Chronicles of Eldoria

Vincenzo Barreca
Giovanni Casano
Emanuele Castronovo
Vincenzo Galia
Manuele Mormorio

11/01/2024

Indice

1 Introduzione

Lo scopo del progetto *Chronicles of Eldoria* è sviluppare un coinvolgente gioco in Java, che integri elementi di Ingegneria e Sicurezza del Software per offrire un'esperienza avvincente per gli utenti. Il gioco si struttura seguendo specifiche chiave:

- Un'avventura coinvolgente con un protagonista principale.
- Decisioni cruciali del giocatore che influenzano lo sviluppo della trama.
- Introduzione di vari personaggi, tra cui compagni di viaggio e personaggi non giocabili.
- Risoluzione di prove e enigmi per progredire nell'avventura.

Il gioco *Chronicles of Eldoria* è un'avventura testuale che mette al centro le scelte del giocatore. Ogni decisione influenzerà lo svolgersi della trama e condurrà a finali differenti, aggiungendo una componente interattiva alla narrazione.

Nel regno magico di Eldoria, la prosperità alimentata dalla Pietra dell'Aurora è minacciata dal malvagio Signore delle Ombre, Sauron. La trama segue il protagonista, erede dei Sacerdoti della Luce Astrale, incaricato di salvare Eldoria dalle forze oscure. La Pietra dell'Aurora, fonte di magia del regno, è ora al centro di una lotta epica.

Gideon, l'anziano sacerdote, ha sacrificato se stesso per proteggere la pietra, trasferendo una parte della sua anima al suo interno. L'avventura inizia con l'indebolimento dello scudo di luce, costringendo il protagonista, Gideon, a una scelta epica. Il giocatore segue il percorso del protagonista attraverso scelte e sfide, esplorando Eldoria, interagendo con personaggi e svelando la verità dietro la caduta del regno.

Il giocatore sara' chiamato ad affrontare enigmi, creature oscure e infine le forze di Sauron. Superare le prove è essenziale per avanzare nella storia, garantendo un'esperienza coinvolgente.

2 Analisi e specifica dei requisiti

Nell'ambito dello sviluppo di software, i requisiti funzionali definiscono le funzionalità specifiche che il sistema deve svolgere, mentre i requisiti non funzionali descrivono le caratteristiche del sistema come la performance, l'affidabilità e altri aspetti che non riguardano direttamente le funzionalità. Entrambi sono cruciali per garantire il successo del progetto.

2.1 Requisiti Funzionali

I requisiti funzionali delineano le funzionalità specifiche che il gioco deve fornire agli utenti. Alcuni requisiti funzionali per il nostro gioco includono:

– Apertura e chiusura della schermata di gioco

– Implementazione di un sistema di scelte interattive per il giocatore.

– Creazione di un sistema di dialogo coinvolgente con personaggi non giocabili.

– Integrazione di enigmi e prove per la risoluzione durante l'avventura.

– Creazione di un sistema per gestire lo stato del gioco, mantenendo traccia delle decisioni del giocatore, delle risorse acquisite e degli eventi chiave.
– Creazione di diverse trame narrative che portano a finali differenti.

2.2 Requisiti Non Funzionali

I requisiti non funzionali definiscono gli aspetti globali e trasversali di un sistema che influenzano le sue prestazioni, sicurezza, affidabilità e usabilità. Questi requisiti descrivono le caratteristiche che non sono strettamente legate alle funzionalità specifiche del software, ma piuttosto ne influenzano il comportamento e le prestazioni complessive. Di seguito sono specificati alcuni requisiti non funzionali per il gioco:

2.2.1 Affidabilità

Metrica	**Descrizione**	**Obiettivo**
Tempo di funzionamento	Percentuale di uptime nel sistema	$> 99.9\%$
Backup e ripristino	Tempo medio per il ripristino da un backup completo	< 1 ora
Log e tracciamento degli errori	Percentuale di eventi di sistema registrati nei log	$> 95\%$

Tabella 1: Affidabilità

2.2.2 Prestazioni

Metrica	**Descrizione**	**Obiettivo**
Caricamento del gioco	Tempo di caricamento	< 30 secondi
Utilizzo efficiente delle risorse	Utilizzo medio della CPU	$< 70\%$
Tempo di risposta	Tempo medio di risposta durante l'uso normale	< 1 secondo
Compilazione ed esecuzione del codice	Tempo medio di compilazione	< 30 secondi

Tabella 2: Prestazioni

2.2.3 Spazio

Metrica	Descrizione	Obiettivo
Utilizzo dello spazio su disco	Percentuale di spazio occupato su disco rigido	$< 80\%$
Efficienza di memoria	Utilizzo medio della memoria RAM	$< 70\%$
Velocità di accesso ai dati	Tempo medio di accesso ai dati su disco e database	< 10 millisecondi

Tabella 3: Spazio

2.2.4 Usabilità

Metrica	Descrizione	Obiettivo
Apprendimento delle meccaniche di gioco	Tempo per apprendere le dinamiche di gioco	< 15 minuti
Leggibilità del testo	Leggibilità' considerata buona o eccellente	$> 90\%$

Tabella 4: Usabilità

3 Progettazione

3.1 Architettura MVC

L'architettura Model-View-Controller (MVC) è il modello architetturale adottato per lo sviluppo del nostro progetto. Tale modello suddivide l'applicazione in tre componenti principali: il Modello (Model), la Vista (View) e il Controllore (Controller). Ognuno di questi componenti svolge un ruolo specifico all'interno del sistema.

- Il *Model* si occupa della gestione e dell'elaborazione dei dati, garantendo la coerenza e l'integrità del sistema. Non è consapevole di come i dati vengano presentati all'utente o di come l'utente interagisce con essi. La sua responsabilità principale è fornire una rappresentazione interna dei dati e rispondere alle richieste provenienti dal Controller.
- La *View* gestisce l'interfaccia utente e la presentazione dei dati provenienti dal Modello. Si occupa di visualizzare i dati in modo comprensibile all'utente. La View è separata dal Model e può essere aggiornata automaticamente ogni volta che i dati nel Model cambiano.
- Il *Controller* risponde agli eventi dell'utente, decide come gestire tali eventi e aggiorna il Modello e la Vista di conseguenza. Agisce come intermediario tra la View e il Model, gestendo la logica dell'applicazione e garantendo la coerenza tra i dati e la loro presentazione.

L'architettura Model-View-Controller (MVC) offre notevoli vantaggi nello sviluppo del software:

- **Separazione delle Preoccupazioni:** La divisione dei componenti favorisce la manutenzione agevole del codice, rendendolo flessibile ed estensibile.
- **Riutilizzabilità del Codice:** Grazie alla progettazione indipendente dei componenti, è possibile riutilizzare le componenti stesse in contesti diversi, migliorando la modularità complessiva.
- **Testabilità Semplificata:** La separazione dei ruoli facilita il testing indipendente di ciascun componente, migliorando l'efficacia dei test unitari e la robustezza complessiva del sistema.

3.2 Applicazione dell'MVC

La scelta dell'architettura Model-View-Controller (MVC) per il nostro progetto è stata guidata dalla volontà di adottare una struttura semplice e modulare in risposta alle esigenze specifiche del software.

Attraverso tale architettura, siamo stati in grado di suddividere chiaramente le responsabilità principali.

3.2.1 Model

La componente Model è stata implementata utilizzando classi Java, assets (sfondi, icone ecc...) e operazioni di input/output fornite da *java.io*.

3.2.2 View

La componente View è stata sviluppata utilizzando Java Swing, un package grafico per l'interfaccia utente in Java. Swing ci ha fornito una vasta gamma di componenti grafici per creare un'interfaccia utente intuitiva e interattiva come i *JButton*, *JFrame*, *JScrollPane* e molti altri .

3.2.3 Controller

La componente Controller è stata implementata utilizzando classi Java del package *java.util* e *java.awt*, insieme a operazioni batch. Le classi di *java.util* sono state utilizzate per gestire strutture dati e operazioni correlate, mentre *java.awt* è stato impiegato per manipolare gli eventi dell'interfaccia utente tramite l'uso degli ascoltatori.

3.2.4 Conclusioni e infografica

L'architettura MVC, con la sua suddivisione chiara delle responsabilità, si è dimostrata ideale per la gestione di un progetto di questa portata, semplificandone lo sviluppo, la manutenzione e l'estensione futura.

Di seguito, si riporta l'immagine esplicativa della nostra architettura.

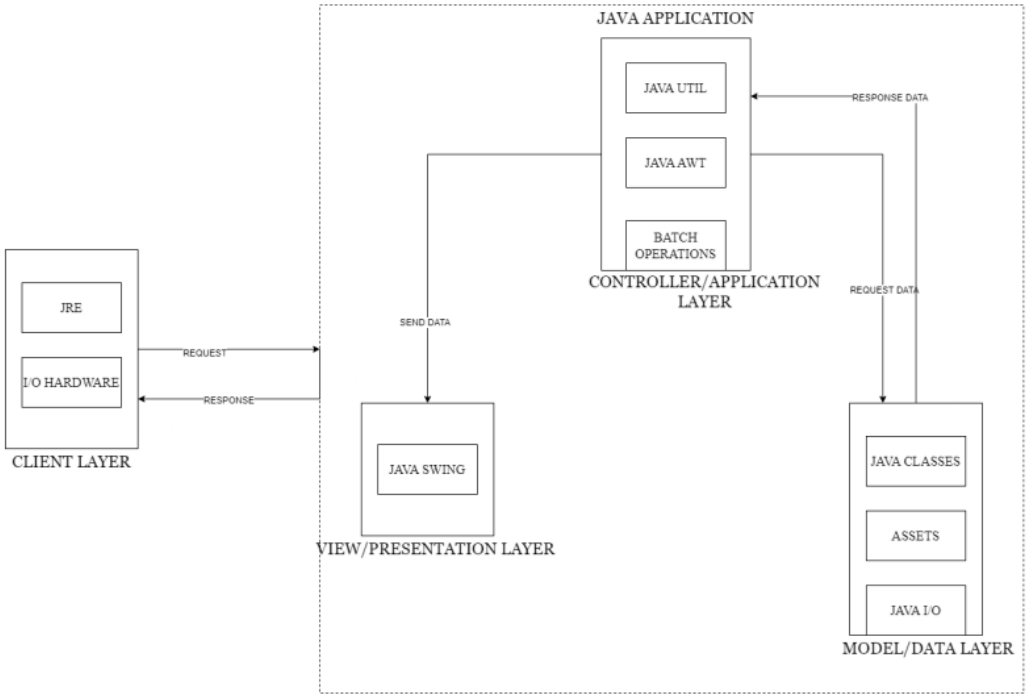

Figura 1: Architettura MVC

3.3 Design Patterns

Nel corso dell'analisi dettagliata delle User Story e dei requisiti di sviluppo del progetto, abbiamo individuato l'opportunità di impiegare alcuni Design Pattern significativi al fine di garantire una struttura solida e flessibile al nostro software. Tra i Design Pattern adottati, ricorriamo a:

- **Singleton**: Questo pattern è stato implementato per garantire che alcune classi del nostro sistema abbiano una sola istanza, fornendo un punto unico di accesso a tale istanza da qualsiasi punto del programma. L'utilizzo del Singleton è stato particolarmente rilevante nella creazione di oggetti di gioco critici e nella gestione di componenti grafiche.

- **Builder**: Il pattern Builder è stato sfruttato per la creazione degli oggetti complessi di gioco. Questo ci ha permesso di separare la creazione di un oggetto dalla sua rappresentazione, consentendo una costruzione flessibile e modulare. In particolare, il Builder è stato utilizzato nella generazione dei personaggi non giocabili, offrendo un'approccio strutturato e chiaro alla loro creazione.

- **Strategy**: Il Design Pattern Strategy è stato adottato per la definizione di alcune dinamiche di gioco, in particolare per la creazione di enigmi. Questo pattern permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili.

L'uso dello Strategy ci ha fornito un modo efficiente per gestire diverse strategie di risoluzione degli enigmi all'interno del nostro gioco.

L'impiego di questi Design Pattern non solo migliora la struttura del nostro codice, ma offre anche una maggiore flessibilità e manutenibilità durante lo sviluppo. La scelta accurata di questi pattern si è dimostrata essenziale per affrontare in modo efficiente le sfide specifiche poste dalle User Story e per garantire una base solida per il nostro progetto di sviluppo del gioco.

Vediamo adesso nello specifico dove sono stati applicati questi Design Pattern.

3.3.1 Creazione dei bottoni

Per la creazione dei bottoni del pannello di gioco abbiamo deciso di utilizzare il pattern Builder. Abbiamo dichiarato l'interfaccia *ButtonBuilder*, contenenti i metodi per settare rispettivamente dimensione, testo, colore di background, colore di foreground, font, opacità, bordo, ascoltatore e tool-tip. Oltre ad essi, sono presenti i metodi specifici del pattern, ovvero *build* e *reset*. Tutti i metodi sono stati opportunamente implementati dalla classe *ConcreteButtonBuilder*.

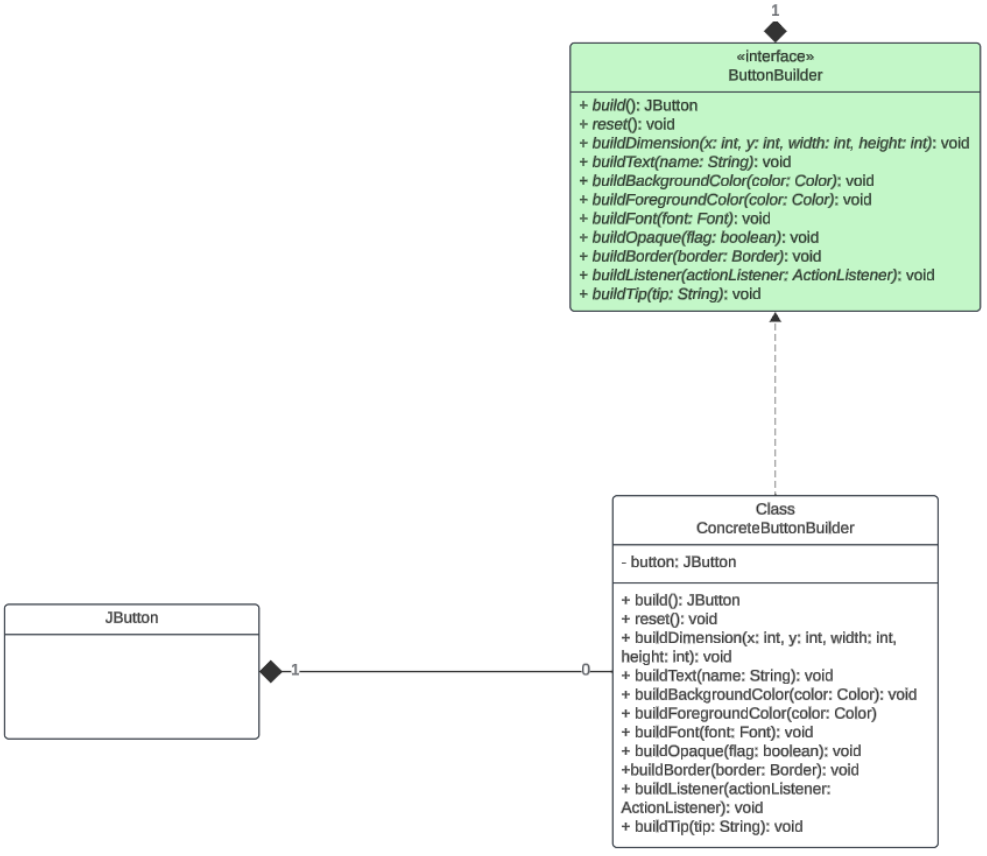

Figura 2: Button Builder

3.3.2 Creazione pannelli scrollabili

Anche per la creazione dei panelli scrollabili abbiamo optato per l'uso del pattern Builder. Nell'interfaccia *ScrollBuilder* sono dichiarati i metodi propri del pattern, *reset* e *builder*, e due metodi per settare rispettivamente il testo e le dimensioni. Tutti i metodi sono stati opportunamente implementati dalla classe *ConcreteScrollBuilder*.

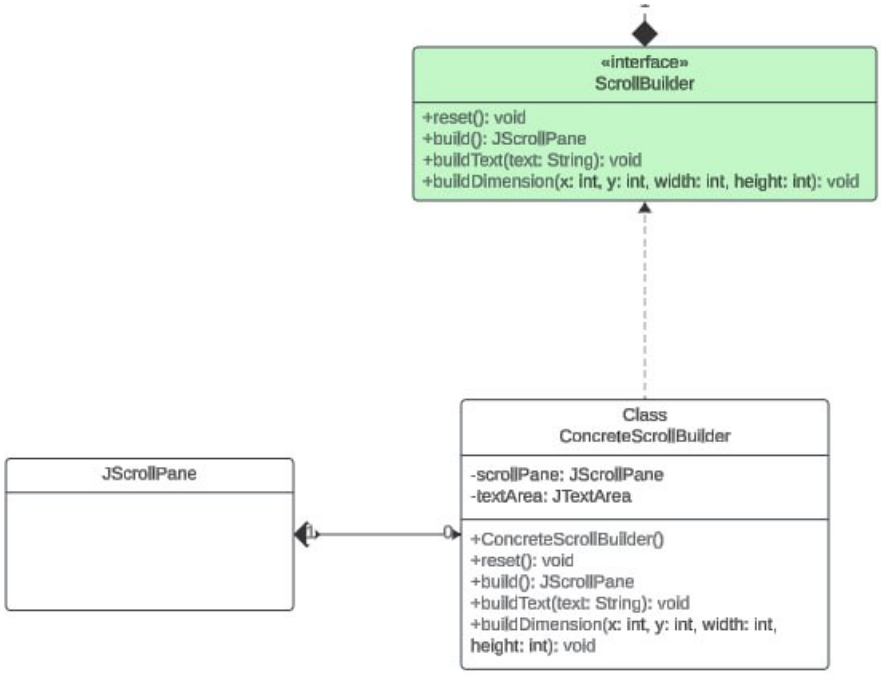

Figura 3: Scroll Builder

3.3.3 Creazione dell'inventario

Per la creazione dell'inventario si è deciso di utilizzare il pattern Singleton. Dunque, la classe *Inventory* contiene un attributo privato di tipo *Inventory* e un metodo *static getInventory* che costruisce la prima volta l'oggetto e lo assegna all'attributo mentre le successive volte restituisce la stessa istanza. Inoltre, *Inventory* contiene una lista di oggetti di tipo *Item*, e dei metodi per inserire, ottenere e usare essi.

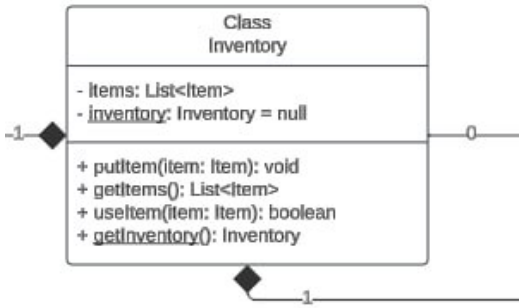

Figura 4: Inventory Singleton

3.3.4 Creazione degli oggetti

Per la creazione degli Item si è deciso di utilizzare il pattern Builder. La classe Item, che funge nel nostro caso da Product, possiede come attributi una stringa per il nome ed un'icona. I metodi di questa classe consentono rispettivamente di settare il nome, l'immagine, ottenere il nome, l'immagine. Vi sono anche gli override dei metodi *equals* e *toString*. L'interfaccia *ItemBuilder*, oltre ai metodi tipici del pattern *build* e *reset*, dichiara i metodi *buildName* e *buildImageIcon* per settare rispettivamente il nome e l'icona dell'oggetto di tipo Item da costruire. Tali metodi sono implementati nella classe *ConcreteItemBuilder*. Abbiamo utilizzato anche un *ItemDirector*, che possiede un attributo di tipo *ItemBuilder* inizializzato tramite il costruttore, un metodo *changeBuilder* per cambiare il tipo di builder concreto e infine i metodi *constructStone*, *constructBoard* e *constructLight* per costruire i tre item di nostro interesse.

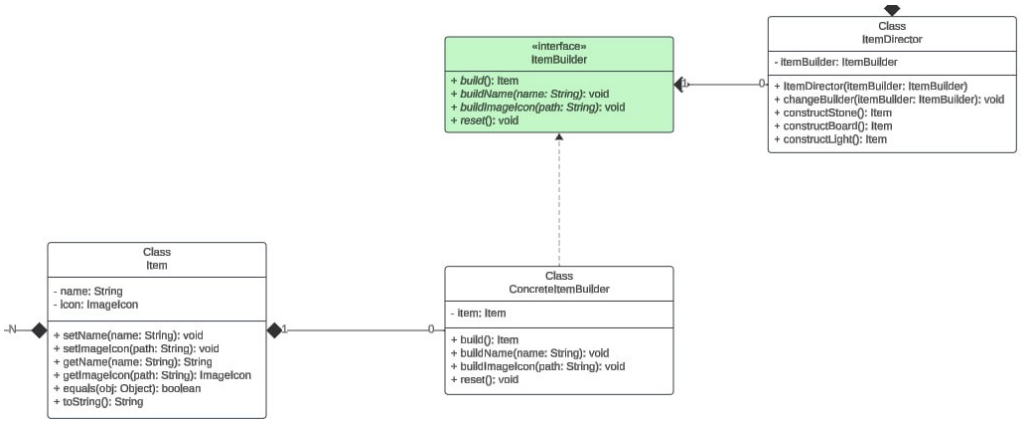

Figura 5: Item Builder

3.3.5 Creazione degli NPC

Per la creazione dei nostri tre NPC di interesse, ovvero Cedric, Lythien e Malgrim, abbiamo inanzitutto estratto una classe astratta NPC, che possiede come attributi due stringhe, una per il nome ed una per il path dell'immagine, un costruttore che prende a parametro queste due stringhe, e due metodi per ottenerle. I tre NPC vengono creati dalle tre rispettive classi che estendono NPC, e su di esse è stato applicato il pattern Singleton.

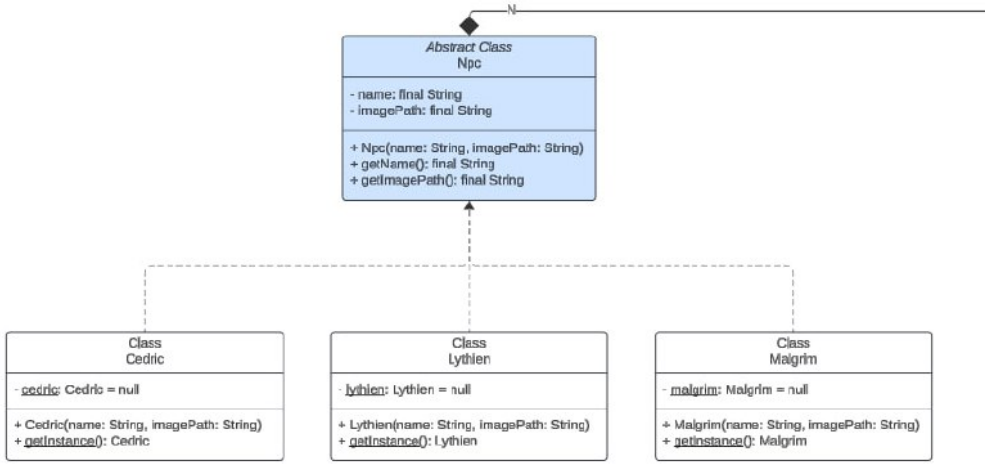

Figura 6: NPC Singleton

3.3.6 Creazione del giocatore

Per la costruzione del giocatore abbiamo ritenuto opportuno utilizzare il pattern Singleton. Innanzitutto abbiamo estratto una classe astratta *Player* avente come attributi due stringhe contenenti il nome e il path dell'icona, un oggetto di tipo *Inventory* e una lista di NPC, utilizzata poi successivamente per gestire le diramazioni verso gli stati finali. Oltre ad un costruttore che inizializza le due stringhe, questa classe contiene dei metodi *get* per ottenere i vari attributi, un metodo *npcFollow* per aggiungere un NPC sulla lista, e due metodi *getItem* e *useItem*, rispettivamente per ottenere ed usare gli item raccolti durante l'avventura. La classe concreta *Eren* estende tale classe astratta concretizzandone i metodi ed, inoltre, contiene gli attributi e i metodi propri del pattern Singleton.

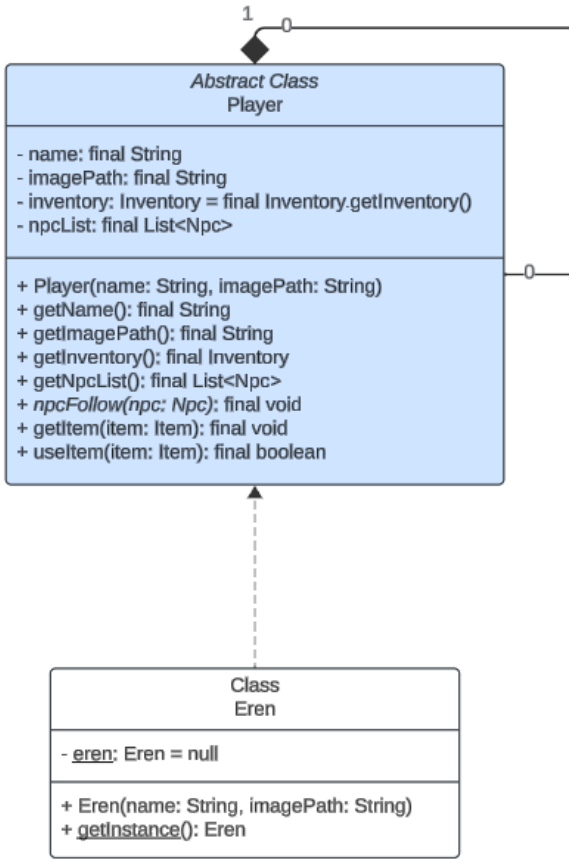

Figura 7: Player Singleton

3.3.7 Stato cripta

La creazione di questo stato contenente un'enigma è stata svincolata dal resto degli stati per la sua maggiore complessità. Per la classe *Crypt*, la quale coincide con questo stato, si è deciso di utilizzare il pattern Singleton. Oltre agli attributi e ai metodi propri di questo pattern, essa possiede come attributi un oggetto di tipo *Strategy* che rappresenta uno dei tre diversi enigmi che il giocatore può scegliere, una flag booleana per aprire o chiudere la cripta ed una stringa contenente un messaggio. I metodi di tale classe permettono rispettivamente di settare il tipo di enigma, settare ed ottenere il messaggio ed aprire la cripta. Per la scelta dell'enigma si è scelto di utilizzare invece il pattern Strategy. Abbiamo estratto una classe astratta Strategy contenente per attributo una stringa che rappresenta il tipo di enigma, e per metodi uno per ottenere l'enigma ed uno per eseguirlo. Questa classe astratta è estesa delle tre classi *ConcretePalindromic*, *ConcreteHistory* e *ConcretePrimes* che implementano il metodo *execute*, il quale prende a parametro una stringa e restituisce un valore di verità in base alla risposta inserita dall'utente. Su queste

tre classi concrete si è deciso di applicare il pattern Singleton.

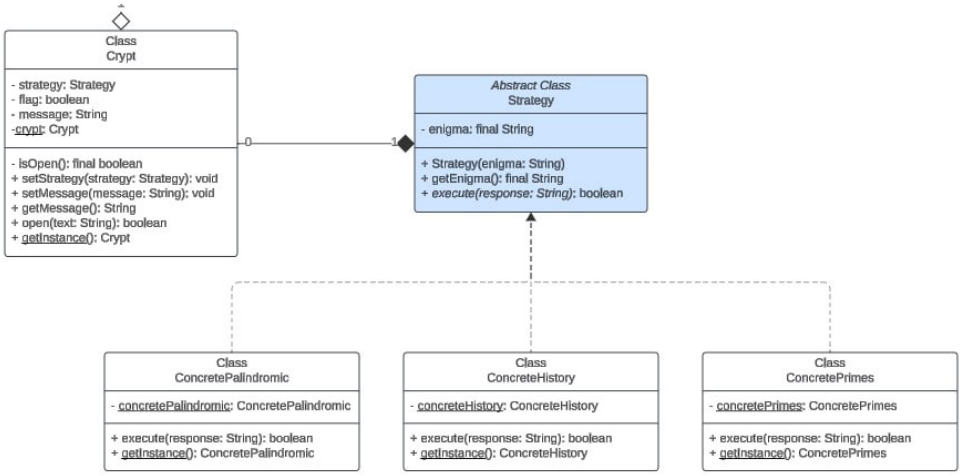

Figura 8: Crypt Strategy

3.4 Testing

3.4.1 Introduzione ai Test

I test rappresentano un elemento fondamentale nel processo di sviluppo del software, fornendo un mezzo affidabile per verificare la correttezza e la robustezza del nostro programma. L'obiettivo principale dei test è identificare e correggere gli errori nel codice, riducendo così la probabilità di errori durante l'esecuzione del software. Questa pratica è essenziale per garantire la qualità del gioco e la soddisfazione del giocatore che non si imbatte in possibili bug.

3.4.2 JUnit

Nell'implementazione del nostro progetto abbiamo scelto di utilizzare JUnit come framework di testing per il linguaggio di programmazione Java. La scelta è dovuta alla sua semplicità nell'organizzare i test in classi di test, di eseguirli in modo indipendente e di fornire i rispettivi output.

3.4.3 Struttura dei Test

Nel nostro approccio ai test all'interno delle classi seguiamo una struttura ben definita che coinvolge la gestione delle precondizioni(facoltativa) e l'esecuzione di test.

- **PRECONDIZIONE**. Utilizziamo il metodo *setUp* annotato con *@BeforeEach* per eseguire operazioni mirate a garantire la veridicità delle precondizioni. Questo metodo viene eseguito prima di ogni caso di test all'interno della classe. Successivamente vengono utilizzate le asserzioni(espressioni booleane) per verificare la precondizione. Come asserzione abbiamo utilizzato: *assertNotNull(Object object, String message);*

per verificare che un determinato oggetto non sia nullo. Il metodo accetta due parametri: l'oggetto che si desidera verificare e un messaggio opzionale che verrà visualizzato in caso di fallimento dell'asserzione.

- **TEST**. Il test, preceduto da *@Test*, riguarda l'esecuzione del metodo da testare con gli eventuali parametri di input relativi a quel caso di test. Come asserzione abbiamo utilizzato: *assertTrue(boolean condition)* che verifica se l'espressione fornita tra parentesi è *true*. Se l'espressione è *true*, il test ha successo; altrimenti il test fallisce

Di seguito sono riportati due esempi di test eseguiti all'interno del nostro progetto software.

```
class PlayerTest {
    Inventory inventory;
    ItemDirector itemDirector;
    Item item;
    Player eren;

    @BeforeEach
    void setUp() {
        inventory = Inventory.getInventory();
        itemDirector = new ItemDirector(new ConcreteItemBuilder());
        item = itemDirector.constructStone();
        eren = Eren.getInstance();
        eren.getItem(item); //Put the item in to the Inventory
        assertNotNull(inventory, "Non è stato possibile istanziare
            un oggetto di tipo Inventory");
        assertNotNull(itemDirector, "Non è stato possibile istanziare
                un oggetto di tipo ItemDirector");
        assertNotNull(item, "Non è stato possibile istanziare
            un oggetto di tipo Item");
        assertNotNull(eren, "Non è stato possibile istanziare
            un oggetto di tipo Player");
    }

    @Test
    void useItem() {
        assertTrue(eren.useItem(item));
    }
}

class ConcretePalindromicTest {
    @BeforeEach
    void setUp() {
        ConcretePalindromic concretePalindromic =
            ConcretePalindromic.getInstance();
        assertNotNull(concretePalindromic, "Non è stato possibile
            stanziare un oggetto di tipo " +
```

```
                "ConcretePalindromic");
    }

    @org.junit.jupiter.api.Test
    void execute() {
        String palindromicWordsNumber = "3";
        assertTrue(ConcretePalindromic.
            getInstance().
            execute(palindromicWordsNumber));
    }
}
```

4 Metodologia Agile

4.1 Scrum

Per lo sviluppo di questo progetto, abbiamo adottato la *Metodologia Agile*, focalizzandoci sul framework *Scrum*. Attraverso le diverse fasi di Scrum, il team ha saputo affrontare positivamente le sfide emergenti, consentendo una gestione flessibile del progetto.

4.1.1 Definizione delle User Stories e delle Tasks

Inizialmente, abbiamo definito in dettaglio le User Stories, ognuna rappresentante un'unità di lavoro significativa. Successivamente, suddividendo tali storie in Tasks più gestibili, abbiamo agevolato la distribuzione equa del lavoro tra i membri del team.

4.1.2 Assegnazione delle Priorità e degli Story Points

Ad ogni User Story è stata assegnata una priorità, fase cruciale per garantire l'implementazione delle funzionalità critiche iniziali. Successivamente, utilizzando la tecnica del Planning Poker, abbiamo stimato collaborativamente gli Story Points, fornendo così una base per la pianificazione futura.

4.1.3 Creazione del Product Backlog

Dopo aver definito gli Story Points, abbiamo creato il *Product Backlog*, consentendoci di avere una visione chiara e completa del lavoro complessivo in anticipo.

4.1.4 Sprint Planning e Sprint Backlog

In questa fase, il team ha selezionato le User Stories prioritarie dal Product Backlog, stimato i tempi e pianificato il lavoro da svolgere durante lo Sprint. Abbiamo organizzato lo sviluppo in due Sprint, ciascuno della durata di due settimane. Definendo lo *Sprint Backlog*, contenente tutte le User Stories e le relative Tasks assegnate a ciascun membro del team, abbiamo garantito che queste fossero completate entro la fine di ciascuno Sprint.

4.2 Product Backlog

La costruzione del Product Backlog è avvenuta immediatamente dopo l'assegnazione delle priorità e degli Story Points. Questo ci ha permesso già dal primo momento di capire quali fossero le User Stories e le relative Tasks più rilevanti che avremmo dovuto portare avanti.

Alla fine, il Product Backlog presentava una struttura piuttosto chiara e organizzata, basata sui livelli di priorità assegnati.

Priorità	**User Story**	**Story Points**
1	As a player, I want to access the game menu so that I can play the game	2 SP
1	As a player I want to decide whether to collect the stone or not so that I can continue the adventure	8 SP
1	As a player I want to be able to choose the puzzle so that I can know the question	8 SP
1	As a player I want to enter the answer so that I can see if it's right or wrong	3 SP
1	As a player I want to choose the plate so that I can go on	13 SP
2	As a player I want to choose whether to go left or right so that I can continue the adventure	2 SP
2	As a player I want to cross the river so that I can continue the adventure	5 SP
2	As a player I want to be able to cross the river via the wooden board	3 SP
2	As a player I want to come back in the previous state so that I can pick up the wooden board	3 SP

Tabella 5: Product Backlog

Priorità	User Story	Story Points
2	As a player I want to enter the answer for the first option so that I can see if it's right or wrong	5 SP
3	As a player I want to choose whether or not to pick up the wooden board so that I can use it in the future	3 SP
3	As a player I want to choose one possibility so that I can continue the adventure	3 SP
4	As a player I want to grab the torch so that I can light the crypt	2 SP
4	As a player I want to put the stone on top of the statue so that I can defeat Sauron	2 SP
5	As a player I want to be able to go and talk to Cedric to understand my mission	2 SP
5	As a player I want to help the elf so that I can continue the adventure	2 SP
5	As a player I want to don't help the elf so that I can continue the adventure	1 SP
5	As a player I want to press the button so that I can continue the adventure	1 SP
5	As a player I want to press the button so that I can answer to Shadow Malgrim	1 SP
5	As a player I want to press the button so taht I can exit the crypt	1 SP
5	As a player I want to continue in the Forest of Shrouded Spirits so that I can continue the adventure	1 SP
5	As a player I want to go towards the statue so that I can put the stone on top of it	1 SP
5	As a player I want to use the stone so that I can defeat Sauron	1 SP

Tabella 6: Product Backlog

4.3 Primo Sprint

Durante il primo sprint, che si è svolto nel periodo dal 01/12/2023 al 18/12/2023, il focus principale del team è stato rivolto allo sviluppo delle fondamentali meccaniche di gioco e all'implementazione dei design pattern selezionati. Questa fase iniziale ha richiesto uno sforzo considerevole, ma ha garantito una base solida e ben strutturata per il proseguimento del progetto. La decisione di concentrarci su queste componenti critiche sin dall'inizio si è rivelata strategica, permettendoci di affrontare con successo le successive parti di sviluppo.

Di seguito sono riportate le *User Story*, con relative *Tasks* e ore di lavoro, che sono state portate a termine durante questo Sprint.

User Story	**Tasks**	**Hours**
As a player I want to access the game menu so that I can play the game	Create the menu Create starting button Create Builder for the buttons Create the player	6 0.5 1 2
As a player I want to be able to choose the puzzle so that I can know the question	Create the button to choose the first puzzle Create the button to choose the second puzzle Create the button to choose the third puzzle Create a push-up window	3 3 3 1
As a player I want to be able to cross the river via the wooden board	Create the button to cross the river Search the object in the inventory Create a push-up window	2 2 1
As a player I want to be able to go and talk to Cedric to understand my mission	Create Cedric Create a push-up window	3 1.5
As a player I want to choose one possibility so that I can continue the adventure	Create the button to choose the first option Create the button to choose the second option Create Malgrim Create a push-up window	3 3 2 3
As a player I want to choose whether or not to pick up the wooden board so that I can use it in the future	Create the button to collect the board Create the button to not collect the board Create a push-up window Put the object in to the inventory	2 2 3 0.2
As a player I want to choose whether to go left or right so that I can continue the adventure	Create the button to go left Create the button to go right Create a push-up window	3 3 3
As a player I want to come back in the previous state so that I can pick up the wooden board	Create the button to come back	3
As a player I want to continue in the Forest of Shrouded Spirits so that I can continue the adventure	Create the button to enter the forest Create a push-up window	2 1.5
As a player I want to cross the river so that I can continue the adventure	Create a text input window Create a button to cross the river Create a button to try again the puzzle Create a push-up window	2 3 3 1

User Story	Tasks	Hours
As a player I want to decide whether to collect the stone or not so that I can continue the adventure	Create the button to collect the stone	2
	Create the button to not collect the stone	2
	Create a push-up window	3
	Create the inventory	4
	Open and close the inventory	4
	Create Builder for the states	2
	Create Builder for the panels	2
As a player I want to don't help the elf so that I can continue the adventure	Create the button to don't help the elf	3
	Create a push-up window	3
As a player I want to enter the answer so that I can see if it's right or wrong	Create Strategy for the crypt	3
	Create a push-up window	1
	Create a text input window	2
	Create Singleton for crypt and strategies	1
As a player I want to grab the torch so that I can light the crypt	Create the button to collect the light	3
	Put the object in to the inventory	4
	Create a push-up window	2
As a player I want to help the elf so that I can continue the adventure	Create the button to help the elf	3
	Create Lythien	3
	Create a push-up window	1
As a player I want to press the button so that I can continue the adventure	Create the button to go on	2
	Create a push-up window	1

Tabella 7: Sprint Backlog 1

4.4 Secondo Sprint

Durante il secondo sprint, che ha coperto il periodo dal 19/12/2023 al 05/01/2024, abbiamo potuto dedicare più risorse allo sviluppo dei finali basati sulle scelte dei giocatori, perfezionare gli aspetti grafici, e concentrarci sull'enigma delle placche, un elemento cruciale nella trama del gioco.

In sintesi, il secondo sprint ha rappresentato una fase snella in cui ci siamo concentrati sulla rifinitura di elementi chiave grazie ai benefici ottenuti dal primo sprint.

Di seguito sono riportate le *User Story*, con relative *Tasks* e ore di lavoro, che sono state portate a termine durante questo Sprint.

User Story	Tasks	Hours
As a player I want to choose the plate so that I can go on	Create the plate puzzle	12
As a player I want to enter the answer for the first option so that I can see if it's right or wrong	Create the JComboBoxes Create a push-up window	3 1
As a player I want to go towards the statue so that I can put the stone on top of it	Create the button to go towards the statue Create a push-up window	3 1
As a player I want to press the button so that I can answer to Shadowy Malgrim	Create the button to answer Create a push-up window	3 1
As a player I want to press the button so that I can exit the crypt	Create the button to exit the crypt Create a push-up window	3 1
As a player I want to put the stone on top of the statue so that I can defeat Sauron	Create a text window with the final Create a push-up window	4 1
As a player I want to use the stone so that I can defeat Sauron	Create the button to take the stone Create a push-up window	1 3
As a player I want to use the stone so that I can defeat Sauron	Create a push-up window Create a text window with the final tips	1 1
As a player I want to use the stone so that I can defeat Sauron	Create a text window with the final tips Create a push-up window	4 1

Tabella 8: Sprint Backlog 2

5 Conclusioni

Durante lo sviluppo di questo progetto, attraverso l'applicazione di metodologie agili, come Scrum, abbiamo sperimentato una gestione dinamica del lavoro che ha favorito la collaborazione e la flessibilità. La scelta di Design Pattern, quali *Singleton*, *Builder* e *Strategy*, ha giocato un ruolo chiave nella strutturazione del nostro codice, contribuendo a rendere il progetto più modulare, mantenibile e coeso.

Durante il percorso, abbiamo affrontato sfide significative, che spaziavano dalla definizione delle User Stories alla risoluzione di problemi complessi legati agli enigmi e alle dinamiche di gioco. Il progresso graduale attraverso gli sprint ha consentito di gestire la complessità in modo incrementale, facilitando l'evoluzione del progetto.

Il secondo sprint, caratterizzato da un lavoro più focalizzato sugli aspetti finali del gioco, ha evidenziato l'importanza della pianificazione e dello sviluppo progressivo. La coerenza e l'efficacia della progettazione iniziale hanno notevolmente facilitato l'implementazione degli ultimi dettagli e la creazione di finali diversificati in base alle scelte del giocatore.

In conclusione, questo progetto ha offerto l'opportunità di applicare concretamente le conoscenze acquisite durante il corso. Attraverso le sfide e le decisioni prese, abbiamo consolidato la comprensione dei processi di sviluppo software, contribuendo al nostro percorso formativo.