



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Game AI: Progettazione e Sviluppo di una Componente di Intelligenza Artificiale nel Contesto di Videogame

RELATORE

Prof. Fabio Palomba

Università degli Studi di Salerno

CANDIDATO

Luigi Salvatore Pio Petrillo

Matricola: 0512114122

Questa tesi è stata realizzata nel



A chi crede di non potercela fare.

Abstract

I videogiochi, negli ultimi anni, hanno compiuto enormi progressi sotto il profilo grafico, diventando sempre più realistici e immersivi. Tuttavia, un aspetto che non ha seguito lo stesso ritmo di evoluzione è l'intelligenza artificiale degli NPC (personaggi non giocanti), in particolare nel genere horror.

Questa tesi descrive la progettazione e lo sviluppo di un'Intelligenza Artificiale capace di aumentare il realismo comportamentale degli NPC all'interno di un videogioco. Il gioco, realizzato con il motore Unity, propone un'esperienza in cui il giocatore può interagire con diversi oggetti, esplorare l'ambiente e trovare la chiave per accedere a una stanza segreta, il tutto cercando di evitare l'avvicinarsi del killer.

Il nemico sviluppato, è in grado di analizzare le azioni del giocatore per dedurre il suo stile di gioco e adattare di conseguenza il proprio comportamento.

L'obiettivo del progetto è mostrare come l'integrazione di una IA più sofisticata possa migliorare significativamente l'esperienza videoludica, rendendola più coinvolgente, dinamica e realistica.

Indice

Elenco delle Figure	iii
1 Introduzione	1
1.1 Contesto Applicativo	1
1.2 Motivazioni e Obiettivi	2
1.3 Risultati	2
1.4 Struttura della Tesi	2
2 Stato dell'Arte	3
2.1 Videogioco	3
2.1.1 Gioco Procedurale	6
2.1.2 Grafica	8
2.1.3 Fisica	9
2.1.4 Intelligenza Artificiale	10
2.2 L'evoluzione dell'Intelligenza Artificiale nei Videogiochi	11
2.2.1 Origini e Teorie dell'Intelligenza Artificiale	11
2.2.2 Principali Ambiti dell'IA Applicata	12
3 Applicazione Sviluppata	14
3.1 Progettazione del Videogioco	14
3.1.1 Studio della Mappa di Gioco	15

3.1.2 Scelta dell'Art Design	16
3.2 Sviluppo dell'Ambiente in Unity	17
3.2.1 Organizzazione degli elementi	18
3.2.2 Luci e Suoni	19
3.3 Integrazione dell'IA	20
3.3.1 Implementazione della Libreria AI Navigation	21
3.3.2 Avvistamento del Player	23
3.3.3 Player non visibile	35
4 Conclusioni	51
4.1 Sviluppi Futuri e Possibili Miglioramenti	52
Bibliografia	53

Elenco delle figure

2.1	Macchina che eseguiva OXO	4
2.2	Schermata di gioco di OXO	4
2.3	Esempio di partita a Pong	5
2.4	Copertina di Beneath Apple Manor	6
2.5	Mappa generata proceduralmente	6
2.6	Mappa di Minecraft generata proceduralmente	7
2.7	Pianeti in No Man's Sky	7
2.8	Copertina Terraria	8
2.9	Copertina Inmost	8
2.10	Panorama Red Dead Redemption 2	8
2.11	Panorama Horizon Zero Dawn	8
2.12	Fisica di SpaceWar	9
2.13	Fisica della distruttibilità	9
2.14	Interazioni fisiche complesse	9
3.1	Planimetria della mappa di gioco.	16
3.2	Camera da Letto.	18
3.3	Cucina.	18
3.4	Nascondigli del player.	18
3.5	Doccia che funge da nascondiglio.	18

3.6 Esempio Torcia Accesa	19
3.7 Esempio Torcia Spenta	19
3.8 In celeste l'area percorribile per l'NPC	21

CAPITOLO 1

Introduzione

1.1 Contesto Applicativo

Attualmente esistono più di 100.000 titoli videoludici differenti e, nonostante questa vasta disponibilità, è sorprendente notare come oltre il 70% dei giocatori ritenga che le intelligenze artificiali (IA) all'interno dei videogiochi manchino di consapevolezza situazionale [1]. Questo dato evidenzia un diffuso senso di insoddisfazione nei confronti delle IA videoludiche, nonostante i notevoli progressi tecnologici compiuti negli ultimi anni.

Solamente di recente stanno emergendo i primi prototipi di IA avanzata, come Nvidia ACE¹ o Suck UP². Tuttavia, si tratta ancora di tecnologie sperimentali, non definitive né largamente adottate nel panorama videoludico.

¹NACE – <https://developer.nvidia.com/ace>

²SuckUp – <https://www.playsuckup.com/>

1.2 Motivazioni e Obiettivi

Questa tesi nasce dall'idea di integrare nei videogiochi un'intelligenza artificiale realmente reattiva, in grado di apprendere lo stile di gioco del giocatore e rispondere di conseguenza, al fine di ottenere un'esperienza più realistica e coinvolgente.

A tal fine è stato utilizzato **Unity**, uno dei motori grafici più diffusi per la realizzazione di videogiochi. L'ambiente di gioco sviluppato consiste in un'abitazione suddivisa in diverse stanze, in cui è presente un personaggio non giocante (NPC) dotato di intelligenza artificiale. Il progetto è stato pensato per essere fruito in modalità desktop.

1.3 Risultati

L'introduzione dell'IA per il comportamento dell'NPC ha permesso di migliorare significativamente l'atmosfera horror del gioco e la dinamicità complessiva dell'esperienza. L'interazione con un personaggio in grado di adattarsi e rispondere alle azioni del giocatore rende il gameplay più immersivo e avvincente.

Inoltre, durante lo sviluppo è stato possibile approfondire l'utilizzo di alcune librerie specifiche per l'implementazione dell'intelligenza artificiale, ampliando le competenze tecniche legate a questo ambito.

1.4 Struttura della Tesi

La tesi è articolata in quattro capitoli principali:

1. **Introduzione**, in cui viene presentato il contesto, le motivazioni e gli obiettivi del lavoro.
2. **Stato dell'arte**, che analizza l'evoluzione dei videogiochi e delle tecniche di intelligenza artificiale a essi applicate.
3. **Applicazione sviluppata**, dove si descrive nel dettaglio il progetto, le tecnologie e le librerie utilizzate.
4. **Conclusioni**, con una sintesi dei risultati ottenuti e possibili sviluppi futuri.

CAPITOLO 2

Stato dell'Arte

2.1 Videogioco

Cos'è un videogioco? Il videogioco è un'opera multimediale interattiva. Questa definizione evidenzia tre aspetti fondamentali:

1. **Opera:** Il videogioco è un prodotto culturale e il risultato di un lavoro intellettuale, al pari di altre forme artistiche come il cinema o la letteratura.
2. **Multimediale:** Integra diversi linguaggi espressivi, tra cui immagini, suoni e testi, creando un'esperienza sensoriale complessa.
3. **Interattiva:** Richiede la partecipazione attiva del giocatore, che interagisce con il contenuto e ne influenza lo sviluppo.

Tutte queste caratteristiche sono il frutto di una progressiva evoluzione che ha reso i videogiochi sempre più sofisticati, fino a essere oggi considerati vere e proprie forme d'arte [2].

Il videogioco nasce negli anni '50, in ambito accademico e scientifico, ma è solo negli anni '70 che inizia a diffondersi commercialmente.

Storia dei Videogiochi

Il primo videogioco mai realizzato, seppur in modo involontario, fu OXO (2.2), una versione digitalizzata del gioco del tris, sviluppata nel 1952 presso l'Università di Cambridge dallo studente Alexander S. "Sandy" Douglas, come esempio per la sua tesi di dottorato¹.



Figura 2.1: Macchina che eseguiva OXO



Figura 2.2: Schermata di gioco di OXO

¹<https://it.wikipedia.org/wiki/Videogioco>

Il primo videogioco a scopo ludico e commercializzato fu invece *Pong*², prodotto da Atari nel 1972 come arcade e nel 1975 come console domestica [3]. Si trattava di un simulatore di ping-pong con una grafica bidimensionale estremamente semplificata, in bianco e nero (2.3). Il giocatore controllava una racchetta, rappresentata da una barra verticale, per colpire una palla e segnare punti contro l'avversario.

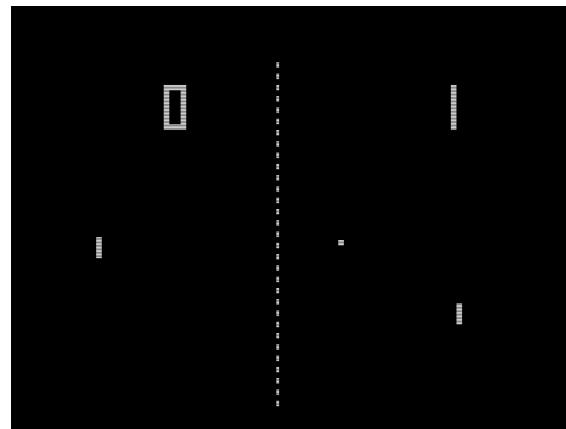


Figura 2.3: Esempio di partita a Pong

²<https://it.wikipedia.org/wiki/Pong>

2.1.1 Gioco Procedurale

La generazione procedurale nei videogiochi si riferisce alla creazione automatica di contenuti tramite algoritmi, piuttosto che tramite progettazione manuale. Questo approccio consente di risparmiare tempo nello sviluppo e aumenta significativamente la rigiocabilità, poiché ogni partita può risultare diversa.

Il primo gioco a introdurre tale tecnica fu *Beneath Apple Manor* (1978) [4], dove l’obiettivo era recuperare una mela d’oro in una mappa generata proceduralmente (2.5).

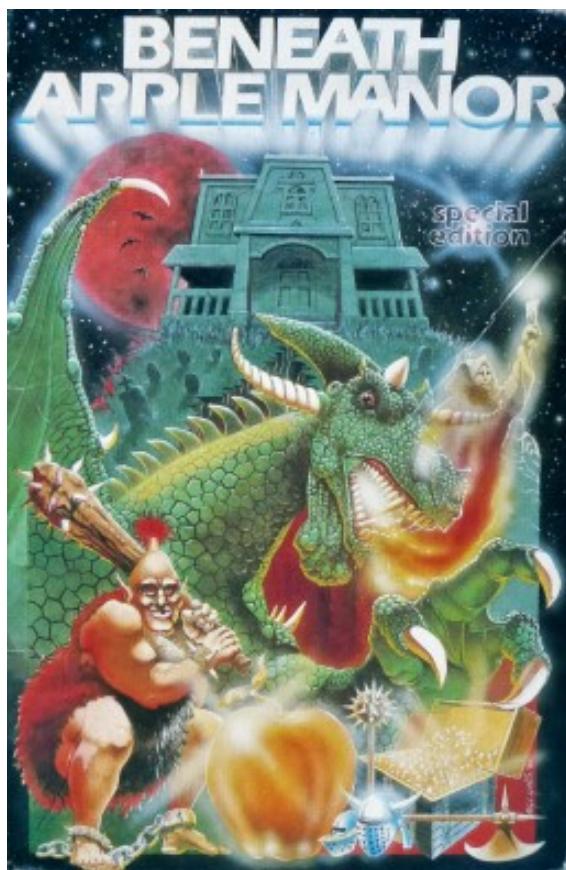


Figura 2.4: Copertina di Beneath Apple Man-
nor

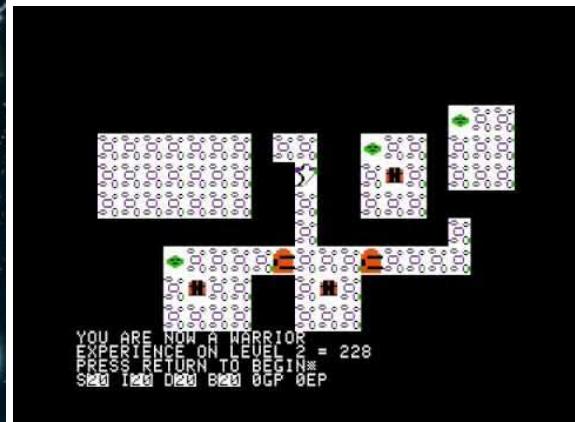


Figura 2.5: Mappa generata proceduralmen-
te

Giochi più moderni come *Minecraft* (2.6) e *No Man's Sky* (2.7) ne rappresentano evoluzioni avanzate, sfruttando la generazione procedurale per creare mondi vasti e unici.



Figura 2.7: Pianeti in No Man's Sky

Figura 2.6: Mappa di Minecraft generata
proceduralmente

2.1.2 Grafica

La grafica è uno degli aspetti che ha subito la maggiore evoluzione:

Anni '80: Dominava la pixel art, con sprite bidimensionali realizzati da pixel colorati. Questo stile, pur semplice, è ancora oggi apprezzato e utilizzato soprattutto dai giochi indie, sviluppati da singoli o piccoli team, come *Terraria* 2.8, *Moonlighter* e *Inmost* 2.9.



Figura 2.8: Copertina Terraria



Figura 2.9: Copertina Inmost

Anni '90: Vengono introdotti modelli poligonali tridimensionali e nuove tecniche di rendering. Uno dei primi esempi fu *Spasim*³, un gioco 3D in wireframe. Questo progresso ha aperto la strada a produzioni moderne di grande impatto visivo come *Red Dead Redemption 2* 2.10 o *Horizon Zero Dawn* 2.11 [5].



Figura 2.10: Panorama Red Dead Redemption 2



Figura 2.11: Panorama Horizon Zero Dawn

³<https://en.wikipedia.org/wiki/Spasim>

2.1.3 Fisica

La simulazione fisica è sempre stata presente nei videogiochi, anche in titoli storici come *Spacewar!*⁴ 2.12, dove venivano simulate le forze gravitazionali. Con il tempo, la fisica nei videogiochi è diventata più sofisticata: *Teardown e Instruments of Destruction* 2.13 introdusse un realismo alle distrubilità mai vista, mentre *The Legend of Zelda: Breath of the Wild* 2.14 permise interazioni fisiche complesse, come la conduzione elettrica tramite oggetti metallici e liquidi. [6]



Figura 2.12: Fisica di SpaceWar

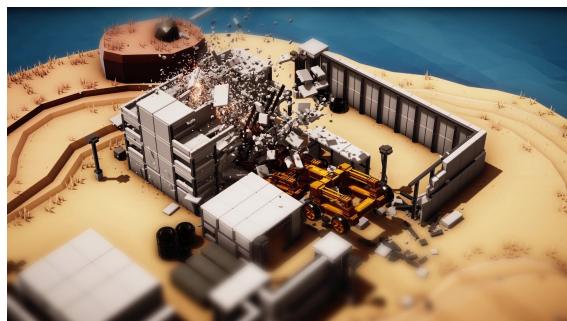


Figura 2.13: Fisica della distruttibilità



Figura 2.14: Interazioni fisiche complesse

⁴<https://it.wikipedia.org/wiki/Spacewar!>

2.1.4 Intelligenza Artificiale

L'intelligenza artificiale (IA) è fondamentale nei giochi, in particolare per il comportamento degli NPC. Fin dagli albori, come nel caso di *OXO*, è stata utilizzata per creare sfide logiche. L'IA in *OXO* era semplice e deterministica, basata su regole predefinite. Anche se primitiva, ha posto le basi per lo sviluppo di NPC sempre più complessi e realistici. Questo tema sarà approfondito nei paragrafi successivi.

2.2 L’evoluzione dell’Intelligenza Artificiale nei Videogiochi

L’Intelligenza Artificiale (IA), nel suo significato più generale, è la capacità di un sistema artificiale di emulare forme di intelligenza tipiche dell’essere umano, come il ragionamento logico, l’apprendimento, la pianificazione, la risoluzione di problemi e la creatività⁵. Nel contesto dei videogiochi, l’IA riveste un ruolo fondamentale nella creazione di comportamenti realistici per personaggi non giocanti (NPC), nella generazione dinamica di contenuti e nell’adattamento dell’esperienza di gioco alle azioni del giocatore.

2.2.1 Origini e Teorie dell’Intelligenza Artificiale

Le basi teoriche dell’IA risalgono agli anni ’50, periodo in cui l’informatico britannico Alan Turing pubblicò l’articolo Computing Machinery and Intelligence, in cui introduceva il celebre Test di Turing. Secondo questo test, una macchina può essere considerata “intelligente” se, in una conversazione scritta, un essere umano non è in grado di distinguerla da un altro essere umano. [7]

A partire da queste premesse, si sono sviluppate due principali correnti di pensiero riguardanti l’IA:

- **Intelligenza Artificiale Forte:** mira allo sviluppo di sistemi dotati di una vera e propria coscienza e consapevolezza, in grado di replicare l’intelligenza umana in tutte le sue sfaccettature. Questo tipo di IA è ancora in fase teorica e rappresenta una delle frontiere più affascinanti (e controverse) della ricerca.
- **Intelligenza Artificiale Debole:** si concentra su algoritmi in grado di eseguire compiti specifici in modo efficiente, senza avere coscienza o comprensione del contesto. Questa è la forma di IA attualmente più diffusa e utilizzata, anche nel mondo videoludico.

⁵https://it.wikipedia.org/wiki/Intelligenza_artificiale

2.2.2 Principali Ambiti dell’IA Applicata

Nel campo dell’informatica e dei videogiochi, l’IA si suddivide in diversi sotto-campi. Di seguito vengono descritte le principali aree di applicazione:

Machine Learning (Apprendimento Automatico): Il machine learning è una branca dell’IA che consente ai sistemi di apprendere dai dati e migliorare nel tempo senza essere esplicitamente programmati. Si suddivide in tre principali modalità di apprendimento:

- *Apprendimento supervisionato*: il sistema viene addestrato su un insieme di dati etichettati, ossia con input e output noti. Questo approccio è molto efficace per compiti come il riconoscimento di immagini o la classificazione di contenuti.
- *Apprendimento non supervisionato*: non vengono forniti output target, e il sistema deve identificare autonomamente pattern e correlazioni nei dati. Viene spesso impiegato per il clustering e l’analisi esplorativa.
- *Apprendimento per rinforzo (Reinforcement Learning)*: il sistema interagisce con un ambiente dinamico e apprende tramite un meccanismo di premi e penalità. Questo tipo di apprendimento è molto utilizzato nei videogiochi per creare agenti intelligenti capaci di migliorare le proprie strategie nel tempo.

Natural Language Processing (NLP): Il Natural Language Processing è la disciplina dell’IA dedicata alla comprensione, interpretazione e generazione del linguaggio naturale umano. È usato in numerose applicazioni, tra cui:

- Traduzione automatica;
- Generazione automatica di dialoghi in videogiochi o chatbot;
- Sistemi di risposta alle domande;
- Sintesi e riassunti automatici di testi;
- Interazione vocale tramite assistenti digitali.

2.2 – L’evoluzione dell’Intelligenza Artificiale nei Videogiochi

Nel contesto videoludico, il NLP sta trovando impieghi innovativi: dai dialoghi dinamici con gli NPC, fino a giochi che utilizzano il linguaggio naturale come meccanica principale.

CAPITOLO 3

Applicazione Sviluppata

3.1 Progettazione del Videogioco

Il progetto di tesi è stato sviluppato durante il tirocinio formativo interno all'università, sotto la supervisione del professore **Fabio Palomba**. È stato proprio il docente a proporre l'idea di progettare un *videogioco horror*, con particolare attenzione allo sviluppo di un'**Intelligenza Artificiale** dedicata al comportamento di un personaggio non giocante (NPC). La proposta nasce anche dalla consapevolezza, da parte del professore, della mia passione per il mondo videoludico, elemento che ha rappresentato una forte motivazione personale nello sviluppo del lavoro. In questo capitolo verranno analizzate nel dettaglio tutte le fasi di **progettazione e realizzazione** del progetto, dall'ideazione iniziale, alle scelte tecnologiche, fino all'implementazione concreta delle meccaniche di gioco e del comportamento dell'IA.

3.1.1 Studio della Mappa di Gioco

Il primo passo nello sviluppo del progetto è stato individuare una mappa adatta agli obiettivi del gioco. Per creare un ambiente credibile e funzionale, sono state analizzate le mappe di alcuni titoli appartenenti alla mia esperienza videoludica, in particolare:

- *Outlast*
- *Alien: Isolation*
- *Poppy Playtime*
- *TJOC:R (The Joy of Creation: Reborn)*

Questi giochi condividono alcune caratteristiche fondamentali, come la presenza di lunghi corridoi, ambienti oscuri e angusti, numerosi nascondigli per il giocatore e piccoli enigmi ambientali. Questi elementi hanno lo scopo di incentivare l'esplorazione e aumentare la tensione, spingendo il giocatore a cercare indizi e oggetti utili per progredire nel gioco. Una particolare ispirazione per la mappa del progetto è stata tratta dal livello di *Freddy* in *TJOC:R*, anch'esso ambientato in una casa a un solo piano. Tuttavia, è stata introdotta una sostanziale differenza: la mappa è stata progettata affinché, indipendentemente dalla posizione del giocatore, sia sempre presente almeno un'uscita percorribile per sfuggire all'NPC. Questa scelta progettuale è stata presa per evitare di rendere l'esperienza di gioco eccessivamente frustrante o punitiva. Nella Figura 3.1 è mostrata la planimetria di base della mappa di gioco realizzata. Tale struttura garantisce al giocatore la possibilità di nascondersi o fuggire dall'NPC da qualsiasi punto della casa.

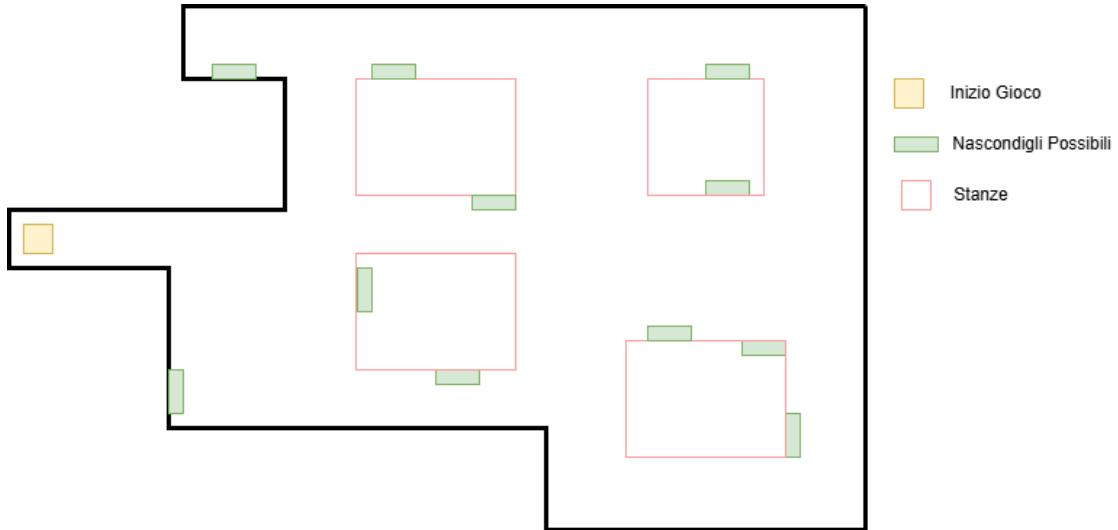


Figura 3.1: Planimetria della mappa di gioco.

3.1.2 Scelta dell'Art Design

Per lo stile grafico è stato scelto un approccio fotorealistico, con l'obiettivo di ricreare un'atmosfera quanto più immersiva e coinvolgente possibile. La rappresentazione realistica degli ambienti consente al giocatore di immedesimarsi maggiormente nel contesto narrativo e nelle dinamiche del gioco. In linea con questo obiettivo, è stata adottata la visuale in prima persona: una scelta fondamentale per mantenere un elevato livello di realismo. Una prospettiva diversa, come quella in terza persona, avrebbe infatti compromesso il senso di immedesimazione e la tensione psicologica tipica del genere horror. L'intera ambientazione è stata concepita per evocare lo stile degli anni '80, attraverso l'uso di arredi, palette cromatiche e oggetti di scena coerenti con il periodo. Questa scelta non solo contribuisce a rafforzare la coerenza narrativa, ma aggiunge anche un elemento di nostalgia e riconoscibilità.

3.2 Sviluppo dell’Ambiente in Unity

Per la realizzazione dell’ambiente di gioco è stato utilizzato **Unity**, una piattaforma di sviluppo software ampiamente impiegata nella creazione di videogiochi 2D e 3D, esperienze di realtà virtuale (VR) e applicazioni interattive multipiattaforma.¹

Unity si compone di un *game engine*, un ambiente di sviluppo integrato (IDE) e una serie di servizi ausiliari, ed è sviluppato da *Unity Technologies*. Per questo progetto è stata adottata la versione 2022.3.19f1, che ha offerto stabilità e compatibilità con i pacchetti più recenti utili alla progettazione degli ambienti.

L’uso di Unity ha permesso una gestione efficiente della scena, dei materiali, delle luci e della fisica di gioco, offrendo al contempo un’interfaccia intuitiva e strumenti di visual scripting utili nelle fasi preliminari di prototipazione.

¹Geek and Job - <https://tinyurl.com/hjffynvb>

3.2.1 Organizzazione degli elementi

Seguendo la planimetria generale del livello di gioco, è stata costruita l’intera mappa, rispettando una coerenza logica nella disposizione delle stanze. Ogni ambiente è stato arredato in modo da richiamare l’organizzazione tipica di una casa reale, con l’intento di aumentare il senso di familiarità e immersione per il giocatore, come mostrato in Figura 3.2 e Figura 3.3. Inoltre, all’interno delle stanze sono stati inseriti diversi possibili nascondigli per il giocatore. Un esempio è visibile in Figura 3.4. Alcuni di questi nascondigli sono stati progettati in modo tale da essere perfettamente integrati nell’ambiente, ovvero percepiti come normali componenti dell’arredo. Un esempio significativo è rappresentato dalla doccia nel bagno (Figura 3.5). Questa scelta progettuale contribuisce a mantenere la tensione durante l’esplorazione, incentivando l’osservazione dell’ambiente circostante e offrendo al contempo possibilità strategiche per sfuggire all’NPC antagonista.



Figura 3.2: Camera da Letto.



Figura 3.3: Cucina.

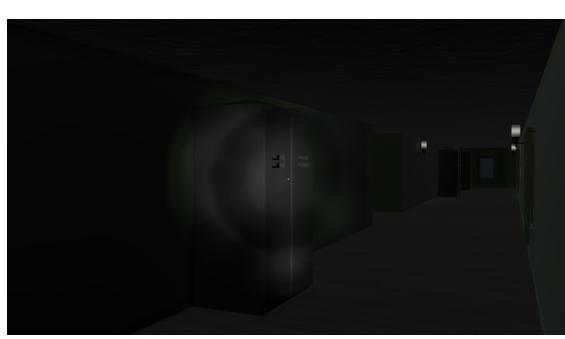


Figura 3.4: Nascondigli del player.

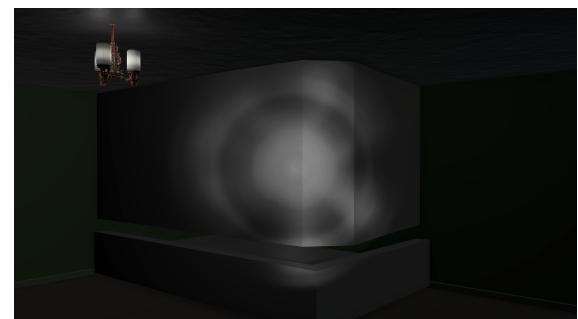


Figura 3.5: Doccia che funge da nascondiglio.

3.2.2 Luci e Suoni

L’intera ambientazione di gioco è caratterizzata da un’illuminazione cupa e poco diffusa, scelta mirata a rafforzare l’atmosfera horror e a coinvolgere maggiormente i sensi del giocatore. Questo tipo di illuminazione limita volutamente la visibilità, generando un senso di disorientamento e tensione. La ridotta percezione visiva agisce sulla psicologia del giocatore, inducendo stati di ansia e stress derivanti dalla mancanza di controllo sull’ambiente circostante. Tale condizione porta il giocatore a rimanere costantemente in allerta, accentuando il coinvolgimento emotivo e sensoriale. Per affrontare l’oscurità, il personaggio controllato dal giocatore è equipaggiato con una torcia, la cui luce ha un raggio e una distanza di illuminazione limitati. Questa scelta progettuale obbliga il giocatore a muoversi con cautela, osservando continuamente l’ambiente circostante. La torcia può essere accesa e spenta manualmente. La differenza tra torcia accesa e spenta è visibile in Figura 3.6 e Figura 3.7. Anche l’aspetto sonoro ha ricevuto particolare attenzione: l’ambiente di gioco è arricchito da effetti audio ambientali, calibrati per aumentare la sensazione di immersione. In particolare, alcuni suoni diventano progressivamente più intensi man mano che il giocatore si avvicina ai confini esterni della mappa. Questa soluzione sonora contribuisce a guidare l’esplorazione e a creare un senso di oppressione e minaccia costante.

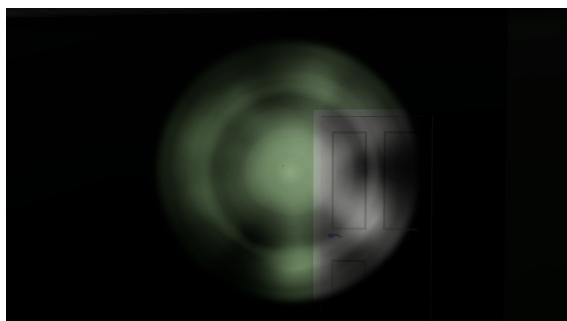


Figura 3.6: Esempio Torcia Accesa

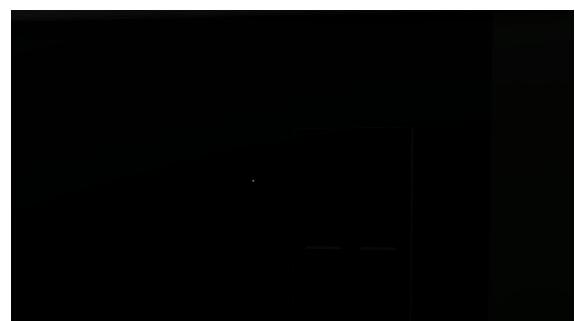


Figura 3.7: Esempio Torcia Spenta

3.3 Integrazione dell'IA

In questo capitolo viene approfondito lo sviluppo dell'Intelligenza Artificiale utilizzata per l'NPC all'interno del videogioco. Il primo passo è stato identificare il tipo di IA più adatta al contesto, facendo riferimento alle principali categorie descritte nella sottosezione 2.2.2. Per progettare un'intelligenza artificiale efficace, si è partiti dall'analisi dei bisogni specifici che l'NPC doveva soddisfare in relazione al genere del gioco. Essendo un survival horror basato sull'inseguimento, uno degli obiettivi principali era quello di far sì che l'NPC fosse in grado di inseguire il giocatore in maniera intelligente, sfruttando il percorso più breve disponibile. Questo comportamento avrebbe garantito una sfida bilanciata, riducendo il margine di vantaggio del giocatore e aumentando la tensione durante il gameplay. Per soddisfare questa esigenza, si è reso necessario l'utilizzo di un sistema di *pathfinding* (ricerca del percorso) efficiente, capace di gestire in tempo reale percorsi alternativi e ostacoli non percorribili. Tenendo conto delle tecnologie disponibili al momento dello sviluppo, la scelta è ricaduta sulla libreria **AI Navigation**, sviluppata da *Unity Technologies Inc.*, che ha offerto le funzionalità necessarie per gestire la navigazione spaziale dell'NPC in maniera ottimizzata. Un'ulteriore necessità era la gestione del comportamento dell'NPC una volta perso di vista il giocatore, ovvero l'adozione di una logica di ricerca autonoma e realistica. Le soluzioni gratuite reperibili online non si sono dimostrate adeguate a soddisfare i requisiti definiti nel sezione 1.2. Per questo motivo, la logica comportamentale è stata progettata e implementata interamente da zero. La soluzione sviluppata è un sistema decisionale adattivo, basato su un meccanismo di apprendimento probabilistico che osserva e memorizza le abitudini del giocatore nel tempo. Nei paragrafi successivi verrà illustrata nel dettaglio l'architettura di tale sistema e le logiche che ne regolano il comportamento.

3.3.1 Implementazione della Libreria AI Navigation

L'utilizzo della libreria **AI Navigation** ha rappresentato una scelta strategica per la gestione efficace del *pathfinding*, sia durante la fase di inseguimento del giocatore, sia nei momenti in cui l'NPC si muove casualmente all'interno della mappa in assenza di un obiettivo specifico.

Prima di analizzare il codice implementato, è opportuno fornire una panoramica generale sul funzionamento della libreria e su come essa gestisca le aree percorribili all'interno della scena di gioco.

La libreria è strutturata attraverso l'impiego di diversi componenti in C#, ciascuno con funzionalità ben precise. I componenti utilizzati maggiormente in questo progetto sono i seguenti:

- **NavMeshSurface**: componente (così definiti i codici C# assegnati agli oggetti 3D in Unity) incaricato di analizzare l'ambiente di gioco e generare dinamicamente una superficie percorribile, basandosi sulla presenza di oggetti etichettati come *Agent*. È il punto di partenza per la creazione della mappa navigabile. Ecco un esempio della mappa dopo aver calcolato le aree percorribili per l'npc in 3.8.



Figura 3.8: In celeste l'area percorribile per l'NPC

- **NavMeshAgent:** viene assegnato all'NPC e ha la funzione di identificarlo come un *agente navigabile*. Questo componente interagisce con la superficie generata da *NavMeshSurface*, permettendo all'NPC di muoversi autonomamente lungo percorsi calcolati in tempo reale.
- **NavMeshModifier:** consente di modificare le proprietà di specifiche superfici, sovrascrivendo i parametri predefiniti generati dal componente *NavMeshSurface*. Grazie a questo componente è possibile, ad esempio, rendere alcune aree non percorribili dall'agente. Ad esempio l'area rosa in Figura 3.8 indica un area con un costo maggiore (ovvero un area che sarebbe meglio evitare).
- **NavMeshObstacle:** etichetta un oggetto come ostacolo. Questo componente può essere configurato per essere statico o dinamico. Nel caso dinamico, il sistema provvede al ricalcolo automatico della superficie percorribile ogni volta che l'ostacolo cambia posizione.

Grazie all'integrazione di questi componenti, è stato possibile sviluppare un sistema di navigazione flessibile e reattivo, in grado di adattarsi dinamicamente ai cambiamenti nella scena e di garantire un comportamento realistico da parte dell'NPC, sia durante l'inseguimento del player, sia durante la fase di pattugliamento.

3.3.2 Avvistamento del Player

Vediamo ora il processo che si attiva nel momento in cui l'NPC rileva la presenza del giocatore. Questo comportamento può essere suddiviso in quattro fasi principali:

- **Avvistamento:** il player entra nel campo visivo dell'NPC.
- **Elaborazione:** viene rilevata la posizione attuale del player e, tramite il componente *NavMeshAgent*, viene calcolato il percorso ottimale per raggiungerlo.
- **Inseguimento:** l'NPC si muove lungo il percorso calcolato, aggiornando in tempo reale la destinazione in base agli spostamenti del player. Durante questa fase vengono raccolti dati sul comportamento del giocatore.
- **Apprendimento:** l'NPC, se possibile, analizza le azioni del player durante l'inseguimento al fine di apprendere il suo stile di gioco. Queste informazioni verranno poi utilizzate in fasi successive, ad esempio nella fase di ricerca, per migliorare l'efficacia dell'intelligenza artificiale.

Avvistamento Il campo visivo dell'NPC è limitato sia in termini di distanza massima sia di angolo di visuale, per evitare comportamenti irrealistici (come il riconoscimento del player alle sue spalle). Quando il player rientra in questo campo, viene eseguito un controllo sulla linea di vista: tramite un *raycasting*, si verifica che non vi siano ostacoli tra l'NPC e il giocatore. Solo se la visuale è libera, il processo di inseguimento può avere inizio. Questo controllo è implementato come mostrato nel codice riportato nel codice sezione 3.3.2.

Elaborazione e Inseguimento Dopo la conferma visiva, l'NPC utilizza la funzione `setDestination()` offerta dalla libreria *AI Navigation*, passando come parametro la posizione corrente del player. In questo modo viene generato in tempo reale un percorso tramite la superficie NavMesh, che l'NPC seguirà per tentare di raggiungere il giocatore.

```

1  public class FieldOfView : MonoBehaviour
2  {
3      public float radius, angle;
4      public LayerMask obstructionMask, targetMask;
5      private void Start()
6      {
7          StartCoroutine(FOVRoutine());
8      }
9      private IEnumerator FOVRoutine()
10     {
11         while(true)
12         {
13             yield return new WaitForSeconds(0.01f);
14             FieldOfViewCheck();
15         }
16     }
17     private void FieldOfViewCheck()
18     {
19         Collider[] rangeChecks=Physics.OverlapSphere(transform.position,
20             radius, targetMask);
21         if (rangeChecks.Length != 0)
22         {
23             Transform target = rangeChecks[0].transform;
24             Vector3 directionToTarget=(target.position-transform.position)
25                 .normalized;
26             if(Vector3.Angle(transform.forward,directionToTarget)<angle/2)
27             {
28                 float distanceToTarget=Vector3.Distance(transform.position,
29                     target.transform.position);
30                 if (!Physics.Raycast(transform.position,directionToTarget,
31                     distanceToTarget,obstructionMask))
32                 {
33                     lastrotation=true;
34                     nemico.stoppingDistance=2f;
35                     canSeePlayer = true;
36                     LastDirection = target.position;
37                     nemico.SetDestination(LastDirection);

```

```
38         RaycastHit hit;
39
40         Physics.Raycast(LastDirection, Vector3.down, out hit, 2f,
41         Floor);
42
43         if (!Floors_Player.Contains(hit.collider))
44         {
45             Floors_Player.Add(hit.collider);
46         }
47
48     }
49 }
```

Listing 1: Classe C# per la vista dell'NPC

Apprendimento Durante la fase di inseguimento, l'NPC segue il player con un duplice obiettivo: raggiungerlo e, contemporaneamente, analizzare le sue azioni. In particolare, l'intelligenza artificiale osserva le scelte compiute dal giocatore nel tentativo di fuga, al fine di identificare un possibile stile di gioco ricorrente.

Questo è reso possibile grazie all'analisi del percorso effettuato dal player durante l'inseguimento. Come mostrato nel listato 3.3.2, viene utilizzata una funzione di Unity per individuare gli oggetti con cui il player ha avuto collisioni o che ha ignorato durante la fuga.

```

1  public class Apprendimento : MonoBehaviour
2  {
3      Vector3 boxSize = new Vector3(3, 3, 4);
4      Collider[] objects;
5      [SerializeField]
6      Transform Player;
7      Vector3 LastPlayerMemory, OverlappPosition;
8      float count_view_Update = 0f;
9      bool player_recentview = false;
10     bool right = false, distant = false, updateside = false,
11         ↳ updatedistance = false;
12     Quaternion OverlappRotation = new Quaternion(1, 0, 0, 0);
13     public void ElementsBack(Vector3 LastPositionPlayer, Vector3
14         ↳ FirstPositionPlayer, LayerMask target, Vector3 EnemyPosition,
15         ↳ int side, List<Collider> Floors)
16     {
17         bool checkz;
18         if(player_recentview)
19         {
20             FirstPositionPlayer = LastPlayerMemory;
21         }
22         Quaternion rotationOverlapp;
23         float minScale = 0;
24         float distance = Vector3.Distance(LastPositionPlayer,
25             ↳ FirstPositionPlayer);
26         Vector3 direzioneMovimento = (FirstPositionPlayer -
27             ↳ LastPositionPlayer).normalized;

```

```

23         if (direzioneMovimento.magnitude < 0.1f)
24             return;
25         else
26     {
27             OverlappPosition = LastPositionPlayer + (direzioneMovimento
28                 ↪ * (distance / 2));
29             float DistanceX = Mathf.Abs(LastPositionPlayer.x -
30                 ↪ FirstPositionPlayer.x);
31             float DistanceZ = Mathf.Abs(LastPositionPlayer.z -
32                 ↪ FirstPositionPlayer.z);
33             if (DistanceX >= DistanceZ)
34             {
35                 checkz = true;
36                 foreach (var floor in Floors)
37                 {
38                     if (floor.transform.eulerAngles.y == 0)
39                     {
40                         OverlappPosition.z = floor.transform.position.z;
41                         minScale = floor.GetComponent<Renderer>().bound_
42                             ↪ s.size.z;
43                         break;
44                     }
45                 }
46                 boxSize = new Vector3(distance / 2, 1, minScale / 2);
47             }
48             else
49             {
50                 checkz = false;
51                 foreach (var floor in Floors)
52                 {
53                     if (floor.transform.eulerAngles.y == -90)
54                     {
55                         OverlappPosition.x = floor.transform.position.x;
56                         minScale = floor.GetComponent<Renderer>().bound_
57                             ↪ s.size.x;
58                         break;
59                     }
60                 }
61             }
62         }
63     }
64 }
```

```

55         }
56         boxSize = new Vector3(minScale / 2, 1, distance / 2);
57     }
58
59     Debug.Log(Vector3.Distance(FirstPositionPlayer,
60                               LastPositionPlayer));
60
61     Collider[] objectsBack =
62     Physics.OverlapBox(OverlappPosition, boxSize,
63                         Quaternion.identity, target);
64
65     List<Collider> objects = objectsBack.ToList();
66 }
67
68 }
69
70 }

```

Per rendere l'apprendimento efficace, è necessario escludere gli oggetti che non apportano informazioni significative. Un esempio sono le porte chiuse a chiave, che il player sa già essere inaccessibili: ignorarle è un comportamento ovvio e non rappresenta una scelta tattica.

Per questo motivo è stato implementato un sistema di **filtraggio** che scarta gli elementi non rilevanti. Oltre a migliorare la precisione dell'apprendimento, questa operazione è fondamentale anche per motivi prestazionali: riducendo il numero di elementi da analizzare, si riduce anche il carico computazionale, considerando che queste operazioni vengono eseguite frequentemente.

Il sistema di filtraggio si basa su due controlli principali:

- **Elementi escludibili:** oggetti che si trovano dietro all'NPC al momento dell'avvistamento, poiché è altamente improbabile che il player abbia interagito con essi dopo essere stato avvistato. Questo filtro è implementato nel codice illustrato in sezione 3.3.2.
- **Nascondigli visibili all'NPC:** i nascondigli che sono completamente visibili all'NPC non vengono considerati, poiché un giocatore difficilmente si nasconde alla vista diretta del nemico. Tuttavia, viene gestito il caso in cui il player commetta questo errore: in tal caso, viene eseguito un controllo specifico sull'ultima posizione nota del giocatore, come mostrato in sezione 3.3.2.

```

1  public class Apprendimento : MonoBehaviour
2  {
3      private void WorkData(List<Collider> objects, Vector3
4          ↳ FirstPositionPlayer, float distance_Player_Do, ref Collider
5          ↳ Player_Chose, Vector3 LastPositionPlayer, float
6          ↳ distance_Player_Chose, ref int count_side, ref int
7          ↳ count_another_side, bool instantiate, ref float side, bool
8          ↳ checkz)
9      {
10         for (int i = objects.Count - 1; i >= 0; i--)
11         {
12             float distance_Player_Coll=Vector3.Distance(objects[i].tran
13                 ↳ sform.position,LastPositionPlayer);
14             if ((objects[i].tag == "Hide_Locked" || objects[i].tag ==
15                 ↳ "Hide_Link") &&distance_Player_Coll>1.5f)
16             {
17                 objects.RemoveAt(i);
18                 continue;
19             }
20             if(Player_Chose==null)
21             {
22                 Player_Chose=objects[i];
23                 distance_Player_Chose=distance_Player_Coll;
24             }
25             if(distance_Player_Coll<distance_Player_Chose)
26             {
27                 Player_Chose=objects[i];
28                 distance_Player_Chose = distance_Player_Coll;
29             }
30             if(objects[i].tag!="Intersection")
31             {
32                 if(checkz)
33                 {
34                     if (!instantiate)
35                     {
36                         instantiate=true;
37                         side=objects[i].transform.position.z;
38                         count_side++;
39                     }
40                 }
41             }
42         }
43     }
44 }
```

```

31
32     }
33
34     else if(objects[i].transform.position.z==side)
35     {
36         count_side++;
37     }
38     else
39     {
40         count_another_side++;
41     }
42     else
43     {
44         if (!instantiate)
45         {
46             instantiate=true;
47             side=objects[i].transform.position.x;
48             count_side++;
49         }
50         else if(objects[i].transform.position.x==side)
51         {
52             count_side++;
53         }
54         else
55         {
56             count_another_side++;
57         }
58     }
59 }
60 }
61 }
```

Listing 2: Classe C# per filtraggio Dati

Dopo il filtraggio dei dati, è possibile passare all'**analisi degli oggetti ignorati dal player durante la fuga**, ovvero tutti quegli elementi presenti nel suo percorso ma non utilizzati. Lo scopo di questa analisi è riconoscere lo *stile di gioco* adottato dal

player, così da adattare la strategia dell'NPC durante le fasi successive.

Abbiamo analizzato tre tipologie principali di comportamento:

- **Impulsivo:** il giocatore tenta di far perdere le sue tracce nel minor tempo possibile, scegliendo nascondigli o deviazioni molto vicine al punto in cui è stato avvistato. In questo caso, è più efficace cercarlo nei pressi dell'ultima posizione nota.
- **Coraggioso:** il giocatore tende a correre verso obiettivi strategici o a lungo termine, anche se più lontani, accettando il rischio di un inseguimento più duraturo. Per contrastare questo stile, l'NPC dovrà esplorare zone distanti o vicine ad obiettivi noti del gioco.
- **Destro/Mancino:** in caso di svincoli simmetrici o porte specchiate, si analizza se il player predilige deviare a destra o a sinistra. Questa tendenza permette all'NPC di aumentare le probabilità di ritrovare il giocatore seguendo il lato preferito.

L'analisi viene eseguita **ogni volta che si verifica un inseguimento**, in modo da aggiornare costantemente il profilo comportamentale del giocatore. Questo consente all'NPC di adattarsi anche in caso di variazioni nello stile di gioco: ad esempio, se il player modifica strategia per esigenze legate alla trama o a eventi inaspettati.

Per rappresentare lo stile di gioco del player, è stato deciso di **memorizzare le sue ultime 5 azioni**. Con queste viene calcolata una media, utile per definire lo stile attuale.

- Se il player non ha ancora uno stile assegnato, le prime 5 azioni vengono utilizzate per determinarlo.
- Se è già presente uno stile dominante, la nuova valutazione viene mediata con quella precedente, evitando un cambiamento radicale dovuto a singole azioni anomale. In questo modo il sistema è più resistente a comportamenti momentanei o casuali.

Il processo è riassunto nella logica implementata nel listato 3.3.2, che descrive il meccanismo di aggiornamento e classificazione dello stile di gioco del player.

```

1  public class PlayerStyle : MonoBehaviour
2  {
3      static float right_value=0.5f;
4      static float left_value=0.5f;
5      static float near_value=0.5f;
6      static float distant_value=0.5f;
7
8      int[] distance= new int[5], side=new int[5];
9
10     private int count_for_update_side=0, count_for_update_distance=0;
11
12     public void Update_PlayerStyle(bool updatedistance, bool
13         → updateside, bool right, bool distant)
14     {
15         if(updatedistance)
16         {
17             distance[count_for_update_distance] = distant==true ? 1:0;
18
19             if(count_for_update_distance==6)
20             {
21                 float percent_value = Percent_values(distance);
22                 updateValues(ref distant_value, ref near_value,
23                     → percent_value);
24                 count_for_update_distance=0;
25             }
26         }
27     }
28
29     if(updateside)
30     {
31         side[count_for_update_side]= right==true ? 1:0;
32
33         if(count_for_update_side==6)
34         {
35             float percent_value = Percent_values(side);
36             updateValues(ref right_value, ref left_value,
37                     → percent_value);
38         }
39     }
40 }
```

```

35             count_for_update_side=0;
36         }
37     else
38     {
39         count_for_update_side++;
40     }
41 }
42 }
43
44 public static float[] Get_PlayerStyle()
45 {
46     return new float []{right_value, left_value, near_value,
47     ↵ distant_value};  

48 }
49
50
51     private float Percent_values(int[] dates)
52     {
53         float sum_dates=0;
54         foreach(float value in dates)
55             sum_dates+=value;
56         return sum_dates/dates.Length;
57     }
58
59     private void updateValues(ref float main_values, ref float
60     ↵ opposite_value, float percent_value_Dates)
61     {
62         if(percent_value_Dates>0.5f)
63         {
64             main_values=Mathf.Lerp(main_values, 1f, 0.1f);
65         }
66         else if(percent_value_Dates<0.5f)
67         {
68             main_values=Mathf.Lerp(main_values, 0, 0.1f);
69         }
70     }
71 }
```

```
70         }
71         opposite_value=1-main_values;
72     }
73 }
74 }
```

Listing 3: Classe C# per stile di gioco player

3.3.3 Player non visibile

In questa sezione analizziamo la logica dell'NPC nel caso in cui il player non sia più visibile. La reazione dell'IA cambia in base a differenti contesti, in particolare:

- **Ricerca del giocatore:** il player è stato avvistato durante un inseguimento, ma successivamente è stato perso di vista. In questo caso, l'IA attiva un sistema di deduzione per identificare il possibile nascondiglio del player.
- **Giocatore mai avvistato:** in questa situazione l'IA può reagire in due modi, a seconda del comportamento del player:
 - *Percezione di un rumore:* se il player si trova troppo vicino all'NPC o sta correndo, potrebbe essere localizzato a causa del rumore dei suoi passi.
 - *Assenza di rumore:* se il player si muove in modo silenzioso, evitando di correre, l'NPC seguirà un percorso di pattuglia prestabilito.

Ricerca del Giocatore Durante un inseguimento possono verificarsi due casi:

- **Giocatore raggiunto:** il player viene preso e perde la partita.
- **Giocatore perso:** l'IA avvia una procedura per identificare il nascondiglio più probabile.

Nel secondo caso, l'NPC sfrutta i dati raccolti durante gli inseguimenti precedenti per scegliere dove cercare il player. Il primo passo, una volta raggiunta l'ultima posizione nota del player, è individuare tutti i nascondigli accessibili nell'area. Questa area viene definita come un cerchio con raggio pari alla distanza che il player avrebbe potuto percorrere nel tempo trascorso dall'ultima visuale, tenendo conto della sua velocità. Questo permette di escludere automaticamente i nascondigli troppo lontani per essere stati raggiunti.

Un secondo filtro viene applicato analizzando la direzione del movimento del player prima di essere perso di vista. I nascondigli posizionati nella direzione opposta vengono scartati, poiché sarebbe improbabile che il player abbia invertito la marcia senza essere notato.

Assegnazione della probabilità Per ogni nascondiglio rimasto, viene calcolata una probabilità di utilizzo in base a due parametri fondamentali:

- **Distanza:** classificata in "vicina" o "lontana" rispetto all'NPC. Un nascondiglio è considerato "vicino" se si trova nella prima metà del raggio calcolato.
- **Lato:** se il nascondiglio si trova alla destra o alla sinistra della direzione dell'NPC.

Queste due caratteristiche vengono poi confrontate con lo stile di gioco del player. Ad esempio, se il player è stato classificato come *impulsivo* e *destro*, i nascondigli **vicini e a destra** avranno una probabilità maggiore di essere selezionati rispetto a quelli lontani o a sinistra. La priorità dei nascondigli, quindi, segue questa logica:

$$\text{vicino + destra} > \text{vicino + sinistra} > \text{lontano + destra} > \text{lontano + sinistra}$$

Dopo aver assegnato una probabilità a ciascun nascondiglio, vengono selezionati:

- i 3 nascondigli con le probabilità più alte,
- 1 nascondiglio scelto casualmente tra i rimanenti (escludendo i 3 precedenti),

Infine, l'NPC sceglierà uno tra questi 4, tenendo conto della loro probabilità ma includendo anche una componente casuale. Questo approccio introduce una certa imprevedibilità che simula la possibilità che il player abbia cambiato stile o sia stato influenzato da fattori esterni.

```

1  public class IASearchPlayerHide : MonoBehaviour
2  {
3      [SerializeField]
4      LayerMask Mask_Location, Mask_Door_or_LinkHide;
5      [SerializeField]
6      public float player_velocity;
7      public float raycastDistance = 2.5f;
8      Vector3 positionSphereDraw;
9      float sizeSphereDraw;
10     public Vector3 NextIAPosition(float angle, float Time, int right=-1)

```

```

11  {
12      Collider floorunderfoot = Enemy_Link_Or_Room_Raycast(transform);
13      Vector3 position_Overlapp=transform.position;
14      position_Overlapp.y=floorunderfoot.transform.position.y;
15      sizeSphereDraw=player_velocity*Time;
16      positionSphereDraw=position_Overlapp;
17      List<Collider> objects_infrontme =
18          → Physics.OverlapSphere(position_Overlapp,
19          → player_velocity*Time,
20          → Mask_Door_or_LinkHide).ToList<Collider>();
21      List<Transform> objects_in_front_me = new List<Transform>();
22      foreach(Collider coll in objects_infrontme)
23          objects_in_front_me.Add(coll.transform);
24      FilterInFront(objects_in_front_me, 170,floorunderfoot.name);
25      IDictionary<Transform, float> choices = new
26          → Dictionary<Transform,float>();
27      foreach(Transform object_infrontme in objects_in_front_me)
28      {
29          if(object_infrontme.name==floorunderfoot.name)
30          {
31              choices.Add(object_infrontme,
32                  → Value_Option(0,null,true,right));
33          }
34          else
35          {
36              AnalyzeFloor(object_infrontme,choices,objects_in_front_]
37                  → me,angle,player_velocity*Time);
38          }
39      }
40      System.Random rnd = new System.Random();
41      List<Transform> KeyValueMax =
42          → choices.OrderBy(pair=>pair.Value)
43              .Take(3)
44              .Select(pair => pair.Key)
45              .ToList();
46      List<Transform> rimanenti= choices
47          .Where(pair => !KeyValueMax.Contains(pair.Key))

```

```

41         .Select(pair => pair.Key)
42         .ToList();
43     List<Transform> random_elements = rimanenti
44         .OrderBy(_ => rnd.Next())
45         .Take(1)
46         .ToList();
47     List<Transform> final_elements =
48         ↳ KeyValueMax.Concat(random_elements).ToList();
49     float total_sum=0f;
50     foreach (Transform op in final_elements)
51     {
52         if(choices[op]>=0.2 && choices[op]<=0.8)
53             total_sum+=choices[op];
54         else if(choices[op]>=0.2)
55             total_sum+=0.2f;
56         else
57             total_sum+=0.8f;
58     }
59     float partial_sum=0f;
60     float random_num = UnityEngine.Random.Range(0, total_sum);
61     foreach(Transform op in final_elements)
62     {
63         if(choices[op]>=0.2 && choices[op]<=0.8)
64             partial_sum+=choices[op];
65         else if(choices[op]>=0.2)
66             partial_sum+=0.2f;
67         else
68             partial_sum+=0.8f;
69         if(partial_sum>=random_num)
70             {
71                 if(op.transform.tag=="HideLink")
72                     return op.transform.position;
73                 else
74                 {
75                     Debug.Log("La scelta è "+op.name);
76                     Transform root = getRootRoomFromChild(op);
77                     Debug.Log("La scelta ha coem root "+root.name);
78                     Vector3 chooces = getRandomHideFromRoom(root);
79                     return chooces.y!=-200? chooces:op.position;
80                 }
81             }
82     }

```

```

77
78         }
79         return new Vector3(0,300,0);
80     }
81     private void FilterInFront(List<Transform> lista, float angle,
82                               string floornameunderfoot)
83     {
84         for(int i=lista.Count-1;i>=0;i--)
85         {
86             if(lista[i].name!=floornameunderfoot)
87             {
88                 Vector3 direction = (lista[i].transform.position-transf_
89                               .orm.position).normalized;
90                 if(lista[i].tag=="FloorLink")
91                 {
92                     lista.RemoveAt(i);
93                     continue;
94                 }
95                 else if(Vector3.Angle(transform.forward,
96                               direction)>angle/2)
97                 {
98                     foreach(Transform coll in
99                         getDoorFromRoot(getRootRoomFromChild(lista[i])))
100                     {
101                         if(Vector3.Angle(coll.position,
102                                         direction)<=angle/2)
103                         {
104                             continue;
105                         }
106                     }
107                 }
108             }

```

```

109     private bool AnalyzeFloor(Transform floor_hide,
110         ↳ IDictionary<Transform, float> choices, List<Transform> options,
111         ↳ float angles, float distance_player)
112     {
113         if(floor_hide.tag=="HideLink")
114         {
115             choices.Add(floor_hide,
116                 ↳ Value_Option(distance_player,floor_hide.transform));
117             return true;
118         }
119         else
120         {
121             if(options.Contains(floor_hide))
122             {
123                 if(choices.ContainsKey(floor_hide))
124                 {
125                     return true;
126                 }
127                 else
128                 {
129                     Transform choose_door=null;
130                     float distance_object=float.MaxValue;
131                     List<Transform> doors_floor=getDoorFromRoot (getRoot |
132                         ↳ RoomFromChild(floor_hide));
133                     if(doors_floor==null)
134                     {
135                         return false;
136                     }
137                     foreach(Transform door in doors_floor)
138                     {
139                         Vector3 direction = (door.position-transform.po |
140                             ↳ sition).normalized;
141                         if(Vector3.Angle(transform.forward,
142                             ↳ direction)<angles/2 &&
143                             ↳ door.tag!="HideLocked" && distance_player>=|
144                             ↳ Vector3.Distance(transform.position,
145                             ↳ door.position))

```

```

137     {
138         if(door.name.Split("_") [1] != "Link")
139         {
140             if(AnalyzeFloor(door.GetComponent<Door_]
141                 ↪ Two_Room> () .Rooms [0].transform,choi]
142                 ↪ ces,options,angles,
143                 ↪ distance_player) ||
144             AnalyzeFloor(door.GetComponent<Door_Two_]
145                 ↪ _Room> () .Rooms [1].transform,choices_]
146                 ↪ ,options,angles,distance_player))
147             if(distance_object>=Vector3.Distance(
148                 ↪ e(transform.position,
149                 ↪ door.transform.position))
150             {
151                 choose_door=door;
152                 distance_object=Vector3.Distance(
153                     ↪ e(transform.position,
154                     ↪ door.transform.position);
155             }
156         }
157     }
158     if(choose_door!=null)
159     {
160         choices.Add(floor_hide,Value_Option(distance_pl]
161             ↪ ayer,choose_door.transform));

```

```

161                     return true;
162                 }
163             else
164                 return false;
165             }
166         }
167     else
168         return false;
169     }
170 }
171 public Collider Enemy_Link_Or_Room_Raycast(Transform point)
172 {
173     RaycastHit hit;
174
175     if (Physics.Raycast(point.position, Vector3.down, out hit,
176     → raycastDistance, Mask_Location))
177     {
178         return hit.collider;
179     }
180     else
181     {
182         return null;
183     }
184 }
185 public Collider Enemy_Link_Or_Room(Transform point_radius)
186 {
187     Collider[] options =
188     → Physics.OverlapSphere(point_radius.position, 2f,
189     → Mask_Location);
190     if(options.Length!=0)
191         return options[0];
192     return null;
193 }
194 public Collider Enemy_Link_Or_Room_with_Position(Vector3
195     → point_radius)

```

```

194     {
195         Collider[] options = Physics.OverlapSphere(point_radius, 2.5f,
196             ↳ Mask_Location);
197         if(options.Length!=0)
198             return options[0];
199         return null;
200     }
201     private Transform getRootRoomFromChild(Transform child)
202     {
203         Transform parent;
204         Debug.Log("Il figlio che stiamo analizzando è "+child.name);
205         parent = child.transform.parent;
206         try{
207             while (parent.tag!="RootRoom")
208                 parent = parent.parent;
209             return parent;
210         }
211         catch
212         {
213             Debug.LogWarning("non è presente il padre RootRoom
214                 ↳ dell'oggetto "+child.name+" nella ricerca del padre si è
215                 ↳ raggiunto l'oggetto "+parent.name);
216             return null;
217         }
218     }
219     private Vector3 getRandomHideFromRoom(Transform root)
220     {
221         List<Transform> hides = getHideRoomFromRoot(root);
222         if(hides.Count-1== -1)
223             return new Vector3(0,-200,0);
224         return hides[UnityEngine.Random.Range(0,
225             ↳ hides.Count-1)].position;
226     }
227     private List<Transform> getDoorFromRoot(Transform root)
228     {
229         if(root==null)
230             return null;

```

```

227     try
228     {
229         return root.GetComponent<Room>().getObjects("RootDoor");
230     }
231     catch
232     {
233         Debug.LogWarning("Errore nell'ottinimento delle porte
234             ↳ tramite l'oggetto "+root.name);
235         return null;
236     }
237
238     private List<Transform> getHideRoomFromRoot(Transform root)
239     {
240         Debug.Log("Il root analizzato con lo script room è "+root.name);
241         return root.GetComponent<Room>().getObjects("HideRoom");
242     }
243     private float Value_Option( float distance_player, Transform
244         ↳ option=null, bool nea=false, int rig=-1)
245     {
246         bool near;
247         int right;
248         if(option!=null)
249         {
250             near = near_or_distant(transform.position,
251                 ↳ option.transform.position, distance_player);
252             right = right_left(option.transform);
253         }
254         else
255         {
256             near=nea;
257             right=rig;
258         }
259         return Calculate_Value(right,near,
260             ↳ PlayerStyle.Get_PlayerStyle());
261     }

```

```

260     private bool near_or_distant(Vector3 nemico, Vector3 option, float
261         ↳ distance_player)
262     {
263         return Vector3.Distance(nemico, option) <= (distance_player/2);
264     }
265
266     private int right_left(Transform option)
267     {
268         Vector3 direction = (transform.position -
269             ↳ option.position).normalized;
270
271         float angle = Vector3.SignedAngle(transform.position,
272             ↳ direction, Vector3.up);
273
274         if(angle>0)
275             return 1;
276
277         else if(angle<0)
278             return -1;
279
280         return 0;
281     }
282
283
284     private float Calculate_Value(int right, bool near, float[] info)
285     {
286
287         float value=0;
288
289         if(right>0)
290             value+=info[0];
291
292         if(right<0)
293             value+=info[1];
294
295         if(near)
296             value+=info[2];
297
298         else
299             value+=info[3];
300
301         return value;
302     }
303
304 }
```

Listing 4: Classe C# per la scelta del nascondiglio

Giocatore non Visto - Percezione Rumore In questo caso, anche se il player non è visibile, l'NPC può comunque entrare in stato di allerta. Questo avviene quando il player si trova in una posizione relativamente vicina e compie un'azione che genera rumore, come ad esempio correre o interagire bruscamente con l'ambiente.

In presenza di un rumore, l'NPC reagisce dirigendosi verso la posizione in cui il suono è stato generato. Se, una volta raggiunta, non rileva alcuna presenza o indizio del player, l'NPC abbandona l'allerta e riprende il suo percorso di pattuglia predefinito.

```

1  public class Movement : MonoBehaviour
2  {
3      [SerializeField] float velocità = 5;
4      Rigidbody Rigidbody;
5      [SerializeField]
6      FieldOfView enemy;
7      float timerun=0f;
8      AudioSource audiosource;
9      void Update ()
10     {
11         if(velocità==8)
12         {
13             timerun+=Time.deltaTime;
14             if(timerun>=5 && Vector3.Distance(transform.position,
15                 ↪ enemy.transform.position)<7f)
16             {
17                 enemy.HearSomeThing_Or_IA(transform.position);
18             }
19         else
20         {
21             if(timerun>0)
22                 timerun-=Time.deltaTime;
23             }
24         }
25         void FixedUpdate()
26         {
```

```

27         if (Input.GetKey(KeyCode.LeftShift))
28     {
29         velocità=8;
30     }
31     else
32     {
33         velocità=5;
34     }
35     Vector2 targetVelocity = new
36         → Vector2(Input.GetAxis("Horizontal") * velocità,
37         → Input.GetAxis("Vertical") * velocità);
38     RigidBody.velocity = transform.rotation * new
39         → Vector3(targetVelocity.x, RigidBody.velocity.y,
40         → targetVelocity.y);
41     if (RigidBody.velocity!=Vector3.zero)
42     {
43         if (velocità>5)
44             audiosource.pitch=1.5f;
45         else
46             audiosource.pitch=1f;
47         if (audiosource.isPlaying)
48             { }
49         else
50             {
51                 audiosource.Play();
52             }
53     }
54 }
55 }
```

Listing 5: Classe C# per movimento del player

```

1   public void HearSomeThing_Or_IA(Vector3 position)
2   {
3       if (!canSeePlayer)
4       {
5           if (transform.GetComponent<StaticPointPAtHEnemy>().enabled)
6           {
7               transform.GetComponent<StaticPointPAtHEnemy>().enabled= ]
8               → false;
9           }
10          StartCoroutine(HearSomeThingCoroutine(position));
11      }
12  public IEnumerator HearSomeThingCoroutine(Vector3 position)
13  {
14      nemico.SetDestination(position);
15      yield return null;
16      while (Vector3.Distance(transform.position, position)>0.5f &&
17          → !canSeePlayer)
18      {
19          yield return null;
20      }
21      if (!canSeePlayer)
22      {
23          transform.GetComponent<StaticPointPAtHEnemy>().enabled=true;
24      }
}

```

Listing 6: Classe C# per movimento dell'npc verso la fonte di rumore

Giocatore non Visto - Assenza di Rumore Nel caso in cui l'NPC né veda né senta il player, allora seguirà la sua routine di base, ovvero un percorso di pattuglia che lo porterà a muoversi all'interno della mappa con l'obiettivo di trovare il giocatore.

Per evitare che i suoi movimenti diventino troppo prevedibili — cosa che comprometterebbe l'esperienza di gioco — il percorso è stato suddiviso in più segmenti, i quali possono essere riorganizzati in modo casuale. Questo approccio rende i movimenti dell'NPC imprevedibili, costringendo il giocatore a restare sempre vigile e attento durante la partita.

```

1  public class StaticPointPAtHEnemy : MonoBehaviour
2  {
3      [SerializeField]
4      Transform[] MappaPoints;
5      NavMeshAgent nemico;
6      private int points, current_point=0, nextpoint=0;
7      void Start()
8      {
9          points = MappaPoints.Length;
10         nemico = GetComponent<NavMeshAgent>();
11         setStartCoroutine();
12         nemico.stoppingDistance=0;
13     }
14     void Update()
15     {
16         if(nemico.remainingDistance==0 && current_point!=nextpoint)
17         {
18             current_point=nextpoint;
19             nemico.SetDestination(MappaPoints[current_point].position);
20         }
21     }
22     void setStartCoroutine()
23     {
24         nemico.stoppingDistance=0;
25         nemico.SetDestination(MappaPoints[current_point].position);
26         StartCoroutine(StaticPath());
}

```

```
27     }
28 
29     void setStopCoroutine()
30     {
31         nemico.stoppingDistance=2.2f;
32         nemico.isStopped=true;
33         StopCoroutine(StaticPath());
34     }
35 
36     IEnumerator StaticPath()
37     {
38         while (true)
39         {
40             if(current_point == nextpoint)
41             {
42                 while(true)
43                 {
44                     nextpoint = Random.Range(0, points);
45                     if(nextpoint != current_point)
46                     {
47                         break;
48                     }
49                 }
50             }
51         }
52     }
53 }
```

Listing 7: Classe C# per movimento casuale dell'npc

CAPITOLO 4

Conclusioni

Lo sviluppo di questo videogioco mi ha permesso di approfondire tematiche legate al Game Design e all'Intelligenza Artificiale, argomenti che mi hanno sempre affascinato e incuriosito. Proprio per questo motivo sono riuscito a dedicarmi con passione, ottenendo nuove conoscenze che prima non avevo mai acquisito.

L'Intelligenza Artificiale si è dimostrata capace di inseguire e cercare il player in modo coerente con il contesto, migliorando così l'esperienza di gioco e rendendola più immersiva. La scelta di utilizzare la libreria *AI Navigation* si è rivelata vincente, in quanto ha permesso una gestione efficace dei percorsi e delle risorse, rendendo l'NPC reattivo alle azioni del giocatore. Questo ha contribuito a evitare una sensazione di lentezza o prevedibilità, aumentando ulteriormente la qualità dell'esperienza.

4.1 Sviluppi Futuri e Possibili Miglioramenti

Il progetto offre numerosi spunti di miglioramento e ampliamento:

- **Testing e Bug Fixing migliorati:** distribuire il gioco a un gruppo di tester permetterebbe di ottenere un quadro più completo del comportamento del sistema. Essendo un progetto realizzato interamente da me, non è stato possibile testarlo su diverse configurazioni hardware o piattaforme. È probabile che esistano casistiche non ancora emerse, che potrebbero compromettere la piena giocabilità del titolo su determinati dispositivi.
- **Miglioramento dell'NPC Intelligente:** includere ulteriori interazioni per il player, come la possibilità di distrarre l'NPC lanciando oggetti per creare rumore. Un altro sviluppo interessante sarebbe quello di memorizzare informazioni contestuali divise per aree della mappa, invece di basarsi solo sugli ultimi cinque eventi.
- **Grafica:** aggiunta di modelli 3D più dettagliati per il player e l'NPC, con l'introduzione di animazioni e cinematiche per rendere l'esperienza visiva più coinvolgente.
- **Contenuti Aggiuntivi:** espansione della mappa di gioco, aumentando la varietà e la complessità degli oggetti interattivi e degli enigmi. Si potrebbero introdurre sezioni in cui è necessario risolvere enigmi in corridoi stretti, aumentando la tensione per il rischio di essere scoperti.
- **Sistema di Salvataggio:** con l'introduzione di missioni più lunghe e cinematiche, sarà importante offrire un sistema di salvataggio per evitare frustrazione da parte del giocatore che non può completare il gioco in una sola sessione.

In conclusione, la realizzazione di questo videogioco ha dimostrato come l'Intelligenza Artificiale possa essere utilizzata per arricchire l'esperienza videoludica, rendendola più dinamica, realistica e stimolante. Un'esperienza che non solo diverte, ma coinvolge davvero il giocatore, immergendolo in una sfida sempre diversa.

Bibliografia

- [1] G. Miller, “99% of gamers are excited by the potential of smart npcs powered by advanced artificial intelligence,” *European Gaming*, 2023. (Citato a pagina 1)
- [2] M. A. Rickards, *Che cos’è un videogioco.* Bussole, 2021. (Citato a pagina 3)
- [3] A. G.M., “Il primo videogioco, un passatempo universale,” *Storica National Geographic*, 2024. (Citato a pagina 5)
- [4] R. Capiotto, “Generazione procedurale di contenuti applicata ai livelli di un gioco,” Master’s thesis, Politecnico Milano 1863, 2014-2015. (Citato a pagina 6)
- [5] R. Allegri, “Grafica nei videogiochi: la sua evoluzione,” *Enkey WebMagazine*, 2023. (Citato a pagina 8)
- [6] S. D. Marzo, “I migliori giochi che sfruttano la fisica alla grande per il gameplay,” *Everyeye.it*, 2023. (Citato a pagina 9)
- [7] G. D. Flavio, “Cos’è il test di turing e perché se ne parla sempre di più in ambito intelligenza artificiale,” *Geopop*, 2023. (Citato a pagina 11)

Ringraziamenti

Ad essere sincero non so che dire, ho passato più tempo a vedere questo foglio bianco che a scrivere l'intera tesi ed ogni volta non sapevo come iniziare il discorso, come ora che sto tergiversando. La prima difficoltà nello scrivere questi ringraziamenti è stata dover guardare tutto da un punto di vista esterno. Alla stesura dei ringraziamenti il primo pensiero è stato: "la mattina mi svegliavo io per andare all'università, sono stato io ad affrontare i professori dovrei ringraziare solo me stesso", ma riflettendoci di più, tutto questo non l'avrei fatto o non sarebbe stato lo stesso senza di voi. Non ho la minima idea di come ordinare le cose, sappiate che è tutto fatto a caso. Per aiutarmi un minimo con la scrittura dei ringraziamenti ho deciso di prendere come esempio una gioranta tipo all'università.

Detto ciò iniziamo.

Ringrazio mio padre (per gli amici Peppino by mamma) poichè, nonostante io non sopporti il suo carattere e il nostro continuo litigare, devo dargli il merito di essersi svegliato ogni mattina per accompagnarmi alla fermata del bus, tranne quell'unica volta che si è dimenticato di suo figlio il quale, per tornare a casa, si è dovuto fare 45 minuti di camminata sotto la pioggia ma almeno è successo soltanto una volta (e sì, se ve lo state chiedendo, l'ho chiamato più di una volta ma non ha mai risposto). Ringrazio mia madre che nonostante fosse impegnata o stanca, ogni mattina si svegliava per prepararmi il pranzo. Per lo stesso motivo ringrazio anche quella

rottura di scatole di mia sorella Carmina (sì, avete sentito bene, Carmina) che ha il talento di saper cucinare qualsiasi cosa ed ognuna di queste è squisita. Ringrazio anche mia sorella, Giovanna, che non ha fatto nulla per aiutarmi direttamente con il mio percorso ma credo che tutti gli schiaffi che ha ricevuto sul sedere (dati per giocare, ovviamente) e tutti i momenti in cui la infastidivo siano stati di grande aiuto per sfogare tutto il mio stress, quindi grazie di aver subito tutto questo.

Ringrazio la palestra WorldFitness, per aver distribuito nel 2020 le mascherine per il covid; è stata la miglior maschera per la notte che io abbia mai usato (la mia salvezza per dormire in autobus). Ringrazio il mio compagno di autobus più fidato, Angelo, per avermi fatto arrivare all'università ogni mattina e per aver resistito alla tentazione di lasciarmi dormire in autobus per poi scoprire dopo ore dove sarei arrivato ma non ti ringrazio per tutti gli schiaffi datomi per svegliarmi quando bastava semplicemente chiamarmi.

Un ringraziamento al Professor Fabio Palomba, relatore di questa tesi e uno dei migliori professori mai incontrati; grazie alla sua professionalità e disponibilità, oggi sono qui, a completare il mio percorso accademico, con la consapevolezza di aver trattato con massimo riguardo un tema molto complesso, l'Intelligenza Artificiale, applicata ad una mia grande passione: i Videogiochi.

Ringrazio tutti i miei amici di disavventure e orrori universitari, perché ad ogni esame fatto insieme, ci siamo aiutati come una squadra, al punto da creare una nuova religione, grazie all'illuminato Alfredo Baratta, che ha divulgato il nome del messia Carlo. Voglio continuare a spargere il nome del messia e organizzare la serata giochi da tavolo; "STAYAWAY" ci aspetta.

Un profondo ringraziamento a tutto il gruppo "Carlo nostro Re", in particolare a:

Buco, per avermi fatto provare il brivido di prendere il pullman all'ultimo secondo. Ci sarò sempre per farti uscire dai buchi in cui cadrà e ti ringrazio anche per essere stato un ottimo compagno di viaggi in macchina, se non fosse per la tua Playlist di musica. Consiglio: non ascoltate mai la canzone "CAZZI FREESTYLE".

Raffaele, mio fedele compagno di guerra, per essere stato l'unico che ha sempre accettato e affrontato le mie idee suicide, pur di completare tutti gli esami nei modi più veloci e "semplici" possibili. Se non fosse stato per la tua chiaroveggenza non sarei mai riuscito a superare l'esame più terribile di tutti, PA. Spero che prima di leggere questi ringraziamenti saremo riusciti a fare quella crostata. Se così non fosse, mi metto il naso da clown e comunico qui davanti a tutti che in questa settimana faremo questa maledetta crostata. Dammi un giorno e un orario e io ci sarò.

Un ringraziamento al gruppo "Scolarette Anime" per aver condiviso idee stupide come le mie. Vi aspetto per la prossima sfilata in gonnella, magari al Termoli Comics. Un grazie al Conad di Fisciano, che ci ha sempre supportato grazie ai suoi panini e alle sue monster, fondamentali per le sessioni di studio. Grazie alle salviettine imbevute di Maury's che mi hanno sempre accompagnato e pulito ad ogni esame (sì, esattamente, io sfogo l'ansia in bagno).

Ringrazio tutti i miei amici che mi hanno sempre fatto divertire malgrado la mia frequente assenza alle uscite ma, nonostante questo, hanno continuato ad invitarmi senza mai lamentarsi (non tutti). Tra questi ringrazio in particolare:

Luca, mio grandissimo amico, che ormai mi ha soprannominato "munnezz" per i molteplici "no" alle uscite, che mi pento di non aver fatto e che spero di recuperare al più presto.

Noemi, spero per prima cosa che sia andato tutto bene e che ora tu stia qui. Tu sei stata una delle persone che ha più inficiato sul mio percorso accademico, anche se non in modo diretto, forse non lo sai, ma sei il motivo principale del perché ora io sia qui a fare questo "discorso", nonostante la nostra grande distanza sei stata una delle persone che mi è stata più vicino, forse anche troppo essendo che non mi mollavi nemmeno quando dormivamo (intendo che ad ogni notte stiamo in videochiamata per poter dormire), sì, è molto strano da dire e non è normale, ne sono consapevole, ma penso che questo sia dato dalla nostra relazione a distanza, però se dovessi vederci qualcosa di positivo nella nostra realzione è che proprio questa nostra distanza mi

abbia portato a dare il 1000%, in questo ultimo anno il forte desiderio di liberarmi dall'università e stare con te, mi ha dato la motivazione e la forza di dare in 1 anno e 3 mesi(periodo di lezioni) ben 13 esami su 22, questo mi sembra assurdo, soprattutto dopo aver passato i primi 2 anni ad aver fatto in totale solo 9 esami, praticamente non facevo nulla ma nonostante questo ci sono riuscito solo grazie a te, considera parte di questa laurea tua,

Grazie di esserci stata.

Questa tesi ha contribuito a piantare un albero in Kenya tramite il progetto Treedom.

<https://www.treedom.net/it/user/sesalab/event/sesa-random-forest>