Relazione di progetto: PacMan 2.0

Ambrogiani Alessandro: 0001080493
Gambacorta Giuseppe: 0001070834
Penserini Nicolò: 0001080348
Zarrilli Antonio: 0001078463

18 febbraio 2024

Indice

Capitolo 1

Analisi

L'applicazione è un clone del famoso gioco PacMan[1], con alcune aggiunte. Il giocatore può comandare Pacman in varie mappe, raccogliendo punti e scappando dai fantasmini. All'interno della mappa sono presenti oggetti raccoglibili semplici, che aumentano il punteggio del giocatore, e oggetti speciali, che attivano, al momento della raccolta, effetti speciali che possono favorire o sfavorire il giocatore. Al termine degli oggetti raccoglibili all'interno della mappa, viene generata una nuova mappa e la partita continua conservando il punteggio e il numero di vite. Le vite di pacman diminuiscono quando si scontra con uno dei fantasmi nella mappa, a meno che i fantasmi non siano sotto l'effetto di una condizione speciale che li rende mangiabili, oppure quando pacman raccoglie determinati oggetti. Al termine delle vite la partita termina.

1.1 Requisiti

Requisiti funzionali

- Ogni fantasmino avrà il proprio modo di agire con propria intelligenza artificiale. Dovranno seguire il giocatore all'interno della mappa e ritornare alla propria base in caso di morte.
- Oggetti raccoglibili che influenzano il comportamento di Pacman e dei fantasmi, ad esempio aumentandone o diminuendone la velocità.

[1] `https://it.wikipedia.org/wiki/Pac-Man`

- I fantasmi possono diventare mangiabili quando Pacman raccoglie un oggetto con un determinato effetto speciale, in questo caso smetteranno di seguire Pacman e torneranno alla base.
- Pacman dovrà muoversi all'interno della mappa interagendo con gli oggetti da raccogliere e i fantasmi. Alle collisioni tra Pacman e oggetti dovrà corrispondere la raccolta dell'oggetto, a collisioni tra Pacman e un fantasma la morte di PacMan, e il suo respawn se non ha terminato le vite, oppure del fantasma, nel caso in cui sia "mangiabile".
- Alla raccolta di un oggetto dovrà essere aumentato il punteggio e, se l'oggetto ha un effetto, dovrà essere applicato l'effetto a Pacman o ai fantasmi. In caso di oggetti con effetti speciali, dovrà essere mostrata una breve descrizione dell'effetto ottenuto.
- All'interno della mappa sono presenti blocchi non oltrepassabili nè da Pacman nè dai fantasmi, teletrasporti ai limiti della mappa e la base dei fantasmi, a cui PacMan non può accedere.
- Dovranno essere presenti varie mappe, che si alternano casualmente nel corso della partita.
- Al termine degli oggetti nella mappa, la partita dovrà continuare con il caricamento di una nuova mappa e il riempimento di essa con nuovi oggetti.
- Durante la partita dovranno essere visualizzate all'interno dello schermo informazioni riguardo il numero di vite rimanenti di Pacman e il punteggio attuale.
- Al termine delle vite di PacMan, la partita dovrà terminare e dovrà essere mostrato il punteggio.

Requisiti non funzionali

- Il gioco dovrà funzionare su tutte le piattaforme.
- Il gioco dovrà essere visualizzato correttamente su qualsiasi display, garantendo buone prestazioni durante il rendering grafico.

1.2 Analisi e modello del dominio

Il gioco è composto da una **mappa**, che può essere cambiata durante la partita. Ogni mappa è composta da diverse celle che possono essere di più tipi:

- **pavimenti**: celle che possono essere attraversate sia da Pacman che dai fantasmi.
- **muri**: celle che non possono essere attraversate nè da PacMan nè dai fantasmi.
- **cancello**: cella che protegge l'area in cui si generano i fantasmi, può essere attraversata dai fantasmi, ma non da PacMan.

All'interno della mappa ci sono due tipi di oggetti di gioco:

- **non animati**: oggetti che non mutano la loro posizione all'interno della mappa durante la partita.
- **animati**: oggetti che si muovono durante la partita con diverse velocità, in seguito a comandi inviati dal giocatore o in base alla posizione degli altri oggetti.

Gli oggetti animati sono **Pacman** e i **fantasmi**. Pacman si muove all'interno della mappa secondo le istruzioni del giocatore, che ne può impostare la direzione, mentre i fantasmi si muovono in base alla posizione di pacman all'interno della mappa e alla loro tipologia di comportamento specifico. Pacman ha un numero di vite che può essere aumentato, in caso di raccolta di un particolare oggetto , o diminuito, in caso di collisione con un fantasma o di raccolta di un determinato oggetto. Ad ogni partita corrisponde un punteggio, che viene aumentato e diminuito nel corso della partita a causa della raccolta degli oggetti nella mappa e delle collisioni con i fantasmi.
I raccoglibili sono oggetti statici divisi in 2 categorie:

- **Semplici**: quelli semplici che una volta mangiati da pacman aumentano il suo punteggio
- **Con effetto**: che una volta mangiati danno, oltre al punteggio, anche un effetto particolare o a pacman o ai fantasmi.

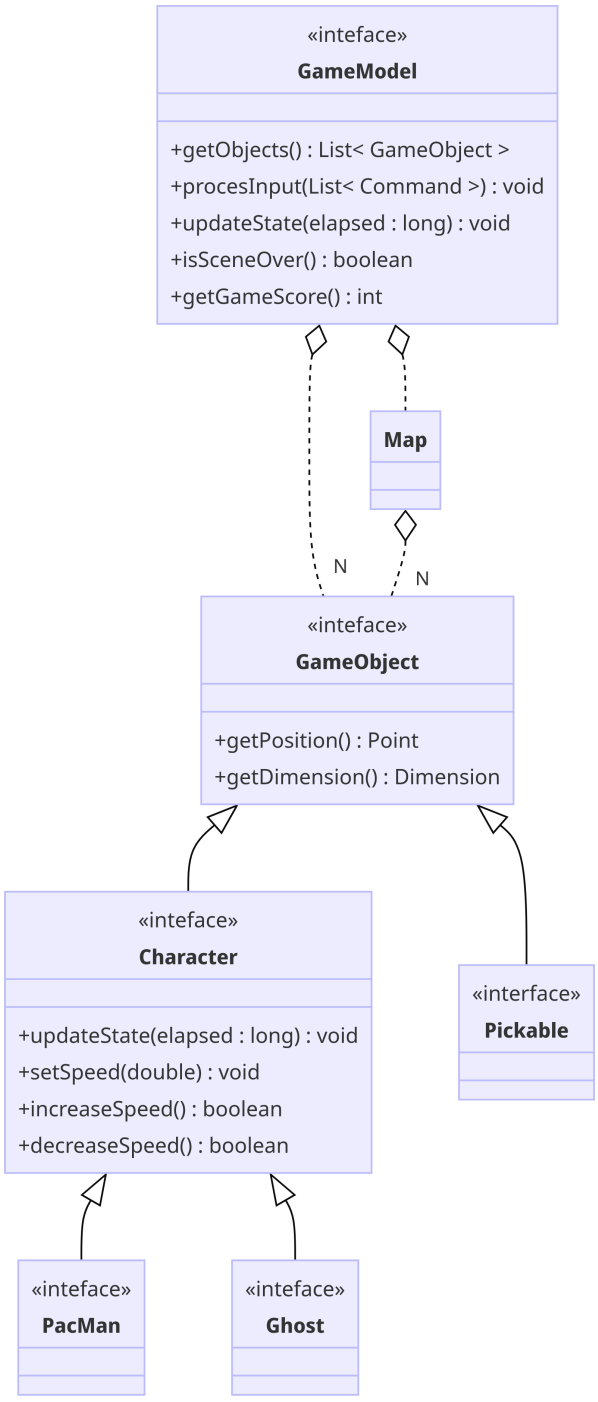

Figura 1.2.1 : Flowchart rappresentativo degli elementi del dominio

Capitolo 2

Design

2.1 Architettura

2.1.1 Engine

Ogni scena di gioco è strutturata seguendo il modello MVC (Model-View-Controller), in cui il controller facilita la comunicazione tra il model e la view. Il compito di processare il controller e gestire l'aggiornamento della finestra è affidato a un motore di gioco (Engine). Il motore di gioco, tuttavia, non si occupa direttamente del caricamento delle diverse scene. Invece, fa riferimento a una classe dedicata chiamata SceneManager. Questa classe ha il compito di monitorare l'inizio e la fine di ciascuna scena e fornisce al motore di gioco l'informazione sulla scena attuale. Lo SceneManager assume anche il ruolo di mediatore tra i singoli controller delle scene. Si occupa di gestire l'effettiva terminazione di ciascuna scena e prende decisioni riguardo a quale scena caricare successivamente. Quando avviene un cambio di scena, lo SceneManager notifica il motore di gioco, il quale richiede allo SceneManager l'attuale scena da processare. In questo modo, la separazione delle responsabilità è mantenuta chiaramente, con il motore di gioco concentrato sul processing e l'aggiornamento della finestra, mentre lo SceneManager gestisce il ciclo di vita delle scene e le transizioni tra di esse.

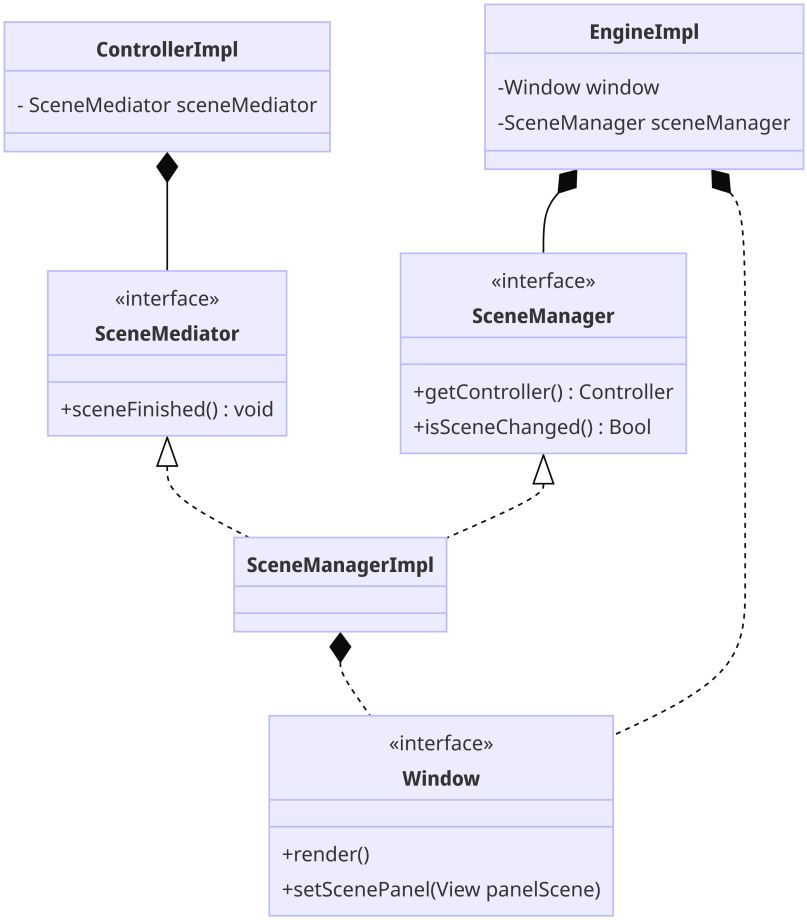

Figura 2.1.1.1 : Schema Engine di gioco

2.1.2 MVC

Nel contesto del nostro sistema di gioco, ogni scena è strutturata seguendo il modello architetturale MVC, acronimo di Model-View-Controller. Questo approccio organizza il codice in tre componenti distinti, ciascuno con responsabilità specifiche, promuovendo una chiara separazione delle preoccupazioni e facilitando la manutenzione e l'estensione del software. Il Model e la View non hanno dipendenze tra di loro, ma il loro comportamento è coordinato dal controller. Questo permette anche di cambiare agevolmente l'implementazione della View, per esempio per passare a una diversa libreria grafica, senza impattare in alcun modo l'implementazione del model e del controller. Per quanto riguarda le scene non di gioco attualmente implementate, abbiamo scelto che il controller assuma temporaneamente anche il ruolo di model.

Tale decisione è motivata dal fatto che le attuali scene non di gioco non richiedono una logica complessa o dati particolarmente elaborati. In questo modo, semplifichiamo la struttura evitando la creazione di un modello separato.
Tuttavia, la nostra progettazione fornisce un elevato grado di flessibilità. Nel caso in cui emergesse la necessità di introdurre logiche più complesse o dati specifici nelle scene non di gioco, il software è progettato per consentire l'implementazione agevole di un modello dedicato. Questo approccio permette di adattare la struttura delle scene del menu senza compromettere l'integrità dell'architettura MVC complessiva. In questo modo, manteniamo una struttura modulare e facilmente estendibile per il nostro sistema di gioco.

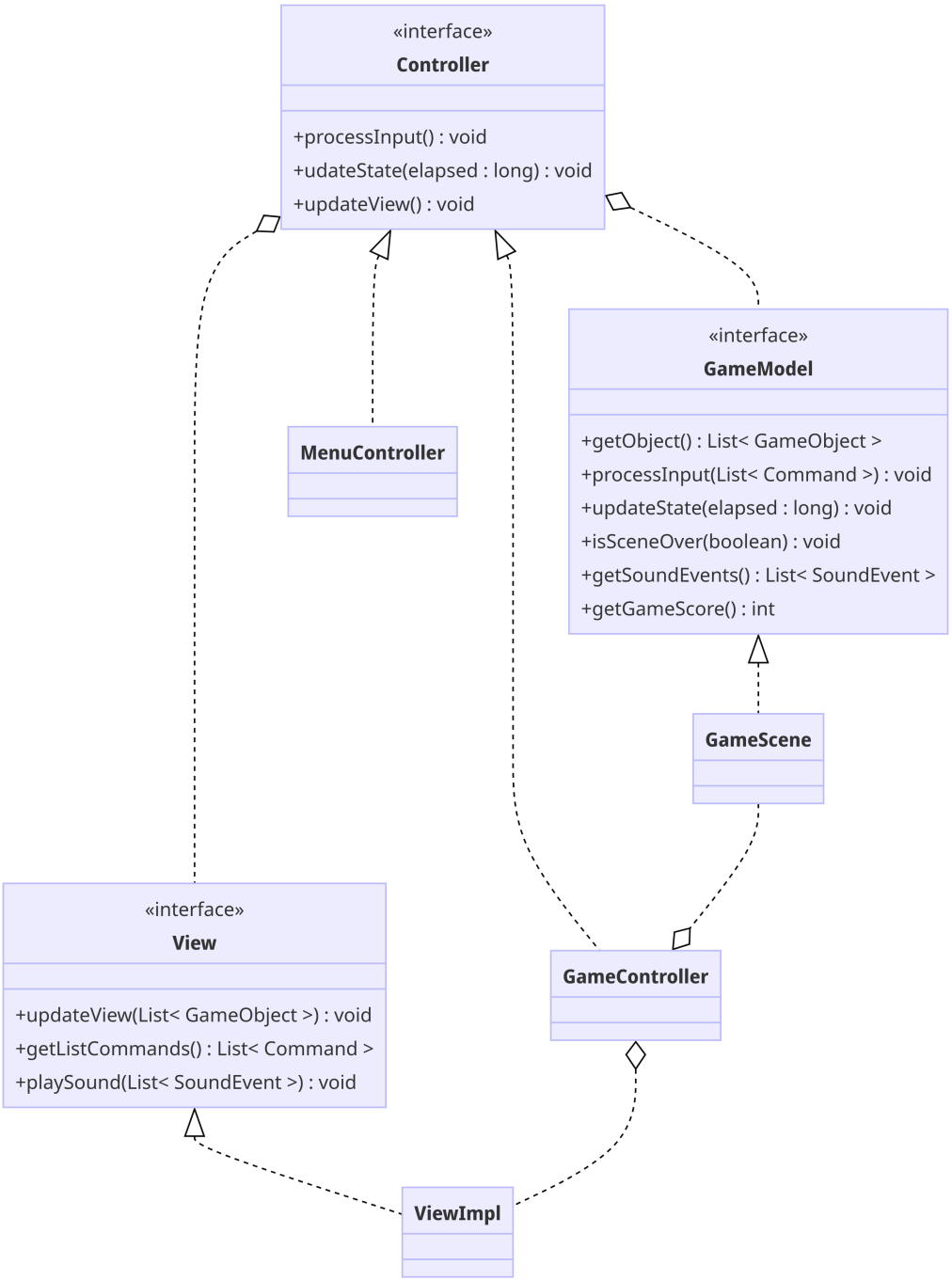

Figura 2.1.2.1 : Schema architetturale MVC di una scena di gioco

Model

Il model rappresenta la logica della scena attuale. Nella scena di gioco, viene aggiornato in base al tempo trascorso dall'ultimo aggiornamento e mantiene i dati relativi allo stato di ogni oggetto di gioco. Il model si occupa inoltre anche di verificare se la scena di gioco è terminata, cosa che avviene al termine delle vite di Pacman, e di conservare le vite di Pacman e il suo punteggio, che viene mostrato alla fine della partita.

View

La View rappresenta l'interazione dell'applicazione con l'utente. Si occupa di ricevere i comandi dall'esterno e di mostrare al giocatore lo stato attuale dell'applicazione. Per suddividere le varie fasi dell'applicazione, abbiamo deciso di sviluppare diverse classi e interfacce responsabili di mostare l'applicazione all'utente, ognuna controllata da uno specifico controller e con specifici comandi possibili.

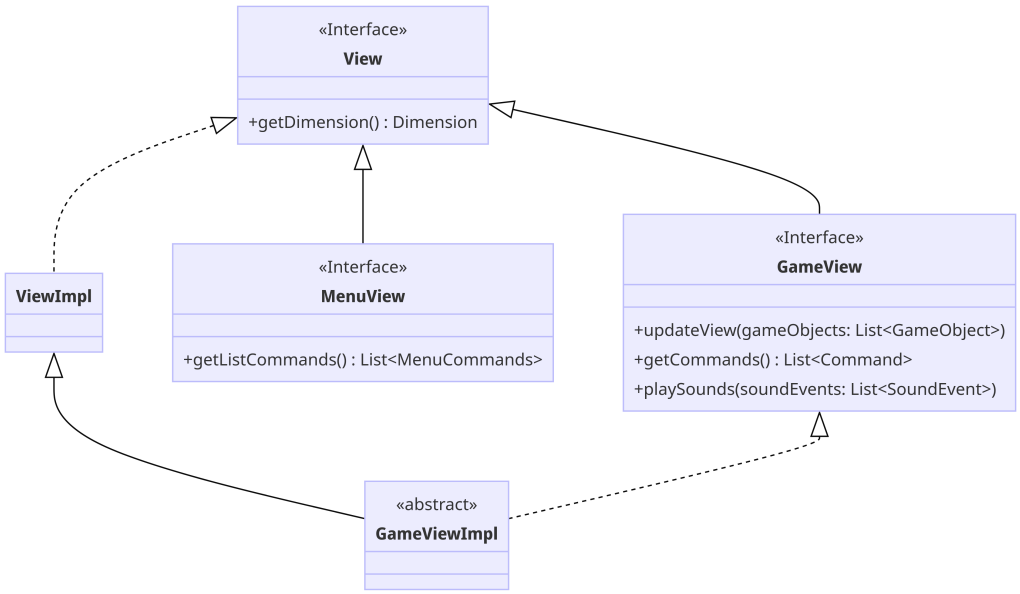

Figura 2.1.2.2 : Diagramma delle classi View

Controller

Il Controller gestisce l'esecuzione del Model e della View e lo scambio di dati tra di essi. Nello specifico il Controller si occupa di aggiornare il Model e di passare i comandi che arrivano dalla View al Model e di passare gli oggetti

del Model alla View, in modo che possano essere correttamente mostrati a schermo. Oltre ai Controller di gioco, che coordinano l'esecuzione delle scene di gioco, possono essere presenti più controller che si occupano di gestire le varie fasi dell'applicazione.
Il Controller si occupa di comunicare con l'esterno, in modo da isolare completamente la View e il Model e non renderli accessibili dall'esterno.

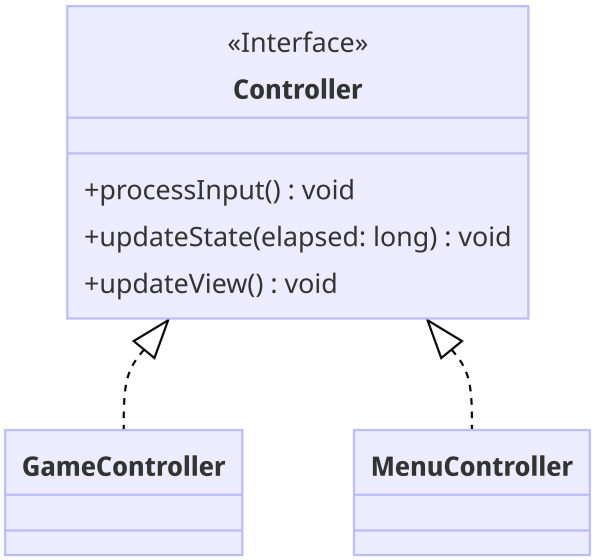

Figura 2.1.2.3 : Diagramma delle classi controller

2.2 Design dettagliato

Nel nostro progetto è presente un'interfaccia da cui derivano tutti gli elementi presenti nella scena di gioco chiamata GameObject. Questa interfaccia è formata da 3 metodi che sono:

- *getPosition* che restituisce la posizione dell'oggetto
- *getImage* che restituisce l'url del png da renderizzare sulla view
- *getDimension* che restituisce le dimensioni dell'oggetto

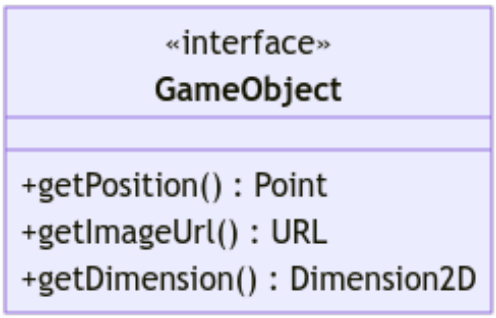

Figura 2.2.1 : Diagramma dell'interfaccia GameObject

2.2.1 Ambrogiani Alessandro

Nel corso dello sviluppo di questo progetto nella parte relativa al model, mi sono occupato della gestione delle mappe,mentre per la view degli effetti sonori in sottofondo.

Gestione delle mappe

Per garantire una gestione efficiente delle mappe, ho deciso di leggerle da file, facilitando così l'aggiunta di nuove mappe in futuro. Il file di ogni mappa è strutturato nel seguente modo: la prima riga contiene il numero di righe e colonne, mentre le righe successive rappresentano la matrice di gioco tramite numeri che corrispondono ai diversi tipi di celle:
Pickable: pavimento dove si generano gli oggetti raccolgibili.
No-Pickable: pavimento dove non si generano oggetti raccoglibili.
Spawn-Ghost: punti della mappa dove si generano i fantasmi.
Gate-Ghost: cancello da cui i fantasmi escono.
Wall: muri della mappa di gioco.
Behind-Wall: aggiunta per il grafo dei fantasmi (spiegato successivamente).
La gestione di questa funzionalità è affidata alla classe **MapReaderImpl**, che implementa l'interfaccia **MapReader**. Questa classe legge la prima riga del file per ottenere il numero di righe e colonne e successivamente scorre il resto del file per convertire la matrice in numeri interi e restituirla.

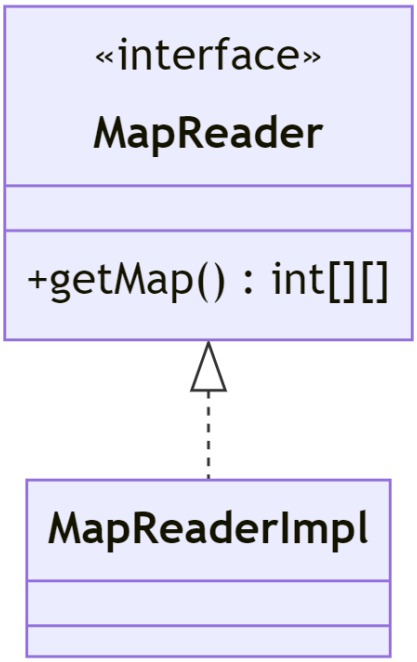

Figura 2.2.1.1 : Diagramma gestione MapReader

Successivamente, ho sviluppato la classe **MapBuilderImpl**, implementando l'interfaccia **MapBuilder**, che utilizza la mappa di interi ottenuta dalla classe **MapReaderImpl**. Grazie all'enumerazione **MapTypes**, questa classe suddivide i vari tipi di celle e inizializza la mappa restituendo le relative coordinate. Inoltre, si occupa di creare i **GameObject** utilizzando la classe **GameObjectFactoryImpl**, che si avvale di **MapImageImpl** per ottenere le immagini corrispondenti ai diversi tipi di oggetti sulla mappa.

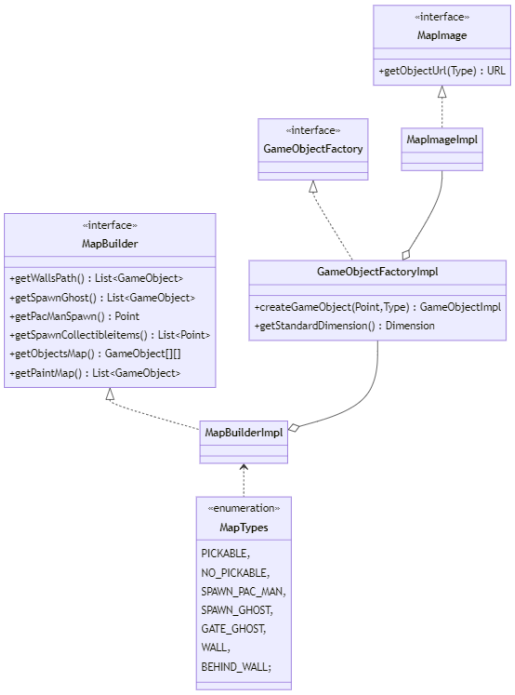

Figura 2.2.1.2 : Diagramma funzionamento MapBuilder

Per gestire i movimenti dei fantasmi rispetto a Pac-Man, ho implementato la classe **MapGraphImpl**, che implementa l'interfaccia **MapGraph**. Utilizzando un apposito algoritmo, questa classe crea un grafo che rappresenta i percorsi percorribili dai fantasmi. La numerazione Behind-Wall è stata aggiunta per evitare problemi nel considerare i pavimenti dietro i muri come percorsi percorribili.

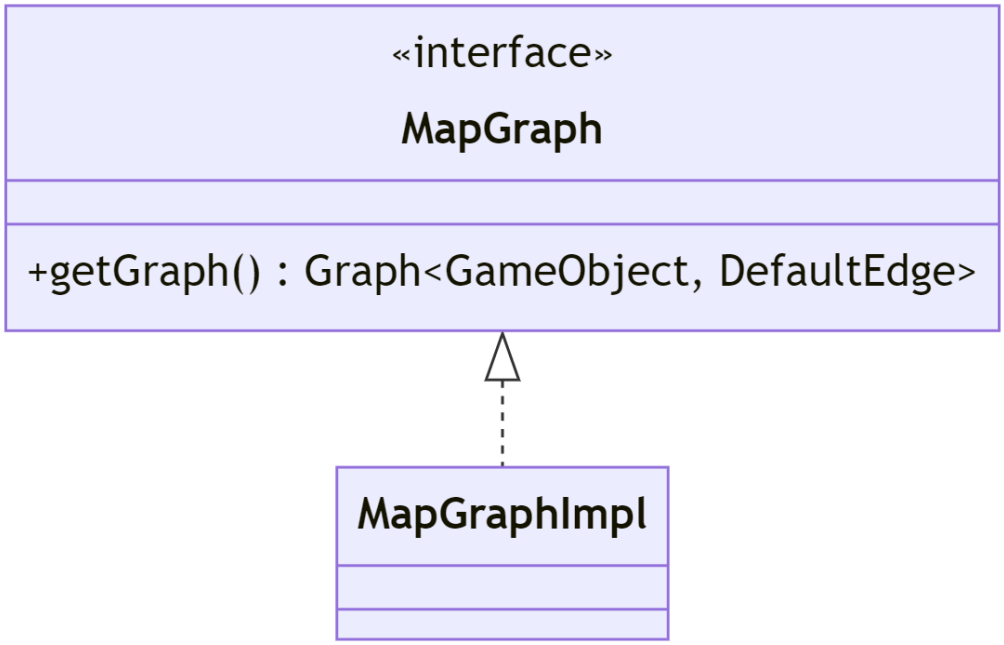

Figura 2.2.1.3 : Diagramma gestione MapGraph

Infine, per garantire una varietà di mappe, ho creato la classe **MapSelectorImpl**, che implementa **MapSelector**. Utilizzando un algoritmo random, questa classe restituisce un nome di mappa diverso da quello precedente, garantendo così una varietà di esperienze di gioco.

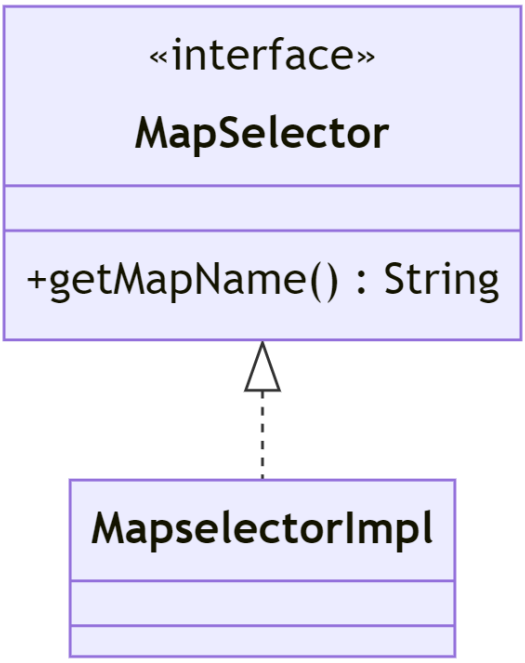

Figura 2.2.1.4 : Diagramma gestione MapSelector

Effetti sonori

Per gestire gli effetti sonori all'interno del gioco, ho creato un'interfaccia chiamata **SoundEffect**, che viene implementata da due classi: **SoundEffectImpl** e **PacManSound**. **SoundEffectImpl** si occupa della riproduzione dei suoni per eventi singoli, come quando Pac-Man mangia un bonus o perde una vita. D'altra parte, **PacManSound** gestisce l'audio del movimento di Pac-Man.

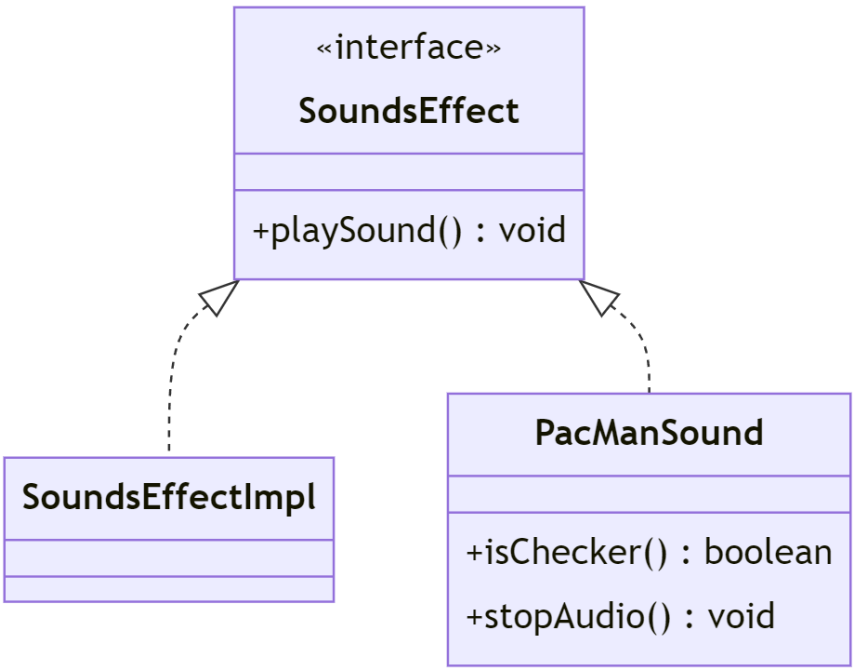

Figura 2.2.1.5 : Diagramma SoundEffect

Per organizzare e gestire gli eventi sonori, ho introdotto un'enumerazione chiamata **SoundEvent**. Questa enumerazione è utilizzata all'interno di **GameScene**, che aggiunge tutti gli eventi sonori in attesa di essere riprodotti. Successivamente, questi eventi vengono passati al **GamePanel**, che si occupa di riprodurre il suono appropriato in base al tipo di evento.

- **Gestione degli Effetti Sonori:**
  - Per gli eventi singoli, ho deciso di gestirli con l'apertura e la chiusura della clip audio all'occorrenza. Dato che non sono eventi continui, ho optato per liberare la memoria una volta esaurita la clip. Tuttavia, per gli eventi continui, ho riscontrato i seguenti problemi:
    * Overhead di apertura e chiusura: L'apertura e la chiusura ripetuta delle clip audio comporta un overhead, poiché sono coinvolte operazioni di I/O per aprire il file audio, configurare il sistema audio e chiudere le risorse associate.
    * Gestione delle risorse: Aprire e chiudere continuamente le risorse audio senza riutilizzarle potrebbe causare un'eccessiva

gestione delle risorse da parte del sistema operativo e della JVM. Questo potrebbe portare a un consumo elevato di memoria e a una maggiore pressione sul garbage collector.

– Per affrontare questi problemi, ho deciso di tenere la clip audio caricata in memoria una volta aperta. Nei casi in cui la clip non fosse necessaria, invece di chiuderla, ho optato per metterla in pausa o resettare il frame della clip a 0 e riprodurla da capo per il prossimo evento. Questo approccio riduce l'overhead e contribuisce a una gestione più efficiente delle risorse.

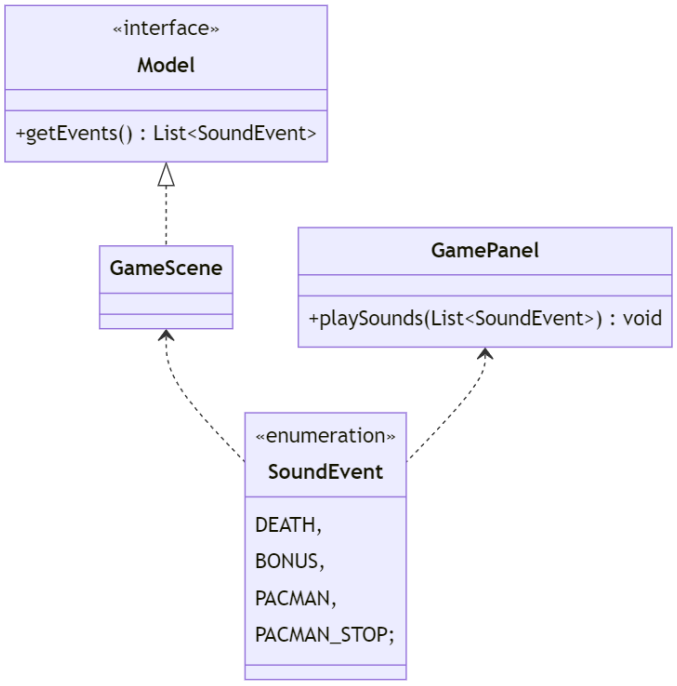

Figura 2.2.1.6 : Diagramma riproduzione degli effetti sonori

2.2.2 Gambacorta Giuseppe

Nel progetto, ho realizzato la parte principale dell'engine basandomi su un mini engine che è stato dettagliatamente illustrato dal Professor Ricci durante l'evento "Game-as-lab" tenuto al campus durante le lezioni.
Dopo uno studio approfondito, ho destrutturato e adattato il codice per integrarlo nel nostro progetto. In particolare, ho introdotto un sistema di gestione delle scene, garantendo che l'engine rimanga all'oscuro dei dettagli specifici delle scene attuali.
Per quanto riguarda il Model della scena di gioco, mi sono occupato della gestione dei movimenti dei fantasmi all'interno della mappa.

Scene Mediator

Il ruolo centrale dello SceneManager nel nostro sistema è quello di fornire all'engine il controllore dell'attuale scena e di integrare il pannello corrispondente all'interno della finestra del programma. Questo compito viene svolto attraverso l'implementazione del pattern ***Mediator*** , dove lo SceneManager agisce come mediatore principale, mediando tra i singoli controller delle scene che comunicano esclusivamente con lo SceneManager, evitando interazioni dirette tra di loro.

I controller notificano lo SceneManager quando una scena è completata, conferendo al manager il compito di prendere decisioni informate su quale scena caricare successivamente. Questo approccio consente una gestione centralizzata delle transizioni tra le scene, migliorando la modularità e la chiarezza del codice.

Un elemento chiave nella progettazione è l'adesione al principio di segregazione delle interfacce (ISP), noto anche come principio di segregazione dell'interfaccia. Tale principio sottolinea l'importanza di fornire alle classi solo le interfacce necessarie, evitando l'esposizione di funzionalità superflue. Nel nostro progetto, abbiamo applicato questo principio allo SceneManager e ai controller di scena, sebbene lo SceneManager sia identificato come Gestore di Scena dall'engine, per i controller agisce unicamente come mediatore.

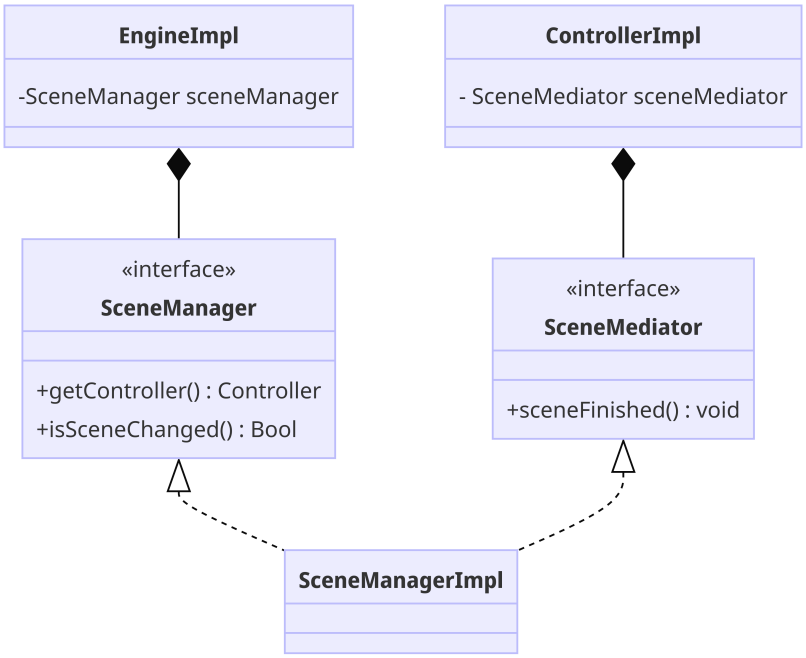

Figura 2.2.2.1 SceneManager FlowChart

Fantasmi

Ho implementato i fantasmi attraverso un'interfaccia dedicata estesa dall'interfaccia Character. In una versione di base, il fantasma è trattato come un personaggio controllabile, analogamente a Pacman, con la capacità di muoversi nella mappa. Tuttavia, è stato introdotto un concetto di stati per i fantasmi, che possono essere normali, spaventati o morti. In base allo stato corrente, l'implementazione di base prevede solo la modifica dell'aspetto grafico del fantasma.

Nel contesto della versione avanzata, denominata "FollowingGhost" e ottenuta tramite il pattern ***Decorator*** , ho esteso le modifiche anche al comportamento effettivo del fantasma all'interno della mappa. Questa versione consente ai fantasmi di adottare un comportamento più sofisticato, che può includere l'inseguimento dinamico di Pacman o il movimento verso altre porzioni di mappa in base al loro stato attuale.

Nella realizzazione del FollowingGhost, ho adottato una struttura basata sul pattern ***Template Method***. Ho progettato una classe astratta che definisce la struttura generale del comportamento del FollowingGhost, implementando tre stati come metodi astratti. Questi metodi astratti fungono da "template" che le classi specializzate possono successivamente concretizzare per ottenere comportamenti specifici.

L'obiettivo di questa implementazione è consentire la creazione di comportamenti distinti senza dover riscrivere la struttura di base di un FollowingGhost. Le classi derivate possono estendere la classe astratta e fornire implementazioni concrete per i metodi astratti, adattando così il FollowingGhost ai requisiti specifici.
Nell'implementazione del FollowingGhost, ho valutato l'opzione di utilizzare il pattern ***State***. Questo avrebbe comportato la creazione di classi specifiche per ciascun tipo di stato, che poi sarebbero potuto essere combinati all'interno di un oggetto FollowingGhost per personalizzarne ulteriormente il comportamento. Tuttavia, ho preferito adottare il pattern Template Method, Questa decisione è stata influenzata dal fatto che, almeno per il momento, i fantasmi presentano solo tre tipologie di stati e non richiedono una complessità tale da giustificare l'utilizzo del pattern State. Considerando che il pattern State coinvolge la creazione di classi specifiche per ogni tipo di stato, ho valutato che l'introduzione di un numero aggiuntivo di classi nel progetto potrebbe risultare eccessiva per la situazione attuale. Nel caso in cui il comportamento dei fantasmi dovesse ulteriormente complicarsi in futuro, si potrebbe rivalutare l'adozione del pattern State per una gestione più modulare delle diverse casistiche.
Inoltre, la scelta del pattern Template Method è stata guidata anche dal fatto che questo pattern è stato ampiamente illustrato e approfondito durante il corso. Mi sono sentito più a mio agio con questa soluzione, considerando la familiarità che ho acquisito nel corso delle spiegazioni e degli esempi forniti durante le lezioni.

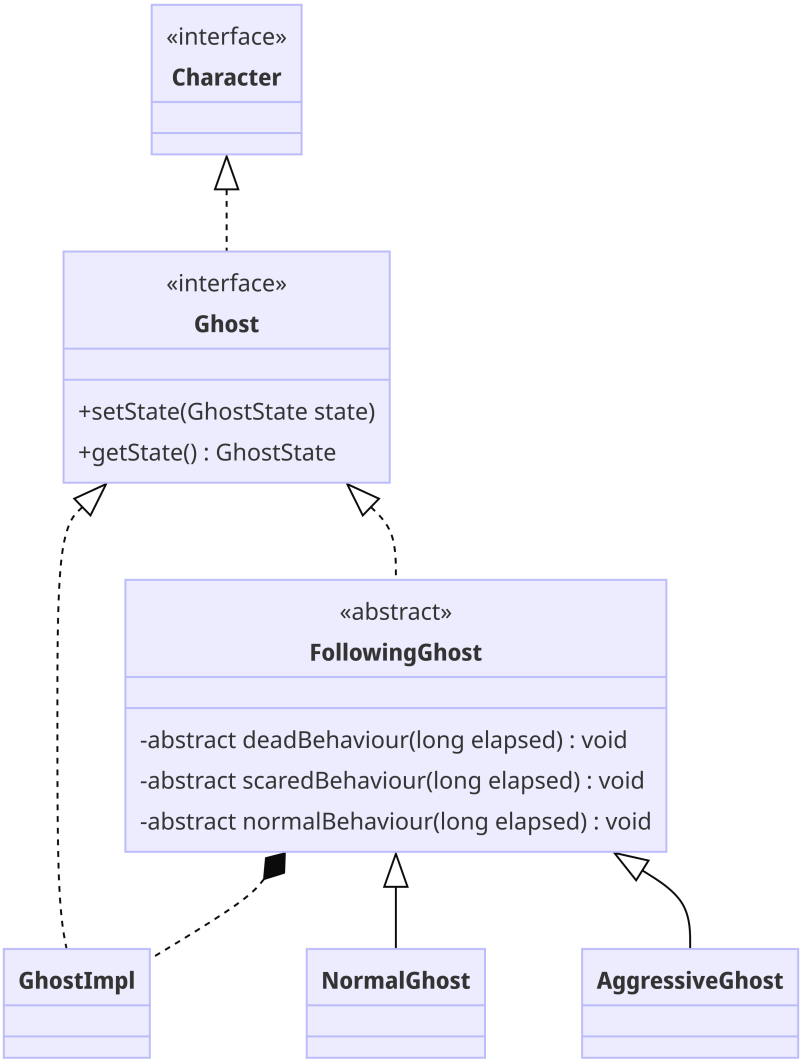

Figura 2.2.2.2 Ghosts FlowChart

GhostCoordinates

All'interno dell'implementazione di FollowingGhostImpl, ho adottato il pattern Strategy per gestire le coordinate e i bersagli del fantasma. In particolare, viene utilizzata l'interfaccia GhostCoordinates.

La classe GhostCoordinatesOnGraph è una implementazione di questa interfaccia che offre un pacchetto di coordinate su un grafo di gioco, consentendo al fantasma di muoversi all'interno dei cammini di questo grafo. La parte fondamentale di questa implementazione è la presenza del CharacterMover, una componente del pacchetto di coordinate, che consente di determinare la direzione del movimento del fantasma in modo intelligente, basandosi sulla struttura del grafo.

Tuttavia, poiché si è scelto di implementare la gestione delle coordinate attraverso il pattern ***Strategy*** , è possibile creare altre implementazioni di GhostCoordinates, offrendo così la flessibilità di assegnare a ogni singolo fantasma diversi tipi di target e comportamenti di movimento. Ad esempio, potrebbe essere creato un altro tipo di pacchetto coordinate che segue una logica di movimento differente, questo pacchetto di coordinate potrebbe anche non essere basato su un grafo.

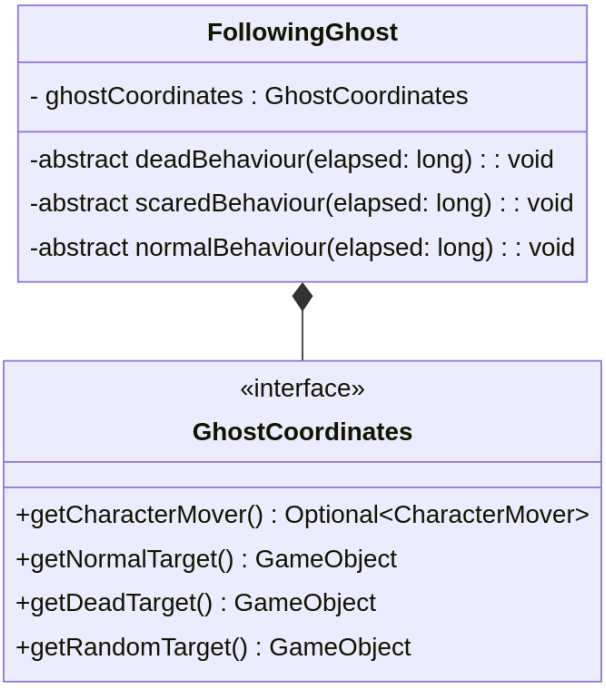

Figura 2.2.2.3 GhostsCoordinates FlowChart

CharacterMover

L'interfaccia CharacterMover fornisce un'astrazione per il movimento di un personaggio verso un qualsiasi oggetto di gioco. Nell'ambito di questa interfaccia, sono state sviluppate implementazioni specifiche per gestire diverse modalità di movimento. In particolare, una delle implementazioni consiste in un CharacterMover che avvicina muovendosi in linea d'aria un personaggio a un determinato oggetto calcolandone la distanza euclidea. Dall'altra parte, c'è un'implementazione avanzata che consente al personaggio di muoversi su un grafo per raggiungere un obiettivo.
Questa implementazione utilizza una combinazione di ***Decorator*** e ***Strategy*** : viene decorato un CharacterMover base con una strategia specifica di movimento su grafo, definendo così il vero e proprio spostamento.
In questo modo, l'uso del Decorator consente di estendere e modificare il comportamento di base di un CharacterMover aggiungendo la capacità di muoversi su un grafo, mentre il pattern Strategy viene utilizzato per rendere la decorazione più flessibile.
È fondamentale sottolineare che, per implementare la gestione del grafo, ho adottato la libreria ***JGraphT*** . Inoltre, per l'esecuzione dell'algoritmo di ricerca del percorso, ho utilizzato l'algoritmo A* (A-Star) già integrato nella libreria, parametrizzato in modo da adattarsi alla modalità di ricerca specificata tramite la modalità di approsimazione specificata dalla classe Approximator passato come riferimento all'algoritmo.

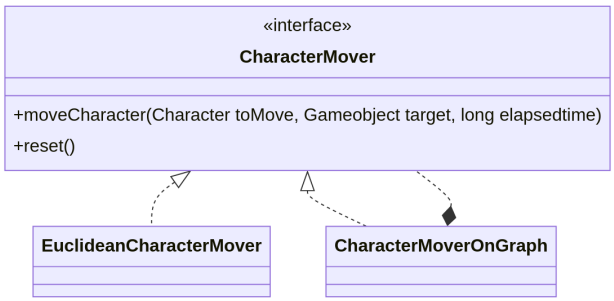

Figura 2.2.2.3 CharacterMover FlowChart

Factory fantasmi

Nel progetto abbiamo adottato una strategia di creazione degli oggetti basata sul pattern ***Factory***. Questa decisione è stata guidata dall'obiettivo di

delegare la responsabilità della creazione degli oggetti a una specifica entità, denominata fabbrica di oggetti. Questa fabbrica è stata progettata in modo da essere opportunamente parametrizzata, consentendo di gestire aspetti come le dimensioni di ogni oggetto in modo centralizzato. In particolare, questa approccio è stato esteso alla creazione dei fantasmi, dove è stata creata una factory dedicata ai fantasmi. Questa factory dei fantasmi è stata poi integrata nella factory generale del modello, seguendo un approccio coerente per la creazione di tutti gli oggetti presenti nella scena. Il pattern ***Builder*** era un'opzione alternativa, la scelta di utilizzare la factory sembrava più adatta alle esigenze del progetto data la limitata varietà di combinazioni di colori e comportamenti dei fantasmi, l'uso del pattern Builder potrebbe essere risultato eccessivo.

2.2.3 Zarrilli Antonio

Durante tutto il progetto, nella parte relativa al Model, mi sono occupato dei pickable, ovvero degli oggetti raccolti da Pacman per completare il livello e ottenere effetti sul mondo di gioco. Ho cercato di mantenere questa parte il più estensibile e fedele possibile ai principi KISS (Keep It Simple, Stupid) e DRY (Don't Repeat Yourself).

Pickable

Per la gestione dei Pickable, ho scelto di creare un'interfaccia da cui tutte le altre derivano. L'interfaccia principale è chiamata **Pickable**, che estende GameObject e condivide le sue stesse caratteristiche. Da **GameObject**, **Pickable** eredita le tre funzioni fondamentali, ovvero *getPosition* (che restituisce la posizione dell'oggetto), *getImage* (che restituisce l'immagine) e *getDimension* (che restituisce le dimensioni dell'oggetto).
Per quanto riguarda **Pickable**, questa ha tre funzioni specifiche che caratterizzano un oggetto raccoglibile da Pacman: *addPointsPickable* (che richiede un oggetto PacMan in ingresso a cui vengono aggiunti i punti), *getCollisionDimension* (per ritornare la dimensione del pickable scalata alla sua dimensione effettiva) e *getCollisionPoint* (che ritorna il punto da cui calcolare la collisione tra il pickable e qualsiasi altro oggetto).
Successivamente, c'è l'interfaccia **EffectPickable** che estende l'interfaccia **Pickable**. Questa si occupa dei pickable che hanno un effetto da applicare al gioco e dispone di due funzioni: *doEffect* (per eseguire l'effetto del raccoglibile sul PacMan passato o sui Ghost presenti nella lista) e *getEffectText* (per ottenere il testo dell'effetto appena applicato e visualizzarlo nella view successivamente).

Segue la classe **PickableImpl**, che implementa l'interfaccia **Pickable** gestendo le funzioni sia di **Pickable** che di **GameObject**.
C'è anche **EffectPickableImpl**, che implementa solo le funzioni di **EffectPickable**, estendendo a sua volta **PickableImpl** per evitare duplicazioni di codice inutili.
Infine, ci sono le classi effettive dei pickable che hanno l'effetto desiderato. Queste sono otto classi che estendono **EffectPickableImpl** e sovrascrivono la funzione *doEffect* per ottenere effetti diversi tra loro.

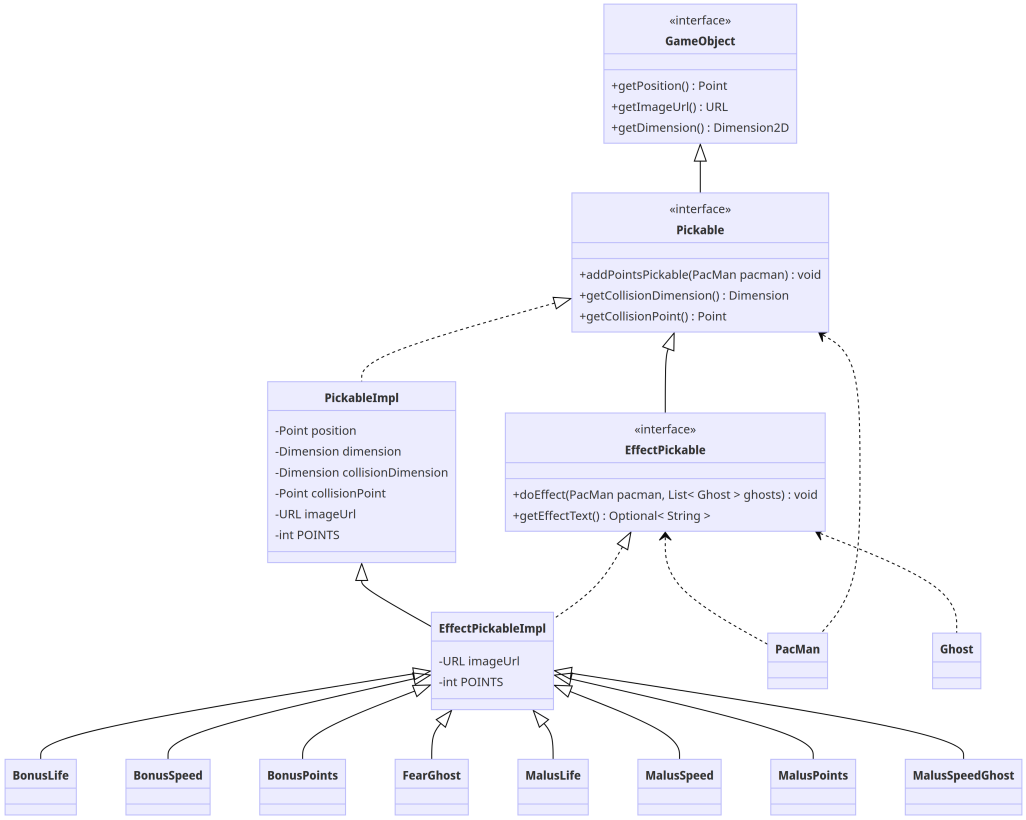

Figura 2.2.3.1 : Pickable Flowchart

Pickable Generator

Per la generazione dei pickable in base alla mappa, ho optato per la creazione di un'interfaccia chiamata **PickableGenerator**, che contiene quattro funzioni per la generazione e la gestione dei pickable.
La funzione *generateMap* si occupa di generare una mappa di pickable in base alla lista dei punti e alla dimensione passata. La funzione *getPickableList* restituisce la lista di pickable generati. La funzione *takePickable* gestisce la raccolta dei pickable tramite un punto che corrisponde alla loro posizione, un oggetto PacMan e i Ghost che verranno passati a ogni pickable per eseguire l'effetto (nel caso siano EffectPickable). Questa funzione restituisce una stringa opzionale perché nei casi in cui il pickable non ha effetto, la stringa non viene restituita. Infine, la funzione *finishedPickable* ritorna un booleano in base al fatto che siano stati raccolti tutti i pickable nella mappa o meno.
L'interfaccia **PickableGenerator** viene implementata dalla classe **PickableGeneratorImpl**, che utilizza un enumeratore chiamato **EffectChose** per la scelta basata su percentuali di spawn di tipo tra pickable normale e pickable con un effetto da applicare.

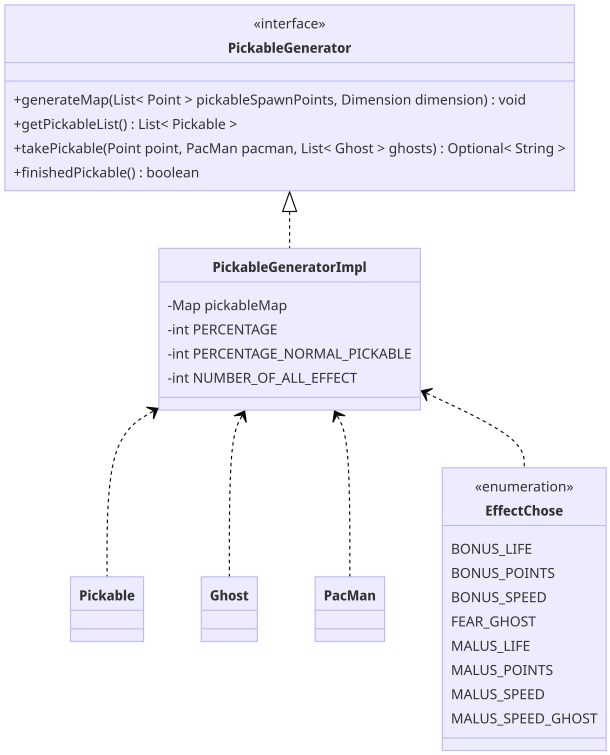

Figura 2.2.3.2 : PickableGenerator Flowchart

UI

Per la parte grafica, è necessario creare dei **GameObject** che rappresentino il testo da visualizzare, sia per il punteggio che per gli effetti dei raccoglibili, e un oggetto che rappresenti la vita di Pacman. Per farlo, ho creato un'interfaccia chiamata **GameObjectText** che estende **GameObject** e aggiunge due funzioni: *getText* (che restituisce il testo) e *setText* (che serve a modificarlo all'interno dell'oggetto)
Successivamente, ho implementato una classe chiamata **GameObjectTextImpl** che implementa sia le funzioni di **GameObjectText** che di **GameObject**.
Infine, ho creato una classe chiamata **GameObjectLife** che implementa l'interfaccia **GameObject** e si occupa di rappresentare una vita.

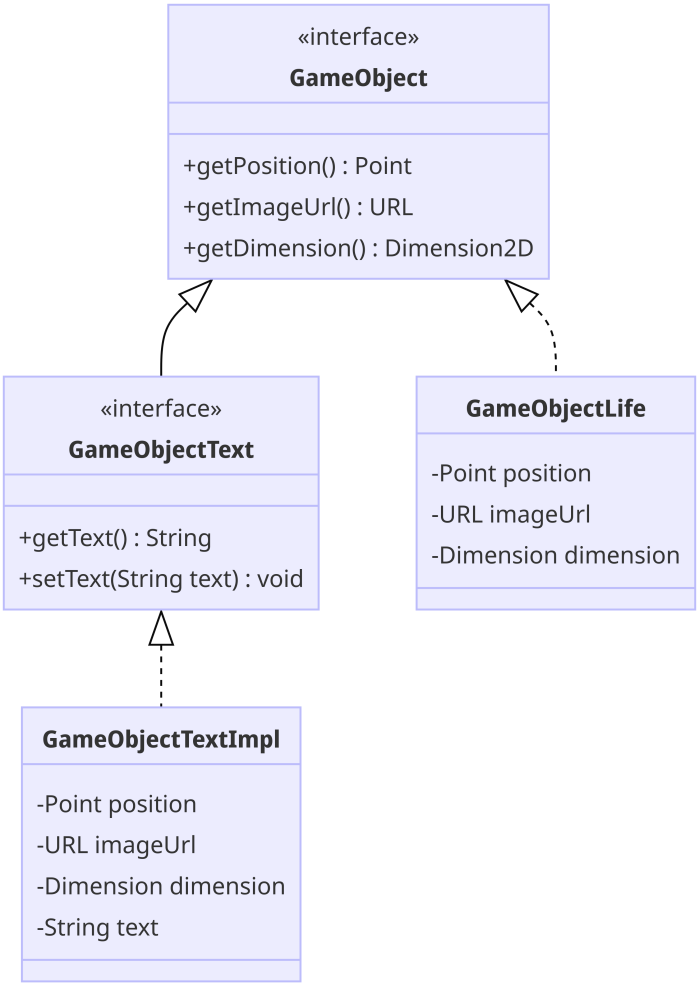

Figura 2.2.3.3 : UI Flowchart

2.2.4 Penserini Nicolò

Movimento dei personaggi

Problema Nonostante i diversi personaggi, cioè Pacman e i fantasmi, abbiano logiche di movimento diverse (Pacman è comandato dal giocatore, mentre i fantasmi "reagiscono" ai movimenti di PacMan), hanno degli elementi in comune che devono essere sfruttati per generalizzare il movimento di un qualsiasi personaggio, evitando ripetizioni di codice.

Soluzione Interfaccia Character con la possibilità di stabilire la direzione del personaggio, che può essere presente o no, e di aggiornare la posizione in base alla direzione e al tempo passato dall'ultimo aggiornamento.
Implementazione dell'interfaccia da parte di una classe astratta CharacterImpl che permette solo alle sottoclassi di stabilire la velocità del personaggio. L'aumento e la diminuzione della velocità sono metodi astratti che le sottoclassi dovranno implementare.

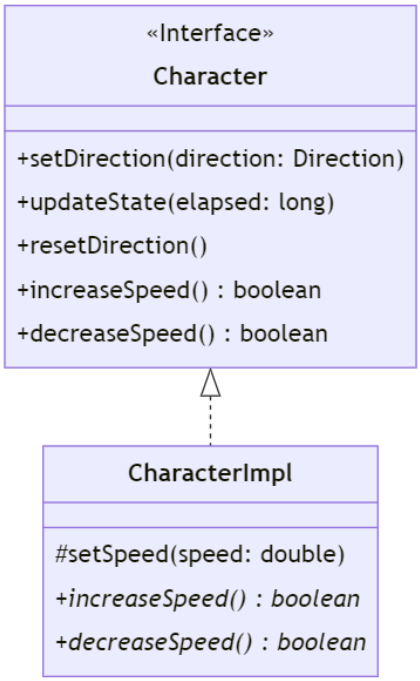

Figura 2.2.4.1 : Diagramma UML dell'interfaccia Character e della classe CharacterImpl

Collisioni tra oggetti

Problema All'interno del gioco sono presenti collisioni tra oggetti dello stesso tipo che devono essere controllate quando viene aggiornato lo stato del GameModel, in modo da verificare se avvengono e, in caso, generare gli eventi necessari alla gestione delle collisioni(ad esempio, aumento del punteggio di Pacman nel caso in cui mangi un oggetto raccoglibile).

Soluzione La classe GameScene mantiene al suo interno un oggetto di tipo CollisionChecker, un'interfaccia funzionale generica con un metodo che, dati due oggetti dello stesso tipo, restituisce true se i due oggetti stanno collidendo, false altrimenti. Sfruttando il pattern ***factory***, la responsabilità della creazione di questi oggetti è lasciata all'interfaccia CollisionCheckerFactory e alla classe che la implementa CollisionCheckerFactoryImpl, che forniscono un metodo per costruire un CollisionChecker di GameObject. Questa organizzazione permette di essere aperti alla creazione di CollisionChecker di tipi diversi semplicemente estendendo l'interfaccia CollisionCheckerFactory.

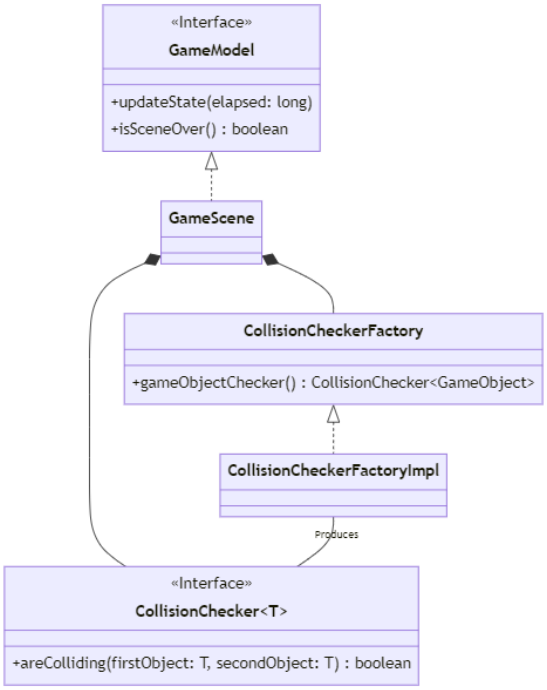

Figura 2.2.4.2: Diagramma UML delle relazioni tra GameModel e CollisionCheker.

Aggiunta di responsabilità a PacMan

Problema Oltre a una semplice implementazione dell'interfaccia PacMan che gestisce un personaggio che si muove libero in uno spazio "illimitato" e senza ostacoli, erano necessarie specializzazioni che permettessero a PacMan di muoversi in uno spazio con dei bordi(oltrepassati i quali si torna al bordo opposto) e con degli ostacoli, rappresentati dai muri.

Soluzione Attraverso il pattern ***decorator*** è stata realizzata un'interfaccia PacManDecorator che estende l'interfaccia PacMan aggiungendo il metodo correctPosition, che corregge la posizione di Pacman in base alle sue limitazioni(collisioni con muri o uscita dai confini di gioco). L'implementazione di questa interfaccia è realizzata da una classe atsratta PacManDecoratorImpl[1], che mantiene al suo interno un oggetto di tipo PacMan, modificando il suo movimento. Tutti i metodi della classe PacManDecoratorImpl delegano all'oggetto decorato, con l'aggiunta della correzione della posizione(metodo astratto la cui implementazione è lasciata alle sottoclassi) dopo l'aggiornamento dello stato. Questa soluzione pecca leggermente di ripetizione di codice, infatti quasi ogni metodo consiste semplicemente nella chiamata dello stesso metodo sull'oggetto decorato e la classe non estende da PacManImpl(implementazione di base dell'interfaccia PacMan), ma ho fatto questa scelta di design perchè semplifica la gestione dello stato di PacMan. L'uso del pattern decorator permette però di combinare diverse "limitazioni" al movimento di Pacman, creando oggetti con dinamiche di movimento complesse concatenando oggetti relativamente semplici. Una delle estensioni di PacManDecoratorImpl è PacManBordered, che tiene memorizzati al suo interno le posizioni dei bordi e corregge la posizione in caso esca dai bordi. Inoltre, questa classe cambia il comportamento del metodo respawn, per evitare che Pacman rinasca fuori dalla mappa.

[1]all'interno di questa classe è stato necessario sopprimere un warning di Spotbugs relativo all'esposizione dello stato interno. Ho preso questa decisione perchè volevo che i cambiamenti all'oggetto decorato influenzassero anche il decoratore. Inoltre, nella creazione di decorazioni di PacManDecorator ho usato chaining dei costruttori, quindi non è possibile modificare l'oggetto decorato perchè non se ne conserva il riferimento

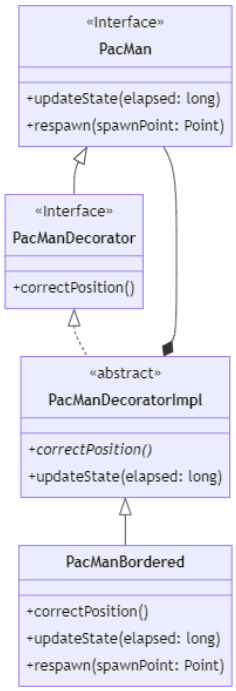

Figura 2.2.4.3: Rappresentazione UML del pattern decorator per modificare il movimento di Pacman

Gestione di Pacman e muri

Problema È necessario creare un'implementazione di Pacman che, data una lista di muri, si muova non oltrepassando i muri, ma fermandosi quando si scontra contro uno di essi. Questa lista deve anche poter essere cambiata durante il corso della partita, in caso di superamento di un livello e di conseguente passaggio al livello successivo.

Soluzione Per risolvere questo problema ho sfruttato il pattern decorator già utilizzato per aggiungere responsabilità di movimento a Pacman introducendo anche un'altra interfaccia che estende PacManDecorator: GamePacMan. Questa interfaccia ha un metodo che permette di cambiare la mappa in cui Pacman si muove specificando la nuova lista di muri e il nuovo punto di nascita. La classe PacManWalls quindi estende la classe astratta PacManDecoratorImpl e implementa l'interfaccia GamePacMan. Per correggere la posizione di PacMan in caso di collisione con un muro ho tenuto, all'interno della classe PacManWalls, un campo con l'ultima posizione di Pacman, aggiornato dopo ogni movimento. Questo permette, in caso di collisione con un muro, di correggere la posizione tornando a quella precedente. Questa scelta risulta un'approssimazione accettabile per il fatto che la velocità e l'intervallo di tempo tra due aggiornamenti sono valori contenuti.

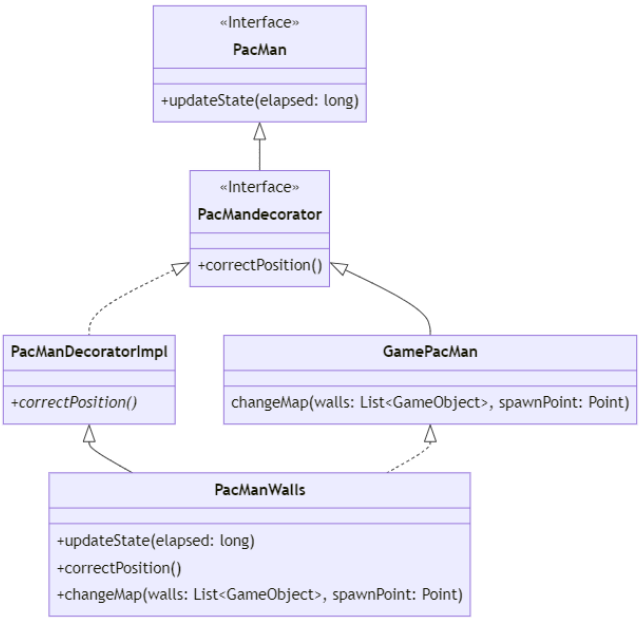

Figura 2.2.4.3: Rappresentazione UML delle classi per la gestione di Pacman e i muri

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per testare il funzionamento delle varie classi è stata utilizzata la libreria JUnit 5 per automatizzare il testing delle varie funzionalità dei componenti del modello.

3.1.1 Ambrogiani Alessandro

Per gestire le mappe, ho scelto di utilizzare un approccio basato su test per garantire un corretto funzionamento del sistema. Ho implementato due tipi di test:

Test di lettura da file: Questo test verifica se la lettura della mappa da file viene eseguita correttamente, senza errori. Viene verificato che il processo di lettura non generi eccezioni e che i dati della mappa siano letti correttamente dal file.

Test di inizializzazione della mappa: Una volta letta la mappa dal file, viene eseguito un secondo test che si assicura che la mappa venga inizializzata nel modo corretto. Questo test verifica che i punti di spawn, le coordinate e il tipo di oggetto siano correttamente assegnati in base ai dati letti dal file.
Classi di test realizzate:

- TestMapReader
- TestMapBuilder

3.1.2 Gambacorta Giuseppe

Ho eseguito test sulla parte relativa ai gestori dei movimenti dei caratteri verso un determinato oggetto in base alla loro posizione, logica sulla quale

si basa tutta la movimentazione dei fantasmi. Tuttavia, è importante sottolineare che la fase di testing specifica per il movimento dei caratteri non è stata effettuata personalmente da me, siccome sono stati implementati e testati da Penserini.

Parallelamente ho eseguito il testing di piccole implementazioni di timer. Queste implementazioni sono state attentamente valutate per garantire un corretto funzionamento e una gestione efficiente del tempo all'interno delle diverse funzionalità della mia parte di sistema.
Classi di test realizzate:

- TestMovingCharacterinDirection
- TestPositionApproximator
- TestTimer
- TestClockTimer

3.1.3 Zarrilli Antonio

Per ogni classe riguardante gli oggetti che PacMan può raccogliere ho creato un test che potesse evidenziare eventuali errori in fase di esecuzione, in modo da poter avere un codice sicuro e che dà i risultati aspettati.
Classi di test realizzate:

- BonusLifeTest
- BonusPointTest
- FearGhostTest
- MalusLifeTest
- MalusPointTest
- MalusSpeedGhostTest
- MalusSpeedTest
- PickableGeneretorImplTest
- PickableImplTest

3.1.4 Penserini Nicolò

Occupandomi della parte di model riguardante Pacman e le collisioni tra oggetti di gioco, i test che ho realizzato mirano a verificare in modo automatizzato il comportamento di oggetti di classi che aderiscono all'interfaccia PacMan, con particolare attenzione al movimento in caso di classi che implementano anche l'interfaccia PacManDecorator, e la verifica della presenza di collisioni tra diversi oggetti di gioco. La realizzazione di questi test mi ha permesso, in seguito ad alcuni cambiamenti nelle mie classi necessari per gestire i rapporti con le altre classi, di verificare che il comportamento delle classi modificate non fosse stato compromesso e rispettasse i requisiti per cui era stato creato.
Classi di test realizzate:

- TestSimplePacMan
- TestBorderedPacMan
- TestWallsPacMan
- TestCollisions

3.2 Note di sviluppo

3.2.1 Ambrogiani Alessandro

Feature avanzate del linguaggio e librerie che ho adoperato:

- Lambda:1)link 1, link2
- libreria sampled: link 1, link 2
- libreria esterna JgraphT:link 1

Crediti
Per questo progetto ho preso come riferimento la repository del prof ricci visionata al seguente link:`https://github.com/pslab-unibo/oop-game-prog-patterns-2022`. mentre per l'utilizzo della libreria Jgrapht ho fatto riferimento alla documentazione ufficiale presso il seguente link:`https://jgrapht.org/guide/UserOverview`.

3.2.2 Gambacorta Giuseppe

Feature avanzate del linguaggio e librerie che ho adoperato:

- **Optional**: utilizzato per non esporre esternamente a modifiche un oggetto che viene utilizzato da una classe esterna.
  Permalink di un esempio: link
- **libreria esterna JgraphT**: Inizialmente avevo utilizzato una lambda come parametrizzazione dell'algoritmo, poi sostituita utilizzando strategy con il metodo di approssimazione dell'attuale classe utilizzata per approssimare le posizioni.
  Permalink di un esempio: link

Crediti

- Repository ufficiale del corso tenuto dal professor Ricci: Repository Game-as-a-lab
- Repository personale dove è presente un piccolo engine basato su quello presentato dal professor Ricci e adattato per le nostre esigenze e preso poi come riferimento 2DGameEngine
- documentazione ufficiale di JgraphT presso il seguente link:`https://jgrapht.org/guide/UserOverview`
- Sito di riferimento per Pattern OOP: link

3.2.3 Zarrilli Antonio

Feature avanzate del linguaggio e librerie che ho adoperato:

- **Optional** Permalink di un esempio: link
- **Libreria Timer** Permalink di un esempio: link
- **Libreria Grafica Swing** Permalink di un esempio: link
- **Lambda** Permalink di un esempio:link

Crediti
Per la parte grafica ho preso spunto dal codice mostrato dal professor Ricci e anche da alcuni corsi su internet che mi hanno consentito di rendere la grafica più accattivante.

3.2.4 Penserini Nicolò

Utilizzo di Optional

Spesso utilizzate all'interno del codice. Permalink: link

Utilizzo di lambda expressions

Spesso utilizzate per implementazione di interfacce funzionali. Permalink: link

Utilizzo di stream

Utilizzate nell'implementazione di PacManWalls per verificare la collisione con i muri. Permalink: link

Creazione di interfaccia funzionale generica

Interfaccia CollisionChecker. Permalink: link

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Ambrogiani Alessandro

Il mio coinvolgimento in questo primo progetto di "grosse dimensioni" nel campo dello sviluppo software con un piccolo team è stata un'esperienza estremamente interessante e produttiva, che ha portato alla mia crescita personale in diversi ambiti. Durante il progetto, ho avuto l'opportunità di esplorare le dinamiche del lavoro di squadra e di comprendere l'importanza della collaborazione efficace. Abbiamo imparato come dividere il lavoro in modo equo, sfruttando le competenze e le preferenze individuali di ciascun membro del team. La comunicazione aperta e la condivisione delle idee sono state fondamentali per mantenere il progresso costante e superare eventuali ostacoli lungo il percorso. Uno degli aspetti più gratificanti del progetto è stato il continuo scambio di conoscenze e il supporto reciproco tra i membri del team. In caso di difficoltà o punti di stallo, ci siamo aiutati a vicenda offrendo consigli, suggerimenti e risorse per superare gli ostacoli. Questo ambiente di apprendimento collaborativo ha accelerato il nostro sviluppo e ci ha permesso di affrontare questo progetto con fiducia e determinazione. L'utilizzo di Git come sistema di controllo versione è stato un elemento chiave del nostro processo di sviluppo. Approfondire il funzionamento di Git, oltre a quanto appreso durante le lezioni, ci ha permesso di lavorare in modo efficiente, mantenere una cronologia dettagliata delle modifiche e gestire con successo le varie branche di sviluppo. È emersa la consapevolezza dell'importanza di studiare Git in modo più approfondito in futuro, data la sua cruciale rilevanza nello sviluppo software moderno.Partecipare a questo progetto mi ha aperto nuove prospettive sul mondo dello sviluppo software e mi ha fornito una solida base di conoscenze e competenze pratiche.

4.1.2 Gambacorta Giuseppe

L'esperienza di lavorare a un progetto in collaborazione con altri è stata davvero formativa. Ho potuto apprezzare i vantaggi del lavoro di gruppo, non solo in termini di condivisione delle conoscenze e delle competenze, ma anche di creatività e di problem solving. In passato mi è già capitato di lavorare in un team ma non abbiamo mai lavorato direttamente sulla stesso progetto.

Strumenti come Git e GitHub si sono rivelati essenziali per la gestione del codice e per una collaborazione efficiente. In particolare, ho capito l'importanza di un sistema di controllo versione, che permette di tenere traccia delle modifiche, di risolvere i conflitti e di ripristinare versioni precedenti del progetto. Pensavo che per prenderci mano ci avrei messo più tempo, ma al di la di qualche piccolo errore è andato tutto bene. Ho apprezzato molto la possibilità di poter sviluppare a parte una feature per poi accantonarla in attesa che tale feature sia effettivamente usabile all'interno del progetto.

Inoltre l'utilizzo di strumenti come l'analisi del codice statica, ha contribuito a migliorare la qualità del nostro lavoro, identificando potenziali errori e migliorando la leggibilità del codice, in futuro cercherò di integrare tool simili nei miei progetti futuri.

4.1.3 Zarrilli Antonio

Questo progetto è stato molto stimolante, mi ha permesso di lavorare in un piccolo team e di interagire con persone che hanno idee diverse. Inoltre, mi ha offerto l'opportunità di imparare e comprendere meglio le potenzialità di alcuni strumenti, come Git, che ci hanno consentito di gestire il codice in modo più comodo e pratico. Il progetto mi ha anche permesso di mettermi alla prova e di sviluppare delle competenze che sono state molto utili sia per la realizzazione del progetto, sia per migliorare le mie capacità nella scrittura del codice, nell'analisi precedente e nella progettazione. Inoltre, mi ha dato un'idea più concreta di come sarà un futuro lavoro in questo settore, seppur in modo limitato, considerando che si tratta di un progetto di dimensioni ridotte e un team composto da sole 4 persone.

4.1.4 Penserini Nicolò

All'interno del gruppo il mio ruolo è stato occuparmi delle feature riguardanti Pacman e delle collisioni tra gli oggetti di gioco. Il lavoro è stato molto stimolante e mi è servito molto lavorare soprattutto alla parte di progettazione, che non avevo mai svolto così dettagliatamente. Il lavor di gruppo

mi ha permesso di approfondire l'uso di strumenti importanti come Git e GitHub, che sicuramente mi saranno utili in futuro. La stesura della relazione mi ha aiutato a cercare di spiegare il mio modo di lavorare durante tutto il progetto. Tra i punti di forza del mio lavoro penso di poter citare la realzizzazione di interfacce e classi per gestire le collisioni degli oggetti di gioco e anche la realizzazione, attraverso il pattern decorator, di funzionalità di movimento più complesse per Pacman. Una delle parti che invece mi ha causato più difficoltà è stata la realizzazione del movimento di PacMan con i muri, infatti, se in futuro volessi portare avanti questo progetto, mi vorrei concentrare sul rendere il movimento di Pacman più simile al gioco originale di cui la nostra applicazione è un clone.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Ambrogiani Alessandro

4.2.2 Gambacorta Giuseppe

4.2.3 Zarrilli Antonio

Il corso è stato ottimo, considerando la vasta quantità di nuovi concetti da apprendere e le ore di lezione dedicate. È stato un corso completo che fornisce una panoramica approfondita della programmazione ad oggetti, consentendo di sviluppare sia applicazioni gestionali che giochi, come nel nostro caso. Inoltre, ho trovato i laboratori estremamente utili, sia per la varietà degli esercizi proposti, sia per le soluzioni fornite in modo graduale, il che permette di investire più tempo nella comprensione dello sviluppo del codice anche quando la soluzione non è immediata. Questo approccio favorisce anche l'interazione con i compagni di corso e la possibilità di imparare nuovi concetti. Infine, il progetto è stato fondamentale per consolidare le conoscenze acquisite durante le lezioni sia in aula che nei laboratori.

4.2.4 Penserini Nicolò

La realizzazione del progetto è impegnativa e, per questo, permette di affrontare, anche con l'aiuto degli altri componenti del gruppo, problemi comuni nello sviluppo di software consentendo di sviluppare l'approccio giusto per questo tipo di lavori e fornire un'idea del corretto metodo di lavoro in gruppo. La correzione degli esercizi effettuata in laboratorio dal professore o dai

tutor è molto utile per controllare errori nella risoluzione o per farsi suggerire strategie migliori di quella adottata, ma, a causa del tempo ridotto a disposizione e del gran numero di studenti presente alle lezioni, spesso è difficile riuscire a svolgere e correggere tutti gli esercizi proposti.

Appendice A

Guida utente

A.0.1 Avvio dell'applicativo

Per avviare l'applicazione occorre aver installato all'interno della propria macchina una versione di Java 17 o più recente.

- Posizionarsi all'interno della cartella contenente il jar relativo all'applicazione (solitamente chiamato OOP23-pac-man-all.jar).
- Eseguirlo tramite il comando java -jar JarPackage.jar.

A.0.2 Menù di gioco

Nel menù di gioco sono presenti divesi elementi, in alto al centro è presente il nome del nostro gioco (Pac-Man 2.0), poi subito sotto al centro dello schermo sono presenti 2 tasti, uno start che avvia il gioco e uno quit che lo chiude. Sotto ai tasti è presente la spiegazione minimale dei tasti che si utilizzano per giocare.

Figura A.0.2.1 : Menu iniziale

A.0.3 Schermata di gioco

Nella schermata di gioco è possibile premere 4 tasti che corrispondono alle frecce direzionali della tastiera, ciascuno dei quali fa muovere Pacman nella rispettiva direzione. Inoltre, quando Pacman raccoglie un oggetto, viene aggiornato il punteggio in alto a sinistra dello schermo. Nel caso in cui raccoglie un oggetto speciale, oltre al punteggio, viene visualizzato anche il tipo di effetto applicato. In alto a destra sono visibili le icone delle vite e il resto del pannello è riservato alla visualizzazione del campo di gioco e dei vari componenti. Al termine degli oggetti nella mappa, viene generata un'altra mappa con nuovi oggetti e la partita continua. Al termine delle vite di Pacman, la partita termina.

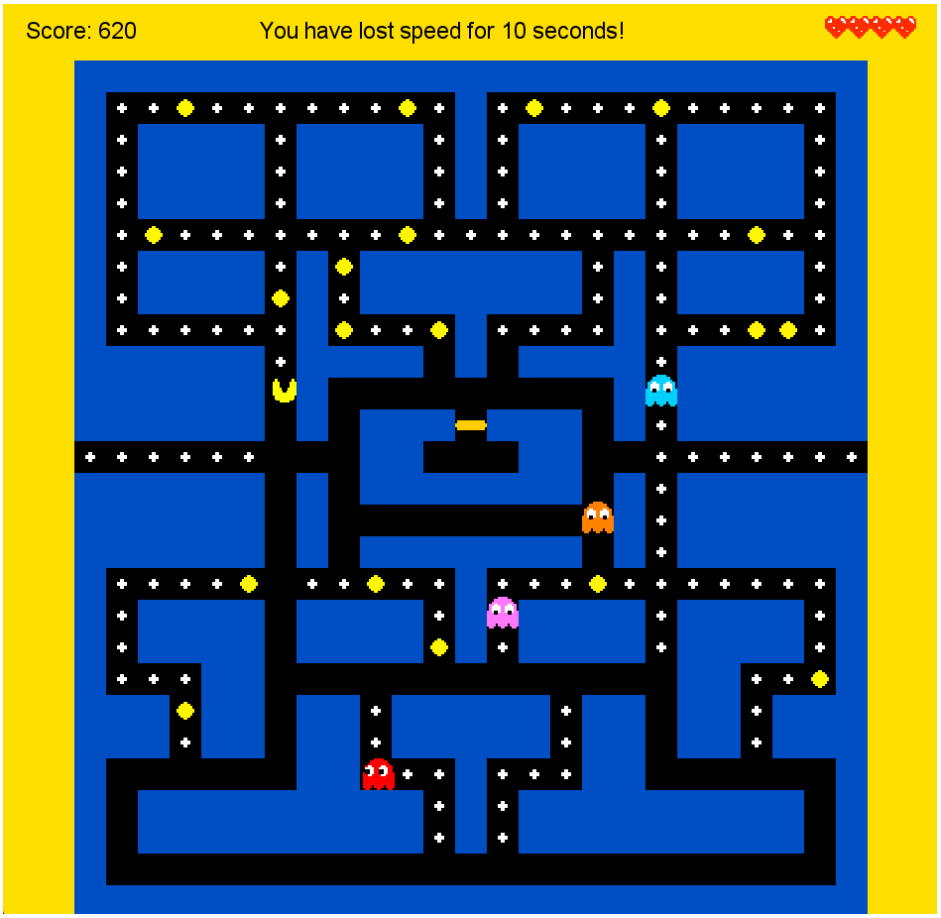

Figura A.0.3.1 : Schermata di gioco

A.0.4 Schermata di fine gioco

Quando il gioco è terminato dalla perdita di tutte le vite di Pacman allora si visualizza una schermata con al centro il punteggio effettuato e al di sotto 2 tasti, uno con scritto restart che ti permette di cominciare una nuova partita e un altro con scritto quit che chiude il gioco.

Figura A.0.4.1 : Schermata di fine gioco

Appendice B

Esercitazioni di laboratorio

B.0.1 alessandr.ambrogian4@studio.unibo.it

B.0.2 giuseppe.gambacorta2@studio.unibo.it

B.0.3 antonio.zarrilli@studio.unibo.it

- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=147598#p209269`
- Laboratorio 09: `https://virtuale.unibo.it/mod/forum/discuss.php?d=149231#p211574`

B.0.4 nicolo.penserini@studio.unibo.it

- Laboratorio 07: `https://virtuale.unibo.it/mod/forum/discuss.php?d=147598#p209252`
- Laboratorio 08: `https://virtuale.unibo.it/mod/forum/discuss.php?d=148025#p210230`
- Laboratorio 10: `https://virtuale.unibo.it/mod/forum/discuss.php?d=150252#p212492`