumento in grado di ingannare la vista e portare l'essere umano in un mondo sintetico è giusto dire che non è l'unico visore presente sul mercato. Anche Sony ha da poco presentato alla Game Developers Conference di San Francisco il prototipo del proprio dispositivo di realtà virtuale: Project Morpheus [10].

Fig 1.5 - Project Morpheus

Un limite di entrambi i visori citati è sicuramente il cablaggio, che rende il movimento del corpo dell'utilizzatore un po' impacciato.
Recentemente sono stati ideati dei visori senza fili che hanno risolto il problema appena descritto. Va detto che questi visori senza fili funzionano con dispositivi mobile (telefoni cellulari) e quindi la qualità dell'esperienza è ridotta.
La qualità dei diversi "visori wireless" prodotti dalle diverse case costruttrici varia a seconda del prezzo.
Google dà la possibilità a chiunque di possedere il proprio dispositivo di realtà virtuale con appena 4$. I Google Cardboard [11] non son altro che una maschera di cartoncino dotata di lenti, dentro la quale può essere posizionato uno smartphone. Avviando una specifica applicazione di realtà virtuale,è possibile godersi l'esperienza osservando lo schermo attraverso le lenti.

Fig 1.6 - Google Cardboard

Una soluzione più sofisticata di "visore wireless" è il Dive [12], un dispositivo che sostanzialmente ha lo stesso funzionamento dei Cardboard con l'unica differenza che le lenti possono esser spostate per regolare la messa a fuoco. Il Dive è sul mercato ad una modica cifra di 89$.

Fig 1.7 - Dive

Con Samsung Gear VR [13] la famosa azienda sudcoreana Samsung in collaborazione con Oculus VR ha prodotto il miglior dispositivo di realtà virtuale wireless presente sul mercato fino ad oggi. Infatti Samsung Gear VR ha prezzi molto più elevati dei dispositivi prima citati, si parla di cifre che si aggirano attorno ai 199$ a cui deve essere aggiunto il costo del phablet Galaxy Note 4 [14] che è l'unico dispositivo compatibile con il visore.

Fig 1.8-Samsung Gear VR

1.2 Introduzione all'applicazione

Dopo aver definito la realtà virtuale, parlato dei suoi campi di sviluppo e di applicazione ed elencato gli strumenti che ci portano ad ottenerla, è opportuno parlare dell'applicazione di realtà virtuale in questione.

1.2.1 Descrizione generale dell'applicazione

L'obbiettivo principale di questa tesi è utilizzare la tecnologia Oculus e l'innovativo sistema di interfacce proposto da Unity3D versione 4.6 Beta [15] per progettare un'applicazione di tipo culturale che permetta ad un utente, immerso all'interno di un ambiente grafico realizzato, di coglierne delle informazioni grazie ad un pannello che descrive l'oggetto osservato.
Si tratta quindi di un prototipo di museo virtuale in stile moderno dove l'osservatore, oltre a contemplare la scena, con un semplice sguardo verso un oggetto di interesse è in grado di aprire un pannello a mo' di leggio sul quale vengono scritte le informazioni che lo riguardano.
Una volta aperto, il pannello può essere richiuso tramite lo stesso meccanismo: l'utente dovrà ruotare il capo sino a soffermarsi sul pulsante di chiusura.
Alcuni oggetti in scena, oltre ad essere dotati di descrizione, sono muniti di suono e animazione.
Fino ad ora si è parlato esclusivamente della selezione di un oggetto, ma per poter osservarlo al meglio si ha la necessità di avvicinarsi ad esso. L'applicazione fornisce la possibilità di utilizzare due diversi metodi di spostamento in scena da parte dell'utente: il primo con il classico metodo di selezione tramite sguardo rivolto verso appositi oggetti per il movimento e il secondo tramite joypad, tastiera, o "Arduino".
L'operazione di selezione tramite il movimento del capo è il fulcro dell'applicazione, per questo viene utilizzato un mirino che ha la funzionalità di "cursore" e serve a guidare l'utente durante l'operazione. L'introduzione del mirino rovina un po' l'esperienza, ma è stata fatta questa scelta perché è stato ritenuto di fondamentale

importanza: la selezione senza questo accorgimento risulta difficile perché troppo dispersiva.

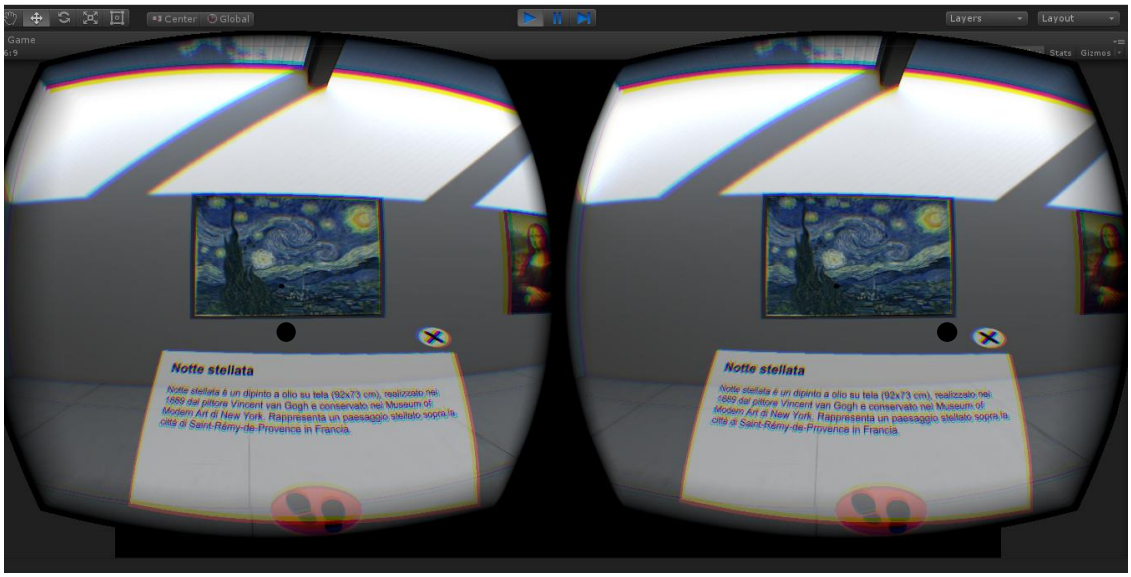


Fig 1.9- Esempio dell'applicazione in esecuzione (notare il mirino al centro).

1.2.2 Oggetti in scena

Solo alcuni degli oggetti del museo virtuale sono degli oggetti interattivi e si possono raggruppare in tre tipologie:

1. oggetto con descrizione: è quell'oggetto dotato di un nome, una descrizione, e in certi casi di suono o animazione, se questo oggetto viene selezionato tali proprietà saranno visibili all'utente;

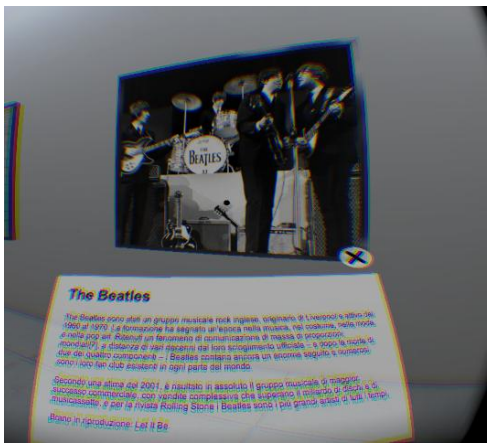


Fig1.10 - Oggetto con descrizione.

2. pedana di movimento: è quell'oggetto che se selezionato permette di spostare l'utente fino a quel punto;

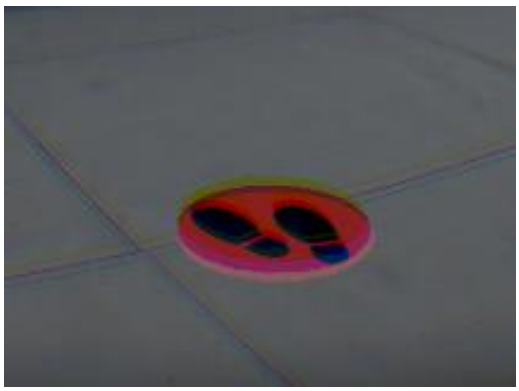


Fig 1.11 - Pedana di movimento.

3. pulsanti di interfaccia: sono due pulsanti visibili solamente all'apertura del pannello di descrizione, uno permette di chiuderlo, e l'altro permette di attivare l'animazione dell'oggetto in questione, se ne è dotato.

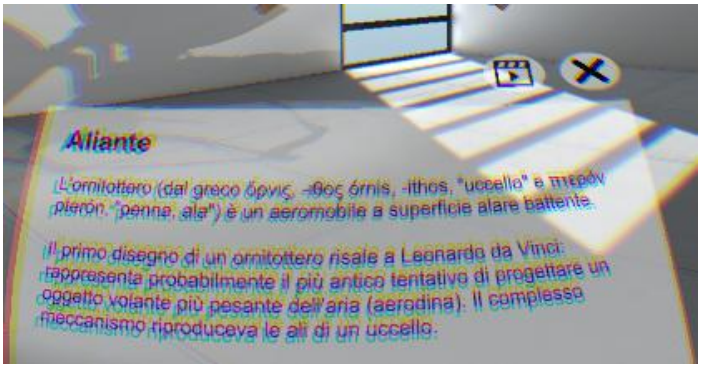


Fig 1.12 - Pulsanti di interfaccia.

1.2.3 Ambiente grafico

In un'applicazione di realtà virtuale l'ambiente grafico scelto gioca un ruolo fondamentale. Affinché si abbia la sensazione di "immersione" all'interno del mondo sintetico, esso deve essere il più possibile simile al mondo reale. Ogni particolare è di fondamentale importanza partendo dall'illuminazione, dal modello 3D di ogni singolo oggetto e dai materiali assegnati.

La scena ed ogni oggetto in essa presente è stata modellata mediante l'editor grafico 3DStudioMax [16] di proprietà Autodesk e successivamente è stata importata in Unity3D [15] nel quale sono stati assegnati i materiali e disposta l'illuminazione.

La scena non è stata modellata direttamente in Unity3D, perché il suo editor grafico non è abbastanza completo da poter creare degli oggetti complessi.

Unity3D è dotato di un proprio sistema di illuminazione. Una volta disposte in scena le luci, l'illuminazione viene calcolata dinamicamente durante l'esecuzione dell'applicazione.

Come per le luci, Unity3D ha i propri materiali che assegnati ai vari oggetti permettono di renderli più realistici.

Come già accennato in precedenza, la scena realizzata è una stanza di un museo in stile moderno, nella quale si possono distinguere tre aree principali:

1. Area dei quadri classici : formata da oggetti interattivi che al momento della selezione forniscono solamente una descrizione.

2. Area dei poster musicali: formata da oggetti interattivi che al momento della selezione, oltre a fornire delle informazioni, emettono anche un suono.
3. Area della scultura: formata da un oggetto interattivo che al momento della selezione emette un suono, un'informazione ed un'animazione.

1.2.4 Tecnologie impiegate per lo sviluppo

L'elaborato è stato sviluppato con il software Unity3D che è un software dedicato allo sviluppo di videogiochi [15].
E' stata fatta questa scelta perché è abbastanza semplice integrare il dispositivo Oculus in esso, inoltre Unity3D è un software abbastanza diffuso ed è stato facile trovare la documentazione necessaria per imparare ad utilizzarlo.
L'altra tecnologia usata è Oculus DK2 [9]. E' stato scelto questo particolare visore principalmente perché l'azienda presso la quale è stata svolta questa tesi possedeva già questo dispositivo, ma anche perché è il visore più conosciuto.
Durante lo sviluppo del progetto si è pensato di utilizzare anche il Leap Motion [8]. Come accennato precedentemente, questo è un dispositivo che permette di catturare i movimenti delle mani. Si era pensato di sfruttare questa tecnologia per riconoscere eventuali "gesture" fatte dall'utente. L'applicazione avrebbe dovuto rispondere in base al gesto fatto. Non è stato possibile inserire la tecnologia Leap Motion per problemi di incompatibilità con la versione di Unity3D utilizzata al momento dello sviluppo.
L'applicazione prevede anche l'utilizzo opzionale del Makey Makey [17], questo dispositivo può essere paragonato ad un "Arduino", quindi una scheda elettronica di piccole dimensioni utile a creare dei prototipi.
Semplicemente Makey Makey è in grado di trasformare ogni oggetto metallico in una tastiera collegandolo all'oggetto con dei cavi. Se si vuole utilizzare questo dispositivo è sufficiente collegarlo al PC tramite un cavo USB. Con due componenti metalliche attaccate al visore e collegate al Makey Makey è possibile semplificare l'esperienza all'utente che non utilizza spesso dei joypad. Toccando le parti metalliche l'utente è in grado di muoversi in direzione del proprio sguardo.

Come già detto questa componente è opzionale perché l'utente può avvalersi di oggetti speciali in scena, che se inquadrati, muovono l'osservatore verso l'oggetto stesso.

Capitolo 2

Unity3D e OculusDK2

In questo capitolo vengono presentate nel dettaglio le due principali tecnologie utilizzate per la creazione dell'applicativo: Unity3D [15], ambiente di sviluppo ed OculusDK2[9], dispositivo di realtà virtuale.

2.1 Unity3D

Unity3d è un software prodotto dalla Unity Tecnologies [15], è uno strumento multipiattaforma che permette di creare videogiochi 3D, animazioni 3D in tempo reale, visualizzazioni architettoniche e altri contenuti interattivi.

2.1.1 Le peculiarità di Unity3D

Il pregio più grande di Unity3D è sicuramente il fatto di essere un software multipiattaforma e quindi un software che in possesso di determinate licenze permette di creare delle applicazioni che possono girare sulle più diffuse piattaforme [18]. Tra

queste ricordiamo: Microsoft Windows [19], Mac OS X [20], Linux [21], Google Android [22], iOS [23], Adobeflash [24], Nintendo Wii [25], Microsoft Xbox [26], Sony PlayStation [27], e tante altre.

Un ulteriore caratteristica di Unity3D è il fatto che supporta l'integrazione con i software di grafica più diffusi: 3DStudioMax [16], Maya [28], Blender [29], Modo [30], Photoshop [31], Adobe Fireworks [32] ecc. e questo permette di utilizzare modelli 3D realizzati con altri software.

Unity3D supporta tre linguaggi di scripting che sono: javascript, C# e Boo.

Per il suo funzionamento Unity3D utilizza delle librerie che sono:

- Direct3D in Windows [33];
- OpenGL in Mac, Windows, Linux [34];
- OpenGL ES in iOS e Android [34].

2.1.2 L'interfaccia di Unity3D

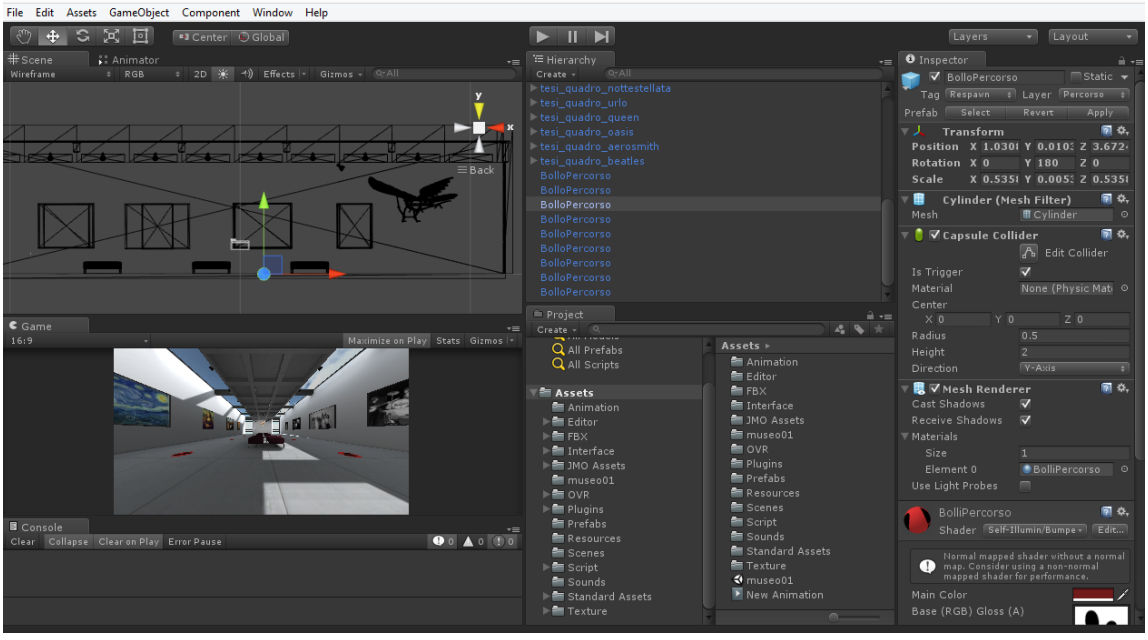


Fig 2.1 - Interfaccia grafica di Unity3D.

Al suo avvio Unity3D si presenta come mostrato in figura. Si può notare che l'interfaccia è suddivisa in diverse aree:

- *Scene*: nella quale viene visualizzato il modello della scena. In quest'area è possibile spostare, scalare, ruotare gli oggetti presenti. Si possono importare dei modelli creati tramite altri software e aggiungere degli oggetti direttamente da Unity3D (es: terreni, primitive geometriche, alberi, ecc).

- *Hierarchy*: in quest'area sono riportati tutti gli elementi presenti nell'area di lavoro *"Scene"*. Ogni oggetto riportato in questo spazio, può essere padre di altri oggetti. La relazione di paternità di un elemento verso l'altro implica che il secondo eredita comportamenti e caratteristiche dal primo.
  Per creare una gerarchia basta trascinare l'oggetto "figlio" sopra l'oggetto "padre". Da questo deriva il nome di quest'area di lavoro.
  In quest'area troviamo anche il pulsante *"Create"* che permette di inserire nuovi elementi in scena (oggetti 3D/2D, luci, telecamere, elementi di interfaccia, ecc.).

- *Game*: in questa finestra viene mostrata la scena inquadrata dalla telecamera principale. L'immagine vista è anche l'aspetto finale dell'applicazione, infatti premendo il tasto play nella parte alta dell'interfaccia è possibile avere una anteprima di quello che si sta facendo.

- *Project*: all'interno di quest'area vengono visualizzate tutte le cartelle e i file che fanno parte del progetto. L'ordine e il contenuto delle cartelle è delegato all'utilizzatore di Unity3D, pertanto è bene mantenere questa sezione ordinata.
  Trascinando degli elementi dall'area *Hierarchy* nell'area *Project* è possibile creare un "prefab", cioè un oggetto che viene salvato per poter essere riutilizzato. In seguito vedremo in maniera più approfondita il significato di "prefab".

- *Inspector*: il contenuto di questa sezione varia in base all'oggetto selezionato all'interno dell'area *"Hierarchy"*, questo perché contiene le *"component"* (le

componenti) relative ad ogni oggetto. In seguito viene approfondito il significato di *"component"*.

- *Console*: all'interno di quest'area vengono riportati tutti i messaggi di errore, di warning e di stampa da parte dell'applicazione.

2.1.3 Il progetto Unity3D

Tutte le risorse necessarie per il funzionamento di un'applicazione creata con Unity3D devono essere contenute all'interno dell'area *"Project"*.

In questa sezione devono essere, quindi, salvati tutti i materiali, le scene, gli script, le animazioni, i suoni, ecc.

Un progetto Unity3D può avere più scene, per questo motivo anche ogni singola scena deve essere salvata.

Una scena è composta da *GameObject* [35] ovvero: primitive geometriche, luci, telecamere ecc. Non è necessario salvare questi oggetti nell'area *"Project"* per il semplice motivo che sono parte integrante di una scena già salvata. E' comunque possibile farlo, e di conseguenza viene creata un'entità denominata *prefab* [36] da Unity3D.

Un *prefab* è sostanzialmente un elemento che oltre ad essere salvato in una scena viene memorizzato singolarmente tra le risorse del progetto. Il nome "prefab" deriva dal termine "prefabbricato" ed è utile quando un oggetto deve comparire più volte in una scena e deve avere delle caratteristiche uguali o simili. Ogni volta che un prefab viene inserito in scena si dice che quella è un'istanza del prefab stesso, quindi ogni eventuale modifica fatta al *prefab "padre"* viene apportata ad ogni singola istanza.

Ogni *GameObject* che fa parte del progetto ha delle proprietà che sono visibili nell'area di lavoro *Inspector* se quel particolare *GameObject* viene selezionato.

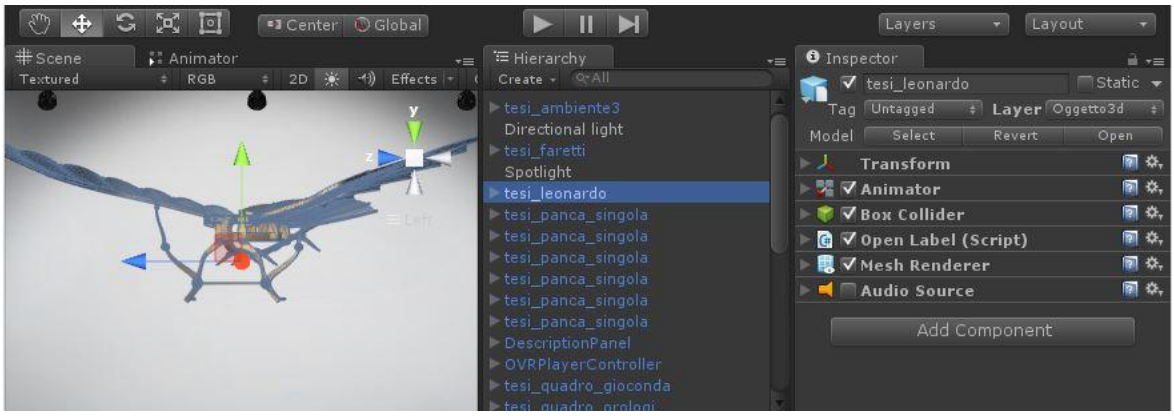


Fig 2.2 - Esempio di component nell'area Inspector.

Queste caratteristiche, che vengono chiamate *"component"* [37], possono essere tolte o aggiunte ad ogni *Gambe Object*.
Per spiegare al meglio il concetto di *component* è opportuno farne alcuni esempi, in particolare vengono riportati quelli più utilizzati per l'applicazione in questione:

- *Collider*: è una componente che se aggiunta ad un *GameObject* permette di gestire le collisioni di quell'oggetto con qualsiasi altro in scena;
- *Script*: è una componente che permette di associare uno script ad *GameObject* per renderlo interattivo;
- *Transform*: è una componente che determina la posizione di un *GameObject* in scena;
- *Animator*: è una componente che associa ad un *GameObject* un'animazione;
- *Mesh Renderer*: è una componente che associa ad un *GameObject* un materiale;
- *Audio Source*: è una componente che associa ad un *GameObject* una risorsa audio.

2.1.4 Lo scripting di Unity3D

Unity3D permette ai propri utenti di generare degli script tramite tre linguaggi di programmazione differenti: C#, Boo e Javascript.
In questa applicazione è stato utilizzato il linguaggio di programmazione C#.
Ad ogni *GameObject* è possibile assegnare uno script, che premette di renderlo interattivo, aggiungendogli la *component "Script"* dall'area *"Inspector"*.
Rendere un *GameObject* interattivo vuol dire fare in modo che nel corso dell'esecuzione del programma il "comportamento" di quest'ultimo varia in base a degli eventi previsti dal programmatore.
Ad un solo *GameObject* è possibile assegnare più script e un solo script può essere assegnato a più *GameObject*, inoltre, uno script può essere importato da altri progetti ed utilizzato nel progetto a cui si sta lavorando.
Quando si realizza un nuovo script nel progetto, Unity3D crea una classe che deriva da *MonoBheaviour* [38], questa classe contiene tutte le funzioni più utilizzate per la

creazione di un'applicazione Unity3D. Di seguito, a titolo di esempio, vengono riportati i due principali metodi della classe *MonoBheaviour* utilizzati nell'applicazione:

- *Start()* [39]: questo metodo viene richiamato quando lo script viene abilitato prima di qualsiasi metodo di Update, questo viene eseguito esattamente una volta nel corso dell'esecuzione del programma. Può essere utilizzato per inizializzare dei valori all'avvio dell'applicazione.
- *Update()* [40]: è quel metodo che viene chiamato ad ogni frame per eseguire di continuo determinati comandi durante l'esecuzione del programma.
  Può essere utilizzata per aggiornare dei valori durante l'esecuzione dell'applicazione.

```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Fig 2.3 - Esempio di uno script vuoto con le funzioni Update() e Start().

Per la creazione di questa applicazione si è reso necessario fare in modo che due script potessero comunicare tra di loro e quindi scambiarsi delle informazioni.
In Unity3D è possibile richiamare una funzione di uno script da un'altro tramite *SendMessage("")* [41]. Questo metodo permette di mandare messaggi ad un

determinato *GameObject*, inviando come parametro una stringa contenente il nome di una funzione. Se tra le *component* del *GameObject* "destinatario" è presente uno script che ha una funzione il cui nome combacia con il parametro inserito in *SendMessage("")* tale funzione viene eseguita.
Per determinare a quale *GameObject* deve essere inviato il messaggio basta utilizzare la funzione *Find("")* [42] che permette di trovare un *GameObject* tramite il nome.

Esempio di utilizzo di *SendMessage("")* e *Find("")*:

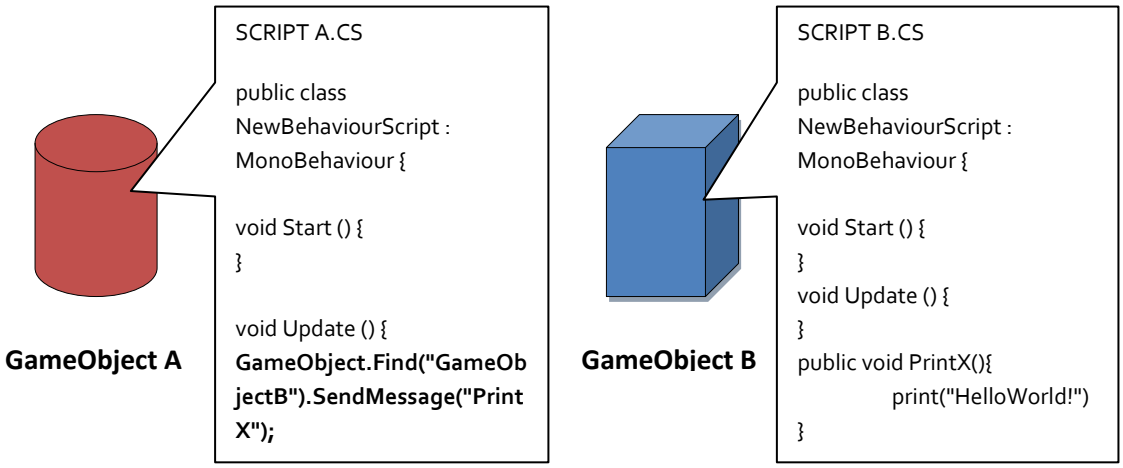


Fig2.4 - L'oggetto A attiva la funzione PrintX dell'oggetto B.

Se, invece, si volesse inviare lo stesso messaggio a più *GameObject* che condividono lo stesso script ma che hanno nomi diversi, al posto di utilizzare le funzione *Find("")* è possibile usare la funzione *FindGameObjectWithTag("")* [43] che permette di trovare tutti gli oggetti che condividono lo stesso *Tag* all'interno del progetto.

Tutti gli script di questo elaborato sono stati realizzati nell'ambiente di sviluppo MonoDevelop [44] che è un implementazione open-source del .Net Framework [45].
Nell'installazione di Unity3D è inclusa una versione proprietaria in modo da poter utilizzare Monodevelop per scrivere del codice.

2.1.5 Le animazioni

In Unity3D è possibile creare delle *Animation Clip* che possono essere attribuite a vari *GameObject* tramite la *component Animator* [46].

Più precisamente un'*AnimationClip* [47] è una clip nella quale vengono memorizzate tutte le caratteristiche di un *GameObject* ad ogni *Frame*.

Unity3D fornisce la possibilità di creare all'interno del software stesso un'animazione tramite un'apposita area di lavoro denominata *Animation*:

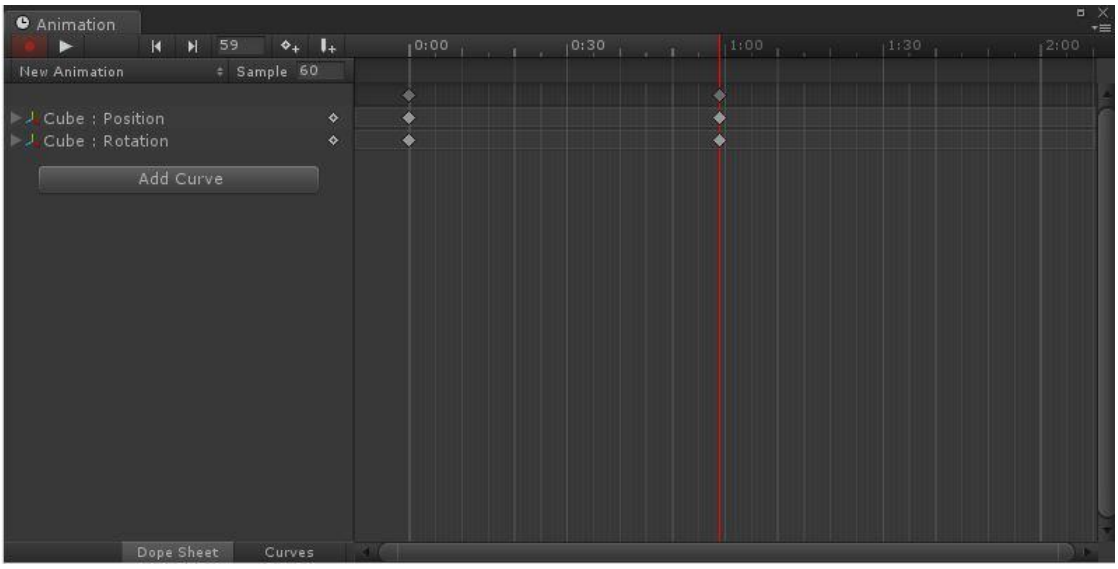


Fig 2.5 - Animazione composta da una rotazione e traslazione di un cubo.

Per creare una nuova clip è sufficiente selezionare l'oggetto desiderato, posizionarsi con il cursore sopra un determinato *frame* e premere il tasto "*Add Curve*" per apportare una modifica che compone l'animazione.

Un'*AnimationCurve* [48] in Unity3D è quindi una modifica (*scalatura*, *rotazione*, *translazione ecc*.) fatta ad uno o più *GameObject* in un determinato *frame*. Più *AnimationCurve* messe assieme formano un *AnimationClip*.

Ad un *GameObject* possono essere assegnate diverse animazioni in base allo stato in cui si trova.

Per esempio, un *GameObject* che rappresenta una figura umana nel corso dell'esecuzione di un videogioco potrebbe disporre di diverse animazioni che

simulano: la camminata, la corsa, il salto ecc. Per questo è necessario associare le animazioni al personaggio e gestirle in modo che ne venga attivata una alla volta.

A tale scopo nasce la componente *Animator* che associa ad un *GameObject* un *Animator Controller* [49].

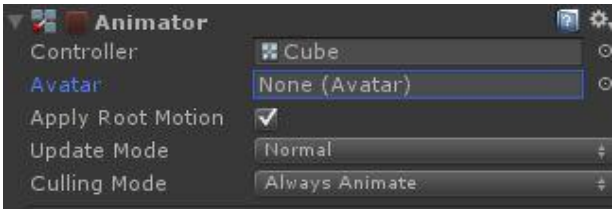


Fig 2.6 - Component Animator.

L'*AnimatorController* che viene creato per l'oggetto da animare é salvato tra le risorse del progetto di Unity3D e consente di mantenere una serie di animazioni per l'oggetto stesso e passare da una all'altra quando si verificano determinate condizioni di gioco.
L'*Animator Controller* gestisce le transizioni tra le animazioni usando uno *State Machine* [50], una sorta di semplice programma scritto in linguaggio di programmazione visivo all'interno di Unity3D.

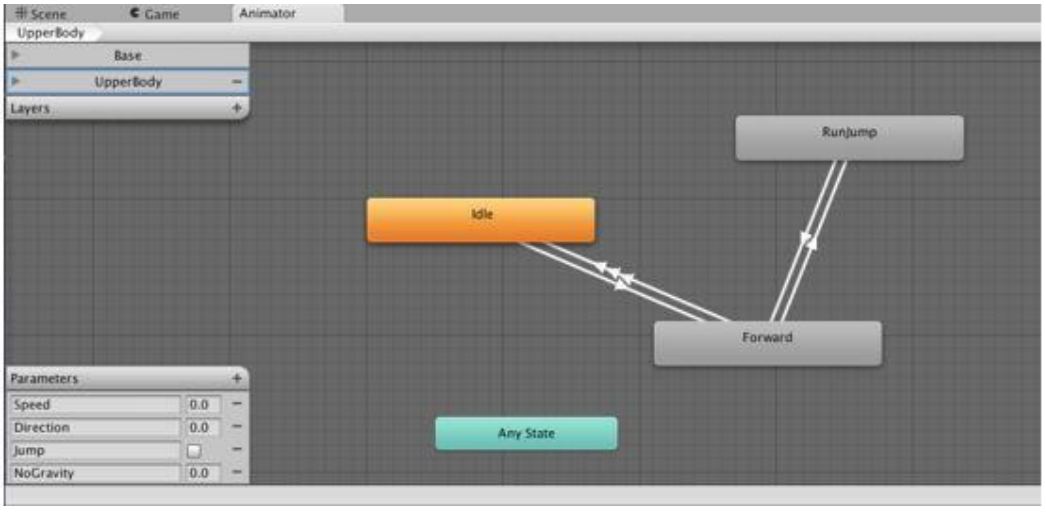


Fig 2.7 - State Machine.

2.1.6 I suoni

In Unity3D è possibile dotare un *GameObject* di un suono semplicemente aggiungendogli la *component AudioSource* [51].
Questa componente associa un file audio, salvato tra le risorse del progetto, ad un oggetto e gestisce le proprietà che deve avere il suono emesso.

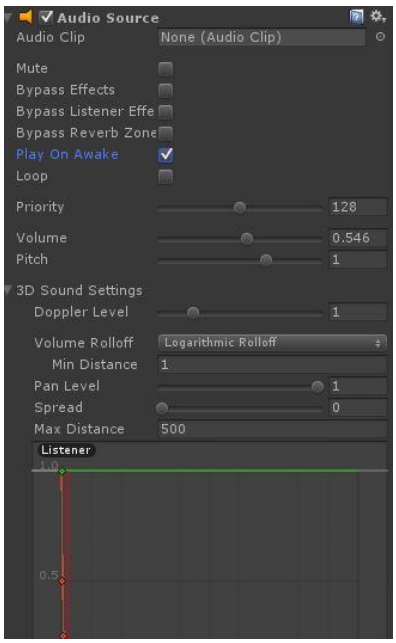


Fig 2.8- Component AudioSource.

2.1.7 Il nuovo sistema di interfaccia

Dalla versione 4.6 beta, Unity3D mette a disposizione dei propri utilizzatori un nuovo sistema che consente di realizzare interfacce utente *(UI, User Interface)* per giochi e applicazioni interattive.
Unity UI [52] fornisce agli sviluppatori un insieme di strumenti potenti e flessibili, ottime prestazioni e funzionalità originali per le animazioni.

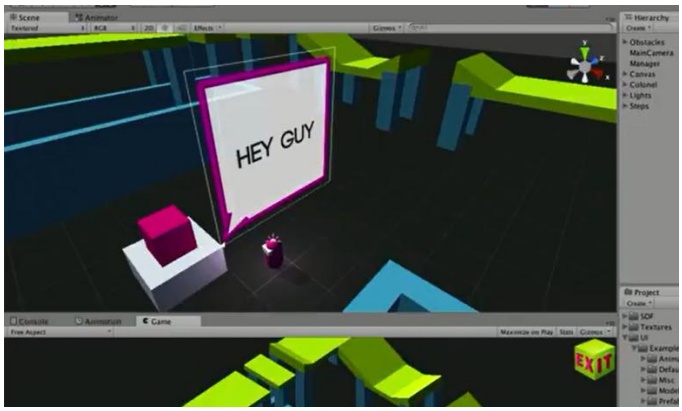


Fig 2.9 - Esempio del nuovo sistema di interfaccia.

Ora descriviamo nel particolare gli strumenti per la creazione di un'interfaccia utente utilizzati nello sviluppo di questa applicazione:

- *Canvas* [53]: può essere definita come l'area contenitore di tutti gli elementi che compongono un'interfaccia utente. Il *Canvas* non è altro che un *GameObject* a cui è assegnata la *component "Canvas"*, tutti gli elementi che faranno parte dell'interfccia utente devono essere figli di tale *Canvas*.

  Il *Canvas* si presenta come un'area rettangolare nella quale vengono disegnati tutti gli elementi figli secondo un criterio d'ordine: il primo figlio viene disegnato per primo, il secondo per secondo e così via. Se due elementi si sovrappongono quello successivo apparirà sopra il precedente.

  Il *Canvas* ha tre principali modalità di rendering: *screen space overlay*, *screen space camera* o *world space*.

  La modalità *screen space overlay* pone gli elementi dell'interfaccia sullo schermo renderizzato sopra la scena, se lo schermo viene ridimensionato o cambia risoluzione anche il *Canvas* cambia automaticamente le sue dimensioni.

  La modalita *screen space camera* è simile alla precedente con la differenza che in questa il *Canvas* è collocato a una data distanza di fronte ad una specifica telecamera. Gli elementi della *UI* sono quindi renderizzati da questa telecamera, il che significa che sono le impostazioni della telecamera a determinare l'aspetto dell'interfaccia utente. Se la telecamera è impostata su prospettiva anche gli elementi dell'interfaccia verranno renderizzati in prospettiva.

  La modalità *world space* prevede che il *Canvas* si comporti come qualsiasi oggetto in scena, le sue dimensioni possono essere impostate attraverso apposite proprietà e gli elementi della *UI* saranno renderizzati davanti o dietro ad altri oggetti in base al posizionamento nella scena 3D. Questa modalità è utile per le interfacce utente che sono destinate ad essere parte del mondo.

- *Text* [54]: è una *component* che può essere assegnata direttamente al *Canvas* oppure ad un suo oggetto figlio, serve a riservare all'interno di esso un area di testo.

  Oltre ad inserire del testo nell' *UI* è possibile dargli una formattazione modificando i parametri di questa *component*.

- *Image* [55]: è una *component* che può essere assegnata direttamente al *Canvas* oppure ad un suo oggetto figlio, indica come apparirà una determinata immagine all'interno dell'interfaccia utente.
- *Button* [56]: è una *component* che può essere assegnata direttamente al *Canvas*, oppure ad un suo oggetto figlio; diversamente dalle precedenti questa è una componente che permette di rendere l'interfaccia interattiva.

  Su questo tipo di componente Unity prevede l'evento *OnClick* che definisce quando il bottone è stato premuto.

Gli strumenti di interfaccia elencati sono solo alcuni di tutti quelli forniti da Unity3D, in questa tesi si è ritenuto opportuno riportare solamente quelli utilizzati nell'applicazione in questione.

Per avere un elenco completo di tutti gli elementi basta visitare la *Unity Documentation* [52] sul sito di Unity3D.

2.1.8 Asset ed Asset Store

Gli *Asset* in Unity3D sono file utili per il progetto; non si tratta solamente di *modelli 3D* ma anche di *texture*, *shaders*, *script* o addirittura intere scene sviluppate con altri programmi (3DStudioMax, Maya).

Può essere considerato *Asset* ogni file contenuto all'interno dell'area Project di Unity3D.

Durante lo sviluppo di un'applicazione oltre a creare nuovi Asset, quindi nuovi file, è possibile importarne altri da vecchi progetti o dalla rete.

A tale scopo nasce l'*Asset Store* [57], un sito dove ogni utente di Unity3D se registrato può pubblicare e scaricare dei file che possono essere utili ad altri utenti nel primo caso, oppure che possono servire nel proprio progetto nel secondo caso.

L'*Asset Store* è anche fonte di guadagno per quegli utenti di unity3D che sono registrati al sito e hanno pubblicato dei file scaricabili a pagamento.

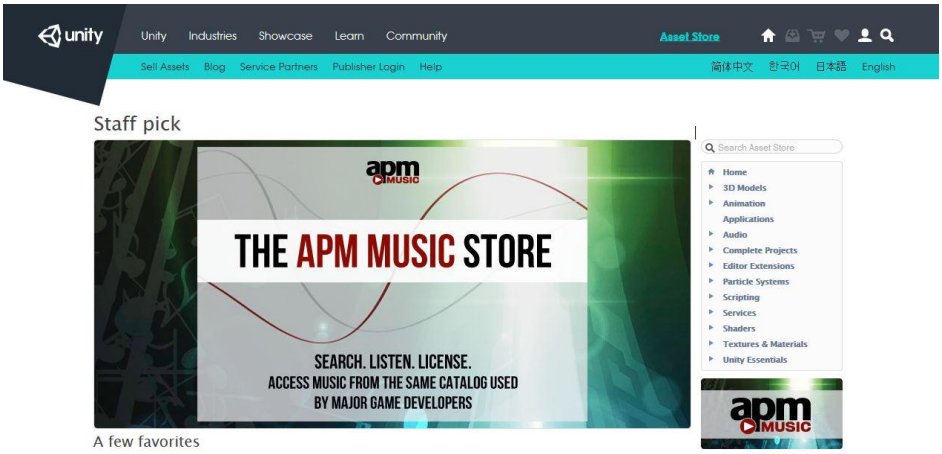


Fig 2.10 - Asset store

2.1.9 Il Package OVR

Per utilizzare Oculus in un progetto Unity3D è necessario importare all'interno del progetto un package che contiene gli *Assets* necessari per il suo utilizzo.
Il package da importare per il visore OculusDK2 è scaricabile dal sito ufficiale di Oculus assieme all'SDK [58] (*Software Development Kit*) per i vari sistemi operativi.
All'interno del package OVR sono contenute tutte le risorse necessarie per l'integrazione del dispositivo Oculus nel programma: prefab, script, materiali, texture, shaders ecc.

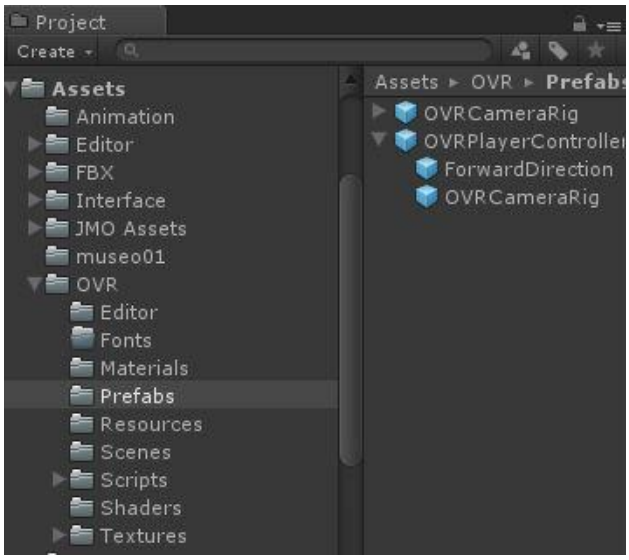


Fig 2.11 - PackageOVR.

Una volta importato questo package nel progetto per utilizzare OculusDK2 è sufficiente recarsi nella cartella *"Prefabs"* e trascinare dentro la scena l'intero *"OVRPlayerController"*.
OVRPlayerController è un prefab dotato di collider che simula la presenza di un corpo umano all'interno della scena.
OVR PlayerController è padre di *Forward Direction* e di *OVRCameraRig*.
ForwardDirection è un *GameObject* che ha come sola componente *"Transform"* : le coordinate che individuano l'oggetto all'interno della scena. La *ForwardDirection* serve ad individuare la direzione di osservazione del dispositivo Oculus all'interno della scena.

Il *GameObject "OVRCameraRig"* è invece un oggetto a cui sono allegati due script contenuti nel *packageOVR*. *L'OVRCameraRig* è a sua volta padre di un *GameObject "CenterEyeAnchor"* che indica un punto centrale situato tra i due occhi e di due telecamere *"LeftEyeAnchor"* e *"RightEyeAnchor"* che rappresentano rispettivamente l'occhio sinistro e destro dell'essere umano.
Queste due telecamere sono posizionate una di fianco all'altra ad una distanza tale da simulare al massimo la distanza tra le due lenti del visore.
Per comprendere al meglio la struttura del prefab descritto viene riportata di seguito un'immagine che lo mostra inserito all'interno della scena:

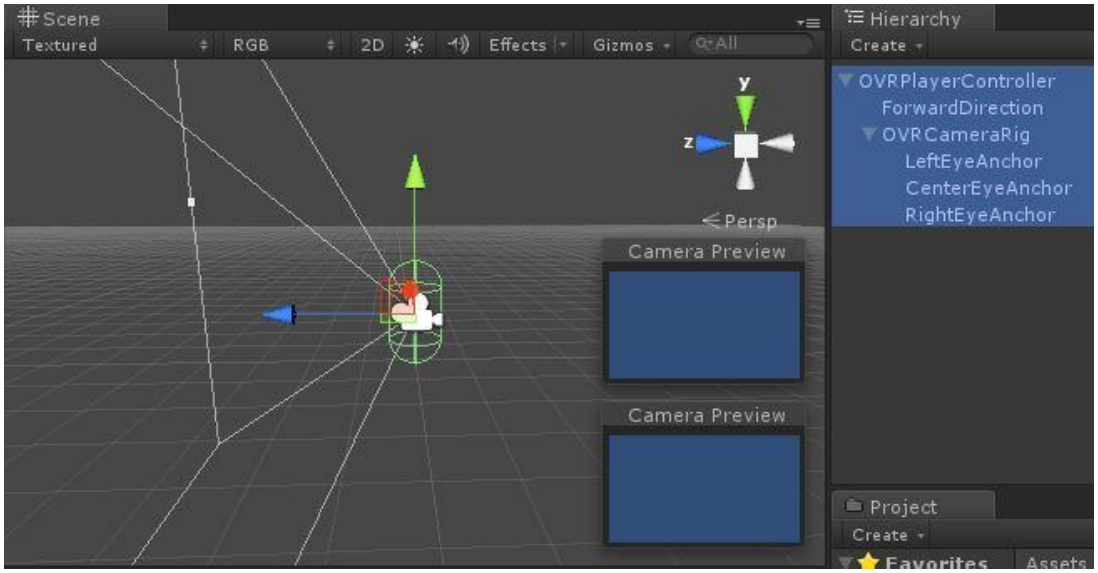


Fig 2.12 - OVRPlayerController in scena.

L'oggetto che nella figura precedente è colorato di verde è un collider e sta a indicare la presenza fisica in scena dell'*OVRPlayerController.*
Quest'oggetto può spostarsi nelle direzioni X e Z e può essere ruotato attorno all'asse Y grazie a degli appositi comandi. Questi movimenti simulano il moto abituale di un corpo umano. L'*OVRCameraRig* figlia dell'*OVRPlayerController* imita invece lo spostamento di una testa. Quest'ultima, infatti, si può ruotare nell'asse Y indipendentemente dall'oggetto padre ma lo spostamento nei restanti due assi dipende solo dal movimento del corpo.

Nella figura precedente si possono notare due quadrati blu che mostrano le due viste ottenute dalle telecamere figlie dell'*OVRCameraRig*. Le due viste in figura sono blu per il semplice motivo che è rappresentata una scena vuota.

2.2 Oculus DK2

OculusDK2 è un visore per realtà virtuale composto da due lenti che proiettano le immagini su due schermi a bassissima latenza che lavorano in simbiosi per ogni occhio coprendo completamente la vista di chi lo usa, immergendolo totalmente nella realtà creata dal programma.
Anche se OculusDK2 è l'evoluzione del prodotto OculusRift [59] e offre prestazioni molto più elevate, non è ancora disponibile al grande pubblico. Infatti la sua casa produttrice OculusVR è chiara sul fatto che OculusDK2 è ancora un kit di sviluppo ed è acquistabile solo sul sito di OculusVR.

2.13 - OculusDK2

2.2.1 Caratteristiche Tecniche

Nella sua parte anteriore OculusDK2 monta uno schermo che è stato scelto appositamente per l'utilizzo con la realtà virtuale. Il pannello 1920x1080 utilizza la tecnologia *PenTile* di *Samsung* [60], che divide i pixel nei colori RGB che li compongono. Grazie a questa tecnologia risulta meno evidente il reticolato nero che si creava per lo spazio tra i pixel nell' OculusRift.
Un'altro enorme vantaggio di questo schermo OLED è la bassa latenza [61], che ha ridotto fortemente gli effetti di nausea causati dall'OculusRift. Grazie a questa caratteristica anche i movimenti più rapidi della testa non producono del caos, e le immagini sono più nitide e reattive.

Il refresh a 75Hz [61] dell'OculusDK2 permette di diminuire, se non eliminare, il senso di nausea che si può avere utilizzando un dispositivo di realtà virtuale. Questa caratteristica sebbene ottima da un lato, introduce nuove problematiche. Un aggiornamento più rapido implica una minor differenza tra l'orientamento della testa e il fotogramma visualizzato sullo schermo.
Facendo partire il sistema ci si trova di fronte ad un problema più canonico: la potenza del sistema. Per sfruttare la visualizzazione a 75Hz è necessario far girare le applicazioni a 75 fotogrammi al secondo, e questo non è un problema molto facile da risolvere.
L'ultima innovativa caratteristica introdotta con questo visore è il tracciamento della posizione [61], quindi oltre a percepire l'orientamento della testa il dispositivo è in grado di capire gli spostamenti effettuati. Tutto questo è possibile grazie a dei led a infrarossi che inviano segnali ad una videocamera che deve essere rivolta verso l'utilizzatore durante l'esperienza. Uno svantaggio introdotto da questa nuova caratteristica è sicuramente la dipendenza da una telecamera che costringe l'utente a mantenere una posizione limitata di fronte ad essa.

2.2.2 Come Funziona OculusDK2

Come si è ben capito, OculusDK2 è un hardware e come tale ha bisogno di un software dedicato per poter funzionare. Non basta collegarlo al proprio PC per poter navigare tra le cartelle in 3D come si potrebbe pensare, ma è necessario che qualche sviluppatore abbia creato un programma pensato per questa tecnologia che sfrutti una visione stereoscopica.
Si riesce ad avere coscienza della profondità poiché l'immagine unica che il cervello elabora è formata da due immagini leggermente diverse acquisite ognuna da ogni occhio come succede nella realtà.

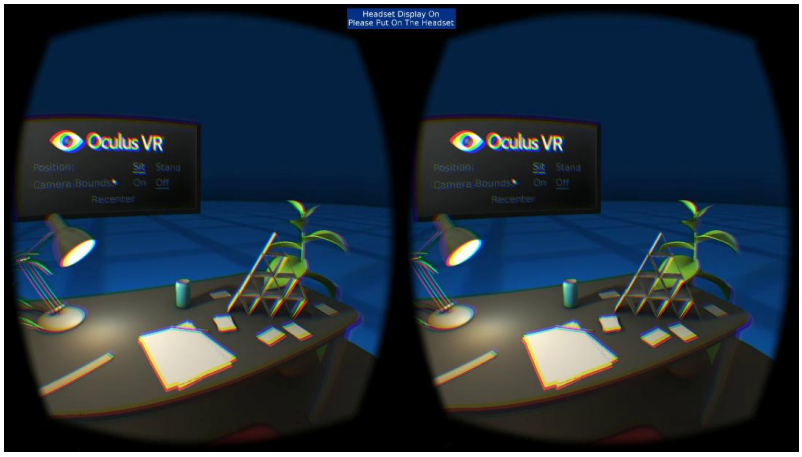


Fig 2.14 - Le due immagini dei due schermi.

Questo dispositivo è dotato di un giroscopio molto sensibile che segue i movimenti sui quattro assi di visuale (alto, basso, destra, sinistra) in modo da potersi guardare attorno durante l'esperienza.

Come già detto in precedenza il visore è dotato anche di led a infrarossi in modo tale da percepire oltre alla rotazione anche i movimenti di spostamento della testa a destra, sinistra, in alto e in basso.

Lo spostamento all'interno della scena è delegato ad altre soluzioni hardware o software.

Capitolo 3

Implementazione del progetto

In questo capitolo vengono analizzati tutti gli script creati per il funzionamento della suddetta applicazione. Gli script sono tre, e vengono assegnati come *component* a diversi tipi di oggetti.

3.1 Osservatore e oggetto interattivo

All'interno di questo progetto possiamo suddividere tutte le tipologie di oggetti in due insiemi:

1. Osservatore: è l'oggetto che rappresenta l'utente in scena e sostanzialmente è il *GameObject* "*OVRPlayerController*" di cui si è parlato nel capitolo precedente.
2. Oggetto interattivo: è quell'oggetto che, se inquadrato dall'osservatore per un certo periodo di tempo, fornisce delle informazioni che riguardano se stesso.

Ad ogni oggetto che rientra in queste due tipologie sono stati assegnati uno o più script vediamoli:

1. Script *OggettoInquadrato.cs*: tale script è assegnato ad un unico *GameObject* figlio dell'*OVRPlayerController* (oggetto di tipologia 1).

Fig 3.1 - La component script Oggetto Inquadrato.

2. Script *OpenLabel.cs*: questo script è assegnato ad ogni *GameObject* che deve fornire delle informazioni riguardanti se stesso (oggetti di tipologia 2).

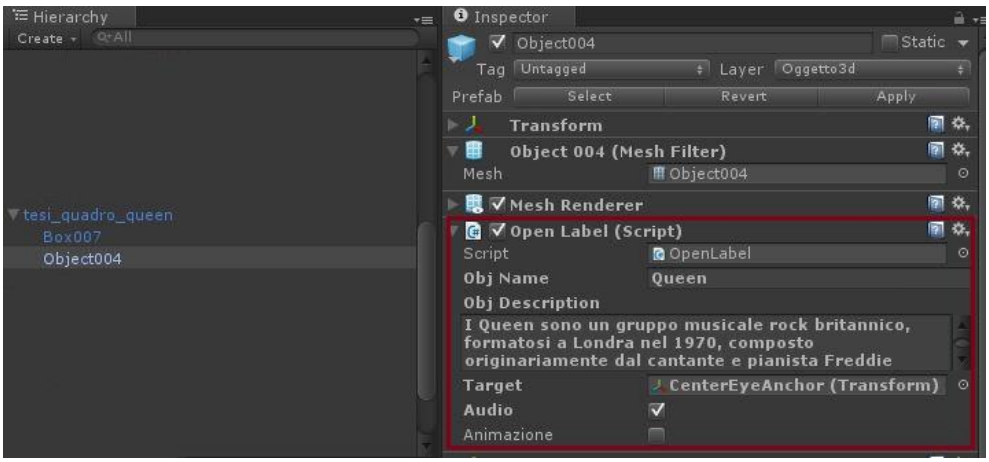


Fig 3.2 - La component script Open Label.

3. Script *MovePlayer.cs*: tale script è assegnato ad un unico oggetto: il *GameObject OVRPlayerController* (oggetto di tipologia 1).

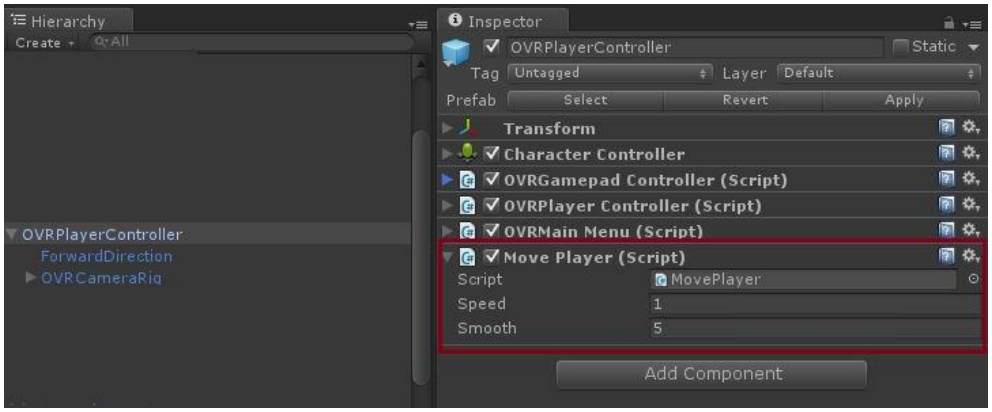


Fig 3.3 - La component script MovePlayer.

Tutti gli script citati sono salvati all'interno la cartella "Script" del progetto.

3.2 Le variabili esposte

Prima di procedere con la spiegazione degli script è bene conoscere il significato di variabile esposta. Le variabili esposte di uno script sono dei parametri utilizzati nello script stesso che possono essere modificati dall'area *Inspector* di Unity3D

.

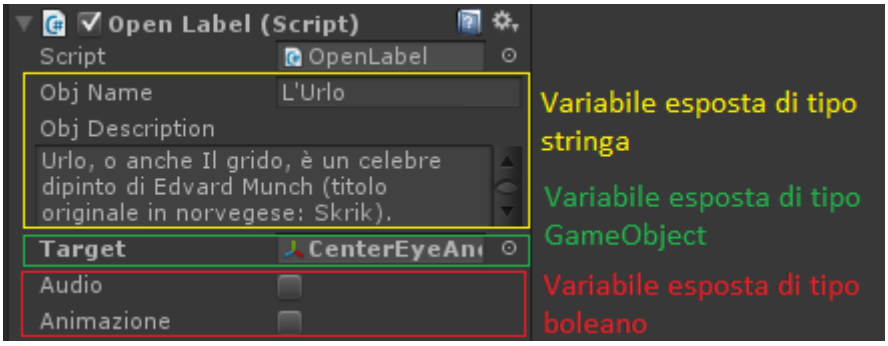


Fig 3.4 - Variabili esposte.

Queste variabili sono molto utili nel caso in cui si vogliono modificare dei valori nello script senza dover mettere mano al codice.
Le variabili esposte devo essere dichiarate come tali all'interno dello script tramite la seguente dicitura:

```
public TipoVariabile NomeVariabile;
```

3.3 L'operazione di intersezione "Raycast"

Prima di iniziare a commentare singolarmente ogni script è opportuno parlare dell'operazione fondamentale che sta alla base del funzionamento di questa applicazione: il *Raycast* [62].
Precisamente il *Raycast* è un metodo che deriva dalla classe *Phisics* di Unity. Questo metodo serve ad indicare se il raggio, che parte da un determinato punto e prosegue per una certa direzione, interseca qualche oggetto durante il suo percorso.

Se l'esito è positivo, quindi se il raggio ha intersecato qualche collider, il metodo di *Raycast* restituisce il valore *"true"* altrimenti *"false"*.
Vediamolo:

```
public static bool Raycast(Vector3 origine,Vector3 direzione,float
distanza);
```

Come si può notare il metodo restituisce un valore booleano e prende in input tre parametri:

- origine: punto di partenza del raggio da sparare;
- direzione: la direzione del raggio;
- distanza: la lunghezza del raggio.

Avendo a disposizione questo utilissimo metodo si è pensato di utilizzarlo per determinare quale oggetto sta osservando l'utente che indossa OculusDK2. In pratica, tramite questo metodo, viene sparato un raggio invisibile che ha origine nel punto "*CenerEyeAnchor*" del prefab "*OVRPlayerController*" e direzione uguale a quella di osservazione in modo tale da capire se un oggetto è intersecato da quel raggio.
Vediamo il caso in cui l'indossatore dell'OculusDK2 è fermo ad osservare un punto di fronte ad esso:

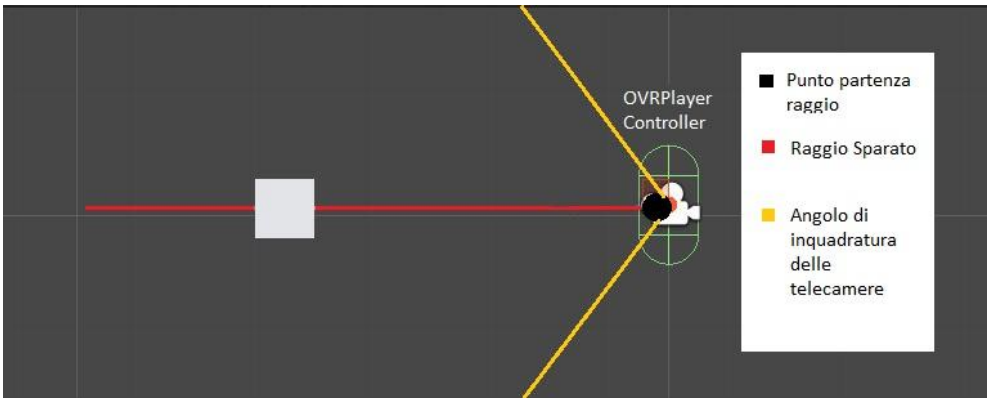


Fig 3.5 - Raycast: esito positivo.

In questo caso il raggio ha colpito un oggetto: l'osservatore lo sta inquadrando.
Ora vediamo il caso in cui l'osservatore sta rivolgendo la testa verso l'alto:

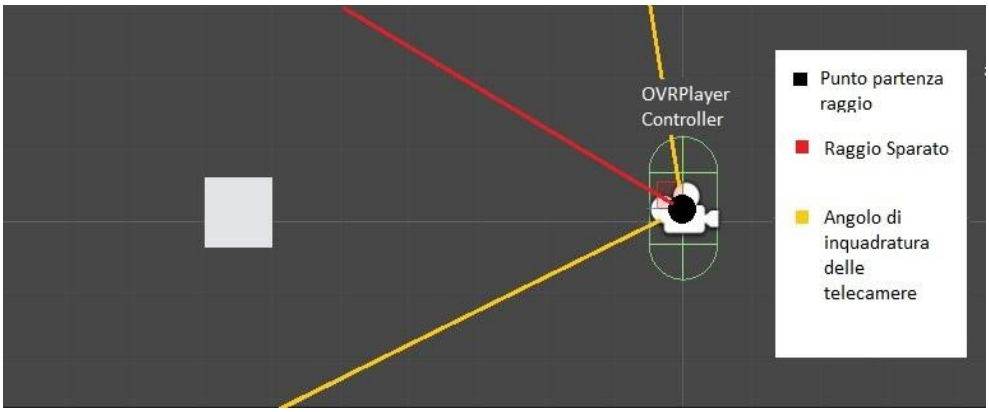


Fig 3.6 - Raycast: esito negativo.

In questo caso il *Raycast* non ha colpito nessun oggetto, il cubo non è inquadrato direttamente ma solamente contenuto all'interno del campo visivo.

Nel caso in cui il raggio avente la stessa origine e la stessa direzione interseca più oggetti, solamente il più vicino all'osservatore viene considerato come "oggetto colpito".
Affinché il metodo di *Raycast* ritorni un valore positivo è necessario che l'oggetto colpito dal raggio sia dotato della *component Collider*, altrimenti l'oggetto verrebbe ignorato.
L'interazione tra l'utente e gli oggetti in scena, in questa applicazione, è gestita completamente tramite questo importantissimo metodo.

3.4 Livelli in scena

Come è stato accennato nei precedenti paragrafi, possiamo raggruppare tutti gli oggetti presenti nella scena in due categorie generali: osservatore e oggetto interattivo. In questo capitolo vengono esaminate tutte le sottocategorie che riguardano la tipologia "oggetto interattivo".
In scena esistono più tipi di oggetti interattivi:

- Oggetti con descrizione: quadri, poster, sculture.
- Oggetti di movimento: pedane che permetto lo spostamento dell'osservatore.
- Pulsante chiusura interfaccia: bottone che permette la chiusura dell'interfaccia.
- Pulsante avvio animazione: bottone che permette l'avvio di un'animazione.
- Resto della scena: tutti gli oggetti che non fanno parte delle precedenti categorie.

Ogni volta che l'osservatore (*OVRPlayerController*) spara il raggio di *Raycast* verso uno degli oggetti in scena deve poter capire con quale tipologia di oggetti il raggio si è intersecato.
Per fare questo si è deciso di adottare la tecnica dei *Layers* (livelli) [63] di Unity3D, questa tecnica viene utilizzata principalmente per renderizzare oggetti che appartengono ad un determinato *Layer* e ignorare gli altri, ma può essere usata anche dal *Raycasting* per gestire le collisioni e capire a quale *Layer* appartiene l'oggetto che è stato colpito dal raggio.
E' quindi possibile assegnare ad ogni *GameObject* il suo *Layer* dall'area *Inspector*.

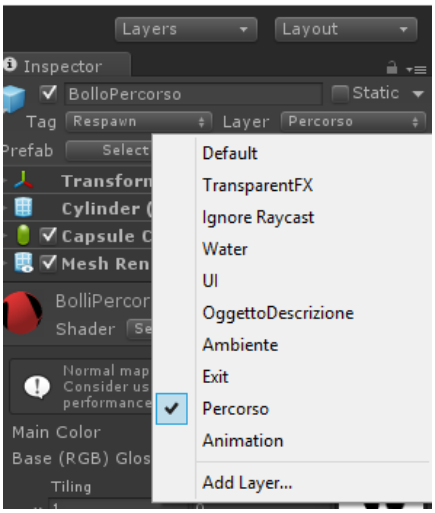


Fig. 3.7 - Esempio assegnazione di un Layer ad un GameObject.

Come si può notare, nella Fig. 3.7 l'elenco dei *Layer* è più numeroso di quello relativo alle sottocategorie previste dalla nostra applicazione. Questo perché in Unity3D sono stabilite per default cinque tipologie di *Layer*: *Default*, *TransparentFX*, *IgnoreRaycast*, *Water*, *UI*.

Oltre ad utilizzare quelli citati, il programmatore è libero di creare nuovi *Layer* e quindi definire delle categorie a proprio piacimento.

In questa applicazione sono stati creati cinque nuovi *Layer*, ognuno dei quali corrisponde ad una sottocategoria degli oggetti interattivi:

1. *OggettoDescrizione*: *Layer* assegnato alla sottocategoria *Oggetto con descrizione*.
2. *Percorso*: *Layer* assegnato alla sottocategoria *Oggetti di movimento*.
3. *Exit*: *Layer* assegnato alla sottocategoria *Pulsante chiusura interfaccia*.
4. *Animation*: *Layer* assegnato alla sottocategoria *Pulsante avvio animazione*.
5. *Ambiente*: *Layer* assegnato a tutti gli oggetti che non fan parte di alcuna categoria.

Ogni *Layer* viene associato ad un numero che lo identifica e per sapere a quale categoria appartiene l'oggetto che viene intersecato dal *Raycast* è sufficiente valutare le proprietà del *GameObject* colpito:

```
Int IdLayer = hit.collider.gameobject.layer;
```

3.5 Lo script "OggettoInquadrato.cs"

Il primo Script che sarà analizzato è lo script "*OggettoInquadrato.cs*".
Questo script è assegnato all'elemento osservatore, quindi, al *GameObject* "*CenterEyeAnchor*" a sua volta figlio del *GameObject* "*OVRPlayerController*".
Generalmente questo script ha la funzione di sparare il raggio di *Raycast*, partendo dall'oggetto stesso e riconoscere a quale *Layer* appartiene l'oggetto che è stato intersecato da quest'ultimo. In base all'esito avuto dalle operazioni descritte lo script eseguirà dei comandi specifici definiti per ogni categoria (*Layer*).

3.5.1 Struttura generale

Lo script si suddivide in due metodi principali che derivano dalla classe *MonoBheavior* e sono *Start()* e *Update()*.
Il primo viene eseguito una sola volta durante l'esecuzione dell'applicazione, più precisamente al suo avvio, vediamolo:

```
    void Start ()
    {
        ObjectH = new GameObject();
        U_time = 1.5f; // Timer a 3 sec.
        ObjectH = null; // Oggetto Temp nullo.
        ObjectPrecAudio = null;
        ObjectPrecAnim = null;
        anim = GameObject.Find ("Mirino").GetComponent
        <Animator>();
        move = false;
        PrecPos = GameObject.Find("OVRPlayerController").
        transform.position;
    }
```

Questo metodo non fa altro che inizializzare delle variabili che sono state definite globali. La dichiarazione delle variabili globali non è stata riportata in questa tesi perché non è stata ritenuta di fondamentale importanza.

Il metodo *Update()* può esser considerato la parte significativa dello script.
Questo metodo viene eseguito continuamente durante l'esecuzione dell'applicazione.
Il metodo in questione non viene riportato in un unico blocco poiché risulta essere di grandi dimensioni, per questo andremo ad analizzarlo e spiegarlo riportandolo suddiviso in parti più piccole.

```
void Update () {
        // Punto colpito dal raggio.
        RaycastHit hit;
        // Direz. osservaz.
        Vector3 fwr = transform.TransformDirection(Vector3.forward);
        if (Physics.Raycast (transform.position, fwr, out hit))
        // viene sparato il raggio e se colpisce qualche
        superficie vengono eseguite le seguenti istruzioni
        {
        if (hit.collider.gameObject.layer == 8 && move == false)
              {
                 // 1° condizione
                 // esegui le istruzioni previste per il layer
                 "OggettoInterattivo"
                 ....
              }
              } else if (hit.collider.gameObject.layer == 10)
              {
                 // 2° condizione
                 // esegui le istruzioni previste per il layer
                 "Exit"
                 ....
              }
```

```
            else if (hit.collider.gameObject.layer == 11)
            {
                // 3° condizione
                // esegui le istruzioni previste per il layer
                "Percorso"
                ....
            }
            else if (hit.collider.gameObject.layer == 12)
            {
                // 4° condizione
                //esegui le istruzioni previste per il layer
                "Animation"
                ....
            }
            else
            {
                //5° condizione
                // esegui le istruzioni previste per gli oggetti
                che hanno un layer diverso da quelli previsti
                precedentemente
                ....
            }
    else
    {
        // esegui queste istruzioni se il raggio non interseca
        alcuna superficie
        ....
    }

    ....
}
```

Il codice riportato qua sopra rappresenta la struttura generale del metodo *Update()* dello script *"OggettoInquadrato.cs"*.

Ad ogni frame viene sparato il raggio che parte dal punto *transform.position* e procede per la direzione di osservazione *fwd*.

Il valore *transform.position* rappresenta le coordinate in scena del *GameObject* a cui è assegnato lo script, quindi in questo caso *"CenterEyeAnchor"* (il punto che si trova al centro delle due telecamere che rappresentano le due viste oculari).
Se l'operazione di *Raycast* (operazione booleana) ha esito positivo si procede controllando a quale *Layer* appartiene l'oggetto colpito.

3.5.2 Intersezione con il layer "OggettoInterattivo"

Quando il raggio di *Raycast* si interseca con un oggetto che ha il *Layer* "OggettoInterattivo" e l'*OVRPlayerController* non è in movimento, si entra nella prima condizione del metodo *Update()*.
Oltre ai vari controlli, il compito principale di questa pozione di codice è di verificare se il raggio interseca lo stesso oggetto per più di tre secondi. Se questo avviene lo script provvederà inviando un messaggio ad uno script allegato all'oggetto colpito (che analizzeremo in seguito).
Nel fare tutto questo deve essere gestito anche un timer, che dovrà essere azzerato ogni volta che il raggio non colpisce più l'oggetto.
E' stata fatta la scelta di far trascorrere tre secondi prima di inviare il messaggio allo script che apre il pannello di descrizione perché, altrimenti ogni volta che lo sguardo si sarebbe incrociato con qualche oggetto interattivo, si sarebbe aperto immediatamente il pannello di descrizione anche se l'utente non avrebbe voluto.
Ogni attesa viene segnalata tramite un'animazione sul mirino che ricorda il caricamento.

Fig 3.8 - Animazione mirino durante l'attesa di apertura del pannello.

Ora vediamo il codice che gestisce il comportamento dell'applicazione a seguito dell'intersezione di *Raycast* con un oggetto del *Layer* "OggettoInterattivo". Ricordiamo che all'interno del progetto il *Layer* "OggettoInterattivo" ha come identificativo il numero 8:

```
if (hit.collider.gameObject.layer == 8 && move == false) {

        ////////////////////// 1° CONDIZIONE//////////////////////
        // Caso in cui il raggio interseca per la prima volta il
             Layer "OggettoInterattivo"

        if (layercollision != 8) {
             // Inizializziamo il tempo a 3 secondi
             U_time = 1.5f;
             // Disattivo le animazioni del mirino usate per gli
                              altri Layer
             anim.SetBool ("Inquadrato", false);
             anim.SetBool ("walk",false);
        }

        // layercollision è una variabile che memorizza a quale
             Layer appartiene l'ultimo oggetto colpito
        layercollision = 8;

        ////////////////////// 2° CONDIZIONE//////////////////////
        // Caso in cui spostiamo il raggio verso un oggetto
        diverso da quello colpito precedentemente

        if (hit.collider.gameObject != ObjectH && ObjectD !=
             hit.collider.gameObject) {
        // viene attivata l'animazione di caricamento, impostato
        il timer a tre secondi
             anim.SetBool ("Inquadrato", true);
             U_time = 1.5f;
```

```
        //setpos è una variabile booleana che memorizza un
        valore positivo se è stata stabilita una posizione
        corretta in scena per il pannello rispetto
        all'oggetto colpito dal raggio
        setpos = false;

        // effettuate delle operazioni di controllo
        sull'audio e sull'animazione relativi agli oggetti
        colpiti
            if (ObjectH != null && ObjectH.audio != null &&
                ObjectH.audio.enabled == true) {
                 ObjectPrecAudio = ObjectH;
            }

            if(ObjectH !=null &&
            ObjectH.GetComponent<Animator>()!= null &&
                   AnimazioneAttiva==true ){
                 ObjectPrecAnim = ObjectH;
            }

            if(ObjectH!= null)
            {
                 anim.SetBool("Inquadrato",false);
            }
////////////////////3°CONDIZIONE//////////////////////
} else {

if (U_time > 0 && ObjectD != hit.collider.gameObject) {
     anim.SetBool ("Inquadrato", true);
}

// La variabile  U_Time viene decrementata ad ogni frame
U_time -= Time.deltaTime;
```

```
            // Caso in cui il timer arriva a 0 .
            if (U_time <= 0.0f) {
            //vengono inviati due messaggi allo script allegato
            all'oggetto colpito con il primo gli si  chiede di
            impostare la posizione del pannello in scena con il
            secondo gli si chiede di far apparire il pannello

            hit.collider.gameObject.SendMessage ("Setpos",setpos);

            hit.collider.gameObject.SendMessage ("HitObject"
            transform.forward.normalized);

            setpos = true;
            anim.SetBool ("Inquadrato", false);
            Active_Play=false;
            description = true;

            //ObjectD è una variabile che memorizza l'oggetto
            attivo
            ObjectD = hit.collider.gameObject;
            if (ObjectPrecAudio != null && ObjectPrecAudio !=  ObjectD)
{
            ObjectPrecAudio.audio.enabled = false;
             }
                if (ObjectPrecAnim != null && ObjectPrecAnim !=
                ObjectD){
                ObjectPrecAnim.SendMessage("DisactiveAnimation");
                     AnimazioneAttiva= false;
            }
        }
        // ObjectH è una variabile che memorizza l'oggetto colpito
            dal raggio in questo frame, questa variabile è diversa da
            ObjectD perchè l'oggetto memorizzato quà può anche non
            essere attivo.
        ObjectH = hit.collider.gameObject;
} else if...
```

Nella prima condizione viene esaminato se l'operazione di *Raycast* nel *frame* corrente ha dato luogo ad un'intersezione con un oggetto di un *Layer* diverso rispetto al precedente *frame*. Per esempio, se l'utente rivolge lo sguardo verso un oggetto del *Layer* "OggettoInterattivo" dopo aver inquadrato un oggetto appartenente al *Layer* "Percorso".
In particolare tramite questa condizione viene settata la variabile *timer* e vengono disattivate le animazioni di caricamento relative agli oggetti di *Layer* diversi da quello corrente.
La seconda condizione è dedicata ai controlli; viene dettato il comportamento che deve avere il programma quando il *Raycast* colpisce un oggetto diverso da quello colpito in precedenza.
Facciamo un esempio pratico per comprendere quali sono le operazioni di controllo di cui si sta parlando.
Quando il *Raycast* colpisce un oggetto del *Layer* "OggettoInterattivo" per più di tre secondi, quest'ultimo viene attivato, quindi vengono abilitate le sue animazioni, i suoi suoni, il suo panello,ecc.. Quando l'utente vuole attivare un nuovo oggetto tutte le proprietà del precedente oggetto devono essere automaticamente disattivate altrimenti si creerebbe del caos.
I controlli di tale porzione di codice sono atti a svolgere questo tipo di compito.
Nella terza condizione viene definito il comportamento assunto dall'applicazione nel caso in cui l'operazione di *Raycast* interseca lo stesso oggetto per più *frame* conseguenti. Quindi il tempo viene decrementato ad ogni frame fino a quando quest'ultimo raggiunge un valore inferiore allo zero. In tal caso vengono inviati dei messaggi allo script allegato all'oggetto colpito, viene disattivata l'animazione di caricamento, e vengono settate delle variabili utilizzate da altre porzioni di codice.

3.5.3 Intersezione con il layer "Exit"

Quando il raggio di *Raycast* si interseca con l'oggetto a cui è assegnato il *Layer* "Exit" (bottone di chiusura del pannello), vengono eseguiti i comandi previsti per questo *Layer*.

Tramite il codice scritto per questa condizione è possibile disattivare l'oggetto interattivo attivato, quindi chiudere il pannello di descrizione, interrompere i suoni emessi e le animazioni ad essi associati.

Come per gli oggetti facenti parte del *Layer* "OggettoInterattivo", anche in questo caso prima di eseguire le istruzioni, a seguito dell'intersezione del raggio con il bottone di chiusura, viene fatto trascorrere un secondo e mezzo.

Anche questa attesa è segnalata con un'animazione fatta sul mirino, molto simile a quella spiegata precedentemente, con la differenza che cambia l'icona e dura la metà del tempo.

Osserviamo il codice ricordando che il *Layer* "Exit", all'interno del progetto, ha come identificativo il numero 10:

```
...else if (hit.collider.gameObject.layer == 10) {
    // caso in cui il raggio interseca per la prima volta layer
    "Exit", dopo aver intersecato un oggetto appartenente ad un
    altro layer

    if (layercollision != 10) {
        // vengono disattivate tutte le animazioni di caricamento
        riguardanti ad altri layer

        anim.SetBool ("Inquadrato", false);
        anim.SetBool("PlayAnimation",false);
        anim.SetBool("Close",false);
        anim.SetBool("walk",false);
        // viene impostato il timer a tre secondi
        U_time = 1.5f;
    }
```

```
// al primo frame viene settato questo valore in modo tale da
non rientrare più nella condizione precedente nei successivi
frame
layercollision = 10;
//viene resa attiva l'animazione di caricamento per il layer
"Exit"
anim.SetBool ("Close", true);
//ad ogni frame viene decrementato il tempo
U_time -= Time.deltaTime;
//quando la varibile U_time ha raggiunto un valore inferiore
a 0.75    e quindi è passato un secondo e mezzo vengono
effettuate   le seguenti   istruzioni.
if (ObjectH != null && U_time <= 0.75) {
     // viene disattivato l'eventuale audio dell'oggetto
     attivo
     if(ObjectH.audio!=null){
          ObjectH.audio.enabled = false;
     }
     //viene disattivata l'eventuale animazione relativa
     all'oggetto inviando un messaggio allo script allegato
     all'oggetto stesso
     if(ObjectH.GetComponent<Animator>()!= null ){
          ObjectH.SendMessage("DisactiveAnimation");
          AnimazioneAttiva = false;
     }
     //viene annullata la variabile che memorizzava l'oggetto
     attivo
     ObjectD = null;

     //viene settata a false la variabile che serve per il
     posizionamento del pannello di descrizione

     setpos = false;
     //viene inviato un messaggio allo script
     ObjectH.SendMessage ("Setpos", setpos);
```

```
            //viene rimossso il pannello di descrizione dalla scena
            ObjectH.SendMessage ("HidePanel");
            //viene impostata la variabile booleana che esprime se il
                  pannello di descrizione è aperto o chiuso
            description = false;
            //viene interrotta l'animazione di caricamento
            anim.SetBool ("Close", false);
        }
} else if..
```

3.5.4 Intersezione con il layer"Percorso"

Se si entra in questa parte di codice significa che è avvenuta un'intersezione del raggio di *Raycast* con una delle pedane che permette il movimento dell'*OVRPlayerController* all'interno della scena.
Anche in questo caso vi è un'attesa di tre secondi prima che l'*OVRPlayerController* si sposti e il tutto viene segnalato con un'apposita animazione di caricamento.
Una volta passati i tre secondi questo script invia due messaggi allo script allegato al *GameObject OVRPlayerController*, il quale provvederà a spostarlo in scena.
Ricordiamo che se l'*OVRPlayerController* è già in movimento l'intersezione del raggio di *Raycast* con la "pedana di movimento" non viene considerata. Pertanto non si entrerà in questa parte di codice.
Osserviamo il codice ricordando che il *Layer* "Percorso", all'interno del progetto, ha come identificativo il numero 11:

```
...else if (hit.collider.gameObject.layer == 11 && move == false)
    {

        // caso in cui il raggio interseca per la prima volta
        il layer "Percorso", dopo aver intersecato un oggetto
        appartenente ad un altro layer

        if (layercollision != 11)
        {
              //viene settato il tempo a tre secondi
              U_time = 1.5f;
              //vengono disattivate le animazioni relative al
              caricamento di altri layer
              anim.SetBool ("Inquadrato", false);
        }
        // al primo frame viene settato questo valore in modo

    tale da non  rientrare più nella condizione precedente nei
    successivi frame
```

```
        layercollision = 11;

        //ad ogni frame viene decrementato il tempo
        U_time -= Time.deltaTime;

        //viene attivata l'animazione per il caricamento
        relativo al percorso
        anim.SetBool ("walk", true);

        //trascorsi i tre secondi si entra in questa condizione
        if ( U_time <= 0.0f) {
              //viene disattivata l'animazione di caricamento del
                 movimento
              anim.SetBool ("walk", false);
              //viene memorizzato il punto di arrivo dello
              spostamento
              EndPosition =
              hit.collider.gameObject.transform.position;
              //viene inviato un messaggio
              all'ovrplayercontroller con la posizione di
              destinazione, lo script destinatario
              provvederà a spostarlo verso quella direzione.
              GameObject.Find("OVRPlayerController").
              SendMessage("MoveP",EndPosition);

              // viene impostata questa variabile a true per
              segnalare che il gameobject ovrplayervcontroller si
              sta muovendo
              move=true;
        }
    }else if...
```

3.5.5 Intersezione con il layer"Animation"

Il raggio di *Raycast* si interseca con l'oggetto di *Layer* "Animation" ogniqualvolta che l'osservatore rivolge lo sguardo verso il bottone che attiva l'animazione di un oggetto interattivo.

Mentre per gli oggetti che sono dotati solamente di un suono, quest'ultimo si avvia automaticamente all'attivazione dell'oggetto stesso, per gli oggetti che possiedono sia suono che animazione non è cosi. Infatti , è possibile godersi l'animazione e il suono di un oggetto solamente attivandoli tramite un apposito bottone.

Il bottone di cui si sta parlando compare a fianco del bottone di chiusura del pannello e fa parte dell'interfaccia.

Anche in questo caso, prima di eseguire i comandi previsti per questa condizione vi è un'attesa di un secondo e mezzo, segnalata tramite un'animazione.

Ora vediamo il codice eseguito nel caso in cui il raggio si intersechi con il bottone di animazione avente il *Layer* "Animation".

Ricordiamo che il *Layer* in questione è identificato all'interno al progetto con il numero 12:

```
...else if (hit.collider.gameObject.layer == 12) {

    // caso in cui il raggio interseca per la prima volta
    il layer "Animation", dopo aver intersecato un oggetto
    appartenente ad un altro layer
    if (layercollision != 12)
    {
        //viene settato il tempo a tre secondi
        U_time = 1.5f;
        //vengono disattivate le animazioni relative al
        caricamento di altri layer
        anim.SetBool ("Inquadrato", false);
        anim.SetBool("wlak",false);
        anim.SetBool("Close",false);
    }
```

```
    // al primo frame viene settato questo valore in modo
    tale da non  rientrare più nella condizione precedente
    nei successivi frame
    layercollision = 12;

    //se l'animazione non è ancora attiva
    if(Active_Play == false)
    {
        //parte l'animazione di caricamento per questo layer
        anim.SetBool("PlayAnimation",true);

        //viene decrementato il timer
        U_time -= Time.deltaTime;
        // se è passato un secondo e mezzo e c'è un oggetto
        interattivo attivo
        if ( ObjectD!=null&& U_time<= 0.75 )
        {
            //interrompi l'animazione di caricamento
            anim.SetBool ("PlayAnimation", false);
            //invia un messaggio allo script relativo
            all'oggetto attivo per attivare l'animazione
            ObjectH.SendMessage("ActiveAnimation");
            //viene impostata a true questa variabile per non
            entrare     più in questa sezione
            Active_Play=true;
            //viene attivato l'audio
            ObjectD.audio.enabled = true;
        }
    }
}
```

3.5.6 Intersezione con il resto della scena

Ogni volta che il raggio di *Raycast* non si interseca con nessun oggetto, o che si interseca con oggetti a cui è stato assegnato un *Layer* diverso da quelli descritti precedentemente, il programma non deve assumere nessun comportamento particolare. In questo caso il suo compito sarà quello di resettare il timer, e disattivare eventuali animazioni di caricamento già attive.

3.5.7 Attivazione e disattivazione delle pedane di movimento

Come è stato detto nel primo capitolo, il movimento dell'*OVRPlayerController* all'interno della scena può essere gestito in diversi modi.
Se, per qualche motivo, l'utilizzatore decidesse di usare un joystick al posto delle pedane di movimento per spostarsi in scena, è stata studiata una soluzione che permette di rimuoverle dalla vista. Premendo il tasto "p" le pedane di movimento scompaiono dalla scena. Il tutto è gestito sempre all'interno del metodo *Update()*.
Vediamo il codice:

```
// Abilitare  e disabilitare i segnali di percorso
if (Input.GetKeyDown ("p"))
{
    GameObject[] objpercorso =
    GameObject.FindGameObjectsWithTag("Respawn");
    //Variabile esposta dello script
    PathEnable=!PathEnable;
    foreach (GameObject go in objpercorso)
    {
        go.renderer.enabled = PathEnable;
        go.collider.enabled = PathEnable;
    }}
```

3.5.8 Animazioni di caricamento

Nel corso dei precedenti paragrafi si è visto che quando il raggio di *Raycast* interseca alcuni degli oggetti in scena si attiva un'animazione di caricamento nell'attesa di compiere determinate operazioni (es: apertura pannello, chiusura pannello, ecc.).
Ad ogni *Layer* è associata una propria animazione di caricamento, in modo tale da contraddistinguere il tipo di oggetto che si sta attivando.
L'unico elemento che differenzia tra di loro le animazioni di caricamento è l'icona che viene applicata sullo sfondo.

Fig 3.9 - Le icone delle diverse animazioni di caricamento.

L' oggetto coinvolto in queste animazioni è il mirino che viene scalato e trasformato per simulare un caricamento.
Ora vediamo tramite una sequenza di immagini una delle animazioni di caricamento utilizzate nell'applicazione:

Fig 3.10 - Animazione "Loading"applicata al mirino.

Per gestirne tutte le animazioni , al mirino è stato assegnato un *Animation Controller* tramite la *component* "Animator".

Per coordinare le transizioni tra un'animazione e l'altra è stato creato il seguente *State Machine:*

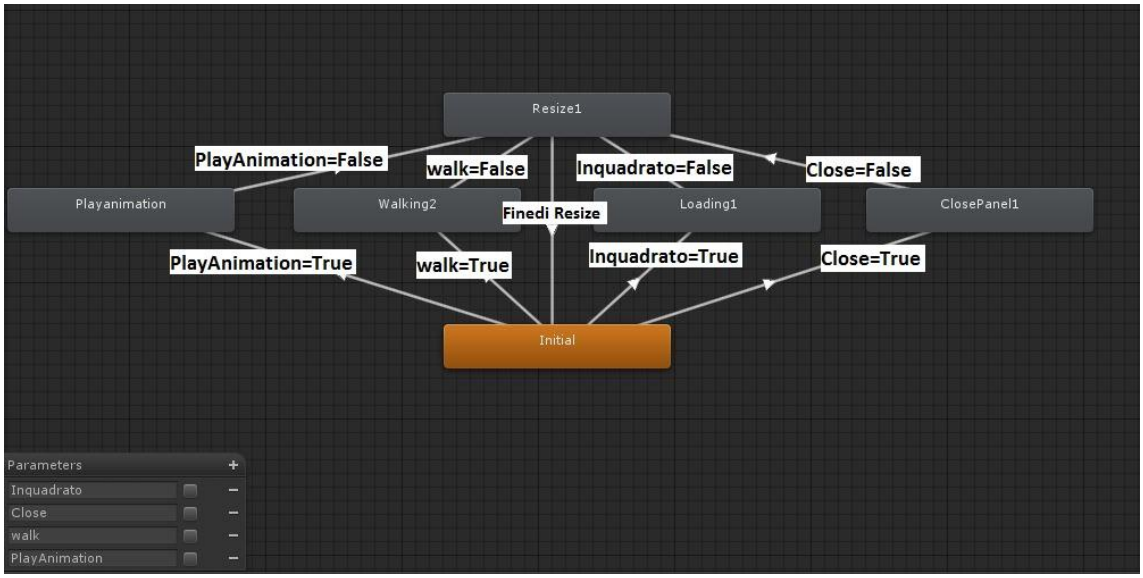


Fig 3.11 - State machine relativo al GameObject "Mirino".

Nella figura 3.11 , in basso a sinistra, si possono notare i nomi di quattro variabili booleane che vengono richiamate nel codice dello script *OggettoInquadrato.cs.*

Si passa da uno stato all'altro ogniqualvolta che il valore delle variabili booleane viene alterato.

Come si può notare, ogni volta che si attiva un'animazione di caricamento il mirino viene modificato tramite l'animazione e successivamente riportato alla sua forma originale tramite un'ulteriore animazione: *Resize*.

3.6 Lo script "OpenLabel.cs"

Il secondo script che sarà analizzato è lo script "*OpenLabel.cs*", questo script è una componente di ogni oggetto che fa parte del *Layer* "Oggetto Interattivo".
Il compito principale di questo script è quello di gestire un pannello che contiene dati relativi all'oggetto colpito.
Nel precedente paragrafo abbiamo visto che lo script "*OggettoInquadrato.cs*" determina su quale oggetto si è soffermato l'osservatore per poi inviargli dei messaggi richiamando delle funzioni.
A rispondere ai messaggi inviati dal precedente script è "*OpenLabel.cs*" che è allegato ad ogni oggetto che deve fornire una descrizione di esso.
Ora analizziamo nel dettaglio questo script.

3.6.1 Assegnazione dello script "OpenLabel.cs"

Nel paragrafo 3.4 abbiamo visto che gli oggetti che sono in scena sono suddivisi in base a dei *Layer*.
Per un funzionamento corretto dell'applicazione ad ogni oggetto che fa parte del *Layer* "Oggetto Interattivo" deve essere assegnata la *component* script "*OpenLabel.cs*". Osserviamola:

Fig 3.12 - La component OpenLabel.cs.

Notiamo subito dall'immagine 3.12 che nello script sono presenti più variabili esposte. Queste variabili servono ad immettere le informazioni che riguardano l'oggetto a cui è stato assegnato lo script. In particolare vediamo tra di esse: il nome dell'oggetto colpito, la descrizione, la presenza di un'eventuale elemento audio associato ad esso e la presenza di un'eventuale animazione.
In conclusione per rendere un oggetto interattivo all'interno della scena è sufficiente assegnargli il *Layer* "OggettoInterattivo" e allegargli la *component* "*OpenLabel.cs*".

3.6.2 Struttura generale

Differentemente dallo script "OggettoInquadrato.cs" in questo script è utilizzato solo uno dei metodi derivanti dalla classe *MonoBheavior*: il metodo *Start()*.
Anche in questo caso, questo metodo è utilizzato per inizializzare delle variabili definite globali.
Vediamo il nel dettaglio il contenuto del metodo *Start()*:

```
void Start () {

    //Memorizza nella variabile CanvasSchermo il Pannello nel
    quale vengono scritte le informazioni

    CanvasSchermo = GameObject.Find("DescriptionPanel").
    GetComponent<Canvas> ();

    //All'avvio rende invisibile il pannello
    CanvasSchermo.enabled = false;

    //Rimuove il collider del pannello all'avvio
    CollidCan = CanvasSchermo.GetComponent<BoxCollider>();
    CollidCan.enabled = false;
```

```
    //se l'oggetto è dotato di animazione inizializza la   variabile
    ObjDescription con la component animator
    if (Animazione == true) {
        ObjAnimation = this.gameObject.GetComponent<Animator> ()
            }
}
```

Il resto dello script è composto da funzioni che vengono richiamate dallo script "OggettoInquadrato.cs" o da altre funzioni di questo script.

Tali funzioni sono:

- *HitObject*: questa funzione viene richiamata dallo script "OggettoInquadrato.cs" ogni volta che un oggetto del *Layer* "OggettoInterattivo" viene attivato.

```
    public void HitObject (Vector3 raydirection) {
        // se la posizione del pannello di descrizione non
        è ancora stabilita per quell'oggetto
        if(!setpos)
        {
            //richiama la funzione che posiziona il pannello
            PositionPanel(raydirection);
        }
        // scrivi all'interno del pannello le informazioni
        relative all'oggetto inquadrato
        WriteInPanel();
        // se l'oggetto attivato è dotato di una componente
        audio, attivala
        if (audio) {
            this.gameObject.audio.enabled = true;
        }
    }
```

- *WriteInpanel*: questa funzione viene richiamata dalla precedente e serve a riportare nel pannello tutti i dati e le informazioni relative all'oggetto attivato.

```
void WriteInPanel(){
    //abilito il pannello
    CanvasSchermo.enabled = true;
    CollidCan.enabled = true;

    // scrivo nel pannello la descrizione dell oggetto
    Font arialnormale = Resources.Load("ARIAL",  typeof(Font))as
    Font;

    CanvasTesto = GameObject.Find
    ("CText").GetComponent<Text> ();

    CanvasTesto.font = arialnormale;
    CanvasTesto.text = ObjDescription;

    // scrivo nel pannello il titolo dell oggetto
    Font arialgrassetto = Resources.Load("ARIALBD",
    typeof(Font))as Font;
    CanvasTitle = GameObject.Find
    ("CTitle").GetComponent<Text> ();
    CanvasTitle.font = arialgrassetto;
    CanvasTitle.text = ObjName;

    //attiva i bottoni di interfaccia quando si apre il
    pannello il primo bottone viene attivato solamente se è
    disponibile  l'animazione per quell'oggetto
    GameObject.Find ("DescriptionPanel").transform.FindChild
    ("ButtonAnimation").gameObject.SetActive(Animazione);

    GameObject.Find ("DescriptionPanel").transform.FindChild
    ("PlaneExit").gameObject.SetActive(true);}
```

- *Hidepanel*: questa funzione è richiamata dallo script "OggettoInquadrato.cs" quando viene disattivato il pannello di descrizione. Ricordiamo che quest'ultimo viene chiuso quando ci si sposta in scena, quando si attiva un altro oggetto, oppure quando viene premuto il bottone apposito.

Vediamo la funzione che svolge effettivamente questo compito:

```
public void HidePanel()
{
    //disattiva il pannello
    CanvasSchermo.enabled=false;
    CollidCan.enabled = false;

    //disattiva i bottoni dell'interfaccia
    GameObject.Find ("DescriptionPanel").transform.FindChild
    ("ButtonAnimation").gameObject.SetActive(false);

    GameObject.Find ("DescriptionPanel").transform.FindChild
    ("PlaneExit").gameObject.SetActive(false);
}
```

- *Setpos* : questa funzione è richiamata dallo script "*OggettoInquadrato.cs*" e serve ad assegnare un valore alla variabile *"setpos"*. Questa variabile è dichiarata globalmente ed è utilizzata dalla funzione *HitObject*. Se il valore di tale variabile risulta essere positivo, al pannello viene assegnata una nuova posizione.

  Osserviamo la funzione *Setpos*:

```
public void Setpos(bool pos)
    {
        setpos = pos;
    }
```

- *PositionPanel*: tale funzione viene richiamata dalla funzione *HitObject* se il valore booleano *"setpos"* risulta essere positivo.

  Quest'ultima serve ad assegnare al pannello di descrizione una posizione corretta rispetto quella dell'osservatore e al suo orientamento.

```
void PositionPanel(Vector3 raydirection)
{
    //player è una variabile dove è memorizzato il gameobject
    OVRPlayerController
    player=GameObject.Find("OVRPlayerController");

    //viene definita la posizione del pannello in scena
    Vector3 direzm = new Vector3(0.7f,0.0f,0.7f);
    CanvasSchermo.transform.position = new Vector3
    (player.transform.position.x,1.2f,player.transform.positi
    on.z)+ Vector3.Scale(raydirection,direzm);

    //viene definito l'orientamento del pannello
    CanvasSchermo.transform.rotation = new Quaternion (0,
    target.transform.rotation.y, 0,
    target.transform.rotation.w) *new Quaternion(0,180,60,0);

    //viene abilitato il bottone di animazione se presente  per
    quell'oggetto
    GameObject.Find ("DescriptionPanel").transform.FindChild
    ("ButtonAnimation").gameObject.SetActive(Animazione);
}
```

- *ActiveAnimation*: questa funzione è richiamata dallo script "*OggettoInterattivo.cs*" nel caso in cui il raggio di *Raycast* intersechi il bottone di avvio animazione. Il compito che ha questa porzione di codice è di attivare l'animazione relativa all'oggetto attivo.

```
void ActiveAnimation(){
    ObjAnimation.SetBool ("ActiveAnim", true);
}
```

- *DisactiveAnimation*: questa funzione svolge il compito inverso della funzione *ActiveAnimation*, infatti, si occupa di disattivare l'animazione relativa ad un oggetto ogniqualvolta che esso viene attivato.

```
void DisactiveAnimation(){
    ObjAnimation.SetBool ("ActiveAnim", false);
```

3.7 Lo Script "MovePlayer.cs"

Il terzo ed ultimo script analizzato è lo script *MovePlayer.cs*. Quest'ultimo viene utilizzato nel caso in cui lo spostamento in scena dell'*OVRPlayerController* è gestito tramite la solita operazione di *Raycast*.
Questo script che è assegnato come componente all'*OVRPlayerController* riceve dei messaggi dallo script "*OggettoInquadrato.cs*" quando il raggio di *Raycast* interseca un oggetto che fa parte del Layer "*Percorso*".

3.7.1 Struttura Generale

Questo script è composto dai due metodi *Start()* e *Update()* derivanti dalla classe *MonoBheavior* e da un'ulteriore funzione *"MoveP"* che riceve i messaggi dallo script "OggettoInquadrato.cs".
Iniziamo a descrivere lo script osservando il primo metodo citato: *Start()*. Come si è ben capito, questo metodo serve ad inizializzare dei valori all'avvio dell'applicazione.

```
void Start () {
    //memorizza la posizione iniziale dell'OVRPlayerController
    startPosition = this.gameObject.transform.position;
    //quando move è false vuol dire che l'OVRPlayerController è
    fermo in scena
    move = false;
    //impostiamo la velocità di spostamento da un punto ad
    un'altro
    speed = 0.2f;
}
```

Procediamo ora con l'analizzare il metodo *Update()*, che viene richiamato ad ogni frame durante l'esecuzione dell'applicazione.

```
void Update () {
    //ad ogni frame viene memorizzata la posizione attuale
    dell'OVRPlayerController nella variabile startPosition
    startPosition = this.gameObject.transform.position;

    //endPosition è la variabile che memorizza le coordinate del
    punto di destinazione
    if (endPosition != null && move==true)
    {
        // gestione dello spostamento dal punto startPosition al
        punto endPosition
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / journeyLength;
        transform.position = Vector3.Lerp (startPosition,
        endPosition, fracJourney);

        // se l'OVRPlayerController è arrivato a destinazione
        if (this.gameObject.transform.position == endPosition)
        {
            startPosition = endPosition;
            move=false;
        }

    }
}
```

Si può notare che tramite questo metodo avviene l'effettivo spostamento in scena dell'*OVRPlayerController*, tramite un'approssimazione del percorso da un punto iniziale ad uno finale.

L'ultima funzione che compone questo script è la funzione "*MoveP*", che riceve messaggi dallo script "*OggettoInterattivo.cs*" in caso di intersezione con una pedana di movimento. La funzione non fa altro che aggiornare il valore della variabile globale "*endPosition*"(utilizzata anche dal metodo *Update()*) ed inizializzare il valore delle

variabili *startTime* e *journeyLenght* se il punto di destinazione attuale è diverso da quello memorizzato precedentemente. Osserviamola:

```
public void MoveP (Vector3 Position)
{
     move = true;
     endPosition = new Vector3 (Position.x,
     this.gameObject.transform.position.y, Position.z);

     if (precPosition != endPosition)
     {
          startTime = Time.time;
          journeyLength = Vector3.Distance
          (this.transform.position,endPosition);
          precPosition = endPosition;
     }
}
```

Capitolo 4

Risultato finale e sviluppi futuri

In questo quarto ed ultimo capitolo viene illustrata tramite delle immagini l'applicazione in esecuzione e vengono riportate alcune idee per possibili sviluppi futuri.

4.1 Applicazione in esecuzione

L'utente che decide di utilizzare questa applicazione deve indossare il dispositivo Oculus collegato ad un PC, dove sta girando la suddetta applicazione.
Una volta indossato il dispositivo, l'utente è immerso in un ambiente virtuale che simula la stanza di un museo.

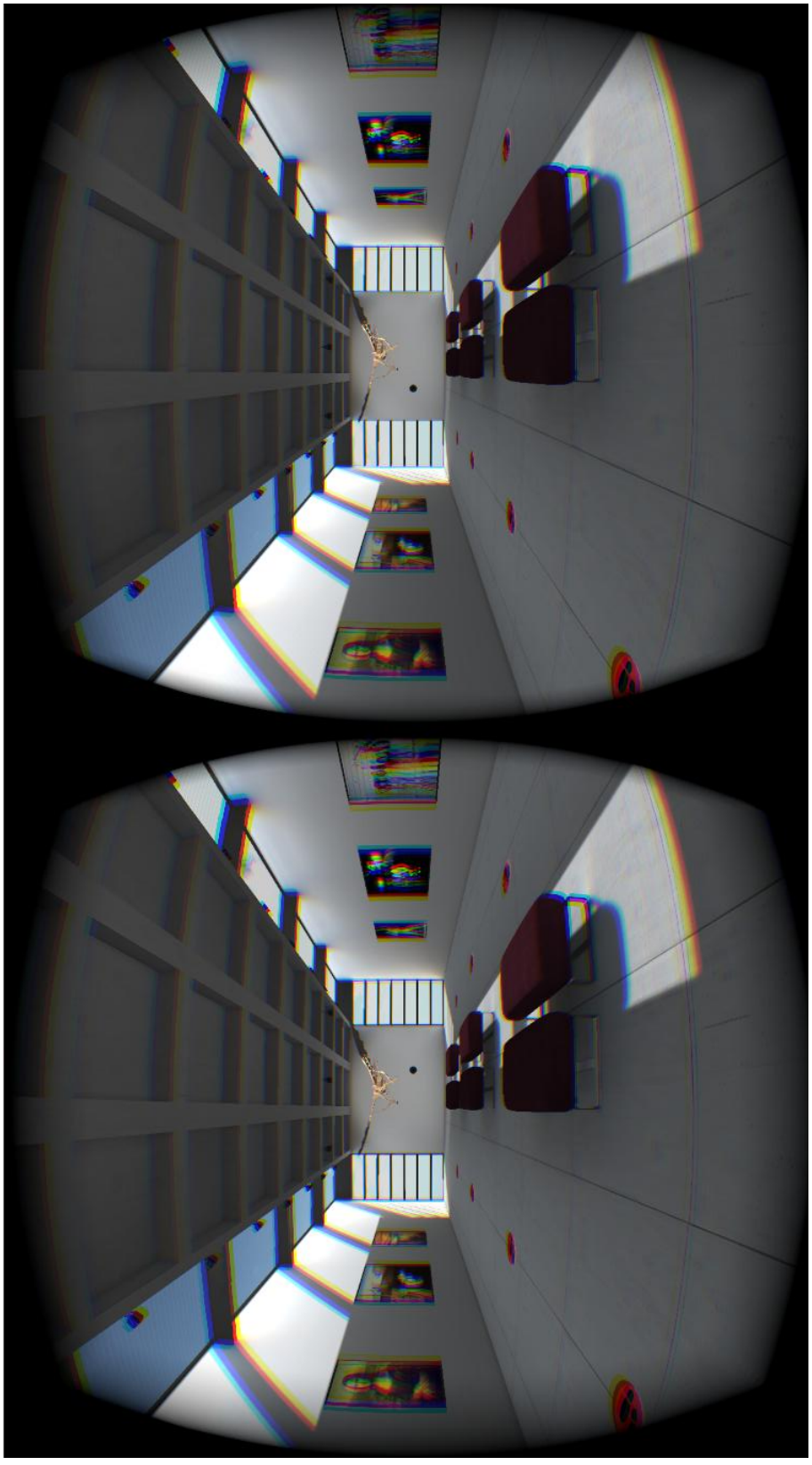


Fig 4.1 - Ambiente virtuale realizzato.

A questo punto l'utente può muoversi all'interno dell'ambiente e avvicinarsi agli oggetti di interesse orientando lo sguardo verso una delle pedane di movimento.

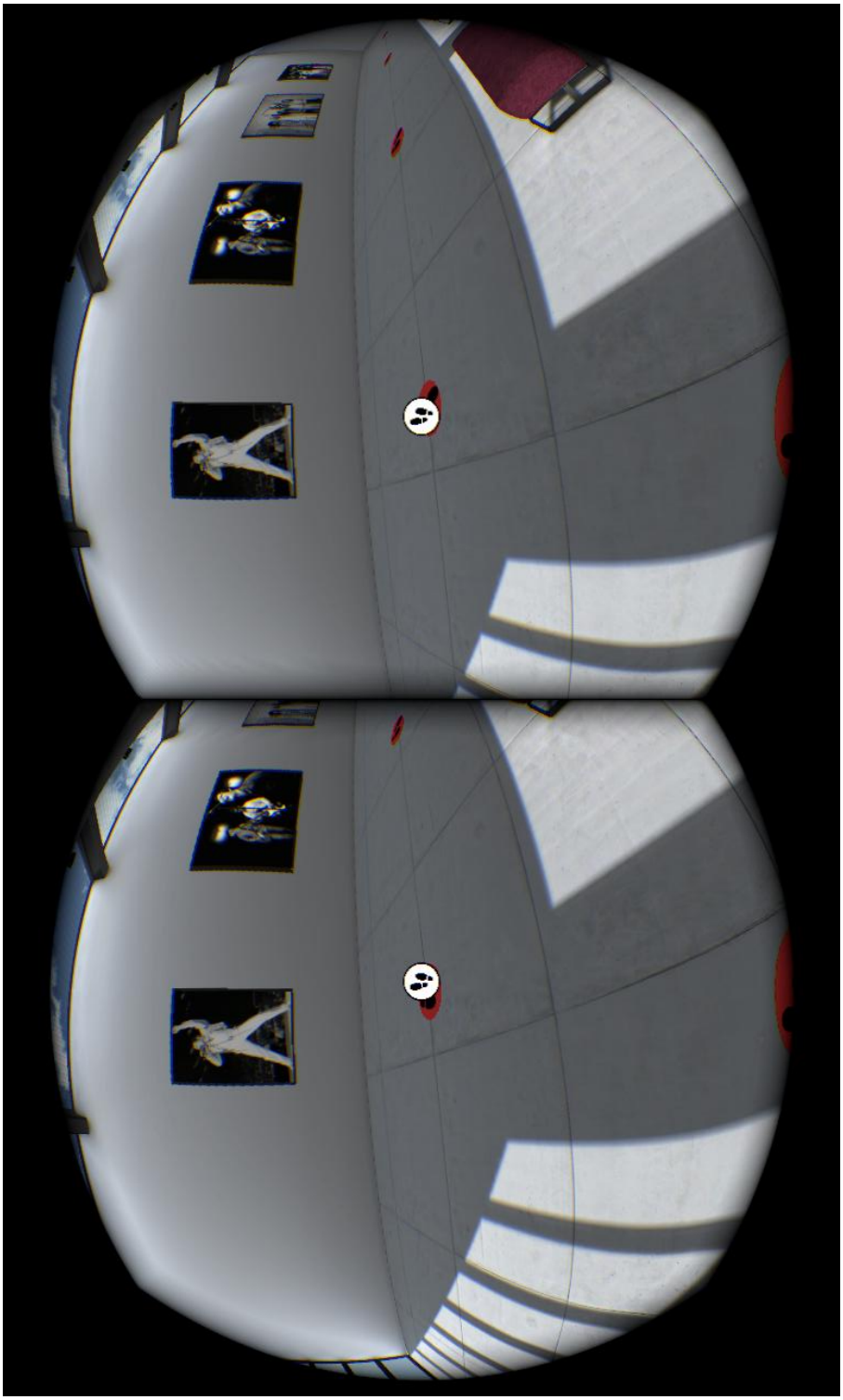


Fig 4.2 - Selezione pedana di movimento.

L'*OVRPlayerController* si sposta in scena sino ad arrivare al punto inquadrato e una volta giunto a destinazione l'utente può selezionare l'oggetto di suo interesse per coglierne delle informazioni. Quest'ultime appaiono in un pannello, la cui posizione in scena varia in base quella dell'*OVRPlayerController*.

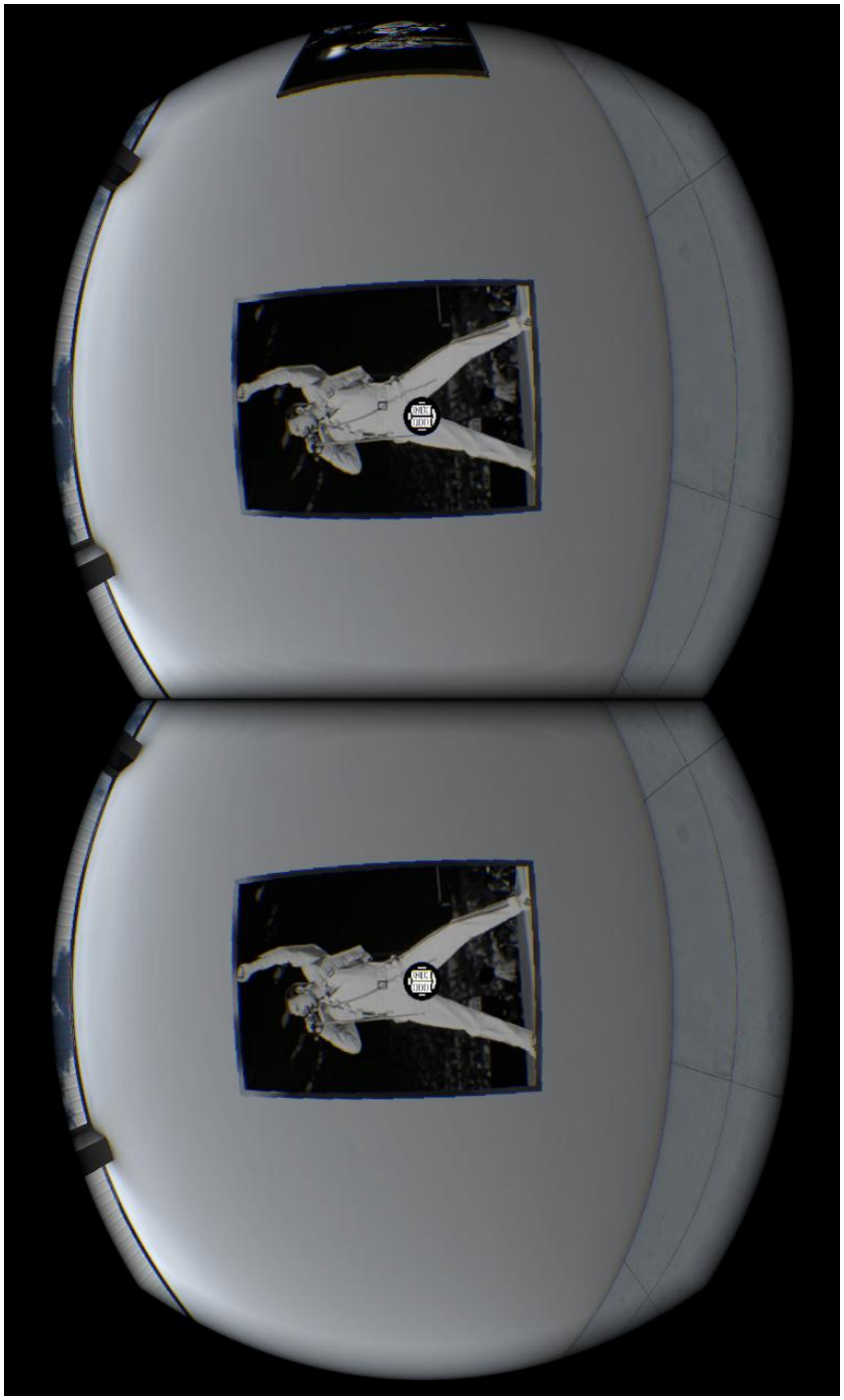


Fig 4.3 - Selezione di un oggetto.

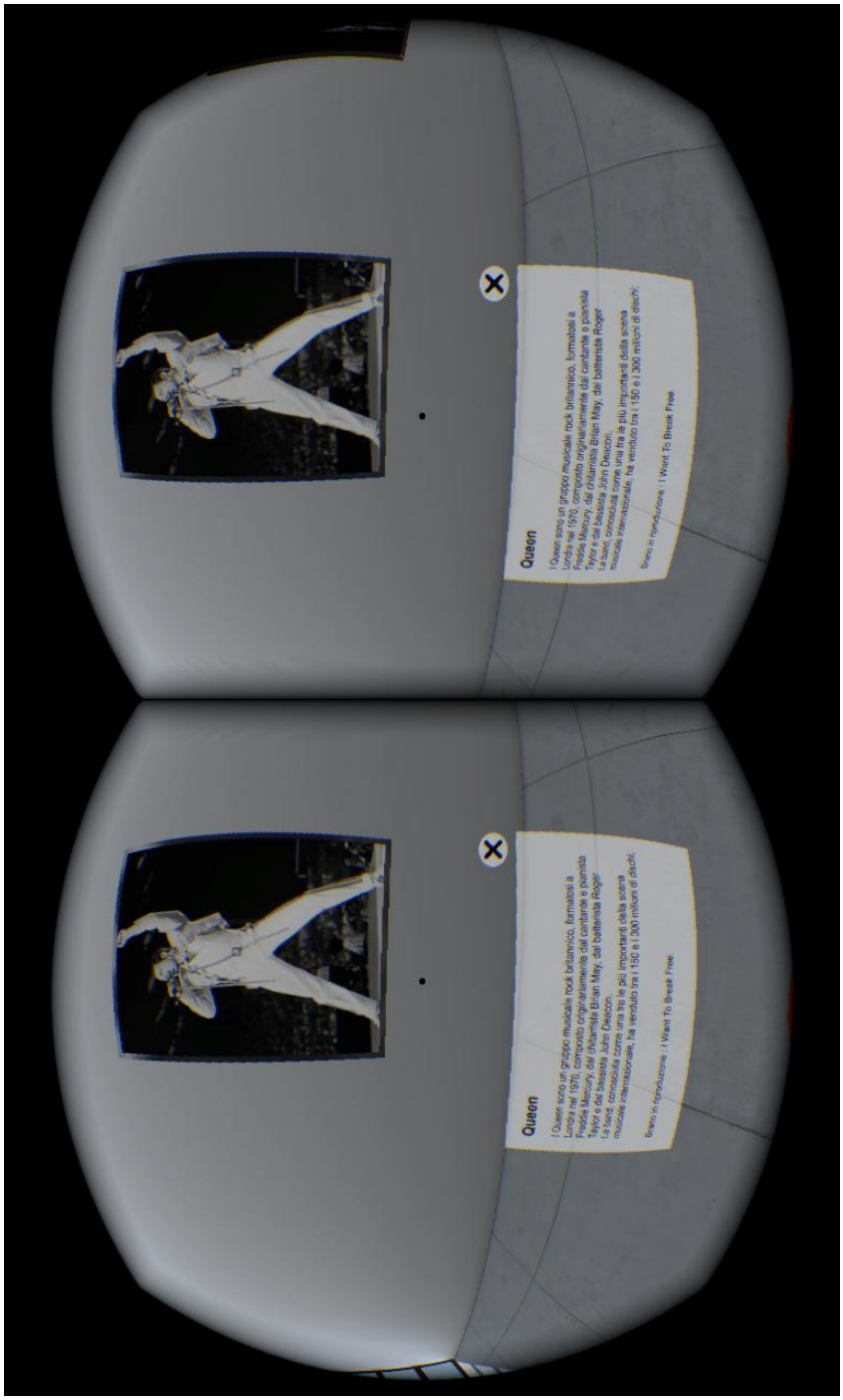


Fig 4.4 -Attivazione di un oggetto.

Per chiudere il pannello di descrizione è sufficiente muoversi in scena, oppure utilizzare l'apposito bottone di chiusura.

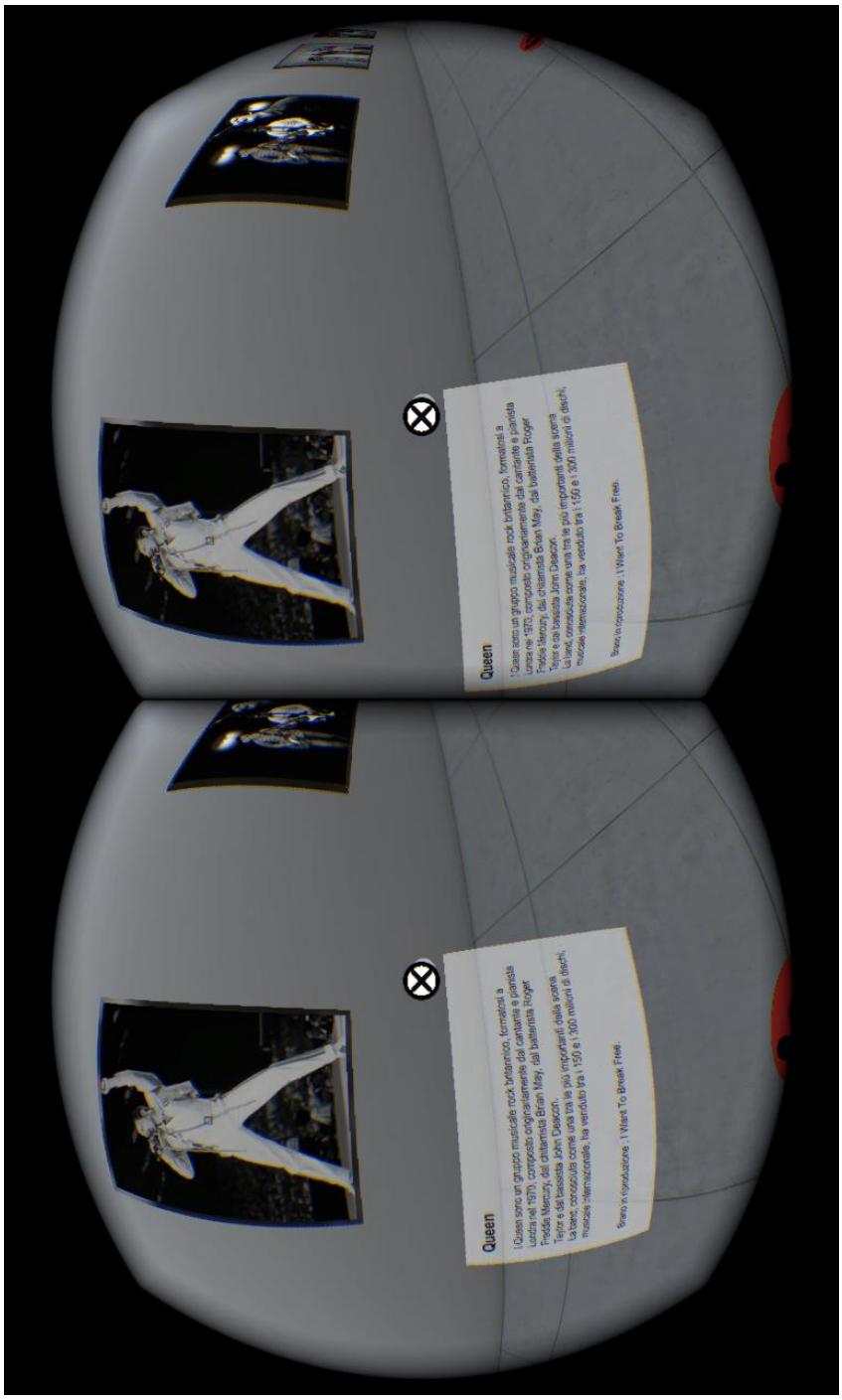
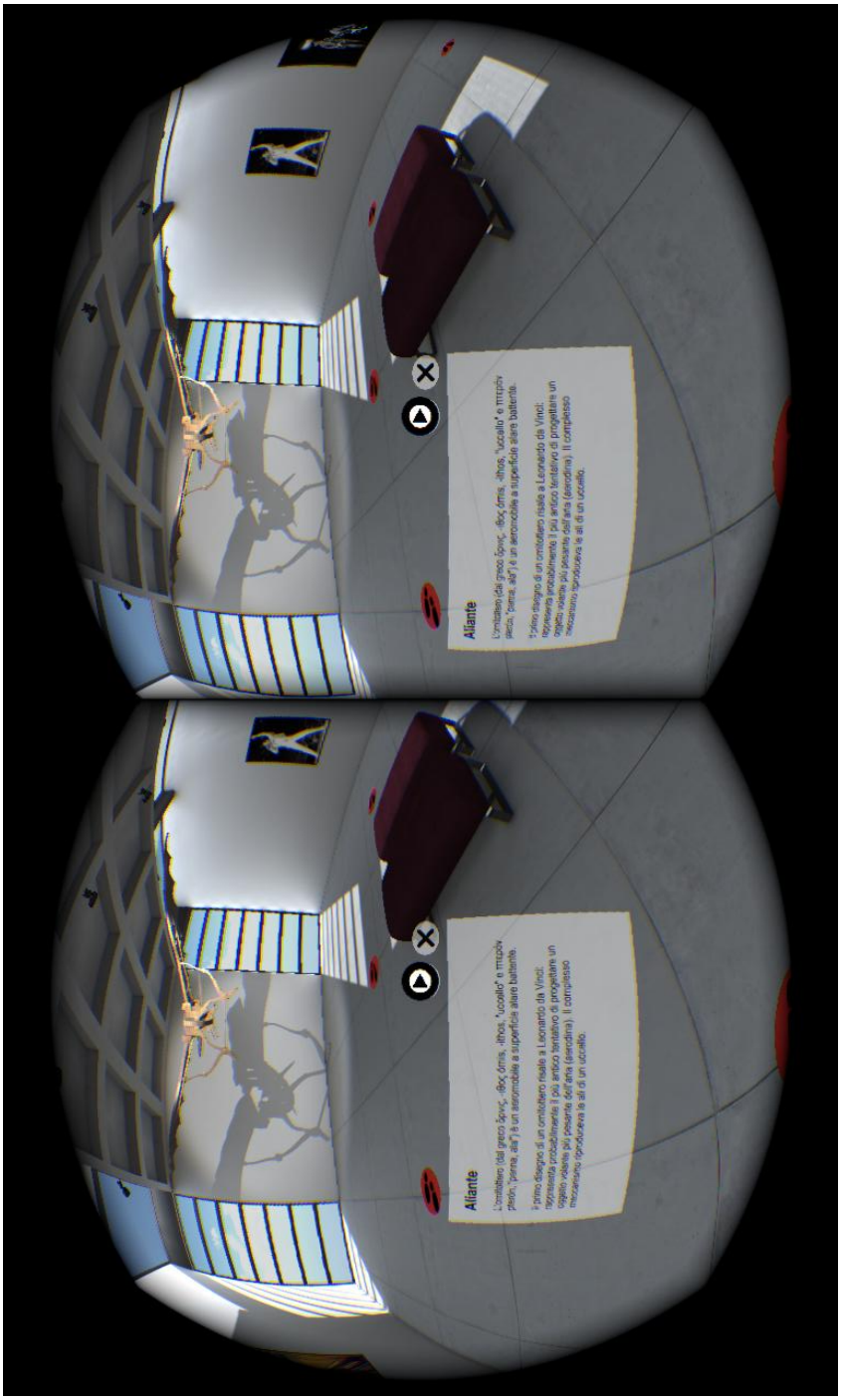


Fig 4.5 - Selezione del pulsante di chiusura.

L'ultima cosa che rimane da illustrare è la selezione del bottone di attivazione dell'animazione di un oggetto.



Aliante
Aliante

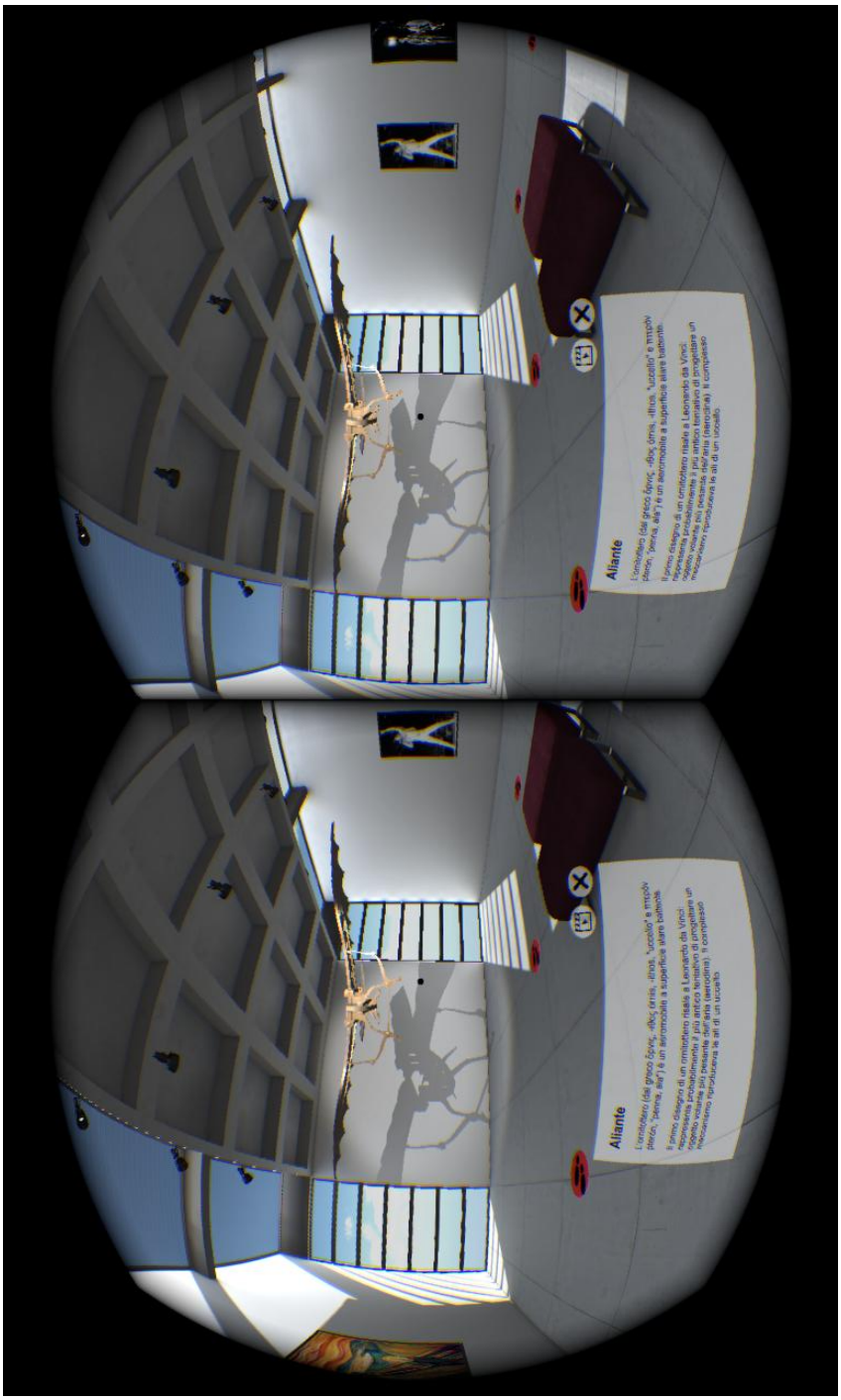


Fig.re 4.6 - Selezione del pulsante di attivazione dell'animazione dell'aliante.

Purtroppo in questo materiale cartaceo non è possibile riportare l'effetto sonoro, ma è bene ricordare che nell'attivazione della sua animazione, la macchina di Leonardo Da Vinci, oltre a muovere le ali, riproduce anche un suono.

4.2 Possibilità di utilizzo

L'applicazione sviluppata è un prototipo di quella che potrebbe essere un'applicazione utilizzata da moderni musei per illustrare e riprodurre antiche ambientazioni. Per esempio, un ambiente sintetico da proporre potrebbe essere una stanza di una "*Domus*" (tipologia di abitazione utilizzata al tempo degli antichi romani), nel quale il "visitatore virtuale" può aggirarsi e scoprire quali erano le funzionalità di determinati oggetti che compongono l'ambiente.

4.3 Sviluppi Futuri

Quest'applicazione sicuramente non può essere definita completa, quindi i miglioramenti che possono essergli apportati sono molteplici, sia a livello di codice che a livello di ambiente grafico. Per esempio potrebbe essere implementato uno *shader di otuline* che permetterebbe di capire meglio quale oggetto della scena si sta selezionando.

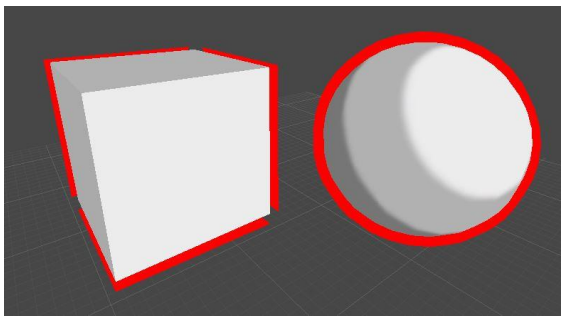


Fig 4.7 - Esempio di shader di outline applicato.

Oltre a questo tipo di miglioramenti, si potrebbero prevedere delle modifiche in modo tale che essa possa esser compatibile con diversi dispositivi.

Come è stato detto precedentemente OculusDK2 è uno dei miglior visori prodotti sul mercato, ma ha una limitazione: il cablaggio e la necessità di essere collegato ad un PC.

Attualmente l'applicativo è compatibile solamente con questo tipo di visore, in un futuro si potrebbe generalizzarla in modo tale da poterla utilizzare con diversi dispositivi (anche mobile).

Infinte un'altro sviluppo da considerare potrebbe essere quello di integrare altri dispositivi che permettono di rendere l'esperienza più immersiva (Leap Motion, Virtux Omni, ecc.).

Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno aiutato nella realizzazione di questa tesi.

Ringrazio anzitutto Marco e Federico dello Studio Rossetti che mi hanno fornito l'idea, il materiale e dedicato il loro tempo per dare vita a questa tesi.

Proseguo con il ringraziare la professoressa Damiana Lazzaro, relatrice di questa tesi, per la grande disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la stesura.

Un ringraziamento particolare va ai miei amici e ai miei colleghi che mi hanno sempre incoraggiato o speso parte del loro tempo per leggere e discutere con me le bozze del lavoro.

Vorrei ringraziare infine le persone a me più care, la mia famiglia, e mia nonna, a cui questo lavoro è dedicato.

Bibliografia e Sitografia

[1] Brooks, F. P. (1999). What's real about virtual reality? IEEE Computer Graphics and Applications, 19(6), 16-27.

[2] Burdea, G. C., & Coiffet, P. (2003). Virtual reality technology (2nd ed.). New Brunswick, NJ: Wiley-IEEE Press.

[3] Ibi p.24

[4] Nicholas Negroponte, Essere digitali, p.118

[5] Sito ufficiale di OculusVR - *https://www.oculus.com/company*

[6] Sito ufficiale di Dexta Robotics - *http://dextarobotics.com*

[7] Sito ufficiale di Virutx - http://www.virtuix.com/products

[8] Sito ufficiale di Leap Motion - https://www.leapmotion.com

[9] Sito ufficiale di OculusVR - *https://www.oculus.com/dk2*

[10] Articolo di Eurogamer su Project Morpheus - *http://www.eurogamer.net /articles/2014-03-19-sony-announces-project-morpheus-virtual-reality-headset-for-playstation-4*

[11] Sito ufficiale di Google - https*://www.google.com/get/cardboard*

[12] Sito ufficiale di Durovis - *http://www.durovis.com*

[13] Sito ufficiale di Samsung *http://www.samsung.com/global/ microsite/gearvr*

[14] Sito ufficiale di Samsung - *http://www.samsung.com/global/microsite/ galaxynote4/note4_main.html*

[15] Sito ufficiale di Unity3D - *http://unity3d.com/*

[16] Sito ufficiale di Autodesk - *http://www.autodesk.com/products/3ds-max/*

[17] Sito ufficiale di Makey Makey - *http://makeymakey.com/*

[18] Wikipedia, caratteristiche di Unity 3D -*http://it.wikipedia.org/*

wiki/Unity_%28motore_di_gioco%29

[19] Sito ufficiale di Microsoft Windows - *http://windows.microsoft.com/it-it/windows/home*

[20] Sito ufficiale di Apple - *https://www.apple.com/it/osx/*

[21] Sito ufficiale di Linux - *http://www.linux.it/*

[22] Sito ufficiale di Android - *https://www.android.com/*

[23] Sito ufficiale Apple - *https://www.apple.com/it/ios/*

[24] Sito ufficiale di Adobe - *http://www.adobe.com/it/products/flash.html*

[25] Sito ufficiale della Nintendo - *https://www.nintendo.it/Wii/*

[26] Sito ufficiale dell'Xbox - *www.xbox.com*

[27] Sito ufficiale di Sony PlayStation - *https://www.playstation.com/it-it/home/*

[28] Sito ufficiale di Autodesk - *http://www.autodesk.com/products/maya/overview*

[29] Sito ufficiale di Blender - *http://www.blender.org/*

[30] Sito ufficiale di Modo - *http://www.thefoundry.co.uk/ab/modo/*

[31] Sito ufficiale di Adobe Photoshop - *http://www.adobe.com/it/products/photoshop.html*

[32] Sito ufficiale di Adobe Fireworks - *https://creative.adobe.com/it/products/fireworks*

[33] Sito ufficiale di Microsoft Windows (Direct3D) - *http://windows.microsoft.com/en-us/windows7/products/features/directx-11*

[34] Sito ufficiale OpenGL - *https://www.opengl.org/*

[35] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/GameObject.html*

[36] Unity Scripting Reference -*http://docs.unity3d.com/Manual/Prefabs.html*

[37] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/Component.html*

[38] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/MonoBehaviour.html*

[39] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html*

[40] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html*

[41] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/GameObject.SendMessage.html*

[42] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/GameObject.Find.html*

[43] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/GameObject.FindGameObjectsWithTag.html*

[44] Sito ufficiale Monodevelop - *http://www.monodevelop.com/*

[45] Definizione .NetFramework - *http://it.wikipedia.org/wiki/.NET_Framework*

[46] Unity Scripting Reference - *http://docs.unity3d.com/Manual/class-Animator.html*

[47] Unity Scripting Reference - *http://docs.unity3d.com/Manual/class-AnimationClip.html*

[48] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/AnimationCurve.html*

[49] Unity Scripting Reference - *http://docs.unity3d.com/Manual/Animator.html*

[50] Unity Scripting Reference - *http://docs.unity3d.com/Manual/AnimationStateMachines.html*

[51] Unity Scripting Reference - *http://docs.unity3d.com/Manual/class-AudioSource.html*

[52] Unity Scripting Reference - *http://docs.unity3d.com/Manual/UISystem.html*

[53] Unity Scripting Reference - *http://docs.unity3d.com/Manual/UICanvas.html*

[54] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/UI.Text.html*

[55] Unity Scripting Reference - *http://docs.unity3d.com/460/Documentation/ScriptReference/UI.Image.html*

[56] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/UI.Button.html*

[57] Asset Store di Unity3D - *https://www.assetstore.unity3d.com*

[58] Sito ufficiale di OculusVR (integrazione Unity) - *https://developer.oculus.com/downloads/*

[59] Sito ufficiale OculusVR - *https://www.oculus.com/rift/*

[60] Definizione di tecnologia PenTile - *http://it.wikipedia.org/wiki/PenTile*

[61] Sito ufficiale Oculus DK2 - *https://www.oculus.com/dk2/*

[62] Unity Scripting Reference - *http://docs.unity3d.com/ScriptReference/Physics.Raycast.html*

[63] Unity Scripting Reference -*http://docs.unity3d.com/Manual/Layers.html*