Warlords

Relazione per
"Programmazione ad Oggetti"

Mainardi Giosuè Giocondo
Tartagni Filippo
Magrini Giacomo
Zacconi Andrea

25 aprile 2021

Indice

Capitolo 1

Analisi

Il gruppo si pone come obbiettivo quello di realizzare un gioco di strategia pvp 2D ispirato a Warlords 2 – Rise of Demons, gioco sviluppato in "Flash"[1], tecnologia browser non più supportata da dicembre 2020. Il gioco consiste nello scontro tra due giocatori, con entrambi dei punteggi, che dovranno schierare una truppa alla volta in una delle linee di combattimento con l'obbiettivo di raggiungere il vertice opposto della corsia. I giocatori hanno a disposizione diverse tipologie di truppe da schierare in combattimento, nelle corsie, contro le unità dell'avversario con lo scopo di raggiungere la fine della corsia per fare punto. Con truppa si intende un tipo di unità per il combattimento (soldato, arciere, ecc.). Una corsia di combattimento è un percorso orizzontale in cui le truppe si spostano nella direzione opposta a quella da cui sono apparse e combattono con le unità nemiche.

1.1 Requisiti

Requisiti funzionali

- L'applicativo dovrà dare la possibilità ai giocatori di impostare diversi settaggi che costruiranno l'ambiente di gioco.
- Durante la partita dovrà essere permesso tornare ai menù o terminare l'applicazione senza dover per forza concludere l'incontro.
- Dovrà essere permesso schierare una sola unità alla volta dopo aver atteso la scadenza del rispettivo tempo di ricarica.

[1] `https://it.wikipedia.org/wiki/Videogioco_per_browser`

- Lo scontro dovrà terminare quando uno dei due giocatori ha raggiunto un certo punteggio oppure alla scadenza del timer della partita.
- Dovrà essere contemplato il pareggio tra i due giocatori nel caso in cui la partita termini con lo stesso punteggio per entrambi.
- I comandi per lo svolgimento della partita saranno diversi per i due giocatori in modo tale che non si intralcino.
- I risultati delle partite dovranno essere memorizzati con i relativi punteggi.

Requisiti non funzionali

- L'applicazione dovrà funzionare ugualmente su diversi sistemi operativi (Windows, Mac OS, Linux).
- L'interfaccia di gioco e le varie schermate si dovranno adattare alla risoluzione dello schermo in cui viene eseguita l'applicazione.

1.2 Analisi e modello del dominio

Warlords si basa principalmente su un terreno di gioco diviso in corsie orizzontali. I principali componenti sono organizzati gerarchicamente:

- Il terreno di gioco verrà identificato dal **Field** il quale raccoglierà le corsie,che potranno essere una o più, in una sola entità che le aggiorna.
- Le corsie saranno modellate dall'entità **Lane**, ogni corsia sarà indipendente dall'altra e mapperà la posizione di ogni unità, inoltre si occuperà di muoverle e farle interagire tra loro. Ogni Lane avrà una stessa lunghezza e terrà traccia dei punteggi che i giocatori hanno segnato su quella singola corsia.

  Una corsia rimuoverà un'unità qualora questa sia stata sconfitta o abbia fatto punto.
- Le truppe saranno rappresentate dalla **Unit**, un entità generica la quale avrà le sue caratteristiche (come l'attacco, i punti vita,il passo o il tempo di attesa) definite dal tipo di unità a cui appartiene cioè lo **UnitType**, un'unità inoltre apparterrà a uno solo dei due giocatori, identificato dal **Player**.

Una unità potrà muoversi, attaccare o fare punto, fare punto significa raggiungere e oltrepassere l'ultima posizione della Lane, rispetto alla sua direzione di movimento.

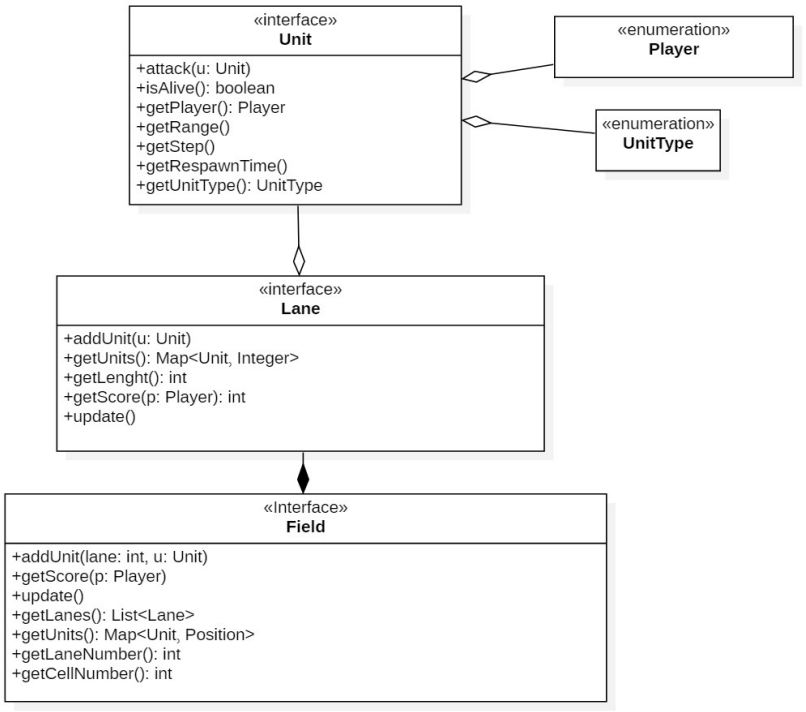


Figura 1.1: Schema UML dell'analisi del problema, con rappresentate le entità principali ed i rapporti fra loro

Capitolo 2

Design

2.1 Architettura

Per realizzare l'applicazione è stato usato il pattern architetturale MVC che garantisce l'indipendenza a evenutali modifiche future grazie alla suddivisione nelle tre parti.

In particolare il Model, rappresentato da Field, definisce la logica di dominio dell'applicazione. Questo componente non è al corrente dell'esistenza di Controller o View, dunque non contiene riferimenti ad essi, sarà compito del Controller utilizzarlo adeguatamente.

La View gestisce l'implementazione grafica della visualizzazione del gioco, eventualmente facendo uso di librerie esterne. Le azioni dell'utente vengono intercettate dalla View che notificherà il Controller di conseguenza. Utilizzare una diversa libreria per rappresentare l'aspetto grafico, stravolgere o sostituire completamente la View, non comporterà modifiche al Controller nè tanto meno al Model.

Il Controller invece agisce come mediatore fra l'ambiente visuale e la logica di gioco, il suo scopo è quello di fornire alla View lo stato attuale del gioco e modificare di conseguenza l'interfaccia grafica, a seguito di azioni dell'utente che hanno cambiato lo stato del Model.

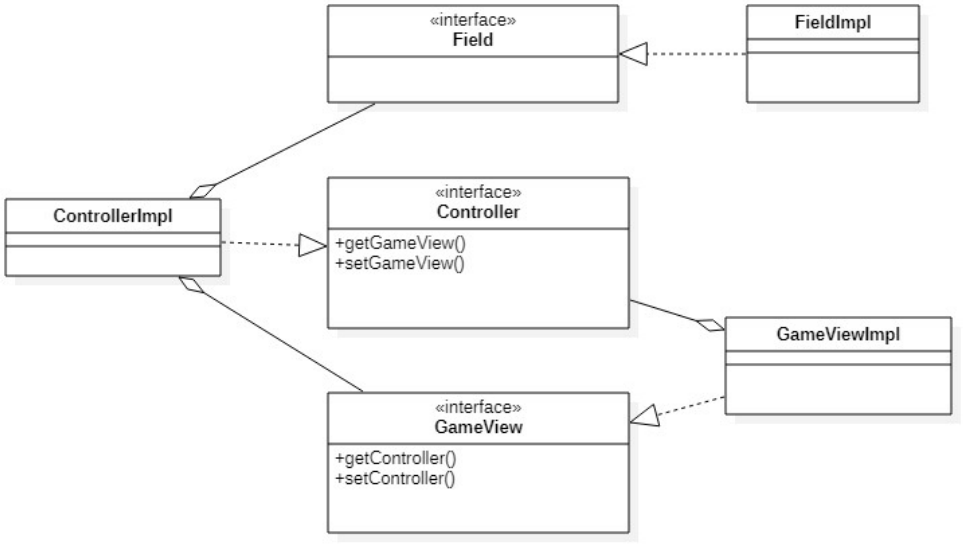


Figura 2.1: Schema UML architetturale di Warlords.

2.2 Design dettagliato

In questa sezione vengono approfonditi nel dettaglio alcuni elementi del design, ciascun membro del gruppo ha una propria sotto-sezione di cui è il solo responsabile.

Giosuè Giocondo Mainardi

Nella fase di analisi sono state definite le varie entità di gioco e ho avuto il compito di progettare la parte di modello del campo di gioco e della relativa trasposizione grafica.

Model Field

Ho modellato le entità campo e corsia tratte dall'analisi del dominio come da Figura 2.2: in modo tale che un `Field` sia composto da una o più `Lane` che possono essere anche di diversa implementazione, oppure estendere quella base sovrascrivendo i metodi non final che definiscono l'effettivo comportamento che assumono le unità in quella Lane. Verso la fine della fase di sviluppo ho visto come la rappresentazione che ho scelto permetta una valide

estensioni dell'Implementazione di base `LaneImpl` consentendo di sfruttare il pattern Decorator per aggiungere funzionalità a un eventuale corsia speciale. Il `Field` notificherà ogni `Lane` nel momento in cui lo stato del gioco vada aggiornato, ogni Lane si occuperà di aggiornare il suo contenuto in base alla propria implementazione. Non avendolo compreso del tutto, non mi sono accorto inzialmente che questo mi ha portato effettivamente ad usare il pattern Observer senza accorgermi.

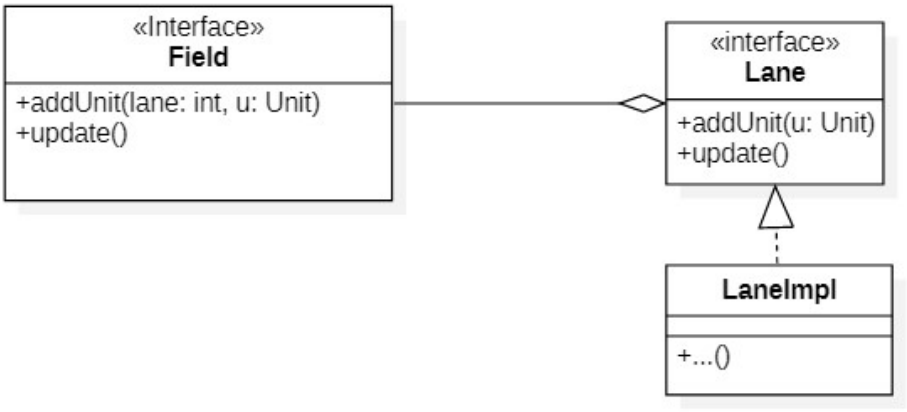


Figura 2.2: Rappresentazione UML del pattern Observer per Field e Lane.

View Field

Il campo da gioco verrà visualizzato graficamente grazie al `FieldView` che si occupa di stampare a video il tipo di unità richiesto dalla GameView principale. Dato che certe unità potrebbero non essere mai schierate in campo durante uno scontro ho scelto di gestire una cache interna la quale va estrapolare l'immagine solo quando effettivamente serve che venga stampata e ne tenga traccia, senza doverla andare ad estrarre di nuovo. Schema grafico in Figura 2.3.

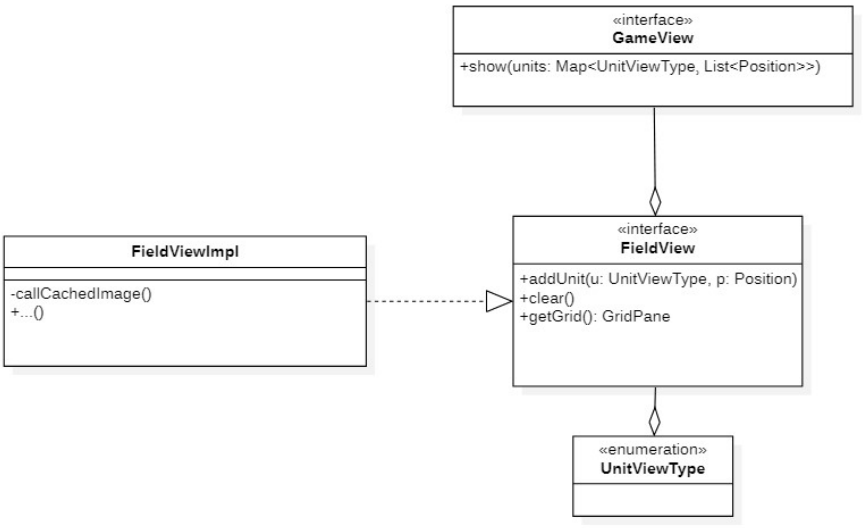


Figura 2.3: Rappresentazione UML di FieldView con il metodo di cache.

Music

Durante la fase di sviluppo è stata introdotta la riproduzione di suoni, inzialmente non pensata, che accompagnano il gioco e i menù, mi sono occupato a riprogettare questa parte, inzialmente introdotta da un altro membro, poichè mi sono accorto di poter far uso di un Singleton dato che anche in future estensioni, non è fondamentale per l'applicazione l'esecuzione di più musiche in contemporanea. Inoltre grazie alla visione estesa del Singleton sarà possibile attivare/disattivare l'audio nel menù principale o durante il gioco, mantenendo salvata la scelta. Una semplice rappresentazione in Figura 2.4.

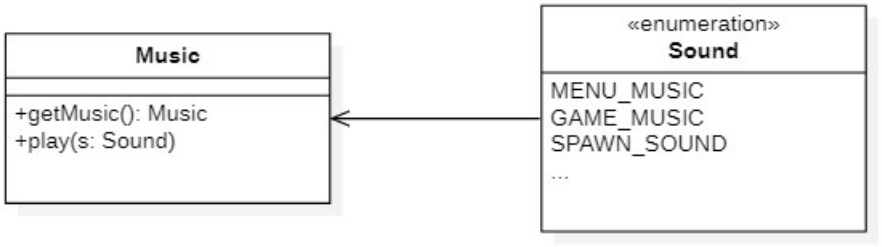


Figura 2.4: Rappresentazione UML del Singleton per la Music.

View Factory

Nella ricerca di fare più ordine nel codice ho visto che nelle classi di View i miei compagni usavano frequentemente certe metodologie ricorrenti per la creazione di elementi. Così ho suggerito di introdurre una Factory per questi elementi fornendo ai compagni i metodi richiesti, per creare i suddetti elementi con lo stesso risultato di prima ma in maniera più compatta e universale, in modo tale che una diversa scelta per la creazione di questi elementi possa essere redatta implementando diversamente l'interfaccia `ViewFactory` come mostrato nello schema:

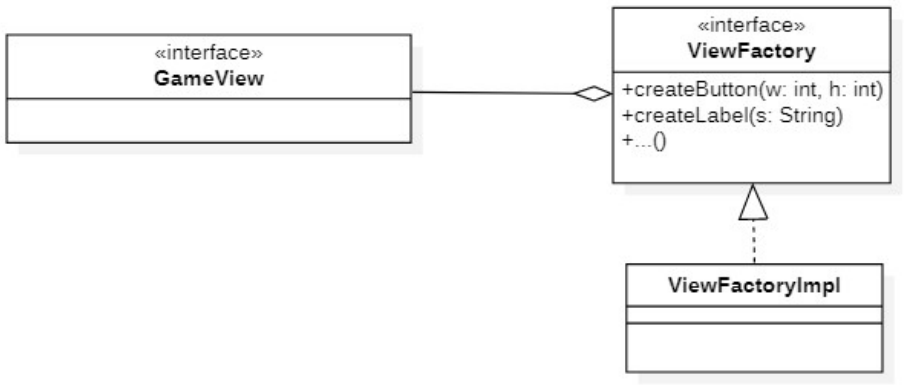


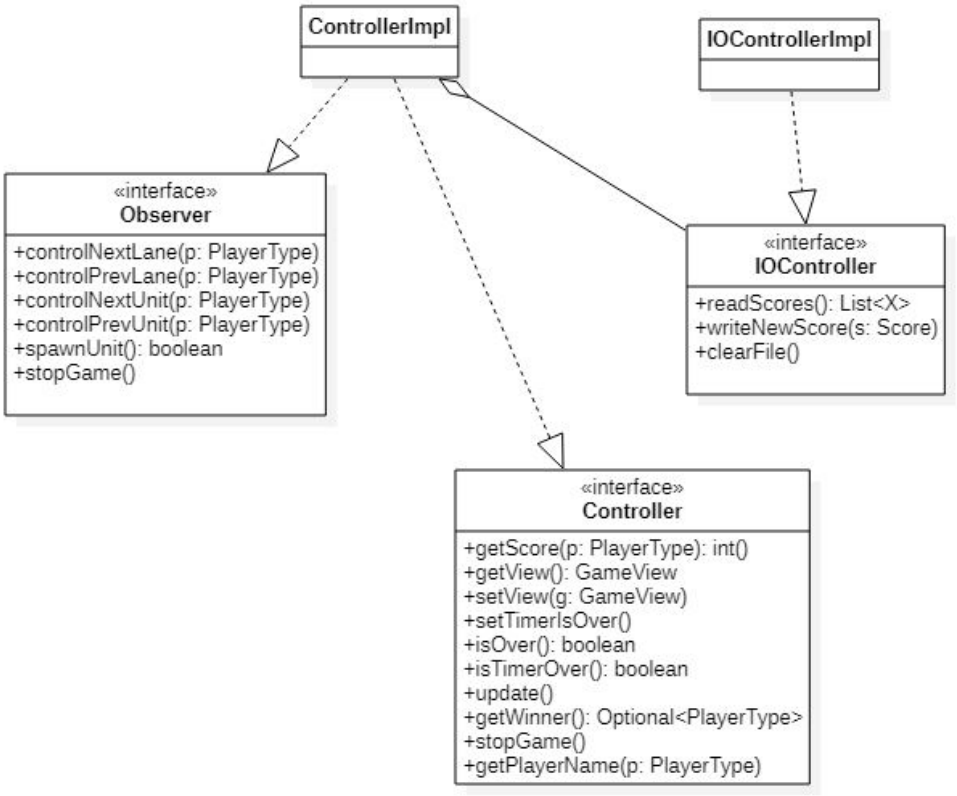
Figura 2.5: Rappresentazione UML della ViewFactory pensata.

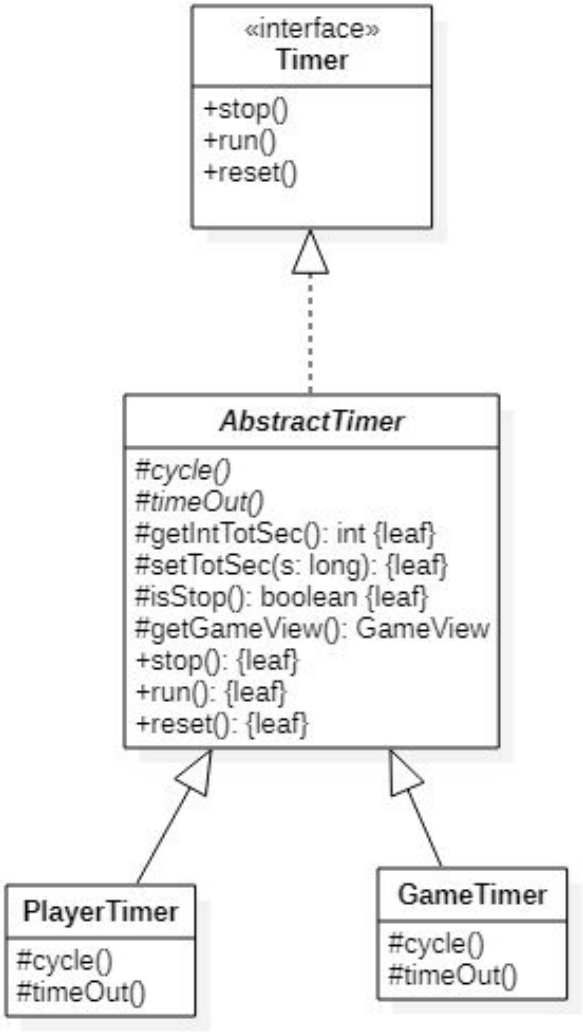
Filippo Tartagni

Per realizzare le classi del timer di gioco principale e quelli che mostrano gli intervalli di spawn dei player ho utilizzato un Template Method (run()). In questo modo le specializzazioni di un Timer, utilizzeranno tutti i metodi definiti nella classe astratta come "final", cambiando però la relativa implementazione dei metodi cycle() e timeOut(). I Timer vengono creati ognuno su un thread diverso, che termina la sua esecuzione nel momento del termine di una partita.

Il ControllerImpl è il cuore del gioco, esso implementa due interfacce: Controller e Observer. La prima definisce i metodi che interagiscono con le entità del model per la vera e propria realizzazione della partita (raggiungimento del fine partita, vincitore, spawn di truppe etc..). Al suo interno la classe del Controller, ha un oggetto GameLoop, un thread incaricato di aggiornare continuamente con una certa frequenza,(update()) lo stato interno del gioco. L'altra interfaccia, Observer, contiene metodi che vengono richiamati dall'oggetto di View "GameViewImpl" nel momento in cui accadono determinati eventi: nel nostro caso l'input da tastiera di uno dei due giocatori, richiede l'invocazioni di questi metodi, che aggiornano la struttura interna del gioco. Il controller, inoltre, comunica alla view i cambiamenti. Questa classe ControllerImpl, ha riferimenti a Timer, IOController e GameView (oltre che a SettingsController).

L'interfaccia IOController definisce i metodi per la lettura e scrittura su file dei risultati. Per memorizzare i punteggi dei giocatori, una volta terminata una partita, è stato utilizzato un "POJO" Score, in cui mettere appunto i dati relativi alla partita, successivamente questo oggetto viene serializzato e deserializzato. Per fare ciò, vengono utilizzati metodi appositi della libreria GSON, oltre ad oggetti FileWriter e FileReader all'interno di costrutti try-with-resources.


ControllerImpl
IOControllerImpl
«interface»
Observer
+controlNextLane(p: PlayerType)
+controlPrevLane(p: PlayerType)
+controlNextUnit(p: PlayerType)
+controlPrevUnit(p: PlayerType)
+spawnUnit(): boolean
+stopGame()
«interface»
IOController
+readScores(): List<X>
+writeNewScore(s: Score)
+clearFile()
«interface»
Controller
+getScore(p: PlayerType): int()
+getView(): GameView
+setView(g: GameView)
+setTimerIsOver()
+isOver(): boolean
+isTimerOver(): boolean
+update()
+getWinner(): Optional<PlayerType>
+stopGame()
+getPlayerName(p: PlayerType)



Giacomo Magrini

La mia parte nel progetto di OOP prevede la creazione della maggior parte della View e la gestione degli action event dei vari bottoni presenti. Per lo sviluppo delle componenti grafiche ho decise di utilizzare JavaFX essendo una libreria versatile e con elementi aggiuntivi molto utili rispetto a Swing. La risoluzione delle varie schermate è stata impostata tramite un metodo statico che controlla le dimensioni dello schermo dell'utente e restituisce la

risoluzione appropriata per il tipo di schermo che l'utente sta utilizzando. Tale metodo è stato inserito da un mio collega nella sua classe appropriata, io richiamerò il metodo all'inizio di ogni classe per impostare delle costanti statiche che serviranno poi per la disposizione corretta degli elementi all'interno della schermata. In questa maniera ogni volta che l'utente ridimensionerà la sua risoluzione del monitor il gioco cambierà di risoluzione in automatico ridisponendo tutti gli elementi in modo corretto. Per la costruzione delle varie classi ho deciso di non utilizzare il software SceneBuilder per avere del codice più chiaro, pulito e avere controllo della qualità del codice. Il MainMenu estende la classe Application e sarà la prima schermata che verrà mostrata all'avvio dell'applicazione all'interno avremo 2 metodi principali: -start: tramite questo metodo creo lo Stage e la Scena che rimarranno unici per tutto la durata d'esecuzione del gioco. -createPane: all'interno di questo metodo invece è presente il cuore della classe MainMenu, ovvero il metodo che creerà tutti i vari bottoni e layout. In realtà Il metodo createPane è presente in tutte le cassi essendo necessario per la creazione dei layout, bottoni, label e tutti gli altri elementi di JavaFX utilizzati In tutte le classi sviluppate ho deciso di utilizzare un Borderpane inserito poi nel Pane che verrà restituito dal metodo createPane. Ciò ci servivà poi come verrà spiegato successivamente per eseguire il cambio di schermata corretto quando si premono i bottoni. Dal MainMenu si accede tramite click di bottoni alla schermata del Tutorial, dello Scoreboard e della GameModeSelection. E pure presente un bottono per silenziare la musica di gioco (aspetto di cui parlerò in seguito). Il cambio di scena avviene tramite action event dei rispettivi bottoni, all'interno dell'evento si istanzia la classe nella quale si vuole cambiare schermata e tramite il metodo setAll si passa il createPane della nuova classe su cui si vuole effettuare il rispettivo cambiamento di scena. Tramite questo metodo ci assicuriamo di avere sempre un'unica Scena e un unico Stage, per questa ragione parlare di cambio di scena non è del tutto appropriato essendo la scena sempre una sola. In GameModeSelection do la possibilità all'utente di scegliere le impostazioni della partita, tali impostazioni (che saranno inserite di default nella classe tramite i campi) potranno essere cambiate dall'utente con il semplice click del bottone con l'impostazione desiderata. Gli action event dei bottoni provvederanno a cambiare i campi della classe con i nuovi valori, ciò è importante perché serviranno poi al controllore per generare la mappa di gioco (GameViewImpl) con le impostazioni scelte dall'utente. In Scoreboard è presente una ListView nel quale sono registrati i risultati di tutte le partite svolte. In GameTutorial è presente una miniguida sull'obbiettivo del giocatore e i vari tasti premibili. GameViewImp è la principale interfaccia grafica del gioco, nel quale avvengono le partite multigiocatore, questa classe è stata sviluppata in parte da tutti i membri del gruppo di

progetto, io (oltre ad impostare i layout, bottoni, label ecc.) ho gestito i vari metodi per terminare correttamente la partita tramite l'ausilio della classe confirmBox (di cui parlerò a breve) e gli action event dei bottoni per uscire dal gioco o tornare al menu principale.

NB: le interfacce (inizialmente non presenti) sono poi state create dal mio collega Andrea Zacconi.

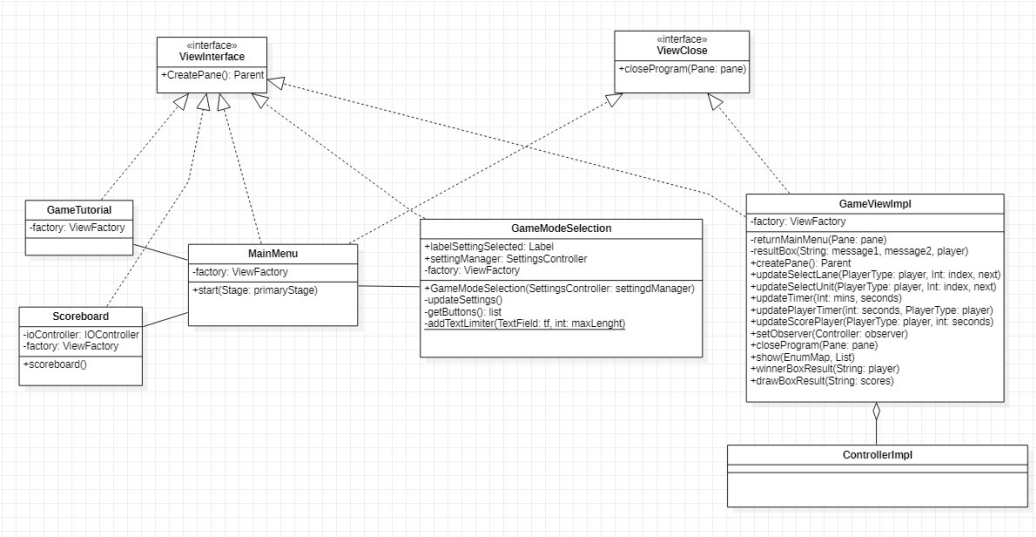


Figura 2.6: Collegamenti tra le varie classi.

Ho poi creato la classe ConfirmBox con un metodo statico display che genererà il layout e i bottoni di questa schermata ritornando un booleano a seconda della scelta dell'utente, ciò mi è servito in ogni occasione in cui è necessario chiedere all'utente una conferma o l'esecuzione di un'azione obbligatoria da parte dell'utente: in dettaglio quando si vuole uscire dal gioco, tornare al menu o quando la partita finisce (in pareggio o vincita di uno dei due giocatori) e verrà chiesto all'utente di scegliere se tornare al menu o uscire dal gioco. Saranno poi i metodi a decidere cosa succede dopo che l'utente ha selezionato una delle 2 opzioni premendo i rispettivi pulsanti del ConfirmBox e sono sempre i metodi a decidere tramite passaggio di parametri il messaggio nella label, il testo nei bottoni e il nome della finestra creata. Questo ci garantisce una riusabilità di questa classe senza dover ogni volta crearne una nuova qualora si voglia chiedere una conferma all'utente. Grazie a questa finestra garantiamo un'uscita corretta dall'applicazione, terminando anche tutti i thread creati dai timer (di cui si è occupato Filippo Tartagni), la X rossa della finestra è stata disabilitata per non creare bug nel caso si

cerchi di chiudererla in quel modo, e di conseguenza killare i thread, anche quando non ce ne sono di attivi (ad esempio se nel main menu si cerca di uscire dal gioco prima dell'avvio della partita).

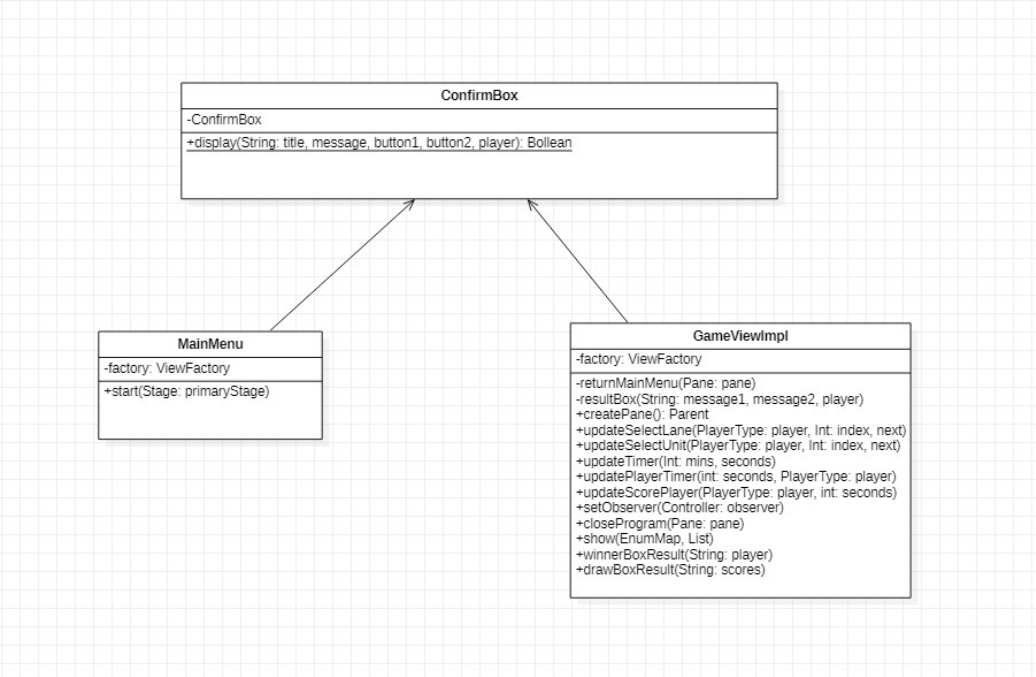


Figura 2.7: UML di confirmBox in relazione con MainMenu e GameViewImpl.

Confrontandomi con i miei colleghi abbiamo notato che per la creazione degli elementi di JavaFX si poteva utilizzare il pattern Factory riducendo notevolmente il codice ripetuto, lo stesso pattern è poi stato utilizzato per la gestione delle immagini Si istanzia l'oggetto Factory e si utilizza il metodo appropriato di Factory per la creazione di ciò che ci interessa passando in ingresso i parametri richiesti, per esempio se si vuole creare un bottone si passerà in ingresso il testo del bottone, lo style del bottone e la grandezza (widht, height) del bottone, si esegue questo procedimento ogni volta che si vuole creare un elemento.

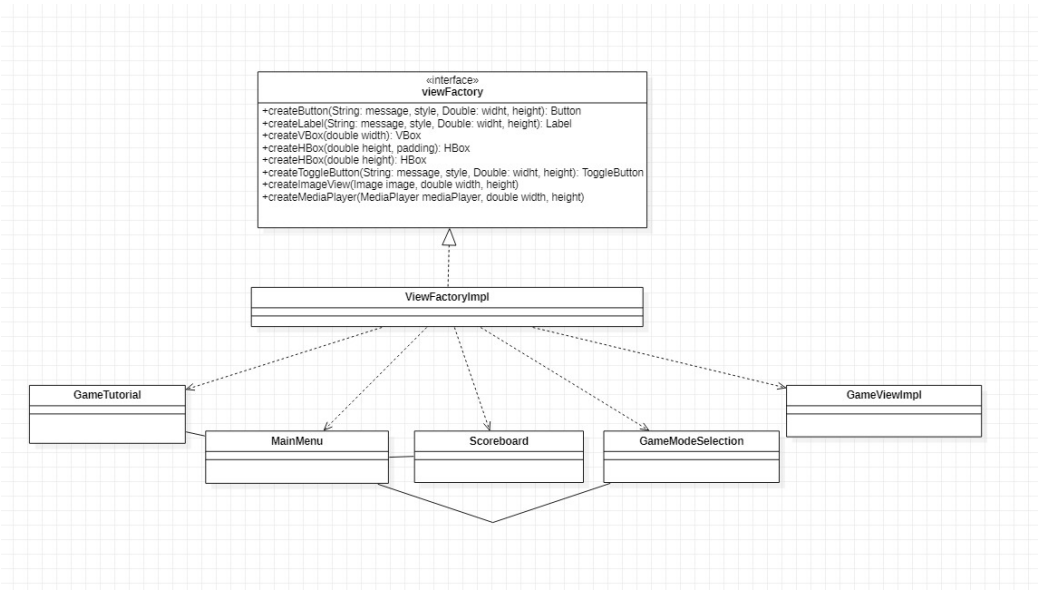


Figura 2.8: UML Factory.

Andrea Zacconi

La parte del progetto affidatami, riguarda la creazione nel Model della classe Unit, l'Interfaccia di gioco della View(insieme al mio collega Giacomo Magrini) e eventuali collegamenti Controller-View.

UnitModel

Le classi Unit nel Model hanno il compito della creazione delle unita' come oggetto, specificandone le caratteristiche(vita, danno, ecc.), le azioni basi(attacco ad un'altra unita', movimento, ecc.) e di differenziarne la tipologia(unita' spadaccino, unita' arcere, ecc.). La differenzazione delle unita' avviene tramite una classe Enum che viene utilizzata dalla classe UnitImpl. Formato l'oggetto Unit(vedi Figura 2.9) questo verra' utilizzato nel Model della Lane/Field per essere impegato nei combattimenti e spostamenti.

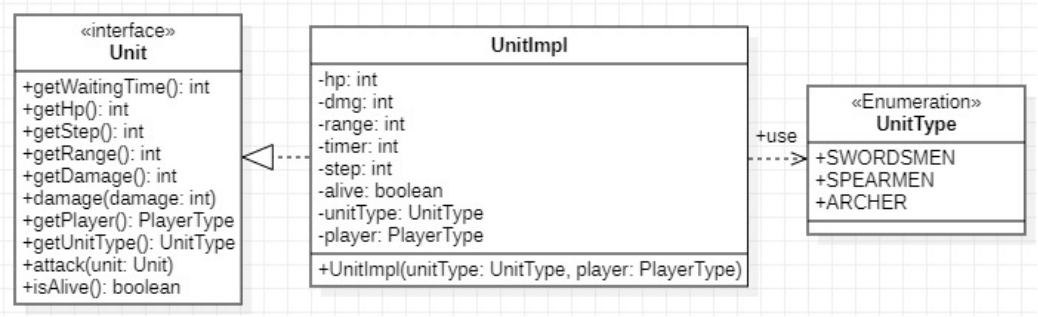


Figura 2.9: Schema UML della UnitModel.

Interfaccia grafica e collegamenti Controller-View

Alcune interfacce e classi utilizzate nella view sono di assoluta importanza e necessitano di essere citate.

L'interfaccia ViewClose e' un'interfaccia utilizzata da alcune classi della View per la chiusura del programma via bottone. Essa permette l'utilizzo del metodo closeProgram(), utilizzato dalle classi della View, per il richiamo di un'altra classe, la ConfirmBox(vedi Figura 2.10). Per fare il punto, questa interfaccia permette il collegamento/richiamo della classe ConfirmBox da parte delle altre classi della View. Per maggiori dettagli sulla ConfirmBox vedere la parte di Magrini Giacomo.

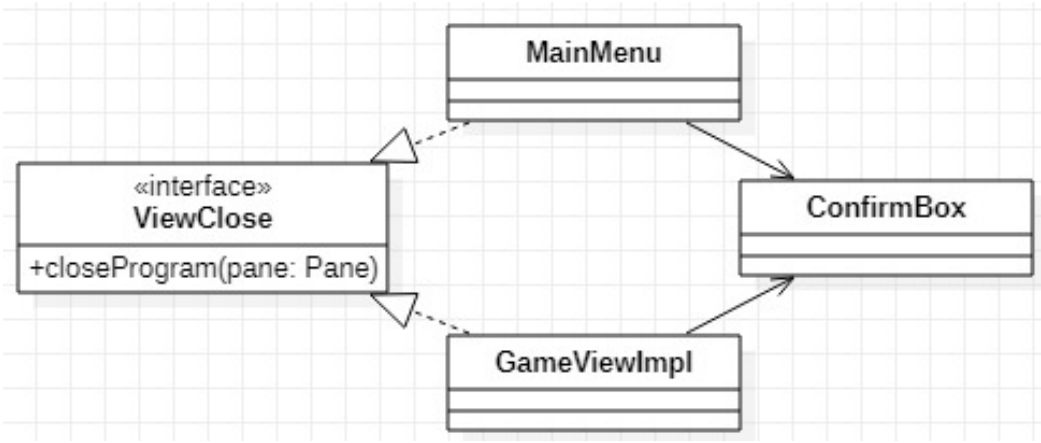


Figura 2.10: Schema UML interfaccia ViewClose.

L'interfaccia ViewInterface e' l'interfaccia principale utilizzata per la creazione delle scene. Essa presenta il metodo createPane(), che permette alle

classi della View di creare la propria scena e tutte le sue caratteristiche. Come per l'interfaccia ViewClose, la ViewInterface permette il collegamento tra le varie classi della View e il loro rispettivo funzionamento(vedi Figura 2.11). Per maggiori dettagli sul collegamento delle scene e il loro design, consultare la parte di Magrini Giacomo.

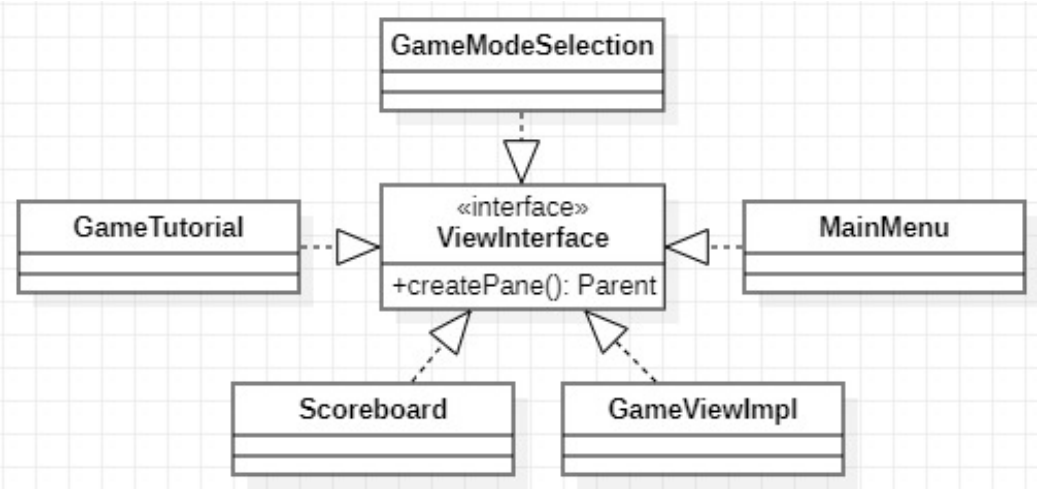


Figura 2.11: Schema UML interfaccia ViewInterface.

La classe ViewResolution e' utilizzata da tutte le classi della View per il ridimensionamento e adattamento degli oggetti grafici(label, bottoni, ecc.), alla risoluzione dello schermo. Questa prende le dimensioni dello schermo(altezza e grandezza) e le divide per una costantante della ViewConstants a nostro piacimento/necessita', ottenendo un ridimensionamento dell'oggetto(vedi Figura 2.12).

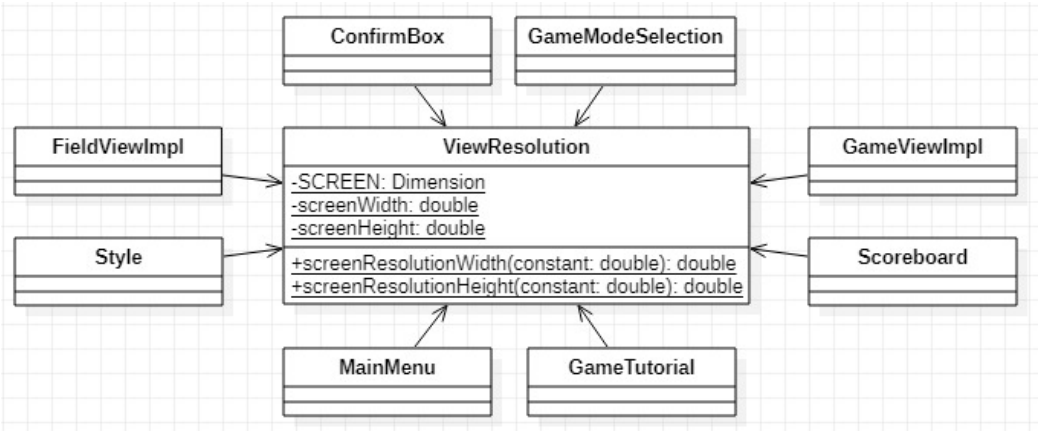


Figura 2.12: Schema UML interfaccia ViewInterface.

Passaggio di dati dalla View al Controller. Questo puo' essere un grosso problema nel rispetto del pattern, ma nel design sviluppato nel progetto, il seguente problema non si presenta. Prendiamo in esempio il dato su cui ho lavorato, I nomi dei player. I player possono inserire I propri nomi nella classe GameModeSelection attraverso delle TextField, per poi essere inseriti nel costruttore di SettingsControllerImpl implementato dentro la classe di GameModeSelection. Da questo costruttore vengono passati I dati al costruttore del ControllerImpl. Il controller salva i dati in variabili locali per poi essere utilizzati da esso e passati eventualmente alla GameViewImpl(vedi Figura 2.13). Questa parte e' stata principalmente sviluppata dal mio collega Filippo.

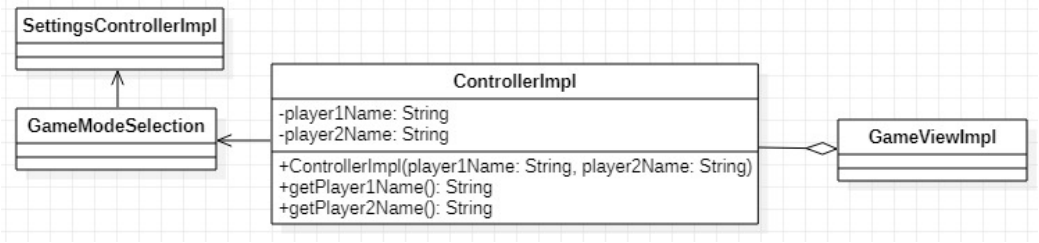


Figura 2.13: Schema UML interfaccia ViewInterface.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Durante la creazione del modello e prima della costruzione del relativo Controller si è avuto il bisogno di considerare l'uso di test per verificare che ciò che era stato fatto fosse corretto e per controllare che eventuali modifiche intraprese non compromettessero il lavoro fatto. Per fare questo è stata utilizzata la libreria Junit (versione 5), per rendere i test automatici. Le parti controllate in questa fase sono state:

- Gestione del combattimento tra diverse unità
- Schieramento e rimozione di un'unità in una corsia
- Movimento delle unità nella corsia
- Generazione del Campo (Field) e interazione con le corsie (Lane)

Non volendosi cimentare nel test di interfcce grafiche, sono stati eseguiti periodicamente test manuali del gioco che hanno accompagnato la fase di sviluppo.

3.2 Metodologia di lavoro

Per il lavoro in team è stato adottato il DVCS Git con un Workflow semplificato che coinsiste nello sviluppo dell'applicazione su un branch develop, dal quale poi un membro in caso di necessità creava un branch separato "feature-nomesperimentazione" in cui ha poteva lavorare autonomamente in parallelo su una funzionalità a lui assegnata, senza interferire significativamente sulla linea di sviluppo principale da cui periodicamente ne estraeva eventuali

modifiche dei compagni. Al termine della corretta implementazione della funzionalità stabilità, si eseguiva un ulteriore pull dal develop correggendo eventuali conflitti, per poi riunirsi alla linea di sviluppo principale.

In seguito al raggiungimento di ogni significativo traguardo dell'applicazione si è poi fatta una merge del branch develop su quello principale, il master.

Questo processo si è iterato fino al completamento delle funzionalità poste come obbiettivo che hanno portato al termine del lavoro.

Si vuole indicare in questa sezione che tutti e quattro i membri del gruppo hanno lavorato principalmente in ambiente Windows. Si è poi testata la funzionalità dell'applicazione in ambiente Linux, più precisamente nella sistema operativo Ubuntu 20.04.2 LTS.

Giosuè Giocondo Mainardi

Come primo passo per l'effettivo sviluppo abbiamo fatto uso di schemi UML in cui rappresentavamo le varie interfacce che saremmo andati a costruire.

Mi sono concentrato su come poter modellare il dominio applicativo secondo quanto stabilito nelle riunioni periodiche con il gruppo cercando di riuscire ad offrire, alle classi dipendenti dalle miei, ciò che era richiesto. Per prima cosa mi sono cimentato nella realizzazione di Field e Lane che modellano lo scontro, essendo agli inizi non avevo modo di testare quanto fatto se non con dei Test automatizzati. Grazie a questi ho scovato vari bug man mano e ho modificato poi le varie implementazioni dei metodi in base a ciò che doveva risultare dai test. Ad esempio, inizialmente le unita' non venivano correttamente rimosse dalle lane, questo succedeva perché nel metodo update rimuovevo dal Set delle unita', un elemento in un ciclo del foreach, quindi il ciclo si interrompeva perché non riusciva a proseguire con l'iterazione. Dopo poco tempo mi sono ricordato che questo problema era stato affrontato in laboratorio e mi sono andato a cercare la soluzione che era per l'appunto usare un iteratore per ciclare sulla collezione, così da poter rimuovere senza problemi un oggetto dal Set anche durante l'iterazione.

Dopo diverso tempo, quando il campo di gioco è stato realizzato nella view, la mia parte di Model ha saputo rendersi già pronta e funzionante senza alcun bisogno di modifiche, grazie ai vari test eseguiti nel corso dello sviluppo.

Per realizzare la parte grafica del campo di gioco ho seguito quanto deciso con il gruppo cioè di adottare una GridPane e di riempirla con immagini, che potevano anche essere vuote, la griglia verra' poi riempita dalle unita' aggiunte.

Per il tracciamento della posizione delle unita' nella singola corsia e del punteggio dei giocatori ho deciso di usare una mappa la quale però in caso di sovrascrittura di una chiave non offre prestazioni ottime, quindi ho deciso di importare in un package utilities i vari Counter, visti come esempi nel corso delle lezioni, così da poter modificare il valore mappato semplicemente andando a richiamare un increment Inoltre i Counter mi offrivano anche la possibilità di impostare un limite e di fare un multi incremento, funzionalità che ho sfruttato dato che ho associato ad ogni unita' il numero di passi che ha compiuto. Così che il raggiungimento della fine della corsia sia il raggiungimento di un certo numero di passi, indipendetemente dalla direzione percorsa.

Ho cercato sempre di mantenere un codice chiaro e pulito fin da subito, predisponendo la documentazione delle interfacce quasi immediatamente così da poter fornire agli altri membri una chiara specifica dell'uso della mia parte.

Classi/Package quasi esclusivamente miei:

- Field e Implementazione
- Lane e Implementazione
- Utilities Counter
- FieldView e Implementazione
- UnitViewType
- Converter
- Test di Field e Lane
- ViewFactory e Implementazione

Classi/Package in cui ho dato un contributo:

- Music e Sound
- Controller e Implementazione
- Timer
- Unit e UnitType
- GameView e Implementazione
- Unit, Implementazione e UnitType

Filippo Tartagni

Realizzazione package controllers.io (classe e interfaccia per salvataggio lettura/scrittura su file del risultato), package controllers.timer (classi e interfaccia che modellano i timer di gioco), package controllers.settings (classe e interfaccia per settare le impostazioni di una partita), le classi e interfacce all'interno di controllers (tranne Converter) che si occupano del mero svolgimento della partita. Ho lavorato inoltre in parte alla classe di View "Scoreboard" per potere inserire la visualizzazione dei risultati e in parte anche ad alcuni metodi nella classe GameViewImpl, tra i quali quelli che vengono chiamati dal Controller per la visualizzazione delle unità e lane selezionate e timers. Contributo nella classe GameModeSelection negli aspetti relativi al passaggio delle scelte dell'utente al controller.

Giacomo Magrini

Menù di gioco, impostazioni partita, interfaccia di gioco, Media di gioco (suoni e immagini). Classi sviluppate:

- ConfirmBox
- GameModeSelection (parte grafica esclusa le textfield)
- GameTutorial
- MainMenu
- ScoreBoard (parte grafica)
- Style
- ResourcesConstants
- GameViewImpl (parte grafica in collaborazione con Andrea Zacconi)
- Music
- Sounds

Andrea Zacconi

Unit Model, View e eventuali collegamenti View-Controller.

Le parti piu' nello specifico sviluppate da me sono:

- Classi Unit, UnitType, UnitImpl, UnitConstants.

- Interfacce ViewResolution, ViewClose, ViewInterface.
- Classe ViewResolution.
- MainMenu background.
- TextField GameModeSelection con Listener e passaggio dei nomi nel Controller.
- Prototipo iniziale dell'Input da tastiera(modificato in alcuni aspetti dal mio collega Filippo per il rispetto dell'MVC).
- GameViewImpl (insieme al collega Magrini Giacomo).
- Collegamento tra le scene nella View(insieme al collega Magrini Giacomo).

Sottogruppo: Magrini Zacconi

Collegamento tra scene della View: Dopo la realizzazione delle scene della View, Zacconi e il collega Magrini si sono imbattuti in un lavoro piuttosto complesso, quello del collegamento tra di esse. Per torvare una soluzione, il "team View" si e' diviso per maggiori possibilita' di riuscita nel compito: Magrini consultava le wiki e guardava tutorial sulla piattaforma YouTube. Zacconi, in base anche alle direttive del collegha, testava su Eclipse i vari tool di javafx per una possibile soluzione. Dopo del tempo, si era giunto ad un bivio con 2 possibili metodi per il collegamento:

- Zacconi proponeva l'utilizzo della funzione astratta start(PrimaryStage). Essa implementa lo Stage(scena, bottoni, label, ecc.) e da questa funzione si puo' modificare il complessivo della scena senza problemi. Fornendo ad ogni scena la funzione start, era possibile l'utilizzo di un singolo Stage, quello del MainMenu, che attraverso un EventHandler andava a modificare se stesso, richiamando la start di un'altra scena. Questo metodo era gia' stato testato e funzionava correttamente.
- Magrini proponeva l'utilizzo della funzione createBorderPane(). Essa si basava sullo stesso ragionamento del metodo Zacconi, ma le differenze sostanziali erano 2:-1) Invece di andare a lavorare sullo Stage, questo veniva creato prima e ci si concentrava sul nodo del BorderPane. -2) La funzione createBorderPane() restituiva un Parent, permettendo a questa di essere utilizzata facilmente in funzione della javafx. Questo metodo e' stato testato e risultava "buggato".

In conclusione, la soluzione finale e' stato il metodo Magrini con alcune modifiche da parte del gruppo. Invece di utilizzare un BorderPane si utilizza un Pane, nodo piu' completo e per ragioni al gruppo sconoscute non "buggato", a differenza della sua controparte(nei test, il BorderPane non ritornava correttamente e completamente modificato senza alcuna modifica da parte del gruppo). La funzione e' stata rinominata createPane() e testata con esito positivo.

La scelta del metodo Magrini e' stata appurata anche per altri due fattori: in base alla genericita' e alla latenza. Essendo nuovi nella libreria javafx, a rigor di logica, il gruppo aveva stabilito che il metodo Magrini modificato era il migliore in entrambi gli aspetti. Genericita' perche' il ritorno di un Parent permetteva l'utilizzo della funzione anche in altri ambiti e facilitava l'utilizzo di essa con le varie funzioni della javafx. Latenza(qui e' dove siamo andati piu' a logica) perche' modificare un Pane a differenza di uno Stage, sembrava molto piu' leggero. Facendo un esempio: se noi consideriamo lo Stage come una bottiglia d'acqua e il Pane l'acqua(dato che il Pane e' un nodo creato dentro lo Stage), ci e' molto piu' economico cambiare l'acqua all'interno della bottiglia, piuttosto che cambiarla completamente.

3.3 Note di sviluppo

Inizialmente non sapendo bene come architettare l'applicazione abbiamo sfogliato un po' il repository con i progetti degli anni passati, anche se non abbiamo trovato applicazioni che si avvicinavano al nostro particolare tipo di gioco.

Giosuè Giocondo Mainardi

Elenco feature avanzate utilizzate:

- Lambda expressions e Stream dove possibile
- Optional
- Libreria apache.commons per il Pair
- JavaFX
- Meccanismo di chaching delle immagini

Filippo Tartagni

Ho utilizzato la libreria GSON (https://github.com/google/gson) di Google per permettere la scrittura e lettura dei risultati di gioco nel formato JSON. - Libreria JavaFX. Dove possibile sono state utilizzate Lambda Expressions e Stream per un codice più compatto, leggibile e pulito.

Giacomo Magrini

- ho utilizzato JavaFX, una libreria da me sconosciuta, ho consultato spesso siti web come stackoverflow e la documentazione di java per capire meglio il funzionamento dei componenti e la rispettiva sintassi.
- ho sfruttato le lamda expression per creare gli actionevent dei bottoni.
- ho creato una classe Style e un file listView.css in cui passo stringhe di codice CSS per impostare lo style dei componenti di javaFX
- ho caricato le varie immagini come campi nelle classi una volta sola per ottimizzare il programma e ridurre il lag durante la fase di gioco

Andrea Zacconi

- Utilizzo delle lambda expression nella View. Essa e' costituita per lo piu' da Event Handler che permettono le azioni dei suoi vari elementi come bottoni, input da tastiera/mouse, ecc.
- Utilizzo della javafx. Libreria inizialmente complessa soprattutto per l'assenza di SceneBuilder, io e il mio collega siamo riusciti a maneggiarla e ad implementare la View a codice. Ci e' capitato piu' volte di dover consultare diverse wiki, principalmente Stack Overflow e Oracle.
- Encoding di file video e audio per la libreria javafx.media e utilizzo di software di conversione file. Mi son ritrovato a dover cercare nelle wiki di Oracle, tutte le tipologie di conversioni compatibili per la libreria javafx.media. Questo solo per il corretto funzionamento dello sfondo animato.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Giosuè Giocondo Mainardi

Penso che nel complesso la mia parte l'ho fatta in modo ottimale, durante le discussioni ho suddiviso diverse volte i ruoli così da potersi organizzare e arrivare alla volta successiva ognuno con una parte in più. Tutto il carico di lavoro è stato appesantito dalle lezioni del nuovo semestre iniziate in concomitanza e dal momento delicato della pandemia che ci ha imposto a dover stare parecchie ore in chiamate per un motivo o per un altro. Penso che se riniziassi il progetto lo farei abbastanza diverso dato che ora ho un altro livello di preparazione in merito alla progettazione. Oggettivamente penso che la mole del mio lavoro sia un po' maggiore di quella del resto del gruppo, ma ho cercato di fare il meglio che potevo essendo anche nuovo nel mondo dell'informatica, non avevo per niente idea di che approccio adottare come primo progetto così complesso. Nel gruppo non siamo omogenei come livello di preparazione il che ha intaccato lo svolgimento del progetto.

Filippo Tartagni

Considerato il momento storico che ci ha fatto svolgere tutto il progetto a distanza, attraverso diverse chiamate settimanali con i compagni, mai visti nè conosciuti prima, e il fatto che sono entrato nel gruppo per ultimo, a progetto già scelto, la realizzazione del progetto non è stata per nulla facile. Avevo già partecipato a lavori di gruppo, ma questo è stato sicuramente il più complicato. Questo progetto, nonostante non mi renda molto soddisfatto penso che sia stato comunque molto utile per il mio bagaglio di conoscenza ed esperienza e, mi ha insegnato sia "cosa fare" in certe situazioni sia cosa

non fare (fatto non scontato). Ritengo di essere stato disponibile ad aiutare e collaborare con i miei compagni, cercando di fornire sempre il mio punto di vista riguardo problemi riscontrati e decisioni da prendere. Non credo che nel futuro estenderò il gioco realizzato, in ogni caso ritengo che per come sono state fatte le cose non sarebbe difficile cambiare la tecnologia della parte di View e ad esempio realizzarei una interfaccia di gioco che includa animazioni e dettagli sulla vita delle truppe. Un altro aspetto che andrebbe a migliorare quanto fatto sarebbe l'utilizzo di thread e concorrenza con i dovuti meccanismi di sincronizzazione.

Giacomo Magrini

Mi ritengo abbastanza soddisfatto del lavoro svolto, ha sicuramente ampliato la mia conoscenza dell'aspetto grafico di un applicazione java grazie allo studio individuale della libreria di JavaFX. Il lavoro di gruppo è stato essenziale specialmente durante le fasi di analisi per avere ben chiaro come sviluppare l'applicazione. Non avendo alcun tipo di esperienza nel campo dello sviluppo di un videogioco è stata un po' dura all'inizio entrare nella giusta ottica, una volta studiata bene la libreria di JavaFX le cose sono diventate sicuramente più chiare. È stato molto utile l'utilizzo dei design patterns studiati durante questo corso, ciò ha evitato molto lavoro di ripetizione di codice che sarebbe anche risultata in una cattiva programmazione. In futuro mi piacerebbe poter espandere il progetto o svilupparne uno per conto mio per implementare delle feature che per mancanza di tempo non sono state potute aggiungere.

Andrea Zacconi

Posso ritenermi soddisfatto del progetto e di come esso sia stato sviluppato nel tempo da parte mia e dei miei colleghi. Il mio ruolo principale e' stato "qua e la" a causa della semplicita' delle mie classi(non richiedevano nessun particolare algoritmo o altro di avanzato), percio' ho cercato il piu' possibile di rendermi utile anche in sottigliezze come il testing della View e lo sviluppo di alcune parti di essa, il fixing di warnins, la progettazione di alcune parti del progetto e il rendermi utile per I miei colleghi in caso di necessita'. Per il futuro sarei molto propenso a continuare il progetto per un eventuale data di uscita su piattaforme, ma dato il tempo a disposizione, questa cosa e' un'incognita.

4.2 Difficoltà incontrate e commenti per i docenti

Giosuè Giocondo Mainardi

Non ci sono dubbi che questo corso sia molto più pesante di altri, però questo denso carico di lavoro penso sia stato utile per l'apprendimento di tante cose sia del linguaggio che di progettazione o di strumenti avanzati come il DVCS o Gradle.

In generale il corso mi ha fatto fare quello che penso sia un bel salto di qualità in termini di programmatore. Se prima di questo corso avevo dubbi se proseguire della carriera universitaria o meno ora non ne ho dati i continui richiami che i professori hanno fatto come "questo lo vedrete alla Magistrale".

Filippo Tartagni

Ho seguito tutte le lezioni del mio piano di studi di laurea Triennale e reputo questo, come il corso che mi ha offerto le lezioni più utili e fornito più conoscenze degli altri. L'impegno richiesto è molto e il "monte ore" necessario per migliorare le proprie "skills" da permettere di passare i due esami (mi sono confrontato anche con amici che li hanno già passati) forse sono troppo alti. Ho trovato un po' troppo riassuntiva e veloce la spiegazione di JavaFx. Essendo un corso che insegna molte cose, dal mio punto di vista si potrebbe pensare di delegare qualcosa ad altri corsi del CdL, ad esempio si potrebbe insegnare Git in un corso precedente, in modo tale da massimizzare il tempo di laboratorio dedicato al "vero e proprio codice". Stesso discorso per il software Gradle, spiegato a mio avviso troppo velocemente senza poterne capire le vere funzionalità/potenzialità. Il fatto che nel corso vengano spiegati i Design Patterns è una cosa molto positiva, penso che sarebbero ancora più apprezzati se, ad esempio, negli ultimi laboratori vengano forniti degli esercizi che prevedano l'implementazione di questi concetti. Non so se i Design Patterns sono un argomento che si sviluppa di più alla Magistrale ma valuterei anche di aggiungere la spiegazione ancora più dettagliata di questi nel corso di Ingegneria del Software, un corso che secondo la mia opinione è bello ma rimane troppo "teorico".

Andrea Zacconi

Le maggiori difficolta' incontrate prevalentemente sono:

- La gestione e il lavoro per il pattern MVC, cosa poco trattata nelle lezioni.
- Alcuni casi di non comunicazione con i miei colleghi, scaturendo "la-draggine" di lavoro altrui.
- La completa estranezza nell'ambiente Git e Gradle(meno per gradle).

Appendice A

Guida utente

L'applicazione all'avvio mostra il menù principale (Figura A.1). Da cui accedere ai diversi sottomenù facendo uso dei bottoni nell'interfaccia grafica.

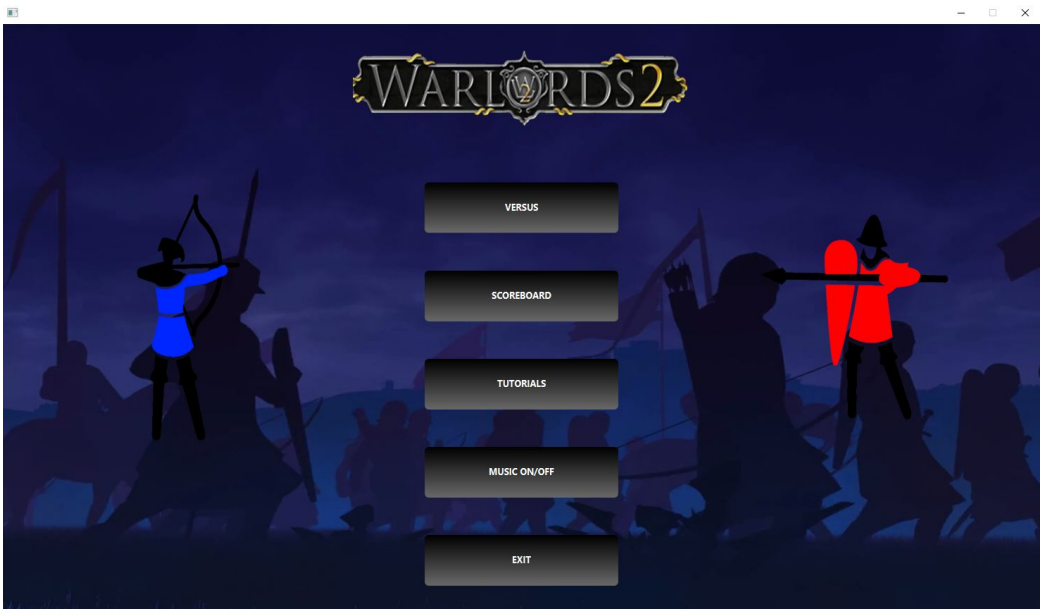


Figura A.1: Menù principale.

Nella sezione *Tutorials*, si potranno vedere i comandi da tastiera assegnati, simmetricamente, ad ognuno dei due giocatori. Inoltre si avrà un elenco più o meno dettagliato delle regole e delle meccaniche dello scontro.

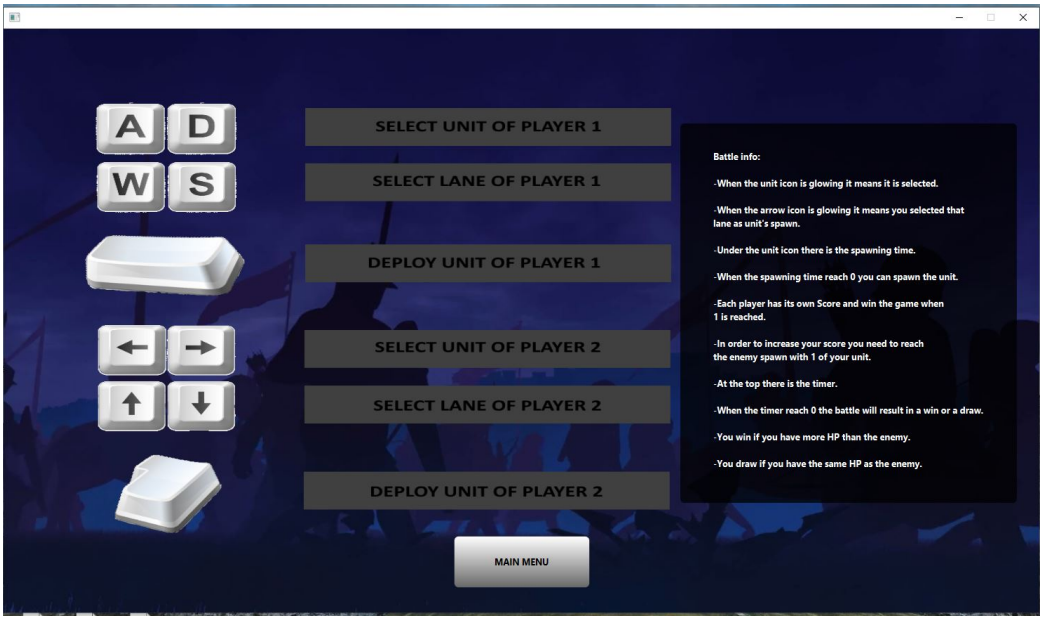


Figura A.2: Schermata di Tutorial.

Dopo aver consultato il "Tutorial" è possibile passare alla sezione *Versus* che permetterà di accedere alla schermata per il settaggio delle preferenze per l'avvio della partita e l'inserimento dei nomi dei giocatori, Figura A.3.

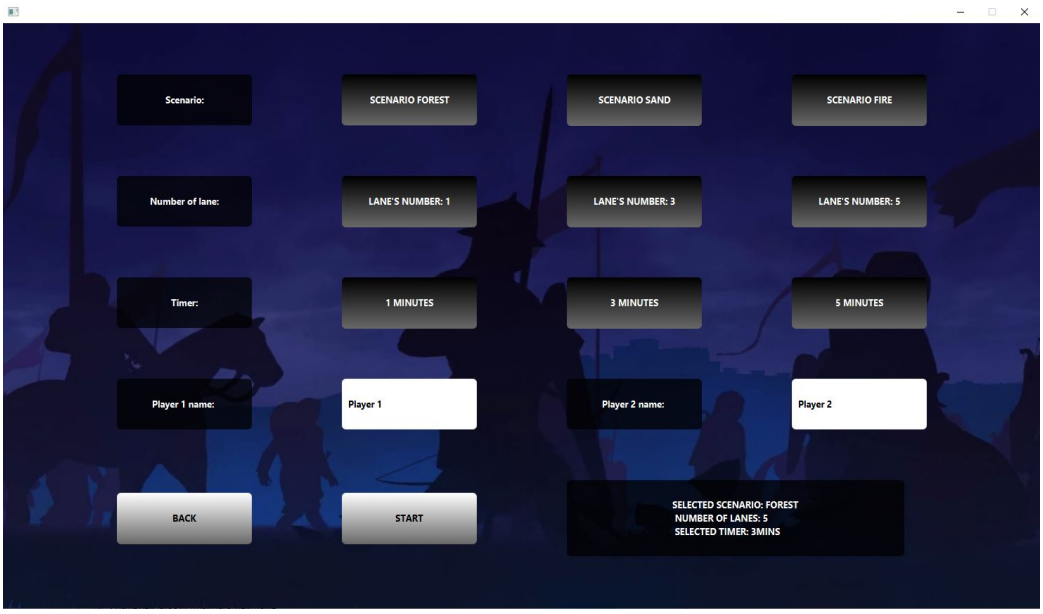


Figura A.3: Settaggio gioco, in basso a destra è disponibile visualizzare la selezione corrente.

Successivamente ad aver selezionato con un click le varie preferenze, oppure anche senza selezionare nulla (il che provvederà all'uso del settaggio di default), sarà possibile avviare la partita con il pulsante *Start*.

All'avvio della partita ogni giocatore potrà far uso dei suoi comandi per competere con l'avversario schierando le proprie unità, e cercando di fare punto. L'obbiettivo è difendersi e attaccare.

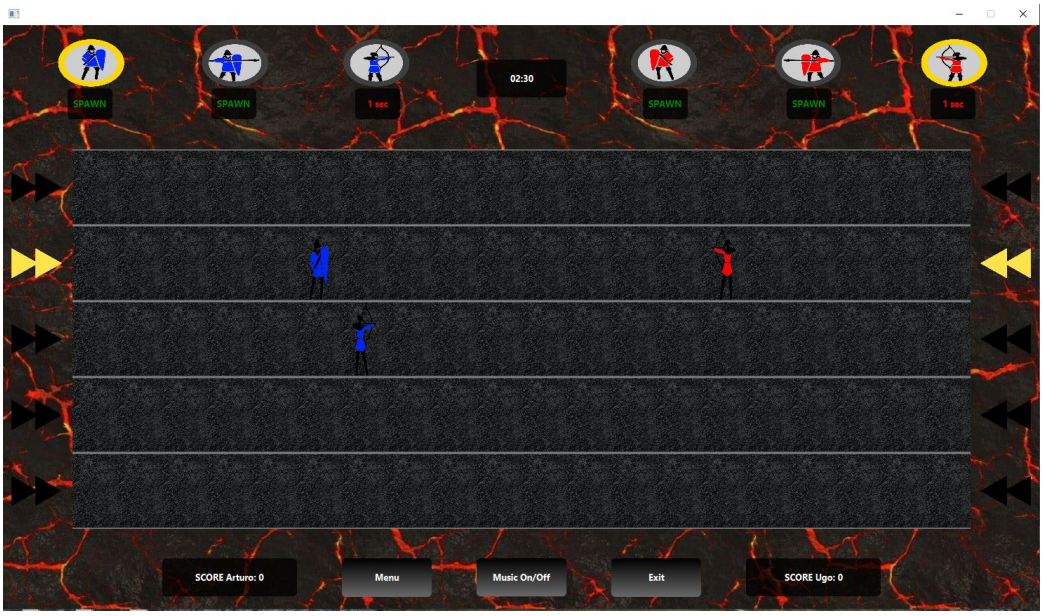


Figura A.4: Interfaccia del gioco durante lo scontro.

Si concluderà la partita quando uno dei due giocatori arriverà a 5 punti oppure allo scadere del timer. La partita finirà in pareggio nel caso in cui al termine dello scontro i giocatori avranno lo stesso punteggio.

Il risultato della partita, che comprende i nomi dei giocatori e i relativi punteggi verra' salvato e sarà consultabile nella sezione *Scoreboard* del menù principale.

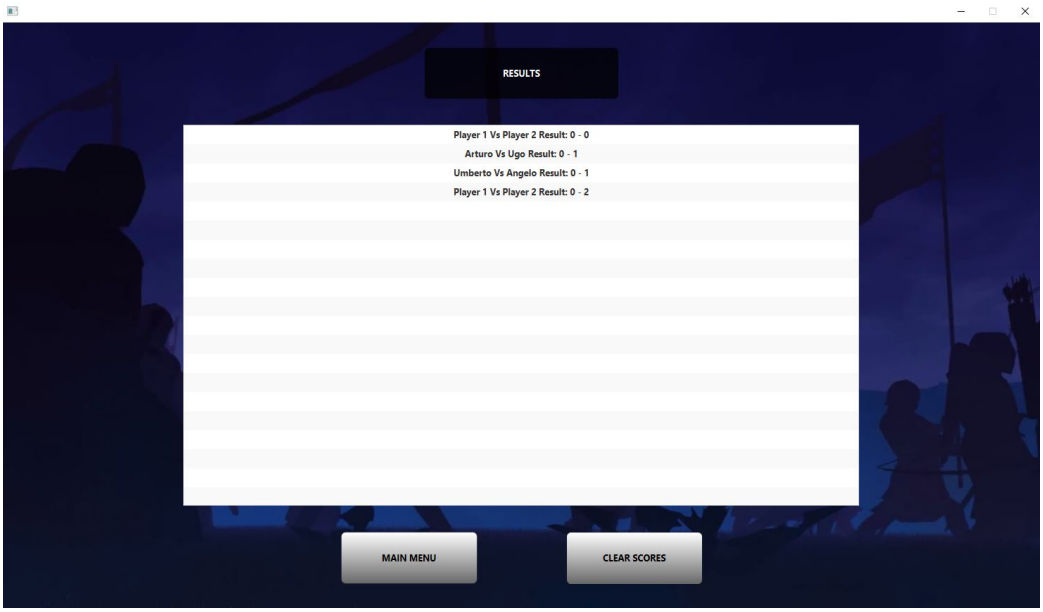


Figura A.5: Tabella con i risultati delle partite.

Appendice B

Esercitazioni di laboratorio

B.1 Mainardi Giosuè Giocondo

- Laboratorio 04: https://virtuale.unibo.it/mod/forum/discuss.php?d=62685#p101504
- Laboratorio 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=62684#p101506
- Laboratorio 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=62579#p100909
- Laboratorio 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=62582#p100911
- Laboratorio 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=63865#p103957
- Laboratorio 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=64639#p104117
- Laboratorio 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=66753#p106596

B.2 Filippo Tartagni

Non presenti in quanto essendo fuori corso, in questo anno non ho seguito il laboratorio.

B.2.1 Andrea Zacconi

A causa di problemi familiari, non mi e' stato possibile seguire alcuna lezione di laboratorio. Arrangiandomi e rimboccandomi le maniche da solo e con l'aiuto dei miei colleghi.