Politecnico di Torino

POLITECNICO DI TORINO

Corso di Laurea magistrale in
Ingegneria del cinema e dei mezzi di comunicazione

Tesi di Laurea magistrale

Tipi di testi nell'antico Egitto

Candidata: Simona Galantino

Relatore: Prof. Riccardo Silvio Antonino

Anno accademico 2023/2024

A papà,

A mamma,

Ad Andrea,

A Mattia,

A Serena,

A me stessa,

E a te se sei rimasto con me

fin proprio alla fine

RINGRAZIAMENTI

Vorrei dedicare questo spazio per ringraziare tutte le persone che mi sono state vicine in questo percorso.

Ai miei primi sostenitori, mamma e papà, per aver sempre creduto in me. Siete stati il mio primo pensiero quando uscivo dall'università dopo un esame e i primi a sapere gli esiti di questi, belli o brutti che fossero. Grazie per avermi consolata dopo ogni fallimento e per aver festeggiato con me ogni successo.

Ai miei fratellini, Andrea, Mattia e Serena, per avermi sopportata e supportata in questi anni. Grazie per essere stati i miei accompagnatori al sushi ogni volta che mi sentivo giù per l'università o ogni volta che mi sentivo felice dopo un esame.

Ai miei amici, per aver sempre creduto in me anche quando io avevo smesso di farlo. Grazie per avermi ascoltata con pazienza quando parlavo di cose a voi incomprensibili e probabilmente noiose. Grazie per aver riso con me delle mie disavventure all'università. Grazie per aver condiviso con me i banchi del Poli ed essere stati i miei compagni per ogni progetto. Grazie per tutte quelle volte in cui abbiamo studiato insieme perché anche se mi avete vista impazzire per progetti, tesi e esami vari mi avete teso una mano e accompagnata fino a questo traguardo.

Infine grazie a me stessa, perché nonostante tutti i momenti di sconforto e di difficoltà, ora sono qui a festeggiare. Affrontare le proprie debolezze è sempre difficile, e ammetto di aver pensato più volte di abbandonare gli studi perché era "più semplice", ma oggi mi guardo indietro e mi dico: Simona, sono orgogliosa di te.

Se puoi sognarlo, puoi farlo

ABSTRACT

L'elaborato presenta lo sviluppo dell'applicazione *Tipi di testi nell'antico Egitto*, nata come collaborazione tra Robin Studio e il Museo Egizio di Torino. Nel dicembre 2023 è stata inaugurata una nuova sala, la Galleria della Scrittura, con l'idea di far immergere gli spettatori e guidarli in un viaggio alla scoperta della storia della scrittura egizia, passando dal geroglifico allo ieratico, dal demotico al copto. All'interno di questa nuova sala sono presenti diversi papiri ma non solo, sono esposti anche capolavori della statuaria, oggetti in alabastro, statuine lignee, e alcune installazioni interattive che permettono di esplorare la scrittura in modo più coinvolgente e immersiva, usando strumenti innovativi.

La cura di questa nuova sala è stata affidata alla responsabile della Papiroteca del Museo Egizio, Susanne Toepfer, nonché figura di riferimento dell'applicazione *Tipi di testi nell'antico Egitto*. Questa applicazione è stata sviluppata attraverso il software Unity, ed è stata installata su un monitor touch all'interno della Galleria, invogliando i visitatori ad approfondire ciò che ha visto precedentemente durante la visita della sala. Immersi in un'ambientazione virtuale della città di Deir El Medina, gli utenti hanno la possibilità di esplorare il luogo, incontrare alcuni scribi dell'epoca e scoprire qualcosa in più su di loro ed entrare all'interno di un'abitazione della città per poter esplorare i papiri che più interessano.

L'obiettivo dell'applicazione è quello di raccogliere in un unico ambiente tutti i papiri mostrati durante la visita nella sala, per poter dare la possibilità all'utente di interagire con essi e approfondire quelli che ritiene più interessanti, in modo del tutto libero. La semplicità dell'interfaccia e le poche e chiare indicazioni su come utilizzare l'applicazione che il visitatore si trova davanti fanno in modo che quest'ultimo si muova in totale autonomia all'interno dell'applicazione.

La semplicità d'utilizzo dell'applicazione è anche data dal tipo di dispositivo su cui è installata *Tipi di testi nell'antico Egitto*. È stato deciso di utilizzare uno

schermo touch di facile intuizione perché l'utente ha maggiore familiarità e ne fa uso ormai quotidianamente, e questo permette a chiunque, anche ai visitatori più giovani, di essere in grado di immergersi in questo viaggio virtuale e scoprire le informazioni inserite all'interno.

INDICE

1. INTRODUZIONE

L'applicazione *Tipi di testi nell'antico Egitto* è stata pensata per essere fruita su uno schermo touch presente nella Galleria della Scrittura del Museo Egizio di Torino. Qui un qualsiasi visitatore può "giocare" con l'applicazione esplorando la città di Deir El Medina e visualizzando diversi papiri con le relative traduzioni. Inoltre, il pubblico a cui si espande l'applicazione è sia italiano che internazionale in quanto ogni testo è stato scritto in italiano che in inglese.

All'interno dell'elaborato viene presentato il lavoro svolto per la produzione dell'applicazione *Tipi di testi nell'antico Egitto* e il lavoro svolto in prima persona all'interno del progetto.

Nel primo capitolo viene mostrato il funzionamento generale dell'applicazione. L'idea di questo capitolo è quella di affrontare il viaggio all'interno dell'applicazione per come è stata pensata dal team di sviluppo, e come quindi un visitatore del museo dovrebbe interagire trovandosi davanti a *Tipi di testi nell'antico Egitto* se avesse il tempo e la possibilità di farlo. Gli utenti però difficilmente riusciranno a seguire tutti i passaggi presenti nell'applicazione in quanto non sempre saranno fortunati a trovarsi davanti alla schermata di inizio dell'applicazione. Nonostante ciò le esperienze di alcuni visitatori, che verranno dettagliatamente descritte in uno dei capitoli seguenti, sono state comunque positive, e non sembra che il punto di partenza abbia influito molto sul giudizio degli utenti riguardo all'esperienza.

In seguito, nel secondo capitolo e nel terzo capitolo, viene mostrato più in dettaglio lo sviluppo di una parte dell'applicazione, riguardo alla lettura del database e come è stata implementata la sottoscena Testo. La scelta di approfondire solo questi due argomenti è dato dal fatto che a lavorare sul progetto erano presenti più persone, ognuna delle quali si è dedicata a una sola parte di questo, e il lavoro svolto in prima persona è stato quello di implementare la logica

del database e di sviluppare la parte dedicata agli approfondimenti dei papiri presenti e in alcuni casi della loro traduzione.

Nei capitoli ancora successivi verranno presentati alcuni metodi per profilare un'applicazione in generale, in modo tale da capire dove sono presenti eventuali colli di bottiglia ed eventuali rallentamenti di un'applicazione. Anche *Tipi di testi nell'antico Egitto* si è sottoposta a queste tecniche e trovando rallentamenti su alcuni punti dell'applicazione sono state mostrate alcune tecniche di ottimizzazione che potrebbero essere utili per migliorare il funzionamento e l'esperienza utente.

Infine viene mostrato uno sguardo sul futuro. L'idea della curatrice del museo e del team di sviluppo è quella di rendere l'applicazione sempre più completa e più ricca di contenuti.

2. TIPI DI TESTI NELL'ANTICO EGITTO

In questo capitolo verrà spiegato il funzionamento generale dell'applicazione *Tipi di testi nell'antico Egitto*. Il percorso descritto successivamente è un percorso teorico e non sempre corrisponde a quello affrontato dai visitatori.

L'applicazione è divisa in diverse scene, in modo tale che il team di sviluppo avesse la possibilità di lavorare in contemporanea senza creare problemi durante i *commit*.

La prima scena che un visitatore si trova davanti è la scena Deir El Medina, e può esplorare la città in lungo e in largo fino ad arrivare alla casa di uno scriba che è la *call to action* per passare alla scena successiva. Questa scena è molto simile a Deir El Medina 2, in cui sono presenti oltre alla città, anche le figure di scribi importanti dell'epoca con cui l'utente può interagire. La scelta di utilizzare due scene con un'interfaccia e una modalità d'uso simile è stata fatta per un più semplice controllo sul codice scritto. È presente una scena Home con due sottoscene, Home e Testo. La scelta di avere due sottoscene all'interno di una scena singola e non due scene separate è data dalla semplicità di collegamento tra le informazioni di una determinata riga del database di una sottoscena e l'altra. Infine la scena PermanentScene è una scena in cui sono presenti le indicazioni di *call to action*, e l'interfaccia in cui è presente il titolo e la scena di riferimento in alto, e i bottoni che permettono il cambio di scena in basso.

2.1 TECNOLOGIE UTILIZZATE

L’applicazione, per essere realizzata, è stata sviluppata con Unity 3D, un motore grafico sviluppato per creare applicazioni interattive e videogiochi. L’ambiente Unity è disponibile sia su Windows, sia su macOS, sia su Linux così come ciò che viene sviluppato può essere eseguito in Windows, macOS, Linux, Android.

È semplice importare modelli 3D da software come Blender su Unity grazie a una serie di tool presenti sul programma, ed è semplice programmare grazie a IDE (ambiente di sviluppo integrato) come VisualStudio utilizzando il linguaggio di programmazione C#.

2.2 SCENA DEIR EL MEDINA

Un visitatore che entra nella Galleria della scrittura si trova davanti alla schermata di avvio dell'applicazione. La schermata mostra una scritta in italiano e in inglese che cambia di intensità in modo tale da invogliare il visitatore ad avvicinarsi e ad interagire con l'applicazione. Quando si avvicina, l'utente si trova davanti ad una scelta, infatti può decidere se passare direttamente alla parte di visualizzazione delle varie categorie di papiri toccando il tasto *Skip,* oppure scoprire la città di Deir El Medina, toccando il tasto *Start*.

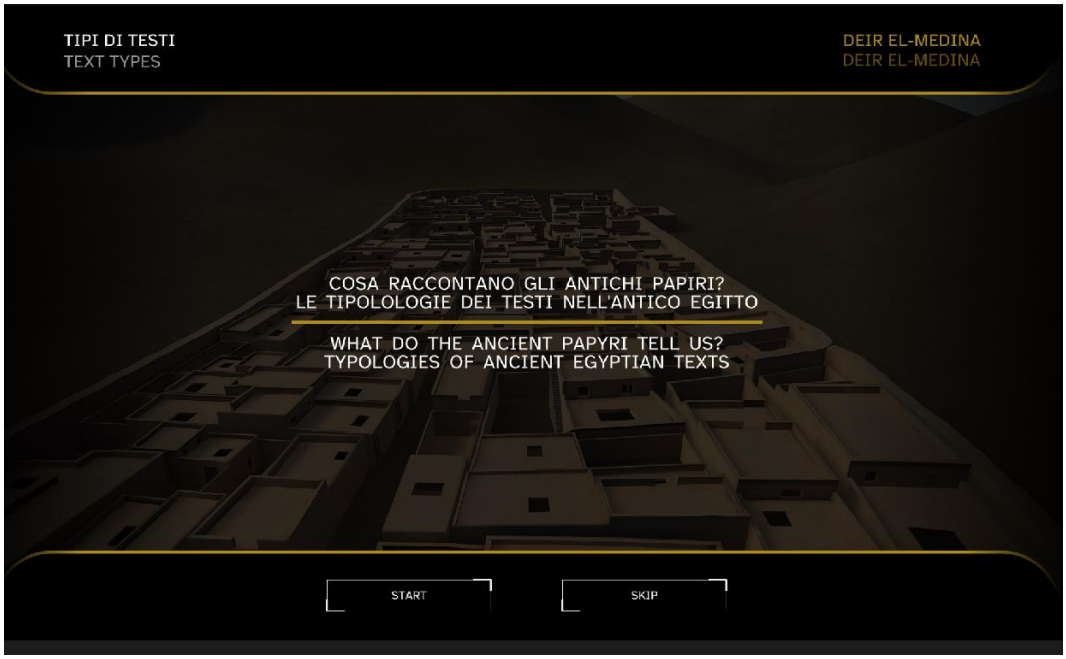


Figura 1: Schermata di avvio dell'applicazione *Tipi di testi nell'antico Egitto*

Scegliendo di iniziare ad esplorare la città il visitatore preme il tasto *Start* e può scoprire di più sulla città, ruotarla e seguire le *call to action* per poter andare avanti. Durante questa scena l'utente può iniziare a prendere confidenza con l'applicazione attraverso i vari comandi da seguire, toccando lo schermo e scoprendo alcune curiosità sulla città. Sono presenti dei box in cui sono inserite delle informazioni sul villaggio di Deir El Medina che verranno riprese anche nella scena Deir El Medina 2, nella schermata di info. Questi box hanno una durata limitata e alla fine del tempo stabilito, tendenzialmente basta per riuscire a

leggere il contenuto, questi scompaiono, lasciando il tempo all'utente di scegliere cosa poter fare.

Figura 2: Esplorazione della città di Deir El Medina

Quando l'utente è soddisfatto della sua esplorazione della città, può avvicinarsi alla casa dello scriba che lo porterà a scoprire i papiri. Per farlo deve seguire le indicazioni presenti sulla schermata, che lo guideranno davanti alla casa. Qui una *call to action* lo porterà ad aprire la porta in modo da riuscire ad entrare nell'abitazione.

Figura 3: Schermata di entrata nella casa dello scriba

L'ultima schermata prima di passare alla scena successiva, presenta l'interno della casa dello scriba in cui sono presenti alcuni elementi di decorazione che ricordano elementi dell'antico Egitto come giare, papiri. Lo scriba è posto al centro della schermata e ha una tavoletta in mano. Questa è la *call to action* che porta alla scena successiva, la scena Home, infatti la tavoletta presenta un *glow* attorno in modo tale da essere evidenziata e risultare visibile al visitatore e rendere più chiara l'indicazione da seguire per procedere.

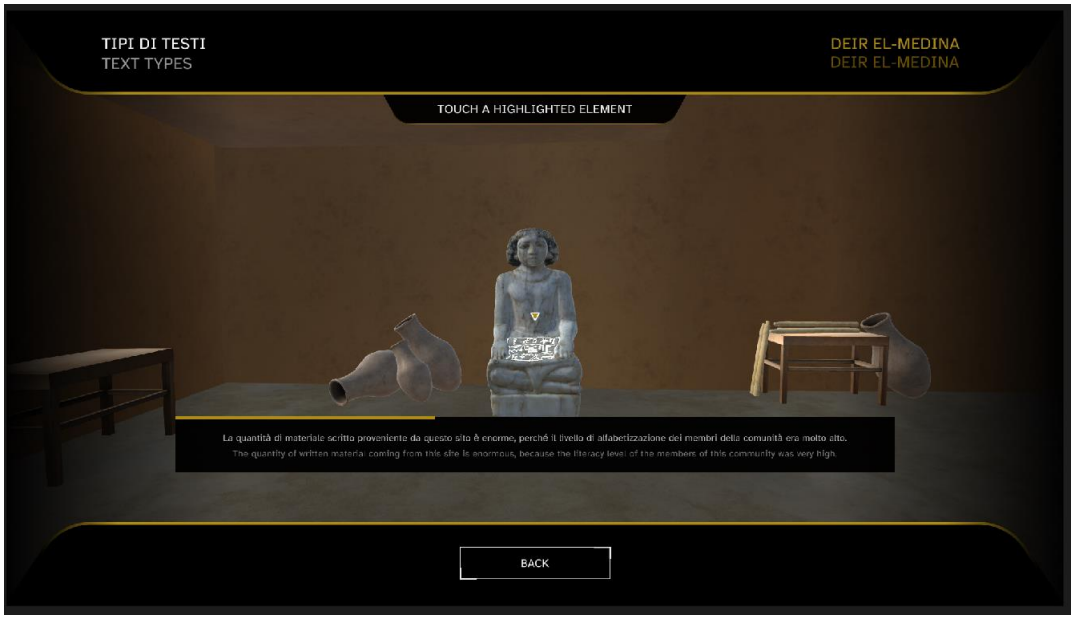


Figura 4: Casa dello scriba

2.3 SCENA HOME

La scena Home è composta da due sottoscene: Home e Testo. Nella prima avviene una scelta della categoria e sottocategoria, nella seconda c'è un approfondimento e la visualizzazione della traduzione di alcuni papiri.

2.3.1 SOTTOSCENA HOME

Attraverso una dissolvenza l'utente passerà dalla scena Deir El Medina alla scena Home, sottoscena Home. Per creare una continuità, lo scriba che era presente nella casa è presente anche in questa sottoscena, ed è l'elemento centrale dello schermo. In questa sottoscena si possono esplorare le diverse categoria in cui sono suddivisi i papiri.

Le categorie sono inserite all'interno di un cerchio che può ruotare in senso orario o antiorario in base al movimento che fa l'utente con il dito.

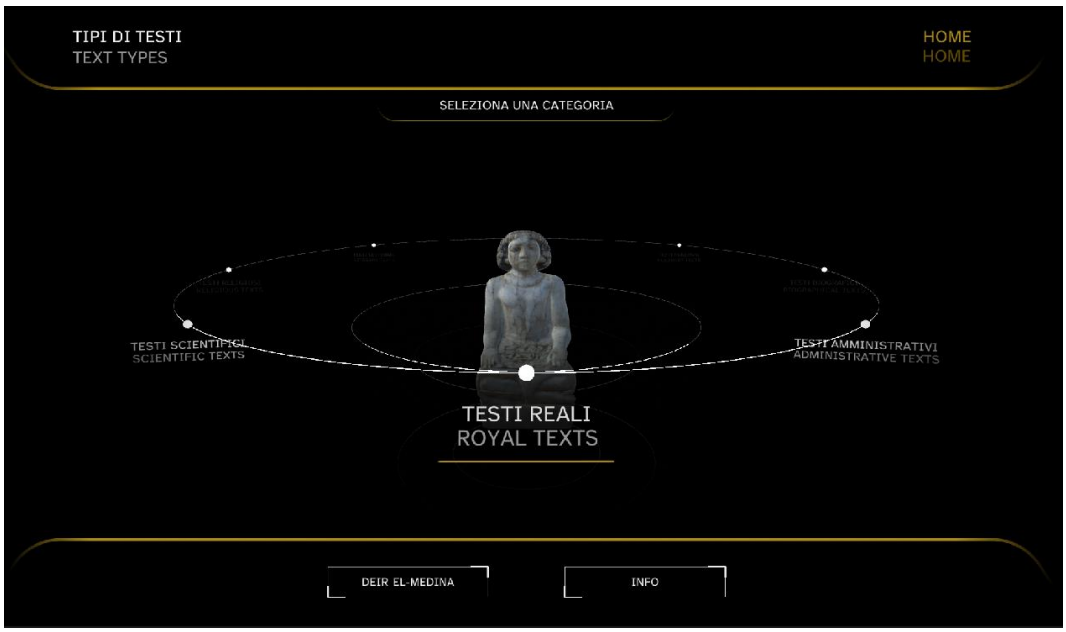


Figura 5: Categorie nella scena Home

Se è incuriosito da una certa categoria l'utente può toccarla e compariranno, sempre disposte in cerchio, le sottocategorie.

Il funzionamento della schermata è uguale sia per le categorie che per le sottocategorie. Infatti l'utente può ruotare anche le sottocategorie in base al suo movimento del dito. Ciò che cambia è l'elemento centrale della schermata perché non è più presente lo scriba, bensì il papiro di riferimento.

Figura 6: Sottocategorie nella scena Home

Le sottocategorie cliccabili, cioè quelle che permettono di passare alla sottoscena Testo e che quindi hanno una descrizione di riferimento, sono caratterizzate dall'immagine del proprio papiro a colori, mentre quelle non cliccabili sono caratterizzate dall'immagine di un papiro standard in bianco e nero in modo tale che anche visivamente l'utente capisce quali papiri possono essere approfonditi e quali no. Questo tipo di papiri sono disabilitati perché corrispondono a quei papiri ancora in fase di studio e presto inseriti all'interno dell'applicazione.

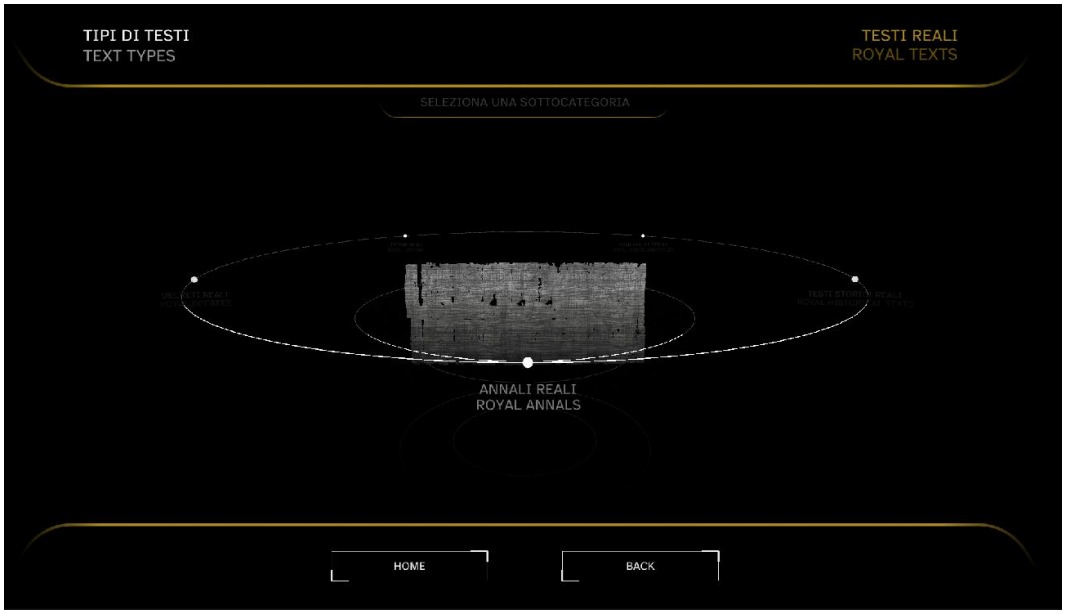


Figura 7: Sottocategorie non cliccabili

2.3.2 SOTTOSCENA TESTO

Se il visitatore è interessato a un tipo di testo in particolare può cliccare sulla sottocategoria e visualizzare più informazioni. Grazie ad un'animazione, si passa dalla sottocena Home alla sottoscena Testo. Le informazioni presenti compaiono un po' per volta tramite delle assolvenze e degli spostamenti verticali o orizzontali in base al tipo di informazione che presentano: nome del papiro, descrizione del papiro, *card* con il tipo di scrittura, *card* con il numero di inventario, immagine del papiro. Tutto ciò che è scritto è presente sia in italiano che in inglese in modo da agevolare eventuali visitatori stranieri. Se una descrizione è particolarmente lunga allora l'utente può scorrere verso il basso e visualizzare le informazioni nascoste. Questo accade anche per i papiri lunghi, e grazie a una *call to action*, l'utente può scorrere l'immagine e visualizzare le parti mancanti. Sono presenti infine delle frecce laterali che permettono all'utente di visualizzare il papiro precedente o successivo in base a quale delle due frecce viene toccata.

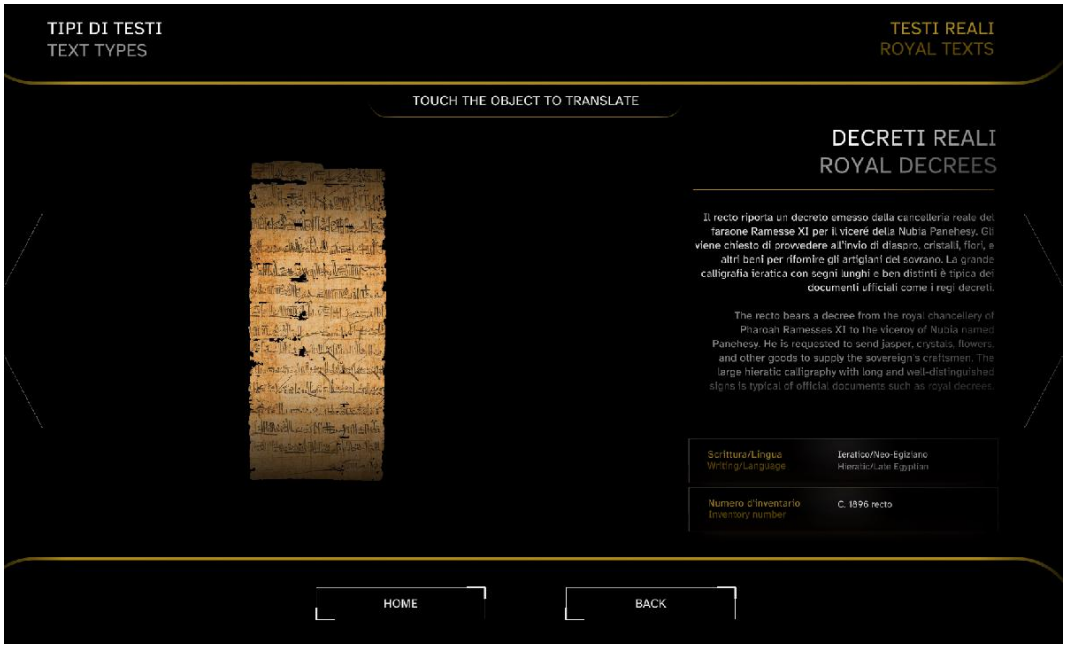


Figura 8: Schermata della sottoscena Testo

Sempre grazie ad una *call to action*, che si alterna a quella di scorrimento se è presente, si può capire quando un papiro ha la traduzione o meno. Se questa è presente vuol dire che l'utente può toccare il papiro per leggere la traduzione. Il passaggio dalla schermata di informazioni generali alla schermata di traduzione è segnato da una dissolvenza e dall'animazione delle varie informazioni come comparivano nella schermata precedente. Anche la struttura è molto simile, è presente un titolo, un testo e l'immagine. Ciò che mancano sono le card con il tipo di scrittura e il numero di inventario.

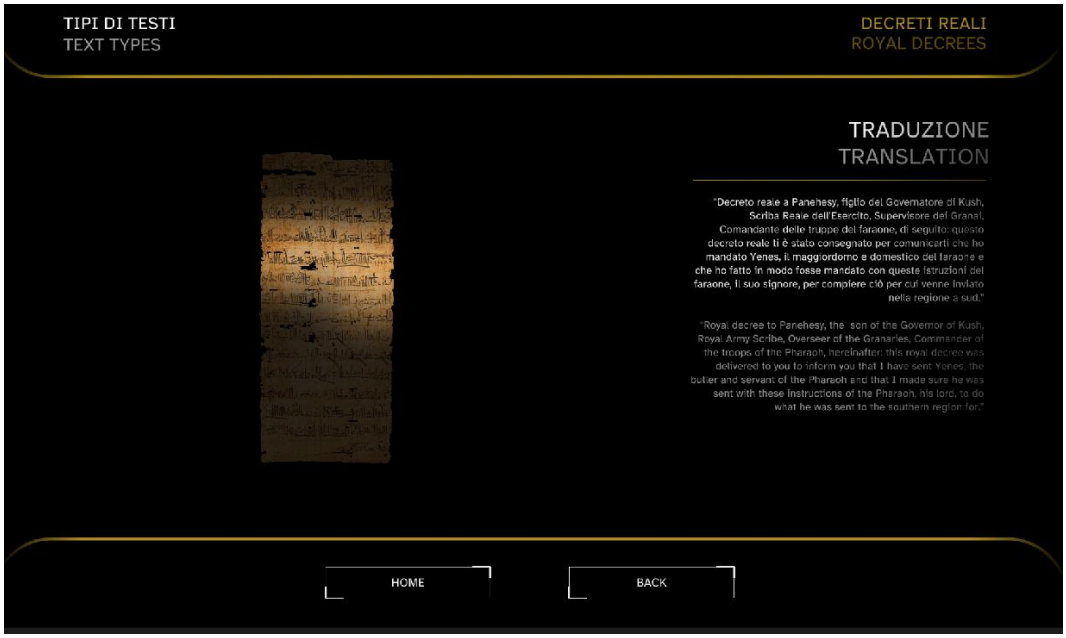


Figura 9: Schermata di traduzione

È possibile che alcuni papiri abbiano più traduzioni. In questo caso nella schermata saranno presenti delle frecce che permetteranno di procedere a visualizzare le traduzioni successive.

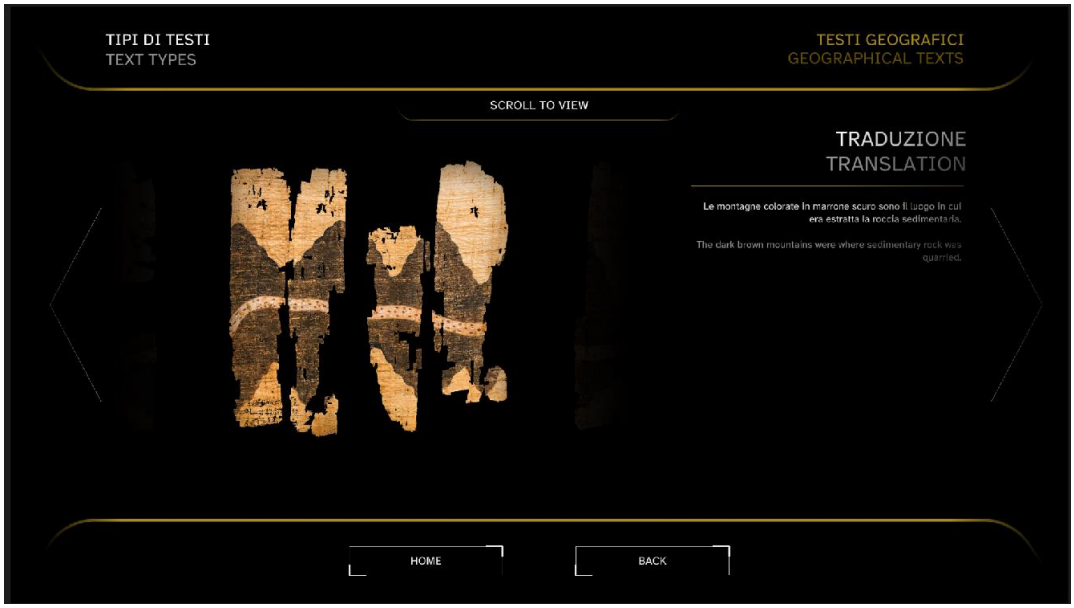


Figura 10: Schermata di traduzione con più di una traduzione

2.4 SCENA DEIR EL MEDINA 2

Nella sottoscena Home non sono stati analizzati i bottoni presenti nella schermata: Deir El Medina e Info.

Il tasto Info permette di visualizzare una schermata con alcune informazioni relativi ai papiri e come procedere nella comprensione dell'applicazione.

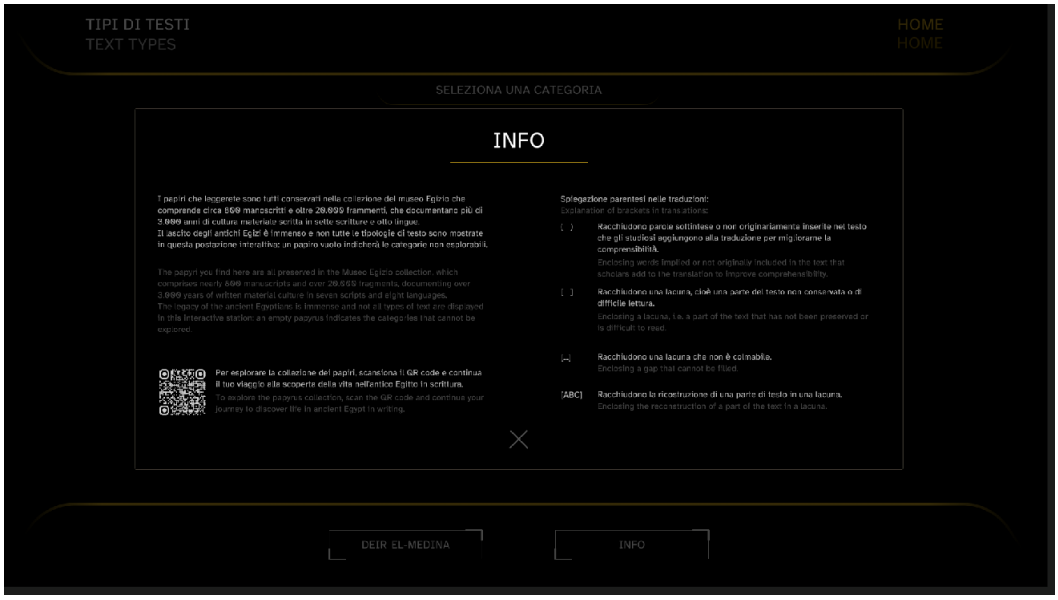


Figura 11: Schermata Info

Il tasto Deir El Medina invece porta alla scena Deir El Medina 2. Si chiama in questo modo perché è molto simile alla scena iniziale, con il modellino della città presente nella scena ma a differenza della prima, si possono conoscere alcuni scribi. Sono presenti infatti all'interno della città tre personaggi che possono essere cliccati per scoprire di più su di loro.

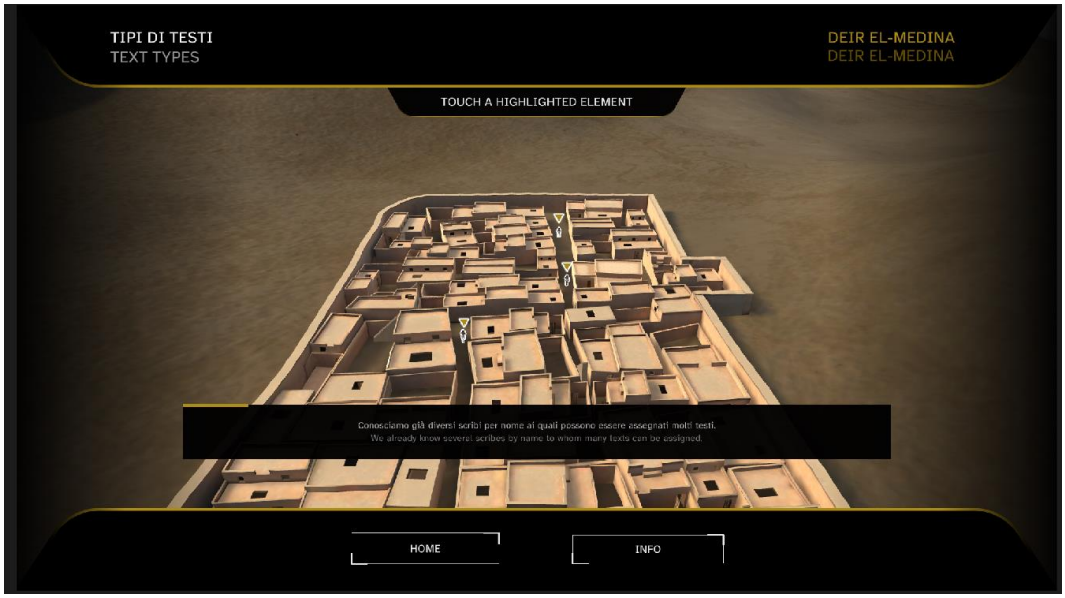


Figura 12: Scribi in Deir El Menina 2

Quando l'utente tocca uno degli scribi, la camera si avvicina ad esso presentandolo, come mostrato in figura.

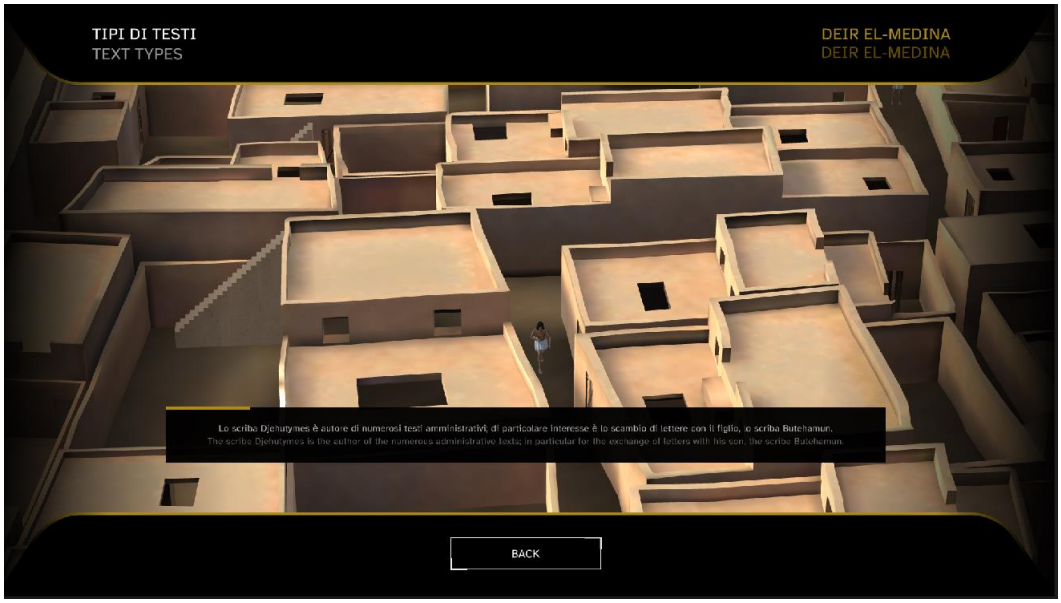


Figura 13: Dettaglio su uno scriba

Anche in questa scena è presente un tasto Info. All'interno della schermata sono state inserite le informazioni che sono state presentate nella prima scena di Deir El Medina in modo tale che se un visitatore salta per sbaglio la scena di inizio o

arriva davanti all'applicazione quando non è ancora stata riavviata, può comunque informarsi su ciò che è avvenuto prima.

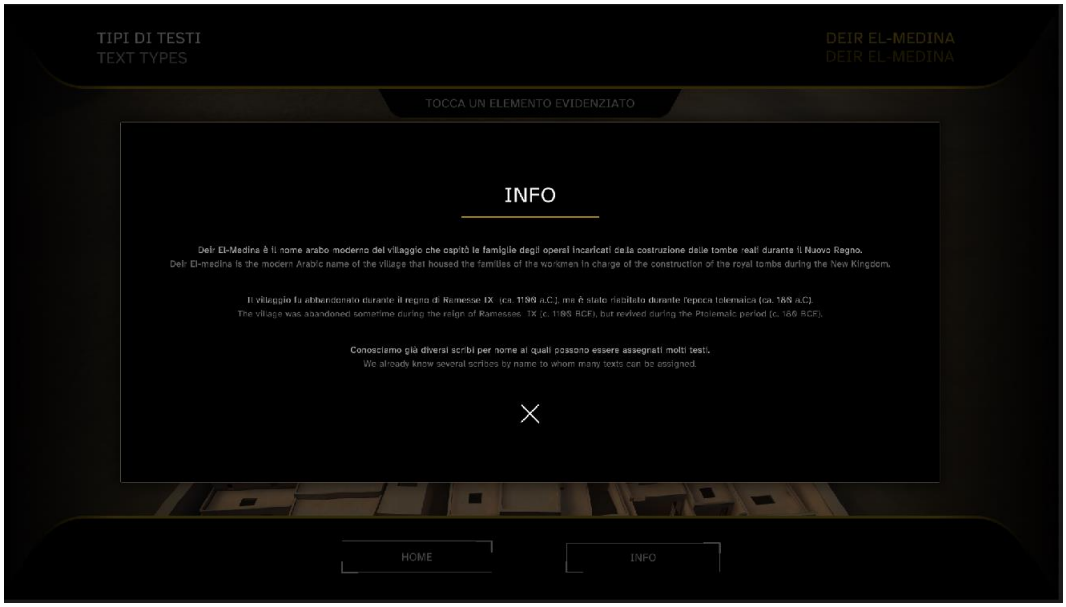


Figura 14: Schermata info in Deir El Medina 2

2.5 SCENA PERMANENTSCENE

La PermanentScene è una scena che è sempre presente ed è composta da un *header*, un *footer* e le *call to action*. Nell'*header* sono presenti le informazioni riguardanti il titolo dell'applicazione, *Tipi di testi*, e il suo corrispettivo titolo in inglese, a sinistra, e il titolo della scena o il papiro che si sta visualizzando a destra e anche questo sia in lingua italiana che in lingua inglese.

Figura 15: Header scena Deir El Medina

Nel *footer* sono presenti i bottoni che variano a seconda della scena in cui si è in quel momento:

- Back: per tornare indietro, presente in tutte le scene.

- Home: per tornare nella sottoscena Home se ci si trova nella sottoscena Testo o in Deir El Medina 2.
- Info: per poter visualizzare le informazioni relativa all'applicazione o a Deir El Medina in base alla scena in cui è presente il tasto, Deir El Medina o Deir El Medina 2.
- Start: per poter iniziare l'applicazione, presente solo all'inizio della scena Deir El Medina
- Skip: per saltare dalla scena Deir El Medina direttamente alla sottoscena Home, presente solo all'inizio della scena Deir El Medina.

Figura 16: Footer scena Deir El Medina

Infine sono presenti le *call to action*, ciò che fa capire all'utente cosa fare, come ad esempio le frecce per poter muovere il villaggio di Deir El Medina, per aprire la porta, o ancora le scritte presenti nella schermata della sottoscena Home o Testo in cui viene detto all'utente di scegliere una categoria, una sottocategoria o toccare il papiro per tradurlo o spostarlo per vederlo nella sua interezza.

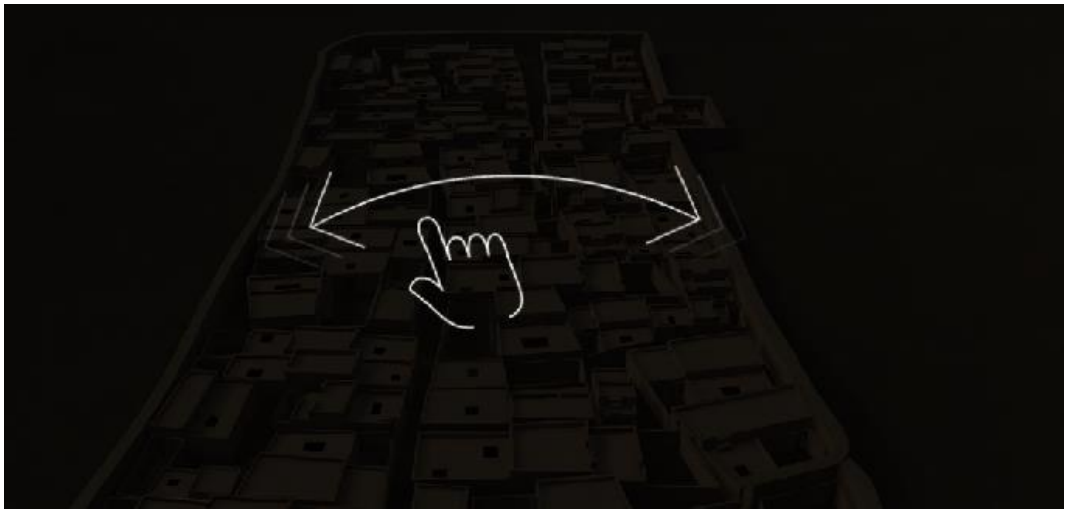


Figura 17: Esempio di call to action

Figura 18: Esempio di call to action

Ciò che si trova all'interno della PermanentScene deve essere sempre visibile, infatti durante il *play* dell'applicazione il contenuto viene spostato in una scena DontDestroyOnLoad che permette agli oggetti che non devono essere distrutti di sopravvivere dopo i vari cambi scena.

2.6 SCHERMATA DI PAUSA

Se l'applicazione rimane in stand by per un certo periodo allora la schermata che appare, in qualsiasi scena in cui ci si trova, è la schermata di pausa.

La schermata si presenta in un modo molto semplice: uno sfondo nero su cui sono presenti due scritte, "Sei ancora qui?" e "clicca per continuare". Così come tutto il resto dell'applicazione, anche in questo caso è presente il testo in inglese che si alterna a quello in italiano.

Figura 19: Schermata di pausa

L'idea di inserire una schermata di pausa così semplice è in linea con tutto il resto dell'applicazione, perché deve risultare intuitiva e di facile utilizzo. Poche informazioni aiutano l'utente a capire subito cosa deve fare.

L'utente può toccare un qualsiasi punto dello schermo per poter continuare ad andare avanti nella sua esplorazione, partendo dallo stesso punto in cui si era bloccato. Se invece la schermata di pausa rimane fissa per un minuto circa allora ciò che succede è che l'applicazione si resetta e riparte dall'inizio, dalla scena Deir El Medina.

3. LETTURA DEL DATABASE

In questo capitolo verrà mostrata la struttura del database e come tramite gli script JsonReader.js, EgyptianText.js, Category.js e SubCategory.js viene utilizzato nell'applicazione. Il database serve per contenere tutte le informazioni utili di un determinato papiro.

3.1 STRUTTURA DEL DATABASE

Il database nasce come tabella su Fogli Google. La scelta di creare un database online è stata fatta in modo tale da poterlo modificare in contemporanea da parte di tutti i componenti del team e avere una versione sempre aggiornata di questo. Il database è composto da 64 righe corrispondenti ai papiri e 14 colonne, che rappresentato:

- categoryNameITA: rappresenta il nome della categoria in italiano. È utile nella sottoscena Home per popolare il cerchio delle categorie e nella scena PermanentScene per inserire il titolo della categoria in alto a destra quando si è presenti nella schermata di visualizzazione del papiro.
- categoryNameENG: rappresenta il nome della categoria in inglese che come quello in italiano serve per le sottoscene Home e Testo.
- subCategoryNameITA: rappresenta il nome della sottocategoria in italiano. È utile nella sottoscena Home per popolare il cerchio delle sottocategorie, nella sottoscena Testo per il titolo del papiro che si sta visualizzando se si è nella schermata della descrizione del papiro altrimenti se si è nella schermata della traduzione questo elemento si trova come titolo della PermanentScene.

- subCategoryNameENG: rappresenta il nome della sottocategoria in inglese. Così come il suo corrispettivo in italiano serve per visualizzare i nomi inglesi delle sottocategorie nella sottoscena Home e nella sottoscena Testo.
- egyptianTextID: è un numero utile per il team per distinguere i papiri con il textInfoITA da quelli senza. Questo numero serve per capire quale immagine del papiro inserire all'interno del cerchio delle sottocategorie nella sottoscena Home, infatti gli elementi senza il textInfoITA sono quelli caratterizzati dall'immagine in bianco e nero dei papiri standard. Di conseguenza questi sono i papiri di cui non si può vedere la descrizione più dettagliata.
- catalogo: rappresenta il numero di catalogo del rispettivo papiro che è stato dato dal museo. È presente nella sottoscena Testo, nella *card* dedicata al numero di catalogo.
- tipo: definisce la lingua del testo in italiano. È presente nella sottoscena Testo, nella *card* dedicata alla lingua in cui è scritto il papiro.
- language: definisce la lingua del testo in inglese. Così come Tipo, è presente nella sottoscena Testo, nella *card* dedicata alla lingua.
- textTitleITA: rappresenta il titolo del testo in italiano.
- textTittleENG: rappresenta il titolo del testo in inglese.
- textInfoITA: contiene la descrizione in italiano del papiro di riferimento. È presente nella sottoscena Testo, nella schermata dedicata alla descrizione del papiro.
- textInfoENG: contiene la descrizione in inglese del papiro di riferimento. Viene utilizzato per il testo riguardante la descrizione del papiro presente nella sottoscena Testo.
- translationInfoITA: contiene la traduzione in italiano di una parte del papiro. È presente nella sottoscena Testo, nella schermata dedicata alla traduzione del papiro.
- translationInfoENG: contiene la traduzione in inglese di una parte del papiro. Viene utilizzato per il testo riguardante la traduzione del papiro presente nella sottoscena Testo.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	categoryNameITA	categoryNameENG	subcategoryNameITA	subcategoryNameENG	egyptianTextID	catalogo	tipo	language	textTitleITA	textTitleENG	textInfoITA	textInfoENG	translationInfoITA	translationInfoENG
2					-	-	-	-	-	-	-	-	-	-
3	Testi Reali	Royal Texts	Annali Reali	Royal Annals	-	-	-	-	-	-	-	-	-	-
4	Testi Reali	Royal Texts	Decreti Reali	Royal Decrees	0	C. 1896 recto	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Decreto Reale	Royal decree	Il recto riporta un decreto emesso dalla cancelleria reale del	The recto bears a decree from the royal chancellery of	'Decreto reale a Panehesy, figlio del Governatore di	'Royal decree to Panehesy, the son of the Governor of Kush,
5	Testi Reali	Royal Texts	Testi Storici Reali	Royal Historical Texts	1	C. 1875 recto	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Il Papiro della Congiura dell'Harem	The Harem Conspiracy Papyrus	Il papiro riporta i fatti relativi al processo che	This papyrus recounts the details of a trial	'Io (= il re) incaricai due fedelieri , , il	"I (= the king) commissioned two
6	Testi Reali	Royal Texts	Lettere Reali	Royal Letters	2	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Lettera reale	Royal letter	Il verso del 'Papiro delle Miniere' contiene	The verso of the 'Goldmine Papyrus'	'La bella statua del Re (realizzata) in	"The beautiful statue of the King (made) in fine
8	Testi Reali	Royal Texts	Nomi Reali e Titoli	Royal Names and Titles	3	C. 1874 verso	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Nomi e titoli reali	Royal names and titles	Si tratta di una lista cronologica scritta in ieratico conservata in 11	This is a chronological list written in hieratic, preserving 11	Osiride Seth Horus	Osiris Seth Horus
7	Testi Scientifici	Scientific Texts	Testi Astronomici/Astrologici	Astronomical/Astrological Texts	-	-	-	-	-	-	-	-	-	-
8	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne colorate in rosa	The pink-coloured mountains indicated the
9	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne colorate in marrone	The dark brown mountains were where
10	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne sono accessibili attraverso	The mountains are accessible by the
11	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	La grande struttura bianca in alto a	The large white structure at the top left
12	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Il tracciato al centro del papiro,	The spotted white and brown track running
13	Testi Scientifici	Scientific Texts	Testi Medico-Magici	Medical-Magical Texts	5	C. 1993 recto	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Testo magico	Magical text	Il manoscritto ieratico contiene diversi	This hieratic manuscript contains	'Ora, Iside era una donna saggia. Il suo	"Now, Isis was a wise woman. her heart was
14	Testi Scientifici	Scientific Texts	Testi Matematici	Mathematical Texts	-	-	-	-	-	-	-	-	-	-
15	Testi Scientifici	Scientific Texts	Testi Medici	Medical Texts	-	-	-	-	-	-	-	-	-	-
16	Testi Scientifici	Scientific Texts	Onomastica	Onomastica	6	S. 17507/005 recto	Iteratico/Egizio Antico	Hieratic/Old Egyptian	Papiro amministrativo con nomi	Amministrativ papyrus with names	Il recto presenta un testo amministrativo con	The recto contains an administrative list with	-	-
17	Testi Religiosi	Religious Texts	Testi Amuletici	Textual Amulets	7	P. 3586 recto	Iteratico/Egizio Classico	Hieratic/Classical Egyptian	Amuleto in papiro	Papyrus amulet	La striscia di papiro, dedicata al defunto	This papyrus strip, dedicated to the	'Parole dette da Osiride di	"Words spoken by Osiris of Pashermontu,
18	Testi Religiosi	Religious Texts	Decreti Divini	Divine Decrees	-	-	-	-	-	-	-	-	-	-

Figura 20: Tabella per il database (dalla riga 1 alla riga 18)

Tutti i papiri hanno i campi categoryNameITA e subCategoryNameITA (e i rispettivi campi in inglese) con il rispettivo contenuto. Quegli elementi che hanno delle colonne vuote sono identificate con il simbolo "-". I papiri che hanno il campo textInfoITA vuoto, sono quelli che, come detto precedentemente, sono i papiri relativi alle sottocategorie non cliccabili. Quelli invece che hanno il campo translationInfoITA vuoto sono quelli che non possono essere cliccati nella schermata di visualizzazione della descrizione del papiro nella sottoscena Testo per scoprire la traduzione, visto che non è presente nel database.

I papiri che hanno più traduzioni hanno più righe presenti nel database come mostrato nella figura.

8	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne colorate in rosa	The pink-coloured mountains indicated the
9	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne colorate in marrone	The dark brown mountains were where
10	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Le montagne sono accessibili attraverso	The mountains are accessible by the
11	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	La grande struttura bianca in alto a	The large white structure at the top left
12	Testi Scientifici	Scientific Texts	Testi Geografici	Geographical Texts	4	C. 1879+C. 1969+C. 1899+C. 2083/174+C.	Iteratico/Neo-Egiziano	Hieratic/Late Egyptian	Mappa geografica	Geographical map	Questo manoscritto, noto anche come	This manuscript, also known as 'The	Il tracciato al centro del papiro,	The spotted white and brown track running

Figura 21: Dettaglio database - Testi Geografici

Se si prende come riferimento il testo presentato in figura, i testi geografici sono scritti su 5 righe diverse e presentano gli stessi identici campi a differenza dei due campi finali riguardanti la traduzione in italiano e in inglese. Questo testo, così come gli altri che hanno più traduzioni, sono quelli che nella sottoscena Testo hanno le frecce anche nella parte delle traduzioni.

Una volta completata la scrittura del database, questo è stato scaricato e convertito in un file .json in modo da renderlo leggibile dagli script creati per il database.

Il file .json si compone come mostrato in figura.

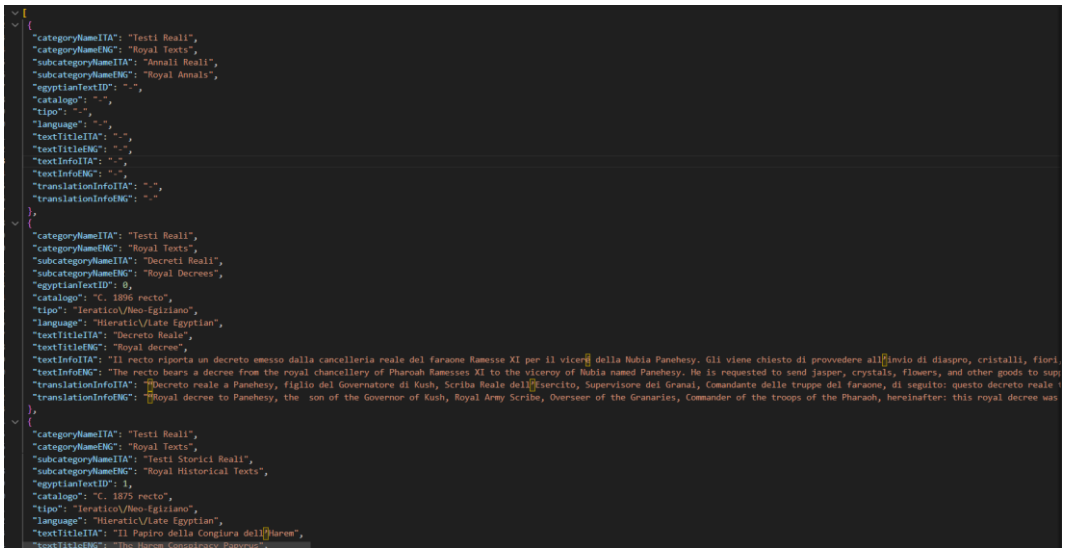


Figura 22: File .json del database

Ogni riga del database su file online, nel file .json diventa un elenco chiave: valore, racchiuso tra parentesi graffe.

3.2 EGYPTIANTEXT.JS

Lo script EgyptianText.js serve per gestire e memorizzare le informazioni che si trovano all'interno del database.

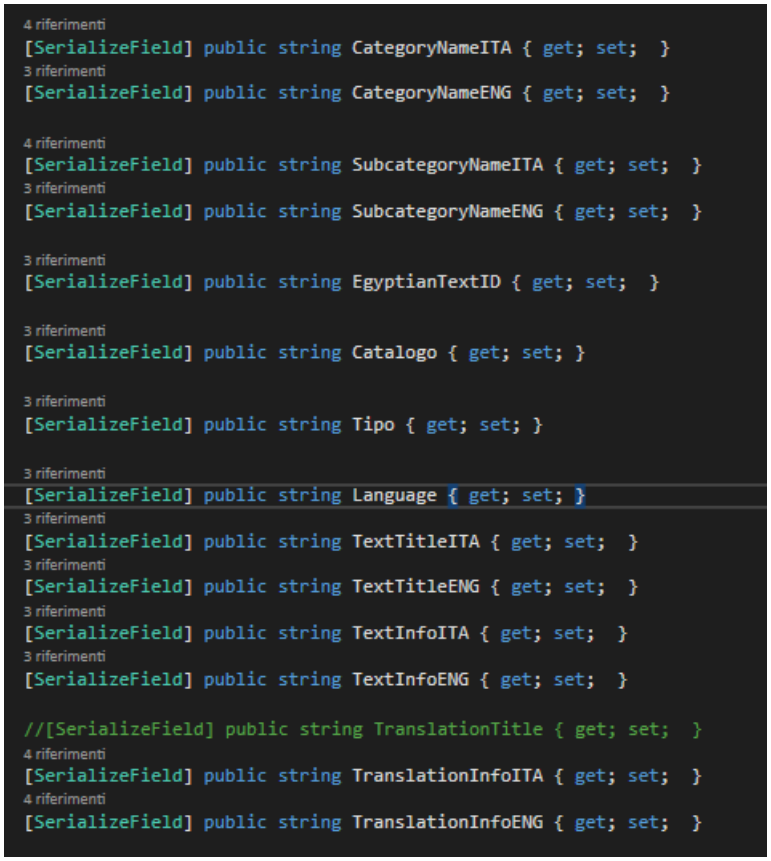


Figura 23: EgyptianText metodi get e set

Nella figura vengono mostrati tutti i campi del database. Sono tutti identificati come [SerializedField] così da essere visibili nell'inspector di Unity e così che gli altri script possano accedervi, anche perché sono di tipo public. I campi del database poi sono tutte stringhe infatti sono di tipo string.

Ogni campo ha un metodo set e un metodo get. Il primo imposta i valori, mentre il secondo legge i valori.

Il costruttore mostrato in questa figura serve per inizializzare le proprietà dell'istanza con i valori dell'oggetto che viene passato.

```
1 riferimento
public EgyptianText(EgyptianText p)
{
    CategoryNameITA = p.CategoryNameITA;
    CategoryNameENG = p.CategoryNameENG;

    SubcategoryNameITA = p.SubcategoryNameITA;
    SubcategoryNameENG = p.SubcategoryNameENG;

    EgyptianTextID = p.EgyptianTextID;

    Catalogo = p.Catalogo;
    Tipo = p.Tipo;
    Language = p.Language;

    TextTitleITA = p.TextTitleITA;
    TextTitleENG = p.TextTitleENG;

    TextInfoITA = p.TextInfoITA;
    TextInfoENG = p.TextInfoENG;

    //TranslationTitle = p.TranslationTitle;
    TranslationInfoITA = p.TranslationInfoITA;
    TranslationInfoENG = p.TranslationInfoENG;
    //LayoutType = p.LayoutType;

}
```

Figura 24: Costruttore di EgyptianText

3.3 JSONREADER.JS

Lo script JsonReader.js serve per elaborare i dati del file .json creato in precedenza per poterli utilizzare all'interno dell'applicazione.

```
// [SerializeField] private string jsonPath;
 [SerializeField] public DatabaseTest database;
 [SerializeField] public GameObject circles;
 public List<Category> categories = new List<Category>();

 Category emptyCategory ;
 SubCategory emptySubCategory;

 Comparison<SubCategory> customComparison = (x,y) =>
 {
     if(x.textInfoITA == "-" && y.textInfoITA != "-")
     {
         return 1;
     }
     else if(x.textInfoITA != "-" && y.textInfoITA == "-")
     {
         return -1;
     }
     else
     {
         return x.subCategoryTextITA.CompareTo(y.subCategoryTextITA);
     }
 };
```

Figura 25: Parametri di JsonReader.js

DatabaseTest database contiene una lista di oggetti di tipo EgyptianText che serve per popolare le categorie e le sottocategorie. La classe DatabaseTest viene mostrata in figura.

```
[System.Serializable]
2 riferimenti
public class DatabaseTest
{
    [SerializeField] public List<EgyptianText> egyptianTextArray = new List<EgyptianText>();

    0 riferimenti
    public DatabaseTest(List<EgyptianText> egyptianTextArray)
    {
        this.egyptianTextArray = egyptianTextArray;
    }
}
```

Figura 26: Classe DatabaseTest

La lista categories rappresenta le categorie presenti all'interno del database.

emptyCategory e emptySubCategory sono degli oggetti utilizzati come placeholder nel momento in cui vengono create delle nuove categorie o nuove sottocategorie.

customComparison è una funzione che serve per ordinare le sottocategorie.

Durante l'analisi dello script non ci occuperemo dell'oggetto circles in quanto è un oggetto utile nella sottoscena Home.

È presente una coroutine ReadJson() che legge il contenuto del database "TipiDiTesti_Installazione_Database" e lo salva come stringa grazie al metodo ToString(). Una volta letto e salvato come stringa, grazie al metodo JsonConvert.DeserializeObject la stringa viene salvata in una lista di oggetti di tipo EgyptianText e per ogni informazione vengono aggiunte le categorie grazie al metodo AddCategories().

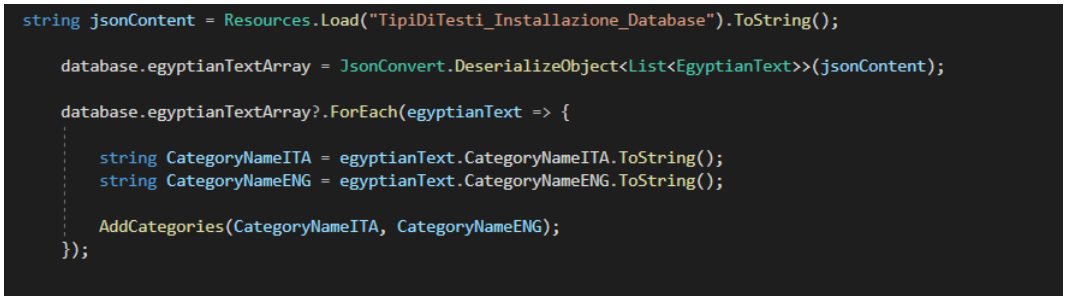


Figura 27: Coroutine ReadJson()

Il metodo AddCategories(string categoryNameITA, string categoryNameENG), mostrato in figura, aggiunge una nuova categoria nella lista delle categorie solo se non esiste già una categoria con lo stesso nome.

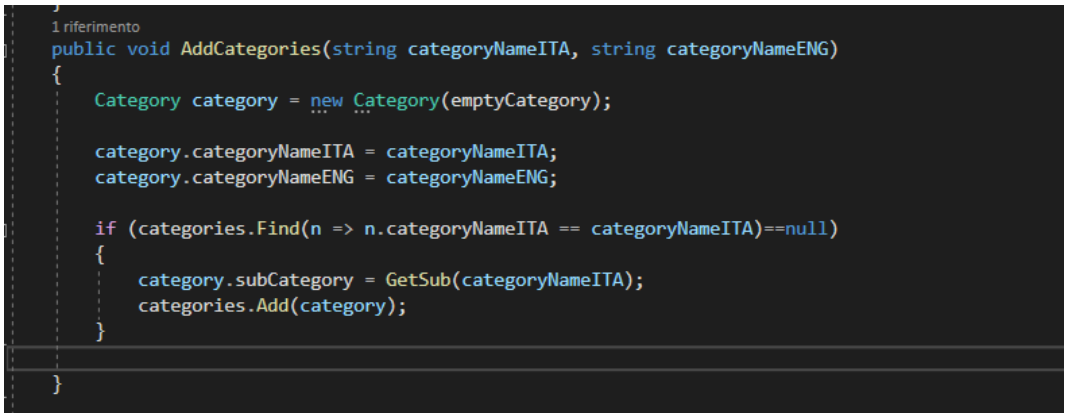


Figura 28: AddCategories(string categoryNameITA, string categoryNameENG)

All'interno di questa funzione viene chiamata la funzione GetSub(string categoryNameITA), che come mostrato in figura, prende una categoria come

input, restituendo la lista delle sottocategorie relative a quella specifica categoria. All'interno della lista delle sottocategorie sono presenti tutte le informazioni utili viste anche precedentemente nella lettura del database.

```
1 riferimento
public List<SubCategory> GetSub(string categoryNameITA)
{
    List<SubCategory> tempList = new List<SubCategory>();
    database.egyptianTextArray?.ForEach((egyptianText) =>
    {
        if (egyptianText.CategoryNameITA == categoryNameITA)
        {
            SubCategory temp = tempList.Find(s => s.subCategoryTextITA == egyptianText.SubcategoryNameITA);
            if (temp == null)
            {
                temp = new SubCategory(emptySubCategory);
                temp.subCategoryTextITA = egyptianText.SubcategoryNameITA;
                temp.subCategoryTextENG = egyptianText.SubcategoryNameENG;
                temp.egyptianTextID = egyptianText.EgyptianTextID;
                temp.catalogo = egyptianText.Catalogo;
                temp.tipo = egyptianText.Tipo;
                temp.lanaguage = egyptianText.Language;
                temp.textTitleITA = egyptianText.TextTitleITA;
                temp.textTitleENG = egyptianText.TextTitleENG;
                temp.textInfoITA = egyptianText.TextInfoITA;
                temp.textInfoENG = egyptianText.TextInfoENG;

                temp.translationInfoITA.Add(egyptianText.TranslationInfoITA);
                temp.translationInfoENG.Add(egyptianText.TranslationInfoENG);
                tempList.Add(temp);
            }
            else
            {
                temp.translationInfoITA.Add(egyptianText.TranslationInfoITA);
                temp.translationInfoENG.Add(egyptianText.TranslationInfoENG);
            }


        }
    });
    tempList.Sort(customComparison);
    return tempList;
}
```

Figura 29: GetSub(string categoryNameITA)

3.4 CATEGORY.JS

Questo script contiene una lista subCategory e due stringhe, categoryNameITA e categoryNameENG. Gestisce le categorie e le sottocategorie all'interno dell'applicazione.

Le due stringhe rappresentano i nomi in italiano e in inglese della categoria, mentre la lista è una lista di oggetti di tipo SubCategory che contiene le sottocategorie associate a quella categoria. Più avanti verrà mostrato lo script relativo a SubCategory.

Il costruttore Category(Category c) inizializza una nuova istanza con i valori dell'oggetto che viene passato.

Infine l'enumeratore CategoryID definisce le categorie possibili: reali, scientifici, religiosi, letterali, legali, funerari, biografici, amministrativi.

Nella figura viene mostrato il codice che definisce lo script Category.js.

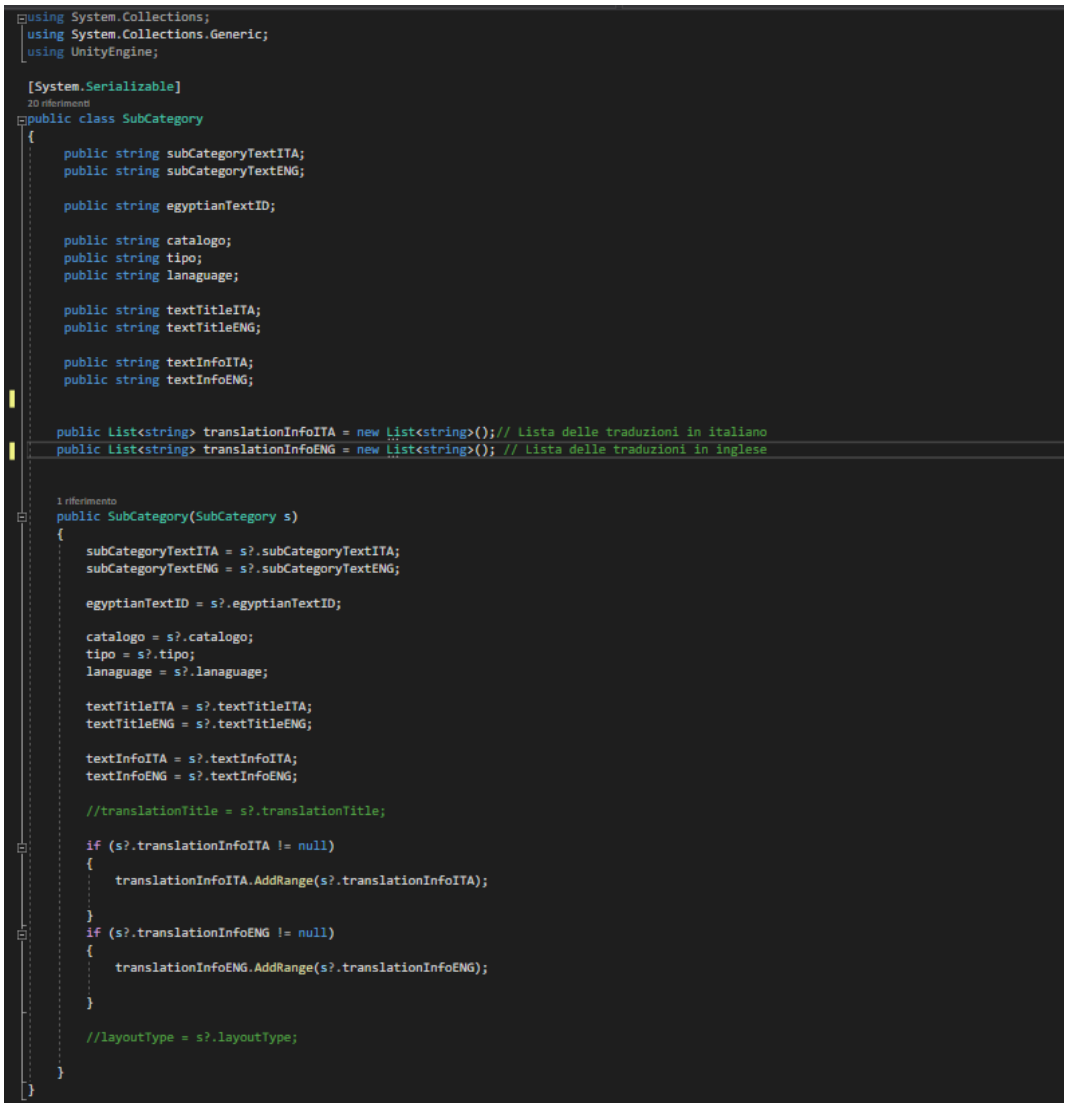
```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

[System.Serializable]

24 riferimenti
public class Category
{
    public List<SubCategory> subCategory = new List<SubCategory>();

    public string categoryNameITA;
    public string categoryNameENG;


    1 riferimento
    public Category(Category c)
    {

        categoryNameITA = c?.categoryNameITA;
        categoryNameENG = c?.categoryNameENG;
        subCategory = c?.subCategory;
    }

}

0 riferimenti
public enum CategoryID
{
    Reali, Scientifici, Religiosi, Letterari, Legali, Funerari, Biografici, Amministrativi
}
```

Figura 30: Script Category.js

3.5 SUBCATEGORY.JS

Lo script SubCategory.js è molto simile allo script precedente, Category.js. Sono presenti diverse stringhe che rappresentano i diversi campi del database e una lista di stringhe per quanto riguarda le traduzioni. Rispetto a Category.js, ciò che cambia nel costruttore di SubCategory è che è presente l'operatore if per quanto riguarda la lista di traduzioni. Per ogni sottocategoria, se sono presenti più traduzioni, queste vengono aggiunte nella lista.

Nella figura viene mostrato il codice dello script SubCategory.js.



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

[System.Serializable]
20 riferimenti
public class SubCategory
{
    public string subCategoryTextITA;
    public string subCategoryTextENG;

    public string egyptianTextID;

    public string catalogo;
    public string tipo;
    public string lanaguage;

    public string textTitleITA;
    public string textTitleENG;

    public string textInfoITA;
    public string textInfoENG;


    public List<string> translationInfoITA = new List<string>();// Lista delle traduzioni in italiano
    public List<string> translationInfoENG = new List<string>(); // Lista delle traduzioni in inglese


    1 riferimento
    public SubCategory(SubCategory s)
    {
        subCategoryTextITA = s?.subCategoryTextITA;
        subCategoryTextENG = s?.subCategoryTextENG;

        egyptianTextID = s?.egyptianTextID;

        catalogo = s?.catalogo;
        tipo = s?.tipo;
        lanaguage = s?.lanaguage;

        textTitleITA = s?.textTitleITA;
        textTitleENG = s?.textTitleENG;

        textInfoITA = s?.textInfoITA;
        textInfoENG = s?.textInfoENG;

        //translationTitle = s?.translationTitle;

        if (s?.translationInfoITA != null)
        {
            translationInfoITA.AddRange(s?.translationInfoITA);

        }
        if (s?.translationInfoENG != null)
        {
            translationInfoENG.AddRange(s?.translationInfoENG);

        }

        //layoutType = s?.layoutType;

    }
}
```

Figura 31: Script SubCategory.js

3.6 DATABASE NELL'INSPECTOR DI UNITY

Tutto ciò che viene eseguito negli script visti precedentemente ha un riscontro su Unity. Nella scena Home è presente un oggetto Database a cui viene passato come script JsonReader.js. Se l'applicazione viene messa in *play*, ciò che si vede

nell'Inspector è un elenco delle categorie classificati come 8 elementi come mostrato in figura.

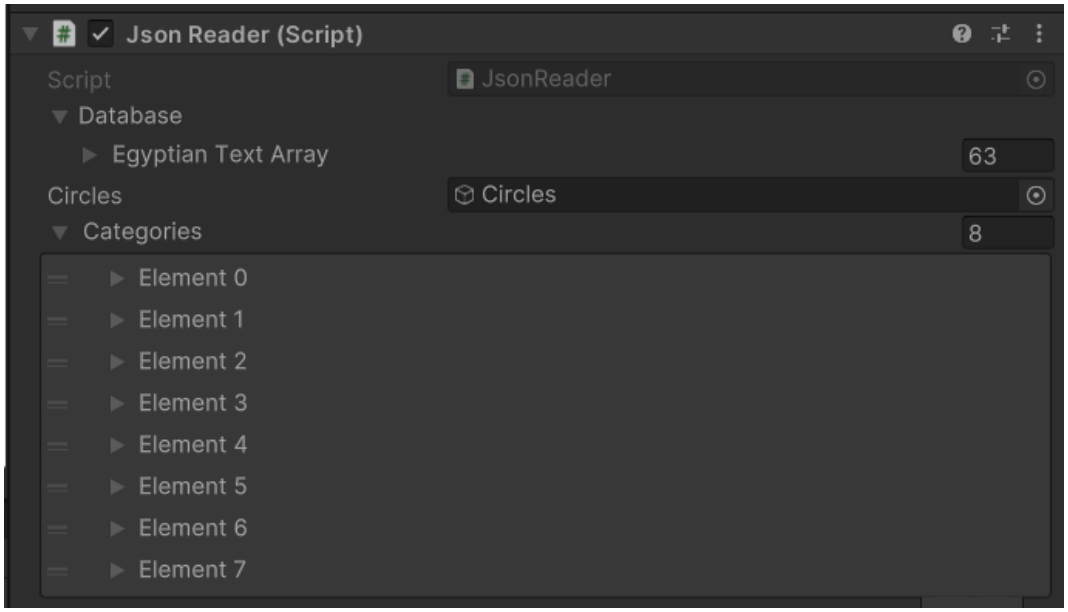


Figura 32: Database nell'inspector Unity

Ogni elemento rappresenta una categoria. Prendendo come riferimento Element 0, ciò che viene mostrato è il nome della categoria sia in italiano che in inglese e una lista di sottocategorie, in questo caso, testi reali, sono presenti 5 sottocategorie.

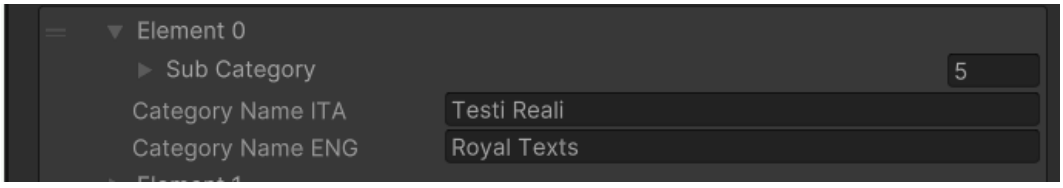


Figura 33: Categoria Testi Reali nell'inspector Unity

Scendendo più nel dettaglio, la lista delle sottocategorie è a sua volta una lista che, per ogni sottocategoria, racchiude le informazioni necessarie.

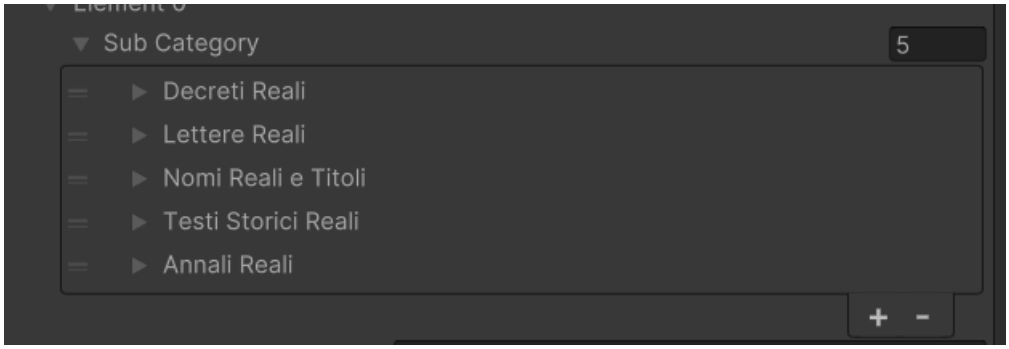


Figura 34: Lista sottocategorie

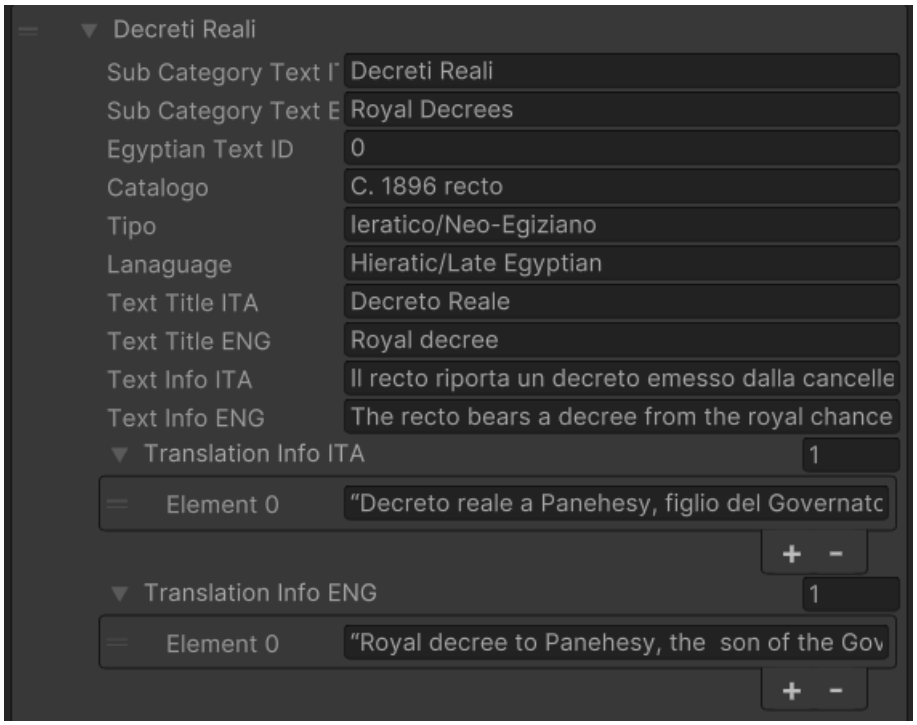


Figura 35: Dettaglio sottocategoria Decreti Reali

Nel caso dei decreti reali, il numero di traduzioni è 1 e infatti la lista presenta solo un elemento. Se si prendesse come riferimento una sottocategoria con più traduzioni, come nel caso dei testi geografici, ciò che si vede nell'Inspector è una lista con tutte le traduzioni presenti, come mostrato in figura.

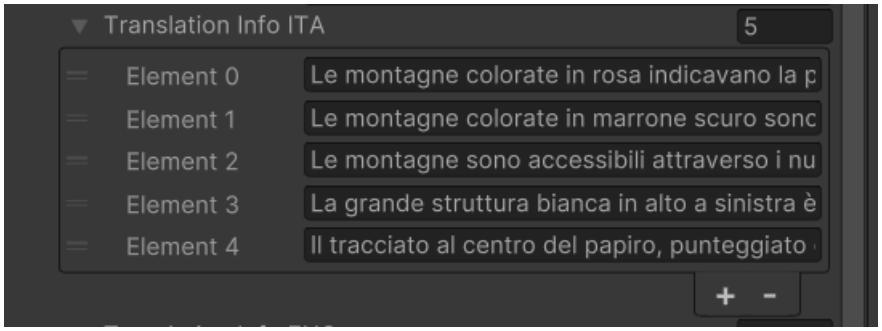


Figura 36: Lista di traduzioni

Se sono presenti delle sottocategorie con delle voci vuote, ciò che viene mostrato in figura è il simbolo "-". La lista delle traduzioni presenta un valore che è sempre il simbolo "-".

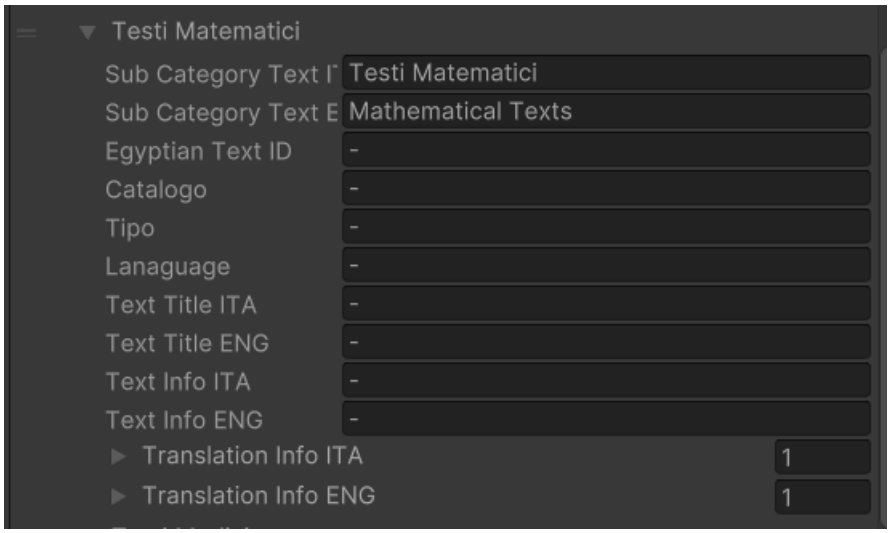


Figura 37: Sottocategoria vuota

4. SOTTOSCENA TESTO

In questo capitolo viene mostrato il funzionamento della sottoscena Testo, attraverso gli oggetti utilizzati su Unity e tramite gli *script* ad essi associati, MainPanelManager, PanelPapiroManager, ManagerPanelTraduzione, ArrowButtonsPapiroManager, ArrowButtonsTraduzioneManager. Verrà fatta un'analisi più approfondita sugli script relativi alla parte del papiro rispetto a quelli relativi alle traduzioni perché il funzionamento di essi è sostanzialmente lo stesso.

4.1 HIERARCHY E INSPECTOR UNITY

Nella scena Home. Per distinguere le due sottoscene sono stati inseriti due empty che racchiudono gli elementi di una o dell'altra sottoscena. Nel caso della sottoscena Testo, l'*empty* di riferimento è Testo_TOTAL, che racchiude un canvas con all'interno il BG che è un background di colore nero e due panel, Panel_Papiro e Panel_Traduzioni.

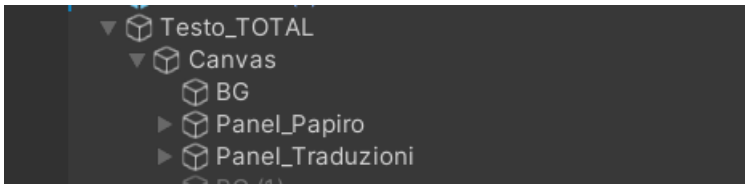


Figura 38: Empty Testo_TOTAL

I due pannelli sono molto simili tra loro, presentano gli stessi elementi e il funzionamento di essi è quasi uguale. È stato scelto di inserire due "copie" per rendere più semplice ed efficace il codice. Utilizzare infatti diverse funzioni sullo stesso elemento avrebbe portato ad errori di continuità all'interno dell'applicazione ed a enormi *bug* difficilmente risolvibili.

All’interno di questi due panel sono racchiusi tutti gli elementi utili nella sottoscena. Entrambi i panel hanno associato un canvas group e i suoi parametri sono controllati dagli script che verranno analizzati più avanti.

Figura 39: CanvasGroup

Alpha permette al pannello di essere visibile o meno. Se alpha è 1 allora questo sarà visibile, se è 0 no. Interactable e block raycasts permettono al pannello di essere interagibile o meno. Attraverso il codice questi parametri cambieranno in quanto i pannelli devono essere visibili e interagibili solo se l’altro non è visibile e non è interagibile.

All’interno di Panel_Papiro sono presenti diversi oggetti:

- Image_Anchor è un *empty* utile per tenere la posizione finale in cui l’immagine del papiro scelto verrà centrata.
- Panel_immagine_scrollArea_lettere è il pannello in cui verrà impostata l’immagine e il bottone per la traduzione delle lettere.
- Panel_immagine_scrollArea_immagini è uguale come struttura al pannello precedente ma le immagini che sono inserite sono tutte quelle dei restanti papiri. Non è stato possibile utilizzare un panel solo per tutti i papiri in quanto i papiri lunghi sono scorribili orizzontalmente, mentre le lettere devono essere scorribili in verticale. Così come la soluzione di avere due pannelli papiro e traduzione risulta efficace, anche quella di inserire due pannelli lettere e immagini risulta di più semplice controllo tramite codice.
- Shadow_orizzontale racchiude due immagini di ombre che permettono alle immagini dei papiri lunghi di essere visibili solo in una certa area e di essere nascosti fuori da questa. La separazione tra visibile e nascosto non è

netta perché le sfumature delle immagini presenti qui dentro servono apposta a rendere più morbida la transizione.

- Shadow_lettere si comporta allo stesso modo della Shadow_orizzontale ma così come per i Panel_immagine, questa shadow è personalizzata per le lettere. Anche in questo caso si è preferito separare i due elementi shadow per non avere problemi di esecuzione dell'applicazione.
- Panel_ColonnaTesti racchiude tutti i testi presenti.
- ArrowButtonsPapiro ha al suo interno le due frecce utili per cambiare papiro.

Figura 40: Dettaglio su Panel_Papiro

Più nel dettaglio, se si prende in considerazione Panel_immagine_scrollArea_immagini, si ha che all'interno di questo panel è presente una gerarchia di oggetti. Quest'ultima è stata creata con l'intento di impostare l'immagine del papiro collegata a una *scrollbar* in modo tale da poter scorrere l'immagine del papiro in modo semplice.

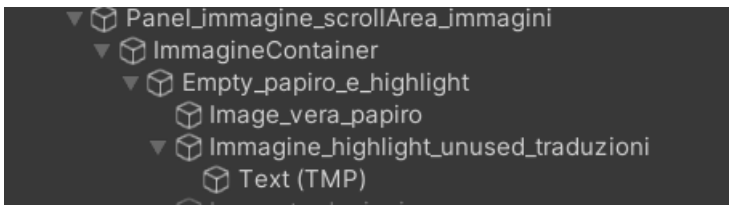


Figura 41: Panel_immagine_scrollArea_immagini

L'Empty_papiro_e_highlitht contiene:

- Image_vera_papiro: l'immagine del papiro vera e propria.

- Immagine_highlight_unused_traduzioni: è un bottone che si attiva o disattiva in base alla presenza o meno della traduzione del papiro che si sta visualizzando. Se il papiro di riferimento ha la traduzione, allora questo bottone permetterà di attivare una funzione onClick() che porta a visualizzare la schermata di traduzione, attraverso il codice scritto all'interno dello script MainPanelManager.js.

Anche Panel_immagine_scrollArea_lettere si comporta nello stesso modo, ma il senso di scroll invece di essere orizzontale, è verticale.

Panel_ColonnaTesti è un pannello che contiene diversi testi:

- Title_Sottocategoria rappresenta il nome della sottocategoria in italiano.
- Title_Subcategory rappresenta il nome della sottocategoria in inglese.
- Line è una linea dorata che serve per distinguere il titolo dato dalla sottocategoria dal resto composto da testo e card su inventario e lingua.
- CanvasColonnaTesti contiene una scrollArea con il testo in italiano e in inglese. Questo testo è scrollabile verticalmente e così come le immagini, il testo che esce dall'area di visualizzazione viene nascosto ed è presente una sfumatura, Shadow_testi per visualizzare una dissolvenza graduale man mano che il testo scorre.
- EmptyInvScrit è un *empty* che contiene due card. La prima contiene il tipo di scrittura/lingua, la seconda invece contiene il numero di inventario. Le card si presentano anch'esse con una gerarchia al loro interno:
    - Background è lo sfondo della *card*.
    - ScrLing nella *card* Scrittura/Lingua e il suo corrispettivo NInventario Testo nella *card* Inventario contengono i titoli in italiano e inglese di ciò che si può leggere nella *card*, quindi la lingua del papiro e il numero di inventario.
    - ScritturaIta, ScritturaEng nella *card* Scrittura/Lingua corrispondono alla lingua in italiano e inglese del papiro e questo cambia in base all'elemento visualizzato
    - Cat_inventario presente nella *card* Inventario corrisponde al numero di inventario dell'elemento selezionato. Più avanti verrà

mostrato che se un numero di inventario è corto, allora questo si dispone al centro della card, altrimenti si dispone su due righe.

Figura 42: Panel_ColonnaTesti

Panel_Traduzioni ha una struttura pressoché uguale a quella di Panel_Papiro. Rispetto a quest'ultimo però, Panel_Traduzioni presenta un solo Panel_immagine_scrollArea perché l'immagine delle lettere nel caso delle traduzioni non è scrollabile. Di conseguenza non serve avere una shadow specifica solo per le lettere. Infine ciò che cambia è l'assenza delle card del numero di inventario e il tipo di scrittura/lingua.

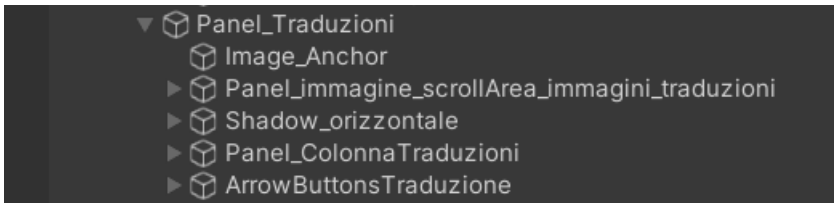


Figura 43: Panel_Traduzioni

4.2 MAINPANELMANAGER.JS

Lo script MainPanelManager.js è lo script di riferimento della sottoscena e contiene alcuni metodi di controllo sul passaggio dal pannello del papiro al pannello delle traduzioni e viceversa.

Lo script si apre con il metodo Start(), un metodo che permette di inizializzare i parametri che ci interessano. In questo caso i due pannelli del papiro e della traduzione vengono impostati come non visibili, e non interagibili. Questo perché questo metodo viene richiamato nel momento in cui si passa dalla scena di Deir El Medina alla scena Home, sottoscena Home. Qui i due pannelli devono essere invisibili e non devono intralciare in nessun modo la parte di scelta del papiro da visualizzare.

```
private void Start()
{

    SetInteractive(_panelPapiro_CG, 0, false);
    SetInteractive(_panelTraduzioni_CG, 0, false);
    //Debug.Log("inizializzo tutto a false riga 92");

    //_background_CR.SetAlpha(0f); //perche usa canvasrenderer?
    _background_CG.alpha = 0;

    InitializePanelPapiroUI(); //inizializzo tutti i settings relativi a PanelPapiroManager

    InitializePanelTraduzioniUI();//inizializzo tutti i settings relativi a PanelTraduzioniManager

}
```

Figura 44: Start()

All'interno dello script sono inseriti dei metodi utilizzati per controllare alcuni parametri dell'interfaccia.

I metodi GetCategoriaITA(), GetCategoriaENG(), GetSottoCategoriaITA(), GetSottoCategoriaENG() sono metodi che prendono la categoria o la sottocategoria di riferimento e la inseriscono come titolo della schermata che si sta osservando in quel momento. I metodi PapiroScrolling(), TraduzioneScrolling() e Lettere() servono per mostrare o meno la call to action nella schermata. Se la call to action è visibile allora il papiro è scrollabile altrimenti no. Il metodo

HaTraduzione() mostra o meno la call to action che porta l'utente a cliccare sull'immagine del papiro per aprire la traduzione di riferimento.

Non vengono analizzati più approfonditamente in quanto sono metodi utili per la PermanentScene, scena non oggetto di studio dell'elaborato.

```
2 riferimenti
public string GetCategoriaITA() //serve per dare il nome della categoria in interfaccia
{
    return ButtonActions.selectedCategory.categoryNameITA;
}


2 riferimenti
public string GetCategoriaENG() //serve per avere il nome della categoria eng in interfaccia
{
    return ButtonActions.selectedCategory.categoryNameENG;
}

1 riferimento
public string GetSottoCategoriaITA() //serve per dare il nome della sottocategoria in interfaccia
{
    return ButtonActions.selectedSubcategory.subCategoryTextITA;
}

1 riferimento
public string GetSottoCategoriaENG() //serve per dare il nome della sottocategoria eng in interfaccia
{
    return ButtonActions.selectedSubcategory.subCategoryTextENG;
}
```

Figura 45: Metodi per prendere i nomi di categorie e sottocategorie

```
2 riferimenti
public bool PapiroScrolling() //serve per sapere se devi mostrare le CTA quando e scrollabile
{
    if(PanelPapiroManager.Instance.immagine_vera_papiro.rectTransform.rect.width > 2132.473f) return true;
    else return false;

}
1 riferimento
public bool TraduzioneScrolling() //serve per sapere se devi mostrare le CTA quando e scrollabile
{
    if (ManagerPanelTraduzioni.Instance.immagine_vera_con_shadow_traduzione.rectTransform.rect.width > 2132.473f)  return true;
    else return false;

}
1 riferimento
public bool Lettere() //serve per sapere se devi mostrare le CTA quando e verticale (lettere)
{
    if (PanelPapiroManager.Instance.IsLettere())  return true;
    else return false;
}
1 riferimento
public bool HaTraduzione() //serve per sapere se devi mostrare le CTA quando puoi aprire il pannello traduzione
{
    if (PanelPapiroManager.Instance.button_traduzioni_highlight.gameObject.activeSelf) return true;
    else return false;
}
```

Figura 46: Metodi per capire se un papiro è scrollabile o ha la traduzione

Il metodo OpenPapiro() serve per aprire la schermata di riferimento del papiro. All'interno di questo metodo vengono impostati alcuni parametri come il background visibile (alpha a 1) e il valore booleano isPapiro a true. Questo valore

è utile per capire in quale dei due pannelli visibili mi trovo, infatti esiste anche il suo corrispettivo isTraduzione. Serviranno anche più avanti per la funzione che permette di tornare indietro e visto che è stata scritta una funzione unica sia nel caso della traduzione che nel caso del papiro, era utile un controllo di tipo booleano.

In più vengono richiamate alcune funzioni di PanelPapiroManager. Queste funzioni servono per inserire le informazioni del papiro nella parte destra della schermata (SetPapiro()), inserire l'immagine nella parte della schermata dedicata all'immagine (SetImmaginePapiro()), animare la pagina (AnimatePage()). Queste funzioni verranno dettagliate più avanti.

```
public void OpenPapiro() //Serve per impostare e aprire la pagina del papiro
{
    //Tutorial.instance.SetDepth(2);//papiro
    _background_CG.alpha = 1;
    //_background_CR.SetAlpha(1f);

    isPapiro = true;

    //prendere informazioni della sottocategoria che ho cliccato: le prende in SetPapiro con ButtonActions.selectedSubcategory

    PanelPapiroManager.Instance.SetPapiro(); //mettere le informazioni nel posto giusto

    PanelPapiroManager.Instance.SetImmaginePapiro(); //mettere l'immagine giusta e l'immagine sopra della traduzione

    //TODO: animare la pagina
    //PanelPapiroManager.Instance.AnimatePage();
    PanelPapiroManager.Instance.currentTween = PanelPapiroManager.Instance.AnimatePage();
}
```

Figura 47: OpenPapiro()

Così come OpenPapiro() esiste anche un metodo OpenTraduzioni() che permette di aprire il pannello di traduzione. Il valore booleano isPapiro quindi viene impostato a false, mentre isTraduzione a true.

Viene eseguita una animazione di uscita del pannello del papiro tramite il metodo ExitFade() e subito dopo vengono impostati i parametri corretti della traduzione di riferimento attraverso alcune funzioni descritte da ManagerPanelTraduzioni ed eseguita una animazione di entrata del pannello della traduzione attraverso AnimatePageTraduzione(). Anche questi metodi verranno dettagliati più avanti.

```
public void OpenTraduzioni()
{
    //Tutorial.instance.SetDepth(3);//traduzioni
   // Debug.Log("Clicco l'immagine");
    Sequence sequence = DOTween.Sequence();

    isPapiro = false;
    isTraduzione = true;
    //_panelPapiro_CG.blocksRaycasts = false;
    //_panelPapiro_CG.interactable = false;
    SetInteractive(PanelPapiroManager.Instance.PortaFrecce, 2, false); //1 o 2? meglio 2
    SetInteractive(_panelPapiro_CG, 2, false);//idem
    sequence.Append(PanelPapiroManager.Instance.ExitFade())
        .Append(_panelPapiro.GetComponent<CanvasGroup>().DOFade(0, duration));
           /*
            .AppendCallback(() =>
           {

           });
            */

        ManagerPanelTraduzioni.Instance.tradIndex = 0;
        ManagerPanelTraduzioni.Instance.CheckIndex();

        ManagerPanelTraduzioni.Instance.SetTraduzione(); //metto le informazioni al posto giusto

        //TODO: mettere l'immagine di traduzione giusta e l'immagine sopra della traduzione
        Tutorial.instance.GoDeeper(3);
        //TODO: animo la pagina
        ManagerPanelTraduzioni.Instance.currentTraduzioneTween = ManagerPanelTraduzioni.Instance.AnimatePageTraduzione();


}
```

Figura 48: OpenTraduzioni()

Il metodo SetInteractive(CanvasGroup group, int value, bool val) serve per impostare i valori corretti in funzione del pannello che si sta visualizzando.

```
public void SetInteractive(CanvasGroup group, int value, bool val)
{
    if (value != 2) //2 = non eseguire questa parte.
    {
        group.alpha = value; //metto a 0 o 1 alpha del panel papiro
    }
    group.blocksRaycasts = val; //abilita o disabilita il raycast interessato
    group.interactable = val; //abilita o disabilita interattività

}
```

Figura 49: SetInteractive(CanvasGroup group, int value, bool val)

Questo metodo viene utilizzato nelle funzioni Start(), BackInPapiro() e BackInSubcategory() di MainPanelManager per impostare i parametri di raycasts e interactable a false, mentre negli script PanelPapiroManager.js e MainPanelTraduzione.js per impostare questi parametri a true o a false in base alla funzione in cui vengono richiamati.

Se ci si trova nella schermata delle traduzioni e si vuole cliccare Back per tornare nella schermata del papiro viene richiamata la funzione BackInPapiro(). All'interno di questa funzione viene eseguita una dissolvenza grazie a ExitFade() che è una funzione all'interno di ManagerPanelTraduzioni e una volta completata l'animazione viene richiamato OpenPapiro(), visto in precedenza, per permettere di visualizzare il papiro.

```
public void BackInPapiro() //viene chiamata dai tasti back SOLO da traduzione a papiro
{
    //_panelTraduzioni_CG.blocksRaycasts = false;
    //_panelTraduzioni.GetComponent<CanvasGroup>().interactable = false;
    SetInteractive(_panelTraduzioni_CG, 2,false); //TODO:quando clicchi disattiva immediatamente il pannello
   // Debug.Log("mainpanelmanager backinpapiro riga 167 false false");

    currentExitTween = ManagerPanelTraduzioni.Instance.currentTraduzioneTween;

    if (currentExitTween != null)
    {
        currentExitTween.Kill(); // Interrompi l'animazione corrente se c'è una
    }

    currentExitTween = ManagerPanelTraduzioni.Instance.ExitFade();

    isTraduzione = false;

    currentExitTween.OnComplete(() =>
    {
        _panelTraduzioni_CG.DOFade(0, duration); //qui metti alpha zero

        OpenPapiro();

    });
}
```

Figura 50: BackInPapiro()

Se si vuole tornare invece dalla traduzione alla schermata di Home oppure dal papiro alla schermata di Home bisogna cliccare il tasto Home. Questo richiama la funzione BackInSubCategory(). In base al punto in cui si è viene eseguito un pezzo di codice o un altro. Se si vuole passare dalla traduzione alla Home si considera il codice mostrato in figura.

```
if (isTraduzione) //se stai cliccando back quando vedi la traduzione
{
    currentExitTween = ManagerPanelTraduzioni.Instance.currentTraduzioneTween;

    if (currentExitTween != null)
    {
        currentExitTween.Kill(); // Interrompi l'animazione corrente se c'è una
    }

    //chiara introduce questo al posto di quello di simona
    SetInteractive(_panelTraduzioni_CG, 2,false);
    //Debug.Log("mainpanelmanager backinsc isTraduzione riga 203 false false");

    currentExitTween = ManagerPanelTraduzioni.Instance.ExitFade();

    isTraduzione = false;

    //modifiche di chiara: erano accesi
    //Debug.Log("mainpanelmanager backingsc isTraduzione riga 238 cose spente da mettere a false");
    //MainPanelManager.Instance._panelTraduzioni.GetComponent<CanvasGroup>().blocksRaycasts = false;
    //_panelTraduzioni.GetComponent<CanvasGroup>().interactable = false;

    currentExitTween.OnComplete(() =>
    {
        //chiara: spegne questo che era acceso
        _panelTraduzioni_CG.DOFade(0, duration);
        isPapiro = true;

    });
}
```

Figura 51: if(isTraduzione) all'interno di BackInSubCategory()

In questo if(isTraduzione) viene eseguita l'animazione di dissolvenza grazie a ExitFade() che appartiene a ManagerPanelTraduzioni e vengono disattivati tutti i pannelli.

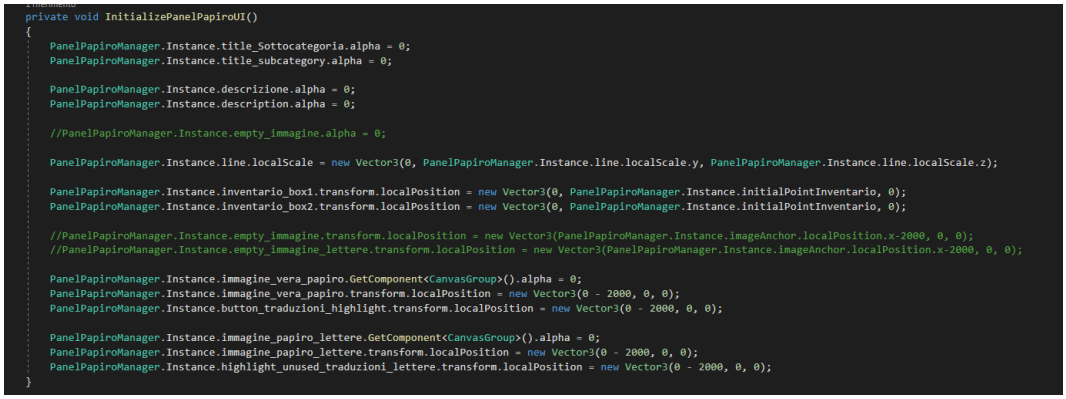
Se invece si è nella schermata del papiro e si vuole tornare indietro, ciò che viene eseguito si trova all'interno dell'if(isPapiro). Anche in questo caso viene fatta una dissolvenza grazie a ExitFade() che però appartiene a PanelPapiroManager e viene disattivato il pannello del papiro. Questo codice viene eseguito sia se si clicca "Back" sia che si clicca "Home". Nel primo caso si torna a visualizzare la schermata con i papiri relativi alle sottocategorie, nel secondo caso invece si torna nella schermata con lo scriba e le categorie. La scelta tra uno o l'altro caso viene considerata in script che sono estranei al caso di studio di questa tesi.

```
if (isPapiro) //se stai cliccando back quando vedi il papiro
{
    SetInteractive(_panelPapiro_CG, 2, false);//idem
    currentExitTween = PanelPapiroManager.Instance.currentTween;

    if (currentExitTween != null)
    {
        currentExitTween.Kill(); // Interrompi l'animazione corrente se c'è una
    }

    SetInteractive(PanelPapiroManager.Instance.PortaFrecce, 2, false); //1 o 2? meglio 2
    SetInteractive(_panelPapiro_CG, 2, false);//idem

    currentExitTween = PanelPapiroManager.Instance.ExitFade();

    currentExitTween.OnComplete(() =>
    {
        //Tutorial.instance.SetDepth(1);
        //chiara: spegne questo che era acceso
        _panelPapiro_CG.DOFade(0, duration).OnComplete(() =>
        {
            PanelPapiroManager.Instance.SfondoBloccaClic.raycastTarget = false;
            SetInteractive(_panelPapiro_CG, 2, false);//idem

        });

        //isPapiro = true;
        isPapiro = false;
    });


}
```

Figura 52: if(isPapiro) all'interno di BackInSubCategory()

Gli ultimi due metodi presenti nello script sono InitializePanelPapiroUI() e InitializePanelTraduzioniUI().

Questi metodi vengono richiamati nel metodo Start() e permettono di inizializzare i parametri. Come detto in precedenza nel momento in cui viene chiamato il metodo, ci si trova nella schermata con lo scriba, quindi ciò che si trova nei vari pannelli deve essere nascosti. Tutti i valori impostati in questi due metodi infatti hanno come alpha 0. In più tutto ciò che grazie alle animazioni si sposta, deve essere impostato correttamente e infatti questi vengono inizializzati con la transform.localPosition nel loro punto di partenza.



```
private void InitializePanelPapiroUI()
{
    PanelPapiroManager.Instance.title_Sottocategoria.alpha = 0;
    PanelPapiroManager.Instance.title_subcategory.alpha = 0;

    PanelPapiroManager.Instance.descrizione.alpha = 0;
    PanelPapiroManager.Instance.description.alpha = 0;

    //PanelPapiroManager.Instance.empty_immagine.alpha = 0;

    PanelPapiroManager.Instance.line.localScale = new Vector3(0, PanelPapiroManager.Instance.line.localScale.y, PanelPapiroManager.Instance.line.localScale.z);

    PanelPapiroManager.Instance.inventario_box1.transform.localPosition = new Vector3(0, PanelPapiroManager.Instance.initialPointInventario, 0);
    PanelPapiroManager.Instance.inventario_box2.transform.localPosition = new Vector3(0, PanelPapiroManager.Instance.initialPointInventario, 0);

    //PanelPapiroManager.Instance.empty_immagine.transform.localPosition = new Vector3(PanelPapiroManager.Instance.imageAnchor.localPosition.x-2000, 0, 0);
    //PanelPapiroManager.Instance.empty_immagine_lettere.transform.localPosition = new Vector3(PanelPapiroManager.Instance.imageAnchor.localPosition.x-2000, 0, 0);

    PanelPapiroManager.Instance.immagine_vera_papiro.GetComponent<CanvasGroup>().alpha = 0;
    PanelPapiroManager.Instance.immagine_vera_papiro.transform.localPosition = new Vector3(0 - 2000, 0, 0);
    PanelPapiroManager.Instance.button_traduzioni_highlight.transform.localPosition = new Vector3(0 - 2000, 0, 0);

    PanelPapiroManager.Instance.immagine_papiro_lettere.GetComponent<CanvasGroup>().alpha = 0;
    PanelPapiroManager.Instance.immagine_papiro_lettere.transform.localPosition = new Vector3(0 - 2000, 0, 0);
    PanelPapiroManager.Instance.highlight_unused_traduzioni_lettere.transform.localPosition = new Vector3(0 - 2000, 0, 0);
}
```

Figura 53: InitializePanelPapiroUI()

```
private void InitializePanelTraduzioniUI()
{
    ManagerPanelTraduzioni.Instance.title_Sottocategoria.alpha = 0;
    ManagerPanelTraduzioni.Instance.title_subcategory.alpha = 0;

    ManagerPanelTraduzioni.Instance.descrizione.alpha = 0;
    ManagerPanelTraduzioni.Instance.description.alpha = 0;

    //ManagerPanelTraduzioni.Instance.empty_immagine.alpha = 0;

    ManagerPanelTraduzioni.Instance.line.localScale = new Vector3(0, PanelPapiroManager.Instance.line.localScale.y, PanelPapiroManager.Instance.line.localScale.z);

    ManagerPanelTraduzioni.Instance.immagine_vera_con_shadow_traduzione.GetComponent<CanvasGroup>().alpha = 0;
    ManagerPanelTraduzioni.Instance.immagine_vera_con_shadow_traduzione.transform.localPosition = new Vector3(0 - 2000, 0, 0);
    ManagerPanelTraduzioni.Instance.highlight_unused_traduzione.transform.localPosition = new Vector3(0 - 2000, 0, 0);

    //ManagerPanelTraduzioni.Instance.empty_immagine.transform.localPosition = new Vector3(ManagerPanelTraduzioni.Instance.imageAnchor.localPosition.x - 2000, 0, 0);
}
```

Figura 54: InitializePanelTraduzioniUI()

4.3 PANELPAPIROMANAGER.JS

Il PanelPapiroManager.js è lo script che controlla il pannello del papiro. Viene richiamato quando nel MainPanelManager è eseguita la funzione OpenPapiro().

La funzione SetPapiro() permette di impostare le informazioni necessarie nel posto corretto.

In title_Sottocategoria e title_subcategory vengono inseriti i nomi delle sottocategorie in italiano e in inglese. In descrizione e description vengono inseriti i testi delle descrizioni dei papiri in italiano e in inglese. Il testo in inglese ha qualche riga vuota in più per permettere la lettura fino all'ultima riga quando il testo scorre. Infatti al fondo dello spazio dedicato al testo è presente un gradiente che permette di nascondere in modo non troppo netto ciò che va oltre a quello spazio e senza le righe vuote questo nasconderebbe la parte finale del testo

inglese. In _catInventario viene inserito in numero di inventario, in _scrittura e _writing il tipo di scrittura in lingua italiana e in lingua inglese. Queste sono tutte variabili temporanee in quanto se viene scelto di cambiare testo anche le variabili devono cambiare.

```
public void SetPapiro() //mettere le informazioni nel posto giusto nel pannello dentro openpapiro
{
    SubCategory temp = ButtonActions.selectedSubcategory;

    title_Sottocategoria.text = temp.subCategoryTextITA;
    title_subcategory.text = temp.subCategoryTextENG;
    descrizione.text = temp.textInfoITA;
    description.text = temp.textInfoENG + " \n \n \n \n \n";

    _catInventario.text = temp.catalogo;
    _scrittura.text = temp.tipo;
    _writing.text = temp.lanaguage;

    PositionTexts();
    PositionInventario();

}
```

Figura 55: Funzione SetPapiro()

I parametri che qui vengono settati, vengono posizionati in modo corretto grazie alle funzioni PositionText() e PositionInventario().

All'interno di questa funzione viene calcolata l'altezza del testo in italiano e in inglese. Il testo in italiano viene posizionato in una posizione data dal vettore (107, 0-30), mentre il testo in inglese viene posizionato al di sotto del testo in italiano con un offset rispetto a quest'ultimo di 10.

Per definire se un testo ha la possibilità di essere scrollato o meno viene calcolata l'altezza di un *empty* che racchiude i due testi. Se questa altezza supera una soglia allora la *scrollbar* sarà visibile e quindi il testo scrollabile. Altrimenti la *scrollbar* non sarà visibile e il testo non sarà scrollabile.

```
private void PositionTexts()
{
    //Ottieni la larghezza preferita del testo italiano e inglese
    textHeightITA = descrizione.preferredHeight;
    textHeightENG = description.preferredHeight;

    // Trova la larghezza massima tra i due testi
    float maxHeight = Mathf.Max(textHeightITA, textHeightENG);

    descrizione.rectTransform.pivot = new Vector2(0.5f, 1);
    descrizione.rectTransform.sizeDelta = new Vector2(descrizione.rectTransform.sizeDelta.x, textHeightITA);
    descrizione.rectTransform.anchoredPosition =
        new Vector2(107, 0 - 30); //30 è offset per fare animazione in animatepage()

    description.rectTransform.pivot = new Vector2(0.5f, 1);
    description.rectTransform.sizeDelta = new Vector2(description.rectTransform.sizeDelta.x, textHeightENG);
    description.rectTransform.anchoredPosition = new Vector2(107, 0 - textHeightITA - 10 - 30);

    empty_testi_ita_eng_papiro.rectTransform.pivot = new Vector2(0.5f, 1);
    empty_testi_ita_eng_papiro.rectTransform.sizeDelta =
        new Vector2(empty_testi_ita_eng_papiro.rectTransform.sizeDelta.x, textHeightITA + textHeightENG);

    //Debug.Log("size delta y: " + empty_testi.rectTransform.sizeDelta.y);

    if (empty_testi_ita_eng_papiro.rectTransform.sizeDelta.y - 25 > 136.66) //controlla se mettere la scrollbar visibile o meno
    {
        scrollbar.GetComponent<CanvasGroup>().alpha = 1;
        slider.handleRect.GetComponent<Image>().color = Color.white;
    }
    else
    {
        scrollbar.GetComponent<CanvasGroup>().alpha = 0;
        slider.handleRect.GetComponent<Image>().color = Color.clear;
    }
}
```

Figura 56: Funzione PositionTexts()

Per quanto riguarda invece PositionInventario(), è presente un valore che salva l'altezza dell'inventario. Infatti alcuni testi hanno più numeri di inventario e questo porta ad avere un testo lungo tanto da essere scritto su più righe. Se questo accade allora la posizione sarà diversa rispetto a quando non accade. Il controllo per posizionare questo parametro è presente quindi in questa funzione.

```
private void PositionInventario()
{
    float heightInventario = _catInventario.preferredHeight;
    //Debug.Log("altezza inventario: " + heightInventario);

    if (heightInventario == 10.31f)
        _catInventario.rectTransform.anchoredPosition = new Vector2(350f, (-152 - 165) / 2);
    else _catInventario.rectTransform.anchoredPosition = new Vector2(350f, -152);
}
```

Figura 57: Funzione PositionInventario()

In OpenPapiro() viene anche richiamata la funzione SetImmaginePapiro(). Questa funzione permette di inserire correttamente l'immagine del papiro di riferimento.

Per prima cosa viene fatto un controllo sul tipo di immagine che si vuole inserire. Infatti, come è stato sottolineato in precedenza, le lettere hanno regole diverse

riguardo allo scorrimento dell'immagine, e quindi è conveniente trattarle separatamente, inserendo nella funzione un controllo.

Se l'utente ha scelto di visualizzare le lettere allora il pannello dell'immagine che verrà visualizzato sarà quello corrispondente alle lettere, e infatti viene impostato un *alpha* a 1 e un *raycast* a *true*, mentre il pannello delle altre immagini non viene reso visibile e non viene reso interagibile. In più è mostrata la shadow delle lettere e non quella orizzontale delle altre immagini. L'immagine che viene presa come quella da visualizzare segue il path delle lettere e ha come dimensioni la larghezza quella standard mentre l'altezza è impostata in proporzione alla larghezza.

```
if (ButtonActions.selectedSubcategory.subCategoryTextITA == "Lettere")
{
    panel_immagine_scrollArea_lettere.alpha = 1;
    panel_immagine_scrollArea_lettere.blocksRaycasts = true;
    Panel_immagine_scrollArea_immagini_papiro.alpha = 0;
    shadowHorizontal.alpha = 0;
    Panel_immagine_scrollArea_immagini_papiro.blocksRaycasts = false;

    imagePath_lettere = "Papiri/Testi Biografici/Slider_TestiBiografici";
    traduzioniPath_lettere = "Papiri/Testi Biografici/Slider_Traduzioni";

    immagine_papiro_lettere.GetComponent<RectTransform>().sizeDelta =
        new Vector2(fixedWidth,
            fixedWidth * Resources.Load<Sprite>(imagePath_lettere).rect.height /
            Resources.Load<Sprite>(imagePath_lettere).rect.width);
    highlight_unused_traduzioni_lettere.GetComponent<RectTransform>().sizeDelta =
        new Vector2(fixedWidth,
            fixedWidth * Resources.Load<Sprite>(imagePath_lettere).rect.height /
            Resources.Load<Sprite>(imagePath_lettere).rect.width);

    empty_papiro_e_highlight_lettere.GetComponent<RectTransform>().sizeDelta =
        new Vector2(fixedWidth,
            fixedWidth * Resources.Load<Sprite>(imagePath_lettere).rect.height /
            Resources.Load<Sprite>(imagePath_lettere).rect.width + 450);

    shadowLettere.alpha = 1;

}
```

Figura 58: Funzione SetImmaginePapiro() - dettaglio su lettere

Se invece l'utente sceglie una qualsiasi altra sottocategoria diversa dalle lettere il codice che viene eseguito è il seguente.

```
else
{
    panel_immagine_scrollArea_lettere.alpha = 0;
    panel_immagine_scrollArea_lettere.blocksRaycasts = false;
    Panel_immagine_scrollArea_immagini_papiro.alpha = 1;
    shadowHorizontal.alpha = 1;
    Panel_immagine_scrollArea_immagini_papiro.blocksRaycasts = true;

    shadowLettere.alpha = 0;
    string textID = ButtonActions.selectedSubcategory.egyptianTextID;

    string safeCatInventario = _catInventario.text.Replace("/", "_"); // Sostituisci "/" con "_"
    //se voglio piu simboli da sostituire aggiungo .repale(...).replace(...)
    imagePath = "Papiri/" + ButtonActions.selectedCategory.categoryNameITA + "/" + safeCatInventario +
                " papiro";
    traduzioniPath = "Papiri/" + ButtonActions.selectedCategory.categoryNameITA + "/" + textID + " Traduzioni";

    immagine_vera_papiro.GetComponent<RectTransform>().sizeDelta =
        new Vector2(
            fixedHeight * Resources.Load<Sprite>(imagePath).rect.width /
            Resources.Load<Sprite>(imagePath).rect.height, fixedHeight);
    button_traduzioni_highlight.GetComponent<RectTransform>().sizeDelta =
        new Vector2(
            fixedHeight * Resources.Load<Sprite>(imagePath).rect.width /
            Resources.Load<Sprite>(imagePath).rect.height, fixedHeight);

    empty_papiro_e_highlight.GetComponent<RectTransform>().sizeDelta =
        new Vector2(
            fixedHeight * Resources.Load<Sprite>(imagePath).rect.width /
            Resources.Load<Sprite>(imagePath).rect.height + 450, fixedHeight);

}
```

Figura 59: Funzione SetImmaginePapiro() - dettaglio su immagini

È molto simile a quello delle lettere. La differenza è che il pannello visibile e interagibile non è più quello delle lettere ma quello delle immagini e l'immagine ha come dimensione fissa l'altezza e ciò che cambia in proporzione è la larghezza.

Una volta che il controllo sul tipo di immagine è stato fatto, bisogna controllare se quel tipo di sottocategoria ha o meno la traduzione. Il controllo viene fatto tramite un *if*. Se esiste un *path* che rimanda alla traduzione corrispondente allora il bottone per la traduzione diventa interagibile e quindi cliccabile per poter visualizzare la schermata corrispondente alla traduzione, altrimenti non viene attivato.

```
if (Resources.Load<Sprite>(traduzioniPath) != null || Resources.Load<Sprite>(traduzioniPath_lettere) != null)
{
    button_traduzioni_highlight.gameObject.SetActive(true);
    button_traduzioni_highlight.GetComponent<CanvasGroup>().alpha = 0;
    button_traduzioni_highlight.GetComponent<Button>().interactable = true;
    button_traduzioni_highlight.GetComponent<Image>().sprite = Resources.Load<Sprite>(traduzioniPath);
    button_traduzioni_highlight.GetComponent<Button>().enabled = true;

    highlight_unused_traduzioni_lettere.gameObject.SetActive(true);
    highlight_unused_traduzioni_lettere.GetComponent<CanvasGroup>().alpha = 0;
    highlight_unused_traduzioni_lettere.GetComponent<Button>().interactable = true;
    highlight_unused_traduzioni_lettere.GetComponent<Image>().sprite = Resources.Load<Sprite>(traduzioniPath_lettere);
    highlight_unused_traduzioni_lettere.GetComponent<Button>().enabled = true;

    SetBoxColliderSizeToImage();
}
else
{
    button_traduzioni_highlight.gameObject.SetActive(false);
    highlight_unused_traduzioni_lettere.gameObject.SetActive(false);
}

SetBoxColliderSizeToImage();
```

Figura 60: Funzione SetImmaginePapiro() - dettaglio su bottone di traduzione

All'interno di questo if viene chiamata la funzione SetBoxColliderSizeToImage() che permette di impostare la dimensione del box collider del bottone di traduzione grande quanto l'immagine corrispondente in modo che in qualsiasi punto dell'immagine l'utente clicchi, si possa passare alla traduzione.

```
void SetBoxColliderSizeToImage()
{
    Vector2 imageSize = immagine_vera_papiro.rectTransform.sizeDelta; //Ottieni la dimensione dell'immagine

    BoxCollider2D boxCollider = button_traduzioni_highlight.GetComponent<BoxCollider2D>(); //Ottieni o aggiungi il BoxCollider2D

    if (boxCollider == null) boxCollider = button_traduzioni_highlight.gameObject.AddComponent<BoxCollider2D>();

    boxCollider.size = imageSize; //Imposta la dimensione del BoxCollider2D per corrispondere all'immagine
}
```

Figura 61: Funzione SetBoxColliderSizeToImage()

L'ultima funzione che viene chiamata da OpenPapiro() è AnimatePage(), una funzione che permette di animare la schermata. Le animazioni vengono fatte usando la libreria di DOTween, creando una sequenza di comparse e spostamenti.

Per poter eseguire alcune delle animazioni in modo quasi simultaneo viene utilizzato Insert, cosicché quando ancora non è finita un'animazione, quella successiva può già cominciare. Altre animazioni invece sono eseguite in simultanea tra loro grazie a Join, e altre ancora vengono eseguite una volta che l'animazione precedente viene conclusa tramite AppendCallBack.

Ogni animazione ha una durata che viene impostata come ultimo parametro delle funzioni DOFade, DOLocalMoveX, DOLocalMoveY, DOScaleX.

```
public Tween AnimatePage()
{
    //Debug.Log("Sono entrato in AnimatePage()");

    Sequence sequenceAnimatePagePapiro = DOTween.Sequence();
    sequenceAnimatePagePapiro.Restart();

    sequenceAnimatePagePapiro
        .Append(MainPanelManager.Instance._panelPapiro.GetComponent<CanvasGroup>().DOFade(1, duration))
        .AppendCallback(() => { })

        .Join(shadowHorizontal.DOFade(1f, duration))
        .Join(shadowTesti.DOFade(1f, duration))//devo avere le shadow dei testi prima che entrino le immagini e i testi
        .Join(shadowLettere.DOFade(1f,duration))

        .Insert(0.3f, line.DOScaleX(1, duration)) //entra riga arancione
        .Insert(0.5f, title_Sottocategoria.DOFade(1, duration - 0.3f)) //entra titolo
        .Insert(0.7f, title_subcategory.DOFade(1, duration - 0.3f)) //entra titolo eng

        .Insert(0.7f, background_testi_scrollArea_papiro.DOFade(1,duration -0.3f)) //entra bg scrollArea testi itaeng
        .Insert(0.9f, descrizione.DOFade(1, duration)) //entra descrizione
        .Insert(0.9f, descrizione.transform.DOLocalMoveY(0, duration - 0.3f)) //entra in movimento il testo

        .Insert(1f, scrollContenitoreTesto.GetComponent<CanvasGroup>().DOFade(1, duration))

        .Insert(1.1f, description.DOFade(1, duration))
        .Insert(1.1f, description.transform.DOLocalMoveY(0 - textHeightITA - 10, duration - 0.3f))

        .Insert(1f, inventario.DOFade(1, duration - 0.3f))
        .Insert(1.2f, inventario_box1.transform.DOLocalMoveY(endPointInventario, duration - 0.3f))
        .Insert(1.4f, inventario_box2.transform.DOLocalMoveY(endPointInventario - 40, duration - 0.3f))

        .Join(IsLettere()? panel_immagine_scrollArea_lettere.DOFade(1f, duration): Panel_immagine_scrollArea_immagini_papiro.DOFade(1f, duration))
```

Figura 62: Funzione AnimatePage()

```
        .Insert(1.6f, immagine_vera_papiro.transform.DOLocalMoveX(0, duration))
        .Insert(1.6f, button_traduzioni_highlight.transform.DOLocalMoveX(0, duration))
        .Insert(1.9f, immagine_vera_papiro.GetComponent<CanvasGroup>().DOFade(1, duration))

        .Insert(1.6f, empty_papiro_e_highlight_lettere.transform.DOLocalMoveX(130, duration))
        .Insert(1.6f, immagine_papiro_lettere.transform.DOLocalMoveX(0, duration))
        .Insert(1.6f, highlight_unused_traduzioni_lettere.transform.DOLocalMoveX(0, duration))
        .Insert(1.9f, immagine_papiro_lettere.GetComponent<CanvasGroup>().DOFade(1, duration))
        .Insert(1.6f, portaFrecce.DOFade(1, duration))
        .AppendCallback(() =>
        {
            MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelPapiro_CG, 1, true);

            MainPanelManager.Instance.SetInteractive(portaFrecce, 2, true);
            //va bene 1 perche aveva appena inserito 1 animazione con DoFade  poche righe sopra

            if (MainPanelManager.Instance._panelPapiro_CG.alpha == 0 &&
                MainPanelManager.Instance._panelPapiro_CG.interactable &&
                MainPanelManager.Instance._panelPapiro_CG.blocksRaycasts) //caso: 0,true,true: demoliscilo.
            {
                //previeni questa diamine di eccezione
                MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelPapiro_CG, 0, false);

            }
        });


    sequenceAnimatePagePapiro.Play().OnComplete(() => currentTween.Kill());


    return sequenceAnimatePagePapiro;
}
```

Figura 63: Funzione AnimatePage()

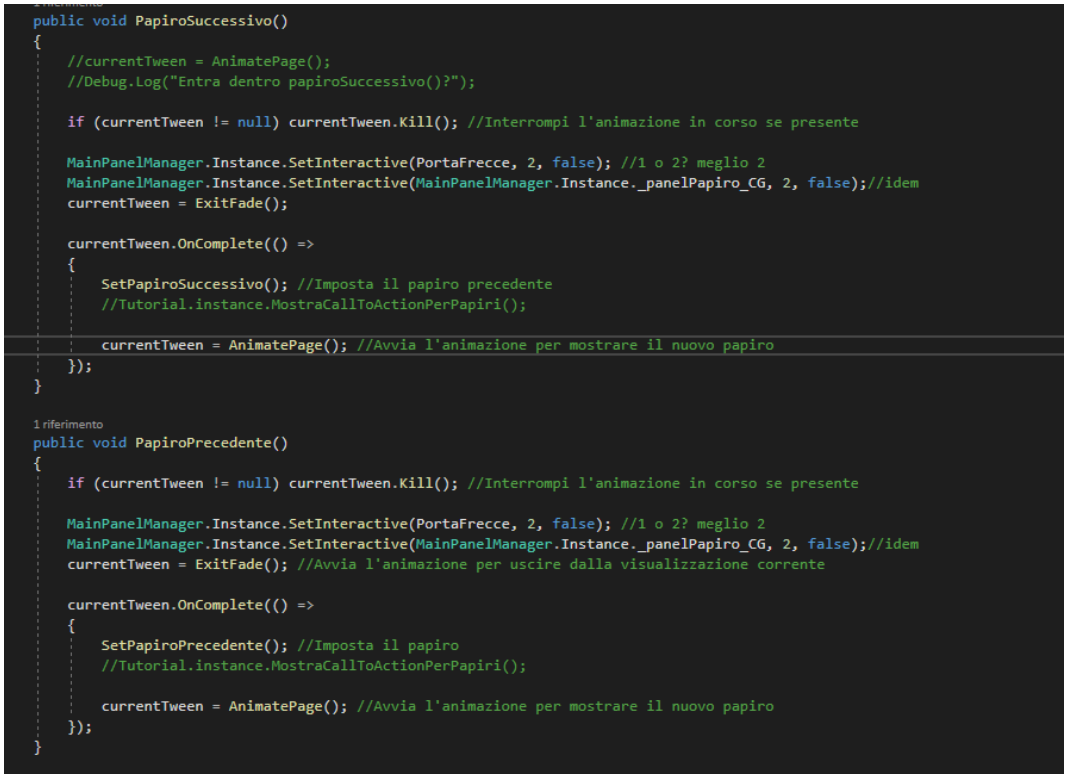
Se dalla schermata che si sta visualizzando si vuole passare alla schermata di traduzione oppure ad un papiro diverso oppure ancora tornare indietro per visualizzare la schermata di scelta della categoria o sottocategoria, viene eseguita la funzione ExitFade(). In questa funzione tutto ciò che è stato animato in AnimatePage(), viene nuovamente animato ma al contrario. Vengono eseguite

delle dissolvenze e gli spostamenti sono importanti per far tornare gli elementi nel loro punto di partenza. Così come in AnimatePage(), anche in ExitFade() viene eseguito tutto con la libreria di DOTween. L'animazione di uscita deve essere contemporanea per tutti gli elementi interessati ed è per questo motivo che viene utilizzato il Join.

```
public Tween ExitFade()
{
    Sequence sequenceExitFadePapiro = DOTween.Sequence();
    sequenceExitFadePapiro.Restart();

    sequenceExitFadePapiro.Append(title_Sottocategoria.DOFade(0, duration))
        .Join(title_subcategory.DOFade(0, duration))
        .Join(portaFrecce.DOFade(0, duration))
        .Join(line.DOScaleX(0, duration / 2))

        .Join(descrizione.DOFade(0, duration))
        .Join(description.DOFade(0, duration))
        .Join(empty_testi_ita_eng_papiro.transform.DOLocalMoveY(68.32909f, duration))

        //sparizione testi, scrollArea, scrollbar

        .Join(background_testi_scrollArea_papiro.DOFade(0,duration))
        .Join(scrollContenitoreTesto.GetComponent<CanvasGroup>().DOFade(0, duration / 4))

        .Join(inventario.DOFade(0, duration))
        .Join(inventario_box1.transform.DOLocalMoveY(initialPointInventario, duration))
        .Join(inventario_box2.transform.DOLocalMoveY(initialPointInventario, duration))

        .Join(immagine_vera_papiro.GetComponent<CanvasGroup>().DOFade(0, duration))
        .Join(empty_papiro_e_highlight.transform.DOLocalMoveX(-257.3409f, duration))
        .Join(immagine_vera_papiro.transform.DOLocalMoveX(0 - 2000, duration))
        .Join(button_traduzioni_highlight.transform.DOLocalMoveX(0 - 2000, duration))

        .Join(immagine_papiro_lettere.GetComponent<CanvasGroup>().DOFade(0, duration)) //lo sposti e poi lo muovi assieme all unused highlight
        .Join(immagine_papiro_lettere.transform.DOLocalMoveX(0 - 2000, duration))
        .Join(highlight_unused_traduzioni_lettere.transform.DOLocalMoveX(0 - 2000, duration))
        .Append(immagine_papiro_lettere.transform.DOLocalMoveY(0, duration))
        .Join(highlight_unused_traduzioni_lettere.transform.DOLocalMoveY(0, duration))
        .Join(empty_papiro_e_highlight_lettere.transform.DOLocalMoveY(0, duration))

        //TODO: SECONDO ME MANCA L ANIMAZIONE IN USCITA DELLE BANDE NERE
        .Join(shadowHorizontal.DOFade(0f, duration))
        .Join(shadowLettere.DOFade(0f, duration))
        .Join(shadowTesti.DOFade(0f,duration))

        .Join(Panel_immagine_scrollArea_immagini_papiro.DOFade(0f, duration));


    return sequenceExitFadePapiro;
}
```

Figura 64: Funzione ExitFade()

Per passare da un papiro a un altro bisogna toccare sulle frecce disposte ai lati dello schermo. Ognuna delle due frecce richiama una funzione specifica. Se si tocca la freccia destra verrà eseguita la funzione PapiroSuccessivo(), se invece si tocca la freccia sinistra verrà eseguita PapiroPrecedente(). Queste funzioni sono state implementate in PanelPapiroManager.js ma vengono eseguite grazie al codice che è stato scritto all'interno di ArrowButtonsPapiroManager.js, ma questo verrà analizzato in seguito.



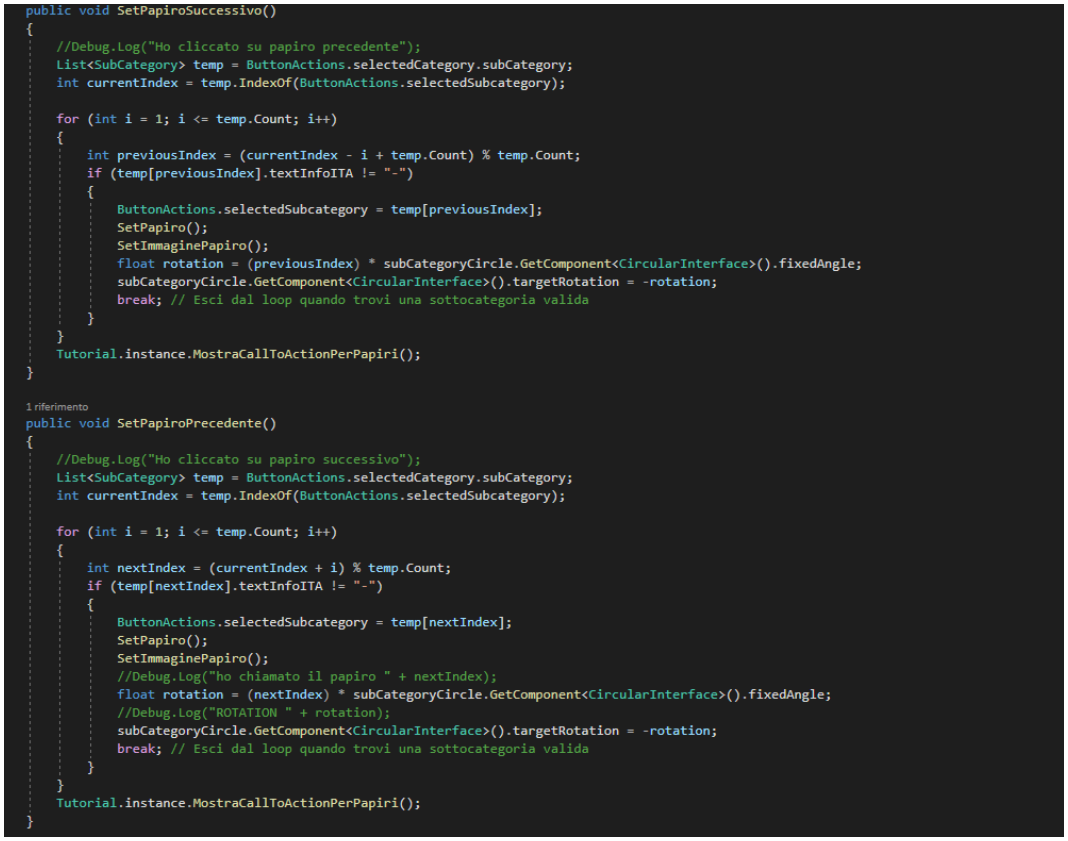
```
public void PapiroSuccessivo()
{
    //currentTween = AnimatePage();
    //Debug.Log("Entra dentro papiroSuccessivo()?");

    if (currentTween != null) currentTween.Kill(); //Interrompi l'animazione in corso se presente

    MainPanelManager.Instance.SetInteractive(PortaFrecce, 2, false); //1 o 2? meglio 2
    MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelPapiro_CG, 2, false);//idem
    currentTween = ExitFade();

    currentTween.OnComplete(() =>
    {
        SetPapiroSuccessivo(); //Imposta il papiro precedente
        //Tutorial.instance.MostraCallToActionPerPapiri();

        currentTween = AnimatePage(); //Avvia l'animazione per mostrare il nuovo papiro
    });
}

1 riferimento
public void PapiroPrecedente()
{
    if (currentTween != null) currentTween.Kill(); //Interrompi l'animazione in corso se presente

    MainPanelManager.Instance.SetInteractive(PortaFrecce, 2, false); //1 o 2? meglio 2
    MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelPapiro_CG, 2, false);//idem
    currentTween = ExitFade(); //Avvia l'animazione per uscire dalla visualizzazione corrente

    currentTween.OnComplete(() =>
    {
        SetPapiroPrecedente(); //Imposta il papiro
        //Tutorial.instance.MostraCallToActionPerPapiri();

        currentTween = AnimatePage(); //Avvia l'animazione per mostrare il nuovo papiro
    });
}
```

Figura 65: Funzione PapiroSuccessivo() e PapiroPrecedente()

In entrambi i casi quello che succede è interrompere l'animazione se è in corso, disattivare le frecce e il pannello in modo da non creare problemi di accavallamenti di animazioni, richiamare la funzione ExitFade(), eseguire una funzione per settare i parametri corretti che vedremo in seguito e infine animare la pagina con AnimatePage() così da poter visualizzare il contenuto corretto.

Nelle funzioni che vengono richiamate, SetPapiroPrecedente() e SetPapiroSuccessivo() viene aggiornato l'indice tale per cui si passa all'elemento successivo o precedente del database. Vengono richiamate poi le funzioni di SetPapiro() e SetImmaginePapiro() in modo da impostare correttamente le immagini e le posizioni dei vari testi senza prendere in considerazione le righe del database che hanno come textInfoITA il contenuto "-".

Infine vengono aggiornate le posizioni delle sottocategorie nella sottoscena Home e richiamata la funzione MostraCallToActionPerPapiri() che si trova in Tutoria.js, ma questi non verranno analizzati in quanto non soggetti di studio della tesi.

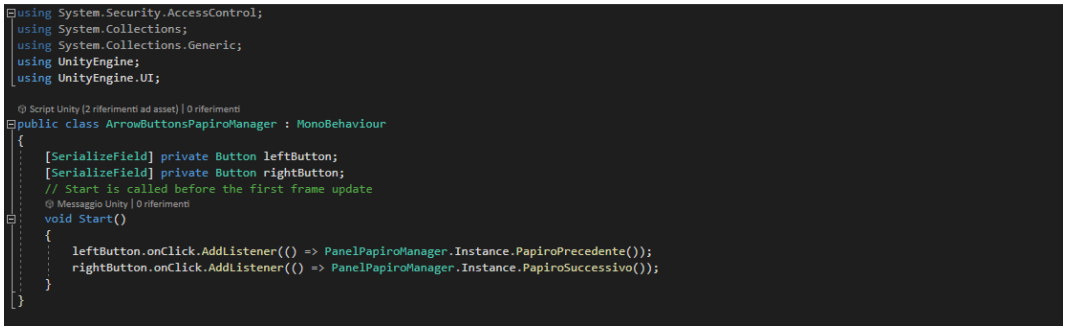


```
public void SetPapiroSuccessivo()
{
    //Debug.Log("Ho cliccato su papiro precedente");
    List<SubCategory> temp = ButtonActions.selectedCategory.subCategory;
    int currentIndex = temp.IndexOf(ButtonActions.selectedSubcategory);

    for (int i = 1; i <= temp.Count; i++)
    {
        int previousIndex = (currentIndex - i + temp.Count) % temp.Count;
        if (temp[previousIndex].textInfoITA != "-")
        {
            ButtonActions.selectedSubcategory = temp[previousIndex];
            SetPapiro();
            SetImmaginePapiro();
            float rotation = (previousIndex) * subCategoryCircle.GetComponent<CircularInterface>().fixedAngle;
            subCategoryCircle.GetComponent<CircularInterface>().targetRotation = -rotation;
            break; // Esci dal loop quando trovi una sottocategoria valida
        }
    }
    Tutorial.instance.MostraCallToActionPerPapiri();
}

1 riferimento
public void SetPapiroPrecedente()
{
    //Debug.Log("Ho cliccato su papiro successivo");
    List<SubCategory> temp = ButtonActions.selectedCategory.subCategory;
    int currentIndex = temp.IndexOf(ButtonActions.selectedSubcategory);

    for (int i = 1; i <= temp.Count; i++)
    {
        int nextIndex = (currentIndex + i) % temp.Count;
        if (temp[nextIndex].textInfoITA != "-")
        {
            ButtonActions.selectedSubcategory = temp[nextIndex];
            SetPapiro();
            SetImmaginePapiro();
            //Debug.Log("ho chiamato il papiro " + nextIndex);
            float rotation = (nextIndex) * subCategoryCircle.GetComponent<CircularInterface>().fixedAngle;
            //Debug.Log("ROTATION " + rotation);
            subCategoryCircle.GetComponent<CircularInterface>().targetRotation = -rotation;
            break; // Esci dal loop quando trovi una sottocategoria valida
        }
    }
    Tutorial.instance.MostraCallToActionPerPapiri();
}
```

Figura 66: Funzione SetPapiroSuccessivo() e SetPapiroPrecedente()

4.4 ARROWBUTTONSPAPIROMANAGER.JS

Il codice di ArroButtonsPapiroManager.js è molto snello e semplice. Le uniche due variabili che vengono inserite sono leftButton e rightButton, che in Unity vengono associate rispettivamente alle frecce sinistra e destra. Una volta che ci si trova nella scena Home, viene eseguita la funzione Start() in cui viene attivato un AddListener() per entrambe le frecce. Questo metodo serve per associare l'evento del click sulla freccia alla funzione PapiroPrecedente() o PapiroSuccessivo(), che sono state analizzate precedentemente, in base a quale delle due frecce viene toccata.

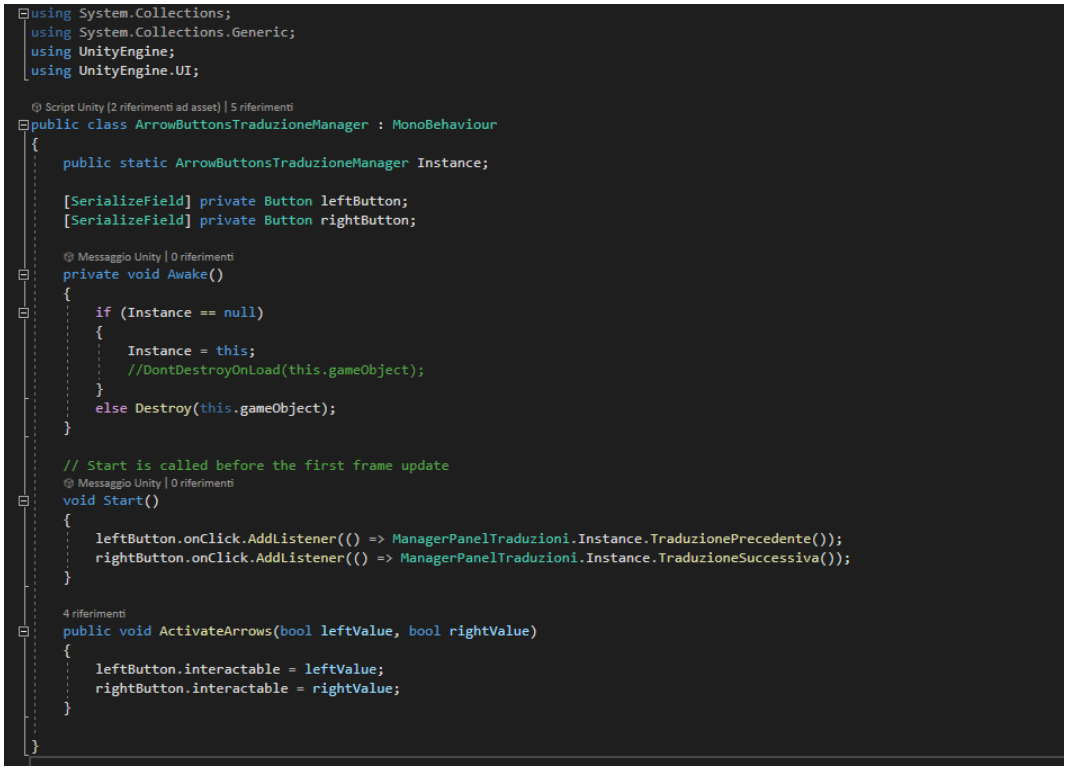


```
using System.Security.AccessControl;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

@ Script Unity (2 riferimenti ad asset) | 0 riferimenti
public class ArrowButtonsPapiroManager : MonoBehaviour
{
    [SerializeField] private Button leftButton;
    [SerializeField] private Button rightButton;
    // Start is called before the first frame update
    @ Messaggio Unity | 0 riferimenti
    void Start()
    {
        leftButton.onClick.AddListener(() => PanelPapiroManager.Instance.PapiroPrecedente());
        rightButton.onClick.AddListener(() => PanelPapiroManager.Instance.PapiroSuccessivo());
    }
}
```

Figura 67: ArrowButtonsPapiroManager.js

4.5 ARROWBUTTONSTRADUZIONEMANAGER.JS

All'interno di ArrowButtonsTraduzioneManager.js quello che succede è molto simile a ciò che è presente in ArrowButtonsPapiroManager.js. Infatti sono presenti come variabili due bottoni per le frecce destra e sinistra che hanno come metodo un AddListener() a cui viene passata la funzione TraduzionePrecedente() o TraduzioneSuccessiva() in base a quale delle due frecce viene toccata. Ciò che è in più è la funzione ActivateArrows(bool leftValue, bool rightValue) in quanto non tutti i papiri hanno più traduzioni. Questa funzione permette di attivare o disattivare le frecce in base al numero di traduzioni presenti. Se il papiro ha una sola traduzione allora entrambi i valori di ActivateArrows saranno false, se sono presenti più traduzioni, le frecce verranno attivate o disattivate in base al numero di traduzione che l'utente sta visualizzando.



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

Script Unity (2 riferimenti ad asset) | 5 riferimenti
public class ArrowButtonsTraduzioneManager : MonoBehaviour
{
    public static ArrowButtonsTraduzioneManager Instance;

    [SerializeField] private Button leftButton;
    [SerializeField] private Button rightButton;

    Messaggio Unity | 0 riferimenti
    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            //DontDestroyOnLoad(this.gameObject);
        }
        else Destroy(this.gameObject);
    }

    // Start is called before the first frame update
    Messaggio Unity | 0 riferimenti
    void Start()
    {
        leftButton.onClick.AddListener(() => ManagerPanelTraduzioni.Instance.TraduzionePrecedente());
        rightButton.onClick.AddListener(() => ManagerPanelTraduzioni.Instance.TraduzioneSuccessiva());
    }

    4 riferimenti
    public void ActivateArrows(bool leftValue, bool rightValue)
    {
        leftButton.interactable = leftValue;
        rightButton.interactable = rightValue;
    }

}
```

Figura 68: ArrowButtonsTraduzioneManager.js

4.6 MAINPANELTRADUZIONE.JS

MainPanelTraduzione.js ha una struttura molto simile a PanelPapiroManager.js. Le funzioni presenti all'interno di questo script vengono eseguite nel momento in cui l'utente clicca sul papiro di cui ha interesse a visualizzare la traduzione. Viene richiamata la funzione OpenTraduzioni() all'interno di MainPanelManager.js ed eseguite la funzione CheckIndex(), SetTraduzione() e AnimatePageTraduzione().

La funzione CheckIndex() serve per capire quante traduzioni ha il papiro che l'utente ha intenzione di approfondire. È presente un intero chiamato numeroTrad che conta il numero di traduzioni e con questo viene calcolato l'indice massimo di traduzioni (l'intero maxIndex).

Viene in seguito fatto un controllo con una sequenza di if:

- Se tradIndex (che è una variabile inizializzata a 0 all'interno della funzione OpenTraduzioni() di MainPanelManager.js in quanto la prima traduzione da visualizzare deve avere come indice 0) è 0 ed è uguale a maxIndex allora le frecce non verranno visualizzate. Questo succede se è presente solo una traduzione.
- Se tradIndex è 0 ed è minore di maxIndex allora vuol dire che sono presenti più di una traduzione e per visualizzare le successive occorre attivare la freccia destra.
- Nel momento in cui è stata visualizzata la prima traduzione ma sono presenti più di due traduzioni allora le frecce devono essere entrambe visualizzabili in modo da poter andare avanti e indietro nella visualizzazione ed è quello che succede nel controllo tradIndex maggiore di 0 e tradIndex minore di maxIndex.
- Se tradIndex è uguale a maxIndex vuol dire che l'utente è arrivato a visualizzare l'ultima traduzione disponibile. Quello che accade è che la freccia di destra viene disattivata mentre rimane visibile solo la freccia sinistra.

```
public void CheckIndex()
{
    List<string> tempITA = ButtonActions.selectedSubcategory.translationInfoITA;
    List<string> tempENG = ButtonActions.selectedSubcategory.translationInfoENG;

    int numeroTrad = tempITA.Count;
    maxIndex = numeroTrad - 1;

    //VERIFICARE A QUALE NUMERO DI TRADUZIONE SONO E COSA FARE
    if (tradIndex == 0 && tradIndex == maxIndex)
    {
        //Debug.Log("ho solo una traduzione");
        //devo spegnere tutte le frecce
        ArrowButtonsTraduzioneManager.Instance.ActivateArrows(false, false);
    }
    else if (tradIndex == 0 && tradIndex < maxIndex)
    {
        //Debug.Log("ho più di una traduzione e sono nella prima");
        //TODO: devo spegnere la freccia sinistra e accendo freccia destra
        ArrowButtonsTraduzioneManager.Instance.ActivateArrows(false, true);
    }
    else if (tradIndex > 0 && tradIndex < maxIndex)
    {
        //Debug.Log("ho più di una traduzione e non sono nella prima nè nell'ultima");
        //TODO: accendo tutte e due le frecce
        ArrowButtonsTraduzioneManager.Instance.ActivateArrows(true, true);
    }
    else if (tradIndex == maxIndex)
    {
        //Debug.Log("ho più di una traduzione e sono nell'ultima");
        //TODO: spengo freccia destra e accendo solo freccia sinistra
        ArrowButtonsTraduzioneManager.Instance.ActivateArrows(true, false);
    }
    else
    {
        // Debug.Log("CASO NON ANALIZZATO");
    }
}
```

Figura 69: Funzione CheckIndex()

La funzione SetTraduzione() si comporta allo stesso modo della funzione SetPapiro() di PanelPapiroManager.js.

```
3 riferimenti
public void SetTraduzione() //cambia la traduzione
{
    SubCategory temp = ButtonActions.selectedSubcategory;

    //la prima traduzione che vedo è quella con l'indice 0
    descrizione.text = temp.translationInfoITA[tradIndex];
    description.text = temp.translationInfoENG[tradIndex] + " \n \n \n \n \n";

    title_Sottocategoria.text = "TRADUZIONE";
    title_subcategory.text = "TRANSLATION";

    SetImmagineTraduzione();
    PositionTexts();
}
```

Figura 70: Funzione SetTraduzione()

Le funzioni che vengono richiamate, SetImmagineTraduzione() e PositionText() sono anch'esse uguali nella struttura alle corrispondenti funzioni in PanelPapiroManager.js.

```
public void SetImmagineTraduzione()
{
    float fixedHeight = 1089.818f;
    float fixedWidth = 1937.455f / 2;

    if (title_Sottocategoria.text == "Lettere")
    {
        imagePath = "Papiri/Testi Biografici/Slider_TestiBiografici";
        traduzioniPath = "Papiri/Testi Biografici/Slider_Traduzioni";

        immagine_vera_con_shadow_traduzione.GetComponent<RectTransform>().sizeDelta =
            new Vector2(fixedWidth, fixedWidth * Resources.Load<Sprite>(imagePath).rect.height / Resources.Load<Sprite>(imagePath).rect.width);
        highlight_unused_traduzione.GetComponent<RectTransform>().sizeDelta =
            new Vector2(fixedWidth, fixedWidth * Resources.Load<Sprite>(imagePath).rect.height / Resources.Load<Sprite>(imagePath).rect.width);
    }
    else
    {
        string textID = ButtonActions.selectedSubcategory.egyptianTextID;
        string safeCatInventario = ButtonActions.selectedSubcategory.catalogo.Replace("/", "_");

        //se voglio piu simboli da sostituire aggiungo .repale(...).replace(...)
        imagePath = "Papiri/" + ButtonActions.selectedCategory.categoryNameITA + "/" + textID + " trad" + tradIndex;
        if(maxIndex == 0)
        {
            traduzioniPath = "Papiri/" + ButtonActions.selectedCategory.categoryNameITA + "/" + textID + " Traduzioni";
        }
        else if(maxIndex != 0)
        {
            traduzioniPath = "Papiri/" + ButtonActions.selectedCategory.categoryNameITA + "/" + textID + " Traduzioni" + tradIndex;
        }

        immagine_vera_con_shadow_traduzione.GetComponent<RectTransform>().sizeDelta =
            new Vector2(fixedHeight * Resources.Load<Sprite>(imagePath).rect.width / Resources.Load<Sprite>(imagePath).rect.height, fixedHeight);
        highlight_unused_traduzione.GetComponent<RectTransform>().sizeDelta =
            new Vector2(fixedHeight * Resources.Load<Sprite>(imagePath).rect.width / Resources.Load<Sprite>(imagePath).rect.height, fixedHeight);

        empty_immagine.GetComponent<RectTransform>().sizeDelta =
            new Vector2(fixedHeight * Resources.Load<Sprite>(imagePath).rect.width / Resources.Load<Sprite>(imagePath).rect.height + 450, fixedHeight);

    }

    immagine_vera_con_shadow_traduzione.sprite = Resources.Load<Sprite>(imagePath);
    highlight_unused_traduzione.sprite = Resources.Load<Sprite>(traduzioniPath);

    Color alphaTrad = highlight_unused_traduzione.color;
    //Imposta il valore alpha desiderato
    alphaTrad.a = 0f;
    //Applica il colore modificato all'immagine dell'handle
    highlight_unused_traduzione.color = alphaTrad;
}
```

Figura 71: Funzione SetImmagineTraduzione()

```
private void PositionTexts()
{
    // Ottieni la larghezza preferita del testo italiano e inglese
    textHeightITA = descrizione.preferredHeight;
    textHeightENG = description.preferredHeight;

    //Debug.Log(descrizione.bounds);

    // Trova la larghezza massima tra i due testi
    float maxHeight = Mathf.Max(textHeightITA, textHeightENG);

    descrizione.rectTransform.pivot = new Vector2(0.5f, 1);
    descrizione.rectTransform.sizeDelta = new Vector2(descrizione.rectTransform.sizeDelta.x, textHeightITA);
    descrizione.rectTransform.anchoredPosition = new Vector2(105, 0 - 30); //30 è offset per fare animazione in animatepage()

    description.rectTransform.pivot = new Vector2(0.5f, 1);
    description.rectTransform.sizeDelta = new Vector2(description.rectTransform.sizeDelta.x, textHeightENG);
    description.rectTransform.anchoredPosition = new Vector2(105, 0 - textHeightITA - 10 - 30);

    empty_testi_ita_eng_traduzione.rectTransform.pivot = new Vector2(0.5f, 1);
    empty_testi_ita_eng_traduzione.rectTransform.sizeDelta = new Vector2(empty_testi_ita_eng_traduzione.rectTransform.sizeDelta.x, textHeightITA + textHeightENG);


    //Debug.Log("anchored position di descrizione è: " + (descrizione.rectTransform.anchoredPosition.y));
    //Debug.Log("grandezza di descrizione è: " + textHeightITA);

    empty_testi_ita_eng_traduzione.rectTransform.pivot = new Vector2(0.5f, 1);
    empty_testi_ita_eng_traduzione.rectTransform.sizeDelta = new Vector2(empty_testi_ita_eng_traduzione.rectTransform.sizeDelta.x, textHeightITA + textHeightENG);

    //Debug.Log("grandezza y: " + empty_testi_ita_eng_traduzione.bounds.size.y);
    //Debug.Log("size delta y: " + empty_testi_ita_eng_traduzione.rectTransform.sizeDelta.y);

    Image scrollbarHandle = scrollbar.transform.Find("Sliding Area/Handle")?.GetComponent<Image>();
    if (empty_testi_ita_eng_traduzione.rectTransform.sizeDelta.y-25 > 190.66) //controlla se mettere la scrollbar visibile o meno
    {
        scrollbar.GetComponent<CanvasGroup>().alpha = 1;
        slider.handleRect.GetComponent<Image>().color = Color.white;
    }
    else
    {
        scrollbar.GetComponent<CanvasGroup>().alpha = 0;
        slider.handleRect.GetComponent<Image>().color = Color.clear;
    }
}
```

Figura 72: Funzione PositionText()

L’ultima funzione richiamata da OpenTraduzioni() è AnimatePageTraduzione(), anch’essa simile come struttura ad AnimatePage() per la schermata dei papiri.

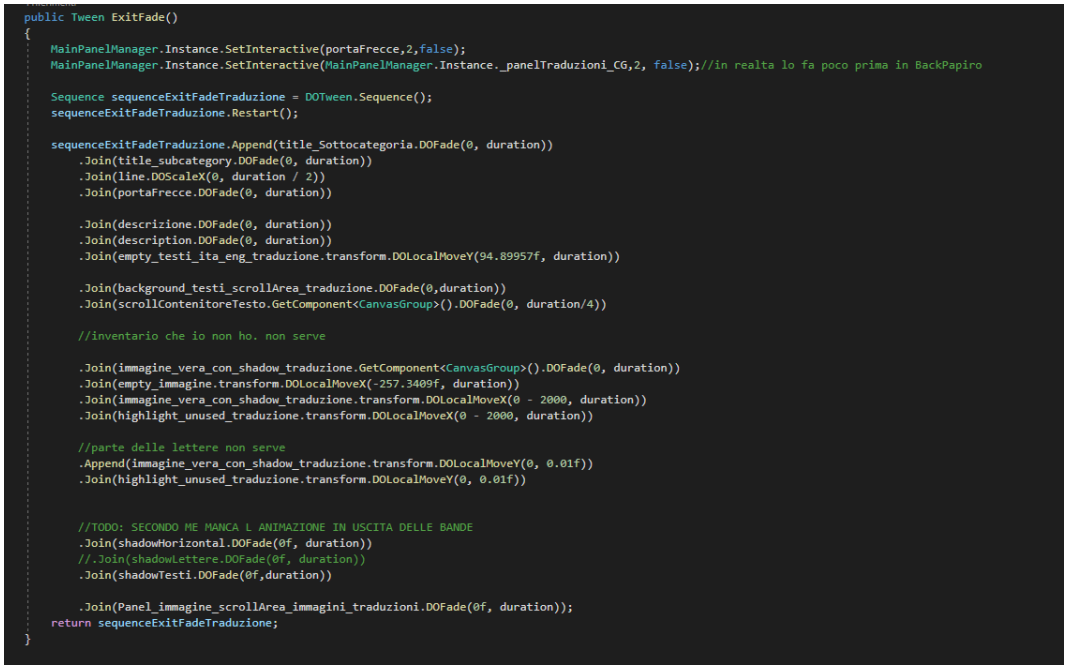
```
public Tween AnimatePageTraduzione()
{
    //Debug.Log("posizione x di imageanchor: " + imageAnchor.localPosition.x);
    Sequence sequenceAnimatePageTraduzione = DOTween.Sequence();

    sequenceAnimatePageTraduzione.Append(MainPanelManager.Instance._panelTraduzioni.GetComponent<CanvasGroup>().DOFade(1, duration))
        .AppendCallback(() => { })

        .Join(shadowHorizontal.DOFade(1f, duration))
        .Join(shadowTesti.DOFade(1f, duration))//devo avere le shadow dei testi prima che entrino le immagini e i testi

        .Insert(0.3f, line.DOScaleX(1, duration))
        .Insert(0.5f, title_Sottocategoria.DOFade(1, duration - 0.3f))
        .Insert(0.7f, title_subcategory.DOFade(1, duration - 0.3f))

        .Insert(0.7f, background_testi_scrollArea_traduzione.DOFade(1,duration -0.3f)) //entra bg scrollArea testi itaeng
        .Insert(0.9f, descrizione.DOFade(1, duration))
        .Insert(0.9f, descrizione.transform.DOLocalMoveY(0, duration - 0.3f))

        .Insert(1f, scrollContenitoreTesto.GetComponent<CanvasGroup>().DOFade(1, duration))

        .Insert(1.1f, description.DOFade(1, duration))
        .Insert(1.1f, description.transform.DOLocalMoveY(0 - textHeightITA - 10, duration - 0.3f))

      //inventario qui non c'e

        .Join(Panel_immagine_scrollArea_immagini_traduzioni.DOFade(1f, duration)) //aggiunto da chiara

        .Insert(1.1f, immagine_vera_con_shadow_traduzione.transform.DOLocalMoveX(0, duration))
        //qui non ho button perche non devo cliccare niente
        .Insert(1.1f, highlight_unused_traduzione.transform.DOLocalMoveX(0, duration))
        .Insert(1.4f, immagine_vera_con_shadow_traduzione.GetComponent<CanvasGroup>().DOFade(1, duration))
        .Insert(1.1f, portaFrecce.DOFade(1, duration)).
        AppendCallback(()=> {
            MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelTraduzioni_CG,1,true);
            MainPanelManager.Instance.SetInteractive(portaFrecce,2,true);
            //homesso2 ma in prec: va bene 1 perche aveva appena inserito 1 animazione con DoFade  poche righe sopra
        });

    if (MainPanelManager.Instance._panelTraduzioni_CG.alpha == 0 &&
        MainPanelManager.Instance._panelTraduzioni_CG.interactable &&
        MainPanelManager.Instance._panelTraduzioni_CG.blocksRaycasts) //caso: 0,true,true: demoliscilo.
    {
        //previeni questa diamine di eccezione
        MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelTraduzioni_CG, 0, false);
    }
```

Figura 73: Funzione AnimatePageTraduzione()

```
    sequenceAnimatePageTraduzione.Play().OnComplete(() => currentTraduzioneTween.Kill());

    if (MainPanelManager.Instance._panelTraduzioni_CG.alpha == 0 &&
        MainPanelManager.Instance._panelTraduzioni_CG.interactable &&
        MainPanelManager.Instance._panelTraduzioni_CG.blocksRaycasts) //caso: 0,true,true: demoliscilo.
    {
        //previeni questa diamine di eccezione
        MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelTraduzioni_CG, 0, false);
    }

    return sequenceAnimatePageTraduzione;
}
```

Figura 74: Funzione AnimatePageTraduzione()

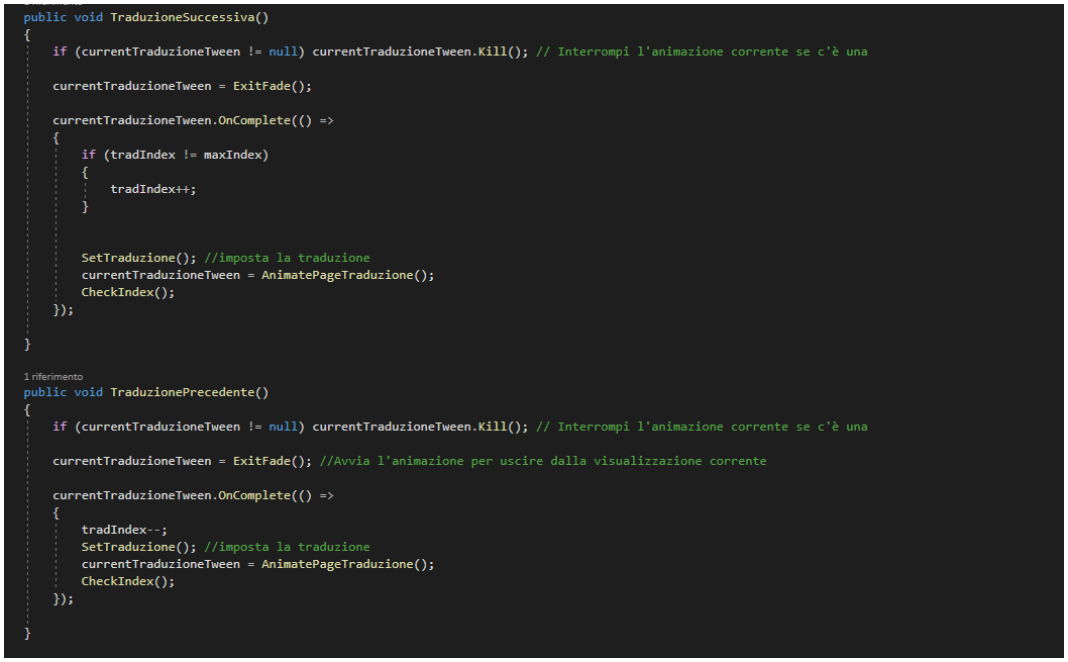
Un’altra funzione analoga a quella vista in precedenza per la schermata dei papiri è ExitFade().



```
public Tween ExitFade()
{
    MainPanelManager.Instance.SetInteractive(portaFrecce,2,false);
    MainPanelManager.Instance.SetInteractive(MainPanelManager.Instance._panelTraduzioni_CG,2, false);//in realta lo fa poco prima in BackPapiro

    Sequence sequenceExitFadeTraduzione = DOTween.Sequence();
    sequenceExitFadeTraduzione.Restart();

    sequenceExitFadeTraduzione.Append(title_Sottocategoria.DOFade(0, duration))
        .Join(title_subcategory.DOFade(0, duration))
        .Join(line.DOScaleX(0, duration / 2))
        .Join(portaFrecce.DOFade(0, duration))

        .Join(descrizione.DOFade(0, duration))
        .Join(description.DOFade(0, duration))
        .Join(empty_testi_ita_eng_traduzione.transform.DOLocalMoveY(94.89957f, duration))

        .Join(background_testi_scrollArea_traduzione.DOFade(0,duration))
        .Join(scrollContenitoreTesto.GetComponent<CanvasGroup>().DOFade(0, duration/4))

        //inventario che io non ho. non serve

        .Join(immagine_vera_con_shadow_traduzione.GetComponent<CanvasGroup>().DOFade(0, duration))
        .Join(empty_immagine.transform.DOLocalMoveX(-257.3409f, duration))
        .Join(immagine_vera_con_shadow_traduzione.transform.DOLocalMoveX(0 - 2000, duration))
        .Join(highlight_unused_traduzione.transform.DOLocalMoveX(0 - 2000, duration))

        //parte delle lettere non serve
        .Append(immagine_vera_con_shadow_traduzione.transform.DOLocalMoveY(0, 0.01f))
        .Join(highlight_unused_traduzione.transform.DOLocalMoveY(0, 0.01f))


        //TODO: SECONDO ME MANCA L ANIMAZIONE IN USCITA DELLE BANDE
        .Join(shadowHorizontal.DOFade(0f, duration))
        //.Join(shadowLettere.DOFade(0f, duration))
        .Join(shadowTesti.DOFade(0f,duration))

        .Join(Panel_immagine_scrollArea_immagini_traduzioni.DOFade(0f, duration));
    return sequenceExitFadeTraduzione;
}
```

Figura 75: Funzione ExitFade()

Se un papiro ha più traduzioni, le funzioni che vengono eseguite se si vuole visualizzare una traduzione successiva o precedente sono TraduzioneSuccessiva() e TraduzionePrecedente(). La struttura è molto simile agli analoghi PapiroSuccessivo() e PappiroPrecedente() di PanelPapiroManager.js, ma in questo caso è presente un controllo in più riguardo alla visualizzazione delle frecce attraverso la funzione CheckIndex() analizzata precedentemente.



```
public void TraduzioneSuccessiva()
{
    if (currentTraduzioneTween != null) currentTraduzioneTween.Kill(); // Interrompi l'animazione corrente se c'è una

    currentTraduzioneTween = ExitFade();

    currentTraduzioneTween.OnComplete(() =>
    {
        if (tradIndex != maxIndex)
        {
            tradIndex++;
        }


        SetTraduzione(); //imposta la traduzione
        currentTraduzioneTween = AnimatePageTraduzione();
        CheckIndex();
    });

}

1 riferimento
public void TraduzionePrecedente()
{
    if (currentTraduzioneTween != null) currentTraduzioneTween.Kill(); // Interrompi l'animazione corrente se c'è una

    currentTraduzioneTween = ExitFade(); //Avvia l'animazione per uscire dalla visualizzazione corrente

    currentTraduzioneTween.OnComplete(() =>
    {
        tradIndex--;
        SetTraduzione(); //imposta la traduzione
        currentTraduzioneTween = AnimatePageTraduzione();
        CheckIndex();
    });

}
```

Figura 76: Funzione TraduzioneSuccessiva() e TraduzionePrecedente()

5. PROFILING DI UN'APPLICAZIONE

In questo capitolo verrà mostrata la profilazione dell'applicazione *Tipi di testi nell'antico Egitto.* I profiler sono degli strumenti utili per identificare i colli di bottiglia della propria applicazione e verificare le prestazioni del codice scritto.

Esistono due metodi per fare profiling della propria applicazione: il profiling basato sui campioni consiste nella raccolta e nell'analisi dei dati. Il sovraccarico di questo tipo di profilazione però è elevato perché è necessario avere alte frequenza perché la profilazione sia accurata [1]. Il profiling della strumentazione prevede un'aggiunta di profile markers che registrano le informazioni utili sul tempo che impiega il codice per essere eseguito [1]. I marker possono anche essere inseriti manualmente utilizzando l'API ProfilerMarker [2].

```
using Unity.Profiling;

public class MySystemClass
{
    static readonly ProfilerMarker s_PreparePerfMarker = new ProfilerMarker("MySystem.Prepare");
    static readonly ProfilerMarker s_SimulatePerfMarker = new ProfilerMarker(ProfilerCategory.Ai, "MySystem.Simulate");

    public void UpdateLogic()
    {
        s_PreparePerfMarker.Begin();
        // ...
        s_PreparePerfMarker.End();

        using (s_SimulatePerfMarker.Auto())
        {
            // ...
        }
    }
}
```

Figura 77: Script per inserire i Marker

5.1 UNITY PROFILER

Unity profiler [3] esegue una profilazione della strumentazione. Misura le prestazioni dell'editor Unity e serve per identificare le aree della propria

applicazione da poter migliorare. Questo strumento influisce sulle prestazioni generali dell'applicazione perché utilizza le stesse risorse di questa quando è in modalità play.

È utile per reperire informazioni necessarie riguardanti le prestazioni di un'applicazione, come CPU, memoria, renderer e audio.

Per poter accedere alla finestra profiler è necessario accedere al menu *Window* > *Analysis* > *Profiler*.

Nella figura viene mostrato Unity profiler dell'applicazione *Tipi di testi nell'antico Egitto*.

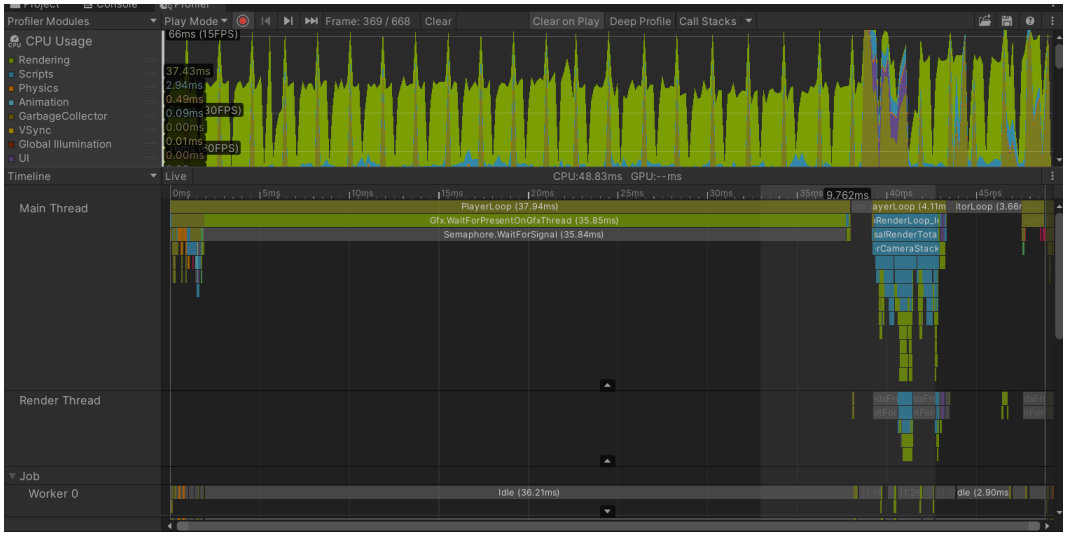


Figura 78: Unity Profiler di *Tipi di testi nell'antico Egitto*

Dalla figura si possono notare tre thread principali: il main thread indica la logica dell'applicazione, ciò che svolgono i vari script ed elabora la scena per passare i propri dati al render thread; il render thread traduce i dati passati dal main thread in graphics API calls che sono indipendenti dalla piattaforma; il job indica ciò che viene eseguito e serve per ridurre il carico di lavoro del main thread.

Nell'applicazione *Tipi di testi nell'antico Egitto*, il metodo che occupa più tempo è Gfx.WaitForPresentOnGfxThread (35.85ms) [4]. Questo indica che il render thread era in attesa che la GPU presentasse il fotogramma. Questo metodo quindi si può definire che sia una "barriera" che impedisce di procedere fino a quando

non viene presentato il fotogramma. Per poter migliorare il tempo di questo metodo è consigliabile ridurre il numero di poligoni dell'oggetto che rappresenta la città di Deir El Medina o ridurre la risoluzione delle texture della città oppure delle immagini dei papiri. Un'altra soluzione che si può adottare è quella di ridurre il numero di chiamate a questo metodo in quanto è possibile che vengano fatte troppe chiamate e un'ottimizzazione è utile per ridurre il tempo di attesa.

5.2 FRAME DEBUGGER

Frame debugger [5] è uno strumento utile per misurare le prestazioni grafiche della propria applicazione, identificare artefatti di rendering e analizza altri problemi. Si può anche utilizzare per osservare come Unity costruisca la scena partendo dagli elementi grafici.

Per poter accedere alla finestra del frame debugger bisogna accedere al menu *Window > Analysis > Frame Debugger*.

È possibile analizzare i passi di render uno ad uno e vedere in che stato di avanzamento si trova il frame in quel momento, in modo tale da conoscere le informazioni riguardanti il comportamento dei materiali, delle luci, degli oggetti.

Mettendo in play l'applicazione, questa partirà e verrà messa in pausa nel momento in cui si cliccherà Enable sul Frame debugger. Si visualizzeranno tutte le chiamate dei vari oggetti del frame corrente e verranno anche segnati dei dettagli aggiuntivi.

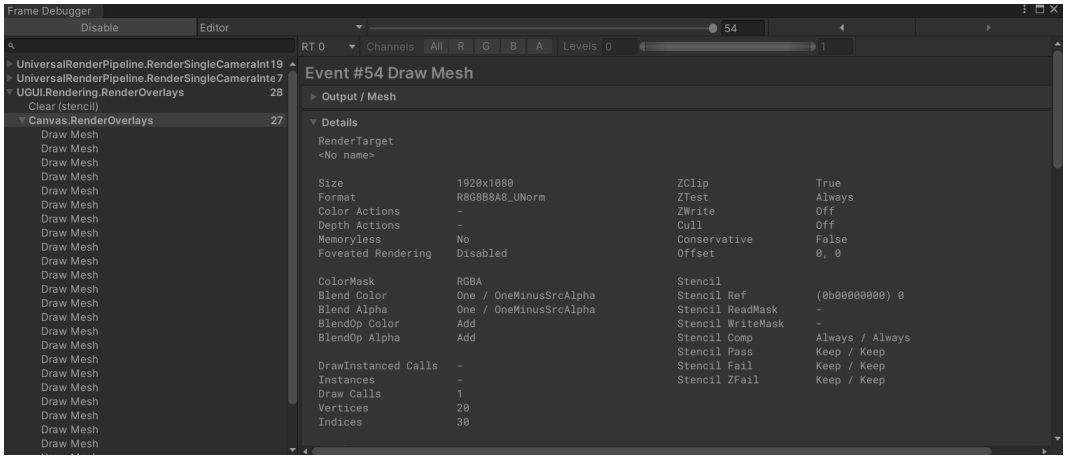


Figura 79: Frame Debugger scena Deir El Medina

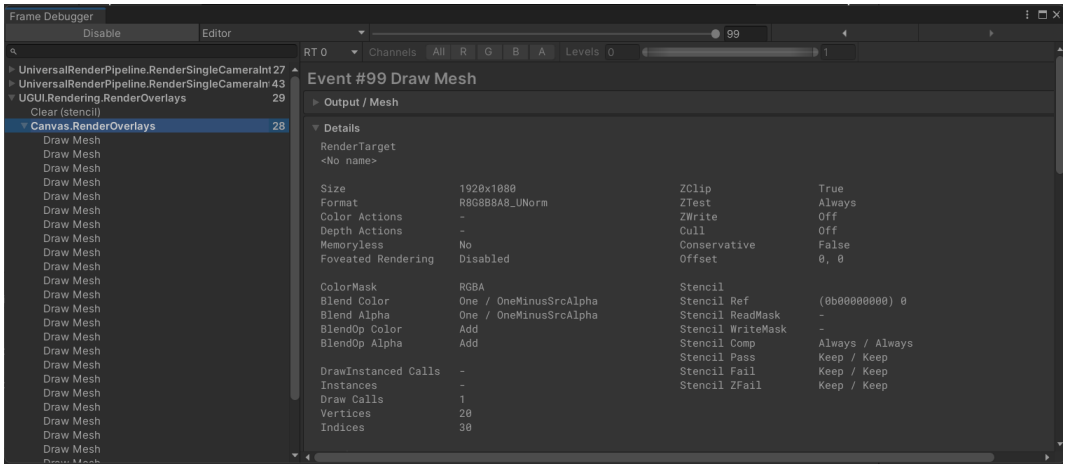


Figura 80: Frame Debugger scena Home - sottoscena Home

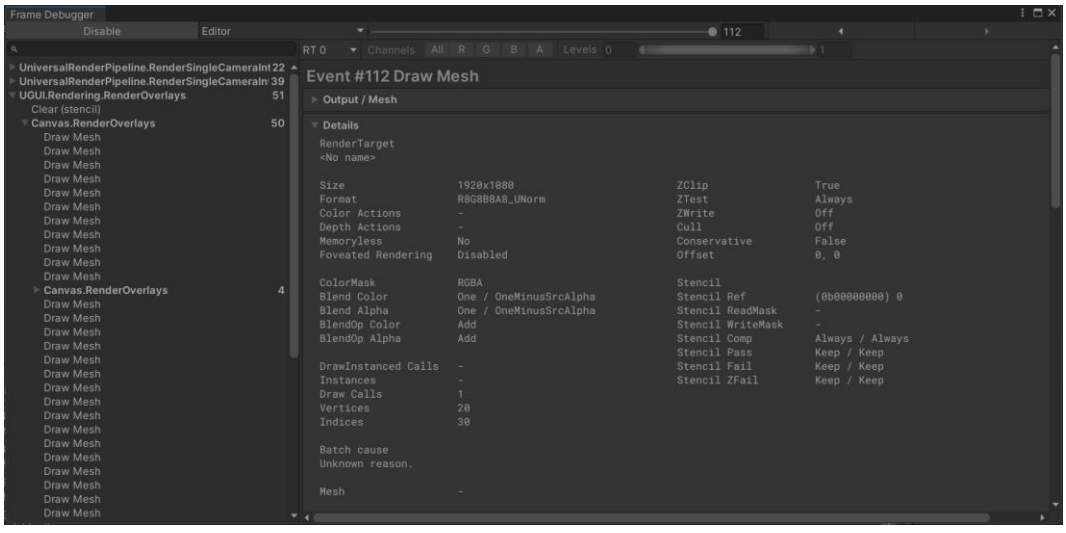


Figura 81: Frame Debugger scena Home - sottoscena Testo

6. OTTIMIZZAZIONE DI UN'APPLICAZIONE

In questo capitolo verranno mostrati vari tipi di ottimizzazione di un'applicazione di Unity e quali tra questi possono essere più utili e come per l'applicazione *Tipi di Testi nell'antico Egitto*. Da un'analisi precedente si riscontra che ciò che occupa maggiormente il main thread di Unity è il caricamento del frame perché le immagini e gli oggetti sono pesanti e portano ad avere un carico di lavoro eccessivo. Su Unity vengono offerte alcune tecniche di ottimizzazione: LOD, Draw call batching, combine meshes, GPU Instancing, Culling, Bake delle luci.

6.1 LOD

Il LOD [6], level of detail, è una tecnica che si fonda sull'ottimizzazione di una mesh in base alla distanza della camera. Renderizzare oggetti con grande dettaglio anche se è distante dalla camera è dispendioso in termini di risorse e di tempo di caricamento delle varie scene. Per ovviare a questo problema si possono create varie versioni della mesh con diverso numero di triangoli che la compongono, in base alla distanza dalla camera. Ciò che viene visualizzato quindi è una versione sempre diversa dell'oggetto rispetto alla posizione della camera.

Per poter utilizzare i LOD serve inserire nel GameObject il componente LOD Group. Questo componente permette di impostare le distanze per cui l'oggetto cambierà la propria mesh e la transizione tra queste può avvenire attraverso un cross-fade che permette di rendere la transizione più morbida. Il componente LOD Group si presenta in questo modo

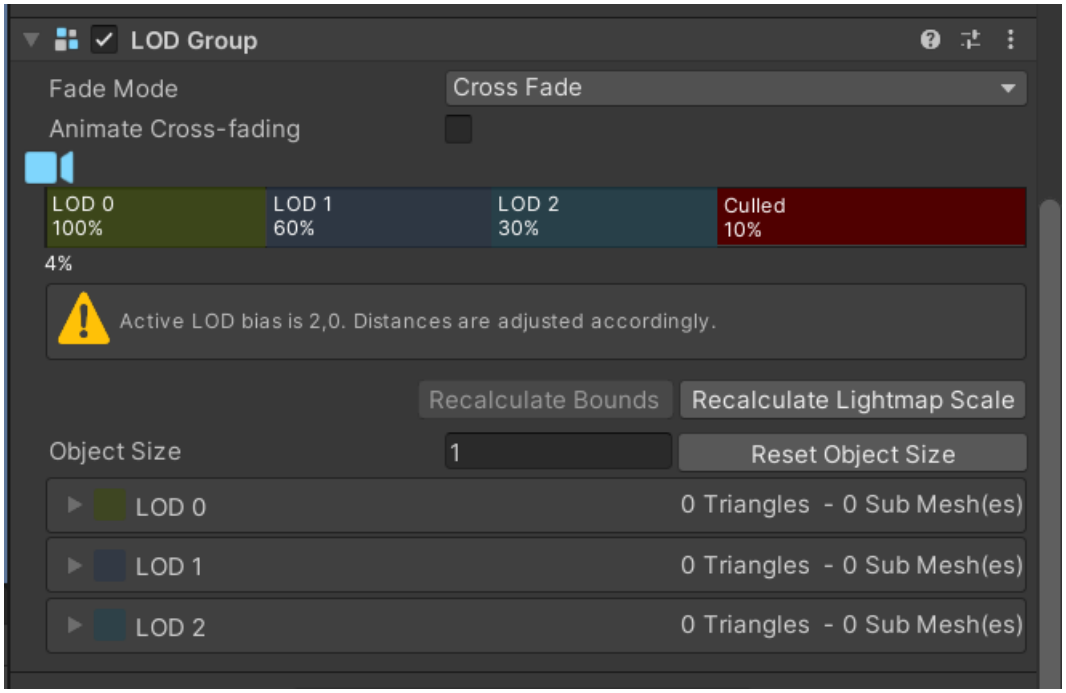


Figura 82: LOD

6.2 DRAW CALL BATCHING

Il draw call batching [7] è un metodo che combina le mesh in modo tale che Unity possa eseguire il rendering in un tempo breve e nel minor numero di draw call. Unity ha due metodi di draw call batching:

- Static batching [8]: Unity combina le mesh degli oggetti statici e li renderizza insieme. In questo modo verranno utilizzate meno draw calls ma ci sarà un maggiore impiego di memoria. Infatti, se diversi oggetti condividono lo stesso materiale, Unity creerà tante copie quanti sono gli oggetti per poi creare un'unica mesh combinata. Per poter attivare il batching static serve attivare la spunta sulla finestra dell'Inspector dell'oggetto.

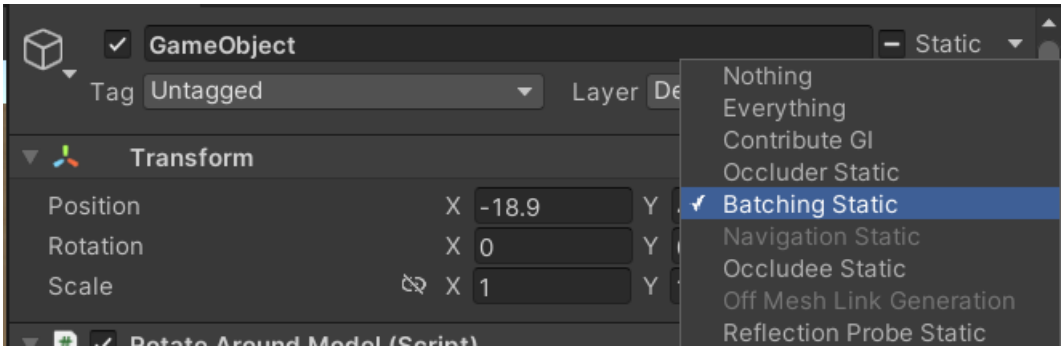


Figura 83: Come impostare static batching

Per poter utilizzare lo static batching è importante che l'oggetto sia attivo, che abbia un numero di vertici maggiore di 0, che la mesh non sia già stata combinata con altre mesh. Ogni batch statico può includere fino a 64000 vertici e se ce ne sono di più allora Unity creerà un altro batch.

- Dynamic batching [9]: Unity raggruppa gli oggetti con caratteristiche simili renderizzando poi una singola draw call. Il dynamic batching utilizza la CPU invece che la GPU. Per poter utilizzare il dynamic batching bisogna andare su *Edit* > *Project settings* > *Player*. Su Other Setting impostare Dynamic Batching. Questo metodo però ha delle limitazioni: non ci può essere raggruppamento dinamico per le mesh con più di 300 vertici; non si può usare il dynamic batching se l'oggetto utilizza diversi materiali; non può essere applicato il dynamic batching se viene utilizzato il multi-pass shader.

6.3 COMBINE MESHES

Un metodo simile allo static batcing è il combine meshes [10], un metodo che permette di creare un'unica mesh con diverse mesh che condividono lo stesso materiale e la stessa texture. Rispetto allo static batching però, questa è una tecnica automatica di Unity.

6.4 GPU INSTANCING

GPU Instancing [11] è un metodo di ottimizzazione delle draw call perché più copie di una mesh con lo stesso materiale vengono renderizzate in una sola draw call. GPU Instancing è disponibile su tutte le piattaforme diverse da WebGL 1.0, e se si volesse usare la GPU Instancing si possono effettuare queste operazioni:

- Utilizzare l'API Ghraphics.RenderMeshInstanced che permette alle mesh di essere di tipo MeshRenderer e utilizza dei parametri specifici per poter disegnare una mesh sullo schermo.

- Rimuovere la compatibilità di SRP Batcher.

Per poter utilizzare la GPU Instancing bisogna selezionare un materiale e nell'Inspector l'opzione abilitare l'opzione "Enable GPU Instancing".

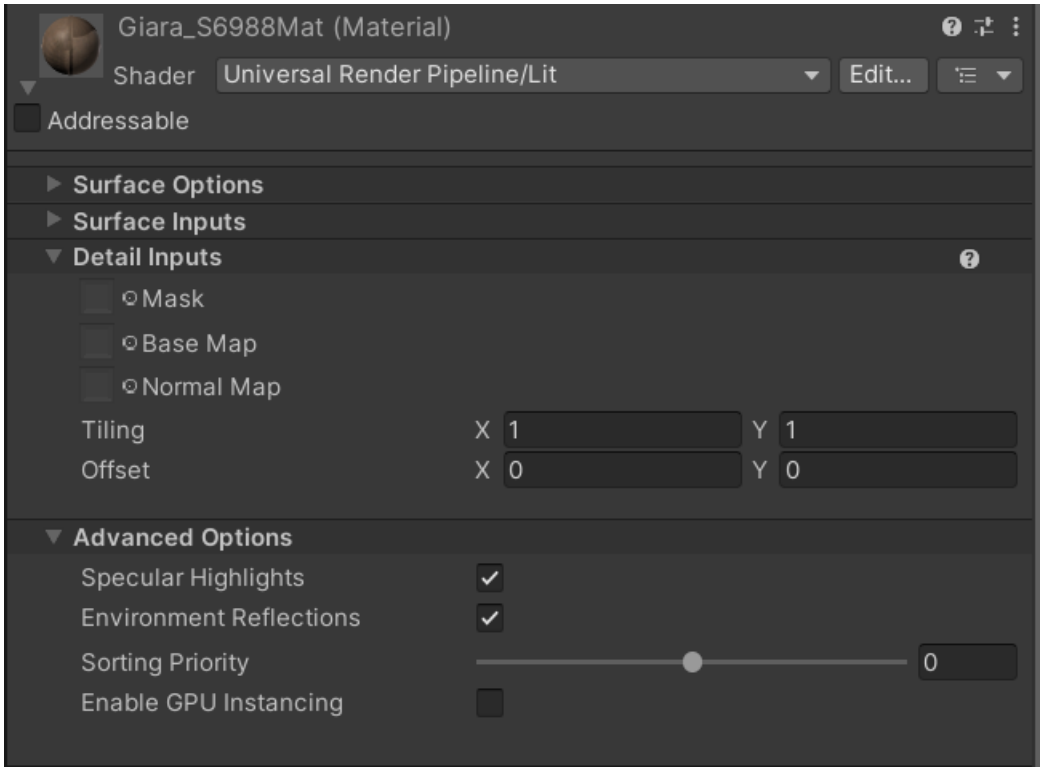


Figura 84: Come impostare GPU Instancing

Questo metodo ha delle implicazioni sulle prestazioni dell'applicazione. Infatti le mesh con un numero di vertici basso non possono essere elaborate efficacemente perché la GPU non riesce a distribuire il lavoro in modo tale da poter usare tutte le proprie risorse. In conclusione quindi la GPU Instancing si utilizza per mesh che hanno più di 256 vertici. Se invece si vuole utilizzare questa tecnica con mesh che hanno pochi vertici si consiglia di creare un buffer che contenga le informazioni della mesh per poterlo utilizzare e disegnare le mesh.

6.5 CULLING

Il culling [12] è una tecnica che permette di renderizzare solo gli oggetti che sono visibili dalla camera. Per ogni frame, la camera analizza la scena e considera solo gli oggetti visibili. L'analisi del frame avviene seguendo questa modalità: la camera esegue il frustrum culling, una tecnica che permette di escludere tutti gli oggetti che sono nascosti dalla vista della camera, quindi ciò che non entra nel

cono visivo della camera non viene disegnato. Il limite di questa tecnica è che non controlla se gli oggetti sono nascosti da altri oggetti. In questo caso è necessario fare uso di un'altra tecnica, l'occlusion culling. Si può attivare questa tecnica tramite un'impostazione nell'Inspector.

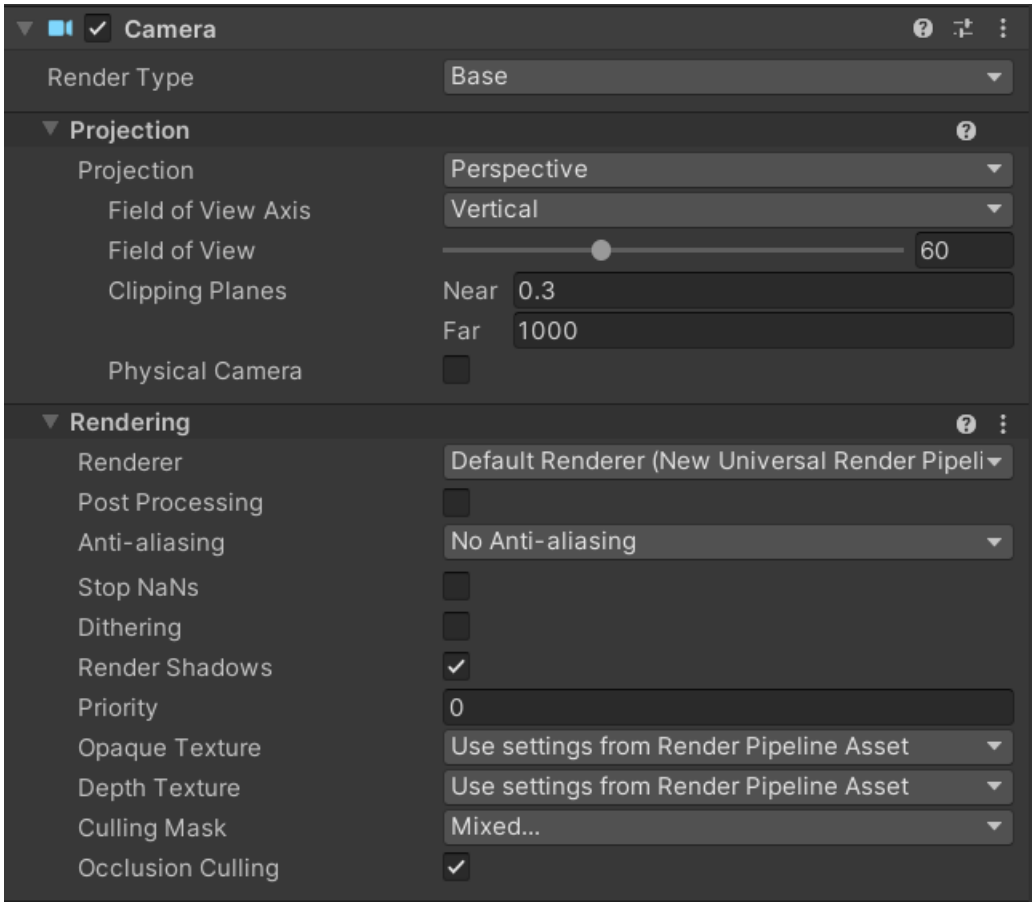


Figura 85: Come impostare Occlusion Culling

Questo procedimento viene usato per far risparmiare tempo sia alla CPU che alla GPU perché vengono evitati calcoli inutili riguardanti il rendering di oggetti che non si vedono dalla camera.

Gli oggetti possono essere di tipo occluder o occludee. Se un oggetto è occluder allora può coprirne un altro, mentre se è occludee può essere coperto da un altro oggetto. Per far sì che un elemento possa nascondere ed essere nascosto è necessario che vengano attivati entrambi i tag.

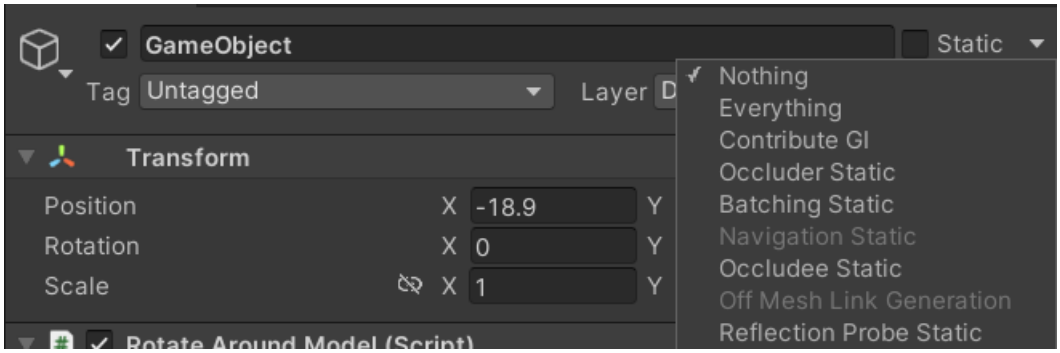


Figura 86: Come impostare un oggetto Occluder o Occludee

Una volta definito che cosa possono essere i vari oggetti si può passare al bake dei dati che consente di generare dei dati per conservarli nella memoria e utilizzarli quando sarà necessario. Una volta che l'applicazione parte, Unity carica i dati salvati in memoria e verifica in ogni istante quali oggetti sono visibili e quali sono nascosti.

6.6 BAKING DELLE LUCI

Con le tecniche viste in precedenza vengono ottimizzati gli oggetti presenti in scena. Per migliorare la propria applicazione si può anche pensare di ottimizzare il carico riguardante l'illuminazione. Utilizzare un'illuminazione realtime in un'applicazione infatti può essere pesante per il PC ed è per questo che viene eseguito il baking delle luci [13]. Unity esegue dei calcoli riguardanti l'illuminazione e salva i risultati in memoria e in fase di esecuzione li riutilizza per poter illuminare correttamente la scena. Questo metodo permette di ridurre i costi di rendering di luci e ombre.

6.7 COME MIGLIORARE TIPI DI TESTI NELL'ANTICO EGITTO

Dopo un'analisi delle tecniche di ottimizzazione si può appurare che in Tipi di testi nell'antico Egitto le tecniche più utili possono essere:

- LOD per quanto riguarda il modello di Deir El Medina. Durante la scena, la camera inquadra da diverse distanze il modello. Ciò vuol dire che è possibile creare diverse mesh con un diverso numero di triangoli e in base alla distanza dalla camera sostituirle.

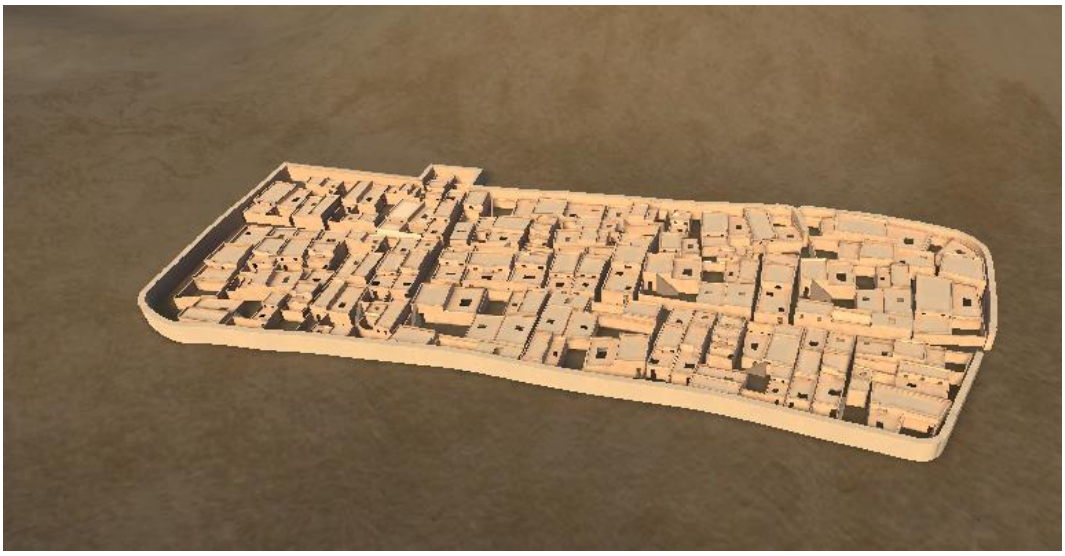


Figura 87: Deir El Medina

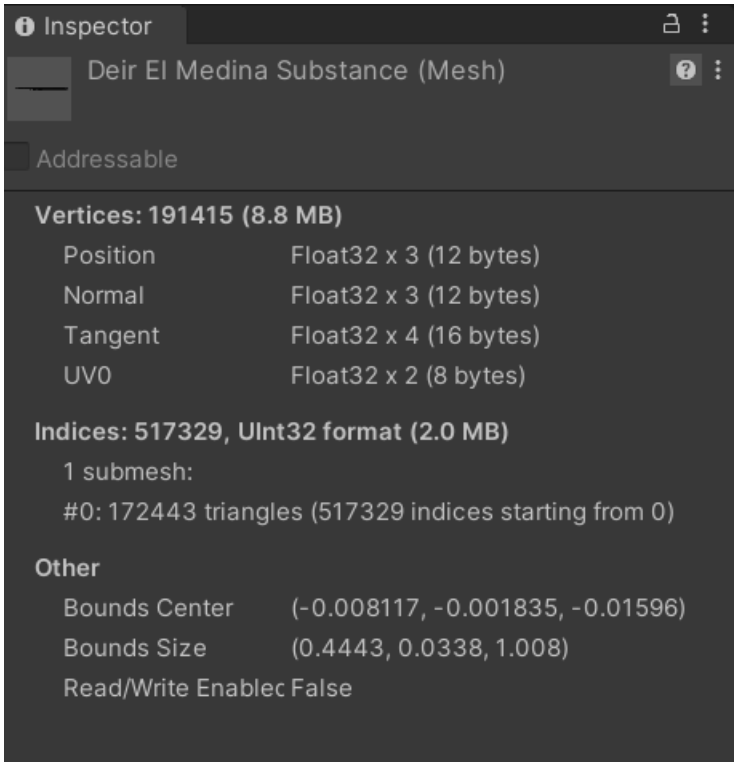


Figura 88: Caratteristiche di Deir El Medina

- Sempre sullo stesso oggetto si può utilizzare draw call batching o combine meshes in quanto la città è composta da diversi pezzi tutti composti dallo stesso materiale.
- Il culling può essere utilizzato nella scena di Deir El Medina in due momenti: quando l'utente sta esplorando la città, ciò che si trova dentro l'abitazione dello scriba è nascosto, e può essere non renderizzato se i muri della casa sono di tipo occluder; quando l'utente si trova all'interno dell'abitazione, ciò che può non essere renderizzato è l'intero villaggio perché non si trova nel campo visivo della camera.
- Per quanto riguarda lo scriba non possono essere utilizzate molte tecniche di ottimizzazione. L'unica tecnica utile è quella di mettere l'oggetto come occludee così da non essere renderizzato fino a quando non è visibile nella scena. Tecniche come la LOD non è utile utilizzarle perché una volta che lo scriba è renderizzato, rimane sempre alla stessa distanza dalla camera. Se fosse cambiata la distanza allora sarebbe conveniente usarla perché l'oggetto possiede più di 30000 vertici.

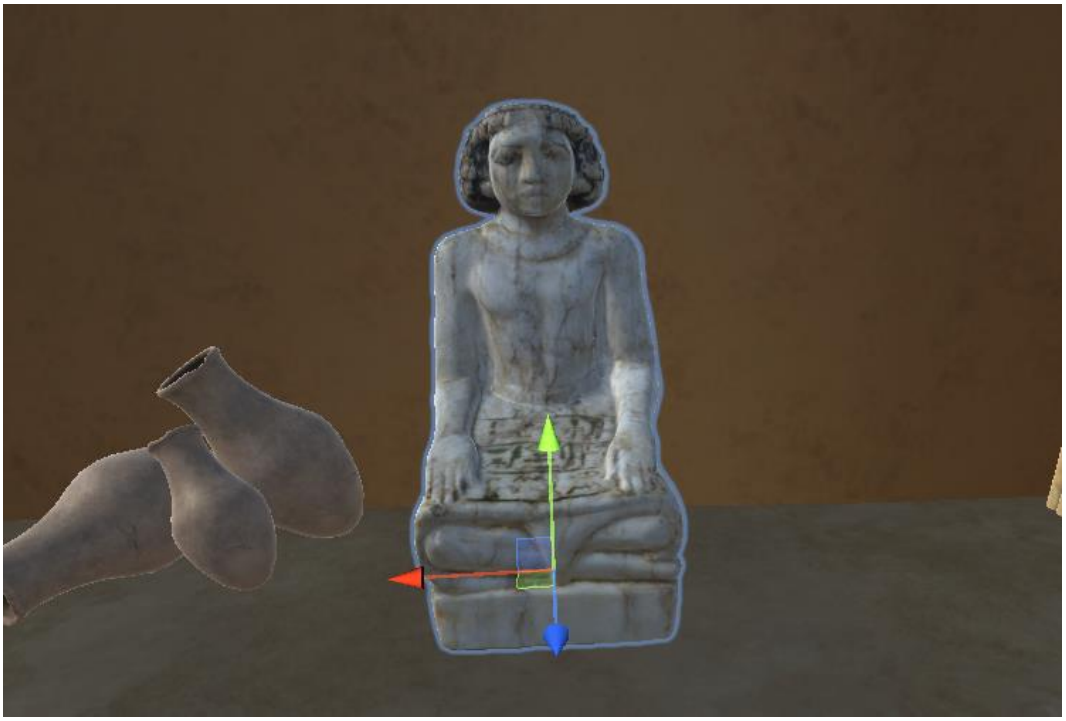


Figura 89: Scriba

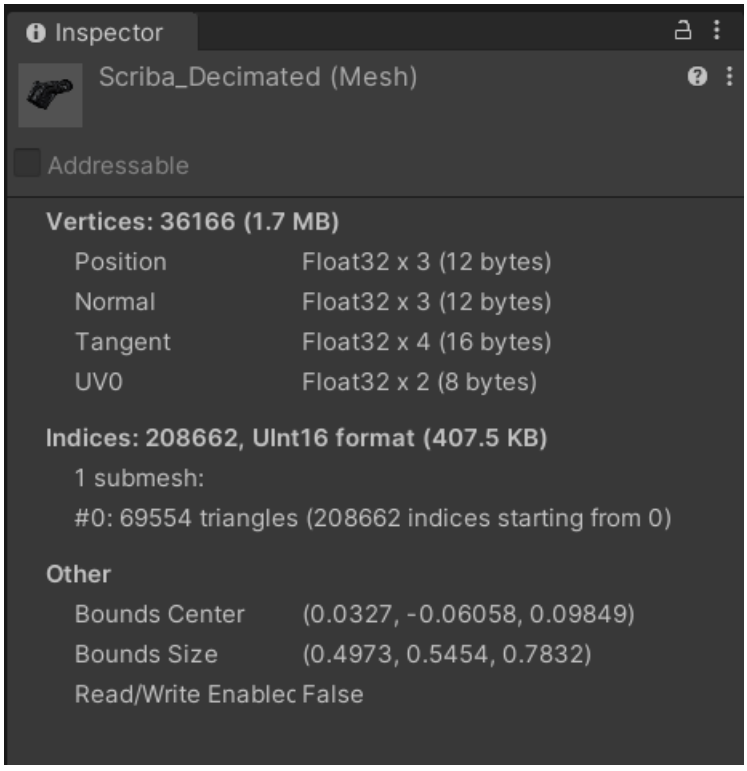


Figura 90: Caratteristiche dello scriba

7. ESPERIENZA UTENTE

In questo capitolo verranno raccontate delle esperienze che alcuni utenti hanno vissuto durante la visita al museo. Verrà messo in mostra come si sono approcciati all'applicazione e le loro considerazioni. Sono stati scelti utenti di diversa età per avere un quadro generale della riuscita o meno dell'applicazione.

7.1 ESPERIENZA UTENTE 1

Il visitatore in questione è una donna di 49 anni, insegnante di sostegno in una scuola elementare e madre di quattro figli. Ha portato i suoi figli in giro per musei e visto che ha sentito che al Museo Egizio aveva aperto una nuova sala non ha esitato ad andare a scoprire di cosa si trattasse. Si è subito sentita immersa all'interno dell'antico Egitto e quando si è trovata davanti all'applicazione *Tipi di testi nell'antico Egitto*, è stata catturata dalla grafica. La schermata che si è trovata davanti era la schermata di "pausa" e volendo provare l'esperienza dall'inizio ha aspettato che si resettasse il tutto. Partendo dalla prima scena, Deir El Medina, ha potuto godere appieno dell'esperienza e ha trovato molto interessanti i contenuti presenti, dalla descrizione dei papiri, alle traduzioni, e alle descrizioni degli scribi presenti nell'applicazione. Ha trovato l'applicazione intuitiva e di facile utilizzo.

7.2 ESPERIENZA UTENTE 2

Il visitatore della seconda esperienza è un ragazzo di 24 anni, studente di Lingue all'Università di Torino. Ha sentito parlare da alcuni amici dell'apertura della Galleria della scrittura e un giorno è entrato al Museo Egizio per scoprirla. Una volta arrivato davanti a *Tipi di testi nell'antico Egitto* si è stupito positivamente di

come un'applicazione digitale possa essere stata inserita in un contesto storico e culturale come il Museo Egizio. L'inserimento della piattaforma, secondo l'utente, stimola la curiosità di tutte le generazioni che si interessano sempre di più ai particolari che non sempre sono fruibili con facilità. Ha provato l'applicazione esplorando tutte le scene presenti, soffermandosi sulla descrizione e traduzione di alcuni papiri che aveva visto precedentemente in formato "cartaceo". Ha notato come prima del suo arrivo e dopo la sua esperienza, come molta gente di tutte le età si sia fermata a provare l'applicazione e ha notato che proprio la sua semplicità di utilizzo sia il punto di forza, perché anche i più piccoli possono addentrarsi facilmente in un mondo molto lontano da loro.

7.3 ESPERIENZA UTENTE 3

L'utente della terza esperienza ha 33 anni, studia infermieristica all'Università di Torino, ed è molto appassionato di musei e cultura in generale. Quando ha sentito parlare della nuova sala che avrebbe inaugurato il Museo Egizio non ha esitato ad andare a vederla. Quando si è trovato davanti a *Tipi di testi nell'antico Egitto*, questa era impostata sulla scena Home, nella sottoscena Home. Ciò che ha scelto l'utente di fare è stato cliccare il tasto "Deir El Medina", portandolo così ad esplorare la città e conoscere qualcosa in più riguardo agli scribi presenti. È proprio questo ciò che ha incuriosito maggiormente l'utente, l'idea di poter interagire con personaggi "virtuali", e secondo lui è stato questo tipo di approccio in un'opera di un museo a far sì che ci fosse la coda per provare l'applicazione. Sostiene che questi tipi di opere interattive siano una novità per i musei, ma se ci fossero altre nuove non esiterebbe a provarle. Ha valutato l'applicazione con un punteggio di 4 su 5, probabilmente perché ha iniziato la sua esperienza non dall'inizio, ma portando avanti quella di un altro visitatore.

7.4 ESPERIENZA UTENTE 4

L'utente della quarta esperienza è una bambina di 9 anni che ha visitato il Museo Egizio in gita con la scuola. La classe non sapeva che in quel periodo stava aprendo una nuova sala quindi l'arrivo davanti all'applicazione *Tipi di testi nell'antico Egitto*, è stato una sorpresa. Subito la piccola visitatrice è rimasta colpita dagli scribi che camminavano nella città di Deir El Medina, ed è proprio questo ciò che le è piaciuto di più, tanto che ha voluto sapere tutto riguardo a quei personaggi. Quando è passata alla parte di esplorazione dei papiri, non ha potuto vivere appieno l'esperienza in quanto non aveva più tempo, ma ha comunque capito come funzionava la scelta prima di una categoria e poi di una sottocategoria, riuscendo a leggere solamente uno dei papiri presenti all'interno dell'applicazione. Nel complesso l'applicazione le è risultata facile e tornerà per provare tutto con più calma.

7.5 ESPERIENZA UTENTE 5

Il visitatore è una ragazza di 24 anni, lavoratrice e appassionata di musei e cultura. Quando ha saputo che a dicembre il Museo Egizio avrebbe inaugurato una nuova sala ha subito comprato il biglietto per andare a vederla. Quando si è trovata all'interno della Galleria della scrittura è subito stata colpita dalla quantità di papiri presenti al suo interno e ha iniziato a provare le applicazioni installate in modo da non perdersi nessun dettaglio. Prima di uscire dalla sala si è imbattuta nell'applicazione *Tipi di testi nell'antico Egitto*, che le è sembrata subito intuitiva e di facile utilizzo. La grafica le è parsa nitida e le è piaciuta la coerenza dei colori che richiamano l'intera sala espositiva, infatti il filo conduttore della Galleria della scrittura è il colore oro accompagnato dal bianco. Non ha quindi esitato a provarla e si è subito sentita immersa nella città di Deir El Medina, trovando i contenuti interessanti sia per un pubblico adulto che per i più piccoli.

CONCLUSIONI

Questo elaborato ha presentato come è stata sviluppata l'applicazione *Tipi di testi nell'antico Egitto.* Questo progetto si è posto come obiettivo quello di racchiudere in un unico luogo i papiri studiati e analizzati dal Museo Egizio, in modo che un visitatore possa interagire direttamente con essi. È stata creata l'applicazione che permette agli utenti di esplorare e scegliere in totale autonomia ciò che vuole visualizzare o meno. Alcuni aspetti vanno ancora migliorati per consentire ai visitatori di vivere l'esperienza senza che ci siano lunghi tempi di caricamento delle scene o problemi riguardo ad alcune funzionalità dell'applicazione.

Il lavoro in team è stato molto efficace durante lo sviluppo dell'applicazione, la comunicazione tra i componenti è stata importante nella risoluzione di alcuni problemi e nel trovare soluzioni efficaci per impostare il progetto in maniera semplice per un visitatore medio del Museo Egizio.

SVILUPPI FUTURI

Tipi di testi nell'antico Egitto è ancora in evoluzione. Il Museo Egizio infatti, continua a studiare ed analizzare diversi papiri. L'idea è quella di ampliare il database con nuovi papiri o completando quelli già presenti. Questo comporterà una completezza dell'applicazione in quanto non saranno più presenti le categorie o sottocategorie non cliccabili.

Altre modifiche saranno anche fatte nell'interfaccia dell'applicazione. C'è la possibilità di aggiungere una nuova card, oltre a quelle già presenti del numero di catalogo e della lingua, all'interno della sottoscena Testo. Inoltre è presente l'idea di inserire nuovi modelli di scribi all'interno della scena Deir El Medina 2 in

modo da differenziarli fisicamente dagli altri ed eventualmente farli muovere in modo indipendente dagli altri.

In generale l'applicazione *Tipi di testi nell'antico Egitto*, non sarà mai completata, ma sempre in fase di sviluppo per poter soddisfare anche le richieste del pubblico.

BIBLIOGRAFIA

[1] Unity, Ultimate guide to profiling unity games, Unity, 2022.

[2] «ProfilerMarker» https://docs.unity3d.com/2021.2/Documentation/ScriptReference/Unity.Profiling.ProfilerMarker.html?utm_source=demand-gen&utm_medium=pdf&utm_campaign=profiling-for-performance&utm_content=the-ultimate-guide-to-profiling-ebook.

[3] «Profiler overview» https://docs.unity3d.com/Manual/Profiler.html.

[4] «Common Profiler markers» https://docs.unity3d.com/Manual/profiler-markers.html.

[5] «FrameDebugger» https://docs.unity3d.com/Manual/FrameDebugger.html.

[6] «LOD» https://docs.unity3d.com/Manual/LevelOfDetail.html.

[7] «Draw call batching» https://docs.unity3d.com/Manual/DrawCallBatching.html.

[8] «Static batching» https://docs.unity3d.com/Manual/static-batching.html.

[9] «Dynamic batching» https://docs.unity3d.com/Manual/dynamic-batching.html.

[10] «Combine meshes» https://docs.unity3d.com/ScriptReference/Mesh.CombineMeshes.html.

[11] «GPU instancing» https://docs.unity3d.com/Manual/GPUInstancing.html.

[12] «Culling» https://docs.unity3d.com/Manual/OcclusionCulling.html.

[13] «Baking delle luci»: https://docs.unity3d.com/Manual/LightMode-Baked.html.