Mind Escape

Relazione per il progetto di *Programmazione ad Oggetti*

Filippo Greppi, Elena Fucci, Spaccini Ettore, Spagnoli Marcello

16 febbraio 2025

Indice

Capitolo 1

Analisi

1.1 Descrizione e requisiti

Il software mira allo sviluppo del videogioco "Mind-Escape", un'avventura psicologica e di suspense. Il giocatore assume il ruolo di un personaggio che si sveglia in una stanza sconosciuta, senza memoria di come ci sia arrivato. L'obiettivo del gioco è fuggire da questa prigionia, risolvendo una serie di enigmi che si presentano in diverse stanze. Ogni enigma risolto porta il giocatore più vicino alla vittoria, aprendo la via verso la stanza successiva. Per avanzare, il giocatore deve raccogliere e utilizzare vari oggetti, che vengono gestiti attraverso un inventario. Questi oggetti sono essenziali per risolvere enigmi e accedere a nuove stanze. La partita si concluderà quando il giocatore raggiungerà e risolverà l'enigma finale, riuscendo a fuggire dall'ultima stanza.

Requisiti funzionali

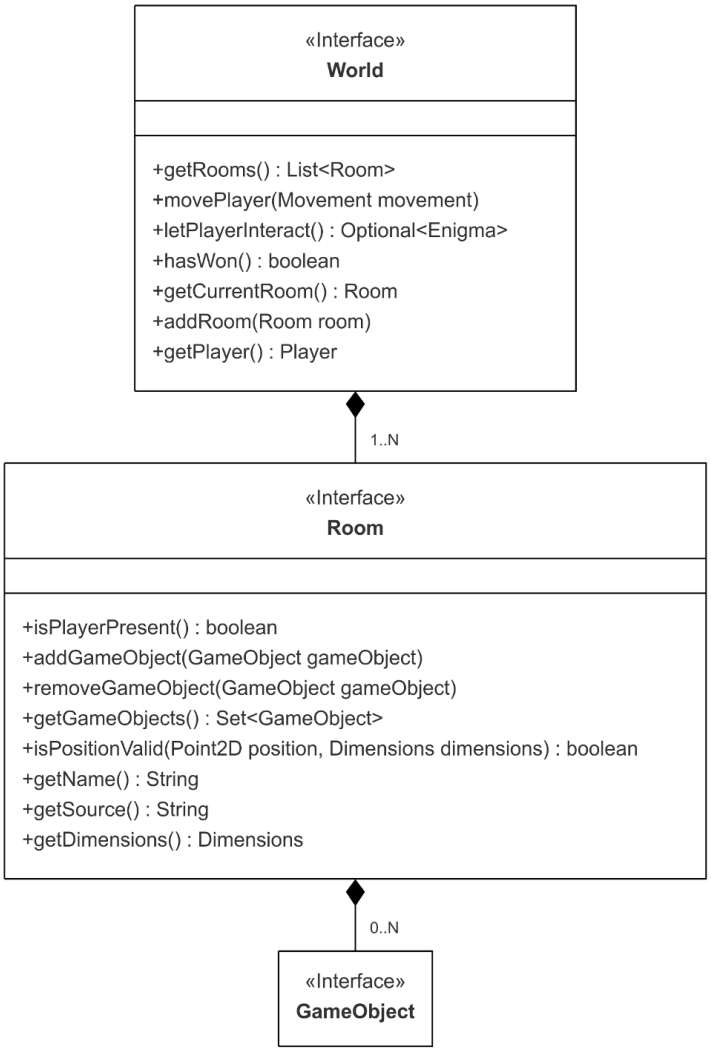
- Il personaggio deve poter muoversi nella mappa, gestendo le collisioni con ostacoli e pareti
- Il gioco deve permettere al personaggio di interagire con gli oggetti nelle stanze (es. raccogliere, esaminare, usare).
- Il personaggio deve disporre di un inventario che permetta di gestire gli oggetti raccolti.
- Il sistema deve verificare automaticamente la correttezza delle soluzioni degli enigmi.

Requisiti non funzionali

- Salvataggio dello stato della partita
- Deve essere garantita una modularità del codice per consentire l'aggiunta futura di nuove stanze o enigmi.
- Ridimensionabilità della finestra di gioco

1.2 Modello del Dominio

Mind Escape è un gioco ambientato in un mondo suddiviso in più stanze, attraverso le quali il giocatore può muoversi ed interagire con vari oggetti. L'avventura ha inizio in una stanza iniziale, da cui il giocatore potrà accedere progressivamente alle altre stanze. Il passaggio tra le stanze avviene mediante porte, che possono essere sbloccate risolvendo enigmi o utilizzando oggetti raccolti durante il gioco. L'obiettivo finale è raggiungere l'ultima stanza e interagire con uno specifico oggetto, evento che determina la vittoria.

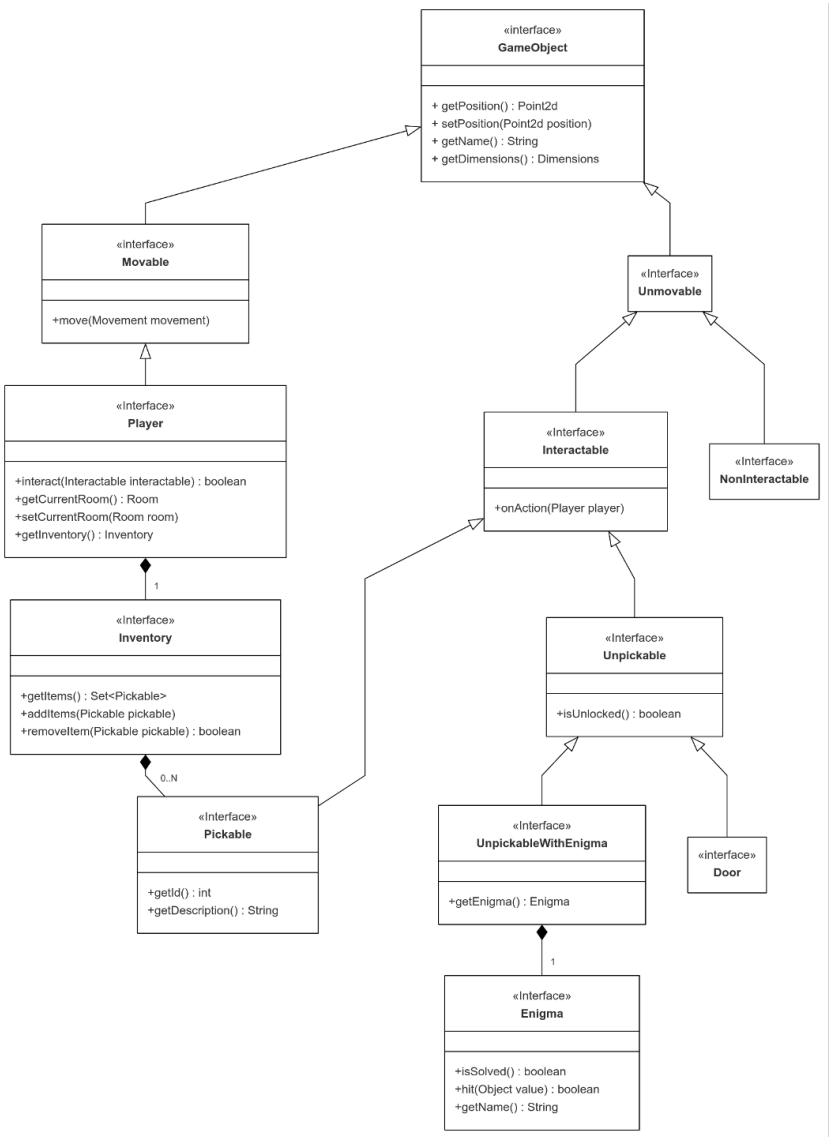


All'interno di ogni stanza possono essere presenti oggetti di gioco (GameObject), con i quali il giocatore può collidere. Gli oggetti di gioco si suddividono in due categorie principali:

- Oggetti interagibili (Interactable): si distinguono in due sottocategorie:
  - Oggetti raccoglibili (Pickable): possono essere raccolti dal giocatore ed inseriti nell'inventario, dove potranno essere utilizzati successivamente.
  - Oggetti non raccoglibili (Unpickable): non possono essere raccolti, ma possono essere sbloccati tramite determinate azioni. Alcuni di essi possono essere associati alla risoluzione di un enigma.

L'interazione tra il giocatore e un oggetto interagibile può avvenire esclusivamente a seguito di una collisione. In seguito all'interazione, l'oggetto esegue un'azione predefinita che determina il suo comportamento e le conseguenze dell'interazione.

- Oggetti non interagibili (NonInteractable): questi oggetti hanno una funzione puramente estetica o di ambientazione, e non influenzano direttamente il gameplay. Tuttavia, il giocatore può collidere con essi.



Capitolo 2

Design

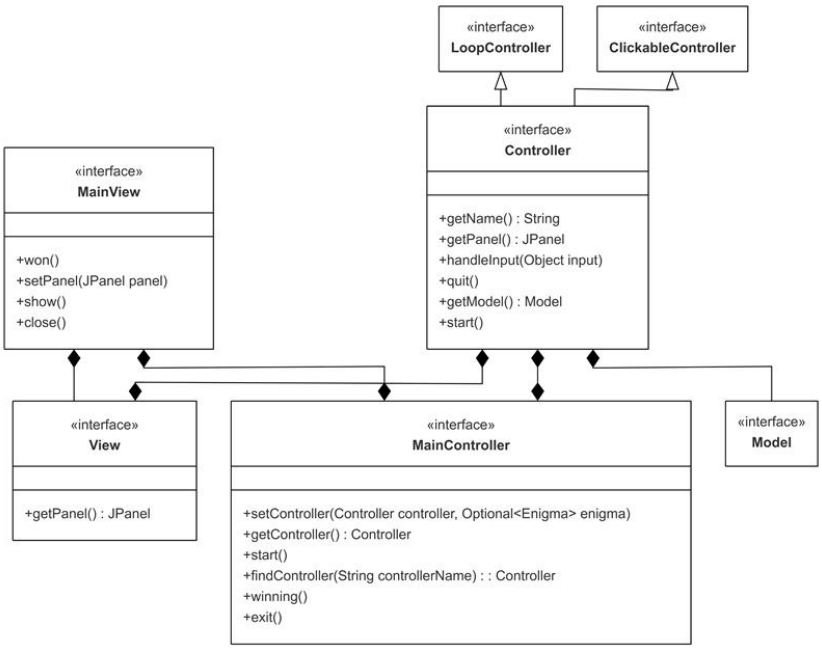
2.1 Architettura

L'architettura di Mind-Escape segue il pattern architetturale MVC (Model-View-Controller), ma si compone di più strutture MVC modulari. Il sistema implementa un'interfaccia chiamata MainController, che ha il compito di cercare e attivare i controller appropriati in base alle azioni dell'utente. I controller disponibili sono quelli associati ai vari MVC, e sono suddivisi in due categorie principali:

- LoopController: Gestisce gli MVC che richiedono un aggiornamento costante sia dal punto di vista logico che grafico. L'esecuzione di questi aggiornamenti può essere avviata o interrotta in base alle decisioni del MainController.
- 2. ClickableController: Gestisce gli MVC che necessitano di un aggiornamento sia logico che grafico, ma solo in seguito a un'interazione esplicita dell'utente, come l'uso del mouse o la tastiera.

Ogni controller è in grado di notificare il MainController riguardo al passaggio al controller successivo. Il MainController è responsabile della creazione e gestione dei controller necessari, ottimizzando l'uso della memoria in base alle esigenze specifiche. Questo approccio consente di aggiungere un numero arbitrario di controller senza compromettere la struttura dell'applicazione. Inoltre, Mind-Escape supporta la registrazione di input e output all'interno delle view di ciascun MVC. Gli input rappresentano nuove informazioni inviate ai controller, che possono includere comandi da mouse, tastiera, o l'inserimento di password. Questi input sono generati dall'utente e possono scatenare eventi che modificano il controller corrente. Una volta ricevuti gli input, il MainController aggiorna la view corretta in output, permettendo

un'interazione dinamica e fluida. In sintesi, l'aggiunta di nuovi Model, Controller e View non richiede modifiche alla struttura centrale dell'applicazione, garantendo così una scalabilità e una modularità efficaci.



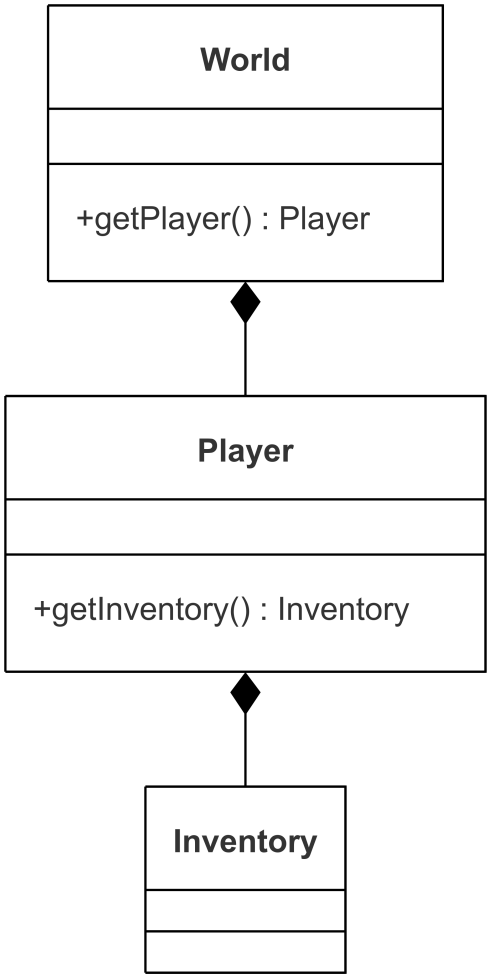
2.2 Design dettagliato

2.2.1 Elena Fucci

Problema Nel gioco Mind-Escape, quando un giocatore raccoglie degli oggetti, questi devono essere visualizzati all'interno di un inventario, che deve poter essere modificato durante il gioco.
La sfida è quella di garantire che l'inventario possa essere aggiornato dinamicamente e salvato in modo persistente, affinchè le modifiche possano essere recuperate anche in sessioni di gioco future.

Soluzione Il problema viene gestito tramite la creazione di un'interfaccia Player che si compone di un'interfaccia Inventory. L'inventario può essere aggiornato in tempo reale quando il giocatore raccoglie oggetti.
Inoltre, il Player è direttamente collegato al mondo collegato a sua volta ad un file di salvataggio, il che consente di salvare lo stato corrente del giocatore, compreso l'inventario, su un file. In questo modo, il gioco è in grado di man-

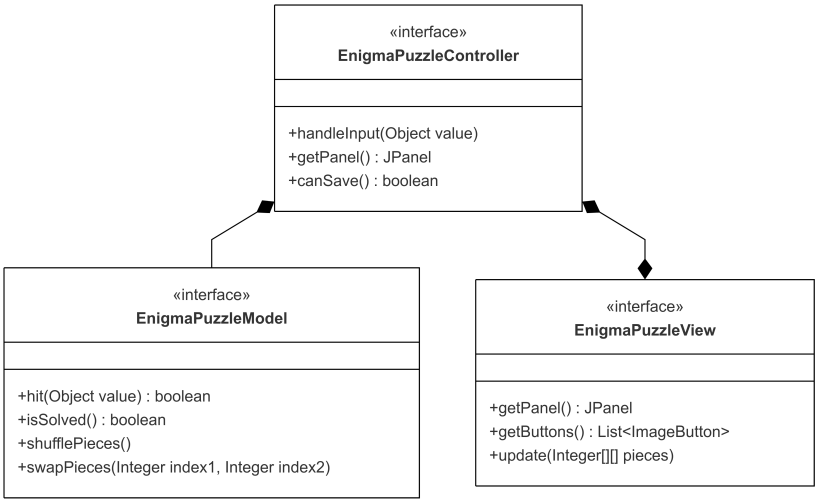
tenere le modifiche correnti all'inventario e ripristinarle quando il giocatore riprende la partita.



Problema All'interno del gioco sono presenti vari enigmi. Uno degli enigmi da risolvere è un puzzle. Per implementare questa parte, è necessario permettere al gioco di passare da una vista principale a una vista dedicata al puzzle, in modo che il giocatore possa interagire con il puzzle senza interruzioni.

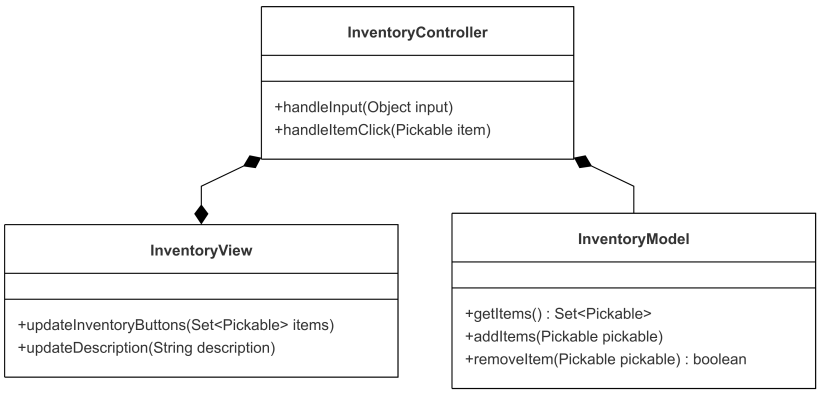
Soluzione Per risolvere questo problema, ho adottato il pattern MVC (Model-View-Controller). Questo approccio consente al mainController di trasferire temporaneamente il controllo al controller dell'enigma, che si occupa di gestire la logica del puzzle e di visualizzare la vista del puzzle corri-

spondente. L'uso del pattern consente di separare chiaramente la logica del gioco dalla parte di view rendendo il codice più modulare. Poichè modifiche alla logica del puzzle non influenzano la vista, e viceversa, il che facilita l'aggiornamento o l'aggiunta di nuove funzionalità.



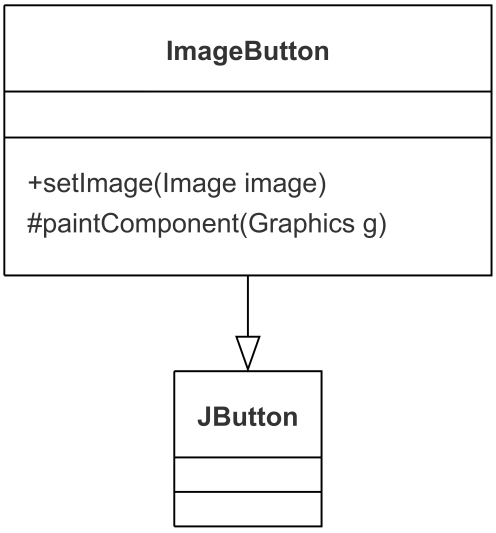
Problema Nel gioco, l'inventario è utilizzato per visualizzare gli oggetti raccolti e alcuni indizi. La gestione dell'inventario deve permettere una visualizzazione chiara e aggiornata durante il gioco. Questo richiede una separazione della logica di gestione degli oggetti e dei dati dalla loro visualizzazione, al fine di garantire una gestione efficiente e flessibile dell'inventario.

Soluzione Per risolvere il problema, è stato adottato il pattern MVC (Model-View-Controller). In questo schema, la logica di gestione dell'inventario è separata dalla sua visualizzazione, permettendo una chiara distinzione tra i dati (modello), l'interfaccia utente (vista) e il controllo (controller). Questa separazione consente di isolare le modifiche in una singola componente senza influire sulle altre.Poiché ogni componente ha un ruolo chiaro e definito, è più facile mantenere e aggiornare il codice, aggiungere nuove funzionalità o migliorare quelle esistenti diventa più semplice senza rischio di causare conflitti tra le diverse logiche.



Problema All'interno del gioco sia nell'inventario che nell'enigma del puzzle era necessario che le immagini fornite si adattassero alla dimensione del pulsante (JButton) in cui erano contenute.

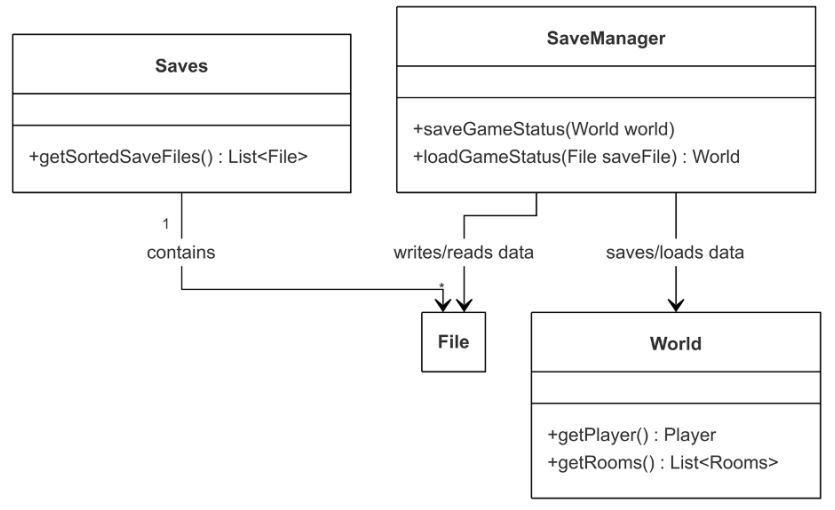
Soluzione Per risolvere questo problema, ho creato la classe ImageButton. Questa classe estende JButton e gestisce il ridimensionamento delle immagini in modo che si adattino automaticamente alla dimensione del pulsante. Ho dunque utilizzato questa classe sia nell'inventario che nel puzzle per ridimensionare le immagini direttamente all'interno dei pulsanti.In questo modo, è stato possibile limitare la duplicazione del codice rispettando il principio DRY. Inoltre centralizzando la logica del ridimensionamento delle immagini in una classe specifica, la manutenzione del codice diventa più semplice.



2.2.2 Filippo Greppi

Problema Salvataggio e caricamento dello stato di gioco

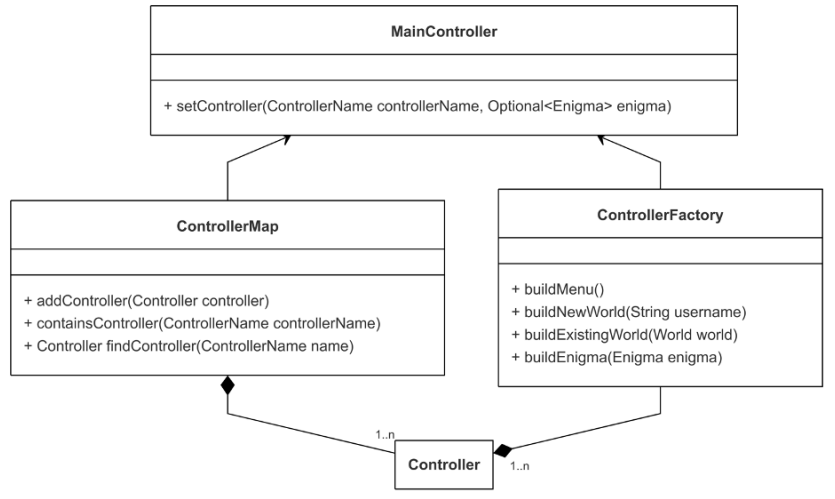
Soluzione Il sistema di salvataggio e caricamento utilizza la serializzazione per convertire lo stato del gioco in un formato salvabile su file. Quando il giocatore decide di salvare la partita, il SaveManager prende l'istanza attuale di World che contiene tutte le informazioni sullo stato del gioco, seleziona i dati necessari, e li scrive su file. Quando il giocatore vuole riprendere una partita salvata, il SaveManager legge i dati dal file selezionato ricreando il mondo. Per facilitare la gestione dei salvataggi, la classe Saves permette di ottenere una lista ordinata dei file disponibili, così da poter scegliere facilmente quale caricare.



Problema Nel contesto del progetto, il controller principale (MainController) deve gestire dinamicamente diversi componenti del gioco, tra cui menu, enigmi, mondo, e altre sezioni del gioco. Ogni componente del gioco è gestito da un controller separato, ma la creazione e la gestione di questi controller può risultare complessa e propensa a errori se non organizzata correttamente.

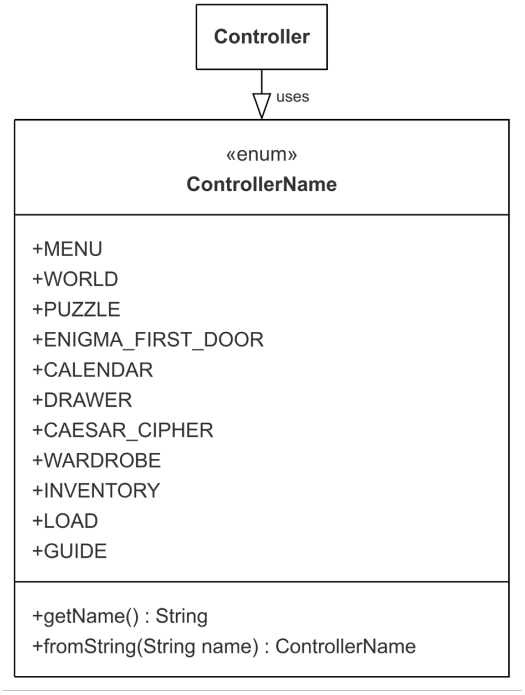
Soluzione La gestione dinamica dei controller è stata implementata utilizzando una combinazione del Factory Pattern e della Controller Map. La classe ControllerFactory funge da punto centrale per la creazione di tutti i controller necessari al gioco. In questo modo, ogni controller viene creato tramite la factory, evitando la ripetizione della logica di creazione in più punti del codice. Se in futuro fosse necessario aggiungere un nuovo controller

basterebbe aggiungerlo alla factory. Per facilitare la gestione dei controller durante l'esecuzione del gioco, ho utilizzato una Controller Map, una mappa che tiene traccia di tutti i controller attivi. Ogni controller è associato a un nome univoco, che permette di recuperarlo rapidamente quando necessario. L'integrazione tra la ControllerFactory e la ControllerMap è cruciale per il corretto funzionamento del sistema. Quando un controller è richiesto, se non gia presente, la factory lo crea e lo aggiunge alla mappa dei controller.



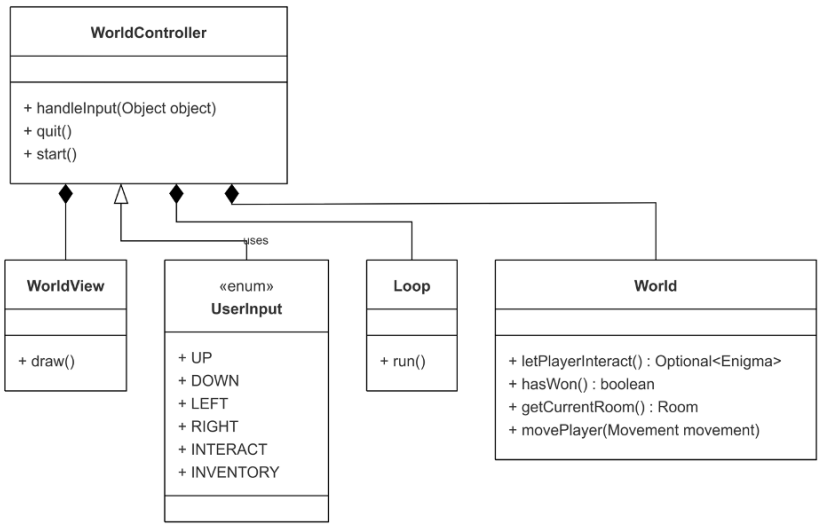
Problema Identificare i controller diversi all'interno di tutto il sistema.

Soluzione Per identificare i vari controller all'interno del sistema, ho utilizzato l'enumerazione ControllerName, che rappresenta i nomi dei controller e le loro stringhe corrispondenti. Ogni costante dell'enum è associata a una stringa unica che descrive il controller corrispondente, consentendo una gestione centralizzata e semplificata dei controller.



Problema Garantire un'esperienza di gioco fluida all'interno del mondo principale

Soluzione Il WorldController segue un approccio reattivo e dinamico, garantendo una risposta immediata agli input dell'utente senza introdurre ritardi o interruzioni nell'esperienza. Il controller utilizza un game loop a 60 FPS, regolando il tempo di esecuzione per mantenere costante la fluidità. Il ciclo principale verifica lo stato del gioco, controlla se il giocatore ha vinto, gestisce gli input e aggiorna la visualizzazione. Grazie a questa struttura, il giocatore percepisce un movimento naturale e senza scatti. Il sistema distingue tra movimenti direzionali (UP, DOWN, LEFT, RIGHT) e azioni specifiche come interagire con oggetti o aprire l'inventario, permettendo al giocatore di esplorare l'ambiente senza interruzioni. Inoltre, se un'azione come INTERACT o INVENTORY viene eseguita, il loop evita input multipli indesiderati.



2.2.3 Ettore Spaccini

Problema Creazione delle porte con differenti meccanismi di sblocco. Nel nostro gioco Mind Escape, le porte possono avere tre stati distinti:

- **Sempre aperte:** permettono il passaggio senza restrizioni.
- **Bloccate da un enigma:** il giocatore deve risolverlo per sbloccarle.
- **Bloccate da un oggetto (Pickable):** il giocatore deve possedere un oggetto specifico nell'inventario per sbloccarle.

Soluzione Utilizzo del Pattern Decorator e del Template Method. Per garantire flessibilità ed estensibilità, ho adottato due pattern di progettazione:

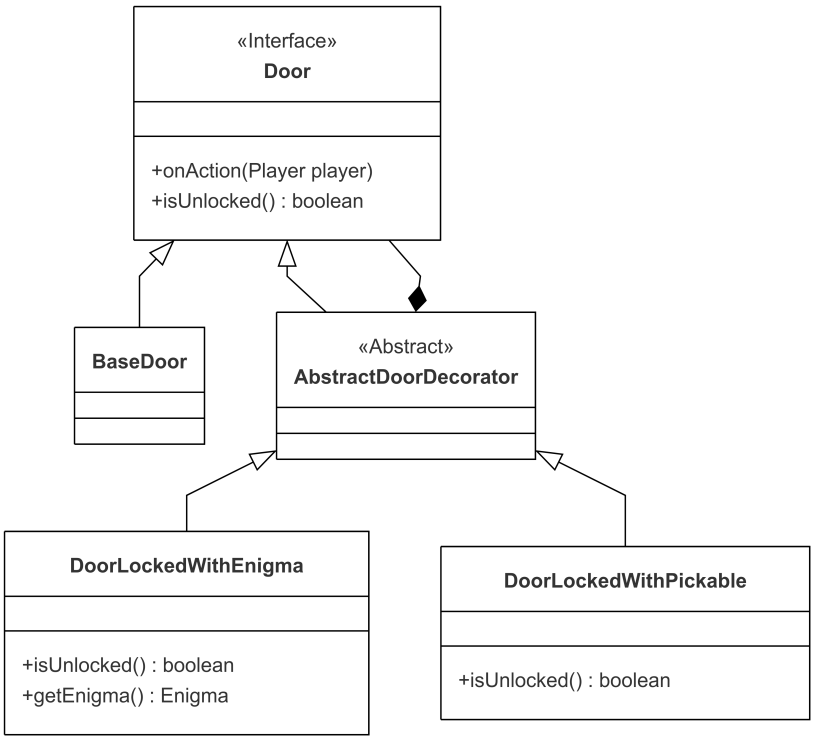
- **Pattern Decorator:** consente di estendere dinamicamente il comportamento delle porte senza modificare direttamente la classe base.
- **Pattern Template Method:** garantisce che il comportamento delle porte bloccate venga gestito in modo strutturato nelle sottoclassi, evitando duplicazioni di codice.

Classi:

- **Creazione della Porta Base:** la classe BaseDoor rappresenta una porta semplice, sempre aperta. Implementa il metodo onAction(Player player), che permette al giocatore di raggiungere la stanza di destinazione.

- **Decorazione con Enigma:** DoorLockedWithEnigma estende AbstractDoorDecorator e richiede la risoluzione di un enigma per lo sblocco.
- **Decorazione con Pickable:** La classe DoorLockedWithPickable segue lo stesso principio, ma invece di un enigma controlla se il giocatore possiede un oggetto con un ID specifico.

La logica di sblocco viene gestita attraverso il Template Method isUnlocked() in AbstractDoorDecorator, il quale viene sovrascritto nelle classi derivate per specificare il comportamento di sblocco. Questa implementazione permette di aggiungere nuovi tipi di blocco senza modificare la classe DoorImpl, garantendo un design scalabile, mantenibile e seguendo il principio Open-Closed della programmazione a oggetti.



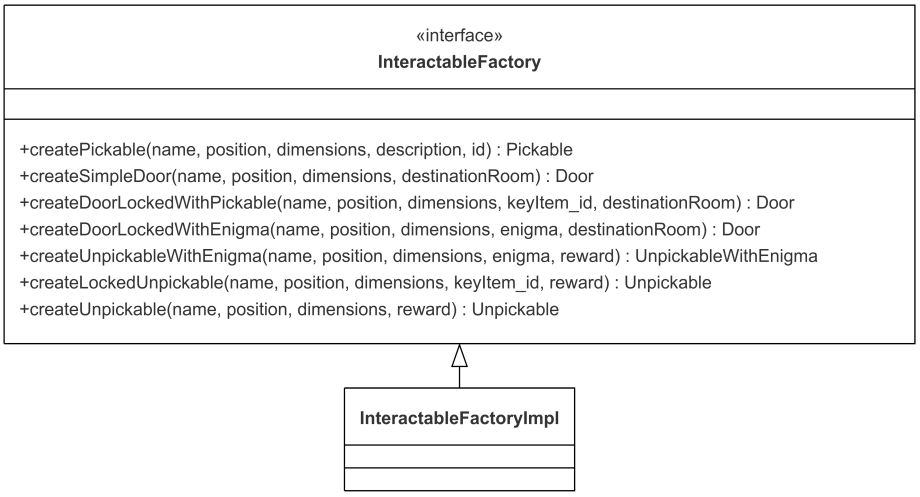
Problema Creazione degli Oggetti Interagibili (Interactable). Nel gioco Mind Escape, il personaggio può interagire con diversi tipi di oggetti, ognuno con comportamenti specifici:

- **Pickable:** oggetti raccolti nell'inventario.
- **Simple Door:** porte sempre accessibili senza restrizioni.

- **DoorLockedWithEnigma:** porte sbloccate solo dopo aver risolto un enigma.
- **DoorLockedWithPickable:** porte sbloccate se il giocatore possiede un oggetto specifico.
- **UnpickableWithEnigma:** oggetti non raccoglibili che attivano un enigma.
- **LockedUnpickable:** oggetti non raccoglibili che si sbloccano con un oggetto presente nell'inventario.
- **Unpickable:** oggetti non raccoglibili che, se interagiti, possono rilasciare un oggetto raccoglibile.

La gestione separata di questi oggetti avrebbe reso il codice rigido, con molte classi duplicate e difficili da estendere.

Soluzione Implementazione di una Factory per gli oggetti interagibili. Per evitare ripetizioni e rendere il codice più modulare, ho utilizzato il Factory Pattern, centralizzando la creazione di tutti gli oggetti interagibili (Interactable). La InteractableFactory fornisce metodi per creare ogni tipo di oggetto interagibile, accettando solo gli attributi essenziali per costruire ciascun oggetto senza duplicare la logica. Il gioco può così facilmente supportare nuovi oggetti interagibili senza modifiche invasive al codice. Grazie a questo design, il sistema è più scalabile, chiaro e manutenibile, garantendo una gestione efficiente di tutti gli oggetti con cui il giocatore può interagire.



Problema Nel gioco Mind Escape, gli enigmi rappresentano un aspetto fondamentale della progressione. Tuttavia, gestire ogni enigma con una logica monolitica avrebbe reso il codice difficile da mantenere e scalare. Il problema principale era separare la logica di ogni enigma dalla sua interfaccia grafica e dal controllo dell'interazione con l'utente.

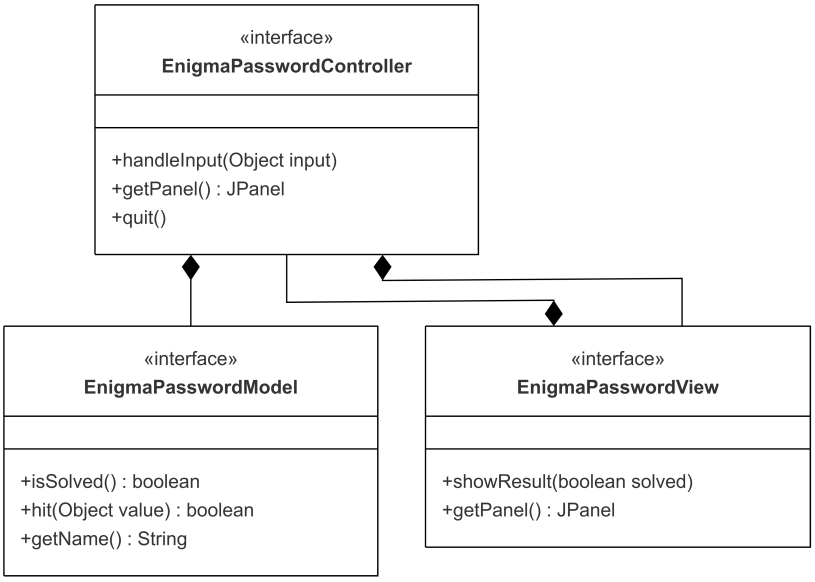
Soluzione Implementazione del Pattern MVC. Per gestire gli enigmi in modo modulare e scalabile, ho utilizzato il pattern Model-View-Controller (MVC). Ogni enigma è trattato come un'istanza indipendente di MVC, con tre componenti chiave:

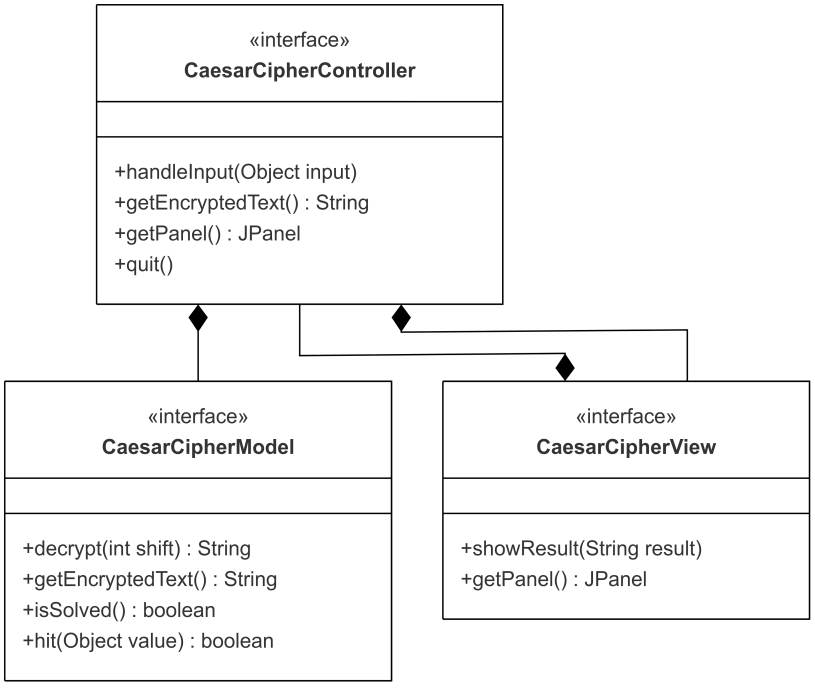
- **Model** (CaesarCipherModel, EnigmaPasswordModel)
  - Contiene la logica dell'enigma (es. cifratura del testo nel caso del Caesar Cipher).
  - Memorizza lo stato dell'enigma (risolto o meno).
  - Implementa metodi per verificare la soluzione.
- **View** (CaesarCipherViewImpl, EnigmaPasswordViewImpl)
  - Gestisce l'interfaccia grafica, visualizzando l'enigma e i risultati dell'interazione.
  - Riceve input dall'utente (es. shift per la cifratura, password per l'enigma testuale).
  - Mostra il feedback all'utente.
- **Controller** (CaesarCipherControllerImpl, EnigmaPasswordControllerImpl)
  - Interpreta l'input dell'utente e aggiorna il Model di conseguenza.
  - Chiede alla View di aggiornare la UI in base ai cambiamenti nel Model.
  - Comunica con il MainController per gestire il flusso del gioco e notificarlo a seguito della terminazione di un enigma, reimpostando come controller corrente quello del World.

Vantaggi:

- **Separazione delle responsabilità:** ogni componente è responsabile di un solo aspetto.

- **Manutenibilità migliorata:** è possibile aggiornare la logica di un enigma senza toccare la UI o il controller.
- **Maggiore modularità:** gli enigmi possono essere facilmente riutilizzati o modificati per adattarsi a nuove esigenze.





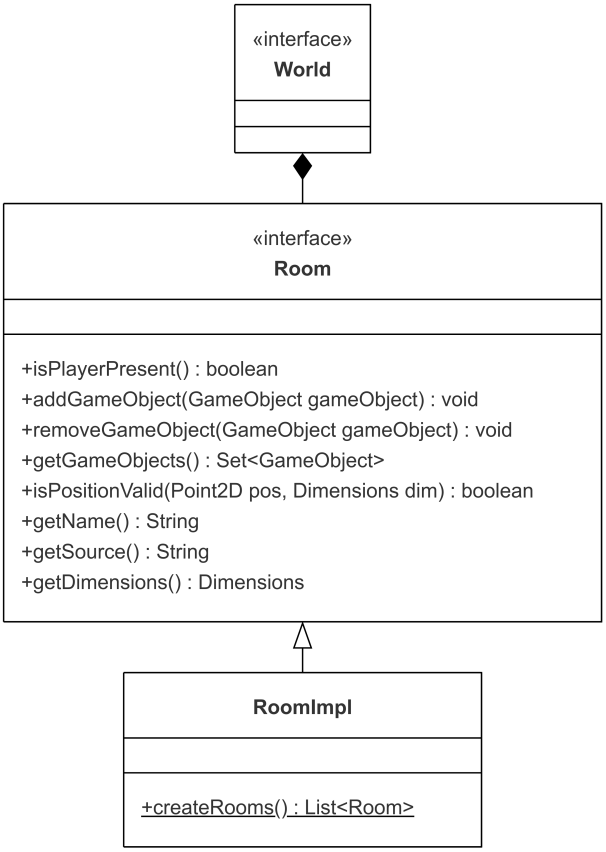
2.2.4 Marcello Spagnoli

Problema Nel gioco, le stanze rappresentano l'ambiente di gioco in cui il giocatore può muoversi ed interagire con gli oggetti. Il problema principale è modellare le stanze come spazi navigabili e gestire in modo efficiente la loro creazione.

Soluzione Le stanze sono caricate da file .tmx. Per gestire la loro creazione e la loro interazione con il giocatore, la classe RoomImpl si occupa di:

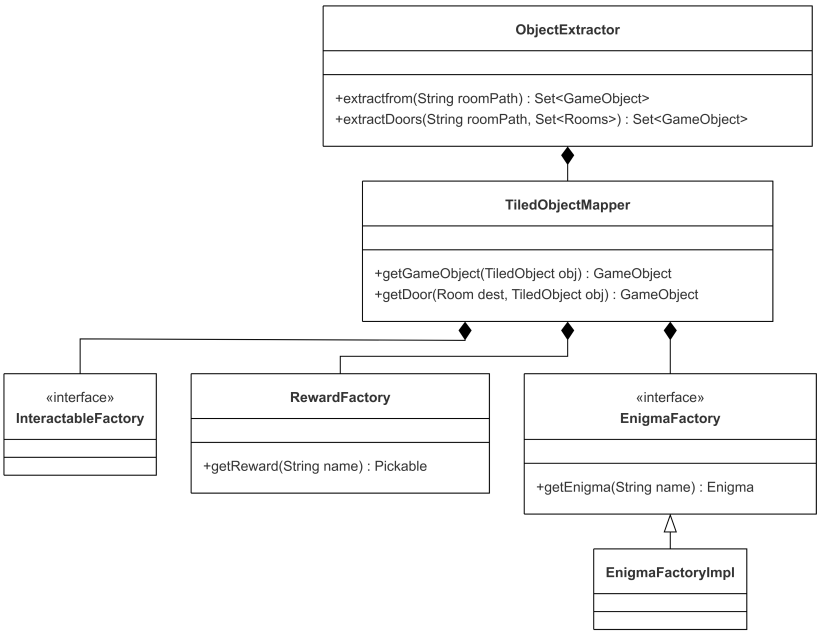
- Leggere i dati della stanza da file .tmx utilizzando la libreria TiledReader.
- Memorizzare gli oggetti di gioco.
- Controllare la validità della posizione degli oggetti, assicurando che non si trovino fuori dai limiti della stanza.

Per gestire la creazione delle stanze ho usato una Static Factory, che a partire dalla cartella contenente le stanze crea una lista di stanze e le popola di oggetti. Ho usato questo approccio perché essendo che le Door per essere create hanno bisogno del riferimento alla stanza di destinazione, è importante che prima vengano create tutte le stanze e che poi vengano popolate. In questa maniera per aggiungere una stanza al gioco basterebbe mettere il suo file tmx nella cartella risorse apposita e aggiungere il nome della stanza nell'enum RoomNames Per evitare la creazione in modo scorretto di stanze, ho messo il costruttore di RoomImpl privato e reso questo l'unico modo di generarle.



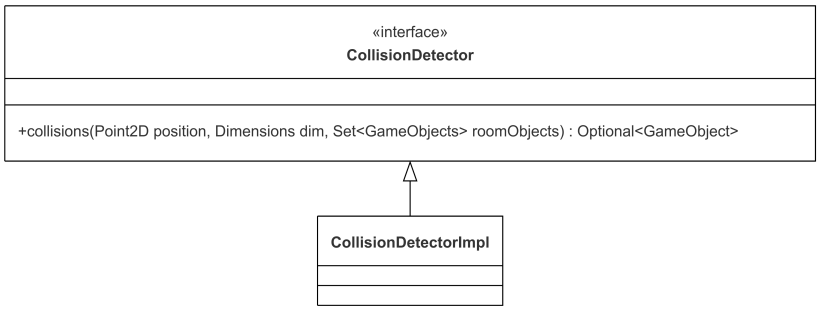
Problema Le stanze hanno bisogno di essere popolate con gli oggetti di gioco sempre presenti nel file .tmx.

Soluzione Ho realizzato una classe ObjectExtractor che si occupa di estrarre gli oggetti da file e di crearli. La conversione da TiledObject (la modellazione di oggetto fornita da TiledReader) a GameObject è operata da una classe TiledObjectMapper che si serve della Factory di Gameobject di Spaccini. Siccome certi oggetti sono associati a specifici enigmi o rewards, ho creato due Simple Factory a cui viene passata la stringa che identifica l'oggetto di cui si fa richiesta e che lo creano (EnigmaFactoryImpl e RewardFactory). Nel caso di aggiunta di nuove stanze per ciò che riguarda questo problema non si deve fare nulla se non avere l'accortezza di aggiornare le Simple Factory se i nuovi GameObject incorporano rewards o enigmi



Problema Ad ogni movimento del giocatore, è necessario verificare se entra in contatto con altri oggetti della stanza, impedendo il passaggio attraverso ostacoli.

Soluzione È stata creata un'interfaccia CollisionDetector, con un metodo confronta la posizione e le dimensioni del giocatore gli oggetti nella stanza. Se viene rilevata una collisione, il metodo restituisce un Optional di GameObject. L'implementazione di CollisionDetector viene quindi usata per determinare se il player può avanzare o meno e per avere il riferimento all'oggetto con cui nel caso egli può interagire



Problema L'immagine della stanza mostrata a schermo deve essere gestita in modo efficiente, evitando di ridisegnare ripetutamente gli stessi elementi e garantendo un aggiornamento fluido durante il gioco.

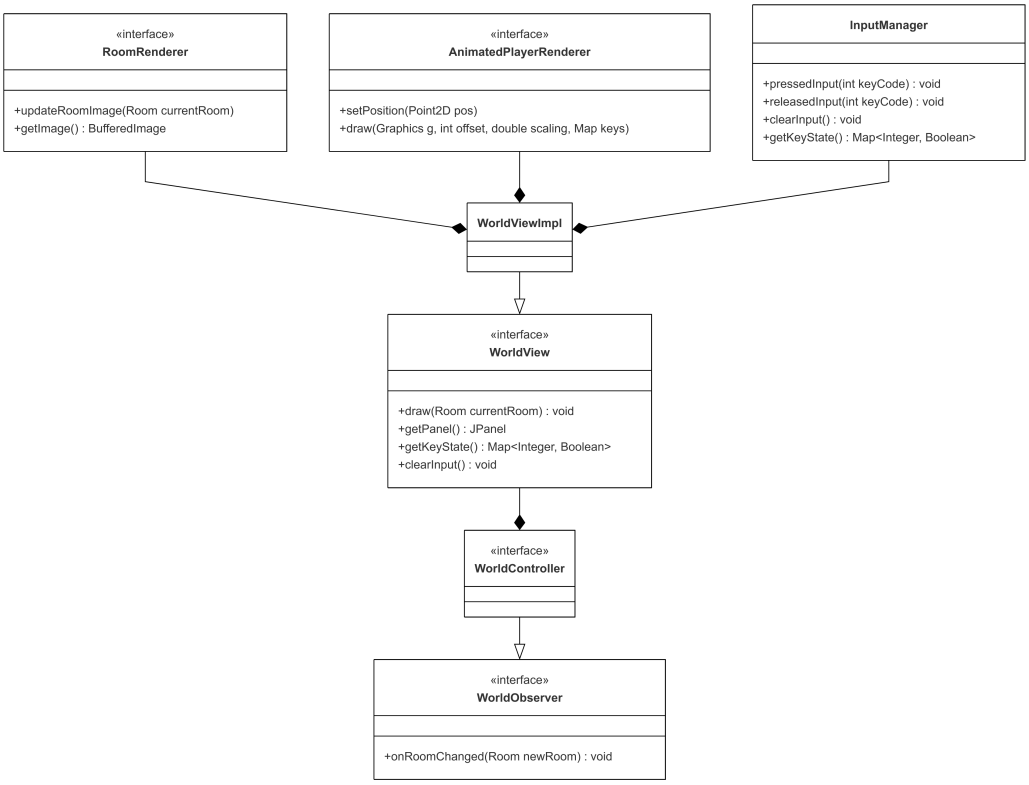
Soluzione La gestione della visualizzazione della stanza è affidata a un Room Renderer, che mantiene una BufferedImage generata a partire da un tileset. Per ottimizzare le prestazioni:

- Si utilizza una cache delle tile, evitando di ricaricare immagini identiche più volte.
- Solo quando avvengono cambiamenti si ridisegna l'immagine.

Il giocatore è disegnato separatamente sopra l'immagine della stanza da un'altra classe AnimatedPlayerRenderer, che ne gestisce anche l'animazione. La rilevazione dei cambiamenti che implicano l'aggiornamento dell'immagine, ovvero raccolta di oggeti pickable e cambio stanza, è affidata al controller che implementa l'interfaccia WorldObserver, mentre il World è l'observable. Quando viene rilevato un cambiamento si impone la rigenerazione dell'immagine della stanza. Di fatto una implementazione di WorldView offre la possibiltà di essere composta modularmente, rispettando il SRP da queste due classi sopracitate e da un gestore di input (segue)

Problema WorldView deve essere in grado di catturare in modo efficiente gli input dell'utente e di organizzarli per far si che il controller ne possa fruire e gestire.

Soluzione La classe InputManager si occupa di raccogliere gli eventi della tastiera tramite l'interfaccia KeyListener. Essa registra gli eventi di pressione e rilascio dei tasti e memorizza lo stato dei tasti. InputManager quale tasto è stato premuto e mantiene il suo stato (premuto o rilasciato). A ogni ciclo di gioco il controller richiede lo stato dei tasti e sulla base di questo esegue le opportune operazioni


«interface»
RoomRenderer
+updateRoomImage(Room currentRoom)
+getImage() : BufferedImage
«interface»
AnimatedPlayerRenderer
+setPosition(Point2D pos)
+draw(Graphics g, int offset, double scaling, Map keys)
InputManager
+pressedInput(int keyCode) : void
+releasedInput(int keyCode) : void
+clearInput() : void
+getKeyState() : Map<Integer, Boolean>
WorldViewImpl
«interface»
WorldView
+draw(Room currentRoom) : void
+getPanel() : JPanel
+getKeyState() : Map<Integer, Boolean>
+clearInput() : void
«interface»
WorldController
«interface»
WorldObserver
+onRoomChanged(Room newRoom) : void

Capitolo 3

Sviluppo

3.1 Testing automatizzato

I principali elementi del Model sono stati testati utilizzando JUnit. Di seguito le classi in esame:

3.1.1 Elena Fucci

- Per l'inventario viene testato che siano aggiunti e rimossi correttamente gli oggetti pickable e che la lista di items contenuta in esso sia corretta.
- Nel puzzle viene testato il corretto mescolamento dei pezzi e che avvenga lo swap tra due celle del puzzle dopo che entrambe sono state cliccate.

3.1.2 Filippo Greppi

- Garantire che GameObjectImpl gestisca correttamente posizione, nome e dimensioni, fornendo metodi getter e setter funzionanti.

3.1.3 Ettore Spaccini

- L'interazione con gli oggetti di gioco viene verificata controllando che un oggetto Pickable, una volta raccolto, venga correttamente aggiunto all'inventario del giocatore. Si testa inoltre che le porte, dopo la risoluzione dell'eventuale vincolo di sblocco, consentano il passaggio alla stanza di destinazione. Infine, viene verificato che l'interazione con un oggetto in grado di restituire un Pickable avvenga correttamente, aggiungendolo all'inventario.

3.1.4 Marcello Spagnoli

- Ho testato CollisionDetectorImpl controllando tutte le collisioni con gli oggetti di una stanza noti
- Ho testato RoomImpl provando l'aggiunta e la rimozione di oggetti, la presenza del player, la validità delle posizioni e il getter degli oggetti

3.2 Note di sviluppo

3.2.1 Elena Fucci

- **Utilizzo di Stream e lambda expressions:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1cceb`
  `src/main/java/mindescape/model/enigma/enigmapuzzle/impl/EnigmaPuzzleModelImp`
  `java#L113C4-L117C6`

3.2.2 Filippo Greppi

- **Utilizzo di Stream e lambda expressions:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1cceb`
  `src/main/java/mindescape/model/world/impl/WorldImpl.java#L50C9-L53C20`
- **Utilizzo di Optional:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1ccebcd882424284b0/src/main/java/mindescape/model/world/impl/WorldImpl.java#L71`
- **Utilizzo di Ordering di Google Guava:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1cceb`
  `src/main/java/mindescape/model/saveload/impl/SavesImpl.java#L37C9-L40C47`

3.2.3 Ettore Spaccini

- **Utilizzo di Stream e lambda expressions:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1cceb`
  `src/main/java/mindescape/model/world/items/interactable/impl/DoorLockedWithPickable.java#L51C8-L53C62`
- **Utilizzo di Optional:** Permalink: `https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1ccebcd882424284b0/src/main/java/mindescape/controller/caesarcipher/impl/CaesarCipherControlle`
  `java#L73C1-L74C1`

3.2.4 Marcello Spagnoli

- **Utilizzo di Stream e lambda expressions:** Permalink: https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1cceb
src/main/java/mindescape/model/world/rooms/impl/RoomImpl.java#L113
- **Utilizzo della libreria TiledReader:** Link: https://github.com/AlexHeyman/TiledReader Permalink: https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1ccebcd882424284b0/src/main/java/mindescape/view/world/RoomRendererImpl.java#L61
- **Utilizzo della libreria Guava (per gestire estensione file):** Permalink: https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1ccebcd882424284b0/src/main/java/mindescape/model/world/rooms/impl/RoomImpl.java#L41
- **Utilizzo di Optional:** Permalink: https://github.com/GreppiFilippo/OOP24-mind-escape/blob/6b13a346f9c2334868ee8e1ccebcd882424284b0/src/main/java/mindescape/model/world/core/impl/CollisionDetectorImpl.java#L20
- **Link a risorse** Capire se si runna da jar https://stackoverflow.com/questions/482560/can-you-tell-on-runtime-if-youre-running-java-from-wit
- **Link a risorse** Manipolazione immagini https://www.google.com/search?client=safari&rls=en&q=flipping+image+java&ie=UTF-8&oe=UTF-8

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Elena Fucci

Nel contesto del progetto, mi sono occupata dell'implementazione del modello del Player, dell'inventario e di due enigmi, nello specifico quello del puzzle e quello del calendario. Ritengo di aver dedicato un notevole impegno a questa attività, cercando di fare del mio meglio nel tempo a disposizione. Qualora avessi l'opportunità in futuro, mi piacerebbe rendere il gioco più coinvolgente, aggiungendo funzionalità come, ad esempio, l'introduzione di un timer per la risoluzione di alcuni enigmi. La parte che ha richiesto il maggior sforzo è stata principalmente quella iniziale, relativa alla creazione della struttura dell'intero progetto. In particolare, l'inserimento degli enigmi all'interno dell'UML non è stato un processo semplice. Durante lo sviluppo del progetto, c'è stata una forte collaborazione all'interno del gruppo; anche nella progettazione delle singole componenti, quando necessario, è stato sempre possibile entrare in contatto con i membri del team per supporto e confronto.

4.1.2 Filippo Greppi

Nel contesto del progetto, mi sono occupato dell'implementazione del modello del mondo, del menu, del sistema di salvataggio e caricamento su file, della creazione del gameobject di base, nonché dello sviluppo di MainController, MainView e WorldController. Ho dedicato un grande impegno alla realizzazione del progetto, collaborando attivamente con il team e rendendomi sempre disponibile. La parte più complessa è stata senza dubbio la progettazione dell'architettura, che ha richiesto un notevole sforzo per garantire una

struttura solida e ben organizzata. Nel complesso, considero questa esperienza altamente formativa, soprattutto per quanto riguarda il lavoro di squadra e l'organizzazione di un progetto complesso. In futuro, difficilmente tornerò a lavorare su questo progetto, poiché preferisco guardare avanti anziché soffermarmi sul passato. Tuttavia, il percorso svolto è stato prezioso per la mia crescita personale e professionale.

4.1.3 Ettore Spaccini

Il mio ruolo nel progetto Mind Escape è stato quello di occuparmi della creazione degli oggetti di gioco e degli enigmi. Questa esperienza mi ha permesso di crescere non solo dal punto di vista tecnico, ma anche a livello personale, insegnandomi a lavorare in gruppo, confrontarmi con i miei compagni e collaborare efficacemente per raggiungere un obiettivo comune. Il clima all'interno del team è sempre stato positivo e stimolante, rendendo il tempo trascorso su questo progetto estremamente piacevole e formativo. In futuro, mi piacerebbe aggiungere nuovi enigmi con meccaniche più varie e migliorare l'esperienza di gioco con dettagli visivi e sonori più curati. Sarebbe anche interessante rendere il gioco più interattivo, magari permettendo ai giocatori di creare i propri enigmi o espandendo la storia.

4.1.4 Marcello Spagnoli

Il mio ruolo è stato quello di creare le room e popolarle oltre che organizzarne il rendering. Ho fatto abbatsnza ricerca per capire come muovermi. L'unico rimpianto è non aver subito guardato bene come funziona TiledReader, tant'è che la prima volta che abbiamo fatto partire da jar non andava nulla. In seguito mi sono accorto che succedeva perché l'oggetto che serve a caricare la mappa fornito da tiled reader era scritto solo per funzionare da filesystem normale, e all'interno di un archivio jar non trovava nulla. Dopo aver distinto i casi è ricominciato ad andare. Sono soddisfatto del mio lavoro, è stata un'esperienza formativa siccome non avevo mai lavorato a un progetto così in gruppo. Tuttavia anche se sono contento del risultato finale non penso rimetterò mano a questo progetto

Appendice A

Guida utente

I comandi di gioco sono i seguenti:

- **W, A, S, D**: per muovere il personaggio
- **E**: per interagire con gli oggetti
- **I**: per aprire l'inventario

Appendice B

Guida soluzione

B.1 Stanza 1 - Camera da Letto

- Interagendo con letto si raccoglie un biglietto
- Interagendo con il tavolo si apre un puzzle
- la porta si sblocca inserendo la password “Sergio Mattarella”

B.2 Stanza 2 - Mensa

- Raccogliere il biglietto sul tavolo
- Guardare il calendario (vicino alla porta di entrata) come suggerisce il biglietto
- Interagire con la cassettiera vicino al calendario (Password: “12-13”)
- Interagire di nuovo con la cassettiera per prendere la chiave (che sblocca una delle due porte)
- Raccogliere la chiave inglese
- Provare a sbloccare una delle 2 porte

B.3 Stanza 3 - Ufficio

- Raccogliere la torcia
- Interagire col computer (Shift: “3”)

- Tornare alla mensa
- Tentare di sbloccare la porta rimasta

B.4 Stanza 4 - Archivio

- Aprire la cassa raccogliendo il martello
- Andare nell' armadio in mezzo alla stanza e inserire la password del cifrario: "oblivion"
- Interagire di nuovo con l'armadio per prendere il biglietto al suo interno (che sblocca l'ultima porta)

B.5 Stanza 5 - Finale

- Interagire con lo specchio

Appendice C

Esercitazioni di laboratorio

C.0.1 ettore.spaccini@studio.unibo.it

- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=179154#p248293`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=180101#p248884`

C.0.2 marcello.spagnoli2@studio.unibo.it

- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=179154#p248384`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=180101#p248819`
- Laboratorio 11: `https://virtuale.unibo.it/mod/forum/discuss.php?d=181206#p250415`