UNIVERSITÀ
POLITECNICA
DELLE MARCHE

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

Dal Concept all'implementazione: game design, logiche di interazione e collaudo del videogioco *Bloody Shift*

Candidato:
Nicolò Gioacchini

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

Sommario

Il presente lavoro di tesi documenta la progettazione, l'implementazione e la valutazione di *Bloody Shift*, un videogioco tridimensionale sviluppato in Unreal Engine 5.6.1 prevalentemente attraverso il sistema di *visual scripting* Blueprint. L'obiettivo del lavoro è analizzare la fattibilità dell'impiego di Blueprint nello sviluppo di un prototipo videoludico completo, con particolare attenzione alla modularità dell'architettura, alla comunicazione tra componenti e alla gestione delle risorse.

L'architettura software è stata progettata adottando strategie di disaccoppiamento tra i principali moduli applicativi. In particolare, l'impiego di *Blueprint Interfaces* ed *Event Dispatchers* ha consentito di limitare il ricorso al *Casting* tra classi personalizzate e di ridurre la presenza di dipendenze rigide. Tali scelte hanno favorito la separazione delle responsabilità, la manutenibilità del codice e l'integrazione parallela dei sistemi sviluppati dai diversi membri del team.

Dal punto di vista del *game design*, *Bloody Shift* sperimenta l'ibridazione tra le meccaniche tipiche dei simulatori di pulizia (*cleaning simulator*) e un'ambientazione horror-grottesca. Il progetto introduce un ritmo di gioco più rapido, una struttura basata su obiettivi intermedi e una componente narrativa finalizzata a ridurre la percezione di monotonia associata alle attività ripetitive del genere.

Il contributo personale ha riguardato la programmazione di alcune delle principali meccaniche del *core loop*, tra cui la gestione del carrello delle pulizie, delle porte interattive e dei personaggi non giocanti infestanti, oltre all'implementazione del sistema di illuminazione e delle logiche di onboarding e progressione narrativa.

La valutazione del prototipo ha coinvolto 15 partecipanti, di cui 10 durante sessioni di *playtesting* supervisionato allo Svilupparty 2026 e 5 mediante test da remoto. I risultati hanno fornito indicazioni preliminari positive: 14 partecipanti su 15 hanno valutato favorevolmente sia la facilità di apprendimento delle meccaniche sia l'utilizzo del carrello delle pulizie. Al tempo stesso, il testing ha permesso di individuare alcune criticità relative alla chiarezza del tutorial, alla gestione delle collisioni durante il distacco dal carrello e alle limitazioni del sistema di inventario. Tali risultati delineano una base utile per le successive attività di rifinitura, ampliamento dei contenuti e validazione tecnica del progetto.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Lo sviluppo contemporaneo di videogiochi richiede l'impiego di motori grafici capaci di integrare rendering, gestione degli asset, simulazione fisica, intelligenza artificiale e logiche di interazione all'interno di architetture software articolate. In questo contesto, Unreal Engine 5.6.1 mette a disposizione un modello di sviluppo ibrido che affianca al linguaggio C++ il sistema di *visual scripting* Blueprint [1]. Il C++ risulta particolarmente indicato per l'implementazione di sistemi a basso livello e di componenti sensibili alle prestazioni, mentre i Blueprint consentono di rappresentare graficamente le logiche applicative e di *gameplay*, favorendone la prototipazione, la modifica e l'ispezione durante lo sviluppo. Dal punto di vista esecutivo, le logiche definite tramite Blueprint vengono compilate in un *bytecode* interpretato dalla Blueprint Virtual Machine. Questa modalità introduce un costo computazionale superiore rispetto all'esecuzione diretta di codice nativo; tuttavia, la documentazione tecnica di Unreal Engine evidenzia come la scelta tra Blueprint e C++ debba essere valutata in relazione alla natura e alla frequenza delle operazioni implementate [2]. Per le logiche di interazione ad alto livello, che non richiedono necessariamente elaborazioni intensive a ogni fotogramma, i Blueprint possono quindi rappresentare una soluzione adeguata, soprattutto quando risultano prioritari la rapidità di iterazione e il coordinamento tra figure con competenze differenti. La rilevanza del *visual scripting* all'interno della *pipeline* produttiva risiede infatti nella possibilità di prototipare, verificare e modificare rapidamente le meccaniche di gioco. Tale caratteristica può agevolare l'organizzazione modulare dei sistemi e il bilanciamento progressivo delle interazioni durante le diverse fasi di produzione [3]. Inoltre, rispetto alla gestione manuale delle risorse tipica del C++, l'uso dei Blueprint riduce l'esposizione diretta ad alcune categorie di errori legati alla memoria, pur richiedendo un'adeguata progettazione delle dipendenze tra classi e asset.

A partire da queste premesse, il problema tecnico-progettuale affrontato nel presente elaborato consiste nel valutare la fattibilità dello sviluppo di un videogioco tridimensionale completo basato prevalentemente su Blueprint, prestando particolare attenzione alla modularità dell'architettura, alla comunicazione tra componenti e alla gestione delle risorse. A tale scopo vengono documentate la progettazione, l'implementazione e la valutazione di ***Bloody Shift***, assunto come caso di studio per analizzare l'intero ciclo di produzione di un prodotto videoludico indipenden-

te. La definizione del concept è stata preceduta da un'analisi del mercato Steam e dei principali titoli appartenenti al segmento dei simulatori di pulizia (*cleaning simulator*). L'osservazione dei competitor ha evidenziato alcune criticità ricorrenti, tra cui la ripetitività delle attività, la presenza di tempi morti e la limitata varietà degli obiettivi nelle sessioni prolungate. Questi elementi hanno orientato il progetto verso un ritmo più rapido e verso l'integrazione di componenti narrative e ambientali finalizzate a diversificare il ciclo principale di gioco.

Bloody Shift colloca pertanto le tradizionali attività di pulizia all'interno di un hotel infernale caratterizzato da un'estetica horror-grottesca. Il giocatore interpreta un inserviente incaricato di rimuovere macchie di sangue, smaltire rifiuti e contrastare infestazioni di scarafaggi. Alla struttura tipica del simulatore sono stati affiancati obiettivi narrativi a breve termine, rappresentati da un sistema di *quest* assegnate da tre ospiti speciali. L'intento progettuale è verificare se la combinazione tra un *core loop* immediato, una progressione scandita da obiettivi intermedi e un'ambientazione atipica possa contribuire a ridurre la percezione di monotonia associata al genere di riferimento.

L'obiettivo dell'elaborato è quindi descrivere e analizzare le soluzioni progettuali e tecniche adottate per la realizzazione del caso di studio. Particolare attenzione è riservata alle strategie di comunicazione tra classi implementate in Blueprint, tra cui l'impiego di *Blueprint Interfaces* ed *Event Dispatchers*, utilizzati per limitare le dipendenze rigide e favorire la separazione delle responsabilità tra i diversi moduli. All'interno del progetto, il contributo personale ha riguardato la programmazione di alcune delle principali meccaniche interattive, tra cui il funzionamento delle porte, la gestione del carrello delle pulizie e il comportamento dei personaggi non giocanti ostili. Il lavoro ha inoltre incluso l'implementazione del sistema di illuminazione, l'integrazione di asset tridimensionali realizzati da altri membri del team e lo sviluppo delle logiche a supporto dell'onboarding e della progressione narrativa.

La fase conclusiva del progetto ha previsto un'attività di *user testing* finalizzata a raccogliere indicazioni preliminari sull'usabilità, sulla chiarezza delle meccaniche e sulla stabilità della versione Beta. Una parte delle sessioni è stata condotta in presenza durante l'evento Svilupparty 2026, consentendo di osservare direttamente il comportamento degli utenti e di raccogliere reazioni spontanee durante l'interazione. Ulteriori sessioni sono state svolte da remoto. Al termine delle prove, i partecipanti hanno compilato un questionario appositamente predisposto, strutturato secondo principi generali di valutazione dell'usabilità. I dati ottenuti sono stati successivamente analizzati in forma descrittiva e qualitativa, con l'obiettivo di individuare punti di forza, criticità tecniche e possibili direzioni di miglioramento.

L'elaborato si articola in sette capitoli. Dopo la presente introduzione, il Capitolo 2 presenta lo Stato dell'Arte, analizzando il genere dei *cleaning simulator* e i principali titoli assunti come riferimento. Il Capitolo 3 descrive la metodologia progettuale, il *game design*, la struttura del *core loop* e il protocollo di valutazione. Il Capitolo 4 approfondisce l'implementazione tecnica in Unreal Engine, con particolare attenzione

all'architettura Blueprint, alle meccaniche interattive e ai sistemi visivi e sonori. Il Capitolo 5 presenta il software realizzato e i risultati emersi dalle sessioni di *playtesting*, inclusi i dati raccolti durante Svilupparty. Il Capitolo 6 discute criticamente i risultati, le scelte progettuali, i limiti metodologici e il confronto con i prodotti di riferimento. Infine, il Capitolo 7 riassume i principali contributi del lavoro e delinea le possibili direzioni di sviluppo futuro.

Capitolo 2

Stato dell'arte

Questo capitolo propone un'analisi del panorama videoludico di riferimento, con l'obiettivo di inquadrare il genere dei simulatori di pulizia (*cleaning simulator*) e comprenderne le dinamiche fondamentali. Prima di addentrarsi nella progettazione e nello sviluppo tecnico di *Bloody Shift*, risulta infatti essenziale esaminare i titoli che hanno definito e consolidato gli standard di questa nicchia di mercato. Attraverso lo studio dei principali *competitor*, verranno analizzate l'evoluzione delle meccaniche di gioco, le soluzioni narrative adottate e le criticità strutturali ricorrenti. Questa indagine preliminare fornirà il contesto teorico e di *game design* necessario per giustificare le successive scelte architetturali e le deviazioni stilistiche introdotte in questo elaborato.

2.1 Analisi dei simulatori e "Cleaning Games"

Il successo commerciale dei simulatori di professioni e dei cosiddetti *cleaning games* (come *PowerWash Simulator* o *House Flipper*) affonda le proprie radici in dinamiche psicologiche ben precise, legate al fenomeno emergente del *cozy gaming* [4]. Questi titoli si discostano dai tradizionali canoni videoludici orientati all'azione e alla competizione, offrendo al giocatore un ambiente focalizzato sulla sicurezza emotiva e sulla ripetizione familiare.

Dal punto di vista psicologico, l'attrattiva di rimettere ordine al caos virtuale trova spiegazione nella Teoria dell'Autodeterminazione (*Self-Determination Theory*) [4], la quale postula che i comportamenti umani siano guidati dal bisogno di sentirsi autonomi, di raggiungere obiettivi tangibili e di connettersi emotivamente. I simulatori di pulizia assecondano questi bisogni primari fornendo mansioni chiare e facilmente realizzabili: il giocatore osserva una superficie sporca tornare progressivamente pulita, ricevendo una gratificazione immediata senza dover sottostare a minacce esterne, limiti di tempo punitivi o stati di ansia. Questo genera un senso di "trasporto" mentale verso un luogo digitale percepito come rassicurante e privo di stress, rendendo il videogioco un vero e proprio strumento terapeutico per la modulazione dell'umore [4].

2.1.1 La deriva "Dark" e il problema della ripetitività

Nonostante il comprovato effetto rilassante, il genere dei simulatori soffre di un intrinseco tallone d'Achille: la difficoltà di mantenere a lungo l'ingaggio del giocatore in assenza di conflitti. Sviluppare un *gameplay loop* soddisfacente basato sulla totale libertà, senza far subentrare la noia o la mancanza di stimoli, rappresenta una delle sfide progettuali più ardue per i *game designer* [4]. Come confermato dalle tendenze di mercato analizzate nel Capitolo 1, il ritmo eccessivamente dilatato di queste produzioni porta spesso a un rapido abbandono del titolo da parte dell'utenza, poiché l'effetto appagante della pulizia cede il passo a una sensazione di "lavoro virtuale" ripetitivo (*grinding*).

Per ovviare a questo problema di natura strutturale, il mercato indipendente ha iniziato a esplorare l'ibridazione dei generi, sovvertendo il concetto di rilassamento per introdurre elementi narrativi grotteschi o macabri. Alcuni sviluppatori hanno dimostrato come le dinamiche pacifiche di questi titoli possano coesistere efficacemente con atmosfere tensive, malinconiche o addirittura legate alla paura [5]. L'inserimento di contesti atipici, come la detersione di scene del crimine o di laboratori infestati, crea un contrasto ironico e disturbante che riaccende l'interesse dell'utente, sfidandone le aspettative.

È partendo da questa debolezza strutturale dei simulatori puri, e ispirandosi a questa recente corrente di ibridazione estetica, che *Bloody Shift* si propone di innovare il mercato: sostituendo il ritmo pacato tipico del *cozy gaming* con una progressione incalzante e calando le rassicuranti meccaniche di pulizia all'interno di un hotel infernale, ostile e imprevedibile.

2.2 Studio dei Competitor

Per posizionare correttamente *Bloody Shift* all'interno del mercato e comprenderne il valore innovativo, risulta fondamentale analizzare i titoli che hanno definito e successivamente fatto evolvere il sottogenere dei simulatori di pulizia a tema macabro. Lo studio dei *competitor* non ha uno scopo puramente nozionistico, ma mira a decostruire le scelte di *game design*, il ritmo della progressione (*pacing*) e l'approccio narrativo adottato da altre produzioni di rilievo.

A tal fine, l'indagine si concentra su tre opere chiave che delineano l'evoluzione concettuale del genere: *Viscera Cleanup Detail*, pioniere indiscusso che ha codificato le meccaniche fondanti della detersione virtuale; *Crime Scene Cleaner*, che ha recentemente integrato una struttura narrativa e di progressione più marcata; e infine *The WereCleaner*, un efficace esempio di ibridazione meccanica. L'analisi critica di questi tre titoli permette di estrapolare le pratiche virtuose da cui *Bloody Shift* trae ispirazione, evidenziando al contempo le frizioni ludiche e i fisiologici cali di ritmo che il progetto si propone di superare.

2.2.1 Il capostipite: Viscera Cleanup Detail

Sviluppato dal team indipendente RuneStorm, *Viscera Cleanup Detail* è universalmente riconosciuto come il pioniere del genere dei simulatori di pulizia a tema macabro [6]. Rilasciato originariamente in *Early Access*, il titolo si propone come una parodia esplicita dei cruenti sparatutto in prima persona (*FPS*) a tema fantascientifico. Invece di combattere minacce aliene, il giocatore viene calato nei panni di un semplice inserviente spaziale, incaricato di ripulire le stazioni orbitanti dai cadaveri, dal sangue e dai bossoli lasciati in seguito al passaggio dell'eroe di turno [6].

L'opera sovverte brillantemente i canoni del genere *horror* e d'azione: decontestualizzando gli scenari tipici degli *FPS* e obbligando l'utente a svolgere mansioni banali in un ambiente devastato, il gioco genera una forma di umorismo macabro senza la necessità di ricorrere a una narrazione esplicita. Dal punto di vista ludico, il *core loop* ruota attorno all'uso di un mocio, di secchi d'acqua (forniti dai distributori *Slosh-O-Matic*) e di un inceneritore per smaltire i resti, il tutto arricchito da una forte componente *sandbox* multigiocatore [6].

Tuttavia, è proprio nell'implementazione di queste meccaniche che emergono le maggiori criticità in termini di ritmo e accessibilità. La gestione della fisica degli oggetti risulta spesso punitiva: urtare accidentalmente un secchio d'acqua sporca, far cadere resti umani o semplicemente calpestare una superficie non ancora detersa provoca la diffusione di nuove macchie ematiche, vanificando i progressi del giocatore e costringendolo a ripetere intere sezioni di pulizia. La critica specializzata ha evidenziato come questa rigidità trasformi l'esperienza in un *grind* metodico ed estenuante, rendendo il completamento dei livelli un compito gravoso ed eccessivamente dilatato, specialmente nelle sessioni in giocatore singolo [7].

È proprio dall'analisi di questi limiti fisiologici che *Bloody Shift* prende le distanze dal suo predecessore spirituale. Pur mutuando l'idea di base della pulizia grottesca, il progetto descritto in questo elaborato rimuove deliberatamente le meccaniche punitive legate alla fisica dei liquidi (come i secchi rovesciabili) e snellisce le procedure di smaltimento, barattando il realismo metodico di *Viscera* in favore di un *pacing* nettamente più veloce, arcade e meno frustrante per l'utente singolo.

Figura 2.1: Key art ufficiale di *Viscera Cleanup Detail*.

2.2.2 L'evoluzione narrativa: Crime Scene Cleaner

Se *Viscera Cleanup Detail* ha gettato le basi meccaniche del genere, *Crime Scene Cleaner* ne rappresenta una significativa evoluzione strutturale, avendo dimostrato come l'integrazione di una solida impalcatura narrativa possa trasformare la ripetitività intrinseca dei simulatori in un'esperienza coinvolgente. Il titolo mette il giocatore nei panni di un padre che, per far fronte alle gravose cure mediche della figlia, si trova costretto a svolgere del "lavoro sporco" per conto della malavita organizzata, ripulendo meticolosamente efferate scene del crimine [8].

A differenza dei suoi predecessori, il titolo introduce una progressione strutturata mediante un vero e proprio albero delle abilità (*skill tree*) che permette all'utente di accumulare punti esperienza e investirli nel potenziamento del proprio arsenale [9]. Questa meccanica, unita alla possibilità di sottrarre oggetti di valore dalle case delle vittime, trovare indizi per ricostruire la storia dell'omicidio [8] e alternare attrezzi specifici in base alla fragilità o all'estensione della superficie da trattare (mocio, spugna, idropulitrice), eleva il simulatore verso le dinamiche tipiche di un *mystery thriller* interattivo [9]. La critica specializzata ha elogiato questa direzione, sottolineando come la combinazione di una scrittura ironica e di un sistema di progressione stratificato trasformi un compito di pura *drudgery* digitale in una sfida logica gratificante per l'utente, sebbene permangano limiti legati alla rigiocabilità a lungo termine e all'assenza di modalità cooperative [10].

Tuttavia, nonostante l'aggiunta di elementi RPG e narrativi, *Crime Scene Cleaner* rimane ancorato a un contesto e a un'estetica prettamente realistici, impostando un ritmo di gioco ponderato che richiede un'esplorazione meticolosa e paziente dello scenario [9]. È in questo solco che si inserisce la divergenza progettuale di *Bloody Shift*. Mentre il titolo in esame punta sul realismo cupo e su un'indagine quasi investigativa, *Bloody Shift* predilige un approccio volutamente surreale e *arcade*: l'atmosfera tesa e drammatica della malavita lascia spazio all'assurdo di un hotel infernale, dove la progressione non passa per la cautela investigativa, ma per il superamento di sfide frenetiche e caotiche che esasperano il concetto di pulizia fino al grottesco.

Figura 2.2: Key art ufficiale di *Crime Scene Cleaner*.

2.2.3 Ibridazione di genere: The WereCleaner

Il percorso di decostruzione dei simulatori culmina con l'esperienza offerta da *The WereCleaner*, un titolo indie rilasciato nel 2024 che porta il concetto di "ibridazione" ai suoi massimi livelli [11]. Abbandonando la finta serietà delle indagini forensi o i ritmi flemmatici dei titoli convenzionali, il gioco si presenta come una "stealth-comedy" [12] in cui l'utente controlla Kyle, un inserviente aziendale che, costretto a turni di lavoro notturni e non retribuiti, si trasforma in un licantropo [13, 11].

L'innovazione progettuale di *The WereCleaner* risiede nell'aver innestato le meccaniche tipiche dei simulatori di pulizia all'interno di un ecosistema popolato da personaggi non giocanti (NPC) [13]. Il giocatore è chiamato a detergere i pavimenti dell'ufficio utilizzando strumenti convenzionali (come idropulitrici e aspirapolveri) [13], ma deve contemporaneamente gestire i pattern di movimento dei colleghi per non essere scoperto [13]. Se l'utente viene individuato, perde il controllo degli istinti del protagonista e divora il testimone [11]. Sebbene la critica di settore abbia elogiato il tono ironico e la critica satirica allo sfruttamento aziendale [11], è sul piano del *game design* che il titolo brilla: uccidere un NPC non comporta un convenzionale *Game Over*, ma genera una nuova, copiosa macchia di sangue e resti organici che il giocatore è costretto a ripulire in tempo record prima dell'arrivo di altri colleghi [13]. Questo *loop* genera un innalzamento vertiginoso della tensione, trasformando un atto ordinario (pulire) in una lotta per la sopravvivenza [13].

L'approccio caotico, la presenza di minacce attive (gli NPC) [12] e l'ironia grottesca di *The WereCleaner* [11] rappresentano la massima fonte di ispirazione per lo sviluppo di *Bloody Shift*. Proprio come il licantropo Kyle, il protagonista del progetto in esame non è un eroe, ma un lavoratore sottopagato costretto ad arrangiarsi in condizioni ostili. *Bloody Shift* assimila questa lezione di *game design* implementando all'interno dell'hotel infernale un sistema di NPC dinamici (i VIP) capaci di infastidire l'utente e generare ulteriore disordine, confermando che l'introduzione di un elemento di disturbo imprevedibile è la chiave per mantenere elevata la *retention* all'interno del genere *cleaning simulator*.

Figura 2.3: Key art ufficiale di *The WereCleaner*.

Capitolo 3

Metodologia e Game Design

A valle dell'analisi dei titoli concorrenti condotta nel capitolo precedente, questo capitolo si concentra sulla progettazione concettuale e strutturale di *Bloody Shift*. Verranno qui definite le fondamenta del *game design*, delineando le scelte stilistiche, narrative e meccaniche ideate per rispondere alle criticità emerse nello Stato dell'Arte, quali la ripetitività e l'eccessiva dilatazione dei ritmi di gioco. Nelle sezioni seguenti saranno illustrati il *core loop* principale, le logiche di interazione e l'architettura ad alto livello dell'applicativo. Questa fase di formalizzazione progettuale rappresenta uno snodo cruciale per tradurre gli intenti creativi in requisiti software definiti, ponendo le basi logiche per la successiva fase di implementazione tecnica all'interno del motore grafico.

3.1 Analisi dell'utenza e definizione dei requisiti

Durante la fase preliminare dello sviluppo, la definizione del profilo utente e del *target* di riferimento non si è basata su una nuova indagine di mercato indipendente, bensì sul trasferimento di *know-how* e sull'analisi dei dati interni forniti dal *team* direttivo dello studio ospitante. Questa fase di elicitazione dei requisiti ha permesso di allineare la produzione alle reali esigenze del mercato PC sulla piattaforma Steam, sfruttando l'esperienza pregressa dell'azienda nel settore.

Dalle direttive fornite, è emerso che l'utenza tipica dei *cleaning simulator* indipendenti è alla ricerca di un equilibrio tra il senso di appagamento derivante dal completamento di mansioni ripetitive e la necessità di un contesto capace di mitigare la monotonia dell'azione.

Sulla base di queste informazioni aziendali, sono stati delineati i requisiti cardine per la progettazione del *game design* e per la selezione dello *stack* tecnologico.

3.1.1 Requisiti di Game Design e innovazioni meccaniche

Dallo studio dei limiti intrinseci al genere di riferimento, il *team* ha formalizzato le seguenti soluzioni progettuali:

- **Ibridazione del genere:** per differenziare il prodotto, l'utenza *target* è stata individuata in una nicchia di giocatori propensa ad atmosfere tensive. Questo

ha guidato la scelta verso un'estetica horror-grottesca, scartando il realismo asettico tipico dei simulatori tradizionali.

- **Ritmo della progressione:** per evitare la frustrazione derivante da sessioni di pulizia eccessivamente lunghe e prive di sbocchi narrativi, è stato richiesto di inserire elementi di intermezzo (come il sistema di *quest*) e di calibrare le condizioni di vittoria su soglie di completamento parziali anziché totali.
- **Ergonomia e riduzione del backtracking (Il Carrello):** l'analisi dei titoli preesistenti ha evidenziato come il continuo spostamento a piedi tra l'area da pulire e i punti di approvvigionamento (es. i distributori di secchi o l'inceneritore) generi tempi morti che diluiscono il coinvolgimento. Per risolvere questa criticità, è stata concepita la meccanica del carrello portatile. Inedito per il genere in questa forma, il carrello funge da inventario e base operativa mobile, consentendo al giocatore di mantenere un flusso di lavoro ininterrotto e dinamico.

3.1.2 Requisiti tecnici e scelte architetturali

La realizzazione tecnica del *concept* ha imposto l'adozione di strumenti di sviluppo adeguati alle dimensioni del *team* e ai vincoli prestazionali:

- **Scelta del motore grafico (Unreal Engine 5.6.1):** la necessità di creare un'atmosfera immersiva basata fortemente sui contrasti chiaroscurali ha orientato la scelta su Unreal Engine 5.6.1. Il motore nativo di Epic Games garantisce, infatti, una gestione avanzata dell'illuminazione globale e della resa dei materiali (PBR), fornendo al contempo *tool* integrati per la navigazione dell'intelligenza artificiale (NavMesh).
- **Sviluppo logico tramite Visual Scripting:** considerando le dimensioni ridotte del comparto di programmazione (due sviluppatori) e la necessità di iterazioni rapide sulle meccaniche, si è optato per l'utilizzo esclusivo dei *Blueprint* al posto del codice sorgente nativo in C++. Questo approccio ha agevolato la prototipazione veloce del *gameplay*, richiedendo tuttavia l'applicazione di rigorosi *pattern* di disaccoppiamento per mantenere il codice visivo pulito e manutenibile.
- **Accessibilità hardware:** considerando che una vasta fetta dell'utenza Steam dispone di macchine di fascia media (come confermato dai dati pubblici dello *Steam Hardware & Software Survey* [14]), il titolo necessitava di un'architettura logica e grafica ottimizzata, capace di preservare il *framerate* senza richiedere calcoli computazionali eccessivamente onerosi.

3.2 Concept e visione progettuale

"Bloody Shift" si propone di innovare le dinamiche tradizionali dei simulatori di pulizia, calando il giocatore nei panni di un inserviente assegnato a una struttura alberghiera dalle connotazioni infernali. Questa sezione analizza la visione alla base del progetto: dalla sintesi dell'impianto narrativo alla definizione delle regole che strutturano il *gameplay* e l'atmosfera.

3.2.1 Premessa narrativa e progressione

L'esperienza interattiva è circoscritta all'interno di un hotel fatiscente, in cui l'obiettivo del protagonista è l'ottenimento della redenzione attraverso l'assolvimento delle proprie mansioni lavorative. Il percorso di esplorazione si sviluppa progressivamente dalle aree comuni (raggiungibili al termine della fase introduttiva di *onboarding*) verso tre stanze esclusive, abitate da particolari ospiti.

L'accesso a queste aree chiave, essenziali per la conclusione del gioco, richiede il completamento di specifiche missioni (*quest*) volte al recupero di oggetti smarriti. Per attivare queste missioni, il sistema richiede al giocatore di soddisfare prima una quota prestabilita di interazioni ambientali (come la rimozione di macchie o rifiuti), incentivando l'operatività generale.

Per favorire il ritmo di gioco e mitigare la ripetitività riscontrabile nei titoli *competitor*, la condizione di vittoria non esige una bonifica totale dell'ambiente. Il *trigger* della sequenza finale si innesca al completamento delle *quest* principali e al raggiungimento di specifici target quantitativi (es. numero di macchie deterse e rifiuti smaltiti). Raggiunti tali obiettivi, la narrazione si chiude con una transizione spaziale: il giocatore accede a una nuova area visivamente celestiale, solo per scoprire di essere intrappolato in un ciclo di lavoro perpetuo.

3.2.2 Core Loop e meccaniche generali

Il *Core Loop* del gioco si fonda su un set di meccaniche principali, strutturate per costituire il fulcro dell'esperienza interattiva offerta da "Bloody Shift".

Il pilastro della navigazione e della gestione delle risorse è la meccanica del carrello. Il giocatore ha la facoltà di agganciarsi o sganciarsi dalla struttura (*cart*) in qualsiasi punto della mappa. Questa impostazione elimina il vincolo restrittivo di un trascinamento continuo, pur incentivandone l'utilizzo strategico: mantenere la prossimità con il carrello ottimizza i tempi di percorrenza per l'interazione con l'inventario. Il carrello funge da base operativa portatile, ospitando i tre strumenti di lavoro (il mocio, lo scacciamosche e lo spolverino) e il secchio per la raccolta dei rifiuti. L'intero ambiente di gioco è stato dimensionato a livello spaziale per risultare fluidamente navigabile anche mantenendo il vincolo di aggancio.

Le interazioni di pulizia, che definiscono il ciclo operativo primario, si suddividono in quattro categorie:

- **Pulizia delle macchie di sangue:** le superfici dell'hotel presentano macchie ematiche che richiedono l'utilizzo del mocio per la rimozione. Al superamento di una soglia prestabilita di utilizzi, il mocio subisce un degrado visivo (tramite cambio di texture) e inverte la sua funzione, espandendo le macchie. L'utensile necessita quindi di essere risciacquato nel secchio dell'acqua (`BP_WaterBucket`) per ripristinarne l'efficacia;
- **Raccolta e smaltimento dei rifiuti:** al giocatore è richiesto di individuare, raccogliere singolarmente e depositare nel secchio del carrello i vari scarti disseminati per le stanze. Al riempimento della sacca dell'immondizia, questa deve essere prelevata e conferita nell'inceneritore situato nel corridoio principale;
- **Disinfestazione dagli scarafaggi:** tramite lo scacciamosche, l'utente deve neutralizzare i PNG infestanti. Questa meccanica innesca un'interazione a catena: l'insetto eliminato genera una nuova macchia sul pavimento, richiedendo un successivo intervento con il mocio;
- **Rimozione delle ragnatele:** il giocatore impiega lo spolverino per ripulire le aree ostruite, localizzate prevalentemente negli angoli e lungo i soffitti.

Figura 3.1: Il carrello delle pulizie completo dell'equipaggiamento in dotazione al giocatore: mocio, spolverino, scacciamosche e secchio.

Un ulteriore elemento del sistema di navigazione è l'interazione con gli accessi. Le porte possono essere azionate indipendentemente dallo stato di aggancio al carrello. Per prevenire compenetrazioni poligonali (*clipping*) e collisioni indesiderate, le cerniere virtuali sono istruite per ruotare in direzione opposta rispetto al vettore di posizione

del giocatore. Inoltre, è stato integrato un *feedback* visivo: in caso di interazione con una porta bloccata, o nel tentativo di chiusura qualora l'attore occupi l'area di collisione della soglia, la *mesh* riproduce una micro-oscillazione per notificare l'invalidità dell'azione.

Infine, l'esplorazione ambientale è supportata dalla gestione dinamica dell'illuminazione tramite interruttori a parete, azionabili esclusivamente quando il personaggio è svincolato dal carrello. L'architettura di queste dinamiche forma una struttura gerarchica schematizzabile tramite un *Onion Diagram* (Figura 3.2).

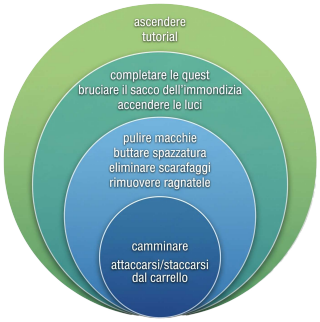

Figura 3.2: *Onion Diagram* illustrante la gerarchia delle meccaniche di gioco: dal *core loop* centrale agli strati di interazione ambientale, fino al livello esterno della progressione narrativa.

3.2.3 Visione artistica e Atmosfera

Per posizionare "Bloody Shift" in discontinuità rispetto alle convenzioni estetiche del genere, la direzione artistica ha collocato l'azione all'interno di un'infrastruttura dalle tinte cupe, fatiscenti e sinistre.

Figura 3.3: Visualizzazione in modalità *Unlit* degli *asset* (ossa e viscere) impiegati come elementi di scarto.

La *palette* cromatica si articola attorno a toni caldi ma caratterizzati da una forte desaturazione. Negli ambienti predominano il rosso, impiegato per la carta da parati e per le tracce ematiche, e il marrone scuro del legno che riveste pavimentazioni e finiture. A supporto dell'impianto visivo contribuisce la tipologia di sporcizia implementata: i comuni rifiuti alberghieri sono stati integrati con elementi anatomici (ossa e viscere) distribuiti lungo le aree esplorabili, conferendo al compito di pulizia un tono grottesco.

Il sistema di illuminazione concorre in modo significativo alla percezione dello spazio: le luci dei corridoi mantengono una bassa intensità e integrano un effetto procedurale di *flickering* (sfarfallio), restituendo un'atmosfera instabile. Anche negli ambienti interni, l'accensione degli interruttori è calibrata per mantenere diffuse zone d'ombra.

A livello di *post-processing*, è stato applicato un filtro grafico che emula il degrado di una registrazione analogica (VHS), introducendo *film grain* e lievi distorsioni visive che amplificano l'estetica inquietante del progetto.

Infine, il *Sound Design* è stato strutturato per affiancare l'esplorazione visiva con un tappeto sonoro a bassa frequenza, caratterizzato da un respiro continuo di sottofondo. A questo *layer* ambientale si sovrappongono effetti acustici spazializzati, tra cui il fruscio dei PNG infestanti, la frizione delle cerniere e il tonfo sordo riprodotto dalla chiusura delle porte.

3.3 User Experience (UX) e immersione

La fase iniziale di un'esperienza videoludica, comunemente definita *onboarding*, rappresenta uno snodo critico per la fidelizzazione dell'utente e per l'apprendimento organico dei sistemi di gioco. In "Bloody Shift", la necessità di istruire il giocatore sulle meccaniche fondamentali si scontrava con il requisito cardine del progetto: mantenere intatta la tensione atmosferica e il senso di isolamento. Per risolvere questa criticità, il *team* ha focalizzato la progettazione sull'ottimizzazione della *User Experience* (UX), ricercando soluzioni di *design* invisibili e integrate nell'ambiente.

3.3.1 Filosofia del diegetic tutoring

L'approccio metodologico adottato per l'insegnamento delle meccaniche si distacca nettamente dalle soluzioni tradizionali del settore. Nei simulatori convenzionali, l'apprendimento è spesso delegato a finestre di dialogo esplicative, testi in sovrimpressione o menù di pausa che interrompono l'azione. Sebbene efficaci dal punto di vista didattico, questi elementi artificiali d'interfaccia (HUD) incidono negativamente sul ritmo di gioco e causano la rottura della sospensione dell'incredulità, un fattore fatale per un titolo dalle tinte *horror*.

Il progetto persegue pertanto una rigorosa filosofia di *diegetic tutoring* [15]. L'utente viene guidato e istruito attraverso stimoli visivi, sonori e spaziali che appartengo-

no coerentemente al mondo di gioco (elementi diegetici). L'obiettivo è favorire l'assimilazione delle regole tramite la deduzione logica e la reazione naturale agli stimoli ambientali, riducendo al minimo il carico cognitivo derivante dalla lettura di istruzioni.

Un esempio pratico di questa filosofia è la gestione dell'attenzione (*focus*). Invece di utilizzare indicatori grafici (come frecce o icone lampeggianti a schermo) per segnalare un nuovo obiettivo, il *game design* prevede transizioni cinematiche controllate. In momenti chiave del tutorial, il sistema inibisce temporaneamente l'input di movimento del giocatore per orientare fluidamente la sua visuale verso un elemento di interesse (es. una porta che si schiude o un accumulo di spazzatura accompagnato da messaggi scritti col sangue sui muri). Questo espediente garantisce che l'utente recepisca l'informazione fondamentale per proseguire, percependo l'evento come parte della narrazione (*environmental storytelling*) e non come un'istruzione di sistema.

3.3.2 Orientamento ambientale tramite illuminazione

In assenza di indicatori di navigazione espliciti, l'impiego strategico dell'illuminazione ha assunto il duplice ruolo di veicolo atmosferico e di principale strumento di *UX design* per l'orientamento implicito dell'utente. Le dinamiche chiaroscurali sono state orchestrate per fungere da mappa invisibile, guidando il giocatore attraverso due metodologie distinte:

- **Illuminazione sequenziale:** all'inizio dell'esperienza, l'ambiente si presenta in condizioni di marcata oscurità, limitando il campo visivo alla sola area dell'ascensore. Questa restrizione spaziale spinge psicologicamente l'utente a concentrarsi sul perimetro immediato, limitando il senso di smarrimento iniziale. Man mano che il giocatore completa i micro-obiettivi richiesti, le fonti di luce a soffitto si attivano progressivamente lungo i corridoi. Questa tecnica crea una scia luminosa che comunica implicitamente la direzione sicura da seguire, assecondando la naturale tendenza umana a muoversi verso le aree illuminate e scoraggiando l'esplorazione prematura di zone oscure.
- **Luminescenza e Affordance degli oggetti:** uno dei problemi principali nell'eliminazione dell'HUD è comunicare al giocatore quali oggetti siano interattivi (*affordance*) e in quale momento esatto del gioco debbano essere utilizzati. Per risolvere questo problema, si è scelto di implementare un sistema di luminescenza diegetica. Quando il copione richiede l'utilizzo di uno specifico utensile (come il mocio, il secchio dell'acqua o lo scacciamosche), il materiale dell'oggetto stesso inizia a emettere un flebile bagliore. Questo contrasto visivo nel buio attira immediatamente l'occhio del giocatore verso l'inventario presente sul carrello, comunicando l'esatta disponibilità dello strumento senza ricorrere a *outline* artificiali o testi a comparsa. L'oggetto stesso "suggerisce" il suo utilizzo, fondendo l'interfaccia utente con la direzione artistica.

3.4 Protocollo di User Testing e validazione

Al fine di validare l'efficacia delle soluzioni architetturali e di *game design* implementate, il progetto è stato sottoposto a una fase di *User Testing* al termine del ciclo di sviluppo principale. Questa sezione descrive la metodologia di indagine adottata, delineando il contesto della somministrazione, il campione coinvolto e la struttura dello strumento di raccolta dati (Google Forms), rimandando l'analisi quantitativa e qualitativa dei risultati ai capitoli successivi.

3.4.1 Contesto, campione e tempistiche

Le sessioni di *playtest* sono state condotte durante la fase di *Beta release* del videogioco, adottando un approccio ibrido strutturato su due modalità complementari:

- **Testing in presenza (Supervisionato):** una parte significativa delle rilevazioni è stata effettuata durante l'evento fieristico *Svilupparty*. Questo contesto ha permesso l'osservazione diretta del comportamento degli utenti, consentendo agli sviluppatori di annotare esitazioni, reazioni emotive e approcci esplorativi non filtrati.
- **Testing da remoto (Non supervisionato):** parallelamente, è stata distribuita una *build* compilata per permettere ad altri utenti di eseguire il gioco in autonomia sulle proprie macchine. Questo ha consentito di simulare le reali condizioni di fruizione dell'utenza e di testare la stabilità software su hardware eterogenei.

Il campione selezionato per l'indagine si è rivelato volutamente eterogeneo. Sebbene il nucleo fosse composto dal *target* primario (giocatori PC abituati all'uso di mouse e tastiera), il test ha coinvolto anche utenti provenienti dall'ecosistema *console* e individui del tutto estranei al *medium* videoludico (non-giocatori). Questa scelta metodologica ha permesso di "stressare" i sistemi di *onboarding* e di valutare l'intuitività dell'interfaccia invisibile (*diegetic tutoring*) su soggetti caratterizzati da diversi gradi di alfabetizzazione digitale.

3.4.2 Struttura dell'indagine e obiettivi

Al termine della sessione di gioco (sia in presenza che da remoto), a ciascun partecipante è stato richiesto di compilare un questionario strutturato tramite la piattaforma *Google Forms*. Il modulo è stato concepito per raccogliere sia metriche quantitative (tramite scale di valutazione e risposte multiple) sia dati qualitativi (attraverso campi a risposta aperta).

L'indagine è stata suddivisa in cinque macro-aree logiche: profilazione dell'utente, usabilità e *onboarding*, valutazione delle meccaniche, analisi tecnica/estetica e, infine, potenziale di mercato. L'obiettivo primario è stato quello di validare l'efficacia delle

soluzioni di *game design* implementate (come il *diegetic tutoring* e la meccanica del carrello) e di misurare la stabilità del software su configurazioni hardware eterogenee.

3.4.3 I quesiti e la giustificazione metodologica

Per garantire il massimo rigore nell'analisi dei dati, il questionario somministrato ai *playtester* è stato strutturato attorno a specifici intenti investigativi. I quesiti sono stati sintetizzati e organizzati per aree di pertinenza, ciascuna associata a una precisa scelta di *design* precedentemente trattata:

- **Profilazione del campione e benchmarking:** le domande introduttive sono state formulate per stabilire il *background* dell'utente (piattaforma preferita, ore di gioco settimanali, conoscenza dei titoli *competitor*). Questa fase è risultata fondamentale per la pesatura dei dati successivi, permettendo di contestualizzare eventuali difficoltà e di valutare la reale divergenza percepita rispetto ai simulatori classici.
- **Accessibilità, controlli e Onboarding:** un blocco sostanzioso dell'indagine è stato dedicato alla validazione della filosofia di *diegetic tutoring*. In assenza di un HUD esplicito e di finestre di testo, era essenziale verificare se le transizioni cinematiche della telecamera e l'attivazione della luminescenza procedurale degli oggetti (`BP_LightingComponent`) risultassero sufficientemente chiare per istruire l'utente, misurando eventuali picchi di frustrazione o disorientamento iniziale.
- **Valutazione del Core Loop e delle Meccaniche:** questa sezione ha indagato il cuore dell'esperienza interattiva. L'obiettivo primario è stato confermare che l'implementazione del carrello mobile (la principale innovazione di *game design* del progetto) risultasse fluida e strategicamente utile. Contestualmente, è stato richiesto di valutare il livello di appagamento derivante dal ciclo di pulizia e l'equilibratura della sfida proposta, per scongiurare l'insorgere di noia o monotonia.
- **Comparto Tecnico ed Estetico:** dal punto di vista ingegneristico, è stato vitale raccogliere dati empirici sulla stabilità del software su macchine eterogenee, richiedendo *feedback* diretti su cali di *framerate* o rilevamento di *bug* e compenetrazioni. Sotto il profilo artistico, i quesiti hanno misurato l'efficacia del comparto visivo e sonoro nel generare l'impatto tensivo e claustrofobico ricercato.
- **Potenziale di mercato e validazione globale:** il questionario si conclude con metriche standardizzate per la valutazione del potenziale commerciale. Tramite scale numeriche e campi aperti, si è indagato l'apprezzamento complessivo, la propensione all'acquisto e si è lasciata l'opportunità di segnalare *pain points* imprevisti o suggerimenti per lo sviluppo di funzionalità future.

Capitolo 4

Implementazione

A valle della definizione concettuale e delle direttive di *game design* esaminate nel capitolo precedente, la trattazione si sposta ora sulla fase di implementazione tecnica del caso di studio. Questo capitolo illustra l'architettura software di *Bloody Shift*, realizzata in Unreal Engine 5.6.1. Verranno inizialmente descritte le metodologie di lavoro collaborativo e le scelte logiche adottate tramite *Blueprint*, con particolare attenzione alle strategie orientate a contenere le dipendenze e ridurre il carico prestazionale in esecuzione (tramite l'uso di *Interfaces*, *Event Dispatchers* e *timer* personalizzati). Successivamente, si analizzerà lo sviluppo ingegneristico degli *Actor* interattivi principali e dell'Intelligenza Artificiale. Infine, si esplorerà l'integrazione tecnica del comparto visivo e sonoro, culminando nell'analisi del `BP_ScriptManager`, il sistema centralizzato preposto all'orchestrazione degli eventi narrativi e del flusso di gioco.

4.1 Architettura generale del progetto

Una volta terminata la fase di *brainstorming* e definiti il *concept* e il *core loop* del gioco, il team ha avviato lo sviluppo tecnico all'interno di Unreal Engine 5.6.1. La programmazione delle meccaniche si è basata su una pianificazione strutturata tra i due programmatori. Si è optato per una divisione mirata del carico di lavoro, con il duplice obiettivo di snellire i tempi di produzione e limitare l'insorgenza di conflitti sui file binari durante le fasi di *merge* dei *Blueprint* tramite il *client* GitKraken.

4.1.1 Architettura logica, modularità e *ownership* dei componenti

La struttura ingegneristica dell'opera è stata concepita seguendo rigorosi paradigmi di disaccoppiamento (*decoupling*). Il sistema si articola in macro-aree funzionali indipendenti: il *Game Management* (che include il `BP_MainGameInstance` per la persistenza dei dati e i gestori `BP_QuestManager` e `BP_ScriptManager`), il *Core Player* (con il `BP_CharacterController` e il carrello `BP_Cart`), e infine le librerie di *Actor* interattivi (come i *Cleaning Tools*, le macchie procedurali e il sistema di smaltimento dei rifiuti).

Per consentire la comunicazione tra i sistemi, la base di codice si affida prevalentemente a *Blueprint Interfaces* (BPI) ed *Event Dispatchers*. Il controllore del giocatore,

ad esempio, implementa la `BPI_Player`, esponendo funzioni pubbliche per il blocco della visuale o l'accesso ai componenti (*LockVision*, *GetCapsuleComponent*), che possono essere richiamate da qualsiasi altro *Actor* senza che quest'ultimo conosca l'identità specifica della classe bersaglio. Analogamente, gli oggetti manipolabili condividono la `BPI_Interact` e comunicano i propri cambiamenti di stato tramite *Dispatcher* asincroni (come `OnObjectCleaned` o `OnTrashBinFull`), ai quali i *Manager* di sistema si iscrivono in fase di *BeginPlay*.

Sotto il profilo organizzativo, l'impiego massiccio delle interfacce ha facilitato la compartimentazione dei compiti tra i membri del team, permettendo di sviluppare i sistemi in parallelo stipulando unicamente i "contratti" delle chiamate pubbliche. All'interno di questa architettura, lo scrivente ha contribuito in maniera diretta alla progettazione e all'implementazione dei seguenti rami logici: l'architettura interattiva del carrello mobile (`BP_Cart`), la programmazione dei comportamenti degli scarafaggi, la logica di funzionamento delle porte, lo sviluppo dei materiali dinamici, l'integrazione pratica del comparto audio e la stesura dell'intero sistema di tutorial orchestrato dal `BP_ScriptManager`. Parallelamente, l'altro programmatore si è concentrato sulla codifica del personaggio giocante (`BP_CharacterController`), la gestione modulare degli strumenti di pulizia, lo sviluppo dell'infrastruttura delle missioni (`BP_QuestManager`) e la realizzazione di tutta l'Interfaccia Utente (UI), comprensiva del sistema di impostazioni. Questa suddivisione delle responsabilità (*ownership*) ha garantito una conduzione del progetto snella e priva di colli di bottiglia logistici.

4.1.2 Integrazione dei sistemi sviluppati in parallelo dal team

L'impiego di un sistema di controllo della versione (VCS) rappresenta uno strumento fondamentale per la gestione di un progetto sviluppato in parallelo. Il team ha pertanto adottato Git per il tracciamento delle modifiche, avvalendosi di GitKraken [16] come *client* grafico. Questa interfaccia visiva ha permesso di monitorare lo storico dei *commit* e di organizzare i vari *branch* (rami di sviluppo) in modo sistematico.

Tuttavia, lo sviluppo collaborativo all'interno di Unreal Engine presenta una specificità tecnica legata alla struttura dei suoi *asset*. A differenza del codice sorgente tradizionale (come il C++), che è basato su testo puro e permette fusioni riga per riga, il motore di Epic Games archivia la maggior parte degli elementi (Blueprint, Mappe, Materiali) sotto forma di file binari proprietari, identificati principalmente dall'estensione `.uasset` [17].

A causa di questa architettura binaria, i file di Unreal Engine non supportano le normali operazioni di *merge*. Qualora due membri del team modifichino simultaneamente il medesimo file `.uasset`, il sistema di versionamento non è in grado di unire le variazioni. Al momento della sincronizzazione si genera pertanto un conflitto, il quale può costringere alla sovrascrittura (con potenziale perdita) del lavoro svolto da uno dei due sviluppatori.

Per mitigare l'incidenza di tali conflitti in fase di merge, lo sviluppo delle logiche è stato strutturalmente compartimentato. Il primo programmatore si è focalizzato su:

- **Personaggio Giocante (*Player Character*):** lo sviluppo dell'architettura di *Bloody Shift* ha preso avvio dalla definizione del protagonista giocabile (implementato all'interno della classe `BP_CharacterController`), l'entità attraverso cui l'utente esplora e interagisce con il mondo virtuale. Data la prospettiva in prima persona (*First-Person*), si è optato per un approccio strutturale mirato a ridurre il carico di elaborazione: l'entità è priva di *Skeletal Mesh* [18] per il corpo, basandosi su un *Capsule Collider* per la gestione della fisica e della navigazione spaziale. Il punto di vista del giocatore è stato affidato a un componente *CineCamera* [19], scelto al fine di ottenere un controllo più preciso sui parametri ottici (come la profondità di campo e l'apertura del diaframma). Inoltre, è stato assegnato un componente *Point Light* con intensità e raggio tarati per supportare la visione in spazi bui, cercando di preservare l'atmosfera cupa prevista dal concept. Dal punto di vista logico, il *Blueprint* elabora due distinti sistemi di proiezione vettoriale (*Line Trace*) [20]: il primo viene eseguito in modo continuo per rilevare la prossimità di elementi interagibili e richiamare dinamicamente l'interfaccia utente (UI) a schermo; il secondo viene innescato selettivamente solo alla pressione del tasto di interazione, calcolando l'evento di collisione con l'oggetto puntato. Infine, la classe è dotata di un set di funzioni per l'interfacciamento con la *GameInstance* e le altre entità di gioco, fungendo da nodo per lo scambio dei dati di stato.

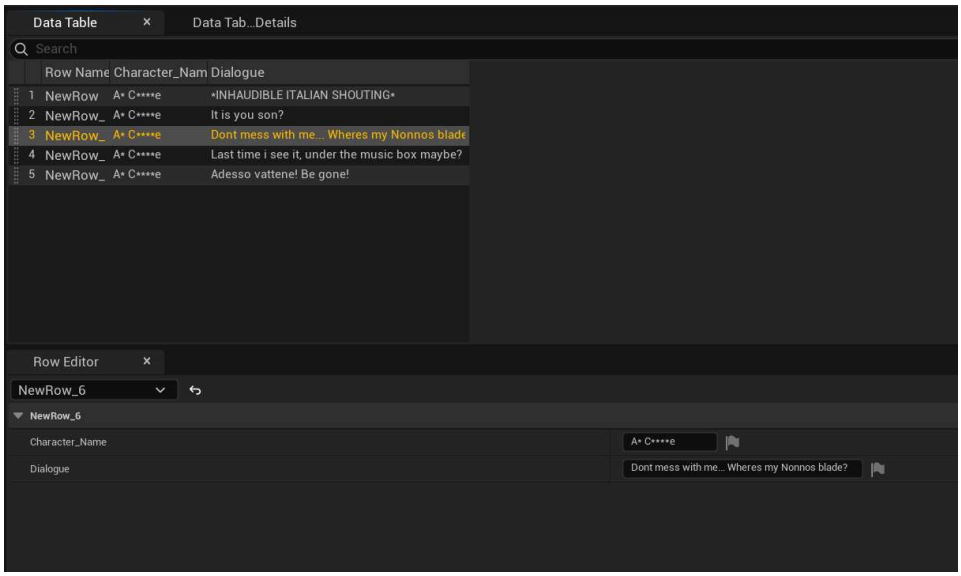

Figura 4.1: Struttura tabellare (*Data Table*) utilizzata per la gestione modulare dei dialoghi e degli stati delle missioni.

- **Sistema di Missioni (*Quest System*):** gestione del flusso degli incarichi speciali e dei relativi testi a schermo. Il programmatore ha strutturato un

sistema a stati per monitorare l'avanzamento delle missioni, subordinandone l'attivazione al raggiungimento di specifiche soglie di interazione ambientale. I dialoghi e i *feedback* testuali, associati ai vari stati della *quest*, sono stati organizzati in modo modulare tramite l'impiego di *User Struct* e archiviati all'interno di *Data Table* (Figura 4.1). Il sistema gestisce inoltre lo *spawn* a *runtime* degli oggetti di missione (`BP_QuestObject`) in punti prefissati, vincolandone l'istanziamento al momento in cui l'incarico viene formalmente avviato.

- **Gestione delle entità di pulizia:** programmazione delle logiche interattive per l'eliminazione dello sporco. Al fine di incrementare la varietà visiva, sono state realizzate nove varianti di macchie ematiche (`BP_BloodStain`) avvalendosi di *Decal Actor* [21] applicati proceduralmente sulle superfici. La gestione dei rifiuti è stata strutturata sfruttando il principio dell'ereditarietà [22]: è stata istituita una classe *Blueprint* padre (`BP_Trash`) da cui derivano gli specifici elementi scartabili. Queste entità sono programmate per distruggersi in risposta agli eventi di collisione con il *trash bin*. Completano il sistema il *Blueprint* dell'inceneritore, che sfrutta il rilevamento delle intersezioni (*overlap*) [23] per lo smaltimento dei sacchi della spazzatura, e le ragnatele, configurate per reagire alle collisioni con lo spolverino.

- **Strumenti di pulizia (*Cleaning Tools*):** in continuità logica con lo sviluppo delle entità interattive, sono stati progettati i tre utensili principali per la sanificazione degli ambienti (mocio, scacciamosche e spolverino). L'impiego di tali strumenti innesca un'animazione procedurale che ne interpola il movimento verso l'entità bersaglio, al fine di generare l'evento di collisione necessario per l'applicazione degli effetti visivi (come la graduale riduzione dell'opacità per le macchie ematiche, lo schiacciamento degli insetti e la distruzione delle ragnatele). Inoltre, per agevolare l'interazione da parte dell'utente, i *tools* implementano un componente *Box Collision* con un volume di tolleranza ampliato, con lo scopo di rendere il rilevamento dell'intersezione tra lo strumento e lo sporco più permissivo.

- **Interfaccia Utente (UI):** si è fatto ricorso ai *Widget Blueprint* [24] per lo sviluppo dell'interfaccia grafica, comprensiva dei menù di navigazione (*main menu*, *pause menu*, *final menu*) e dei *prompt* contestuali a schermo. Per la realizzazione strutturale dell'interfaccia, il programmatore ha sviluppato pulsanti, campi di testo e *slider* personalizzati (Figura 4.2), estendendo le funzionalità grafiche dei componenti nativi di Unreal Engine 5.6.1 per favorire l'allineamento con la direzione artistica del progetto.

- **Sistema di Impostazioni (*Settings*):** sviluppo dei moduli per la configurazione personalizzata dei parametri video (limitatore di *framerate*, risoluzione, qualità delle ombre e modalità finestra) e del comparto audio (suddiviso in

Figura 4.2: Visualizzazione del menù principale all'interno dell'ambiente UMG UI Designer, con pulsanti e *layout* personalizzati.

volume *master*, musica ed effetti sonori). Per consentire la persistenza di tali configurazioni durante i caricamenti e le transizioni tra i vari livelli, i dati sono stati archiviati e gestiti tramite la *GameInstance* [25], la classe di Unreal Engine deputata al mantenimento dello stato globale dell'applicazione.

4.1.3 Gestione dello stato globale e logiche di interfacciamento

L'architettura software di *Bloody Shift* è stata progettata adottando un approccio modulare per la gestione dello stato globale, demandando il controllo del flusso logico a specifiche classi manageriali che operano con logiche assimilabili al *pattern Singleton* [26]. L'infrastruttura di base poggia sull'interazione di tre componenti cardine.

In primo luogo, la *GameInstance* è stata impiegata come contenitore persistente primario. Essendo l'unica istanza a non subire il ciclo di distruzione (*garbage collection*) [27] e reinizializzazione durante i caricamenti tra le mappe, essa funge da nucleo di riferimento. Oltre a ospitare le preferenze relative ai *settings* audio e video, mantiene in memoria le variabili di stato necessarie a favorire la continuità e la coerenza dei dati di partita.

L'interfacciamento tra le intenzioni dell'utente e il *Pawn* controllato è gestito dalla classe *PlayerController* [28]. Per l'assegnazione e la decodifica dei comandi si è scelto di adottare l'*Enhanced Input System* [29], il *framework* nativo introdotto nelle recenti versioni del motore. Questa architettura offre un'alternativa alle associazioni dirette tipiche dei sistemi *legacy*: ogni potenziale comando logico (es. l'interazione o l'uso degli strumenti) è stato astratto e incapsulato all'interno di specifiche *Input Action* (IA). Tali azioni sono state successivamente mappate sui dispositivi periferici (tastiera e mouse) mediante un *Input Mapping Context* (IMC), come illustrato in Figura 4.3.

Questo livello di disaccoppiamento mira a favorire la flessibilità, orientando il sistema verso una maggiore modularità in vista di eventuali integrazioni.

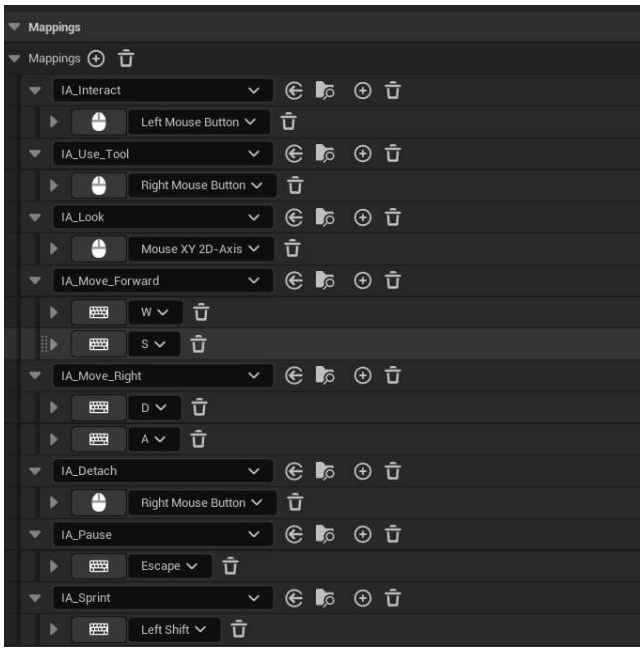

Figura 4.3: Configurazione dell'*Input Mapping Context* (IMC) per l'assegnazione delle *Input Action* ai dispositivi periferici.

Infine, a livello di singola mappa, la validazione delle regole e la definizione della struttura della sessione sono state affidate alla classe *GameMode* [30]. Con l'intento di strutturare adeguatamente le responsabilità (*Separation of Concerns*) e limitare il caricamento di logiche superflue, il progetto implementa due varianti distinte: una *GameMode* semplificata dedicata esclusivamente alla mappa del *Main Menu*, preposta all'inizializzazione dei *Widget Blueprint* dell'interfaccia; e una *GameMode* operativa, istanziata nei livelli giocabili, incaricata di orchestrare le regole specifiche del turno di lavoro.

4.2 Progettazione architetturale tramite Blueprint

La realizzazione delle logiche interattive e sistemiche di *Bloody Shift* è stata condotta integralmente attraverso il sistema di *Visual Scripting* nativo di Unreal Engine, denominato *Blueprint*. Sebbene tale strumento consenta una prototipazione rapida, lo sviluppo di un progetto stratificato richiede l'applicazione di consolidati principi di ingegneria del software. Per prevenire il deterioramento dell'architettura logica (comunemente noto come *spaghetti code*) e mitigare l'insorgere di colli di bottiglia prestazionali, la stesura dei nodi visivi è stata orientata verso paradigmi di modularità e disaccoppiamento. I paragrafi seguenti analizzano le direttive tecniche e

le soluzioni architetturali adottate con l'intento di favorire l'efficienza computazionale e l'ottimizzazione delle risorse in memoria.

4.2.1 Strategie di modularità e contenimento delle risorse: Blueprint Interfaces, Event Dispatchers e animazione procedurale

Secondo i dati dello *Steam Hardware & Software Survey*, 16 GB rappresentano la configurazione di memoria RAM più diffusa tra gli utenti che hanno partecipato all'indagine [14]. Questo dato ha motivato l'adozione di scelte architetturali orientate a mantenere il progetto compatibile anche con configurazioni hardware di fascia media. In particolare, il team ha cercato di limitare le dipendenze rigide tra i Blueprint, le operazioni eseguite inutilmente a ogni fotogramma e l'impiego di asset più complessi del necessario. Poiché il consumo effettivo di memoria non è stato misurato mediante strumenti di profiling, le soluzioni descritte di seguito devono essere considerate strategie preventive di progettazione e non ottimizzazioni prestazionali quantitativamente dimostrate.

- ***Blueprint Interfaces* e limitazione del *Casting*:** nel *Visual Scripting* di Unreal Engine, il metodo tradizionale per accedere alle funzioni o verificare il tipo di un altro *Blueprint* è l'utilizzo del nodo di *Cast* [31]. Sebbene l'operazione in sé non presenti criticità a livello di esecuzione computazionale, l'esecuzione di un *Cast* verso una classe personalizzata comporta una dipendenza in memoria nota come *hard reference* [32]. Ciò implica che, al caricamento in memoria della classe chiamante (ad esempio il *Player*), il motore grafico provveda a caricare preventivamente anche la classe bersaglio e il suo intero albero di dipendenze, incrementando il potenziale consumo di RAM. Per mitigare questo fenomeno, si è deciso di limitare l'utilizzo del *Casting* verso i *Blueprint* di gioco. L'eccezione applicata a tale regola ha riguardato il *Cast* verso le classi native o entità intrinsecamente persistenti (come *Pawn*, *GameMode* o *GameInstance*), per le quali il costo di caricamento aggiuntivo risulta marginale in quanto già residenti in memoria.

  Per far comunicare gli *asset* personalizzati mantenendo il disaccoppiamento, si è fatto ricorso alle *Blueprint Interfaces* [33]. Trattandosi di insiemi di firme di funzioni prive di implementazione, permettono di inviare messaggi ad altri oggetti senza generare *hard reference*: le entità comunicano indirettamente, riducendo l'impronta teorica sulla RAM, mentre ciascuna classe ricevente fornisce una propria implementazione logica indipendente.

- **Comunicazione ad eventi (*Event Dispatchers*):** per gestire i cambiamenti di stato globali evitando il paradigma del *polling* continuo tramite l'evento *Tick*, l'architettura si è avvalsa degli *Event Dispatchers* [34]. Questo strumento implementa una logica di tipo *Observer* (uno-a-molti): un singolo *Actor* [35] funge da emettitore (ad esempio, il giocatore che raccoglie un rifiuto) e trasmette

l'evento nell'ambiente di gioco. Tutti i sistemi interessati a tale informazione si iscrivono (*bind*) a questo richiamo, aggiornandosi in modo asincrono al verificarsi dell'azione. Questo approccio mira a minimizzare le operazioni ridondanti sulla CPU, favorendo un flusso di dati più efficiente.

- **Animazione Procedurale (*Timeline*):** una potenziale ottimizzazione a carico della memoria video (VRAM), che lo *Steam Hardware & Software Survey* [14] indica aggirarsi mediamente intorno agli 8 GB, deriva dalla gestione dei movimenti ambientali. Invece di importare *Skeletal Meshes* [18] e file di animazione dedicati, generalmente più onerosi in termini di occupazione di memoria, i movimenti meccanici di porte, scompartimenti e degli stessi strumenti di pulizia sono stati implementati proceduralmente tramite calcoli matematici a *runtime*. L'uso di curve di interpolazione (*Timeline* [36]) combinate a trasformazioni vettoriali ha permesso di simulare i movimenti manipolando esclusivamente le coordinate (*Transform*) di semplici *Static Meshes* [37], contribuendo ad alleggerire il carico di *rendering*.

- **Ottimizzazione dei processi continui (*Custom Polling* e *Timer*):** per le entità che richiedono un monitoraggio ambientale reiterato, la pratica comune in Unreal Engine prevede l'impiego dell'evento nativo `Event Tick`. Tuttavia, poiché tale nodo viene eseguito a ogni singolo fotogramma renderizzato, rischia di generare un *overhead* computazionale sul *Game Thread*, elaborando calcoli che spesso non necessitano di una precisione temporale al millisecondo. Per contenere l'impatto prestazionale, l'utilizzo del *Tick* è stato sostituito, ove possibile, da una routine di *polling* personalizzata. Questa soluzione si basa sul nodo `Set Timer by Function Name` [38], che consente di richiamare ciclicamente una funzione impostando un intervallo di tempo esplicito. Svincolando l'esecuzione dal *framerate* e operando un sottocampionamento della frequenza di aggiornamento (ad esempio, passando a 10 esecuzioni al secondo anziché 60), si ottiene una potenziale riduzione del carico sulla CPU. L'architettura risultante ha l'obiettivo di preservare le risorse computazionali, mantenendo al contempo una reattività del sistema adeguata all'esperienza utente.

4.3 Implementazione delle entità e logiche di gioco

A valle della definizione architetturale e delle direttive di ottimizzazione, i paragrafi successivi espongono l'implementazione pratica del contributo allo sviluppo del progetto. L'analisi si concentrerà inizialmente sulla strutturazione degli *Actor* interattivi, esaminando nello specifico la logica dei *Blueprint* relativi alle porte e la gestione del carrello delle pulizie. Successivamente, la trattazione si sposterà sullo sviluppo dell'Intelligenza Artificiale (IA), illustrando i sistemi progettati per governare il comportamento spaziale e decisionale delle entità infestanti (scarafaggi).

4.3.1 Programmazione degli Actor interattivi e logica del carrello

L'architettura logica degli elementi interattivi è stata progettata per garantire la massima coerenza tra lo stato del mondo e le azioni dell'utente. Sono state implementate tre varianti di porte, ciascuna con responsabilità specifiche.

- **`BP_NeutralDoor`:** Questi *Actor* fungono da principali elementi di transizione. La classe utilizza una logica di sottoscrizione agli eventi globali (*Event Dispatchers*) per reagire autonomamente a cambiamenti dello stato di gioco, come lo sblocco delle aree operative al termine del tutorial o la chiusura forzata durante le sequenze narrative finali.

  L'interazione è stata ingegnerizzata per prevenire compenetrazioni tra la *mesh* e gli attori circostanti. Il sistema esegue un calcolo direzionale confrontando il vettore di posizione del giocatore con il cardine della porta; l'utilizzo di proiezioni vettoriali e volumi di collisione permette di determinare se l'utente si trovi all'interno o all'esterno della soglia, forzando la rotazione del battente in direzione opposta. L'animazione è protetta da un sistema di controllo degli stati che inibisce nuovi *input* durante il movimento. In caso di serratura bloccata, una routine di interpolazione rapida genera una micro-oscillazione procedurale come *feedback* di inaccessibilità.

- **`BP_GuestRoomDoor`:** Estensione della porta neutrale, questa classe gestisce l'interfaccia con il sistema delle missioni (*Quest System*). Al momento dell'attivazione di una *quest*, la porta coordina l'istanziazione procedurale dell'oggetto richiesto. Il sistema interroga l'ambiente per individuare i punti di generazione (*Spawn Points*) e inietta nell'oggetto generato i parametri logici e gli identificativi (*Tags*) necessari per la successiva validazione dell'obiettivo.

- **`BP_ElevatorDoor`:** Progettata seguendo il principio della separazione delle responsabilità, questa classe è delegata esclusivamente all'esecuzione del movimento meccanico (*sliding*). La logica decisionale e i controlli di sicurezza sono stati centralizzati nello `ScriptManager`.

Il fulcro operativo dell'esperienza è il carrello delle pulizie (**`BP_Cart`**). La sua implementazione ha richiesto una gestione complessa della gerarchia e del controllo del giocatore.

A causa delle instabilità fisiche riscontrate nell'ancoraggio diretto del personaggio al carrello, si è optato per un sistema di trasferimento del possesso (*Possession*). Ad ogni interazione, l'utente assume il controllo diretto del carrello. Per mitigare il disorientamento spaziale, il sistema gestisce una transizione fluida della telecamera che interpola la visuale tra i due punti di osservazione.

La gestione dell'attrezzatura sul carrello è affidata alla funzione `PutObjectOnCart`. Questo sistema gestisce l'ancoraggio gerarchico degli utensili (mocio, scacciamosche, spolverino) ai rispettivi scomparti del modello 3D. L'algoritmo identifica la tipologia

di oggetto tramite metadati (*Tags*), calcola la matrice di trasformazione relativa rispetto al punto di aggancio e vincola l'utensile alla struttura del carrello (*attachment*), ripristinando al contempo le corrette coordinate di rotazione per assicurarne l'allineamento visivo.

La meccanica di distacco (*Detach*) si avvale di un algoritmo di riposizionamento procedurale. Prima di restituire il controllo al giocatore, il sistema analizza lo spazio circostante tramite proiezioni volumetriche per identificare una superficie libera da ostacoli, garantendo un rientro in gioco privo di anomalie fisiche. Poiché il carrello è privo della logica nativa di movimento, l'aggiornamento costante dei suoi sottosistemi è affidato a un ciclo di *polling* personalizzato (`Fake_Tick`), che gestisce la correzione delle coordinate verticali e l'aggiornamento dei *feedback* visivi legati al movimento.

4.3.2 Comportamento dei PNG infestanti

Lo sviluppo degli insetti infestanti è stato orientato alla creazione di un comportamento reattivo e autonomo.

- **`BP_NotInteractableCockroach`:** Queste entità seguono una logica di movimento deterministico calcolato tramite interpolazione spaziale verso destinazioni fisse, con successiva rimozione automatica dalla memoria per l'ottimizzazione delle risorse.

- **`BP_Cockroach`:** Il comportamento di queste unità è basato su un ciclo di navigazione stocastica. L'entità interroga il sistema di navigazione (*NavMesh*) per individuare punti casuali raggiungibili. L'allineamento e lo spostamento sono gestiti tramite routine temporizzate che eliminano la necessità di calcoli ad ogni fotogramma.

  Il sistema è reattivo: la prossimità del giocatore altera dinamicamente i parametri di velocità dell'IA, simulando un comportamento di fuga. Inoltre, l'entità è integrata nel ciclo di *gameplay*: alla sua distruzione, il sistema innesca la generazione di una macchia ematica, collegando funzionalmente la meccanica di disinfestazione a quella di pulizia.

4.4 Comparto visivo, sonoro e integrazione degli Asset

Come anticipato nelle sezioni precedenti, lo sviluppo ha previsto la collaborazione di due modellatori 3D, incaricati della realizzazione degli *asset* fisici destinati a comporre l'ambiente di gioco. La modellazione poligonale e la generazione delle primitive di collisione sono state eseguite tramite il software Blender, mentre per il *texturing* ci si è avvalsi della suite Adobe Substance 3D. La *pipeline* grafica ha previsto l'esportazione e l'integrazione di cinque mappe fondamentali per favorire una corretta interazione con l'illuminazione secondo il paradigma PBR (*Physically Based Rendering*) [39]:

Mappa PBR	**Funzione tecnica**	**Impatto sul *Rendering* / Scopo**
Base Color	Mappa RGB che definisce il colore diffuso della superficie, priva di informazioni di illuminazione precalcolate.	Fornisce la base cromatica pura su cui il motore calcola l'illuminazione fisicamente accurata.
Metallic	Maschera in scala di grigi per distinguere materiali dielettrici (valore 0) da metallici (valore 1).	Regola la risposta fisica del materiale, governando l'eventuale colorazione dei riflessi.
Ambient Occlusion	Mappa in scala di grigi che simula l'occlusione della luce ambientale nelle intersezioni geometriche.	Genera micro-ombreggiature per aggiungere profondità senza gravare sul calcolo dell'illuminazione globale.
Normal	Mappa vettoriale RGB che altera la direzione apparente della normale superficiale per deviare la luce.	**Ottimizzazione GPU:** simula dettagli geometrici complessi (es. graffi o rilievi) su modelli *low-poly*.
Roughness	Mappa in scala di grigi che determina la micro-ruvidità della superficie.	Controlla la dispersione dei raggi luminosi incidenti, stabilendo il grado di opacità e la nitidezza dei riflessi.

Tabella 4.1: Riepilogo delle mappe utilizzate nel *workflow* PBR (*Physically Based Rendering*), con relative funzioni.

Per quanto concerne il comparto audio, l'ambiente sonoro di *Bloody Shift* è stato strutturato selezionando tracce ed effetti da librerie *open source* libere da vincoli di *copyright*. I file sono stati importati nel formato non compresso `.wav`, raccomandato dalla documentazione di Unreal Engine [40], al fine di preservare la qualità acustica e facilitare la spazializzazione sonora all'interno del motore.

4.4.1 Progettazione del sistema di illuminazione

L'impianto illuminotecnico di "Bloody Shift" è stato concepito per supportare la dimensione atmosferica e, al contempo, fungere da strumento di *level design* per l'orientamento del giocatore. Per enfatizzare il contrasto chiaroscurale, l'illuminazione

ambientale globale è stata ridotta al minimo, affidando la leggibilità dello spazio esclusivamente a sorgenti artificiali localizzate (*Spot Light* e *Point Light*).

La gerarchia delle fonti luminose è stata strutturata richiamando i principi della programmazione orientata agli oggetti (OOP) per garantirne la modularità:

- **Illuminazione ambientale e procedurale:** le unità luminose dei corridoi (`BP_HallwayLamp`) derivano da una classe genitrice che centralizza la logica di gestione dell'intensità. Attraverso l'impiego di interfacce dedicate, queste entità gestiscono in modo astratto i comandi di attivazione e disattivazione. Per incrementare il realismo, è stata implementata una routine di *flickering* che modula l'intensità luminosa nel tempo, simulando proceduralmente un malfunzionamento elettrico tramite interpolazione.

  Per favorire il disaccoppiamento tra logica e *rendering*, queste unità comunicano con il `BP_LightingComponent`, il quale sincronizza l'emissione fisica della luce con il *feedback* visivo del modello 3D (canale emissivo del materiale).

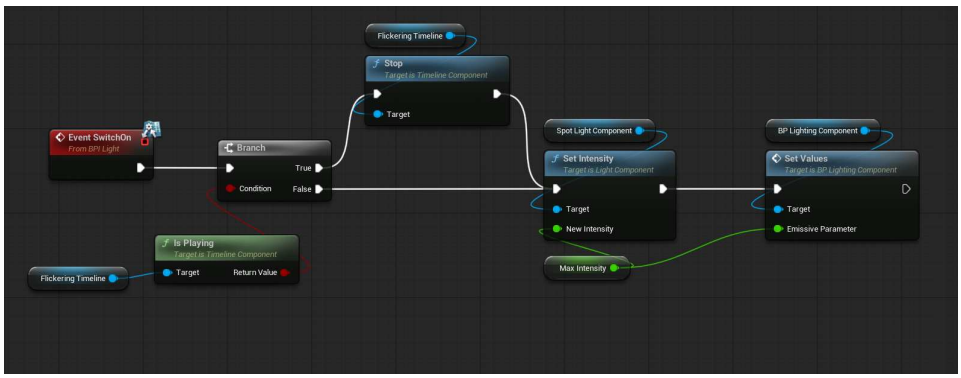

Figura 4.4: Architettura logica dell'evento di attivazione luminosa.

- **Sistemi di illuminazione delle stanze:** Nelle aree private, l'illuminazione è gestita tramite un'interazione diretta con interruttori fisici (`BP_LightSwitch`). Ciascuna stanza è stata progettata come un set illuminotecnico autonomo: l'attivazione dell'interruttore aggiorna lo stato logico di un gruppo di sorgenti pre-associate, modulando l'intensità volumetrica della camera.

  Il sistema garantisce la coerenza audiovisiva dell'azione: l'interazione innesca simultaneamente un campionamento acustico e un'animazione procedurale di rotazione della *mesh* dell'interruttore, fornendo all'utente un *feedback* immediato sulla riuscita dell'operazione.

Gestione dei Materiali e ottimizzazione visiva Per ottimizzare l'impiego delle risorse della GPU e contenere il carico sulla VRAM, il flusso di lavoro è stato basato sulla separazione tra *Master Material* e Istanze di Materiale (*Material Instance*). Questa metodologia prevede lo sviluppo di un numero limitato di *shader* complessi e

parametrizzati, dai quali generare istanze leggere che ereditano la logica di base ma consentono variazioni estetiche a *runtime*.

L'architettura dei materiali *Master* è stata strutturata per essere flessibile e modulare:

Materiale	**Funzione**	**Motivazione (*Design* / Ottimizzazione)**
`MM_Opaque Material`	Costituisce la base per la maggior parte degli *asset*, integrando mappatura UV dinamica, canali PBR e tecnica di *Albedo Darkening*.	**Ottimizzazione visiva:** previene distorsioni (*stretching*) e forza la resa delle micro-ombreggiature per compensare i limiti della *Global Illumination*.
`MM_StainDecal`	Gestisce le macchie procedurali tramite *Deferred Decal*, con variazione dinamica di colore, rilievo e un canale emissivo animato.	**UX Design:** genera un effetto pulsante che guida l'attenzione del giocatore, facilitando l'individuazione degli ostacoli.
`MM_Dissolve Material`	Impiega una *texture* di rumore procedurale per pilotare l'erosione della maschera di opacità, generando un bordo incandescente matematico.	**Feedback visivo:** simula realisticamente l'effetto di combustione durante lo smaltimento nel forno inceneritore.
`MM_Glass`	Applica un modello di *rendering* traslucido semplificato alle superfici trasparenti, combinato a mappe di rugosità.	**Ottimizzazione:** simula imperfezioni superficiali mantenendo un basso costo computazionale, per non inficiare il *framerate* globale.

Tabella 4.2: Riepilogo dei materiali *Master* implementati, con relative funzioni e finalità progettuali.

4.4.2 Implementazione e spazializzazione del sistema audio

Il comparto sonoro è stato progettato per garantire un'immersione profonda attraverso l'uso della spazializzazione tridimensionale. Ogni campionamento audio è stato incapsulato in *Sound Cue*, che fungono da contenitori logici per definire le curve di attenuazione, la distanza di decadimento (*falloff*) e la modulazione dinamica del tono.

Dal punto di vista implementativo, l'orchestrazione dei suoni a *runtime* è stata vincolata a rigidi controlli di validità per prevenire fenomeni di "acoustic clutter"

(sovrapposizione caotica di tracce). Prima di autorizzare la riproduzione di un effetto (come il suono dei passi o lo scorrimento delle porte), il sistema verifica se l'istanza sonora precedente sia ancora attiva. Solo in caso di risposta negativa, ovvero quando l'oggetto audio non risulta più presente in memoria, viene autorizzato l'avvio di un nuovo ciclo. Questa logica di controllo, basata su variabili di stato booleane, assicura che il paesaggio sonoro resti pulito e coerente con le azioni compiute dall'utente o dall'Intelligenza Artificiale.

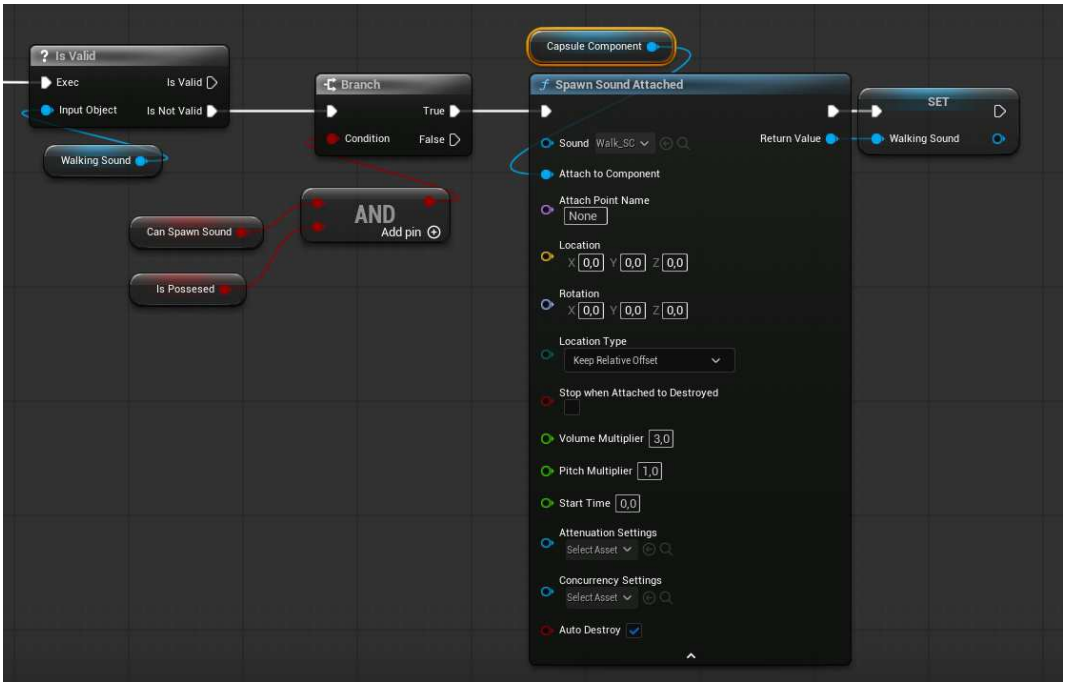

Figura 4.5: Logica di gestione dell'audio cinetico: sistema di validazione delle istanze per la prevenzione della saturazione acustica.

4.4.3 Importazione e configurazione dei modelli tridimensionali

La *pipeline* di importazione e configurazione dei modelli tridimensionali ha seguito un flusso di lavoro standardizzato. Gli *asset* modellati in Blender sono stati trasferiti nel motore grafico adottando il formato di interscambio `.fbx` (*Filmbox*), allo scopo di favorire la compatibilità geometrica.

Una volta importata, ciascuna *Static Mesh* è stata isolata e revisionata all'interno dell'editor nativo di Unreal Engine, prestando attenzione al comparto fisico. Sebbene la generazione delle primitive di collisione (*custom collision*) venisse spesso pre-calcolata durante la modellazione, si è talvolta reso necessario un intervento di rifinitura o sostituzione. Qualora i *collider* importati presentassero difetti di allineamento (risolvibili tramite alterazioni del parametro *Transform*) o risultassero geometricamente complessi, venivano rimossi in favore di primitive generate nativamente dal motore grafico.

Sostituire collisioni altamente triangolate con forme matematiche basilari (*Box* o *Capsule Collider*) rappresenta una tecnica di ottimizzazione: riduce l'alberatura dei calcoli fisici delegati alla CPU per la gestione degli urti e delle sovrapposizioni (*Overlap*), mantenendo inalterata la fedeltà percepita dal giocatore.

A valle della configurazione fisica, a ciascun modello venivano assegnate le rispettive *Material Instance*. Conclusa la configurazione, l'oggetto veniva posizionato nell'ambiente di gioco per la composizione della scena (*Level Design* statico), oppure veniva incapsulato come `StaticMeshComponent` all'interno di un *Actor Blueprint*, per potergli associare la logica interattiva prevista.

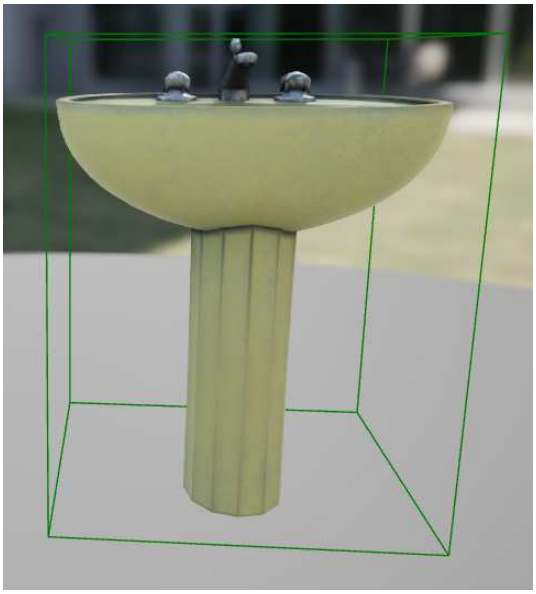

Figura 4.6: Implementazione di una collisione ottimizzata basata su una primitiva geometrica semplice (*Box Collider*) per l'asset del lavandino.

4.5 Sistemi logici a supporto dell'Onboarding

A valle delle direttive di *User Experience* definite in sede metodologica, l'implementazione della fase di *onboarding* ha richiesto lo sviluppo di sistemi progettati per gestire in modo procedurale le transizioni visive e l'illuminazione dinamica degli *asset*.

4.5.1 Transizioni cinematiche e manipolazione della telecamera

Per orchestrare le sequenze narrative senza ricorrere a interruzioni esterne, le transizioni della visuale verso elementi di interesse sono state implementate operando direttamente sui parametri della telecamera. Questa dinamica è stata integrata nelle classi `BP_CharacterController` e `BP_Cart` tramite una funzione di orientamento procedurale (`RotateToObject`).

All'attivazione di uno specifico *trigger*, il sistema inibisce temporaneamente l'accettazione degli *input* utente per garantire l'integrità della sequenza. A livello matematico, l'algoritmo calcola il vettore direzionale necessario per allineare la telecamera al bersaglio e utilizza routine di interpolazione sferica per spostare fluidamente lo sguardo verso l'obiettivo narrativo. Questo approccio garantisce che l'utente assista a eventi critici per la progressione (come l'apertura coordinata di un infisso) prima di riacquisire la piena manovrabilità dei controlli.

4.5.2 Gestione dinamica dell'illuminazione: il `BP_LightingComponent`

Per supportare la navigazione intuitiva in assenza di un'interfaccia utente (HUD), l'infrastruttura si avvale di una gestione dinamica delle proprietà emissive dei materiali, centralizzata nel `BP_LightingComponent`.

L'architettura di questo componente è stata progettata per massimizzare la modularità e l'efficienza:

- **Inizializzazione dinamica:** Durante l'avvio, il componente scansiona gerarchicamente l'attore a cui è assegnato, individuando le geometrie e convertendo i materiali statici in istanze dinamiche (*Dynamic Material Instances*). Questo processo permette di modificare le proprietà dello *shader* a *runtime* senza influenzare gli altri oggetti della scena.
- **Modulazione procedurale:** Attraverso una funzione dedicata, il sistema aggiorna i parametri scalari associati al canale emissivo del *Master Material*. Ciò consente di variare l'intensità luminosa superficiale di qualsiasi strumento o interruttore in tempo reale, comunicandone l'importanza operativa (*affordance*) attraverso contrasti visivi controllati.

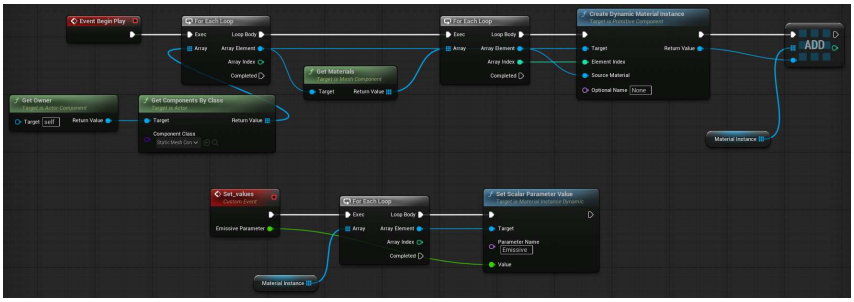

Figura 4.7: Logica dell'istanza modulare `BP_LightingComponent`: orchestrazione procedurale del canale emissivo.

4.6 Il copione di Bloody Shift: lo `ScriptManager`

L'architettura software più complessa del progetto risiede nello `ScriptManager`, un *Actor* globale incaricato di governare il *game flow* e di orchestrare l'avanzamento degli stati di gioco.

Il sistema si fonda sull'interazione tra volumi di collisione (*trigger zone*) e un selettore logico centrale. Al momento dell'intersezione spaziale del giocatore con un'area critica, l'algoritmo valida l'identità dell'attore coinvolto e invoca le istruzioni previste per quel determinato segmento del livello, coordinando accensioni luminose, transizioni di camera e animazioni ambientali.

Un'ottimizzazione rilevante riguarda l'uso del *Construction Script* per il posizionamento dei *trigger*. Invece di configurare manualmente le coordinate di ogni volume, è stato implementato un sistema di manipolazione visiva tramite *3D Widget*. Questa soluzione permette al *designer* di riposizionare e scalare le aree di interazione direttamente nella *viewport*, leggendo i dati da variabili strutturate e riducendo drasticamente il margine di errore nella disposizione spaziale degli eventi narrativi.

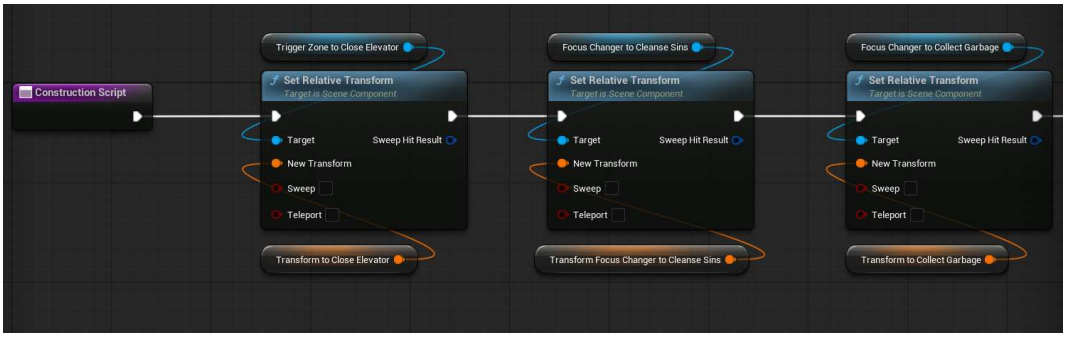

Figura 4.8: Ottimizzazione del *workflow* tramite manipolatori spaziali per la disposizione delle *trigger zone*.

L'avanzamento della sessione è governato dall'evento `PlayScript`, che opera come una macchina a stati finiti. Ogni attivazione incrementa un contatore di stato deterministico, indirizzando l'esecuzione verso blocchi logici sequenziali. Questo garantisce una progressione lineare e impedisce l'attivazione asincrona di eventi successivi.

Di seguito si analizzano le fasi principali coordinate dal sistema:

- **Fasi 1-2 (Inizializzazione e Isolamento):** Il sistema gestisce l'ancoraggio iniziale dell'avatar al carrello e l'apertura automatica della cabina dell'ascensore. All'uscita dell'utente, un evento di *overlap* comanda la chiusura forzata dell'infisso, ottimizzando il carico computazionale e stabilendo il tono claustrofobico dell'esperienza.
- **Fasi 3-5 (Apprendimento e Usura):** Lo `ScriptManager` guida l'attenzione verso la prima macchia ematica, abilitando le meccaniche di detersione. La logica monitora l'utilizzo degli strumenti: al raggiungimento della saturazione del mocio, il sistema innesca *feedback* diegetici (monologhi e luminescenza del secchio d'acqua) per suggerire la manutenzione dell'attrezzatura.
- **Fasi 6-7 (Smaltimento e Logistica):** Attraverso transizioni cinematiche, l'utente viene introdotto alla raccolta dei rifiuti. Il sistema tiene traccia della

capienza del sacco e orienta visivamente il giocatore verso l'inceneritore tramite l'attivazione sequenziale delle sorgenti luminose a soffitto.

- **Fasi 8-10 (Anticipazione e Loop di Gameplay):** Una coreografia ambientale di *foreshadowing* anticipa la minaccia dei PNG infestanti. Il sistema concatena le meccaniche: la distruzione di una blatta genera dinamicamente una macchia, forzando l'utente a riutilizzare gli strumenti precedenti e consolidando il ciclo di apprendimento.
- **Fase 11 (Conclusione dell'Onboarding):** Al completamento degli obiettivi didattici, lo `ScriptManager` sblocca le aree restanti e restituisce all'utente piena libertà esplorativa, inaugurando il *core loop* principale.
- **Fasi 12-15 (Epilogo e Transizione procedurale):** Il raggiungimento della soglia operativa finale attiva il rientro nell'ascensore. Durante il movimento della cabina, il sistema esegue una manipolazione massiva dell'ambiente circostante: gli *Actor* interattivi vengono rimossi e i materiali delle superfici vengono sostituiti dinamicamente a *runtime* (*Texture Swapping*). Questa transizione oculata prepara il set per la risoluzione narrativa finale e la visualizzazione dell'interfaccia di chiusura.

Per tutelare l'integrità di questa progressione, è stato implementato un sistema di protezione contro il *sequence breaking*. Ogni interazione è subordinata a un blocco logico condizionale: le variabili di stato inibiscono la manipolazione di oggetti e utensili finché il "copione" non ne prevede l'introduzione formale, assicurando che l'assimilazione delle regole avvenga in modo incrementale e controllato.

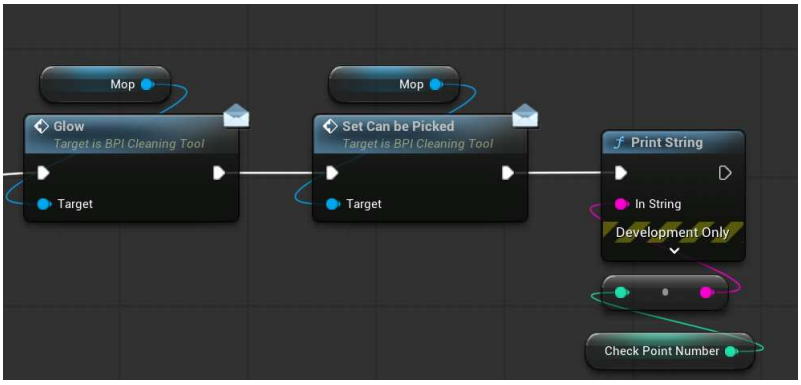

Figura 4.9: Sistema di protezione contro il *sequence breaking*: validazione degli stati di interazione.

Capitolo 5

Risultati

Il presente capitolo costituisce la fase conclusiva della trattazione, dedicata alla presentazione e alla valutazione critica delle risultanze ottenute al termine del ciclo di sviluppo. Dopo aver analizzato le premesse metodologiche e l'architettura tecnica del caso di studio, si procede qui alla validazione dell'opera prodotta, verificando l'effettiva corrispondenza tra i requisiti iniziali e l'output finale.

La disamina si articola in due fasi complementari. In primo luogo, verrà analizzato il software sotto il profilo estetico e funzionale, documentando attraverso sessioni di *gameplay* l'efficacia del *level design* e dell'atmosfera ricercata. In seconda istanza, verranno elaborati i dati raccolti durante la fase di *User Testing*, seguendo il protocollo di indagine delineato nel Capitolo 3. L'analisi dei *feedback*, sia quantitativi che qualitativi, permetterà di misurare il grado di usabilità del titolo, l'impatto della filosofia di *onboarding* diegetico e la stabilità tecnica complessiva, fornendo un quadro esaustivo sul valore dell'esperienza videoludica realizzata.

5.1 Analisi del software realizzato e *visual design*

Il percorso di sviluppo descritto nei capitoli precedenti è culminato nella realizzazione di una *Beta release* di "Bloody Shift". In questa fase del ciclo di vita del software, il prodotto presenta l'intero set di funzionalità e meccaniche di gioco previste, risultando idoneo per le sessioni di *testing* e per una valutazione critica del comparto estetico e funzionale.

L'impatto visivo complessivo riflette la volontà di operare una sintesi tra due linguaggi apparentemente antitetici: la precisione metodica del genere simulativo e l'inquietudine viscerale dell'estetica *horror-grottesca*. La sfida progettuale è stata quella di trasformare mansioni ripetitive e "asettiche" in un'esperienza tensiva, dove l'hotel infernale non funge da semplice fondale, ma da attore protagonista che interagisce con il giocatore attraverso stimoli sensoriali e visivi.

L'integrazione di tecnologie avanzate, quali l'illuminazione globale *Lumen* e un sistema di *post-processing* mirato (filtro VHS), ha permesso di ottenere una coerenza stilistica definita, capace di immergere l'utente in un'atmosfera claustrofobica che valorizza la narrazione ambientale (*environmental storytelling*). Nelle sezioni seguenti verrà operata una disamina dettagliata dei risultati raggiunti, supportata dalla

documentazione fotografica delle sessioni di gioco e dall'analisi dei dati raccolti sul campo.

5.1.1 Documentazione fotografica e analisi delle sessioni di *gameplay*

In questa sezione viene presentata una selezione di fermo-immagine tratti da sessioni di gioco effettive sulla *build* definitiva del progetto. La documentazione fotografica mira a validare la resa estetica dei materiali, l'efficacia del sistema di illuminazione e la coerenza della narrazione ambientale.

Si rende opportuna una precisazione in merito all'interfaccia utente (UI) visibile negli *screenshot*. In conformità con la filosofia di *design* esposta nel Capitolo 3, l'HUD è stato ridotto esclusivamente al tracciamento numerico dei requisiti minimi di pulizia e delle missioni attive (*Objective List*).

La Figura 5.1 illustra il nucleo operativo dell'esperienza interattiva. È possibile osservare l'integrazione spaziale dei PNG infestanti (gli scarafaggi) in reazione all'avvicinamento del giocatore armato. L'assenza di reticoli di mira artificiali (*crosshair* dinamici) conferma la volontà di mantenere l'azione ancorata al mondo di gioco, affidando la precisione del colpo unicamente all'allineamento visivo e all'ampio *collider* dello strumento.

Figura 5.1: Dinamiche di *core loop*: neutralizzazione dei PNG infestanti mediante l'uso dello scacciamosche e gestione degli elementi macabri a terra.

Nella Figura 5.2 viene documentata l'efficacia del `BP_LightingComponent` analizzato nel Capitolo 4. Al raggiungimento della soglia di usura del mocio, il secchio dell'acqua sul carrello attiva una luminescenza gialla procedurale. Questo *feedback* visivo cattura immediatamente l'attenzione del giocatore, comunicando la necessità di sciacquare lo strumento per proseguire il turno di lavoro, ovviando alla necessità di finestre di dialogo invasive.

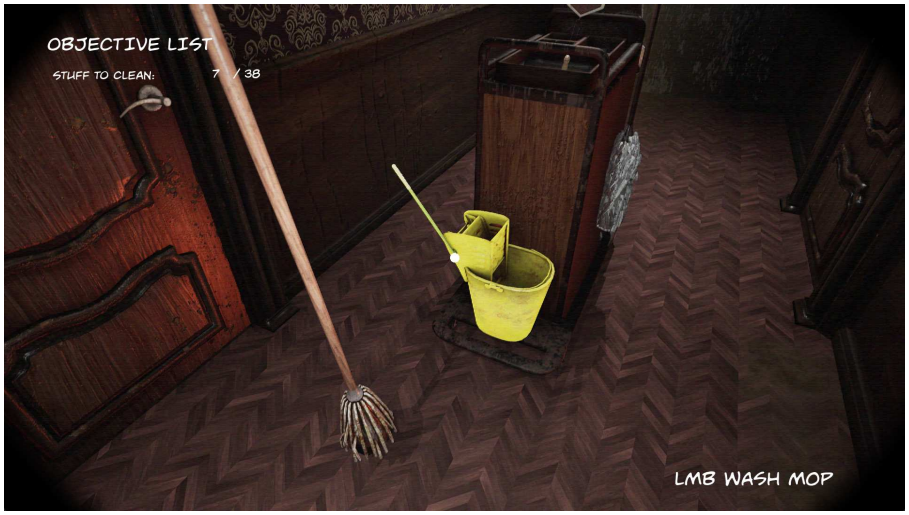

Figura 5.2: Dettaglio del carrello delle pulizie e attivazione del `BP_LightingComponent`. L'emissione luminosa dal secchio dell'acqua funge da *feedback* diegetico: un indicatore visivo integrato direttamente nel mondo di gioco che comunica all'utente lo stato di interazione senza gravare sull'interfaccia utente (HUD), favorendo così l'immersività.

Il concetto di narrazione ambientale trova la sua massima espressione nelle stanze degli ospiti VIP. Come illustrato nella Figura 5.3, la presenza di elementi caratterizzanti (quali ampolle chimiche e tracce ematiche specifiche) permette all'utente di identificare immediatamente il profilo dell'occupante. Questo allestimento scenico non ha solo una funzione estetica, ma introduce coerentemente le *quest* speciali che il giocatore dovrà risolvere per sbloccare la progressione narrativa.

Figura 5.3: Esempio di *environmental storytelling* in una delle stanze VIP. Gli *asset* distribuiti suggeriscono l'identità dell'ospite prima dell'interazione testuale.

Infine, la Figura 5.4 propone un confronto diretto tra l'estetica dei corridoi nelle

fasi iniziali e l'epilogo del gioco. Questa giustapposizione evidenzia la riuscita tecnica del sistema di manipolazione ambientale orchestrato dallo `ScriptManager`: attraverso la sostituzione istantanea dei materiali (*Texture Swapping*) e la variazione dei parametri di illuminazione globale, l'hotel muta radicalmente la propria estetica in tempo reale. Il marcato contrasto tra la saturazione cupa della Fase Iniziale (a) e la luminosità eterea della Fase Finale (b) comunica visivamente l'illusione dell'ascensione celestiale del protagonista, prima della chiusura dell'esperienza ludica.

(a) Fase Iniziale: estetica *horror-grottesca*.

(b) Fase Finale: illuminazione eterea.

Figura 5.4: Confronto visivo diretto che illustra il risultato dell'operazione di *texture swapping* a *runtime* orchestrata dallo `ScriptManager`.

5.2 Acquisizione dei dati: l'esperienza di testing allo Svilupparty

La maggior parte dei dati analizzati in questo capitolo è stata raccolta il 9 maggio 2026 in occasione dello Svilupparty, storico evento italiano dedicato ai creatori di videogiochi indipendenti. La partecipazione alla fiera ha rappresentato un'opportunità metodologica fondamentale per sottoporre *Bloody Shift* a un *playtest* intensivo e dal vivo, raccogliendo il *feedback* di un pubblico eterogeneo composto sia da semplici appassionati che da sviluppatori del settore.

Arrivati sul posto, il team ha allestito una postazione dedicata, configurata in modo da massimizzare l'attrattiva visiva e l'ergonomia. L'hardware a disposizione comprendeva un personal computer dotato di periferiche standard (mouse e tastiera) e una configurazione a doppio schermo. Il monitor principale era riservato esclusivamente al *gameplay*, mentre il secondo trasmetteva in *loop* un *trailer* promozionale realizzato internamente, rivelatosi uno strumento essenziale per catturare l'attenzione dei passanti. A completamento della postazione, era stato predisposto un codice QR che rimandava direttamente ai contatti e alle pagine ufficiali del progetto.

Il collaudo ha assunto la forma di un testing supervisionato. Nel corso dell'evento, 10 utenti incuriositi dal *trailer* si sono seduti alla postazione per provare la *Beta* e hanno successivamente compilato il questionario di valutazione. Le sessioni di gioco hanno avuto una durata altamente variabile, strettamente legata all'esperienza pregressa e alle abilità ludiche dei singoli partecipanti: alcuni giocatori sono riusciti

a completare l'intera sequenza prevista dalla *demo*, mentre altri hanno interrotto la prova prima della conclusione naturale del livello.

Durante le sessioni, i membri del team si sono posizionati alle spalle dei giocatori, adottando un approccio di osservazione non intrusiva. Questo ha permesso di intervenire fornendo leggeri suggerimenti solo nei casi di stallo prolungato, evitando che la frustrazione compromettesse l'esperienza complessiva e sfalsasse la percezione del gioco. Questa modalità in presenza si è rivelata nettamente superiore al testing da remoto per due ragioni fondamentali. In primo luogo, ha consentito di studiare in tempo reale il linguaggio del corpo, le esitazioni e le reazioni spontanee dell'utenza di fronte alle meccaniche. In secondo luogo, le considerazioni "a caldo" espresse a voce dai giocatori al termine della partita hanno fornito consigli e intuizioni di inestimabile valore, impossibili da catturare in un semplice modulo digitale e fondamentali per pianificare i futuri aggiornamenti.

È interessante notare come l'osservazione diretta abbia funto da prima validazione empirica dei dati quantitativi successivi: le criticità ergonomiche, gli ostacoli di navigazione e i *bug* tecnici notati visivamente dal team durante le prove sullo schermo si sono poi riflessi con assoluta coerenza nei problemi riportati dagli utenti all'interno dei questionari.

Figura 5.5: Logo ufficiale dell'evento Svilupparty, contesto fieristico in cui si è svolta la fase di *playtesting* supervisionato.

5.3 Analisi dei *feedback* e dei dati raccolti

In questa sezione si procederà all'analisi quantitativa e qualitativa dei dati raccolti tramite il modulo di valutazione, sottoposto a un campione totale di 15 partecipanti. L'obiettivo dell'indagine è esaminare l'esperienza empirica vissuta dagli utenti durante le sessioni di *playtesting* della *Beta release*. Lo studio di tali metriche consentirà di raccogliere indicazioni sull'efficacia delle soluzioni di *game design* implementate e, contestualmente, di isolare le eventuali criticità del sistema, fornendo una base

utile e oggettiva per definire le aree di potenziale miglioramento in vista di iterazioni future del progetto.

5.3.1 Valutazione quantitativa e analisi dei grafici

In coerenza con l'impostazione metodologica delineata nel Capitolo 3, l'esposizione dei dati estratti dai questionari segue la medesima ripartizione logica in macro-aree. Tale strutturazione permette di incrociare le intenzioni progettuali iniziali con il *feedback* empirico restituito dal campione di *playtester*.

Profilazione del campione e benchmarking

La prima fase dell'indagine è stata dedicata alla profilazione del campione, un passaggio utile per la contestualizzazione dei dati sull'usabilità e sul *game design*.

Dalle risposte a campo aperto relative ai generi preferiti, emerge una compresenza di videogiocatori assidui (orientati verso produzioni AAA, *Survival Horror*, FPS e GDR) e utenti occasionali (fruitori di titoli *mobile* o *platform*). Tale varietà si riflette nel monte ore settimanale (Figura 5.6): il 46,7% (7/15) degli intervistati dichiara sessioni di gioco superiori alle 10 ore, controbilanciato da un 33,3% (5/15) che si attesta sotto le 5 ore e un 20% (3/15) nella fascia intermedia.

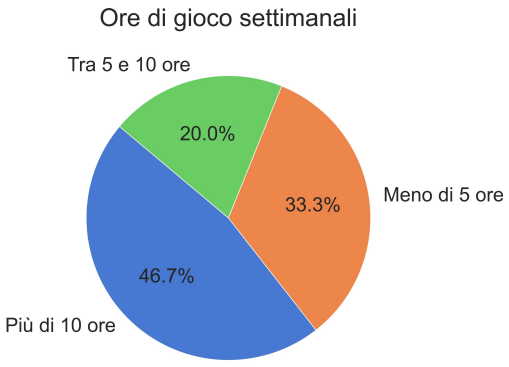

Figura 5.6: Distribuzione delle ore di gioco settimanali del campione.

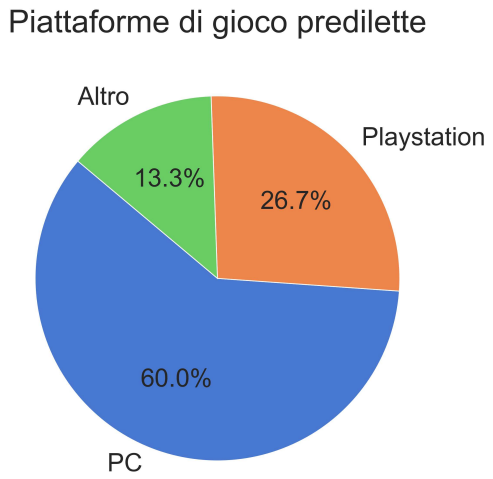

Figura 5.7: Piattaforme di gioco predilette dagli intervistati.

Per quanto concerne l'hardware di riferimento (Figura 5.7), il personal computer risulta la piattaforma dominante, selezionata dal 60% (9/15) degli utenti, seguita dall'ecosistema PlayStation (26,7%, 4/15) e da altre piattaforme (13,3%, 2/15), mentre le console Xbox non registrano preferenze. L'utilizzo maggioritario del PC allinea l'utenza con l'ambiente nativo di sviluppo della *Beta*.

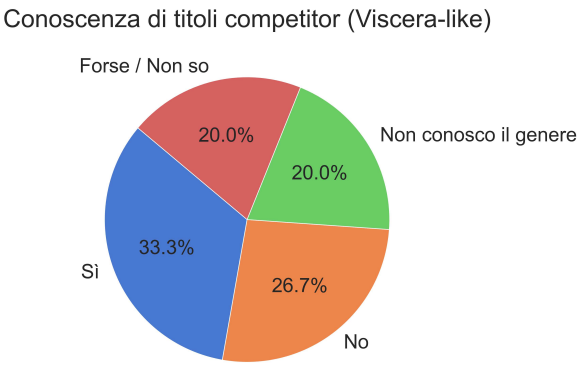

Figura 5.8: Esperienza pregressa degli utenti con titoli *competitor* (es. *Viscera Cleanup Detail*).

Infine, la Figura 5.8 illustra il livello di conoscenza del genere di riferimento. Un terzo del campione (33,3%, 5/15) ha maturato esperienze pregresse con titoli *competitor* diretti. Il restante 66,7% (10/15) si divide tra chi non conosce il genere e chi non ha saputo esprimere una preferenza netta.

Accessibilità, controlli e Onboarding

Il secondo blocco dell'indagine è stato dedicato alla valutazione del sistema di apprendimento iniziale in assenza di finestre esplicative testuali.

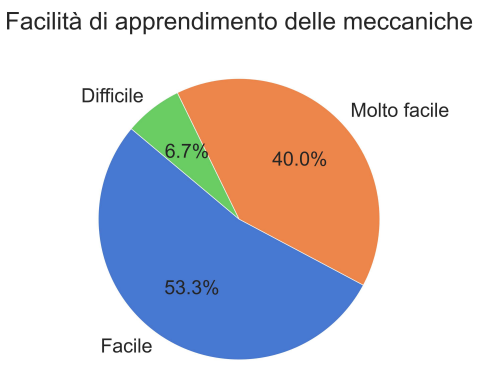

Figura 5.9: Facilità di apprendimento delle meccaniche.

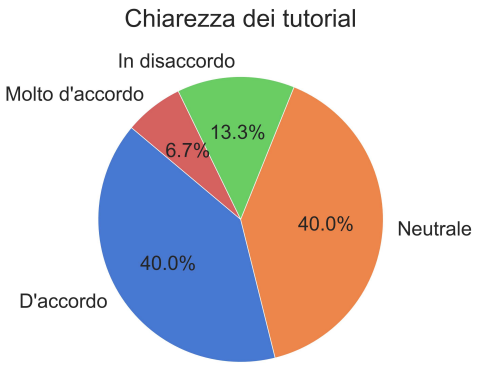

Figura 5.10: Percezione della chiarezza dei tutorial diegetici.

Figura 5.11: Metriche relative alla curva di apprendimento (*Onboarding*).

Come illustrato nella Figura 5.11, alla domanda relativa alla facilità di assimilazione delle meccaniche (*"Did you find the game easy to pick up?"*), il 93,3% (14/15) del campione ha risposto in modo positivo (53,3% "facile" e 40% "molto facile"), a fronte di un 6,7% (1/15) che ha riscontrato difficoltà. In merito alla chiarezza dei tutorial (*"Were the instructions/tutorials provided clear and sufficient?"*), i dati mostrano

una diversa distribuzione: il 46,7% (7/15) ha espresso un parere favorevole (40% d'accordo e 6,7% molto d'accordo), il 40% (6/15) si è posizionato su un giudizio neutrale e il 13,3% (2/15) ha manifestato disaccordo.

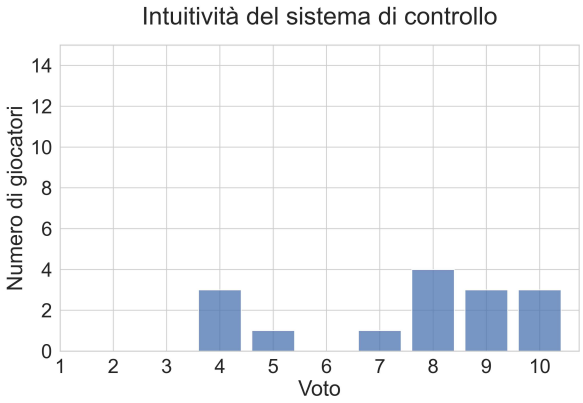

Figura 5.12: Intuitività del sistema di controllo su scala decimale.

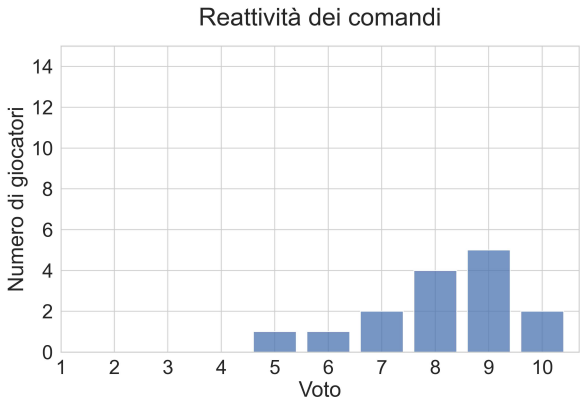

Figura 5.13: Reattività percepita (*responsiveness*).

Figura 5.14: Valutazione dell'ergonomia e della mappatura dei controlli.

Sotto il profilo dell'ergonomia e della reattività dei comandi, misurati su una scala decimale (Figura 5.14), oltre il 66% (10/15) degli utenti ha valutato l'intuitività con punteggi tra 8 e 10, con le restanti preferenze distribuite sui valori centrali. La reattività percepita (*responsiveness*) ha visto il 100% (15/15) del campione assegnare un voto pari o superiore a 5, con una concentrazione del 73,3% (11/15) sui punteggi compresi tra l'8 e il 10.

Per quanto riguarda la leggibilità degli scopi ludici (Figura 5.17), il 73,3% (11/15) dei partecipanti non ha riscontrato difficoltà nel comprendere l'obiettivo principale della sessione, mentre il 26,7% (4/15) ha manifestato incertezza (risposta "forse"). La chiarezza della *User Interface* è stata valutata positivamente dall'86,6% (13/15) dei *playtester* (53,3% "facile", 33,3% "molto facile").

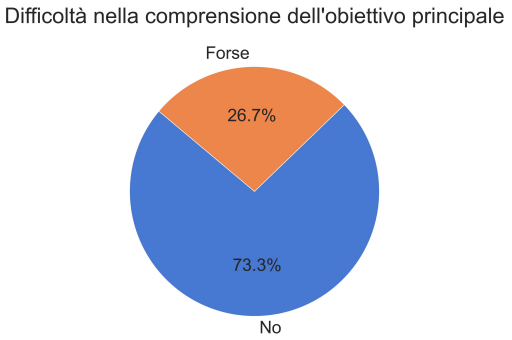

Figura 5.15: Difficoltà percepita nella comprensione dell'obiettivo principale.

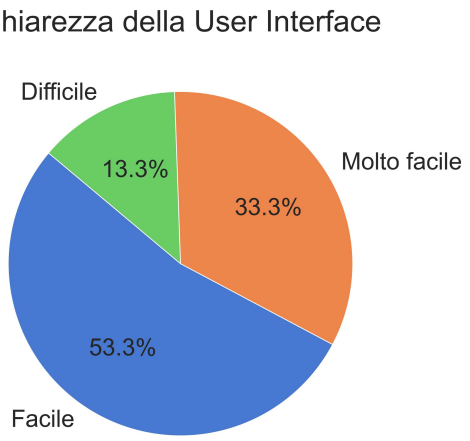

Figura 5.16: Livello di chiarezza della *User Interface*.

Figura 5.17: Metriche relative alla leggibilità delle interfacce e degli obiettivi.

Valutazione del Core Loop e delle Meccaniche

Il terzo blocco dell'analisi quantitativa espone i dati sul nucleo interattivo dell'opera (*core loop*).

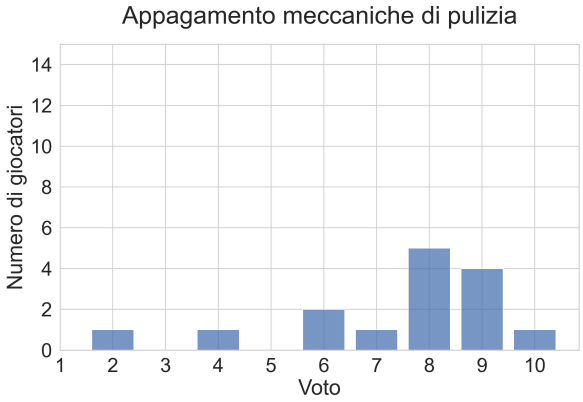

Figura 5.18: Livello di appagamento derivante dal ciclo di pulizia su scala decimale.

Il livello di soddisfazione legato al ciclo primario di pulizia, valutato su scala decimale (Figura 5.18), mostra che il 66,7% (10/15) del campione ha concentrato i propri voti nella fascia alta (33,3% per il voto 8, 26,7% per il 9 e 6,7% per il 10). Le restanti valutazioni si distribuiscono sui valori compresi tra 2 e 7.

In relazione all'uso del carrello mobile (Figura 5.19), il 93,3% (14/15) degli utenti ha espresso valutazioni positive (60% "facile" e 33,3% "molto facile"), rispetto a un 6,7% (1/15) che ha riscontrato difficoltà nell'utilizzo.

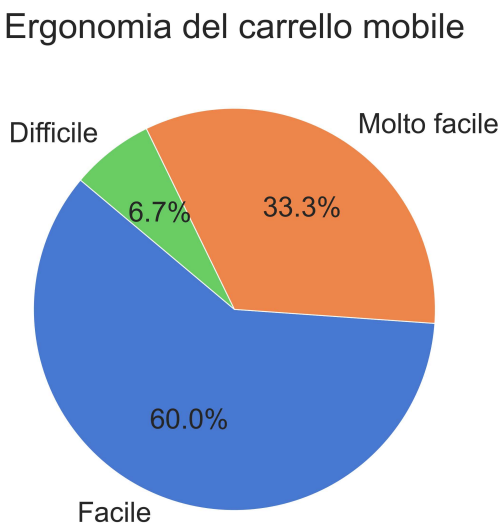

Figura 5.19: Valutazione dell'ergonomia e della gestione del carrello mobile.

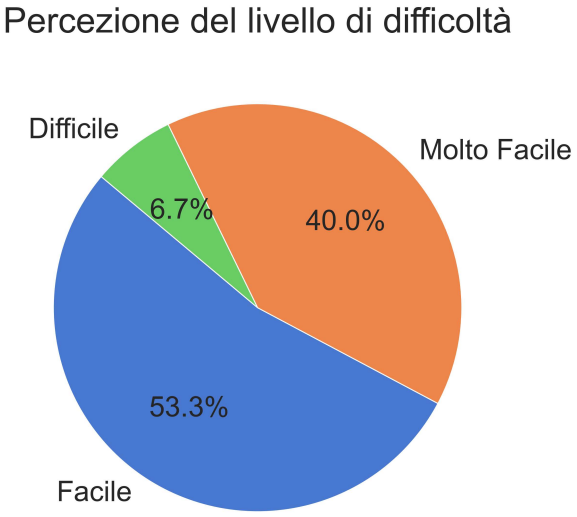

Figura 5.20: Percezione del livello di difficoltà e bilanciamento della sfida proposta.

La Figura 5.20 esamina il bilanciamento della difficoltà complessiva. Il 93,3% (14/15) dei *playtester* ha ritenuto la sfida poco punitiva (53,3% "facile" e 40% "molto facile"), mentre il 6,7% (1/15) ha giudicato l'esperienza "difficile". Nessun utente ha selezionato i valori intermedi disponibili nel questionario originario.

Comparto Tecnico ed Estetico

Il quarto blocco dell'indagine riporta i dati sulla stabilità tecnica e sull'accoglienza del comparto grafico e sonoro.

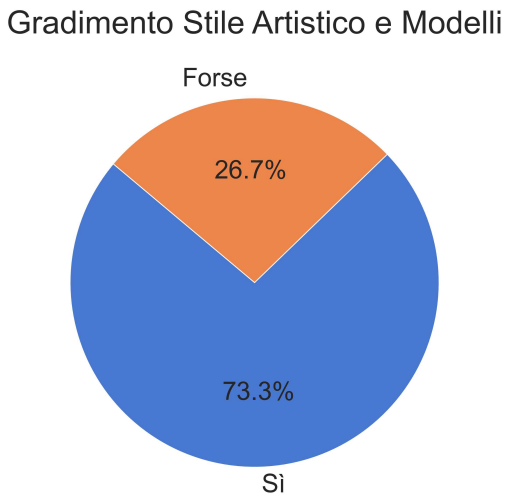

Figura 5.21: Valutazione sul gradimento dello stile artistico e dei modelli.

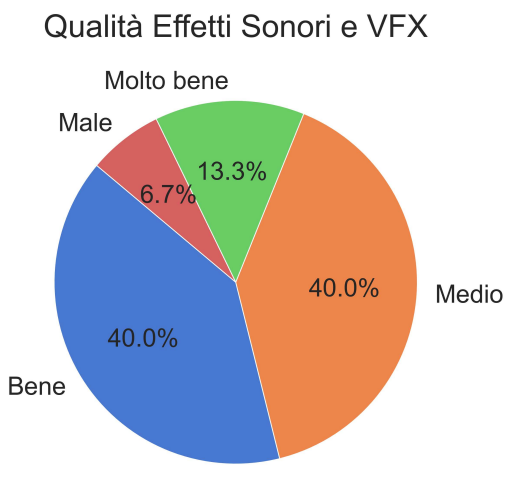

Figura 5.22: Qualità percepita degli effetti sonori e dei feedback visivi.

Figura 5.23: Metriche relative al comparto artistico ed estetico del titolo.

Riguardo al gradimento estetico (*"Did you like the game's art style and models?"*, Figura 5.21), il 73,3% (11/15) del campione ha espresso un parere favorevole, mentre il 26,7% (4/15) si è posizionato sulla valutazione "forse".

In relazione alla qualità della componente sonora e dei *feedback* visivi (Figura 5.22), il 13,3% (2/15) ha assegnato un giudizio "molto bene", il 40% (6/15) "bene", il 40% "medio" e il 6,7% (1/15) ha indicato una valutazione "male".

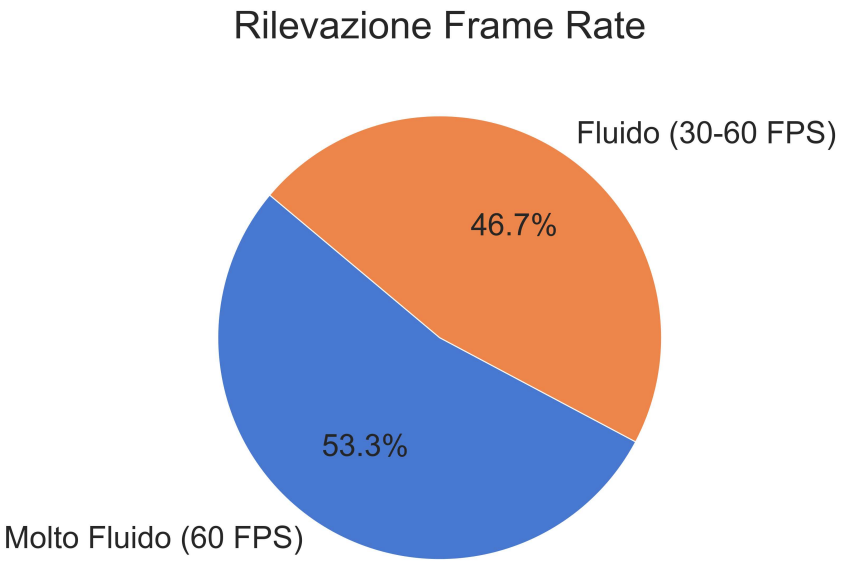

Figura 5.24: Rilevazione del livello di fluidità (Frame Rate) sui dispositivi dell'utenza.

Dal punto di vista prestazionale (Figura 5.24), il 53,3% (8/15) dei *playtester* ha registrato un *frame rate* di 60 FPS, mentre il 46,7% (7/15) ha riscontrato una fluidità compresa tra i 30 e i 60 FPS. Nessun utente ha segnalato cali di *framerate* inferiori ai 30 FPS.

Potenziale di mercato e valutazione globale

L'ultima sezione quantitativa raccoglie i dati sulle prospettive commerciali e sull'apprezzamento complessivo.

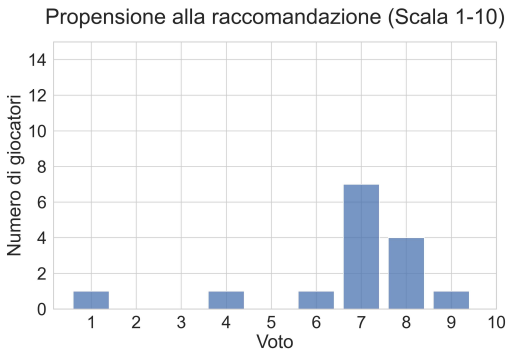

Figura 5.25: Propensione alla raccomandazione su scala decimale.

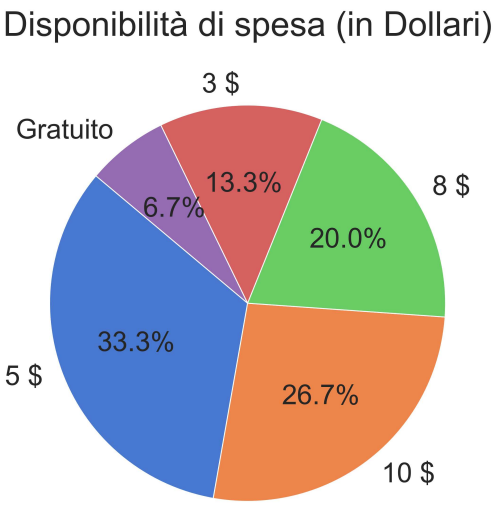

Figura 5.26: Valutazione della disponibilità di spesa (in dollari).

Figura 5.27: Metriche relative al potenziale commerciale del prodotto.

Sulla propensione alla raccomandazione (Figura 5.25), il 73,3% (11/15) del campione ha assegnato un punteggio pari a 7 o 8. Le valutazioni comprese tra 1 e 6 costituiscono il 20 % (3/15) dei voti, mentre il punteggio 9 rappresenta il 6,7% (1/15). Non sono stati registrati punteggi pari a 10.

Relativamente alla disponibilità di spesa (Figura 5.26), la fascia dei 5 dollari è stata selezionata dal 33,3% (5/15) degli intervistati. Il 20% (3/15) e il 26,7% (4/15) del campione si sono orientati rispettivamente sui prezzi di 8 e 10 dollari, un 13,3% (2/15) sulla fascia dei 3 dollari, mentre il 6,7% (1/15) ha optato per un rilascio gratuito.

5.3.2 Analisi qualitativa delle osservazioni degli utenti

A compendio dei dati statistici, i campi aperti del questionario hanno permesso di raccogliere osservazioni descrittive. I riscontri forniti dai *playtester* sono stati aggregati in quattro categorie principali.

Punti di forza percepiti dagli utenti

Dalle risposte relative agli elementi maggiormente apprezzati, gli utenti hanno citato l'ibridazione dei generi, l'uso del carrello logistico e il senso di appagamento derivante dal completamento delle mansioni primarie. Ulteriori menzioni hanno riguardato la meccanica di disinfestazione e la narrazione ambientale, in particolare per quanto concerne l'integrazione di atmosfere *horror* all'interno delle stanze VIP. Sono state inoltre evidenziate positivamente le *quest* secondarie assegnate dai personaggi non giocanti (NPC).

Criticità di usabilità e Quality of Life

L'indagine sulle componenti meno gradite ha raccolto segnalazioni relative al limite di trasporto dell'inventario, che impone la raccolta di un singolo rifiuto per volta obbligando il giocatore al *backtracking*. Altre criticità rilevate riguardano alcune ambiguità tipografiche della UI (es. la porta "3S" letta come "35"), l'assenza di un tracciamento persistente a schermo per le missioni assegnate dagli NPC e il fastidio visivo generato dall'opzione non disattivabile del *Motion Blur*.

Problemi tecnici e bug emersi

Le segnalazioni relative alla stabilità tecnica hanno identificato malfunzionamenti legati al sistema di collisione. Gli utenti hanno riportato episodi di compenetrazione (*stuck*) o di caduta attraverso le geometrie del livello (*out of bounds*) in concomitanza con l'azione di separazione dal carrello vicino a pareti o letti. Un secondo problema tecnico ha coinvolto uno *script* in fase di dialogo finale con un NPC: il sistema ha generato una sovrapposizione di linee vocali che ha causato un blocco logico (*softlock*), impedendo l'accesso all'ascensore e la conseguente prosecuzione del gioco.

Suggerimenti per sviluppi futuri

Tra le proposte per le versioni successive del prodotto, l'utenza ha espresso la volontà di poter utilizzare il denaro ottenuto nel corso del gioco per l'acquisto di potenziamenti, come l'aumento della capacità di trasporto. I *tester* hanno inoltre richiesto una maggiore interazione con la *lore* attraverso documenti o elementi di scena aggiuntivi e hanno suggerito l'inserimento di minacce attive all'interno della mappa per incrementare il tasso di sfida generale.

Capitolo 6

Discussioni

In questo capitolo vengono esaminati e discussi criticamente i riscontri empirici, le potenzialità e i limiti emersi a seguito delle scelte progettuali e architetturali intraprese dal team durante l'intera *pipeline* di sviluppo.

6.1 Analisi critica delle scelte progettuali e architetturali

La valutazione a posteriori dell'infrastruttura software sviluppata in Unreal Engine 5.6.1 permette di analizzare criticamente l'efficacia delle scelte metodologiche adottate, pesandone i benefici in termini di stabilità e pulizia del codice rispetto alle complessità introdotte in fase di sviluppo. Il pilastro portante dell'intera architettura è stato il perseguimento del massimo disaccoppiamento (*decoupling*) tra i moduli di *gameplay*, una necessità ingegneristica affrontata opponendosi alla proliferazione di collegamenti rigidi tra le classi.

6.1.1 Modularità e disaccoppiamento

L'impiego sistematico di tecniche di disaccoppiamento, in particolare *Blueprint Interfaces* ed *Event Dispatchers*, ha permesso di mitigare in modo significativo la proliferazione di *hard references* all'interno dell'architettura di gioco. Sebbene in assenza di un *profiling* tecnico rigoroso non sia possibile stabilire un nesso di causalità assoluto, è ragionevole ipotizzare che il mantenimento di un'infrastruttura di memoria snella abbia influito positivamente sulla stabilità e sulla fluidità del prodotto in fase di esecuzione. Tale premessa trova un parziale riscontro empirico nei dati presentati nel Capitolo 5, dove si evidenzia che nessun utente ha registrato prestazioni inferiori alla soglia dei 30 *frame* al secondo.

Parallelamente all'ottimizzazione logica, un ulteriore fattore che ha verosimilmente concorso a preservare le prestazioni risiede nella gestione del comparto grafico. In fase di modellazione si è prestata attenzione all'economia poligonale, affiancata dal controllo sulla risoluzione delle *texture*. La direzione artistica è stata calibrata per restituire un impatto visivo coerente con l'atmosfera *horror*, evitando tuttavia derive iper-realistiche che avrebbero saturato rapidamente l'allocazione della memoria video (VRAM). Questo compromesso sembra aver consentito di mantenere il carico di lavoro accessibile anche per macchine di fascia media.

6.1.2 Scalabilità dei sistemi

L'approccio architetturale adottato tramite *Blueprint* e lo sfruttamento della logica di ereditarietà pongono basi stabili per una potenziale espansione futura del progetto. L'infrastruttura attuale è stata concepita per agevolare l'integrazione di nuovi livelli, *asset* inediti, missioni e personaggi secondari.

L'utilizzo combinato di *Master Material*, *Actor Component* modularizzati e interfacce di comunicazione, unito alla centralizzazione del flusso degli eventi tramite il `BP_ScriptManager`, favorisce l'estensione del *gameplay* limitando la necessità di riprogrammare i sistemi da zero. Questa flessibilità logica tende a ridurre i tempi di iterazione, permettendo alle nuove meccaniche di ereditare i comportamenti standardizzati già presenti nella base di codice.

6.1.3 Gestione della fisica e dei componenti

L'introduzione del carrello (`BP_Cart`) come perno del *core loop* si è rivelata una scelta di *game design* funzionale, supportata dai riscontri positivi emersi nei questionari. Tuttavia, sotto il profilo ingegneristico, l'implementazione di questa meccanica ha rappresentato una sfida tecnica di notevole entità.

La gestione fisica di un *Actor* che funge simultaneamente da inventario visivo e da estensione controllabile del giocatore ha introdotto criticità spaziali, in particolar modo nei calcoli vettoriali durante le fasi di aggancio e sgancio (*attach* e *detach*) e nel posizionamento dinamico degli oggetti. Queste complicazioni cinematiche sono alla base delle anomalie riportate in fase di test, come le occasionali compenetrazioni o i casi di *clipping* che spingono il giocatore al di fuori dei limiti geometrici (*out of bounds*).

Ciononostante, il sistema ha dimostrato una discreta robustezza: la gestione delle collisioni durante la navigazione e l'operazione logica di *switch* dei controller (il passaggio della possessione dal *Character* al carrello) hanno restituito risultati stabili, indicando la validità concettuale della meccanica ma sottolineando la necessità di ulteriori affinamenti per risolvere le instabilità nei casi limite.

6.2 Valutazione del raggiungimento degli obiettivi di *design*

L'analisi incrociata tra i propositi concettuali e i riscontri restituiti dal campione di *playtester* consente di tracciare un bilancio sul raggiungimento degli obiettivi ludici, evidenziando le scelte funzionali e le aree suscettibili di miglioramento.

6.2.1 Ibridazione dei generi e atmosfera

Uno degli intenti primari del progetto consisteva nel fondere le dinamiche tipiche dei simulatori di pulizia con l'estetica di un titolo *horror*. La scelta di decontestualizzare il *cleaning simulator* calandolo all'interno di un hotel infernale ha fornito indicazioni

positive. I dati qualitativi illustrati nel Capitolo 5 suggeriscono che le missioni legate agli ospiti speciali e la tensione ambientale abbiano contribuito a limitare la ripetitività delle mansioni di base. Questo risultato sembra avvalorare l'efficacia dell'ibridazione e l'uso dell'*environmental storytelling* come strumento per sostenere l'ingaggio emotivo.

6.2.2 Accessibilità e approccio diegetico

Un secondo obiettivo riguardava la costruzione dell'immersività attraverso l'impiego di un tutorial diegetico e di un HUD minimale. L'indagine ha mostrato che l'apprendimento delle meccaniche è risultato ampiamente accessibile per la quasi totalità dell'utenza. Tuttavia, come emerso dai grafici presentati precedentemente, le valutazioni specifiche sulla chiarezza del tutorial sono risultate più contrastanti, con una fetta rilevante di utenti che ha espresso pareri neutrali o di disaccordo. Questo scostamento indica che, pur essendo il gioco facile da padroneggiare, il sistema di indizi ambientali necessita di essere perfezionato. Le future iterazioni richiederanno di calibrare ulteriormente l'illuminazione e i contrasti visivi per rendere l'approccio diegetico più inequivocabile.

6.2.3 Sostenibilità del *Core Loop*

La volontà di rendere gratificante una routine virtuale è stata affrontata centralizzando l'azione attorno al carrello logistico. Sebbene l'attuale *Beta* difetti ancora di sovrastrutture avanzate, come un'economia in-game, il livello di appagamento restituito dal ciclo di detersione è risultato incoraggiante. I dati sembrano indicare che le fondamenta del *core loop* siano in grado di mantenere l'interesse dell'utente, costituendo una base su cui sarà possibile stratificare future espansioni.

6.3 L'esperienza Svilupparty: valore e limiti del testing supervisionato

L'opportunità di presentare *Bloody Shift* allo Svilupparty ha rappresentato uno snodo cruciale per la validazione del progetto, fornendo un contesto di analisi significativamente diverso dal testing in remoto. La possibilità di osservare i giocatori in presenza ha consentito di mappare le immediate reazioni e di cogliere le frizioni ergonomiche nel momento esatto in cui si presentavano, superando i limiti di un questionario asincrono in cui i dettagli possono andare persi. Le indicazioni verbali fornite a caldo dall'utenza e la discussione tra colleghi sviluppatori hanno arricchito i dati quantitativi raccolti, indirizzando le priorità per gli sviluppi futuri.

È tuttavia necessario inquadrare i risultati ottenuti tenendo conto delle dinamiche fisiologiche di un evento fieristico. La postazione in loco comporta intrinsecamente una potenziale alterazione dell'esperienza: la presenza costante dei membri del team

alle spalle del giocatore potrebbe aver generato un effetto di "desiderabilità sociale", inducendo l'utente a un giudizio più indulgente. Inoltre, l'ambiente rumoroso della fiera, unito alla necessità di mantenere sessioni brevi per permettere la rotazione dei partecipanti, ha inevitabilmente limitato la fruizione della componente audio e frammentato il livello di immersione, due pilastri essenziali in un prodotto di stampo *horror*. L'intervento occasionale degli sviluppatori nei momenti di stallo prolungato ha evitato l'insorgere della frustrazione, ma ha anche attutito i difetti del tutorial diegetico, spiegando in parte perché l'apprendimento sia risultato percepito come "facile" nonostante le ambiguità progettuali. Tali considerazioni confermano il valore inestimabile del test dal vivo, ma impongono di pesare i dati raccolti come indicazioni orientative per una fase *Beta*, rimandando a futuri collaudi in ambiente isolato la validazione definitiva.

6.4 Limiti del lavoro e criticità emerse in fase di sviluppo

Le tempistiche imposte dalla *pipeline* di produzione hanno fisiologicamente limitato la fase di rifinitura, rendendo i *playtest* determinanti per l'individuazione delle lacune progettuali. Sotto il profilo del *game design* e dell'usabilità, la criticità più marcata ha riguardato il sistema di trasporto e smaltimento dei rifiuti tramite la *trash bag*. La costrizione a smaltire un singolo oggetto per volta ha generato un ripetitivo *backtracking*, un fattore che rischia di appesantire il ritmo di gioco nelle sessioni prolungate. A questo si lega la necessità di rendere l'intero sistema di interazione più accattivante: una parte dei collaudatori ha evidenziato l'assenza di un incentivo forte che stimoli la progressione fino alla conclusione dell'esperienza. Tale mancanza suggerisce la necessità di esplicitare con maggiore chiarezza le premesse narrative e il ruolo dell'utente all'interno dell'hotel infernale, al fine di mantenere alta la motivazione del giocatore.

Sul fronte tecnico, i collaudi hanno evidenziato instabilità strutturali e bug ancora presenti nella versione attuale. La problematica più grave si verifica durante l'azione di distacco (*detach*) dal carrello delle pulizie: il sistema di collisioni può fallire, causando la compenetrazione del *Player Character* negli oggetti di scena o proiettandolo all'interno di stanze non ancora sbloccate, bloccando di fatto la progressione del giocatore (*softlock*). Inoltre, il `BP_ScriptManager` ha mostrato incertezze nella gestione della coda degli eventi, sfociando in sovrapposizioni delle linee di dialogo che talvolta corrompono il corretto flusso della logica di gioco. Infine, l'utenza ha segnalato fastidi visivi derivanti da un'implementazione aggressiva del *Motion Blur*, sottolineando l'utilità di un menu grafico per poterne modulare l'intensità.

In conclusione, è doveroso discutere in modo esplicito i limiti metodologici della fase di collaudo. In primo luogo, lo strumento di valutazione impiegato è un questionario costruito internamente dal team di sviluppo e somministrato a un campione di convenienza di 15 partecipanti. Sebbene costituisca una base esplorativa utile, tale campione risulta numericamente troppo esiguo, rendendo necessari futuri *playtest* su

scala più ampia e con profili di videogiocatori maggiormente diversificati. In secondo luogo, l'analisi ha aggregato i dati provenienti dai test supervisionati in presenza allo Svilupparty e dai test condotti da remoto. Questa unione rappresenta un limite metodologico intrinseco: i test in fiera hanno beneficiato dell'osservazione diretta e del supporto in tempo reale, mentre i collaudi da remoto mancavano di controllo ambientale, un fattore che potrebbe aver introdotto *bias* nelle valutazioni. Infine, è importante precisare che i dati relativi alle prestazioni tecniche, come la tenuta del *frame rate*, derivano esclusivamente dalle dichiarazioni soggettive fornite dagli utenti, e non da misurazioni oggettive effettuate tramite strumenti di *profiling* del motore grafico.

6.5 Valutazione della *pipeline* produttiva in team

La valutazione della *pipeline* produttiva restituisce un bilancio positivo in merito alla gestione del flusso di lavoro condiviso. L'adozione di GitKraken come *client* grafico ha facilitato la sincronizzazione della *codebase*. Data la natura binaria dei file *asset* di Unreal Engine, il rischio di sovrascritture è stato arginato attraverso una rigorosa divisione delle responsabilità e una comunicazione costante. Assegnando l'esclusività di modifica su specifici *Blueprint* a ciascun programmatore, è stato possibile operare in parallelo mitigando la generazione di conflitti logistici.

Questa compartimentazione dei compiti sembra aver giovato ai tempi di sviluppo. Un ruolo rilevante è stato ricoperto dalla fase di pre-produzione: la pianificazione architetturale preliminare ha fornito linee guida operative per l'implementazione logica delle meccaniche, pur mantenendo il margine di flessibilità necessario per accogliere le revisioni in corso d'opera.

Infine, la collaborazione interdisciplinare tra programmatori e modellatori 3D ha permesso di ottimizzare la creazione degli elementi grafici. Il ciclo di *feedback* continuo ha favorito l'allineamento dei modelli ai vincoli architetturali del codice, ai punti di aggancio (*socket*) e alle metriche della mappa, contribuendo alla coerenza tecnica ed estetica dell'opera limitando la necessità di onerosi rimaneggiamenti in fase di integrazione.

6.6 Confronto interpretativo con i prodotti di riferimento

Per valutare l'effettivo posizionamento di *Bloody Shift* all'interno del mercato, risulta utile proporre un breve confronto interpretativo, di natura non sperimentale, con i *competitor* esaminati nello Stato dell'Arte. Rispetto a *Viscera Cleanup Detail*, il progetto riprende la tematica del lavoro manuale decontestualizzato, ma ne snellisce deliberatamente i ritmi, abbandonando la lentezza metodica e le fisiche punitive in favore di un approccio più permissivo. Rispetto a *Crime Scene Cleaner*, l'applicativo eredita il concetto di una narrazione integrata e della progressione guidata da *quest*, ma se ne distanzia abbandonando il realismo investigativo per abbracciare

un'estetica grottesca e irriverente. Infine, l'influenza di *The WereCleaner* risulta evidente nella scelta di contaminare le meccaniche di pulizia con l'imprevedibilità di personaggi non giocanti (NPC), sebbene *Bloody Shift* cerchi di sostituire le dinamiche puramente *stealth* con una frenesia più affine al genere *arcade*. Questa sintesi progettuale suggerisce il tentativo di unire gli elementi più efficaci dei predecessori, pur necessitando di ulteriori rifiniture per raggiungere analoga profondità.

Capitolo 7

Conclusioni e Sviluppi Futuri

Il presente elaborato ha documentato la progettazione, l'implementazione e la valutazione di *Bloody Shift*, un videogioco tridimensionale sviluppato in Unreal Engine 5.6.1 prevalentemente attraverso il sistema di *visual scripting* Blueprint. Il lavoro ha seguito l'intero ciclo produttivo: dall'analisi dei titoli concorrenti e definizione dei requisiti di *game design*, fino allo sviluppo, all'integrazione degli asset e al collaudo.

Dal punto di vista tecnico, il progetto ha mostrato la fattibilità dello sviluppo di un prototipo videoludico completo mediante Blueprint. L'impiego di *Blueprint Interfaces* ed *Event Dispatchers* ha limitato il ricorso al *Casting* tra classi personalizzate, riducendo le dipendenze rigide tra i moduli. Questa impostazione ha favorito la suddivisione delle responsabilità e l'integrazione parallela del lavoro del team. Tra i sistemi principali rientrano il carrello delle pulizie, le porte interattive, i personaggi non giocanti infestanti, l'illuminazione dinamica e il `BP_ScriptManager`, incaricato di coordinare onboarding e progressione.

I dati prestazionali raccolti durante il collaudo hanno fornito indicazioni preliminari positive: nessuno dei quindici partecipanti ha segnalato un frame rate inferiore ai 30 FPS. Tale risultato non consente tuttavia di quantificare l'effetto delle scelte architetturali, poiché le prestazioni non sono state misurate mediante strumenti di profiling e l'hardware non è stato controllato. Le soluzioni adottate vanno pertanto considerate strategie progettuali orientate alla modularità, non ottimizzazioni dimostrate quantitativamente.

Sotto il profilo del *game design*, *Bloody Shift* ha fuso le meccaniche dei *cleaning simulator* con un'ambientazione horror-grottesca, caratterizzata da ritmi rapidi e obiettivi a breve termine. Il questionario, somministrato a quindici partecipanti (di cui dieci in sessioni supervisionate allo Svilupparty 2026), ha evidenziato una ricezione positiva. Quattordici utenti su quindici hanno apprezzato la facilità di apprendimento e l'uso del carrello mobile, mentre il livello di appagamento del ciclo di pulizia ha ottenuto valutazioni tra 8 e 10 da dieci partecipanti.

È emersa tuttavia una discrepanza sulla chiarezza del tutorial diegetico: solo sette utenti su quindici hanno valutato positivamente le istruzioni, mentre sei sono risultati neutrali e due negativi. Ciò suggerisce che l'illuminazione dinamica, la telecamera e i feedback emissivi costituiscano una buona base per l'onboarding, ma necessitino di maggiore evidenza visiva e coerenza nei momenti critici.

Il testing ha inoltre individuato alcune limitazioni tecniche, in primis il sistema di distacco dal carrello, che può causare compenetrazioni o il posizionamento del giocatore fuori dai limiti previsti. Ulteriori criticità riguardano occasionali sovrapposizioni dei dialoghi gestiti dal `BP_ScriptManager`, l'eccessivo *backtracking* dovuto al trasporto di un solo rifiuto per volta, alcune ambiguità nell'interfaccia e l'impossibilità di disattivare il *Motion Blur*.

A breve termine, le attività future dovranno rendere più robusto il riposizionamento del giocatore durante il *detach*, introdurre controlli sulla coda degli eventi narrativi e ampliare le opzioni grafiche. Parallelamente, sarà opportuno migliorare la leggibilità del tutorial e ridurre i tempi morti della raccolta rifiuti, ad esempio incrementando la capacità di trasporto.

In ottica futura, l'architettura modulare agevolerà l'aggiunta di nuovi ambienti e missioni. I feedback indicano inoltre forte interesse verso una maggiore profondità narrativa (*lore*), l'introduzione di minacce attive e un'economia interna per l'acquisto di potenziamenti. Per consolidare i risultati serviranno test con un campione più ampio, separando i dati supervisionati da quelli da remoto, e affiancando ai questionari misurazioni oggettive tramite *profiler* per valutare usabilità, memoria e stabilità.

In conclusione, *Bloody Shift* costituisce un solido caso di studio sull'impiego dei Blueprint in un progetto indipendente e sull'ibridazione tra simulazione e horror. Pur non essendo ancora pronto per la distribuzione commerciale, individua una chiara base tecnica e progettuale su cui costruire successive iterazioni in vista di un'eventuale pubblicazione su piattaforme come Steam.

Bibliografia e sitografia

[1] Epic Games, "Blueprint visual scripting in unreal engine," Unreal Engine Documentation, 2024, disponibile online: https://docs.unrealengine.com/5.3/en-US/blueprints-visual-scripting-in-unreal-engine/ (ultimo accesso: aprile 2026).

[2] Epic Games, "Coding in unreal engine: Blueprint vs. c++," Unreal Engine Documentation, 2024, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/coding-in-unreal-engine-blueprint-vs-cplusplus (ultimo accesso: aprile 2026).

[3] R. Nystrom, *Game Programming Patterns*. Genever Benning, 2014.

[4] D. Tach, "The rise of cozy games," Epic Games Store News, 2025, disponibile online: https://store.epicgames.com/news/best-cozy-games-why-cozy-feature-interview?lang=en-US (ultimo accesso: giugno 2026).

[5] A. Waszkiewicz and M. Tymińska, "Cozy games and resistance through care," *Replay. The Polish Journal of Game Studies*, vol. 11, no. 1, pp. 7–16, 2024.

[6] RuneStorm, "Viscera cleanup detail," Steam Store Page, disponibile online: https://store.steampowered.com/app/246900/Viscera_Cleanup_Detail/ (ultimo accesso: giugno 2026).

[7] D. Whitehead, "Viscera cleanup detail early access review," Eurogamer, 2014, disponibile online: https://www.eurogamer.net/viscera-cleanup-detail-early-access-review (ultimo accesso: giugno 2026).

[8] President Studio, "Crime scene cleaner," Steam Store Page, disponibile online: https://store.steampowered.com/app/1040200/Crime_Scene_Cleaner/ (ultimo accesso: giugno 2026).

[9] Gare, "Crime scene cleaner review – cleaning up after criminals has never been this relaxing," GTOGG, 2024, disponibile online: https://www.gtogg.com/News/en/Crime_Scene_Cleaner_Review_Cleaning_up_after_criminals_has_never_been_this_relaxing/ (ultimo accesso: giugno 2026).

[10] Playday, "Crime scene cleaner review," Playday.one, 2025, disponibile online: https://playday.one/2025/04/21/crime-scene-cleaner-review/ (ultimo accesso: giugno 2026).

[11] A. Acevedo, "The werecleaner review: Werewolves against capitalism," MetaStellar, 2024, disponibile online: https://www.metastellar.com/nonfiction/reviews/the-werecleaner-review/ (ultimo accesso: giugno 2026).

[12] Howlin' Hugs, "The werecleaner," Steam Store Page, disponibile online: https://store.steampowered.com/app/2795000/The_WereCleaner/ (ultimo accesso: giugno 2026).

[13] O. Davies, "The werecleaner review — a bite-sized indie," GamingTrend, 2025, disponibile online: https://gamingtrend.com/reviews/the-werecleaner-review-a-bite-sized-indie/ (ultimo accesso: giugno 2026).

[14] Valve Corporation, "Steam hardware & software survey," Steam, disponibile online: https://store.steampowered.com/hwsurvey/ (ultimo accesso: maggio 2026).

[15] A. Stonehouse, "User interface design in video games," Game Developer (ex Gamasutra), 2014, disponibile online: https://www.gamedeveloper.com/design/user-interface-design-in-video-games (ultimo accesso: maggio 2026).

[16] GitKraken, "Gitkraken - legendary git tools," Sito Ufficiale, disponibile online: https://www.gitkraken.com/ (ultimo accesso: maggio 2026).

[17] Epic Games, "Source control in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/source-control-in-unreal-engine (ultimo accesso: maggio 2026).

[18] Epic Games, "Skeletal mesh animation system," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/skeletal-mesh-assets-in-unreal-engine (ultimo accesso: maggio 2026).

[19] Epic Games, "Cinematic cameras in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/cinematic-cameras-in-unreal-engine (ultimo accesso: maggio 2026).

[20] Epic Games, "Tracing with raycasts in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/traces-in-unreal-engine---overview (ultimo accesso: maggio 2026).

[21] Epic Games, "Decal actors in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/decal-actors-in-unreal-engine (ultimo accesso: maggio 2026).

[22] P. J. Deitel and H. M. Deitel, *Java: How to Program, Early Objects*, 11th ed. Pearson, 2017.

[23] Epic Games, "Collision in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/collision-in-unreal-engine---overview (ultimo accesso: maggio 2026).

[24] Epic Games, "Umg ui designer in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/widget-blueprints?application_version=4.27 (ultimo accesso: maggio 2026).

[25] Epic Games, "Game instance in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/community/learning/tutorials/pw96/unreal-engine-data-persistence-game-instance-vs-level-blueprint (ultimo accesso: maggio 2026).

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[27] R. Jones, A. Hosking, and E. Moss, *The Garbage Collection Handbook: The Art of Automatic Memory Management*. Chapman and Hall/CRC, 2011.

[28] Epic Games, "Player controller in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/player-controllers-in-unreal-engine (ultimo accesso: maggio 2026).

[29] Epic Games, "Enhanced input system in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/enhanced-input-in-unreal-engine (ultimo accesso: maggio 2026).

[30] Epic Games, "Game mode and game state in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/game-mode-and-game-state-in-unreal-engine (ultimo accesso: maggio 2026).

[31] Epic Games, "Casting quick start guide in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/casting-quick-start-guide-in-unreal-engine (ultimo accesso: maggio 2026).

[32] Epic Games Developer Community, "Myth busting: Best practices in unreal engine," Epic Developer Community, disponibile online: https://dev.epicgames.com/community/learning/tutorials/l3E0/myth-busting-best-practices-in-unreal-engine (ultimo accesso: maggio 2026).

[33] Epic Games, "Blueprint interfaces in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprint-interface-in-unreal-engine (ultimo accesso: maggio 2026).

[34] Epic Games, "Event dispatchers in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/event-dispatchers-in-unreal-engine (ultimo accesso: maggio 2026).

[35] Epic Games, "Actors in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/actors-in-unreal-engine (ultimo accesso: maggio 2026).

[36] Epic Games, "Timelines in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/timelines-in-unreal-engine (ultimo accesso: maggio 2026).

[37] Epic Games, "Static meshes in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/static-meshes (ultimo accesso: maggio 2026).

[38] Epic Games, "Set timer by function name - blueprint api reference," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Utilities/Time/SetTimerbyFunctionName (ultimo accesso: maggio 2026).

[39] Epic Games, "Physically based materials in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/en-us/unreal-engine/physically-based-materials-in-unreal-engine (ultimo accesso: maggio 2026).

[40] Epic Games, "Audio files in unreal engine," Unreal Engine Documentation, disponibile online: https://dev.epicgames.com/documentation/unreal-engine/importing-audio-files (ultimo accesso: maggio 2026).