

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame
e Realtà Virtuale

Progettazione e sviluppo di un videogioco orientato al mercato: il caso "Bloody Shift", dallo sviluppo al testing

Market-Driven Video Game Design: The Case of "Bloody Shift" — From Development to Testing

Candidato:
Giuseppe Cervelli

Relatore:
Dott. Lorenzo Stacchio

Correlatore:
Prof. Adriano Mancini

Anno Accademico
2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame
e Realtà Virtuale

Progettazione e sviluppo di un videogioco orientato al mercato: il caso "Bloody Shift", dallo sviluppo al testing

Market-Driven Video Game Design: The Case of "Bloody Shift" — From Development to Testing

Candidato:
Giuseppe Cervelli

Relatore:
Dott. Lorenzo Stacchio

Correlatore:
Prof. Adriano Mancini

Anno Accademico
2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione per Videogame e Realtà Virtuale
Via Brecce Bianche – 60131 Ancona (AN), Italia

DEDICA

Sommario

Il presente elaborato espone in dettaglio la pipeline metodologica e procedurale seguita durante l'attività di tirocinio per lo sviluppo del software videoludico *Bloody Shift*, ingegnerizzato mediante l'utilizzo del motore grafico *Unreal Engine 5.6.1*.

La soluzione software è stata concepita a monte per allinearsi sinergicamente alle tendenze commerciali del *marketplace* di *Steam* e alle specifiche esigenze del pubblico target; tale approccio ha richiesto una preliminare ed essenziale fase di *brainstorming* collettivo incentrata sul *game design*, integrata da un'attenta analisi comparativa dei *competitor* già esistenti sulla piattaforma.

La trattazione si focalizza, inoltre, sulla descrizione rigorosa dell'architettura logica implementata dal team di programmazione per la traduzione delle meccaniche fondamentali del *core gameplay loop*. Il processo di sviluppo è stato strutturato a partire dalla definizione di classi base in *C++*, successivamente ereditate e specializzate all'interno dei rispettivi *Blueprint*.

Infine, viene delineato il contesto operativo in cui è stato effettuato il primo *deployment* del prototipo, corredato da un'analisi critica e quantitativa dei risultati empirici e dei *feedback* ottenuti dal cluster di utenti *beta-tester*.

Indice

1 Introduzione **1**
- 1.1 Contesto . . . 1
- 1.2 Problema . . . 1
- 1.3 Obiettivo . . . 2
- 1.4 Contributi . . . 3

2 Stato dell'arte e analisi competitiva **4**
- 2.1 Definizione genere Viscera like . . . 4
  - 2.1.1 Viscera Cleanup Detail . . . 4
  - 2.1.2 Crime Scene Cleaner . . . 6
  - 2.1.3 Monster Mopup . . . 7
  - 2.1.4 The Were Cleaner . . . 8
  - 2.1.5 Cash Cleaner Simulator . . . 10
  - 2.1.6 Leaf Blower Co . . . 11
  - 2.1.7 Goblin Cleanup . . . 12
- 2.2 La nostra soluzione . . . 13

3 Metodologia **16**
- 3.1 Brainstorming col team di sviluppo . . . 16
- 3.2 Indagine di mercato . . . 17
- 3.3 Implementazione . . . 18
- 3.4 Testing e Polishing . . . 19

4 Bloody Shift Development **20**
- 4.1 Design & Engineering . . . 20
  - 4.1.1 Deployment . . . 20
  - 4.1.2 Tutorial . . . 21
  - 4.1.3 Gioco Intermedio . . . 24
  - 4.1.4 Fine Gioco . . . 25
- 4.2 Implementation . . . 25
  - 4.2.1 Giocatore . . . 26
  - 4.2.2 Oggetti Interagibili . . . 32
  - 4.2.3 Porte e Ascensore . . . 46
  - 4.2.4 Luci . . . 51
  - 4.2.5 Game Management . . . 53
  - 4.2.6 User Interface . . . 60

4.2.7 Gestione Audio 66
4.3 Testing & Improvements 67

5 Risultati **70**
5.1 Contesto 70
5.1.1 Svilupparty 70
5.1.2 Discussione sui Feedback 71

6 Conclusioni **86**

Elenco delle figure

2.1 Copertina su Steam [1] 5
2.2 Core Loop 5
2.3 Copertina su Steam [2] 6
2.4 Core Loop 6
2.5 Copertina su Steam [3] 8
2.6 Core Loop 8
2.7 Copertina su Steam [4] 9
2.8 Core Loop 9
2.9 Copertina su Steam [5] 10
2.10 Core Loop 10
2.11 Copertina su Steam [6] 11
2.12 Core Loop 11
2.13 Copertina su Steam [7] 12
2.14 Core Loop 12

3.1 Parte della board Miro utilizzata, contenente il pitch del gioco, il level design e alcuni spunti sui sistemi e meccaniche 16
3.2 Foglio Excel sul quale sono elencati i maggiori titoli Viscera-like presenti al momento su Steam 17
3.3 Parte della repository vista tramite interfaccia GitKraken 18

4.1 Diagramma di Deployment 20
4.2 Diagramma delle attività per la prima parte del tutorial 21
4.3 Diagramma delle attività per la seconda parte del tutorial 21
4.4 Diagramma delle attività per la terza parte del tutorial 22
4.5 Diagramma delle attività per la quarta parte del tutorial 23
4.6 Diagramma delle attività per la quarta parte del tutorial 23
4.7 Diagramma delle attività per la quinta parte del tutorial 24
4.8 Diagramma delle attività per parte intermedia del gioco 24
4.9 Diagramma delle attività per la fase finale del gioco 25
4.10 UML delle classi per giocatore e controller 26
4.11 Creazione dei dispatcher in PlayerController 26
4.12 Binding in classe target 26
4.13 Chiamata ai dispatcher in PlayerController 26
4.14 UML delle classi specifico per il giocatore 27
4.15 Parte del Blueprint CharacterController mostrante le funzioni di Tick 28

4.16 Parte del Blueprint CharacterController mostrante la funzione PlayerInput . . . 28
4.17 Parte del Blueprint CharacterController mostrante una sezione del controllo a cascata in PlayerInput . . . 29
4.18 Carrello all'inizio del gioco . . . 29
4.19 UML delle classi per il carrello . . . 30
4.20 Parte del Blueprint Cart mostrante la funzione OnDetach . . . 31
4.21 Parte del Blueprint Cart mostrante la funzione TraceToSpawnPlayer 31
4.22 UML delle classi per InteractableObject . . . 32
4.23 UML delle classi per CleaningTool . . . 32
4.24 Parte del Blueprint di CleaningTool mostrante il BeginPlay . . . 33
4.25 Parte del Blueprint CleaningComp mostrante la funzione CleanMop 34
4.26 Parte del Blueprint CleaningComp mostrante la funzione AddFilthToTool . . . 34
4.27 Mocio . . . 34
4.28 Funzione di interfaccia chiamata dal Giocatore . . . 35
4.29 Movimento tramite Timeline . . . 35
4.30 Rilevamento delle collisioni . . . 36
4.31 Schiacciascarafaggi . . . 36
4.32 Spolverino . . . 36
4.33 Secchio d'acqua . . . 37
4.34 Funzione di distaccamento nel Giocatore . . . 37
4.35 Movimento verticale del mocio . . . 37
4.36 Funzione nel secchio che ridà il mocio al Giocatore . . . 37
4.37 Funzione nel Giocatore che riprende possesso del mocio . . . 37
4.38 UML delle classi per CleanableObject . . . 38
4.39 Parte del Blueprint CleanableObject mostrante il BeginPlay . . . 38
4.40 Primo residuo ematico da pulire . . . 39
4.41 Stanza colma di residui ematici . . . 39
4.42 Parte del Blueprint BloodStain mostrante la funzione BeginPlay . . 39
4.43 Parte del Blueprint BloodStain mostrante il costruttore . . . 40
4.44 Singolo residuo compostabile . . . 41
4.45 UML delle classi per Immondizia, cestino e inceneritore . . . 41
4.46 Sacco di rifiuti & inceneritore . . . 42
4.47 Parte del Blueprint CharacterController, mostrante la logica di generazione della busta di immondizia nella funzione PlayerInput . . . 42
4.48 Contenitore per i rifiuti sul carrello . . . 43
4.49 Parte del Blueprint TrashBin mostrante la funzione DestroyTrash . . 44
4.50 Parte del Blueprint Incinerator mostrante la funzione DestroyTrash e parte della logica di dissolvenza . . . 44
4.51 Parte della funzione OpenIncinerator . . . 45
4.52 Parte della funzione CloseIncinerator . . . 45

4.53 UML delle classi per gli scarafaggi 45
4.54 Parte del Blueprint Cockroach mostrante parte della logica di movimento 46
4.55 Una delle porte neutrali 46
4.56 UML delle classi per le porte 47
4.57 Parte del Blueprint LeftNeutralRoomDoor mostrante parte della logica di apertura chiusura 48
4.58 Parte del Blueprint GuestRoomDoor mostrante parte della funzione StartDialogue 49
4.59 Ascensore aperto 50
4.60 UML delle classi per l'ascensore 50
4.61 Pulsanti nell'ascensore 51
4.62 UML delle classi per le luci 51
4.63 UML delle classi per la GameInstance 53
4.64 Parte del Blueprint GameInstance mostrante l'evento Init 54
4.65 Parte del Blueprint GameInstance mostrante uno dei metodi "Update" 55
4.66 Parte del Blueprint GameInstance mostrante il metodo InitializeUI . 56
4.67 Parte della funzione BeginPlay mostrante il binding ai dispatcher di alcuni oggetti fondamentali 57
4.68 Una delle routine di collisione associate a una delle Trigger VBox . . 57
4.69 Parte della funzione PlayerScript 58
4.70 Parte della funzione UpdateQuest 60
4.71 UML delle classi per la UI 60
4.72 UML delle classi per la UI 61
4.73 Menù principale 62
4.74 Schermata di Fine Partita 63
4.75 Menù di pausa 63
4.76 UML delle classi per la UI 63
4.77 Schermata con il messaggio finale 64
4.78 Dialogo con ospite 64
4.79 Dialogo con ospite 64
4.80 UML delle classi per il comparto audio 66

5.1 Logo dello Svilupparty [8] 71
5.2 Parte delle risposte alla domanda 71
5.3 Diagramma a torta per le risposte a questa domanda 72
5.4 Diagramma a torta per le risposte a questa domanda 72
5.5 Diagramma a torta per le risposte a questa domanda 73
5.6 Diagramma a colonne per le risposte a questa domanda 74
5.7 Diagramma a righe per le risposte a questa domanda 75
5.8 Alcune risposte a questa domanda 75
5.9 Diagramma a colonne per le risposte a questa domanda 76
5.10 Diagramma a torta per le risposte a questa domanda 78

5.11 Diagramma a righe per le risposte a questa domanda 79
5.12 Diagramma a colonne per le risposte a questa domanda 79
5.13 Diagramma a righe per le risposte a questa domanda 80
5.14 Diagramma a colonne per le risposte a questa domanda 81
5.15 Diagramma a torta per le risposte a questa domanda 81
5.16 Diagramma a righe per le risposte a questa domanda 82
5.17 Diagramma a torta per le risposte a questa domanda 83
5.18 Diagramma a colonne per le risposte a questa domanda 84
5.19 Diagramma a colonne per le risposte a questa domanda 84
5.20 Diagramma a colonne per le risposte a questa domanda 85

Elenco delle tabelle

2.1 Matrice comparativa titoli Viscera-like 14

Capitolo 1

Introduzione

1.1 Contesto

Il videogioco rappresenta oggi una delle principali forme di intrattenimento digitale e costituisce un prodotto software caratterizzato dalla convergenza di aspetti tecnologici, progettuali e artistici. A partire dalla diffusione commerciale dei primi videogiochi arcade negli anni Settanta, il settore ha attraversato una progressiva evoluzione sia nelle modalità di fruizione sia nei processi di produzione [9].

Parallelamente alla crescita del mercato videoludico, anche gli strumenti impiegati per lo sviluppo hanno subito una significativa trasformazione. La realizzazione di videogiochi moderni si basa sempre più frequentemente sull'impiego di *game engine*, che integrano all'interno di un unico ambiente funzionalità per il rendering, la gestione della fisica, l'audio, l'interazione e la programmazione delle logiche di gioco. Tra questi, Unreal Engine mette a disposizione sia un ambiente di sviluppo basato su C++ sia il sistema di *visual scripting* Blueprint, consentendo di combinare componenti native e logiche visuali all'interno della medesima architettura software.

Le versioni più recenti del motore introducono inoltre tecnologie quali Nanite e Lumen, progettate rispettivamente per la gestione di geometrie ad alta complessità e per l'illuminazione globale dinamica. Tali strumenti contribuiscono a ridurre alcune attività tradizionalmente onerose della pipeline grafica, come la gestione manuale di differenti livelli di dettaglio e il precalcolo dell'illuminazione, favorendo processi di prototipazione e iterazione più rapidi.

Questa evoluzione degli strumenti di sviluppo assume particolare rilevanza nel contesto delle produzioni indipendenti, nelle quali team di dimensioni ridotte devono spesso conciliare vincoli di tempo e risorse con la necessità di realizzare sistemi di gioco sufficientemente complessi, modulari e facilmente iterabili.

1.2 Problema

La possibilità per molte più persone di accedere a mezzi avanzati ha reso lo sviluppo di un videogioco molto più agevole rispetto al passato. Ciò significa che, al giorno d'oggi, prolifera un gran numero di studi di sviluppo indipendenti che, pur disponendo di risorse umane e tecnologiche limitate, riescono a produrre soluzioni appetibili per

l'ampio mercato [10]. Nonostante ciò, la vita dello sviluppatore indipendente non è certamente facile. Il primo scoglio da superare è l'esiguità delle risorse tecniche, umane ed economiche: nonostante la quantità di titoli prodotti penda a favore degli studi indie ([11] oltre il 90% dei rilasci su Steam), gli studi AAA prevalgono dal punto di vista delle risorse disponibili. Si noti, infatti, che il budget medio di uno studio di quel calibro può superare agevolmente i trecento milioni, mentre le compagnie indipendenti raramente superano i due milioni[12]. Si osservi, inoltre, come molti studi indipendenti tendano a assecondare le fluttuazioni della popolarità o ad allinearsi ai trend di determinati generi. Lo sviluppo videoludico presenta inoltre peculiarità rispetto ad altri processi software, poiché integra componenti di programmazione, design, produzione artistica e audio all'interno di un processo fortemente iterativo [13]. In tale ottica, il concepimento e la progettazione del codice sorgente secondo paradigmi di modularità divengono fondamentali ; la possibilità di riutilizzare i sistemi in più aree del progetto — consente di avere una maggiore manutebilità ed estendibilità del progetto.

Superati i palesi limiti strutturali, le ragioni che impediscono a un progetto indipendente di ottenere il successo sperato sono da rintracciarsi nella fase di progettazione e, successivamente, in quella di marketing. Infatti, il pubblico di Steam (target di riferimento principale) risulta particolarmente esigente e difficile da fidelizzare: la spiccata propensione al rimborso [14] (con tassi compresi tra l'8% e il 12% nel 2024) entro le prime due ore di fruizione conduce molte startup al fallimento immediato, impedendo loro di raggiungere il punto di pareggio o break-even [15] (condizione che interessa circa il 40% dei rilasci). Qualora l'idea alla base della progettazione risulti debole, il titolo è destinato all'insuccesso commerciale anche qualora fosse tecnicamente impeccabile. L'ostacolo principale per gli sviluppatori emergenti non risiede, dunque, soltanto nella scarsità di risorse, ma nella difficoltà di ideare una soluzione capace di catturare istantaneamente l'attenzione dell'utente, sin dal primo contatto visivo con la pagina Steam del prodotto. Ciò impone nella fasi preliminari del progetto un'indagine mirata e settoriale del proprio pubblico target (approccio market-driven); questo viene fatto studiando i titoli di successo rilasciati dai competitor, per poi elaborare scelte di design efficaci che distinguano la soluzione da quello che già c'è sul mercato.

1.3 Obiettivo

Alla luce di quanto premesso, la presente tesi sperimentale documenta le attività svolte durante il tirocinio presso BadVices Games per lo sviluppo di un'esperienza videoludica funzionale e accessibile. L'elaborato approfondisce l'indagine di mercato preliminare, il contributo tecnico offerto nelle fasi di progettazione, implementazione e debugging del codice in sinergia con il team, nonché le scelte di game design adottate sotto la supervisione aziendale per garantire una fruizione fluida e intuitiva del titolo.

1.4 Contributi

Il contributo tecnico apportato dal sottoscritto ha riguardato l'implementazione dei sistemi di locomozione del personaggio e la gestione delle interazioni con gli oggetti di gioco, nonché la definizione delle relazioni logiche intercorrenti tra gli stessi. Tale attività si è estesa allo sviluppo del sistema di obiettivi (quest system) e alla realizzazione dell'interfaccia utente (UI) mediante l'impiego di Widget Blueprint. Infine, è stata curata la logica per la configurazione e la persistenza delle opzioni di sistema, unitamente alla gestione del comparto audio.

Capitolo 2

Stato dell'arte e analisi competitiva

In base ai suggerimenti di ognuno dei membri, si è deciso unanimamente di sviluppare un titolo "Viscera-like", ispirato al più celebre titolo *Viscera Cleanup Detail*.

2.1 Definizione genere Viscera like

Nel corso dell'analisi sulle attuali tendenze di mercato della piattaforma Steam, è emersa una significativa espansione del segmento relativo ai titoli basati sulla simulazione di specifiche mansioni lavorative. All'interno di tale filone si colloca, acquisendo sempre maggiore rilevanza, il sottogenere dei cosiddetti "Viscera-like". Si identifica come "Viscera-like" infatti una branca dei simulatori di pulizia, nel quale il giocatore veste i panni di un'operatore ecologico il quale compito è ripristinare l'ordine in scenari post-catastrofici, solitamente di matrice sci-fi o horror. Le direttrici (anche fulcro dell'elaborato progettuale) fondamentali che definiscono il genere sono:

- **Pulizia e riordino:** Il *core loop* si basa sulla trasformazione di un ambiente caotico e visivamente disturbante in uno spazio asettico e ordinato. Questo processo genera una soddisfazione psicologica legata al completamento metodico di compiti ripetitivi.
- **Narrazione ambientale:** L'assenza di una trama esplicita viene compensata dalla ricostruzione degli eventi tramite l'osservazione dei resti sulla scena e la disposizione degli oggetti pulibili/interagibili in gioco. Ciò permette al giocatore di dedurre la cronologia del disastro appena avvenuto, rendendo l'esplorazione parte integrante dell'esperienza.

2.1.1 Viscera Cleanup Detail

2.1Tale titolo rappresenta il capostipite del sottogenere dei "Viscera-like", nonché l'opera di maggior successo tra quelle analizzate. La sua influenza è stata determinante per lo sviluppo di ogni produzione analoga successiva, fungendo da principale riferimento sia per l'implementazione delle meccaniche di gioco che per la definizione

del ciclo d'interazione principale (core loop). Ambientato in un contesto fantascientifico dalle atmosfere futuristiche, il titolo richiede al giocatore di bonificare e ripristinare l'ordine in un'area teatro di una cruenta battaglia. É possibile il multigiocatore.



Figura 2.1: Copertina su Steam [1]

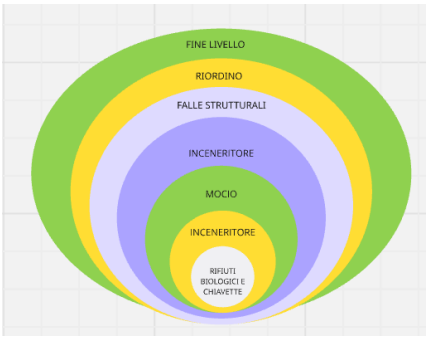


Figura 2.2: Core Loop

Meccaniche e sistemi

- **Mocio:** Utilizzato per la rimozione di macchie e fluidi di varia natura.
- **Inceneritore:** Impiegato per lo smaltimento dei rifiuti organici (es. resti anatomici).
- **Contenitori:** Elementi necessari per il trasporto simultaneo di molteplici frammenti biologici verso l'inceneritore, ottimizzando i tempi di smaltimento.
- **Paperwork:** Registro documentale impiegato per tracciare gli obiettivi attivi (missioni) e riepilogare il punteggio ottenuto al termine di ciascun livello.
- **Tablet:** Dispositivo impiegato per l'annotazione di dati funzionali al completamento degli obiettivi. È inoltre equipaggiato con una funzione di scansione per leggere le targhette identificative rinvenute sui cadaveri, ai fini dell'incremento del punteggio finale.
- **Fucile saldatore:** Strumento adibito alla riparazione delle falle strutturali (muri), la cui efficacia meccanica è subordinata al corretto tempismo d'esecuzione (*timing*) da parte del giocatore.
- **Sniffer:** Dispositivo di rilevamento utilizzato per acquisire informazioni spaziali (*intelligence*) sui residui ancora da bonificare e per localizzarne l'esatta posizione nell'ambiente.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.

- **Distributore:** Terminale d'acquisto per l'approvvigionamento di oggetti di rimpiazzo, i quali dovranno essere ricollocati strategicamente all'interno della mappa durante la fase di riordino.
- **Punteggio:** Sistema di valutazione che associa a ogni missione completata un valore specifico, consultabile direttamente dall'hub principale dell'applicazione.

Analisi Contestuale

Rappresentando la principale fonte di ispirazione sia per il progetto in esame sia per le produzioni successive afferenti al medesimo genere, Viscera Cleanup Detail non mostra criticità di game design degne di nota. Il titolo si distingue tuttavia dai prodotti concorrenti per l'implementazione di una modalità multigiocatore (multiplayer) e per la spiccata varietà e originalità della strumentazione di bonifica fornita all'utente.

2.1.2 Crime Scene Cleaner

2.3Titolo fortemente incentrato sulla componente narrativa, in cui il protagonista è incaricato di bonificare svariate tipologie di scene del crimine, teatro di efferati delitti.



Figura 2.3: Copertina su Steam [2]

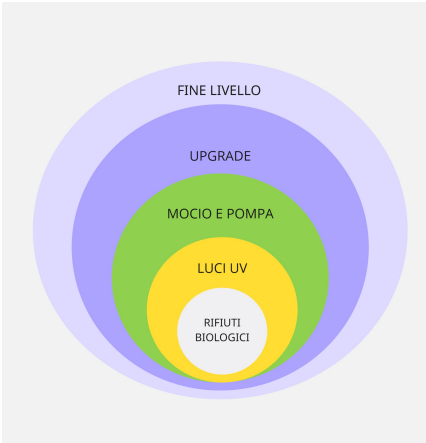


Figura 2.4: Core Loop

Meccaniche e sistemi

- **Mocio:** Strumento base utilizzato per la rimozione di macchie e fluidi di varia natura.
- **Pompa ad acqua:** Dispositivo con funzionalità analoghe a quelle del mocio, ma progettato per consentire la bonifica delle superfici mantenendo una distanza di sicurezza.

- **Automezzo (Pick-up):** Veicolo impiegato per la raccolta e lo smaltimento simultaneo di molteplici frammenti biologici e rifiuti, fungendo da punto di scarico principale.
- **Lampada UV:** Strumento impiegato per il rilevamento di fluidi e residui organici non altrimenti visibili a occhio nudo.
- **Tablet:** Dispositivo impiegato per l'annotazione di dati funzionali al completamento degli obiettivi. È inoltre equipaggiato con una funzione di scansione per leggere le targhette identificative rinvenute sui cadaveri, ai fini dell'incremento del punteggio finale.
- **Accetta:** Strumento adibito all'abbattimento e alla distruzione di ostacoli ambientali che precludono l'esplorazione.
- **Albero delle abilità (*Skill Tree*):** Sistema di progressione che consente al giocatore di monitorare e potenziare specifiche caratteristiche. Tale sistema è temporaneo, non prevedendo la persistenza dei potenziamenti tra missioni differenti.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.
- **Hub Centrale:** Area neutrale (zona di intermezzo) dalla quale l'utente può selezionare e avviare le diverse missioni, oltre ad approfondire l'arco narrativo del protagonista.

Analisi Contestuale

Crime Scene Cleaner si distingue nel panorama dei titoli concorrenti per la complessità della componente narrativa e per l'approfondita caratterizzazione del protagonista tramite la consultazione di reperti rinvenuti nella hub centrale, oltre che per un elaborato sistema di progressione strutturato attraverso un albero delle abilità (skill tree). Un ulteriore elemento di rilievo è costituito dalla parziale distruttibilità ambientale, introdotta nel ciclo di interazione tramite la meccanica dell'accetta. Di contro, la totale assenza di elementi procedurali all'interno dei livelli rischia di rendere l'esperienza di gioco (gameplay) ripetitiva una volta portata a compimento la campagna principale.

2.1.3 Monster Mopup

Questo videogioco, la cui copertina è riportata in 2.5, si distingue per tonalità decisamente più solari e distese rispetto alle produzioni precedentemente esaminate, risultando totalmente priva di componenti cruente o di natura gore e splatter. Il fulcro dell'esperienza risiede nel compito affidato al protagonista incaricato di bonificare il

disordine generato da creature dall'aspetto rassicurante e caricaturale, provvedendo contestualmente al loro recupero e alla loro messa in sicurezza.



Figura 2.5: Copertina su Steam [3]

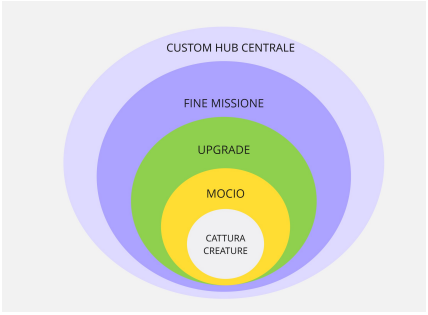


Figura 2.6: Core Loop

Meccaniche e sistemi

- **Mocio e secchio:** Strumenti base utilizzato per la rimozione di macchie e fluidi di varia natura; il mocio va pulito nel secchio.
- **Inceneritore:** Impiegato per lo smaltimento dei rifiuti organici.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.
- **Trappole:** Usate per catturare le creature che corrono in giro per la mappa.
- **Mercatino:** Utilizzabile por il potenziamento degli strumenti per l'aumento dell'efficienza.
- **Hub Centrale:** Area neutrale (zona di intermezzo) dalla quale l'utente può selezionare e avviare le diverse missioni; è possibile personalizzare i colori e l'arredamento.

Analisi Contestuale

Questo titolo si differenzia dalla concorrenza per il totale distacco da tematiche e immaginari cruenti, a favore di un'estetica stilizzata e vivace (cartoon), che rende l'esperienza sensibilmente più accessibile a un target demografico più giovane. Di contro, anche in questa produzione si riscontra una ridotta rigiocabilità (replayability), in quanto l'esperienza di gioco non è stata strutturata per incentivare successive sessioni dopo il primo completamento.

2.1.4 The Were Cleaner

2.7Assumendo il ruolo di un protagonista capace di tramutarsi in licantropo, il giocatore ha l'incarico di bonificare gli spazi di un ambiente d'ufficio. Durante

lo svolgimento delle missioni, l'utente dovrà prestare attenzione alle routine di pattugliamento dei Personaggi Non Giocanti (NPC), adottando un approccio furtivo al fine di non destare sospetti qualora operi nella sua forma ferina.



Figura 2.7: Copertina su Steam [4]

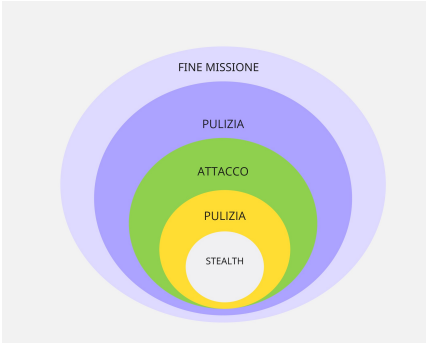


Figura 2.8: Core Loop

Meccaniche e sistemi

- **Pompa ad acqua:** Dispositivo progettato per consentire la bonifica delle superfici e di rifiuti compostabili..
- **Minimappa:** Utilizzata per orientarsi nella mappa in base alla posizione degli NPC.
- **Furtività:** Il giocatore dovrà prestare attenzione allo steto degli NPC e alle loro ronde di perlustrazione, per evitare di incrementare il sospetto. Lo stato del personaggio non giocante è indicato da un'icona sulla testa del suddetto.
- **Attacco:** Nel caso in cui un NPC passi in stato di allerta, si può scegliere di aggredirlo ed eliminarlo. Questo causerà la comparsa di nuovi residui da pulire con la pompa dell'acqua.

Analisi Contestuale

Questo titolo si differenzia principalmente dai prodotti concorrenti per l'adozione di una prospettiva isometrica in terza persona, in netto contrasto con la canonica inquadratura in prima persona che caratterizza il genere. Sotto il profilo estetico e del gameplay, emerge una marcata dicotomia tra lo stile grafico stilizzato (cartoon) e la presenza di elementi gore, a cui si affianca un'interazione sistemica con agenti dotati di intelligenza artificiale (AI) durante le fasi furtive (stealth). L'assenza di funzionalità multigiocatore e di logiche procedurali ne limita tuttavia la rigiocabilità complessiva, riducendo l'incentivo a sessioni di gioco reiterate; a ciò si aggiunge una limitata varietà per quanto concerne la strumentazione di bonifica messa a disposizione dell'utente.

2.1.5 Cash Cleaner Simulator

2.9All'interno di questo titolo, il giocatore assume il ruolo di un operatore incaricato di ricevere, bonificare e riordinare del denaro di dubbia provenienza. A seguito dell'accettazione di specifici contratti, l'utente riceverà diverse sacche contenenti banconote contaminate, le quali verranno rilasciate dall'alto all'interno dell'area di lavoro. Il ciclo d'interazione prevede lo svuotamento di tali contenitori, la pulizia fisica del denaro e il successivo posizionamento dello stesso su un apposito nastro trasportatore, al fine di finalizzarne la consegna.



Figura 2.9: Copertina su Steam [5]

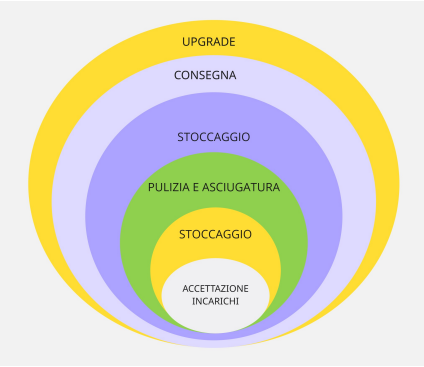


Figura 2.10: Core Loop

Meccaniche e sistemi

- **Lavatrice:** Macchinario progettato per la bonifica automatizzata di considerevoli volumi di banconote contaminate.
- **Asciugatrice:** Apparecchiatura impiegata per il processo di asciugatura del denaro precedentemente sottoposto al ciclo di lavaggio.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.
- **Smartphone:** Dispositivo mobile adibito all'accettazione dei contratti e alla consultazione degli obiettivi attivi. Viene impiegato, inoltre, come interfaccia per l'acquisto di vari potenziamenti (*upgrade*) legati alla progressione di gioco.
- **Contaminazione:** Meccanica sistemica per la quale il contatto fisico tra banconote ancora sporche, o umide, e quelle già bonificate ne provoca l'immediata alterazione, vanificando il lavoro svolto.
- **Contenitori:** Elementi logistici impiegati per il trasporto simultaneo di molteplici mazzette di denaro, facilitandone sia la movimentazione all'interno dell'area di lavoro che le operazioni di stoccaggio e organizzazione.

Analisi Contestuale

Questo prodotto reinterpreta la definizione di Viscera-like alla luce di una premessa fondamentale: il giocatore non è più chiamato a bonificare interi ambienti tridimensionali, bensì a ripulire letteralmente il denaro sporco. Nonostante l'assenza di un comparto narrativo strutturato o di funzionalità multigiocatore, questo incipit ha permesso agli sviluppatori di elaborare meccaniche e un core loop magnetico e accattivante. Quest'ultimo si basa sull'accumulo incrementale del punteggio e sul conseguente acquisto di potenziamenti (upgrade) e di ulteriori elementi logistici, convertendo di fatto l'esperienza di gioco in una vera e proprio dinamica gestionale.

2.1.6 Leaf Blower Co

2.11Utilizzando l'abitazione della protagonista come base operativa, il giocatore è chiamato ad assumere molteplici contratti per la bonifica e il ripristino delle aree verdi all'interno del quartiere residenziale.



Figura 2.11: Copertina su Steam [6]

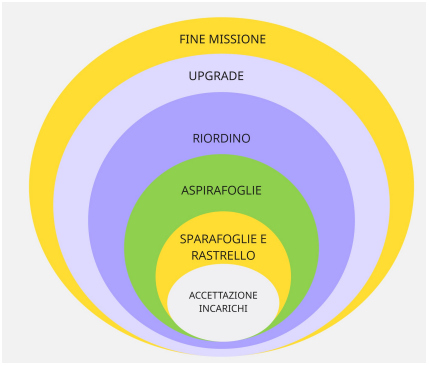


Figura 2.12: Core Loop

Meccaniche e sistemi

- **Soffiatore e Rastrello:** Strumenti impiegati per lo spostamento e l'accumulo strategico di detriti vegetali e oggetti di varia natura.
- **Aspira-foglie:** Dispositivo adibito all'aspirazione e alla rimozione di considerevoli accumuli di fogliame.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.
- **Lanciafiamme:** Strumento impiegato per l'incenerimento e la rapida rimozione di elementi infestanti, quali funghi, rovi ed erbacce.

- **Scatto (Sprint):** Meccanica di locomozione che incrementa la velocità di spostamento del personaggio, inibendo tuttavia temporaneamente l'utilizzo degli strumenti equipaggiati.
- **Contenitori:** Elementi logistici impiegati per la raccolta e il trasporto simultaneo di detriti e scarti vegetali, facilitandone le operazioni di stoccaggio e smaltimento.
- **Ricarica energetica:** Meccanica gestionale che impone all'utente di monitorare e ripristinare ciclicamente il livello di batteria delle apparecchiature elettriche in uso.
- **Negozio:** Interfaccia o area dedicata all'acquisto di potenziamenti (*upgrade*) ed equipaggiamenti supplementari, funzionali alla progressione del gioco.

Analisi Contestuale

Leaf Blower Co. si discosta ulteriormente dalla canonica bonifica di ambienti interni, focalizzandosi sulla pulizia e sul riordino di spazi esterni. Questa variazione di ambientazione determina l'introduzione di un'attrezzatura specifica legata alle attività di giardinaggio, nonché la totale esclusione di stilemi visivi cruenti. In aggiunta a ciò, risulta di rilevante importanza l'implementazione di un hub centrale, un'interfaccia o area di gioco dedicata dalla quale l'utente ha la possibilità di selezionare e avviare le diverse missioni.

2.1.7 Goblin Cleanup

2.13Assumendo il ruolo di una giovane protagonista, il giocatore è chiamato a bonificare e ripristinare l'ordine all'interno di vari dungeon, intervenendo successivamente al passaggio e alle incursioni di gruppi di avventurieri. L'opera si discosta nettamente dalle tematiche cruente e dalle componenti gore, in virtù di una direzione artistica marcatamente cartoon e fortemente ispirata all'animazione giapponese. I livelli sono presettati o generati proceduralmente.



Figura 2.13: Copertina su Steam [7]

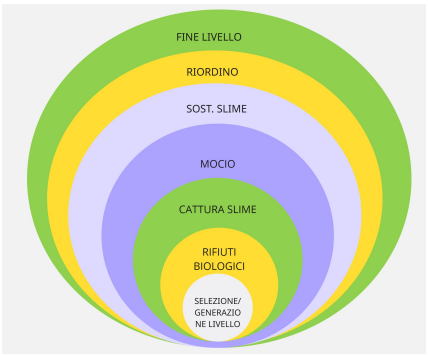


Figura 2.14: Core Loop

Meccaniche e sistemi

- **Mocio e Slime:** Strumento impiegato per la rimozione di macchie di varia natura. A differenza delle meccaniche tradizionali basate sull'uso di un secchio, l'utilizzo di questo strumento è vincolato alla raccolta di specifiche creature melmose (*slime*). L'esaurimento delle cariche a disposizione è segnalato visivamente dal mutamento cromatico della creatura (che transita dal blu al rosso), indicandone all'utente la necessaria e immediata sostituzione.
- **Presa oggetti (Grab):** Meccanica basilare implementata per consentire al giocatore di afferrare, manipolare e ispezionare fisicamente i vari elementi di gioco.
- **Riordino:** Meccanica ambientale che richiede la ricollocazione spaziale degli oggetti: ogni elemento decontestualizzato o fuori posto deve essere riposizionato nella sua sede originaria.
- **Trappole:** Pericoli ambientali e meccanismi ostili in grado di infliggere danni al protagonista, potendo condurre al fallimento della missione (*game over*).
- **Mimic:** Creatura iconica dell'immaginario *fantasy*, impiegata all'interno del ciclo d'interazione come unità di smaltimento per gli *slime* esauriti e per i vari rifiuti organici.
- **Torcia:** Specifica entità o creatura impiegata per l'interazione con l'ambiente, preposta in particolar modo alla riaccensione delle fonti d'illuminazione (fiaccole) all'interno del *dungeon*

Analisi Contestuale

Goblin Cleanup si differenzia dai prodotti concorrenti principalmente per una presenza contenuta di elementi cruenti — i quali risultano comunque fortemente stilizzati — e per l'introduzione della generazione procedurale dei livelli, un fattore che garantisce un'elevata varietà strutturale e rende ogni sessione di gioco unica. Inoltre, la scelta di un comparto artistico orientato verso uno stile caricaturale e vicino all'estetica anime consente di reinterpretare in modo originale e creativo meccaniche d'interazione già ampiamente consolidate all'interno del genere.

2.2 La nostra soluzione

Sulla base dei risultati emersi dall'indagine di mercato, si è proceduto a un'analisi comparativa delle soluzioni già presenti nel settore. Tale approccio è stato adottato al fine di concepire meccaniche e scelte di design originali, necessarie per rendere il prodotto competitivo e attraente per l'utenza della piattaforma Steam.

Tabella 2.1: Matrice comparativa titoli Viscera-like

Titolo / Gioco	**Mocio**	**Disfacimento rifiuti vari**	**Upgrade**	**Contenitor**	**Storia sviluppata**	**Gore / Splatter**
Viscera Cleanup Detail	✓ ✓		✓		✓	
Crime Scene Cleaner	✓	✓	✓	✓	✓	✓
Monster Mopup	✓	✓	✓			
The Were Cleaner		✓			✓	✓
Cash Cleaner Simulator			✓	✓		
Leaf Blower Co		✓	✓	✓		
Goblin Cleanup	✓	✓				
Bloody Shift	✓	✓		✓	✓	✓

Come illustrato nella tabella comparativa, l'analisi sinottica dei diversi titoli appartenenti al filone Viscera-like evidenzia una marcata convergenza di caratteristiche e precise analogie strutturali nelle meccaniche di gioco. Si evince, infatti, come l'interazione legata al mocio sia costantemente riproposta in tutte le produzioni incentrate sulla rimozione di residui liquidi; allo stesso modo, ricorrono con frequenza sistemi dedicati allo smaltimento dei rifiuti e all'impiego di appositi contenitori, volti a ottimizzare la gestione logistica e ad agevolare il trasporto simultaneo di un maggior quantitativo di materiali. Un ulteriore tratto ampiamente condiviso è rappresentato dall'inclusione di un sistema di progressione che consente il potenziamento dell'equipaggiamento a disposizione dell'utente lungo l'arco dell'esperienza. Al contrario, elementi quali un comparto narrativo approfondito e l'adozione di espliciti stilemi gore costituiscono caratteristiche decisamente meno diffuse e peculiari soltanto di specifiche produzioni.

Alla luce delle opportune riflessioni sui pattern ricorrenti all'interno delle meccaniche dei titoli analizzati, si è giunti alla conclusione che la gratificazione e l'appagamento dell'utente, nei giochi appartenenti al filone dei Viscera-like, derivano principalmente dai seguenti fattori:

- **Bonifica e lucidatura:** Il processo di pulizia profonda e di ripristino estetico di ambienti caratterizzati da pavimentazioni piastrellate e superfici riflettenti, elementi in grado di restituire un immediato e soddisfacente *feedback* visivo.
- **Sistemi logistici dinamici:** La disponibilità di soluzioni di locomozione o di trasporto mobile, funzionali all'ottimizzazione dei processi di smaltimento dei rifiuti e al corretto stoccaggio degli utensili da lavoro.

- **Estetica e tematiche *gore*:** L'impatto visivo di scenari marcatamente cruenti, unito all'interazione fisica e al successivo smaltimento di fluidi e resti anatomici.
- **Diversificazione delle metodologie di bonifica:** L'introduzione di differenti e distinte modalità di pulizia, volte a garantire la varietà dell'esperienza di gioco e a evitare che il ciclo d'interazione principale (*core loop*) risulti eccessivamente monotono o ripetitivo.

Si è pertanto deciso di focalizzare l'attività di innovazione su questi elementi mirati e definiti:

- **Carrello logistico:** Elemento che consente al giocatore di avere sempre a immediata disposizione i vari utensili per la bonifica, unitamente al contenitore per lo smaltimento dei rifiuti, ottimizzando in tal modo gli spostamenti e i flussi di lavoro.
- **Strumentazione diversificata:** Il sistema prevede la possibilità di bonificare tre distinte tipologie di contaminazione, in aggiunta alla necessaria gestione e rimozione dei detriti solidi distribuiti all'interno del livello.
- **Contesto narrativo e NPC (Ospiti):** L'esperienza è implicitamente ambientata in una dimensione infernale caratterizzata da atmosfere cupe. All'interno di specifiche aree del livello sono collocate tre entità, ispirate al folklore popolare, incaricate di assegnare al giocatore obiettivi dinamici e missioni da completare.
- **Estetica *gore*:** Il titolo è caratterizzato da una massiccia e rilevante presenza di elementi visivi cruenti e resti organici, distribuiti coerentemente in tutto l'ambiente di gioco.

Capitolo 3

Metodologia

3.1 Brainstorming col team di sviluppo

In fase embrionale, precedentemente all'avvio dell'attività tecnica sul motore grafico, è stato condotto un processo di brainstorming e confronto tra i membri del team. In tale ambito, la scelta del genere videoludico è avvenuta in modo collegiale, basandosi sulle inclinazioni e sulle visioni dei singoli componenti[**?**].



Figura 3.1: Parte della board Miro utilizzata, contenente il pitch del gioco, il level design e alcuni spunti sui sistemi e meccaniche

Per facilitare tale operazione è stata utilizzata la piattaforma Miro 3.1, la quale ha consentito la condivisione di idee in tempo reale e la tracciabilità sistematica dei requisiti preliminari e delle caratteristiche fondamentali del progetto.

Nella fase iniziale, attraverso l'utilizzo di una board di progetto, il team ha collaborato sinergicamente per la selezione del genere di riferimento: ciascun membro ha esposto le proprie abitudini di consumo videoludico, giungendo infine alla decisione unanime di sviluppare un titolo afferente al filone Viscera-like. Successivamente, si è proceduto alla formulazione del pitch (l'incipit) del gioco; questa specifica fase ha presentato una notevole complessità, poiché ha richiesto un'attenta immedesimazione nel profilo dell'utente medio della piattaforma Steam. L'obiettivo primario era infatti l'individuazione di efficaci espedienti di design e comunicazione, capaci di catturare l'attenzione del potenziale acquirente fin dalla prima consultazione della pagina di presentazione del prodotto. In base ai suggerimenti di ognuno dei membri, si è deciso unanimamente di sviluppare un titolo "Viscera-like", ispirato al più celebre titolo *Viscera Cleanup Detail*.

In seguito, il team ha provveduto alla ripartizione delle mansioni relative alla fase di progettazione iniziale. Nello specifico, i modellatori si sono dedicati alla ricerca e alla raccolta di riferimenti visivi (reference) e alla prima strutturazione del level

design, mentre il comparto di programmazione ha curato la definizione concettuale delle diverse meccaniche e dei sistemi di gioco (gameplay systems).

L'approccio adottato segue quindi una logica iterativa e centrata sull'esperienza del giocatore, in cui ideazione, prototipazione e valutazione vengono progressivamente raffinate durante lo sviluppo [16].

3.2 Indagine di mercato

In seguito alla decisione unanime del team di sviluppo, si è proceduto a un'analisi comparativa dei titoli già presenti sulla piattaforma Steam, esaminandone dettagliatamente le caratteristiche, l'ambientazione (setting), le meccaniche, i sistemi di gioco e il ciclo d'interazione principale (core loop), in virtù di produrre un titolo che proponga dal punto di vista di design soluzioni innovative.

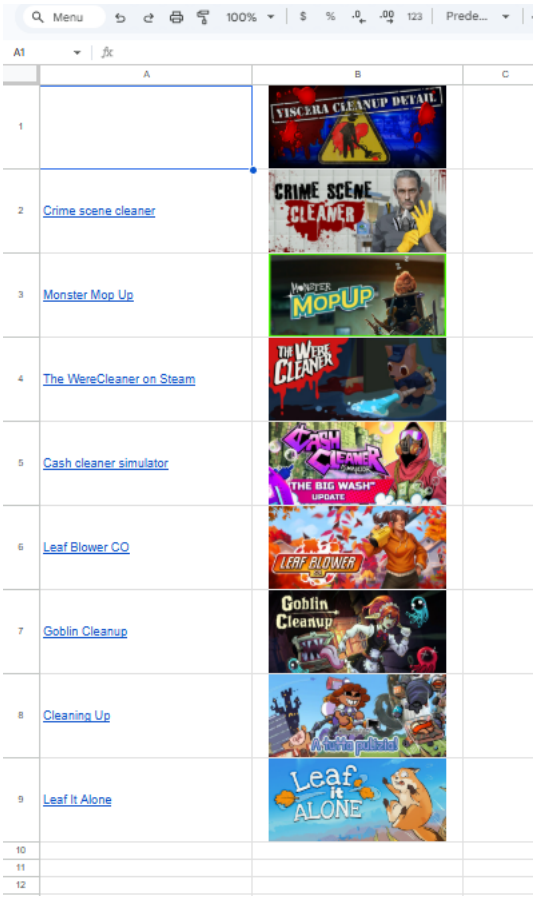


Figura 3.2: Foglio Excel sul quale sono elencati i maggiori titoli Viscera-like presenti al momento su Steam

Al fine di strutturare in modo sistematico i titoli oggetto di studio, l'impiego del software Excel 3.2 si è rivelato fondamentale per raggruppare le produzioni appartenenti a generi affini all'interno di fogli di lavoro distinti.

3.3 Implementazione

Successivamente, si è proceduto a un'analisi delle funzionalità principali e delle classi da implementare, definendo contestualmente l'architettura generale del software. Sotto il profilo dello sviluppo, si è optato per l'implementazione della quasi totalità delle logiche di gioco avvalendosi dei Blueprint di Unreal Engine (sistema di visual scripting), strutturati a partire da classi base redatte in C++. Infine, è stata pianificata la suddivisione modulare delle classi e la configurazione del sistema di controllo di versione tramite repository GitHub, al fine di garantire una tracciatura completa e rigorosa dell'evoluzione del progetto.

Al fine di ottimizzare le tempistiche di sviluppo, si è rivelato fondamentale progettare l'architettura del codice secondo un approccio modulare e scalabile; in questo modo, l'eventuale integrazione di nuove meccaniche nelle fasi successive della produzione avrebbe potuto realizzarsi in modo fluido, senza richiedere una ristrutturazione radicale della base di codice preesistente. Tale obiettivo è stato conseguito sfruttando i paradigmi dell'ereditarietà e i costrutti della Programmazione Orientata agli Oggetti (OOP) nativi di Unreal Engine. Parallelamente, si è fatto ricorso, ove possibile, a interfacce specifiche (blueprint interfaces) destinate a raggruppare le funzioni cardine di determinati gruppi di classi. Questa scelta architetturale risponde alla necessità di ottimizzare le prestazioni complessive dell'applicativo, riducendo l'accoppiamento tra le classi e aumentando la manitebiòità e scalabilità del codice [17].

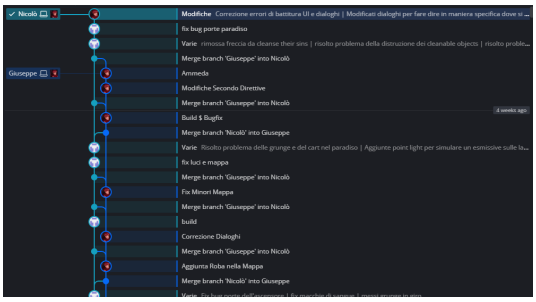


Figura 3.3: Parte della repository vista tramite interfaccia GitKraken

L'impiego del sistema di controllo versione (versioning) offerto da GitHub, coadiuvato dall'interfaccia grafica (GUI) GitKraken 3.3, si è rivelato di fondamentale importanza per coordinare lo sviluppo parallelo delle diverse componenti del codice all'interno di progetti distinti. Questa metodologia ha agevolato la successiva operazione di fusione (merge) in un unico ambiente di lavoro unificato, passaggio propedeutico all'implementazione di classi basate sul design pattern Singleton, necessarie alla gestione (management) della sessione di gioco e del sistema di missioni (quest). Parimenti cruciale è risultata l'integrazione del modulo Git LFS (Large File Storage) unitamente alla corretta strutturazione del file .gitignore; tali accorgimenti hanno consentito, rispettivamente, il tracciamento ottimale dei file binari e degli

asset ad alta risoluzione e l'esclusione sistematica dai commit di file temporanei o non pertinenti alla build.

Successivamente, si è proceduto all'implementazione delle meccaniche fondamentali alla base del titolo, partendo dal sistema di locomozione del personaggio e dalle logiche di interazione con gli oggetti. In una seconda fase, lo sviluppo si è concentrato sulla macro-gestione delle missioni e sulla progressione dell'utente all'interno del livello di tutorial. Infine, l'attività si è conclusa con la configurazione del comparto audio e con l'importazione all'interno del motore di gioco degli asset e dei modelli 3D, precedentemente realizzati in Blender, procedendo al loro posizionamento nella mappa del livello.

3.4 Testing e Polishing

Quest'ultima fase di sviluppo si è focalizzata prevalentemente sull'affinamento delle logiche fondamentali e del design di gioco, nonché sull'ottimizzazione dell'impatto visivo complessivo. Tale risultato è stato perseguito attraverso l'impiego di caratteri tipografici personalizzati, l'integrazione di effetti visivi (VFX) e l'utilizzo di volumi di post-elaborazione (post-process volume) all'interno della scena.

Il prototipo è stato inizialmente sottoposto a sessioni di collaudo (playtesting) da parte di una cerchia ristretta di utenti vicini al team di sviluppo. Il playtesting costituisce infatti una componente centrale del processo iterativo di game design, poiché consente di confrontare le intenzioni progettuali con l'esperienza effettivamente osservata negli utenti [16]. Successivamente, il progetto è stato presentato e testato su più vasta scala di fronte a un pubblico più ampio, in occasione dell'evento Svilupparty tenutosi a Bologna.

Capitolo 4

Bloody Shift Development

4.1 Design & Engineering

All'interno di questa sezione, propedeutica alla fase di implementazione tecnica, verrà delineata una panoramica dell'architettura generale dell'applicativo. Nello specifico, avvalendosi di un dettagliato diagramma delle attività (Activity Diagram), sarà illustrato il percorso d'interazione ideale che l'utente è chiamato a seguire durante l'esperienza di gioco. Contestualmente, saranno analizzate le scelte di design adottate al fine di indirizzare in modo intuitivo anche il giocatore meno metodico verso la curva di progressione stabilita dal team di sviluppo.

4.1.1 Deployment

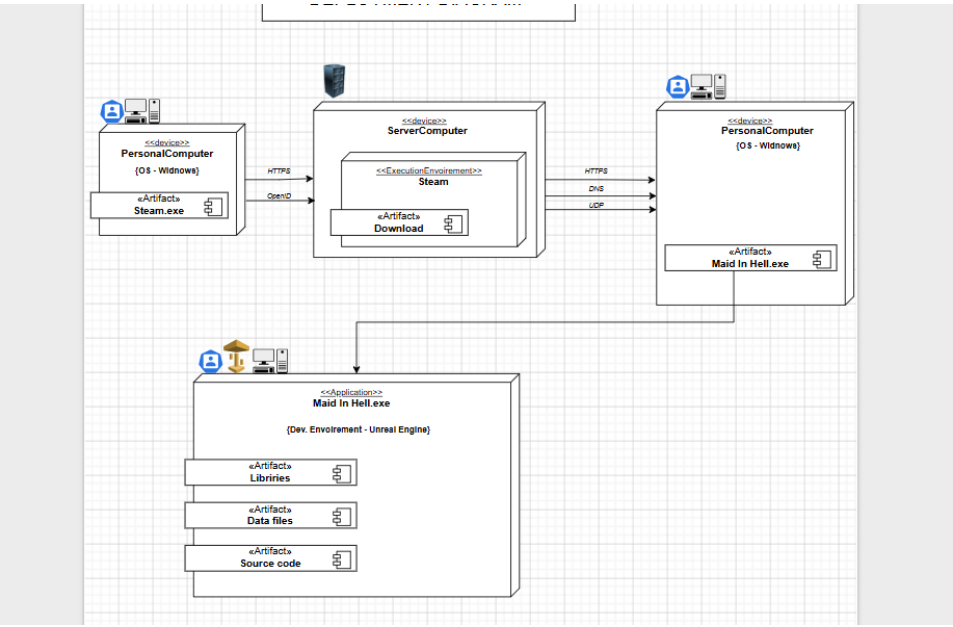


Figura 4.1: Diagramma di Deployment

In fase preliminare, l'utente è chiamato ad accedere al client desktop della piattaforma Steam al fine di procedere con l'acquisizione del titolo. Attraverso l'apposita

pagina del negozio digitale, avviando la procedura di download, il sistema provvederà a scaricare e configurare l'eseguibile dell'applicativo direttamente sul terminale locale.

4.1.2 Tutorial

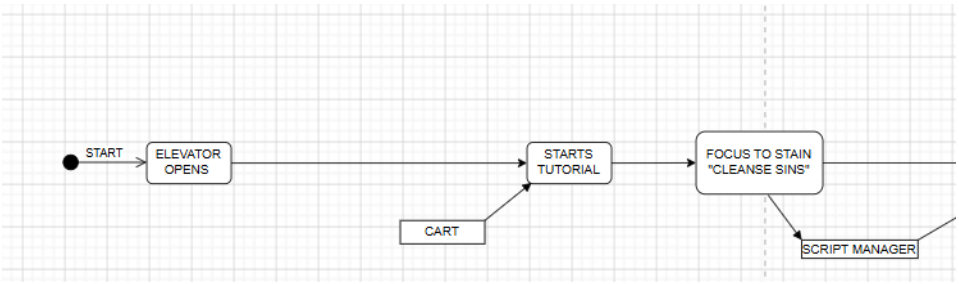


Figura 4.2: Diagramma delle attività per la prima parte del tutorial

In seguito alla selezione dell'opzione di avvio dal menù principale, l'utente inizia la sessione di gioco all'interno di un ascensore, trovandosi già al controllo dell'unità logistica mobile (il carrello). All'apertura delle porte, il giocatore è indotto ad avanzare nell'ambiente tridimensionale avvalendosi del canonico sistema di movimento basato sui tasti WASD.

Percorsa una breve distanza, il sistema impone una rotazione scriptata della telecamera, focalizzando l'attenzione dell'utente su un messaggio tracciato sulla parete, il quale costituisce il primo residuo liquido da bonificare. La natura della scritta ("Cleanse their sins") e la simultanea evidenziazione visiva del mocio posto sul carrello forniscono un chiaro feedback contestuale: tali espedienti visivi sono progettati per suggerire in modo intuitivo al giocatore la necessità di disancorarsi dall'unità mobile, equipaggiare lo strumento e procedere alle operazioni di pulizia. La pulizia di ogni residuo liquido, o dopo di qualsiasi elemento pulibile causerà l'immediato aggiornamento della lista obiettivi presente nell'angolo superiore a sinistra.

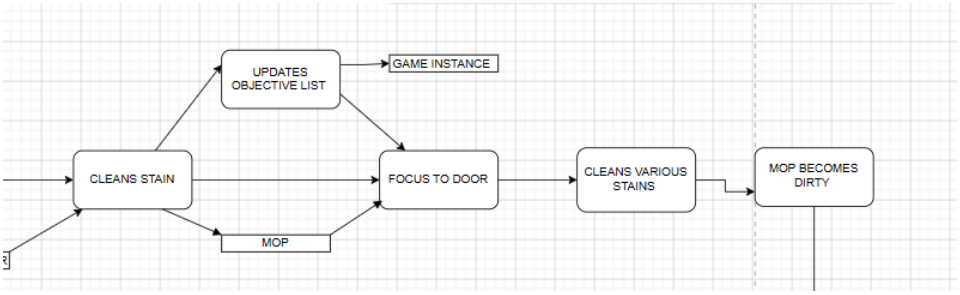


Figura 4.3: Diagramma delle attività per la seconda parte del tutorial

In seguito alla rimozione del primo residuo, la telecamera ruota automaticamente in direzione di una porta che si apre più avanti nel percorso. Questo specifico stimolo visivo suggerisce indirettamente all'utente di riposizionare il mocio sul carrello

logistico, riprenderne il controllo operativo e direzionarsi oltre la soglia. Una volta effettuato l'accesso al nuovo ambiente e attivate le fonti di illuminazione tramite l'interazione con l'interruttore, il giocatore si troverà di fronte a diverse contaminazioni analoghe a quella precedentemente trattata. In questo modo, l'utente è indotto a reiterare la meccanica appena appresa, consolidando l'apprendimento del ciclo di interazione.

Al raggiungimento di una determinata soglia di utilizzo, il sistema notifica all'utente — attraverso un'interfaccia testuale a schermo e un segnale acustico (audio cue) — la saturazione del mocio, segnalando la necessità di effettuarne il lavaggio. Il giocatore è quindi indotto a ritornare presso il carrello logistico per interagire con il secchio, il quale viene evidenziato visivamente tramite un effetto di illuminazione (highlighting) analogo a quello applicato poco prima allo strumento di pulizia per guidare l'attenzione dell'utente.

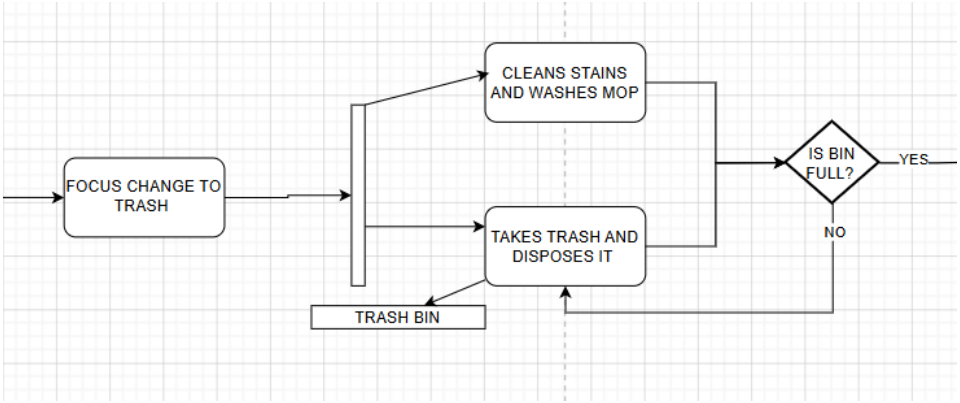


Figura 4.4: Diagramma delle attività per la terza parte del tutorial

Una volta completata la bonifica dell'intero ambiente, l'utente viene guidato a proseguire lungo il corridoio; superata una determinata distanza, si attiva un ulteriore movimento programmato (scripted) della telecamera, focalizzato sulla zona della libreria. L'iscrizione sul pavimento "Dispose of their regrets" funge da espediente di environmental storytelling e da indicatore diegetico, suggerendo all'utente la successiva attività da svolgere: raccogliere i diversi detriti e conferirli nell'apposito contenitore situato sul carrello logistico. Coerentemente con il sistema di affordance del titolo, la presenza di un oggetto interattivo viene segnalata sia dal dinamico ingrandimento del mirino (crosshair) a schermo, sia dall'aggiornamento degli elementi dell'interfaccia utente (UI) posizionati nell'angolo inferiore destro.

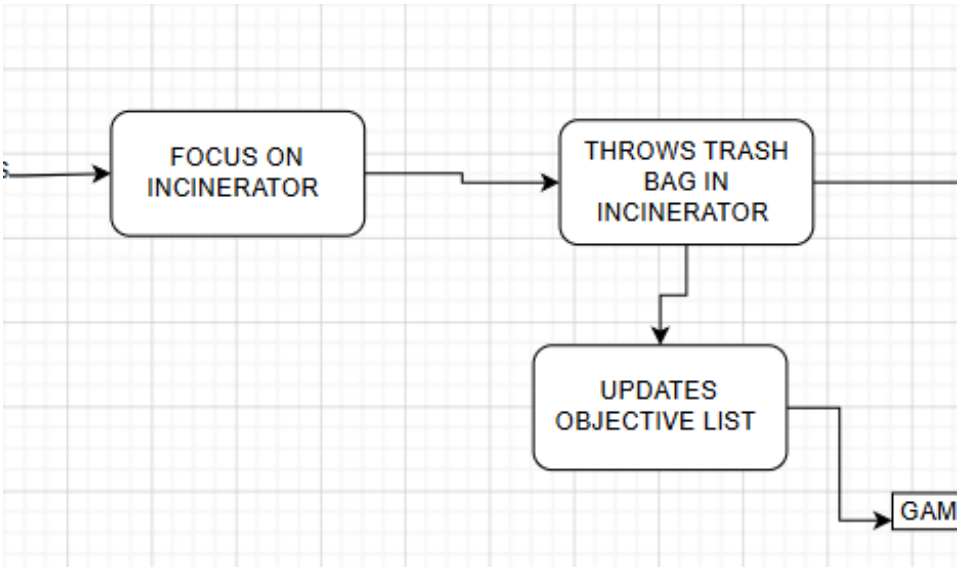


Figura 4.5: Diagramma delle attività per la quarta parte del tutorial

Una volta inserito un determinato quantitativo di rifiuti, il raggiungimento della capacità massima del contenitore viene comunicato all'utente in modo duplice: attraverso una notifica testuale a schermo e tramite il ripristino automatico dell'asset d'interazione nella sua posizione originaria. Questo sistema di feedback espliciti indica chiaramente la saturazione del cestino e induce il giocatore a interagire con esso per prelevare il sacco dei rifiuti accumulati; l'utente viene così direzionato verso l'inceneritore situato al termine del corridoio, al fine di completare il processo di smaltimento definitivo.

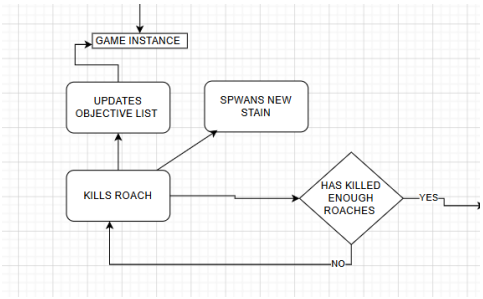


Figura 4.6: Diagramma delle attività per la quarta parte del tutorial

Durante il tragitto di ritorno dall'inceneritore, si attiva un ulteriore movimento programmato (scripted) della telecamera focalizzato su una porta, la quale si apre per mostrare la fuoriuscita di diversi scarafaggi. Il giocatore viene così indotto a esplorare il nuovo ambiente — ormai infestato dagli insetti — e a fare ritorno al carrello logistico al fine di equipaggiare lo schiacciascarafaggi, strumento necessario a eliminare le creature. Ciascun elemento neutralizzato rilascia sulla superficie un

residuo liquido, generando una nuova macchia che richiede un successivo intervento di bonifica tramite il mocio.

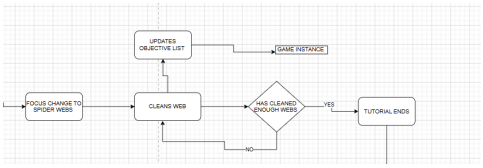


Figura 4.7: Diagramma delle attività per la quinta parte del tutorial

In quest'ultima sezione del livello introduttivo (tutorial), ripercorrendo lo scenario a ritroso, si attiva un ulteriore movimento programmato (scripted) della telecamera, volto a mostrare all'utente un'area caratterizzata dalla presenza di ragnatele. Analogamente a quanto avvenuto in precedenza, il giocatore viene indirizzato verso il carrello logistico per equipaggiare lo spolverino e procedere alla rimozione delle ragnatele. Una volta rimosso un determinato quantitativo di elementi, l'attivazione combinata di un feedback acustico (il suono di porte che si aprono) e di una notifica testuale a schermo segnala esplicitamente all'utente il completamento con successo del tutorial.

4.1.3 Gioco Intermedio

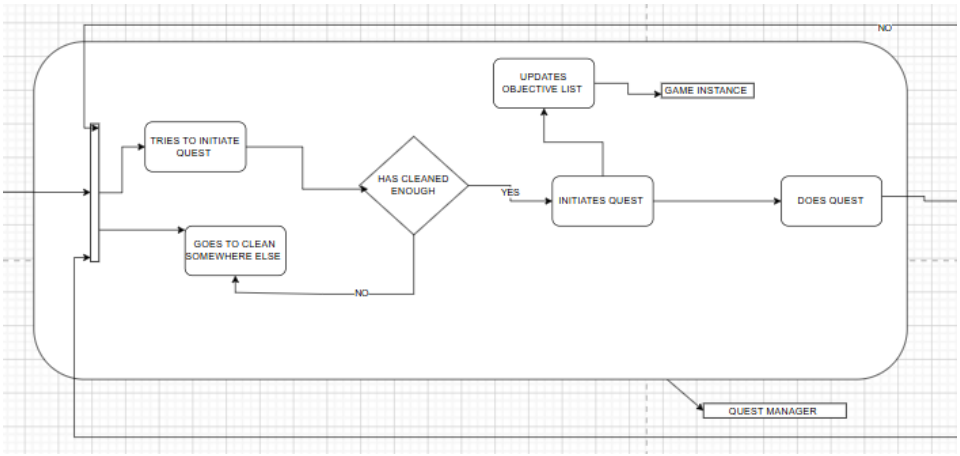


Figura 4.8: Diagramma delle attività per parte intermedia del gioco

Al termine della fase introduttiva (tutorial), il giocatore intraprende l'opera di bonifica estesa, la quale coinvolge sia le aree già esplorate sia i locali fino a quel momento preclusi. In questa fase, l'utente ha inoltre la possibilità di interagire con tre varchi d'accesso speciali, ciascuno dei quali risulta correlato a un ospite peculiare. Il raggiungimento di una determinata soglia metrica, calcolata sul computo complessivo degli elementi igienizzati, funge da prerequisito strutturale per sbloccare l'avvio delle effettive missioni di gioco, implemenate come ricerca e ritrovamento di reperti speciali.

4.1.4 Fine Gioco

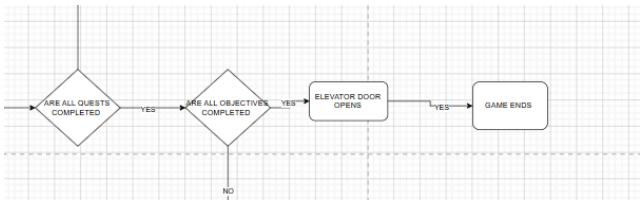


Figura 4.9: Diagramma delle attività per la fase finale del gioco

Esclusivamente a seguito del completamento delle tre missioni associate agli ospiti e della rimozione di una quantità predefinita di residui, un'interfaccia testuale a schermo segnala al giocatore la possibilità di dirigersi verso l'ascensore per concludere l'esperienza di gioco. In alternativa, il sistema consente di prolungare la fase di esplorazione, al fine di conseguire il totale completamento della bonifica ambientale. Una volta effettuato l'accesso all'elevatore, l'interazione con l'apposita pulsantiera attiva il meccanismo di transizione, il quale trasferisce l'avatar all'interno di un'area di gioco inedita. In questo nuovo scenario, dopo aver percorso una breve distanza, si assiste a una dissolvenza visiva (fade to black), seguita dalla visualizzazione di un ironico messaggio finale; quest'ultimo espediente narrativo rende noto all'utente che il protagonista sarà tenuto a perpetuare la propria mansione di addetto alla pulizia nonostante la sopravvenuta ascensione paradisiaca. La sequenza si chiude con la visualizzazione della schermata di riavvio (restart menu), tramite la quale l'utente può scegliere se fare ritorno al menu principale o intraprendere una nuova partita.

4.2 Implementation

In questa sezione verranno approfondite le logiche e i criteri progettuali alla base dell'implementazione delle diverse classi Blueprint che costituiscono l'ossatura del software di gioco, incluse le componenti sviluppate congiuntamente nel corso dell'attività di tirocinio. Per ciascuna classe saranno illustrate la specifica collocazione all'interno dell'architettura generale — supportata dai relativi diagrammi delle classi UML — e le rispettive funzionalità principali.

4.2.1 Giocatore

Player Controller

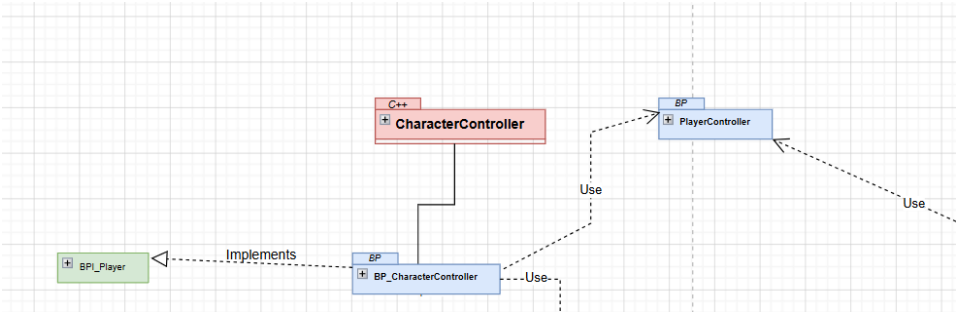


Figura 4.10: UML delle classi per giocatore e controller

Questa classe Blueprint fornisce i metodi fondamentali per l'associazione (binding) dei comandi di input alle classi Pawn deputate al controllo degli avatar di gioco. Tale processo viene strutturato attraverso un'architettura orientata agli eventi (event-driven), che impiega in modo sinergico eventi personalizzati (custom events) ed emettitori di eventi (event dispatchers). Nello specifico, i dispatcher vengono dichiarati all'interno del PlayerController, mentre le classi Pawn eseguono la sottoscrizione (binding) delle proprie funzioni locali a tali emettitori. Di conseguenza, al riscontro di un input fisico da parte dell'utente, il PlayerController provvede a invocare il dispatcher di riferimento, determinando l'esecuzione dinamica e disaccoppiata delle relative logiche di gioco.

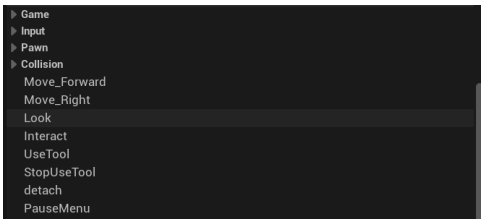


Figura 4.11: Creazione dei dispatcher in PlayerController

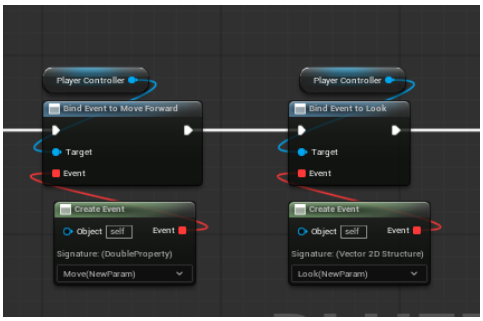


Figura 4.12: Binding in classe target

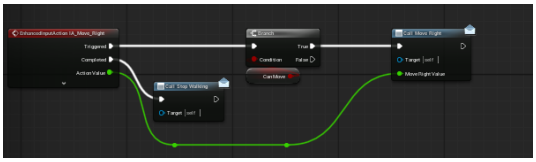


Figura 4.13: Chiamata ai dispatcher in PlayerController

Character Controller

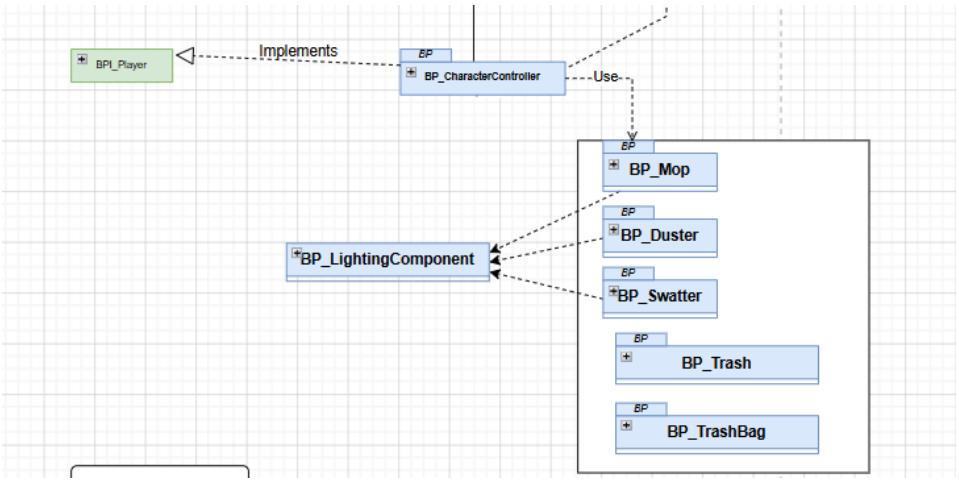


Figura 4.14: UML delle classi specifico per il giocatore

Tale componente si configura come una delle classi cardine per il corretto funzionamento dell'intera architettura software. Essa sovrintende a tutte le logiche relative all'avatar del giocatore e alla sua interazione con gli elementi interattivi dello scenario — quali utensili, porte e il carrello logistico —, occupandosi al contempo dell'elaborazione e della mappatura dei flussi di input e della gestione delle relazioni sistematiche con le classi globali basate sul design pattern Singleton, prima fra tutte la GameInstance. Questa eredita da una classe generica in C++, e implementa l'interfaccia dedicata BPI_Player.

Evento Begin Play Questo metodo viene invocato una sola volta in concomitanza con l'esecuzione del primo fotogramma (frame) di gioco. La sua funzione principale consiste nel gestire la sottoscrizione (binding) delle diverse routine locali ai corrispettivi emettitori di eventi (dispatcher) originati dalle classi PlayerController e GameInstance. Contestualmente, il costrutto sovrintende all'inizializzazione e alla generazione a schermo dei vari elementi dell'interfaccia utente (UI), tra i quali figura il mirino centrale (crosshair).

Tick e Fake Tick Queste due routine vengono eseguite, rispettivamente, a ogni fotogramma (Event Tick) e a intervalli regolari dieci volte al secondo (Custom/Fake Tick). La prima funzione è deputata alla gestione dell'oscillazione verticale della telecamera, configurata per simulare realisticamente il movimento del capo (head bobbing) tipico della visuale in prima persona. Il Fake Tick, invece, è stato introdotto per ridurre la frequenza di operazioni che non richiedono aggiornamento a ogni frame: esso sovrintende all'esecuzione ciclica di una tracciatura sferica (Sphere Trace), finalizzata sia a rilevare gli oggetti interattivi presenti nello scenario, sia ad

aggiornare dinamicamente gli elementi dell'interfaccia utente (UI) relativi ai comandi di input posizionati nell'angolo inferiore destro.



Figura 4.15: Parte del Blueprint CharacterController mostrante le funzioni di Tick

Uso delle Timeline All'interno dell'oscillogramma principale degli eventi (Event Graph principale), diverse logiche si avvalgono dell'impiego di componenti Timeline, i quali non risultano integrabili o eseguibili all'interno delle canoniche funzioni dotate di parametri di input e output. Tali componenti vengono adoperati, ad esempio, nell'architettura che governa lo smaltimento dei rifiuti, dove assolvono il compito di calcolare l'interpolazione lineare (Lerp) della posizione del detrito, guidandone la transizione fluida da un punto di origine A (la mano dell'avatar) a un punto di destinazione B (il contenitore di raccolta).

Player Input Questa articolata funzione governa il ciclo principale di interazione con gli elementi interattivi dello scenario. Il flusso logico prevede, in prima istanza, la verifica di validità dell'oggetto rilevato (validity check); successivamente, viene eseguito il controllo sulle interfacce implementate dalla classe colpita. Qualora l'oggetto implementi la specifica interfaccia BPI_Interact, il sistema attiva una struttura di controllo condizionale sequenziale (branch a cascata) atta a identificare e categorizzare l'entità target in base al valore memorizzato in una variabile Tag di tipo Name.

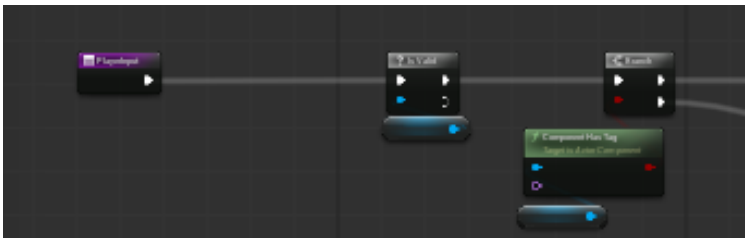


Figura 4.16: Parte del Blueprint CharacterController mostrante la funzione PlayerInput



Figura 4.17: Parte del Blueprint CharacterController mostrante una sezione del controllo a cascata in PlayerInput

ShowScreenInputs Presentando un'architettura analoga a quella deputata alla gestione degli input del giocatore (Player Input), questa routine sovrintende all'aggiornamento dinamico dell'interfaccia utente (UI) dei comandi a schermo. Il processo si articola attraverso una struttura di controllo condizionale sequenziale (a cascata), la quale invoca una specifica funzione esposta dalla GameInstance; quest'ultima, fungendo da intermediario architetturale, provvede a sua volta a sollecitare il Widget Blueprint responsabile della visualizzazione finale degli input a schermo.

Cart



Figura 4.18: Carrello all'inizio del gioco

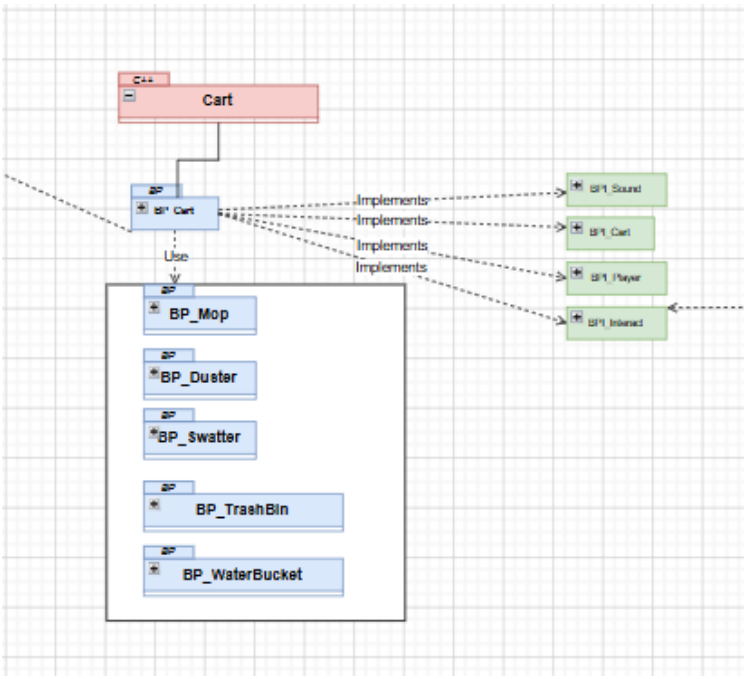


Figura 4.19: UML delle classi per il carrello

Questa componente costituisce l'ulteriore classe Pawn posta sotto il controllo diretto dell'utente. Al raggiungimento di uno specifico punto di interazione (trigger) in prossimità dell'asset, il sistema esegue il rilascio del controllo (unpossess) dell'istanza del Character Controller principale per trasferire il possesso (possess) all'entità carrello. In questa specifica transizione, l'avatar precedentemente utilizzato non viene rimosso dalla memoria (distrutto), bensì traslato al di sotto della superficie calpestabile dello scenario (culling/occlusion manuale), al fine di non inficiare la navigabilità dell'ambiente di gioco. Parallelamente, il carrello logistico funge da hub centrale per l'equipaggiamento e il riposizionamento degli utensili da lavoro, nonché da raccoglitore per lo smaltimento dei detriti accumulati. La classe implementa le interfacce dedicate BPI_Sound, BPI_Player, BPI_Cart e BPI_Interact.

Gestione Input e UI Anche durante la fase di possesso del carrello logistico, il sistema consente l'interazione diretta con molteplici elementi interattivi dello scenario. L'elaborazione e il controllo delle entità intercettate nello spazio di gioco vengono eseguiti mediante una logica architetturale analoga a quella implementata all'interno della classe CharacterController.

Distaccamento dal carrello L'implementazione della logica di rilascio (unposses-s/detach) dal carrello logistico ha presentato notevoli complessità, correlate alla necessità di eseguire controlli condizionali differenziati in base al contesto spaziale in cui viene invocata l'azione. L'algoritmo principale prevede l'esecuzione simultanea di tre tracciature sferiche (Sphere Trace) a partire dal punto di disancoraggio dell'utente,

orientate rispettivamente in direzione posteriore, laterale destra e laterale sinistra. Qualora uno o più vettori intercettino un ostacolo geometrico, il sistema elegge come coordinate di spawn valide per l'avatar del giocatore quelle restituite dal tracciamento che non ha rilevato alcuna collisione. Parallelamente, sono state integrate verifiche geometriche addizionali per prevenire anomalie di posizionamento in prossimità dei varchi d'accesso; tale accorgimento evita che l'utente si materializzi accidentalmente oltre una porta chiusa, dinamica che determinerebbe una condizione di blocco permanente (soft-lock) del ciclo di gioco qualora le stanze risultino ancora precluse nelle fasi antecedenti al completamento del livello introduttivo (tutorial). Questo viene gestito dall'evento onDetach, il quale invoca la funzione TraceToSpawnPlayer.

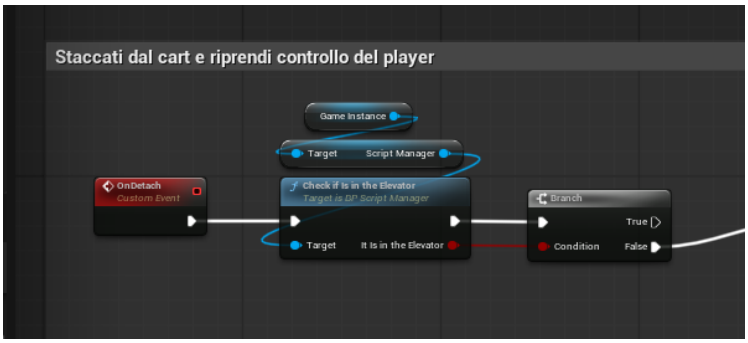


Figura 4.20: Parte del Blueprint Cart mostrante la funzione OnDetach

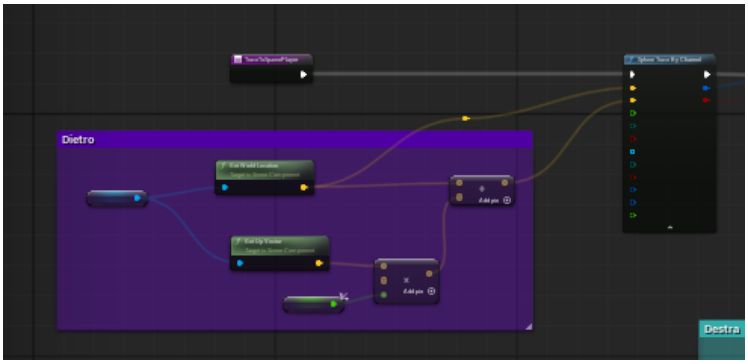


Figura 4.21: Parte del Blueprint Cart mostrante la funzione TraceToSpawnPlayer

Riposizionamento utensili sul carrello Il processo di riposizionamento dei componenti viene governato dalla funzione PutObjectOnCart. Nel momento in cui l'utente interagisce con il carrello logistico senza averne assunto il possesso diretto, il sistema esegue una verifica preliminare sull'entità correntemente equipaggiata dall'avatar. Qualora l'oggetto appartenga alla classe o sottoclasse dei sistemi di lavoro (Tool), viene effettuata un'analisi condizionale sui relativi metadati (Tag); in base all'esito di tale riscontro, l'asset viene rimosso dalla mano del personaggio e vincolato dinamicamente al punto di ancoraggio (socket) specifico predisposto sulla struttura del carrello.

4.2.2 Oggetti Interagibili

Interactable Object

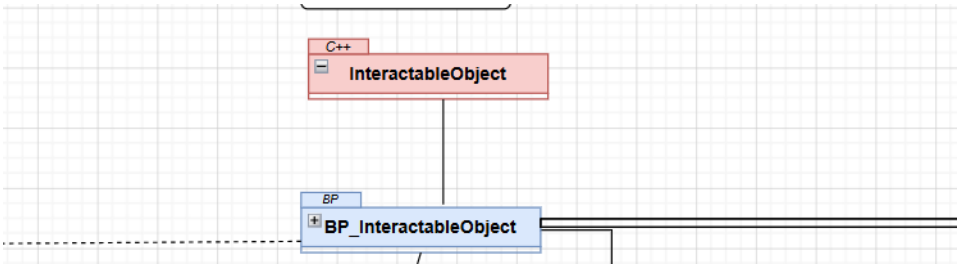


Figura 4.22: UML delle classi per InteractableObject

Questa componente si configura come la classe base astratta dalla quale ereditano tutte le classi deputate alla gestione degli elementi interattivi dello scenario, inclusi gli utensili, le entità igienizzabili, il contenitore dei rifiuti e il secchio dell'acqua. Essa non presenta logiche applicative complesse al proprio interno, limitandosi a esporre un metodo di scrittura (setter) per la proprietà ReceiveDecals integrata nella componente di rendering geometrico (StaticMesh) dell'asset. La classe implementa, inoltre, l'interfaccia strutturale nativa BPI_Interact, garantendo la conformità ai requisiti di interazione del sistema. Molti oggetti di questo ramo fanno anche uso di un componente illuminante (LightingComponent), per evidenziare l'oggetto in determinate sezione del livello introduttivo.

Cleaning Tool

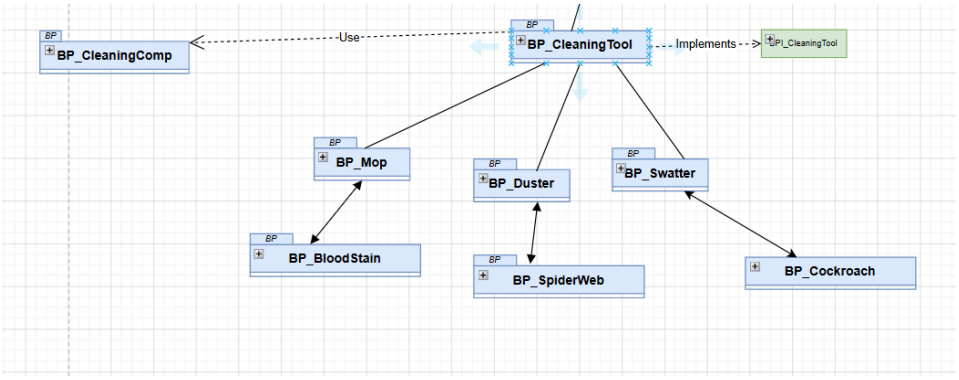


Figura 4.23: UML delle classi per CleaningTool

Tale componente si configura come una classe derivata (figlia) di Interactable_Object e funge da modello strutturale (classe base) per tutti gli strumenti adibiti alle attività di pulizia. Anch'essa non integra logiche applicative complesse, limitandosi a ospitare una routine di inizializzazione per il componente deputato alla rimozione dei residui (CleaningComp) e una variabile di tipo intero (integer) atta a

definire la soglia massima di utilizzi consentiti per lo specifico attrezzo prima della saturazione. Implementa l'interfaccia BPI_CleaningTool.

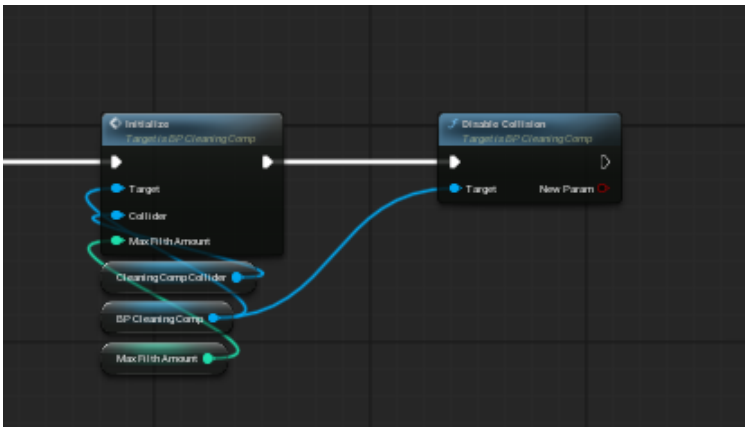


Figura 4.24: Parte del Blueprint di CleaningTool mostrante il BeginPlay

Cleaning Comp

Questa componente eredita dalla classe nativa ActorComponent e, in quanto tale, non presenta sotto-componenti geometriche proprie, come le StaticMesh. Il suo ruolo strutturale all'interno dell'architettura consiste nel cooperare in stretta sinergia con la classe Cleaning_Tool: essa memorizza i riferimenti alle mesh e alle variabili relative allo stato di imbrattamento, utilizzandoli per governare l'intera logica di rimozione dei residui. La scelta di implementare un ActorComponent risponde alla volontà di adottare un paradigma di sviluppo modulare e scalabile, fortemente orientato alla composizione (Component-Based Design). Questa impostazione architetturale ottimizza la manutenibilità del software, semplificando sensibilmente l'eventuale integrazione di nuove meccaniche di pulizia o la modifica di quelle preesistenti, senza dover alterare la struttura profonda degli attori principali.

Gestione sporco del mocio Attraversando i metodi CleanMop e AddFilthToTool, il sistema governa il comportamento del mocio sia durante l'interazione con l'entità secchio, sia al contatto con un residuo liquido. Nello specifico, la prima routine provvede alla bonifica dello strumento qualora si verifichi un'interazione con il secchio dell'acqua; al contrario, la seconda gestisce la saturazione del mocio a seguito della collisione con una macchia presente sullo scenario. Nell'eventualità in cui l'utente tenti di igienizzare una superficie imbrattata utilizzando un attrezzo già saturo, il metodo AddFilthToTool invoca la funzione MakeStainBigger esposta dalla macchia colpita; tale espediente logico determina un incremento dell'area del residuo, aumentandone lo stato di contaminazione e, di conseguenza, la complessità delle successive operazioni di pulizia.

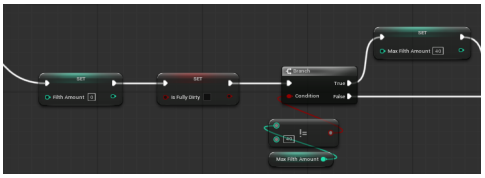


Figura 4.25: Parte del Blueprint CleaningComp mostrante la funzione CleanMop

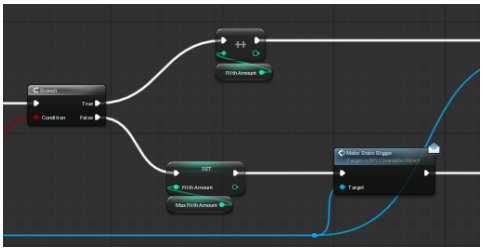


Figura 4.26: Parte del Blueprint CleaningComp mostrante la funzione AddFilthToTool

Mop



Figura 4.27: Mocio

Questa componente, derivata dalla classe Cleaning_Tool, rappresenta l'entità mocio, strumento adibito alla rimozione dei residui liquidi rinvenuti all'interno dei locali e delle aree comuni dello scenario. Coerentemente con la struttura gerarchica del software, essa integra tra i propri moduli ereditati il CleaningComp, preposto al monitoraggio del livello di saturazione dell'attrezzo e alla validazione della sua idoneità operativa. Nelle fasi iniziali, al fine di introdurre l'utente alla meccanica di bonifica del mocio, la soglia di tolleranza dello strumento è limitata a dieci interazioni consecutive prima della completa saturazione; tale valore metrico viene successivamente esteso a quaranta utilizzi a seguito del completamento della relativa sezione introduttiva (tutorial), traducendosi in una progressione diretta dell'efficacia dell'equipaggiamento.

Logica di pulizia La logica di rimozione dei residui associata al mocio, in analogia con gli altri strumenti di gioco, viene governata mediante il rilevamento delle collisioni volumetriche (OnComponentBeginOverlap), l'analisi dei metadati (Tag) e la verifica di un tipo enumerativo (Enum). Qualora il mocio entri in collisione con un asset contrassegnato dal medesimo valore enumerativo dello strumento (Mop), il sistema invoca la routine di pulizia, subordinandone l'esecuzione allo stato di idoneità operativa

dell'attrezzo. È stata inoltre implementata una logica suppletiva atta a generare una nuova macchia qualora il mocio saturo entri in contatto con le pareti o la superficie calpestabile; tuttavia, tale funzionalità non risulta attiva nell'ambiente di produzione, poiché l'abilitazione dello strumento è vincolata esclusivamente all'intercettazione di un residuo preesistente tramite lo Sphere Trace del giocatore. Di conseguenza, quando l'utente equipaggia il mocio dal carrello logistico e preme l'apposito comando di input in prossimità di una macchia, il sistema attiva la funzione di interfaccia Interact; quest'ultima avvia l'animazione dell'attrezzo mediante una componente Timeline e provvede alla contestuale abilitazione del volume di collisione (collider) preposto alla bonifica.

Movimento dell'attrezzo Il sistema di movimentazione spaziale dello strumento e il relativo vincolo di ancoraggio alla mano dell'avatar vengono arbitrati mediante due componenti direzionali nativi (ArrowComponent). Il primo, contraddistinto dal codice cromatico rosso e denominato AttachToPlayerArrow, definisce i parametri di posizione e orientamento (Transform) necessari per il corretto aggancio dell'asset all'arto del personaggio. Il secondo, identificato dal colore blu e denominato CleaningArrow, stabilisce invece i vettori di traiettoria e spostamento che lo strumento deve seguire durante l'esecuzione dell'animazione di pulizia.

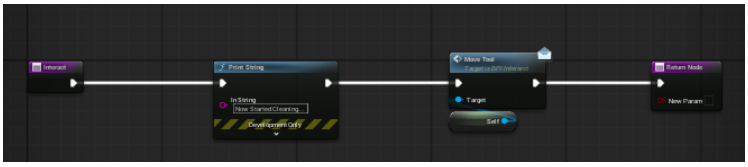


Figura 4.28: Funzione di interfaccia chiamata dal Giocatore

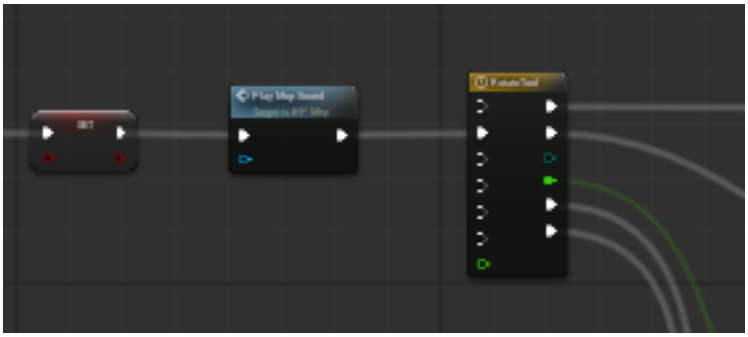


Figura 4.29: Movimento tramite Timeline

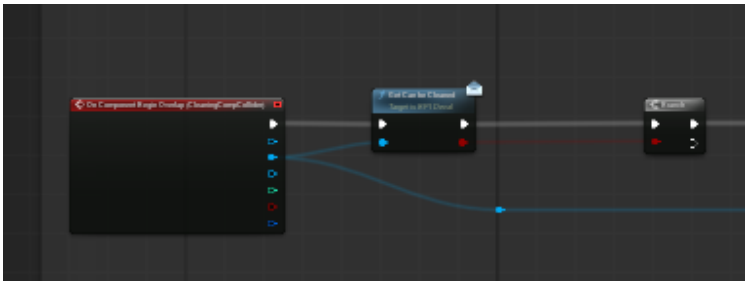


Figura 4.30: Rilevamento delle collisioni

Duster e Swatter



Figura 4.31: Schiacciascarafaggi



Figura 4.32: Spolverino

Tali classi rappresentano, rispettivamente, gli strumenti dedicati alla rimozione delle ragnatele (Duster) e all'eliminazione delle infestazioni da blatte (Swatter). Sotto il profilo architetturale, l'esecuzione delle logiche di bonifica ricalca fedelmente il modello implementato per l'entità mocio (Mop), discriminando le diverse categorie di residui interattivi attraverso il controllo combinato di tipi enumerativi (Enum) e metadati (Tag). Tuttavia, a differenza di quest'ultimo, tali attrezzature presentano una caratteristica di asaturabilità permanente durante il ciclo di utilizzo; di conseguenza, non necessitano dell'integrazione di routine dedicate alla pulizia o al ripristino dello strumento.

Water Bucket



Figura 4.33: Secchio d'acqua

Questa classe rappresenta l'entità secchio d'acqua, infrastruttura logica deputata alla bonifica del mocio a seguito della sua saturazione. Il flusso algoritmico che governa il processo di ripristino prevede, in prima istanza, il disancoraggio temporaneo (detachment) dello strumento dall'avatar del giocatore, preservando tuttavia il puntatore al riferimento in memoria dell'asset. Successivamente, l'attrezzo viene vincolato dinamicamente (attachment) a una componente specifica predisposta sulla mesh del secchio. Questa transizione logico-spaziale consente l'attivazione di una componente Timeline, la quale riproduce un'oscillazione sull'asse verticale volto a simulare l'immersione; l'esecuzione di tale routine determina l'azzeramento delle variabili di imbrattamento e il conseguente ripristino dell'idoneità operativa dello strumento. Al termine del ciclo, il sistema esegue un nuovo riparentamento, riassegnando il possesso e l'ancoraggio dell'asset alla mano del personaggio.

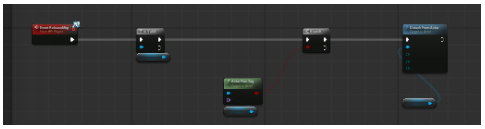


Figura 4.34: Funzione di distaccamento nel Giocatore

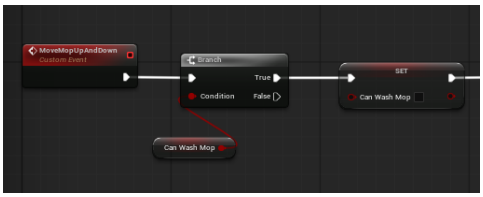


Figura 4.35: Movimento verticale del mocio

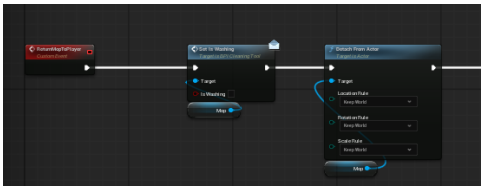


Figura 4.36: Funzione nel secchio che ridà il mocio al Giocatore

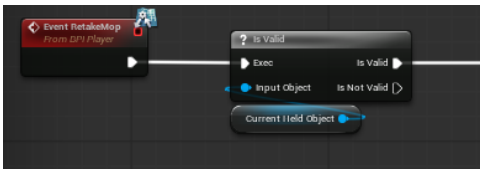


Figura 4.37: Funzione nel Giocatore che riprende possesso del mocio

Cleanable Object

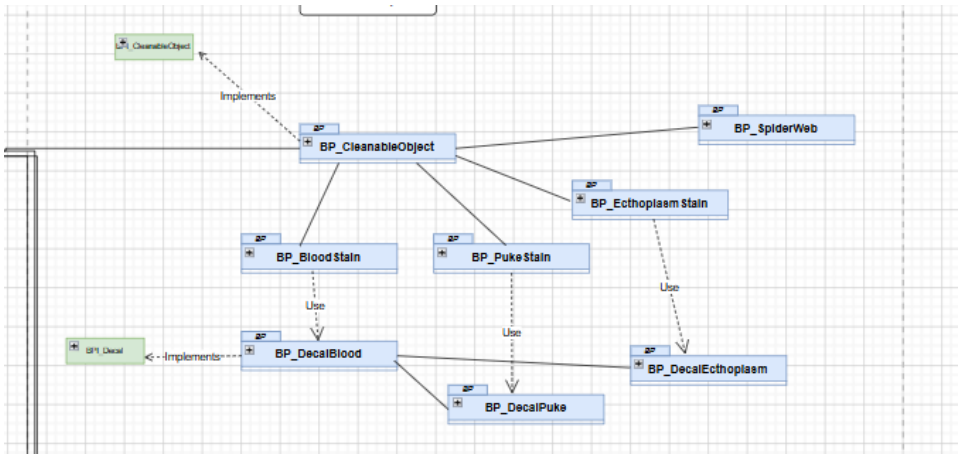


Figura 4.38: UML delle classi per CleanableObject

Questa classe si configura come l'entità base (classe genitore) dalla quale derivano, per ereditarietà, le differenti categorie di imbrattamento dislocate all'interno dello scenario di gioco, quali le macchie superficiali, i residui liquidi e le ragnatele. Al fine di uniformare e standardizzare i flussi di igienizzazione dei vari attori, la classe implementa l'interfaccia strutturale dedicata BPI_CleanableObject. Essa integra, inoltre, un componente attore figlio (ChildActorComponent) denominato Stain; all'interno delle classi derivate (figlie), tale modulo viene specializzato mediante l'istanziamento di specifici attori decalcomania (DecalActor), deputati alla gestione dinamica della visibilità, della morfologia e della resa estetica della macchia stessa.

Begin Play All'interno di questa routine viene gestito l'incremento della variabile contatore memorizzata nella GameInstance, deputata al tracciamento quantitativo dei residui da rimuovere, e viene contestualmente eseguita la sottoscrizione (binding) delle funzioni locali ai corrispettivi emettitori di eventi (event dispatcher). Nella fattispecie, è stata implementata una logica di filtraggio dinamico delle collisioni: nel momento in cui l'avatar equipaggia lo strumento mocio, il sistema provvede a disattivare i canali di collisione relativi ai rifiuti solidi; viceversa, qualora l'utente stringa un utensile differente, vengono disabilitate le collisioni associate alle macchie liquide.

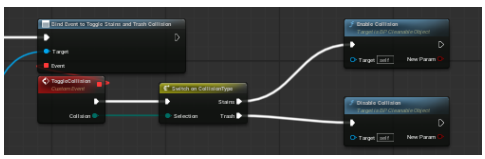


Figura 4.39: Parte del Blueprint CleanableObject mostrante il BeginPlay

Logica di distruzione All'atto della distruzione del residuo, il sistema invoca la specifica funzione esposta dalla GameInstance al fine di aggiornare il contatore globale degli elementi igienizzati; contestualmente, viene eseguita la deallocazione (destruction) del componente attore figlio (ChildActorComponent) responsabile della resa grafica della decalcomania (Decal).

Blood Stain



Figura 4.40: Primo residuo ematico da pulire



Figura 4.41: Stanza colma di residui ematici

Questa componente rappresenta una delle molteplici classi derivate (figlie) di Cleanable_Object; nello specifico, l'attore implementa uno dei cinque modelli tipologici di residuo ematico (sangue) previsti all'interno dell'ambiente di gioco. Le logiche applicative ed esecutive esplicate per questa singola classe sono da considerarsi strutturalmente analoghe e direttamente estensibili a tutte le altre varianti omologhe non esplicitamente menzionate nella trattazione.

Begin Play Oltre a invocare il metodo BeginPlay della classe genitore (padre), all'interno di questa routine viene eseguita l'assegnazione (set) del vettore (array) di decalcomanie nel componente figlio (ChildComponent). Tale accorgimento è finalizzato alla gestione modulare e scalabile di molteplici elementi grafici (Decal) associati a una singola entità logica. Contestualmente, il costrutto sovrintende alla memorizzazione del riferimento locale alla GameInstance e all'iscrizione (binding) della classe ai corrispettivi emettitori di eventi (event dispatcher).

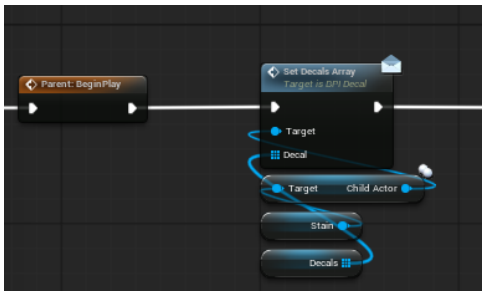


Figura 4.42: Parte del Blueprint BloodStain mostrante la funzione BeginPlay

Aggravamento contaminazione Attraverso l'esecuzione del metodo MakeStainBigger, viene definita la logica comportamentale secondo cui l'interazione con un residuo mediante lo strumento mocio già saturo determina una penalizzazione per l'utente, incrementando la complessità delle successive operazioni di igienizzazione; contestualmente, il sistema interviene sulle proprietà del materiale dell'asset, accentuando il coefficiente di opacità (opacity) del Decal per fornire un riscontro visivo immediato del maggiore stato di contaminazione.

Molteplici Decal su una singola macchia All'interno del costruttore della classe è stata implementata una logica architetturale atta ad associare molteplici decalcomanie (Decal) a una singola istanza di oggetto presente nello scenario, con la finalità di garantire la diversificazione morfologica ed estetica dei vari residui. Direttamente dall'editor della scena, l'utente ha la facoltà di mappare fino a nove variabili di posizionamento spaziale (Transform) all'interno di appositi vettori (array); il costruttore provvede successivamente a iterare tali strutture dati, istanziando per ciascun elemento un Decal differente. Questo approccio procedurale conferisce piena libertà di orientamento e scalabilità geometrica a ogni singolo componente grafico aggiunto.

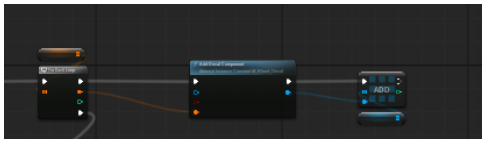


Figura 4.43: Parte del Blueprint BloodStain mostrante il costruttore

Decal Blood

Questa classe opera in stretta sinergia con l'entità BloodStain, dalla quale viene invocata ogniqualvolta si renda necessario modificare dinamicamente il parametro di opacità (opacity) delle decalcomanie (Decal) associate all'asset scenico.

Modifica opacità Attraverso le funzioni evento CleanStain e MakeStainBigger, invocate dall'istanza di classe BloodStain a cui il componente figlio (ChildComponent) risulta vincolato, il sistema manipola dinamicamente i coefficienti di opacità di tutte le decalcomanie memorizzate all'interno del vettore (array) dedicato, costantemente aggiornato dalla classe genitore. Qualora, a seguito di progressive operazioni di igienizzazione, il valore di tale parametro diventi minore o uguale a zero, l'asset viene rimosso dal ciclo di esecuzione e definitivamente deallocato (distrutto).

Trash



Figura 4.44: Singolo residuo compostabile

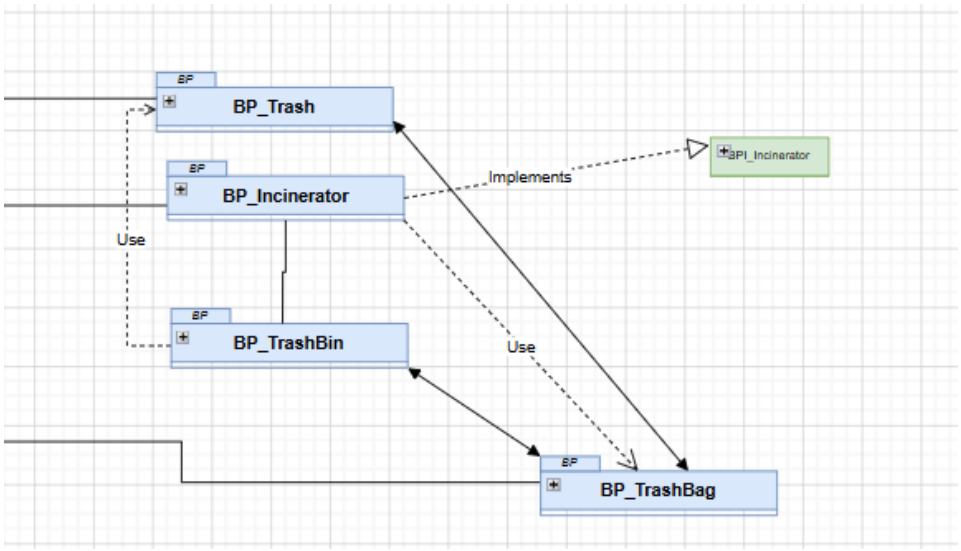


Figura 4.45: UML delle classi per Immondizia, cestino e inceneritore

Questa componente si configura come la classe base astratta (classe genitore) dalla quale derivano, per ereditarietà, le molteplici entità logiche deputate alla gestione e alla rappresentazione dei rifiuti di natura organica e compostabile all'interno dell'ambiente di gioco.

Begin Play All'interno di questa routine viene eseguita l'inizializzazione (set) delle variabili di stato fondamentali, quali le coordinate di posizionamento iniziale dell'asset, e viene contestualmente gestito l'incremento del parametro contatore memorizzato nella GameInstance, deputato al tracciamento dei rifiuti complessivi presenti nello scenario. Infine, il costrutto sovrintende alla sottoscrizione (binding) delle funzioni locali ai corrispettivi emettitori di eventi (event dispatcher).

Illuminazione Attraverso le funzioni evento EventGlow e StopGlowing, il sistema governa la logica di illuminazione dinamica dell'asset, invocando i metodi esposti dal componente di illuminazione (LightingComponent) integrato nell'oggetto. Tale meccanismo viene attivato in corrispondenza di specifiche fasi del ciclo di gioco, con la finalità di modulare la luminescenza dell'attore e fornire un riscontro visivo (visual feedback) orientativo per l'utente.

TrashBag

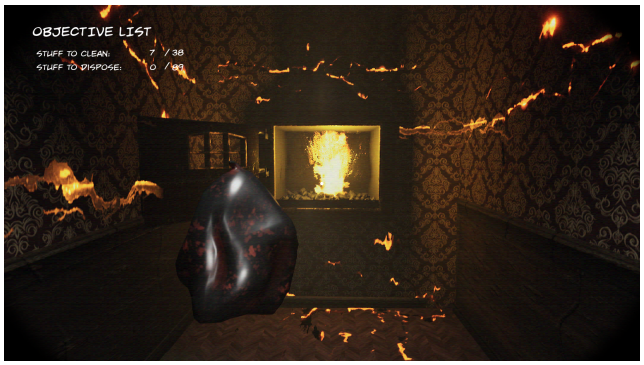


Figura 4.46: Sacco di rifiuti & inceneritore

Questa classe rappresenta l'entità sacco dei rifiuti, asset che viene equipaggiato direttamente dall'avatar del giocatore a seguito dell'interazione con il contenitore principale (TrashBin) situato sul carrello logistico. L'attore integra una proprietà denominata ContainedTrash, deputata a registrare quantitativamente il volume di rifiuti accumulato nel cestino fino a quel momento. Qualora la verifica sullo stato del contenitore dia esito positivo e l'avatar disponga dello slot della mano libero, il sistema estrae il valore del suddetto parametro dall'istanza del cestino e lo trasferisce al costruttore del sacco in fase di istanziamento; contestualmente, viene eseguito l'azzeramento della variabile d'origine all'interno del cestino per consentire un nuovo ciclo di raccolta.



Figura 4.47: Parte del Blueprint CharacterController, mostrante la logica di generazione della busta di immondizia nella funzione PlayerInput

TrashBin

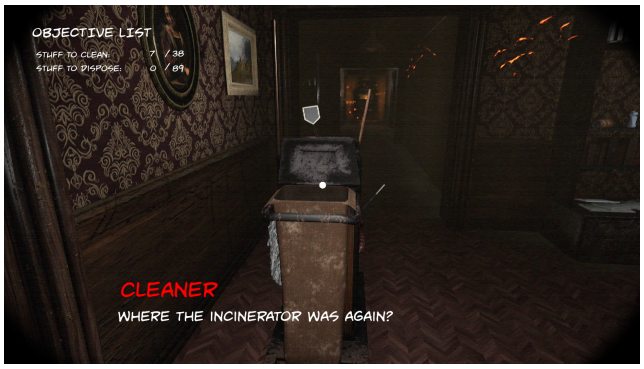


Figura 4.48: Contenitore per i rifiuti sul carrello

Questa classe rappresenta uno dei molteplici componenti attore figlio (ChildActor-Component) integrati nella struttura del carrello logistico. Il suo ruolo architetturale consiste nel monitorare quantitativamente i rifiuti depositati dall'utente e nel gestire il rilascio del sacco dei rifiuti nel momento in cui l'avatar avvia un'operazione di interazione con il modulo stesso.

Begin Play All'interno di questa routine viene eseguita la memorizzazione del riferimento (reference) locale alla GameInstance e viene contestualmente gestita la sottoscrizione (binding) delle funzioni ai rispettivi emettitori di eventi (event dispatcher).

Disfacimento immondizia All'atto della collisione tra un attore provvisto del metadato Tag "Trash" e uno specifico volume di innesco (Trigger Collider), il sistema intercetta l'evento di sovrapposizione (overlap) o contatto. L'attivazione di tale routine determina l'immediata rimozione dell'oggetto dal ciclo di esecuzione e la sua conseguente deallocazione (distruzione) dalla memoria di gioco, tramite la chiamata della funzione di interfaccia DestoryTrash. All'interno di questa routine, il sistema esegue in prima istanza la verifica del metadato Tag associato all'attore interagente. Una volta validato l'identificativo, viene operato l'incremento della variabile intera locale ContainedTrash, deputata al conteggio interno dei residui accumulati. Successivamente, si procede all'immediata deallocazione (distruzione) dell'oggetto in esame e alla contestuale esecuzione del modulo audio preposto alla riproduzione degli effetti sonori di feedback.

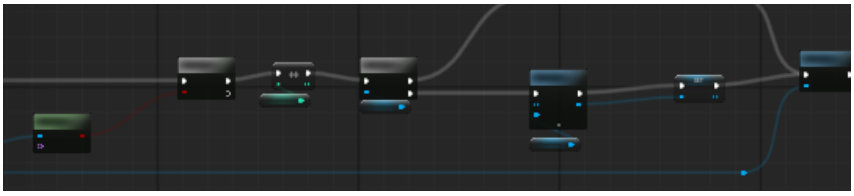


Figura 4.49: Parte del Blueprint TrashBin mostrante la funzione DestroyTrash

Incinerator

Questa classe, derivata (figlia) dall'entità TrashBin, sovrintende — in analogia con le logiche della classe genitore — alla deallocazione (distruzione) del sacco dei rifiuti a seguito dell'interazione diretta da parte dell'avatar, provvedendo al contestuale aggiornamento del parametro contatore memorizzato nella GameInstance. La classe implementa l'interfaccia dedicata BPI_Incinerator.

Disfacimento busta All'interno della funzione di interfaccia DestroyTrash, il sistema governa il processo di rimozione del sacco dei rifiuti a seguito della rilevazione di un evento di collisione con il relativo volume di innesco (collider). In questa routine viene inoltre implementato, mediante l'ausilio di una componente Timeline, un effetto di transizione visiva in dissolvenza; tale comportamento agisce direttamente su un'istanza di materiale dinamico (DynamicMaterialInstance) applicata alla superficie del sacco, manipolandone progressivamente i parametri scalari di opacità o scomposizione.

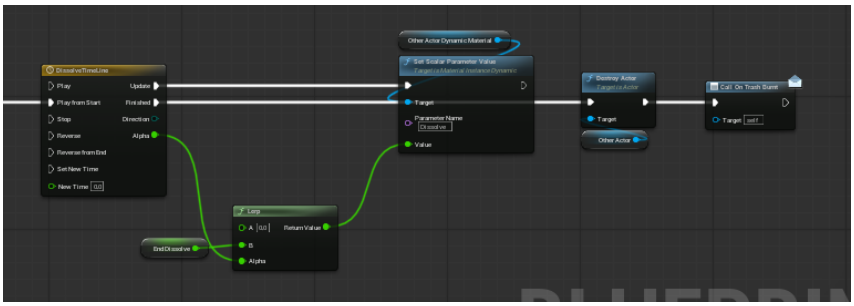


Figura 4.50: Parte del Blueprint Incinerator mostrante la funzione DestroyTrash e parte della logica di dissolvenza

Logica sportello Attraverso l'esecuzione delle funzioni dedicate OpenIncinerator e CloseIncinerator, il sistema governa i processi di apertura e chiusura dello sportello dell'inceneritore, attivandoli in funzione della prossimità spaziale dell'avatar del giocatore. Tale transizione cinematica viene realizzata mediante l'ausilio di una componente Timeline, la quale pilota un'interpolazione lineare (Lerp) tra due valori di rotazione (Rotator); le coordinate angolari, elaborate come variabili in virgola mobile (float), consentono di ottenere un movimento fluido e cinematicamente coerente delle componenti geometriche dello sportello.

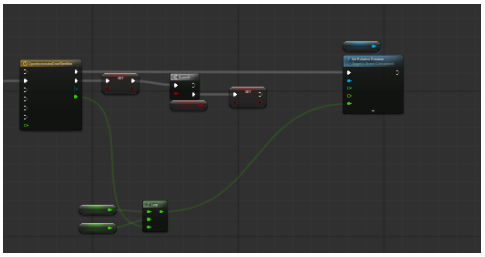


Figura 4.51: Parte della funzione OpenIncinerator

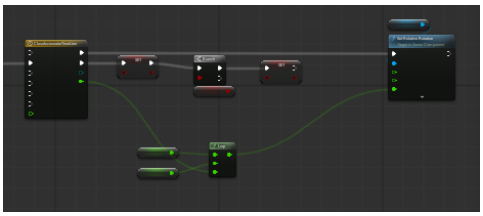


Figura 4.52: Parte della funzione CloseIncinerator

Cockroach

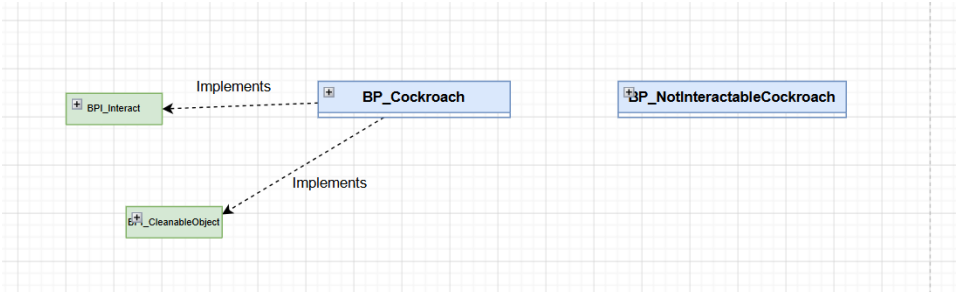


Figura 4.53: UML delle classi per gli scarafaggi

Questa classe rappresenta l'entità logica associata alle blatte (scarafaggi), agenti dinamici dislocati all'interno dell'ambiente di gioco che l'avatar è deputato a eliminare mediante l'utilizzo dello specifico strumento di disinfestazione (Swatter). La classe implementa le interfacce BPI_Interact e BPICleanableObject.

Logica di movimento L'agente esegue la propria routine di navigazione nello scenario attraverso semplici operazioni di traslazione e rotazione bidimensionale; in virtù di tale linearità cinematica, la gestione del movimento non ha richiesto l'implementazione di un'architettura decisionale complessa o di una macchina a stati basata su alberi di comportamento (Behavior Tree). Mediante l'esecuzione della funzione ChooseANewLocation, l'entità calcola una coordinata target casuale entro un raggio predefinito, orienta il proprio vettore di orientamento (Forward Vector) verso tale destinazione e ne pianifica lo spostamento attraverso una componente Timeline. Qualora l'avatar del giocatore entri all'interno di un determinato raggio di rilevamento (portata), il sistema incrementa considerevolmente la frequenza esecutiva e la velocità di tali routine, simulando un comportamento di fuga o allerta.

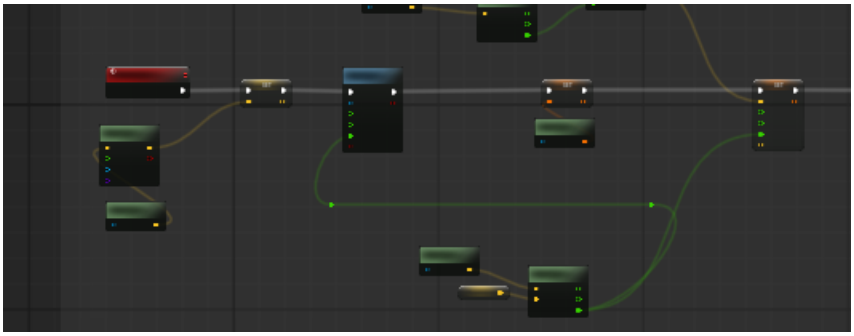


Figura 4.54: Parte del Blueprint Cockroach mostrante parte della logica di movimento

NotInteractableCockroach

Questa classe di tipo minimale modella le blatte (scarafaggi) prive di logica interattiva, il cui istanziamento ed esecuzione sono vincolati esclusivamente alla sequenza introduttiva (tutorial) dedicata all'apprendimento dello strumento di disinfestazione (Swatter). All'avvio di tale fase, gli agenti fuoriescono dalla soglia dell'ambiente di gioco. Nello specifico, attraverso l'attivazione della funzione CockroachGoOutOfTheRoom, il sistema seleziona in modo pseudo-casuale una coordinata di destinazione all'interno di un set predefinito di vettori di arrivo; successivamente, mediante l'ausilio di una componente Timeline, viene eseguita un'interpolazione lineare cinematica per traslare l'attore dal punto di generazione (spawn) alla posizione target calcolata.

4.2.3 Porte e Ascensore

LeftNeutralRoomDoor



Figura 4.55: Una delle porte neutrali

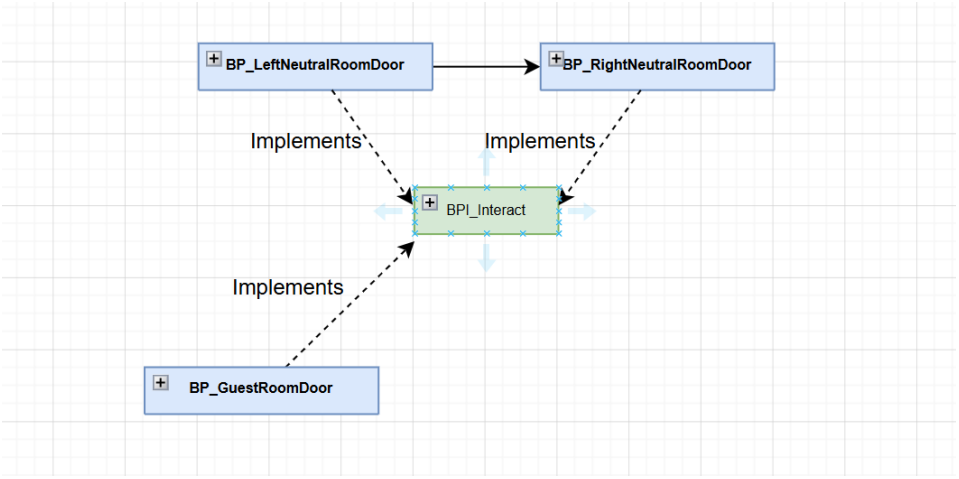


Figura 4.56: UML delle classi per le porte

Questa classe modella l'entità logico-geometrica dello sportello sinistro della porta neutrale, il cui meccanismo di apertura è subordinato al completamento della sequenza introduttiva (tutorial) o vincolato all'attivazione di specifiche fasi di quest'ultima. Al fine di completare l'infrastruttura dell'infisso, è stata implementata una classe speculare per la componente destra, la quale adotta il medesimo framework algoritmico e le stesse logiche comportamentali della controparte sinistra. Questa come le altre, implementa l'interfaccia BPI_Interact.

Logica di apertura e chiusura Lo stato dell'infisso e le relative animazioni cinematiche di transizione vengono governati dalle funzioni OpenCloseDoorFromInsideRoom e OpenCloseDoorFromOutsideRoom. L'attivazione dell'uno o dell'altro metodo è determinata in modo dinamico dalla posizione spaziale occupata dall'avatar all'atto dell'interazione; la separazione formale in due routine distinte risponde alla necessità di game design di prevenire la compenetrazione geometrica della porta con la mesh del personaggio.

Sotto il profilo algoritmico, se l'utente avvia l'interazione dall'interno dell'ambiente, il tracciamento sferico (Sphere Trace) originato dal personaggio interseca un volume di collisione provvisto del metadato Tag "RoomDoorFromInside", decretando l'esecuzione della prima funzione; viceversa, qualora l'interazione avvenga dall'esterno, il flusso intercetta la mesh statica identificata dal Tag "RoomDoorFromOutside", invocando la seconda procedura.

- **Verifica dello stato di blocco:** In prima istanza, il sistema interroga la variabile di stato della porta. Qualora l'infisso risulti serrato (*locked*), viene avviata una componente *Timeline* deputata a riprodurre un'animazione oscillatoria ad alta frequenza (*tremolio*), finalizzata a fornire un *feedback* visivo di impedimento.

- **Controllo del ciclo di apertura/chiusura:** Se la porta risulta sbloccata, il sistema valuta la variabile booleana di apertura.

- ◇ **Stato Chiuso:** Il sistema esegue la routine di apertura, calcolando il vettore di rotazione in direzione opposta rispetto alla posizione dell'avatar.
- ◇ **Stato Aperto:** Prima di procedere alla chiusura, l'algoritmo esegue una verifica di prossimità spaziale sul personaggio. Se l'esito rileva la presenza dell'avatar all'interno del raggio di rotazione del battente (evitando così una collisione forzata), l'azione viene temporaneamente inibita; in caso contrario, viene eseguita l'animazione di chiusura.

Infine, la classe implementa il metodo dedicato DoorSlightlyOpenForTutorial, concepito per forzare un'apertura parziale controllata (pari al 50% dell'ampiezza massima) in corrispondenza di specifiche tappe della sequenza introduttiva.

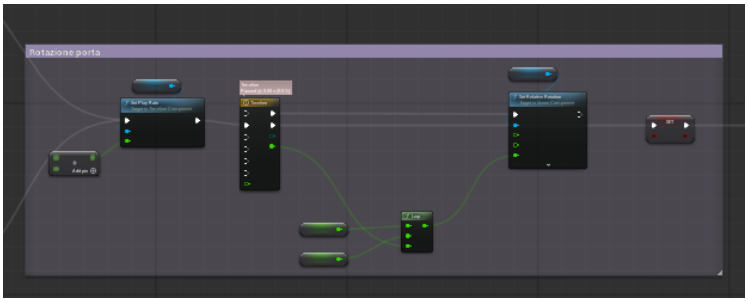


Figura 4.57: Parte del Blueprint LeftNeutralRoomDoor mostrante parte della logica di apertura chiusura

GuestRoomDoor

Questa componente modella l'infrastruttura logico-geometrica degli infissi d'accesso alle stanze private destinate ai diversi personaggi non giocanti (*NPC*), identificati all'interno della narrazione come ospiti della struttura. Attraverso tali elementi, l'avatar ha la facoltà di attivare le routine di interazione con i rispettivi occupanti nel corso della sessione di gioco.

Logica di apertura e chiusura Il framework algoritmico che governa la cinematica di movimento degli infissi speciali ricalca fedelmente la logica implementata per le componenti standard. L'unica divergenza strutturale risiede nell'introduzione di un nodo di controllo condizionale preliminare: antecedentemente alla verifica dello stato di blocco della porta, il sistema interroga il sottosistema di gestione delle missioni (Quest System) per valutare lo stato di avanzamento della linea narrativa associata a quello specifico varco.

Integrazione nel sistema di quest Attraverso l'implementazione di metodi dedicati ed esclusivi, la classe si integra nativamente all'interno del sistema di gestione delle missioni (Quest System).

Nella fase iniziale del ciclo di vita dell'attore, la mesh statica dell'infisso presenta il metadato Tag "GuestRoomDoor"; tale identificativo indirizza il flusso d'esecuzione verso la routine di attivazione dei dialoghi con l'ospite, inibendo temporaneamente il convenzionale algoritmo di apertura e chiusura. Ogni istanza di porta speciale è associata a una variabile intera denominata CurrentCheckpoint, deputata a tracciare lo stato di avanzamento lineare della missione, e a un set di quattro strutture dati di tipo Data Table, le quali memorizzano le stringhe di testo destinate a essere visualizzate su schermo all'invocazione dello specifico metodo esposto dalla GameInstance.

All'avvio della missione, il modulo centrale QuestManager invoca il metodo SpawnQuestObject esposto dalla porta. Questa funzione esegue un'iterazione globale sugli attori presenti nello scenario per identificare le coordinate spaziali esatte in cui istanziare l'oggetto richiesto dalla sotto-quest. Infine, una volta soddisfatti i requisiti della missione, viene eseguito il metodo ModifyTag: quest'ultimo sovrascrive il Tag "GuestRoomDoor" con la stringa "RoomDoorFromOutside", convertendo dinamicamente l'infisso speciale in un componente standard e ripristinandone le normali funzionalità di accesso.

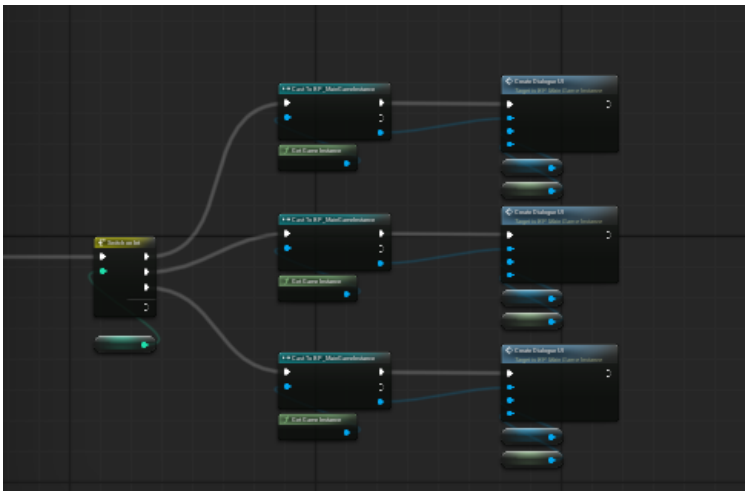


Figura 4.58: Parte del Blueprint GuestRoomDoor mostrante parte della funzione StartDialogue

ElevatorDoor_SX



Figura 4.59: Ascensore aperto

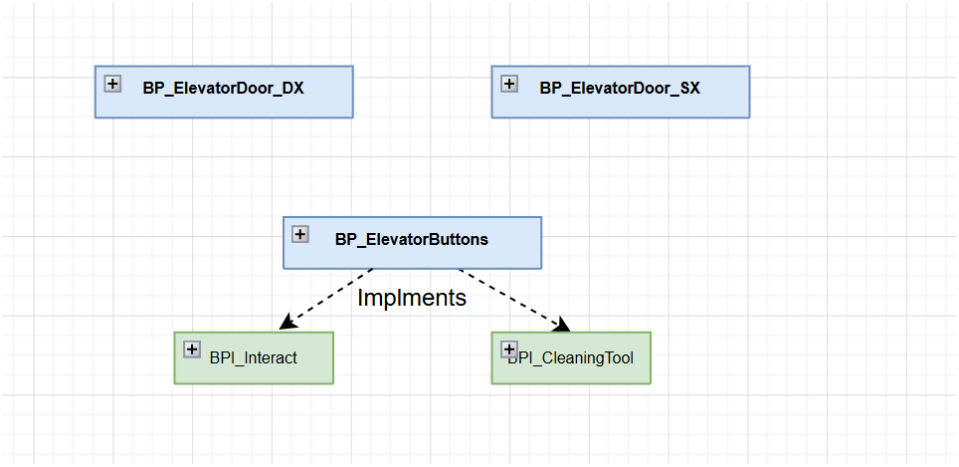


Figura 4.60: UML delle classi per l'ascensore

Questa classe modella il battente sinistro dello sportello dell'ascensore, infrastruttura logico-geometrica che delimita i punti di inizio e di conclusione della sessione di gioco. In analogia con l'architettura degli infissi precedentemente descritti, il sistema integra una componente speculare per il lato destro, la quale condivide la medesima base algoritmica. I processi di apertura e chiusura dell'elevatore vengono invocati in modo univoco dal modulo centrale ScriptManager e si traducono in operazioni di traslazione lineare lungo l'asse orizzontale delle due componenti geometriche.

ElevatorButtons

Questa classe modella la console dei pulsanti integrata all'interno dell'elevatore, i cui moduli risultano interattivi esclusivamente nelle fasi conclusive della sessione di gioco. Quando l'avatar si appresta a completare il livello, i componenti grafici vengono retroilluminati mediante l'attivazione del componente di illuminazione (LightingComponent), segnalando la disponibilità all'interazione. Una volta sollecitato dall'utente, il costrutto invoca la routine di chiusura dei battenti dell'ascensore e

inizializza la logica di transizione procedurale allo scenario successivo; quest'ultima fase sovrintende al caricamento asincrono delle nuove texture sulle superfici verticali dell'ambiente e alla contestuale deallocazione dei residui liquidi e degli agenti dinamici rimasti in scena. Questa implementa le interfacce BPI_Interact e BPI_CleaningTool.

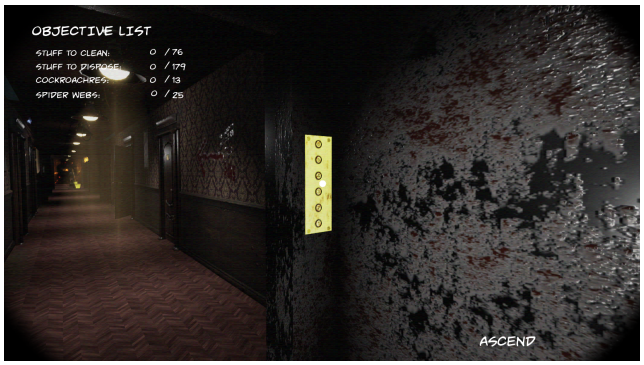


Figura 4.61: Pulsanti nell'ascensore

4.2.4 Luci

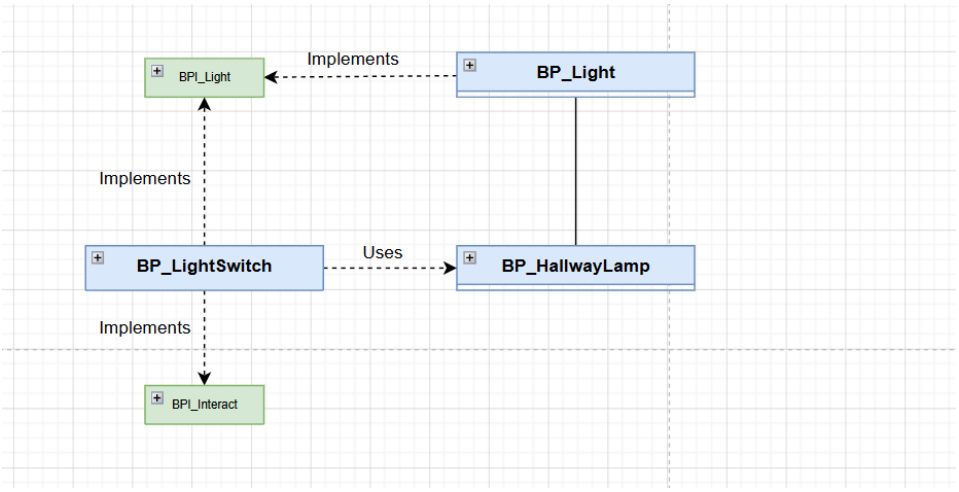


Figura 4.62: UML delle classi per le luci

Light

Questa classe si configura come l'entità genitore della componente HallwayLamp, la quale rappresenta l'asset concretamente istanziato e configurato all'interno dell'ambiente di gioco. L'architettura software implementa due funzioni fondamentali deputate all'attivazione e alla disattivazione delle sorgenti luminose, oltre a un metodo specifico ingegnerizzato per la simulazione dell'effetto di sfarfallio (flickering) a intervalli regolari, guidato dall'esecuzione di una componente Timeline.

Nello specifico, le routine di accensione e spegnimento interrogano preventivamente lo stato operativo della Timeline; in funzione del valore di ritorno di tale controllo

condizionale, il sistema modula l'intensità della luce agendo direttamente su un parametro in virgola mobile (float). La classe implementa l'interfaccia dedicata BPI_Light.

HallwayLamp

All'interno di questa classe, che modella l'asset concretamente istanziato all'interno dello scenario, viene implementata la gestione della visibilità di un componente scenico deputato alla simulazione dell'illuminazione volumetrica attraverso il pulviscolo. Tale effetto visivo (*God Ray*) viene realizzato mediante l'applicazione e la manipolazione dinamica di un'apposita istanza di materiale (*Dynamic Material Instance*), che ne governa la resa e l'interazione con l'ambiente circostante.

BeginPlay All'interno di questa routine, il sistema esegue l'operazione di conversione di tipo (Cast) verso l'istanza globale GameInstance, memorizzandone l'indirizzo in una variabile di riferimento locale. Contestualmente, viene effettuata l'associazione (Binding) ai relativi delegati d'evento (Event Dispatcher) e la configurazione dello stato del modulo GodRay; nello specifico, l'algoritmo valuta lo stato operativo della sorgente luminosa, inibendo la visibilità del componente volumetrico qualora la lampada risulti spenta.

Accensione Il metodo ereditato SwitchOn esegue in prima istanza l'invocazione della routine omonima definita all'interno della classe genitore, preservandone la logica di inizializzazione originaria. Contestualmente all'attivazione della sorgente luminosa, la funzione interviene sul modulo GodRay, ripristinando la visibilità del componente volumetrico per simulare in modo coerente la diffusione della luce attraverso il pulviscolo atmosferico.

LightSwitch

Questa classe modella il modulo di interfaccia hardware-utente con cui l'avatar interagisce per governare lo stato di attivazione o disattivazione dei sistemi di illuminazione dislocati nelle diverse stanze. Al fine di garantire il corretto instradamento dei comandi, ogni singola istanza di interruttore memorizza e mantiene una coppia di referenze locali ai rispettivi attori di tipo luce presenti all'interno dello specifico ambiente di competenza. La componente implementa l'interfaccia BPI_Interact.

Accensione e Spegnimento Mediante l'esecuzione del metodo ToggleLight, il sistema sovraintende alla logica di commutazione dello stato elettrico degli interruttori associati all'ambiente di competenza. La routine esegue preliminarmente una verifica condizionale su una variabile di stato booleana, deputata a tracciare la condizione dell'interruttore (*attivo/inattivo*); in funzione dell'esito di tale controllo, l'algoritmo

interpella gli attori referenziati per modificarne i parametri di intensità luminosa, simulando in modo cinematicamente coerente il comportamento e la risposta transitoria del sistema elettrico della stanza.

4.2.5 Game Management

Game Instance

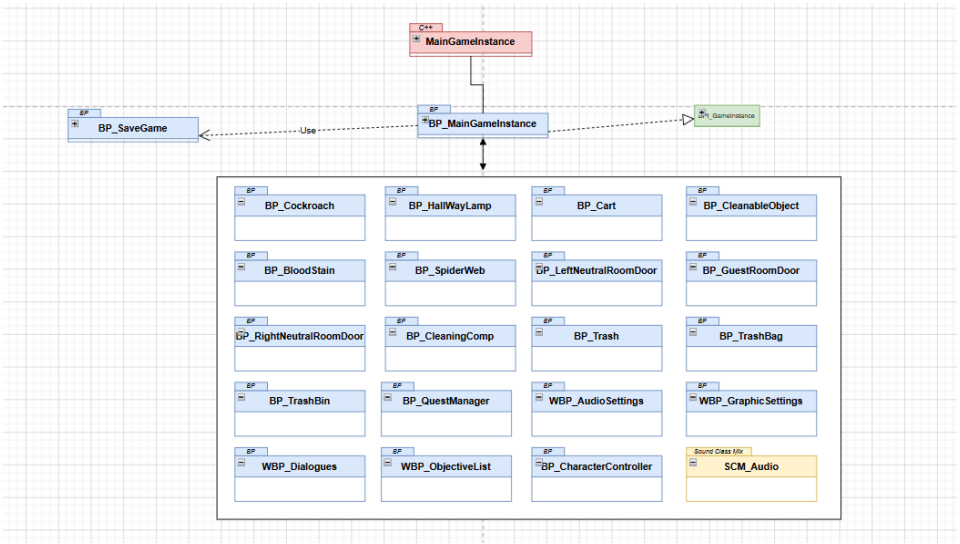


Figura 4.63: UML delle classi per la GameInstance

La classe MainGameInstance si configura come l'infrastruttura software centrale deputata alla persistenza e alla gestione dello stato macroscopico della sessione di gioco; in virtù di tale ruolo architetturale, essa risulta referenziata dalla quasi totalità dei Blueprint che compongono il codice di progetto.

Al suo interno vengono governate le logiche di serializzazione e salvataggio dei parametri audio, nonché l'aggiornamento dei vettori di stato che tracciano il completamento degli obiettivi e delle missioni attive (provvedendo, contestualmente, al loro ripristino (reset) a ogni riavvio della partita). La classe funge inoltre da modulo di interfacciamento e disaccoppiamento tra gli attori dello scenario e i moduli globali implementati come Singleton — nello specifico ObjectiveList, ScriptManager e QuestManager — e presiede all'istanziamento e alla propagazione di eventi in broadcasting mediante l'ausilio di delegati d'evento (Event Dispatcher). Questa classe implementa l'interfaccia dedicata BPI_GameInstance.

Event Init Questo metodo costituisce l'equivalente funzionale dell'evento BeginPlay applicato al ciclo di vita della GameInstance (corrispondente alla routine di inizializzazione Init); di conseguenza, la sua esecuzione avviene in modo univoco durante l'elaborazione del primo frame della sessione di gioco.

All'interno di questa procedura, il sistema attiva determinati delegati d'evento (Event Dispatcher) interni, deputati alla configurazione e al setup delle condizioni geometriche e fisiche iniziali di specifici attori di scenario (come, a titolo esemplificativo, la gestione dei profili di collisione del carrello). Contestualmente, l'algoritmo provvede alla creazione dello slot di salvataggio persistente denominato "Save", all'interno del quale vengono allocati e memorizzati i parametri di riferimento predefiniti relativi al volume generale, agli effetti sonori e alle componenti musicali, e alcuni parametri relativi alle impostazioni grafiche.

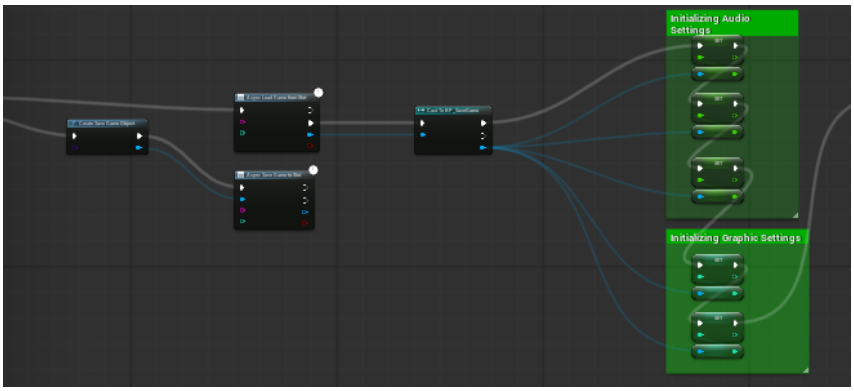


Figura 4.64: Parte del Blueprint GameInstance mostrante l'evento Init

Save Questo metodo, invocato dai moduli dell'interfaccia utente (*UI*) deputati alla configurazione dei parametri audio e delle impostazioni grafiche, presiede alle operazioni di caricamento, serializzazione e salvataggio su disco delle nuove variabili di configurazione all'atto della loro applicazione runtime.

Aggiornamento parametri di pulizia La classe MainGameInstance traccia, mediante l'utilizzo di variabili di tipo intero, sia la quantità complessiva sia la quota parziale progressivamente rimossa di ogni tipologia di residuo interattivo presente nello scenario, quali contaminazioni ematiche (macchie di sangue), sacchi dei rifiuti, agenti dinamici (scarafaggi) e ragnatele.

Sotto il profilo algoritmico, all'atto del proprio istanziamento (BeginPlay), ciascuno di questi attori interroga la GameInstance per recuperare il valore corrente del contatore globale associato alla propria categoria, provvedendo a incrementarlo per registrare la propria presenza nella scena. Successivamente, in concomitanza con la rimozione e deallocazione di ogni singolo residuo a seguito dell'interazione dell'avatar, viene invocato uno specifico metodo di aggiornamento (Update). Quest'ultimo provvede ad aggiornare il bilancio della quantità epurata per la rispettiva classe di oggetti e avvia contestualmente una routine condizionale deputata alla verifica dei requisiti della condizione di vittoria.

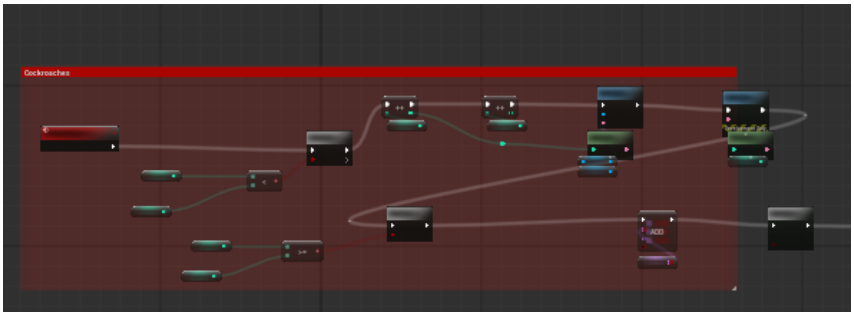


Figura 4.65: Parte del Blueprint GameInstance mostrante uno dei metodi "Update"

Tracciamento delle quest MainGameInstance monitora lo stato di avanzamento delle missioni attraverso l'esecuzione del metodo QuestAdvanced, invocato direttamente dagli attori di scenario interessati.

Tale routine interroga in prima istanza la funzione UpdateQuest esposta dal modulo centrale QuestManager; successivamente, avvia la verifica sul completamento dell'obiettivo analizzando il parametro intero ricevuto in input, il quale notifica lo stadio (step) corrente della missione.

Qualora il valore di tale argomento sia equivalente a due (condizione che attesta il superamento della sotto-quest), il sistema accede alla mappa chiave-valore (Map / Dictionary) che indicizza le diverse missioni di gioco e, utilizzando il Tag della quest come chiave di ricerca, commuta a true la relativa variabile booleana di completamento. A conclusione di tale sequenza, viene eseguito il consueto controllo condizionale sulla pipeline globale per la convalida della condizione di vittoria del livello.

Controllo condizione di vittoria Mediante l'esecuzione del metodo CheckWinCondition, il sistema sovraintende alla verifica dello stato di soddisfacimento dei requisiti necessari al completamento del livello. L'algoritmo effettua un controllo relazionale di uguaglianza, confrontando il valore memorizzato nella variabile numerica CompletedObjectivesCounter con l'ampiezza complessiva (dimensione/size) della mappa associativa MainQuestMap. Qualora l'esito di tale operazione condizionale risulti positivo — attestando la risoluzione di ogni obiettivo prefissato — il sistema interroga il modulo centrale ScriptManager per avviare la routine cinematica di apertura dei battenti dell'ascensore, consentendo all'avatar l'accesso alla sequenza finale di gioco.

Gestione vari elementi di UI a runtime Il sistema si configura come il modulo di interfacciamento centrale (middleware) deputato a fare da tramite tra gli attori presenti nello scenario e le diverse classi preposte alla gestione dei principali elementi dell'interfaccia utente (*UI*), quali l'elenco degli obiettivi, il riquadro dei dialoghi (*Dialogue Box*) e i prompt di comando a schermo. L'architettura software sovraintende, di conseguenza, alla modifica, al ciclo di istanziamento e alla deallocazione dinamica di tali moduli a runtime.

Prendendo in esame il metodo specifico InitializeUI — responsabile dell'inizializzazione dell'elenco obiettivi e dei prompt di input — la routine esegue la pipeline classica di creazione dell'asset grafico (Create Widget), provvede al suo ancoraggio alla visuale del giocatore (Add to Viewport) e memorizza l'indirizzo dell'istanza all'interno di una variabile di riferimento locale, rendendola successivamente accessibile dall'esterno.

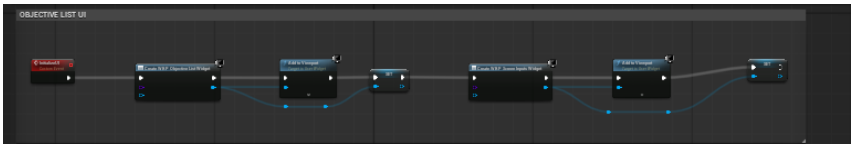


Figura 4.66: Parte del Blueprint GameInstance mostrante il metodo InitializeUI

Script Manager

Questa classe sovraintende alla gestione della progressione della sessione di gioco, la quale si articola in una sequenza ordinata di fasi (step) sequenziali, e centralizza la logica di esecuzione delle diverse tappe che compongono il tutorial introduttivo.

Al fine di monitorare gli spostamenti dell'utente, l'ambiente tridimensionale integra una serie di volumi di rilevamento geometrico (Trigger Box) disposti secondo criteri strategici. Nel momento in cui questi moduli registrano la collisione per sovrapposizione (Overlap) dell'avatar del giocatore o del carrello, invocano in modo diretto la sotto-routine responsabile del monitoraggio e dell'avanzamento dello stato complessivo della partita.

Begin Play All'interno di questa routine di inizializzazione vengono eseguiti i consueti processi di associazione (Binding) ai delegati d'evento (Event Dispatcher) esposti dalla GameInstance. Contestualmente, il sistema provvede alla memorizzazione persistente di alcune variabili di riferimento fondamentali — quali le istanze dell'avatar del giocatore (Player) e della GameInstance stessa — e configura lo stato dei parametri booleani iniziali, instradando le chiamate verso le opportune funzioni di interfaccia al fine di garantire la corretta inizializzazione della nuova sessione di gioco.

In determinate sezioni dello scenario, l'avanzamento della linea narrativa e logica non è vincolato alla diretta rilevazione geometrica da parte di un volume di collisione (Trigger Box), bensì al completamento di specifiche azioni atomiche fondamentali. In virtù di tale architettura guidata dagli eventi, durante la fase di attivazione dell'attore, l'algoritmo effettua il Binding della funzione locale PlayScript ai delegati d'evento degli oggetti direttamente coinvolti nel superamento di tali snodi logici.



Figura 4.67: Parte della funzione BeginPlay mostrante il binding ai dispatcher di alcuni oggetti fondamentali

Logica di avanzamento della partita La gestione centralizzata della progressione della sessione di gioco è interamente demandata all'esecuzione della funzione PlayScript. All'atto della sua invocazione, il sistema incrementa la variabile intera CheckPointNumber, deputata a tracciare lo stadio (step) corrente di avanzamento della partita; successivamente, in funzione del valore assunto da tale contatore, il flusso di esecuzione viene instradato verso uno dei 15 rami condizionali che compongono una struttura di controllo logico di tipo Switch. Ogni ramo corrisponde univocamente a una specifica sequenza deterministica di routine e sotto-funzioni procedurali.

La funzione in esame viene interpellata secondo una logica flessibile: attraverso il meccanismo di associazione (Binding) ai delegati d'evento (Event Dispatcher) di vari attori di scenario, oppure in modo diretto a seguito del transito dell'avatar attraverso determinati volumi di rilevamento geometrico (Trigger Box). Alle funzioni di collisione di tali volumi sono associate logiche di controllo dedicate, configurate specificamente per i requisiti del singolo stadio operativo.

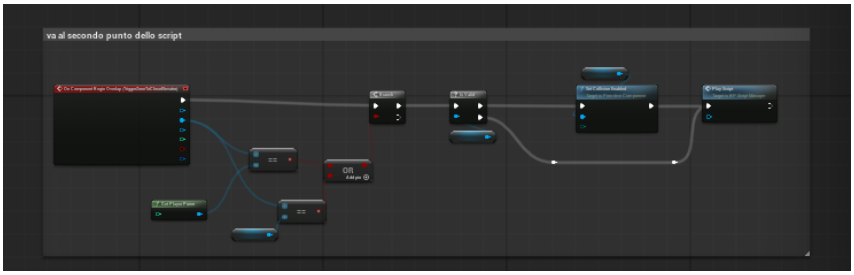


Figura 4.68: Una delle routine di collisione associate a una delle Trigger VBox

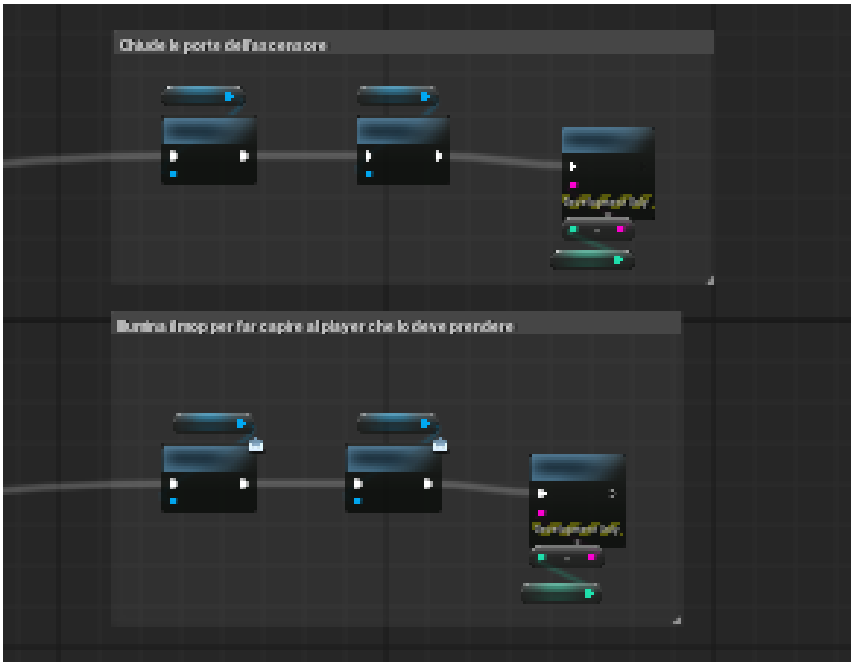


Figura 4.69: Parte della funzione PlayerScript

Gestione delle Luci Mediante l'esecuzione delle funzioni SwitchOnAllTheLights e SwitchOffAllTheLights, il sistema sovraintende all'attivazione e alla disattivazione simultanea dell'intero vettore (array) di sorgenti luminose dislocate lungo il corridoio. L'invocazione di tali metodi sincroni è strutturata per governare lo stato dell'illuminazione ambientale in corrispondenza del raggiungimento di determinate sezioni e snodi logici della sessione di gioco.

Change Focus Mediante questa funzione, il sistema governa l'orientamento forzato della visuale dell'avatar verso specifici elementi d'interesse dello scenario durante le fasi del tutorial, pilotando l'interpolazione del punto di vista attraverso un componente Timeline.

Sotto il profilo del flusso algoritmico, la routine invoca inizialmente apposite funzioni di interfaccia deputate all'inibizione temporanea dei vettori di movimento e di rotazione della telecamera del giocatore; successivamente, esegue la traslazione cinematica guidata del punto di vista e, al termine dell'animazione, provvede al ripristino delle normali capacità di input dell'utente.

Sebbene routine analoghe siano originariamente implementate anche all'interno delle classi del giocatore e del carrello, al fine di preservare i requisiti di scalabilità e modularità del software si è optato per un totale disaccoppiamento architetturale, centralizzando l'intera logica di controllo all'interno di questo modulo di gestione.

Quest Manager

Questa componente sovraintende alla gestione e al tracciamento dello stato di avanzamento delle singole missioni a runtime, venendo invocata in modo diretto dalla funzione QuestAdvanced integrata all'interno della GameInstance. All'atto di tale chiamata, l'attore mittente (la porta dell'ospite) trasmette alla funzione un puntatore a se stesso (self-reference), unitamente ai valori correnti delle proprie variabili locali CurrentCheckpoint e QuestActivationThreshold.

Begin Play All'interno di questa routine, la classe provvede a interpellare l'istanza globale GameInstance al fine di memorizzare il proprio indirizzo (self-reference) all'interno di una variabile di riferimento locale di tipo omonimo, salvaguardando la persistenza del modulo in memoria per le successive operazioni.

UpdateQuest Questa funzione sovraintende alla progressione delle missioni collegate ai diversi personaggi non giocanti (NPC) dislocati all'interno delle stanze.

In prima istanza, la routine memorizza all'interno di variabili locali temporanee il puntatore all'attore mittente e i valori correnti delle sue proprietà CurrentCheckpoint e QuestActivationThreshold. Successivamente, il sistema interroga la GameInstance per estrarre il valore del contatore AllObjectsToCleanAndDispose, il quale esprime la quantità complessiva di elementi d'area rimossi dall'utente; tale metrica viene posta a confronto con la soglia definita in QuestActivationThreshold al fine di verificare se sia stato raggiunto il requisito di pulizia minimo necessario per l'attivazione della specifica missione.

Qualora il controllo relazionale restituisca esito positivo, l'algoritmo esegue una verifica condizionale secondaria sul parametro CurrentCheckpoint: un valore equivalente a due attesta la completa risoluzione della quest, determinando l'interruzione precoce della routine per evitare elaborazioni ridondanti.

In caso contrario, sfruttando il riferimento memorizzato dell'attore mittente, viene invocato il metodo StartDialogue, deputato al ciclo di istanziamento del Widget dei dialoghi a schermo. Il flusso logico viene poi indirizzato verso una struttura di controllo di tipo Switch basata sul valore di CurrentCheckpoint, la quale impartisce le direttive procedurali all'oggetto chiamante per i singoli stadi operativi:

- **Stadio 1:** la routine istruisce l'attore porta per eseguire l'istanziamento (*Spawn*) dell'oggetto di missione in corrispondenza di precise coordinate spaziali della mappa, abilitandone la successiva raccolta da parte del giocatore.
- **Stadio 2:** non viene invocata alcuna routine specifica, in quanto questa fase modella lo stato di stallo computazionale e logico intercorrente tra la generazione dell'*asset* di missione e l'individuazione dello stesso da parte dell'utente.

- **Stadio 3:** attivato in concomitanza con la riconsegna dell'oggetto corretto presso l'istanza della porta associata, il sistema esegue la routine cinematica di apertura del varco geometrico.

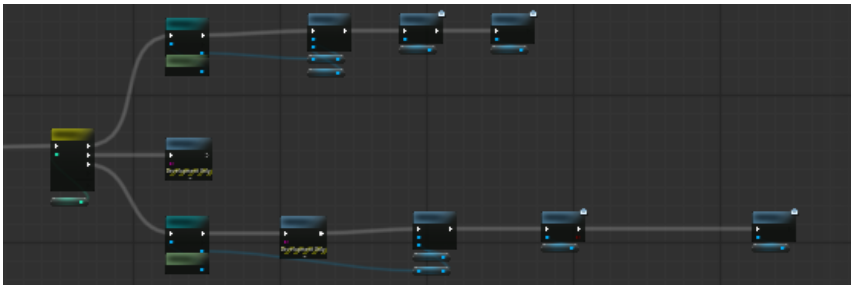


Figura 4.70: Parte della funzione UpdateQuest

4.2.6 User Interface

Classi base custom

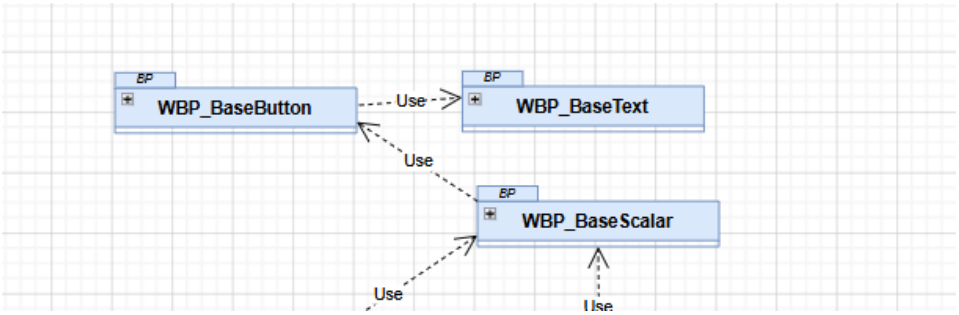


Figura 4.71: UML delle classi per la UI

BaseButton Questa classe eredita e sovrascrive le funzionalità del componente pulsante nativo fornito da Unreal Engine, configurandosi come un'estensione specializzata del widget di base.

Sotto il profilo architetturale, l'implementazione di questa classe derivata risponde a criteri di astrazione e manutenibilità del software: essa consente infatti di centralizzare e rifattorizzare le proprietà stilistiche e logiche di tutti i pulsanti integrati nell'interfaccia utente (UI) agendo esclusivamente sul codice di questo singolo modulo, evitando la duplicazione di configurazioni ridondanti a runtime.

Al fine di garantire il corretto disaccoppiamento tra i livelli dell'applicazione e favorire una programmazione guidata dagli eventi, il componente espone un delegato d'evento specifico denominato On_Clicked, deputato a propagare in broadcasting la notifica di interazione verso le classi gerarchicamente superiori.

BaseText Questa classe si configura come un'estensione specializzata del componente testuale nativo di Unreal Engine, implementata come modulo alternativo per ottimizzare la gestione tipografica dell'interfaccia.

Al suo interno, la classe espone il metodo pubblico Initialize, la cui invocazione è demandata direttamente alla classe BaseButton. Questa routine presiede alla pipeline di configurazione iniziale dell'elemento grafico, determinando e allocando a runtime i parametri relativi alla dimensione del corpo del carattere, alla famiglia del font (Font Family) e alla stringa testuale predefinita.

BaseScalar Questa classe custom definisce l'architettura logica e grafica del componente di scorrimento (Slider) predefinito, implementato per governare la rimodulazione dei parametri relativi alle impostazioni grafiche e audio all'interno dei menu di configurazione.

Sotto il profilo strutturale, l'interfaccia del modulo integra e aggrega due istanze specializzate di tipo BaseButton, deputate rispettivamente all'esecuzione delle routine di incremento e decremento discreto del valore corrente. La visualizzazione dello stato logico e della stringa informativa associata è invece demandata a un elemento centrale di tipo BaseText, preservando così la coerenza tipografica e la modularità complessiva del sistema di interfaccia utente.

Menù & Impostazioni

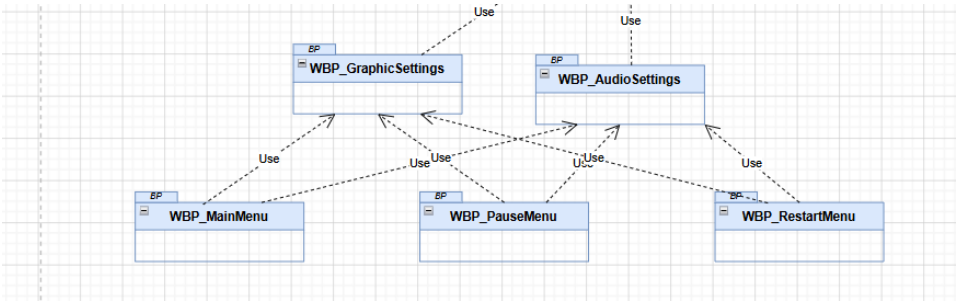


Figura 4.72: UML delle classi per la UI

GraphicSettings Questa classe modella l'entità architetturale preposta alla configurazione e alla manipolazione runtime dei parametri di rendering grafici del motore di gioco. Il modulo governa in modo diretto variabili critiche quali la frequenza di aggiornamento dei fotogrammi (Frame Rate), la risoluzione video, la modalità di visualizzazione della finestra (Window Mode) e il livello di fedeltà e qualità delle ombre (Shadow Quality).

Sotto il profilo dell'interfaccia grafica (UI Layout), la struttura del widget si articola simmetricamente su quattro righe orizzontali, ciascuna dedicata a una singola opzione di configurazione. Ogni riga integra un elemento informativo di tipo BaseText e un modulo di scorrimento e selezione discreta di tipo BaseScalar. L'architettura del

pannello viene completata da due componenti di tipo BaseButton disposti alla base, deputati rispettivamente all'applicazione e alla serializzazione delle modifiche grafiche correnti e all'attivazione della routine di chiusura per il ripristino della schermata di interfaccia precedente.

AudioSettings Questa classe costituisce la controparte speculare del modulo GraphicSettings, risultando specificamente adibita alla gestione e alla manipolazione runtime del comparto audio del gioco.

Sotto il profilo architetturale e funzionale, il componente mutua la medesima logica operativa del pannello grafico.

MainMenu Questo componente modella l'infrastruttura del menu principale, l'interfaccia grafica di primo livello presentata all'utente all'avvio dell'applicazione. Sotto il profilo visivo, il modulo integra l'asset grafico di sfondo, la stringa testuale identificativa del titolo di gioco e tre istanze specializzate di tipo BaseButton, deputate rispettivamente all'avvio della sessione di gioco (Start Game), all'accesso al pannello di configurazione delle opzioni e alla chiamata di interruzione del processo per la chiusura dell'applicazione (Quit Game).

La transizione e l'istanziamento dei sottomenu di configurazione sono interamente governati mediante l'ausilio di un componente nativo denominato WidgetSwitcher. Questa struttura logica fornita dal motore grafico permette di ottimizzare la gestione della visibilità runtime di molteplici moduli di interfaccia sovrapposti, orchestrando lo switch e l'attivazione dei singoli pannelli (grafica, audio o menu principale) in modo simultaneo ed efficiente all'interno del medesimo spazio di viewport.

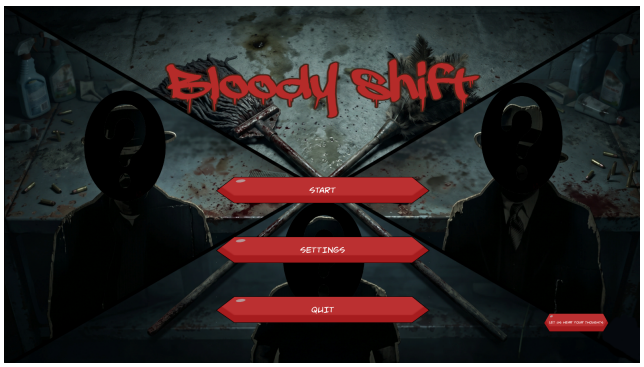


Figura 4.73: Menù principale

PauseMenu & RestartMenu Queste classi si configurano come entità strutturalmente speculari al modulo MainMenu, mutuandone la quasi totalità delle logiche operative, delle pipeline di gestione dell'interfaccia e dei componenti grafici interni.

Nello specifico, l'architettura si articola in due moduli distinti deputati a governare differenti stati della macchina di gioco:

- **Pannello di Pausa (*PauseMenu*):** viene istanziato a schermo in modo dinamico in concomitanza con la rilevazione dell'*input* utente associato alla sospensione della partita, intercettando il flusso di gioco e congelandone temporaneamente le routine di aggiornamento.
- **Schermata di Fine Partita (*RestartMenu*):** viene invocata e renderizzata a *runtime* esclusivamente al soddisfacimento dei requisiti finali del livello, guidando l'avatar verso l'epilogo dell'esperienza e la chiusura della sessione.

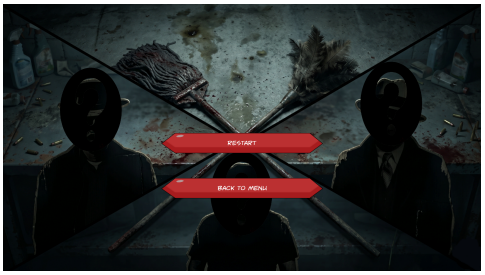


Figura 4.74: Schermata di Fine Partita

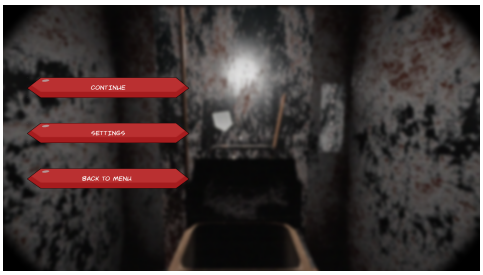


Figura 4.75: Menù di pausa

Miscellaneous

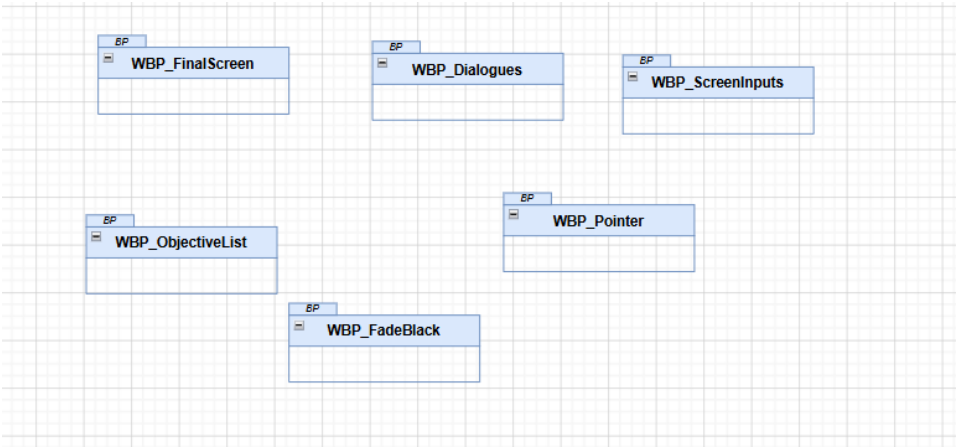


Figura 4.76: UML delle classi per la UI

FinalScreen Questo modulo implementa l'interfaccia grafica conclusiva, renderizzata a schermo esclusivamente all'atto del soddisfacimento delle condizioni di vittoria globali. Sotto il profilo funzionale, la schermata gestisce la scomposizione e la riproduzione temporizzata di una sequenza ordinata di stringhe testuali, finalizzate a veicolare all'utente il messaggio testuale conclusivo attraverso una transizione narrativa dal tono goliardico.



Figura 4.77: Schermata con il messaggio finale

Dialogues Questo modulo si configura come la componente logico-grafica deputata al rendering e alla formattazione dinamica dei dialoghi di gioco a schermo.

A seguito del suo istanziamento, l'attore mittente trasmette a questa classe il riferimento a una specifica struttura dati tabellare (DataTable), le cui righe memorizzano i record relativi all'identificativo dell'oratore (Character Name) e alle rispettive stringhe testuali delle battute.

Successivamente, mediante l'invocazione sequenziale della funzione Next_Line, il sistema esegue l'iterazione algoritmica sulla tabella, estraendone i contenuti e aggiornando gli elementi grafici della viewport. Al fine di ottimizzare il coinvolgimento dell'utente e la resa visiva, è stata inoltre implementata una sotto-routine di scansione temporizzata, la quale simula l'effetto di digitazione progressiva dei caratteri (Typewriter effect) mediante l'inserimento sequenziale delle singole lettere a intervalli di tempo regolari.



Figura 4.78: Dialogo con ospite



Figura 4.79: Dialogo con ospite

ScreenInputs Questa classe sovraintende alla gestione dinamica e al controllo della visibilità dei prompt di comando e dei suggerimenti d'input a schermo. L'attivazione di tale modulo viene determinata a runtime mediante l'interrogazione diretta da parte delle classi dell'avatar del giocatore o del carrello; nello specifico, la routine scatta nel momento in cui l'algoritmo di tracciamento geometrico sferico (Sphere Trace) rileva una collisione valida con un attore contrassegnato come interagibile all'interno dello scenario.

ObjectiveList Questo modulo presiede al tracciamento dinamico e alla visualizzazione a schermo dei contatori relativi alle diverse categorie di residui e contaminazioni d'area, nonché dello stato di avanzamento delle missioni attive.

Sotto il profilo algoritmico, la gestione del livello di visibilità dei dati è vincolata allo stato della sessione: durante la fase di tutorial, il sistema maschera parzialmente le metriche globali, visualizzando esclusivamente una quota ridotta (metà) dei quantitativi totali da epurare; solo al completamento della fase introduttiva, la routine provvede all'aggiornamento e al disvelamento dei parametri reali ed effettivi dello scenario.

Con riferimento al sotto-sistema delle missioni, gli elementi grafici associati rimangono in uno stato di inibizione della visibilità (Hidden / Collapsed) fino all'effettivo istanziamento e attivazione della specifica quest. All'atto dell'attivazione, il widget provvede a esporre dinamicamente i metadati informativi, visualizzando nello specifico l'identificativo testuale dell'oggetto bersaglio da individuare.

FadeToBlack Questo modulo implementa un'interfaccia grafica minimale dedicata alla gestione delle transizioni visive ambientali, configurandosi come una maschera di campitura nera in grado di variare in modo lineare la propria opacità (Fade Effect).

Sotto il profilo algoritmico, il componente interviene in due snodi critici del flusso di gioco, operando una modulazione dinamica del canale alfa (Alpha Channel) del widget: all'avvio della sessione, esegue una transizione decrescente (da opaco a completamente trasparente) per introdurre l'utente allo scenario; all'atto della conclusione della partita, esegue la routine inversa in modo crescente (da trasparente a opaco), determinando l'oscuramento totale della viewport per accogliere i moduli di epilogo.

Pointer Questo modulo grafico minimale funge da elemento di puntamento centrale (Reticle), implementato per incrementare il livello di immersività dell'utente e ottimizzare il sistema di feedback visivo in tempo reale.

Sotto il profilo funzionale, il comportamento del widget è direttamente asservito ai risultati dell'algoritmo di tracciamento sferico (Sphere Trace) condotto dall'avatar del giocatore o dal carrello: nell'istante in cui la sottomisura geometrica rileva un'intersezione valida con un attore contrassegnato come interagibile, il sistema innesca una transizione di scala che incrementa il raggio del punto geometrico centrale. Viceversa, qualora il vettore di tracciamento non intercetti alcuna entità attiva o entri in collisione con elementi di scenario statici e non interagibili, il modulo esegue la routine inversa, ripristinando istantaneamente le dimensioni minime predefinite del puntatore.

4.2.7 Gestione Audio

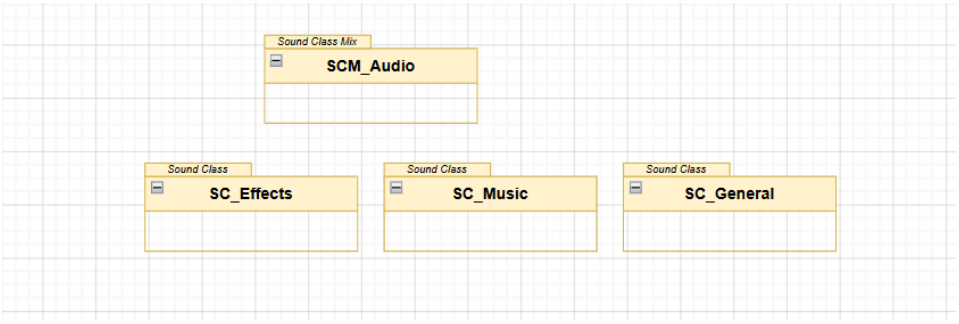


Figura 4.80: UML delle classi per il comparto audio

SCM_Audio Questo modulo implementa l'infrastruttura del missaggio audio centrale (Sound Class Mixer) posta al vertice dell'architettura sonora del progetto. Sotto il profilo funzionale, la classe è deputata alla macrogestione coordinata delle diverse classi audio singole (Sound Classes), orchestrando la manipolazione in tempo reale di parametri globali quali l'attenuazione dell'ampiezza (Volume Ducking), il guadagno del segnale e il coefficiente di intonazione (Pitch).

Attraverso tale centralizzazione, il sistema consente di raggruppare gerarchicamente i flussi acustici per macro-categorie omogenee — come gli effetti sonori ambientali (SFX) e le tracce musicali — permettendo alle schermate di configurazione dell'interfaccia utente (AudioSettings) di rimodulare dinamicamente le relative costanti di volume a runtime in modo sincrono ed efficiente.

SC_General, SC_Music & SC_Effects Queste entità rappresentano le singole classi audio (Sound Classes) configurate per segmentare e controllare i flussi sonori relativi al volume generale (Master), alla componente musicale e agli effetti sonori ambientali (SFX). All'interno della pipeline di assetting, a ogni singola sorgente d'onda (Sound Wave) o istanza logica di missaggio (Sound Cue) viene associata la rispettiva classe audio di pertinenza.

Il motore grafico consente di strutturare tali classi secondo una gerarchia ad albero di tipo padre-figlio. All'interno di questa architettura, la classe radice General (Master) sovrintende gerarchicamente alle classi derivate Musica ed Effetti. Di conseguenza, qualsiasi variazione apportata ai parametri acustici della classe radice si propaga per ereditarietà a tutti i nodi figli dell'albero sonoro.

Sotto il profilo matematico, le modifiche introdotte sui singoli nodi secondari operano in regime di cascata, accumulandosi al valore ereditato dal nodo padre mediante un algoritmo di attenuazione di tipo moltiplicativo.

4.3 Testing & Improvements

Al termine della fase di implementazione — durante la quale sono state consolidate le fondamenta logico-architetturali necessarie a garantire il funzionamento minimale del software in termini di meccaniche e sistemi di gioco — il ciclo di vita del progetto è evoluto verso una fase iterativa di Testing, Debugging e ottimizzazione progressiva. Tale transizione è stata finalizzata a elevare l'indice di fruibilità e l'esperienza utente, rendendo il prodotto accessibile a soggetti estranei al contesto aziendale o al team di sviluppo interno.

Sotto il profilo operativo, sono state generate molteplici compilazioni eseguibili (Build) della soluzione software, distribuite sequenzialmente e sottoposte a verifica presso tre cluster distinti di utenti: il team di sviluppo originario, il personale interno dell'azienda ospitante e un campione statistico di utenti esterni. Il feedback empirico derivato da tali sessioni di verifica incrociata ha permesso di mappare e categorizzare diverse criticità tecniche e di game design, sulle quali si è successivamente intervenuti in ottica di rifattorizzazione.

Relativamente agli interventi effettuati sui *Blueprint* durante la fase di *Testing*, l'attività di sviluppo si è focalizzata sull'ottimizzazione prestazionale e logica dei seguenti moduli architetturali:

- **Utensili e attori interagibili (oggetti pulibili):** l'attenzione è stata rivolta al perfezionamento delle routine di interazione reciproca e alla sincronizzazione degli stati logici tra le primitive di pulizia e le rispettive geometrie bersaglio.
- **Sistemi di apertura (porte e sportelli):** sin dalle prime fasi del progetto, la simulazione deterministica del comportamento fisico e cinematico dei varchi ha presentato notevoli complessità computazionali. Tale criticità è riconducibile alla necessità di coordinare simultaneamente un'ampia matrice di costanti condizionali, variabili in virgola mobile (*float*), componenti *Timeline* per l'interpolazione dei movimenti e vincoli fisici specifici per ogni singola istanza.
- **Routine di *Spawn* dell'avatar al rilascio del carrello:** come precedentemente illustrato nel capitolo dedicato all'implementazione, nel momento in cui l'utente interrompe l'interazione con il carrello, l'attore principale (abilitato all'uso degli utensili) viene istanziato nuovamente in corrispondenza del punto di ancoraggio del veicolo, simulando l'abbandono dello strumento logistico. In questa fase, è risultato particolarmente complesso isolare e risolvere i casi limite in cui l'avatar, a seguito della procedura di disaccoppiamento (*Detach*), subiva compenetrazioni geometriche rimanendo bloccato all'esterno dei confini del livello.
- **Prompt di comando a schermo (*Input UI*):** il lavoro ha riguardato l'affinamento della macchina a stati preposta alla visibilità dinamica delle

stringhe di comando. Nello specifico, sono state modellate le condizioni logiche necessarie a inibire, attivare o rimuovere i suggerimenti d'*input* in funzione dello stato di usabilità *runtime* degli oggetti mirati dal giocatore.

- **Condizioni di terminazione della sessione (*End Game*):** il *testing* ha permesso di validare la routine di chiusura della partita e il rispettivo metodo di controllo centralizzato nella *GameInstance*. In questo ambito, la gestione della linea narrativa finale del protagonista ha richiesto una profonda revisione logica: a causa di anomalie nella *pipeline* d'ascolto degli eventi, il dialogo conclusivo tendeva a registrarsi in modo ridondante (invocazioni multiple) o a manifestarsi in concomitanza di nodi temporali errati.

Sotto il profilo della revisione del *Game Design*, sono state adottate precise scelte stilistiche ed estetiche finalizzate a incrementare il coinvolgimento atmosferico dell'esperienza e a ottimizzare l'orientamento dell'utente (*User Guidance*), indirizzandolo in modo intuitivo verso le corrette meccaniche di gioco.

- **Effetto *post-processing* della telecamera (*VHS Effect*):** attingendo ai canoni estetici e alla popolarità dei *trend* legati ai *Liminal Spaces* e al genere cinematografico dei *Found Footage*, si è proceduto all'integrazione di un pacchetto di *asset* mediante il *marketplace* nativo di Unreal Engine (Fab). Tale modulo ha permesso di applicare alla telecamera di gioco un filtro di *Post-Processing* video e un *layer* sonoro dinamico, capaci di simulare fedelmente la distorsione analogica e l'aberrazione cromatica tipiche dei supporti a nastro magnetico (VHS).

- **Integrazione di dialoghi contestuali e scriptati:** al fine di mitigare potenziali ambiguità informative durante la fase di *tutorial*, il sistema è stato arricchito con l'inserimento di linee di dialogo scriptate del protagonista. Tali stringhe testuali si attivano in corrispondenza di specifici eventi di gioco (ad esempio: saturazione del contenitore dei rifiuti, completamento della fase introduttiva, contaminazione del mocio o attivazione cinematica dell'ascensore nella fase di epilogo), fungendo da *feedback* narrativo diretto per l'utente.

- **Ottimizzazione e diversificazione dei materiali ematici:** per elevare il grado di realismo visivo e la varietà geometrica delle contaminazioni di sangue, si è intervenuti sui parametri fisici del materiale (*Material Instances*), manipolandone i canali di opacità e di lucentezza (*Specular / Roughness*) per simulare la consistenza del sangue fresco. Contestualmente, come approfondito nel capitolo dedicato all'implementazione, è stata ingegnerizzata una logica di allocazione multipla che consente di associare dinamicamente più *texture* di proiezione (*Decal*) a un singolo attore di scenario.

- **Inclusione di cartonati antropomorfi per gli NPC:** l'approccio preliminare prevedeva la caratterizzazione visiva delle stanze dei diversi ospiti esclusivamente

mediante la disposizione di elementi di arredo e oggettistica tematica unica. Successivamente, con l'obiettivo di inserire una componente goliardica e ironica nel ciclo di gioco, si è optato per la modellazione tridimensionale di sagome bidimensionali (cartonati) raffiguranti i tre ospiti, posizionate secondo criteri strategici di *Level Design* all'interno dei rispettivi ambienti.

- **Effetti di illuminazione volumetrica artificiale (*God Rays*):** per simulare il fenomeno ottico dello *scattering* della luce attraverso il pulviscolo atmosferico, sono stati modellati dei solidi cuneiformi estrusi. A tali geometrie è stato applicato un materiale personalizzato con proprietà di trasparenza e mascheratura sfumata, ottenendo un effetto di colonna di luce volumetrica a basso impatto computazionale.

Capitolo 5

Risultati

5.1 Contesto

Successivamente alla fase di Testing e Debugging preliminare, il software è stato presentato ufficialmente all'evento Svilupparty, tenutosi a Bologna in data 9 maggio 2026. Il team si è presentato come Spin-off dell'azienda ospitante e rappresentante dell'Università Politecnica delle Marche. La partecipazione a tale manifestazione ha costituito uno snodo fondamentale all'interno della pipeline di sviluppo, configurandosi come un'opportunità strategica per condurre sessioni estensive di Playtesting sul campo. Questa attività ha permesso di raccogliere un cospicuo volume di dati empirici e feedback qualitativi da parte di un pubblico eterogeneo e completamente estraneo al contesto aziendale e al team di sviluppo originario, fornendo metriche preziose per la successiva ottimizzazione della User Experience.

5.1.1 Svilupparty

Lo Svilupparty si configura come una manifestazione pubblica a cadenza annuale, istituita e organizzata dall'associazione IPID (Italian Party of Indie Developers) con l'obiettivo di valorizzare e celebrare la produzione degli sviluppatori di videogiochi indipendenti sul territorio nazionale.

Nel corso delle tre giornate in cui si articola l'evento, molteplici studi di sviluppo di diverse dimensioni e segmenti di mercato hanno l'opportunità di riservare uno spazio espositivo dedicato per presentare i propri prodotti a un pubblico eterogeneo e specializzato. Tale contesto non rappresenta esclusivamente una preziosa isola di Playtesting sul campo per raccogliere metriche e feedback empirici diretti sul ciclo di gioco, ma costituisce un ecosistema strategico di networking professionale, volto a favorire lo scambio di competenze, la condivisione di metodologie e la nascita di sinergie tra professionisti con percorsi accademici e lavorativi affini.



Figura 5.1: Logo dello Svilupparty [8]

5.1.2 Discussione sui Feedback

Ai molteplici utenti che hanno dedicato il proprio tempo alla fruizione della soluzione software sviluppata è stato successivamente somministrato un questionario di valutazione composto da 26 quesiti mirati, nella forma di 4 domande a risposta aperta, 13 a scelta chiusa e 9 su una scala di valutazione in decimi. La struttura e il contenuto metrico di tale strumento di indagine sono stati preventivamente definiti e validati in stretta sinergia con il team di sviluppo e il management dell'azienda ospitante.

Nelle sezioni successive si procederà all'analisi sistematica dei macro-risultati emersi da ciascun quesito, discutendone criticamente le implicazioni logiche, tecniche e di game design, al fine di delineare le linee guida strategiche per le future iterazioni e l'ottimizzazione incrementale del prodotto. In virtù di rendere il questionario più accessibile, le domande e possbili risposte sono state trasposte in lingua inglese.

"What do you usually play?"

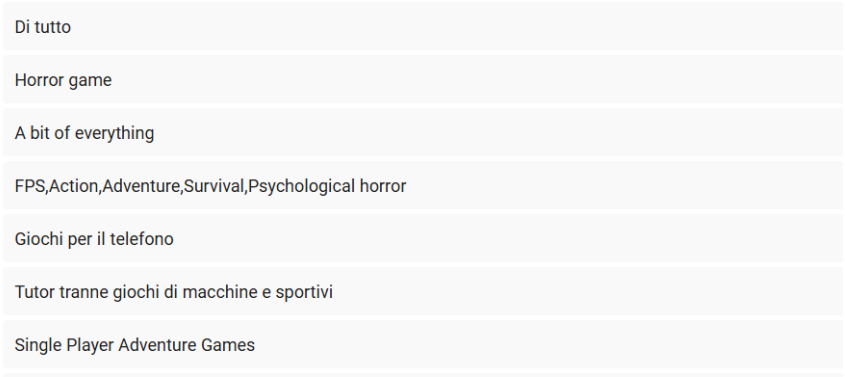


Figura 5.2: Parte delle risposte alla domanda

"How many hours do you play per week on average?"

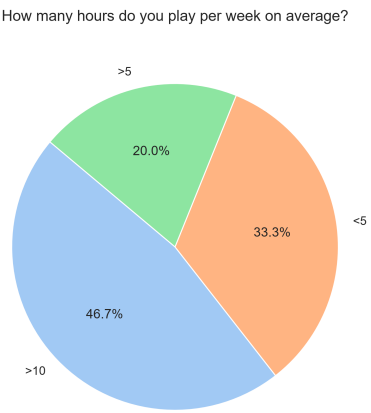


Figura 5.3: Diagramma a torta per le risposte a questa domanda

Questo secondo quesito è stato formulato con l'obiettivo di quantificare il volume di tempo settimanale che il campione di utenti dedica all'attività videoludica, delineandone così l'intensità di consumo ordinaria.

Dall'analisi dei risultati empirici emerge che quasi il 50% dei partecipanti dichiara un tempo di fruizione medio superiore alle 10 ore settimanali, configurandosi come un cluster di utenti ad alta frequenza (Core Gamers). Al contrario, le restanti quote del campione si distribuiscono in percentuali minori lungo fasce di coinvolgimento più moderate: una prima frazione attesta il proprio consumo tra le 5 e le 10 ore settimanali, mentre la restante parte dichiara un monte ore inferiore alle 5 ore settimanali, ascrivibile a un profilo di consumo occasionale (Casual Gamers).

"What platform do you usually play on?"

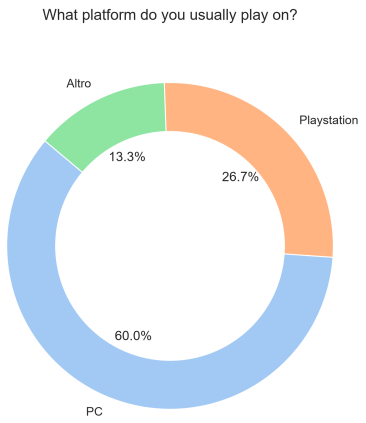


Figura 5.4: Diagramma a torta per le risposte a questa domanda

Questo quesito è stato integrato nell'indagine al fine di mappare la distribuzione dell'ecosistema hardware di riferimento, individuando le piattaforme computazionali maggiormente impiegate dal cluster di utenti per l'attività videoludica ordinaria.

Sotto il profilo dei risultati, i dati evidenziano una netta ed egemone prevalenza di utenti associati all'architettura Personal Computer (PC Gaming). A fronte di questa forte polarizzazione tecnologica, si registra una presenza decisamente più marginale e contratta di fruitori afferenti ai sistemi Console (sia domestici che portatili) e ai dispositivi mobili (Mobile Gaming).

"Have you played other viscera titles like Viscera Cleanup?"

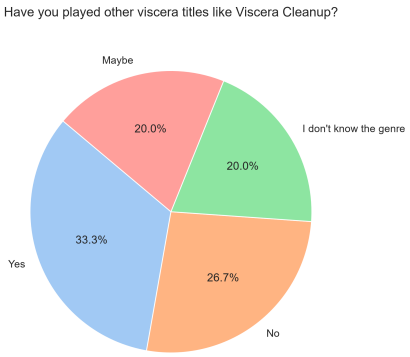


Figura 5.5: Diagramma a torta per le risposte a questa domanda

Questo quesito è stato introdotto al fine di verificare il grado di familiarità e l'esperienza pregressa del campione di utenti con produzioni ascrivibili al medesimo genere del progetto in esame, nello specifico il sotto-genere dei cosiddetti Viscera-like.

I dati empirici derivati dall'interrogazione mostrano uno spettro di risposte estremamente variegato ed eterogeneo. Il campione si divide infatti in quote equivalenti tra utenti che dichiarano di aver già fruito di titoli analoghi, soggetti che non hanno mai avuto esperienze con tale genere, una percentuale di rispondenti incerti e, infine, un cluster che dichiara di non possedere alcuna conoscenza teorica o pratica in merito a questa specifica nicchia di mercato.

"If so, how much does ours differ from similar games?"

Questo quesito è stato formulato con l'obiettivo di determinare il posizionamento competitivo del software sviluppato, analizzando come la soluzione proposta si differenzi rispetto ai prodotti concorrenti presenti sul mercato secondo la percezione diretta dell'utenza.

Escludendo la frazione di rispondenti che ha dichiarato di non possedere esperienze pregresse con il genere Viscera-like, l'indagine ha evidenziato precisi elementi di forte discontinuità e originalità stilistico-funzionale. Nello specifico, i fattori maggiormen-

te valorizzati dal campione d'utenza sono stati l'ambientazione caratterizzata da un'estetica infernale, l'integrazione degli ospiti (NPC) supportata dalle rispettive linee di dialogo contestuali, l'introduzione della meccanica del carrello logistico e, infine, l'ampia diversificazione e specificità funzionale degli utensili da pulizia messi a disposizione del giocatore.

"On a scale of zero to ten, how likely are you to recommend "Blody Shift" to a friend?"

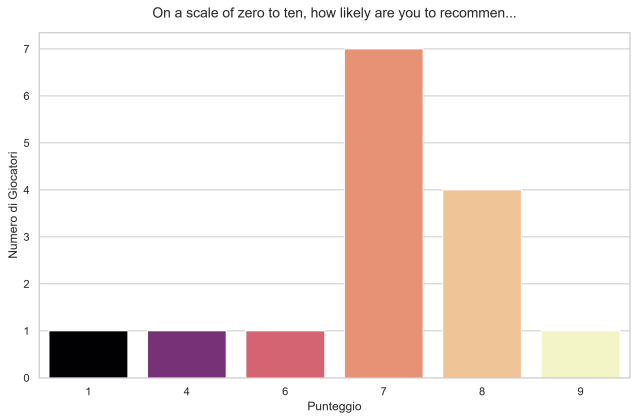


Figura 5.6: Diagramma a colonne per le risposte a questa domanda

Questo quesito valuta l'indice di gradimento globale e la propensione alla raccomandazione del software da parte degli utenti, invitando il campione a quantificare su una scala Likert da 1 a 10 la probabilità di consigliare il titolo a terzi.

Sotto il profilo empirico, i dati raccolti delineano una distribuzione fortemente polarizzata verso i valori più alti della scala: la netta maggioranza del campione si posiziona nella fascia d'eccellenza, esprimendo valutazioni comprese tra i punteggi 7 e 9, a testimonianza di un elevato grado di validazione dell'esperienza d'uso complessiva. Al contrario, si registra una quota nettamente minoritaria e residuale di valutazioni inferiori o uguali al punteggio 6, ascrivibili a utenti che manifestano riserve o un minor coinvolgimento nei confronti delle meccaniche proposte.

"Did you find the game easy to pick up?"

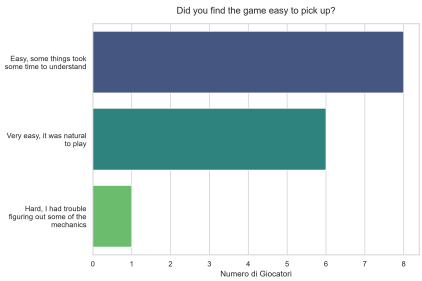


Figura 5.7: Diagramma a righe per le risposte a questa domanda

Il settimo quesito è stato formulato con l'obiettivo di valutare l'accessibilità iniziale del software, quantificando l'indice di apprendibilità (Learnability) e la facilità di comprensione immediata delle logiche e dei sistemi di interazione da parte dell'utente alla prima sessione d'uso.

Sotto il profilo analitico, i dati raccolti evidenziano un riscontro estremamente positivo: la quasi totale interezza del campione d'utenza ha classificato il prototipo software come "facile" o "molto facile" da comprendere e padroneggiare. Questa convergenza sottolinea l'efficacia delle scelte di design applicate alle prime fasi di gioco, confermando come la curva di apprendimento risulti ottimizzata e priva di barriere cognitive significative per il fruitore.

"What other things would you have liked to see in the game? We're all ears :)"

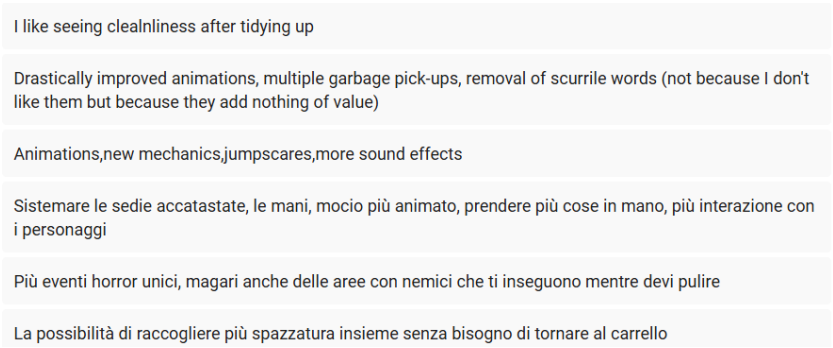


Figura 5.8: Alcune risposte a questa domanda

"How much would you pay for the game on release?"

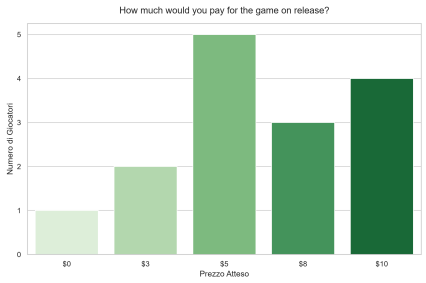


Figura 5.9: Diagramma a colonne per le risposte a questa domanda

Questo quesito è stato introdotto al fine di condurre un'analisi di sensibilità al prezzo (Price Sensitivity Analysis), valutando direttamente il valore economico percepito del software da parte dei consumatori in vista di una sua potenziale immissione sul mercato. Dall'analisi dei dati empirici emerge una distribuzione variegata e caratterizzata da una configurazione di tipo bimodale, che frammenta il campione in due segmenti commerciali ben definiti. Circa la metà dell'utenza si dichiara infatti disposta a investire una cifra compresa tra gli 8 e i 10 dollari, posizionandosi in una fascia di prezzo medio-alta per le produzioni indipendenti di questa scala e attestando il riconoscimento di un solido valore intrinseco nel prototipo. Al contrario, la restante quota del campione si colloca in una fascia più contratta ed economica, stimando il costo ideale tra i 3 e i 5 dollari, un posizionamento tipico di un'utenza orientata a prodotti di stampo più casual o a micro-produzioni di portata contenuta.

"Which part of the game did you like the most?"

Questo quesito è stato introdotto con l'obiettivo di identificare il nucleo di massimo coinvolgimento del software, mappando le funzionalità e le componenti di gioco che hanno generato il più alto indice di gradimento e divertimento all'interno del campione.

I dati qualitativi raccolti evidenziano una netta convergenza da parte della maggioranza dei rispondenti, i quali hanno espresso un forte apprezzamento per due elementi cardine dell'esperienza. In primo luogo, spicca la componente d'azione legata alla meccanica di eliminazione delle blatte, che evidentemente funge da ottimo elemento di rottura dinamica rispetto alle routine di pulizia più statiche. In secondo luogo, il campione ha valorizzato ampiamente le componenti goliardiche e ironiche integrate nelle linee di missione dei vari ospiti (NPC), a conferma di come la caratterizzazione narrativa e l'estetica scanzonata del mondo di gioco rappresentino un pilastro fondamentale per l'identità del prodotto e per la fidelizzazione dell'utente.

"Which part of the game did you like the least?"

Specularmente al quesito precedente, questa interrogazione è stata formulata per identificare i punti di attrito dell'esperienza, mappando le funzionalità e le dinamiche che hanno generato il minor indice di gradimento o che hanno introdotto colli di bottiglia nell'esperienza d'uso.

Anche in questo caso, l'analisi qualitativa dei dati evidenzia una netta convergenza attorno a due criticità strutturali ben definite, che confermano e rafforzano i requisiti futuri emersi nei precedenti nodi dell'indagine. Da un lato, gli utenti hanno manifestato una chiara insoddisfazione nei confronti del vincolo logico che impedisce la raccolta e la movimentazione simultanea di molteplici elementi di rifiuto, costringendo a un ciclo di interazione frammentato e ripetitivo per il trasporto dei singoli attori al contenitore. Dall'altro, i riscontri convergono sulla percezione di una precoce saturazione del ciclo di gioco, giudicato ridondante nel lungo periodo a causa di una temporanea carenza di eventi dinamici di disturbo o di una progressione narrativa più articolata capace di sostenere la ritenzione dell'utente.

"If you feel that the game needs that one thing to become much better, please share it:"

In perfetta coerenza con le criticità strutturali emerse nei precedenti nodi dell'indagine, questo quesito è stato strutturato per raccogliere in modo diretto le proposte di ottimizzazione espresse dall'utenza, traducendo i bisogni dei fruitori in requisiti funzionali per lo sviluppo futuro.

L'analisi qualitativa dei suggerimenti evidenzia una netta convergenza attorno a due precise direttrici di intervento. La prima riguarda l'implementazione del sistema di raccolta multipla degli oggetti, una modifica architetturale ritenuta imprescindibile per snellire i flussi di inventario e abbattere i tempi morti nelle routine di pulizia. La seconda richiesta si focalizza invece sul potenziamento dell'orientamento dell'utente all'interno della fase introduttiva. Per mitigare i margini di ambiguità del tutorial, il campione ha suggerito l'integrazione di elementi visivi dedicati nell'interfaccia grafica (HUD), nello specifico attraverso l'introduzione di un indicatore dinamico di obiettivo posizionato nella parte superiore centrale dello schermo, preposto a comunicare in tempo reale lo stato di avanzamento e la sotto-missione corrente.

"Do you think you found a bug? Report it below and we'll fix it."

Questo quesito è stato introdotto al fine di condurre un'attività di tracciamento dei difetti (Bug Reporting) basata sull'esperienza sul campo, consentendo di identificare le anomalie software non emerse durante le sessioni di test interne.

Sotto il profilo del controllo qualità, i dati raccolti evidenziano un quadro complessivo di soddisfacente stabilità strutturale del codice e dei Blueprint. Escludendo infatti alcuni casi limite circoscritti, in cui l'avatar ha subito compenetrazioni geometriche

tali da causarne la caduta al di fuori dei confini del livello (Out of Bounds) o il blocco fisico a seguito della routine di disaccoppiamento (Detach) dal carrello logistico, non sono emerse criticità o regressioni funzionali a carico degli altri moduli architetturali del gioco. Questo riscontro positivo evidenzia l'efficacia del lavoro di debugging preliminare condotto sul resto dei sistemi interattivi.

"How smooth did you find the game (Frame Rate) on your device?"

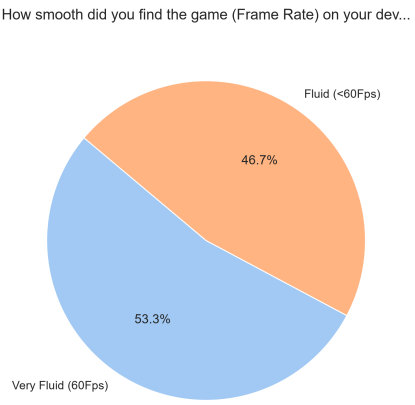


Figura 5.10: Diagramma a torta per le risposte a questa domanda

Sebbene questo quesito sia stato formulato per valutare la fluidità percepita dell'esperienza, i relativi dati analitici rivestono un'importanza puramente indicativa nel contesto globale dell'indagine; l'intero ciclo di playtesting allo Svilupparty, infatti, è stato condotto su una singola configurazione hardware preimpostata e fornita direttamente dal team di sviluppo, annullando di fatto la variabilità dei test in ambiente eterogeneo.

Ciononostante, il riscontro empirico fornito dal campione si è rivelato ampiamente positivo, descrivendo un'esperienza fluida e caratterizzata da una stabilità di rendering priva di fenomeni significativi di latenza (lag) o cali improvvisi di fotogrammi (framerate drops). Questo risultato, pur condizionato dall'ambiente hardware controllato, offre una parziale validazione del lavoro svolto in fase di profilazione e ottimizzazione degli asset tridimensionali, dei materiali e delle routine di calcolo, confermando l'efficienza computazionale del prototipo software su configurazioni di fascia target.

"How did you find the mechanics of the cart?"

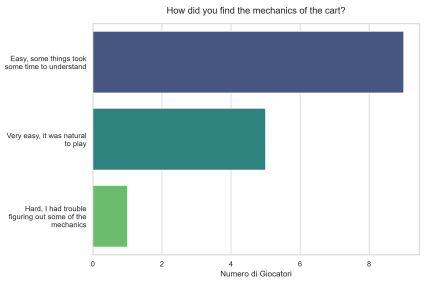


Figura 5.11: Diagramma a righe per le risposte a questa domanda

Questo quesito è stato strutturato per valutare specificamente il tasso di gradimento e l'efficacia del carrello logistico, una delle meccaniche core del ciclo di gioco. I dati empirici raccolti evidenziano un riscontro ampiamente positivo da parte del campione, confermando come questa dinamica di movimentazione sia stata percepita come un elemento caratterizzante, in grado di arricchire il valore funzionale e l'originalità dell'intera esperienza d'uso.

"How satisfying did you find the cleaning mechanics?"

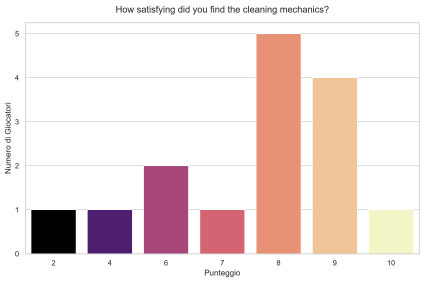


Figura 5.12: Diagramma a colonne per le risposte a questa domanda

Questo quesito espande la metrica analizzata nel nodo precedente, approfondendo il livello di soddisfazione generale percepito dall'utenza in merito all'intero ecosistema delle meccaniche di pulizia.

Sotto il profilo qualitativo, la netta maggioranza del campione ha espresso valutazioni ampiamente superiori alla sufficienza, attestandosi stabilmente nella fascia alta della scala di gradimento. Questo posizionamento dei dati empirici evidenzia il successo complessivo del modello interattivo sviluppato, confermando l'efficacia del Game Design nel catturare l'interesse del giocatore e nel rendere stimolante e appagante il ciclo di interazione principale su cui si fonda l'intera architettura del titolo.

"Would you like it if there were other mechanics? If so which ones?"

Questo quesito, parzialmente sovrapponibile ad alcune delle metriche analizzate in precedenza, è stato formulato per raccogliere indicazioni puntuali e proattive in merito all'espansione del sistema di gioco, invitando gli utenti a proporre l'introduzione di specifiche meccaniche interattive.

A differenza di altri nodi dell'indagine, i dati qualitativi raccolti non mostrano una convergenza netta su un'unica funzionalità, delineando invece una dispersione di suggerimenti eterogenei ma ugualmente significativi per lo sviluppo futuro. Tra le proposte emerse con maggiore frequenza spiccano tre filoni concettuali: l'introduzione di meccaniche dedicate esplicitamente al riordino e al posizionamento millimetrico degli oggetti scenici, l'inserimento di elementi di disturbo orrorifico improvviso (Jumpscares) per accentuare la tensione atmosferica e, infine, l'implementazione di un sistema di progressione economica che permetta al giocatore di investire le risorse guadagnate nell'acquisto di potenziamenti e modifiche funzionali per la strumentazione in dotazione (Upgrade System).

"Is the UI clear and understandable?"

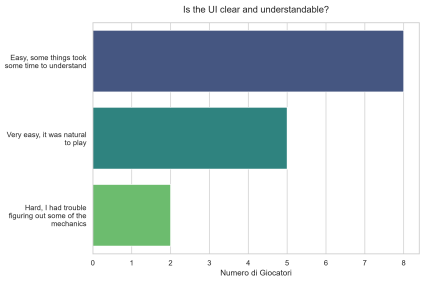


Figura 5.13: Diagramma a righe per le risposte a questa domanda

Questo quesito è stato integrato nell'indagine al fine di raccogliere riscontri mirati in merito all'accessibilità visiva e all'usabilità dell'interfaccia utente (UI) e dei sistemi di visualizzazione a schermo (HUD - Heads-Up Display), metriche fondamentali per valutare la corretta trasmissione delle informazioni di gioco al fruitore.

Dall'analisi dei dati empirici emerge un quadro ampiamente positivo, caratterizzato da una netta polarizzazione delle risposte: la maggioranza assoluta dei partecipanti ha valutato l'interfaccia come pienamente chiara, intuitiva e priva di barriere cognitive significative nell'organizzazione dei layout. Al contrario, si registra una contrazione dei dissensi, limitati a una ridotta quota minoritaria di utenti che ha riscontrato margini di ambiguità o criticità nella leggibilità degli elementi visivi; questo dato residuale offre comunque spunti preziosi per una futura fase di rifattorizzazione dell'interfaccia in ottica di User Experience.

"How would you rate the quality of the sound effects and visual feedback?"

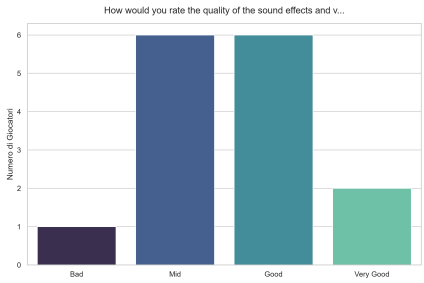


Figura 5.14: Diagramma a colonne per le risposte a questa domanda

Questo quesito è stato introdotto al fine di quantificare l'indice di gradimento relativo al comparto degli effetti visivi (VFX) e sonori (SFX), valutando l'impatto dell'esperienza sensoriale e della componente sinestetica sulla qualità percepita del software.

I dati raccolti evidenziano un riscontro non particolarmente favorevole, delineando una valutazione globale che si attesta su livelli di sufficienza o di moderata criticità. Questo andamento rispecchia fedelmente i vincoli operativi che hanno caratterizzato la genesi del progetto: l'assenza di un budget iniziale allocato allo sviluppo ha inevitabilmente limitato la pipeline di produzione, precludendo la possibilità di implementare soluzioni acustiche e particellari elaborate o strutturate a regola d'arte. Il feedback del campione crea la necessità di un futuro investimento mirato all'ottimizzazione del design audiovisivo per innalzare il valore produttivo complessivo dell'opera.

"Did you like the game's art style and models?"

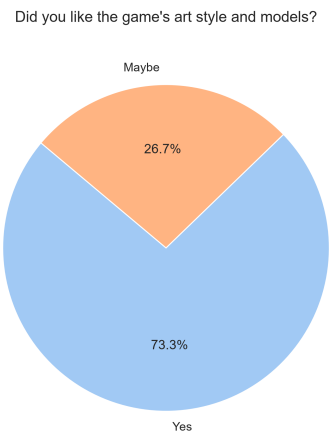


Figura 5.15: Diagramma a torta per le risposte a questa domanda

Questo quesito è stato formulato con l'obiettivo di valutare la risposta estetica del pubblico nei confronti del progetto, analizzando nello specifico il grado di apprezzamento della direzione artistica e la qualità realizzativa degli asset e dei modelli tridimensionali impiegati nello scenario.

I dati empirici delineano un riscontro pressoché unanime e ampiamente favorevole, caratterizzato da una netta concentrazione di valutazioni positive all'interno del campione. A fronte di questa spiccata convergenza verso l'approvazione delle scelte stilistiche e della resa visiva del software, si registra una quota del tutto marginale e contratta di rispondenti che si posiziona in una fascia di incertezza. Questo bilancio mostra che l'efficacia delle linee guida estetiche adottate durante la pipeline di produzione tridimensionale, confermando l'ottimo impatto visivo del titolo sull'utenza di riferimento.

"Were the instructions/tutorials provided clear and sufficient?"

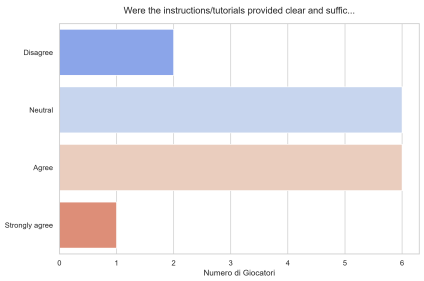


Figura 5.16: Diagramma a righe per le risposte a questa domanda

Questo quesito riveste un ruolo di fondamentale importanza nel processo di validazione metodologica dell'esperimento, in quanto analizza in modo mirato l'efficacia della fase introduttiva e la qualità delle istruzioni fornite. L'obiettivo risiede nel valutare la solidità delle soluzioni di Game Design implementate per agevolare l'apprendimento, con particolare riferimento alla gestione del carico cognitivo dell'utente distratto o meno propenso alla lettura dei testi di gioco.

I dati empirici raccolti delineano una distribuzione fortemente divisiva e polarizzata, che frammenta il campione in due macro-aree equivalenti. Circa la metà dei rispondenti ha manifestato criticità nell'approccio iniziale, segnalando una parziale ambiguità o una carenza di incisività nelle istruzioni del tutorial; al contrario, la restante metà del cluster si è dichiarata pienamente soddisfatta della chiarezza del flusso informativo proposto.

Questo andamento, pur evidenziando una parziale efficacia del sistema attuale per una frazione del pubblico, evidenzia la necessità di operare una profonda rifattorizzazione della pipeline di inserimento dell'utente. In ottica di sviluppo futuro, emerge l'urgenza di implementare soluzioni di User Guidance più dinamiche, intuitive e meno dipendenti dal testo statico, riducendo i tempi di assimilazione delle regole fondamentali ed eliminando i punti di attrito emersi durante l'indagine.

"Did you find it difficult to understand the main objective of the game?"

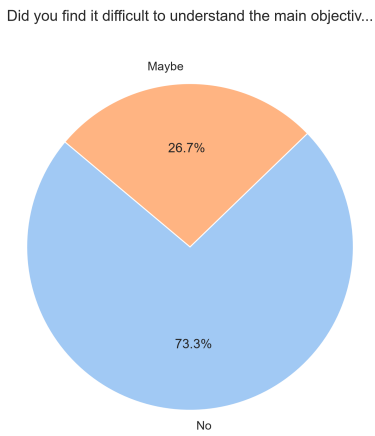


Figura 5.17: Diagramma a torta per le risposte a questa domanda

Questo quesito approfondisce l'analisi avviata nella sezione precedente, focalizzandosi in modo specifico sulla comprensione dell'obiettivo principale dell'esperienza di gioco al fine di isolare eventuali barriere cognitive macroscopiche.

A differenza delle criticità emerse nella struttura dettagliata del tutorial, l'indagine non ha evidenziato particolari elementi di criticità sotto questo profilo. I dati raccolti confermano che lo scopo primario del software — ovvero l'attività di pulizia e ripristino ambientale — risulta immediatamente intellegibile e trasparente fin dall'inizio della sessione.

"Did you find parts of the game confusing? If so which ones?"

Questo quesito è stato formulato con l'obiettivo di mappare in modo olistico eventuali deficit di chiarezza strutturale all'interno dell'esperienza di gioco, isolando i momenti in cui l'utente ha percepito un disorientamento spaziale o concettuale.

In perfetta continuità con i riscontri emersi nei nodi analitici precedenti, i dati qualitativi evidenziano una convergenza critica attorno a due specifiche carenze del sistema informativo del software. Da un lato, il campione ha segnalato un deficit nei sistemi di navigazione e nell'orientamento spaziale, lamentando una scarsità di indicazioni generali atte a guidare i flussi di movimento all'interno dello scenario. Dall'altro, sono emerse chiare riserve in merito alla gestione dell'interfaccia utente dedicata al tracciamento delle missioni; la configurazione attuale della UI è stata percepita come poco incisiva nel comunicare gli obiettivi attivi. Questi risultati confermano l'esigenza di una revisione congiunta tra layout dell'interfaccia e segnaletica di gioco per azzerare i tempi di attrito cognitivo.

"How would you rate the difficulty level of the proposed challenge?"

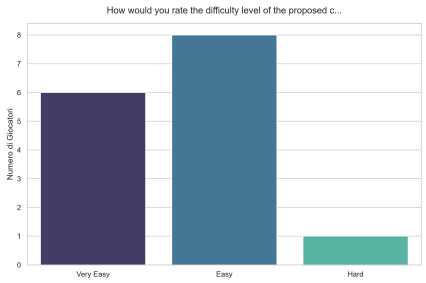


Figura 5.18: Diagramma a colonne per le risposte a questa domanda

Questo quesito è stato introdotto al fine di mappare la difficoltà percepita dal campione, valutando il bilanciamento del livello di sfida e l'accessibilità complessiva dell'esperienza ludica.

I dati raccolti delineano una distribuzione fortemente asimmetrica e convergente: escludendo una ristretta e marginale minoranza di rispondenti, la quasi totalità del campione ha classificato l'esperienza come facile e ampiamente accessibile. Sotto il profilo del design delle regole, questo riscontro empirico conferma l'assenza di barriere d'ingresso frustranti o picchi di difficoltà anomali, rendendo il titolo come un prodotto accessibile a più segmenti di utenza.

"How responsive did you find the controls?"

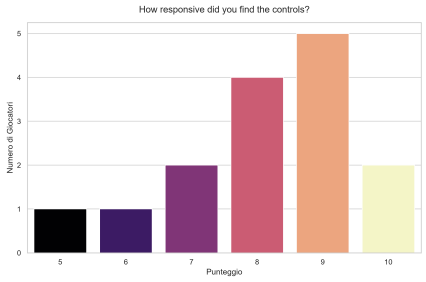


Figura 5.19: Diagramma a colonne per le risposte a questa domanda

Questo quesito è stato integrato nell'indagine al fine di esaminare sotto il profilo tecnico la responsività percepita del sistema di input, offrendo una metrica di valutazione diretta sul lavoro di ingegnerizzazione e mappatura delle funzioni di comando implementate all'interno delle classi dei personaggi controllabili.

L'analisi quantitativa dei dati evidenzia un riscontro estremamente positivo e strutturalmente solido. Le valutazioni espresse dal campione si collocano tutte ampiamente al di sopra della soglia di sufficienza, mostrando peraltro una spiccata concentrazione verso i valori di eccellenza per un'ampia porzione dei rispondenti. Questo risultato evidenzia l'efficacia del codice di gestione degli input, confermando

come il sistema di controllo garantisca un'interazione fluida, immediata e priva di attriti cinetici tra l'utente e l'avatar virtuale.

"How intuitive was it to understand the game controls from the first minutes?"

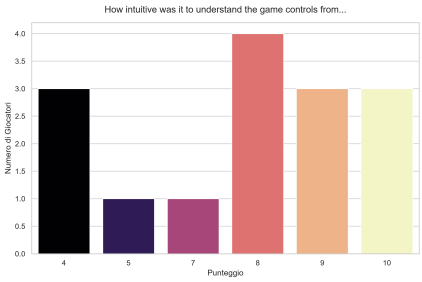


Figura 5.20: Diagramma a colonne per le risposte a questa domanda

Questo quesito è stato formulato per analizzare in modo specifico il tasso di apprendimento iniziale, valutando il grado di intuitività e l'accessibilità dei comandi principali nei primi minuti della sessione di gioco, un intervallo temporale critico per l'esperienza di inserimento dell'utente.

L'analisi qualitativa dei dati ha evidenziato una netta polarizzazione del campione, che si è strutturato attorno a due precise e antitetiche linee di pensiero. Da un lato, una porzione significativa di utenti ha dichiarato di aver assimilato la mappatura dei comandi in modo immediato e privo di ostacoli cognitivi; dall'altro, una quota equivalente ha segnalato di aver riscontrato palesi difficoltà e incertezze nelle battute iniziali.

Questa spaccatura nei risultati empirici non fa che confermare e supportare le criticità emerse nei precedenti nodi analitici della presente indagine. Il riscontro evidenzia inequivocabilmente la necessità di operare una profonda revisione metodologica e strutturale delle strategie di User Guidance; per lo sviluppo futuro del titolo, diviene prioritario riprogettare l'architettura con cui vengono presentati e veicolati i sistemi di interazione di base, ottimizzando la curva di apprendimento per uniformare l'esperienza d'uso dell'intero target di riferimento.

Capitolo 6

Conclusioni

Il presente elaborato ha analizzato nel dettaglio la pipeline di sviluppo di Bloody Shift, un titolo di genere Viscera-like concepito come progetto di tirocinio e prima esperienza all'interno del panorama del game development professionale.

L'avvio del progetto ha richiesto una rigorosa declinazione strategica per allineare le ambizioni del team alle risorse limitate tipiche di una produzione indipendente. Attraverso un'attenta analisi del mercato di Steam e dei diretti competitor, si è strutturata un'offerta commerciale capace di riproporre le dinamiche consolidate del genere all'interno di una cornice narrativa inedita e fortemente caratterizzata. Fondamentale, prima ancora dell'apertura dell'editor e della scrittura dei Blueprint, è stata la fase di coordinamento sinergico con il team di modellazione 3D; la definizione congiunta del level design, del parco asset primario e del pitch di gioco ha permesso di blindare i requisiti di produzione, riducendo al minimo i colli di bottiglia e i rallentamenti in fase di sviluppo.

Sotto il profilo strettamente implementativo, la sfida principale ha riguardato il bilanciamento tra l'adozione di un'architettura interamente visuale in Blueprint e l'impatto computazionale del framework di Unreal Engine. In quest'ottica, si è rivelato fondamentale il ricorso sistematico alle Blueprint Interfaces (in primis BPI_Interact) al fine di disaccoppiare le classi e diminuire le referenze dirette (hard references) tra gli attori di scena. Parallelamente, l'ottimizzazione delle prestazioni è stata supportata da un uso rigoroso e limitato delle strutture iterative (For e ForEach Loop) su grandi moli di dati, preservando la stabilità del frame time.

La fase di testing preliminare e la generazione delle prime build stabili hanno evidenziato la complessa gestione fisica e logica delle porte e del carrello logistico, elementi caratterizzati da una vasta casistica di conflitti di collisione e comportamenti imprevisti da parte dell'utente. Successivamente, la fase di deployment e il collaudo sul campo in occasione dello Svilupparty hanno fornito dati preziosi: sebbene l'impatto generale sul pubblico sia stato ampiamente positivo, i riscontri hanno evidenziato i limiti strutturali legati all'assenza di budget e alcune lacune sul fronte del Game Design, legate principalmente alla mancanza di un elemento catalizzatore capace di consolidare la ritenzione a lungo termine.

In virtù di una futura release commerciale su Steam, la roadmap di sviluppo dovrà necessariamente prevedere:

- Una profonda rifattorizzazione dei sistemi di *User Guidance* e del *tutorial*, ottimizzando l'interfaccia per scongiurare il disorientamento del giocatore.
- L'introduzione di una meccanica emergente (legata al senso di urgenza o a eventi dinamici) capace di spezzare la ciclicità delle *routine* di pulizia.
- Un *refactor* completo del comparto audiovisivo, sfruttando il sistema particellare Niagara per innalzare il valore produttivo percepito.

Nel complesso, il caso di Bloody Shift evidenzia come, in un progetto indipendente caratterizzato da risorse limitate, l'integrazione tra analisi competitiva, progettazione iterativa, sviluppo modulare e playtesting possa supportare la definizione delle priorità di sviluppo. I risultati raccolti durante il playtest pubblico hanno tuttavia evidenziato criticità relative soprattutto alla user guidance e alla varietà del ciclo di gioco. Tali evidenze costituiscono indicazioni progettuali per le successive iterazioni del prototipo, ma dovranno essere validate attraverso ulteriori sessioni di testing con campioni più ampi e procedure di valutazione maggiormente strutturate.

Bibliografia

[1] RuneStorm, "Viscera cleanup detail," Piattaforma Steam, 2015, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/246900/Viscera_Cleanup_Detail/

[2] President Studio, "Crime scene cleaner," Piattaforma Steam, 2024, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/1040200/Crime_Scene_Cleaner/

[3] Terahard Studios, "Monster mop up," Piattaforma Steam, 2024, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/2451310/Monster_Mop_Up/

[4] Sleepyhead Games, "The werecleaner," Piattaforma Steam, 2024, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/2795000/The_WereCleaner/

[5] M. C. Games, "Cash cleaner simulator," Piattaforma Steam, 2026, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/2488370/Cash_Cleaner_Simulator/

[6] L. Games, "Leaf blower co." Piattaforma Steam, 2026, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/3571710/Leaf_Blower_Co/

[7] C. Games, "Goblin cleanup," Piattaforma Steam, 2026, ultimo accesso: 3 giugno 2026. [Online]. Available: https://store.steampowered.com/app/2748340/Goblin_Cleanup/

[8] Associazione IPID. (2026) Svilupparty. Italian Party of Indie Developers. [Online]. Available: https://www.ipid.dev/svilupparty/

[9] D. Arsenault, "Video game genre, evolution and innovation," *Eludamos: Journal for computer game culture*, vol. 3, no. 2, pp. 149–176, 2009. [Online]. Available: https://septentrio.uit.no/index.php/eludamos/article/view/6003

[10] G. Freeman, L. Li, N. J. McNeese, and K. Schulenberg, "Understanding and mitigating challenges for non-profit driven indie game development to innovate game production," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23. New York,

NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3544548.3580976

[11] Video Game Insights, "Global indie games market report 2024," Sensor Tower, Tech. Rep., September 2024, data as of September 30, 2024. [Online]. Available: https://app.sensortower.com/vgi/assets/reports/VGI_Global_Indie_Games_Market_Report_2024.pdf

[12] F. B. I. F. Portela, "Predicting success in the indie video game market," 2026.

[13] C. M. Kanode and H. M. Haddad, "Software engineering challenges in game development," in *2009 Sixth International Conference on Information Technology: New Generations*. IEEE, 2009, pp. 260–265. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5070627/

[14] S. Carless. (2024) Gamediscoverco: Steam refunds research. GameDev Reports / GameDiscoverCo. Analisi statistica basata su dati aggregati riguardante i tassi di rimborso (refund rates) sulla piattaforma Steam. [Online]. Available: https://gamedevreports.substack.com/p/gamediscoverco-steam-refunds-research

[15] Gamalytic. (2025) Steam market analytics: Games released between 2025-01-01 and 2025-10-21. Gamalytic. Piattaforma di analisi dati per il mercato Steam. Dati estratti per l'analisi della saturazione e dei trend di rilascio nel settore indie durante il 2025. [Online]. Available: https://gamalytic.com/steam-analytics?first_release_date_min=2025-01-01&first_release_date_max=2025-10-21

[16] T. Fullerton, *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*, 5th ed. A K Peters/CRC Press, 2024. [Online]. Available: https://books.google.com/books?hl=it&lr=&id=7xjYCwAAQBAJ&oi=fnd&pg=PP1&dq=Game+Design+Workshop:+A+Playcentric+Approach+to+Creating+Innovative+Games&ots=CHDbWpWLw4&sig=rX8_B9UYYU4WiwtHTU_zzpFZVHU

[17] Epic Games. UInterface | Unreal Engine C++ API Reference. [Online]. Available: https://dev.epicgames.com/documentation/unreal-engine/API/Runtime/CoreUObject/UInterface

Ringraziamenti

Ringraziamenti

Jesi, Ottobre 2026

Giuseppe Cervelli