



UNIVERSITÀ DEGLI STUDI DI NAPOLI  
**FEDERICO II**

*Facoltà di Ingegneria Informatica*

**PROGETTO SISTEMI MULTIMEDIALI:**  
***Virtual Reality for CH***

**Professore:**

**PICCIALLI Francesco**

**A.A. 2018/2019**

**Studenti:**

**COLANTUONO Antonio N46003371**

**COPPOLA Vincenzo N46003356**

**DELLA TORCA Salvatore N46003208**

**IZZO Alberto N46003425**

# **INDICE**

<b>1. Introduzione</b>	<i>pag.3</i>
<b>2. Specifiche di progetto</b>	<i>pag.4</i>
<b>3. Strumenti utilizzati</b>	<i>pag.5</i>
<b>3.1. Unity 3D</b>	<i>pag.5</i>
<b>3.2. Visual Studio</b>	<i>pag.6</i>
<b>4. Descrizione applicazione: MuseumTOUR</b>	<i>pag.7</i>
<b>4.1. Scena 1: Lingua</b>	<i>pag.9</i>
<b>4.2. Scena 2: Audio</b>	<i>pag.11</i>
<b>4.3. Scena 3: Intro</b>	<i>pag.12</i>
<b>4.4. Scena 4: Menù</b>	<i>pag.13</i>
<b>4.5. Scena 5: Entrata</b>	<i>pag.15</i>
<b>4.6. Scena 6: Modifica</b>	<i>pag.20</i>
<b>4.7. Scena 7: Credits e Scena 8: CreditsEN</b>	<i>pag.22</i>
<b>4.8. Opzioni</b>	<i>pag.23</i>
<b>5. Testing</b>	<i>pag.24</i>
<b>6. Ringraziamenti</b>	<i>pag.25</i>

## 1. Introduzione



La *realtà virtuale*, per sua stessa definizione, simula la realtà effettiva e non deve essere confusa con la *realtà aumentata* che ha come obiettivo arricchire il mondo reale intorno a noi con contenuti digitali.

La realtà virtuale consente all'utente di entrare in un mondo immaginario creato digitalmente e di calarsi completamente in esso tramite *visori* appositi che isolano l'utente dall'ambiente circostante e rendono l'esperienza totalmente immersiva.

L'immersività può essere aumentata ulteriormente con device che digitalizzano i movimenti delle mani, in modo da rendere l'esperienza ancora più realistica.

Un visore VR può essere un casco o dei semplici occhiali in cui gli schermi vicini agli occhi annullano il mondo reale dalla visuale dell'utente.

Il visore può inoltre contenere dei sistemi per la rilevazione dei movimenti, in modo che girando la testa da un lato, ad esempio, si ottenga la stessa azione anche nell'ambiente virtuale.

## **2. Specifiche di progetto**

*Virtual Reality for CH (VR4CH)* è un progetto che ha come obiettivo quello di valorizzare i luoghi culturali del nostro territorio, fornendo all'utente la possibilità di visitare virtualmente tali luoghi utilizzando un visore VR.

Si richiede che l'utente possa, semplicemente guardandosi attorno, osservare in prima persona il luogo in questione e che possa interagire con esso.

L'applicazione deve essere sviluppata in UNITY 3D e si consiglia l'utilizzo del componente aggiuntivo Google VR SDK for Unity, plug-in offerto da Google per il supporto alla realizzazione di progetti utilizzabili tramite visori 3D per smartphone.

Il gioco deve essere costituito da varie scene, ognuna delle quali rappresenta un particolare punto del percorso virtuale che l'utente intraprenderà.

All'avvio dell'applicazione l'utente si troverà dinnanzi ad una basilare scena principale, costituita semplicemente da un menù che permetterà a quest'ultimo di avviare il gioco o di chiudere l'applicazione.

Le scene di un tour possono essere realizzate effettuando dei sopralluoghi negli ambienti da ricostruire, al fine di scattare un insieme di fotografie del luogo per la realizzazione di fotosfere, oppure utilizzando le tecniche di costruzione 3D messe a disposizione da Unity 3D.

### **3. Strumenti utilizzati**

#### **3.1 Unity**



L'applicazione è stata realizzata utilizzando **UNITY 3D** (versione 2019.1.4f1).

Unity è un motore grafico multiplatforma, sviluppato da Unity Technologies, che consente lo sviluppo di videogiochi e altri contenuti interattivi, come visualizzazioni architettoniche o animazioni 3D in tempo reale.

Unity è definito “multiplatforma” perchè il suo motore permette di progettare l'applicazione una sola volta e rilasciarla per ambienti diversi: PC (Windows, Mac), PlayStation, Nintendo, sistema operativo Android, iOS, Windows Phone ecc.

Il linguaggio di programmazione per Unity 3D è chiamato UnistyScript ed è sviluppato usando due tipi di linguaggi di programmazione: Javascript e C#.

### 3.2 Visual Studio



Microsoft Visual Studio, o più comunemente Visual Studio, è un ambiente di sviluppo integrato (*IDE*) sviluppato da Microsoft.

Visual Studio è multi-linguaggio ed attualmente supporta la creazione di progetti per varie piattaforme, tra cui anche Mobile e Console.

Visual Studio integra la tecnologia IntelliSense che permette di correggere eventuali errori sintattici (ed alcuni logici) senza compilare l'applicazione.

Possiede inoltre un debugger interno per il rilevamento e la correzione degli errori logici nel codice in runtime e fornisce diversi strumenti per l'analisi prestazionale.

Visual Studio offre un supporto per gli strumenti nativi Python e applicazioni Linux, l'integrazione con Unity per lo sviluppo di videogiochi, il simulatore Android e iOS e la possibilità di gestire e modificare cursori, icone e immagini all'interno dell'applicazione.

#### 4. Descrizione applicazione



MuseumTOUR

“MuseumTOUR” vuole dare all’utente la possibilità di visitare la **Galleria degli Uffizi** e di interagire con le opere presenti al suo interno.

All’avvio dell’applicazione l’utente avrà la possibilità di selezionare la lingua tra due possibili scelte: italiano o inglese.

Nel menù iniziale l’utente potrà:

- iniziare il tour cliccando sul pulsante “INIZIA”;
- visualizzare i “CREDITS”, ossia le informazioni relative agli sviluppatori dell’applicazione;
- modificare le impostazioni relative all’audio cliccando su “Opzioni”, scegliendo se utilizzare o meno una traccia audio come sottofondo durante la visita della Galleria;
- creare una propria esposizione attraverso il comando “CREA”, che consente all’utente di poter selezionare quali opere inserire all’interno del museo, scegliendo tra quelle disponibili nell’applicazione, e anche di poter selezionare la posizione in cui deve trovarsi quell’opera;
- uscire dall’applicazione attraverso il comando “ESCI”.

All’avvio del tour l’utente si troverà all’interno della prima stanza della Galleria e potrà iniziare la visita.

Inclinando la testa in avanti l’utente potrà muoversi all’interno della stanza e ruotando la testa potrà invece osservarsi intorno.

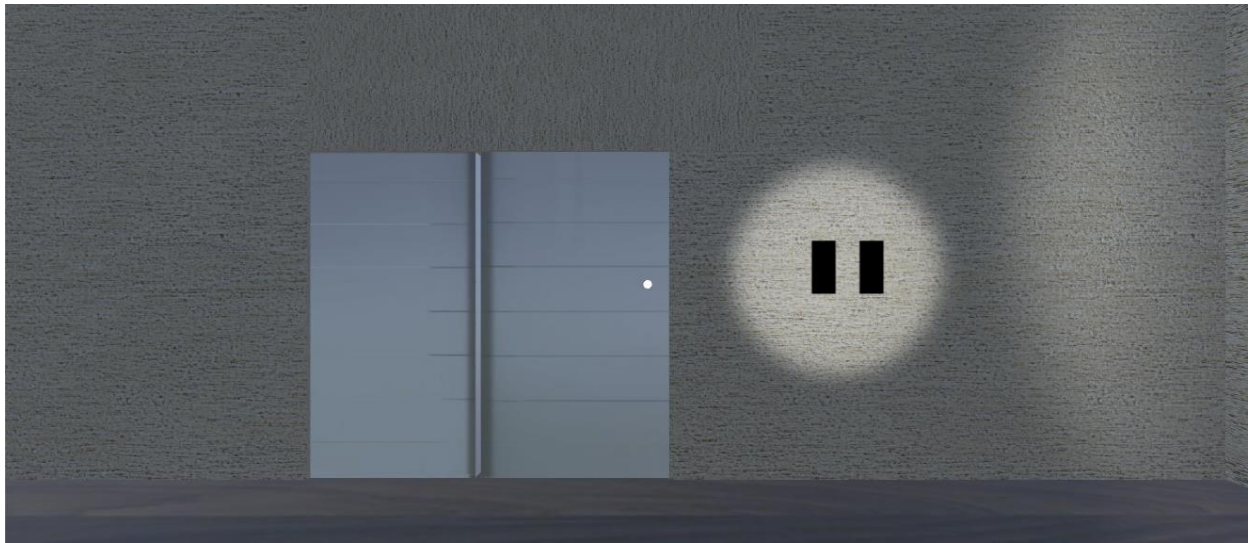
Potrà poi interagire con le opere presenti all’interno di una stanza focalizzando il “mirino” sulla targhetta col nome dell’opera: in questo



modo al lato dell'opera comparirà una descrizione in cui è possibile leggere il nome dell'artista, la data di realizzazione dell'opera, la tecnica utilizzata, le dimensioni e la spiegazione dell'opera.

La struttura, realizzata con UNITY 3D, è costituita da otto stanze, ognuna delle quali contiene 2 quadri.

L'utente può interagire quante volte vuole con ogni quadro e, una volta terminato il percorso, potrà cliccare sul pulsante di *pausa* presente nell'ultima stanza e tornare al menù principale.



Tuttavia, se l'utente volesse terminare il tour senza visitare tutte le stanze, è presente un tasto *pausa* anche nella quarta stanza attraverso il quale l'utente può ritornare al menù principale ed uscire dall'applicazione.

Di seguito vengono descritte le tecniche che sono state utilizzate per la realizzazione di ogni scena.

N.B: Il team ha deciso di concentrarsi maggiormente sulle funzionalità da fornire all'utente piuttosto che al lato estetico dell'applicazione.



## 4.1 Scena 1: Lingua

La prima scena è stata realizzata per consentire all'utente di selezionare la lingua.

Sono state fornite due possibili scelte: Italiano e Inglese.



Per realizzare la scena è stato utilizzato un *canvas*, necessario per poter interagire con i *bottoni* per la selezione della lingua.

Al canvas è stato assegnato lo script “*LinguaSkip*”, all’interno del quale sono state dichiarate due variabili di tipo **button** (*it* e *eng*) alle quali sono stati associati i due bottoni realizzati attraverso la UI.

```
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class LinguaSkip : MonoBehaviour
{
    public Button it;
    public Button en;

    // Start is called before the first frame update
    void Start()
    {
        variabile.italiano = false;
        variabile.inglese = false;
        it.onClick.AddListener(Ita);
        en.onClick.AddListener(Eng);
    }

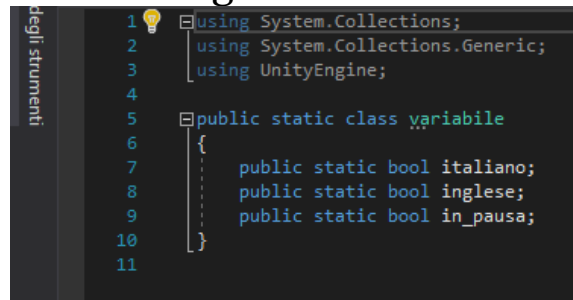
    // Update is called once per frame
    void Ita()
    {
        variabile.italiano = true;
        SceneManager.LoadScene("2. Audio");
    }

    void Eng()
    {
        variabile.inglese = true;
        SceneManager.LoadScene("2. Audio");
    }
}
```

Lo script utilizza la funzione ***onClick.AddListener()***, il cui parametro è l'azione che bisogna eseguire al click, per realizzare il passaggio alla scena successiva.

Nel nostro caso è stata fornita, come parametro, una nuova funzione: ***Ita*** se il bottone cliccato è quello per la lingua italiana, ***Eng*** se il bottone cliccato è quello per la lingua inglese.

Sia la funzione *Ita* che la funzione *Eng* utilizzano la funzione ***LoadScene*** dello ***SceneManager*** per caricare la scena successiva. Ovviamente la lingua utilizzata nelle scene successive dipende da quale lingua è stata selezionata; pertanto è stata realizzata una classe statica, denominata *variabile*, che contiene due variabili statiche di tipo bool che gestiscono la lingua, e un altro booleano che gestisce invece la pausa nelle scene di gioco.



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public static class variabile
6  {
7      public static bool italiano;
8      public static bool inglese;
9      public static bool in_pausa;
10 }
11
```

Queste variabili sono inizializzate a **false** in “*LinguaSkip*” e, se il bottone cliccato è Ita, allora la variabile *italiano* assume valore true, altrimenti sarà la variabile *inglese* ad assumere valore true.

Queste due variabili saranno poi utilizzate in tutte le scene successive in modo tale che, valutando il valore che esse assumono, sia possibile visualizzare i testi in lingua italiana o in lingua inglese.

Lo stesso vale per la variabile *in\_pausa* che sarà utilizzata in tutte le scene successive per verificare che l’applicazione sia in pausa oppure no.

## 4.2 Scena 2: Audio

La seconda scena è stata realizzata per consigliare all'utente di attivare l'audio, al fine di usufruire di un'esperienza di gioco ottimale. Si tratta di un canvas con all'interno un'immagine a sfondo nero e un testo.

Al testo, attraverso l'*animator*, è stata applicata un'animazione grazie alla quale si è realizzato l'effetto dissolvenza sia iniziale che finale.

Anche in questa scena al canvas è associato uno script "*AudioSkip*" che permette, dopo un certo intervallo di tempo (nel nostro caso 6 secondi), di passare alla scena successiva attraverso l'utilizzo della funzione ***LoadScene*** dello ***Scene Manager***.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5
6 public class AudioSkip : MonoBehaviour
7 {
8     private float time= 0.0f;
9     // Start is called before the first frame update
10    void Start()
11    {
12    }
13
14    // Update is called once per frame
15    void Update()
16    {
17        time = time + Time.deltaTime;
18        if (time > 6.0f)
19        {
20            SceneManager.LoadScene("3. Intro");
21        }
22    }
23 }
24
25
```

Nella scena 2 è stata utilizzata la classe *variabile*, che è stata precedentemente citata, per far sì che il testo fosse in lingua italiana o inglese a seconda della lingua scelta dall'utente.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class Testo : MonoBehaviour
{
    public Text testo;
    // Start is called before the first frame update
    void Start()
    {
        testo = GetComponent<Text>();
        if (testo)
        {
            if (variabile.italiano)
            {
                testo.text = "SI CONSIGLIA DI ATTIVARE L'AUDIO \nAL FINE DI UN'ESPERIENZA OTTIMALE";
            }
            else if (variabile.inglese)
            {
                testo.text = "IT IS RECOMMENDED TO TURN ON THE AUDIO IN ORDER\n TO ACHIEVE THE BEST POSSIBLE EXPERIENCE";
            }
        }
    }
}

```

### 4.3 Scena 3: Intro

La terza scena è un'intro in cui viene visualizzata un'immagine della Galleria degli Uffizi sulla quale, attraverso l'*animator*, è stato applicato un effetto di dissolvenza in entrata.

Inoltre, nella scena, sono stati inseriti una traccia audio ed un testo al quale è stato associato uno script che realizza l'effetto *scorrimento da destra verso sinistra*.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpostamentoTesto : MonoBehaviour
{
    public float time = 0.0f;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        time = time + Time.deltaTime;
        if(transform.position.x > 0 && time > 2.5f)
        {
            transform.Translate(Vector3.left * 0.09f);
        }
        if(transform.position.x > -17 && time > 8.5f)
        {
            transform.Translate(Vector3.left * 0.09f);
        }
    }
}
```

Lo script "*IntroSkip*", così come "*AudioSkip*" nella scena precedente, permette di caricare la scena successiva dopo un certo intervallo di tempo(nel nostro caso 13 secondi).

Anche in questa scena, così come in quella precedente, è stato poi realizzato uno script "*TestoIntro*" che permette di modificare il testo in base alla lingua scelta.

## 4.4 Scena 4: Menù

La quarta scena è quella utilizzata per la realizzazione del menù.

Alla camera è stato associato uno script “*movCamera*” che è stato realizzato per creare un’animazione sulla camera attraverso la quale quest’ultima, ad ogni frame, viene ruotata verso il basso.

In questo script è dichiarata una variabile di tipo *float* che rappresenta il tempo trascorso.

Questa variabile è inizializzata a zero e nella funzione *update*, che viene invocata una volta per frame, si incrementa di un fattore **Time.deltaTime**.

Fin tanto che il tempo trascorso è inferiore a 7 secondi, la camera viene ruotata verso il basso e traslata all’indietro attraverso le funzioni **Rotate** e **Translate** (entrambe usano come parametro un *vector3*).

Infine, quando l’animazione della camera è terminata, si attiva il canvas per la visualizzazione del menù attraverso l’istruzione **SetActive(true)**.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class movCamera : MonoBehaviour
{
    //public float time = 0.0f;
    public GameObject can;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        //time = time + Time.deltaTime;
        // if (time < 7.0f)
        if(this.gameObject.transform.localEulerAngles.x<40)
        {
            transform.Rotate(Vector3.right * 0.1f);
            transform.Translate(Vector3.back * 0.005f);
        }

        if(this.gameObject.transform.localEulerAngles.x>39)
        {
            can.SetActive(true);
        }
    }
}
```

Anche al testo è stato applicato uno script, *TranslationTEXT*, che sfruttando la funzione *translate* della classe Transform permette al testo di spostarsi automaticamente verso l’alto.

Alla camera è associato anche un altro script, *TwoD*, che serve per attivare la modalità 2D nel menù.

La camera “contiene” un canvas che a sua volta contiene tutti i bottoni presenti all’interno del menù.

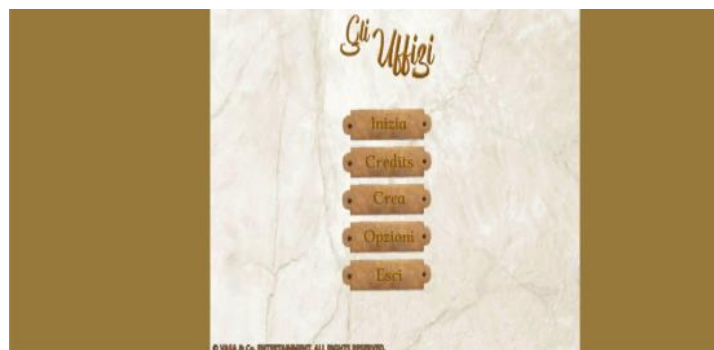
A ciascun bottone è associato lo script che consente, a run-time, o di cambiare scena o, nel caso in cui viene cliccato il bottone “Esci”, di terminare l’applicazione.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class QuitScript : MonoBehaviour
6 {
7     public void EsciApp()
8     {
9         Application.Quit();
10    }
11 }
12
```

Il menù presenta diversi bottoni:

- Inizia;
- Credits;
- Opzioni;
- Crea;
- Esci.

Anche in questo caso sono stati realizzati tutti gli script necessari per far apparire i bottoni del menù in lingua diversa a seconda di quella selezionata



## 4.5 Scena 5: Entrata

Una volta aver selezionato “Inizia” si aprirà la quinta scena. L’utente si trova all’interno della Galleria, precisamente nella prima stanza.

Ovviamente, a partire dall’inizio della quinta scena sarà attivo il VR e l’utente potrà quindi indossare il visore e cominciare il tour.

Per lo switch alla VR è stata utilizzata la funzione *vrO2d5* che, effettuando un controllo sulle variabili definite, attiva o meno la modalità VR.

```
// Start is called before the first frame update
void Start()
{
    Switch5.toggle = true;
}

// Update is called once per frame
void Update()
{
    if (Switch5.toggle)
    {
        StartCoroutine(SwitchVR5());
    }
    else
    {
        StartCoroutine(SwitchTo2D5());
    }
}

IEnumerator SwitchVR5()
{
    if (String.Compare(XRSettings.loadedDeviceName, "cardboard", true) != 0)
        XRSettings.LoadDeviceByName("cardboard");
    yield return null;
    XRSettings.enabled = true;
}

IEnumerator SwitchTo2D5()
{
    XRSettings.LoadDeviceByName("");
    yield return null;
}
```



Il player è stato realizzato utilizzando una *sfera*, alla quale sono stati applicati gli script **MovimentoPlayerVR** e **RotazionePlayer**.

In *MoviemntoPlayerVR*, per ogni frame, viene eseguita la funzione *update* in cui se l'angolo di inclinazione del VR è compreso fra *toggleAngle* (nel nostro caso 20°) e 90° allora la variabile *moveForward* viene inizializzata a true, altrimenti viene inizializzata a false.

Successivamente, se *moveForward* è true, allora si chiama la funzione **Translate** per realizzare il movimento del player.

```
public class MovimentoPlayerVR : MonoBehaviour
{
    public Transform vrCamera;
    public float toggleAngle = 15.0f;
    public bool moveForward;
    //public Rigidbody rb;
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        if (vrCamera.eulerAngles.x > toggleAngle && vrCamera.eulerAngles.x < 90.0f)
        {
            moveForward = true;
        }
        else
        {
            moveForward = false;
        }

        if (moveForward)
        {
            transform.Translate(Vector3.forward * 0.2f, Space.Self);
            //rb.AddRelativeForce(Vector3.forward * speed);
        }
    }
}
```

Ovviamente il player può solo muoversi in avanti e per potersi muovere lateralmente o all'indietro è necessaria la rotazione.

Lo script **RotazionePlayer** consente al player di ruotare infatti, per ogni frame, viene invocata la *update* che assegna a *transform.rotation* il valore di *rotazione.rotation* dove rotazione è una variabile di tipo Transform alla quale è stata associata la main camera.

```
public class RotazionePlayer : MonoBehaviour
{
    public Transform rotazione;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        transform.rotation = rotazione.rotation;
    }
}
```

Alla *sphere* è stato applicato anche un altro script, *AudioSet*, che serve per riprodurre come audio di sottofondo la traccia selezionata dall'utente.

```
public class AudioSet : MonoBehaviour
{
    public AudioSource Audio1;
    public AudioSource Audio2;
    // Start is called before the first frame update
    void Start()
    {
        if (audioSelezione.selezione % 3 == 1)
        {
            Audio1.Play();
        }
        if(audioSelezione.selezione % 3 == 2)
        {
            Audio2.Play();
        }
    }
}
```

Lo script **PosizioneCamera**, assegnato all'oggetto *supporto*, serve invece per garantire che la camera si muova contemporaneamente al player.

Affinché ciò sia possibile sono state dichiarate due variabili:

- *Pallista*, di tipo *Transform*, alla quale è stata associata la sfera che rappresenta il player;
- *Offset*, di tipo *Vector3*, che permette di settare la posizione della camera rispetto alla sfera.

Per ogni frame la funzione *update* associa a *Trasform.position* la somma di *pallista.position* e *offset* in modo tale che la camera si trovi, a meno dell'offset, nella stessa posizione della sfera.

A questo punto, definite le modalità attraverso le quali il player può muoversi all'interno della scena, bisogna capire come può interagire con le opere.

Come è stato riportato nella descrizione generale, se il player posiziona il mirino sulla targa col nome dell'opera, dopo una breve attesa, compariranno la descrizione dell'opera stessa e le informazioni relative all'artista.

Lo script realizzato per tradurre la visualizzazione della targhetta in click è **Pressione2**, che è stato associato ad ogni bottone col nome dell'opera.

In questo script, la funzione *gvrOn*, grazie all' *Event Trigger* aggiunto al bottone, viene invocata quando il player visualizza la

targhetta e setta a true la variabile booleana *gvrStatus* che viene poi utilizzata nella funzione *update*:

se *gvrStatus* è true allora *gvrTimer* viene incrementata di un fattore *Time.deltaTime* e viene avviata l'animazione di interazione attraverso la funzione ***fillAmount***.

Quando *gvrTimer* diventa maggiore di *totalTime*, che rappresenta la durata dell'animazione dell'interazione, viene invocata la ***onClick***, che simula il click del bottone, e viene portato *gvrTimer* a 0.

La funzione *gvrOff* è stata realizzata per resettare i parametri nel caso in cui il player dovesse distogliere lo sguardo dalla targa.

```
public class Pressione2 : MonoBehaviour
{
    public Image imgCircle;
    public UnityEvent gvrClick;
    public float totalTime = 2;
    bool gvrStatus = false;
    public float gvrTimer;

    // Update is called once per frame
    void Update()
    {
        if (gvrStatus)
        {
            gvrTimer = gvrTimer + Time.deltaTime;
            imgCircle.fillAmount = gvrTimer / totalTime;
        }
        if (gvrTimer > totalTime)
        {
            GetComponent<Button>().onClick.Invoke();
            gvrTimer = 0;
        }
    }

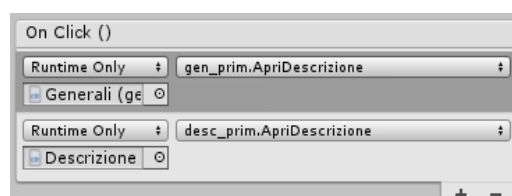
    public void gvrOn()
    {
        gvrStatus = true;
    }

    public void gvrOff()
    {
        gvrStatus = false;
        gvrTimer = 0;
        imgCircle.fillAmount = 0;
    }
}
```

Quando viene invocata la funzione *onClick*, a run-time saranno eseguiti altri due script: ***Generali e Descrizione***.

Queste due funzioni servono per aprire le descrizioni legate all'opera e all'artista: il tutto è basato su una variabile *contatore* di tipo int che si incrementa ogni volta che viene invocata la *onClick*.

Essendo inizializzata a 0, quando *contatore* diventa 1 vuol dire che il bottone è stato cliccato e quindi bisogna far comparire la descrizione: per fare ciò è stato utilizzato un ***if*** in cui si verifica che *contatore%2* sia una quantità diversa da 1 e, se ciò è vero, bisogna rimuovere la descrizione altrimenti bisogna farla comparire.



Le stanze sono separate da una porta in grado di aprirsi automaticamente quando il player si trova ad una certa distanza da essa.

Per rendere ciò possibile, è stato realizzato uno script denominato **PortaAuto** che è basato sulla funzione **Lerp** di Vector3 che realizza l'*interpolazione lineare* tra due vettori.

```
public class PortaAuto : MonoBehaviour
{
    public GameObject porta1, posAperto1;
    public GameObject porta2, posAperto2;

    public float velocidade = 2;

    Vector3 posicInicP1, posicInicP2;
    int numObjDentro;
    void Start()
    {
        numObjDentro = 0;
        posicInicP1 = porta1.transform.localPosition;
        posicInicP2 = porta2.transform.localPosition;
    }

    void Update()
    {
        if (numObjDentro > 0)
        {
            porta1.transform.localPosition = Vector3.Lerp(porta1.transform.localPosition, posAperto1.transform.localPosition, velocidade * Time.deltaTime);
            porta2.transform.localPosition = Vector3.Lerp(porta2.transform.localPosition, posAperto2.transform.localPosition, velocidade * Time.deltaTime);
        }
        else
        {
            porta1.transform.localPosition = Vector3.Lerp(porta1.transform.localPosition, posicInicP1, velocidade * Time.deltaTime);
            porta2.transform.localPosition = Vector3.Lerp(porta2.transform.localPosition, posicInicP2, velocidade * Time.deltaTime);
        }
    }

    void OnTriggerEnter()
    {
        numObjDentro++;
    }

    void OnTriggerExit()
    {
        numObjDentro--;
        if (numObjDentro < 0)
        {
            numObjDentro = 0;
        }
    }
}
```

Nella stanza numero 4 e nell'ultima stanza è presente un tasto *pausa* che, se cliccato, consente all'utente di mettere in pausa il tour oppure di tornare al menù.

Anche in questo caso è stato realizzato uno script apposito.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.XR;

public class TastoPausa5 : MonoBehaviour
{
    public GameObject menupausaUI;
    public GameObject suppPlay;

    public void Pausa()
    {
        Switch5.toggle = false;
        //StartCoroutine(SwitchTo2D());
        suppPlay.SetActive(false);
        menupausaUI.SetActive(true);
        Time.timeScale = 0f;
        variabile.in_pausa = true;
    }

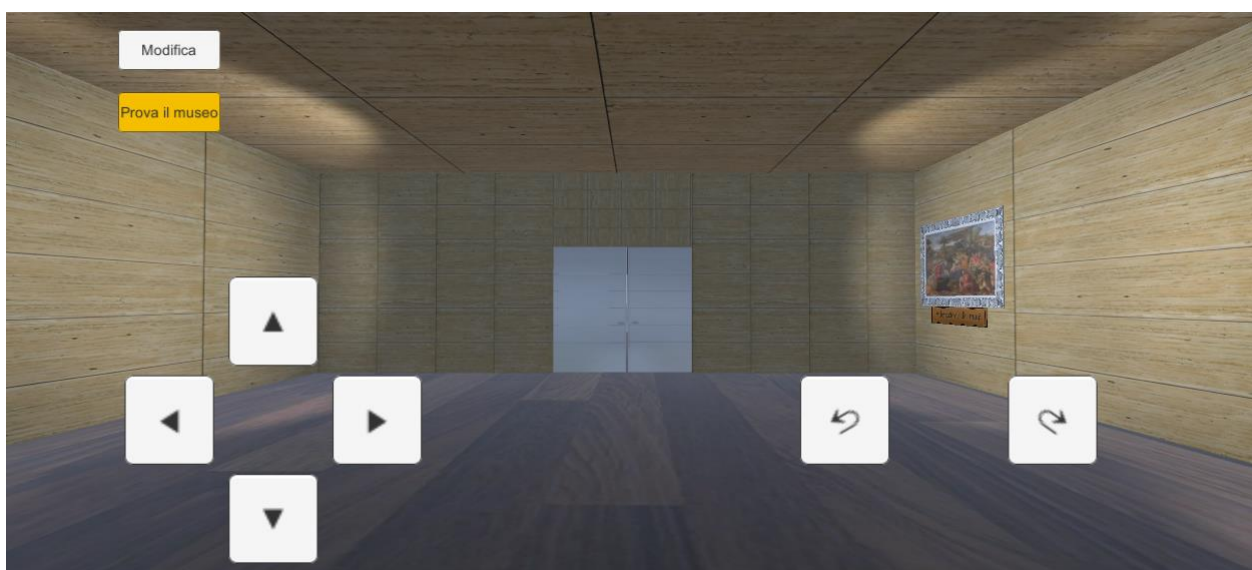
    IEnumerator SwitchTo2D()
    {
        XRSettings.LoadDeviceByName("");
        yield return null;
    }
}
```

## 4.6 Scena 6: Modifica

Cliccando sul bottone “Crea” del menù lo SceneManager caricherà la scena numero 6.

In questa scena è gestito il **Back-End** dell'applicazione.

Questo vuol dire che si dà la possibilità all'utente di poter realizzare una propria esposizione, il che significa che l'utente, data la struttura, potrà selezionare quali tra i quadri presenti nell'applicazione esporre e dove questi ultimi dovranno essere posizionati.



Affinchè ciò sia possibile l'utente deve cliccare sulla targa del quadro da modificare, in modo tale da far comparire la descrizione, e a quel punto può modificarlo.

Tutto questo è possibile grazie all'implementazione di opportuni script, ad esempio:

- *AudioSet*, che serve per caricare la traccia audio che l'utente seleziona nel menù;
- *EditareScript*, associato al gameobject SupportEdit, che consente di bloccare il movimento del player e abilitare i tasti direzionali per modificare la posizione del quadro;
- *aumX*, *aumY*, *dimX*, *dimY* che servono per spostare il quadro selezionato in alto, in basso, a destra e a sinistra;
- *vai Sx e vai Dx*, che servono per far scorrere il vettore delle immagini che possono essere inserite all'interno di un quadro;
- *cambiaImmagine*, che serve per caricare nella cornice l'immagine corrispondente all'indice del vettore di immagini;

- *applicaImmagine* serve per sostituire il quadro selezionato con il quadro presente in *cambiaImmagine*;
- *provaCambiamenti*, che serve per consentire al player di visitare l'esposizione appena realizzata;
- *MovimentoEdit* e *RotazioneEdit* che servono per consentire al player di spostarsi e ruotarsi nella modifica dei quadri;

Lo script *settaPerEdit* è uno degli script più importanti, che serve per associare al gameobject *editare* il quadro da modificare.

Questo script fa in modo che, cliccando la targhetta di un quadro, nel gameobject *editare* ci sia un riferimento al quadro da cambiare.

#### ***4.7 Scena 7: Credits e Scena 8: CreditsEN***

Queste due scene sono state realizzate per fornire delle informazioni relative agli sviluppatori dell'applicazione.

È possibile accedere ad esse cliccando sul bottone “Credits” del menu, La scena è costituita da un semplice canvas nel quale è inserito un testo, un'immagine che rappresenta il logo dell'Università Federico II di Napoli, e un bottone “*Indietro*” che come è possibile intuire serve per ritornare al menù.



## 4.8 Opzioni

Cliccando sul bottone Opzioni del menù si apre un nuovo canvas nel quale troviamo due bottoni:

- Audio, attraverso il quale l'utente può selezionare quale traccia audio usare come sottofondo;
- Indietro, per tornare al menù.

Per aprire il canvas relativo alle opzioni è stato realizzato lo script *Opzioni*.

```
Assembly-CSharp
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Opzioni : MonoBehaviour
6 {
7     public GameObject canvas1;
8     // Start is called before the first frame update
9     void Start()
10    {
11    }
12
13    // Update is called once per frame
14    public void opzioni()
15    {
16        canvas1.SetActive(true);
17        this.gameObject.transform.parent.gameObject.SetActive(false);
18    }
19
20 }
21
```

Per settare invece la traccia audio scelta dall'utente abbiamo lo script *TestoAudio*.

```
Assembly-CSharp
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6 public static class audioSelezione
7 {
8     public static int selezione = 0;
9 }
10
11 public class TestoAudio : MonoBehaviour
12 {
13     // Start is called before the first frame update
14     public void cambiaAudio()
15     {
16         audioSelezione.selezione++;
17     }
18
19     // Update is called once per frame
20     void Update()
21     {
22         if (audioSelezione.selezione % 3 == 0)
23         {
24             this.gameObject.transform.GetChild(0).GetComponent<Text>().text = "No Audio";
25         }
26         if (audioSelezione.selezione % 3 == 1)
27         {
28             this.gameObject.transform.GetChild(0).GetComponent<Text>().text = "Audio 1";
29         }
30         if (audioSelezione.selezione % 3 == 2)
31         {
32             this.gameObject.transform.GetChild(0).GetComponent<Text>().text = "Audio 2";
33         }
34     }
35 }

```

## 5. Testing

Il testing dell'applicazione è stato realizzato in due diverse fasi:

- *Alpha testing*;
- *Beta testing*.

La fase di Alpha Testing è stata realizzata dai membri del VASA&Co. ad ogni aggiunta di funzionalità all'applicazione.

In primis sono stati inseriti negli script una grande quantità di *Debug.Log* in modo da avere conoscenza degli errori commessi.

Successivamente è stata provata più volte l'applicazione e sono stati utilizzati principalmente due dispositivi:

- Samsung Galaxy S5;
- Huawei P10 Lite.

Per quanto riguarda il Beta Testing, l'apk dell'applicazione è stato fornito ad amici e familiari dei componenti del team, che hanno potuto utilizzare l'applicazione al fine di poter individuare eventuali problematiche e fornire consigli riguardanti le funzionalità presenti e non.

I dispositivi utilizzati per il Beta-Testing sono:

- Samsung Galaxy S4;
- Samsung Galaxy S8 Plus;
- Huawei Mate 20 Pro;
- Huawei P10;
- Huawei P30.

## 6. Ringraziamenti

Il team Vasa&Co. ci tiene a ringraziare:

- il professore Piccialli Francesco, per essere stato sempre disponibile e per aver sempre chiarito in modo chiaro come affrontare la progettazione dell'applicazione;
- familiari e amici che sono stati fondamentali, non solo per il testing dell'applicazione, ma anche e soprattutto per il supporto morale fornito;
- tutti coloro che, anche se in piccola parte, hanno contribuito con i loro consigli a rendere questa applicazione migliore.

Si spera che **MuseumTOUR** sia di vostro gradimento.  
BUON DIVERTIMENTO.

Il team Vasa&Co.