UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un serious game per la descrizione delle patologie causate dai virus sui bovini da utilizzarsi in un Corso di Laurea in Veterinaria

Design and implementation of a serious game for the description of diseases caused by viruses on cows for use in a Veterinary Degree Course

Relatore

Prof. Domenico Ursino

Correlatrice

Dott.ssa Marta Giacomazzo

Candidato

Loisi Squarcella

ANNO ACCADEMICO 2023-2024

"Ci sono cose che non si possono fermare: la volontà ereditata,
i sogni delle persone, l'andirivieni delle ere.
Finché le persone avranno sete di libertà,
queste cose dureranno per sempre"

Gol D. Roger

Sommario

I serious game stanno acquisendo un ruolo sempre più significativo nell'innovazione dei metodi di apprendimento, combinando interattività e tecnologia per migliorare l'acquisizione di competenze. In questa tesi vengono descritte la progettazione e l'implementazione di un serious game in realtà virtuale, finalizzato all'apprendimento delle patologie causate dai virus nei bovini, da utilizzarsi in un corso di laurea veterinaria. Il lavoro si è avviato con una fase di raccolta e analisi dei requisiti, che ha permesso di delineare le classi principali e il flusso generale del gioco tramite i diagrammi UML, tra cui attività e casi d'uso. La successiva fase di implementazione, condotta utilizzando il motore grafico Unity, ha consentito di realizzare un ambiente immersivo e interattivo. Infine, il sistema è stato testato per garantire funzionalità e coerenza, ponendo le basi per future espansioni del progetto.

Keyword: Realtà Virtuale, Meta Quest, Serious Game, E-learning, Gamification, Diagrammi UML, Analisi dei Requisiti, Unity, Apprendimento, Virus, Grafica 3D, Veterinaria, Blender, Modelli 3D, Motori grafici

Indice

Introduzione **1**

1 La realtà virtuale **3**
1.1 Definizione e caratteristiche della realtà virtuale . . . 3
1.1.1 La magia del mondo virtuale . . . 4
1.1.2 Un po' di storia . . . 4
1.2 Realtà virtuale e serious game . . . 5
1.2.1 Realtà virtuale e apprendimento . . . 5
1.2.2 Vantaggi ed esempi applicativi dei serious game basati sulla realtà virtuale . . . 6
1.3 Applicazioni della realtà virtuale in altri settori . . . 6
1.3.1 Medicina e sanità . . . 6
1.3.2 Industria e produzione . . . 7
1.3.3 Intrattenimento e media . . . 7
1.3.4 Architettura e design . . . 7
1.3.5 Militare e difesa . . . 8
1.3.6 Turismo e beni culturali . . . 8

2 Specifiche dei requisiti **9**
2.1 Descrizione del sistema in linguaggio naturale . . . 9
2.2 Glossario dei termini . . . 10
2.3 Specifiche dei requisiti . . . 10
2.3.1 Requisiti funzionali . . . 10
2.3.2 Requisiti non funzionali . . . 13

3 Progettazione **15**
3.1 Obiettivi di progettazione e target di pubblico . . . 15
3.2 Struttura del gioco . . . 15
3.3 Diagramma delle classi . . . 15
3.3.1 GameLogic . . . 16
3.3.2 GameHandler . . . 19
3.3.3 Virus . . . 22
3.3.4 Utility . . . 26
3.4 Diagramma di attività . . . 31
3.5 Diagramma dei casi d'uso . . . 32

3.5.1 Selezione del virus . . . 32
3.5.2 Movimento del virus . . . 33
3.5.3 Selezione delle cellule . . . 36
3.5.4 Selezione delle lesioni . . . 39
3.5.5 Menù di pausa . . . 42

4 Implementazione **47**
4.1 Practical Design . . . 47
4.1.1 Unity . . . 47
4.1.2 Blender . . . 48
4.1.3 Visual Studio Code . . . 49
4.1.4 C# . . . 49
4.2 Creazione dei modelli 3D . . . 49
4.2.1 Anatomia del bovino . . . 50
4.2.2 Cellule epitelio respiratorio e alveoli . . . 50
4.2.3 Stalla . . . 51
4.3 Implementazione delle funzionalità principali . . . 52
4.3.1 ActivatePathogenesis . . . 53
4.3.2 ActivateVirusAnimator . . . 54
4.3.3 AnimationSliderController . . . 55
4.3.4 CellVirusData . . . 57
4.3.5 ChooseSectionPause . . . 57
4.3.6 CycleObjects . . . 59
4.3.7 DuplicateVirus . . . 60
4.3.8 EpiManager . . . 61
4.3.9 FadeAndSwitchObjects . . . 62
4.3.10 GameManager . . . 63
4.3.11 ImageSelectionManager . . . 64
4.3.12 InteractionTrasmissionZone . . . 67
4.3.13 MaterialChanger . . . 69
4.3.14 PauseManager . . . 69
4.3.15 QuitApplicationHandler . . . 71
4.3.16 SceneInformationManager . . . 71
4.3.17 SceneTransitionHandler . . . 72
4.3.18 SceneTransitionManager . . . 74
4.3.19 StartAnimationPosition . . . 74
4.3.20 StartThirdIteration . . . 75
4.3.21 ThirdInteraction . . . 77
4.4 Implementazione realtà virtuale . . . 80

5 Manuale utente **81**
5.1 Avvertenze per la salute . . . 81
5.2 Installazione e avvio . . . 81
5.3 Elenco completo dei comandi e interfaccia utente . . . 82
5.3.1 Schermata iniziale . . . 82
5.3.2 Ambiente stalla . . . 83
5.3.3 Selezione delle cellule . . . 84
5.3.4 Selezione delle lesioni (gross) . . . 84
5.3.5 Selezione delle lesioni (histo) e fine del gioco . . . 85
5.3.6 Menù di pausa . . . 87
5.4 Come si gioca . . . 88

6 Discussione in merito al lavoro svolto **91**
6.1 Analisi SWOT 91
6.1.1 Strengths 91
6.1.2 Weaknesses 93
6.1.3 Opportunities 93
6.1.4 Threats 94

Conclusioni e sviluppi futuri **95**

Bibliografia **96**

Ringraziamenti **98**

Elenco delle figure

1.1 Morton Heiling mentre prova il Sensorama 5
1.2 Esempio di utilizzo della VR in ambito medico 7
1.3 Esperienza di realtà virtuale in uno scavo sotto l'Abbazia di Bath, Somerset, Inghilterra 8

3.1 Blocco `GameLogic` e relative classi 16
3.2 Blocco `GameHandler` e relative classi 19
3.3 Blocco `Virus` e relative classi 25
3.4 Blocco `Utility` e relative classi 28
3.5 Diagramma di attività del gioco 31
3.6 Diagramma di attività del menù di pausa 32
3.7 Diagramma dei casi d'uso relativi alla selezione del virus 33
3.8 Diagramma dei casi d'uso relativi al movimento del virus 34
3.9 Diagramma dei casi d'uso relativi alla selezione delle cellule 37
3.10 Diagramma dei casi d'uso relativi alla selezione delle lesioni 40
3.11 Diagramma dei casi d'uso relativi alla selezione del menù di pausa 42

4.1 Modello dell'anatomia del bovino 50
4.2 Modello delle cellule 51

5.1 Touch Controller del Meta Quest 3 82
5.2 Schermata iniziale 83
5.3 Schermata di selezione del virus 83
5.4 Schermata di uscita 84
5.5 Schermata iniziale nella stalla 85
5.6 Schermata relativa all'animazione del virus 85
5.7 Schermata relativa alla selezione delle cellule 86
5.8 Schermata relativa alla patogenesi 86
5.9 Schermata relativa ai segni clinici 87
5.10 Schermata relativa alla selezione delle immagini di gross 87
5.11 Schermata relativa alla selezione delle immagini histo 88
5.12 Schermata relativa alle informazioni sulle lesioni 88
5.13 Schermata relativa al menù di pausa 89
5.14 Schermata relativa al cambio sezione 89

6.1 Analisi SWOT del serious game 92

Elenco delle tabelle

2.1 Glossario dei termini relativi al serious game da realizzare 10
2.2 Requisiti funzionali e non funzionali del sistema 11
2.3 Requisiti funzionali del sistema 11
2.4 Requisiti funzionali: Scelta virus 12
2.5 Requisiti funzionali: Scelta via di trasmissione 12
2.6 Requisiti funzionali: Scelta cellule infette 12
2.7 Requisiti funzionali: Scelta lesioni 13
2.8 Requisiti funzionali: Menù di pausa 13
2.9 Requisiti funzionali: Utilizzo della realtà virtuale 13
2.10 Analisi dei requisiti non funzionali del sistema 13
2.11 Requisiti non funzionali 14

3.1 Blocco `GameLogic`: elementi della classe `ActivePathogenesis` 17
3.2 Blocco `GameLogic`: elementi della classe `ChooseSectionPause` 18
3.3 Blocco `GameLogic`: elementi della classe `EpiManager` 18
3.4 Blocco `GameLogic`: elementi della classe `GameManager` 19
3.5 Blocco `GameLogic`: elementi della classe `ImageSelectionManager` 20
3.6 Blocco `GameLogic`: elementi della classe `SceneInformationManager` . . . 21
3.7 Blocco `GameLogic`: elementi della classe `StartThirdIteration` 21
3.8 Blocco `GameLogic`: elementi della classe `ThirdInteraction` 22
3.9 Blocco `GameHandler`: elementi della classe `PauseManager` 23
3.10 Blocco `GameHandler`: elementi della classe `QuitApplicationHandler` . . 23
3.11 Blocco `GameHandler`: elementi della classe `SceneTransitionHandler` . . 24
3.12 Blocco `GameHandler`: elementi della classe `SceneTransitionManager` . . 24
3.13 Blocco `Virus`: elementi della classe `ActivateVirusAnimator` 24
3.14 Blocco `Virus`: elementi della classe `CellVirusData` 25
3.15 Blocco `Virus`: elementi della classe `DuplicateVirus` 26
3.16 Blocco `Virus`: elementi della classe `InteractionTrasmissionZone` . . . 27
3.17 Blocco `Virus`: elementi della classe `VirusDrag` 27
3.18 Blocco `Utility`: elementi della classe `FadeAndSwitchObjects` 28
3.19 Blocco `Utility`: elementi della classe `AnimationSliderController` . . . 29
3.20 Blocco `Utility`: elementi della classe `CycleObjects` 30
3.21 Blocco `Utility`: elementi della classe `MaterialChanger` 30
3.22 Blocco `Utility`: elementi della classe `StartAnimationPosition` 30
3.23 Caso d'uso relativo alla visualizzazione del virus 33

3.24 Caso d'uso relativo alla selezione del virus 33
3.25 Caso d'uso relativo al cambio scena 34
3.26 Caso d'uso relativo allo spostamento del virus nello spazio 35
3.27 Caso d'uso relativo al raggiungimento della via di trasmissione 35
3.28 Caso d'uso relativo alla segnalazione della via di trasmissione errata 35
3.29 Caso d'uso relativo alla visualizzazione dell'interno del bovino 36
3.30 Caso d'uso relativo alla visualizzazione dell'animazione di trasmissione . . . 36
3.31 Caso d'uso relativo alla visualizzazione delle informazioni sulla via di trasmissione 37
3.32 Caso d'uso relativo alla visualizzazione delle cellule 37
3.33 Caso d'uso relativo selezione delle cellule 38
3.34 Caso d'uso relativo alla colorazione delle cellule 38
3.35 Caso d'uso relativo alla visualizzazione della patogenesi del virus 39
3.36 Caso d'uso relativo alla visualizzazione dei segni clinici 39
3.37 Caso d'uso relativo alla visualizzazione delle lesioni 40
3.38 Caso d'uso relativo allo scorrimento delle immagini 40
3.39 Caso d'uso relativo alla selezione della lesione 41
3.40 Caso d'uso relativo alla segnalazione della selezione di una lesione errata . . 41
3.41 Caso d'uso relativo alla visualizzazione della schermata finale 42
3.42 Caso d'uso relativo alla ripresa del gioco nel menù 43
3.43 Caso d'uso relativo al cambio di sezione nel menù 43
3.44 Caso d'uso relativo all'uscita dal gioco nel menù 44
3.45 Caso d'uso relativo alla scelta del menù iniziale nel menù 44
3.46 Caso d'uso relativo alla scelta della via di trasmissione nel menù 45
3.47 Caso d'uso relativo alla scelta delle cellule nel menù 45
3.48 Caso d'uso relativo alla scelta delle macro nel menù 46
3.49 Caso d'uso relativo alla scelta delle micro nel menù 46

5.1 Elenco dei comandi per la schermata iniziale 83
5.2 Elenco dei comandi per le interazioni nella stalla 84
5.3 Elenco dei comandi per la selezione delle cellule 85
5.4 Elenco dei comandi per la selezione delle immagini di gross 86
5.5 Elenco dei comandi per la selezione delle immagini histo 87
5.6 Elenco dei comandi per il menù di pausa 88

Introduzione

I serious game rappresentano un approccio innovativo all'apprendimento, combinando elementi ludici con finalità educative e formative. Questi strumenti consentono agli utenti di apprendere attraverso l'esperienza diretta, l'interazione e la risoluzione di problemi, offrendo un metodo di insegnamento coinvolgente e motivante.

L'aspetto ludico consente di mantenere alta la motivazione degli utenti, che, sentendosi coinvolti in un'esperienza piacevole e stimolante, sono più inclini a proseguire l'apprendimento senza percepirlo come un obbligo.

Quando integrati con la realtà virtuale, i serious game acquisiscono un ulteriore livello di efficacia, poiché la tecnologia immersiva consente di simulare ambienti tridimensionali realistici, favorendo una maggiore partecipazione e comprensione da parte dell'utente.

La capacità di integrare elementi visivi, sonori e tattili in un ambiente tridimensionale permette agli utenti di apprendere in modo multisensoriale, rendendo più durature le conoscenze acquisite.

Questa sinergia tra gioco e realtà virtuale trasforma il processo educativo, rendendolo più dinamico e interattivo rispetto ai tradizionali metodi didattici.

In questo contesto, la presente tesi si propone di esaminare la creazione e l'implementazione di un serious game in realtà virtuale per supportare l'apprendimento delle patologie causate dai virus nei bovini, destinato agli studenti di corsi di laurea in veterinaria.

L'obiettivo principale del serious game è quello di migliorare la comprensione degli aspetti legati alla trasmissione, alla patogenesi e alle lesioni causate dai virus vaccini, offrendo agli studenti un'esperienza educativa più pratica e immersiva.

Il lavoro si è articolato in diverse fasi, partendo dalla raccolta e analisi dei requisiti forniti dal Dipartimento di Medicina Veterinaria dell'Università di Padova. L'analisi dei requisiti ha consentito di delineare le funzionalità chiave del gioco e di progettare l'architettura del sistema attraverso strumenti di modellazione come i diagrammi UML.

Successivamente, la fase di implementazione è stata realizzata utilizzando il motore grafico Unity; le funzionalità richieste sono state implementate mediante script in C#.

Inoltre, in questa fase è risultato necessario ricorrere alla modellazione 3D tramite il software Blender, poiché non vi erano modelli esistenti adatti allo scopo.

Il progetto si distingue per il connubio tra contenuti scientifici accurati e l'utilizzo della realtà virtuale, che garantisce agli studenti un ambiente simulato in cui esplorare attivamente i concetti trattati.

È stato elaborato un manuale utente per facilitare l'approccio al gioco, mentre un'analisi SWOT ha permesso di valutare i punti di forza e debolezza del sistema, identificando opportunità di miglioramento e possibili sviluppi futuri.

Questa tesi si pone come un contributo concreto all'innovazione didattica, proponendo uno strumento capace di arricchire il percorso formativo degli studenti di veterinaria e di integrare nuove tecnologie nell'insegnamento delle scienze mediche.

Il presente elaborato è composto da 6 capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 introdurremo i concetti chiave relativi alla realtà virtuale e ai serious game, evidenziando i loro vantaggi nel processo di apprendimento, con particolare attenzione all'integrazione della realtà virtuale per aumentare il coinvolgimento degli utenti.
- Nel Capitolo 2 analizzeremo i requisiti del progetto, suddivisi in funzionali e non funzionali, definiti sulla base delle richieste effettuate dalla Facoltà di Veterinaria dell'Università di Padova.
- Nel Capitolo 3 illustreremo la progettazione del serious game, includendo i diagrammi UML che descrivono la struttura del software, come i diagrammi delle classi, di attività e dei casi d'uso.
- Nel Capitolo 4 descriveremo l'implementazione del serious game, dettagliando gli strumenti e i linguaggi utilizzati, e mostrando le porzioni di codice relative alle funzionalità principali.
- Nel Capitolo 5 presenteremo il manuale utente, che descrive i comandi necessari per utilizzare il serious game e fornisce esempi visivi delle schermate principali del gioco.
- Nel Capitolo 6 valuteremo il serious game attraverso un'analisi SWOT, identificando i punti di forza, i punti di debolezza, le opportunità e le minacce del progetto.

CAPITOLO 1

La realtà virtuale

Nel capitolo introduttivo di questa tesi, si esamineranno le caratteristiche principali della realtà virtuale, approfondendo le tecnologie che la compongono e le peculiarità che la distinguono dalle altre forme di simulazione digitale. Inoltre, verranno analizzati i suoi ambiti di applicazione, con particolare attenzione agli scenari formativi.

1.1 Definizione e caratteristiche della realtà virtuale

La *realtà virtuale*, comunemente nota come *virtual reality* (VR), rappresenta una tecnologia avanzata che consente l'esplorazione e l'interazione con ambienti tridimensionali artificiali. Attraverso l'utilizzo di dispositivi quali visori e controller, gli utenti possono immergersi in mondi simulati, sperimentando un marcato senso di "presenza", amplificata dall'isolamento dalle distrazioni esterne e da un coinvolgimento sensoriale completo. Questa tecnologia, distinta da altre forme di visualizzazione e simulazione, offre esperienze completamente immersive, particolarmente adatte per un'ampia gamma di applicazioni, che spaziano dall'istruzione all'intrattenimento, dalla terapia alla formazione professionale.

Per comprendere appieno il fenomeno della VR, è utile considerare i sette concetti teorici delineati da Michael Heim nel 1993, che ne descrivono le basi filosofiche e funzionali:

- *Simulazione*: la realtà virtuale riproduce ambienti artificiali che simulano la realtà fisica o danno vita a scenari completamente inventati. Ciò consente di replicare situazioni realistiche, come esercitazioni di volo o simulazioni chirurgiche, le quali sarebbero eccessivamente rischiose o complesse da attuare nel mondo reale.

- *Interazione*: l'interazione rappresenta un elemento centrale della realtà virtuale, consentendo agli utenti di influenzare attivamente l'ambiente virtuale. Attraverso gesti, movimenti o comandi, l'utente modifica in tempo reale l'ambiente virtuale, favorendo così un'esperienza immersiva in cui diviene parte integrante della simulazione.

- *Artificiosità*: la realtà virtuale introduce il concetto di artificiosità, creando mondi che non sono soggetti alle leggi fisiche o alla logica del mondo reale. Questa libertà consente l'esplorazione di scenari e idee che nella realtà sarebbero impossibili o estremamente difficili da rappresentare.

- *Immersione psicologica*: la sensazione di immersione si riferisce alla capacità della realtà virtuale di trasportare psicologicamente l'utente all'interno del mondo virtuale. Grazie

a visori, cuffie e altre periferiche, l'utente è avvolto dall'ambiente simulato, percependo un senso di presenza che lo distacca temporaneamente dal mondo fisico.

- *Telepresenza*: questo concetto è strettamente correlato alla percezione di trovarsi fisicamente in un luogo remoto o in uno scenario virtuale. La telepresenza risulta particolarmente utile nelle simulazioni che richiedono la rappresentazione di ambienti lontani o inaccessibili, come nel caso di missioni di esplorazione spaziale o ambienti estremi.
- *Immersione totale*: l'immersione totale coinvolge tutti i sensi dell'utente, frequentemente attraverso dispositivi quali controller manuali, sensori di movimento e visori. Tale livello di coinvolgimento accresce il realismo e l'efficacia della simulazione, poiché l'utente percepisce fisicamente l'ambiente e reagisce ad esso in modo più naturale.
- *Rete di comunicazione*: Heim sottolinea l'importanza della comunicazione e della condivisione in un ambiente virtuale, che consente a più utenti di interagire e collaborare simultaneamente. Questo aspetto risulta essenziale nei contesti di social VR, in cui utenti reali si incontrano virtualmente per collaborare e condividere esperienze.

1.1.1 La magia del mondo virtuale

La "magia" della realtà virtuale risiede nella sua capacità di creare esperienze che superano la mera rappresentazione visiva, creando estensioni immersive della percezione umana. Attraverso un'attenta progettazione degli ambienti e delle interazioni, la realtà virtuale permette di raggiungere uno stato di "flow", o flusso, durante il quale l'utente è completamente immerso, dimenticando la distinzione tra realtà e simulazione.

Questa combinazione di tecnologia e psicologia crea esperienze capaci di suscitare emozioni profonde, aumentare l'apprendimento e facilitare il coinvolgimento. La realtà virtuale, con i suoi potenziali infiniti, invita l'utente non solo a osservare, ma anche a vivere e a esplorare nuovi orizzonti, aprendo la strada a nuove possibilità in ambito educativo, medico, artistico e ricreativo.

1.1.2 Un po' di storia

Le radici della realtà virtuale affondano negli anni '60, con il primo esempio di esperienza multisensoriale, ovvero il *Sensorama* (Figura 1.1), ideato da Morton Heilig nel 1962, progettato per coinvolgere la vista, l'udito, l'olfatto e il tatto, creando una simulazione sensoriale immersiva.

Pochi anni dopo, nel 1968, Ivan Sutherland sviluppò *The Sword of Damocles*, considerato il primo vero visore VR, in grado di visualizzare immagini tridimensionali che reagivano ai movimenti della testa. Pur essendo ancora rudimentale, segnò un punto di partenza verso la moderna interazione VR.

Durante gli anni '80, la realtà virtuale trovò applicazioni in ambito *militare* e *industriale*, con simulatori di volo e strumenti per l'addestramento astronautico, come quelli sviluppati dalla NASA. Negli anni '90, tentativi commerciali di portare la VR al pubblico, come il *Sega VR* e il *Virtual Boy* di Nintendo, non ebbero successo, limitati dalla tecnologia dell'epoca.

Con l'avvento di nuovi display e sensori di movimento, la VR conobbe una rinascita nel 2012, con il lancio di *Oculus Rift*, che portò la realtà virtuale a una nuova generazione di utenti e applicazioni. Da allora, molte aziende come *HTC*, *Google* e *Sony* hanno sviluppato i loro visori, favorendo la diffusione della VR in settori come l'educazione, la medicina e l'industria. Oggi, la realtà virtuale continua a evolversi, spingendo i limiti delle esperienze immersive.

Figura 1.1: Morton Heiling mentre prova il Sensorama

1.2 Realtà virtuale e serious game

I *serious game* sono strumenti digitali che combinano elementi ludici con scopi educativi, formativi o informativi, andando oltre il semplice intrattenimento. Introdotti formalmente come concetto negli anni '70, i *serious game* enfatizzano il loro utilizzo per obiettivi *seri*, come la formazione professionale, la promozione della salute o l'insegnamento. Essi rappresentano un'intersezione tra gioco e apprendimento, utilizzando il potenziale immersivo e interattivo dei videogiochi per migliorare il coinvolgimento e facilitare il trasferimento delle conoscenze.

Uno dei principali vantaggi dei *serious game* è la loro capacità di simulare situazioni complesse in un ambiente controllato e sicuro. Ciò li rende ideali per settori come l'istruzione, dove facilitano l'apprendimento esperienziale, la formazione aziendale, per lo sviluppo di competenze pratiche, e la medicina, per simulazioni chirurgiche o riabilitazione. Inoltre, grazie alla loro componente ludica, migliorano la motivazione e l'attenzione degli utenti, rendendo il processo di apprendimento più coinvolgente.

1.2.1 Realtà virtuale e apprendimento

La realtà virtuale si integra perfettamente con i *serious game*, potenziandone l'efficacia grazie alla sua capacità di creare ambienti immersivi e interattivi. La realtà virtuale consente agli utenti di *vivere* esperienze realistiche e coinvolgenti, favorendo la percezione di "presenza" e aumentando il livello di partecipazione. Questo è particolarmente utile nei contesti educativi, dove gli utenti possono esplorare ambienti difficili o pericolosi da raggiungere nella realtà, come il corpo umano per scopi medici o ambienti estremi per l'addestramento.

I *serious game* basati sulla realtà virtuale offrono anche un'opportunità unica per personalizzare l'apprendimento, adattandolo ai bisogni specifici degli utenti. La possibilità di monitorare le azioni in tempo reale consente di creare feedback immediati e dinamici, migliorando l'efficacia dell'apprendimento. Grazie a tecnologie come i visori di realtà virtuale, l'accessibilità a esperienze immersive è aumentata significativamente, espandendo i *serious game* in numerosi settori, dall'educazione primaria alla formazione professionale avanzata.

1.2.2 Vantaggi ed esempi applicativi dei serious game basati sulla realtà virtuale

La combinazione tra realtà virtuale e serious game offre un'ampia gamma di benefici, rendendoli strumenti estremamente versatili e potenti in molti ambiti. Grazie alla capacità della realtà virtuale di creare ambienti immersivi, interattivi e altamente personalizzabili, i serious game possono migliorare non solo l'efficacia dell'apprendimento ma anche l'engagement degli utenti. Di seguito vengono illustrati alcuni dei principali vantaggi offerti da questa tecnologia.

Un aspetto chiave è l'*immersione e il coinvolgimento*, che consente agli utenti di concentrarsi pienamente sull'attività, riducendo le distrazioni e favorendo l'apprendimento esperienziale. Inoltre, la realtà virtuale permette la *simulazione di scenari complessi* o difficilmente accessibili nella realtà, garantendo un ambiente sicuro per sperimentare o esercitarsi. La possibilità di offrire *feedback immediati e misurabili*, grazie ai sensori e agli algoritmi integrati, rende questi giochi uno strumento di apprendimento dinamico e adattabile alle esigenze specifiche dell'utente.

Questi vantaggi hanno consentito ai serious game di trovare applicazione in numerosi settori. In ambito educativo, vengono utilizzati per spiegare concetti complessi, come il funzionamento del corpo umano o principi scientifici, attraverso simulazioni coinvolgenti e pratiche. In medicina, i serious game basati su realtà virtuale sono impiegati per l'addestramento di chirurghi, con simulazioni di interventi realistici, o per la riabilitazione, aiutando i pazienti a eseguire esercizi specifici in un ambiente controllato.

Anche la formazione professionale ha beneficiato ampiamente di queste tecnologie, con applicazioni che vanno dalla gestione di emergenze in ambienti industriali all'addestramento militare. Infine, in ambito psicologico, i serious game VR vengono utilizzati per trattare fobie, ansia o stress post-traumatico, attraverso simulazioni che permettono ai pazienti di affrontare gradualmente situazioni difficili in un contesto sicuro e controllato.

Grazie a questi esempi e vantaggi, la realtà virtuale combinata con i serious game sta ridefinendo il modo in cui apprendimento e formazione vengono concepiti, rendendo questi strumenti sempre più essenziali in un mondo in continua evoluzione tecnologica.

1.3 Applicazioni della realtà virtuale in altri settori

La realtà virtuale, grazie alla sua capacità di creare ambienti immersivi e interattivi, ha trovato applicazioni significative in una vasta gamma di settori, che vanno oltre l'ambito educativo e formativo. L'evoluzione delle tecnologie di realtà virtuale ha consentito di sviluppare soluzioni innovative che migliorano le prestazioni, la sicurezza e l'efficienza in contesti altamente complessi. Di seguito vengono analizzati i principali ambiti di applicazione.

1.3.1 Medicina e sanità

In ambito medico, la realtà virtuale si è affermata come uno strumento prezioso per la simulazione, la formazione e la terapia. I visori VR vengono utilizzati per addestrare chirurghi in scenari simulati altamente realistici, dove è possibile esercitarsi su procedure complesse senza alcun rischio per i pazienti. Inoltre, la VR viene impiegata nella riabilitazione motoria e cognitiva, offrendo esercizi interattivi che stimolano il recupero delle capacità dei pazienti in un ambiente controllato.

Un esempio significativo è rappresentato dalle simulazioni di interventi chirurgici cardiaci, in cui i medici possono esplorare un modello tridimensionale del cuore del paziente per pianificare l'operazione, come mostrato in Figura 1.2. Anche la terapia cognitivo-comportamentale beneficia della VR, utilizzata per trattare fobie, disturbi da stress post-traumatico e altre

Figura 1.2: Esempio di utilizzo della VR in ambito medico

condizioni psicologiche, grazie alla possibilità di ricreare situazioni specifiche in modo sicuro e personalizzato.

1.3.2 Industria e produzione

Nell'industria, la realtà virtuale viene utilizzata per migliorare la progettazione, la manutenzione e la sicurezza. Gli ambienti virtuali permettono di simulare interi processi produttivi, consentendo agli ingegneri di identificare inefficienze o errori progettuali prima ancora che i macchinari vengano costruiti. La VR viene, inoltre, utilizzata per l'addestramento dei lavoratori, in particolare in ambienti ad alto rischio come impianti chimici, centrali elettriche o piattaforme petrolifere, dove la formazione pratica sarebbe pericolosa o costosa.

Un ulteriore ambito di applicazione è rappresentato dalla manutenzione predittiva: gli operatori possono interagire con modelli virtuali dei macchinari, analizzandone le prestazioni e anticipando eventuali guasti.

1.3.3 Intrattenimento e media

Il settore dell'intrattenimento è stato uno dei primi ad adottare la realtà virtuale su larga scala, grazie al suo potenziale immersivo. Oltre ai videogiochi, che rappresentano l'applicazione più conosciuta, la VR viene utilizzata per esperienze cinematografiche interattive, parchi a tema e simulazioni virtuali di eventi sportivi o concerti.

Nel campo della produzione mediatica, la VR consente di creare contenuti interattivi che coinvolgono l'utente in modo profondo, offrendo nuove modalità di narrazione e partecipazione.

1.3.4 Architettura e design

In architettura e design, la realtà virtuale permette ai progettisti di esplorare e presentare progetti tridimensionali in scala reale prima della costruzione fisica. I clienti possono camminare all'interno di edifici virtuali, analizzare gli spazi, i materiali e l'illuminazione, fornendo un feedback diretto che migliora il risultato finale. Questo approccio è particolarmente utile per progetti complessi, come edifici pubblici o infrastrutture su larga scala.

La VR viene, inoltre, utilizzata nel design industriale per testare prototipi virtuali di prodotti, riducendo i costi e i tempi di sviluppo.

1.3.5 Militare e difesa

In ambito militare, la realtà virtuale viene adottata per addestrare soldati e piloti in simulazioni di combattimento, scenari operativi e missioni tattiche. Gli ambienti virtuali consentono di esercitarsi in situazioni che sarebbero troppo costose o pericolose da ricreare nel mondo reale, come operazioni in zone di guerra o scenari di evacuazione.

La VR è anche impiegata per la progettazione e la valutazione di nuovi sistemi d'arma, dove i progettisti possono simulare l'uso di equipaggiamenti in condizioni realistiche.

1.3.6 Turismo e beni culturali

Il turismo virtuale e la valorizzazione del patrimonio culturale rappresentano applicazioni emergenti della realtà virtuale. Grazie a questa tecnologia, gli utenti possono visitare virtualmente luoghi remoti o siti archeologici inaccessibili, esplorandoli con un realismo sorprendente (Figura 1.3).

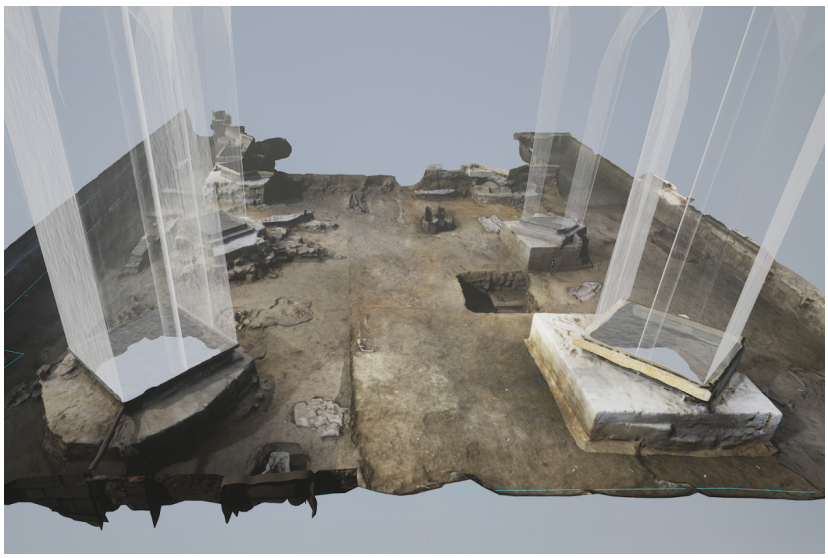


Figura 1.3: Esperienza di realtà virtuale in uno scavo sotto l'Abbazia di Bath, Somerset, Inghilterra

Musei e istituzioni culturali utilizzano la VR per creare mostre interattive che permettono ai visitatori di immergersi nella storia o nella vita quotidiana di epoche passate. Questo approccio non solo rende il patrimonio culturale più accessibile, ma arricchisce l'esperienza educativa.

CAPITOLO 2

Specifiche dei requisiti

In questo capitolo viene presentata una descrizione dettagliata dei requisiti relativi al sistema proposto, con l'obiettivo di fornire una visione chiara e completa delle funzionalità e delle caratteristiche che esso deve offrire. In seguito viene fornita una descrizione in linguaggio naturale del sistema, accompagnata da un glossario dei termini specifici utilizzati. Successivamente, viene illustrata l'analisi dei requisiti, distinguendo tra requisiti funzionali e non funzionali, per definire chiaramente cosa il sistema deve fare e quali qualità deve garantire. Infine, ogni requisito è dettagliato per evidenziare la sua rilevanza all'interno del progetto e l'impatto sull'esperienza utente.

2.1 Descrizione del sistema in linguaggio naturale

Il progetto proposto consiste nella realizzazione di un serious game in realtà virtuale per la descrizione delle patologie causate da alcuni virus sui bovini, da utilizzarsi in un Corso di Laurea in Veterinaria.

Gli aspetti di interesse risultano essere:

- la possibilità di selezionare, da parte dell'utente, il virus con il quale intende proseguire;
- la visualizzazione di una spiegazione preliminare circa il virus da selezionare;
- la capacità di spostarsi dalla scena di selezione del virus a quella di gioco;
- la possibilità di selezionare, da parte dell'utente, il sito di ingresso relativo al virus scelto;
- la visualizzazione di un'animazione e di una spiegazione preliminare di come avviene la trasmissione del virus;
- la possibilità di selezionare, da parte dell'utente, le cellule colpite dal virus scelto;
- la visualizzazione e la spiegazione della patogenesi del virus scelto;
- la possibilità di selezionare, da parte dell'utente, la lesione associata al virus scelto;
- la visualizzazione e la spiegazione delle lesioni provocate dal virus;
- la possibilità, da parte dell'utente, di mettere in pausa il gioco e di spostarsi,tramite un menù, tra le diverse interazioni presenti.

2.2 Glossario dei termini

Il glossario dei termini rappresenta una raccolta di definizioni che offrono spiegazioni chiare e concise per le parole potenzialmente ambigue o poco chiare nel contesto di riferimento. Questo strumento è fondamentale per assicurare una comunicazione precisa e facilmente comprensibile. Nella Tabella 2.1 è riportato il glossario dei termini di interesse.

Termine	Descrizione	Sinonimi
Utente	La persona giocante, colei che giocherà con il serious game	Giocatore, Player
Virus	Agente infettivo microscopico composto da acido nucleico (DNA o RNA) avvolto da una proteina, che può infettare e replicarsi all'interno di cellule vive degli organismi	Agente patogeno, Germe
Patologia	Indica uno stato patologico, una malattia in atto	Malattia, Morbo
Lesione	Danno o anomalia causata dall'infezione virale che colpisce un tessuto o un organo	Effetto della malattia
Sintomi	Segni evidenti di una malattia o infezione virale, che possono includere cambiamenti nel comportamento, nel corpo o, in generale, nella salute	Segni
Scena	Ambiente dove vi è inserito il contenuto, ovvero tutte le componenti di un gioco	Scenario
Animazione	Sequenza di immagini o movimenti di un oggetto di scena	Clip, Cut-Scene
Interazione	Possibilità di influenzare l'ambiente circostante con azioni	Selezione, Scelta
Visualizzazione	Illustrazione delle informazioni o delle immagini relative al virus, alla trasmissione, alla lesione o ai sintomi	-

Tabella 2.1: Glossario dei termini relativi al serious game da realizzare

2.3 Specifiche dei requisiti

Prima di approfondire nel dettaglio i requisiti, è essenziale suddividerli e classificarli in due categorie principali, ovvero requisiti funzionali e requisiti non funzionali. Questa distinzione risulta fondamentale per garantire un'analisi e una progettazione accurate. La suddivisione dei requisiti è rappresentata nella Tabella 2.2.

2.3.1 Requisiti funzionali

I requisiti funzionali descrivono, nel linguaggio naturale, le funzionalità e/o i servizi che il sistema deve fornire in risposta a una richiesta o un'azione dell'utente. Essi specificano cosa il sistema dovrebbe essere in grado di fare, le azioni che dovrebbe svolgere e i risultati che dovrebbe produrre.

Questi requisiti garantiscono che il sistema fornisca un'esperienza coerente, soddisfi le aspettative del target di riferimento e risponda adeguatamente alle azioni dei giocatori. Nella Tabella 2.3 vengono riportati i requisiti funzionali relativi al sistema da realizzare.

Requisiti funzionali	Requisiti non funzionali
- Scelta virus - Scelta via di trasmissione - Scelta cellule infette - Scelta lesioni - Menù di pausa - Utilizzo della realtà virtuale	- Implementazione - Interfaccia grafica - Interattività - Senso di immersione

Tabella 2.2: Requisiti funzionali e non funzionali del sistema

Requisiti funzionali

Scelta virus
RF1 - Selezione virus
RF2 - Spiegazione virus
RF3 - Ingresso nella scena di gioco

Scelta via di trasmissione
RF4 - Selezione via di trasmissione
RF5 - Animazione trasmissione
RF6 - Spiegazione trasmissione

Scelta cellule infette
RF7 - Selezione cellule infette
RF8 - Spiegazione patogenesi
RF9 - Spiegazione sintomi

Scelta lesioni
RF10 - Visualizzazione gross e histo
RF11 - Selezione immagine lesioni
RF12 - Spiegazione lesioni

Menù di pausa
RF13 - Possibilità di mettere in pausa
RF14 - Selezione cambio interazione
RF15 - Uscire dal gioco

Utilizzo della realtà virtuale
RF16 - Utilizzo di visore
RF17 - Utilizzo di controller

Tabella 2.3: Requisiti funzionali del sistema

Scelta virus

Nella Tabella 2.4 sono riportati i requisiti funzionali relativi alla scelta del virus.

Scelta via di trasmissione

Nella Tabella 2.5 sono riportati i requisiti funzionali relativi alla scelta della via di trasmissione.

Requisito	Descrizione
RF1 - Selezione virus	Il sistema dovrà consentire di scegliere tra tutti i virus presenti
RF2 - Spiegazione virus	Il sistema dovrà fornire una breve spiegazione sul virus scelto
RF3 - Ingresso nella scena di gioco	Il sistema dovrà gestire l'ingresso nella scena di gioco dalla scena di selezione

Tabella 2.4: Requisiti funzionali: Scelta virus

Requisito	Descrizione
RF4 - Selezione via di trasmissione	Il sistema dovrà consentire di scegliere la via di trasmissione corretta mediante lo spostamento del virus
RF5 - Animazione trasmissione	Il sistema dovrà riprodurre un'animazione che mostra la modalità di trasmissione del virus
RF6 - Spiegazione trasmissione	Il sistema dovrà consentire la visualizzazione di una spiegazione testuale di come il virus arriva all'organo specifico

Tabella 2.5: Requisiti funzionali: Scelta via di trasmissione

Scelta cellule infette

Nella Tabella 2.6 sono riportati i requisiti funzionali relativi alla scelta delle cellule che vengono colpite dal virus.

Requisito	Descrizione
RF7 - Selezione cellule infette	Il sistema dovrà consentire la selezione delle cellule correlate al virus scelto
RF8 - Spiegazione patogenesi	Il sistema dovrà consentire la visualizzazione della descrizione della patogenesi relativa al virus scelto
RF9 - Spiegazione sintomi	Il sistema dovrà consentire la visualizzazione della descrizione dei sintomi

Tabella 2.6: Requisiti funzionali: Scelta cellule infette

Scelta lesioni

Nella Tabella 2.7 sono riportati i requisiti funzionali relativi alla scelta delle lesioni.

Menù di pausa

Nella Tabella 2.8 sono riportati i requisiti funzionali relativi al menù di pausa.

Utilizzo della realtà virtuale

Nella Tabella 2.9 sono riportati i requisiti funzionali relativi all'utilizzo della realtà virtuale.

Requisito	Descrizione
RF10 - Visualizzazione gross e histo	Il sistema dovrà consentire la visualizzazioni di immagini di gross e immagini istologiche
RF11 - Selezione immagine lesioni	Il sistema dovrà consentire la selezioni delle immagini corrette tra le presenti
RF12 - Spiegazione lesioni	Il sistema dovrà consentire la visualizzazione di una spiegazione circa le lesioni nelle immagini di gross e nelle immagini istologiche

Tabella 2.7: Requisiti funzionali: Scelta lesioni

Requisito	Descrizione
RF13 - Possibilità di mettere in pausa	Il sistema dovrà consentire all'utente di fermare il tempo di gioco, visualizzando un menù di pausa
RF14 - Selezione cambio interazione	Il sistema dovrà consentire all'utente di selezionare, attraverso il menù di pausa, una delle interazioni presenti, in modo da andare avanti e/o tornare indietro
RF15 - Uscire dal gioco	Il sistema dovrà consentire la scelta, da parte dell'utente, di uscire dal gioco

Tabella 2.8: Requisiti funzionali: Menù di pausa

Requisito	Descrizione
RF16 - Utilizzo di visore	Il sistema dovrà consentire l'utilizzo della realtà virtuale attraverso l'uso di visori
RF17 - Utilizzo di controller	Il sistema dovrà consentire l'utilizzo di controller per la realtà virtuale permettendo la gestione delle interazioni

Tabella 2.9: Requisiti funzionali: Utilizzo della realtà virtuale

2.3.2 Requisiti non funzionali

I requisiti non funzionali si concentrano su aspetti come prestazioni, affidabilità, sicurezza, usabilità e altri vincoli che il sistema deve rispettare. Essi non descrivono le funzionalità specifiche del sistema, ma definiscono le qualità generali che esso deve garantire. Nella Tabella 2.10 vengono riportati i requisiti non funzionali del sistema da realizzare.

Requisiti non funzionali

Requisiti non funzionali
RNF1 - Implementazione
RNF2 - Interfaccia grafica
RNF3 - Interattività
RNF4 - Senso di immersione

Tabella 2.10: Analisi dei requisiti non funzionali del sistema

Nella Tabella 2.11 sono riportati i requisiti non funzionali relativi al sistema.

Requisito	Descrizione
RNF1 - Implementazione	Il sistema dovrà essere sviluppato utilizzando l'editor Unity e come linguaggio di programmazione il C#
RNF2 - Interfaccia grafica	Il sistema dovrà essere dotato di un'interfaccia grafica in 3D
RNF3 - Interattività	Il sistema dovrà offrire un'interattività dinamica per garantire un'esperienza coinvolgente e appagante per gli utenti
RNF4 - Senso di immersione	Il sistema dovrà fare in modo che l'utente si senta immerso nella scena, sia in senso fisico sia in senso psicologico

Tabella 2.11: Requisiti non funzionali

CAPITOLO 3

Progettazione

In questo capitolo verranno trattate le fasi di progettazione del serious game oggetto di questa tesi. Si mostreranno i diagrammi delle classi, spiegando i vari attributi e metodi di ogni classe, quindi i diagrammi di attività. Infine, si concluderà con i diagrammi dei casi d'uso.

3.1 Obiettivi di progettazione e target di pubblico

L'obiettivo alla base dello sviluppo di questo progetto è la creazione di uno strumento di supporto nell'apprendimento delle patologie causate dai virus nei bovini. In particolare, si intende mostrare in modo semplice e intuitivo come i virus infettino l'organismo dei bovini, partendo dalla loro trasmissione fino ad arrivare a visualizzare le lesioni causate e la sintomatologia associata. Il pubblico target di questo videogioco sono gli studenti dei corsi di laurea in veterinaria.

3.2 Struttura del gioco

La struttura del gioco prevede una serie di fasi in cui il giocatore dovrà affrontare diverse interazioni. In particolare, il giocatore dovrà:

1. *scegliere il virus* con cui giocare;
2. *muovere il virus* per indirizzarlo verso la via di trasmissione corretta;
3. *scegliere le cellule* coinvolte dal virus;
4. *scegliere le macro-lesioni* associate al virus;
5. *scegliere le micro-lesioni* associate al virus.

3.3 Diagramma delle classi

I diagrammi delle classi sono uno strumento essenziale nell'analisi e nella progettazione orientata agli oggetti, poiché forniscono una rappresentazione visiva delle classi, evidenziando le loro relazioni, i loro attributi e i loro metodi.

Le sezioni seguenti forniranno una descrizione approfondita di ogni classe, illustrandone gli attributi e i metodi.

3.3.1 GameLogic

Il blocco `GameLogic` contiene le classi relative alla logica di base del gioco, quindi quelle che consentono di selezionare il virus con il quale giocare, di passare e di mantenere le informazioni tra una scena e l'altra, di visualizzare e di interagire con le cellule, di far scegliere al giocatore le immagini relative alle lesioni e di gestire il menù di pausa. Le classi che costituiscono questo blocco sono mostrate in Figura 3.1.

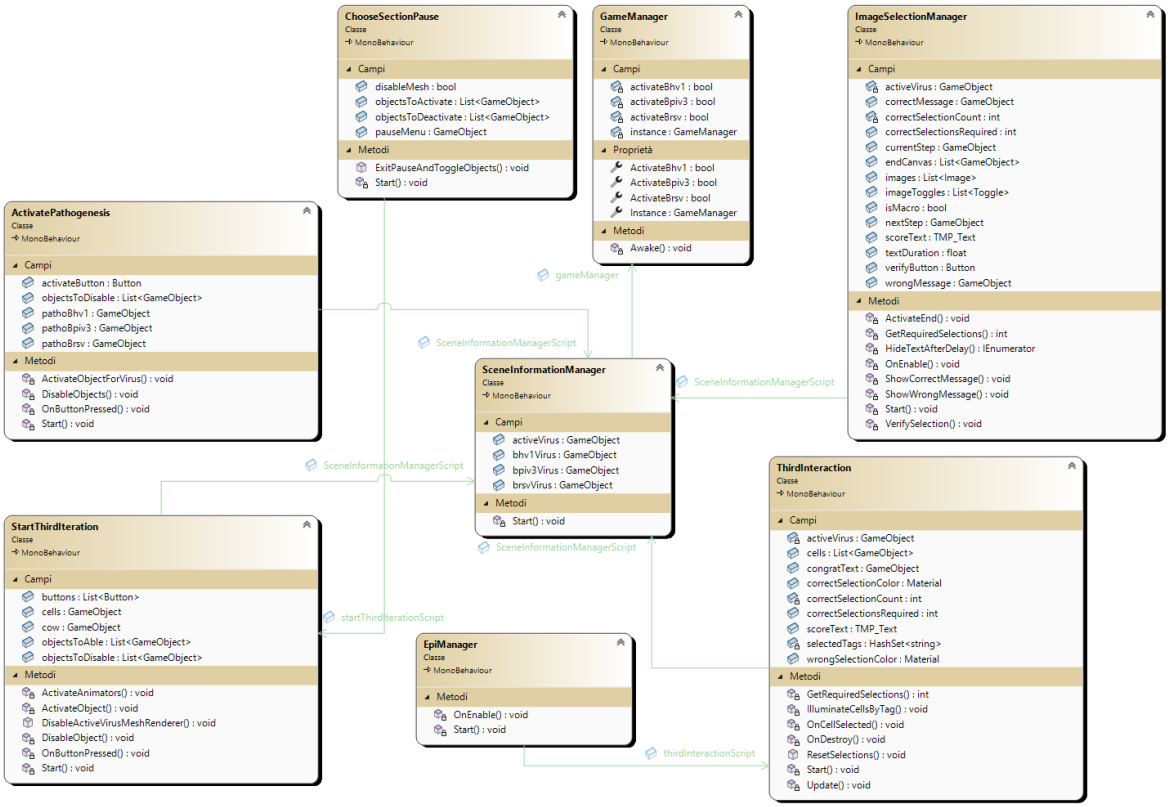


Figura 3.1: Blocco `GameLogic` e relative classi

ActivePathogenesis

La classe `ActivePathogenesis` è responsabile della gestione e visualizzazione delle informazioni relative alla patogenesi del virus all'interno della scena tramite un'interfaccia 3D basata su Unity. Gli elementi che la costituiscono sono mostrati nella Tabella 3.1.

ChooseSectionPause

La classe `ChooseSectionPause` è responsabile della gestione e della visualizzazione degli oggetti nella scena tramite la scelta della fase di gioco in cui si vuole andare. Gli elementi che la costituiscono sono mostrati nella Tabella 3.2.

EpiManager

La classe `EpiManager` gestisce il comportamento del sistema al momento dell'attivazione dell'oggetto 3D in Unity relativo alle cellule, reimpostando le selezioni e riportando le cellule al loro stato originale. Gli elementi che la costituiscono sono mostrati nella Tabella 3.3.

Attributi		
Nome	**Tipo**	**Descrizione**
`activateButton`	`Button`	Riferimento a un oggetto `Button` che viene utilizzato per attivare la visualizzazione della patogenesi relativa
`pathoBhv1`, `pathoBpiv3`, `pathoBrsv`	`GameObject`	Riferimenti al `GameObject` circa la patogenesi da attivare relativa al virus scelto
`objectsToDisable`	`List<GameObject>`	Una lista di `GameObject` che devono essere disattivati prima di attivare la patogenesi
`SceneInformationManagerScript`	`SceneInformationManager`	Riferimento alla classe `SceneInformationManager`, che gestisce le informazioni sul virus attivo nella scena
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Metodo chiamato all'avvio del gioco. Inizializza `SceneInformationManagerScript`; inoltre, associa il metodo `OnButtonPressed()` all' `activateButton`
`OnButtonPressed()`	`void`	Metodo eseguito quando `activateButton` viene premuto. Controlla quale virus è attivo, quindi disattiva gli oggetti nella lista `objectsToDisable` e attiva la patogenesi relativa
`ActivateObjectForVirus(string virusTag)`	`void`	Metodo che controlla il `tag` del virus attivo e attiva la patogenesi associata
`DisableObjects()`	`void`	Metodo che disattiva tutti gli oggetti presenti nella lista `objectsToDisable`

Tabella 3.1: Blocco `GameLogic`: elementi della classe `ActivePathogenesis`

GameManager

La classe `GameManager` gestisce lo stato globale del gioco, permettendo di attivare o disattivare i virus coinvolti. La sua struttura garantisce che esista una sola istanza durante tutta la durata dell'applicazione. Gli elementi che la costituiscono sono mostrati nella Tabella 3.4.

ImageSelectionManager

La classe `ImageSelectionManager` gestisce la selezione e la verifica delle immagini associate al virus scelto. Controlla la correttezza delle scelte effettuate dall'utente e mostra messaggi di feedback (corretti o errati), gestendo anche il passaggio alla fase finale del gioco. Gli elementi che la costituiscono sono mostrati nella Tabella 3.5.

SceneInformationManager

La classe `SceneInformationManager` si occupa di gestire le informazioni relative ai virus attivi nella scena, basandosi sulle impostazioni fornite dal `GameManager`. Attiva il virus appropriato e assegna il riferimento al virus attivo, garantendo una corretta inizializzazione delle risorse. Gli elementi che la costituiscono sono mostrati nella Tabella 3.6.

Attributi		
Nome	**Tipo**	**Descrizione**
`pauseMenu`	`GameObject`	Oggetto che rappresenta il menù di pausa. Viene disattivato quando si esce dalla pausa
`objectsToActivate`	`List<GameObject>`	Lista di oggetti che devono essere attivati quando il gioco esce dalla pausa
`objectsToDeactivat`	`List<GameObject>`	Lista di oggetti che devono essere disattivati quando il gioco esce dalla pausa
`startThirdIterationScript`	`StartThirdIteration`	Riferimento alla classe `StartThirdIteration`, utilizzato per disabilitare il `MeshRenderer` del virus attivo
`disableMesh`	`bool`	Variabile booleana che indica se il `MeshRenderer` del virus attivo deve essere disabilitato
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Metodo chiamato all'avvio del gioco. Inizializza il riferimento a `startThirdIterationScript`
`ExitPauseAndToggleObjects()`	`void`	Metodo principale chiamato per uscire dalla pausa e per gestire gli oggetti nella scena

Tabella 3.2: Blocco `GameLogic`: elementi della classe `ChooseSectionPause`

Attributi		
Nome	**Tipo**	**Descrizione**
`thirdInteractionSc`	`ThirdInteraction`	Riferimento alla classe `ThirdInteraction`, utilizzato per gestire le interazioni delle cellule
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza il riferimento a `ThirdInteraction` se non già assegnato
`OnEnable()`	`void`	Reimposta le selezioni e ripristina lo stato originale delle cellule al momento dell'attivazione dell'oggetto

Tabella 3.3: Blocco `GameLogic`: elementi della classe `EpiManager`

StartThirdIteration

La classe `StartThirdIteration` gestisce il comportamento e le transizioni nella terza fase di gioco. Controlla l'attivazione e la disattivazione di oggetti specifici, abilita animazioni e gestisce le modifiche al `MeshRenderer` del virus attivo. Gli elementi che la costituiscono sono mostrati nella Tabella 3.7.

ThirdInteraction

La classe `ThirdInteraction` gestisce l'interazione dell'utente con le cellule infette nella scena, determinando quali cellule sono associate al virus attivo e aggiornando il punteggio visibile all'utente. Gli elementi che la costituiscono sono elencati nella Tabella 3.8.

Attributi		
Nome	**Tipo**	**Descrizione**
`instance`	`GameManager`	Riferimento statico alla singola istanza della classe (singleton)
`activateBhv1`	`bool`	Indica se il virus `Bhv1` è attivato
`activateBpiv3`	`bool`	Indica se il virus `Bpiv3` è attivato
`activateBrsv`	`bool`	Indica se il virus `Brsv` è attivato
Metodi		
Nome	**Tipo**	**Descrizione**
`Awake()`	`void`	Metodo chiamato prima dell'avvio del gioco. Assicura l'esistenza di una singola istanza della classe e la persistenza dell'oggetto tra le scene
`ActivateBhv1`	`bool`	Proprietà che consente di ottenere o impostare lo stato di attivazione del virus `Bhv1`
`ActivateBpiv3`	`bool`	Proprietà che consente di ottenere o impostare lo stato di attivazione del virus `Bpiv3`
`ActivateBrsv`	`bool`	Proprietà che consente di ottenere o impostare lo stato di attivazione del virus `Brsv`

Tabella 3.4: Blocco `GameLogic`: elementi della classe `GameManager`

3.3.2 GameHandler

Il blocco `GameHandler` contiene le classi relative alla gestione delle scene di gioco, quindi quelle che permettono il passaggio da una scena all'altra tramite la scelta del virus e tramite il menù di pausa. Inoltre, permettono di uscire dal gioco. Le classi che costituiscono questo blocco sono mostrate in Figura 3.2.

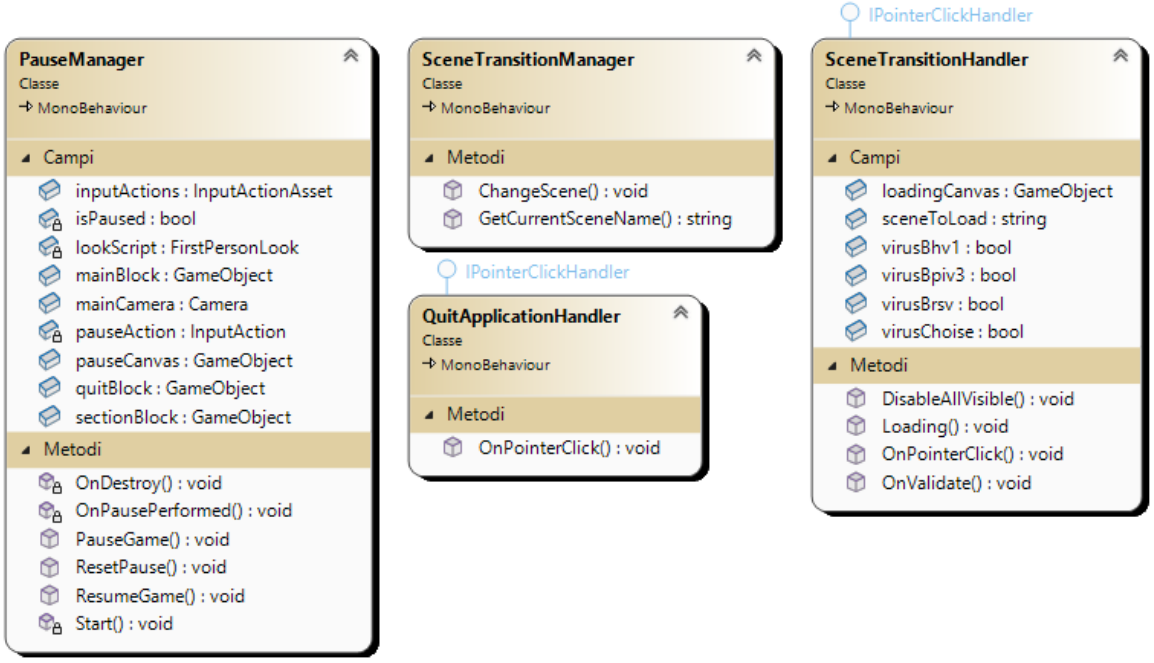


Figura 3.2: Blocco `GameHandler` e relative classi

Attributi		
Nome	**Tipo**	**Descrizione**
images	List<Image>	Lista delle immagini selezionabili
imageToggles	List<Toggle>	Lista dei toggle associati alle immagini
endCanvas	List<GameObject>	Lista delle schermate finali per ciascun virus
verifyButton	Button	Pulsante per verificare le scelte dell'utente
correctMessage	GameObject	Messaggio visualizzato in caso di selezione corretta
wrongMessage	GameObject	Messaggio visualizzato in caso di selezione errata
isMacro	bool	Indica se il contesto è macro (Gross) o micro (Histo)
nextStep	GameObject	Riferimento alla fase successiva del gioco
currentStep	GameObject	Riferimento alla fase corrente del gioco
correctSelection-Count	int	Numero di selezioni corrette effettuate dall'utente
correctSelections-Required	int	Numero di selezioni corrette richieste per completare la fase
scoreText	TMP_Text	Testo che mostra il punteggio e il messaggio iniziale
textDuration	float	Durata in secondi per cui il messaggio di feedback rimane visibile
activeVirus	GameObject	Riferimento al virus attivo nella scena
SceneInformation-ManagerScript	SceneInformation-Manager	Riferimento alla classe che gestisce informazioni globali della scena
Metodi		
Nome	**Tipo**	**Descrizione**
Start()	void	Inizializza il riferimento al virus attivo e imposta i requisiti di selezione in base al contesto (macro/micro)
OnEnable()	void	Reimposta le selezioni e i toggle al momento dell'attivazione dell'oggetto
GetRequired-Selections()	int	Restituisce il numero di selezioni corrette richieste in base al virus attivo e al contesto
VerifySelection()	void	Verifica la correttezza delle selezioni effettuate dall'utente
ShowCorrect-Message()	void	Mostra il messaggio di selezione corretta e avvia il passaggio alla fase successiva
ShowWrongMessage()	void	Mostra il messaggio di selezione errata
HideTextAfter-Delay()	IEnumerator	Nasconde il messaggio di feedback dopo un intervallo di tempo e gestisce il passaggio di fase
ActivateEnd()	void	Attiva il canvas finale associato al virus attivo

Tabella 3.5: Blocco GameLogic: elementi della classe ImageSelectionManager

PauseManager

La classe PauseManager gestisce il menù di pausa del gioco, consentendo di mettere in pausa e riprendere il gioco tramite gli input dei controller. Gli elementi che la costituiscono sono elencati nella Tabella 3.9.

QuitApplicationHandler

La classe QuitApplicationHandler gestisce la chiusura dell'applicazione quando viene cliccato un elemento associato. Gli elementi della classe sono elencati nella Tabella 3.10.

Attributi		
Nome	**Tipo**	**Descrizione**
gameManager	GameManager	Riferimento alla classe GameManager, utilizzata per determinare i virus attivi
bhv1Virus	GameObject	Oggetto 3D associato al virus BHV1
bpiv3Virus	GameObject	Oggetto 3D associato al virus BPIV3
brsvVirus	GameObject	Oggetto 3D associato al virus BRSV
activeVirus	GameObject	Riferimento al virus attivo nella scena
Metodi		
Nome	**Tipo**	**Descrizione**
Start()	void	Determina quale virus attivare in base alle impostazioni del GameManager e assegna il riferimento all'attributo activeVirus

Tabella 3.6: Blocco GameLogic: elementi della classe SceneInformationManager

Attributi		
Nome	**Tipo**	**Descrizione**
buttons	List<Button>	Lista di pulsanti utilizzati per avviare la fase
cow	GameObject	Riferimento all'oggetto che rappresenta la vacca nella scena
cells	GameObject	Riferimento all'oggetto che rappresenta le cellule attivabili
objectsToAble	List<GameObject>	Oggetti da attivare durante la transizione
objectsToDisable	List<GameObject>	Oggetti da disattivare durante la transizione
SceneInformation-ManagerScript	SceneInformation-Manager	Riferimento alla classe per la gestione delle informazioni della scena
Metodi		
Nome	**Tipo**	**Descrizione**
Start()	void	Inizializza il riferimento a Scene-InformationManager e associa i listener ai pulsanti
OnButtonPressed()	void	Attiva le cellule, avvia le animazioni e gestisce l'attivazione/disattivazione degli oggetti nella scena
Activate-Animators()	void	Abilita gli Animator degli oggetti cow e cells
DisableObject()	void	Disattiva tutti gli oggetti nella lista objectsToDisable
DisableActive-VirusMeshRenderer()	void	Disabilita il MeshRenderer del virus attivo nella scena
ActivateObject()	void	Attiva tutti gli oggetti nella lista objectsToAble

Tabella 3.7: Blocco GameLogic: elementi della classe StartThirdIteration

SceneTransitionHandler

La classe SceneTransitionHandler gestisce le transizioni di scena quando viene cliccato un elemento associato, con funzionalità aggiuntive per selezionare uno specifico virus da attivare nel GameManager. Gli elementi della classe sono elencati nella Tabella 3.11.

Attributi		
Nome	**Tipo**	**Descrizione**
cells	List<GameObject>	Lista delle cellule presenti nella scena
correctSelection-Color	Material	Colore da applicare alle cellule selezionate correttamente
wrongSelection-Color	Material	Colore da applicare alle cellule selezionate in modo errato
correctSelections-Required	int	Numero di selezioni corrette richieste per completare l'interazione
scoreText	TMP_Text	Elemento di testo che mostra il punteggio
congratText	GameObject	Messaggio di congratulazioni mostrato al completamento
SceneInformation-ManagerScript	SceneInformation-Manager	Riferimento alla classe che gestisce le informazioni della scena
activeVirus	GameObject	Riferimento al virus attivo nella scena
correctSelection-Count	int	Conteggio delle selezioni corrette effettuate
selectedTags	HashSet<string>	Set di tag delle cellule già selezionate
Metodi		
Nome	**Tipo**	**Descrizione**
Start()	void	Inizializza il riferimento al virus attivo e il numero di selezioni richieste, aggiungendo i listener per le cellule
Update()	void	Aggiorna il testo del punteggio visualizzato all'utente
OnDestroy()	void	Rimuove i listener per le cellule al distruggersi dell'oggetto
GetRequiredSelections(string virusTag)	int	Restituisce il numero di selezioni richieste in base al tag del virus
ResetSelections()	void	Resetta il conteggio delle selezioni corrette e rimuove i tag selezionati
OnCellSelected (SelectEnterEventArgs args)	void	Gestisce l'evento di selezione di una cellula, verificando l'associazione al virus attivo
IlluminateCellsByTag(string cellTag, Material color)	void	Cambia il colore delle cellule con il tag specificato

Tabella 3.8: Blocco GameLogic: elementi della classe ThirdInteraction

SceneTransitionManager

La classe SceneTransitionManager gestisce la transizione tra le scene nel gioco. Gli elementi che la costituiscono sono elencati nella Tabella 3.12.

3.3.3 Virus

Il blocco Virus contiene le classi relative alle dinamiche del virus, come la gestione dello spostamento tramite controller, le animazioni dei virus e, infine, le informazioni relative alle cellule e alle via di trasmissione del virus. Le classi che costituiscono questo blocco sono mostrate in Figura 3.3.

Attributi		
Nome	**Tipo**	**Descrizione**
`mainCamera`	`Camera`	Riferimento alla camera principale nella scena
`pauseCanvas`	`GameObject`	Riferimento al canvas del menù di pausa
`mainBlock`	`GameObject`	Blocco principale visibile nel menù di pausa
`sectionBlock`	`GameObject`	Blocco delle sezioni visibile nel menù di pausa
`quitBlock`	`GameObject`	Blocco per il menù di uscita
`isPaused`	`bool`	Stato del gioco: `true` se il gioco è in pausa
`lookScript`	`FirstPersonLook`	Classe per gestire il movimento della telecamera
`inputActions`	`InputActionAsset`	Asset degli input configurati
`pauseAction`	`InputAction`	Azione associata al comando di pausa
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza i riferimenti agli oggetti, le azioni di input e resetta lo stato del menù di pausa
`OnDestroy()`	`void`	Rimuove il listener per l'azione di pausa quando l'oggetto viene distrutto
`OnPausePerformed (InputAction. CallbackContext context)`	`void`	Alterna tra pausa e ripresa del gioco quando l'azione di pausa è eseguita
`ResetPause()`	`void`	Resetta lo stato del menù di pausa, disattivando il canvas e riabilitando i blocchi predefiniti
`PauseGame()`	`void`	Attiva il menù di pausa, disabilita il tempo di gioco e aggiorna lo stato
`ResumeGame()`	`void`	Disattiva il menù di pausa, ripristina il tempo di gioco e aggiorna lo stato

Tabella 3.9: Blocco `GameHandler`: elementi della classe `PauseManager`

Metodi		
Nome	**Tipo**	**Descrizione**
`OnPointerClick (PointerEventData eventData)`	`void`	Chiude l'applicazione quando si clicca sull'elemento associato

Tabella 3.10: Blocco `GameHandler`: elementi della classe `QuitApplicationHandler`

ActivateVirusAnimator

La classe `ActivateVirusAnimator` gestisce l'animazione e la visualizzazione delle descrizioni dei virus. Quando un pulsante di attivazione viene premuto, attiva l'animazione del virus selezionato e mostra la sua descrizione associata, disabilitando altre descrizioni già visibili. Gli elementi della classe sono elencati nella Tabella 3.13.

CellVirusData

La classe `CellVirusData` gestisce le informazioni relative alle cellule e al loro legame con i virus. Essa memorizza i tag dei virus associati a ciascuna cellula, il materiale originale della cellula e se la cellula è stata selezionata. Gli elementi della classe sono elencati nella Tabella 3.14.

Attributi		
Nome	**Tipo**	**Descrizione**
`sceneToLoad`	`string`	Nome della scena da caricare al click
`loadingCanvas`	`GameObject`	Riferimento al canvas mostrato durante il caricamento della scena
`virusChoise`	`bool`	Determina se è possibile selezionare un virus da attivare
`virusBhv1`	`bool`	Indica se è selezionato il virus Bhv1
`virusBpiv3`	`bool`	Indica se è selezionato il virus Bpiv3
`virusBrsv`	`bool`	Indica se è selezionato il virus Brsv
Metodi		
Nome	**Tipo**	**Descrizione**
`OnValidate()`	`void`	Gestisce la logica di esclusività tra i virus selezionabili. Si attiva automaticamente nell'Editor di Unity quando vengono modificati i valori degli attributi
`OnPointerClick (PointerEventData eventData)`	`void`	Esegue la transizione di scena e gestisce l'attivazione del virus selezionato
`Loading()`	`void`	Disabilita tutti gli oggetti visibili e attiva il canvas di caricamento
`DisableAll-Visible()`	`void`	Disabilita tutti i `Renderer` e `Canvas` attivi nella scena

Tabella 3.11: Blocco `GameHandler`: elementi della classe `SceneTransitionHandler`

Metodi		
Nome	**Tipo**	**Descrizione**
`GetCurrentScene-Name()`	`string`	Restituisce il nome della scena attualmente attiva
`ChangeScene(string sceneName)`	`void`	Carica una nuova scena in base al nome passato come parametro

Tabella 3.12: Blocco `GameHandler`: elementi della classe `SceneTransitionManager`

Attributi		
Nome	**Tipo**	**Descrizione**
`virusMappings`	`List<VirusDescript`	Elenco dei virus e delle relative descrizioni e nomi degli organi associati
`activateButtons`	`List<Button>`	Elenco dei pulsanti che attivano i virus
`descriptionToHide`	`GameObject`	Riferimento al canvas che contiene le descrizioni da nascondere prima di mostrare quella selezionata
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza i pulsanti di attivazione per associarli all'evento di click
`OnActivateButton-Press(Button activateButton)`	`void`	Gestisce l'evento di click sui pulsanti, attivando l'animazione del virus selezionato e visualizzando la sua descrizione

Tabella 3.13: Blocco `Virus`: elementi della classe `ActivateVirusAnimator`

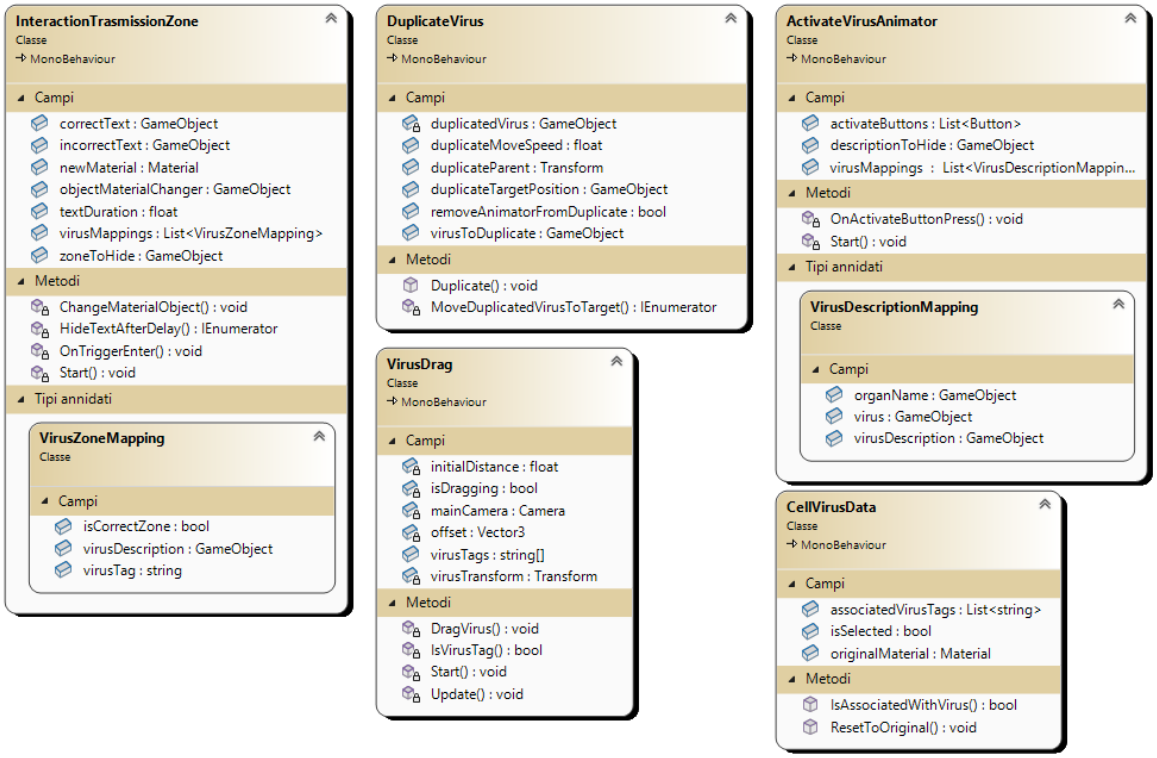


Figura 3.3: Blocco `Virus` e relative classi

Attributi		
Nome	**Tipo**	**Descrizione**
`associatedVirus-Tags`	`List<string>`	Elenco dei tag dei virus associati alla cellula
`originalMaterial`	`Material`	Materiale originale della cellula, che può essere ripristinato
`isSelected`	`bool`	Indica se la cellula è stata selezionata o meno
Metodi		
Nome	**Tipo**	**Descrizione**
`IsAssociatedWith-Virus(string virusTag)`	`bool`	Verifica se la cellula è associata al virus con il tag specificato
`ResetToOriginal()`	`void`	Ripristina il materiale originale della cellula e annulla la selezione

Tabella 3.14: Blocco `Virus`: elementi della classe `CellVirusData`

DuplicateVirus

La classe `DuplicateVirus` gestisce la duplicazione di un virus presente nella scena. Quando viene attivata, crea una copia del virus, la sposta verso una posizione target e, opzionalmente, rimuove l'animatore dal duplicato. Gli elementi della classe sono elencati nella Tabella 3.15.

Attributi		
Nome	**Tipo**	**Descrizione**
`virusToDuplicate`	`GameObject`	Riferimento al virus da duplicare
`duplicateParent`	`Transform`	Trasformazione del genitore del virus duplicato
`duplicateTarget-Position`	`GameObject`	Posizione di destinazione del virus duplicato
`duplicateMoveSpeed`	`float`	Velocità con cui il virus duplicato si muove verso la posizione target
`removeAnimator-FromDuplicate`	`bool`	Determina se rimuovere l'animatore dal virus duplicato
Metodi		
Nome	**Tipo**	**Descrizione**
`Duplicate()`	`void`	Duplica il virus e lo muove verso la posizione target, rimuovendo l'animatore se necessario
`MoveDuplicated-VirusToTarget()`	`IEnumerator`	Sposta il virus duplicato verso la posizione target ad una velocità definita

Tabella 3.15: Blocco `Virus`: elementi della classe `DuplicateVirus`

InteractionTrasmissionZone

La classe `InteractionTrasmissionZone` gestisce la logica di interazione tra le vie di trasmissione dei virus e gli oggetti della scena. Quando un oggetto entra in una zona di interazione, viene verificato se esso è associato al virus corretto, mostrando un testo appropriato e visualizzando gli organi interni del bovino. Gli elementi della classe sono elencati nella Tabella 3.16.

VirusDrag

La classe `VirusDrag` permette di trascinare i virus nella scena tramite un'operazione di "drag and drop" con il controller. La classe gestisce il rilevamento del virus selezionato, calcolando la distanza iniziale tra la camera e l'oggetto, e aggiornando la posizione dell'oggetto durante il trascinamento. Gli elementi della classe sono elencati nella Tabella 3.17.

3.3.4 Utility

Il blocco `Utility` contiene le classi relative alle ultime funzionalità presenti nel gioco, come la dissolvenza tra determinati oggetti dell'interfaccia grafica e la possibilità di scorrere ciclicamente tra una serie di elementi dell'interfaccia, ad esempio le immagini di gross e le immagini histo. Le classi che costituiscono questo blocco sono mostrate in Figura 3.4.

FadeAndSwitchObjects

La classe `FadeAndSwitchObjects` gestisce una transizione visiva tra due oggetti di tipo `GameObject` mediante un effetto di dissolvenza (fade-out e fade-in). Durante la transizione, l'oggetto da disattivare viene oscurato progressivamente e successivamente disattivato, mentre l'oggetto da attivare viene reso gradualmente visibile. Gli elementi della classe sono descritti nella Tabella 3.18.

Attributi		
Nome	**Tipo**	**Descrizione**
`zoneToHide`	`GameObject`	Riferimento all'oggetto da nascondere quando viene selezionata la zona corretta
`correctText`	`GameObject`	Testo da visualizzare quando si entra nella zona corretta
`incorrectText`	`GameObject`	Testo da visualizzare quando si entra nella zona sbagliata
`textDuration`	`float`	Durata per cui il testo corretto o errato sarà visibile
`newMaterial`	`Material`	Materiale semitrasparente da applicare al bovino in modo da rendere visibili gli organi interni
`objectMaterial-Changer`	`GameObject`	Riferimento all'oggetto che deve cambiare materiale
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza gli oggetti di testo e nasconde le descrizioni dei virus
`OnTriggerEnter (Collider other)`	`void`	Gestisce l'interazione tra l'oggetto e la zona, mostrando il testo corretto o errato
`HideTextAfter-Delay(GameObject text, bool isCorrect)`	`IEnumerator`	Nasconde il testo dopo un ritardo
`ChangeMaterial-Object()`	`void`	Cambia il materiale del bovino

Tabella 3.16: Blocco `Virus`: elementi della classe `InteractionTrasmissionZone`

Attributi		
Nome	**Tipo**	**Descrizione**
`virusTags`	`string[]`	Array di tag dei virus che devono essere trascinati
`mainCamera`	`Camera`	Riferimento alla camera principale della scena
`isDragging`	`bool`	Indica se l'oggetto è attualmente in fase di trascinamento
`offset`	`Vector3`	Distanza tra la posizione dell'oggetto e la posizione del controller al momento del click
`virusTransform`	`Transform`	Riferimento al `Transform` dell'oggetto virus selezionato
`initialDistance`	`float`	Distanza iniziale tra la camera e il virus
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza il riferimento alla camera principale
`Update()`	`void`	Gestisce gli input del controller per selezionare e trascinare il virus
`IsVirusTag(string tag)`	`bool`	Verifica se il tag dell'oggetto corrisponde a uno dei tag definiti per il virus
`DragVirus()`	`void`	Gestisce il movimento del virus durante il trascinamento, aggiornando la sua posizione

Tabella 3.17: Blocco `Virus`: elementi della classe `VirusDrag`

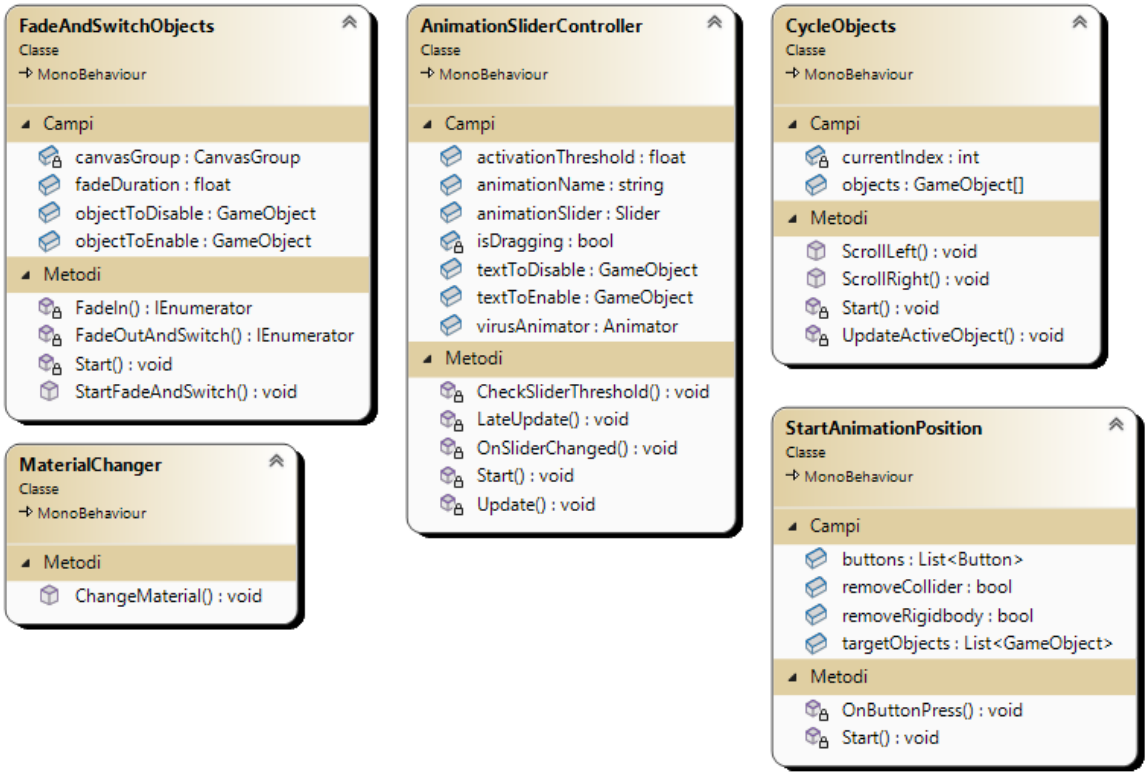


Figura 3.4: Blocco `Utility` e relative classi

Attributi		
Nome	**Tipo**	**Descrizione**
`objectToEnable`	`GameObject`	L'oggetto che verrà attivato durante la transizione
`objectToDisable`	`GameObject`	L'oggetto che verrà disattivato durante la transizione
`fadeDuration`	`float`	La durata della dissolvenza, espressa in secondi
`canvasGroup`	`CanvasGroup`	Componente utilizzato per gestire la trasparenza (`alpha`) dell'oggetto da disattivare
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza il `CanvasGroup` per l'oggetto da disattivare, se non è già presente
`StartFadeAndSwitch()`	`void`	Avvia la transizione di dissolvenza e il passaggio tra gli oggetti
`FadeOutAndSwitch()`	`IEnumerator`	Gestisce la dissolvenza in uscita (`fade-out`) dell'oggetto da disattivare e la dissolvenza in entrata (`fade-in`) dell'oggetto da attivare
`FadeIn(CanvasGroup targetCanvasGroup)`	`IEnumerator`	Applica l'effetto di dissolvenza in entrata (`fade-in`) a un oggetto target specifico

Tabella 3.18: Blocco `Utility`: elementi della classe `FadeAndSwitchObjects`

AnimationSliderController

La classe `AnimationSliderController` permette di sincronizzare un'animazione con uno `Slider` di Unity. L'animazione può essere controllata manualmente tramite lo slider, e

un testo può essere abilitato o disabilitato in base al valore dello slider rispetto a una soglia. Gli elementi della classe sono descritti nella Tabella 3.19.

Attributi		
Nome	**Tipo**	**Descrizione**
`virusAnimator`	`Animator`	L'animatore che gestisce l'animazione da sincronizzare con lo slider
`animationSlider`	`Slider`	Lo slider utilizzato per controllare l'animazione
`animationName`	`string`	Il nome dell'animazione da riprodurre e sincronizzare
`isDragging`	`bool`	Indica se l'utente sta trascinando lo slider
`textToDisable`	`GameObject`	Il testo che verrà disabilitato quando il valore dello slider supera una soglia
`textToEnable`	`GameObject`	Il testo che verrà abilitato quando il valore dello slider supera una soglia
`activation-Threshold`	`float`	Soglia oltre la quale viene abilitato/disabilitato il testo
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza lo stato del testo disabilitando `textToEnable` e registra il listener per il valore dello slider
`Update()`	`void`	Sincronizza lo slider con l'animazione e controlla la soglia di attivazione
`OnSliderChanged (float value)`	`void`	Gestisce il trascinamento dello slider, sincronizzando l'animazione con il valore dello slider
`LateUpdate()`	`void`	Riabilita la riproduzione dell'animazione quando lo slider non viene più trascinato
`CheckSlider-Threshold(float sliderValue)`	`void`	Verifica se il valore dello slider ha superato la soglia e abilita/disabilita i testi di conseguenza

Tabella 3.19: Blocco `Utility`: elementi della classe `AnimationSliderController`

CycleObjects

La classe `CycleObjects` consente di scorrere ciclicamente tra una serie di oggetti attivandone uno alla volta. È possibile muoversi a sinistra o a destra nella lista degli oggetti. Gli attributi e i metodi della classe sono descritti nella Tabella 3.20.

MaterialChanger

La classe `MaterialChanger` consente di modificare dinamicamente il materiale di un oggetto con un nuovo materiale fornito. Gli attributi e i metodi della classe sono descritti nella Tabella 3.21.

StartAnimationPosition

La classe `StartAnimationPosition` permette di spostare e ruotare una lista di oggetti target alla posizione e rotazione dell'oggetto che contiene questa classe quando viene premuto un pulsante associato. Inoltre, fornisce opzioni per rimuovere componenti come `Rigidbody` e `Collider` dagli oggetti target. Gli attributi e i metodi della classe sono descritti nella Tabella 3.22.

Attributi		
Nome	**Tipo**	**Descrizione**
`objects`	`GameObject[]`	Un array di oggetti tra cui scorrere ciclicamente
`currentIndex`	`int`	Indice corrente che indica quale oggetto è attivo
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Inizializza l'oggetto attivo in base all'indice corrente
`ScrollLeft()`	`void`	Scorre a sinistra nella lista degli oggetti, attivando l'oggetto precedente
`ScrollRight()`	`void`	Scorre a destra nella lista degli oggetti, attivando l'oggetto successivo
`UpdateActive-Object()`	`void`	Aggiorna lo stato degli oggetti, attivando solo quello corrispondente all'indice corrente

Tabella 3.20: Blocco `Utility`: elementi della classe `CycleObjects`

Metodi		
Nome	**Tipo**	**Descrizione**
`ChangeMaterial (Material newMaterial)`	`void`	Cambia il materiale dell'oggetto corrente con il materiale fornito come parametro. Se l'oggetto non ha un componente `Renderer`, il metodo non ha effetto.

Tabella 3.21: Blocco `Utility`: elementi della classe `MaterialChanger`

Attributi		
Nome	**Tipo**	**Descrizione**
`targetObjects`	`List<GameObject>`	La lista di oggetti che saranno spostati e ruotati.
`buttons`	`List<Button>`	La lista di pulsanti che attivano l'operazione sugli oggetti target.
`removeRigidbody`	`bool`	Indica se rimuovere i componenti `Rigidbody` dagli oggetti target.
`removeCollider`	`bool`	Indica se disattivare i componenti `Collider` degli oggetti target.
Metodi		
Nome	**Tipo**	**Descrizione**
`Start()`	`void`	Registra gli eventi dei pulsanti associati alla lista `buttons`, collegandoli al metodo `OnButtonPress`.
`OnButtonPress (Button pressedButton)`	`void`	Quando un pulsante viene premuto, sposta gli oggetti target alla posizione e rotazione dell'oggetto corrente. Rimuove `Rigidbody`, `Collider` e componenti opzionali come `XRGrabInteractable`, se specificato.

Tabella 3.22: Blocco `Utility`: elementi della classe `StartAnimationPosition`

3.4 Diagramma di attività

I diagrammi di attività permettono di rappresentare in modo grafico il flusso delle azioni, delle attività e delle decisioni che caratterizzano un processo o un sistema. Essi offrono una visione chiara e organizzata di come le diverse operazioni si sviluppano nel tempo e si interconnettono tra loro. In particolare, nell'ambito dello sviluppo software, vengono utilizzati per delineare il flusso delle operazioni all'interno di un programma o di un'applicazione, facilitando la comprensione delle interazioni tra le diverse componenti del sistema e della sequenzialità con cui vengono eseguite. Nella Figura 3.5 è mostrato il diagramma di attività che evidenzia la sequenza di azioni basilari che compongono il gioco.

Nella Figura 3.6 è mostrato, invece, il diagramma di attività che illustra le possibili sequenze di azioni pertinenti al menù di pausa.

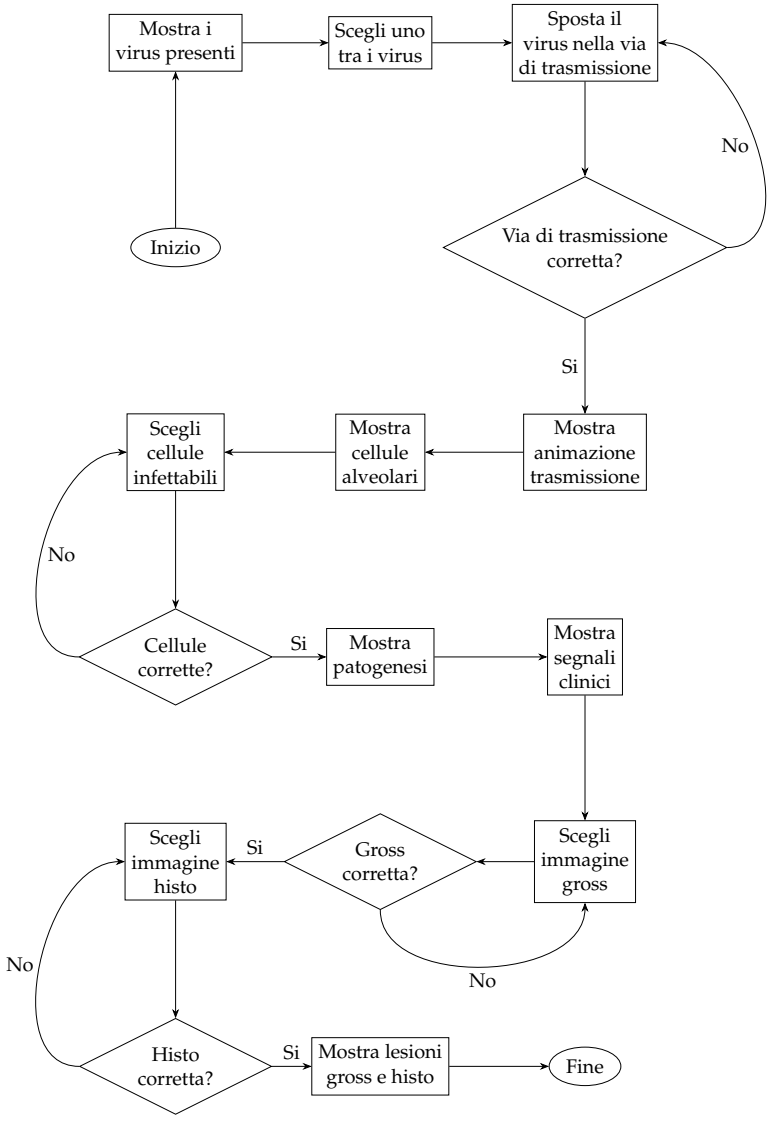


Figura 3.5: Diagramma di attività del gioco

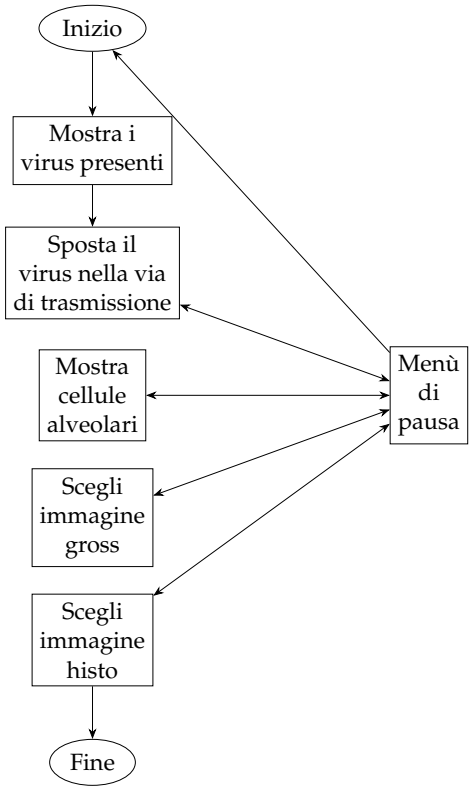


Figura 3.6: Diagramma di attività del menù di pausa

3.5 Diagramma dei casi d'uso

Un caso d'uso rappresenta una descrizione dettagliata delle aspettative e delle azioni che un utente intende compiere durante un'interazione specifica con un sistema. Si tratta di uno strumento fondamentale per comprendere e modellare il comportamento desiderato del software, poiché definisce chiaramente gli obiettivi dell'utente e il modo in cui il sistema dovrebbe rispondere a tali obiettivi. Nel contesto del nostro serious game, questa sezione è dedicata all'analisi dei casi d'uso, con l'obiettivo di identificare le diverse modalità di interazione che caratterizzano l'esperienza utente e le funzionalità chiave che il gioco deve offrire per soddisfare tali aspettative.

3.5.1 Selezione del virus

Questa prima sottosezione ha lo scopo di illustrare i casi d'uso circa la selezione del virus, come mostrato in Figura 3.7.

CU1 - Visualizzazione del virus

Questo caso d'uso si verifica all'avvio del gioco quando il giocatore, prima di effettuare la scelta del virus con cui giocare, può visionarli tutti, scorrendoli. Le informazioni relative sono mostrate nella Tabella 3.23.

CU2 - Selezione del virus

Questo caso d'uso si verifica quando il giocatore sceglie il virus con il quale giocare. Le informazioni relative sono mostrate nella Tabella 3.24.

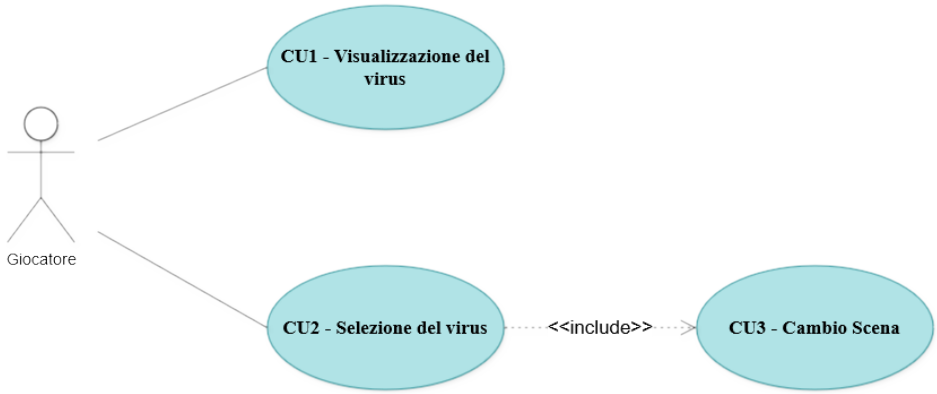


Figura 3.7: Diagramma dei casi d'uso relativi alla selezione del virus

Elementi	Descrizione
Pre-Condizioni	Nessuna
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore preme il pulsante *Play* 2. Il sistema mostra il modello e le informazioni dei virus
Sequenza eventi alternativa	Nessuna

Tabella 3.23: Caso d'uso relativo alla visualizzazione del virus

Elementi	Descrizione
Pre-Condizioni	Nessuna
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore sceglie il virus con cui giocare premendo il pulsante *Start* 2. Il sistema salva il tag del virus selezionato
Sequenza eventi alternativa	Nessuna

Tabella 3.24: Caso d'uso relativo alla selezione del virus

CU3 - Cambio scena

Questo caso d'uso si verifica quando il giocatore ha già scelto il virus con cui giocare. Le informazioni relative sono mostrate nella Tabella 3.25.

3.5.2 Movimento del virus

Questa sottosezione ha lo scopo di illustrare i casi d'uso circa il movimento del virus nella scena, come mostrato in Figura 3.8.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato un virus
Post-Condizioni	Il giocatore si trova in una scena diversa (stalla)
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha scelto il virus con cui giocare 2. Il sistema carica la seconda scena, portando il giocatore all'interno di una stalla
Sequenza eventi alternativa	Nessuna

Tabella 3.25: Caso d'uso relativo al cambio scena

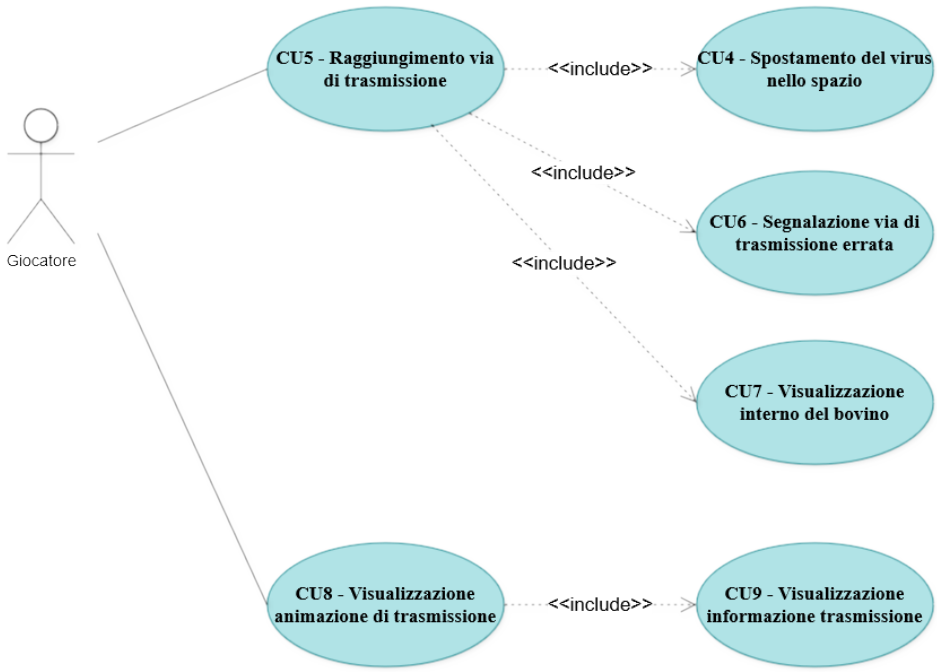


Figura 3.8: Diagramma dei casi d'uso relativi al movimento del virus

CU4 - Spostamento del virus nello spazio

Questo caso d'uso si verifica quando il giocatore muove il virus, prima indicandolo con il controller e premendo l'apposito tasto, poi muovendolo nella scena. Le informazioni relative sono mostrate nella Tabella 3.26.

CU5 - Raggiungimento via di trasmissione

Questo caso d'uso si verifica quando il giocatore muove il virus verso la via di trasmissione. Le informazioni relative sono mostrate nella Tabella 3.27.

CU6 - Segnalazione via di trasmissione errata

Questo caso d'uso si verifica quando il giocatore raggiunge la via di trasmissione errata. Le informazioni relative sono mostrate nella Tabella 3.28.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato un virus
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore indica e seleziona il virus con il controller 2. Il sistema sposta il virus nella zona indicata dai controller
Sequenza eventi alternativa	Nessuna

Tabella 3.26: Caso d'uso relativo allo spostamento del virus nello spazio

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato un virus
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore arriva ad una delle vie di trasmissione 2. Il sistema controlla che la via di trasmissione raggiunta sia quella corretta 3. Il sistema mostra l'anatomia del bovino
Sequenza eventi alternativa	3. Il sistema mostra un messaggio di errore in quanto la via di trasmissione non è quella corretta

Tabella 3.27: Caso d'uso relativo al raggiungimento della via di trasmissione

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato un virus e raggiunto la via di trasmissione errata
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore arriva alla via di trasmissione errata 2. Il sistema mostra un messaggio di errore
Sequenza eventi alternativa	Nessuna

Tabella 3.28: Caso d'uso relativo alla segnalazione della via di trasmissione errata

CU7 - Visualizzazione interno del bovino

Questo caso d'uso si verifica quando il giocatore raggiunge la via di trasmissione corretta. Le informazioni relative sono mostrate nella Tabella 3.29.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato un virus e raggiunto la via di trasmissione corretta
Post-Condizioni	L'anatomia del bovino è mostrata nella scena al posto del bovino stesso
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore arriva alla via di trasmissione corretta 2. Il sistema mostra l'anatomia del bovino al posto del modello del bovino
Sequenza eventi alternativa	Nessuna

Tabella 3.29: Caso d'uso relativo alla visualizzazione dell'interno del bovino

CU8 - Visualizzazione animazione di trasmissione

Questo caso d'uso si verifica quando il giocatore raggiunge la via di trasmissione corretta. Le informazioni relative sono mostrate nella Tabella 3.30.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato un virus e raggiunto la via di trasmissione corretta
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore preme il pulsante *Next* dopo aver raggiunto la via di trasmissione corretta 2. Il sistema fa partire l'animazione corretta 3. Il sistema consente al giocatore di mettere in pausa e riprendere la visione dell'animazione, e consente anche di spostarsi avanti e/o indietro nella timeline
Sequenza eventi alternativa	4. Il sistema consente al giocatore di premere un nuovo pulsante *Next* per saltare l'animazione

Tabella 3.30: Caso d'uso relativo alla visualizzazione dell'animazione di trasmissione

CU9 - Visualizzazione informazione trasmissione

Questo caso d'uso si verifica quando il giocatore raggiunge la via di trasmissione corretta. Le informazioni relative sono mostrate nella Tabella 3.31.

3.5.3 Selezione delle cellule

Questa sottosezione ha lo scopo di illustrare i casi d'uso circa la selezione delle cellule, come mostrato in Figura 3.9.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato un virus e raggiunto la via di trasmissione corretta
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore preme il pulsante *Next* dopo aver raggiunto la via di trasmissione corretta 2. Il sistema mostra le informazioni corrette
Sequenza eventi alternativa	Nessuna

Tabella 3.31: Caso d'uso relativo alla visualizzazione delle informazioni sulla via di trasmissione

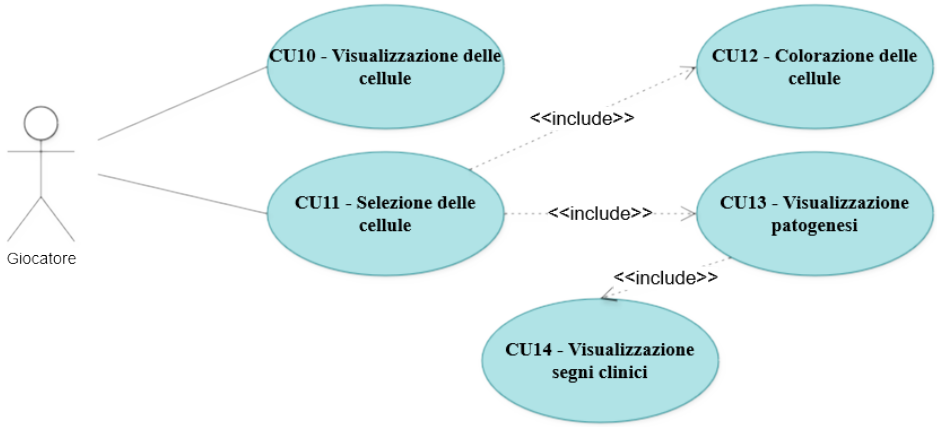


Figura 3.9: Diagramma dei casi d'uso relativi alla selezione delle cellule

CU10 - Visualizzazione delle cellule

Questo caso d'uso si verifica in seguito alla visualizzazione dell'animazione sulla trasmissione del virus. Le informazioni relative sono mostrate nella Tabelle 3.32.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver raggiunto la via di trasmissione corretta
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha terminato la visione dell'animazione di trasmissione 2. Il sistema mostra l'insieme delle cellule
Sequenza eventi alternativa	Nessuna

Tabella 3.32: Caso d'uso relativo alla visualizzazione delle cellule

CU11 - Selezione delle cellule

Questo caso d'uso si verifica in seguito alla visualizzazione dell'animazione sulla trasmissione del virus. Le informazioni relative sono mostrate nella Tabella 3.33.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver raggiunto la via di trasmissione corretta
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore seleziona una cellula tramite controller 2. Il sistema controlla se la cellula selezionata è corretta 3. Il sistema colora le cellule selezionate
Sequenza eventi alternativa	Nessuna

Tabella 3.33: Caso d'uso relativo selezione delle cellule

CU12 - Colorazione delle cellule

Questo caso d'uso si verifica in seguito alla selezione della cellula da parte del giocatore, cambiando colorazione della cellula. Le informazioni relative sono mostrate nella Tabella 3.34.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato una cellula
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha selezionato una cellula 2. Il sistema controlla se la cellula selezionata è corretta 3. Il sistema colora di verde la cellula corretta aumentando il punteggio di uno 4. Il sistema mostra una schermata di congratulazioni
Sequenza eventi alternativa	3. Il sistema colora di rosso la cellula sbagliata 4. Il sistema rimane nella stessa schermata mantenendo il punteggio precedente

Tabella 3.34: Caso d'uso relativo alla colorazione delle cellule

CU13 - Visualizzazione patogenensi

Questo caso d'uso si verifica in seguito alla selezione corretta di tutte le cellule. Le informazioni relative sono mostrate nella Tabella 3.35.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato tutte le cellule corrette
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha selezionato tutte le cellule corrette 2. Il sistema mostra la schermata riguardante la patogenesi del virus
Sequenza eventi alternativa	Nessuna

Tabella 3.35: Caso d'uso relativo alla visualizzazione della patogenesi del virus

CU14 - Visualizzazione segni clinici

Questo caso d'uso si verifica in seguito alla selezione corretta di tutte le cellule e alla visualizzazione della patogenesi. Le informazioni relative sono mostrate nella Tabella 3.36.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato tutte le cellule corrette e visualizzato la patogenesi
Post-Condizioni	Il giocatore visualizza la schermata delle lesioni
Sequenza eventi principali	1. Il caso d'uso inizia una volta che il giocatore ha visualizzato la patogenesi 2. Il sistema mostra la schermata riguardante i segni clinici del virus
Sequenza eventi alternativa	Nessuna

Tabella 3.36: Caso d'uso relativo alla visualizzazione dei segni clinici

3.5.4 Selezione delle lesioni

Questa sottosezione ha lo scopo di illustrare i casi d'uso circa la selezione delle lesioni, come mostrato in Figura 3.10. Questa tipologia di caso d'uso si ripete due volte, una per le macro-lesioni ed una per le micro-lesioni.

CU15 - Visualizzazione delle lesioni

Questo caso d'uso si verifica in seguito alla visualizzazione dei segni clinici del virus. Le informazioni relative sono mostrate nella Tabella 3.37.

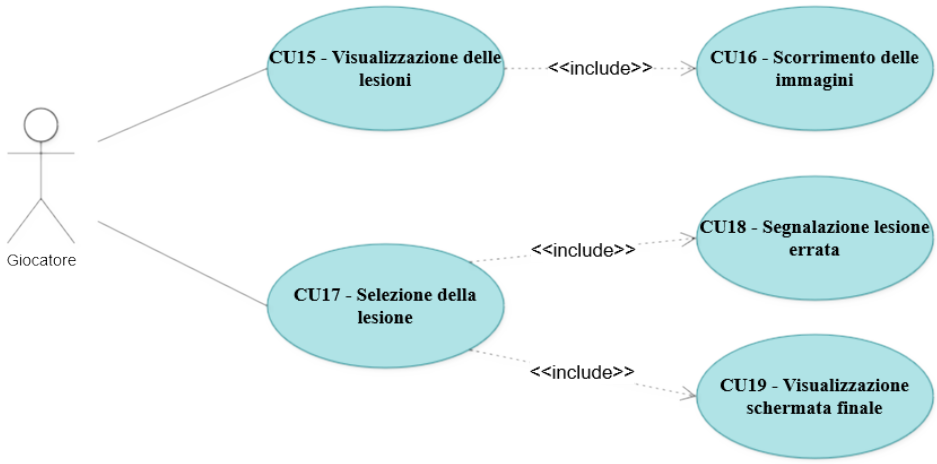


Figura 3.10: Diagramma dei casi d'uso relativi alla selezione delle lesioni

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato le cellule corrette e visualizzato i segni clinici
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha visualizzato la schermata riguardante i segni clinici 2. Il sistema mostra le immagini relative alle lesioni
Sequenza eventi alternativa	Nessuna

Tabella 3.37: Caso d'uso relativo alla visualizzazione delle lesioni

CU16 - Scorrimento delle immagini

Questo caso d'uso si verifica quando il giocatore si trova nella schermata relativa alle lesioni e clicca sulle frecce per scorrere le immagini. Le informazioni relative sono mostrate nella Tabella 3.38.

Elementi	Descrizione
Pre-Condizioni	Il giocatore si deve trovare nella schermata di selezione della lesione
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore clicca le frecce per spostarsi tra le immagini 2. Il sistema cicla tra le immagini presenti
Sequenza eventi alternativa	Nessuna

Tabella 3.38: Caso d'uso relativo allo scorrimento delle immagini

CU17 - Selezione della lesione

Questo caso d'uso si verifica quando il giocatore decide qual è la lesione corretta tra quelle presenti. Le informazioni relative sono mostrate nella Tabella 3.39.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato una lesione
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore seleziona una, o più, immagini tra le lesioni presenti 2. Il sistema controlla che la lesione (o le lesioni) selezionata sia quella corretta 3. Il sistema mostra la schermata successiva
Sequenza eventi alternativa	3. Il sistema mostra un messaggio di errore in quanto la lesione non è quella corretta 4. Il sistema rimane nella schermata attuale invitando il giocatore a riprovare

Tabella 3.39: Caso d'uso relativo alla selezione della lesione

CU18 - Segnalazione lesione errata

Questo caso d'uso si verifica quando il giocatore seleziona la lesione errata. Le informazioni relative sono mostrate nella Tabella 3.40.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver selezionato la lesione errata
Post-Condizioni	Il giocatore rimane nella schermata corrente
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore seleziona la lesione errata 2. Il sistema mostra un messaggio di errore 3. Il sistema continua a mostrare la schermata corrente (fino a che il giocatore non seleziona la lesione corretta)
Sequenza eventi alternativa	Nessuna

Tabella 3.40: Caso d'uso relativo alla segnalazione della selezione di una lesione errata

CU19 - Visualizzazione schermata finale

Questo caso d'uso si verifica quando il giocatore seleziona la lesione corretta. Le informazioni relative sono mostrate nella Tabella 3.41.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver selezionato la lesione corretta
Post-Condizioni	Il giocatore visualizza la schermata finale
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore seleziona la lesione corretta 2. Il sistema mostra la schermata finale
Sequenza eventi alternativa	Nessuna

Tabella 3.41: Caso d'uso relativo alla visualizzazione della schermata finale

3.5.5 Menù di pausa

Quest'ultima sottosezione ha lo scopo di illustrare i casi d'uso circa le funzionalità del menù di pausa, come mostrato in Figura 3.10.

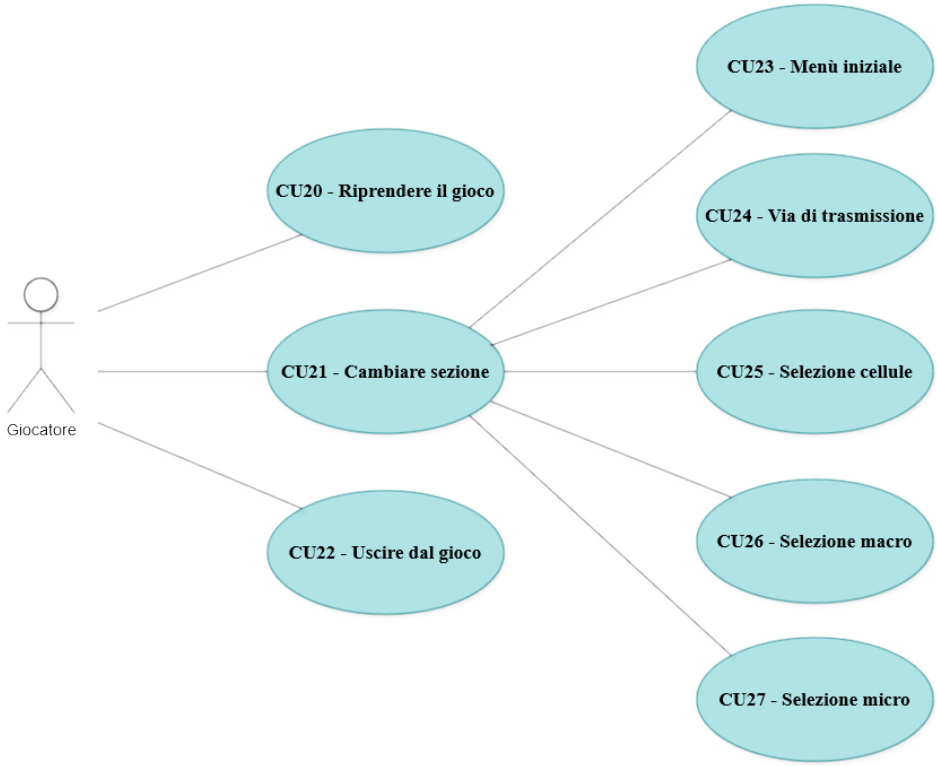


Figura 3.11: Diagramma dei casi d'uso relativi alla selezione del menù di pausa

CU20 - Riprendere il gioco

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller. Le informazioni relative sono mostrate nella Tabella 3.42.

CU21 - Cambiare sezione

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller. Le informazioni relative sono mostrate nella Tabella 3.43.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Il giocatore riprende a giocare
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il tasto per mettere il gioco in pausa 2. Il sistema mostra il menù di pausa 3. Il giocatore seleziona il pulsante di ripresa del gioco
Sequenza eventi alternativa	3. Il giocatore ripreme il tasto del menù di pausa sul controller

Tabella 3.42: Caso d'uso relativo alla ripresa del gioco nel menù

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il tasto per mettere il gioco in pausa 2. Il sistema mostra il menù di pausa 3. Il giocatore seleziona il pulsante per cambiare sezione
Sequenza eventi alternativa	Nessuna

Tabella 3.43: Caso d'uso relativo al cambio di sezione nel menù

CU22 - Uscire dal gioco

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller. Le informazioni relative sono mostrate nella Tabella 3.44.

CU23 - Menù iniziale

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller e ha scelto di cambiare sezione. Le informazioni relative sono mostrate nella Tabella 3.45.

CU24 - Via di trasmissione

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller e ha scelto di cambiare sezione. Le informazioni relative sono mostrate nella Tabella 3.46.

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il tasto per mettere il gioco in pausa 2. Il sistema mostra il menù di pausa 3. Il giocatore seleziona il pulsante di uscita 4. Il sistema chiede conferma di uscita 5. Il giocatore conferma di voler uscire dal gioco 6. Il sistema chiude il gioco
Sequenza eventi alternativa	5. Il giocatore non conferma di voler uscire dal gioco 6. Il sistema rimane nella schermata del menù

Tabella 3.44: Caso d'uso relativo all'uscita dal gioco nel menù

Elementi	**Descrizione**
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il pulsante per cambiare sezione 2. Il sistema mostra le sezioni disponibili 3. Il giocatore seleziona il menù iniziale 4. Il sistema porta il giocatore nel punto d'inizio del gioco (quando deve scegliere il virus)
Sequenza eventi alternativa	3. Il giocatore seleziona il pulsante indietro 4. Il sistema rimane nella schermata del menù

Tabella 3.45: Caso d'uso relativo alla scelta del menù iniziale nel menù

CU25 - Sezione cellule

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller e ha scelto di cambiare sezione. Le informazioni relative sono mostrate nella Tabella 3.47.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il pulsante per cambiare sezione 2. Il sistema mostra le sezioni disponibili 3. Il giocatore seleziona la via di trasmissione 4. Il sistema porta il giocatore nel punto in cui deve muovere il virus nello spazio (quando deve scegliere la via di trasmissione)
Sequenza eventi alternativa	3. Il giocatore seleziona il pulsante indietro 4. Il sistema rimane nella schermata del menù

Tabella 3.46: Caso d'uso relativo alla scelta della via di trasmissione nel menù

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il pulsante per cambiare sezione 2. Il sistema mostra le sezioni disponibili 3. Il giocatore seleziona le cellule 4. Il sistema porta il giocatore nel punto in cui deve selezionare le cellule
Sequenza eventi alternativa	3. Il giocatore seleziona il pulsante indietro 4. Il sistema rimane nella schermata del menù

Tabella 3.47: Caso d'uso relativo alla scelta delle cellule nel menù

CU26 - Sezione macro

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller e ha scelto di cambiare sezione. Le informazioni relative sono mostrate nella Tabella 3.48.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il pulsante per cambiare sezione 2. Il sistema mostra le sezioni disponibili 3. Il giocatore seleziona le macro 4. Il sistema porta il giocatore nel punto in cui deve selezionare le immagini di gross
Sequenza eventi alternativa	3. Il giocatore seleziona il pulsante indietro 4. Il sistema rimane nella schermata del menù

Tabella 3.48: Caso d'uso relativo alla scelta delle macro nel menù

CU27 - Sezione micro

Questo caso d'uso si verifica una volta che il giocatore ha premuto il pulsante del menù sul controller e ha scelto di cambiare sezione. Le informazioni relative sono mostrate nella Tabella 3.49.

Elementi	Descrizione
Pre-Condizioni	Il giocatore deve aver premuto il tasto menù sul controller
Post-Condizioni	Nessuna
Sequenza eventi principali	1. Il caso d'uso inizia quando il giocatore ha premuto il pulsante per cambiare sezione 2. Il sistema mostra le sezioni disponibili 3. Il giocatore seleziona le micro 4. Il sistema porta il giocatore nel punto in cui deve selezionare le immagini histo
Sequenza eventi alternativa	3. Il giocatore seleziona il pulsante indietro 4. Il sistema rimane nella schermata del menù

Tabella 3.49: Caso d'uso relativo alla scelta delle micro nel menù

CAPITOLO 4

Implementazione

In questo capitolo verrà trattata l'implementazione del serious game oggetto di questa tesi. Si mostreranno i tool utilizzati nello sviluppo, quindi la creazione dei modelli 3D e, infine, lo sviluppo delle funzionalità principali.

4.1 Practical Design

Questa sezione è dedicata alla descrizione degli strumenti e delle tecnologie adottati per lo sviluppo del serious game. Verranno illustrati i tool impiegati durante il processo di creazione, nonché il linguaggio di programmazione scelto. In particolare verranno illustrati:

- Unity
- Blender
- Visual Studio Code
- C#

4.1.1 Unity

Unity rappresenta una piattaforma di sviluppo software all'avanguardia, composta da un potente game engine, un ambiente di sviluppo integrato (IDE) e una suite di servizi che abilitano la creazione di esperienze interattive di alto livello. Grazie alle sue capacità avanzate, Unity consente lo sviluppo di videogiochi in 2D e 3D, applicazioni multipiattaforma e ambienti immersivi di realtà virtuale.

Il motore fisico di Unity è particolarmente robusto, supportando dinamiche del corpo rigido, rilevamento delle collisioni e tecniche di raycasting, che consentono di progettare ambienti realistici e interattivi. Questo approccio permette agli sviluppatori di integrare oggetti che rispondono in modo accurato e coinvolgente alle azioni dell'utente, migliorando l'esperienza complessiva.

L'architettura basata sui componenti, un pilastro di Unity, offre un'enorme flessibilità: le funzionalità possono essere facilmente aggiunte, rimosse o personalizzate attraverso l'associazione di componenti (come script, controller di personaggi o effetti visivi) ai GameObject presenti nel progetto.

Dal punto di vista grafico, Unity offre strumenti di alto livello per creare mondi di gioco estremamente realistici. Tra le principali funzionalità grafiche si annoverano:

- *Illuminazione* dinamica e statica, per ambienti credibili;
- *Ombre* di alta qualità, con supporto per configurazioni avanzate;
- *Riflessi* in tempo reale e ambientali, per migliorare il dettaglio visivo;
- *Effetti di post-elaborazione*, come anti-aliasing, bloom e profondità di campo, che aumentano il livello di immersione.

Un altro punto di forza di Unity è il suo ampio supporto multipiattaforma, che include sistemi operativi e dispositivi come Windows, Mac, iOS, Android, console e visori VR. Questo consente agli sviluppatori di distribuire i propri prodotti su una vasta gamma di piattaforme con sforzi di adattamento minimi.

Unity offre strumenti dedicati per creare esperienze altamente immersive in realtà virtuale. Grazie al supporto per visori VR, come Oculus, HTC Vive e altri dispositivi compatibili, gli sviluppatori possono progettare mondi virtuali interattivi che rispondono in tempo reale ai movimenti e alle azioni dell'utente, sfruttando l'integrazione con SDK specifici. Questo rende Unity una scelta ideale per lo sviluppo di applicazioni VR sia in ambito ludico che educativo o professionale.

4.1.2 Blender

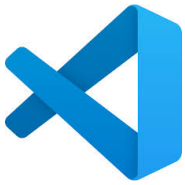
Blender è un software open-source e multipiattaforma progettato per la modellazione, il rigging, l'animazione, il montaggio video, la composizione, il rendering e il texturing di immagini tridimensionali e bidimensionali.

Questo strumento offre una vasta gamma di funzionalità, supportato da una comunità attiva di utenti e sviluppatori.

Blender può essere utilizzato per numerosi scopi, tra cui:

- *Modellazione 3D*: consente di creare modelli complessi e dettagliati di oggetti, personaggi e ambienti.
- *Animazione*: è possibile animare i modelli 3D per ottenere movimenti fluidi e realistici, grazie a una vasta gamma di strumenti e opzioni per l'animazione.
- *Rendering*: il motore di rendering di Blender permette di produrre immagini fotorealistiche dei modelli 3D. Esso include funzionalità avanzate per la gestione dell'illuminazione, degli shader e per ottenere risultati di alta qualità.
- *Simulazione*: Blender offre strumenti per simulare fenomeni fisici, come fluidi, particelle e tessuti; esso, inoltre, consente di effettuare simulazioni di fisica avanzata.
- *Compositing e post-produzione*: Blender dispone di un potente nodo di composizione che consente di combinare diversi passaggi di rendering, applicare effetti speciali, correggere i colori ed effettuare editing video.
- *Effetti visivi (VFX)*: Blender è utilizzato nell'industria cinematografica per creare effetti visivi complessi e realistici, come esplosioni, fuoco, simulazioni di massa e molto altro.
- *Sviluppo di giochi*: Blender supporta la creazione di giochi 3D grazie a strumenti dedicati per modellazione, animazione e rendering specifici per il game development.

4.1.3 Visual Studio Code



Visual Studio Code (o, più semplicemente, VS Code) è un editor di codice sorgente open-source sviluppato da Microsoft, compatibile con i sistemi operativi Windows, Linux e macOS.

VS Code include una serie di funzionalità avanzate, tra cui:

- *Debugging*: permette di eseguire il debug del codice direttamente dall'editor.
- *Controllo Git integrato*: fornisce strumenti per il controllo di versione direttamente nell'interfaccia.
- *Syntax Highlighting*: supporta l'evidenziazione della sintassi per numerosi linguaggi di programmazione.
- *IntelliSense*: offre il completamento automatico intelligente, suggerimenti e informazioni sui parametri.
- *Snippet*: consente di inserire rapidamente frammenti di codice predefiniti.
- *Refactoring del codice*: fornisce strumenti per modificare e ottimizzare il codice in modo rapido ed efficace.

Unity supporta l'integrazione nativa con Visual Studio Code, consentendo agli sviluppatori di utilizzarlo come editor esterno predefinito per gli script, sia su Windows sia su macOS. Inoltre, Visual Studio Code permette di installare numerose estensioni che ampliano le sue funzionalità, rendendolo uno strumento versatile e potente per lo sviluppo di applicazioni e videogiochi.

4.1.4 C#

C# è un linguaggio di programmazione orientato agli oggetti, sviluppato e mantenuto da Microsoft. Questo linguaggio, che condivide molte somiglianze con C++ e Java, è ampiamente riconosciuto per la sua versatilità e potenza.

C# è uno dei linguaggi principali per la manipolazione degli oggetti all'interno del framework .NET, rendendolo una scelta ideale per lo sviluppo di applicazioni robuste, scalabili e performanti. Grazie alla sua sintassi intuitiva e al forte supporto da parte di Microsoft, C# è diventato uno strumento chiave per lo sviluppo di software, inclusi videogiochi, applicazioni desktop, web e mobile.

4.2 Creazione dei modelli 3D

Questa sezione è dedicata alla creazione e modellazione dei vari modelli 3D non disponibili sul web, quali l'anatomia del bovino e le diverse cellule presenti nell'epitelio respiratorio.

4.2.1 Anatomia del bovino

Per la modellazione dell'anatomia si è partiti da un modello preesistente di un bovino. Al modello è stato applicato un materiale che lo rendesse parzialmente trasparente, in modo da garantire una chiara visibilità degli organi interni.

Successivamente, sono stati modellati i diversi organi interni, adottando un approccio basato su ricerche preliminari, volte a raccogliere immagini di riferimento e ispirazioni visive, con l'obiettivo di rendere il modello il più accurato possibile.

In particolare:

- La lingua è stata modellata interamente ex novo per garantirne la fedeltà anatomica.
- Il cervello e l'apparato respiratorio, inizialmente basati su modelli preesistenti, sono stati adattati e modellati per essere compatibili con la struttura anatomica del bovino.
- L'apparato digerente è stato inserito e modellato in modo da rappresentare correttamente le sue peculiarità nel bovino.

Dopo aver perfezionato la forma e la struttura dell'anatomia, è stata dedicata una fase all'applicazione di materiali e texture, per rendere il modello finale visivamente realistico e accattivante. L'utilizzo di texture realistiche ha ulteriormente contribuito a simulare con precisione i tessuti dei vari organi, migliorando la qualità e la fedeltà visiva del modello.

L'elaborazione del modello è stata sviluppata tramite l'utilizzo di Blender. In Figura 4.1 è mostrato il modello nell'editor di Unity.

Figura 4.1: Modello dell'anatomia del bovino

4.2.2 Cellule epitelio respiratorio e alveoli

La modellazione delle cellule dell'epitelio respiratorio e degli alveoli è stata condotta interamente da zero, utilizzando Blender. Questa fase è stata preceduta da un'approfondita ricerca preliminare, finalizzata a raccogliere dati e immagini di riferimento per garantire un giusto compromesso tra accuratezza scientifica nella rappresentazione delle strutture cellulari e usabilità da parte dell'utente.

Il processo di creazione è iniziato con la modellazione della struttura di base delle cellule, prestando particolare attenzione alle caratteristiche morfologiche tipiche dell'epitelio

respiratorio e degli alveoli. Per l'epitelio respiratorio sono state modellate cellule cilindriche dotate di ciglia apicali e cellule cilindriche semplici, nonché cellule con diversa forma, come ad esempio le cellule basali. Mentre per gli alveoli sono state create unità cellulari in grado di rappresentare con precisione gli pneumociti di tipo I e tipo II, elementi chiave nella funzione respiratoria.

Successivamente, si è proceduto al perfezionamento delle mesh, ottimizzandone la geometria per mantenere un elevato livello di dettaglio senza compromettere le prestazioni computazionali del serious game. Durante questa fase, sono stati applicati materiali, utilizzando texture procedurali per differenziare le diverse tipologie di cellule.

Infine, per assicurare una rappresentazione visivamente fedele e scientificamente accurata, ogni cellula è stata dotata di un nucleo interno. L'elaborazione del modello è stata sviluppata tramite l'utilizzo di Blender. In Figura 4.2 è mostrato il modello nell'editor di Unity.

Figura 4.2: Modello delle cellule

4.2.3 Stalla

Il modello della stalla utilizzato nel serious game è stato sviluppato a partire da un modello tridimensionale preesistente, al quale sono state apportate modifiche strutturali e spaziali per adattarlo alle esigenze specifiche del progetto.

Le prime modifiche hanno riguardato la struttura generale del modello: sono stati ridefiniti gli spazi interni ed esterni per ottimizzarne l'utilizzo e renderlo coerente con il contesto didattico e interattivo del gioco. La disposizione degli oggetti, come stalle, recinzioni e altri elementi ambientali, è stata riorganizzata per migliorare la navigabilità e l'interazione all'interno dell'ambiente virtuale.

Successivamente, sono state aggiunte nuove mesh per arricchire il livello di dettaglio visivo e migliorare l'immersività del modello. Grazie a queste modifiche, il modello della stalla risulta un ambiente virtuale realistico, immersivo e funzionale, in grado di supportare efficacemente gli obiettivi educativi del serious game.

4.3 Implementazione delle funzionalità principali

Nella fase di sviluppo del serious game, l'implementazione delle funzionalità gioca un ruolo cruciale nel conferire interattività e dinamicità all'esperienza dell'utente. Per aggiungere nuove funzionalità in Unity, è necessario scrivere script in C# che gestiscano il comportamento degli oggetti nel gioco o eseguano altre operazioni specifiche. Una volta creato lo script, esso va associato all'oggetto a cui si desidera applicare la funzionalità.

Nel contesto dello sviluppo del nostro serious game, le funzionalità implementate sono contenute nei seguenti script:

- `ActivatePathogenesis`
- `ActivateVirusAnimator`
- `AnimationSliderController`
- `CellVirusData`
- `ChooseSectionPause`
- `CycleObjects`
- `DuplicateVirus`
- `EpiManager`
- `FadeAndSwitchObjects`
- `GameManager`
- `ImageSelectionManager`
- `InteractionTrasmissionZone`
- `MaterialChanger`
- `PauseManager`
- `QuitApplicationHandler`
- `SceneInformationManager`
- `SceneTransitionHandler`
- `SceneTransitionManager`
- `StartAnimationPosition`
- `StartThirdIteration`
- `ThirdInteraction`

Nel seguito esamineremo in dettaglio ciascuna di queste funzionalità.

4.3.1 ActivatePathogenesis

Lo script `ActivatePathogenesis`, collegato all'oggetto `ThirdInteractionManager`, è utilizzato per attivare la patogenesi del virus attivo nella scena, ovvero quello scelto dal giocatore, utilizzando il tag del virus come discriminante.

La funzione `Start()` (Listato 4.1) viene eseguita automaticamente all'avvio dello script e svolge due compiti principali. In primo luogo, si assicura che la variabile `SceneInformationManagerScript` contenga un riferimento valido all'oggetto `SceneInformationManager` presente nella scena. Questo viene ottenuto tramite un'operazione di controllo: se la variabile è già inizializzata, viene mantenuta intatta; altrimenti, viene assegnato un riferimento utilizzando il metodo `FindObjectOfType<SceneInformationManager>()`. In secondo luogo, la funzione registra un listener sull'evento di click del pulsante `activateButton`, collegandolo al metodo `OnButtonPressed()`.

```
1  private void Start()
2  {
3      SceneInformationManagerScript = SceneInformationManagerScript !=
           null ? SceneInformationManagerScript : FindObjectOfType<
           SceneInformationManager>();
4
5      activateButton.onClick.AddListener(OnButtonPressed);
6  }
```

Listato 4.1: Funzione che gestisce il pulsante per la patogenesi

La funzione `OnButtonPressed()` (Listato 4.2) rappresenta il punto di ingresso per le interazioni dell'utente. Quando il pulsante viene premuto, questa funzione viene invocata per determinare quale oggetto attivare in base al virus attivo nella scena. Il primo passo consiste nel recuperare il riferimento al virus attivo tramite la variabile `SceneInformationManagerScript.activeVirus`. Se il valore restituito non è `null`, vengono eseguite due operazioni: anzitutto, tutti gli oggetti nella lista `objectsToDisable` vengono disattivati richiamando il metodo `DisableObjects()`; successivamente, viene invocato il metodo `ActivateObjectForVirus()`, passando come parametro il tag del virus attivo per identificare quale oggetto specifico attivare.

```
1  private void OnButtonPressed()
2  {
3      GameObject activeVirus = SceneInformationManagerScript.activeVirus;
4
5      if (activeVirus != null)
6      {
7          DisableObjects();
8          ActivateObjectForVirus(activeVirus.tag);
9      }
10 }
```

Listato 4.2: Funzione che si attiva alla pressione del pulsante

La funzione `ActivateObjectForVirus()` (Listato 4.3) utilizza una struttura `switch` per verificare il valore del tag del virus ricevuto in ingresso. In base al tag, viene attivato l'oggetto corrispondente tra `pathoBhv1`, `pathoBpiv3` e `pathoBrsv`, utilizzando il metodo `SetActive(true)`. Qualora il tag non corrisponda a nessuno dei casi specificati, la funzione non esegue alcuna operazione.

```
private void ActivateObjectForVirus(string virusTag)
{
    switch (virusTag)
    {
        case "Bhv1":
            pathoBhv1.SetActive(true);
            break;
        case "Bpiv3":
            pathoBpiv3.SetActive(true);
            break;
        case "Brsv":
            pathoBrsv.SetActive(true);
            break;
        default:
            break;
    }
}
```

Listato 4.3: Funzione che trova la patogenesi corretta

Infine, la funzione `DisableObjects()` (Listato 4.4) si occupa di iterare attraverso tutti gli oggetti inclusi nella lista `objectsToDisable` e di disattivarli uno per uno, chiamando il metodo `SetActive(false)` su ciascun oggetto. Ciò garantisce che tutti gli oggetti non pertinenti vengano nascosti prima di attivare quello specifico per il virus corrente.

```
private void DisableObjects()
{
    foreach (GameObject obj in objectsToDisable)
    {
        obj.SetActive(false);
    }
}
```

Listato 4.4: Funzione che disabilita gli oggetti

4.3.2 ActivateVirusAnimator

Lo script `ActivateVirusAnimator` gestisce l'attivazione delle animazioni e delle descrizioni associate ai virus nella scena tramite l'interazione con pulsanti dedicati.

La funzione `Start()` (Listato 4.5) viene eseguita al momento dell'inizializzazione dello script. In questa fase, a ciascun pulsante di attivazione presente nella lista viene associata un'azione da eseguire quando il pulsante viene premuto. Questa azione è gestita dal metodo `OnActivateButtonPress()`.

```
private void Start()
{
    foreach (Button activateButton in activateButtons)
    {
        activateButton.onClick.AddListener(() => OnActivateButtonPress(
            activateButton));
    }
}
```

Listato 4.5: Funzione che gestisce il pulsante per l'animazione

La funzione `OnActivateButtonPress()` (Listato 4.6) viene richiamata al momento della pressione di uno dei pulsanti. Prima di tutto, viene nascosta la precedente descrizione per garantire che sia visibile solo quella pertinente all'animazione. Successivamente, vengono esaminati i virus presenti nella scena per identificare quale di essi sia attualmente attivo. Una volta trovato, il metodo abilita l'animazione associata al virus nonché le descrizioni e i dettagli correlati, permettendo, così, di visualizzare le informazioni pertinenti in modo dinamico.

```
1  private void OnActivateButtonPress(Button activateButton)
2  {
3     descriptionToHide.SetActive(false);
4     if (activateButton.gameObject.activeSelf)
5     {
6        foreach (var mapping in virusMappings)
7        {
8           if (mapping.virus != null && mapping.virus.activeSelf)
9           {
10             if (mapping.virus.TryGetComponent<Animator>(out var
                   virusAnimator))
11             {
12                virusAnimator.enabled = true;
13                mapping.virusDescription.SetActive(true);
14                mapping.organName.SetActive(true);
15             }
16             break;
17          }
18       }
19    }
20 }
```

Listato 4.6: Funzione che gestisce la pressione dei pulsanti

4.3.3 AnimationSliderController

Lo script `AnimationSliderController` collega l'animazione di un virus a uno `Slider` nell'interfaccia utente, consentendo di sincronizzare la posizione dello `Slider` con il progresso dell'animazione e di controllare dinamicamente quest'ultima.

La funzione `Start()` (Listato 4.7) disabilita inizialmente il testo associato e collega l'evento di modifica dello `Slider` al metodo `OnSliderChanged()`, che gestisce il comportamento durante l'interazione con lo `Slider`.

```
1  private void Start()
2  {
3     textToEnable.SetActive(false);
4     animationSlider.onValueChanged.AddListener(OnSliderChanged);
5  }
```

Listato 4.7: Funzione che configura le impostazioni iniziali

La funzione `Update()` (Listato 4.8) viene chiamata ad ogni frame e gestisce l'aggiornamento automatico dello `Slider` in base allo stato dell'animazione, a meno che non sia in corso un'interazione diretta da parte del giocatore. Inoltre, verifica la soglia di attivazione dello `Slider` utilizzando il metodo `CheckSliderThreshold()` e disabilita l'animazione quando questa raggiunge il completamento.

```
private void Update()
{
   if (!isDragging)
   {
      AnimatorStateInfo currentState = virusAnimator.
          GetCurrentAnimatorStateInfo(0);

      float normalizedTime = currentState.normalizedTime % 1.0f;
      animationSlider.SetValueWithoutNotify(normalizedTime);

      CheckSliderThreshold(normalizedTime);

      if (normalizedTime >= 0.99f)
      {
         virusAnimator.enabled = false;
      }
   }
}
```

Listato 4.8: Funzione che aggiorna automaticamente lo stato dello `Slider`

La funzione `OnSliderChanged()` (Listato 4.9) viene eseguita ogni volta che lo `Slider` subisce una modifica. Durante l'interazione, l'animazione viene sincronizzata con il valore dello `Slider` e viene aggiornato lo stato dell'interfaccia in base alla soglia di attivazione.

```
private void OnSliderChanged(float value)
{
   isDragging = true;

   if (!virusAnimator.enabled)
   {
      virusAnimator.enabled = true;
   }

   virusAnimator.speed = 0;
   virusAnimator.Play(animationName, 0, value);

   CheckSliderThreshold(value);
}
```

Listato 4.9: Funzione che gestisce le modifiche allo `Slider`

La funzione `LateUpdate()` (Listato 4.10) rileva il rilascio del pulsante del mouse/controller e, se l'utente stava interagendo con lo `Slider`, ripristina la velocità dell'animazione consentendogli di riprendere normalmente.

```
private void LateUpdate()
{
   if (!Input.GetMouseButton(0) && isDragging)
   {
      isDragging = false;
      virusAnimator.speed = 1;
   }
}
```

Listato 4.10: Funzione che ripristina la velocità dell'animazione dopo l'interazione

La funzione `CheckSliderThreshold()` (Listato 4.11) verifica se il valore dello `Slider` ha superato una determinata soglia. Se la soglia viene superata, disattiva un testo predefinito e ne attiva un altro, fornendo un feedback visivo all'utente.

```
private void CheckSliderThreshold(float sliderValue)
{
    if (sliderValue >= activationThreshold)
    {
        if (textToDisable != null)
        {
            textToDisable.SetActive(false);
        }
        if (textToEnable != null)
        {
            textToEnable.SetActive(true);
        }
    }
}
```

Listato 4.11: Funzione che gestisce il comportamento basato sulla soglia dello `Slider`

4.3.4 CellVirusData

Lo script `CellVirusData` fornisce un'implementazione per gestire i dati relativi alle cellule nella scena, in particolare per verificarne l'associazione con determinati virus e ripristinare il loro stato originale.

La funzione `IsAssociatedWithVirus()` (Listato 4.12) consente di determinare se una cellula è associata a un determinato virus. Il metodo accetta il tag del virus come argomento e verifica se questo è contenuto nella lista dei tag associati alla cellula.

```
public bool IsAssociatedWithVirus(string virusTag)
{
    return associatedVirusTags.Contains(virusTag);
}
```

Listato 4.12: Funzione che verifica l'associazione della cellula con un virus

La funzione `ResetToOriginal()` (Listato 4.13) ripristina lo stato originale della cellula. Questo include il ritorno al materiale di default assegnato e la reimpostazione del flag `isSelected` a `false`, indicando che la cellula non è attualmente selezionata.

```
public void ResetToOriginal()
{
    GetComponent<Renderer>().material = originalMaterial;
    isSelected = false;
}
```

Listato 4.13: Funzione che ripristina lo stato originale della cellula

4.3.5 ChooseSectionPause

Lo script `ChooseSectionPause` gestisce la logica per l'uscita dal menù di pausa e la modifica dello stato di attivazione di specifici oggetti nella scena.

La funzione `Start()` (Listato 4.14) si occupa di assicurare che il riferimento all'oggetto `StartThirdIteration` sia valido. Se non è stato assegnato manualmente, il metodo utilizza `FindObjectOfType<>()` per individuarlo dinamicamente nella scena.

```
private void Start()
{
    startThirdIterationScript = startThirdIterationScript != null ?
        startThirdIterationScript : FindObjectOfType<StartThirdIteration
        >();
}
```

Listato 4.14: Funzione che inizializza il riferimento a `StartThirdIteration`

La funzione `ExitPauseAndToggleObjects()` (Listato 4.15) gestisce l'uscita dal menù di pausa e la transizione tra diverse sezioni del gioco. Il metodo esegue le seguenti operazioni:

- disattiva il menu di pausa se il riferimento è valido;
- attiva tutti gli oggetti inclusi nella lista `objectsToActivate`;
- disattiva tutti gli oggetti inclusi nella lista `objectsToDeactivate`;
- se il flag `disableMesh` è impostato su `true`, richiama il metodo `DisableActiveVirusMeshRenderer()` dello script associato a `startThirdIterationScript`;
- ripristina il tempo di gioco impostando `Time.timeScale` a 1.

```
public void ExitPauseAndToggleObjects()
{
    if (pauseMenu != null)
    {
        pauseMenu.SetActive(false);
    }
    foreach (var obj in objectsToActivate)
    {
        if (obj != null)
        {
            obj.SetActive(true);
        }
    }

    foreach (var obj in objectsToDeactivate)
    {
        if (obj != null)
        {
            obj.SetActive(false);
        }
    }

    if (disableMesh)
    {
        startThirdIterationScript.DisableActiveVirusMeshRenderer();
    }

    Time.timeScale = 1f;
}
```

Listato 4.15: Funzione che gestisce l'uscita dal menù di pausa e l'attivazione/disattivazione degli oggetti

4.3.6 CycleObjects

Lo script `CycleObjects` permette di navigare ciclicamente tra un insieme di oggetti, attivandone uno alla volta e disattivando gli altri. La gestione avviene tramite lo scorrimento verso sinistra o destra, con un meccanismo che assicura il ritorno al primo elemento dopo l'ultimo, e viceversa.

La funzione `Start()` (Listato 4.16) richiama il metodo `UpdateActiveObject()` per attivare inizialmente l'oggetto corrispondente all'indice `currentIndex`, che di default è impostato a 0.

```
private void Start()
{
    UpdateActiveObject();
}
```

Listato 4.16: Funzione di inizializzazione dello stato attivo degli oggetti

La funzione `ScrollLeft()` (Listato 4.17) viene utilizzata per scorrere verso sinistra nella lista di oggetti. L'indice corrente `currentIndex` viene decrementato e, se si supera il limite inferiore (valore negativo), viene riposizionato sull'ultimo elemento della lista. Successivamente, il metodo `UpdateActiveObject()` viene chiamato per aggiornare lo stato degli oggetti.

```
public void ScrollLeft()
{
    currentIndex--;
    if (currentIndex < 0)
    {
        currentIndex = objects.Length - 1;
    }

    UpdateActiveObject();
}
```

Listato 4.17: Funzione per scorrere verso sinistra nella lista di oggetti

La funzione `ScrollRight()` (Listato 4.18) consente di scorrere verso destra nella lista di oggetti. In questo caso, l'indice `currentIndex` viene incrementato e, se si supera il limite superiore della lista, viene riportato al primo elemento. Anche qui, il metodo `UpdateActiveObject()` viene invocato per aggiornare lo stato degli oggetti.

```
public void ScrollRight()
{
    currentIndex++;
    if (currentIndex >= objects.Length)
    {
        currentIndex = 0;
    }

    UpdateActiveObject();
}
```

Listato 4.18: Funzione per scorrere verso destra nella lista di oggetti

La funzione `UpdateActiveObject()` (Listato 4.19) è responsabile dell'aggiornamento dello stato degli oggetti. Essa itera attraverso la lista `objects` e attiva solo l'oggetto corrispondente al valore corrente di `currentIndex`, disattivando tutti gli altri.

```
1 private void UpdateActiveObject()
2 {
3    for (int i = 0; i < objects.Length; i++)
4    {
5       objects[i].SetActive(i == currentIndex);
6    }
7 }
```

Listato 4.19: Funzione per aggiornare lo stato degli oggetti

4.3.7 DuplicateVirus

Lo script `DuplicateVirus` è progettato per duplicare un oggetto virus nella scena, per spostare la copia in una posizione predefinita e per rimuovere l'animatore dal duplicato, se necessario. Questo script permette di avere animazioni, circa la trasmissione del virus, più realistiche, in modo da vedere come i virus si moltiplicano all'interno del bovino.

La funzione `Duplicate()` (Listato 4.20) viene utilizzata per creare una copia dell'oggetto `virusToDuplicate`. Se la variabile `duplicatedVirus` non è già impostata e `virusToDuplicate` è valido, viene istanziata una copia del virus nella posizione e rotazione originale dell'oggetto. Se è stato definito un `duplicateParent`, il duplicato viene assegnato come figlio di quest'ultimo. Se la variabile `removeAnimatorFromDuplicate` è impostata su `true`, l'animatore del duplicato viene rimosso. Infine, viene avviata una coroutine `MoveDuplicatedVirus- ToTarget()` per spostare il duplicato verso la destinazione predefinita.

```
1  public void Duplicate()
2  {
3     if (duplicatedVirus == null && virusToDuplicate != null)
4     {
5        duplicatedVirus = Instantiate(virusToDuplicate, virusToDuplicate.
             transform.position, virusToDuplicate.transform.rotation);
6
7        if (duplicateParent != null)
8        {
9           duplicatedVirus.transform.SetParent(duplicateParent);
10       }
11
12       if (removeAnimatorFromDuplicate)
13       {
14          if (duplicatedVirus.TryGetComponent<Animator>(out var animator
                ))
15          {
16             Destroy(animator);
17          }
18       }
19
20       StartCoroutine(MoveDuplicatedVirusToTarget());
21    }
22 }
```

Listato 4.20: Funzione che duplica il virus e avvia il movimento verso la destinazione

La funzione `MoveDuplicatedVirusToTarget()` (Listato 4.21) è una coroutine che si occupa di spostare il duplicato verso la posizione di destinazione `duplicateTarget-Position`. La posizione del duplicato viene aggiornata progressivamente, utilizzando `Vector3.Move- Towards()` per avvicinarsi alla destinazione con una velocità definita dalla variabile `duplicateMove- Speed`. La coroutine continua a spostare il duplicato fino a quando la distanza dalla destinazione è inferiore a una soglia definita (0.1f).

```
1 private IEnumerator MoveDuplicatedVirusToTarget()
2 {
3    while (Vector3.Distance(duplicatedVirus.transform.position,
         duplicateTargetPosition.transform.position) > 0.1f)
4    {
5       duplicatedVirus.transform.position = Vector3.MoveTowards(
           duplicatedVirus.transform.position, duplicateTargetPosition.
           transform.position, duplicateMoveSpeed * Time.deltaTime);
6       yield return null;
7    }
8 }
```

Listato 4.21: Coroutine che sposta il duplicato verso la destinazione

4.3.8 EpiManager

Lo script `EpiManager` gestisce l'interazione tra il giocatore e le cellule presenti nel gioco, garantendo che, quando l'oggetto `EpiManager` viene abilitato, le selezioni siano azzerate e le cellule tornino al loro stato originale.

Nella funzione `Start()` (Listato 4.22), se il riferimento alla variabile `thirdInteraction-Script` non è stato impostato precedentemente, viene cercato un oggetto di tipo `Third-Interaction` nella scena e associato a questa variabile.

```
1 private void Start()
2 {
3    if (thirdInteractionScript == null)
4    {
5       thirdInteractionScript = FindObjectOfType<ThirdInteraction>();
6    }
7 }
```

Listato 4.22: Funzione che inizializza il riferimento al terzo script di interazione

La funzione `OnEnable()` (Listato 4.23) viene chiamata quando l'oggetto `EpiManager` viene attivato. In questa fase, il metodo `ResetSelections()` dello script `thirdInterac-tionScript` viene richiamato per azzerare qualsiasi selezione precedentemente fatta dal giocatore. Inoltre, tutte le celle di tipo `CellVirusData`, contenute come figli dell'oggetto `EpiManager`, vengono ripristinate al loro stato originale chiamando il metodo `ResetToOri-ginal()` su ciascuna di esse.

```
1 private void OnEnable()
2 {
3    thirdInteractionScript.ResetSelections();
4
5    CellVirusData[] cells = GetComponentsInChildren<CellVirusData>();
6    foreach (var cell in cells)
7    {
```

```
8           cell.ResetToOriginal();
9       }
10  }
```

Listato 4.23: Funzione che ripristina le celle e le selezioni quando l'oggetto viene abilitato

4.3.9 FadeAndSwitchObjects

Lo script `FadeAndSwitchObjects` gestisce la transizione tra due oggetti nella scena, applicando una dissolvenza (*fade*) durante il passaggio di visibilità, in cui uno degli oggetti viene disabilitato e l'altro abilitato.

La funzione `Start()` (Listato 4.24) verifica se l'oggetto da disabilitare `objectToDisable` ha già un componente `CanvasGroup` per gestire l'alpha (trasparenza). Se non presente, ne aggiunge uno. Questo componente è necessario per applicare la dissolvenza sugli oggetti UI.

```
1   private void Start()
2   {
3       if (!objectToDisable.TryGetComponent<CanvasGroup>(out canvasGroup))
4       {
5           canvasGroup = objectToDisable.AddComponent<CanvasGroup>();
6       }
7   }
```

Listato 4.24: Funzione di inizializzazione per il componente CanvasGroup

La funzione `StartFadeAndSwitch()` (Listato 4.25) avvia la coroutines `FadeOutAndSwitch()`, che si occupa di eseguire la dissolvenza e il cambio degli oggetti. In particolare, questa coroutine riduce progressivamente l'alpha di `objectToDisable`, fino a renderlo invisibile, quindi abilita `objectToEnable` e ne aumenta l'alpha per farlo apparire.

```
1   public void StartFadeAndSwitch()
2   {
3       StartCoroutine(FadeOutAndSwitch());
4   }
```

Listato 4.25: Funzione che avvia la dissolvenza e il cambio degli oggetti

La funzione `FadeOutAndSwitch()` (Listato 4.26) gestisce la dissolvenza di `objectToDisable` e l'abilitazione di `objectToEnable`. Se `objectToEnable` non ha un `CanvasGroup`, ne viene aggiunto uno, quindi l'oggetto viene reso visibile tramite la funzione `FadeIn()`.

```
1   private IEnumerator FadeOutAndSwitch()
2   {
3       float elapsedTime = 0.0f;
4
5       while (elapsedTime < fadeDuration)
6       {
7           elapsedTime += Time.deltaTime;
8           canvasGroup.alpha = 1.0f - (elapsedTime / fadeDuration);
9           yield return null;
10      }
11
12      canvasGroup.alpha = 0.0f;
13      if (objectToEnable != null)
```

```
14      {
15          if (!objectToEnable.TryGetComponent<CanvasGroup>(out var
                newCanvasGroup))
16          {
17              newCanvasGroup = objectToEnable.AddComponent<CanvasGroup>();
18          }
19
20          newCanvasGroup.alpha = 0.0f;
21          objectToEnable.SetActive(true);
22          yield return StartCoroutine(FadeIn(newCanvasGroup));
23      }
24      objectToDisable.SetActive(false);
25  }
```

Listato 4.26: Funzione che gestisce la dissolvenza e il cambio degli oggetti

La funzione `FadeIn()` (Listato 4.27) aumenta gradualmente l'alpha dell'oggetto `targetCanvasGroup`, rendendolo visibile durante il processo di dissolvenza in entrata. Questa funzione viene chiamata dopo che l'oggetto da abilitare è stato attivato.

```
1   private IEnumerator FadeIn(CanvasGroup targetCanvasGroup)
2   {
3       float elapsedTime = 0.0f;
4       while (elapsedTime < fadeDuration)
5       {
6           elapsedTime += Time.deltaTime;
7           targetCanvasGroup.alpha = elapsedTime / fadeDuration;
8           yield return null;
9       }
10      targetCanvasGroup.alpha = 1.0f;
11  }
```

Listato 4.27: Funzione che applica la dissolvenza in entrata

4.3.10 GameManager

Lo script `GameManager` assicura che ci sia una sola istanza dell'oggetto `GameManager` durante il ciclo di vita del gioco. Esso gestisce anche le variabili che permettono di informare il sistema su quale virus è attivo nella scena.

La funzione `Awake()` (Listato 4.30) viene eseguita quando l'oggetto viene inizializzato. In questa funzione viene controllato se esiste già un'istanza di `GameManager`. Se esiste, essa distrugge l'oggetto corrente, garantendo che ci sia una sola istanza del `GameManager`. Inoltre, imposta l'oggetto per non essere distrutto al cambio di scena.

```
1   private void Awake()
2   {
3       if (instance != null && instance != this)
4       {
5           Destroy(this.gameObject);
6           return;
7       }
8       instance = this;
9       DontDestroyOnLoad(this.gameObject);
10  }
```

Listato 4.28: Funzione di inizializzazione dell'istanza Singleton

Le proprietà `ActivateBhv1`, `ActivateBpiv3`, e `ActivateBrsv` (Listato 4.31) sono utilizzate per gestire lo stato di attivazione di specifiche funzionalità del gioco. Ogni proprietà ha un `getter` e un `setter` che permettono di accedere e modificare i valori booleani privati `activateBhv1`, `activateBpiv3`, e `activateBrsv`.

```
public bool ActivateBhv1
{
    get { return activateBhv1; }
    set { activateBhv1 = value; }
}
public bool ActivateBpiv3
{
    get { return activateBpiv3; }
    set { activateBpiv3 = value; }
}
public bool ActivateBrsv
{
    get { return activateBrsv; }
    set { activateBrsv = value; }
}
```

Listato 4.29: Proprietà per la gestione dello stato di attivazione

4.3.11 ImageSelectionManager

Lo script `ImageSelectionManager` gestisce la logica di selezione delle immagini tramite `Toggle` e verifica se la selezione è corretta in base al virus attivo. Esso gestisce anche il punteggio e la visualizzazione di messaggi di successo o errore.

La funzione `Start()` (Listato 4.30) cerca il virus attivo e inizializza i valori relativi alla quantità di selezioni corrette richieste e al testo del punteggio da visualizzare. Inoltre, aggiunge un listener al pulsante `verifyButton` per avviare la verifica della selezione.

```
private void Start()
{
    SceneInformationManagerScript = SceneInformationManagerScript !=
        null ? SceneInformationManagerScript : FindObjectOfType<
        SceneInformationManager>();

    activeVirus = SceneInformationManagerScript.activeVirus;
    correctSelectionsRequired = GetRequiredSelections(activeVirus.tag);

    if (isMacro)
    {
        if (scoreText != null)
        {
            scoreText.text = "Choose the correct " +
                correctSelectionsRequired + " Gross";
        }
    }
    else
    {
        if (scoreText != null)
        {
            scoreText.text = "Choose the correct " +
                correctSelectionsRequired + " Histo";
        }
    }
```

```
22      verifyButton.onClick.AddListener(VerifySelection);
23  }
```

Listato 4.30: Funzione di inizializzazione iniziale e listener al pulsante

La funzione `OnEnable()` (Listato 4.31) viene chiamata quando lo script viene abilitato; in questa funzione vengono reimpostate le selezioni corrette e i `Toggle` associati alle immagini. Il conteggio delle selezioni corrette viene azzerato.

```
1   private void OnEnable()
2   {
3       correctSelectionCount = 0;
4       foreach (Toggle toggle in imageToggles)
5       {
6           toggle.isOn = false;
7       }
8   }
```

Listato 4.31: Funzione di inizializzazione delle selezioni all'abilitazione

La funzione `GetRequiredSelections()` (Listato 4.32) determina quante selezioni corrette sono richieste in base al tag del virus attivo. Questa funzione restituisce un valore intero che indica il numero di selezioni corrette necessarie per il virus corrente.

```
1   private int GetRequiredSelections(string virusTag)
2   {
3       if (isMacro)
4       {
5           return virusTag switch
6           {
7               "Bhv1" => 1,
8               "Bpiv3" => 1,
9               "Brsv" => 2,
10              _ => 0,
11          };
12      }
13      else
14      {
15          return virusTag switch
16          {
17              "Bhv1" => 1,
18              "Bpiv3" => 1,
19              "Brsv" => 1,
20              _ => 0,
21          };
22      }
23  }
```

Listato 4.32: Funzione per determinare le selezioni corrette richieste

La funzione `VerifySelection()` (Listato 4.33) verifica se la selezione effettuata tramite i `Toggle` è corretta. Per ogni immagine selezionata, confronta il tag dell'immagine con quello del virus attivo. Se tutte le selezioni sono corrette, viene visualizzato un messaggio di successo e, in caso di selezione delle immagini "Macro", viene attivato il passo successivo, ovvero la selezione delle immagini "Micro". In caso contrario, se ci sono selezioni sbagliate, viene visualizzato un messaggio di errore.

```
private void VerifySelection()
{
   correctSelectionCount = 0;
   string activeVirusTag = activeVirus.tag;

   for (int i = 0; i < imageToggles.Count; i++)
   {
      if (imageToggles[i].isOn)
      {
         if (images[i].CompareTag(activeVirusTag))
         {
            correctSelectionCount++;
         }
         else
         {
            ShowWrongMessage();
            return;
         }
      }
   }
   if (correctSelectionCount == correctSelectionsRequired)
   {
      ShowCorrectMessage();
      if (!isMacro)
      {
         ActivateEnd();
      }
   }
   else
   {
      ShowWrongMessage();
   }
}
```

Listato 4.33: Funzione per la verifica delle selezioni

Le funzioni `ShowCorrectMessage()` e `ShowWrongMessage()` (Listato 4.34) sono utilizzate per mostrare i messaggi di successo o errore. Entrambe attivano il messaggio corrispondente e avviano una coroutine per nascondere il messaggio dopo un breve ritardo.

```
private void ShowCorrectMessage()
{
   if (correctMessage != null) correctMessage.SetActive(true);
   StartCoroutine(HideTextAfterDelay(correctMessage, true));
}

private void ShowWrongMessage()
{
   if (wrongMessage != null) wrongMessage.SetActive(true);
   StartCoroutine(HideTextAfterDelay(wrongMessage, false));
}
```

Listato 4.34: Funzioni per visualizzare i messaggi corretti o errati

La coroutine `HideTextAfterDelay()` (Listato 4.35) nasconde il messaggio di testo dopo un ritardo di `textDuration` secondi. Se la selezione è corretta, il passo successivo viene attivato, mentre il passo corrente viene disattivato.

```
private IEnumerator HideTextAfterDelay(GameObject text, bool isCorrect)
{
    yield return new WaitForSeconds(textDuration);

    if (text != null)
    {
        text.SetActive(false);
        if (isCorrect)
        {
            nextStep.SetActive(true);
            currentStep.SetActive(false);
        }
    }
}
```

Listato 4.35: Coroutine per nascondere il messaggio dopo un ritardo

Infine, la funzione `ActivateEnd()` (Listato 4.36) viene chiamata quando la selezione delle immagini "Micro" è corretta. Essa attiva il canvas finale delle lesioni associato al virus attivo.

```
private void ActivateEnd()
{
    foreach (GameObject endCanva in endCanvas)
    {
        if (activeVirus.CompareTag(endCanva.tag))
        {
            endCanva.SetActive(true);
            break;
        }
    }
}
```

Listato 4.36: Funzione per attivare il passo finale

4.3.12 InteractionTrasmissionZone

Lo script `InteractionTrasmissionZone` gestisce l'interazione tra il giocatore e le vie di trasmissione dei virus, attivando i messaggi "corretto o "errato" e modificando il materiale relativo alla pelle del bovino, rendendolo semitrasparente, se la zona è corretta. Inoltre, gestisce la visualizzazione di una descrizione del virus e la disattivazione di oggetti specifici dopo un certo periodo di tempo.

Nella funzione `Start()` (Listato 4.37) vengono disattivati i messaggi "corretto e "errato", così come le descrizioni del virus nella lista `virusMappings`, per garantire che non siano visibili quando la scena viene caricata.

```
private void Start()
{
    if (correctText != null) correctText.SetActive(false);
    if (incorrectText != null) incorrectText.SetActive(false);
    foreach (var mapping in virusMappings)
    {
        if (mapping.virusDescription != null)
        {
            mapping.virusDescription.SetActive(false);
        }
```

```
11      }
12  }
```

Listato 4.37: Funzione di inizializzazione

La funzione `OnTriggerEnter()` (Listato 4.38) viene chiamata quando un oggetto entra nella zona di trigger, ovvero in una delle vie di trasmissione. Essa verifica se l'oggetto che entra nella zona ha un tag che corrisponde a uno dei virus nella lista `virusMappings`. Se il tag del virus è presente, la funzione verifica se la zona è corretta. In caso affermativo, essa attiva il messaggio di successo, cambia il materiale del bovino e mostra la descrizione del virus. Se la zona è errata, viene visualizzato il messaggio di errore.

```
1   private void OnTriggerEnter(Collider other)
2   {
3       VirusZoneMapping mapping = virusMappings.Find(v => other.gameObject.
            CompareTag(v.virusTag));
4
5       if (mapping != null)
6       {
7           if (mapping.isCorrectZone)
8           {
9               if (correctText != null) correctText.SetActive(true);
10              ChangeMaterialObject();
11              if (mapping.virusDescription != null)
12              {
13                  mapping.virusDescription.SetActive(true);
14              }
15
16              StartCoroutine(HideTextAfterDelay(correctText, true));
17
18          }
19          else
20          {
21              if (incorrectText != null) incorrectText.SetActive(true);
22              StartCoroutine(HideTextAfterDelay(incorrectText, false));
23          }
24      }
25  }
```

Listato 4.38: Funzione di gestione dell'ingresso in zona di trigger

La coroutine `HideTextAfterDelay()` (Listato 4.39) è utilizzata per nascondere i messaggi di successo o errore dopo un certo periodo di tempo, definito dalla variabile `textDuration`. Se la via di trasmissione è corretta, tutte le vie di trasmissione (definita da `zoneToHide`) verranno disattivate.

```
1   private IEnumerator HideTextAfterDelay(GameObject text, bool isCorrect)
2   {
3       yield return new WaitForSeconds(textDuration);
4
5       if (text != null)
6       {
7           text.SetActive(false);
8           if (isCorrect) zoneToHide.SetActive(false);
9       }
10  }
```

Listato 4.39: Coroutine per nascondere il messaggio dopo un ritardo

La funzione `ChangeMaterialObject()` (Listato 4.40) cambia il materiale di un oggetto associato a `objectMaterialChanger`, in questo caso il bovino. Se l'oggetto ha un componente `MaterialChanger`, viene chiamato il metodo `ChangeMaterial` per cambiare il materiale dell'oggetto in quello definito in `newMaterial`.

```
void ChangeMaterialObject()
{
    if (objectMaterialChanger.TryGetComponent<MaterialChanger>(out var
        materialChanger))
    {
        materialChanger.ChangeMaterial(newMaterial);
    }
}
```

Listato 4.40: Funzione per cambiare il materiale di un oggetto

4.3.13 MaterialChanger

Lo script `MaterialChanger` è responsabile della modifica del materiale del bovino, passando da un materiale opaco ad uno semitrasparente.

La funzione `ChangeMaterial()` (Listato 4.41) riceve come parametro un `Material` e cambia il materiale dell'oggetto associato.

```
public void ChangeMaterial(Material newMaterial)
{
    if (TryGetComponent<Renderer>(out var objRenderer))
    {
        objRenderer.material = newMaterial;
    }
}
```

Listato 4.41: Funzione di cambio del materiale

4.3.14 PauseManager

Lo script `PauseManager` gestisce la pausa del gioco, permettendo di mettere in pausa e riprendere il gioco tramite input da parte del giocatore. Inoltre, si occupa della visualizzazione e della gestione dei vari elementi dell'interfaccia utente quando il gioco è in pausa.

La funzione `Start()` (Listato 4.42) inizializza il sistema di pausa, associando la camera principale e abilitando l'azione di pausa tramite `InputAction`.

```
void Start()
{
    mainCamera = Camera.main;

    ResetPause();

    var actionMap = inputActions.FindActionMap("XRI LeftHand Interaction
        ");
    pauseAction = actionMap.FindAction("Pause");
    pauseAction.Enable();
    pauseAction.performed += OnPausePerformed;
}
```

Listato 4.42: Inizializzazione dell'input per la pausa

La funzione `OnDestroy()` (Listato 4.43) rimuove il listener dell'azione di pausa quando lo script viene distrutto.

```
1  private void OnDestroy()
2  {
3      if (pauseAction != null)
4          pauseAction.performed -= OnPausePerformed;
5  }
```

Listato 4.43: Rimozione dell'ascoltatore quando lo script viene distrutto

La funzione `OnPausePerformed()` (Listato 4.44) gestisce la commutazione tra la pausa e la ripresa del gioco, basandosi sullo stato della variabile `isPaused`.

```
1  private void OnPausePerformed(InputAction.CallbackContext context)
2  {
3      if (isPaused)
4      {
5          ResumeGame();
6      }
7      else
8      {
9          PauseGame();
10     }
11 }
```

Listato 4.44: Commutazione tra pausa e ripresa

La funzione `ResetPause()` (Listato 4.45) resetta lo stato dell'interfaccia utente di pausa, in modo tale da non avere il menù di pausa visibile una volta che il gioco viene avviato.

```
1  public void ResetPause()
2  {
3      pauseCanvas.SetActive(false);
4      mainBlock.SetActive(true);
5      sectionBlock.SetActive(false);
6      quitBlock.SetActive(false);
7  }
```

Listato 4.45: Reset dello stato di pausa

La funzione `PauseGame()` (Listato 4.46) mette il gioco in pausa e imposta il `Time.timeScale` a 0, fermando il flusso del gioco.

```
1  public void PauseGame()
2  {
3      pauseCanvas.SetActive(true);
4      Time.timeScale = 0f;
5      isPaused = true;
6  }
```

Listato 4.46: Pausa del gioco

La funzione `ResumeGame()` (Listato 4.47) ripristina il gioco dallo stato di pausa, reimposta l'interfaccia del menù e riavvia il flusso del gioco con `Time.timeScale` a 1.

```
public void ResumeGame()
{
    ResetPause();
    Time.timeScale = 1f;
    isPaused = false;
}
```

Listato 4.47: Ripresa del gioco

4.3.15 QuitApplicationHandler

Lo script `QuitApplicationHandler` (Listato 4.48) gestisce l'uscita dall'applicazione.

```
public void OnPointerClick(PointerEventData eventData)
{
    if (eventData.button == PointerEventData.InputButton.Left)
    {
        Debug.Log("QUIT QUIT QUIT");
        Application.Quit();
    }
}
```

Listato 4.48: Gestione dell'uscita dall'applicazione

4.3.16 SceneInformationManager

Lo script `SceneInformationManager` gestisce il virus attivo nella scena, che viene determinato dalle impostazioni nel `GameManager`.

La funzione `Start()` (Listato 4.49) verifica quali virus sono attivi nel `GameManager` e attiva il virus corrispondente, impostando l' `activeVirus` al virus giusto. Inoltre, inizializza il riferimento al `gameManager`, cercandolo se non è stato assegnato.

```
void Start()
{
    if (GameManager.Instance.ActivateBhv1)
    {
        bhv1Virus.SetActive(true);
        activeVirus = bhv1Virus;
    }
    if (GameManager.Instance.ActivateBpiv3)
    {
        bpiv3Virus.SetActive(true);
        activeVirus = bpiv3Virus;
    }
    if (GameManager.Instance.ActivateBrsv)
    {
        brsvVirus.SetActive(true);
        activeVirus = brsvVirus;
    }

    gameManager = gameManager != null ? gameManager : FindObjectOfType<
        GameManager>();
```

```
20
21      if (activeVirus == null)
22      {
23          Debug.LogWarning("NON C' STA LU VIRUUUUUUS!");
24      }
25      if (gameManager == null)
26      {
27          Debug.LogWarning("NON C' STA LU GAMEMANAGEEEER!");
28      }
29  }
```

Listato 4.49: Gestione del virus attivo nella scena

4.3.17 SceneTransitionHandler

Lo script `SceneTransitionHandler` gestisce la logica per il caricamento di una nuova scena e la selezione di un virus da attivare prima di avviare il cambiamento.

La funzione `OnValidate()` (Listato 4.50) è utilizzata per controllare, nel pannello dell'Inspector di Unity, che solo una delle opzioni per la selezione del virus sia attiva. Se `virusChoise` è attivo, uno dei virus (Bhv1, Bpiv3, o Brsv) verrà selezionato, disattivando automaticamente gli altri.

```
1   public void OnValidate()
2   {
3       if (!virusChoise)
4       {
5           virusBhv1 = false;
6           virusBpiv3 = false;
7           virusBrsv = false;
8       }
9       else
10      {
11          if (virusBhv1)
12          {
13              virusBpiv3 = false;
14              virusBrsv = false;
15          }
16          else if (virusBpiv3)
17          {
18              virusBhv1 = false;
19              virusBrsv = false;
20          }
21          else if (virusBrsv)
22          {
23              virusBhv1 = false;
24              virusBpiv3 = false;
25          }
26      }
27  }
```

Listato 4.50: Gestione della mutua esclusione dei virus

La funzione `OnPointerClick()` (Listato 4.51) viene eseguita quando l'utente clicca sull'oggetto, gestendo la scelta del virus, il caricamento della scena e la gestione della transizione. Viene chiamato il metodo `Loading()` e successivamente viene chiamato il `SceneTransitionManager` per cambiare scena.

```
public void OnPointerClick(PointerEventData eventData)
{
   if (eventData.button == PointerEventData.InputButton.Left)
   {
      Loading();

      if (virusChoise && virusBhv1)
      {
         GameManager.Instance.ActivateBhv1 = true;
         GameManager.Instance.ActivateBpiv3 = false;
         GameManager.Instance.ActivateBrsv = false;
      }
      else if (virusChoise && virusBpiv3)
      {
         GameManager.Instance.ActivateBhv1 = false;
         GameManager.Instance.ActivateBpiv3 = true;
         GameManager.Instance.ActivateBrsv = false;
      }
      else if (virusChoise && virusBrsv)
      {
         GameManager.Instance.ActivateBhv1 = false;
         GameManager.Instance.ActivateBpiv3 = false;
         GameManager.Instance.ActivateBrsv = true;
      }

      GameObject sceneTransitionManagerObj = new("
          SceneTransitionManager");
      SceneTransitionManager sceneTransitionManager =
          sceneTransitionManagerObj.AddComponent<SceneTransitionManager
          >();
      Time.timeScale = 1f;
      sceneTransitionManager.ChangeScene(sceneToLoad);
   }
}
```

Listato 4.51: Scelta del virus e cambio di scena

La funzione `Loading()` (Listato 4.52) è utilizzata per mostrare il canvas di caricamento e nascondere gli altri oggetti nella scena, disabilitando i `Renderer` e i `Canvas` visibili tramite `DisableAllVisible()`.

```
public void Loading()
{
   DisableAllVisible();
   loadingCanvas.SetActive(true);
}

public void DisableAllVisible()
{
   Renderer[] renderers = FindObjectsOfType<Renderer>();
   foreach (Renderer rend in renderers)
   {
      rend.gameObject.SetActive(false);
   }

   Canvas[] canvases = FindObjectsOfType<Canvas>();
   foreach (Canvas canvas in canvases)
   {
      canvas.gameObject.SetActive(false);
   }
```

```
20  }
```

Listato 4.52: Caricamento della schermata di loading

4.3.18 SceneTransitionManager

La funzione `ChangeScene(string sceneName)` (Listato 4.53) carica una nuova scena specificata dal parametro `sceneName`, utilizzando il metodo `SceneManager.LoadScene`.

```
1  public void ChangeScene(string sceneName)
2  {
3      SceneManager.LoadScene(sceneName);
4  }
```

Listato 4.53: Gestione del cambio di scena e recupero del nome della scena attuale

4.3.19 StartAnimationPosition

Lo script `StartAnimationPosition` sposta il virus in modo tale da far partire l'animazione nello stesso punto della via di trasmissione

La funzione `Start()` (Listato 4.54) aggiunge un listener per ogni pulsante nella lista `buttons`, che invoca la funzione `OnButtonPress` quando il pulsante viene premuto.

```
1  private void Start()
2  {
3      foreach (Button button in buttons)
4      {
5          button.onClick.AddListener(() => OnButtonPress(button));
6      }
7  }
```

Listato 4.54: Aggiunta dei listener ai pulsanti per la pressione

La funzione `OnButtonPress(Button pressedButton)` (Listato 4.55) viene chiamata quando un pulsante viene premuto. Se il pulsante è attivo, verifica la presenza di oggetti nella lista `targetObjects`, ovvero i virus. Per ogni oggetto viene poi controllato se deve essere rimosso il `Collider` o il `Rigidbody`. Successivamente, l'oggetto viene spostato alla posizione e rotazione dell'oggetto desiderato.

```
1  private void OnButtonPress(Button pressedButton)
2  {
3      if (pressedButton.gameObject.activeSelf)
4      {
5          if (targetObjects != null && targetObjects.Count > 0)
6          {
7              foreach (GameObject targetObject in targetObjects)
8              {
9                  if (targetObject != null)
10                 {
11                     if (removeCollider && targetObject.TryGetComponent<
                           Collider>(out var targetCollider))
12                     {
13                         targetCollider.enabled = false;
14                     }
```

```
15                 if (removeRigidbody && targetObject.TryGetComponent<
                       Rigidbody>(out var targetRigidbody))
16                 {
17                    if (targetObject.TryGetComponent<XRGrabInteractable>(
                          out var targetScript))
18                    {
19                       Destroy(targetScript);
20                    }
21                    Destroy(targetRigidbody);
22                 }
23
24                 targetObject.transform.SetPositionAndRotation(transform.
                       position, transform.rotation);
25              }
26           }
27        }
28     }
29  }
```

Listato 4.55: Gestione della pressione del pulsante e modifica degli oggetti target

4.3.20 StartThirdIteration

La classe `StartThirdIteration` è utilizzata per gestire l'inizio dell'interazione da parte del giocatore con le cellule.

La funzione `Start()` (Listato 4.56) inizializza lo script cercando una referenza al `SceneInformationManager` se non già presente. Inoltre, aggiunge un listener ad ogni pulsante della lista `buttons`.

```
1  private void Start()
2  {
3     SceneInformationManagerScript = SceneInformationManagerScript !=
          null ? SceneInformationManagerScript : FindObjectOfType<
          SceneInformationManager>();
4
5     foreach (Button btn in buttons)
6     {
7        btn.onClick.AddListener(OnButtonPressed);
8     }
9  }
```

Listato 4.56: Inizializzazione dei listener e referenze iniziali

La funzione `OnButtonPressed()` (Listato 4.57) viene eseguita quando uno dei pulsanti nella lista viene premuto. Essa attiva l'oggetto `cells`, abilita gli animatori associati, disabilita l'oggetto `cow`, e gestisce l'attivazione/disattivazione di altri oggetti definiti nelle liste `objectsToAble` e `objectsToDisable`. Inoltre, disabilita il `MeshRenderer` del virus attivo.

```
1  private void OnButtonPressed()
2  {
3     cells.SetActive(true);
4     ActivateAnimators();
5     cow.SetActive(false);
6
7     DisableObject();
8     DisableActiveVirusMeshRenderer();
```

```
9       ActivateObject();
10  }
```

Listato 4.57: Gestione della pressione del bottone

La funzione `ActivateAnimators()` (Listato 4.58) abilita i componenti `Animator` del bovino e dell'insieme delle cellule.

```
1   private void ActivateAnimators()
2   {
3       if (cow != null)
4       {
5           if (cow.TryGetComponent<Animator>(out var cowAnimator))
6           {
7               cowAnimator.enabled = true;
8           }
9       }
10
11      if (cells != null)
12      {
13          if (cells.TryGetComponent<Animator>(out var cellsAnimator))
14          {
15              cellsAnimator.enabled = true;
16          }
17      }
18  }
```

Listato 4.58: Abilitazione degli animatori degli oggetti

La funzione `DisableObject()` (Listato 4.59) disabilita tutti gli oggetti contenuti nella lista `objectsToDisable`.

```
1   private void DisableObject()
2   {
3       if (objectsToDisable != null)
4       {
5           foreach (GameObject obj in objectsToDisable)
6           {
7               obj.SetActive(false);
8           }
9       }
10  }
```

Listato 4.59: Disabilitazione degli oggetti definiti

La funzione `DisableActiveVirusMeshRenderer()` (Listato 4.60) disabilita il `MeshRenderer` di tutte le istanze del virus attivo, identificato tramite il `SceneInformationManager`.

```
1   public void DisableActiveVirusMeshRenderer()
2   {
3       GameObject activeVirus = SceneInformationManagerScript.activeVirus;
4       if (activeVirus == null) return;
5
6       string virusTag = activeVirus.tag;
7
8       GameObject[] viruses = GameObject.FindGameObjectsWithTag(virusTag);
9       foreach (GameObject virus in viruses)
```

```
10      {
11          if (virus.TryGetComponent<MeshRenderer>(out MeshRenderer
                meshRenderer))
12          {
13              Debug.Log("name = " + virus.name);
14              meshRenderer.enabled = false;
15          }
16      }
17  }
```

Listato 4.60: Disabilitazione del MeshRenderer del virus attivo

La funzione `ActivateObject()` (Listato 4.61) abilita tutti gli oggetti presenti nella lista `objectsToAble`.

```
1   private void ActivateObject()
2   {
3       if (objectsToAble != null)
4       {
5           foreach (GameObject obj in objectsToAble)
6           {
7               obj.SetActive(true);
8           }
9       }
10  }
```

Listato 4.61: Abilitazione degli oggetti definiti

4.3.21 ThirdInteraction

La classe `ThirdInteraction` gestisce l'interazione tra le cellule e il virus attivo, consentendo al giocatore di selezionare cellule infette e monitorare il progresso rispetto agli obiettivi richiesti.

La funzione `Start()` (Listato 4.62) inizializza il riferimento al `SceneInformationManager`, determina il numero di selezioni corrette richieste per il virus attivo e collega l'evento di selezione a tutte le cellule configurate.

```
1   private void Start()
2   {
3       SceneInformationManagerScript = SceneInformationManagerScript !=
            null ? SceneInformationManagerScript : FindObjectOfType<
            SceneInformationManager>();
4       activeVirus = SceneInformationManagerScript.activeVirus;
5       correctSelectionsRequired = GetRequiredSelections(activeVirus.tag);
6
7       foreach (var cell in cells)
8       {
9           if (cell.TryGetComponent<XRSimpleInteractable>(out var
                interactable))
10          {
11              interactable.selectEntered.AddListener(OnCellSelected);
12          }
13      }
14  }
```

Listato 4.62: Inizializzazione dello script e configurazione degli eventi

La funzione `Update()` (Listato 4.63) aggiorna il testo dello `scoreText` per visualizzare il progresso del giocatore rispetto alle cellule infette richieste.

```
private void Update()
{
    if (scoreText != null)
    {
        scoreText.text = "Cells infected: " + correctSelectionCount + " /
            " + correctSelectionsRequired;
    }
}
```

Listato 4.63: Aggiornamento dello stato del punteggio

La funzione `OnDestroy()` (Listato 4.64) rimuove tutti i listener aggiunti durante l'inizializzazione, per evitare riferimenti persi.

```
private void OnDestroy()
{
    foreach (var cell in cells)
    {
        if (cell.TryGetComponent<XRSimpleInteractable>(out var
            interactable))
        {
            interactable.selectEntered.RemoveListener(OnCellSelected);
        }
    }
}
```

Listato 4.64: Pulizia degli eventi durante la distruzione

La funzione `GetRequiredSelections()` (Listato 4.65) restituisce il numero di selezioni corrette richieste per un determinato tag di virus.

```
private int GetRequiredSelections(string virusTag)
{
    return virusTag switch
    {
        "Bhv1" => 1,
        "Bpiv3" => 4,
        "Brsv" => 3,
        _ => 0,
    };
}
```

Listato 4.65: Determinazione delle selezioni richieste

La funzione `ResetSelections()` (Listato 4.66) reimposta lo stato dello script, azzerando il conteggio delle selezioni corrette e disattivando il messaggio di congratulazioni.

```
public void ResetSelections()
{
    correctSelectionCount = 0;
    selectedTags.Clear();
    congratText.SetActive(false);
}
```

Listato 4.66: Reset delle selezioni e dello stato iniziale

La funzione `OnCellSelected()` (Listato 4.67) gestisce la logica di selezione di una cellula, controllando se è associata al virus attivo e aggiornando il conteggio delle selezioni corrette. Inoltre, impedisce la selezione ripetuta di cellule già analizzate.

```
private void OnCellSelected(SelectEnterEventArgs args)
{
    GameObject selectedCell = args.interactableObject.transform.
        gameObject;
    string cellTag = selectedCell.tag;

    if (selectedTags.Contains(cellTag))
    {
        return;
    }

    selectedTags.Add(cellTag);

    CellVirusData cellVirusData = selectedCell.GetComponent<
        CellVirusData>();

    if (cellVirusData != null && !cellVirusData.isSelected)
    {
        if (cellVirusData.IsAssociatedWithVirus(activeVirus.tag))
        {
            IlluminateCellsByTag(cellTag, correctSelectionColor);
            correctSelectionCount++;
            cellVirusData.isSelected = true;
        }
        else
        {
            IlluminateCellsByTag(cellTag, wrongSelectionColor);
            cellVirusData.isSelected = true;
        }

        if (correctSelectionCount >= correctSelectionsRequired)
        {
            congratText.SetActive(true);
        }
    }
}
```

Listato 4.67: Gestione della selezione delle cellule

La funzione `IlluminateCellsByTag()` (Listato 4.68) applica un materiale a tutte le cellule con uno specifico tag per indicare la loro selezione come corretta o errata.

```
private void IlluminateCellsByTag(string cellTag, Material color)
{
    foreach (GameObject cell in cells)
    {
        if (cell.CompareTag(cellTag))
        {
            cell.GetComponent<Renderer>().material = color;
        }
    }
```

```
10  }
```

Listato 4.68: Illuminazione delle cellule selezionate

4.4 Implementazione realtà virtuale

L'implementazione su visore VR è stata possibile grazie all'utilizzo del pacchetto *XR Interaction Toolkit* e alla guida XR Interaction Toolkit Examples presente su GitHub (`https://github.com/Unity-Technologies/XR-Interaction-Toolkit-Examples`).

Tale pacchetto è un sistema di interazione di alto livello, *component-based*, per la creazione di esperienze in realtà virtuale e aumentata. Esso fornisce un framework che rende disponibili interazioni 3D e UI da eventi di input Unity. Il nucleo di questo sistema è un set di componenti base, *Interactor* e *Interactable* e un *Interaction Manager* che collega insieme questi due tipi di componenti.

Ogni GameObject, con il quale l'utente deve interagire tramite visore e controller, è stato dotato della classe `TrackedDeviceGraphicRaycaster`, per permettere l'utilizzo del controller.

Ai virus è stato aggiunto la classe `XR Grab Interactable`, per renderli interattivi tramite "presa" da parte del giocatore con i controller, eliminando la classe `VirusDrag`, ormai obsoleta. Le cellule, invece, sono state dotate della classe `XR Simple Interactable`, per renderle interattive e permettere al giocatore di selezionarle.

In più, è stato utilizzato l'oggetto *Simple XR Player with Ray Interactor*, preesistente nell'asset store di Unity, per implementare i comandi base del visore e dei controller.

CAPITOLO 5

Manuale utente

In questo capitolo verrà introdotto il manuale utente del serious game sviluppato come parte integrante della presente tesi. All'interno di queste pagine, i giocatori avranno accesso a un'esposizione esauriente, la quale comprende non soltanto istruzioni dettagliate e l'elenco esaustivo dei comandi di gioco, ma anche immagini delle schermate con le quali l'utente deve interagire. In aggiunta, verranno fornite indicazioni passo passo per l'installazione del videogioco.

5.1 Avvertenze per la salute

Giocare sempre in un ambiente ben illuminato. Fare pause regolari di circa 15 minuti per ogni ora di gioco. Smettere immediatamente di giocare se si viene colti da vertigini, nausea, affaticamento o cefalea. I soggetti sensibili a luci intermittenti, o a particolari forme o configurazioni geometriche, potrebbero soffrire di una forma di epilessia non diagnosticata ed essere soggetti a crisi epilettiche guardando la TV o giocando con i videogiochi. Se si è soggetti ad attacchi epilettici, consultare il medico prima di giocare con i videogiochi e contattarlo immediatamente qualora si dovessero riscontrare uno o più dei seguenti sintomi durante il gioco: alterazioni della vista, contrazioni muscolari, altri movimenti involontari, perdita di coscienza, confusione mentale e/o convulsioni.

5.2 Installazione e avvio

Per installare correttamente il gioco, sarà sufficiente seguire attentamente i passaggi di seguito esposti:

1. *Download dei File di Installazione*: scaricare i file di installazione del serious game dal link: `https://github.com/Jimboide69/VetGo_Cow_Edition`;

2. *Estrazione*: una volta scaricati tutti i file, che saranno in formato archivio (Vet-GO Cow 1.2.6.zip e Vet-GO Cow 1.2.6.z0*), posizionarli nella stessa cartella e procedere con l'estrazione tramite tool specifici, come, ad esempio, *WinRAR* o *WinZIP*;

3. *Abilitare la Modalità Sviluppatore*: il primo passo per poter installare applicazioni non presenti sullo store ufficiale di Meta, è abilitare la Modalità Sviluppatore sul vostro account Meta seguendo le istruzioni presenti al seguente link: `https://developers.meta.com/horizon/documentation/native/android/mobile-device-setup/`;

4. *Installazione SideQuest*: SideQuest è un software che permette di installare APK direttamente dal PC. Scaricare, quindi, l'installer del software direttamente sul vostro PC al seguente link: `https://sidequestvr.com/download`;

5. *Avvio dell'installazione di SideQuest*: una volta completato il download, fare doppio click sul file di installazione. Avviato l'installer, avremo la possibilità di installare il software su un singolo utente o su tutti gli utenti presenti sul PC (scelta consigliata). L'installer chiederà di selezionare la cartella in cui si desidera installare il videogioco. È possibile accettare la cartella predefinita o scegliere una diversa destinazione sul sistema. Fatto ciò, avviare l'installazione.

6. *Abilitare il visore*: una volta installato SideQuest, mentre il software è aperto, collegare il visore al PC. In alto a sinistra della schermata del software ci sarà un pallino verde, ad indicare che la connessione è andata a buon fine.

7. *Installazione del serious game*: dopo tutte le precedenti configurazioni, è possibile installare il serious game, direttamente dal PC sul visore, tramite l'icona  in alto a destra nella schermata del software;

8. *Avvio del gioco*: infine è possibile avviare il gioco. Ciò è possibile mediante l'utilizzo del menù presente nell'interfaccia del visore.

5.3 Elenco completo dei comandi e interfaccia utente

In questa sezione verranno elencati i comandi con i quali l'utente può interagire con l'applicazione e verrà illustrata l'interfaccia grafica mediante la quale tale interazione è possibile.

Nella Figura 5.1 sono mostrati i Touch Controller del Meta Quest 3, con i rispettivi nomi dei pulsanti presenti, per facilitare la comprensione dei comandi.

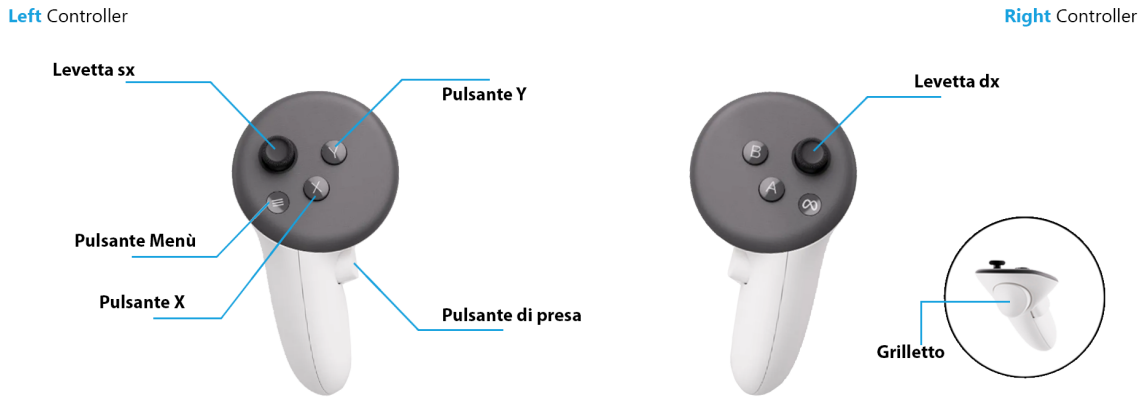


Figura 5.1: Touch Controller del Meta Quest 3

5.3.1 Schermata iniziale

Nella Tabella 5.1 sono mostrati i comandi mediante i quali il giocatore può interagire con l'interfaccia di gioco nella schermata iniziale. Nella Figura 5.2 viene mostrata la schermata iniziale del gioco. Nella Figura 5.3 viene mostrata l'interfaccia grafica relativa alla selezione del virus. Nella Figura 5.4 viene mostrata l'interfaccia grafica relativa all'uscita dal gioco.

Azione	Comando
Iniziare a giocare	Click, con il *Grilletto*, sul pulsante (*PLAY*)
Chiudere il gioco	Click, con il *Grilletto*, sul pulsante (*QUIT*)
Scorrere tra i virus	Click, con il *Grilletto*, sui pulsanti (<), indietro, e (>), avanti
Scegliere il virus	Click, con il *Grilletto*, sul pulsante (*START*)
Confermare la chiusura del gioco	Click, con il *Grilletto*, sul pulsante (*YES*)
Non confermare la chiusura	Click, con il *Grilletto*, sul pulsante (*NO*)

Tabella 5.1: Elenco dei comandi per la schermata iniziale

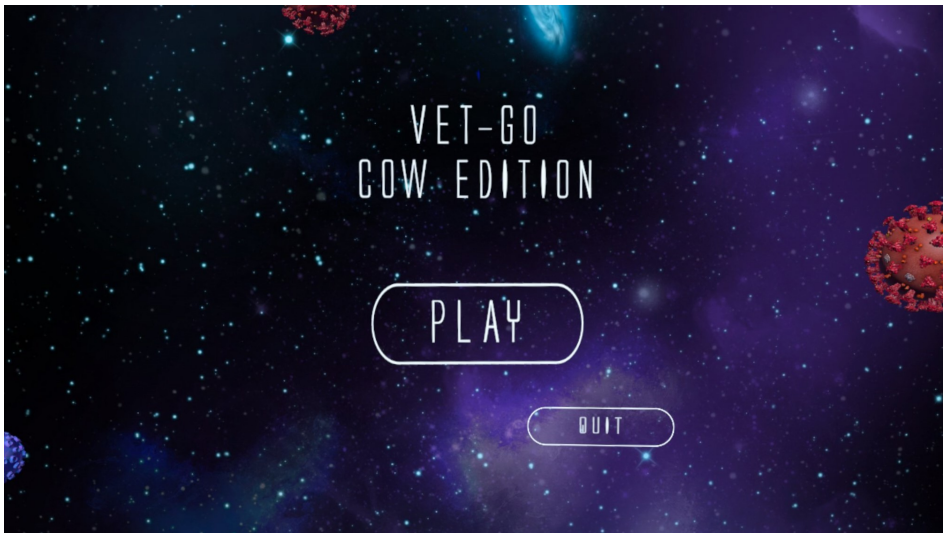


Figura 5.2: Schermata iniziale

Figura 5.3: Schermata di selezione del virus

5.3.2 Ambiente stalla

Nella Tabella 5.2 sono mostrati i comandi mediante i quali il giocatore può spostare il virus e muoversi nell'ambiente circostante. Nella Figura 5.5 viene mostrato l'ambiente di gioco dove il giocatore può muoversi e spostare il virus. Vengono, inoltre, mostrate le diverse vie di

Figura 5.4: Schermata di uscita

trasmissione disponibili (l'ingresso della via di trasmissione è definito da una sfera di colore blu). Nella Figura 5.6 viene mostrata l'interfaccia grafica relativa all'inizio dell'animazione circa la trasmissione del virus.

Azione	Comando
Prendere il virus	Tenere premuto, con il *Pulsante di presa*, sul modello 3D del virus
Spostare il virus	Una volta preso il virus, questo viene spostato nella stessa direzione indicata dal controller con cui esso è stato preso
Proseguire dopo aver letto le informazioni preliminari	Click, con il *Grilletto*, sul pulsante (*NEXT*)
Gestire la visione dell'animazione	Tenere premuto, con il *Grilletto*, sullo *Slider* e spostare il cursore
Proseguire dopo aver visto l'animazione	Click, con il *Grilletto*, sul pulsante (*NEXT*)

Tabella 5.2: Elenco dei comandi per le interazioni nella stalla

5.3.3 Selezione delle cellule

Nella Tabella 5.3 sono mostrati i comandi mediante i quali il giocatore può selezionare le cellule relative al virus scelto. Nella Figura 5.7 viene mostrata la schermata di selezione delle cellule. Nella Figura 5.8 vengono mostrate le informazioni relative alla patogenesi. Nella Figura 5.9 vengono mostrate le informazioni relative ai segni clinici.

5.3.4 Selezione delle lesioni (gross)

Nella Tabella 5.4 sono mostrati i comandi mediante i quali il giocatore può selezionare le immagini delle lesioni di gross relative al virus scelto. Nella Figura 5.10 viene mostrata la schermata di selezione delle immagini di gross.

Figura 5.5: Schermata iniziale nella stalla

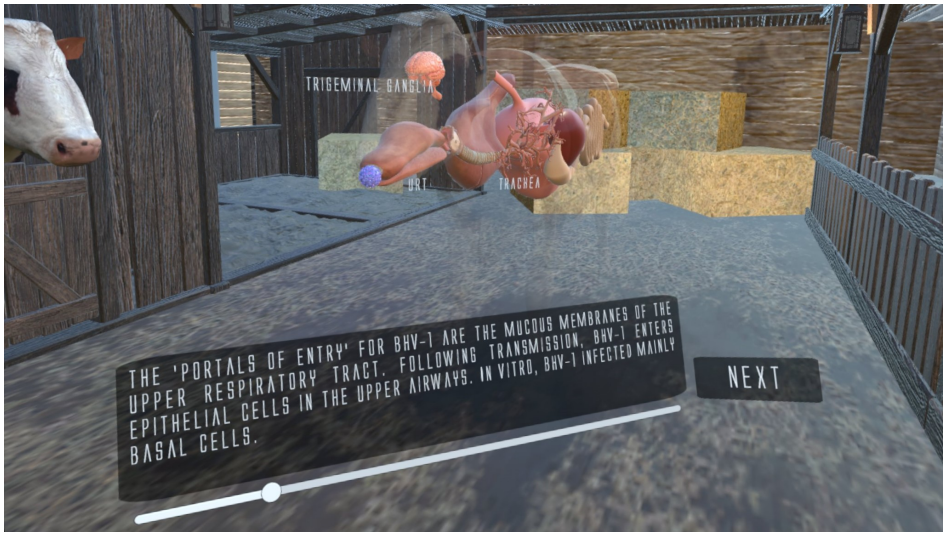


Figura 5.6: Schermata relativa all'animazione del virus

Azione	Comando
Selezionare cellula	Click, con il *Pulsante di presa*, sul modello 3D della cellula
Proseguire dopo aver selezionato le cellule	Click, con il *Grilletto*, sul pulsante (*NEXT*)
Proseguire dopo aver letto le informazioni sulla patogenesi	Click, con il *Grilletto*, sul pulsante (*NEXT*)
Proseguire dopo aver letto le informazioni sui segni clinici	Click, con il *Grilletto*, sul pulsante (*NEXT*)

Tabella 5.3: Elenco dei comandi per la selezione delle cellule

5.3.5 Selezione delle lesioni (histo) e fine del gioco

Nella Tabella 5.5 sono mostrati i comandi mediante i quali il giocatore può selezionare le immagini delle lesioni histo relative al virus scelto. Nella Figura 5.11 viene mostrata la

Figura 5.7: Schermata relativa alla selezione delle cellule

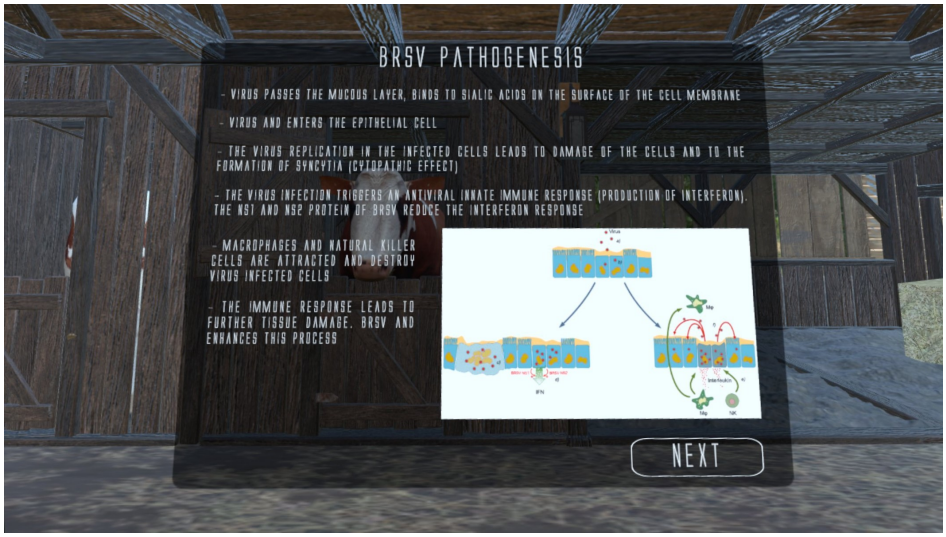


Figura 5.8: Schermata relativa alla patogenesi

Azione	Comando
Scorrere tra le immagini di gross	Click, con il *Grilletto*, sui pulsanti (<), indietro, e (>), avanti
Selezionare/deselezionare le singole immagine	Click, con il *Grilletto*, sulla (*checkbox*) ■
Confermare la selezione	Click, con il *Grilletto*, sul pulsante (*NEXT*)

Tabella 5.4: Elenco dei comandi per la selezione delle immagini di gross

schermata di selezione delle immagini histo. Nella Figura 5.12 viene mostrata la schermata finale del gioco, con le descrizioni delle lesioni.

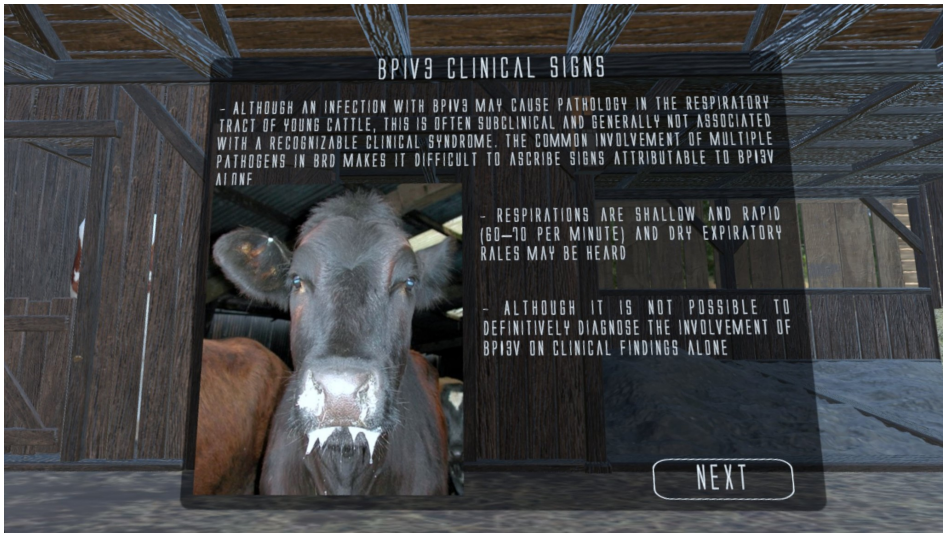


Figura 5.9: Schermata relativa ai segni clinici

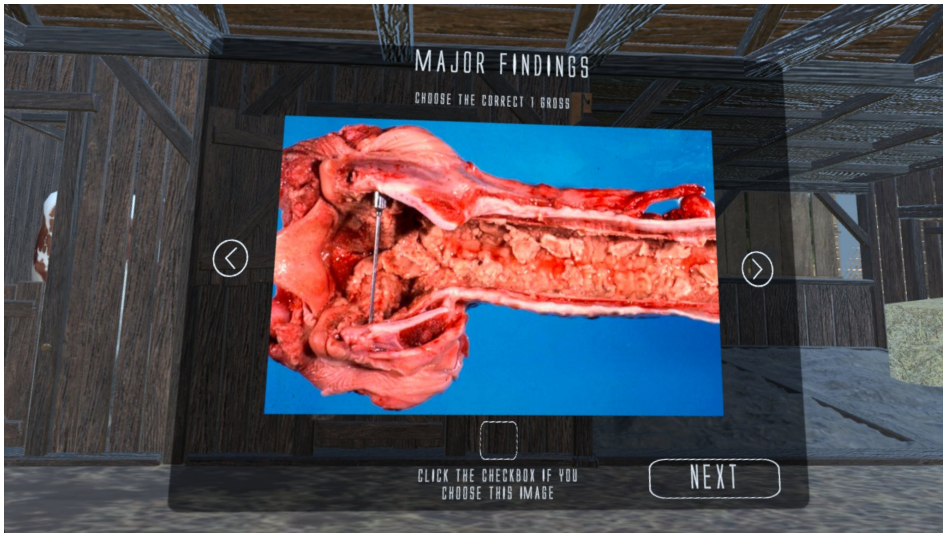


Figura 5.10: Schermata relativa alla selezione delle immagini di gross

Azione	Comando
Scorrere tra le immagini di gross	Click, con il *Grilletto*, sui pulsanti (<), indietro, e (>), avanti
Selezionare/deselezionare le singole immagine	Click, con il *Grilletto*, sulla (*checkbox*)
Confermare la selezione	Click, con il *Grilletto*, sul pulsante (*NEXT*)
Ricominciare il gioco	Click, con il *Grilletto*, sul pulsante (*END GAME*)

Tabella 5.5: Elenco dei comandi per la selezione delle immagini histo

5.3.6 Menù di pausa

Nella Tabella 5.6 sono mostrati i comandi mediante i quali il giocatore può interagire con il menù di pausa. Nella Figura 5.13 viene mostrata la schermata di pausa. Nella Figura 5.14 viene mostrata la schermata per la selezione della sezione di gioco in cui andare.

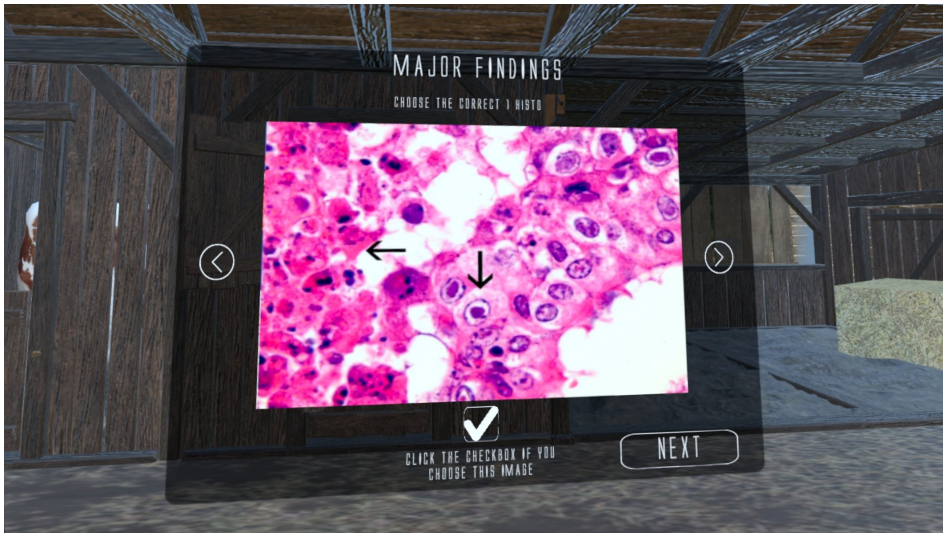


Figura 5.11: Schermata relativa alla selezione delle immagini histo

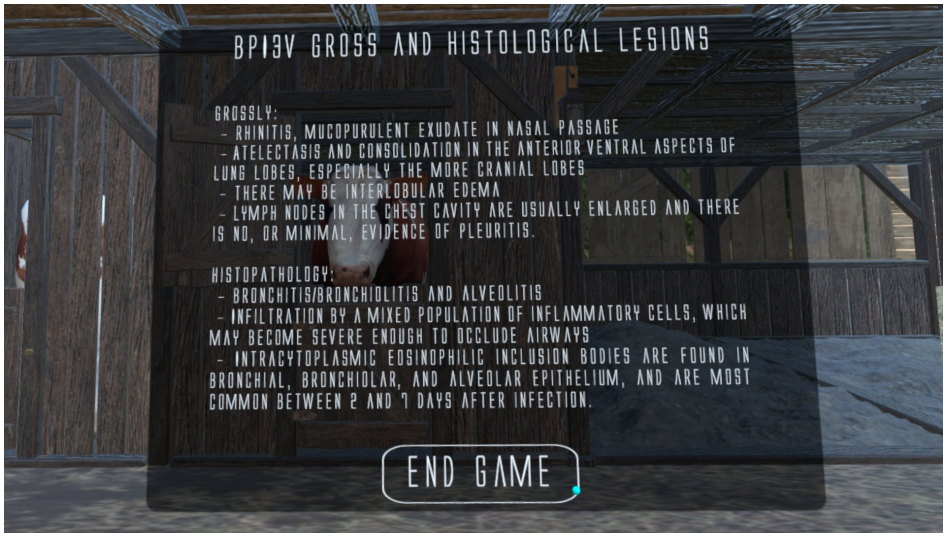


Figura 5.12: Schermata relativa alle informazioni sulle lesioni

Azione	Comando
Riprendere il gioco	Click, con il *Grilletto*, sul pulsante (*RESUME*)
Cambiare sezione	Click, con il *Grilletto*, sul pulsante (*CHANGE SECTION*)
Uscire dal gioco	Click, con il *Grilletto*, sul pulsante (*QUIT*)
Selezionare sezione	Click, con il *Grilletto*, sul pulsante relativo alla sezione in cui si vuole andare

Tabella 5.6: Elenco dei comandi per il menù di pausa

5.4 Come si gioca

Il gioco inizia in un ambiente tridimensionalmente infinito, senza possibilità di movimento da parte dell'utente. In questa scena il giocatore deve selezionare il virus con il quale iniziare a giocare.

In seguito alla selezione, il giocatore si troverà nella seconda scena del gioco, la stalla. In

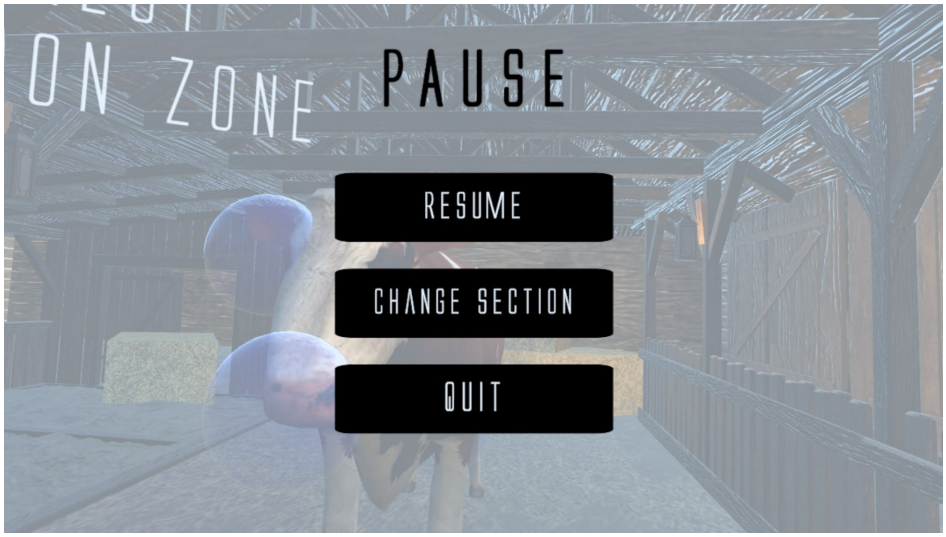


Figura 5.13: Schermata relativa al menù di pausa

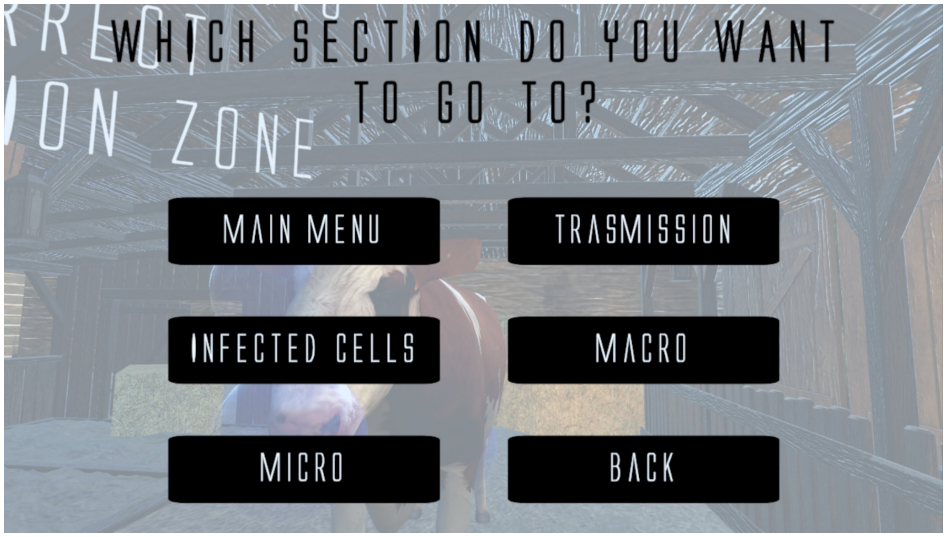


Figura 5.14: Schermata relativa al cambio sezione

questo ambiente il giocatore è libero di muoversi all'interno della stuttura e, utilizzando i comandi esposti nella sezione precedente, dovrà spostare il virus fino a raggiungere la via di trasmissione corretta relativa al virus scelto.

Una volta raggiunta la via di trasmissione corretta, comparirà una breve descrizione su come il virus si trasmette. Dopo di ciò, partirà una breve animazione nella quale viene mostrato come il virus agisce all'interno dell'organismo del bovino. Terminata l'animazione, il giocatore, dopo aver premuto l'apposito pulsante per andare avanti, dovrà selezionare le cellule relative al virus, ovvero le cellule che vengono "colpite" dal virus con cui si sta giocando. Quando il giocatore avrà selezionato tutte le cellule corrette, comparirà un messaggio di congratulazioni permettendo il proseguimento del gioco.

Completata la fase cellulare, il giocatore visionerà le informazioni sulla patogenesi e sui segni clinici del virus in questione. Il giocatore arriva, quindi, a dover scegliere le immagini delle lesioni di gross. Essendo diverso il numero di immagini relative ad ogni virus, il giocatore dovrà selezionare, tramite checkbox, quelle che reputa corrette in base al numero che comparirà sulla schermata, e confermare la selezione tramite l'apposito pulsante.

Se il giocatore seleziona le immagini corrette, comparirà la schermata per la selezione

delle immagini histo, uguale a quella precedente delle immagini di gross. Il giocatore, infine, dopo aver selezionato tutte le immagini corrette, visualizzerà le informazioni relative alle lesioni, con la possibilità di tornare alla schermata iniziale tramite apposito pulsante.

Dopo aver scelto il virus nella schermata iniziale, quindi una volta che è entrato nella stalla, il giocatore può mettere il gioco in pausa attivando il menù tramite apposito pulsante sul controller. Il menù gli permette di riprendere il gioco da dove è stato interrotto, di uscire dal gioco e di cambiare sezione, selezionandone una tra quelle disponibili, ossia ritornare al menù iniziale, arrivare nella scelta della via di trasmissione, nella scelta delle cellule, nella scelta delle lesioni di gross o delle lesioni histo.

CAPITOLO 6

Discussione in merito al lavoro svolto

L'obiettivo di questo capitolo è quello di analizzare in maniera critica il serious game sviluppato, identificandone i punti di forza e di debolezza, nonché le opportunità e le minacce, attraverso un'analisi SWOT (Strengths, Weaknesses, Opportunities, Threats).

6.1 Analisi SWOT

L'analisi SWOT rappresenta uno strumento strategico di fondamentale importanza per valutare in modo approfondito sia gli aspetti interni di un progetto sia le influenze esterne che ne possono condizionare l'esito. Questa metodologia consente di identificare e analizzare i punti di forza (Strengths) e le debolezze (Weaknesses) intrinseci del progetto, oltre a evidenziare le opportunità (Opportunities) offerte dall'ambiente circostante e le potenziali minacce (Threats) che potrebbero ostacolare il raggiungimento degli obiettivi prefissati.

In questa sezione verranno esaminati nel dettaglio ognuna delle quattro categorie dell'analisi SWOT, fornendo una visione strutturata e critica di ciò che rende il progetto competitivo, delle aree che necessitano miglioramenti, delle opportunità di sviluppo e delle sfide da affrontare per garantire un impatto sostenibile e positivo.

Nella Figura 6.1 è riportata una panoramica di quanto segue.

6.1.1 Strengths

Nell'analisi SWOT, i *punti di forza* (strengths) rappresentano fattori interni che contribuiscono positivamente al progetto, differenziandolo dalle soluzioni concorrenti. Dall'analisi condotta sul serious game sviluppato, sono emersi i seguenti punti di forza:

- *Gratuità*: il serious game oggetto di questa tesi si distingue per essere offerto gratuitamente agli utenti come strumento educativo, differenziandosi dalla maggior parte dei software e applicazioni simili presenti sul mercato. Questa scelta elimina le barriere economiche che potrebbero ostacolare l'adozione del gioco, garantendo un accesso semplice e inclusivo a un ampio pubblico. In un contesto in cui l'acquisizione di risorse educative di qualità è spesso associata a costi elevati, la gratuità del nostro serious game rappresenta un punto di forza rilevante, capace di favorire una più ampia diffusione e di incrementarne l'impatto formativo.

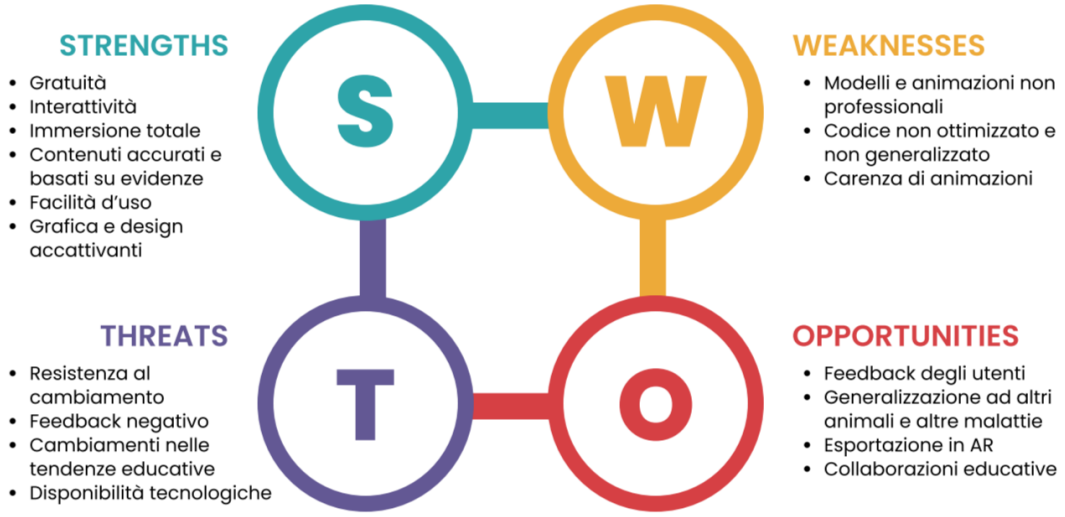


Figura 6.1: Analisi SWOT del serious game

- *Interattività*: il nostro serious game si distingue dagli altri giochi dedicati alla trasmissione dei virus per il livello avanzato di interattività offerto. Nelle varie fasi del gioco, il giocatore interagisce direttamente con l'ambiente circostante tramite le sue scelte. Questo approccio consente all'utente di partecipare in modo diretto e coinvolgente all'esperienza formativa, prendendo decisioni e sperimentando in prima persona i concetti legati ai virus vaccini. L'adozione di una metodologia didattica attiva non solo favorisce una maggiore comprensione dei contenuti, ma stimola anche l'interesse degli utenti, rendendo il serious game più efficace come strumento educativo e contribuendo al suo successo complessivo.

- *Immersione totale*: il nostro serious game è stato sviluppato per sfruttare le potenzialità offerte dalla realtà virtuale, garantendo un livello di immersione totale che amplifica significativamente l'esperienza dell'utente. Grazie all'uso di visori e controller, i giocatori possono interagire con gli elementi del gioco in un ambiente tridimensionale, vivendo in prima persona le dinamiche legate alla trasmissione e agli effetti dei virus. Questa immersione non solo rende il gioco più coinvolgente, ma favorisce anche un apprendimento più efficace, poiché l'utente è pienamente integrato nell'ambiente simulato e può esplorare attivamente i contenuti educativi. La realtà virtuale, combinata con un design interattivo, trasforma il serious game in uno strumento altamente innovativo per l'insegnamento e la formazione.

- *Contenuti accurati e basati su evidenze*: il nostro serious game si basa su informazioni e dettagli accurati riguardanti i virus vaccini, forniti direttamente da un referente del Dipartimento di Veterinaria dell'Università di Padova. Questo approccio assicura che i contenuti presentati agli utenti siano scientificamente validi e affidabili. L'accuratezza dei dati non solo rafforza la credibilità del serious game, ma costituisce anche un valore aggiunto per coloro che cercano informazioni precise. In un contesto in cui la disinformazione è a portata di click, l'attenzione posta sull'affidabilità e sulla validità scientifica dei contenuti rende il nostro serious game una risorsa educativa di grande valore e un punto di riferimento per l'apprendimento.

- *Facilità d'uso*: l'interfaccia utente del nostro serious game è stata progettata con un

design semplice e intuitivo, per garantire una navigazione agevole all'interno del gioco. Questa attenzione alla facilità d'uso è cruciale per raggiungere un pubblico ampio e diversificato, includendo anche coloro che possiedono una limitata esperienza con giochi o applicazioni digitali. Un'interfaccia accessibile favorisce l'efficacia del serious game come strumento educativo, consentendo agli utenti di focalizzarsi sull'apprendimento dei contenuti piuttosto che sull'affrontare difficoltà tecniche o barriere tecnologiche.

- *Grafica e design accattivanti*: l'aspetto visivo del serious game è stato progettato con particolare attenzione per garantire un'esperienza estetica coinvolgente e gradevole per gli utenti. I colori, le illustrazioni e i modelli sono stati accuratamente selezionati per rendere l'ambiente di gioco visivamente accattivante e stimolante. Questa cura per il design grafico contribuisce a catturare l'interesse degli utenti fin dalle prime fasi, favorendo il loro coinvolgimento e mantenendo alta l'attenzione durante l'intero percorso di apprendimento.

6.1.2 Weaknesses

Nell'analisi SWOT, i *punti di debolezza* (weaknesses) sono fattori interni che contribuiscono negativamente al progetto, differenziandolo dalle soluzioni concorrenti, in maniera negativa. Dall'analisi condotta sul serious game sviluppato, sono emersi i seguenti punti di debolezza:

- *Carenza di animazioni*: le animazioni presenti nel nostro serious game sono poche, e limitate al trasferimento del virus all'interno del bovino. La mancanza di competenze specifiche nella creazione di animazioni complesse potrebbe influire negativamente sull'aspetto videoludico del prodotto.

- *Modelli e animazioni non professionali*: alcuni dei modelli, insieme alle poche animazioni attualmente presenti nel gioco, sono stati realizzati da uno studente di Ingegneria con minime conoscenze di base nella modellazione grafica. Sebbene queste competenze abbiano consentito di creare contenuti funzionali al serious game, alcune limitazioni tecniche potrebbero aver influito sulla qualità visiva e sul livello di dettaglio estetico. La mancanza di una formazione avanzata in design grafico potrebbe aver ridotto l'impatto complessivo del gioco, limitandone in parte la capacità di offrire un'esperienza visivamente accattivante e coinvolgente per gli utenti.

- *Codice non ottimizzato e non generalizzato*: sebbene il codice sorgente del nostro serious game fornisca un'esperienza di gioco soddisfacente e funzioni come previsto, è evidente la necessità di ottimizzarlo e generalizzarlo per migliorare la manutenibilità, la scalabilità e la flessibilità del progetto.

6.1.3 Opportunities

Nell'analisi SWOT, le *opportunità* (opportunities) rappresentano fattori esterni che, se ben strutturati, permettono di differenziare il prodotto da quello dei concorrenti. Le opportunità rappresentano, quindi, le condizioni esterne utili a migliorare il prodotto, raggiungendo l'obiettivo prefissato. Dall'analisi condotta sul serious game sviluppato, sono emerse le seguenti opportunità:

- *Feedback degli utenti*: la prima opportunità riguarda la raccolta di feedback da parte degli utenti. Mediante tali feedback possono essere strutturati altri punti di forza e/o punti di debolezza sui quali lavorare per migliorare tutti gli aspetti del nostro serious game.

- *Generalizzazione ad altri animali e altre malattie*: un'altra importante opportunità riguarda la sua generalizzazione, per poter coprire altre specie animali e altre malattie, rendendo il nostro serious game uno strumento educativo scalabile, versatile e flessibile.
- *Collaborazioni educative*: la possibilità di stabilire collaborazioni educative con scuole, università o qualsiasi altra tipologia di istituzione accademica ed educativa, oltre quella già presente con l'Università di Padova, rappresenta un'opportunità di crescita non indifferente. Queste collaborazioni potrebbero portare ad un ampliamento considerevole dell'utenza, andando ad integrare il prodotto in corsi di studio specifici; inoltre, gli esperti accademici del settore potrebbero contribuire in maniera significativa all'aumento delle informazioni scientifiche, già presenti nel nostro serious game.
- *Esportazione in Realtà Aumentata (AR)*: un'ultima opportunità riguarda l'utilizzo della realtà aumentata come nuova tecnologia di sviluppo. Conciliare la realtà virtuale alla realtà aumentata, farebbe sì che il nostro serious game si adatti a qualsiasi tipologia di utenza.

6.1.4 Threats

Nell'analisi SWOT, le *minacce* (threats) sono fattori esterni che possono rappresentare potenziali problemi, ostacoli o sfide. Le minacce rappresentano condizioni sfavorevoli o fattori negativi che possono influire negativamente sull'obiettivo da raggiungere. Dall'analisi condotta sul serious game sviluppato, sono emerse le seguenti minacce:

- *Resistenza al cambiamento*: la tendenza delle persone a resistere ai cambiamenti è una delle minacce più consistenti per il nostro serious game. L'utilizzo della realtà virtuale, per molti sconosciuta e poco utilizzata, potrebbe essere vista come una perdita di tempo nell'educazione, rimarcando metodi ed approcci consolidati da tempo.
- *Feedback negativo*: il feedback negativo da parte degli utenti potrebbe portare ad una sfiducia sul nostro serious game e sulla sua efficacia educativa.
- *Cambiamenti nelle tendenze educative*: un'altra minaccia è rappresentata dai cambiamenti nelle tendenze educative, in quanto le priorità e le modalità educative possono evolversi nel tempo, modificando così l'efficacia del nostro serious game.
- *Disponibilità tecnologiche*: l'ultima minaccia è la disponibilità della tecnologia per garantire il funzionamento del nostro serious game. Non tutte le istituzioni accademiche e/o educative possiedono un visore per la realtà virtuale, o, perlomeno, il numero di dispositivi disponibili potrebbe essere molto limitato. Ciò porterebbe l'utilizzo del nostro serious game ad una nicchia di utenti, limitando il numero di giocatori.

Conclusioni e sviluppi futuri

Nei precedenti capitoli abbiamo illustrato in dettaglio le fasi del processo di progettazione e di sviluppo di un serious game, come strumento didattico per supportare l'apprendimento delle patologie causate dai virus nei bovini, rivolto agli studenti dei corsi di laurea in Veterinaria.

Il progetto è stato avviato grazie a una stretta collaborazione con il Dipartimento di Medicina Veterinaria dell'Università di Padova, che ha fornito i dati scientifici necessari per l'identificazione dei requisiti funzionali e non funzionali.

Una volta definiti i requisiti, abbiamo elaborato la struttura del sistema individuando le classi e i casi d'uso principali, che hanno guidato la successiva implementazione.

L'intero processo di sviluppo ha seguito un approccio iterativo e incrementale, ispirato alla metodologia Agile, con revisioni periodiche condotte insieme ai referenti accademici.

Ciò ha permesso di recepire tempestivamente eventuali modifiche ai requisiti e di garantire che il prodotto finale fosse in linea con le aspettative.

Infine, una volta terminata la fase di implementazione, si è proceduto a testare il nostro serious game e a generare il file utile per l'installazione.

Con questo lavoro, abbiamo fornito un'applicazione educativa, che combina contenuti scientifici accurati con un'esperienza di apprendimento interattiva e immersiva. In prospettiva, ci sono molteplici possibilità per evolvere ulteriormente il progetto. In particolare, si può pensare di procedere lungo le seguenti direzioni:

- il *miglioramento dei modelli* e l'*aggiunta di animazioni*, che renderebbero il gioco più piacevole e coinvolgente per gli utenti;
- l'*integrazione di strumenti di analisi dei dati*, per monitorare il progresso dei giocatori e il loro comportamento, in modo da evidenziare le aree in cui potrebbe essere necessario fornire maggiore supporto;
- il *refactoring del codice*, per renderlo maggiormente leggibile, scalabile e versatile, senza modificare le funzionalità;
- l'*utilizzo della realtà mista (XR)*, in modo da permettere all'utente di scegliere se avere un'immersione totale o parziale all'interno del gioco;
- la *modalità multigiocatore*, per permettere a più utenti di lavorare in gruppo e, quindi, di collaborare per risolvere le sfide del gioco;
- l'*aggiunta di altre malattie e di altri animali*, per rendere il gioco ancora più versatile ed efficiente come supporto allo studio.

Bibliografia

DJAOUTI, D., ALVAREZ, J. e JESSEL, J.-P. (2011), «Classifying Serious Games: The G/P/S Model», in «Serious Games and Edutainment Applications», Springer.

FREINA, L. e OTT, M. (2015), «A Literature Review on Immersive Virtual Reality in Education: State of the Art and Perspectives», in «Proceedings of the 11th International Scientific Conference eLearning and Software for Education (eLSE)», .

HEIM, M. (1993), *The Metaphysics of Virtual Reality*, Oxford University Press.

KAVANAGH, S., LUXTON-REILLY, A., WUENSCHE, B. e PLIMMER, B. (2017), «A systematic review of Virtual Reality in education», *Themes in Science and Technology Education*.

LAMPROPOULOS, G., KERAMOPOULOS, E., KONSTANTINOS, D. e EVANGELIDIS, G. (2023), «Integrating Augmented Reality, Gamification, and Serious Games in Computer Science Education», *Education Sciences*.

LOMBARD, M. e DITTON, T. (1997), «At the Heart of It All: The Concept of Presence», *Journal of Computer-Mediated Communication*.

MAESTRI, A., POLSINELLI, P. e SASSOON, J. (2018), *Giochi da prendere sul serio: Gamification, storytelling e game design*, Franco Angeli.

MICHAEL, D. e CHEN, S. (2006), *Serious Games: Games That Educate, Train, and Inform*, Course Technology PTR.

RICE, J. W. (2012), *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*, Pfeiffer.

RIVA, G. e GAGGIOLI, A. (2019), *Realtà virtuali: Gli aspetti psicologici delle tecnologie simulative e il loro impatto sull'esperienza umana*, Giunti Psychometrics.

RIVA, G. e WIEDERHOLD, B. K. (2002), *Virtual Environments in Clinical Psychology and Neuroscience: Methods and Techniques in Advanced Patient–Therapist Interaction*, IOS Press.

ZIKAS, P., BACHLITZANAKIS, V., PAPAEFTHYMIOU, M., KATEROS, S., GEORGIOU, S., LYDATAKIS, N. e PAPAGIANNAKIS, G. (2016), «Mixed Reality Serious Games and Gamification for Smart Education», in «Proceedings of the International Conference on Advanced Learning Technologies (ICALT)», IEEE.

ZYDA, M. (2005), «From Visual Simulation to Virtual Reality to Games», *IEEE Computer*.

Siti web consultati

- VA - Virtual Arena – `www.virtualarena.tech`
- VR Italia – `www.vr-italia.org`
- Wikipedia – `www.wikipedia.org`
- Wessex Archaeology – `www.wessexarch.co.uk`
- Meta Horizon – `https://developers.meta.com/horizon/develop?redirect=all-platform`
- Unity documentation – `https://docs.unity.com/`

Ringraziamenti

GRAZIE!