Pacman

Relazione per il progetto di
Programmazione e Modellazione ad Oggetti

Giaconi Christian
314045
Rossi Giacomo
314671

Sessione autunnale a.a. 2023-24

Sommario

1. Analisi

1.1 Obiettivi

L’obiettivo principale di questo progetto è la creazione di un'applicazione che riproduca fedelmente il celebre gioco arcade **Pacman**. Il gioco consente ai giocatori di controllare Pacman, il personaggio protagonista, all'interno di un labirinto ricco di sfide. Il fine ultimo è raccogliere tutti i punti sparsi nel labirinto, evitando al contempo i fantasmi, che sono i principali antagonisti del gioco.

Le entità principali coinvolte sono:

- **Pacman**: Il personaggio principale controllato dal giocatore, il cui compito è muoversi attraverso il labirinto per raccogliere i punti e evitare i fantasmi.
- **Fantasmi**: I nemici che inseguono Pacman con l'intento di catturarlo. Se un fantasma riesce a raggiungere e toccare Pacman, quest'ultimo perde una vita.

Il labirinto è strutturato come una griglia, dove ogni cella può contenere tre diversi tipi di componenti di gioco:

- **Small Dots**: Punti piccoli che Pacman deve raccogliere per aumentare il punteggio. Questi sono distribuiti in tutto il labirinto e sono essenziali per il progresso nel gioco.
- **Big Dots**: Conosciuti anche come "power pellets", sono oggetti speciali che conferiscono a Pacman un'abilità temporanea di mangiare i fantasmi. Quando Pacman consuma un *Big Dot*, i fantasmi diventano vulnerabili per un periodo limitato. Durante questo tempo, Pacman può mangiarli per guadagnare punti bonus. Dopo che il potere del *Big Dot* svanisce, i fantasmi ritornano ai loro punti di partenza e ricominciano a inseguire Pacman.
- **Muri**: Strutture fisse che costituiscono la configurazione del labirinto e impediscono il movimento di Pacman e dei fantasmi. I muri sono essenziali per definire i percorsi e le strategie di movimento all'interno del labirinto.

L'obiettivo del giocatore è guidare Pacman attraverso il labirinto, raccogliendo tutti i punti e evitando i fantasmi. La cattura da parte di un fantasma comporta la perdita di una vita. La dinamica di gioco è arricchita dal consumo dei *Big Dot*, che non solo conferisce a Pacman un vantaggio temporaneo, ma introduce anche una strategia di gioco basata sulla gestione del tempo e sulla massimizzazione dei punti bonus.

1.2 Requisiti

1.2.1 Requisiti Funzionali

- **Movimento di Pacman**: Il giocatore deve essere in grado di controllare Pacman utilizzando i tasti direzionali. Pacman deve muoversi attraverso la griglia senza attraversare i muri che delimitano il labirinto.
- **Strategie dei Fantasmi**: Ogni fantasma segue una propria strategia di movimento. I fantasmi possono adottare modalità diverse, come l'inseguimento di Pacman (Chase Mode) o la fuga quando Pacman attiva la supermode (Flee Mode).
- **Supermode**: Quando Pacman raccoglie un Big Dot, i fantasmi diventano temporaneamente vulnerabili. Questo consente a Pacman di mangiarli e ottenere punti bonus. La vulnerabilità dei fantasmi dura per un periodo limitato, dopo il quale essi ritornano al loro stato normale e ricominciano a inseguire Pacman.
- **Gestione del punteggio**: Il punteggio del giocatore aumenta quando Pacman raccoglie i *Small Dots* e mangia i fantasmi durante la supermode. Il sistema di punteggio deve essere aggiornato in tempo reale per riflettere le azioni del giocatore.
- **Vittoria e sconfitta**: Il gioco termina con una vittoria quando Pacman raccoglie tutti i punti presenti nel labirinto. In caso di sconfitta, il gioco termina quando Pacman perde tutte le sue vite. Dopo una vittoria, il gioco può continuare fino a quando il giocatore non subisce una sconfitta.

1.2.2 Requisiti Non Funzionali

- **Performance**: L'applicazione deve garantire una risposta rapida agli input del giocatore, assicurando fluidità nel movimento di Pacman e nelle interazioni generali del gioco. La velocità di risposta deve essere ottimale per un'esperienza di gioco senza lag.
- **Manutenibilità**: Il codice deve essere progettato in modo modulare e seguire i principi della programmazione orientata agli oggetti. Questo approccio facilita la manutenzione e le eventuali estensioni future del software, garantendo una base solida per eventuali miglioramenti.
- **Interfaccia Grafica**: L'interfaccia utente deve essere intuitiva e facile da comprendere. Deve mostrare chiaramente Pacman, i fantasmi, i punti e il punteggio del giocatore. L'interfaccia deve essere progettata per offrire un'esperienza di gioco coinvolgente e visivamente piacevole.

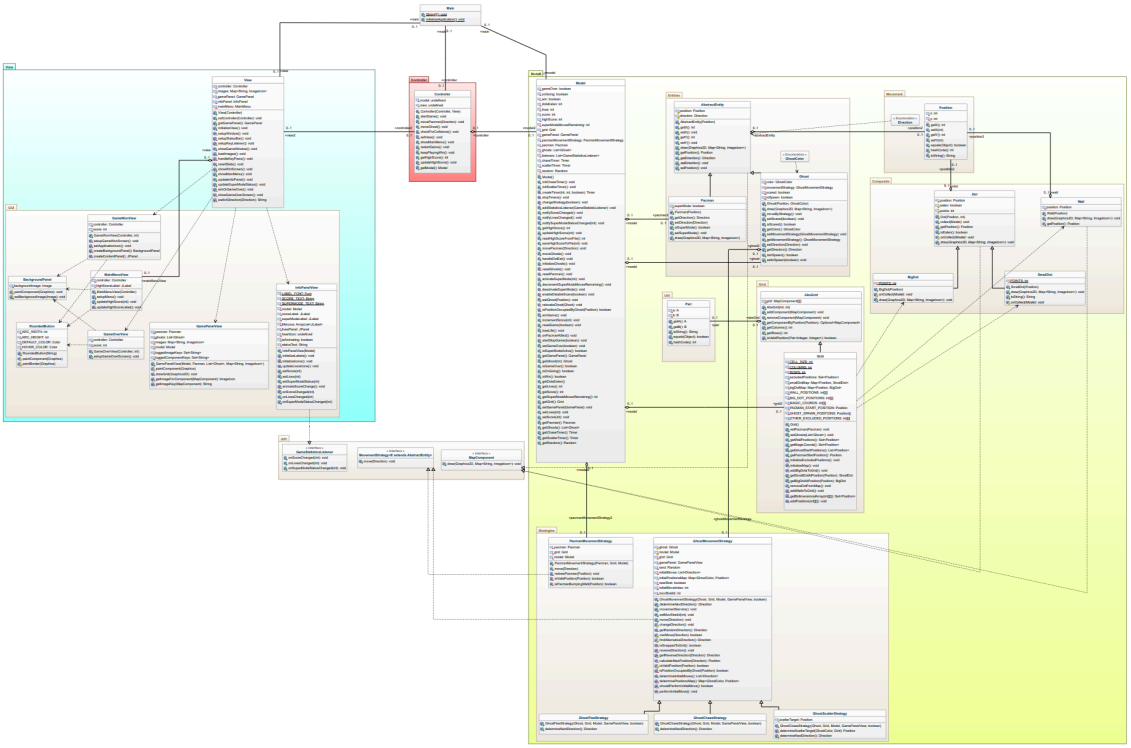
1.3 Modello del Dominio

Il dominio del gioco contiene diverse entità che interagiscono all'interno del labirinto. Queste includono:

- **Pacman**: Il personaggio principale controllato dal giocatore. Il suo obiettivo è raccogliere punti e evitare i fantasmi.
- **Fantasmi**: Nemici che seguono strategie di movimento specifiche. Ogni fantasma ha un comportamento distintivo e cerca di catturare Pacman.
- **Small Dots e Big Dots**: I punti che Pacman deve raccogliere. I *Small Dots* aumentano il punteggio, mentre i *Big Dots* attivano la supermode, rendendo i fantasmi vulnerabili.
- **Muri**: Barriere fisse che costituiscono la struttura del labirinto. I muri impediscono il movimento di Pacman e dei fantasmi, definendo i percorsi nel labirinto.
- **Griglia**: La rappresentazione del labirinto, organizzata in una griglia di dimensioni predefinite (21x19). La griglia definisce l'ambiente di gioco e le posizioni delle diverse entità.

Diagramma UML del Modello del Dominio

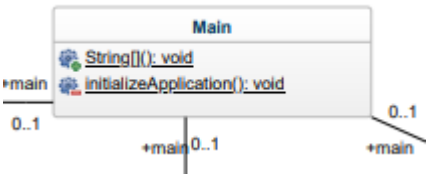
Il diagramma UML seguente rappresenta le relazioni tra le entità principali del gioco, evidenziando le loro interazioni e comportamenti.



2. Design

2.1 Architettura

Il progetto Pacman adotta il pattern architetturale MVC (Model-View-Controller) per garantire una chiara separazione tra la logica del gioco, la visualizzazione grafica e l'elaborazione degli input del giocatore. Questo approccio facilita la modularità del codice, migliorandone la manutenibilità e l'estensibilità futura.



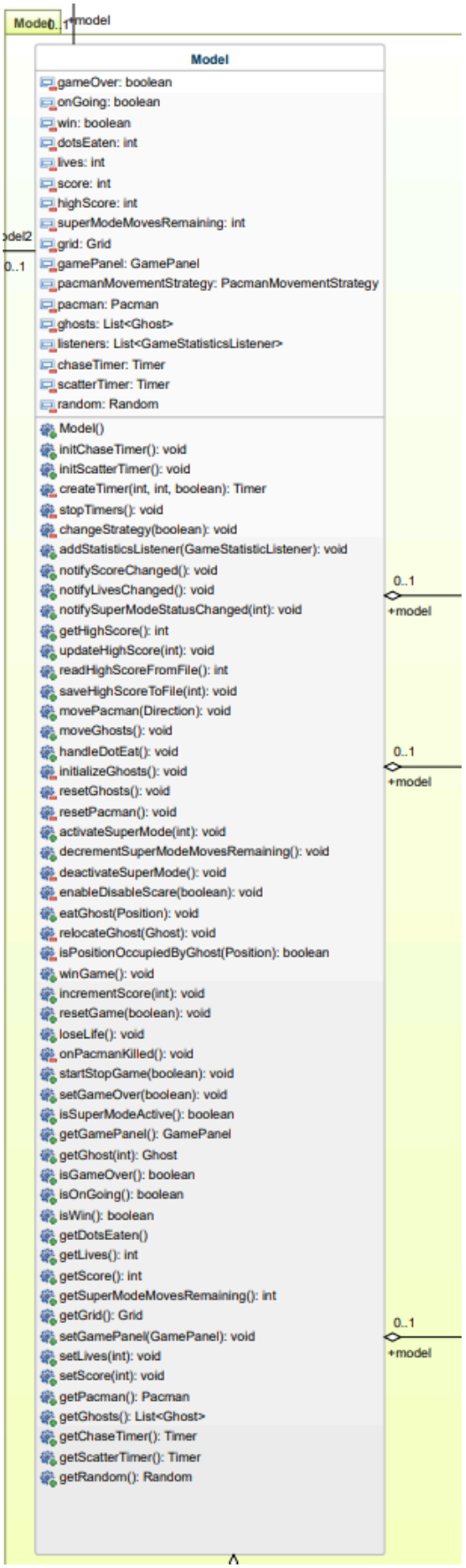
Model.java

2.1.1 Model

Il Model (modello) è responsabile della gestione dello stato del gioco e delle regole operative. Esso comprende la logica relativa a Pacman, ai fantasmi, ai punti e alle vite. Inoltre, si occupa della gestione della supermode, che si attiva quando Pacman raccoglie un *Big Dot*, e dell'aggiornamento del punteggio del giocatore in base alle interazioni tra Pacman e gli oggetti presenti nel labirinto.

Le componenti principali nel Model includono:

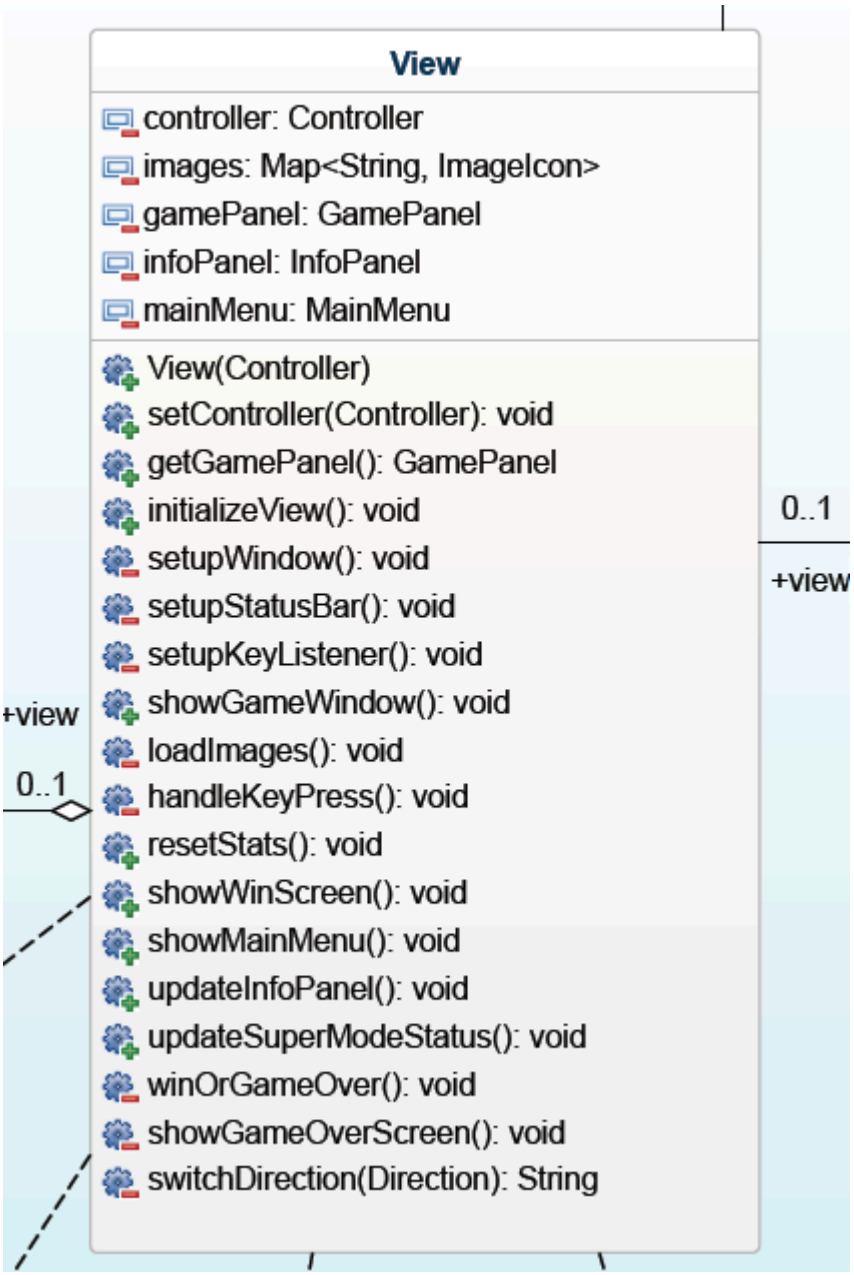
- **PacmanMovementStrategy**: Gestisce il movimento di Pacman e le sue interazioni con gli altri elementi del gioco.
- **Ghosts e Timer**: Gestiscono il comportamento dei fantasmi tramite metodi come `enableDisableScareMode()`, applicando diverse strategie di movimento a seconda della modalità in cui si trovano (inseguimento o fuga).
- **Small Dot e Big Dot**: Vengono conteggiati in `dotsEaten` i punti che Pacman raccoglie. I *Small Dots* aumentano il punteggio, mentre i *Big Dots* oltre a conferire punti attivano anche la supermode.
- **Grid**: Rappresenta la struttura del labirinto, definendo la disposizione dei muri e degli oggetti di gioco.


Model
0..1
+model
Model
gameOver: boolean
onGoing: boolean
win: boolean
dotsEaten: int
lives: int
score: int
highScore: int
superModeMovesRemaining: int
grid: Grid
gamePanel: GamePanel
pacmanMovementStrategy: PacmanMovementStrategy
pacman: Pacman
ghosts: List<Ghost>
listeners: List<GameStatisticsListener>
chaseTimer: Timer
scatterTimer: Timer
random: Random
Model()
initChaseTimer(): void
initScatterTimer(): void
createTimer(int, int, boolean): Timer
stopTimers(): void
changeStrategy(boolean): void
addStatisticsListener(GameStatisticListener): void
notifyScoreChanged(): void
notifyLivesChanged(): void
notifySuperModeStatusChanged(int): void
getHighScore(): int
updateHighScore(int): void
readHighScoreFromFile(): int
saveHighScoreToFile(int): void
movePacman(Direction): void
moveGhosts(): void
handleDotEat(): void
initializeGhosts(): void
resetGhosts(): void
resetPacman(): void
activateSuperMode(int): void
decrementSuperModeMovesRemaining(): void
deactivateSuperMode(): void
enableDisableScare(boolean): void
eatGhost(Position): void
relocateGhost(Ghost): void
isPositionOccupiedByGhost(Position): boolean
winGame(): void
incrementScore(int): void
resetGame(boolean): void
loseLife(): void
onPacmanKilled(): void
startStopGame(boolean): void
setGameOver(boolean): void
isSuperModeActive(): boolean
getGamePanel(): GamePanel
getGhost(int): Ghost
isGameOver(): boolean
isOnGoing(): boolean
isWin(): boolean
getDotsEaten()
getLives(): int
getScore(): int
getSuperModeMovesRemaining(): int
getGrid(): Grid
setGamePanel(GamePanel): void
setLives(int): void
setScore(int): void
getPacman(): Pacman
getGhosts(): List<Ghost>
getChaseTimer(): Timer
getScatterTimer(): Timer
getRandom(): Random
0..1
+model
0..1
+model
0..1
+model

Model.java

2.1.2 View

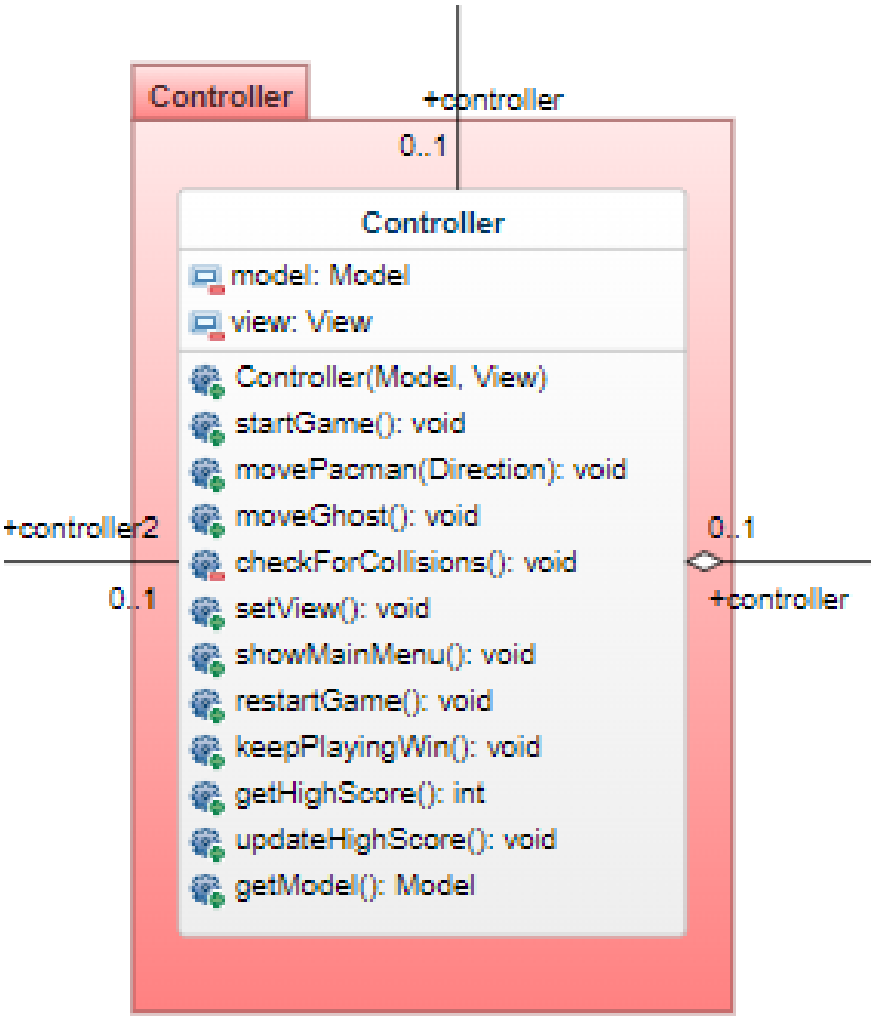
La View (vista) è incaricata della visualizzazione grafica del gioco. Essa rappresenta il labirinto, Pacman, i fantasmi, e gli altri oggetti di gioco, come i punti e le vite. La vista riceve aggiornamenti dal Model tramite il Controller e riflette graficamente i cambiamenti, mantenendo l'interfaccia utente sempre sincronizzata con lo stato corrente del gioco. Questo permette una rappresentazione visiva precisa e aggiornata degli eventi e delle azioni nel gioco.



View.java

2.1.3 Controller

Il Controller (controllore) interpreta gli input dell'utente, come i comandi per muovere Pacman, e li traduce in azioni che influenzano lo stato del Model. Esso è responsabile dell'applicazione delle regole del gioco in base alle interazioni tra le varie entità. Il controller gestisce anche l'attivazione della supermode quando Pacman raccoglie un *Big Dot*, coordinando le modifiche dello stato del gioco in risposta agli input del giocatore e alle dinamiche del gioco.



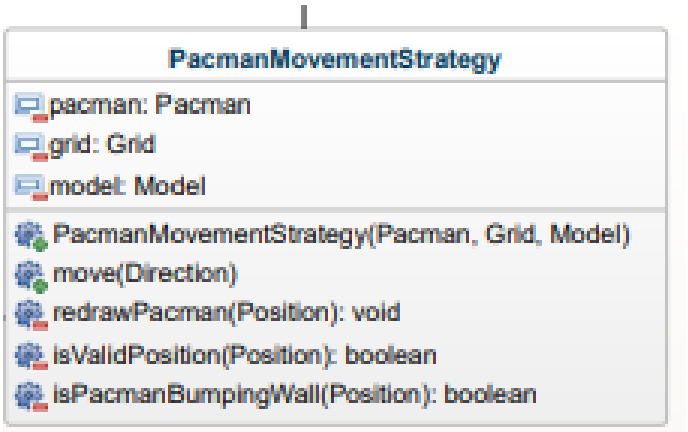
Controller.java

2.2 Design Dettagliato

2.2.1 Christian Giaconi

2.2.1.1 Movimenti e Azioni di Pacman

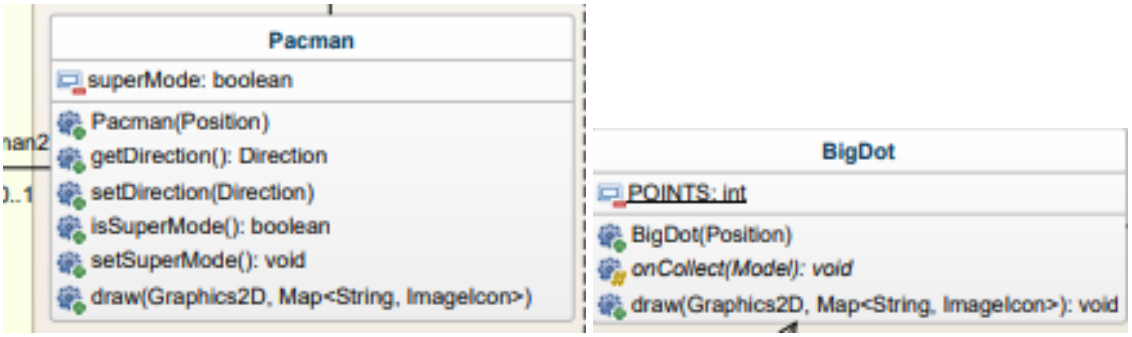
Il sistema di movimento di Pacman è stato realizzato utilizzando una griglia che consente di gestire in modo preciso gli spostamenti del personaggio all'interno del labirinto. Ogni input del giocatore viene elaborato per verificare se la cella di destinazione è libera da ostacoli. Questo approccio garantisce che Pacman possa muoversi solo in spazi percorribili, prevenendo movimenti non consentiti e gestendo adeguatamente le collisioni con i muri.



PacmanMovementStrategy.java

2.2.1.2 Supermode e Gestione del Tempo

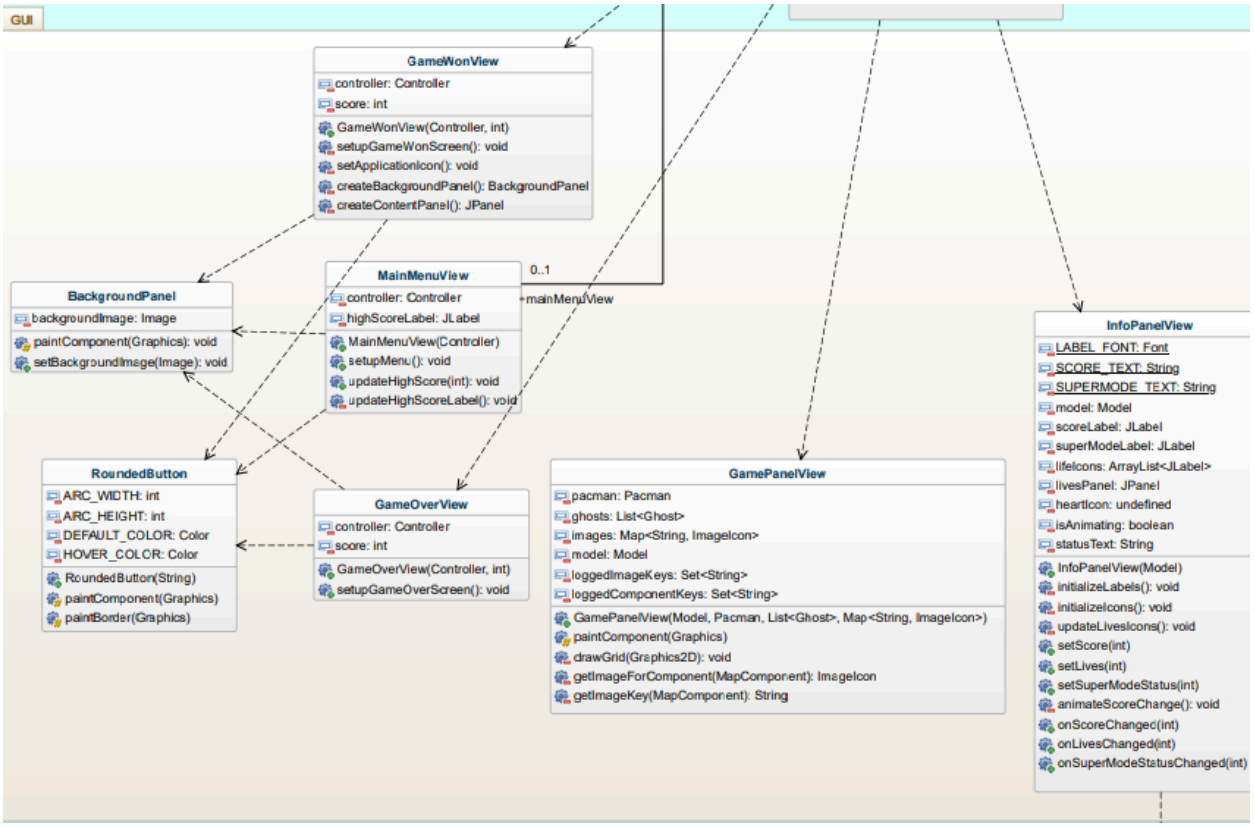
Per quanto riguarda la supermode, è stato implementato nella classe `model` un meccanismo che attiva una modalità temporanea quando Pacman mangia un *Big Dot*. Durante questa fase, i fantasmi diventano vulnerabili e il loro comportamento cambia. La durata della supermode è determinata da un numero prestabilito di mosse, dopo il quale i fantasmi ritornano al loro comportamento normale. Questo sistema assicura una transizione fluida tra la modalità normale e quella vulnerabile dei fantasmi.



Pacman.java e BigDot.java

2.2.1.3 GUI

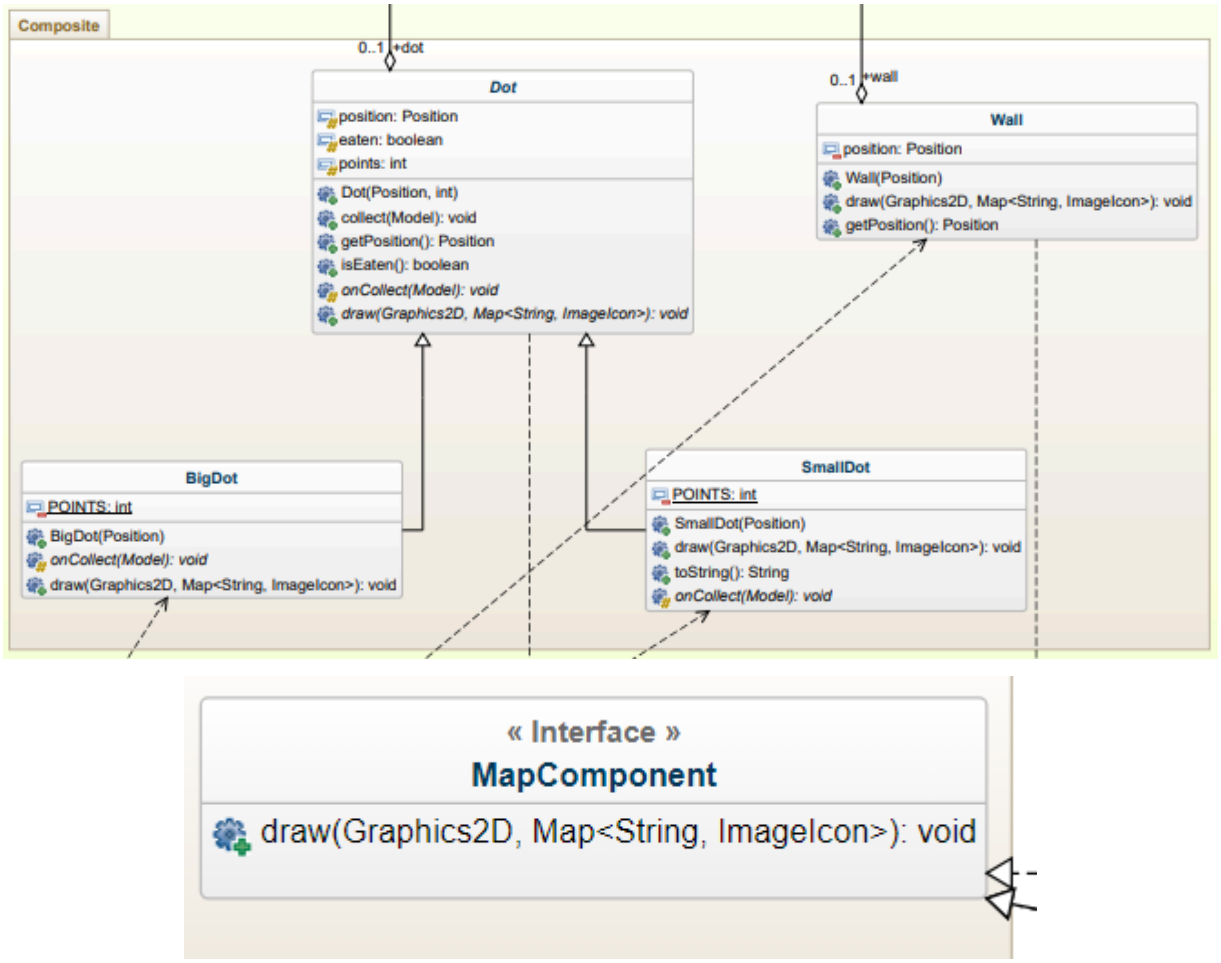
L'interfaccia grafica del gioco è stata progettata e implementata con attenzione ai dettagli visivi. Gli elementi principali, come il labirinto, Pacman, i fantasmi e i punti, sono stati creati e posizionati per garantire un'esperienza utente chiara e reattiva. La GUI riflette accuratamente lo stato del gioco, con un design che facilita l'interazione e la comprensione visiva.



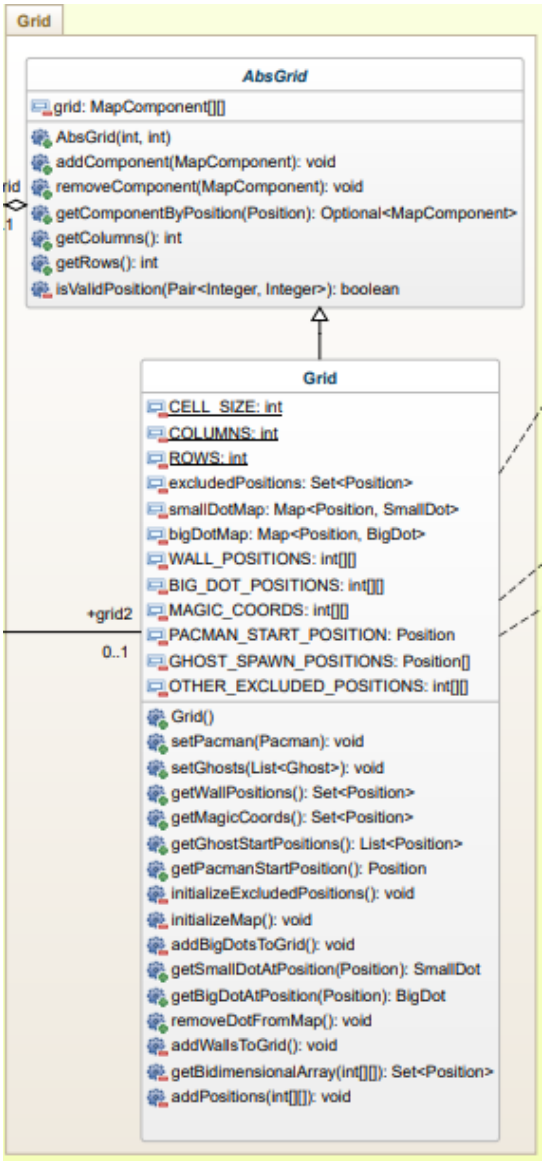
Package GUI

2.2.1.4 Gestione della Griglia e degli Oggetti

La gestione della griglia di gioco è stata realizzata tramite la classe `Grid`, che rappresenta il labirinto come una matrice. Questa griglia è responsabile della posizione di Pacman, dei fantasmi e degli oggetti di gioco, e aggiorna le posizioni in tempo reale. È stato adottato il Composite Pattern per gestire in modo uniforme e coerente i diversi oggetti presenti nella griglia, come punti, muri, Pacman e fantasmi.



Package Composite ed interfaccia MapComponent implementata da `Dot` e `Wall`

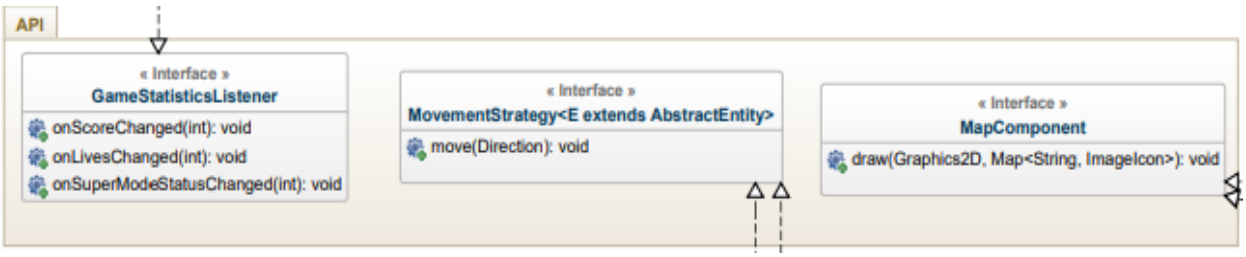


Package Grid

2.2.2 Giacomo Rossi

2.2.2.1 API

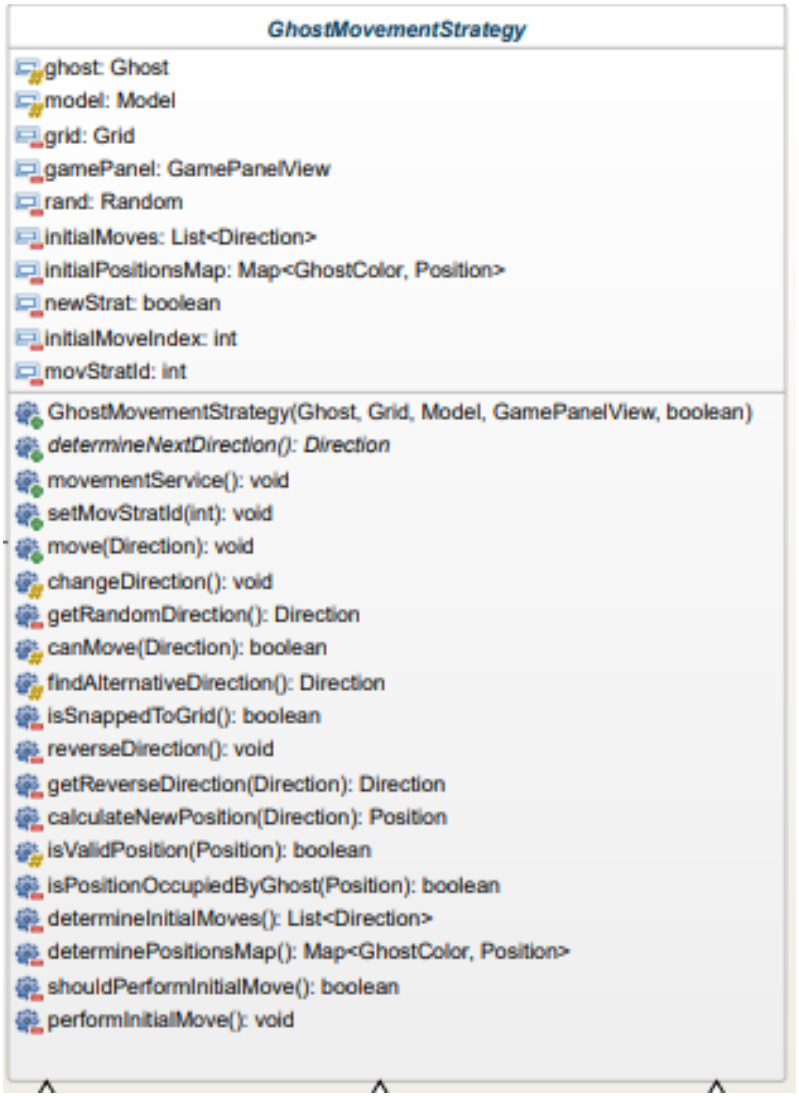
Le API sono state utilizzate per gestire in modo efficace le operazioni sui dati e le interazioni tra gli oggetti di gioco. Questo ha permesso di semplificare e rendere più efficienti le operazioni legate alla gestione delle entità e delle loro interazioni nel contesto del gioco.



Package API

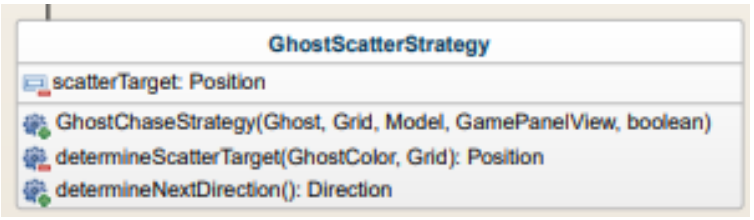
2.2.2.2 Movimenti dei Fantasmi

Per i movimenti dei fantasmi, è stato adottato il Strategy Pattern, che ha consentito di implementare diverse strategie di movimento per i fantasmi, come l'inseguimento di Pacman, la fuga durante la supermode e il movimento in aree specifiche del labirinto. Ogni fantasma segue una strategia separata, che può essere modificata dinamicamente in risposta agli eventi di gioco. Questo approccio ha facilitato la gestione modulare e manutenibile delle diverse logiche di movimento dei fantasmi.

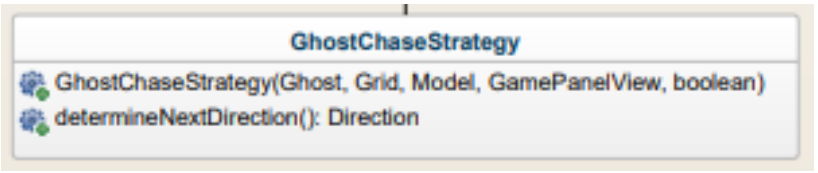


GhostMovementStrategy.java

GhostFleeStrategy.java



GhostScatterStrategy.java



GhostChaseStrategy.java

3. Sviluppo

3.1 Testing Automatizzato

3.1.1 Christian Giaconi

Per garantire la correttezza delle principali funzionalità del gioco Pacman, è stata sviluppata una suite di test automatizzati utilizzando JUnit. L'adozione dei test automatizzati ha permesso di verificare il comportamento del gioco in diversi scenari, assicurando che ciascun componente funzionasse come previsto e che l'integrazione tra i vari moduli fosse fluida.

Le principali aree coperte dai test includono:

- **Interazioni con i punti**: I test assicurano che il punteggio venga incrementato correttamente quando Pacman raccoglie i *Small Dots*. Inoltre, verificano che la *supermode* venga attivata correttamente dopo che Pacman mangia un *Big Dot*, con la conseguente vulnerabilità dei fantasmi.

- **Strategie di movimento dei fantasmi**: I test confermano che i fantasmi seguano le loro strategie di movimento predefinite, come la modalità Chase e Scatter. Viene verificato anche il ritorno dei fantasmi alla posizione iniziale dopo essere stati mangiati

- **Gestione delle Vite e Game Over**: `testLoseAllLivesEndsGame` verifica che il gioco gestisca correttamente la perdita delle vite di Pacman e che lo stato di "Game Over" venga attivato una volta esaurite le vite. Il test inizia controllando che Pacman parta con 3 vite, quindi simula la perdita progressiva di vite, riducendo il conteggio a 2, 1 e infine 0. Alla perdita della terza vita, viene verificato che il gioco sia terminato, assicurando che il metodo `loseLife()` del model funzioni correttamente e che lo stato di Game Over venga impostato quando Pacman ha esaurito tutte le vite. Questo test è fondamentale per garantire che il flusso di gioco relativo alla gestione delle vite si comporti come previsto.

- **Movimento di Pacman**: I test verificano che Pacman si muova correttamente all'interno del labirinto, rispettando le regole di collisione con i muri. Viene inoltre testata la corretta gestione del blocco del movimento quando Pacman cerca di attraversare un muro.

- **Gestione della supermode**: I test verificano che la supermode si attivi correttamente e che i fantasmi cambino comportamento diventando vulnerabili per un tempo limitato, per poi riprendere il loro comportamento originale al termine della modalità.

L'utilizzo di JUnit ha permesso di automatizzare i test, migliorando l'affidabilità e riducendo significativamente il tempo necessario per le verifiche manuali. Ogni modulo è stato testato individualmente per confermare il corretto funzionamento, e sono stati condotti test di integrazione per garantire che tutti i componenti lavorassero insieme senza problemi.

3.2 Metodologia di Lavoro*

3.2.1 Christian Giaconi

- **Movimenti di Pacman**: Si è sviluppato il sistema di controllo per il movimento di Pacman, garantendo che il personaggio si muova all'interno del labirinto in risposta agli input del giocatore. Questo sistema include la gestione delle collisioni con i muri, assicurando che Pacman non possa attraversare ostacoli. La logica implementata verifica continuamente la validità delle direzioni scelte dal giocatore, bloccando i movimenti non permessi e garantendo che Pacman possa solo muoversi in spazi liberi.

- **Supermode**: Si è optato per un sistema basato su un numero prestabilito di mosse che determina la durata della supermode, durante la quale i fantasmi diventano vulnerabili e possono essere mangiati da Pacman. Alla scadenza di questo quantitativo di mosse, i fantasmi ritornano al loro comportamento originario, riprendendo la loro normale strategia di movimento.

- **Gestione della griglia**: La mappa di gioco è stata implementata utilizzando una struttura a griglia, che rappresenta il labirinto come una matrice bidimensionale. Questo sistema gestisce le posizioni di Pacman, dei fantasmi e degli oggetti presenti nel labirinto, e coordina le interazioni tra queste entità e l'ambiente circostante. La griglia è progettata per assicurare che le interazioni siano gestite in tempo reale e che le posizioni delle entità siano aggiornate correttamente.

3.2.2 Giacomo Rossi

- **Strategie dei Fantasmi**: Le strategie di movimento dei fantasmi sono state progettate utilizzando il design “Pattern” per assicurare una gestione flessibile e

modulare del loro comportamento. Ogni fantasma segue una strategia di movimento distinta, che può includere modalità come Chase, dove il fantasma insegue Pacman, Scatter, che comporta il distanziamento temporaneo tra ciascun fantasma, allo scopo di garantire un'esperienza bilanciata ed evitare che tutti i fantasmi si ammucchiano insieme per inseguire Pacman, e Flee, durante la quale il fantasma cerca di fuggire quando Pacman attiva la supermode. Le strategie sono strutturate per essere facilmente modificabili e aggiornabili in tempo reale durante il gioco. Questo approccio consente di adattare dinamicamente il comportamento dei fantasmi in risposta agli eventi di gioco, come l'attivazione della supermode o altre situazioni particolari, garantendo una reattività adeguata alle variazioni nel comportamento di Pacman e mantenendo un'esperienza di gioco coinvolgente e imprevedibile.

- **Componenti della griglia**: I componenti della griglia: `Wall`, `SmallDot` e `BigDot`, sono stati sviluppati mediante il design pattern "Composite", per garantire una struttura gerarchica dei ruoli ricoperti da ciascuno dei tre componenti. Mediante appositi metodi, sono state determinate reazioni specifiche alle interazioni tra pacman ed i tre componenti e quelle tra i fantasmi e i muri.

3.3 Note di Sviluppo*

3.3.1 Christian Giaconi

- **Sviluppo della GUI**: Lavorazione della progettazione e dell'implementazione dell'interfaccia grafica del gioco. Questo include la creazione degli elementi visivi come la rappresentazione del labirinto, Pacman, i fantasmi e i punti, assicurando che tutti gli elementi grafici siano ben allineati e facilmente comprensibili per l'utente. La GUI è stata progettata per essere intuitiva e reattiva, riflettendo correttamente lo stato del gioco in tempo reale.

- **Lambda Expressions**: Sono state adottate per gestire in modo elegante e compatto le operazioni di filtraggio e trasformazione degli input del giocatore, facilitando così l'elaborazione di eventi e migliorando la reattività del sistema.

- **Stream API**: Le Stream API sono state impiegate per ottimizzare il trattamento dei dati della griglia e delle entità, consentendo operazioni fluide e dichiarative. Questo approccio ha facilitato la trasformazione, il filtraggio e l'elaborazione dei dati in modo dichiarativo, contribuendo a un codice più chiaro e più facile da mantenere.

3.3.2 Giacomo Rossi

- **Lambda Expressions**: Le espressioni lambda sono state utilizzate per semplificare il codice relativo alla gestione degli input del giocatore. Questo approccio ha reso il codice più conciso e leggibile, migliorando anche la manutenibilità e riducendo il boilerplate code.

- **Generics**: Utilizzati principalmente come un mezzo per ottenere argomenti per determinate azioni, e.g i metodi della gestione della Griglia di gioco.

- **Stream API**: Sono state impiegate per eseguire operazioni efficienti sui dati della griglia e delle entità. Utilizzando le Stream API, è stato possibile filtrare e manipolare gli elementi della griglia e gestire le liste di oggetti in modo funzionale e pulito, migliorando la leggibilità e l'efficienza del codice.

- **Optional**: Viene utilizzata la classe `Optional` per gestire i risultati delle operazioni che potrebbero restituire valori nulli. Questo approccio ha migliorato la leggibilità e la sicurezza del codice, evitando l'uso di controlli espliciti con “null” e riducendo il rischio di eccezioni non gestite.

- **Timers e Random:** Viene utilizzata la classe `Timer` per garantire ai fantasmi durante la partita, di cambiare la propria strategia di movimento in tempi casuali, determinati dalla classe `Random`, allo scopo di evitare il più possibile il bloccaggio dei fantasmi.
  Inoltre, per `Random` si è trovato come impiego anche la determinazione casuale della direzione di un fantasma.

***Nota sulla Suddivisione dei Compiti**

È importante notare che, sebbene i compiti siano stati inizialmente suddivisi come descritto, la collaborazione tra Christian Giaconi e Giacomo Rossi è stata molto stretta. Si è deciso di lavorare insieme su diverse parti del codice, condividendo le responsabilità e aiutandoci reciprocamente per risolvere problemi complessi e ottimizzare il sistema. In alcuni casi, abbiamo anche lavorato su aree di competenza dell'altro quando necessario, per garantire un'integrazione fluida e una qualità del codice ottimale. Questo approccio collaborativo ha contribuito a migliorare il progetto finale e a garantire una coerenza tra le diverse componenti del sistema.