

Relazione Progetto OOP “Dimension Holiday”

Lorenzo Prati, Elvis Perlika
Emanuele Dajko, Alessandra Versari

June 2, 2023

Contents

1	Analisi	2
1.1	Requisiti	2
1.2	Analisi e modello del dominio	3
2	Design	5
2.1	Architettura	5
2.2	Design dettagliato	7
2.2.1	Lorenzo Prati	7
	Engine	7
3	Sviluppo	9
3.1	Testing automatizzato	9
3.2	Metodologia di lavoro	9
3.3	Note di sviluppo	9
4	Commenti finali	10
4.1	Autovalutazione e lavori futuri	10
4.2	Difficoltà incontrate e commenti per i docenti	10
A	Guida utente	11
B	Esercitazioni di laboratorio	12

Chapter 1

Analisi

1.1 Requisiti

Il gruppo si pone l'obiettivo di realizzare un videogioco roguelike *Dimension Holiday* vagamente ispirato a giochi famosi come *Hades* oppure *The Binding of Isaac*. Il giocatore controllerà un personaggio che è stato trasportato in un altro mondo dove dovrà esplorare un dungeon e affrontare un boss per tornare alla propria dimensione.

Elementi funzionali

- Per attraversare tutto il dungeon il giocatore dovrà passare da stanza in stanza, eliminando tutti i nemici presenti. Per farlo potrà usare la spada o lanciare proiettili energetici. Giocatore e nemici hanno delle vite (esprese in cuori) e se il giocatore perde tutti i cuori e' *Game over* e si deve ricominciare da capo
- Una volta che avrà ucciso tutti i nemici della stanza corrente compariranno dei reward casuali (cuori, monete ecc.) e il portale che ti trasporterà nella prossima stanza.
- Dopo un certo numero di stanze, comparirà una stanza shop dove sarà possibile effettuare acquisti usando le monete creando una piccola progressione
- Il gioco si conclude quando il giocatore sconfigge un nemico speciale chiamato *Boss*

Elementi non funzionali

- Ci si pone l'obiettivo di creare un'architettura del software modulare ed espandibile ad aggiunte future, come l'aggiunta di nuovi nemici, mappe e oggetti.

1.2 Analisi e modello del dominio

Il giocatore potrà muoversi nelle 4 direzioni su, sinistra, giù, destra tramite i tasti, rispettivamente, W, A, S, D ed effettuare due tipi di attacchi:

- *meele*, ovvero ravvicinato, usando una spada e tramite il tasto sinistro del mouse
- dalla distanza, usando un proiettile energetico e tramite il tasto destro del mouse

Il gioco dovrà essere in grado di presentare al **giocatore** una serie di stanze dove affrontare dei **nemici**. Questi potranno avere diversi comportamenti e diverse tipologie di **attacco**. Il giocatore dovrà stare attento ad evitare gli attacchi dei nemici per non perdere cuori e allo stesso tempo non entrarci in contatto, cosa che comporterà ulteriore danno. Saranno presenti degli **oggetti** raccogliabili (cuori, monete ecc.) che dovranno applicare degli **effetti** alle entità con cui entrano in contatto, ad esempio l'incremento della vita oppure l'incremento della valuta posseduta dal giocatore. Lo shop sarà gestito *in game*, nel senso che non comparirà un'interfaccia grafica che permetterà al giocatore di scegliere i potenziamenti, ma il giocatore dovrà interagire dinamicamente con degli oggetti presenti nella mappa per acquistarli. Anche gli attacchi applicheranno degli effetti con le entità con cui entrano in contatto, come ad esempio la perdita di cuori. Il **mondo** di gioco (*dungeon*) sarà composto di una serie di stanze. Tramite l'interazione con un oggetto portale, il giocatore sarà trasportato alla stanza successiva senza possibilità di tornare indietro. All'interno delle stanze sono presenti dei muri, che bloccano il passaggio al giocatore. Esistono tre tipi di stanze: normale, shop, boss. Nella stanza normale compariranno dei nemici, in numero e tipo variabile in base al momento della partita, nella stanza shop invece compariranno gli oggetti rappresentati i potenziamenti acquistabili, e nella stanza boss comparirà solo il nemico boss. Le stanze normali saranno intercambiate randomicamente tra un pool prescelto di mappe create a mano, mentre le stanze shop e boss saranno uniche.

La difficoltà sarà gestita in modo tale che proseguendo nel dungeon risulti più difficile il gioco, ad esempio facendo comparire più nemici nelle stanze

oppure nemici piu' forti. Questo aumento della difficoltà sara' compensato dai potenziamenti che il giocatore potra' acquistare nello shop, che comparira' dopo un numero costante di stanze normali superate.

Una delle maggiori difficoltà consistera' nella creazione di un architettura che permetta la gestione sia di diversi tipi di nemici (zombie, shooter, boss ecc.), ognuno con un proprio comportamento, sia di diversi attacchi utilizzabili sia dal giocatore che dai nemici (proiettili, attacchi meelee). Inoltre, si cerchera' di realizzare un sistema di combattimento *real-time* quanto piu' possibile fluido e responsivo e un'alternanza di mappe e generazione dei nemici in modo tale da far sembrare ogni partita diversa.

Dato il monte ore previsto, si rimanda al futuro una gestione accurata delle performance del gioco.

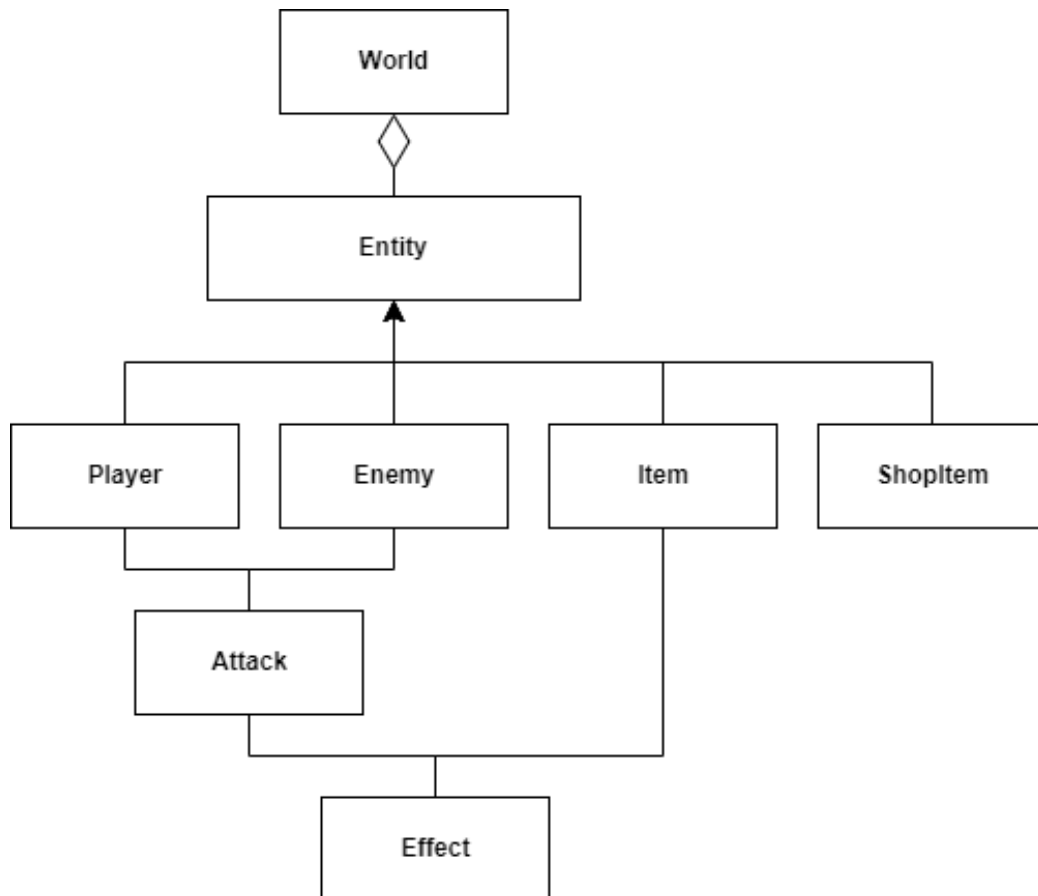


Figure 1.1: Diagramma uml concettuale rappresentante le varie entita' presenti nel dominio del gioco

Chapter 2

Design

2.1 Architettura

Abbiamo deciso di utilizzare per il modello logico del gioco e per la logica di controllo una versione semplificata del pattern Entity-Component-System (ECS) mantenendo la parte grafica separata. Questo pattern si basa come da definizione su tre parti fondamentali, che interconnesse permettono una buona suddivisione delle responsabilità, un alto riuso e un alto grado di composizione (composition over inheritance).

Di seguito spieghiamo le varie parti della nostra architettura:

- **Components:** sono oggetti utili a mantenere dei dati che descrivono un certo aspetto del modello di gioco (ad esempio la posizione sul piano, il movimento ecc.).
- **Entity:** sono dei raccoglitori di Component. Ogni entità è descritta dai suoi componenti che permettono alla stessa di distinguerla dalle altre entità. Ad esempio, diverse entità presenti in gioco nello stesso momento potrebbero contenere un PositionComponent, un MovementComponent, un BodyComponent, un HealthComponent e altri, che ne descrivono le proprietà comuni. Tramite l'aggiunta di AIComponent o di un PlayerComponent, solo per citarne alcuni, è possibile distinguere i nemici dal giocatore.
- **Systems:** sono la parte dell'architettura che si occupa di operare sulle entità, modificandone i componenti e quindi svolgendo la maggior parte della logica del gioco. L'esecuzione sequenziale di vari sistemi, ciascuno che scorre le entità e opera su un determinato set di componenti, permette il funzionamento del gioco. Ad esempio, il MovementSystem opera esclusivamente sulle entità che contengono il Move-

mentComponent e si occupa di muovere tutte le entita'; mentre il CheckHealthSystem si occupa di prendere tutte le entita' che hanno un HealthComponent e di rimuovere quelle che hanno esaurito le vite. E' quindi sufficiente, per inserire nel gioco una nuova meccanica, costruire nuovi componenti che definiscano nuove proprieta' e un nuovo sistema che operi su di essi senza il bisogno di andare a modificare i sistemi o i componenti precedentemente creati.

- Engine e World: queste sono le classi che controllano effettivamente lo svolgersi del gioco. Engine si occupa di gestire il game loop, mentre il World contiene al suo interno le entita' di gioco, schedula i system e passa alla View le informazioni necessarie.
- View: e' gestita in modo indipendente dal resto dell'architettura e si occupa solamente di disegnare lo stato del modello di gioco. Inoltre, gestisce diverse schermate, si occupa di disegnare l'interfaccia grafica e di registrare gli input da mouse e tastiera.

2.2 Design dettagliato

2.2.1 Lorenzo Prati

Engine

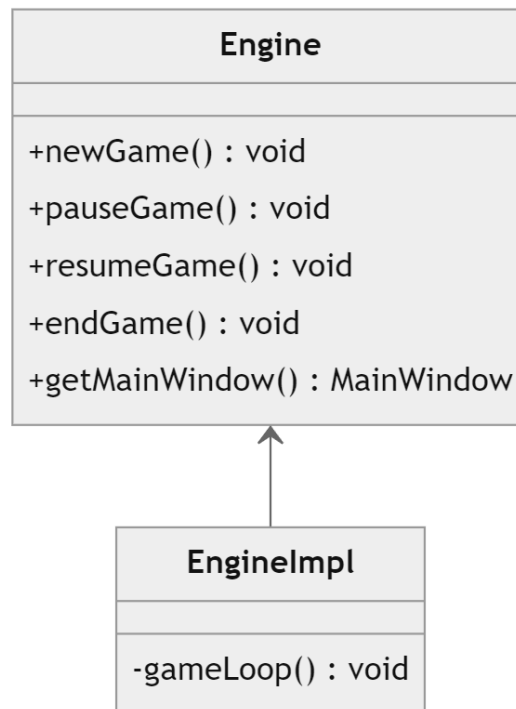


Figure 2.1: Diagramma UML dell'Engine

Problema Realizzare una classe che permetta lo svolgimento effettivo del main loop del gioco, attraverso il quale scandire gli update del modello logico e della rappresentazione grafica, e che contenga tutti gli elementi per mantenere attiva l'applicazione. All'occorrenza, deve anche essere possibile mettere in pausa il gioco e riprenderlo. Deve essere inoltre gestita la fine del gioco.

Soluzione La soluzione e' ricaduta sulla creazione di un'interfaccia **Engine** con relativa implementazione **EngineImpl**. Questa classe fa uso del *pattern game loop*, realizzato in un metodo privato e vengono esposti dall'interfaccia i metodi necessari alle altre classi (ad esempio le schermate della View)

per controllare il loop. E' quindi possibile metterlo in pausa, riprenderlo e arrestarlo. Essendo questa classe la prima che viene creata al lancio dell'applicazione, essa si occupa anche internamente di creare il modello logico e la view.

World

Chapter 3

Sviluppo

3.1 Testing automatizzato

3.2 Metodologia di lavoro

3.3 Note di sviluppo

Chapter 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.2 Difficoltà incontrate e commenti per i docenti

Appendix A

Guida utente

Appendix B

Esercitazioni di laboratorio

Bibliography