OOParty

Emanuele Artegiani
Enrico Tagliaferri
Leonardo Beleffi
Lorenzo Guerrini
Roberto Draghetti

25 agosto 2022

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il progetto che il gruppo deve realizzare, consiste in un gioco da tavolo, chiamato OOParty, in cui i giocatori devono raccogliere il maggior numero di stelle per vincere. Ad ogni turno ogni giocatore lancia il dado, avanza e attiva l'effetto della casella su cui capita. Nel gioco sono presenti dei power-up, ovvero oggetti che possono essere utilizzati contro gli altri giocatori per danneggiarli o per ricevere un vantaggio. Se si capita su una casella su cui è presente una stella, se si possiedono abbastanza monete la si può comprare. Alla fine di ogni turno i giocatori devono partecipare ad un minigioco, che assegna dei premi e stabilisce l'ordine del turno successivo.

1.1.1 Requisiti funzionali

- Il gioco darà la possibilità di creare una partita personalizzata in cui è possibile modificare il numero di giocatori e il numero di turni.
- All'inizio della partita ogni giocatore potrà scegliere il proprio nickname da usare durante il gioco e un colore (scelto da un set predefinito) che lo rappresenti.
- Durante ogni turno ogni giocatore dovrà poter utilizzare dei power-up (se ne possiede).
- I power-up previsti sono i seguenti:
  - **medikit**: il giocatore che lo usa recupera tutta la vita;
  - **fucile**: arreca del danno ad un giocatore scelto;

  - **doppio dado**: tira due volte il dado, permettendo al giocatore di muoversi di più caselle in un solo turno;
  - **magnete**: toglie al giocatore selezionato circa un terzo delle monete in suo possesso.

- Durante ogni turno ogni giocatore dovrà tirare un dado per avanzare sulla mappa.
- Ogni turno dovrà terminare con un minigioco scelto in maniera casuale tra i seguenti:
  - **Mastermind**;
  - **Who risks wins**;
  - **Memo**;
  - **You're The Bob-omb**;
  - **Cut From The Team**.
- Il risultato del minigioco a fine turno, determinerà l'ordine del turno successivo.
- Al termine del minigioco i giocatori riceveranno un premio in monete, che sarà maggiore per il vincitore e via via calante fino ad arrivare all'ultimo classificato.

Tabellone di gioco

Nel tabellone di gioco si potranno vedere i tutti giocatori posizionati nelle relative caselle e le varie caselle speciali presenti nella mappa, la classifica attuale e l'ordine di svolgimento del turno.

Mastermind

Mastermind è un piccolo gioco di logica che consiste nell'indovinare un numero di quattro cifre tutte differenti tra loro. Per indovinare tale numero si avranno a disposizione due indizi:

- **Cifre comuni**: numero di cifre presenti sia nel tentativo sia nel numero da indovinare;
- **Cifre in posizione corretta**: numero di cifre le quali sono nella stessa posizione sia nel tentativo sia nel numero da indovinare.

Le cifre in posizione corretta sono un sottoinsieme delle cifre comuni.

Who risks wins

Who risks wins è un gioco di riflessi che consiste nel fermare la caduta di un blocco il più tardi possibile, ma prima che colpisca l'avatar del giocatore. Il blocco avrà una velocità di caduta generata casualmente ad ogni turno per aumentare la difficoltà e impedire che un giocatore si abitui a tale velocità.

Memo

Memo è un gioco di memoria ispirato al famoso "memory", il gioco da tavolo in cui a turni si gira una coppia di carte; se le carte coincidono il giocatore procede a girare un'altra coppia, diversamente la coppia viene nascosta e il giocatore seguente ripete il procedimento. Il gioco termina quando vengono scoperte tutte le carte.

You're The Bob-omb

You're The Bob-omb è un gioco di fortuna che consiste nel rimanere l'ultimo giocatore in vita. Ogni turno ciascun giocatore sceglie una piattaforma su cui sostare. Dopo che tutti i giocatori hanno scelto, una piattaforma a caso verrà fatta saltare, e tutti i giocatori su quella saranno eliminati. Il gioco termina quando rimane solo una piattaforma o rimane solo un giocatore salvo.

Cut From The Team

Cut From The Team è un gioco di fortuna che consiste nel rimanere l'ultimo giocatore in vita. Ogni turno ciascun giocatore taglia una corda, che può essere collegata a una bomba o no. Ovviamente nel caso la corda sia collegata a una bomba il giocatore corrente perde, diversamente passa al turno successivo. La posizione in classifica sarà data dal numero di bombe scampate.

1.1.2 Requisiti non funzionali

- Le caselle speciali del tabellone di gioco saranno rappresentate attraverso delle immagini.
- Il gioco dovrà essere portabile e quindi sarà possibile utilizzarlo su macchine Windows, Mac OS e UNIX/Linux.

1.2 Analisi e modello del dominio

OOParty consiste in un gioco da tavolo in cui i giocatori competono per ottenere più stelle. Una partita di OOParty si compone di un numero arbitrario di giocatori (scelto al momento di creazione della partita e compreso tra 2 e 4) e da un tabellone di gioco. Il tabellone è composto da delle caselle, alcune delle quali hanno degli effetti speciali. Gli effetti possibili sono:

- far guadagnare al giocatore alcune monete;
- far guadagnare al giocatore un power-up;
- arrecare del danno al giocatore;
- dare la possibilità al giocatore di comprare una stella (se possiede abbastanza monete).

Le caselle che fanno guadagnare le monete, una volta attivate, fanno guadagnare al giocatore un numero casuale di monete.

I power-up presenti nel gioco sono:

- medikit;
- fucile;
- doppio dado;
- magnete.

Le caselle che fanno danno al giocatore gli tolgono un numero casuale di punti vita. Se al giocatore viene azzerata la vita, perde la metà delle monete e viene riposizionato nella prima casella senza effetti dopo la casella contenente la stella (con i punti vita riportati al massimo). In questo modo, per raggiungere la casella con la stella, dovrà fare tutto il giro del tabellone, comportando una penalità non da poco.

La durata della partita è stabilita dal numero di turni, valore impostato dall'utente al momento di creazione della partita stessa. Ogni giocatore è identificato da un nickname e da un colore (scelto tra un set predefinito).

Le fasi che compongono un turno di un giocatore sono: attivazione di power-up (se il giocatore ne possiede), lancio del dado, avanzamento sulla mappa e attivazione dell'effetto localizzato sulla casella su cui si capita, se presente.

Quando tutti i giocatori hanno effettuato il proprio turno si avvia il minigioco, a cui tutti dovranno partecipare. Al termine del minigioco, viene

assegnato un premio in monete ai giocatori, premio che sarà direttamente proporzionale alla posizione nella classifica calcolata dal punteggio ottenuto nel minigioco. Nel caso ci siano dei parimeriti, si procede con uno spareggio effettuato con un lancio di dadi. Viene inoltre assegnato alla prima metà dei giocatori in questa classifica un power-up, scelto casualmente. La classifica del minigioco definisce anche l'ordine di svolgimento del turno successivo.

Alla fine della partita viene mostrata la classifica finale e si viene riportati nel menù principale.

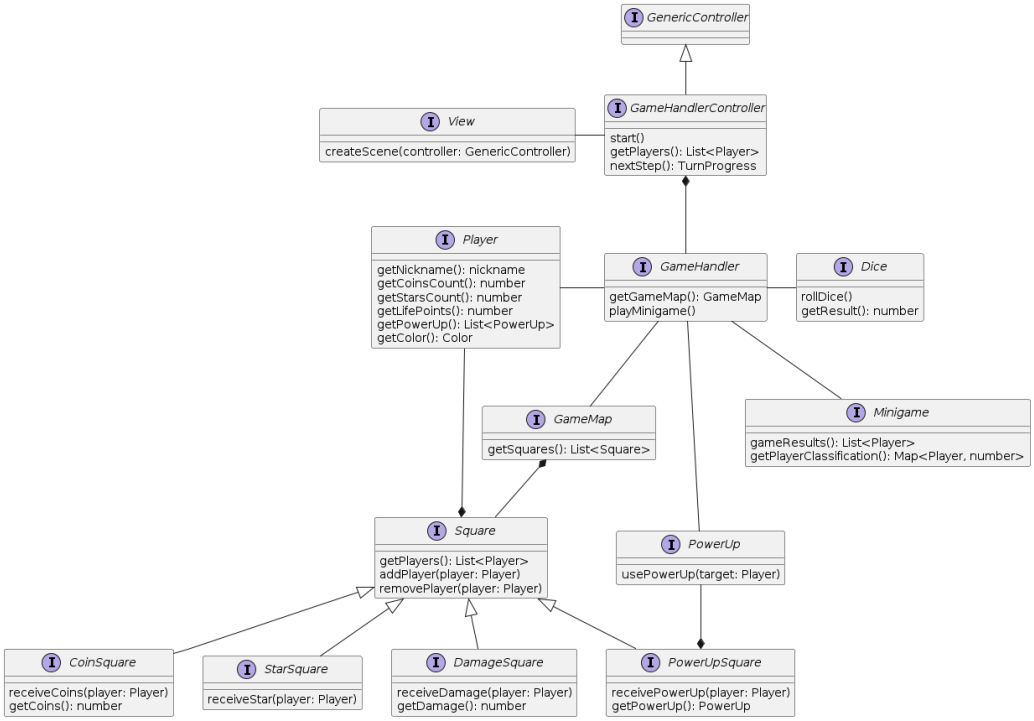

Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

In OOParty abbiamo usato il modello architetturale MVC, così da poter separare la logica dietro all'applicazione, dall'interfaccia con cui questa viene mostrata all'utente.

Abbiamo creato un'interfaccia ControllerFactory che si occupa di creare tutti i controller usati nell'applicazione. Cambiando quindi l'implementazione di ControllerFactory si ha la possibilità di utilizzare un model diverso. Ad ogni controller usato, al momento della creazione, bisogna passare come argomento il model da usare (cosa che viene fatta dai metodi all'interno di ControllerFactory).

Per quanto riguarda la parte di view, ogni volta che bisogna mostrare una schermata di gioco, il controller si occupa di richiamare ViewLoader, oggetto che contiene tutti i metodi per mostrare le schermate necessarie. Il tipo di schermata da visualizzare dipende dall'implementazione di ViewLoader. La componente ViewController è quella che si occupa di gestire le interazioni con la View. Si occupa quindi di catturare gli eventi e di associarci un'azione.

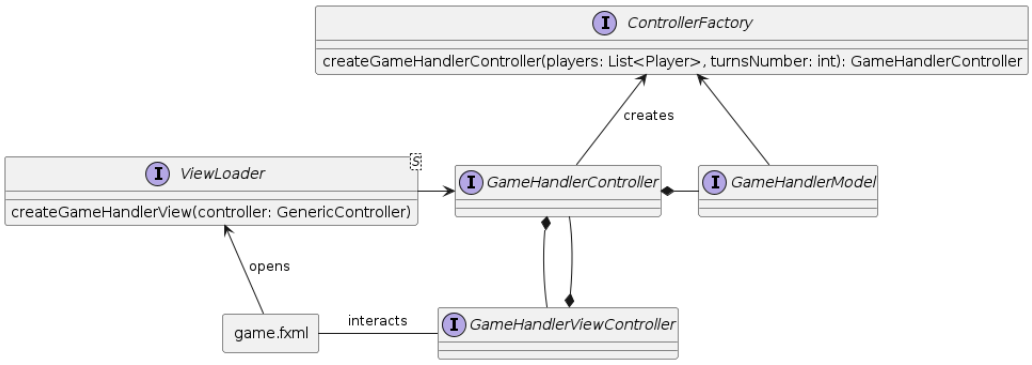

Figura 2.1: Schema UML dell'architettura MVC usata, che mostra l'applicazione di MVC per la classe GameHandler.

2.2 Design dettagliato

2.2.1 Emanuele Artegiani

Gestione delle scene

Problema Vi è la necessità di gestire la gui e le varie scene del gioco, indipendentemente dal tipo di queste ultime. Inoltre, sorge il bisogno di avere una classe che possa fornire le factories necessarie al programma.

Soluzione É stato creato l'oggetto **StageManager** che si occupa di risolvere i problemi sopra citati. Abbiamo utilizzato lo **Strategy pattern** per fare uso del **ControllerFactory** senza conoscerne l'implementazione; inoltre attraverso il **Façade pattern** abbiamo potuto semplificare l'implementazione di **StageManager** delegando la parte di gestione delle scene a **SceneHandler** e la gestione della view a **Gui**. L'utilizzo di quest'ultimo pattern permette di apportare modifiche ed effettuare manutenzione al codice con maggiore semplicità, in quanto da effettuare localmente nelle singole implementazioni delle varie classi.

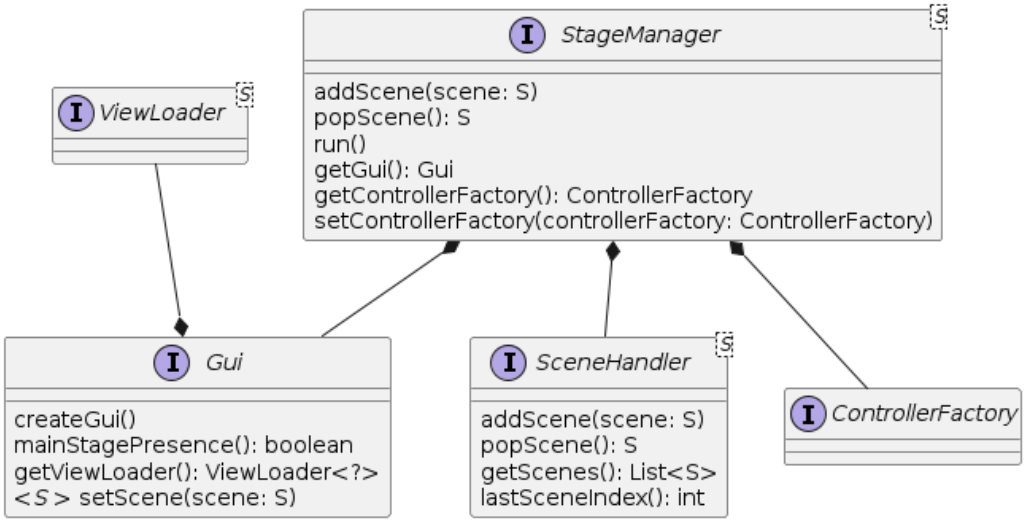

Figura 2.2: Schema UML relativo al gestore delle scene.

Gestione minigiochi

Problema I minigiochi hanno alcuni metodi e campi in comune, mentre altri specifici.

Soluzione Per risolvere questo problema abbiamo usato il **Template method**. Questo pattern è utilizzato per varie classi del progetto. L'esempio più significativo è il model dei minigiochi. **GameModelAbstr** fornisce un'astrazione per la gestione dei giocatori, minimizzando la duplicazione del codice, e lasciando alle sottoclassi il metodo *runGame()* da implementare. Stessa cosa, ma per la gestione dei punteggi, viene fatta da **MinigameModelAbstr** (specializzazione della classe astratta citata in precedenza). I model dei minigiochi estendono **MinigameModelAbstr** e possiedono metodi e campi proprietari relativi al minigioco preso in considerazione.

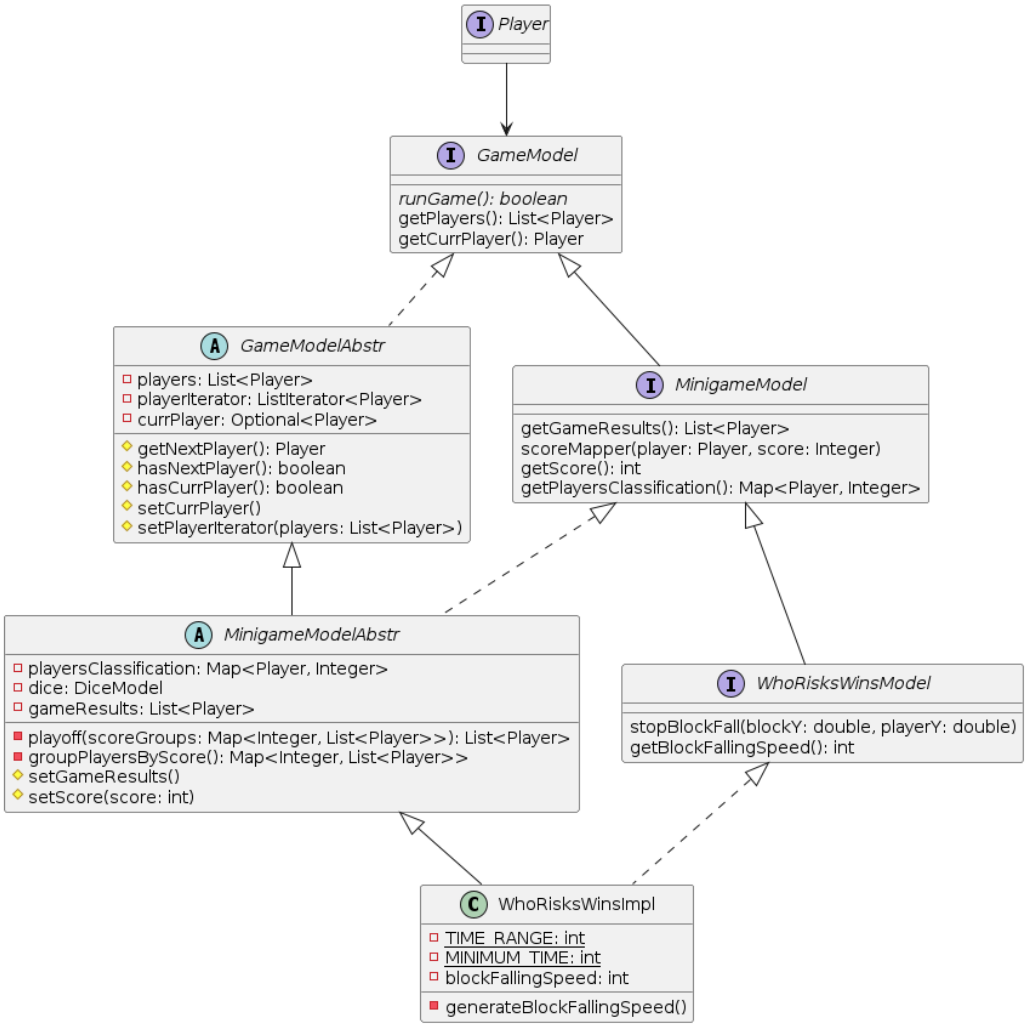

Figura 2.3: Schema UML relativo al model di Who risks wins.

2.2.2 Enrico Tagliaferri

Creazione mappa di gioco

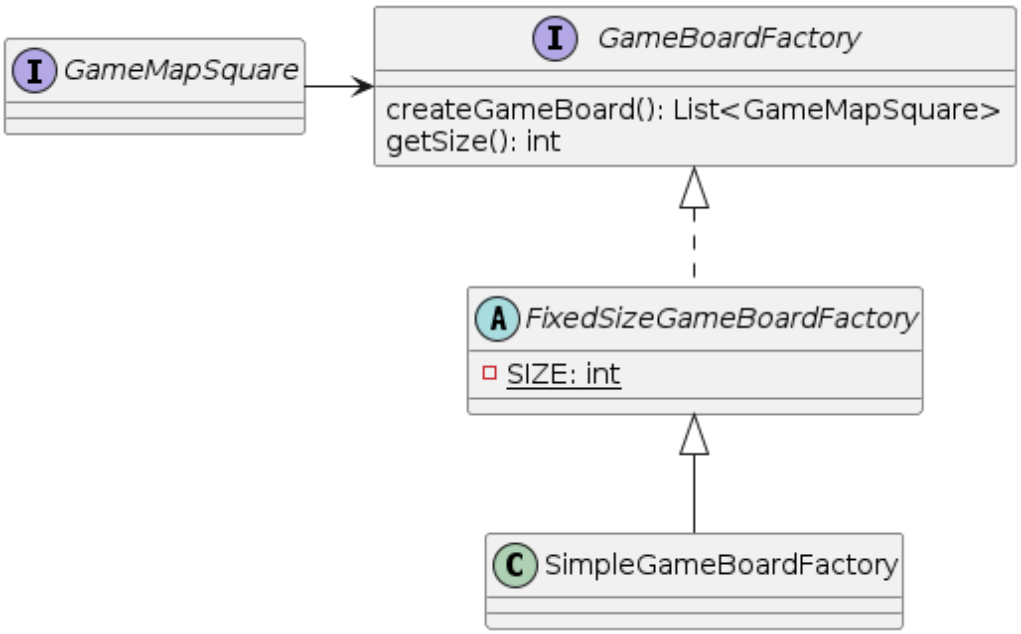

Figura 2.4: Schema UML relativo alla creazione della mappa di gioco.

Problema Per la creazione della mappa di gioco si è incontrato il problema di poter avere diversi tipi di mappa. Un esempio potrebbe essere quello di una mappa con la dimensione fissa e una mappa "personalizzabile", quindi con la dimensione scelta al momento della creazione.

Soluzione La mappa di gioco è rappresentata attraverso una lista di GameMapSquare, quindi per la risoluzione di questo problema abbiamo pensato all'utilizzo del pattern **Factory Method**. Il **Factory Method** è GameBoardFactory. In questo modo abbiamo potuto creare un oggetto che rappresentasse una mappa con una dimensione stabilita a priori e di seguito l'abbiamo implementata. Il compito della classe implementata è quello di decidere, per esempio, in che modo le caselle sono disposte.

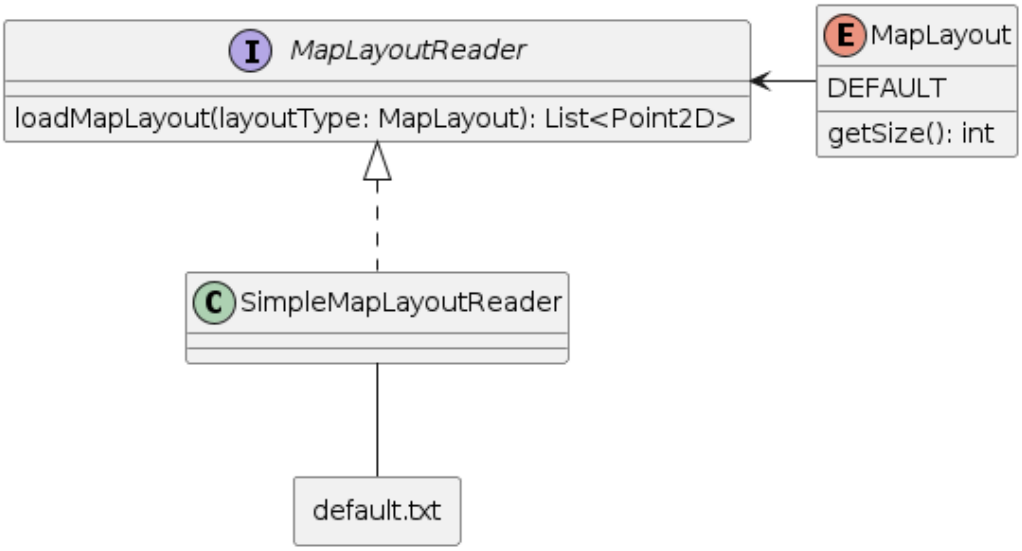

Figura 2.5: Schema UML della gestione del caricamento dei layout. SimpleMapLayoutReader rappresenta un oggetto che carica file con formato txt.

Problema Bisogna gestire il salvataggio dei layout della mappa, così da poter avere mappe con forme diverse.

Soluzione Per la memorizzazione dei layout abbiamo pensato all'utilizzo di file. Un file, il cui nome è uguale a quello del layout che rappresenta, contiene una lista di coppie di valori che rappresentano la posizione di una casella nella griglia usata per mostrare la mappa. Ogni coppia è composta dall'indice della colonna e dall'indice della riga. A lato programma, esiste l'oggetto MapLayoutReader che si occupa del caricamento del layout. L'enumeration MapLayout definisce i vari tipi di layout, con relativa dimensione (ovvero il numero di caselle di cui è composto un layout).

Gestione guide ai minigiochi

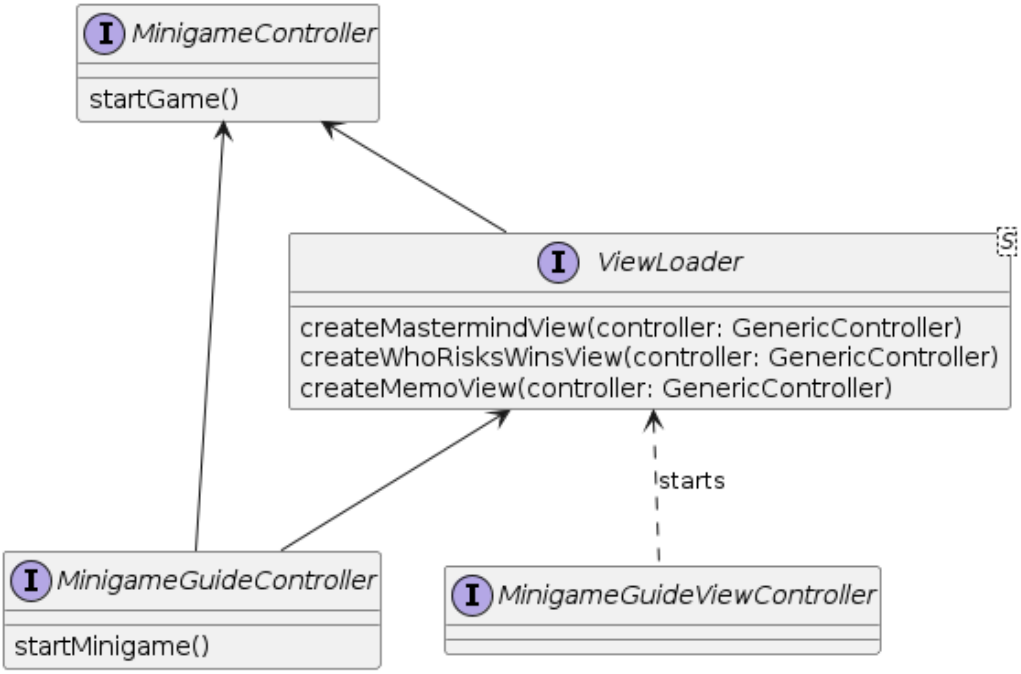

Figura 2.6: Schema UML delle relazioni tra le componenti che si occupano di mostrare la guida di un minigioco.

Problema Ogni minigioco deve mostrare la propria guida quando viene avviato.

Soluzione Quando un minigioco viene avviato, apre la propria guida. Per fare questo il controller del minigioco fa partire la View della sua guida. In quanto tutte le guide sono uguali (cambiano sono le scritte mostrate) viene usato un unico MinigameGuideController che alla pressione di un tasto, si occupa di rimuovere l'ultima scena caricata (quella della guida) facendo così partire il minigioco che l'ha creata.

2.2.3 Leonardo Beleffi

Gestione minigiochi

Problema Gestire i minigiochi tenendo divisi i componenti interni.

Soluzione Applicazione del pattern MVC in modo da separare logiche di gioco da rappresentazioni grafiche dello stesso, e crezione di un controller che permetta l'interazione delle 2 parti. Il controller sarà specifico per il minigioco d'interesse. Per facilitare l'astrazione ci siamo avvalsi di interfacce.

In figura 2.7, per concentrare l'attenzione fra le interazioni tra M V e C, è stato omesso; ma tutte le implementazioni mostrate nell'UML ereditano dalle rispettive classi astratte MinigameModelAbstr, MinigameViewControllerAbstr, e MinigameControllerAbstr.

Per la spiegazione di queste scelte vedere: "Gestione minigiochi" in sez. 2.2.1

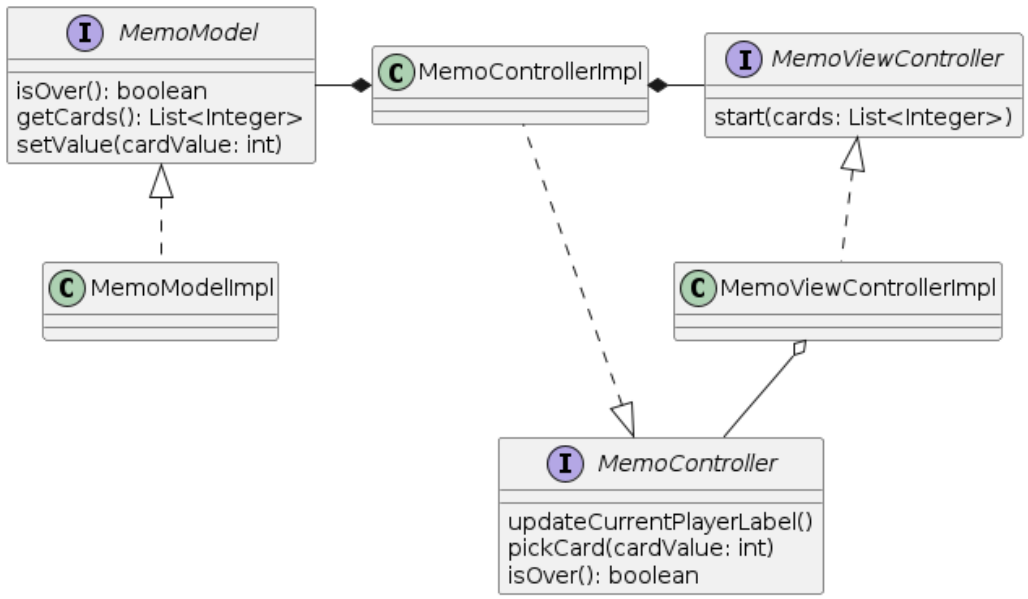

Figura 2.7: Schema UML relativo alla gestione dell'MVC del minigioco Memo.

Integrazione MVC con JavaFX

Problema Riuscire a integrare il pattern **MVC** con la libreria grafica **JavaFX**.

Soluzione Siccome in MVC il controller deve essere in grado di funzionare con tutte le view, non è stato possibile integrare JavaFX direttamente perchè il controller avrebbe dovuto riferirsi esplicitamente a file risorse di JavaFX, quindi sarebbe stato legato a questa libreria specifica. Per risolvere abbiamo creato, oltre al controller del gioco (che è generico) anche un controller per la view (JavaFX-oriented). Questo ha permesso di sollevare il controller di gioco da questi dettagli implementativi, e di fatto non infrangere il pattern succitato.

Uso della reflection in *utils.controller.GenericControllerAbstr*

Problema I metodi **getViewController** e **setViewController** sono metodi astratti che fanno la stessa cosa in tutti i minigiochi e sono implementati allo stesso modo ovunque, solo con un cast alla propria interfaccia.

Soluzione Abbiamo provato a risolvere questo problema nel modo più generico possibile, a partire dalla reflection andando a prendere la classe runtime, solo che nella classe astratta dove sono definiti questi 2 metodi non esiste il campo che questi metodi vanno a leggere/scrivere, quindi dava errore. Ho pensato di creare un setter astratto e richiamarlo da questo metodo, ma poi mi ha dato l'idea di spostare il problema senza risolverlo. Alla fine abbiamo optato per lasciare che tutte le sottoclassi si implementassero questo metodo, non avendo trovato soluzioni migliori.

Testing delle logica di gioco in you're the bob-omb

Problema Riuscire a effettuare testing della classe **YoureTheBobomb-ModelImpl**.

Soluzione Non sono riuscito a trovare una soluzione... Cercando in rete ho trovato che per facilitare il testing è molto usato il **Dependency injection**. Mi sono informato e ho provato a usarlo, come si può vedere nelle classi di testing **TestCutFromTheTeamModel** e **TestMemoModel**, tuttavia mi sono trovato davanti a un muro troppo alto per me al momento: la necessità di generare numeri casuali. Non sono riuscito a capire come passare alla classe di model da testare un "finto" generatore di numeri random, in modo

da poter prevedere l'output dei vari input in anticipo. Siccome la mia classe da testare dipende internamente da un oggetto **java.util.Random**, visto anche che non posso prevedere l'esito di un numero casuale, i test che avrei potuto fare sarebbero stati irrisori. Questo mi ha portato a cancellare la classe **TestYoureTheBobombModelImpl**.

Minigioco memo

Problema A fine turno aspettare qualche secondo e ricoprire le carte.

Soluzione Non avendo a disposizione i thread ho riscontrato vari problemi nell'aspettare del tempo, soprattutto perchè veniva messa in pausa anche la grafica e ne risultava un gioco un po' macchinoso. Ho risolto aggiungendo un bottone che abilita il turno successivo e ho interamente rimosso l'attesa di un tempo predefinito.

2.2.4 Lorenzo Guerrini

Gestione delle caselle

Problema Diversificazione delle caselle.

Soluzione Per questo problema è stata creata un'interfaccia GameMapSquare, che viene implementata dalla classe GameMapSquareImpl, la casella semplice. Ogni casella speciale estende questa classe. L'interfaccia, tra gli altri metodi, dispone di un metodo makeSpecialAction, che viene implementato diversamente per ogni casella e permette alla casella speciale di eseguire la propria azione speciale.

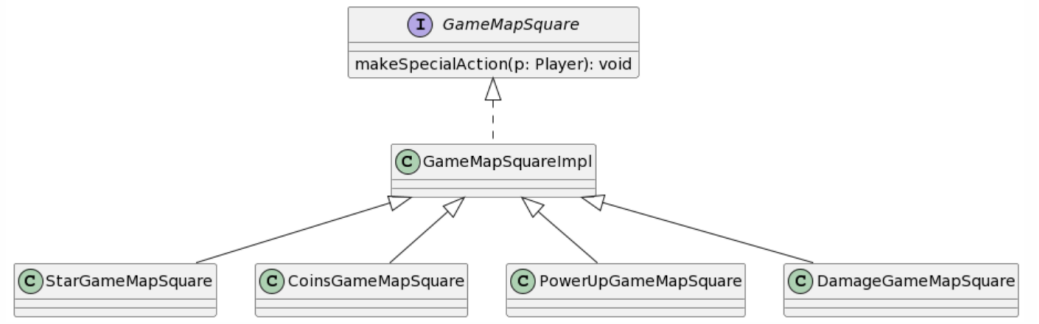

Figura 2.8: Schema UML relativo alla gestione della diversificazione delle caselle.

Gestione dei giocatori sulle caselle

Problema Gestione della posizione dei giocatori nelle varie caselle del tabellone di gioco.

Soluzione Ogni GameMapSquare possiede un Set di Player, che rappresenta i giocatori presenti su quella casella. Ogni casella ha inoltre un metodo per rimuovere e un metodo per aggiungere giocatori su di essa. L'interfaccia GameMap presenta invece un metodo per restituire la casella sulla quale è posizionato un giocatore, passatogli questo come parametro, poiché possiede la lista delle caselle. Il Player si può muovere tramite due metodi, moveForward e goTo. Il metodo initializePlayers si occupa di posizionare tutti i giocatori all'interno della casella di partenza, se questi non sono ancora stati posizionati.

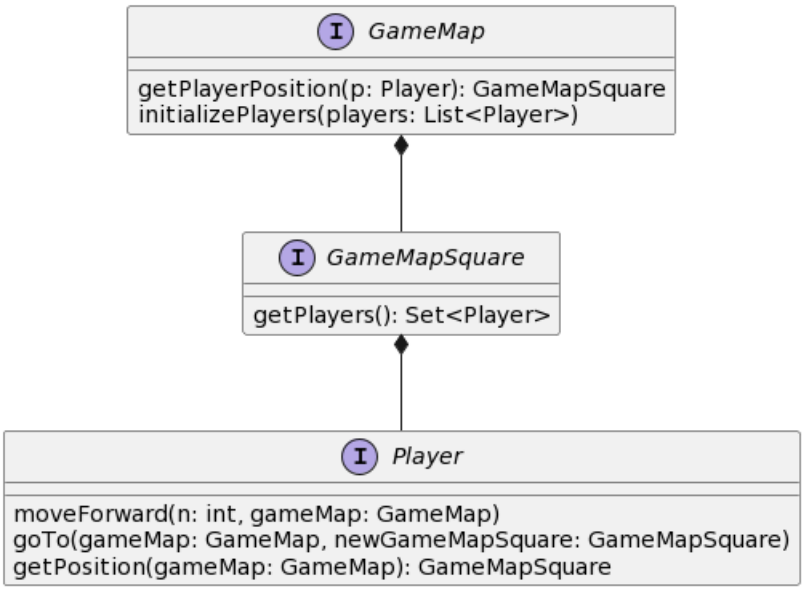

Figura 2.9: Schema UML relativo alla gestione del posizionamento dei giocatori nelle varie caselle.

Schermata della classifica del minigioco e della classifica finale

Problema Dopo ogni mingioco deve comparire la classifica prodotta da esso e alla fine del gioco deve comparire la classifica finale.

Soluzione Ci si è accorti che le due schermate erano molto simili, praticamente uguali. Si è quindi deciso di implementarle tramite le stesse classi

e interfacce. Sono state create le interfacce AfterMinigameMenuController e AfterMinigameMenuViewController. Hanno entrambe due metodi makeLeaderboard e makeEndGameLeaderboard: il GameHandlerController richiamerà una o l'altra a seconda di quello che si vuol mostrare, e l'AfterMinigameMenuController richiamerà l'omonima funzione dell'AfterMinigameMenuViewController.

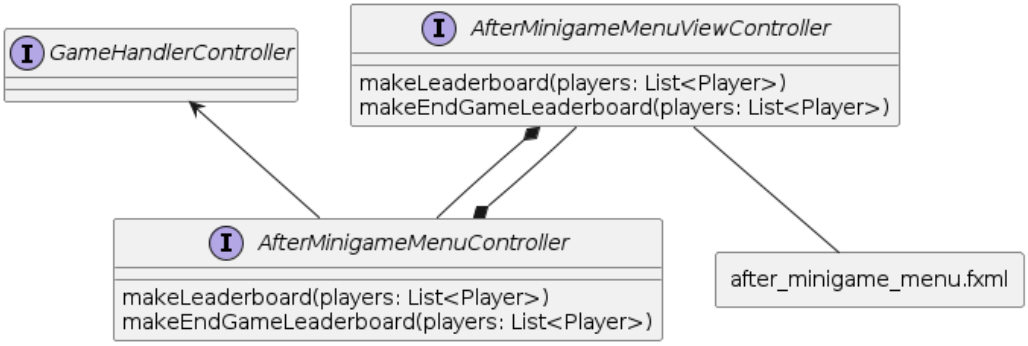

Figura 2.10: Schema UML relativo alla gestione della schermata delle classifiche.

Generazione di un power-up casuale

Problema Generare un power-up casuale tra quelli presenti nel gioco.

Soluzione Per fare questo è stato usato il **Factory Method**. L'interfaccia PowerupFactory dispone di un metodo getRandomPowerup, implementato in PowerupFactoryImpl, che restituisce un power-up casuale.

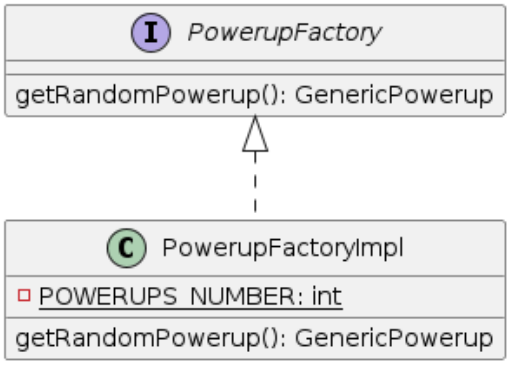

Figura 2.11: Schema UML relativo alla gestione della generazione di un power-up casuale.

2.2.5 Roberto Draghetti

Gestione dei power-up

Problema Diversificazione dei power-up.

Soluzione Per poter avere all'interno del gioco diversi tipi di power-up, è stata creata l'interfaccia GenericPowerup. Essa può essere implementata in diversi modi per avere un funzionamento differente a seconda del tipo di effetto che si vuole dare al power-up. In particolare, l'interfaccia contiene il metodo usePowerup, che applica un diverso effetto al bersaglio (passato come parametro) a seconda dell'implementazione, oltre ai metodi getPowerupType e useOnSelf, che permettono di capire dall'esterno di quale power-up si tratta e se viene utilizzato su sè stessi o sugli altri giocatori.

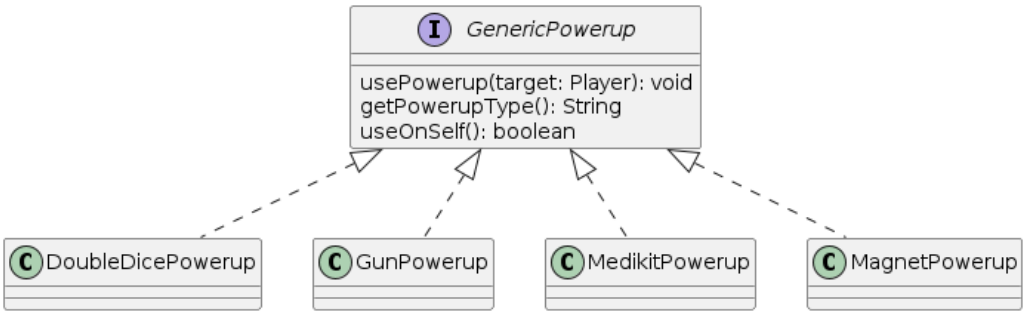

Figura 2.12: Schema UML relativo all'interfaccia dei power-up e alle sue diverse implementazioni.

Gestione del lancio dei dadi.

Problema Ottenimento del risultato del lancio di uno o più dadi e animazione visiva del lancio degli stessi.

Soluzione Il dado è gestito da tre diverse componenti: model, view e controller. Il model si occupa della parte logica del funzionamento del dado, e attraverso la funzione rollDice crea progressivamente una lista che associa i vari risultati ottenuti al giocatore che ha lanciato il dado stesso, la quale si può ottenere attraverso il metodo getResults. L'interfaccia possiede anche altri metodi per poter ottenere il risultato dell'ultimo lancio e il totale dei vari risultati. Il controller si occupa di gestire le interazioni tra model e view, pertanto ha il compito, una volta richiamata la funzione start, di ottenere i risultati dei lanci dal controller e di richiamare la funzione initialize nella view

per inizializzare la scena del lancio del dado. Attraverso l'input dell'utente, la view richiama la funzione nextStep del controller, la quale fa procedere l'animazione grafica richiamando il metodo jumpToDice della view.

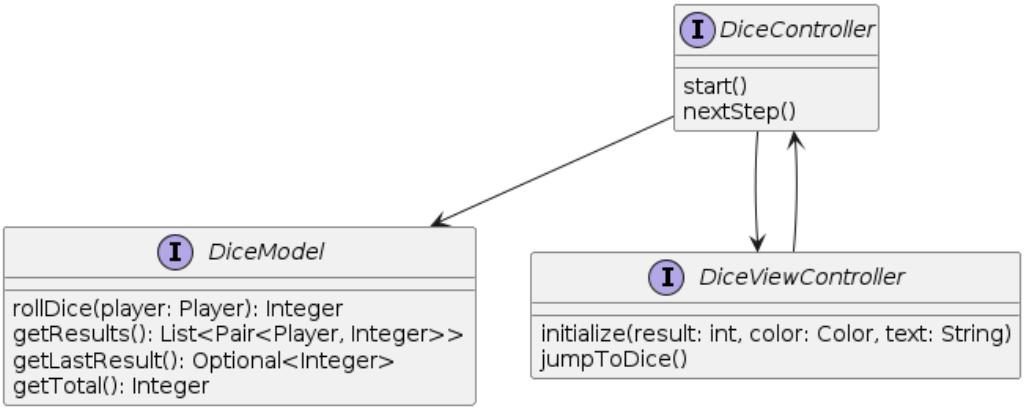

Figura 2.13: Schema UML relativo alle interfacce che gestiscono il funzionamento del dado a livello sia logico che grafico.

Gestione dello spareggio dopo i minigiochi.

Problema Quando il lancio dei dadi serve per lo spareggio di un minigioco, i risultati di ogni giocatore devono essere tutti diversi tra loro.

Soluzione Siccome il dado utilizzato per lo spareggio differisce da quello normale solo per i risultati, che devono essere tutti diversi tra loro, il problema è stato risolto creando la classe DiceModelNoRepeatImpl, la quale estende DiceModelImpl, la prima implementazione dell'interfaccia DiceModel. La differenza tra le due classi in questione riguarda solamente il metodo rollDice, che viene modificato in DiceModelNoRepeatImpl per controllare che il risultato sia diverso da tutti quelli ottenuti in precedenza, e lanciare un'eccezione nel caso non ci siano più risultati diversi disponibili. In DiceModelImpl sono stati creati i metodi protetti setResult e getRandom, che permettono alla nuova classe di interagire con i risultati e il Random presenti all'interno.

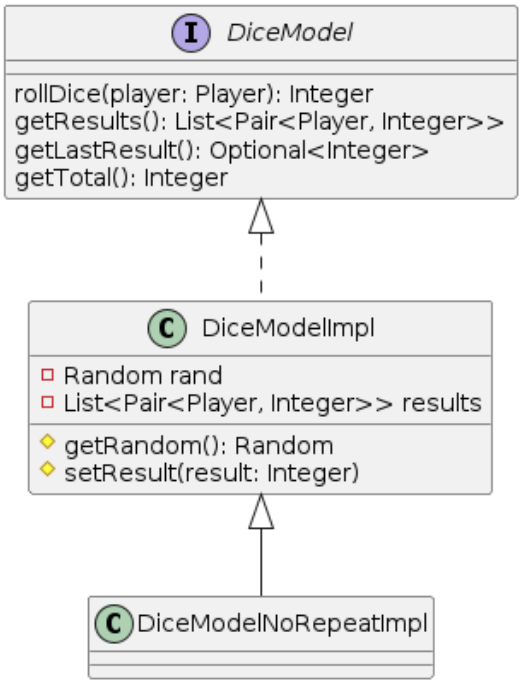

Figura 2.14: Schema UML relativo alla gestione dei due diversi tipi di dado.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Per questo progetto sono state fatte diverse classi di test utilizzando la libreria JUnit. In particolare, sono presenti:

- Due test per verificare la correttezza della classifica;
- Vari test per verificare il corretto funzionamento della mappa di gioco e delle varie caselle, insieme al movimento dei vari giocatori;
- Una classe di test per verificare la correttezza delle azioni dopo la morte di un giocatore e che questa avvenga realmente nel momento giusto;
- Tre test per verificare il corretto avanzamento dei turni;
- Una classe di test per verificare il corretto funzionamento del gestore delle scene;
- Una classe di test per verificare il corretto funzionamento della parte di model comune ad ogni minigame;
- Una classe di test per verificare la correttezza del minigioco Mastermind;
- Una classe di test per verificare la correttezza del minigioco Memo;
- Una classe di test per verificare la correttezza del minigioco Cut From The Team.

3.2 Metodologia di lavoro

3.2.1 Emanuele Artegiani

- Implementazione del gestore delle scene (**utils.graphics.***);
- Realizzazione di una classe astratta estendibile che fornisce funzioni base per la gestione dei giocatori (**game.common.model.***);
- Realizzazione di interfacce e classi generali (**utils.controller.***, **utils.view.GenericViewController**, **minigames.common.***, **menu.common.***);
- Gestione del menu iniziale (**menu.mainmenu.***), del menu di creazione della partita (**menu.gamecreationmenu.***) e del menu di pausa (**menu.pausemenu.***);
- Realizzazione dei minigames: Mastermind (**minigames.mastermind.***) e Who risks wins (**minigames.whoriskswins.***);
- Realizzazione di enumerazioni utili allo sviluppo della view del gioco (**Notice**, **PlayerColor**);
- Realizzazione di alcune classi di utility (**NoticeUser**, **GuiUtils**, **IntSpinnerValueFactory**);
- **TestMinigame**, **TestMastermind** e **TestStageManager**.

Inoltre ho contribuito allo sviluppo di:

- **ControllerFactory** e **ControllerFactoryFx**;
- **ViewLoader** e **ViewLoaderFx**.

3.2.2 Enrico Tagliaferri

- Creazione della mappa di gioco (**utils.factories.board.***);
- Implementazione di un oggetto in grado di caricare il layout della mappa di gioco (**utils.readers.***);
- Realizzazione di enumeration utili allo sviluppo del gioco (**MapLayout**, **OrdinalNumber**, **SquareType)**;

- Gestione delle guide introduttive ai minigiochi (**MinigameGuideController**, **MinigameGuideControllerImpl**, **MinigameGuideViewController** e **MinigameGuideViewControllerImpl**);
- **TestLeaderboard**, **TestGameBoard** e **TurnProgressTest**.

Inoltre ho contribuito allo sviluppo di:

- **GameHandlerViewControllerImpl**;
- enumeration **PlayerTurnProgress**, **TurnProgress**;
- **ControllerFactory** e **ControllerFactoryFx**.

3.2.3 Leonardo Beleffi

- Creazione di tutte le classi necessarie al funzionamento del minigame **minigames.memo.***;
- Creazione di tutte le classi necessarie al funzionamento del minigame **minigames.yourethebobomb.***;
- Creazione di tutte le classi necessarie al funzionamento del minigame **minigames.cutfromtheteam.***;
- Testing automatizzato per le logiche di gioco.

Inoltre ho contribuito allo sviluppo di:

- **ControllerFactory**, **ControllerFactoryFX**;
- **ViewLoader**, **ViewLoaderFx**;
- **MinigameControllerFactoryImpl**;
- **MinigameModelAbstr**;
- Refactoring vari per migliorare **MVC**.

3.2.4 Lorenzo Guerrini

- Tutte le caselle (**GameMapSquare**, **GameMapSquareImpl** e tutte le classi che estendono quest'ultima);
- Schermata che mostra la classifica al termine di ogni minigioco e al termine della partita (**menu.afterminigamemenu.***);
- Generazione di un power-up casuale (**PowerupFactory** e **PowerupFactoryImpl**);
- Eccezione per un giocatore non trovato nel tabellone di gioco (**PlayerNotFoundException**);
- **TestGameMapSquare**, **TestMap** e **TestPlayerDeath**.

Inoltre ho contribuito allo sviluppo di:

- **game.gamehandler.***;
- Svariati metodi dell'interfaccia **Player** e della classe **PlayerImpl**;
- Alcuni metodi di **GameMap** e **GameMapImpl**.

3.2.5 Roberto Draghetti

- Creazione di tutti i power-up (**game.powerup.***);
- Creazione e gestione del menu di utilizzo dei power-up (**menu.powerupmenu.***);
- Gestione del progresso dei turni dei giocatori (**game.gamehandler.***)
- Creazione di enumeration utili a tracciare il progresso dei turni (**TurnProgress**) e dei turni dei giocatori (**PlayerTurnProgress**)
- Creazione e gestione del lancio dei dadi (**game.dice.***)

Inoltre ho contribuito allo sviluppo di:

- Creazione iniziale dell'interfaccia **Player** e della classe **PlayerImpl**;
- **MinigameControllerFactory** e **MinigameControllerFactoryImpl**.

3.3 Note di sviluppo

Per aiutarci nello sviluppo abbiamo utilizzato lo strumento di version control **Git**. I due branch prinicipali sono stati *master* e *develop*. Su *master* abbiamo tenuto un programma che funzionava in maniera completa; su *develop* invece aggiungevamo le funzionalità man mano che erano complete. Da *develop* poi ognuno faceva partire altri branch per le funzionalità personali. Per ogni funzionalità, poi, ognuno poteva far partire altri sotto branch, se lo riteneva necessario. In questo modo *develop* è stato un branch con un programma sempre funzionante.

3.3.1 Emanuele Artegiani

- Uso di **Gradle wrapper** fornito a lezione;
- Uso di **Optional**;
- Uso di **Stream**;
- Uso di **Lambda**;
- Uso della **Reflection** in **StageManagerImpl** per istanziare la gui;
- Uso di **Wildcard**;
- Uso di **Generici**;
- Uso di **JavaFX**.

Per utilizzare Javafx tramite Gradle-plugin mi sono affidato a `https://github.com/openjfx/javafx-gradle-plugin`.

3.3.2 Enrico Tagliaferri

- Uso di **Optional**;
- Uso di **Stream**;
- Uso di **Lambda**;
- Uso della **Reflection** in **SimpleGameBoardFactory** per effettuare dei controlli sul tipo delle caselle;
- Uso di **Wildcard**;

- Uso di **JavaFx**.

Per poter leggere un oggetto di tipo FileInputStream ho seguito il seguente blog: `https://zetcode.com/java/inputstream/`

3.3.3 Leonardo Beleffi

- Uso di **Optional**;
- Uso di **Stream**;
- Uso di **Lambda**;
- Uso di **Method Reference**;
- Uso di **Wildcard** e **Reflection** (anche se in seguito a refactoring abbiamo trovato modi migliori di scrivere il codice);
- Uso di **Generici**;
- Uso di **JavaFX**.

Per poter usare Dependency injection ho guardato qui: `https://it.wikipedia.org/wiki/Dependency_injection` e qui: `https://www.youtube.com/watch?v=eQ90v7HQT-Q&t=1015s&ab_channel=Amigoscode`

3.3.4 Lorenzo Guerrini

- Uso di **Optional**;
- Uso di **Stream**;
- Uso di **Lambda**;
- Uso di **JavaFX**.

Per inserire le immagini nelle caselle del tabellone di gioco mi sono aiutato guardando qui:
`https://www.tutorialspoint.com/how-to-add-an-image-as-label-using-javafx`.

3.3.5 Roberto Draghetti

- Uso di **Optional**;
- Uso di **Stream**;
- Uso di **Lambda**;
- Uso di **JavaFX**

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

4.1.1 Emanuele Artegiani

Questo è stato il primo progetto di discrete dimensioni a cui ho preso parte. Durante lo svolgimento di quest'ultimo ho migliorato le mie capacità di team-working, riuscendo a confrontarmi sempre più efficacemente con i miei compagni, e soprattutto ho avuto la possibilità di applicare le nozioni di git imparate lezione. Inoltre, ho notato che con il procedere del progetto ho migliorato progressivamente il modo nel quale utilizzo git, comunque ancora lontano dall'essere perfetto, e mi sono reso conto in prima persona delle incredibili potenzialità di questo strumento. Sono rimasto positivamente soddisfatto del lavoro di progettazione e implementazione svolto, in quanto effettuare modifiche, anche di diversa entità, è risultato abbastanza facile e veloce. Portare a termine questo progetto è stato tutt'altro che semplice, ma sento che l'impegno che ci ho messo mi abbia ripagato rendendomi più consapevole di come funzioni lo sviluppo software e migliorando il mio stile di programmazione, soprattutto nell'ambito della programmazione a oggetti.

4.1.2 Enrico Tagliaferri

Questo è stato il primo progetto di medie-grandi dimensioni in cui mi sono cimentato. Mi ha aiutato a conoscere meglio lo strumento Git, che avevo già usato in precedenza, ma per progetti decisamente più semplici. Inoltre, le altre volte in cui ho usato Git è stato in progetti a cui lavoravo da solo. La differenza nell'usarlo in un gruppo è notevole e sono rimasto piacevolmente sorpreso dalla sua utilità (che prima non avevo compreso appieno). Sono inoltre consapevole di non averlo sfruttato al suo massimo potenziale. Svi-

luppare questo progetto mi ha anche aiutato meglio a comprendere le mie abilità in quanto programmatore. Sono rimasto stupito dall'essere stato in grado di utilizzare in maniera corretta la tecnica della reflection, in quanto a lezione non avessi compreso fino in fondo la sua utilità. Ritengo che il mio ruolo nel progetto non sia stato completamente centrale, in quanto ho sviluppato aspetti che non appartenevano al cuore dell'applicazione, ma quando ho contribuito su aspetti comuni, penso di aver lavorato bene con i miei compagni di progetto.

4.1.3 Leonardo Beleffi

Questo progetto è stata la sfida più impegnativa della mia vita (dal punto di vista informatico). Mi ha fatto disperare abbastanza, soprattutto l'interazione con la grafica che io ho sempre detestato. È tuttavia stato veramente utile, in quanto mi ha aiutato a sviluppare la consapevolezza di essere in grado di portare avandi progetti così impegnativi. Dubito avrei mai provato se non fosse stato per questo corso.

Purtroppo quest'estate non sono riuscito a organizzarmi bene con tutti gli esami e gli impegni, difatti mi sono messo a lavorare a questo progetto decisamente tardi: ciò ha comportato il dover passare un enorme quantitativo di ore giornaliere a lavorare. La lucidità ne ha risentito e immagino che quando riguarderò questo progetto tra qualche settimana mi renderò conto di pattern che avrei potuto usare e ridondanze che mi sarei potuto risparmiare. A questo punto oltre che a prendere atto e ripromettermi di fare diversamente la prossima volta non posso fare molto.

Mi era già capitato di usare git, ma doverlo usare così intensivamente mi ha fatto raggiungere un altro livello. Tutt'altro vale per gradle: continuo a non avere la minima idea di come funzioni, avendo imparato solo i comandi basilari necessari per portare avanti il programma sviluppato.

Mi rendo conto di non essere riuscito a padroneggiare tutti gli aspetti di java come mi sarebbe piaciuto, ma sicuramente ha aiutato a migliorarmi. Mi ha anche fatto realizzare che ero caduto nell'effetto Dunning-Kruger senza rendermene conto, e mi ha messo molta voglia di imparare molte altre techiche (e pattern) avanzate per rendere i miei codici più semplici, funzionali e leggibili.

4.1.4 Lorenzo Guerrini

Questo è stato il progetto più grande che ho realizzato finora, e affrontarlo è stato sicuramente molto impegnativo, anche se, essendo programmazione ad oggetti, mi è anche piaciuto cimentarmici, siccome mi è sempre piaciuto

(da quando l'ho visto per la prima volta) OOP e anche Java. In generale mi è quindi piaciuto realizzarlo, nonostante varie difficoltà incontrate. Mi ha aiutato a migliorarmi nella programmazione ad oggetti ma anche a capire meglio Git e ad utilizzarlo in maniera corretta in modo che potesse essere molto d'aiuto, più di quanto pensassi. Nel complesso penso che il gruppo abbia lavorato abbastanza bene nella coordinazione e nella programmazione in generale, riuscendo a collaborare bene e a confrontarsi efficacemente, nonostante qualche difficoltà iniziale. In particolare penso di aver migliorato non poco la mia abilità nel collaborare con altre persone in progetti, man mano che ci confrontavamo e che il progetto assumeva dimensioni sempre maggiori, riuscendo a spiegare bene come usare le funzionalità che avevo sviluppato e i problemi che riscontravo testando il programma.

4.1.5 Roberto Draghetti

Affrontare questo progetto è stata per me un'esperienza sicuramente impegnativa, che ha finito per occupare, per vari motivi, molte più ore del previsto. È stato molto interessante mettersi alla prova con un progetto di queste dimensioni, sicuramente uno dei più importanti con cui io abbia mai avuto a che fare, ma allo stesso tempo sono stati molti i momenti di difficoltà che ho affrontato, tra cui più volte problemi con l'utilizzo di Git e di Eclipse, anche se questi non sono per me un novità.

Il progetto mi ha fatto comprendere che ci sono molte difficoltà nello sviluppare un lavoro di questo tipo in gruppo, principalmente nel coordinare il lavoro con gli altri membri: questo vale sia per lo scegliere insieme una direzione e una metodologia da dare al progetto, sia per il collaborare a parti di programmazione comuni o che comunque serviranno agli altri.

Tutto sommato, penso che il lavoro finale svolto dal gruppo sia abbastanza buono, mentre per quanto riguarda le mie parti di programmazione, riconosco di non essere sempre riuscito a programmare nella maniera più chiara o più comprensibile possibile, e probabilmente di non avere utilizzato appieno alcune funzionalità di Java, complicando in alcuni casi il mio lavoro più del dovuto.

4.2 Difficoltà incontrate e commenti per i docenti

4.2.1 Emanuele Artegiani

Le difficoltà sono state di diverso tipo, una tra tutte il lavoro di gruppo. Un progetto di gruppo del genere è stato il primo per tutti, ma ciò che ha complicato di più le cose è il fatto che ogni membro ha avuto i propri impegni personali, oltre allo sviluppo di questo progetto; cosa che ci ha costretti ad iniziare a lavorare in periodi differenti e senza poter fare una progettazione ben strutturata, complicando così "l'assemblaggio" del codice di ognuno.

Un'altra difficoltà, riscontrata praticamente subito, è stato l'utilizzo di JavaFX, essendo una libreria vista solo molto superficialmente a lezione. Anche l'implementazione del modello architetturale mvc non è stata tutta rosa e fiori, ma neanche qualcosa di particolarmente problematico, per quanto mi riguarda.

In merito al parere sul corso, mi trovo pienamente d'accordo con il mio compagno Enrico Tagliaferri.

4.2.2 Enrico Tagliaferri

Una delle maggiori difficoltà riscontrate è stata l'organizzazione iniziale del progetto. Nessuno aveva mai lavorato ad un progetto di queste dimensioni e abbiamo fatto fatica ad iniziare, non sapendo come andasse organizzata la struttura dei file o come andasse strutturato il progetto, utilizzando il modello architetturale mvc.

Un parere sul corso, invece, è che la coppia esame in laboratorio più progetto di gruppo rende il corso molto impegnativo. Quello che rende il corso impegnativo non è però il fatto di avere sia una prova in laboratorio, sia un progetto da sviluppare, ma il fatto che entrambe le prove siano molto toste. In particolare il progetto mi è sembrato eccessivamente complicato. Magari questo è dovuto da un mio approccio sbagliato al problema. Oppure viene dal fatto che per altri corsi non è mai stato necessario fare un progetto così consistente, e quindi magari io mi sono abituato ad esami con un coefficiente di difficoltà minore (questo non per sminuire i corsi affrontati fino ad ora). Una delle difficoltà più grosse è stata sicuramente quella di lavorare in gruppo (difficoltà che prima o poi, sono consapevole del fatto che andasse affrontata). Per quanto riguarda la prova in laboratorio l'unica cosa che non condivido è il fatto che si possa rifiutare il voto solo un certo numero di volte. La motivazione più immediata a questa scelta, ho pensato essere per evitare che gli studenti si presentino all'esame poco preparati, e che tentino e ritentino fino

a quando non lo passano. Se questa può essere una motivazione, la comprendo pienamente, ma, da studente, non la condivido. La mia contraddizione viene dal fatto che la consapevolezza di avere un numero limitato di volte per poter superare l'esame, aumenta notevolmente la pressione durante la prova stessa. E visto che in un esame del genere, la lucidità è la chiave per ottenere un buon risultato, questa consapevolezza aumenta la difficoltà dell'esame.

4.2.3 Leonardo Beleffi

La parte più difficile è stata il mettersi d'accordo sulla direzione da prendere, perché essendo il primo progetto serio per tutti, nessuno aveva idea di come muoversi. Inoltre essendo in 5, il punto di forza è la forza lavoro, ma il punto debole è l'incastro di tutti gli impegni. È stato un problema, e visto che non riuscivamo a conciliare le cose siamo partiti scaglionati. Il danno è evidente, ci è voluto parecchio per capire come gli altri avevano strutturato le loro parti, e il riuscire a farle interagire bene non è stato né veloce né semplice.

Parlando del corso, invece, secondo me non è stata ottimale la gestione degli esercizi di laboratorio, le soluzioni sono estremamente complicate anche nei primi laboratori, e un po' mettono paura (visto che nei primi laboratori non si sa ancora programmare).

Tutto sommato sono contento di questo corso perché mi ha aperto gli occhi sulla programmazione a oggetti e mi ha ispirato ad approfondirla anche da solo.

4.2.4 Lorenzo Guerrini

La prima difficoltà incontrata è stata l'organizzazione iniziale: suddividere il lavoro in parti uguali tra tutti i membri del gruppo. Anche poi iniziare non è stato proprio semplice, in particolare perché non sapevo da che parte farmi e perché la mia conoscenza di Git era molto limitata, e usarlo non era per niente facile. Un'altra difficoltà riscontrata è stata leggere e comprendere il codice già scritto, soprattutto per quanto riguarda la parte grafica, in quanto non avevo mai utilizzato prima JavaFX e ho speso quindi molto tempo nel comprendere a pieno il funzionamento generale delle classi e delle interfacce realizzate dagli altri e di come interagivano tra di loro. Per la parte di relazione, la difficoltà principale è stata quella di comprendere a pieno la teoria dei diagrammi UML e realizzarli in maniera coerente al codice scritto.

4.2.5 Roberto Draghetti

Durante lo sviluppo del progetto, mi sono reso conto che uno dei problemi più grandi è stato riuscire a collaborare con gli altri membri del gruppo: sia perché nessuno di noi aveva mai avuto a che fare con dei lavori di questo calibro, sia perché tutti abbiamo avuto vari impegni, universitari e non, nei mesi precedenti alla consegna del progetto stesso. Mi sono infatti reso conto che la scelta della deadline di agosto non è stata minimamente azzeccata e che sarebbe stato molto più opportuno sviluppare il progetto prima della sessione estiva (periodo per me personalmente sempre incastrato tra studio e lavoro).

Iniziare a lavorare al progetto non simultaneamente ci ha inoltre portati ad affrontare la programmazione con le idee non molto chiare sulla direzione da prendere e sullo stile da seguire, con la conseguenza di dover poi modificare più volte parti di codice che erano già state scritte, perdendo così un sacco di tempo che poteva invece essere impiegato per implementare funzionalità aggiuntive e/o migliorare porzioni di codice rendendole più comprensibili e conformi allo stile di programmazione imparato durante il corso.

Personalmente, mi sono trovato male con il modello architetturale mvc, e non credo di essere sempre riuscito a suddividere in modo ottimale le parti di codice appartenenti a questi diversi aspetti del programma. Ho inoltre riscontrato alcune difficoltà con l'utilizzo di Git, dovendo più volte ri-clonare il progetto per risolvere problemi tutt'ora non compresi, e con l'ambiente di sviluppo Eclipse, anche in questo caso dovendo varie volte re-importare il progetto. È stato inizialmente complicato capire come creare la parte grafica del programma, ma una volta iniziato, questa si è rivelata essere una parte relativamente semplice dello sviluppo del progetto.

Per quanto riguarda il corso, nel complesso trovo che esso sia gestito abbastanza bene, tuttavia credo che il progetto stesso risulti essere una sfida fin troppo impegnativa, specialmente per quanto riguarda il numero di ore richieste per realizzare un lavoro di buon livello. Trovandomi io a dover studiare per gli altri esami, e a lavorare allo stesso tempo, ho fatto molta fatica ad avere il tempo necessario a sviluppare il progetto nel migliore dei modi. Una cosa che non condivido riguardo invece all'esame pratico è il limite di due appelli a studente; trovo che sia una limitazione abbastanza insensata e che può in certi casi compromettere la valutazione finale di studenti che puntano ad un buon voto, e che magari durante un'esame hanno semplicemente avuto sfortuna o non si sono trovati nelle condizioni ideali per affrontare l'esame al pieno delle proprie capacità.

Appendice A

Guida utente

All'apertura del gioco ci si ritrova nel menù principale, contenente due bottoni: uno per creare una partita e uno per uscire dal gioco. Premendo quello per creare una partita si viene portati, appunto, nel menù di creazione della partita. Impostati tutti i parametri, si può iniziare una partita premendo il bottone Start Game. Si noti che tutti i nickname e i colori di ogni giocatore devono essere diversi tra di loro.

A questo punto inizia il gioco vero e proprio: si apre una schermata con il tabellone di gioco e i giocatori posizionati sopra di esso. Per avanzare nel turno, basterà premere uno dei seguenti tasti: invio, barra spaziatrice oppure un qualsiasi tasto del mouse.
Lo scopo del gioco è quello di possedere più stelle possibili alla fine dei turni impostati precedentemente. A parità di stelle, si conteranno le monete (chi ne ha di più è in vantaggio) e a parità di monete si prenderanno in considerazione i punti vita (anche qui, chi ne ha di più è in vantaggio).

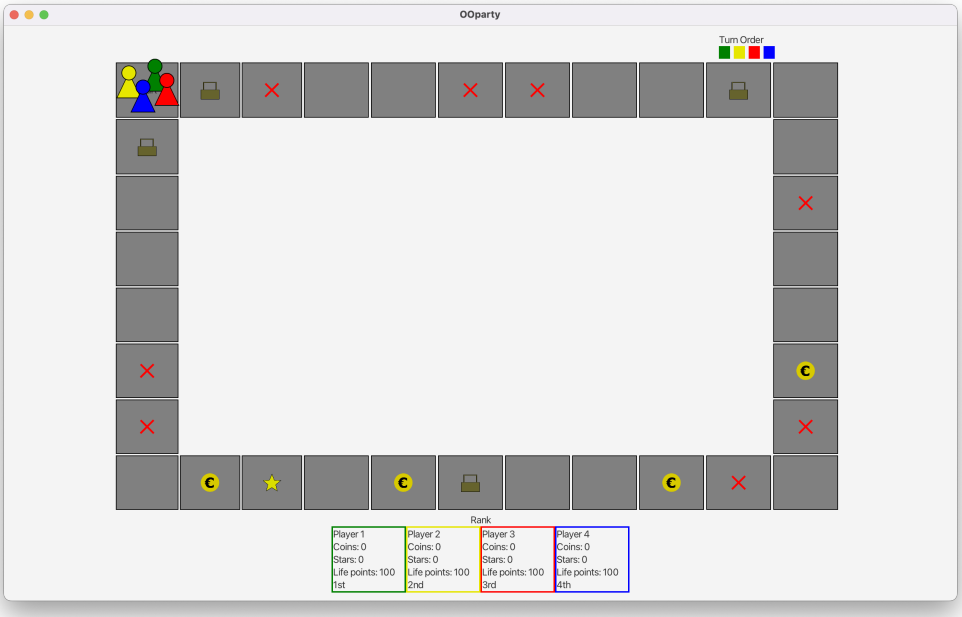

Figura A.1: Un frame del tabellone di gioco.

Il turno di ogni giocatore è composto da varie fasi:

- uso di power-up: se il giocatore possiede uno o più power-up, si apre un menù in cui l'utente può scegliere se usare un power-up e il giocatore sul quale utilizzarlo;
- lancio del dado: il giocatore lancia il dado;
- movimento: il giocatore si muove sulla mappa in base al numero fatto dal dado (o dai dadi lanciati se ha usato il power-up Doppio Dado).

Ogni giocatore, muovendosi nel tabellone di gioco, può capitare su diverse caselle:

- la casella semplice, rappresentata semplicemente da un quadrato vuoto: capitando su questa casella non succede nulla;
- la casella contenente la stella, rappresentata dalla figura A.2: capitando su questa casella, se si possiedono almeno 30 monete, si compra una stella;
- la casella contenente delle monete, rappresentata dalla figura A.3: capitando su questa casella si guadagnano un numero casuale di monete, compreso tra 1 e 20;

- la casella contenente un power-up, rappresentata dalla figura A.4: capitando su questa casella si guadagna un power-up casuale;
- la casella contenente del danno, rappresentata dalla figura A.5: capitando su questa casella si riceve un danno, quindi una riduzione dei punti vita, casuale, compreso tra 1 e metà della vita massima, che è 100.

Figura A.2: Stella.

Figura A.3: Moneta.

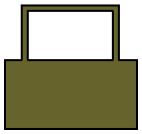

Figura A.4: Power-up.

Figura A.5: Danno.

Quando un giocatore arriva a 0 punti vita, muore. Viene quindi teletrasportato nella prima casella semplice dopo la stella, in modo che riceva una grossa penalità, dovendo fare il giro del tabellone per comprare una stella. Perde anche metà delle monete.

I power-up disponibili nel gioco sono 4:

- medikit: permette al giocatore che lo usa di recuperare tutti i punti vita;
- fucile: permette al giocatore che lo usa di sparare ad un altro giocatore, facendogli perdere 50 punti vita;
- doppio dado: permette al giocatore che lo usa di lanciare due dadi in quel preciso turno, avanzando potenzialmente di più caselle nel tabellone;
- magnete: permette al giocatore che lo usa di far perdere ad un altro giocatore circa un terzo delle monete che possiede.

Il menù per scegliere il power-up da usare è un menù a tendina che rende visibile a primo impatto un solo power-up: per visualizzare gli altri, bisogna scorrere verso il basso.

Quando tutti i giocatori avranno svolto il proprio turno, si dovrà giocare ad un minigioco, scelto casualmente tra questi cinque:

- Mastermind;
- Who risks wins;
- Memo;
- You're The Bob-omb;
- Cut From The Team;

Prima di ogni minigioco apparirà una schermata con la guida per giocarci. Quando ogni giocatore l'avrà letta e sarà pronto per giocare, basterà premere il pulsante sotto di essa per cominciare. Finito il minigioco, il programma stilerà una classifica in base al punteggio ottenuto nel minigioco. Più in alto si arriva, più monete si avrà la possibilità di ricevere. In più, i giocatori posizionati nella prima metà della classifica riceveranno un power-up (nel caso si stia giocando in 3, solo il primo riceverà il power-up).
Se dovesse capitare un pareggio tra due o più giocatori al termine di un minigioco, si effettueranno i playoff: lanciando un dado, chi effettuerà il numero più alto si posizionerà prima in classifica.

Appendice B

Esercitazioni di laboratorio

Nessuno di noi ha consegnato esercizi di laboratorio durante l'anno.