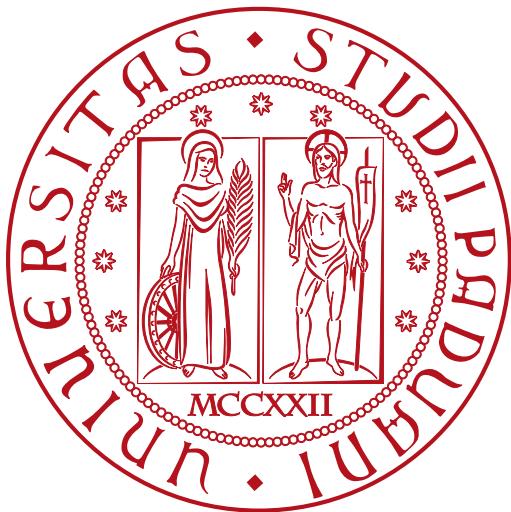


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



**Sviluppo di un progetto di Game Design
con elementi a tema di Machine Learning
e Intelligenza Artificiale**

Tesi di Laurea

Relatore

Prof. Vardanega

Laureando

Filippo Sabbadin

Matricola 2010008

Ringraziamenti

Padova, Settembre 2025

Filippo Sabbadin

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di *stage* curricolare, della durata di circa trecento ore, dal laureando Filippo Sabbadin presso l'azienda Zucchetti S.p.A.. Lo *stage* è stato condotto sotto la supervisione del tutor aziendale Gregorio Piccoli, mentre il prof. Prof. Vardanega ha ricoperto il ruolo di tutor accademico.

Organizzazione del testo

Il primo capitolo presenta l'azienda ospitante, illustrando il contesto organizzativo e produttivo in cui si è svolto lo *stage*, i processi interni adottati e la tipologia di clientela a cui si rivolge. Vengono inoltre descritte le principali tecnologie di supporto utilizzate dal personale e la propensione dell'azienda all'innovazione.

Il secondo capitolo approfondisce il rapporto dell'azienda con gli *stage*, l'interesse verso il progetto svolto e le motivazioni della scelta. Vengono illustrati gli obiettivi e i vincoli concordati con il tutor aziendale, la pianificazione e il calendario delle attività, l'organizzazione del lavoro e le principali tecnologie utilizzate. Infine, si analizza come il progetto si inserisce nel contesto di innovazione e mercato dell'azienda.

Il terzo capitolo descrive in dettaglio l'analisi dei requisiti con i casi d'uso e la lista dei requisiti, l'architettura, *test* e validazione dello *stage*. Seguirà, infine, una descrizione dei risultati che ho raggiunto sul piano qualitativo e quantitativo.

Il quarto capitolo per finire, descrive l'esperienza personale di *stage*, quanti obiettivi soddisfatti rispetto agli obiettivi totali dichiarati nel secondo capitolo, la maturazione durante lo *stage*, con conoscenze ed abilità acquisite durante il periodo.

Infine, verrà fatto un confronto tra le competenze richieste a inizio *stage* rispetto a quelle erogate dal corso di studi, con eventuali lacune su quest'ultimo su competenze che sarebbero state utili per lo *stage* o mondo del lavoro.

Convenzioni tipografiche

Durante la stesura del testo ho scelto di adottare le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini di uso non comune menzionati vengono definiti nel glossario, situato alla fine del documento (p. 75);

- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *termine_G* e ne viene riportata una breve descrizione del termine a piè di pagina;
- i termini in lingua straniera non di uso comune o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- all'inizio di ogni capitolo viene riportato un breve sommario sugli argomenti principali che il capitolo tratta;
- i nomi di funzioni, variabili o classi appartenenti ad un linguaggio di programmazione vengono scritte con un carattere **monospaziato**;
- le citazioni ad un libro o ad una risorsa presente nella bibliografia (p. 80) saranno affiancate dal rispettivo numero identificativo, es. [1];
- ogni immagine sarà accompagnata da un titolo e verrà elencata nel suo indice apposito a inizio documento, esempio:

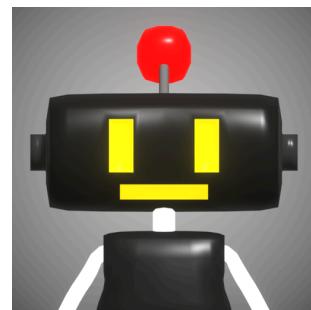


Figura 1: Immagine esempio

- allo stesso modo, ogni tabella sarà seguita da un suo titolo ed inserita nel suo indice apposito. Inoltre ogni riga avrà un colore diverso dalle righe vicine per renderla più accessibile, esempio:

Titolo 1	Titolo 2
Valore 1-1	Valore 1-2
Valore 2-1	Valore 2-2

Tabella 1: Tabella esempio

- i blocchi di codice sono rappresentati nel seguente modo:

```
1 float Q_rsqrt( float number ){
2     long i;
3     float x2, y;
4     const float threehalves = 1.5F;
5     x2 = number * 0.5F;
6     y = number;
7     i = * (long * ) &y;
8     i = 0x5f3759df - (i>>1);
9     y = * (float * ) &i;
10    y = y * ( threehalves - ( x2 * y * y ) );
11    return y;
12 }
```

C C

Codice 1: Codice d'esempio.

Indice

1 L’azienda	1.
1.1 Introduzione	1.
1.1.1 Informazioni sull’azienda	1.
1.1.2 Sede dello <i>stage</i>	1.
1.2 Contesto organizzativo e produttivo	2.
1.3 Processi interni utilizzati	3.
1.3.1 Sviluppo, manutenzione ed organizzazione del lavoro	3.
1.3.2 Tecnologie di supporto	3.
1.4 Clientela e prodotti	4.
1.4.1 Tipologia di clientela	4.
1.4.2 Prodotti e servizi	5.
1.5 Propensione per l’innovazione	5.
2 Lo <i>stage</i>	7.
2.1 Rapporto dell’azienda con gli <i>stage</i>	7.
2.2 Interesse personale e dell’azienda verso lo <i>stage</i>	7.
2.2.1 Proposta del progetto	7.
2.2.2 Motivazioni personali	7.
2.2.3 Scelta dell’azienda	8.
2.2.4 Supporto dell’azienda verso il progetto	8.
2.3 Descrizione del progetto	9.
2.3.1 Struttura del gioco	9.
2.3.2 Rapporto del progetto con l’innovazione	10.
2.3.3 Aspettative	11.
2.4 Obiettivi	11.
2.5 Vincoli	12.
2.5.1 Vincoli temporali e tecnologici	12.
2.5.2 Pianificazione	12.
2.5.3 Calendario	13.
2.5.4 Organizzazione del lavoro	14.
2.5.5 Tecnologie usate	17.
3 Il progetto	21.
3.1 Analisi dei rischi	21.
3.1.1 Rischi organizzativi	21.
3.1.2 Rischi tecnici	22.

3.1.3	Rischi di analisi e progettazione	23.
3.2	Analisi dei requisiti	24.
3.2.1	Attori	24.
3.2.2	Casi d'uso	24.
3.2.3.	Requisiti	41.
3.2.4.	Lista dei requisiti	41.
3.3.	Architettura	45.
3.3.1.	Concetti chiave di <i>Godot</i>	45.
3.3.2.	Funzioni comuni	48.
3.3.3.	Classi del giocatore	49.
3.3.4.	Entità <i>interagibili</i>	53.
3.3.5.	Gestione dei salvataggi	54.
3.3.6.	<code>LevelsTransition</code>	54.
3.3.7.	<code>OptionsSave</code>	54.
3.3.8.	Struttura base di un livello	55.
3.3.9.	<code>LRCannon</code>	56.
3.3.10.	<code>LinearRegressionGraph</code>	56.
3.3.11.	Livello <i>Albero di decisione</i>	58.
3.3.12.	Livello <i>Causalità</i>	60.
3.4.	Verifica e validazione	62.
3.4.1.	Macchina di <i>test</i>	62.
3.4.2.	Nomenclatura <i>test</i>	62.
3.4.3.	<i>Test</i> di unità	63.
3.4.4.	<i>Test</i> di integrazione	67.
3.4.5.	<i>Test</i> di sistema	70.
3.4.6.	<i>Test</i> di accettazione	71.
3.5.	Risultati ottenuti	71.
4	Conclusioni	73.
4.1	Obiettivi stage soddisfatti	73.
4.2	Esperienze acquisite	73.
4.3	Differenza tra stage e percorso studi	73.
4.3.1	Lacune sul percorso studi	73.
4.4	Pensieri finali	73.
Glossario	75.	
Bibliografia	80.	

Elenco delle Figure

Figura 1	Immagine esempio	vi
Figura 2	Settori e temi di cui si occupa l'azienda. Fonte: Zucchetti	4.
Figura 3	Livello <i>Regressione lineare</i>	9.
Figura 4	Livello <i>Albero di decisione</i>	10.
Figura 5	Livello <i>Causalità</i>	10.
Figura 6	Attore principale	24.
Figura 7	Diagramma UML <i>use case</i> sul movimento	24.
Figura 8	Diagramma UML <i>use case</i> sul salto	25.
Figura 9	Rotazione telecamera	26.
Figura 10	Caduta	27.
Figura 11	Raccolta collezionabile	27.
Figura 12	Interazione <i>_entità_</i>	28.
Figura 13	Interazione <i>_entità_</i> automatica	28.
Figura 14	Interazione manuale	29.
Figura 15	Interazione personaggio non giocabile	30.
Figura 16	Prendere un oggetto	31.
Figura 17	Lasciare un oggetto	31.
Figura 18	Interazione cartello	32.
Figura 19	Interazione zona di transizione	33.
Figura 20	Interazione con macchina <i>LR</i>	33.
Figura 21	Scelte albero di decisione	34.
Figura 22	Inserimento dell'oggetto nello spazio dedicato	35.
Figura 23	Apertura menu di pausa	36.
Figura 24	Menu di pausa	36.
Figura 25	Menu opzioni	38.
Figura 26	Accensione unità esterna condizionatore	40.
Figura 27	Visualizzazione <i>input</i> nella <i>UI</i>	40.
Figura 28	<i>Scena</i> del personaggio del giocatore	46.
Figura 29	Diagramma <i>UML</i> delle classi del giocatore	49.
Figura 30	Diagramma <i>UML</i> delle classi della telecamera del giocatore	50.
Figura 31	Diagramma <i>UML</i> sulla struttura della macchina di stati	51.
Figura 32	Diagramma sul flusso degli stati del giocatore	52.
Figura 33	Diagramma <i>UML</i> degli oggetti con cui il giocatore può interagire	53.
Figura 34	Diagramma sul funzionamento dei salvataggi	54.
Figura 35	Diagramma <i>UML</i> delle classi <i>Autoloads</i>	54.
Figura 36	Diagramma delle classi di un livello base	55.

Figura 37 Diagramma sul funzionamento di un grafico <i>Linear Regression</i> nel gioco	56.
Figura 38 Diagramma sul funzionamento dell’Albero di decisione	58.
Figura 39 Diagramma del livello <i>Causalità</i>	60.
Figura 40 Diagramma sul funzionamento dei personaggi non giocabili nella scena di intermezzo	61.

Elenco delle Tabelle

Tabella 1 Tabella esempio	vi
Tabella 2 Obiettivi del progetto	11.
Tabella 3 Vincoli del progetto	12.
Tabella 4 Pianificazione del lavoro in ore	13.
Tabella 5 tabella dei documenti	15.
Tabella 6 Linguaggi di programmazione utilizzati	17.
Tabella 7 <i>Softwares</i> utilizzati	18.
Tabella 8 Strumenti e servizi utilizzati	18.
Tabella 9 Tipi di <i>file</i> utilizzati	19.
Tabella 10 Errata pianificazione dei tempi	21.
Tabella 11 Impegni personali o universitari	21.
Tabella 12 Mancanza di competenze tecniche	22.
Tabella 13 Tecnologie non adeguate	22.
Tabella 14 Cambio dei requisiti	23.
Tabella 15 Errore nella progettazione dell’architettura	23.
Tabella 16 Tabella dei requisiti funzionali	41.
Tabella 17 Tabella dei requisiti di qualità	44.
Tabella 18 Tabella dei requisiti di accessibilità	45.
Tabella 19 Componenti della macchina su cui sono stati eseguiti i <i>test</i>	62.
Tabella 20 <i>Test</i> di unità	63.
Tabella 21 <i>Test</i> di integrazione	67.
Tabella 22 <i>Test</i> di sistema	70.
Tabella 23 Tabella dei <i>test</i> di accettazione	71.

Elenco dei Codici Sorgente

Codice 1 Codice d'esempio.	vii
---------------------------------	-----

Capitolo 1

L’azienda

In questo capitolo descrivo l’azienda, il contesto organizzativo in cui sono stato inserito, i processi interni utilizzati e la tipologia di clientela a cui si rivolge.

1.1 Introduzione

1.1.1 Informazioni sull’azienda

L’azienda Zucchetti Spa opera nel settore informatico da oltre 45 anni ed offre una vasta gamma di soluzioni *software* e servizi per le aziende, mantenendosi sempre aggiornati su tematiche come il diritto civile, contabilità, fiscalità, diritto del lavoro e previdenza.

Zucchetti ha un organico di circa 9.000 persone, con oltre 2.000 di queste dedicate a ricerca e sviluppo, dimostrando una forte attenzione all’innovazione tecnologica e al miglioramento continuo dei propri prodotti.

L’azienda investe costantemente in nuove tecnologie e nella formazione del personale, favorendo un ambiente dinamico e orientato alla crescita professionale.

1.1.2 Sede dello *stage*

La sede dell’azienda dove ho svolto lo *stage* si trova a Padova, in un’area ben collegata e facilmente raggiungibile.

L’ufficio è situato in un edificio moderno e funzionale, dotato di spazi di lavoro aperti e aree dedicate alla collaborazione tra i membri del *team*.

L’ambiente di lavoro si è rivelato collaborativo e stimolante, con una particolare attenzione alla condivisione delle conoscenze e al supporto reciproco tra colleghi. Ho potuto osservare da vicino l’organizzazione del lavoro e l’interazione tra il personale.

Questa esperienza mi ha permesso di comprendere meglio le dinamiche aziendali e di apprezzare l’importanza dell’innovazione continua all’interno dell’azienda.

1.2 Contesto organizzativo e produttivo

Durante lo *stage* sono stato per lo più indipendente, tuttavia qualora avessi avuto bisogno di aiuto, potevo chiedere ad un gruppo composto da professionisti con competenze eterogenee, tra cui sviluppatori, analisti e *project manager*. Ho potuto osservare come la collaborazione e il confronto tra colleghi fossero elementi fondamentali per il buon andamento dei progetti.

L'ambiente lavorativo era caratterizzato da una forte attenzione alla qualità del prodotto e al rispetto delle scadenze, con un approccio orientato al miglioramento continuo dei processi produttivi.

Questo mi ha aiutato a comprendere l'importanza di lavorare in un contesto organizzato e strutturato, dove ogni componente contribuisce al raggiungimento degli obiettivi comuni.

1.3 Processi interni utilizzati

1.3.1 Sviluppo, manutenzione ed organizzazione del lavoro

Durante tutto il periodo di *stage*, ho svolto le mie attività seguendo i processi interni decisi dall’azienda, che prevedevano una gestione strutturata del progetto e una chiara suddivisione dei compiti da eseguire. I processi interni dell’azienda comprendevano fasi distinte per lo sviluppo, la manutenzione e l’organizzazione del lavoro.

- Durante l’organizzazione del lavoro, ho potuto notare una forte comunicazione tra il personale in ufficio ed anche con i membri che lavoravano in *smart working* durante alcuni giorni.
- Per lo sviluppo, ognuno aveva compiti specifici e responsabilità ben definite, con delle eventuali piccole discussioni per chiedere chiarimenti o approfondimenti, ad esempio, sul codice. Molto spesso vedeva due o più membri lavorare insieme su uno stesso argomento, scambiandosi idee e suggerimenti per migliorare il prodotto finale oppure per risolvere eventuali problemi.
- Tutto ciò che veniva sviluppato e completato, era anche mantenuto in base alle esigenze del cliente, in caso di problemi o richieste particolari, come problemi di compatibilità con versioni più vecchie dei *browser*.

1.3.2 Tecnologie di supporto

Per facilitare la comunicazione e la collaborazione tra i membri del *team*, l’azienda ha adottato diverse tecnologie di supporto. La comunicazione tra i membri del *team* avveniva principalmente tramite strumenti digitali di collaborazione, ad esempio *Microsoft Teams*_G, che facilitavano la condivisione delle informazioni e il coordinamento delle attività. Attraverso questa piattaforma, i membri svolgevano anche riunioni o videochiamate, permettendo una comunicazione rapida ed efficace, anche a distanza.

Inoltre, per sincronizzare i cambiamenti e garantire che tutti i membri del *team* fossero aggiornati, venivano utilizzati sistemi di versionamento e *repository* condivisi, in un *database*_G interno. Questi strumenti hanno reso possibile una gestione efficiente delle attività e una rapida risoluzione dei problemi, contribuendo a mantenere un flusso di lavoro fluido e produttivo.

database: insieme organizzato di dati, generalmente memorizzato e gestito in modo da facilitarne l’accesso e la manipolazione

Microsoft Teams: piattaforma di comunicazione e collaborazione sviluppata da Microsoft.

1.4 Clientela e prodotti

1.4.1 Tipologia di clientela

Zucchetti si rivolge a una clientela molto diversificata, che comprende sia piccole e medie imprese, sia grandi aziende, tutte queste private.

Questa varietà di clientela rappresenta uno stimolo costante all'innovazione e all'adattamento dei prodotti alle evoluzioni del mercato, contribuendo a mantenere un'offerta sempre aggiornata e competitiva.



market@zucchetti.it - www.zucchetti.it



Figura 2: Settori e temi di cui si occupa l'azienda. Fonte: Zucchetti

Ho potuto osservare una discussione molto aperta e libera tra i membri del team riguardo alle esigenze e alle aspettative dei clienti, modificando i prodotti in base alle richieste e necessità di questi ultimi.

1.4.2 Prodotti e servizi

L'azienda offre una vasta gamma di prodotti con funzionalità diverse.

Alcuni esempi sono:

- soluzioni gestionali per la contabilità;
- la gestione del personale;
- la gestione della produzione;
- *software* verticali progettati per rispondere alle esigenze di settori specifici come:
 - ▶ sanità;
 - ▶ *retail*;
 - ▶ logistica;
 - ▶ produzione industriale.

Questi prodotti sono pensati per supportare le aziende nell'ottimizzazione dei processi, nella gestione efficiente delle risorse e nell'adeguamento alle normative vigenti.

Oltre ai *software*, Zucchetti fornisce anche servizi, che possono essere di:

- consulenza;
- assistenza tecnica;
- formazione.

In questo modo, Zucchetti garantisce un supporto completo ai propri clienti durante tutte le fasi di adozione e utilizzo delle soluzioni offerte.

1.5 Propensione per l'innovazione

Zucchetti investe costantemente in ricerca e sviluppo, con oltre 2.000 persone dedicate a queste attività. Questo impegno si traduce in una continua introduzione di nuove tecnologie, metodologie di lavoro innovative e aggiornamenti dei prodotti offerti. L'azienda promuove attivamente la formazione del personale e la sperimentazione di soluzioni all'avanguardia, favorendo un ambiente in cui l'innovazione è parte integrante della cultura aziendale.

Durante il mio *stage*, ho potuto osservare come le idee innovative vengano accolte con interesse e valutate attentamente, sia a livello di processo che di prodotto. Ho avuto modo, inoltre, di assistere ad una sessione di *brainstorming* dimostrazione di come l'azienda sia aperta a nuove idee e approcci.

Argomento principale delle ricerche che il personale dell'azienda stava svolgendo, erano gli *LLM*, tema ancora molto nuovo ed inesplorato nel mondo dell'informatica.

brainstorming: tecnica di generazione di idee in gruppo, in cui i partecipanti sono incoraggiati a esprimere liberamente le proprie idee.

LLM - Large Language Model: modello di intelligenza artificiale progettato per comprendere e generare testo in linguaggio naturale.

Per la maggior parte, il personale in azienda si occupava di test e addestramento dei vari modelli, cambio dei parametri, ad esempio, la *temperatura_G*, analizzando gli *output* che questi generavano, la correttezza di questi, e molto altro.

Questa collaborazione contribuisce a generare nuove soluzioni e a mantenere elevato il livello di competitività sul mercato.

La propensione all'innovazione dell'azienda si riflette nella rapidità con cui adotta strumenti digitali e tecnologie emergenti, garantendo così un costante miglioramento dei servizi e delle soluzioni offerte ai clienti.

temperatura: parametro che controlla la casualità delle risposte generate da un LLM.

Capitolo 2

Lo *stage*

In questo capitolo approfondisco il rapporto con l'azienda ospitante verso gli stage in generale, come ha supportato il mio stage, il perché della mia scelta e gli obiettivi e vincoli decisi con il tutor aziendale. Infine verrà messo a confronto il tema dello stage con l'innovazione ed il mercato dove viene inserito il progetto.

2.1 Rapporto dell'azienda con gli *stage*

Da molti anni l'azienda Zucchetti Spa si presenta all'evento *StageIT_G*, incontrando un vasto numero di studenti, proponendo loro nuovi temi da approfondire per i progetti di *stage*.

Oltre a questi progetti, l'azienda è sempre disponibile a valutare nuove idee di progetti proposti dagli studenti, ascoltando le loro esigenze e suggerimenti.

Ho potuto osservare questa disponibilità anche durante lo *stage*, dove erano presenti dei miei colleghi, con alcuni di loro che stavano svolgendo progetti che non erano stati proposti dall'azienda, ma decisi da loro, segno che l'azienda è sempre aperta a nuove idee e proposte.

2.2 Interesse personale e dell'azienda verso lo *stage*

2.2.1 Proposta del progetto

Sempre durante l'evento *StageIT* 2025, ho avuto la possibilità di incontrare il *tutor* aziendale, il quale, prima di presentare i progetti proposti dall'azienda, mi ha dato la possibilità di proporre un tema di progetto diverso.

La mia proposta è stata un progetto di *Game Design_G*.

2.2.2 Motivazioni personali

Il tema di *Game Design* è un argomento che mi appassiona da tempo e su cui ho già avuto modo di lavorare in progetti personali, seppure di piccole dimensioni. Inoltre, il *Game Design* offre l'opportunità di esplorare la creatività e di sviluppare nuove idee in un contesto ludico.

Game Design: disciplina che si occupa della progettazione e dello sviluppo di giochi.

StageIT: evento orientato al lavoro, dedicato agli studenti per aiutarli a trovare aziende dove svolgere l'attività di *stage*.

Questo tema mi ha permesso di combinare la mia passione per lo sviluppo di applicazioni videoludiche con l'apprendimento di nuove competenze tecniche e di sviluppo, rendendo l'esperienza di *stage* più coinvolgente e stimolante.

2.2.3 Scelta dell'azienda

Il motivo principale della mia scelta sull'azienda è stata la disponibilità aperta ad esplorare nuove idee ed approcci.

Ulteriore motivo della mia scelta è stata la posizione della sede dove ho svolto lo *stage*, che ho trovato molto comoda da raggiungere, attraverso i mezzi di trasporto disponibili.

2.2.4 Supporto dell'azienda verso il progetto

Nonostante ritenessi che la mia proposta avesse poco valore, il *tutor* ha mostrato un interesse genuino e ha incoraggiato la mia idea, portandomi a svilupparla ulteriormente.

Successivo all'evento *StageIT*, ho avuto la possibilità di parlare con il *tutor* aziendale, attraverso una videochiamata, il quale mi ha supportato nella definizione del progetto, aiutandomi a capire come svilupparlo al meglio e mostrandomi anche esempi di progetti passati svolti sullo stesso argomento.

Visto che l'argomento del momento nel mondo dell'informatica era l'uso degli *LLM*, il *tutor* ha proposto di usare gli argomenti comuni sul tema dell'Intelligenza Artificiale e **Machine Learning (ML)_G** per arricchire il progetto, ed usarli per creare nuove meccaniche nel gioco per creare livelli unici, che il giocatore deve completare. In questo modo, tramite il gioco, l'utente impara nuovi argomenti riguardo al mondo dell'Intelligenza artificiale e *Machine Learning* ed apprende il loro funzionamento.

Tra questi temi, sono spiccati di più:

- *Regressione lineare_G* ;
- *Alberi di decisione_G* ;
- causalità;
- *Nearest Neighbor_G* ;
- *Support Vector Machines_G*.

alberi di decisione: modello predittivo che rappresenta le decisioni e le loro possibili conseguenze sotto forma di un albero.

Machine Learning: disciplina che si occupa dello sviluppo di algoritmi che permettono ai computer di apprendere dai dati e migliorare le proprie prestazioni nel tempo senza essere esplicitamente programmati.

Nearest neighbor: algoritmo di classificazione che assegna un'etichetta a un campione in base alle etichette dei suoi vicini più prossimi nel dataset.

Regressione lineare: tecnica statistica utilizzata per modellare la relazione tra una variabile dipendente e una o più variabili indipendenti, assumendo una relazione lineare.

Support Vector Machines: classe di algoritmi di apprendimento che cercano di trovare l'iperpiano ottimale che separa le classi nel dataset.

2.3 Descrizione del progetto

2.3.1 Struttura del gioco

Il progetto si tratta di un videogioco educativo, che ha come obiettivo quello di insegnare i concetti base dell'Intelligenza Artificiale e *Machine Learning* in modo semplice e divertente.

Oltre al livello *tutorial* ed al livello principale, dove il giocatore può scegliere il livello che vuole affrontare, sono presenti 3 livelli, ognuno dei quali insegna un concetto diverso:

- il livello della **regressione lineare**, dove la linea nel grafico diventa un ponte su cui il personaggio può camminare, tuttavia la direzione non è corretta e bisogna modificarla aggiungendo nuovi punti nel grafico, modificando la direzione della linea;

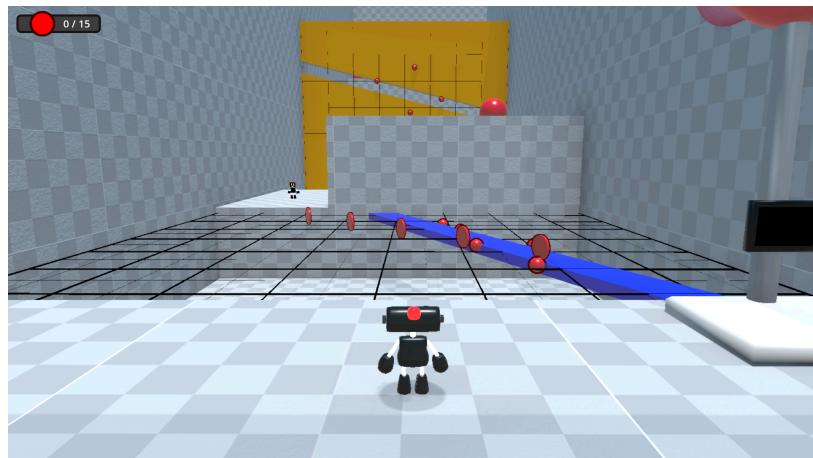


Figura 3: Livello *Regressione lineare*

tutorial: guida passo-passo che insegna come utilizzare un software o completare un'attività specifica.

- il livello dell'**albero di decisione**, dove l'utente dovrà classificare diverse razze di cani in base ai valori, già assegnati, in un albero di decisione;



Figura 4: Livello *Albero di decisione*

- il livello della **causalità**, dove l'obiettivo principale è cercare la giusta causa di quello che sta avvenendo nel livello, ed in caso di risposta corretta, l'utente viene avvisato e premiato.



Figura 5: Livello *Causalità*

2.3.2 Rapporto del progetto con l'innovazione

Il progetto si inserisce in un mercato in crescita, dove l'innovazione rappresenta un elemento chiave per attrarre e coinvolgere nuovi utenti.

La clientela *target* del progetto è costituita principalmente da giovani e appassionati di tecnologia e videogiochi, interessati sia all'aspetto ludico sia a quello educativo. Il prodotto si rivolge a chi desidera apprendere concetti di Intelligenza Artificiale e *Machine Learning* in modo interattivo e coinvolgente, offrendo un'esperienza che unisce apprendimento e divertimento.

2.3.3 Aspettative

Prima di iniziare lo *stage* avevo delle aspettative riguardo al progetto, che si sono rivelate, alla fine, corrette.

Tra queste aspettative, vi erano:

- imparare nuove tecnologie e strumenti per lo sviluppo di giochi;
- migliorare le mie capacità di programmazione e *problem solving*;
- acquisire esperienza pratica nel lavoro di squadra e nella gestione di progetti;
- ricevere *feedback* costruttivo sul mio lavoro e sulle mie idee.

2.4 Obiettivi

Nella seguente tabella, vengono elencati gli obiettivi personali che ho fissato, insieme al *tutor* aziendale, suddivisi in obbligatori e desiderabili.

Gli obiettivi sono elencati con un codice, costituito da un prefisso e un numero. Il prefisso indica con:

- **O**: gli obiettivi obbligatori, rappresentano le competenze e i risultati minimi da raggiungere durante lo *stage*;
- **D**: gli obiettivi desiderabili, sono traguardi aggiuntivi che arricchiscono ulteriormente il progetto e l'esperienza formativa.

Obbligatori	
O-1	Progettazione e realizzazione delle principali meccaniche di gioco (movimento 3D, interazione con altri oggetti...)
O-2	Implementazione degli argomenti di <i>AI</i> e <i>ML</i> al gioco (Regressione lineare, Alberi di decisione, Causalità...)
O-3	Implementazione di salvataggi e caricamenti dei dati attraverso file di tipo <i>.json</i> oppure <i>.ini</i>
Desiderabili	
D-1	Supporto della lingua inglese oltre all'italiano, con opzione di cambiare lingua di gioco
D-2	Implementazione di <i>shaders</i> , utilizzando <i>script</i> di tipo <i>OpenGLG</i>
D-3	Uso dei linguaggi <i>C#</i> o <i>C++</i> per migliorare le prestazioni
D-4	Implementazione di un modello di <i>LLM</i> per conversazioni tra personaggi all'interno del gioco

Tabella 2: Obiettivi del progetto

OpenGL: linguaggio di programmazione grafica utilizzato per creare applicazioni 3D e 2D.

shader: modello di sviluppo software che promuove la collaborazione e la condivisione del codice sorgente.

2.5 Vincoli

2.5.1 Vincoli temporali e tecnologici

Durante il secondo colloquio con il *tutor* aziendale, sono stati definiti dei vincoli obbligatori del progetto da rispettare, concordati tra me e il *tutor* aziendale. I vincoli sono indicati nella seguente tabella, con il prefisso:

- **VTM**: vincoli temporali;
- **VTC**: vincoli tecnologici;

Temporali	
VTM-1	Il progetto deve essere realizzato in un tempo massimo di 320 ore
VTM-2	Il numero di ore settimanali non può essere superiore a 40
Tecnologici	
VTC-1	Il progetto deve essere sviluppato utilizzando il <i>motore di gioco_G</i> <i>open source_G</i> <i>Godot</i>
VTC-2	L'applicazione finale deve essere un eseguibile, senza bisogno di installazione o altri strumenti
VTC-3	Il gioco deve utilizzare una grafica 3D con un movimento del personaggio in terza persona, tridimensionale

Tabella 3: Vincoli del progetto

2.5.2 Pianificazione

Il gioco contiene 3 livelli, per ognuno di questi livelli sono state dedicate due settimane. Nella tabella a pagina successiva vengono riportate le ore totali pianificate del progetto:

open source: modello di sviluppo software che promuove la collaborazione e la condivisione del codice sorgente.

motore di gioco: software progettato per facilitare lo sviluppo di videogiochi, fornendo strumenti e funzionalità per la gestione della grafica, della fisica, dell'audio e di altre componenti del gioco.

Durata (ore)	Descrizione attività
24	Pianificazione struttura applicazione Pianificazione stutura livelli Pianificazione implementazione meccaniche di gioco
63	Stesura della documentazione
24	Stesura documentazione relativa ad analisi e progettazione
16	Stesura delle metriche di qualità
15	Stesura delle norme e piano di progetto
8	Stesura del Manuale Utente
177	Sviluppo dei livelli
59	Sviluppo primo livello
59	Sviluppo secondo livello
59	Sviluppo terzo livello
40	<i>Test e verifica dell'applicazione</i>
Totale ore: 304	

Tabella 4: Pianificazione del lavoro in ore

2.5.3 Calendario

Il periodo di *stage* è suddiviso in 8 periodi, la cui lunghezza di ognuno corrisponde a una settimana. Qui sotto sono elencate le attività pianificate per ogni settimana:

- **Settimana 1 | 18/06 - 20/06 | 24 ore:**
 - Incontro con il personale dell'azienda per discutere i requisiti dell'applicazione da sviluppare.
 - Verifica credenziali e strumenti di lavoro assegnati
 - Pianificazione e progettazione dell'applicazione.
 - Inizio sviluppo.
- **Settimana 2 | 23/06 - 27/06 | 40 ore:**
 - Approfondimento sul tema *Regressione lineare*.
 - Sviluppo degli oggetti principali del primo livello, implementando gli elementi della *Regressione lineare*.

- **Settimana 3 | 30/06 - 04/07 | 40 ore:**
 - Approfondimento sul tema *Alberi di decisione*.
 - Sviluppo degli oggetti principali del secondo livello.
- **Settimana 4 | 07/07 - 11/07 | 40 ore:**
 - Approfondimento sull'argomento *Causalità*.
 - Sviluppo degli oggetti principali del terzo livello.
 - Compilazione del *PoC_G*.
- **Settimana 5 | 14/07 - 18/07 | 40 ore:**
 - Sviluppo del primo livello, sul tema *Regressione Lineare*, con gli oggetti creati nella seconda settimana.
- **Settimana 6 | 21/07 - 25/07 | 40 ore:**
 - Sviluppo del secondo livello, sul tema *Alberi di decisione*, implementando gli oggetti creati nella terza settimana.
- **Settimana 7 | 28/07 - 01/08 | 40 ore:**
 - Sviluppo del terzo livello, sul tema *Causalità*, implementando gli oggetti creati nella quarta settimana.
- **Settimana 8 | 04/08 - 08/08 | 40 ore:**
 - Stesura dei *test*.
 - Compilazione dell'*MVP_G*.

2.5.4 Organizzazione del lavoro

- **Organizzazione generale:**

Ogni giorno, sceglievo le attività da svolgere in base allo stato di avanzamento del progetto, tenendo conto delle priorità e delle scadenze.

Utilizzavo strumenti digitali per la gestione delle attività, come sistemi di versionamento o uso di *ticket_G*, per aiutarmi a tenere traccia dei compiti svolti. Ogni fase del lavoro, dalla raccolta dei requisiti allo sviluppo e alla verifica, veniva documentata e tracciata per garantire trasparenza ed efficienza.

Questi processi interni mi hanno permesso di acquisire una maggiore consapevolezza sull'importanza dell'organizzazione e della comunicazione all'interno di un contesto lavorativo strutturato.

Durante lo sviluppo, sono state imposte le seguenti regole per garantire corretta organizzazione. Le regole sono suddivise in base all'attività.

PoC: descrive una dimostrazione pratica che ha lo scopo di verificare la fattibilità o il potenziale di un'idea, concetto o soluzione.

MVP: descrive la versione minima di un prodotto che include solo le funzionalità essenziali per essere utilizzato dagli utenti.

ticket: segnalazione o richiesta registrata in un sistema di tracciamento che descrive un'attività da svolgere.

- **Documentazione:**

Il processo di fornitura deve essere documentato in modo dettagliato, in modo da garantire la tracciabilità delle attività svolte e delle decisioni prese.

I documenti che il progetto presenta sono:

Nome	Descrizione
Analisi dei requisiti	Definisce tutti gli i casi d'uso e i requisiti funzionali del progetto. Questi sono stati raccolti in collaborazione con il tutor aziendale e sono stati utilizzati come base per la progettazione e lo sviluppo del <i>software</i>
Piano di progetto	Definisce le attività da svolgere e i tempi previsti per lo sviluppo del <i>software</i> , viene descritto in dettaglio ogni periodo di sviluppo, con una retrospettiva delle attività svolte e una pianificazione delle attività future
Norme di progetto	Definisce le regole e le convenzioni da seguire durante lo sviluppo del <i>software</i> , come la nomenclatura dei <i>file</i> , la struttura del codice e le pratiche di programmazione da seguire
Piano di qualifica	Definisce le metriche che vengono usate per garantire la qualità del prodotto <i>software</i> . Vengono inoltre scritte le modalità di <i>test</i> e verifica del <i>software</i> , in modo da garantire che il prodotto soddisfi i requisiti stabiliti
Specifica tecnica	Describe in dettaglio l'architettura del sistema, i componenti <i>software</i> e le loro interazioni
Manuale utente	Fornisce istruzioni dettagliate su come utilizzare il <i>software</i> all'utente e garantirne il corretto funzionamento

Tabella 5: tabella dei documenti

- **Codifica:**

Tutti i *file* contenenti codice del gioco sono salvati come file *.gd*, e sono scritti con il linguaggio di programmazione ***GDScript***. I nomi delle classi sono salvate con una nomenclatura ***PascalCase***, mentre i nomi dei *file* e delle variabili usano ***snake_case***.

GDScript: linguaggio di programmazione specifico per il motore di gioco *Godot*, progettato per essere semplice e intuitivo.

PascalCase: pratica di scrivere parole composte o frasi unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole.

snake_case: pratica di scrivere parole composte separando le parole tramite trattino basso, con tutte le lettere minuscole.

Per maggiori dettagli sulla nomenclatura, si seguono le convenzioni della documentazione ufficiale:

https://docs.godotengine.org/it/4.x/tutorials/scripting/gdscript/gdscript_styleguide.html

- **Modellazione:**

Tutti i modelli sono esportati nel formato *.glb_G*. Il *materiale_G* esportato insieme al modello 3D viene esportato senza immagini, come un *placeholder*, dato che verrà rimpiazzato dal materiale presente nei *file* del gioco.

Nel caso il modello 3D presenti animazioni, queste vengono esportate insieme al modello.

- **Animazione:**

Le animazioni sono incluse nel modello durante l'esportazione. Per semplificare l'attività, viene usato un *rig_G* che dispone di *IK_G*. Le animazioni sono già separate prima dell'esportazione e possono essere trovate nella sezione *NLA_G* del software *Blender_G* e selezionate individualmente premendo la linea con il mouse e modificarle usando la scorciatoia *Shift+TAB*.

- **Creazione e modifica di *texture*:**

Le *texture_G* sono salvate come semplici immagini di tipo *.png_G*.

Entrambe le dimensioni della *texture* (larghezza e altezza) devono essere una potenza di 2.

Risoluzioni esempio:

- 256x256;
- 512x512;
- 1024x1024 (1K);
- 2048x2048 (2K).

Di norma, 1024 *pixels* corrispondono a 1 metro.

.glb: formato standard di un modello tridimensionale che legge il modello 3D come un file binario.

.png: formato di immagine raster senza perdita di qualità, ampiamente utilizzato per la grafica web e il design digitale.

Blender: software di modellazione ed animazione 3D usato per creare modelli 3D ed animazioni. **IK - Inverse Kinematics:** descrive il processo di calcolo della posizione delle articolazioni di un modello 3D in base alla posizione finale di una parte del corpo.

NLA - Nonlinear Animations: sistema di gestione delle animazioni in *Blender* che consente di combinare e sovrapporre diverse animazioni in modo non lineare.

rig: struttura scheletrica applicata a un modello 3D.

materiale: insieme di proprietà che definiscono l'aspetto visivo di un oggetto 3D, come colore, riflessione, trasparenza.

texture: immagine applicata a un modello 3D per fornire dettagli visivi, come colori e *pattern*.

- **Verifica e validazione:**

Il processo di verifica ha lo scopo di garantire che il *software* sviluppato soddisfi i requisiti stabiliti e che sia conforme agli *standard* di qualità richiesti.

Vi sono due tipologie di verifica, ognuna è focalizzata sulla verifica di vari aspetti dell'applicazione:

analisi statica: l'analisi statica comporta il controllo del codice prima della sua esecuzione. Questo tipo di verifica non viene applicata solo al codice, ma anche ai documenti del progetto.

Questo metodo viene applicato nei seguenti casi:

- individuazione di *bug* nel codice;
- individuazione di errori di battitura nei documenti;
- verifica della coerenza e completezza della documentazione prodotta;

analisi dinamica: l'analisi dinamica viene eseguita all'esecuzione del *software*. Viene usata per controllare se ci sono errori durante l'esecuzione dell'applicazione e dei suoi componenti.

Questo tipo di verifica permette di individuare malfunzionamenti, errori logici o comportamenti inattesi che possono emergere solo durante l'esecuzione reale del *software*. Le principali attività di analisi dinamica includono:

- esecuzione di *test* di unità e di integrazione per verificare il corretto funzionamento delle singole componenti e della loro interazione;
- monitoraggio delle prestazioni e dell'utilizzo delle risorse durante l'esecuzione;
- individuazione e correzione di *bug* che si manifestano solo in fase di runtime.

2.5.5 Tecnologie usate

Nome	Descrizione	Versione
GDScript	Linguaggio di programmazione di alto livello, con sintassi simile a <i>Python</i> , viene integrato con il motore di gioco <i>Godot</i>	(Legata a <i>Godot</i>)
GDShader	Linguaggio simile a <i>GLSL ES</i> 3.0, usato per la creazione di materiali e <i>shader</i> più complessi	(Legata a <i>Godot</i>)
Typst	Linguaggio utilizzato per la stesura dei documenti	0.13.1

Tabella 6: Linguaggi di programmazione utilizzati

GLSL ES - OpenGL Shading Language for Embedded Systems: linguaggio di shading utilizzato per scrivere *shader*.

Python: linguaggio di programmazione di alto livello, noto per la sua sintassi semplice e leggibile.

Nome	Descrizione	Versione
Godot	Il motore di gioco <i>open source</i> per lo sviluppo del videogioco	4.5-beta3-mono
Blender	<i>Software</i> di modellazione ed animazione 3D usato per creare i modelli 3D ed animazioni nel gioco	4.4.3

Tabella 7: *Softwares* utilizzati

Nome	Descrizione	Versione
Git	Servizio per il controllo della versione	2.50.1
GitHub	Servizio di <i>hosting</i> _G per i progetti <i>software</i> , utilizzato per la gestione del codice sorgente	-
GitHub Actions	Servizio di integrazione continua e distribuzione continua (<i>CI</i> _G / <i>CD</i> _G), utilizzato per compilare i documenti ad ogni <i>push</i> _G	-
Notion	Applicazione per la gestione dei progetti e la collaborazione	2.53

Tabella 8: Strumenti e servizi utilizzati

CI - Continuos Integration: processo di integrazione continua delle modifiche del codice in un *repository* condiviso, garantendo che il codice sia sempre in uno stato funzionante e testato.

CD - Continuos Delivery: processo di rilascio continuo delle modifiche del codice in produzione, garantendo che il *software* sia sempre in uno stato utilizzabile.

hosting: descrive il servizio che consente di archiviare e rendere accessibili *online* siti *web*, applicazioni o progetti *software*.

push: descrive l'azione di inviare le modifiche del codice a un *repository* remoto.

Nome	Descrizione	Versione
*.csv	« <i>Comma separated values</i> », file utilizzato per memorizzare le frasi nelle lingue diverse supportate dal gioco	-
*.ini	Tipo di file <i>plain-text</i> utilizzato per salvare i dati del gioco	-
*.glb	« <i>GLTF Binary</i> », file utilizzato per memorizzare i modelli 3D e le loro animazioni in formato binario, in modo da risparmiare spazio e migliorare le prestazioni	2.0.1

Tabella 9: Tipi di file utilizzati

Capitolo 3

Il progetto

In questo capitolo approfondisco tutti i processi del progetto: sviluppo, test e validazione. In pratica descrivo cosa ho fatto di preciso, e come l'ho svolto.

3.1 Analisi dei rischi

3.1.1 Rischi organizzativi

Errata pianificazione dei tempi	
Descrizione	Un'errata pianificazione dei tempi può portare a ritardi nello sviluppo del progetto, con conseguente rischio di non rispettare le scadenze stabilite.
Probabilità	Alta
Pericolosità	Alta
Rilevamento	Monitoraggio delle attività pianificate e dei tempi di esecuzione ogni settimana
Piano di contingenza	Controllare le attività svolte tramite uno strumento di gestione del progetto (ad esempio <i>diagrammi di Gannt</i> e uso di <i>checklist</i> su <i>Notion</i>) e rivedere la pianificazione delle attività in caso di ritardi.

Tabella 10: Errata pianificazione dei tempi

Impegni personali o universitari	
Descrizione	Impegni personali o universitari possono influenzare il tempo a disposizione per lo sviluppo del progetto, causando ritardi o interruzioni nello sviluppo.

Impegni personali o universitari	
Probabilità	Alta
Pericolosità	Media
Rilevamento	Monitoraggio delle attività pianificate e dei tempi di esecuzione ogni settimana
Piano di contingenza	Pianificare le attività in modo da tenere conto degli impegni personali o universitari, e rivedere la pianificazione delle attività in caso di imprevisti.

Tabella 11: Impegni personali o universitari

3.1.2 Rischi tecnici

Mancanza di competenze tecniche	
Descrizione	La mancanza di competenze tecniche può influenzare la qualità del prodotto software, causando ritardi nello sviluppo e problemi di integrazione
Probabilità	Media
Pericolosità	Alta
Rilevamento	Monitoraggio delle attività pianificate e dei tempi di esecuzione ogni settimana
Piano di contingenza	Formazione sulle tecnologie utilizzate e revisione della progettazione in caso di problemi tecnici

Tabella 12: Mancanza di competenze tecniche

Tecnologie non adeguate	
Descrizione	L'uso di tecnologie non adeguate può influenzare la qualità del prodotto <i>software</i> , causando problemi di prestazioni basse o <i>bug</i> .
Probabilità	Alta

Tecnologie non adeguate	
Pericolosità	Alta
Rilevamento	Monitoraggio delle attività svolte e dei tempi di esecuzione ogni settimana
Piano di contingenza	Valutazione delle tecnologie utilizzate e revisione della progettazione in caso di problemi tecnici. In caso di problemi con le tecnologie utilizzate, si valuterà la possibilità di modificare la progettazione del gioco per adattarsi alle tecnologie disponibili

Tabella 13: Tecnologie non adeguate

3.1.3 Rischi di analisi e progettazione

Cambio dei requisiti	
Descrizione	Un cambiamento dei requisiti può influenzare la progettazione del sistema, causando ritardi nello sviluppo
Probabilità	Bassa
Pericolosità	Alta
Rilevamento	Comunicazione frequente con il relatore del progetto per garantire che i requisiti siano chiari e stabili
Piano di contingenza	Rivedere la progettazione del sistema in caso di cambiamenti dei requisiti, e valutare l'impatto sui tempi di sviluppo. In caso di cambiamenti significativi dei requisiti, si valuterà la possibilità di modificare la pianificazione delle attività per tenere conto dei nuovi requisiti

Tabella 14: Cambio dei requisiti

Errore nella progettazione dell'architettura	
Descrizione	Un errore nella progettazione dell'architettura può influenzare la qualità del prodotto <i>software</i> ,

Errore nella progettazione dell'architettura	
	causando ritardi nello sviluppo e problemi di integrazione
Probabilità	Media
Pericolosità	Alta
Rilevamento	Monitoraggio delle attività svolte e dei tempi di esecuzione ogni settimana, valutazione della progettazione dell'architettura
Piano di contingenza	Rivedere la progettazione dell'architettura in caso di problemi, e valutare l'impatto sui tempi di sviluppo

Tabella 15: Errore nella progettazione dell'architettura

3.2 Analisi dei requisiti

3.2.1 Attori

Il gioco prevede un solo attore, il **giocatore**, cioè l'utente che interagisce con il videogioco, controllando il personaggio e prendendo decisioni durante il gioco.



Figura 6: Attore principale

Nei seguenti casi d'uso, l'attore principale sarà sempre il giocatore.

3.2.2 Casi d'uso

UC1 - Movimento

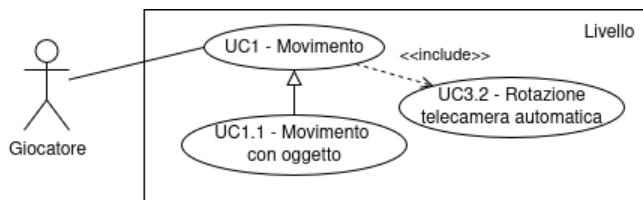


Figura 7: Diagramma UML *use case* sul movimento

Descrizione: il giocatore può muovere il personaggio in avanti, indietro, a sinistra e a destra utilizzando i tasti direzionali della tastiera o lo *stick* analogico sinistro del *joypad_G*.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: il personaggio si muove nella direzione desiderata e interagisce con l'ambiente circostante.

Inclusioni: - Rotazione telecamera automatica.

Generalizzazioni: - Movimento con oggetto.

UC1.1 - Movimento con oggetto

Descrizione: il giocatore può muovere il personaggio in avanti, indietro, a sinistra e a destra, mentre sta portando un oggetto, utilizzando i tasti direzionali della tastiera o lo *stick* analogico sinistro del *joypad*.

Precondizioni: il giocatore deve essere in un livello del gioco e ha un oggetto con sé.

Postcondizioni: il personaggio si muove nella direzione desiderata e sposta con sé l'oggetto.

UC2 - Salto

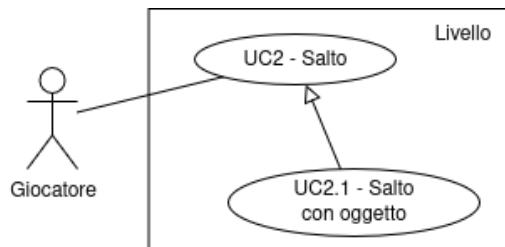


Figura 8: Diagramma UML *use case* sul salto

Descrizione: il giocatore può far saltare il personaggio utilizzando un tasto specifico.

Precondizioni: il giocatore deve essere in un livello del gioco e deve essere libero di muoversi.

Postcondizioni: il personaggio esegue il salto.

Generalizzazioni: - Salto con oggetto.

UC3 - Rotazione telecamera

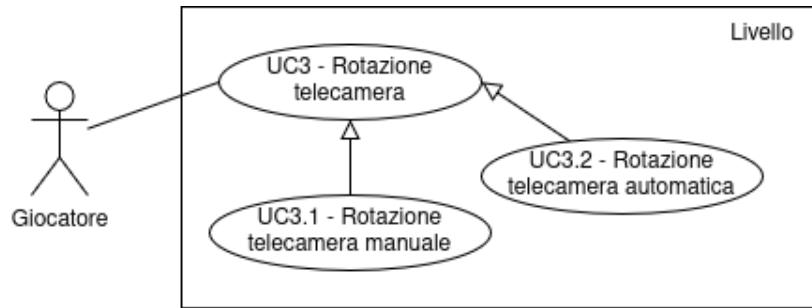


Figura 9: Rotazione telecamera

Descrizione: il giocatore può ruotare la telecamera attorno al personaggio utilizzando i comandi del mouse o del *joypad*.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: la telecamera si muove attorno al personaggio, cambiando l'angolo di visualizzazione.

Generalizzazioni:

- Rotazione telecamera manuale;
- Rotazione telecamera automatica.

UC3.1 - Rotazione telecamera manuale

Descrizione: il giocatore può ruotare la telecamera attorno al personaggio utilizzando i comandi della tastiera o del *joypad*.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: la telecamera si muove attorno al personaggio, cambiando l'angolo di visualizzazione.

UC3.2 - Rotazione telecamera automatica

Descrizione: la telecamera può modificare in autonomia la sua posizione e rotazione per inquadrare ciò che c'è davanti al personaggio quando questo si sta muovendo.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: la telecamera si muove attorno al personaggio, cambiando l'angolo di visualizzazione.

UC4 - Caduta

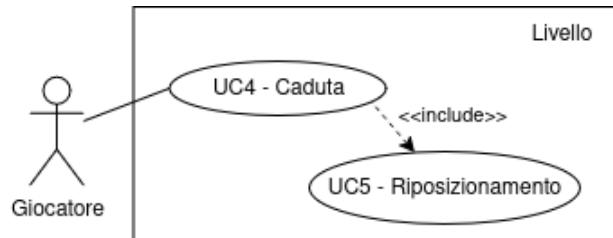


Figura 10: Caduta

Descrizione: il giocatore cade dal livello e viene riportato in una zona dove si trovava prima.

Precondizioni: il giocatore deve essere in un livello del gioco e deve entrare in un'area *di caduta*.

Postcondizioni: il giocatore torna dove si trovava, prima di cadere.

Inclusioni: - Riposizionamento.

UC5 - Riposizionamento

Descrizione: il giocatore viene posizionato in una certa zona del livello.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: il giocatore viene riposizionato.

UC6 - Raccolta *collezionabile*

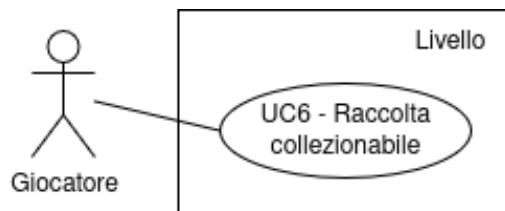


Figura 11: Raccolta collezionabile

Descrizione: il giocatore può raccogliere oggetti *collezionabili* presenti nel livello quando si avvicina.

Precondizioni: il giocatore deve essere in un livello del gioco e deve esserci un oggetto collezionabile nelle vicinanze.

Postcondizioni: il numero di collezionabili sale di un certo valore.

UC7 - Interazione con *entità*

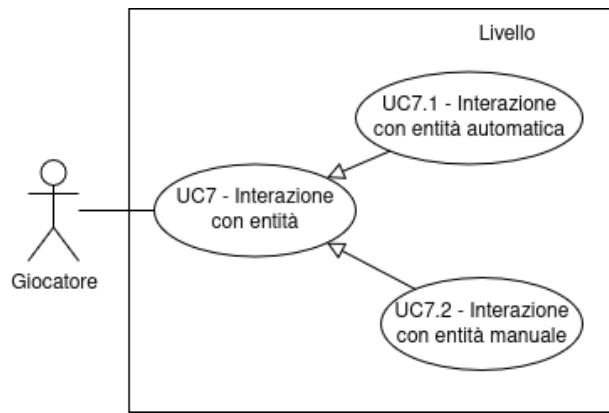


Figura 12: Interazione *_entità_*

Descrizione: il giocatore si avvicina a un'*entità* e vede un messaggio.

Precondizioni: il giocatore deve essere vicino ad un'*entità*.

Postcondizioni: l'*entità* mostra un messaggio.

Generalizzazioni:

- Interazione con un'*entità* automatica.
- Interazione con un'*entità* manuale.

UC7.1 - Interazione con *entità* automatica

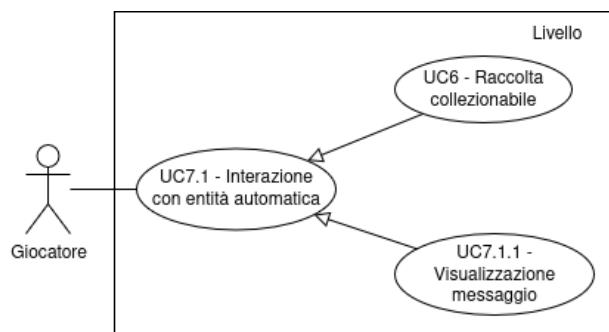


Figura 13: Interazione *_entità_* automatica

Descrizione: il giocatore si avvicina ad un'*entità* e l'interazione avviene automaticamente.

Precondizioni: il giocatore deve essere vicino ad un'*entità*.

Postcondizioni: il giocatore interagisce con l'*entità*.

Generalizzazioni:

- Raccolta collezionabile.
- Visualizzazione messaggio.

UC7.1.1 - Visualizzazione messaggio

Descrizione: il giocatore può visualizzare un messaggio automatico di un'*entità* del gioco.

Precondizioni: il giocatore deve essere vicino all'*entità*.

Postcondizioni: il giocatore visualizza il messaggio.

UC7.2 - Interazione con *entità* manuale

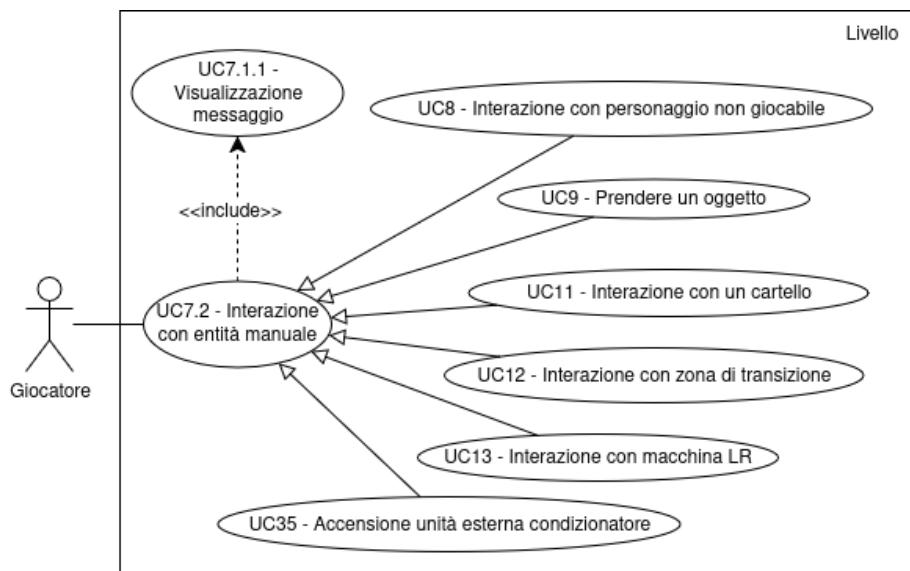


Figura 14: Interazione manuale

Descrizione: il giocatore si avvicina a un'*entità* e vede l'*input* che deve premere per interagire.

Precondizioni: il giocatore deve essere vicino all'*entità*.

Postcondizioni: il giocatore ha premuto l'*input* per interagire.

Inclusioni: Visualizzazione messaggio.

Generalizzazioni:

- Interazione con personaggio non giocabile.
- Prendere un oggetto.
- Interazione con un cartello.
- Interazione con zona di transizione.
- Interazione con macchina *LR*.
- Accensione unità esterna condizionatore.

UC8 - Interazione con personaggio non giocabile

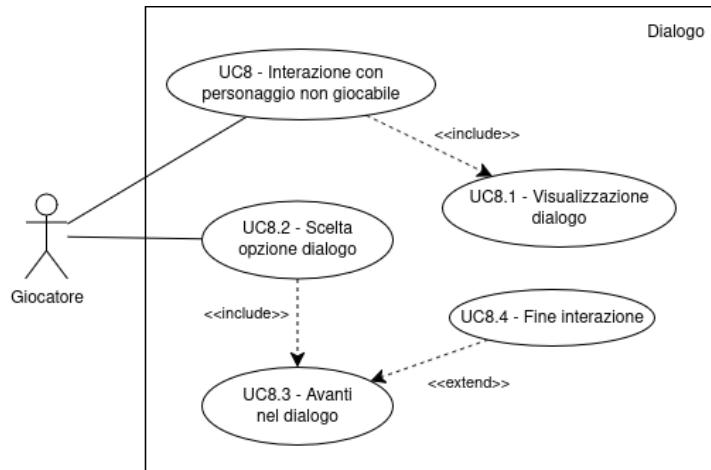


Figura 15: Interazione personaggio non giocabile

Descrizione: il giocatore si avvicina a un personaggio non giocabile e vede l'*input* che deve premere per interagire.

Precondizioni: il giocatore deve essere vicino a un personaggio non giocabile.

Postcondizioni: il giocatore ha premuto l'*input* per interagire.

Inclusioni: - Visualizzazione dialogo.

UC8.1 - Visualizzazione dialogo

Descrizione: il giocatore visualizza il dialogo mostrato dall'*entità*.

Precondizioni: il giocatore ha premuto l'*input* per interagire con l'*entità*.

Postcondizioni: l'*entità* mostra il dialogo, mentre il giocatore non può più più muoversi.

UC8.2 - Scelta opzione dialogo

Descrizione: il giocatore deve scegliere un'opzione per andare avanti nel dialogo.

Precondizioni: il giocatore deve essere in un dialogo e visualizza le opzioni tra cui scegliere.

Postcondizioni: il giocatore ha scelto l'opzione ed il dialogo finisce.

Inclusioni: - Avanti nel dialogo.

UC8.3 - Avanti nel dialogo

Descrizione: il giocatore vuole continuare il dialogo.

Precondizioni: il giocatore ha premuto l'*input* per andare avanti nel dialogo.

Postcondizioni: viene mostrato il messaggio successivo del dialogo.

Estensioni: - Fine interazione.

UC8.4 - Fine interazione

Descrizione: il giocatore vuole terminare l'interazione.

Precondizioni: il giocatore sta interagendo con un'*entità*.

Postcondizioni: il giocatore preme l'*input*, smette di interagire con l'*entità* e può muoversi di nuovo.

UC9 - Prendere un oggetto

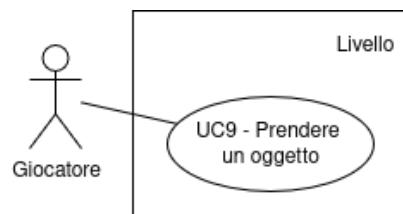


Figura 16: Prendere un oggetto

Descrizione: il giocatore può prendere un oggetto e poi muoversi con esso.

Precondizioni: il giocatore deve essere in un livello del gioco ,deve esserci un oggetto che può raccogliere davanti ad esso e non deve averne già uno.

Postcondizioni: il giocatore interagisce con l'oggetto.

UC10 - Lasciare un oggetto

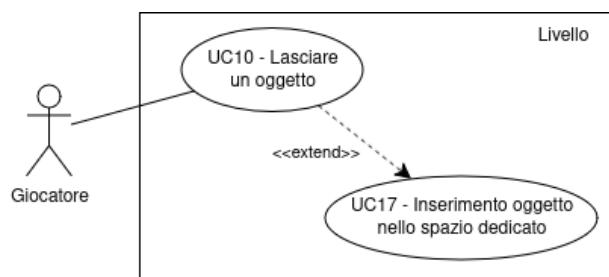


Figura 17: Lasciare un oggetto

Descrizione: il giocatore può lasciare un oggetto.

Precondizioni: il giocatore deve essere in un livello del gioco e sta portando un oggetto.

Postcondizioni: il giocatore lascia l'oggetto e questo rimane nella posizione dove viene lasciato.

Estensioni: - Inserimento oggetto nello spazio dedicato.

UC11 - Interazione con un cartello

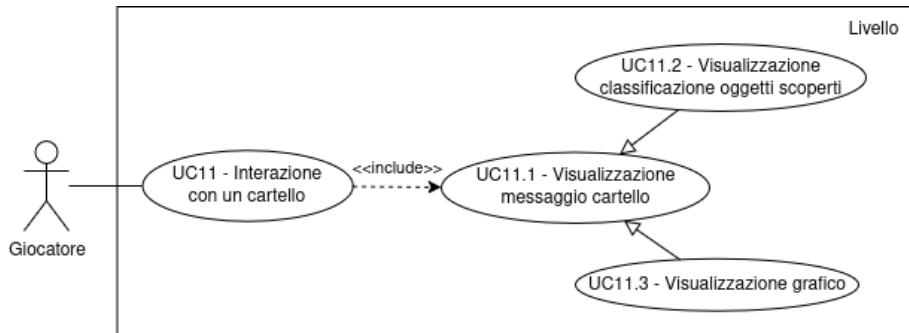


Figura 18: Interazione cartello

Descrizione: il giocatore si avvicina a un cartello e vede l'*input* che deve premere per interagire.

Precondizioni: il giocatore deve essere vicino a un cartello.

Postcondizioni: il giocatore ha premuto il tasto.

Inclusioni: - Visualizzazione messaggio cartello.

UC11.1 - Visualizzazione messaggio cartello

Descrizione: il giocatore visualizza il messaggio del cartello.

Precondizioni: il giocatore deve aver interagito con un cartello.

Postcondizioni: il giocatore visualizza il messaggio del cartello.

Generalizzazioni: - Visualizzazione classificazioni oggetti scoperti.
- Visualizzazione grafico.

UC11.2 - Visualizzazione classificazioni oggetti scoperti

Descrizione: il giocatore vuole visualizzare tutte le classificazioni degli oggetti che è riuscito ad indovinare nel livello.

Precondizioni: il giocatore è dentro il livello *Albero di decisione*.

Postcondizioni: il giocatore visualizza le classificazioni degli oggetti scoperti.

Estensioni: - Fine interazione.

UC11.3 - Visualizzazione grafico

Descrizione: il giocatore vuole visualizzare un grafico presente in un cartello.

Precondizioni: il giocatore deve avere interagito con un cartello.

Postcondizioni: il giocatore visualizza il grafico.

UC11.4 - Aggiornamento dato di un cartello

Descrizione: un dato di un cartello può essere modificato in base a determinate azioni del giocatore all'interno del livello.

Precondizioni: il giocatore ed il cartello devono essere nello stesso livello.
Postcondizioni: il dato del cartello viene aggiornato.

UC12 - Interazione con zona di transizione

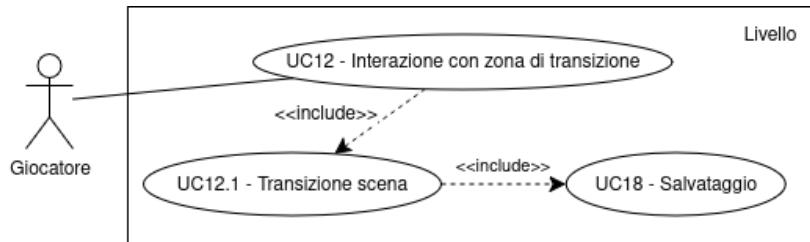


Figura 19: Interazione zona di transizione

Descrizione: il giocatore si avvicina a una zona di transizione per cambiare livello.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: il giocatore ha premuto il tasto.

UC12.1 - Transizione scena

Descrizione: il giocatore cambia livello.

Precondizioni: il giocatore deve aver premuto il tasto per interagire in un'area di transizione.

Postcondizioni: il giocatore è nel nuovo livello.

Inclusioni: - Salvataggio.

UC13 - Interazione con macchina LR

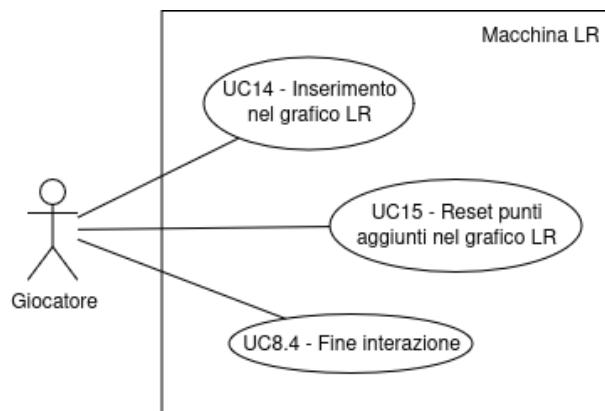


Figura 20: Interazione con macchina *LR*

Descrizione: il giocatore vuole interagire con la macchina per posizionare dei punti sul grafico *LR*.

Precondizioni: il giocatore deve trovarsi in un'area per interagire con la macchina.

Postcondizioni: il giocatore può usare la macchina.

UC14 - Inserimento punto nel grafico *LR*

Descrizione: il giocatore vuole posizionare un punto sul grafico *LR*.

Precondizioni: il giocatore deve essere in utilizzo di una macchina *LR*.

Postcondizioni: il punto viene posizionato sul grafico.

UC15 - Reset punti aggiunti nel grafico *LR*

Descrizione: il giocatore toglie i punti da lui aggiunti nel grafico premendo il tasto *Reset*.

Precondizioni: il giocatore deve essere in utilizzo di una macchina *LR*.

Postcondizioni: i punti aggiunti dal giocatore vengono tolti.

UC16 - Posizionamento sopra un nodo dell'albero di decisione

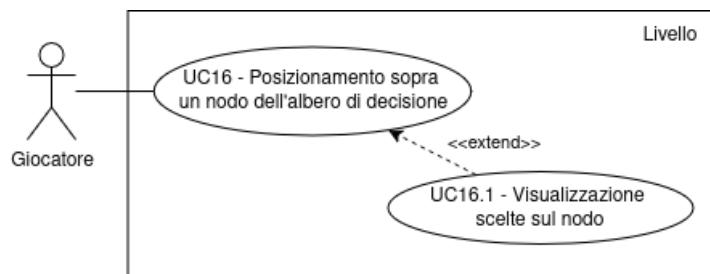


Figura 21: Scelte albero di decisione

Descrizione: il giocatore si posiziona sopra un nodo dell'albero di decisione.

Precondizioni: il giocatore deve essere in un livello del gioco e deve essere presente un *Albero di decisione*.

Postcondizioni: il giocatore è posizionato sopra un nodo dell'albero di decisione.

Estensioni: - Visualizzazione delle scelte sul nodo.

UC16.1 - Visualizzazione scelte sul nodo

Descrizione: il giocatore vuole proseguire nell'albero di decisione.

Precondizioni: il giocatore deve essere sopra un nodo dell'albero di decisione.

Postcondizioni: il giocatore sceglie la direzione in base alle scelte disponibili.

UC17 - Inserimento dell'oggetto nello spazio dedicato

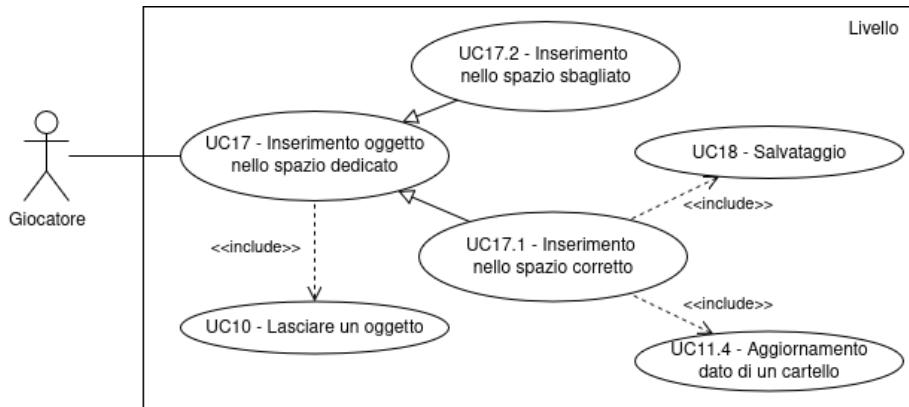


Figura 22: Inserimento dell'oggetto nello spazio dedicato

Descrizione: il giocatore posiziona l'oggetto che sta portando in uno spazio apposito.

Precondizioni: il giocatore deve portare un oggetto ed essere sopra un nodo finale dell'*albero*.

Postcondizioni: se l'oggetto è giusto, il nuovo oggetto viene mostrato nella visualizzazione delle classificazioni degli oggetti scoperti.

Generalizzazioni:

- Inserimento nello spazio corretto.
- Inserimento nello spazio sbagliato.

UC17.1 - Inserimento nello spazio corretto

Descrizione: il giocatore posiziona l'oggetto nello spazio corretto.

Precondizioni: il giocatore deve portare un oggetto.

Postcondizioni: l'oggetto viene mostrato nella visualizzazione delle classificazioni degli oggetti scoperti.

Inclusioni:

- Salvataggio.

UC17.2 - Inserimento nello spazio sbagliato

Descrizione: il giocatore posiziona l'oggetto nello spazio sbagliato.

Precondizioni: il giocatore deve portare un oggetto.

Postcondizioni: l'oggetto non viene mostrato nella visualizzazione delle classificazioni degli oggetti scoperti.

UC18 - Salvataggio

Descrizione: il gioco salva in automatico in momenti specifici.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: la partita viene salvata e il giocatore può riprendere da quel punto in un secondo momento.

UC18 - Salvataggio

Descrizione: il gioco salva in automatico in momenti specifici.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: la partita viene salvata e il giocatore può riprendere da quel punto in un secondo momento.

UC19 - Pausa

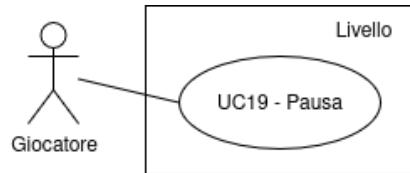


Figura 23: Apertura menu di pausa

Descrizione: il giocatore può mettere in pausa il gioco in qualsiasi momento per accedere al menu di pausa.

Precondizioni: il giocatore deve essere in un livello del gioco.

Postcondizioni: il gioco si interrompe e viene visualizzato il menu di pausa.

UC20 - Riprendi

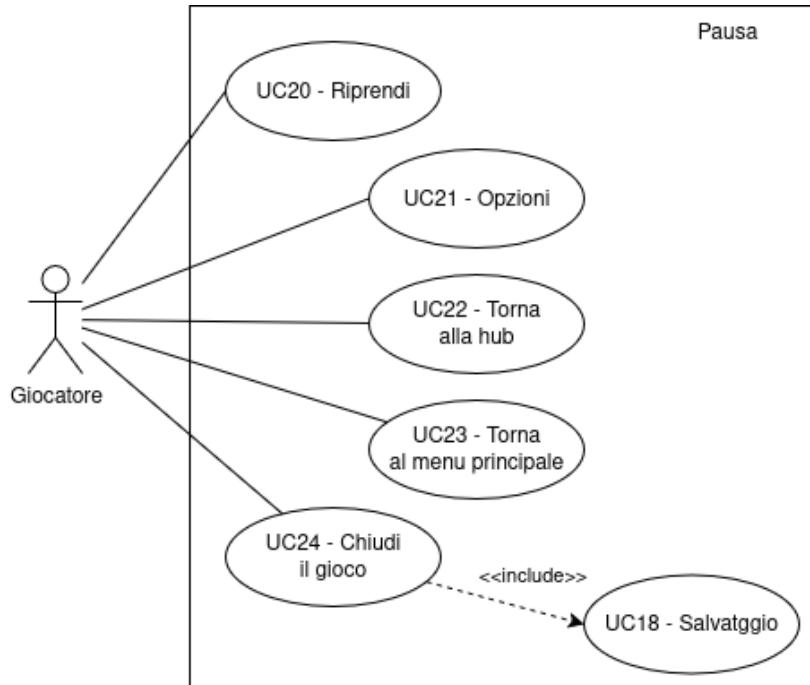


Figura 24: Menu di pausa

Descrizione: il giocatore può riprendere il gioco dal menu di pausa.

Precondizioni: il gioco deve essere in pausa e il menu di pausa deve essere visualizzato.

Postcondizioni: il gioco riprende dalla posizione in cui era stato interrotto.

UC21 - Opzioni

Descrizione: il giocatore può accedere alle opzioni del gioco dal menu di pausa per modificare le impostazioni come volume, risoluzione, modalità finestra o schermo intero e lingua.

Precondizioni: il gioco deve essere in pausa e il menu di pausa deve essere visualizzato.

Postcondizioni: il giocatore può modificare le impostazioni del gioco.

UC22 - Torna alla *hub*

Descrizione: il giocatore può ritornare al livello «*hub*» da qualsiasi altro livello.

Precondizioni: il gioco deve essere in pausa, il menu di pausa deve essere visualizzato ed il giocatore non deve essere già nel livello *hub*.

Postcondizioni: il giocatore torna al livello *hub*.

UC23 - Torna al menu principale

Descrizione: il giocatore può tornare al menu principale dal menu di pausa.

Precondizioni: il gioco deve essere in pausa e il menu di pausa deve essere visualizzato.

Postcondizioni: il gioco torna al menu principale e il giocatore può scegliere di avviare una nuova partita o caricare una partita salvata.

UC24 - Chiudi il gioco

Descrizione: il giocatore può chiudere il gioco premendo l'apposito tasto.

Precondizioni: il gioco deve essere in pausa o nel menu principale.

Postcondizioni: il gioco viene chiuso.

UC25 - Carica partita

Descrizione: il giocatore può caricare una partita salvata dal menu principale.

Precondizioni: il gioco deve essere avviato con il menu principale visualizzato e deve presentare dei dati di salvataggio esistenti.

Postcondizioni: il gioco carica la partita con i valori salvati ed il giocatore viene portato nel livello *hub*.

UC26 - Nuova partita

Descrizione: il giocatore può avviare il gioco dal menu principale.

Precondizioni: il gioco deve essere avviato con il menu principale visualizzato

e non deve presentare dei dati di salvataggio esistenti.

Postcondizioni: il gioco viene resettato ed il giocatore viene portato al livello base.

UC27 - Modifica modalità finestra

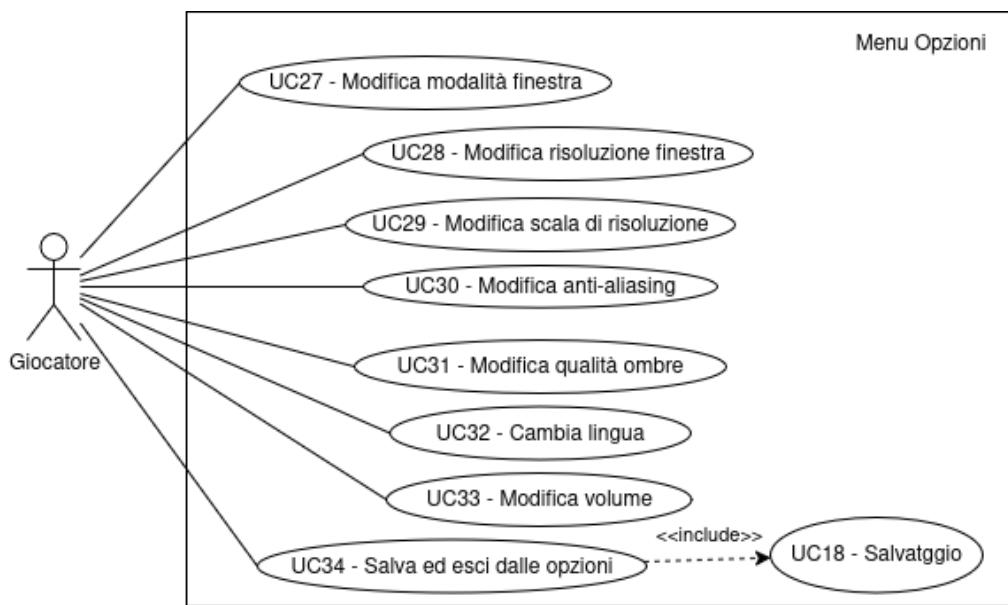


Figura 25: Menu opzioni

Descrizione: il giocatore può modificare la modalità di visualizzazione del gioco (finestra o schermo intero) dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: la modalità di visualizzazione del gioco viene modificata in base alle preferenze del giocatore.

UC28 - Modifica risoluzione finestra

Descrizione: il giocatore può modificare la risoluzione del gioco dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: la risoluzione del gioco viene modificata in base alle preferenze del giocatore.

UC29 - Modifica scala di risoluzione

Descrizione: il giocatore può modificare la scala di risoluzione dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere

visualizzato.

Postcondizioni: la scala di risoluzione viene modificata in base alle preferenze del giocatore.

UC30 - Modifica *anti-aliasing*

Descrizione: il giocatore può modificare la modalità di *anti-aliasing* dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: la modalità dell'*anti-aliasing* viene modificata in base alle preferenze del giocatore.

UC31 - Modifica qualità ombre

Descrizione: il giocatore può modificare la qualità delle ombre dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: la qualità delle ombre viene modificata in base alle preferenze del giocatore.

UC32 - Cambia lingua

Descrizione: il giocatore può modificare la lingua del gioco dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: la lingua del gioco viene modificata in base alle preferenze del giocatore.

UC33 - Modifica volume

Descrizione: il giocatore può modificare il volume del gioco dal menu delle opzioni.

Precondizioni: il gioco deve essere in pausa e il menu delle opzioni deve essere visualizzato.

Postcondizioni: il volume del gioco viene modificato in base alle preferenze del giocatore.

UC34 - Salva ed esci dalle opzioni

Descrizione: il giocatore può salvare le opzioni scelte.

Precondizioni: il giocatore deve essere nel menu di opzioni.

Postcondizioni: il gioco salva ed applica le opzioni.

Inclusioni: - Salvataggio.

UC35 - Accensione unità esterna condizionatore

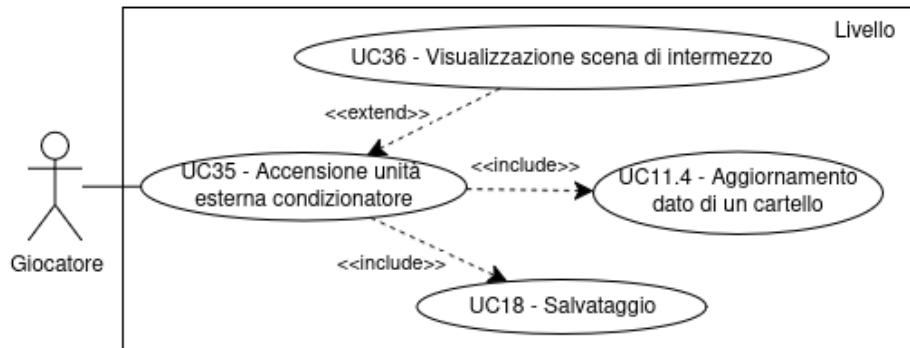


Figura 26: Accensione unità esterna condizionatore

Descrizione: il giocatore accende un'unità esterna di un condizionatore.

Precondizioni: il giocatore deve essere in un livello.

Postcondizioni: l'unità esterna del condizionatore viene accesa ed il giocatore non può più interagire con l'*entità*.

Inclusioni: - Aggiornamento dato di un cartello.
- Salvataggio.

Estensioni: - Visualizzazione scena di intermezzo.

UC36 - Visualizzazione scena di intermezzo

Descrizione: il giocatore visualizza una scena di intermezzo.

Precondizioni: il giocatore deve soddisfare certe condizioni.

Postcondizioni: il giocatore visualizza la scena di intermezzo.

UC37 - Visualizzazione *input* tastiera

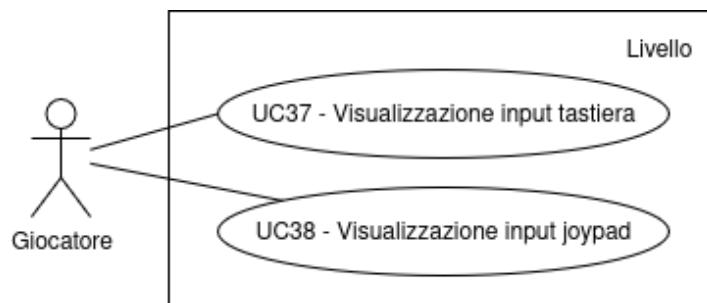


Figura 27: Visualizzazione *input* nella *UI*

Descrizione: il giocatore deve vedere che *input* deve premere dal dispositivo che sta utilizzando.

Precondizioni: il giocatore deve avere la tastiera collegata alla macchina.

Postcondizioni: il giocatore visualizza nella *UI_G* il tasto della tastiera da premere.

UC38 - Visualizzazione *input joypad*

Descrizione: il giocatore deve vedere che *input* deve premere del dispositivo che sta utilizzando dalla *UI* del gioco.

Precondizioni: il giocatore deve soddisfare certe condizioni e deve avere un *joypad* collegato alla macchina.

Postcondizioni: il giocatore visualizza nella *UI* il tasto del *joypad* da premere.

3.2.3. Requisiti

In questa sezione elenco tutti i requisiti del progetto, individuati durante la fase di analisi. Ogni requisito viene identificato da un codice, scelto in base ai seguenti parametri:

R - [numero] - [tipo] - [priorità]

con:

- **Numero:** numero progressivo che identifica il requisito, parte da 01.
- **Tipo:** può essere
 - **F:** requisito funzionale, indica una funzionalità del sistema;
 - **Q:** requisito di qualità, definisce le caratteristiche della qualità del prodotto, come un sistema deve essere o come il sistema deve esibirsi;
 - **A:** requisito di accessibilità, indica una funzionalità da soddisfare per rendere il gioco accessibile ad un numero maggiore di utenti.
- **Priorità:** può essere
 - **O:** Obbligatorio e necessario per considerare il prodotto completo;
 - **D:** Desiderabile, non strettamente necessario ma è un valore aggiunto.

3.2.4. Lista dei requisiti

ID	Descrizione	Fonte
R-01-F-O	Il giocatore deve essere in grado di muoversi in uno spazio tridimensionale	UC1
R-02-F-O	Il giocatore deve essere in grado di muoversi in uno spazio tridimensionale portando un oggetto	UC1.1
R-03-F-O	Il giocatore deve essere in grado saltare	UC2
R-04-F-O	Il giocatore deve essere in grado di saltare con un oggetto in mano	UC2.1
R-05-F-O	La telecamera deve muoversi automaticamente dietro il giocatore quando questo si muove	UC3 UC3.1
R-06-F-O	Il giocatore deve essere in grado di ruotare la telecamera intorno al personaggio	UC3 UC3.2

ID	Descrizione	Fonte
R-07-F-O	Il giocatore deve tornare in una posizione precedente quando cade dal livello	UC4 UC5
R-08-F-O	Il giocatore deve essere in grado di raccogliere <i>collezionabili</i> sparsi per il livello	UC6
R-09-F-O	Il giocatore deve essere in grado di interagire con <i>entità</i> presenti nel livello	UC7
R-10-F-O	Il giocatore deve essere in grado di visualizzare subito il messaggio di un' <i>entità</i> automatica	UC7.1 UC7.1.1
R-11-F-O	Il giocatore deve essere in grado di parlare con un personaggio non giocabile	UC7.2
R-12-F-O	Il giocatore deve poter visualizzare il dialogo quando interagisce con delle <i>entità</i> specifiche	UC8.1
R-13-F-O	Il giocatore deve avere la possibilità di prendere una decisione quando gli viene mostrato nel dialogo	UC8.2
R-14-F-O	Il giocatore deve poter andare avanti nel dialogo	UC8.3
R-15-F-O	Il giocatore deve poter finire l'interazione con un' <i>entità</i>	UC8.4
R-16-F-O	Il giocatore deve poter essere in grado di raccogliere oggetti	UC9
R-17-F-O	Il giocatore deve poter essere in grado di lasciare l'oggetto che sta portando	UC10
R-18-F-O	Il giocatore deve poter interagire con un cartello in un livello	UC11
R-19-F-O	Il giocatore deve poter visualizzare il contenuto di un cartello	UC11.1
R-20-F-O	Il giocatore deve poter visualizzare gli oggetti che ha classificato correttamente	UC11.2
R-21-F-O	Il giocatore deve poter visualizzare il grafico di un cartello	UC11.3
R-22-F-O	Il giocatore deve poter modificare i dati presenti in un cartello	UC11.4

ID	Descrizione	Fonte
R-23-F-O	Il giocatore deve poter essere in grado di interagire con un'area di transizione	UC12
R-24-F-O	Il giocatore deve poter essere in grado di cambiare livello	UC12.1
R-25-F-O	Il giocatore deve poter interagire con la macchina <i>LR</i>	UC13
R-26-F-O	Il giocatore deve poter inserire punti nel grafico <i>LR</i>	UC14
R-27-F-O	Il giocatore deve poter resettare i punti aggiunti nel grafico <i>LR</i>	UC15
R-28-F-O	Il giocatore deve poter salire sopra un nodo dell'albero di decisione	UC16
R-29-F-O	Il giocatore deve poter visualizzare le scelte da prendere sopra il nodo	UC16.1
R-30-F-O	Il giocatore deve poter piazzare un oggetto sopra un nodo dell'albero di decisione	UC17
R-31-F-O	Il giocatore deve poter visualizzare se l'oggetto posto sul nodo sia giusto	UC17.1
R-32-F-O	Il giocatore deve poter visualizzare se l'oggetto posto sul nodo sia sbagliato	UC17.2
R-33-F-O	Il giocatore deve poter salvare il gioco in momenti specifici	UC18
R-34-F-O	Il giocatore deve poter mettere in pausa il gioco	UC19
R-35-F-O	Il giocatore deve poter riprendere il gioco dal menu di pausa	UC20
R-36-F-O	Il giocatore deve poter accedere alle opzioni del gioco	UC21
R-37-F-O	Il giocatore deve poter tornare al livello <i>hub</i> dal menu di pausa	UC22
R-38-F-D	Il giocatore deve poter tornare al menu principale dal menu di pausa	UC23

ID	Descrizione	Fonte
R-39-F-O	Il giocatore deve poter chiudere il gioco dal menu di pausa o principale	UC24
R-40-F-O	Il giocatore deve poter caricare una partita salvata dal menu principale	UC25
R-41-F-O	Il giocatore deve poter avviare una nuova partita dal menu principale	UC26
R-42-F-O	Il giocatore deve poter modificare la modalità della finestra dal menu delle opzioni	UC27
R-43-F-O	Il giocatore deve poter modificare la risoluzione della finestra	UC28
R-44-F-D	Il giocatore deve poter modificare la scala di risoluzione del gioco	UC29
R-45-F-D	Il giocatore deve essere in grado di poter modificare il tipo di anti-aliasing usato nel gioco, oppure non usarlo	UC30
R-46-F-D	Il giocatore deve essere in grado di modificare la qualità delle ombre nel gioco	UC31
R-47-F-D	Il giocatore deve poter cambiare lingua di gioco	UC32
R-48-F-D	Il giocatore deve poter cambiare il volume generale del gioco	UC33
R-49-F-O	Il gioco deve applicare e salvare le opzioni selezionate	UC34
R-50-F-O	Il giocatore deve poter accendere delle unità esterne di un condizionatore premendo un tasto	UC35
R-51-F-O	Il giocatore deve poter vedere scene di intermezzo	UC36
R-52-F-O	Il gioco deve supportare <i>input</i> da tastiera	UC37
R-53-F-D	Il gioco deve supportare <i>input</i> da un <i>joypad</i> generico	UC38

Tabella 16: Tabella dei requisiti funzionali

ID Requisito	Descrizione
R-01-Q-O	È richiesta la presentazione del documento Specifica Tecnica che include dettagli riguardanti la progettazione architettonale
R-02-Q-O	È richiesta la presentazione del documento Specifica Tecnica che include dettagli riguardanti le tecnologie utilizzate
R-03-Q-O	Tutte le attività del progetto devono essere svolte rispettando le Norme di Progetto
R-04-Q-O	Tutto il codice e la documentazione vanno salvati all'interno di un repository pubblico

Tabella 17: Tabella dei requisiti di qualità

ID Requisito	Descrizione
R-01-A-O	Il gioco deve supportare il sistema operativo Windows
R-02-A-D	Il gioco deve supportare il sistema operativo Linux
R-03-A-D	Il gioco deve supportare il sistema operativo Mac-OS
R-04-A-D	La piattaforma deve essere responsive e funzionare correttamente su dispositivi desktop con risoluzione minima di 640×360px
R-05-A-O	Il gioco deve mostrare gli <u>input</u> del dispositivo che si sta usando

Tabella 18: Tabella dei requisiti di accessibilità

3.3. Architettura

3.3.1. Concetti chiave di *Godot*

3.3.1.1. Nodi

I nodi sono i blocchi fondamentali del gioco. Sono come ingredienti in una ricetta. Ci sono dozzine di tipi che possono mostrare un'immagine, riprodurre un suono, rappresentare una camera, e molto altro. Tutti i nodi hanno le seguenti caratteristiche:

- un nome;
- proprietà modificabili;

- ricevono *callback* per aggiornarsi ad ogni *frame*;
- si possono estendere con nuove proprietà e funzioni;
- si possono aggiungere a un altro nodo come figlio. [1]

Inoltre ad ogni nodo si può assegnare uno script, che estende il tipo di quel nodo e aggiunge nuove funzionalità.

I principali tipi di nodi che vengono utilizzati in questo progetto sono:

- **Node**: nodo base da cui vengono estesi tutti gli altri nodi, in questo progetto viene usato principalmente per assegnare classi e inserirle come figlie in altri nodi.
- **Node3D**: rappresenta un oggetto nello spazio tridimensionale.
 - **CharacterBody3D**: rappresenta un personaggio che si può muovere nel gioco, gestendo la sua posizione e interazioni.
 - **Camera3D**: rappresenta una telecamera nello spazio tridimensionale, che può essere utilizzata per visualizzare la scena.
 - **MeshInstance3D**: rappresenta un oggetto tridimensionale.
 - **CollisionShape3D**: rappresenta una forma di collisione nello spazio tridimensionale, utilizzata per gestire le interazioni fisiche tra gli oggetti.
 - **Area3D**: rappresenta un'area nello spazio tridimensionale, utilizzata per gestire le interazioni tra gli oggetti che entrano ed escono da essa.
- **AnimationPlayer**: gestisce le animazioni degli oggetti nella scena, permettendo di riprodurre animazioni sul modello 3D, telecamere e altri nodi.
- **Control**: rappresenta un nodo *UI*, utilizzato per gestire gli elementi dell'interfaccia utente del gioco.

3.3.1.2. Scene

Quando si organizzano nodi in un albero, come il nostro personaggio, possiamo chiamare questa formazione una scena. Una volta salvata, la scena si presenta come un nuovo nodo nell'*editor*, dove possiamo aggiungerlo come figlio di un nodo esistente. In questo caso, l'istanza della scena appare come nodo singolo con interni nascosti. Le scene di consentono di strutturare il codice del gioco in qualunque modo si voglia. Si possono comporre nodi per creare nodi personalizzati e complessi, come un personaggio di gioco che si muove e salta, una barra della vita, una cesta con cui puoi interagire, e molto altro. [1]

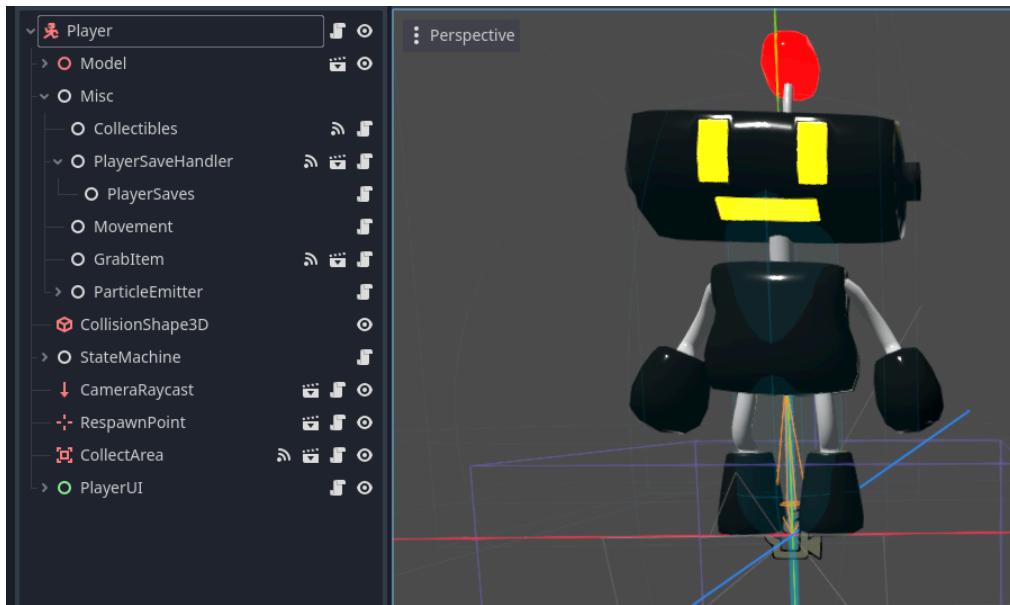


Figura 28: *Scena del personaggio del giocatore*

Oltre che a comportarsi come nodi, le scene hanno anche le seguenti caratteristiche:

- Hanno sempre un nodo **owner**, come il **Player** nel nostro esempio.
- Si possono salvare sul disco locale e caricarle in seguito.
- Si possono creare quante più istanze di una scena si desideri. Ad esempio, si possono avere cinque o dieci personaggi nel gioco, creati da una determinata scena.

3.3.1.3. Segnali

I segnali sono un modo per far comunicare i nodi in maniera asincrona in Godot. Ogni classe presenta dei segnali preimpostati ed emessi in determinati momenti, ad esempio quando un nodo viene caricato, questo emette il segnale **ready()**, oppure quando un bottone viene premuto, viene emesso il segnale **pressed()**. I segnali inoltre possono anche contenere dei parametri, che possono essere utilizzati per passare informazioni tra i nodi.

Infine, si possono creare segnali personalizzati, che possono essere emessi in qualsiasi momento dal nodo che li ha definiti tramite il metodo **signal.emit(...)**.

Ci sono due modi per collegare un segnale ad un altro nodo:

- tramite l'*editor*: selezionando il nodo che emette il segnale e trascinandolo sul nodo che deve ricevere il segnale, e selezionando il metodo che deve essere chiamato quando il segnale viene emesso;
- tramite codice: utilizzando il metodo **signal.connect(callable: Callable)** del nodo che emette il segnale, passando come parametro il nome del metodo che deve essere chiamato quando il segnale viene emesso.

3.3.2. Funzioni comuni

Molte classi del progetto presentano delle funzioni virtuali comuni, che vengono fornite dalle classi base del motore di gioco:

+ready(): void

Questa funzione viene chiamata quando il nodo entra nella scena, ovvero quando tutti i nodi figli sono stati caricati e il nodo è pronto per essere utilizzato.

In questa funzione è possibile inizializzare le variabili, collegare i segnali e impostare le proprietà del nodo.

È importante notare che questa funzione viene chiamata solo una volta, quando il nodo viene caricato per la prima volta nella scena, e non ad ogni *frame* del gioco.

+process(delta: float): void

Questa funzione viene chiamata ad ogni *frame* del gioco e permette di aggiornare lo stato della classe, ad ogni *frame* di inattività.

Il parametro **delta** rappresenta il tempo trascorso dall'ultimo *frame* di inattività, ed è utile per gestire le animazioni e le interazioni in modo fluido e coerente. [2]

+physics_process(delta: float): void

Questa funzione viene chiamata ad ogni *frame* di fisica del gioco, che di default è fisso 60 volte al secondo, anche nel caso il numero di *frame* al secondo del gioco sia inferiore o superiore.

Questa funzione è utile per gestire le interazioni fisiche tra gli oggetti, come ad esempio la gestione delle collisioni e la gestione della gravità.

Il parametro **delta** rappresenta il tempo trascorso dall'ultimo *frame* di fisica. [2]

+_input_(event: InputEvent): void

Funzione chiamata ogni volta che il gioco rileva un qualsiasi *input*, sia da tastiera che dal *joypad*. Il parametro *event* rappresenta l'*input* che chiama la funzione.

3.3.3. Classi del giocatore

3.3.3.1. Player

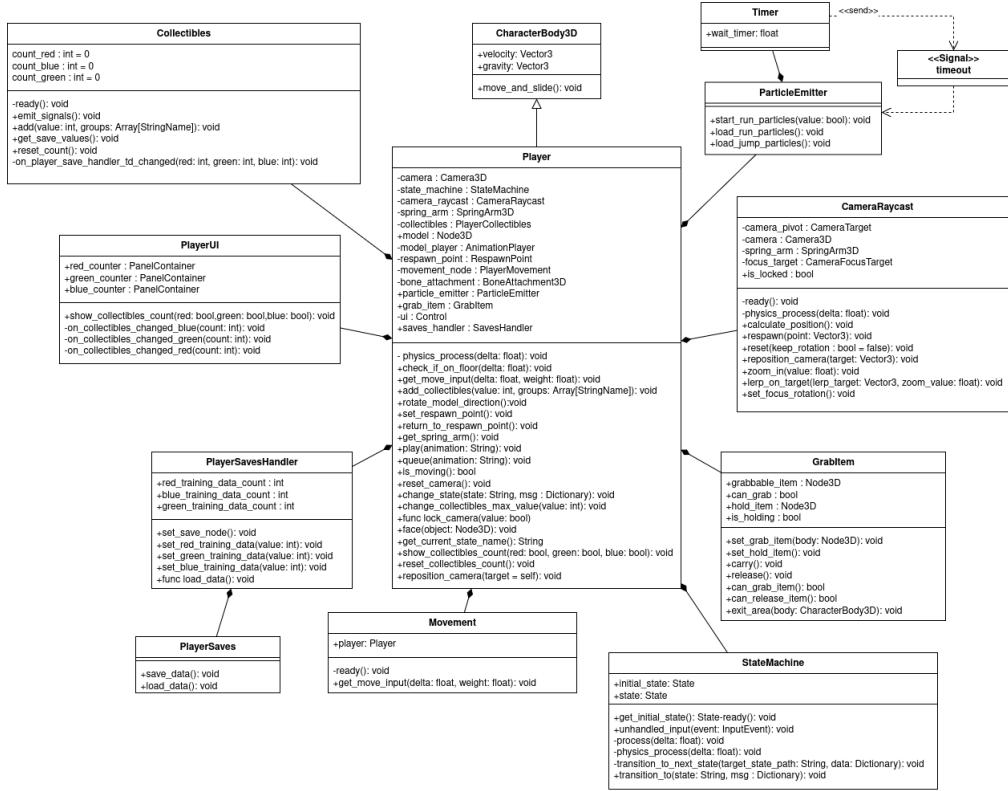


Figura 29: Diagramma UML delle classi del giocatore

Il giocatore può essere considerata la classe principale di tutta l'applicazione, attraverso il quale l'utente può interagire con la maggior parte dell'applicazione. Nonostante ci sia solo un giocatore presente nel gioco, questo non è un *singletone*, poiché per implementare un *singletone* in Godot è richiesto che questo sia caricato come *autoload* in ogni scena, e non c'è motivo di caricare il giocatore nel menu principale, all'avvio del gioco.

Molte variabili presenti nel giocatore sono riferimenti ai suoi nodi figli presenti nella scena, queste variabili sono precedute dalla parola chiave `@onready` nel codice. Similmente, molte funzioni della classe servono solo per accedere alle variabili dei suoi nodi figli.

La classe del giocatore ha associate delle classi che offrono funzionalità diverse, di seguito vengono descritte in dettaglio quelle più importanti.

3.3.3.2. CameraRayCast

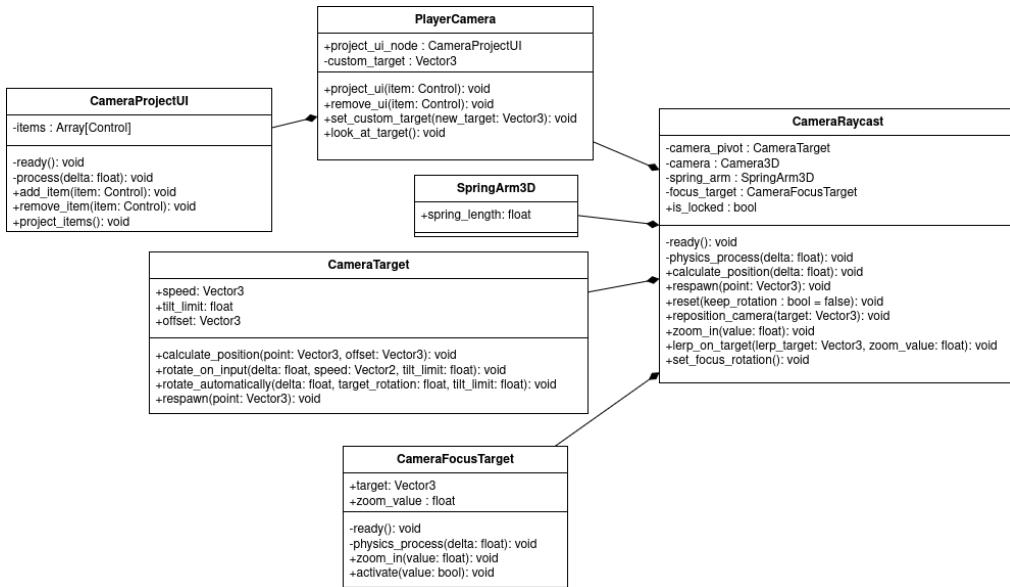


Figura 30: Diagramma UML delle classi della telecamera del giocatore

La telecamera del giocatore viene gestita da più classi per garantire diverse funzionalità tra quali la rotazione intorno al giocatore, proiezione di elementi della *UI* sullo schermo ed evitare che la telecamera passi attraverso i muri, generando il fenomeno chiamato *clipping*.

Oltre che a gestire il lavoro di tutte le altre classi per il corretto funzionamento della telecamera, la classe **CameraRayCast** lancia un raggio dalla posizione del giocatore verso il basso per controllare velocemente la distanza dal terreno del giocatore. Nel caso il raggio tocca ancora il terreno, la telecamera rimane per terra e non si sposta in alto con il giocatore. Di seguito vengono descritte le classi principali associate alla classe **CameraRayCast**:

- **PlayerCamera**: la telecamera effettiva, eredita dalla classe di *Godot* **Camera3D**. Offre il metodo **look_at_target()** che si occupa di girare la telecamera verso un obiettivo, specificato dalla variabile **custom_target** nella classe. Questo metodo viene usato da **CameraRayCast** per girare la telecamera verso un punto calcolata da quest'ultima.
- **CameraProjectUI**: gestisce gli elementi della *UI* la cui posizione viene proiettata dallo spazio 3D del gioco, allo spazio 2D dello schermo. Contiene un array, composto da questi elementi. Nel caso l'array sia vuoto, la modalità di processo viene disabilitata, cioè le operazioni della classe non vengono più effettuate ad ogni fotogramma del gioco, risparmiando risorse.
- **SpringArm3D**: classe fornita da *Godot*. La sua posizione globale corrisponde sempre a quella del giocatore, e si occupa di avvicinare la telecamera quando è vicino ad un muro per evitare il *clipping*.

- **CameraTarget**: eredita dalla classe di *Godot Marker3D*. La sua posizione viene calcolata da **CameraRayCast** e si occupa di gestire gli input per ruotare e muovere la telecamera intorno al giocatore.

3.3.3.3. StateMachine

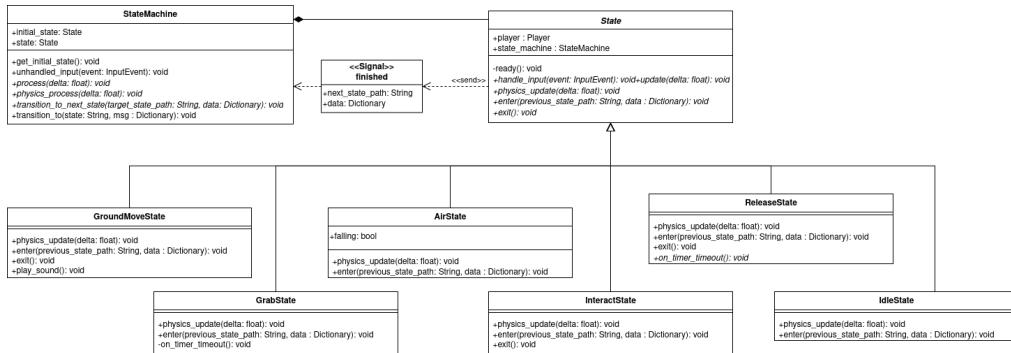


Figura 31: Diagramma UML sulla struttura della macchina di stati

La macchina di stati è stata utilizzata per controllare meglio i diversi stati in cui il personaggio del giocatore può trovarsi, ad esempio: il movimento, salto... L'uso della macchina di stati, inoltre, ha garantito una gestione più semplice del personaggio del giocatore e ha reso più facile aggiungere o modificare funzionalità a questo.

La classe **StateMachine** si occupa di gestire la transizione degli stati.

L'attributo **state** indica lo stato corrente del personaggio del giocatore. Quando riceve il segnale **finished(...)** dallo stato in cui si trova, si occupa di passare allo stato indicato dal segnale, passando gli eventuali dati contenuti nel **Dictionary** allo stato successivo.

Tutti gli stati ereditano dalla classe base astratta **State**. Questa include un riferimento al giocatore ed alla macchina di stati.

- **IdleState**: stato iniziale del giocatore. Questo stato viene chiamato quando il giocatore è fermo per terra. Può passare a tutti gli altri stati in base agli input premuti in diverse condizioni. Ad esempio se il giocatore sta portando qualcosa e preme il tasto di interazione, il personaggio passa allo stato *Release*, ma se non sta portando niente, allora non succede niente. Invece se preme lo stesso tasto dentro un area specifica, il personaggio passa allo stato *Interact*.
- **GroundMovementState**: il personaggio del giocatore passa allo stato *Ground-Movement* quando viene premuto un input per spostarsi rimanendo per terra. Come lo stato *Idle*, si può passare a tutti gli altri stati anche da questo, seguendo le stesse condizioni dello stato *Idle*.
- **AirState**: si può passare a questo stato in due condizioni: il giocatore cade da una piattaforma, o preme il tasto per saltare. Nell'ultimo caso, lo stato precedente manda un valore **jump = true** all'interno del **Dictionary**, in

questo modo lo stato controlla se è presente il medesimo valore ed in caso positivo, esegue il salto, caricando la rispettiva animazione e modificando la velocità verticale.

- **InteractState:** lo stato *Interact* indica che il personaggio del giocatore è impegnato ad interagire con un'altra entità, ad esempio mentre parla con un personaggio non giocabile o legge un cartello. A differenza degli altri stati, non è un input a far cambiare stato, ma i segnali dalle entità esterne.
- **GrabState:** il personaggio del giocatore passa allo stato *Grab* quando il giocatore preme il tasto per prendere un oggetto vicino a uno di questi. Importante notare che questo stato rappresenta solo quando il personaggio prende un oggetto, dopo aver svolto l'azione, il personaggio torna allo stato *Idle*, cambiando le animazioni in modo che rispecchino la situazione.
- **ReleaseState:** quando il giocatore preme di nuovo il tasto per prendere un oggetto mentre il personaggio sta portando un oggetto, questo passa allo stato *Release* e lascia l'oggetto. Come il suo stato opposto, una volta lasciato l'oggetto, il giocatore torna allo stato *Idle*.

La figura a pagina successiva mostra il flusso degli stati.

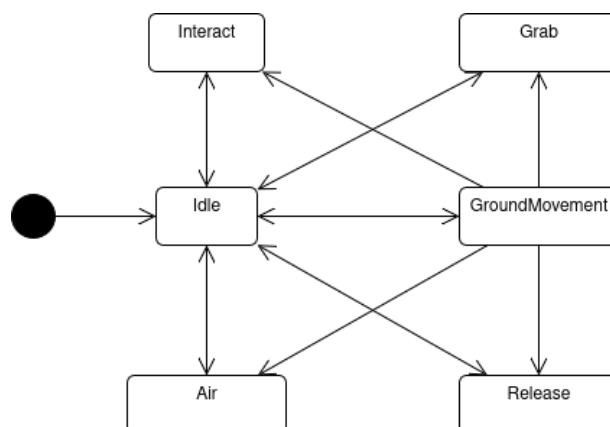


Figura 32: Diagramma sul flusso degli stati del giocatore

3.3.4. Entità interagibili

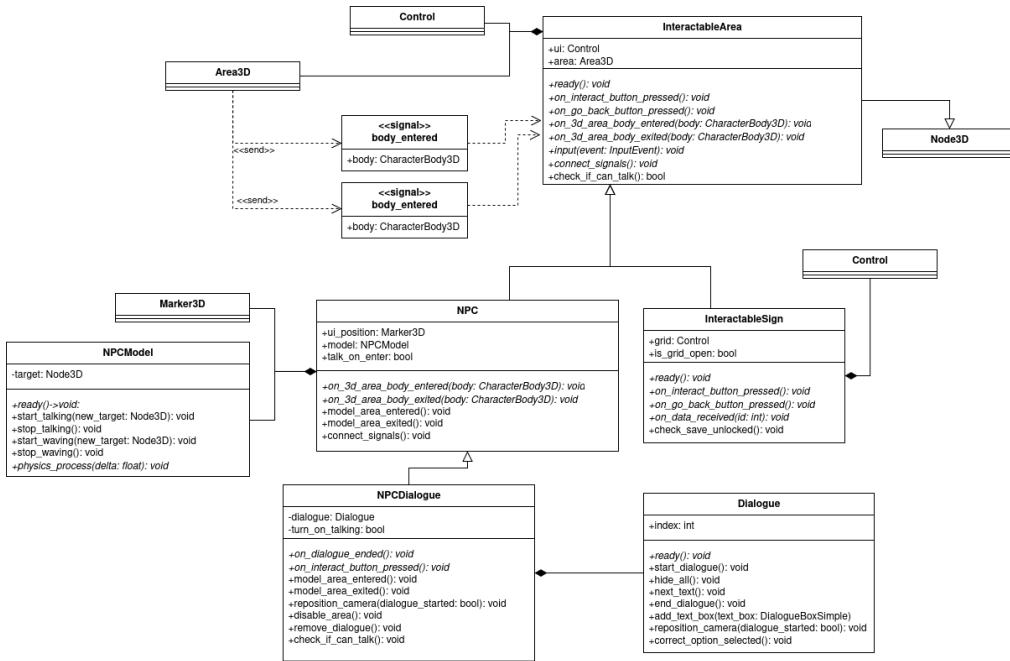


Figura 33: Diagramma UML degli oggetti con cui il giocatore può interagire

Nei livelli sono presenti diverse *entità* con cui il giocatore può interagire. Di seguito vengono descritte i diversi tipi di *entità*, e le classi che le compongono:

- **InteractableArea**: classe base astratta che fornisce i metodi alle classi figlie. La classe è composta da un'Area3D, che invia i segnali quando il giocatore entra ed esce, e da una classe Control che rappresenta la UI che il giocatore visualizza quando entra.
- **NPC**: rappresenta un personaggio non giocabile che ha assegnato una semplice frase come messaggio. Questa frase viene visualizzata in una classe SimpleProjectLabel il cui funzionamento è stato spiegato nella sezione precedente. Presenta anche una classe Marker3D che segna la posizione della UI, e una classe NPCModel che gestisce le animazioni del modello 3D del personaggio.
- **InteractableSign**: rappresenta un cartello che il giocatore può leggere. Il cartello può contenere diverse informazioni, come una lista o un grafico. Il contenuto del cartello è inserito in un'altra classe Control.
- **NPCDialogue**: rappresenta un personaggio non giocabile che, a differenza della classe NPC, presenta un dialogo. Il giocatore può interagire con il personaggio e visualizzare il dialogo premendo il rispettivo tasto.

3.3.5. Gestione dei salvataggi

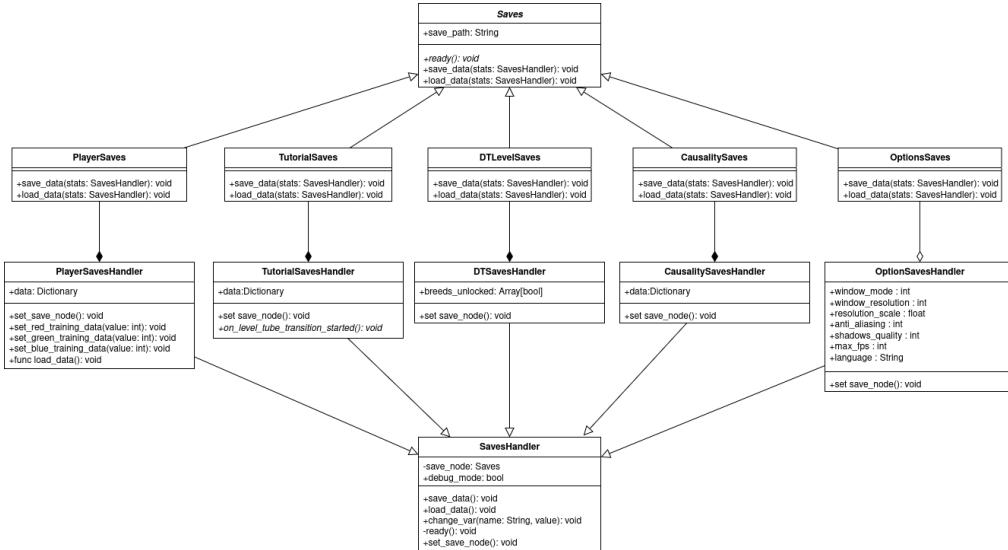


Figura 34: Diagramma sul funzionamento dei salvataggi

3.3.6. LevelsTransition

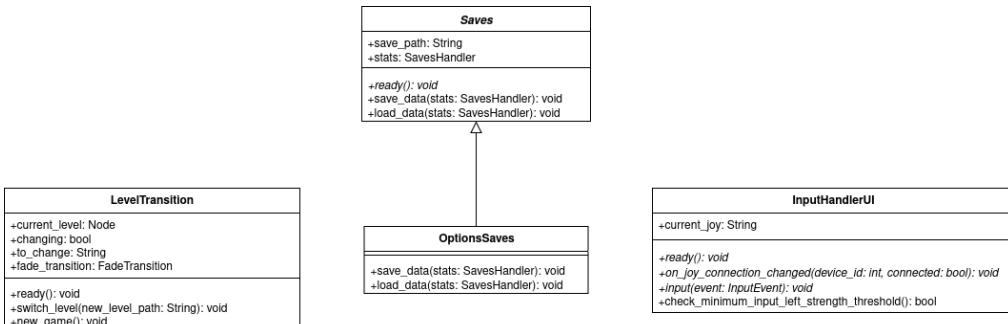


Figura 35: Diagramma UML delle classi *Autoloads*

La classe **LevelsTransition** si occupa della transizione tra due classi. Il metodo **switch_level(new_level_path: String)** carica il livello il cui percorso è fornito come argomento della funzione. Il motivo per cui è stato usato il percorso come **String** anziché il livello stesso come **PackedScene** è stato per evitare riferimenti ciclici, in quanto se due livelli contenevano un riferimento a tra di loro nella scena, il gioco non caricava correttamente il livello successivo.

3.3.7. OptionsSave

OptionsSave è la classe che carica le impostazioni del gioco. Queste opzioni devono essere sempre disponibili, in quanto servono per applicare le modifiche

fatte dal giocatore in qualsiasi momento, come la risoluzione alla finestra, il volume dell'applicazione, etc...

3.3.7.1. InputUIHandler

La classe `InputUIHandler` si occupa di controllare i dispositivi di *input* collegati, e mandare il segnale `device_changed` nel caso il nome del dispositivo dell'ultimo *input* non corrisponda all'attributo `current_joy`. Il metodo `check_minimum_input_left_strength_threshold()` stabilisce la potenza minima che l'*input* deve superare per mandare il segnale. Questo perché alcuni *joypad* presentano il problema del *drifting* e la classe potrebbe leggere *input* che non sono premuti dal giocatore.

3.3.8. Struttura base di un livello

Ogni livello viene creato con le seguenti classi:

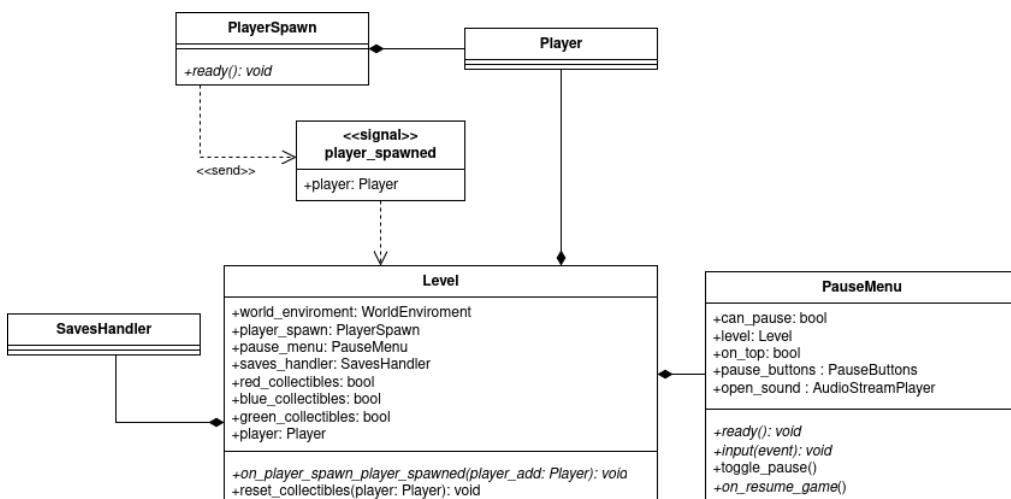


Figura 36: Diagramma delle classi di un livello base

- **Level**: classe del livello, non svolge molte funzioni visto che i componenti possono comunicare tra di loro attraverso segnali. Gli attributi booleani `red_collectibles`, `blue_collectibles` e `green_collectibles` stabiliscono quali tipi di *collezionabili* sono presenti nel livello, e quindi quali far visualizzare nella *UI* del giocatore.
- **PlayerSpawn**: classe che si occupa di generare il giocatore nella posizione in cui si trova. Appena generato il giocatore viene assegnato alla classe **Level**
- **PauseMenu**: il menu di pausa, questo viene caricato quando il giocatore preme il rispettivo tasto, mettendo in pausa tutta la scena.

3.3.9. LRCannon

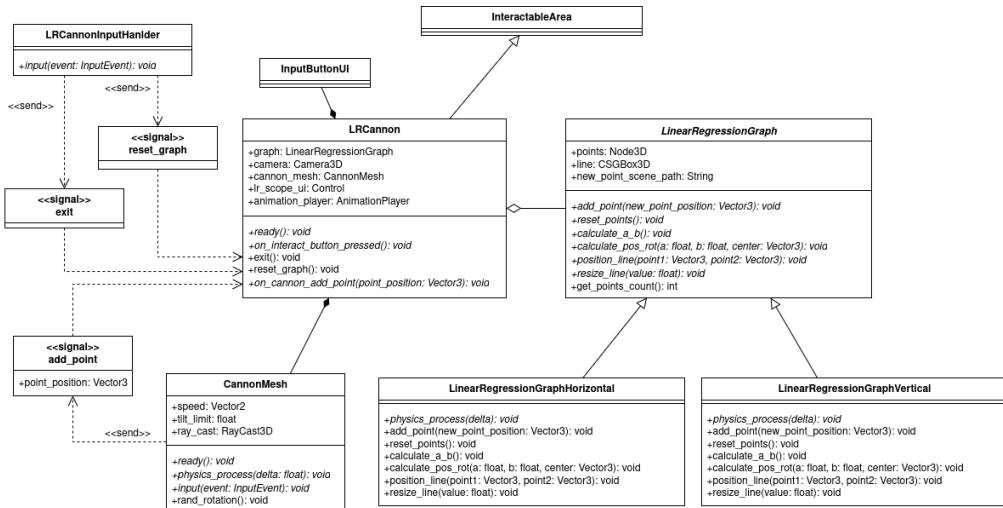


Figura 37: Diagramma sul funzionamento di un grafico *Linear Regression* nel gioco

La classe `LRCannon` rappresenta il cannone nel livello. Eredita da `InteractableArea` e infatti il giocatore può interagirci quando entra dentro l'area apposita.

Quando il giocatore preme l'*input* per interagire, la telecamera viene cambiata ed il giocatore entra nello stato *Interact*. La classe è composta da:

- **CannonMesh**: si occupa della rotazione del cannone quando questo è attivo e quando viene inserito un nuovo punto nel grafico.
 - **LRCannonInputHandler**: gestisce gli *input* del giocatore quando questo sta controllando il *cannone LR*.

3.3.10. LinearRegressionGraph

Classe base astratta usata per i due tipi di grafico presenti nel livello: orizzontale e verticale. Si occupa di svolgere le operazioni di regressione lineare per ottenere la formula della retta $y = a + bx$.

Tuttavia, non si può applicare la formula direttamente ad un oggetto 3D.

Il metodo **calculate_a_b()** si occupa di calcolare le variabili *a* e *b* della formula della retta con la seguente formula:

a =

Il metodo `calculate_pos_rot(a: float, b: float)`, poi, prende la formula della retta, e posiziona due punti nel grafico lungo la retta calcolata dal metodo precedente. Questi due punti vengono passati al metodo successivo.

Il metodo **position_line**(pos1: Vector3, pos2: Vector3), infine, posiziona il modello 3D della retta in mezzo ai due punti e lo ruota in modo che li

intersechi.

Le trasformazioni globali vengono poi modificate in base al tipo della classe:

- **LinearRegressionGraphHorizontal**: il grafico orizzontale, parallelo al terreno.
- **LinearRegressionGraphVertical**: il grafico verticale, perpendicolare al terreno.

3.3.11. Livello *Albero di decisione*

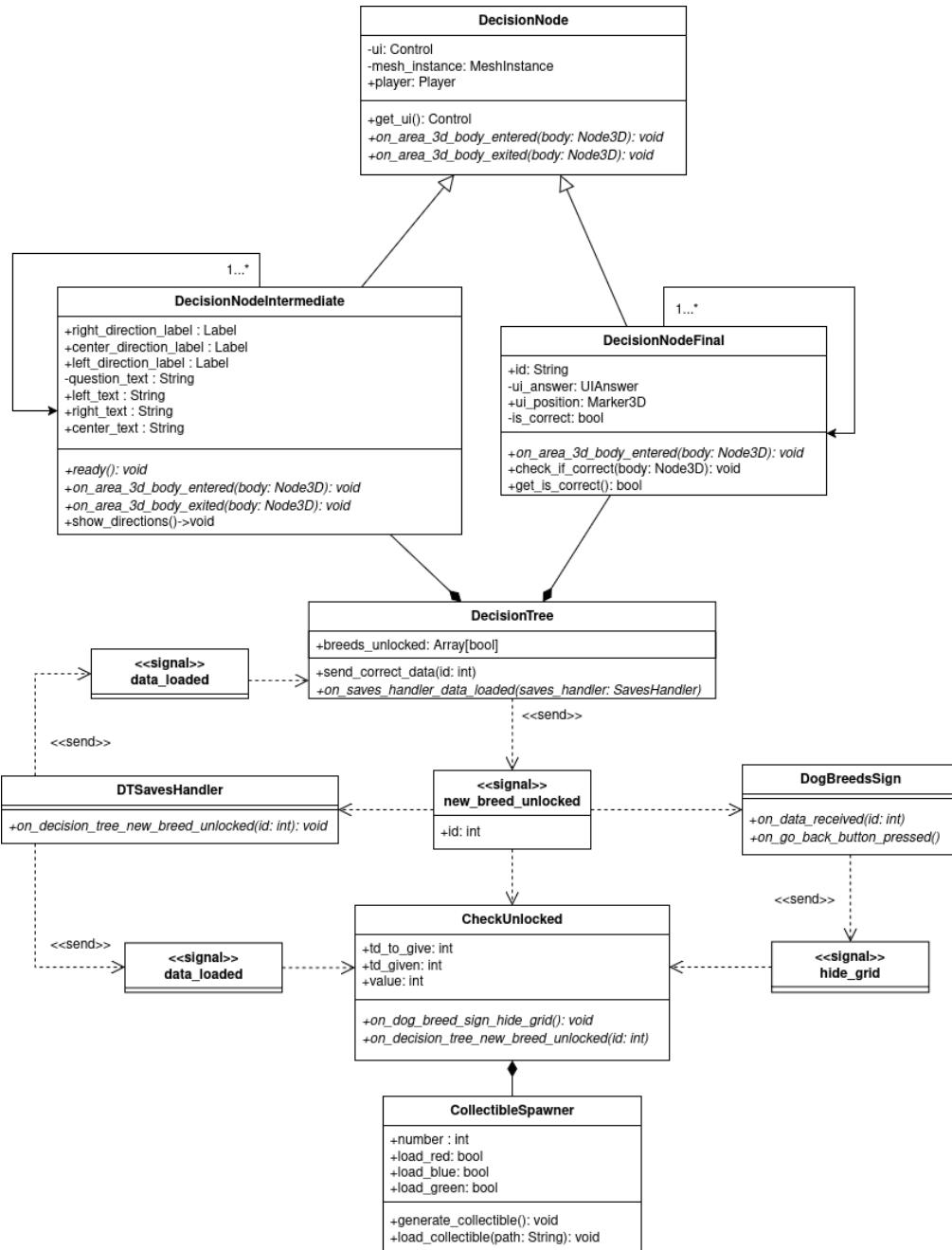


Figura 38: Diagramma sul funzionamento dell'Albero di decisione

- **DecisionTree**: la classe che rappresenta un Albero di decisione nel livello composta da più istanze di **DecisionNodeFinal** e **DecisionNodeIntermediate**, inserite tutte come nodi figli nella scena. Si occupa di inviare i segnali agli altri nodi presenti nel livello.

- **DecisionNode**: classe base astratta per i due tipi di nodi presenti nell'albero: *intermediate* e *final*. Fornisce i metodi virtuali `on_area_3d_body_entered(body: CharacterBody3D)` e `on_area_3d_body_exited(body: CharacterBody3D)` che vengono chiamati quando entra un oggetto di tipo `CharacterBody3D` nell'area sopra la piattaforma.

Il comportamento poi viene modificato dalle classi figlie.

- **DecisionNodeIntermediate**: quando entra il giocatore nell'area, viene visualizzata la *UI* con le indicazioni da seguire;
- **DecisionNodeFinal**: quando entra un cane nell'area, controlla se l'`id` di questo corrisponde all'`id` associato all'istanza.
- **DogBreedsSign**: oltre all'albero di decisione nel livello è presente anche un cartello con cui il giocatore può interagire e visualizzare le razze die cani che ha indovinato.

La classe `DogBreedsSign` rappresenta questo cartello. Questa, è composta da una classe `DogSignUI` che è il contenuto del cartello, contenente tutte le razze dei cani che il giocatore ha indovinato.

Quando il cartello viene chiuso, emette il segnale `hide_grid()` che chiama il metodo `on_dog_breed_sign_hide_grid()` nella classe `CheckUnlocked`.

- **CheckUnlocked**: classe che si occupa di controllare le razze di cani sbloccate e tenere il conto di quelle nuove che il giocatore non ha ancora controllato, nell'attributo `td_to_give`

Il valore di questo attributo viene modificato all'inizio del caricamento del livello e quando il giocatore indovina una nuova razza nell'albero di decisione, ed è la differenza tra l'attributo `value` e il valore `td_given`.

Al caricamento del livello, riceve il segnale `data_loaded` dal nodo che gestisce i salvataggi `DTsavesHandler`, assegna il valore dell'attributo `td_given`. Quando riceve il segnale `new_breed_unlocked` dall'albero di decisione, `value` aumenta di 1, ed aggiorna il valore di `td_to_give`.

Quando riceve il segnale `hide_grid` dal cartello, chiama la funzione per generare i `training_data` tanti quanti il valore di `td_to_give`.

3.3.12. Livello *Causalità*

3.3.12.1. Struttura del livello

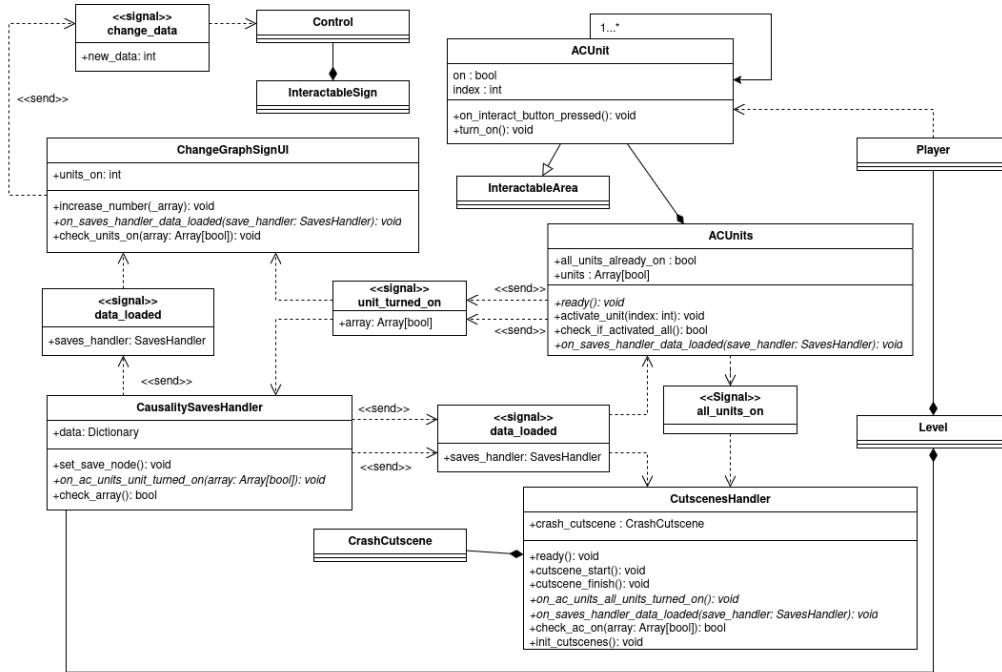


Figura 39: Diagramma del livello *Causalità*

- **ACUnit**: rappresenta un condizionatore. Eredita da **InteractableArea**, il giocatore, quando entra nell'area, può premere il tasto di interazione per accenderlo.
Quando viene acceso, il valore dell'array nodo padre, in questo caso **ACUnits**, viene aggiornato con il giusto indice.
- **ACUnits**: si occupa di gestire tutte le istanze di **ACUnit**, inserite come nodi figli nella scena.
Quando viene acceso un condizionatore, emette il segnale `unit_turned_on(index: int)`, passando direttamente l'array aggiornato come argomento nel segnale.
Quando tutti i condizionatori sono stati accesi, manda il segnale `all_units_on()`, usato in questo caso per far iniziare la scena di intermezzo.
- **CutscenesHandler**: si occupa di gestire le scene di intermezzo nel livello, inserite come nodi figli nella scena. Nonostante la classe è stata pianificata per gestire più scene, alla fine ne è presente solo una.
Questa classe svolge anche il ruolo da mediatore, ricevendo i segnali dal livello e mandandoli ai nodi figli, gestendo il traffico dei segnali.

3.3.12.2. Scena di intermezzo

Dopo la scena di intermezzo, alcuni personaggi possono cambiare il dialogo a loro assegnato, oppure il comportamento con il giocatore.

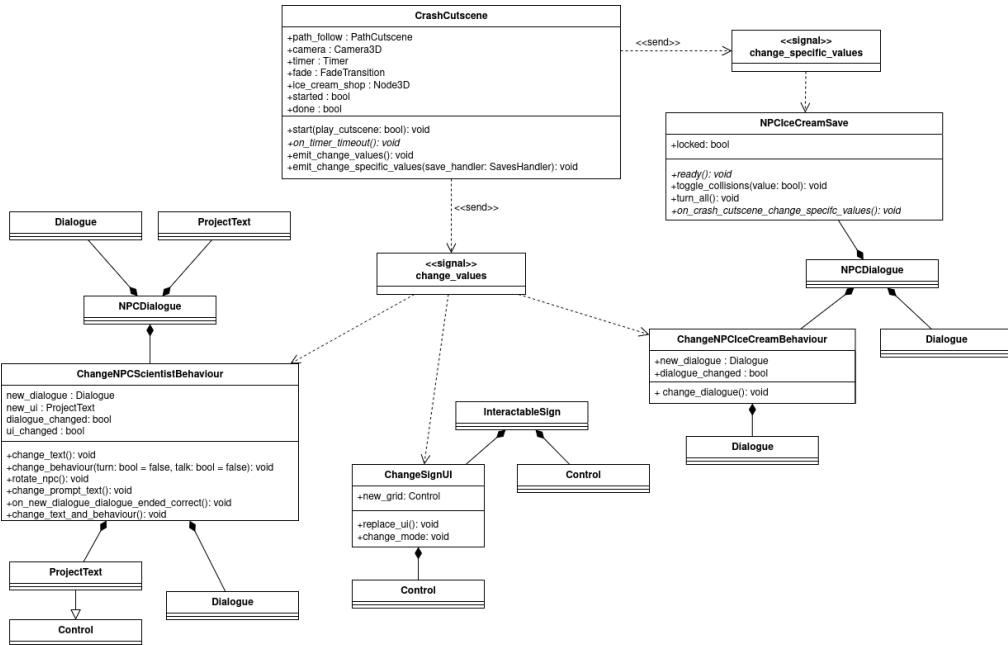


Figura 40: Diagramma sul funzionamento dei personaggi non giocabili nella scena di intermezzo

- **CrashCutscene**: la classe che gestisce la scena di intermezzo principale del livello. Si occupa principalmente di inviare i segnali per iniziare correttamente la scena.

Quando il giocatore accende l'ultimo condizionatore, viene emesso il segnale **change_values()**. Se invece il livello viene caricato con già tutti i condizionatori accessi, viene emesso il segnale **change_specific_values()**.

- **ChangeNPCScientistBehaviour**: si occupa di cambiare il comportamento del rispettivo personaggio non giocabile. Viene assegnata ad un nodo figlio del nodo del personaggio.

All'inizio del livello, il personaggio presenta un dialogo predefinito e non si gira quando parla con il giocatore.

Dopo aver ricevuto il segnale **change_values** dalla classe «CrashCutscene», sostituisce il dialogo del personaggio con il dialogo assegnato alla classe, e cambia il comportamento, in modo che si giri e cambi animazione quando parla con il giocatore.

Inoltre, dopo che il giocatore risponde correttamente alla domanda del nuovo dialogo, il comportamento cambia di nuovo, e viene tolto il dialogo, rimpiazzando il messaggio automatico che appare quando il giocatore entra nell'area di interazione.

- **ChangeNPCIceCreamBehaviour**: si occupa di cambiare il comportamento del rispettivo personaggio non giocabile. Viene assegnata ad un nodo figlio del nodo del personaggio.
Funziona nello stesso modo della classe descritta prima, però avviene solo un cambio del dialogo e non c'è la modifica del comportamento.
- **NPCIceCreamSave**: lo scopo della classe è quello di caricare il gruppo di persone davanti alla gelateria nel caso il livello venga caricato quando già tutti i condizionatori sono stati accesi.
Le persone vengono caricate quando la classe riceve il segnale *change_specific_values*, in quanto vengono caricate solo ed esclusivamente al caricamento del livello.
- **ChangeSignUI**: questa classe si occupa di cambiare il contenuto del rispettivo cartello. Viene assegnata ad un nodo figlio del nodo del cartello.
Funziona nello stesso modo delle classi che cambiano il dialogo o comportamento dei personaggi. Quando riceve il segnale *change_values*, cambia il contenuto del cartello, rimpiazzandolo con l'istanza assegnata alla classe.

3.4. Verifica e validazione

3.4.1. Macchina di *test*

Tutti i test sono stati eseguiti sulla mia macchina con specifiche hardware e software definite nella tabella. Molto importante è specificare le componenti della macchina su cui viene testato il gioco, dato che macchine diverse offrono prestazioni diverse.

Componente	Dettagli
<i>CPU</i>	<i>AMD® Ryzen 5 4500U</i>
<i>GPU</i>	<i>AMD® Radeon Graphics (RADV RENOIR)</i> Integrata alla <i>CPU</i>
<i>RAM</i>	<i>8GB DDR4</i>
Sistema Operativo	<i>Ubuntu 22.04</i>

Tabella 19: Componenti della macchina su cui sono stati eseguiti i *test*

3.4.2. Nomenclatura *test*

Di seguito sono elencate le metodologie di testing che verranno utilizzate per verificare e validare il prodotto software. Le metodologie di testing sono suddivise in quattro categorie:

- **test di unità:** test che verificano il corretto funzionamento di singole unità del codice, questi test sono stati svolti con l'*add-on* della community *GUT - Godot Unit Test*;
- **test di integrazione:** test che verificano il corretto funzionamento dell’interazione tra più unità del codice, anche questi svolti con l'*add-on GUT*;
- **test di sistema:** test che verificano il corretto funzionamento del sistema nel suo complesso, inclusi i requisiti funzionali e non funzionali, comprendono anche test sulle prestazioni, e sono svolti utilizzando gli strumenti forniti da *Godot*;
- **test di accettazione:** test che verificano se il prodotto è pronto per essere rilasciato.

3.4.3. Test di unità

Identificativo	Descrizione	Superato
TU-01	Si verifica che il giocatore stia su una piattaforma con velocità pari a zero	✓
TU-02	Si verifica che se il giocatore è fermo su una piattaforma, il suo stato nella macchina di stati è <i>Idle</i>	✓
TU-03	Si verifica che quando il giocatore è fermo su una piattaforma, utilizza l’animazione <i>idle</i>	✓
TU-04	Si verifica che la rotazione iniziale sull’asse y della telecamera è la stessa del giocatore	✓
TU-05	Si verifica che la telecamera ruoti intorno al giocatore quando viene premuto il rispettivo tasto	✓
TU-06	Si verifica che il giocatore si muovi ad una determinata velocità quando viene premuto il rispettivo tasto	✓
TU-07	Si verifica che se il giocatore si muove su una piattaforma, il suo stato nella macchina di stati è <i>GroundMove</i>	✓
TU-08	Si verifica che quando il giocatore si muove su una piattaforma, utilizza l’animazione della corsa	✓
TU-09	Si verifica che quando il giocatore si muove, la telecamera ruota automaticamente	✓

Identificativo	Descrizione	Superato
TU-10	Si verifica che quando il giocatore salti quando viene premuto il rispettivo tasto	✓
TU-11	Si verifica che quando il giocatore salta, lo stato nella macchina di stati cambia a <i>Air</i>	✓
TU-12	Si verifica che quando il giocatore salta utilizza l'animazione del salto	✓
TU-13	Si verifica che quando il giocatore cade da una piattaforma, con velocità verticale negativa, lo stato nella macchina di stati sia <i>Air</i>	✓
TU-14	Si verifica che quando il giocatore cade da una piattaforma, con velocità verticale negativa, utilizzi l'animazione della caduta	✓
TU-15	Si verifica che il menu di pausa venga visualizzato quando viene premuto il rispettivo tasto	✓
TU-16	Si verifica che quando viene aperto il menu di pausa, l'applicazione viene messa in pausa	✓
TU-17	Si verifica che quando viene premuto il tasto «ripredi», il menu di pausa viene nascosto e viene ripresa l'esecuzione	✓
TU-18	Si verifica che quando viene premuto lo stesso tasto quando il menu di pausa è aperto, questo viene nascosto e viene ripresa l'esecuzione	✓
TU-19	Si verifica che quando viene premuto il tasto delle opzioni, venga aperto il menu di opzioni	✓
TU-20	Si verifica che quando viene premuto il tasto «Ritorna alla hub», il giocatore viene riportato al livello Hub	✓
TU-21	Si verifica che quando viene premuto il tasto del menu principale, il giocatore viene riportato al menu principale	✓

Identificativo	Descrizione	Superato
TU-22	Si verifica che il gioco venga chiuso quando viene premuto il bottone «Esci dal gioco»	✓
TU-23	Si verifica che viene cambiata la modalità della finestra quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-24	Si verifica che viene cambiata la risoluzione della finestra quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-25	Si verifica che viene cambiata la scala di risoluzione quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-26	Si verifica che viene cambiato il valore del volume quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-27	Si verifica che viene cambiato il valore massimo degli FPS quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-28	Si verifica che viene cambiato il metodo di Anti Aliasing quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-29	Si verifica che viene cambiata la qualità/risoluzione delle ombre quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-30	Si verifica che viene cambiata la lingua del gioco quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-31	Si verifica che vengano salvati i nuovi valori delle opzioni quando viene premuto il rispettivo bottone nel menu opzioni	✓
TU-32	Si verifica che venga caricato il gioco con i salvataggi esistenti quando viene premuto il bottone «Carica partita»	✓
TU-33	Si verifica che venga caricata una nuova partita, cancellando i dati di salvataggio	✓

Identificativo	Descrizione	Superato
	esistenti, quando viene premuto il bottone «Nuova partita»	
TU-34	Si verifica che il messaggio del personaggio non giocabile venga nascosto quando il giocatore non è vicino	✓
TU-35	Si verifica che il personaggio non giocabile, sia con dialogo che senza dialogo, usi l'animazione «idle» quando il giocatore non è vicino	✓
TU-36	Si verifica che il grafico orizzontale venga caricato con la giusta rotazione	✓
TU-37	Si verifica che la linea del grafico orizzontale cambi correttamente con l'aggiunta di un punto	✓
TU-38	Si verifica che il grafico verticale venga caricato con la giusta rotazione	✓
TU-39	Si verifica che la linea del grafico verticale cambi correttamente con l'aggiunta di un punto	✓
TU-40	Si verifica che il grafico possa eliminare i punti aggiunti, resettando la linea	✓
TU-41	Si verifica che un cane possa ritornare alla sua posizione iniziale	✓
TU-42	Si verifica che l'NPC che esce dall'appartamento corra verso l'obiettivo	✓
TU-43	Si verifica che l'applicazione rilevi un cambio di dispositivo di Input	✓
TU-44	Si verifica che l'applicazione mostri i rispettivi <i>input</i> del dispositivo che si sta usando nella <i>UI</i>	✓

Tabella 20: *Test di unità*

3.4.4. Test di integrazione

Identificativo	Descrizione	Superato
TI-01	Si verifica che il personaggio non giocabile mostri il messaggio quando il giocatore si avvicina	✓
TI-02	Si verifica che il personaggio non giocabile senza dialogo usi l'animazione per parlare quando il giocatore si avvicina	✓
TI-03	Si verifica che il personaggio non giocabile con il dialogo usi l'animazione per salutare quando il giocatore si avvicina	✓
TI-04	Si verifica che quando il giocatore preme il rispettivo <i>input</i> vicino ad un' <i>entità</i> con cui può interagire, lo stato nella macchina di stati passi a «Interact»	✓
TI-05	Si verifica che quando il giocatore preme il rispettivo <i>input</i> vicino a un cartello, ne visualizzi i contenuti	✓
TI-06	Si verifica che quando il giocatore preme lo stesso <i>input</i> durante un'interazione, smette di interagire con l' <i>entità</i>	✓
TI-07	Si verifica che quando il giocatore preme il rispettivo <i>input</i> vicino a un personaggio con un dialogo, il giocatore si ferma e visualizza il dialogo del personaggio	✓
TI-08	Si verifica che quando il giocatore preme il rispettivo <i>input</i> durante un dialogo, va avanti se esiste un messaggio successivo	✓
TI-09	Si verifica che quando il giocatore preme il rispettivo <i>input</i> durante un dialogo, finisce se non esiste un messaggio successivo	✓
TI-10	Si verifica che quando il giocatore preme il rispettivo <i>input</i> durante la scelta di un'opzione del dialogo, questa viene scelta ed il dialogo procede con il prossimo messaggio	✓

Identificativo	Descrizione	Superato
TI-11	Si verifica che quando il giocatore smette di interagire con un'entità, lo stato nella macchina di stati del giocatore torna a «Idle»	✓
TI-12	Si verifica che il prompt dell' <i>input</i> da premere appaia quando il giocatore si posiziona sopra la piattaforma del cannone <i>LR</i>	✓
TI-13	Si verifica che il giocatore interagisca con il cannone <i>LR</i> quando preme il rispettivo tasto sopra la piattaforma	✓
TI-14	Si verifica che il giocatore possa posizionare un punto sul grafico <i>LR</i>	✓
TI-15	Si verifica che il giocatore possa interrompere l'interazione con il cannone <i>LR</i>	✓
TI-16	Si verifica che quando il giocatore si avvicina ad un oggetto che può raccogliere, viene mostrato sullo schermo il tasto da premere	✓
TI-17	Si verifica che quando il giocatore raccoglie un oggetto, lo stato nella macchina di stati passi a «Grab»	✓
TI-18	Si verifica che l'oggetto raccolto sia lo stesso che il giocatore sta portando	✓
TI-19	Si verifica che il giocatore possa muoversi con l'oggetto	✓
TI-20	Si verifica che il giocatore possa saltare con l'oggetto	✓
TI-21	Si verifica che il giocatore può lasciare l'oggetto	✓
TI-22	Si verifica che quando il giocatore lascia un oggetto, lo stato nella macchina di stati passi a «Release»	✓
TI-23	Si verifica che l'oggetto rimanga nella posizione dove è stato lasciato	✓
TI-24	Si verifica che le informazioni dei rami vengano visualizzate quando il giocatore si	✓

Identificativo	Descrizione	Superato
	posiziona sopra un nodo dell'albero di decisione	
TI-25	Si verifica che il nodo riconosca se il cane posizionato sopra è corretto	✓
TI-26	Si verifica che il nodo riconosca se il cane posizionato sopra è sbagliato	✓
TI-27	Si verifica che il cartello con le razze di cani indovinate si aggiorni quando il giocatore dà una risposta corretta	✓
TI-28	Si verifica che un'unità esterna di un condizionatore venga accesa quando il giocatore preme il rispettivo <i>input</i> quando è vicino	✓
TI-29	Si verifica che il grafico dei condizionatori usati venga aggiornato all'inizio del livello	✓
TI-30	Si verifica che il grafico dei condizionatori usati venga aggiornato quando viene acceso un condizionatore	✓
TI-31	Si verifica che la scena di intermezzo inizi quando il giocatore accende tutti i condizionatori nel livello «Causalità»	✓
TI-32	Si verifica che i personaggi non giocabili che seguono un obiettivo, smettano di seguirlo quando entrano in una specifica area	✓
TI-33	Si verifica che il personaggio «scienziato» cambi dialogo quando vengono accese tutti i condizionatori nel livello «Causalità»	✓
TI-34	Si verifica che il personaggio «gelataio» cambi dialogo quando vengono accesi tutti i condizionatori nel livello «Causalità»	✓
TI-35	Si verifica che quando il giocatore si avvicina ad un «Training data» di colore rosso, questo viene preso ed aumenta il rispettivo contatore	✓

Identificativo	Descrizione	Superato
TI-36	Si verifica che quando il giocatore si avvicina ad un «Training data» di colore verde, questo viene preso ed aumenta il rispettivo contatore	✓
TI-37	Si verifica che quando il giocatore si avvicina ad un «Training data» di colore blu, questo viene preso ed aumenta il rispettivo contatore	✓
TI-38	Si verifica che quando il giocatore cade dal livello, torna in una zona dove si trovava precedentemente	✓

Tabella 21: *Test* di integrazione

3.4.5. *Test* di sistema

Identificativo	Descrizione	Superato
TS-01	Si verifica che il gioco ricevi <code>_input_</code> dalla tastiera	✓
TS-02	Si verifica che il gioco riceva <code>_input_</code> da un joystick generico	✓
TS-03	Si verifica che il gioco mantenga almeno 30fps durante l'esecuzione (caricamenti esclusi)	✓
TS-04	Si verifica che il tempo tra un <code>_frame_</code> e l'altro sia minore di 33.3 millisecondi durante l'esecuzione (caricamenti esclusi)	✓
TS-05	Si verifica che il tempo tra un <code>_frame_</code> di fisica e l'altro rimanga costante a 16.67 millisecondi durante l'esecuzione (caricamenti esclusi)	✓
TS-06	Si verifica che l'uso della memoria video (VRAM) non superi 500MB durante tutta l'esecuzione	✓

Identificativo	Descrizione	Superato
TS-07	Si verifica che l'uso della memoria statica non superi 200MB durante tutta l'esecuzione	✓
TS-08	Si verifica che il tempo necessario alla CPU per caricare un <u>frame</u> sia minore di 2 millisecondi	✓
TS-09	Si verifica che il tempo necessario alla GPU per caricare un <u>frame</u> sia inferiore a 33.3 millisecondi	✓
TS-10	Si verifica che non siano presenti nodi non utilizzati nella scena	✓

Tabella 22: *Test* di sistema

3.4.6. *Test* di accettazione

Identificativo	Descrizione	Superato
TA-01	Si verifica che il gioco funzioni nel sistema operativo Linux	✓
TA-02	Si verifica che il gioco funzioni nel sistema operativo Windows 11	✓
TA-03	Si verifica che il gioco rilevi <i>input</i> da tastiera	✓
TA-04	Si verifica che il giocatore possa muoversi in uno spazio tridimensionale	✓

Tabella 23: Tabella dei *test* di accettazione

3.5. Risultati ottenuti

Qui descrivo i risultati raggiunti rispettivamente su piano qualitativo e quantitativo, con copertura dei requisiti, testing e quantità di prodotti

Capitolo 4

Conclusioni

In questo capitolo effettuo una retrospettiva sul progetto e sulla mia esperienza di stage, analizzando le esperienze acquisite durante il periodo. Infine metto a confronto gli argomenti insegnati dal percorso di studi e quelli richiesti per lo sviluppo del progetto.

4.1 Obiettivi stage soddisfatti

Qui elenco gli obiettivi che erano stati dichiarati a inizio nel capitolo 2 e quelli soddisfatti a fine stage

4.2 Esperienze acquisite

Come dice il titolo della sezione, qui descrivo le esperienze e conoscenze che ho acquisito durante lo stage

4.3 Differenza tra stage e percorso studi

Qui tratto della differenza tra gli argomenti trattati durante lo stage e gli argomenti insegnati durante il percorso di studi

4.3.1 Lacune sul percorso studi

Se presenti, in questa sezione descrivo alcune delle mie lacune verso gli argomenti insegnati nel percorso di studi verso lo stage e/o mondo del lavoro

4.4 Pensieri finali

Qui concludo la tesi con miei pensieri finali sull'ambiente di lavoro, il progetto, e quello che ho imparato per entrare nel mondo del lavoro

Glossario

.glb – GLTF (Graphics Library Transmission Format) Binary: formato standard di un modello tridimensionale che legge il modello 3D come un file binario, permettendo una lettura e *rendering* più veloce e minimizzando lo spazio occupato dal *file*. 16.

.png – Portable Network Graphics: formato di immagine raster senza perdita di qualità, ampiamente utilizzato per la grafica web e il design digitale. 16.

Alberi di decisione: modello predittivo utilizzato in statistica e *Machine Learning*, che rappresenta le decisioni e le loro possibili conseguenze sotto forma di un albero, facilitando l'interpretazione e la visualizzazione delle scelte. 8.

autoload: meccanismo di *Godot* che consente di caricare automaticamente una risorsa all'avvio del gioco, rendendola disponibile in tutte le scene senza doverla caricare manualmente. 49.

Blender: *software* di modellazione ed animazione 3D *open source* usato per creare modelli 3D ed animazioni. 16.

brainstorming: tecnica di generazione di idee in gruppo, in cui i partecipanti sono incoraggiati a esprimere liberamente le proprie idee senza giudizio, al fine di stimolare la creatività e trovare soluzioni innovative a un problema. 5.

CD – Continuos Delivery: pratica di sviluppo *software* che consente di rilasciare frequentemente e in modo affidabile nuove versioni del *software*, garantendo che il codice sia sempre in uno stato pronto per la produzione. 18.

CI – Continuos Integration: pratica di sviluppo *software* che consente di integrare frequentemente le modifiche del codice in un repository condiviso, garantendo che il codice sia sempre in uno stato funzionante e testato. 18.

clipping: fenomeno che si verifica quando la telecamera di un videogioco passa attraverso un oggetto solido, causando la visualizzazione errata della scena. 50.

database: insieme organizzato di dati, generalmente memorizzato e gestito in modo da facilitarne l'accesso e la manipolazione. In ambito *software*, i database sono utilizzati per archiviare informazioni in modo strutturato, consentendo operazioni di ricerca, aggiornamento e gestione dei dati. 3.

diagrammi di Gantt: strumento di gestione dei progetti che rappresenta graficamente le attività pianificate nel tempo, mostrando la durata, le dipendenze e le scadenze delle attività in un formato visivo facile da comprendere. 21.

drifting: fenomeno che si verifica quando un dispositivo di input, come uno *stick* analogico, registra movimenti o pressioni anche quando non viene toccato, causando comportamenti indesiderati nel gioco. 55.

frame: unità di misura temporale utilizzata nei videogiochi e nelle animazioni, che rappresenta un singolo fotogramma di un'animazione o di un ciclo di gioco. Tipicamente sono 60 in un secondo (60 FPS - *Frames Per Second*). 48.

Game Design: disciplina che si occupa della progettazione e dello sviluppo di giochi, sia da tavolo che digitali, considerando aspetti come la meccanica di gioco, la narrazione, l'estetica e l'interazione con il giocatore. 7.

GDScript: linguaggio di programmazione specifico per il motore di gioco Godot, progettato per essere semplice e intuitivo, con una sintassi simile a *Python*. Viene utilizzato per scrivere *script* che controllano la logica del gioco, le interazioni e le funzionalità. 15.

GLSL ES – OpenGL Shading Language for Embedded Systems: linguaggio di shading utilizzato per scrivere shader per applicazioni embedded, come giochi e grafica in tempo reale. 17.

hosting: servizio che consente di archiviare e rendere accessibili online siti web, applicazioni o progetti software, fornendo le risorse necessarie per il loro funzionamento e la loro distribuzione. 18.

IK – Inverse Kinematics: soluzione usata nell'ambito dell'animazione 3D. Si tratta di semplificare l'animazione calcolando il movimento di altre ossa o articolazioni in base all'ultimo osso della catena. Ad esempio, automatizza il movimento del braccio muovendo solo la mano, anziché ruotare singolarmente braccio, avambraccio e mano. Questo metodo risulta anche molto più simile a come ci si muove naturalmente. 16.

joypad: dispositivo di input utilizzato principalmente per come dispositivo di *input* nei videogiochi, dotato di pulsanti, leve e altri controlli per interagire con il gioco. 25.

LLM – Large Language Model: modello di intelligenza artificiale progettato per comprendere e generare testo in linguaggio naturale, addestrato su grandi quantità di dati testuali per svolgere compiti come la traduzione, la risposta a domande e la generazione di contenuti. 5.

ML – Machine Learning: disciplina che si occupa dello sviluppo di algoritmi e modelli statistici che permettono ai computer di apprendere dai dati e migliorare le proprie prestazioni nel tempo senza essere esplicitamente programmati. 8.

materiale: insieme di proprietà che definiscono l'aspetto visivo di un oggetto 3D, come colore, riflessione, trasparenza e altre caratteristiche ottiche. In *Godot*, i materiali possono essere applicati a modelli 3D per controllare il loro aspetto durante il *rendering*. 16.

Microsoft Teams: piattaforma di comunicazione e collaborazione sviluppata da Microsoft, che fornisce chat, videoconferenze, condivisione di file e lavoro di gruppo integrato con gli strumenti forniti da Microsoft. 3.

motore di gioco: software progettato per facilitare lo sviluppo di videogiochi, fornendo strumenti e funzionalità per la gestione della grafica, della fisica, dell'audio e di altre componenti del gioco. 12.

MVP – Minimum Viable Product: Versione minima di un prodotto che include solo le funzionalità essenziali per essere utilizzato dagli utenti. 14.

Nearest Neighbor: algoritmo di apprendimento automatico utilizzato per la classificazione e la regressione, che si basa sull'idea di trovare i punti dati più vicini a un dato punto di input e fare previsioni in base a questi punti. 8.

NLA – Nonlinear Animation: sistema di gestione delle animazioni in *Blender* che consente di combinare e sovrapporre diverse animazioni in modo non lineare, permettendo una maggiore flessibilità e controllo sulle animazioni dei modelli 3D. 16.

open source: modello di sviluppo software che promuove la collaborazione e la condivisione del codice sorgente, consentendo a chiunque di utilizzare, modificare e distribuire il software liberamente. 12.

OpenGL – Open Graphics Language: linguaggio di programmazione grafica utilizzato per creare applicazioni 3D e 2D, fornendo un’interfaccia standardizzata per l’interazione con la scheda grafica del computer. 11.

PascalCase: pratica di scrivere parole composte o frasi unendo tutte le parole tra loro, ma lasciando le loro iniziali maiuscole. 15.

PoC – Proof of Concept: una dimostrazione pratica che ha lo scopo di verificare la fattibilità o il potenziale di un’idea, concetto o soluzione. È spesso utilizzato nelle fasi iniziali di un progetto per validare il funzionamento teorico e pratico, incluso il modo in cui diverse componenti del sistema possono integrarsi tra loro per raggiungere l’obiettivo prefissato. 14.

push: operazione che consente di inviare le modifiche locali del codice a un *repository* remoto, aggiornando così la versione del codice condiviso con altri membri del *team*. 18.

Python: linguaggio di programmazione di alto livello, noto per la sua sintassi semplice e leggibile, ampiamente utilizzato in vari ambiti come lo sviluppo web, l’analisi dei dati, l’intelligenza artificiale e la scienza dei dati. 17.

Regressione lineare: modello statistico utilizzato per analizzare la relazione tra una variabile dipendente e una o più variabili indipendenti, assumendo che questa relazione sia lineare. 8.

rig: struttura scheletrica applicata a un modello 3D che consente di animarlo tramite la manipolazione di ossa e articolazioni. 16.

shader: programma che calcola l’aspetto visivo di un oggetto 3D, determinando come la luce interagisce con le superfici. 11.

singleton: design pattern garantisce che esista un’unica istanza di una classe, garantendo un punto di accesso globale a essa. 49.

snake_case: pratica di scrivere parole composte separando le parole tramite trattino basso, solitamente con le prime lettere delle singole parole in minuscolo. 15.

StageIT: evento orientato al lavoro organizzato dall’Università degli Studi di Padova, dedicato agli studenti per aiutarli a trovare aziende dove svolgere l’attività di *stage*. 7.

Support Vector Machines: algoritmo di apprendimento automatico utilizzato per la classificazione e la regressione, che cerca di trovare il margine ottimale che separa le diverse classi nel piano. 8.

temperatura: parametro che controlla la casualità delle risposte generate da un LLM. Valori più bassi rendono le risposte più conservative e focalizzate, mentre valori più alti aumentano la creatività e la varietà delle risposte. 6.

termine: termine esempio per dimostrare come funziona il glossario. vi

texture: immagine bitmap applicata a un modello 3D per fornire dettagli visivi, come colori e pattern. Ne esistono di vario tipo e possono essere utilizzate, ad esempio, per dare colore al modello 3D o modificare il valore della luce riflessa da questo. 16.

ticket: segnalazione o richiesta registrata in un sistema di tracciamento (come *Github Issues* o *Jira*) che descrive un problema, una funzionalità da implementare o un'attività da svolgere all'interno di un progetto *software*. 14.

tutorial: concetto usato per indicare una guida introduttiva ad un determinato argomento. In questo caso, rappresenta il livello introduttivo di un videogioco, progettato per insegnare all'utente i concetti base del gioco, ad esempio i comandi. 9.

UI – User Interface: insieme di elementi grafici e controlli che consentono all'utente di interagire con un'applicazione o un videogioco. 40.

Bibliografia

- [1] Juan Linietsky, Ariel Manzur, e comunità di Godot, «Nodi e Scene», 2014. [Online]. Disponibile su: https://docs.godotengine.org/it/4.x/getting_started/step_by_step/nodes_and_scenes.html
- [2] Juan Linietsky, Ariel Manzur, e comunità di Godot, «Processi di inattività e fisica», 2014. [Online]. Disponibile su: https://docs.godotengine.org/it/4.x/tutorials/scripting/idle_and_physics_processing.html