



UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione
e Matematica

Tesi di Laurea Triennale in Informatica

Progettazione e sviluppo di un assistente virtuale di cucina in realtà mista

Relatore

Prof. Francesco Tarquini

Correlatore

Dr. Leonardo D'Errico

Laureando

Giacomo Paolocci

278662

Anno Accademico 2024-2025

Indice

1	Introduzione	1
2	Tecnologie Utilizzate	2
2.1	Realtà Mista (MR)	2
2.1.1	Storia	2
2.1.2	Definizione	2
2.1.3	Casi d'uso	3
2.2	Hardware	4
2.2.1	HoloLens 2	4
2.3	Software e AI	5
2.3.1	Unity	5
2.3.2	MRTK	6
2.3.3	Google Gemini	7
3	Tecnologie concorrenti, simili e confronti	8
3.1	Hardware	8
3.1.1	Meta Quest 3	8
3.1.2	HoloLens 1	8
3.2	Tecnologie di Visualizzazione	8
3.2.1	Realtà Virtuale (VR)	9
3.2.2	Realtà Aumentata (AR)	9
3.2.3	Differenze e confronto tra AR e MR	9
3.2.4	Differenze e confronto tra VR e MR	9
4	Processo e tecniche di sviluppo per applicazioni MR per Hololens2	10
4.1	Setup di Unity	10
4.1.1	Installazione di Unity	10
4.1.2	Installazione di Mixed Reality Feature Tool	11
4.1.3	Impostazioni del progetto e della scena	12
4.2	MRTK3.0	15
4.2.1	Tabs	15
4.2.2	Scroll list	16
4.2.3	Slate	17
4.2.4	Near Menu	18
4.3	Programmazione in Unity con C#	19
4.3.1	MonoBehaviour	19
4.3.2	Prefab	20
4.3.3	IEnumerator e Coroutine	20
4.3.4	Singleton	20

5 Utilizzo di Gemini come LLM e Spatial Understanding	22
5.1 Cosa è una LLM	22
5.2 Prompting	23
5.3 Utilizzo di Gemini per Spatial Understanding	23
5.4 Utilizzo di Gemini come LLM	23
6 Progettazione	24
6.1 Casi D'uso	24
6.2 Architettura	25
6.3 Interfaccia grafica	25
6.3.1 Menu	25
6.3.2 Finestra per scatto della foto e generazione degli ingredienti	26
6.3.3 Finestra per visualizzazione delle ricette	29
6.4 Persistenza dei dati	29
7 Implementazione dell'applicazione	30
7.1 Introduzione	30
7.2 Gemini API	30
7.2.1 Structured output	30
7.2.2 Richieste POST	31
7.3 File Handler	33
7.3.1 Classi Serializzabili	33
7.3.2 Metodi	34
7.4 Scatto della foto	35
7.5 Lista ingredienti	37
7.5.1 Prefab	37
7.5.2 Creazione del prefab all'interno della lista	38
7.5.3 Aggiunta, modifica e cancellazione	40
7.6 Tastiera virtuale	42
7.7 Problematiche e soluzioni	43
7.7.1 Il file di config non veniva letto	43
7.7.2 I prefab dentro la scroll list avevano dei collider di grandezza errata	43
8 Conclusioni	44
8.1 La mia esperienza con Unity e MRTK	44
8.2 Sviluppi futuri della mia applicazione	44
8.3 Considerazioni finali	44
Bibliografia	45

Capitolo 1

Introduzione

Questa tesi si propone di esplorare l'uso della Realtà Mista (MR) in un contesto culinario, con particolare attenzione all'uso di visori come gli HoloLens 2.

L'obiettivo principale è quello di sviluppare un'applicazione che possa aiutare a creare ricette con ingredienti inutilizzati evitando anche lo spreco alimentare. Tutto ciò riassunto in un'applicazione semplice e intuitiva che utilizza la MR per sovrapporre informazioni digitali al mondo reale, così da avere pieno controllo di quello che si fa e allo stesso tempo avere le mani libere da qualsiasi dispositivo fisico.

L'applicazione è stata sviluppata utilizzando Unity e il Mixed Reality Toolkit (MRTK), che forniscono gli strumenti necessari per creare esperienze MR. Inoltre, utilizzando L'LLM Google Gemini è possibile generare ricette in base agli ingredienti disponibili facendo una semplice foto, rendendo l'applicazione ancora più utile e versatile.



Windows Mixed Reality

Capitolo 2

Tecnologie Utilizzate

2.1 Realtà Mista (MR)

2.1.1 Storia

Lo studio di tecnologie simili a realtà aumentata (AR), realtà mista (MR) e realtà virtuale (VR) risalgono alla fine degli anni '50, in particolare nel 1957 Morton Heiling sviluppò un simulatore, grande quanto un cabinato per videogiochi, che permetteva all'utilizzatore di visualizzare immagini 3d stereoscopiche, integrando l'esperienza con vento artificiale, vibrazioni, audio stereo e addirittura un dispositivo per la creazione di profumi.

Ma l'applicazione che si avvicina di più a qualcosa di moderno arriva dall'utilizzo in aviazione, in particolare un sistema chiamato "Sword of Democles", un primo casco dotato di lenti per l'AR. Questo dispositivo serviva ad aiutare i piloti di elicottero ad atterrare di notte azionando le telecamere con l'utilizzo della testa. Un problema di questo dispositivo era il peso eccessivo, e da questo deriva proprio il suo nome.

Lo sviluppo continua nel 1992 con Loui B. Rosenberg con lo sviluppo del primo sistema AR immersivo che permetteva di guidare bracci robotici, tecnologia successivamente utilizzata nell'aeronautica militare statunitense. [1]

In tempi recenti abbiamo visto lo sviluppo dei Google Glass, usciti nel 2013 e ritirati dal mercato nel 2023 [2], e dei primi visori di realtà virtuale come l'Oculus Rift CV1, uscito nel 2016 [3], che hanno creato le basi per visori più recenti come il Meta Quest 3 e gli Hololens 2, quest'ultimi utilizzati proprio per la realizzazione di questa tesi.

2.1.2 Definizione

La realtà mista consente di sovrapporre al mondo fisico elementi tridimensionali con il quale l'utente può interagire. La realtà mista si contraddistingue per il fatto di avere un dispositivo, di solito un visore, che riconosce l'ambiente circostante mappandolo tridimensionali, in cui inserire oggetti tridimensionali tramite l'utilizzo di appositi sensori, la comprensione dei movimenti delle mani, degli occhi e la comprensione di input vocali.

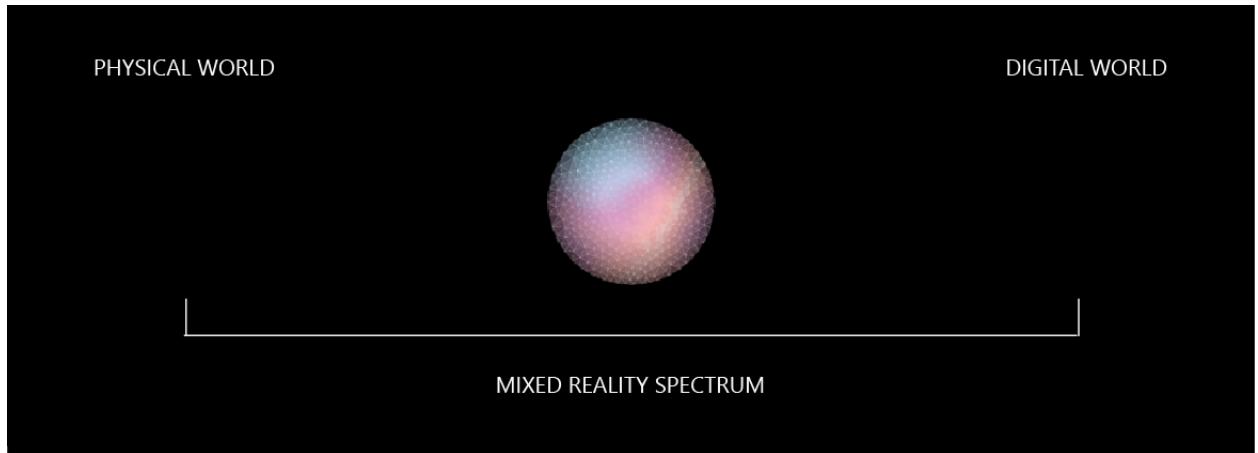


Figura 2.1: Rappresentazione della fusione del mondo fisico con il mondo digitale nella realtà mista.

Per consentire una esperienza immersiva fra realtà e oggetti 3d, chiamati anche ologrammi in ambito Microsoft, è emersa una nuova disciplina chiamata "HIC" o anche "Human-Computer Interaction", che si occupa di studiare come interagire virtualmente con il computer, ad esempio tramite tastiere, mouse, comandi vocali o tracciamento dei movimenti. [4]

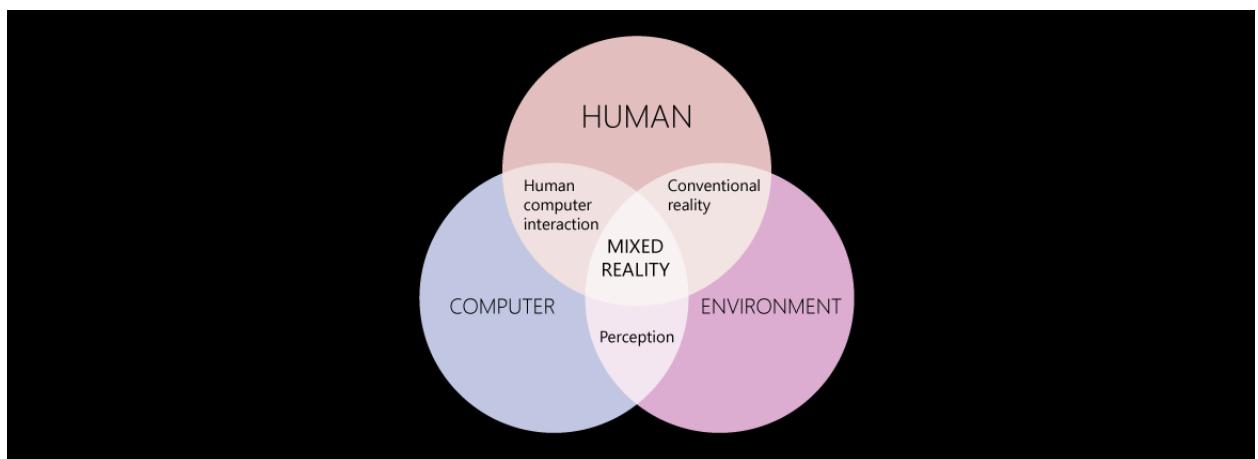


Figura 2.2: interazioni tra computer, utente e ambiente circostante.

2.1.3 Casi d'uso

Formazione di piloti in aereounatica

Tramite l'utilizzo della realtà mista è possibile fare delle lezioni interattive simulate, ad esempio la risoluzione di un guasto all'impianto elettrico visualizzando in realtà mista la schematica dell'impianto, oppure illustrare ed allenare ad un'ispezione esterna dell'aereo. In questo ultimo caso lo studente, in una stanza abbastanza grande, può camminare intorno al modello 3d in scala dell'aereo, venendo aiutato dalla applicazione in MR durante le varie azioni di controllo, ad esempio la rotazione di maniglie o apertura di porte.[5]

Formazione e simulazione in ambito industriale

La realtà aumentata offre un modo efficace e sicuro per imparare attraverso la pratica, soprattutto in situazioni potenzialmente pericolose. Grazie a simulazioni immersive, i tirocinanti possono affrontare

scenari realistici senza correre rischi reali, ricevendo feedback immediato sui propri errori. Questo tipo di formazione unisce la percezione dell'ambiente reale con l'attività virtuale, permettendo di sviluppare competenze pratiche senza creare danni a persone o macchinari [6]



Figura 2.3: Utilizzo della realtà aumentata in ambito industriale

Prototipazione e produzione per le imprese

L'integrazione della realtà aumentata nei processi di prototipazione e produzione può portare a una significativa riduzione dei costi, soprattutto su scala industriale. Spostare la progettazione in uno spazio virtuale consente di evitare la produzione di prototipi fisici costosi e di velocizzare l'intero ciclo di sviluppo grazie a iterazioni rapide e meno dispendiose. Questo approccio non solo ottimizza le tempistiche, ma migliora anche la precisione nella produzione, aumentando la sicurezza e l'affidabilità delle linee produttive. In molti casi, l'introduzione della realtà aumentata ha portato benefici concreti in termini di stabilità delle infrastrutture, sicurezza dei lavoratori e qualità del prodotto finale.[7]

Musei e mostre virtuali

La realtà virtuale si è dimostrata uno strumento potente per il mondo dei musei, delle mostre d'arte e d'intrattenimento, dei servizi di hosting di eventi, dei servizi turistici e di altre attività e servizi simili incentrati sulle mostre [8]. In particolare l'utilizzo della realtà mista in ambienti reali consente una perfetta armonia tra il mondo reale e quello virtuale, permettendo di visualizzare opere d'arte o reperti storici in 3d, con la possibilità di interagire con essi. [9]

2.2 Hardware

Fra i vari visori di realtà mista, il più utilizzato è l'HoloLens 2, un visore sviluppato da Microsoft, che permette di visualizzare ologrammi in 3d e interagire con essi tramite comandi vocali, movimenti delle mani e tracciamento degli occhi.

2.2.1 HoloLens 2

Gli HoloLens 2, successori degli HoloLens 1, sono visori di realtà mista sviluppati da Microsoft e rilasciati nel 2019, sono dotati di un display olografico che consente di sovrapporre oggetti virtuali al mondo reale e si basano sul sistema operativo "Windows Holographic OS" derivato da Windows 10

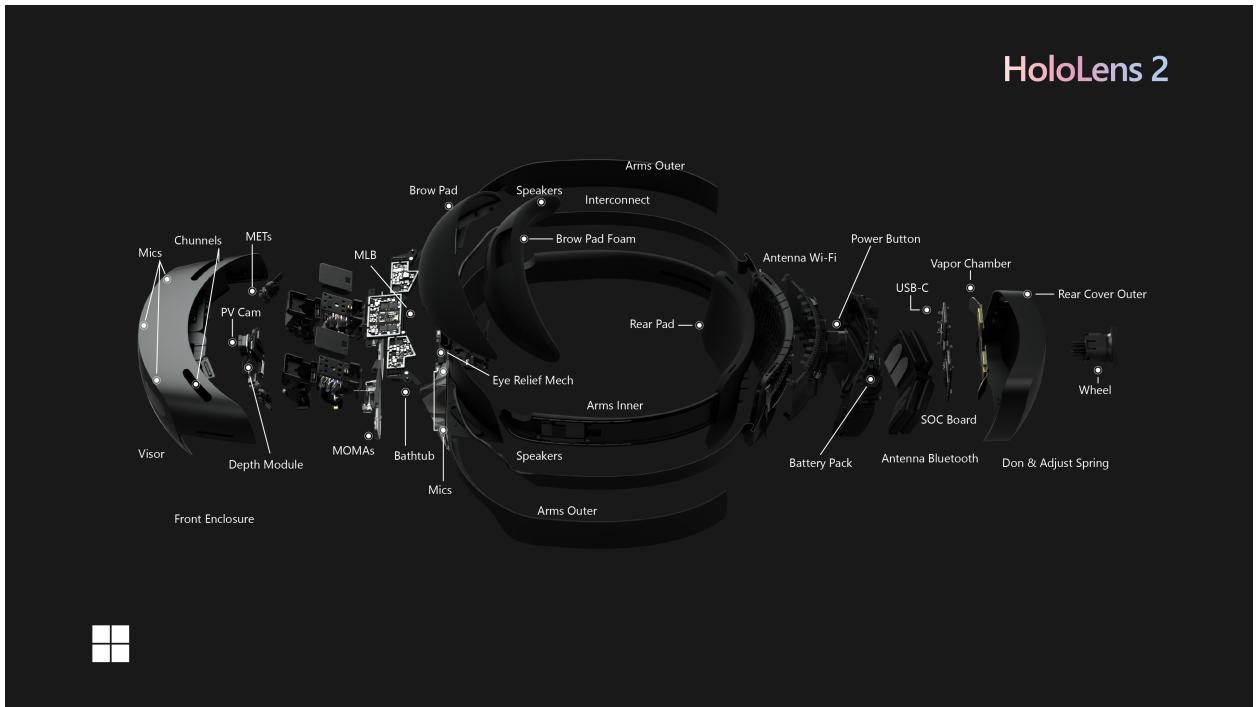


Figura 2.4: Esplosione 3d del visore HoloLens 2

Come si può vedere in figura tutta la parte di elaborazione di trova nella parte posteriore del visore, mentre nella parte anteriore troviamo l'hardware per la proiezione degli ologrammi, i sensori come giroscopio e accelerometro, e le telecamere che permettono il tracciamento delle mani e degli occhi. In particolare abbiamo 4 telecamere frontali per il riconoscimento dell'ambiente circostante, 2 telecamere infrarossi per il tracciamento oculare e una fotocamera frontale a colori da 8megapixel. Il visore è dotato di un processore Qualcomm Snapdragon 850, 4 GB di RAM e 64 GB di memoria interna, la batteria ha una durata di circa 2-3 ore. Il visore è dotato di un sistema audio spaziale e di un microfono a 5 canali usati anche per il riconoscimento vocale. [10]
Per questo progetto è stato adottato questo visore solo perché offriva l'accesso alla telecamera frontale, cosa non disponibile in altri visori come il Meta Quest 3, oltre al fatto di essere quello di più recente produzione da parte di Microsoft.

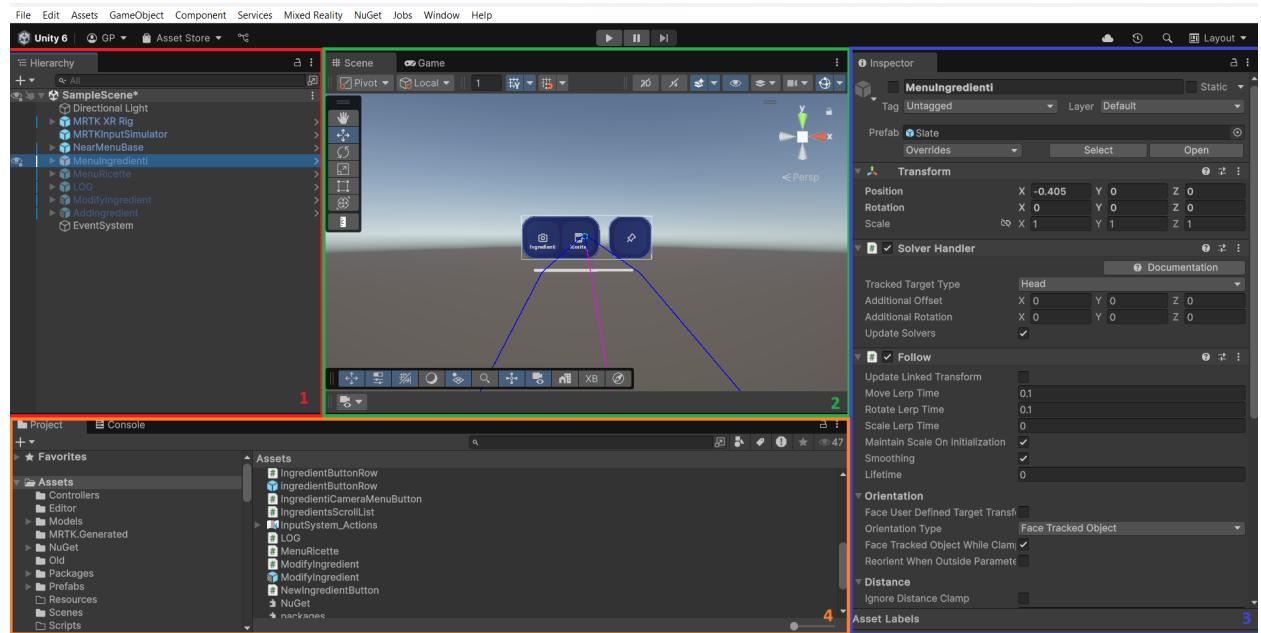
2.3 Software e AI

Per la realizzazione del progetto sono stati utilizzati il motore grafico Unity, il Mixed Reality Toolkit (MRTK) e Google Gemini.

2.3.1 Unity

Unity è un motore grafico multipiattaforma sviluppato da Unity Technologies, utilizzato per la creazione di videogiochi e applicazioni in 2D e 3D. È particolarmente popolare per lo sviluppo di giochi per dispositivi mobili, console e PC. Nel nostro caso è stato utilizzato per la creazione di un'applicazione di realtà mista per gli HoloLens 2 utilizzando il Mixed Reality Toolkit (MRTK) e il linguaggio di programmazione C#.

Overview di Unity



- 1. Gerarchia:** La gerarchia mostra tutti gli oggetti presenti nella scena. Puoi organizzare gli oggetti in una struttura ad albero, creando genitori e figli per raggruppare gli oggetti correlati.
- 2. Scena:** La scena è l'area di lavoro principale in Unity, dove puoi posizionare e organizzare gli oggetti 3D, le luci e le telecamere. Puoi visualizzare la scena in diverse modalità, come la vista 3D o la vista 2D.
- 3. Inspector:** L'Inspector mostra le proprietà e i componenti dell'oggetto selezionato nella gerarchia. Puoi modificare le proprietà degli oggetti, aggiungere componenti e configurare le impostazioni. Un componente potrebbe essere anche uno script C# che definisce il comportamento dell'oggetto.
- 4. Project e Console:** La finestra Project mostra tutti i file e le risorse del tuo progetto, come modelli 3D, texture, suoni e script. Puoi organizzare le risorse in cartelle per una gestione più semplice. La Console mostra i messaggi di errore, avviso e debug generati durante l'esecuzione del gioco o dell'applicazione.

2.3.2 MRTK



L'MRTK è un progetto della Microsoft che fornisce una serie di strumenti e librerie per lo sviluppo di applicazioni di realtà mista su piattaforme Microsoft, in particolare per gli HoloLens. Il toolkit include una serie di componenti e funzionalità per facilitare lo sviluppo di interfacce utente, il

tracciamento delle mani, il riconoscimento vocale e altre funzionalità specifiche per la realtà mista. L'MRTK è progettato per essere facilmente integrato in Unity e offre una serie di esempi e documentazione per aiutare gli sviluppatori a iniziare rapidamente.

Oltre a supportare gli Hololens supporta anche altri visori come gli Oculus e permette anche di creare applicazioni AR (Argumented Reality) per Android e iOS.[11]

2.3.3 Google Gemini

Google Gemini, inizialmente uscita sotto il nome di Google Bard, rappresenta uno dei più recenti sviluppi dell'intelligenza artificiale generativa di Google. Gemini, oltre a elaborare del testo, è in grado di comprendere anche immagini, audio, documenti e altri file di tipo testuale. L'obiettivo di Google con Gemini è offrire un assistente personale basato su intelligenza artificiale che sia non solo utile e produttivo, ma anche creativo e capace di stimolare la curiosità dell'utente. In ambito produttivo, Gemini può ad esempio sintetizzare documenti lunghi, supportare attività di scrittura, aiutare nella programmazione e automatizzare attività complesse.

Nonostante il grande potenziale, Google riconosce che la tecnologia è ancora in una fase iniziale e presenta alcune limitazioni tipiche dei modelli linguistici di grandi dimensioni (LLM). Tra queste, vi sono il rischio di fornire risposte inaccurate, la possibilità che emergano bias nei contenuti generati, e la difficoltà nel rappresentare prospettive multiple su questioni soggettive. Inoltre, l'assistente può occasionalmente mostrare segnali di "personalizzazione" e sembrare esprimere emozioni o opinioni, che in realtà non possiede, essendo un modello statistico. Altri limiti includono i cosiddetti falsi positivi e falsi negativi, ossia casi in cui Gemini rifiuta di rispondere a domande appropriate o, al contrario, fornisce risposte inappropriate. Infine, il sistema può essere vulnerabile a prompt avversari, ovvero tentativi intenzionali di metterne alla prova i limiti o aggirarne i filtri di sicurezza. Per evitare ciò, e rispettare le proprie linee guida, Google ha adottato un approccio responsabile nello sviluppo di Gemini, condudendo degli stress test svolti da team interni incaricati di individuare criticità, formando appositi "red team", composti da esperti e scienziati sociali.[12] [13]

Capitolo 3

Tecnologie concorrenti, simili e confronti

3.1 Hardware

Andiamo a esplorare altri Hardware concorrenti o simili agli Hololens 2, facendo un confronto delle caratteristiche principali e del perchè sono stati scartati per essere utilizzati in questo progetto.

3.1.1 Meta Quest 3

Il Meta quest 3 è il visore di punta sul mercato consumer di Meta, è un visore standalone, ma che si può collegare anche al PC, con un sistema operativo basato su Android chiamato Meta Horizon OS. A differenza degli Hololens 2 ha un processore molto più potente, uno Snapdragon XR2 Gen2, con al massimo 512GB di memoria interna e 8Gb di Ram. A differenza degli Hololens2 ha uno schermo LCD con una risoluzione di 2064x2208 per occhio, e un refresh rate massimo di 120Hz. Dato che ha un display la realtà mista viene garantita tramite un sistema di passthrough, che permette di vedere il mondo reale attraverso le telecamere del visore, che riescono a ricostruire in maniera quasi perfetta la normale visione di un occhio umano. Dal punto di vista dello sviluppo di app questo avviene sempre con Unity e un pacchetto di sviluppo, ma a differenza degli Hololens 2 non è possibile accedere in nessun modo alle fotocamere frontali, diminuendo di molto le capacità delle applicazioni in realtà mista.

3.1.2 HoloLens 1

Gli Hololens di prima generazione, usciti nel 2016, sono i predecessori degli Hololens 2, sono dotati di un Intel Atom x5-Z8100, 2Gb di RAM e 64Gb di memoria interna, le prestazioni, comparate con quelle del visore di nuova generazione, sono nettamente inferiori, inoltre questo visore è in fase di dismissione, in favore della nuova generazione. Quindi la scelta degli Hololens 2 è abbastanza banale, anche solo per il confronto di prestazioni.

3.2 Tecnologie di Visualizzazione

Oltre alla MR (Mixed Reality) esistono altre tecnologie di visualizzazione, le più importanti sono la VR (Virtual Reality) e la AR (Augmented Reality).

3.2.1 Realtà Virtuale (VR)

La Realtà Virtuale crea un ambiente virtuale completamente immersivo dove l'utente può muoversi e interagire, ad esempio prendendo oggetti. Ogni utente può avere anche un avatar e interagire con altri utenti nel mondo virtuale. Di solito questo tipo di tecnologia viene utilizzata con dei visori dotati di schermi che coprono completamente il campo visivo dell'utente, e che sono dotati di sensori di movimento, per permettere all'utente di muoversi in questo mondo virtuale. La VR viene utilizzata principalmente per videogiochi, ma anche per applicazioni professionali, come ad esempio la simulazione di ambienti per la formazione, ma anche per la creazione di riunioni aziendali virtuali.

3.2.2 Realtà Aumentata (AR)

Il più famoso esempio di Realtà Aumentata potrebbe essere il gioco Pokemon Go, dove gli utenti possono vedere i Pokemon nel mondo reale attraverso lo schermo del proprio smartphone per poi catturarli. La AR permette di sovrapporre oggetti virtuali al mondo reale, visualizzando informazioni extra come testi, indicazioni stradali, e oggetti 3d, ma non permette di interagire con essi.

3.2.3 Differenze e confronto tra AR e MR

La Realtà Aumentata e la Realtà Mista sono due tecnologie simili, ma con differenze fondamentali. La Realtà Aumentata sovrappone oggetti virtuali al mondo reale, ma non permette di interagire con essi in modo significativo. La Realtà Mista, invece, integra gli oggetti virtuali nel mondo reale in modo più profondo, permettendo all'utente di interagire con essi come se fossero parte del mondo reale. Ad esempio, in un'applicazione di Realtà Mista, un utente potrebbe afferrare un oggetto virtuale e spostarlo virtualmente nello spazio reale, mentre in un'applicazione di Realtà Aumentata l'oggetto virtuale rimarrebbe statico e non interagirebbe con l'ambiente circostante.

3.2.4 Differenze e confronto tra VR e MR

La Realtà Virtuale e la Realtà Mista sono due tecnologie che offrono esperienze immersive, ma con differenze significative. La Realtà Virtuale crea un ambiente completamente virtuale, isolando l'utente dal mondo reale, mentre la Realtà Mista integra elementi virtuali nel mondo reale, consentendo interazioni più naturali e intuitive per l'utente.

[14] [15]

Capitolo 4

Processo e tecniche di sviluppo per applicazioni MR per Hololens2

In questa sezione andremo a vedere come inizializzare un progetto Unity e utilizzare l'MRTK al suo interno. Successivamente si andrà ad approfondire entrambi, andando a vedere il funzionamento dei componenti utilizzati in questo progetto e le tecniche di programmazione utilizzate.

4.1 Setup di Unity

4.1.1 Installazione di Unity

Andare nel sito ufficiale di Unity e scaricare l'ultima versione disponibile. Dopo l'installazione ci si troverà davanti all'Unity Hub, qui la schermata potrebbe leggermente cambiare a seconda della versione ma delle opzioni 3d disponibili **non** bisogna scegliere la versione **URP** o **HDRP** ma la versione normale **3D**.

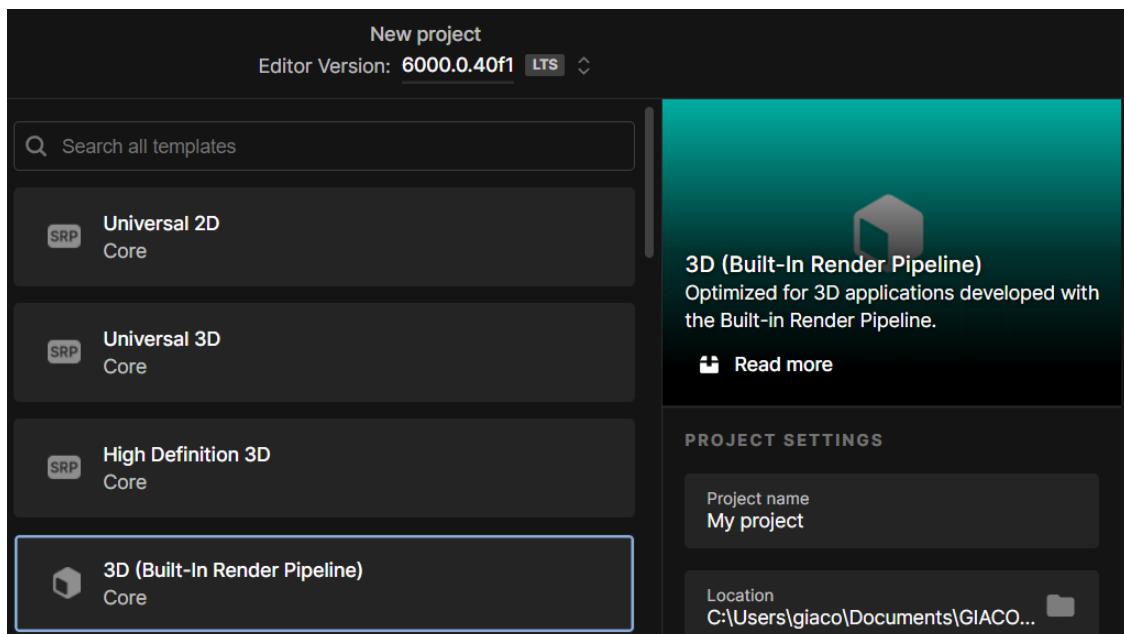


Figura 4.1: In questo caso la versione corretta è quella selezionata

Dopo aver dato nome al progetto e selezionata la cartella di installazione andiamo subito a modificare le impostazioni di build.

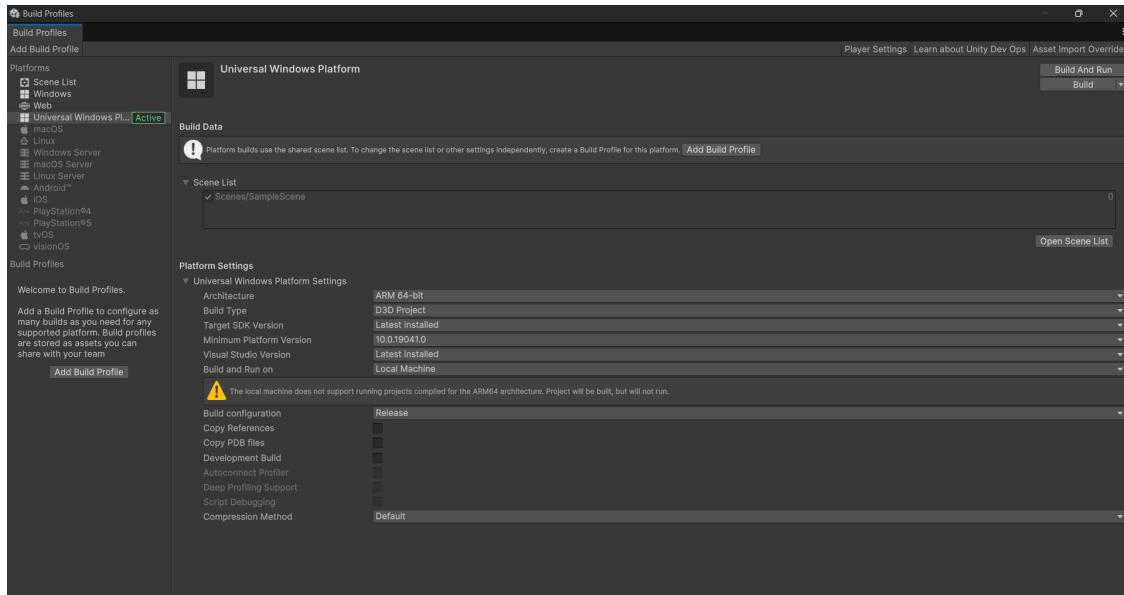


Figura 4.2: Da notare il "Build Type"

Prima di tutto bisogna essere sicuri di aver attivato la build per l'Universal Windows Platform e di aver selezionato la piattaforma **ARM64** come architettura, questo è fondamentale per il corretto funzionamento dell'applicazione su Hololens2, poi bisogna essere sicuri di avere come build type **D3D**.

4.1.2 Installazione di Mixed Reality Feature Tool

Dopo questi passaggi scaricare il "Mixed Reality Feature Tool" dal sito ufficiale di Microsoft, questo strumento ci permette di scaricare e installare l'MRTK (Mixed Reality Toolkit) direttamente all'interno del nostro progetto. Una volta scaricato, aprire l'eseguibile, selezionare il path del progetto Unity, nella schermata successiva selezionare tutta la sezione "MRTK3" e alla sezione "Platform Support" selezionare "Mixed Reality OpenXR Plugin".

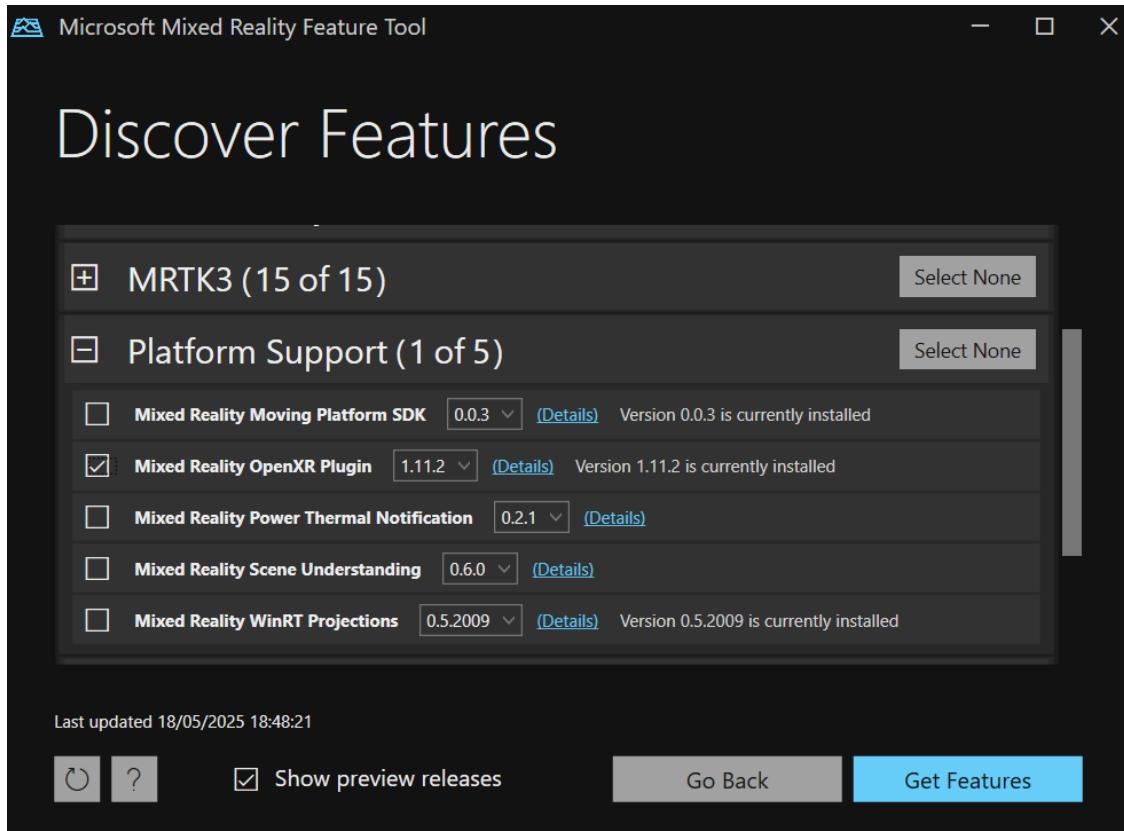


Figura 4.3

Dopo ciò ritornare su Unity e aspettare che il pacchetto venga importato.

4.1.3 Impostazioni del progetto e della scena

Una volta completata l'installazione dei pacchetti andare nelle impostazioni del progetto e seguire le seguenti immagini:

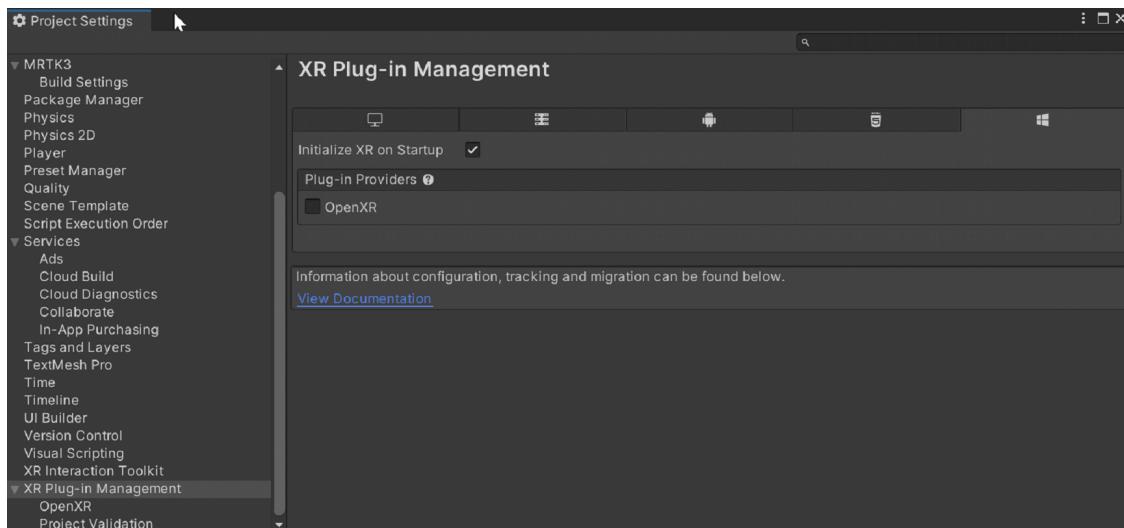


Figura 4.4: Andare nelle impostazioni del progetto e selezionare la sezione "XR Plug-in Management"

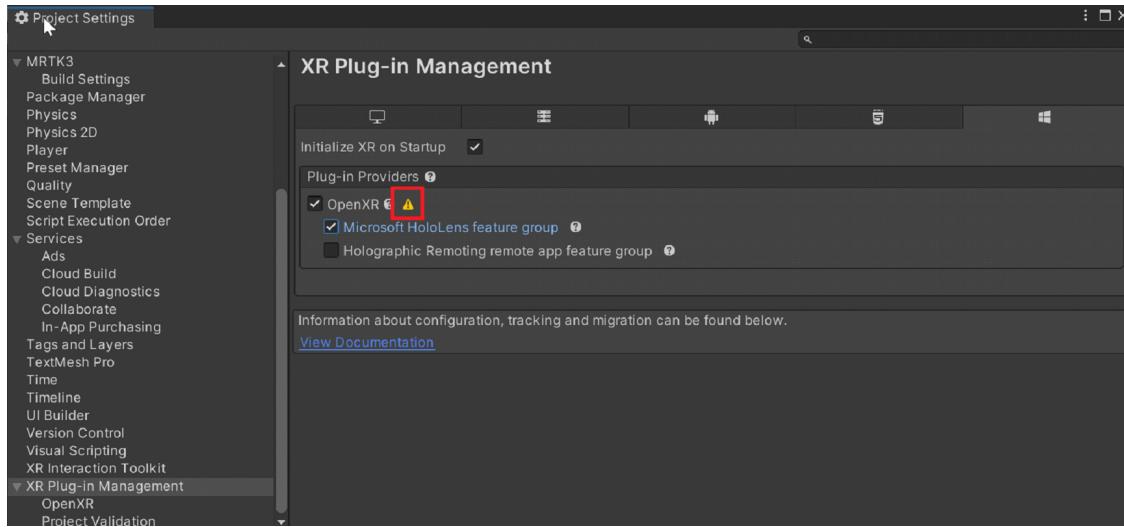


Figura 4.5: Abilitare l'OpenXR

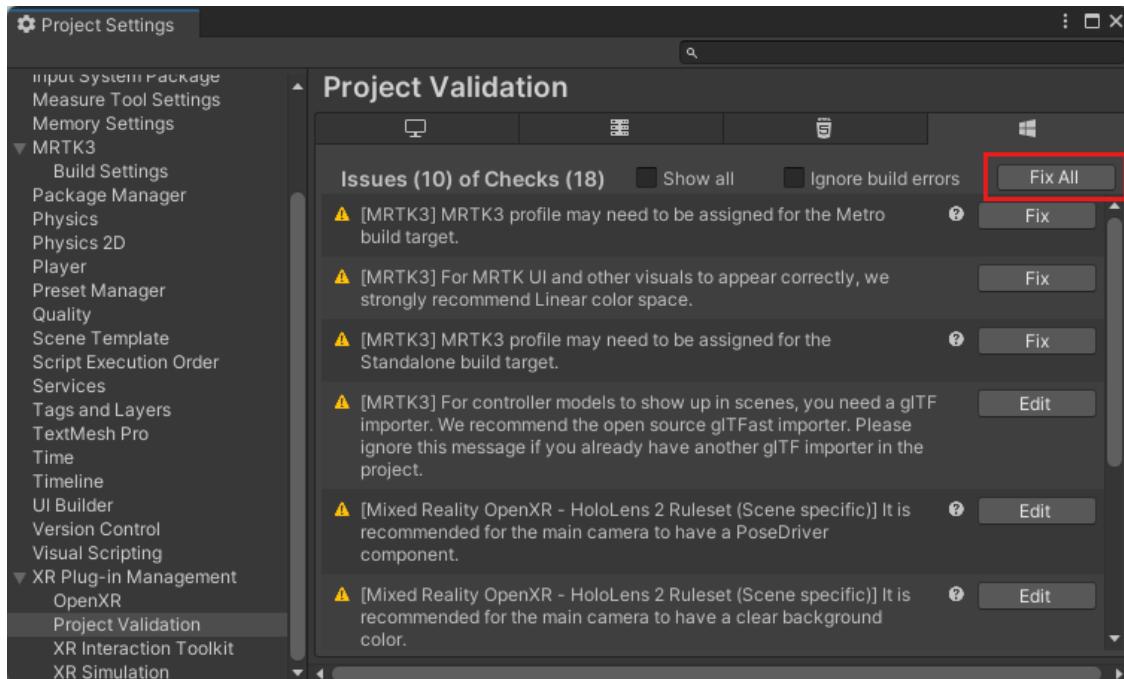


Figura 4.6: Se compare il triangolo giallo, cliccarci sopra e premere "Fix All"

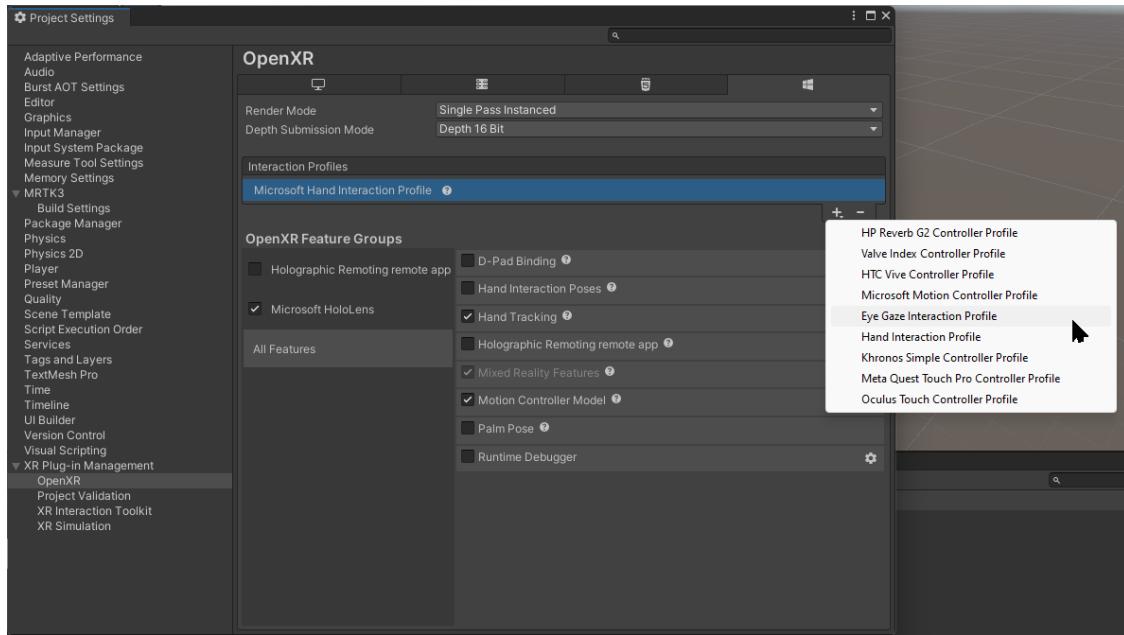


Figura 4.7: Inserire i seguenti profili e abilitare le spunte in foto

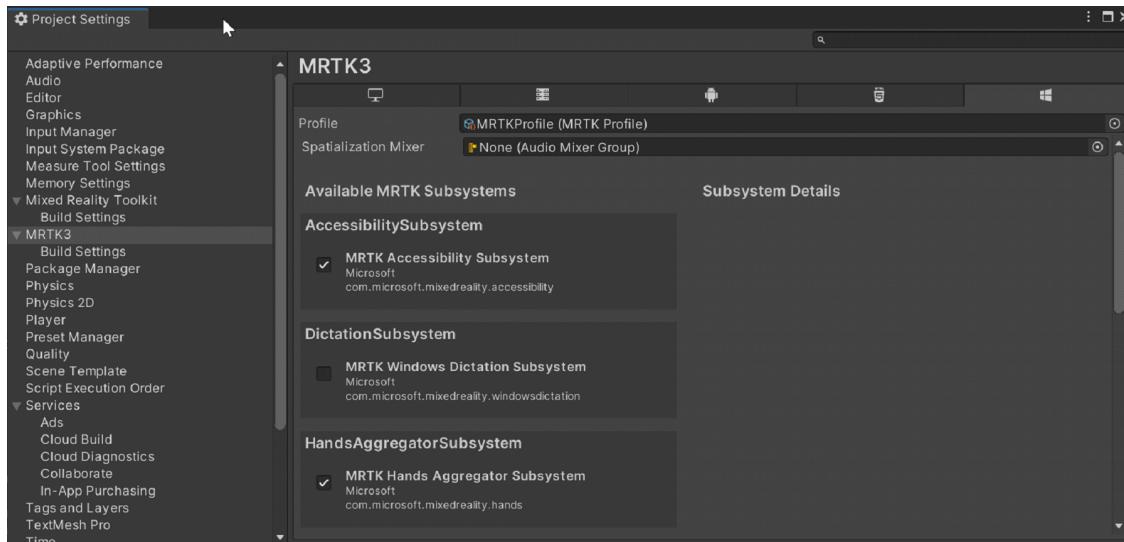


Figura 4.8: Nella sezione MRTK3 selezionare il profilo come in figura

Una volta completati questi passaggi tornare alla scena principale di Unity, andare nella sezione che mostra le gerarchie degli oggetti ed eliminare tutto, poi andare nella sezione in basso denominata "Project", navigare dentro "Pakages" -> "MRTK Input" -> "Assets" -> "Prefabs", trovare "MRTK XR Rig" e trascinarlo nella scena.

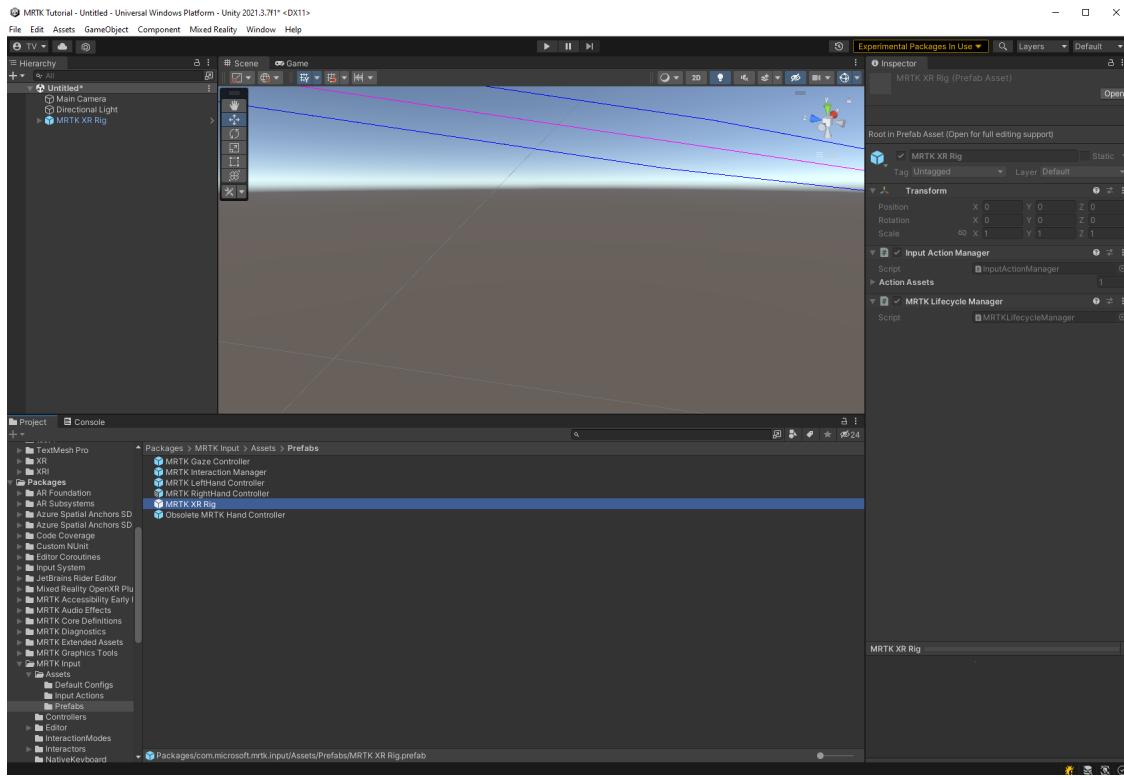


Figura 4.9

Fare la stessa cosa con "MRTK Input Simulator" che si trova in "Packages" -> "MRTK Input" -> "Simulation" -> "Prefabs". Con questo il setup di Unity è completo, ora possiamo iniziare a lavorare con l'MRTK e a sviluppare la nostra applicazione.

4.2 MRTK3.0

Andiamo a vedere i principali componenti utilizzati in questo progetto e il loro funzionamento.

4.2.1 Tabs

Questo componente è stato utilizzato per visualizzare le tre ricette generate da Gemini e, nella Slate principale, per separare la foto scattata dalla lista di ingredienti rilevati dall'AI.



Figura 4.10: Tabs nel menu della fotocamera e ingredienti



Figura 4.11: Tabs nel menu delle ricette

Per creare un menu di Tabs dobbiamo avere due o più bottoni e dei relativi contenuti che verranno nascosti e mostrati a seconda del tab selezionato.

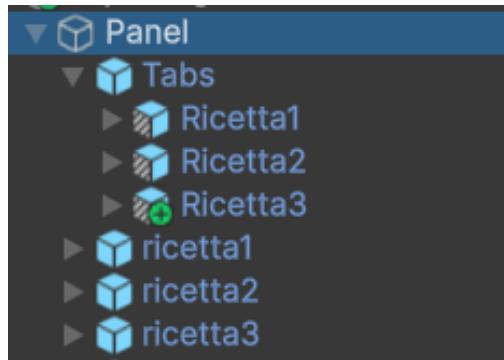


Figura 4.12: Gerarchia dei oggetti

Gli elementi denominati "Ricetta1", "Ricetta2" e "Ricetta3" sono i bottoni che rappresentano le singole tab, mentre "ricetta1", "ricetta2" e "ricetta3" sono i contenuti che verranno mostrati quando si seleziona il rispettivo tab. Per fare ciò dobbiamo avere il seguente componente nel GameObject Tabs:

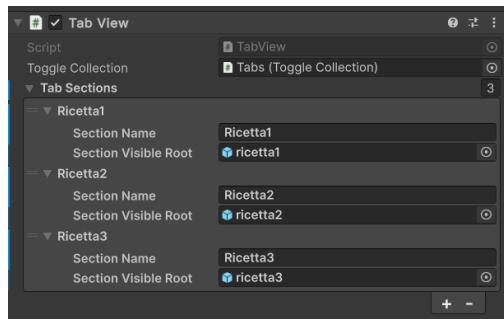


Figura 4.13: Componente per il funzionamento delle tabs

In "Toggle Collection" andiamo a trascinare il GameObject padre "Tabs", collezione dei tre bottoni, e in "Tab Selections" andiamo a impostare gli oggetti da visualizzare quando si premono i bottoni. Da notare che l'ordine in cui abbiamo messo i bottoni nel padre "Tabs" è lo stesso ordine in cui andremo a mettere gli oggetti da visualizzare nel componente, quindi il primo bottone mostrerà il primo oggetto e così via. [16]

4.2.2 Scroll list

La scroll list utilizzata in questo progetto è un prefab che si trova all'interno dei packages dell'MRTK3, precisamente in "MRTK UX Components" -> "Experimental" -> "Scollable". Se apriamo il prefab la gerarchia di GameObject è la seguente:



Figura 4.14: Gerarchia del prefab ScrollablePanel

In Content andiamo a mettere gli oggetti che vogliamo visualizzare e scorrere, Viewport, con i suoi componenti, ci permette di visualizzare solo una parte del contenuto se questo è più grande della finestra di visualizzazione, mentre nell'oggetto ScrollablePanel c'è il componente che ci permette di scorrere il contenuto e definisce la scroll list.

- **Content:** è consigliabile che abbia un componente come un Grid Layout Group o un Vertical Layout Group, in modo da avere una disposizione ordinata degli oggetti al suo interno.
- **Viewport:** è l'oggetto che contiene il contenuto visibile della scroll list, quindi è fondamentale che abbia un componente Rect Mask 2D, in modo da mascherare il contenuto che esce dalla finestra di visualizzazione.
- **Scrollable Panel:** è l'oggetto che contiene il componente Scroll Rect, a cui dobbiamo assegnare il viewport e le due barre di scorrimento se presenti. In questo modo possiamo scorrere il contenuto all'interno della finestra di visualizzazione. Oltre ad altri componenti c'è anche "Scrollable", dove dobbiamo impostare la "Scroll Rect" ed è un componente dell'MRTK3

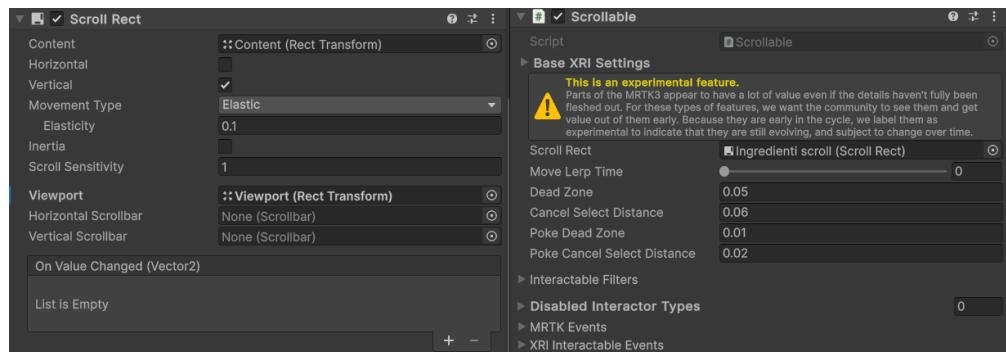


Figura 4.15: Componenti per Scrollable Panel

4.2.3 Slate

La Slate offre una finestra che consente la visualizzazione di contenuti 2D, come testo, immagini e elementi grafici come bottoni e menu, come una vera e propria finestra di una applicazione desktop. La Slate è un oggetto prefab che si trova all'interno dei packages dell'MRTK3, precisamente in "MRTK UX Components (Non-Canvas)" -> "Slates".

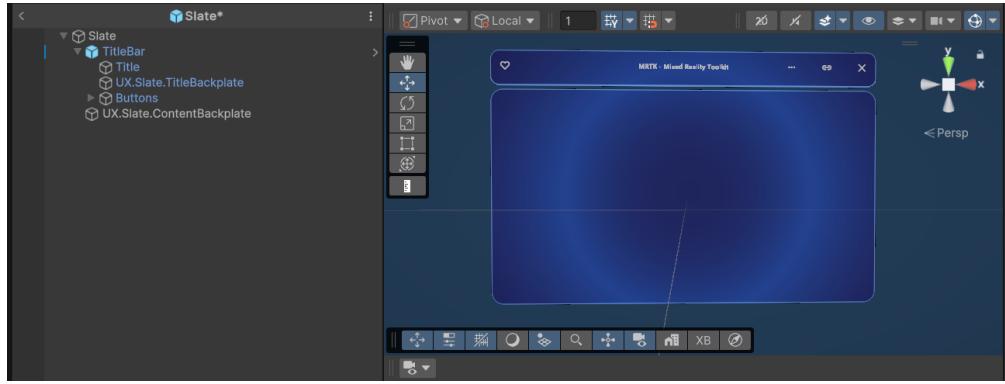


Figura 4.16: Gerarchia e UI della Slate

La Slate è composta principalmente da:

- **TitleBar:** è la barra in alto, ha due bottoni personalizzabili, altri due bottoni per chiudere la finestra e abilitare il follow e il titolo.
- **UX.Slate.ContentBackplate:** il contenitore padre dove andremo a mettere gli oggetti che vogliamo visualizzare all'interno della finestra

La Slate può essere trascinata dove si vuole prendendola dalla TitleBar, può essere chiusa e si può abilitare il follow, in modo che segua la posizione dell'utente. [17]

4.2.4 Near Menu

Il Near Menu offre un menù che può essere spostato o può seguire l'utente, in modo tale da non dare fastidio alle interazioni con gli altri elementi. Il Near Menu lo possiamo trovare sotto forma di Prefab all'interno dei packages dell'MRTK3, precisamente in "MRTK UX Components" -> "Near Menu".

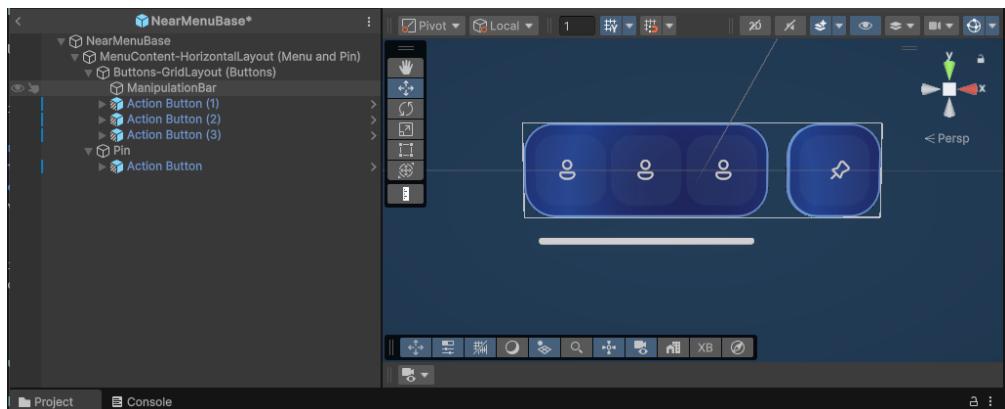


Figura 4.17: Gerarchia e UI del Near Menu

Il Near Menu è composto principalmente da:

- **Bottoni:** Si trovano all'interno di un Grid Layout e ne possiamo aggiungere quanti vogliamo, anche creando più righe e colonne.
- **PinButton:** è il bottone che permette di fissare il menù nella posizione in cui si trova, disattivando il follow

- **ManipulatorBar**: è la barra bianca in basso che permette di prendere e spostare il menù

Comportamenti principali del Near Menu:

- **Tag-along**: il menù segue l'utente ponendosi ad una distanza di 30-60cm
- **Pin**: usando il bottone del pin, situato nella parte più a destra, si fissa il menù nella posizione in cui si trova e si disattiva il follow
- **Grab and move**: essendo il menù un oggetto prendibile e spostabile, si può riposizionare in qualsiasi punto dello spazio prendendolo dalla barra bianca in basso

[18]

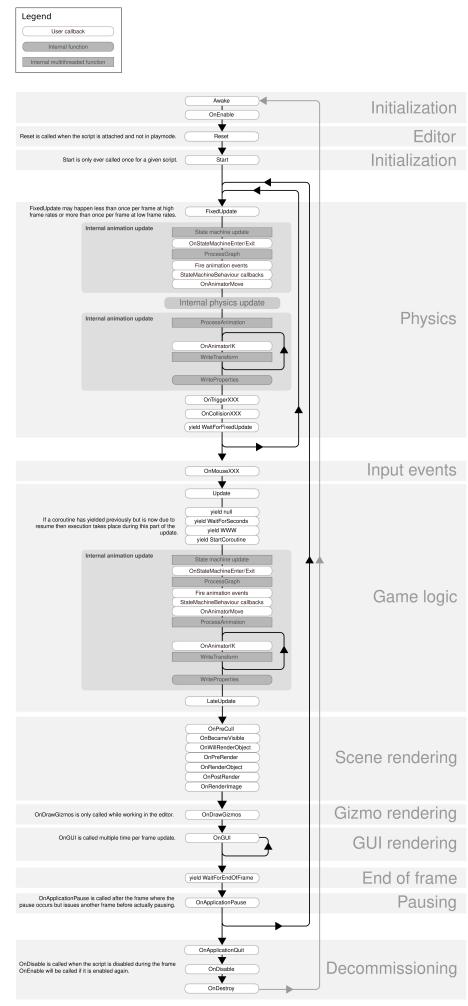
4.3 Programmazione in Unity con C#

4.3.1 MonoBehaviour

Il MonoBehaviour è la classe base da cui derivano tutti gli script in Unity. Quando creiamo uno script in Unity, automaticamente ereditiamo da MonoBehaviour, il che ci permette di utilizzare i metodi e le proprietà che Unity mette a disposizione per gestire il ciclo di vita degli oggetti della scena. [19]

I metodi principali utilizzati nel progetto sono:

- **Awake**: viene chiamato prima di Start quando il GameObject padre è attivo e si inizializza al caricamento della scena o passa da uno stato disattivo ad attivo [20]
- **OnEnable**: viene chiamato quando l'oggetto viene abilitato e diventa attivo [21]
- **Start**: viene chiamato una sola volta all'inizio del ciclo di vita dello script, utile per inizializzare variabili o impostare lo stato iniziale dell'oggetto [22]
- **Update**: viene chiamato una volta per frame, ed è il metodo principale per gestire la logica del GameObject [23]
- **OnDisable**: viene chiamato quando l'oggetto viene disabilitato [24]



4.3.2 Prefab

I Prefab consentono di configurare e creare un GameObject con tutti i suoi componenti e proprietà, così da riutilizzarlo e istanziarlo nella scena quante volte si vuole senza doverlo ricreare da capo ogni volta.

```
IngredientButtonRow row = Instantiate(ingredientButtonRow, transform);
row.initializeRow(ingredient);
```

Figura 4.18: Esempio di creazione di un Prefab

Nella figura sopra vediamo un esempio di creazione di un Prefab tramite la funzione **Instantiate**, che prende come parametro il Prefab da istanziare, nel nostro caso ingredientButtonRow, e il padre in cui posizionarlo (transform). Il Prefab viene poi assegnato ad un oggetto di tipo IngredientButtonRow in modo tale da poter accedere ai suoi metodi e inizializzarlo, cosa fatta nella riga successiva. [25]

4.3.3 IEnumarator e Coroutine

Le Coroutine sono un modo per eseguire codice in modo asincrono, permettendo di sospendere l'esecuzione di uno script per un certo periodo di tempo senza bloccare il thread principale. Questo è particolarmente utile per operazioni che richiedono del tempo. Per eseguire una Coroutine, si utilizza il metodo **StartCoroutine**, passando come parametro un metodo che restituisce un oggetto di tipo IEnumarator.

```
public void GenerateIngredients()
{
    GeminiAPI geminiAPI = new GeminiAPI();
    StartCoroutine(routine: geminiAPI.GetIngredientsPostRequest(photoPath, ingredientsScrollList));
}

| 1 usage
```

```
public IEnumerator GetIngredientsPostRequest(
    string imagePath,
    IngredientsScrollList ingredientsScrollList)
in class GeminiAPI
```

Figura 4.19: Esempio di utilizzo di Coroutine

4.3.4 Singleton

Il **Singleton** è un design pattern che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a essa. Ci sono diversi modi per implementare un Singleton in C#, tra cui l'implementazione normale e basica e quella thread-safe, utilizzata in questo progetto. La versione thread-safe garantisce che l'istanza sia creata in modo sicuro anche in un ambiente multi-thread, evitando problemi di concorrenza.

Tralasciando la versione normale, la versione thread-safe utilizza un blocco chiamato nel codice **padlock** per garantire che solo un thread alla volta possa accedere alla creazione dell'istanza.

```
public sealed class Singleton
{
    private static Singleton instance = null;
    private static readonly object padlock = new object();

    Singleton()
    {
    }

    public static Singleton Instance
    {
        get
        {
            lock (padlock)
            {
                if (instance == null)
                {
                    instance = new Singleton();
                }
                return instance;
            }
        }
    }
}
```

[26]

Capitolo 5

Utilizzo di Gemini come LLM e Spatial Understanding

In questo progetto è stato fatto uso di una LLM per il riconoscimento degli ingredienti e la successiva creazione di tre ricette. Fra tutte le LLM disponibili è stato scelto Gemini, la LLM sviluppata da Google, per la sua capacità di comprendere le immagini e l'API gratuita.

5.1 Cosa è una LLM

Un Large Language Model (LLM) è un modello di intelligenza artificiale in grado principalmente di riconoscere e generare testo. Gli LLM sono addestrati su grandi quantità di dati tramite l'uso del machine learning, nello specifico un particolare tipo di rete neurale chiamato Transformer. Per essere allenati in campi specifici, senza perdere il modello generato, viene utilizzato il fine-tuning, un processo che permette di specializzare il modello su un compito specifico. Le informazioni che le LLM restituiscono sono affidabili quanto lo sono i dati che gli sono stati forniti durante l'allenamento, quindi se nei dati sono presenti delle informazioni sbagliate queste verranno date per giuste alle domande poste all'LLM. A volte invece gli LLM possono creare delle false informazioni dal nulla, questo fenomeno si chiama allucinazione, ad esempio nel 2022 venne chiesto a ChatGPT di creare un articolo sulla compagnia Tesla, l'LLM creò l'articolo, dove però la maggior parte delle informazioni erano inventate.[27]

Gli usi di una LLM sono vari, tra cui:

- **Generazione di testo:** Creazione di email, blog, articoli e molto altro
- **Riassumere testo:** Riassumere lunghi articoli, notizie ma anche report aziendali
- **Generazione di codice:** Generare codice per la maggior parte di linguaggi di programmazione per aiutare il programmatore in compiti ripetitivi
- **Traduzione:** Tradurre i testi in varie lingue, fornendo una traduzione accurata che prende nota anche del contesto delle parole
- **Accessibilità:** Aiutare le persone con disabilità a interagire con l'ambiente circostante, ad esempio generando descrizioni di immagini per persone non vedenti, tecnologia ancora in sviluppo con la recentissima versione di Gemini Live

[28] [29] [30]

5.2 Prompting

I prompt sono utilizzati per interagire con le LLM, di solito sono delle domande, e in generale delle richieste testuali fatte dall'utente. Per ottenere le risposte volute nel prompt, oltre a porre la domanda o l'azione da fare, è necessario fornire, se necessario, il contesto, e se si vuole essere più specifici anche al formato della risposta, ad esempio un elenco puntato oppure un testo di poche righe. Da tenere a mente che però le LLM potrebbero non del tutto seguire le istruzioni fornite, soprattutto se si fanno richieste troppo specifiche come il numero di caratteri. [31] [32]

5.3 Utilizzo di Gemini per Spatial Understanding

Gemini offre la possibilità di utilizzare le immagini come input, e ottenere, oltre che a una descrizione dell'immagine anche informazioni più specifiche e interessanti. Lo Spatial Understanding è una funzionalità che ci permette di farci dure dall'LLM le posizioni in coordinate e i nomi degli oggetti presenti nell'immagine. Si può provare questa funzionalità nel sito di Gemini Studio nella sezione "Build" sotto il nome di "Spatial Understanding". Nel nostro caso si darà alla LLM l'immagine di un tavolo o in generale di un piano in cui sono poggiati gli ingredienti e si chiederà all'LLM di restituire i nomi degli ingredienti e le relative posizioni.

5.4 Utilizzo di Gemini come LLM

Una volta ottenuti i nomi degli ingredienti, questi verranno dati a Gemini, a cui sarà chiesto di generare tre ricette utilizzando solo gli ingredienti forniti.

Capitolo 6

Progettazione

6.1 Casi D'uso

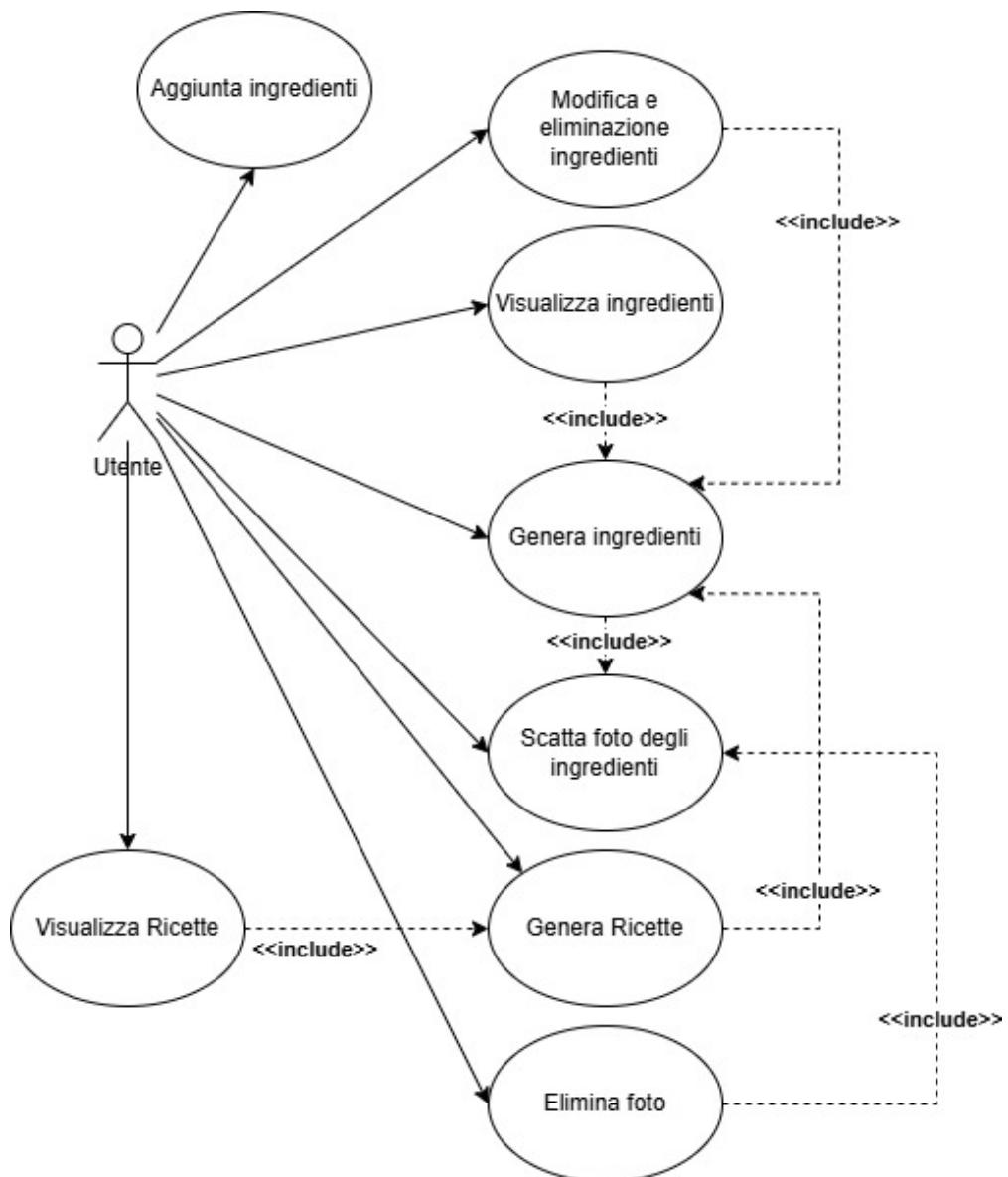


Figura 6.1

Lo use case in fig.6.1 mostra lo scenario in cui l'utente ha già aperto entrambe le finestre dell'applicazione, quella per la fotocamera e generazione degli ingredienti e quella per la visualizzazione delle ricette. L'utente potrà scattare una foto ed eliminarla, generare gli ingredienti solo se la foto è stata scattata, aggiungere un ingrediente indipendentemente dalla foto, modificare un ingrediente già presente, generare tre ricette con gli ingredienti presenti e visualizzare le ricette generate.

6.2 Architettura

L'intera applicazione è standalone sul visore, quindi non necessita di un server per il funzionamento. Le uniche operazioni e calcoli che avvengono all'esterno del visore sono quelle che coinvolgono Google Gemini, che avvengono grazie a delle chiamate API in **POST**.

6.3 Interfaccia grafica

L'interfaccia è composta da due finestre, composte poi da più tabs, e un menu sempre visibile. Ogni finestra ha in alto una barra in cui è presente il nome della finestra e due bottoni posizionati a destra. Il bottone con il simbolo della "X" permette di chiudere la finestra, mentre quello con il simbolo della catena serve a bloccare la finestra nella posizione dove si trova o farsi seguire.

6.3.1 Menu

Il menu che vediamo nella figura 6.2 è il menu principale che permette di mostrare e nascondere le finestre dell'applicazione, è composto da tre bottoni e una barra bianca sotto. Andando da sinistra verso destra troviamo il bottone per aprire la finestra per la fotocamera e generazione degli ingredienti, il bottone per vedere le ricette generate e il bottone per fare il pin della barra, in modo da tenerla ancorata in nella posizione prefissata. Infatti di default la barra seguirà l'utente posizionandosi in basso in modo da non dar fastidio alle azioni con le altre finestre, sarà possibile spostarla utilizzando la barra bianca in basso.

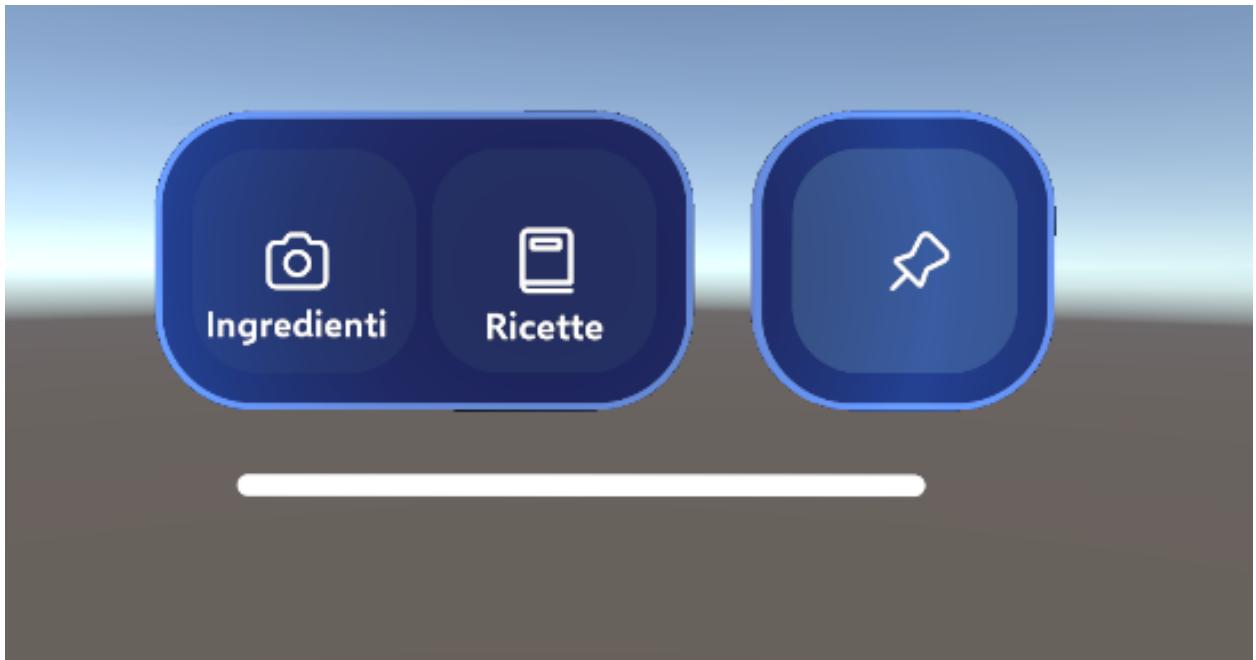


Figura 6.2: Menu principale dell'applicazione

6.3.2 Finestra per scatto della foto e generazione degli ingredienti

Questa finestra è composta da due tabs, una per scattare la foto e visualizzarla come in fig.6.3, l'altra per visualizzare gli ingredienti generati e apportare modifiche come in fig.6.4. Per cambiare da una tab all'altra sono presenti due buttoni in alto con i nomi "Fotocamera" e "Ingredienti".

Tab per la fotocamera

Al centro troviamo la foto scattata, mentre a destra si trova un menu che ci permette di scattare la foto, eliminare la foto già scattata e generare gli ingredienti.



Figura 6.3: tab per la visualizzazione e scatto della foto

Tab per gli ingredienti

Nella parte centrale si trova la scrollist con gli ingredienti, a destra si trova il menu che permette di aggiungere un nuovo ingrediente o creare tre ricette con gli ingredienti presenti. Per la modifica di un ingrediente è necessario cliccare su di esso nella lista, in questo modo si aprirà una finestra di modifica come in fig.6.6.

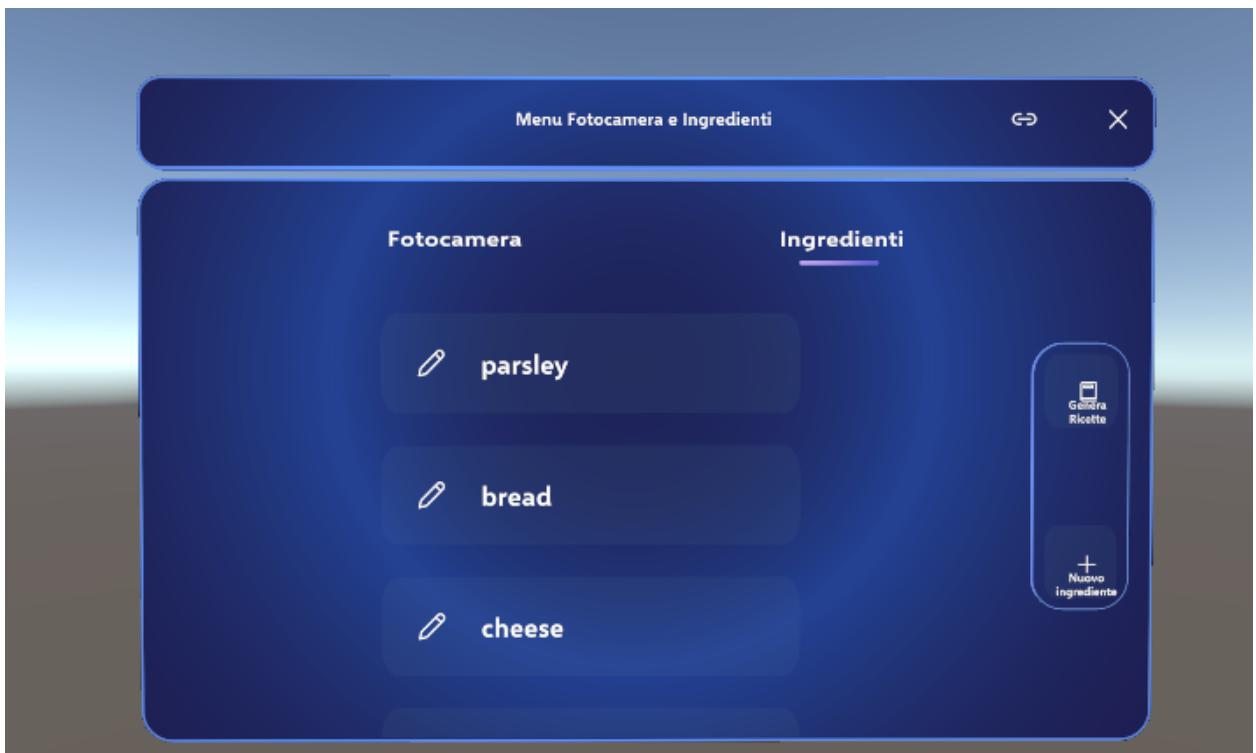


Figura 6.4: Tab per la visualizzazione, inserimento e modifica degli ingredienti

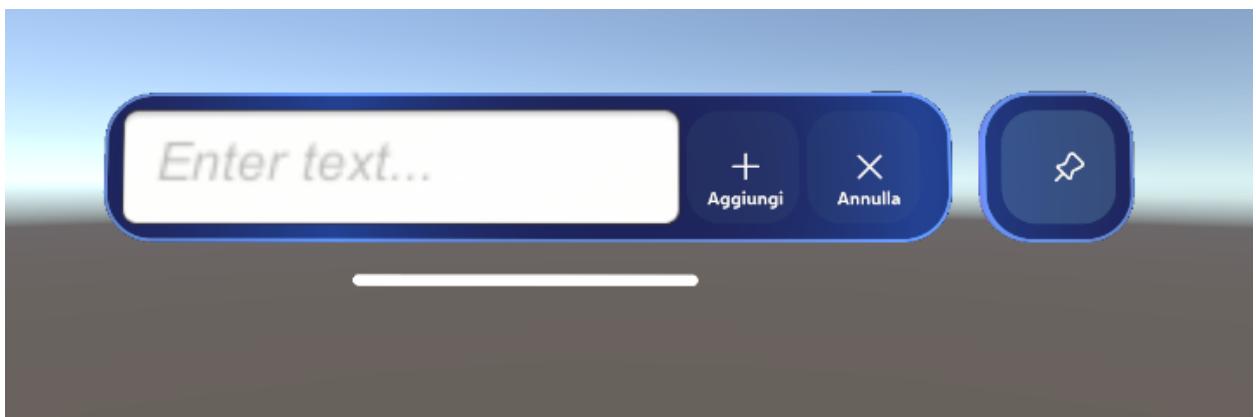


Figura 6.5: Aggiunta di un ingrediente



Figura 6.6: Modifica di un ingrediente

6.3.3 Finestra per visualizzazione delle ricette

In alto troviamo un menu con tre button che ci permettono di cambiare tab, ogni tab è composta da una scrollist con le ricette generate. A destra troviamo un bottone che permette di generare nuove ricette.



Figura 6.7: Interfaccia per la visualizzazione delle tre ricette

6.4 Persistenza dei dati

I dati vengono salvati in due file JSON, uno che contiene gli ingredienti e l'altro le ricette generate, la foto scattata invece viene salvata nella stessa directory dei file JSON sotto un nome specifico. I nomi dei file si trovano all'interno di un file config, nella cartella StreamingAssets, accessibile e leggibile come indicato in fig.6.8

```
string json = File.ReadAllText( path: Application.streamingAssetsPath + "/config.json");
```

Figura 6.8: Codice per leggere il file di config

Successivamente, grazie alla classe generica **FileHandler** possiamo serializzare e deserializzare i file JSON salvati, per modificarli o crearli. I file JSON e l'immagine vengono salvati nel **persistentDataPath** come indicato in fig.6.9.

```
string filePath = Path.Combine(Application.persistentDataPath, jsonNameFile);
```

Figura 6.9: Parte di codice che genera il path in cui salvare il file JSON

Capitolo 7

Implementazione dell'applicazione

7.1 Introduzione

In questa sezione andremo a descrivere l'implementazione andando a dare uno sguardo più profondo al codice e successivamente anche alle problematiche riscontrate durante lo sviluppo con le relative soluzioni.

7.2 Gemini API

7.2.1 Structured output

Lo Structured Output permette di farsi restituire da Gemini una risposta standardizzata, in questo caso un JSON, predefinita sotto forma di uno schema. Lo **Schema** supporta array, tipi e oggetti dentro i quali è possibile definire i tipi dei dati e i loro nomi. [33] I tipi nello **Schema** devono essere uno degli OpenAPI Data Types, oppure un'unione di questi tipi (utilizzando `anyOf`). Per ciascun tipo, è valido solo un determinato sottoinsieme di campi. Nella seguente lista viene riportata la corrispondenza tra ciascun tipo e il relativo insieme di campi validi:

- **string**: enum, format, nullable
- **integer**: format, minimum, maximum, enum, nullable
- **number**: format, minimum, maximum, enum, nullable
- **boolean**: nullable
- **array**: minItems, maxItems, items, nullable
- **object**: properties, required, propertyOrdering, nullable

```
{  
    "type": enum (Type),  
    "format": string,  
    "description": string,  
    "nullable": boolean,  
    "enum": [  
        string  
    ],  
    "maxItems": integer,  
    "minItems": integer,  
    "properties": {  
        string: {  
            object (Schema)  
        },  
        ...  
    },  
    "required": [  
        string  
    ],  
    "propertyOrdering": [  
        string  
    ],  
    "items": {  
        object (Schema)  
    }  
}
```

Figura 7.1: Pseudo-JSON che rappresenta tutti i campi dello Schema

```

{
  "type": "string", "enum": [ "a", "b", "c" ] }

{
  "type": "string", "format": "date-time" }

{
  "type": "integer", "format": "int64" }

{
  "type": "number", "format": "double" }

{
  "type": "boolean" }

{
  "type": "array", "minItems": 3, "maxItems": 3, "items": { "type": ... } }

{
  "type": "object",
  "properties": {
    "a": { "type": ... },
    "b": { "type": ... },
    "c": { "type": ... }
  },
  "nullable": true,
  "required": [ "c" ],
  "propertyOrdering": [ "c", "b", "a" ]
}

```

Figura 7.2: Schemas di esempio

7.2.2 Richieste POST

Le richieste POST vengono fatte all'interno della classe **GeminiAPI**, che è anche la delegata al salvataggio dei dati ricevuti da Gemini in dei file JSON.

```
public IEnumerator GetIngredientsPostRequest(string imagePath, IngredientsScrollList ingredientsScrollList)
```

Figura 7.3: Funzione per generare gli ingredienti

```
public IEnumerator GetRecipesPostRequest(MenuRicette menuRicette, List<Ingredient> ingredients)
```

Figura 7.4: Funzione per generare le ricette

Generazione degli ingredienti

Per generare gli ingredienti useremo la funzione della fig.7.3. Il payload della POST richiede:

- Una immagine in formato **Base64**
- L'API Key e l'API url di Gemini
- Lo schema che definisce i campi che Gemini deve restituire

Per convertire l'immagine in formato **Base64** useremo la funzione **Convert.ToString()** di C#.

```
byte[] imageBytes = System.IO.File.ReadAllBytes(imagePath);
string base64Image = System.Convert.ToString(imageBytes);
```

Figura 7.5: Lettura e conversione dell'immagine in Base64

```
string url = config.getApiUrl()+config.getApiKey();
```

Figura 7.6: Creazione dell'url per la richiesta POST

L'API Key e l'url vengono letti dal file config come in fig.7.6, che poi vengono messi all'interno del payload, insieme allo schema mostrato in fig.7.7 e all'immagine in formato Base64 mostrato in fig.7.8.

```
""generationConfig"": {{
  ""response_mime_type"": ""application/json"",
  ""response_schema"": {{
    ""type"": ""OBJECT"",
    ""properties"": {{
      ""ingredients"": {{
        ""type"": ""ARRAY"",
        ""items"": {{
          ""type"": ""OBJECT"",
          ""properties"": {{
            ""box_2d"": {""type"": ""array"", ""minItems"": 4, ""maxItems"": 4, ""items"": {{"type"": ""number""}}}},
            ""ingredient"": {""type"": ""STRING""}}
          }}}
        }}}
      }}}
    }}
```

Figura 7.7: Schema per la generazione degli ingredienti

```
{{
  ""inline_data"": {{
    ""mime_type"": ""image/jpeg"",
    ""data"": "{base64Image}"
  }}
}}
```

Figura 7.8: Inserimento dell'immagine nel payload della richiesta POST

Infine viene effettuata la POST creando una nuova istanza della classe **UnityWebRequest**, convertendo il payload string in un array di byte per poi invocare il metodo **SendWebRequest()** per inviare la richiesta. Se la richiesta ha successo, Gemini risponderà con un JSON che contiene gli ingredienti, che verranno salvati in un file JSON.

```
var www = new UnityWebRequest(url, method: "POST");
byte[] bodyRaw = Encoding.UTF8.GetBytes(body);
www.uploadHandler = (UploadHandler)new UploadHandlerRaw(bodyRaw);
www.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();
www.SetRequestHeader(name: "Content-Type", value: "application/json");

// Send the request
yield return www.SendWebRequest();
```

Figura 7.9: Codice per effettuare la richiesta POST

Generazione delle ricette

Per generare gli ingredienti useremo la funzione della fig.7.4. In questo caso non ci servirà un'immagine ma la lista degli ingredienti, che verrà letta in un for e concatenata sotto un'unica stringa. La creazione dell'url e dell'API Key avviene come per la generazione degli ingredienti in fig.7.6, ma lo Schema sarà diverso, come mostrato in fig.7.10.

```

    "generationConfig": {
      "response_mime_type": "application/json",
      "response_schema": {
        "type": "OBJECT",
        "properties": {
          "recipes": {
            "type": "ARRAY",
            "items": {
              "type": "OBJECT",
              "properties": {
                "recipe": { "type": "STRING" }
              }
            }
          }
        }
      }
    }
  
```

Figura 7.10: Schema per la generazione delle ricette

Come possiamo facilmente notare lo schema è molto più semplice e richiede soltanto la restituzione di un array di oggetti, ognuno dei quali contiene una ricetta di tipo stringa. Invece la richiesta testuale fatta è la seguente:

```

create three recipes only with the following ingredients: {ingredientsConcat}.
Write it with tags compatible with TextMesh Pro, make the recipes titles colored
in red, make the number of the instruction list colored in red, use bold in
subtitles
  
```

dove `ingredientsConcat` è la stringa che contiene tutti gli ingredienti separati da virgola. L'invio della richiesta POST avviene come per la generazione degli ingredienti in fig.7.9.

7.3 File Handler

la classe **FileHandler** è responsabile della lettura e scrittura e creazione dei file JSON. La particolarità è quella di essere una classe generica, in modo da poter serializzare tramite apposite classi serializzabili qualsiasi JSON.

7.3.1 Classi Serializzabili

Sono presenti quattro classi serializzabili, in particolare:

- **ConfigJSON**: Serve a leggere il file di configurazione, dove sono salvati l'API Key, l'API URL di Gemini e i nomi dei file JSON da salvare e quello della foto.
- **GeminiResponseJSON**: Serve a leggere la risposta di Gemini che contiene gli ingredienti o le ricette e prendere i campi che ci interessano, in particolare il campo `text` tramite apposita funzione `GetText()`.
- **ListOfIngredientsJSON**: Utilizzata per salvare e leggere la lista degli ingredienti e ad assegnargli un ID univoco, che viene generato automaticamente quando si aggiunge un nuovo ingrediente.

- **ListOfRecipesJSON**: Utilizzata per salvare e leggere la lista delle ricette, che contiene un array di tipo **RecipesJSON**, ognuna delle quali contiene una stringa.

La separazione delle classi serializzabili da quelle del model permette di avere un codice più pulito e facilmente mantenibile, in quanto ogni classe ha una responsabilità ben definita.

7.3.2 Metodi

Sono presenti tre metodi:

- `SaveJson(string jsonNameFile, T item)`: Salva un oggetto di tipo generico **T** nel file json con il nome passato tramite **jsonNameFile**. Il file viene creato se non esiste, altrimenti viene sovrascritto. Codice disponibile in fig.7.11.
- `public T LoadJson(string jsonNameFile)`: deserializza un file JSON con il nome passato tramite **jsonNameFile** e restituisce un oggetto di tipo generico **T**. Se il file non esiste, ne viene creato uno vuoto e viene restituito un oggetto di tipo generico **T** vuoto. Codice disponibile in fig.7.12.
- `CreateBlankFile(string filePath)`: Crea un file vuoto con il nome passato tramite **filePath**. Se il file esiste già, non viene creato nulla.

```
public void SaveJson(string jsonNameFile, T item)
{
    string jsonNameFileFormatted = string.Format(jsonNameFile, Time.time);

    string filePath = Path.Combine(Application.persistentDataPath, jsonNameFileFormatted);

    if (File.Exists(filePath))
    {
        File.Delete(filePath);
    }

    File.WriteAllText(filePath, contents: JsonConvert.SerializeObject(item));
}
```

Figura 7.11: Codice per effettuare il salvataggio del JSON

```
0 usages
public T LoadJson(string jsonNameFile)
{
    string filePath = Path.Combine(Application.persistentDataPath, jsonNameFile);
    CreateBlankFile(filePath);
    string json = File.ReadAllText(filePath);
    return JsonConvert.DeserializeObject<T>(json);
}

0+1 usage
private void CreateBlankFile(string filePath)
{
    if (!File.Exists(filePath))
    {
        File.WriteAllText(filePath, contents: null);
    }
}
```

Figura 7.12: Codice per effettuare la deserializzazione del JSON

7.4 Scatto della foto

Per scattare la foto usiamo la classe **PhotoShooter**, che utilizza la libreria **UnityEngine.Windows.WebCam** di Unity per accedere alla fotocamera del dispositivo. La classe è responsabile della gestione della fotocamera, dello scatto della foto e del salvataggio dell'immagine in un file ed è implementata tramite un singleton per garantire che ci sia una sola istanza della classe durante l'esecuzione dell'applicazione. Sono presenti due funzioni principali:

- **TakePhoto(string photoPath)**: controlla se la foto non esiste già, altrimenti la cancella e crea un processo asincrono per scattare e salvare la foto. Fig.7.13
- **DeletePhoto(string photoPath)**: cancella la foto scattata, se esiste. Fig.7.14

```
* 0+ asset usages  0+ 1 usage
public void TakePhoto(string photoPath)
{
    if (System.IO.File.Exists(photoPath))
    {
        System.IO.File.Delete(photoPath);
    }

    PhotoCapture.CreateAsync( showHolograms: false, OnPhotoCaptureCreated);
}
```

Figura 7.13: Inizio funzione per lo scatto della foto

```
public void DeletePhoto(string photoPath)
{
    if (System.IO.File.Exists(photoPath))
    {
        System.IO.File.Delete(photoPath);
    }
}
```

Figura 7.14: Codice per l'eliminazione della foto

```
private void OnPhotoCaptureCreated(PhotoCapture captureObject)
{
    photoCaptureObject = captureObject;

    Resolution cameraResolution = PhotoCapture.SupportedResolutions.OrderByDescending((res:Resolution) => res.width * res.height).First();

    CameraParameters c = new CameraParameters();
    c.hologramOpacity = 0.0f;
    c.cameraResolutionWidth = cameraResolution.width;
    c.cameraResolutionHeight = cameraResolution.height;
    c.pixelFormat = CapturePixelFormat.BGRA32;

    captureObject.StartPhotoModeAsync(c, OnPhotoModeStarted);
}
```

Figura 7.15: Inizializzazione della fotocamera

Nella figura 7.15 il metodo **OnPhotoCaptureCreated** viene inizializzata la fotocamera con la massima risoluzione possibile e iniziata la cattura della foto in maniera asincrona utilizzando il metodo **OnPhotoModeStarted**

```
private void OnPhotoModeStarted(PhotoCapture.PhotoCaptureResult result)
{
    if (result.success)
    {
        string filename = string.Format(config.getimageName(), Time.time);
        string filePath = System.IO.Path.Combine(Application.persistentDataPath, filename);

        if(config.getDebug()) Debug.Log(filePath);

        photoCaptureObject.TakePhotoAsync(filePath, PhotoCaptureFileOutputFormat.JPG, OnCapturedPhotoToDisk);
    }
    else
    {
        if(config.getDebug()) Debug.LogError("Unable to start photo mode!");
    }
}
```

Figura 7.16: Scatto della foto

Nella figura 7.16 il metodo **OnPhotoModeStarted** viene chiamato in maniera asincrona e tramite

```
photoCaptureObject.TakePhotoAsync(filePath, PhotoCaptureFileOutputFormat.JPG,
    OnCapturedPhotoToDisk);
```

viene scattata la foto e utilizzato il metodo **OnCapturedPhotoToDisk** per salvare la foto su disco al path precedentemente definito dal **persistentDataPath**.

```
private void OnCapturedPhotoToDisk(PhotoCapture.PhotoCaptureResult result)
{
    if (result.success)
    {
        if(config.getDebug()) Debug.Log("Saved Photo to disk!");
        photoCaptureObject.StopPhotoModeAsync(OnStoppedPhotoMode);
    }
    else
    {
        if(config.getDebug()) Debug.LogError("Failed to save Photo to disk");
    }
}
```

Figura 7.17: Salvataggio della foto su disco

In figura 7.17 il metodo **OnCapturedPhotoToDisk** è utilizzato per salvare la foto su disco, se l'operazione è compiuta con successo viene chiamato il metodo **OnStoppedPhotoMode** per terminare la modalità foto e rilasciare le risorse della fotocamera.

```
private void OnStoppedPhotoMode(PhotoCapture.PhotoCaptureResult result)
{
    photoCaptureObject.Dispose();
    photoCaptureObject = null;
}
```

Figura 7.18: Procedura per terminare lo scatto della foto

In figura 7.18 il metodo **OnStoppedPhotoMode** viene chiamato per terminare la modalità foto e rilasciare le risorse della fotocamera, in particolare tramite `photoCaptureObject.Dispose()` il cui metodo `Dispose()` viene chiamato per arrestare l'istanza di **PhotoCapture**.

7.5 Lista ingredienti

Possiamo definire la lista degli ingredienti come uno dei più complessi elementi presenti nell'applicazione, in quanto è composta da dei prefab che vengono visualizzati in una lista, ognuno dei quali contiene il nome dell'ingrediente modificabile e eliminabile.

7.5.1 Prefab

Utilizzando il prefab **IngredientButtonRow** possiamo visualizzare gli ingredienti all'interno della lista, riusciendolo a generarlo quanti sono gli ingredienti presenti nel file JSON.

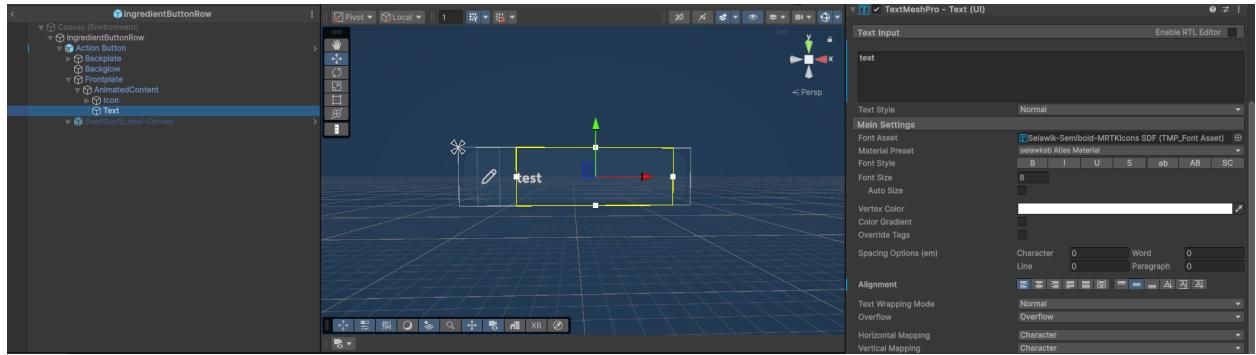


Figura 7.19: Gerarchia del prefab IngredientButtonRow

Il prefab è un bottone al cui interno è presente un **TextMeshPro** per il nome dell'ingrediente, se cliccato verrà aperta la schermata di modifica dell'ingrediente. Al prefab è assegnato uno script **IngredientButtonRow** che contiene i metodi per l'inizializzazione del prefab e il funzionamento generale.

```

✿ 1 asset usage  ⚡ 9 usages  🔍 1 exposing API
public class IngredientButtonRow : MonoBehaviour
{
    public TextMeshProUGUI textField; ✿ Unchanged
    private Ingredient ingredient;

    ✿ Frequently called ⚡ 3 usages
    public void initializeRow(Ingredient ingredient)
    {
        this.ingredient = ingredient;
        textField.text = ingredient.getName();
    }

    ✿ 0+ asset usages
    public void OpenModifyPanel()
    {
        IngredientsScrollList scrollList = transform.parent.GetComponent<IngredientsScrollList>();
        LOG.Instance.addLog( text: scrollList.ToString() );
        ModifyIngredient modifyIngredient = scrollList.getModifyIngredient();
        if (modifyIngredient != null)
        {
            Debug.LogWarning("Panel found");

            LOG.Instance.addLog( text: "OpenModifyPanel()" );
            LOG.Instance.addLog( text: "OpenModifyPanel() ==>" + this.ingredient.getName() );
            modifyIngredient.OpenPanel(ingredient, ingredientButtonRow: this);
        }
        else
        {
            Debug.LogWarning("Panel not found");
        }
    }
}

```

Figura 7.20: Classe IngredientButtonRow

La variabile **TextField** è di tipo **TextMeshProUGUI** e rappresenta il campo di testo del nome dell'ingrediente, componente del prefab, e visto che è serializzabile e dichiarata come pubblica questa verrà mostrata nell'Inspector di Unity, permettendo di assegnare il campo di testo direttamente dal prefab con un drag and drop.

Il metodo **InitializeRow(Ingredient ingredient)** viene chiamato per inizializzare il prefab con i dati dell'ingrediente passato come parametro, in particolare viene assegnato il nome dell'ingrediente al campo di testo **TextField** e salvato nella variabile **ingredient**.

7.5.2 Creazione del prefab all'interno della lista

La scroll list, come visto in fig.4.14, è formata da diversi oggetti nella gerarchia, tra cui il **Content**, ossia l'ultimo elemento della gerarchia ma il padre dei prefab generati. Infatti proprio questo oggetto è delegato alla loro creazione, modifica e cancellazione.

```

public class IngredientsScrollList : MonoBehaviour
{
    public IngredientButtonRow ingredientButtonRow; // ingredientButtonRow (IngredientButtonRow)
    public MenuRicette menuRicette; // Changed in 1 asset
    public ModifyIngredient modifyIngredientGameObject; // Changed in 1 asset
    private Config config = null;

    private List<Ingredient> ingredients = new List<Ingredient>();
    private int lastIndex = 0;

    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        config = Config.Instance;
        FileHandler<ListOfIngredientsJSON> fileHandler = new FileHandler<ListOfIngredientsJSON>();
        if (fileHandler.LoadJson(config.getIngredientsJsonName()) is not null)
        {
            generateRows(fileHandler.LoadJson(config.getIngredientsJsonName()).getIngredients());
        }
    }
}

```

Figura 7.21: Dichiarazione delle variabili e metodo Start

Fra le diverse variabili dichiarate troviamo **ingredients**, una lista di tipo **Ingredient** che andremo a popolare alla chiamata del metodo **Start**.

Nel metodo **Start()** tramite la classe **FileHandler** andremo a leggere il file JSON degli ingredienti, se il metodo **LoadJson()** non è nulla allora chiamiamo la funzione **getIngredients()** implementata nella classe per serializzare il JSON, che risulta in una lista di ingredienti. Questa lista viene passata al metodo **generateRows()** che si occupa di generare i prefab per ogni ingrediente presente nella lista **ingredients** e di aggiornare la variabile **ingredients** per poi generare i prefab all'interno della lista.

```

public void generateRows(List<Ingredient> ingredients)
{
    this.ingredients = ingredients;
    foreach (Transform child in transform)
    {
        Destroy(child.gameObject);
    }

    lastIndex = 0;
    foreach (Ingredient ingredient in this.ingredients)
    {
        if(config.getDebug()) Debug.Log(message: ingredient.getID());
        if (ingredient.getID() > lastIndex) lastIndex = ingredient.getID();
        IngredientButtonRow row = Instantiate(ingredientButtonRow, transform);
        row.initializeRow(ingredient);
    }
}

```

Figura 7.22: Metodo per generare i prefab

Il metodo `generateRows()` si occupa di generare i prefab per ogni ingrediente presente nella lista `ingredients`. Prima di tutto aggiorna la variabile `ingredients` con la lista degli ingredienti passata come parametro, poi se al suo interno ha dei figli vengono distrutti tramite un ciclo `foreach`, successivamente, sempre con un ciclo `foreach` per ogni ingrediente presente nella lista `ingredients` viene generato un prefab. La creazione del prefab viene spiegata al paragrafo 4.3.2, aggiungiamo solo che il prefab viene generato tramite il metodo `Instantiate()` di Unity, che prende come parametro il prefab da generare e la posizione in cui generarlo, in questo caso la posizione è quella del `Content` della lista, per questo per semplicità è stato passato direttamente `transform`. Il prefab viene poi assegnato alla variabile `ingredientButtonRow` e viene chiamato il metodo `initializeRow()` passandogli come parametro l'oggetto di tipo `Ingredient`.

7.5.3 Aggiunta, modifica e cancellazione

Le funzionalità di aggiunta, modifica e cancellazione degli ingredienti sono delegate sempre alla classe `IngredientsScrollView`, tramite gli appositi metodi `addIngredient(string ingredientName)`, `deleteIngredient(Ingredient ingredient, IngredientButtonRow ingredientButtonRow)` e `modifyIngredient(Ingredient oldIngredient, Ingredient newIngredient, IngredientButtonRow ingredientButtonRow)`. Inoltre è presente un metodo `saveList()` che si occupa di salvare la lista degli ingredienti nel file JSON, chiamato alla fine di ogni operazione di modifica della lista.

```

public void addIngredient(string ingredientName)
{
    Ingredient ingredient = new Ingredient(ingredientName, ID: lastIndex + 1);
    ingredients.Add(ingredient);
    lastIndex++;
    IngredientButtonRow row = Instantiate(ingredientButtonRow, transform);
    row.initializeRow(ingredient);
    saveList();
}

```

Figura 7.23: Metodo per aggiungere un ingrediente

L'Inserimento di un nuovo ingrediente avviene tramite il metodo `addIngredient(string ingredientName)`, come in fig.7.23, che prende come parametro il nome dell'ingrediente da aggiungere. Viene creato un nuovo oggetto di tipo **Ingredient** con il nome passato come parametro e un ID generato automaticamente, che viene incrementato di uno rispetto all'ultimo ID presente nella lista degli ingredienti. Successivamente viene aggiunto alla lista **ingredients**, viene aggiornata graficamente la lista e poi viene salvato il JSON.

```

public void modifyIngredient(Ingredient oldIngredient, Ingredient newIngredient, IngredientButtonRow ingredientButtonRow)
{
    int index = ingredients.IndexOf(oldIngredient);
    if (index != -1)
    {
        ingredients[index] = newIngredient;
        ingredientButtonRow.initializeRow(newIngredient);
        saveList();
    }
    else
    {
        Debug.LogError("Ingredient not found in the list.");
    }
}

```

Figura 7.24: Metodo per modificare un ingrediente

La modifica di un ingrediente avviene tramite il metodo `modifyIngredient(Ingredient oldIngredient, Ingredient newIngredient, IngredientButtonRow ingredientButtonRow)`, come in fig.7.24, che prende come parametri l'ingrediente da modificare, il nuovo ingrediente e il prefab dell'ingrediente da modificare. Viene cercato l'ingrediente da modificare nella lista **ingredients** e se trovato viene sostituito con il nuovo ingrediente, aggiornando il prefab con il nuovo nome e salvando la lista degli ingredienti nel file JSON.

```

public void deleteIngredient(Ingredient ingredient, IngredientButtonRow ingredientButtonRow)
{
    ingredients.Remove(ingredient);
    Destroy(ingredientButtonRow.gameObject);
    saveList();
}

```

Figura 7.25: Metodo per cancellare un ingrediente

L'eliminazione di un ingrediente avviene tramite il metodo `deleteIngredient(Ingredient ingredient, IngredientButtonRow ingredientButtonRow)`, come in fig.7.25, che prende come parametri l'ingrediente da eliminare e il prefab dell'ingrediente da eliminare. Tramite il metodo `Remove()` della lista `ingredients` viene eliminato l'ingrediente passato come parametro, successivamente viene distrutto il prefab dell'ingrediente e infine viene salvata la lista degli ingredienti nel file JSON.

```
private void saveList()
{
    FileHandler<ListOfIngredientsJSON> fileHandler = new FileHandler<ListOfIngredientsJSON>();
    ListOfIngredientsJSON listOfIngredients = new ListOfIngredientsJSON();
    listOfIngredients.setIngredients(ingredients);
    fileHandler.SaveJson(config.getIngredientsJsonName(), listOfIngredients);
}
```

Figura 7.26: Metodo per salvare la lista degli ingredienti

7.6 Tastiera virtuale

La tastiera virtuale viene utilizzata per l'inserimento e la modifica degli ingredienti. È implementata tramite la classe **KeyboardManager** che fornisce i metodi per la sua apertura, chiusura e gestione degli eventi di input.

```
public class KeyboardManager
{
    [SerializeField]
    private TouchScreenKeyboard keyboard;

    ⚡ 2 usages
    public void OpenSystemKeyboard()
    {
        keyboard = TouchScreenKeyboard.Open(
            text: "",
            TouchScreenKeyboardType.Default,
            autocorrection: false,
            multiline: false,
            secure: false,
            alert: false
        );
    }

    ⚡ 1 usage
    public void CloseSystemKeyboard()
    {
        if (keyboard != null)
        {
            keyboard.active = false;
            keyboard = null;
        }
    }
}
```

Figura 7.27: Metodi per aprire e chiudere la tastiera

```
⌚ Frequently called ⚖ 2 usages
public string GetKeyboardText()
{
    if (keyboard != null)
    {
        return keyboard.text;
    }
    return string.Empty;
}

⌚ 1 usage
public void SetKeyboardText(string text)
{
    if (keyboard != null)
    {
        keyboard.text = text;
    }
}
```

Figura 7.28: Metodi per ottenere e impostare il testo della tastiera

7.7 Problematiche e soluzioni

In questa sezione andremo a descrivere le problematiche riscontrate durante lo sviluppo dell'applicazione e le relative soluzioni adottate.

7.7.1 Il file di config non veniva letto

il file di config, dopo la compilazione e l'utilizzo standalone, non veniva letto correttamente, in quanto il percorso del file era errato. Per risolvere il problema è stato necessario spostare il file in una cartella chiamata **StreamingAssets** all'interno della cartella **Assets** del progetto Unity. In questo modo il file viene incluso nella build e il percorso viene preso dal metodo **Application.streamingAssetsPath** di Unity, che restituisce il percorso corretto della cartella in cui si trova il file in base alla piattaforma di destinazione.

7.7.2 I prefab dentro la scroll list avevano dei collider di grandezza errata

Inizialmente i prefab all'interno della scrollist avevano un input field e vari bottoni, però man mano che cresceva il numero degli elementi nella scrollist più i collider di questi bottoni diminuivano di dimensione, rendendo impossibile cliccarli. Per risolvere si è optato per un unico bottone con dentro il nome dell'ingrediente, che toccato fa comparire un menù per la modifica dell'ingrediente.

Capitolo 8

Conclusioni

8.1 La mia esperienza con Unity e MRTK

8.2 Sviluppi futuri della mia applicazione

8.3 Considerazioni finali

Bibliografia

- [1] An overview of augmented reality, 2022. URL <https://www.mdpi.com/2073-431X/11/2/28>.
- [2] The end of google glass, 2023. URL <https://www.cnbc.com/2023/03/15/google-discontinues-google-glass-enterprise-end-to-early-ar-project.html>.
- [3] Oculus rift launches as vr market heats up, 2023. URL <https://www.cnbc.com/2016/03/28/oculus-rift-launches-as-vr-market-heats-up.html>.
- [4] What is mixed reality?, 2023. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/discover/mixed-reality>.
- [5] Novel mixed reality use cases for pilot training, 2022. URL <https://www.mdpi.com/2227-7102/12/5/345>.
- [6] Training and simulation for enterprises, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/training-simulation>.
- [7] Prototyping and manufacturing for enterprises, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/prototyping-manufacturing>.
- [8] Virtual museums, exhibits, & tourism, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/enthusiast-guide/virtual-museums#virtual-museums-exhibits-and-tourism-what-it-is-and-how-it-works>.
- [9] Ambient information visualisation and visitors' technology acceptance of mixed reality in museums, 2020. URL <https://repository.falmouth.ac.uk/3973/7/Ambient%20Information%20Visualisation%20and%20Visitors%27%20Technology%20Acceptance%20of%20Mixed%20Reality%20in%20Museums%20-%20Revise%20and%20resubmit.pdf>.
- [10] About hololens 2, 2023. URL <https://learn.microsoft.com/en-us/hololens/hololens2-hardware>.
- [11] What is mixed reality toolkit 2?, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05>.
- [12] Introducing gemini: our largest and most capable ai model, 2023. URL <https://blog.google/technology/ai/google-gemini-ai/>.
- [13] An overview of the gemini app, 2024. URL <https://gemini.google/overview/>.

- [14] What's the difference between ar, vr and mr?, 2024. URL <https://forwork.meta.com/en/blog/difference-between-vr-ar-and-mr/>.
- [15] What is augmented reality?, 2024. URL <https://www.ibm.com/think/topics/augmented-reality>.
- [16] Tab view, 2023. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-uxcore/packages/uxcore/tab-view>.
- [17] Slate — mrtk3, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-uxcomponents-noncanvas/packages/uxcomponents-noncanvas/slate>.
- [18] Near menu — mrtk3, 2022. URL <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-uxcomponents/packages/uxcomponents/near-menu>.
- [19] Monobehaviour, 2025. URL <https://docs.unity3d.com/6000.1/Documentation/Manual/class-MonoBehaviour.html>.
- [20] Monobehaviour.awake(), 2025. URL <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.Awake.html>.
- [21] Monobehaviour.onenable(), 2025. URL <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.OnEnable.html>.
- [22] Monobehaviour.start(), 2025. URL <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.Start.html>.
- [23] Monobehaviour.update(), 2025. URL <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.Update.html>.
- [24] Monobehaviour.ondisable(), 2025. URL <https://docs.unity3d.com/6000.1/Documentation/ScriptReference/MonoBehaviour.OnDisable.html>.
- [25] Prefabs, 2025. URL <https://docs.unity3d.com/6000.1/Documentation/Manual/Prefabs.html>.
- [26] Implementing the singleton pattern in c#. URL <https://csharpindepth.com/articles/singleton>.
- [27] What is a large language model (llm)? URL <https://www.cloudflare.com/learning/ai/what-is-large-language-model/>.
- [28] What are large language models (llms)?, 2023. URL <https://www.ibm.com/think/topics/large-language-models>.
- [29] Gemini live, 2025. URL <https://gemini.google/overview/gemini-live/?hl=en>.
- [30] Share your camera with gemini live to ask questions about what you see, 2025. URL <https://www.youtube.com/watch?v=aQKWQNbAH90&t=33s>.

- [31] An introduction to large language models: Prompt engineering and p-tuning, 2023. URL <https://developer.nvidia.com/blog/an-introduction-to-large-language-models-prompt-engineering-and-p-tuning/>.
- [32] Prompt design strategies. URL <https://ai.google.dev/gemini-api/docs/prompting-strategies>.
- [33] Structured output, 2025. URL <https://ai.google.dev/gemini-api/docs/structured-output?lang=rest>.

Ringraziamenti