

Relazione “Flower Force”

Juri Gugliemi, Nicolas Amadori, Nicolò Monaldini, Riccardo Mazzi

10 aprile 2023

Indice

1	Analisi	2
1.1	Requisiti	2
1.2	Analisi e modello del dominio	4
2	Design	6
2.1	Architettura	6
2.2	Design dettagliato	8
3	Sviluppo	25
3.1	Testing automatizzato	25
3.2	Metodologia di lavoro	25
3.3	Note di sviluppo	28
4	Commenti finali	34
4.1	Autovalutazione e lavori futuri	34
A	Guida utente	36
B	Esercitazioni di laboratorio	40
B.0.1	riccardo.mazzi@studio.unibo.it	40
B.0.2	nicolo.monaldini@studio.unibo.it	40
B.0.3	juri.guglielmi@studio.unibo.it	41
B.0.4	nicolas.amadori@studio.unibo.it	41

Capitolo 1

Analisi

1.1 Requisiti

Il progetto, commissionato dai professori del corso di Programmazione ad Oggetti della facoltà di Ingegneria e Scienze Informatiche, mira alla verifica del nostro apprendimento della programmazione ad oggetti tramite la costruzione di un software. Abbiamo scelto di realizzare un gioco tower-defense, ispirato al famoso titolo Plant vs Zombies¹, intitolato *Flower Force*.

Requisiti funzionali

- Flower Force permetterà al giocatore, all'interno di una schermata principale, di accedere a diverse funzionalità: ad uno dei livelli della modalità ”avventura”, alla modalità ”sopravvivenza” oppure al negozio.
- In qualsiasi tipo di partita il giocatore dovrà difendere la propria casa da orde di ”zombie” (nemici) utilizzando ”piante” (armi di difesa). Ogni tipo di pianta avrà una capacità unica (come sparare semi, produrre soli, ecc) e il giocatore dovrà difendersi dagli attacchi di molteplici zombie (alcuni più resistenti, altri più veloci, ecc) selezionando e posizionando le piante che reputerà migliori. Cosa da tenere in considerazione per il posizionamento delle piante, è che la pianta smetterà di sparare mentre viene mangiata. Il giocatore sarà sconfitto se anche uno solo degli zombie raggiungerà la sua casa (i.e. lo zombie sarà riuscito a percorrere l'intero campo da gioco). Flower Force incrementerà il punteggio della partita per ogni zombie eliminato, di un valore che dipenderà dalla sua difficoltà.

¹<https://www.ea.com/it-it/games/plants-vs-zombies/plants-vs-zombies>

- I livelli della modalità avventura sono ”incrementali”, cioè (partendo dal primo) ogni volta che ne verrà superato uno si sbloccherà quello successivo con nuovi tipi di piante e zombie. Inoltre, una volta superato ognuno di questi livelli, Flower Force darà al giocatore una certa quantità di monete (anche questa incrementale). Il livello verrà superato se tutti gli zombie verranno eliminati.
- La modalità sopravvivenza permetterà di giocare ad un livello senza fine, con le piante sbloccate dal giocatore nella storia fino a quel momento (con l’aggiunta delle piante acquistate nel negozio) ma affrontando tutte le tipologie di zombie presenti nel gioco. Per questa modalità verrà salvato il punteggio più alto ottenuto dal giocatore.
- All’interno del negozio Flower Force permetterà al giocatore di spendere le proprie monete in cambio di nuove piante potenziate da usare nelle partite.
- Durante la partita i soli saranno l’unica risorsa a disposizione per poter posizionare le piante (che ne richiederanno una determinata quantità come costo) e saranno prodotti sia dal gioco che da alcune piante.
- Una volta utilizzata una pianta dal giocatore, Flower Force la renderà momentaneamente non disponibile per un tempo che potrà variare a seconda del tipo di pianta.
- Le piante potranno essere piazzate all’interno del campo da gioco, il quale sarà diviso in celle. Una volta occupata una cella con una pianta, questa non sarà più in grado di accoglierne altre. Se il giocatore vorrà liberare la cella potrà utilizzare la pala (funzionalità sempre disponibile), che renderà la cella nuovamente disponibile senza però restituire i soli spesi per la pianta che la occupava.
- La generazione degli zombie sarà graduale, sia per la frequenza di creazione sia per la difficoltà (nelle fasi iniziali della partita verranno creati solo gli zombie più facili). Inoltre dopo un certo numero di zombie generati, saranno presenti delle orde di zombie. In questo modo i livelli saranno giocabili e superabili.
- Flower Force dovrà avere musica ed effetti sonori.
- Dovranno essere salvati (e successivamente caricati) i progressi di gioco.

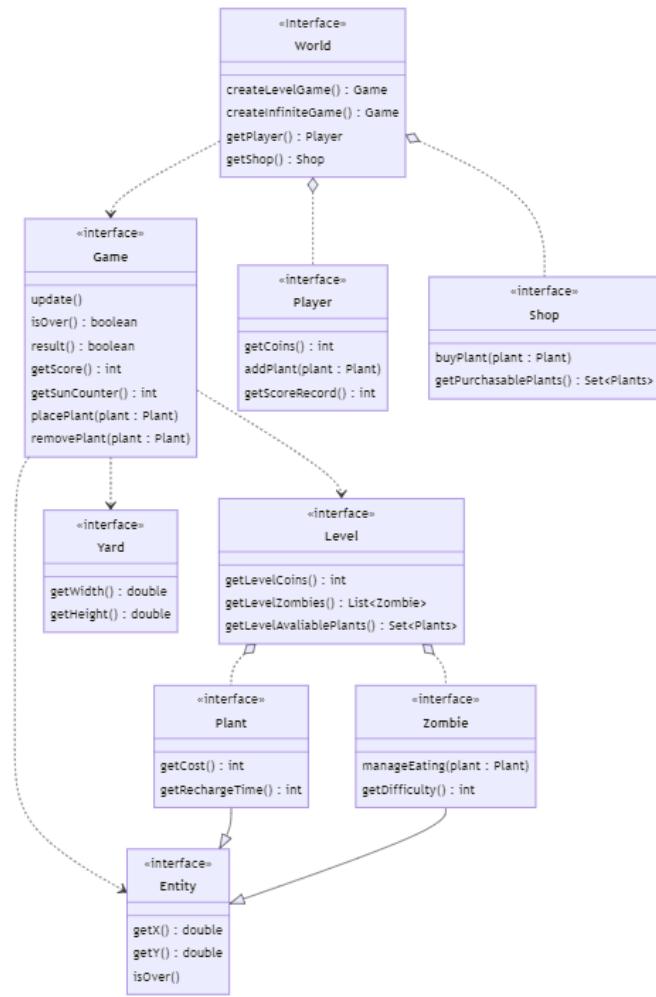
Requisiti non funzionali

- Il salvataggio ed il caricamento dei progressi dovrà avvenire in modo efficiente e con un basso impatto sulla memoria.
- Giocabilità minima garantita, senza necessità di hardware con alte prestazioni.

1.2 Analisi e modello del dominio

Il giocatore si troverà in un mondo di gioco (World) in cui potrà giocare a due tipologie di partita (Game): potrà scegliere tra vari livelli (modalità avventura), superandoli uno ad uno, oppure potrà giocare a partite in modalità sopravvivenza per resistere il più possibile e battere i record di score precedentemente ottenuti. Ogni livello (Level) è unico in quanto conterrà diverse tipologie di zombie (Zombie) e di piante (Plant) in numero variabile. Al progredire dei livelli gli zombie diventeranno incrementalmente sempre più difficili da affrontare, cioè diventeranno più forti e/o più veloci, con diverse caratteristiche. Anche le piante saranno differenziate da alcune caratteristiche, come vita, danno e altre funzionalità; alcune, ad esempio, saranno utilizzate per fare danno agli zombie con l'obiettivo di eliminarli, altre saranno utilizzate per produrre soli al fine di poter posizionare ulteriori piante nel campo di gioco (Yard). Il giocatore (Player) accumulerà monete di gioco man mano che avanza nei livelli con le quali potrà acquistare, nel negozio (Shop), delle piante non presenti all'interno di essi (sbloccando gli slot bonus).

Particolarmente complicata sarà la differenziazione delle funzionalità delle varie piante, in quanto non tutte infliggeranno danno agli zombie, ma potrebbero ad esempio rallentarli o generare soli. Un altro elemento critico che potremmo incontrare è la gestione dell'avanzamento della partita, più in particolare le interazioni tra gli zombie e le piante, il tutto gestendo anche gli input dell'utente. Inoltre sarà particolarmente complicata la generazione delle varie tipologie di zombie in quantità incrementalì aumentando di continuo la difficoltà. Sarà critico anche il salvataggio su disco dei progressi di gioco (ad esempio di score e monete ottenute) e il suo caricamento all'avvio del gioco.



Capitolo 2

Design

2.1 Architettura

Per la realizzazione di Flower Force abbiamo scelto di utilizzare il pattern architettonico MVC (model-view-controller).

FlowerForceApplication, che estende Application di JavaFX, implementa l’interfaccia FlowerForceView, ed è la classe che instanzia il controller. Si occupa di attivare le varie scene (che implementano FlowerForceScene), delegando l’interazione con il controller ai controller della view che le gestiscono. Ogni controller di scena si occupa infatti della gestione dell’input, chiamando poi in modo opportuno le funzionalità del controller per mostrare le informazioni delle quali hanno bisogno. Consistono quindi in classi *reattive*. All’interno di FlowerForceApplication verrà avviata, al momento dell’avvio della partita, l’istanza del GameLoop, ottenuta dal controller, per iniziare l’aggiornamento continuo dello stato di gioco e del suo rendering sulla scena della partita. Il gameloop infatti interagisce con il controller della game view attraverso l’interfaccia GameEngine, invocando il metodo render quando avvengono degli update nel model.

ControllerImpl, che implementa l’interfaccia Controller, è l’entry point del controller e si occupa di fare da tramite tra model e view; viene infatti chiamato da FlowerForceApplication e dai controller delle view, e a sua volta chiama i metodi dell’interfaccia World, istanziata all’interno del suo costruttore.

WorldImpl (che implementa World), consiste nell’entry point del model. Esso si occupa di fornire l’accesso allo Shop, al Player e permette l’inizializzazione di una nuova partita (Game).

In caso di un cambio di libreria grafica, model e controller rimarrebbero intoccati; bisognerebbe solamente cambiare il modo con il quale il metodo

run del Gameloop viene chiamato in maniera continuativa.

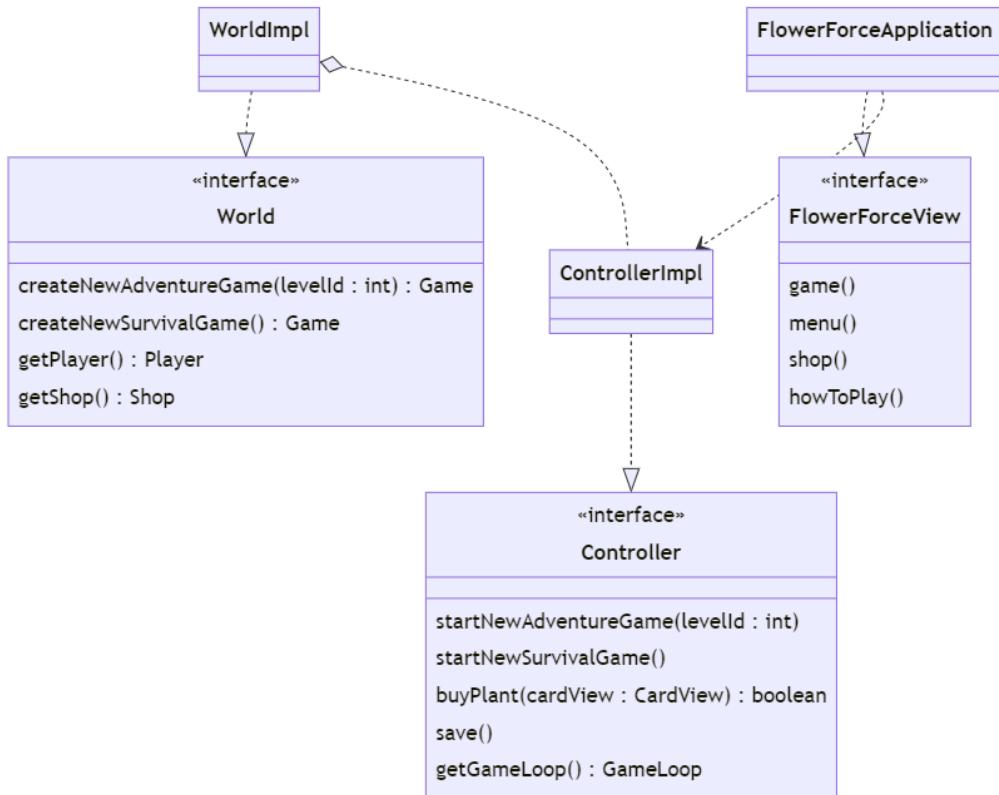


Figura 2.1: Comunicazione tra Model, View, Controller

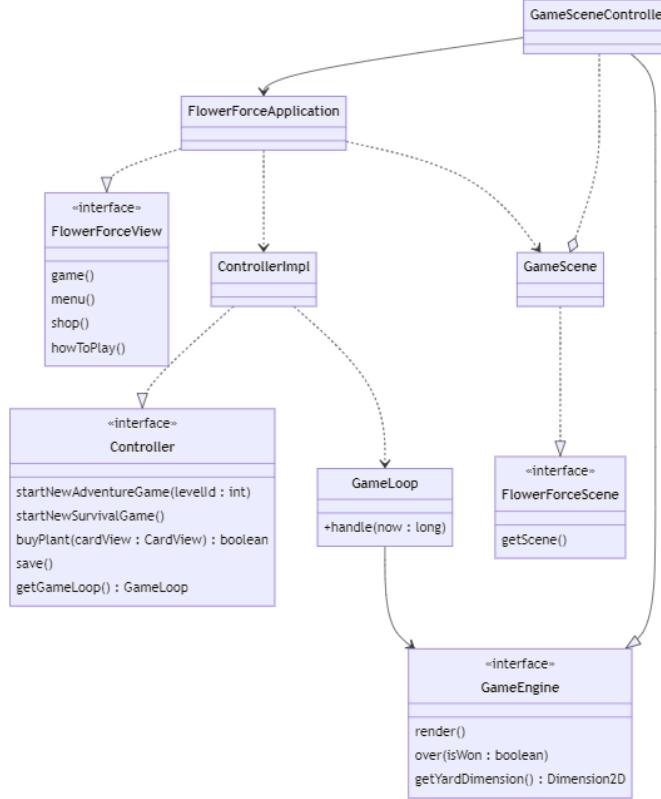


Figura 2.2: Comunicazione tra scene della View e componenti del Controller

2.2 Design dettagliato

Prima di entrare nello specifico descriviamo le parti di design che sono state scelte insieme.

Particolarmente complessa è stata la strutturazione delle entità (Entity) e la loro gestione all'interno del model, per questo abbiamo valutato diverse opzioni. Abbiamo valutato l'uso del pattern Decorator per separare lo stato di un Entity dalle sue capacità, tuttavia abbiamo scelto di scartare questa opzione poiché ci siamo accorti che lo stato delle Entity è legato e influenzato dal loro comportamento, e inoltre ogni entità ha un comportamento specifico non in comune con altre, quindi le varie entità non sarebbero state componibili attraverso l'utilizzo "in catena" di più decoratori. Abbiamo quindi optato per l'ereditarietà: partendo dall'interfaccia più generica `Entity`, avremo delle interfacce che modellano comportamenti specifici (`MovingEntity` e `LivingEntity`) che verranno estese da interfacce che rappresentano piante, zombie e proiettili. Per queste interfacce avremo delle classi di implementa-

zione istanziate attraverso delle factory, per tipologie più specifiche ci saranno invece classi che estendono queste implementazioni.

Riccardo Mazzi

Struttura delle entità Zombie

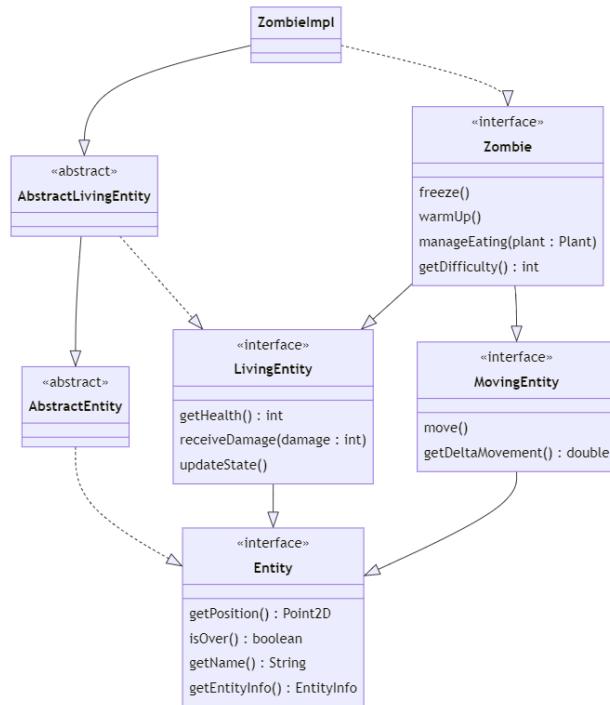


Figura 2.3: Struttura delle entità per definire l'interfaccia Zombie

Problema: Struttura delle classi e delle interfacce delle entità che definiscono Zombie e altre interfacce (Plant e Bullet) minimizzando la ripetizione di codice e rispettando i principi della buona programmazione.

Soluzione: Divisioni delle interfacce (e relative classi astratte, se necessarie) a seconda dello scopo (entità con vita, entità che si muove). Zombie è sia `LivingEntity` che `MovingEntity`.

Creazione degli zombie

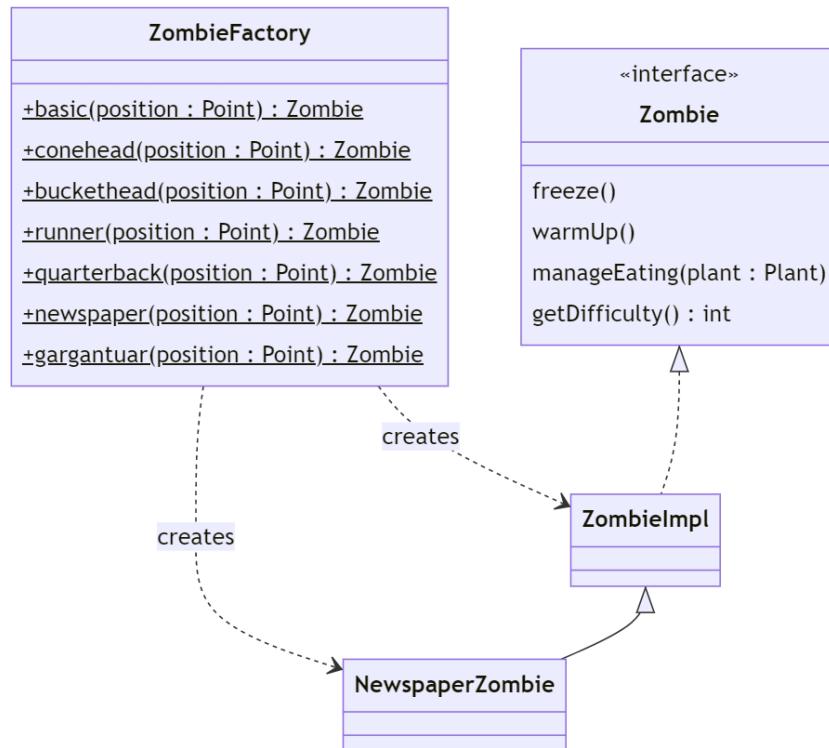


Figura 2.4: Creazione degli Zombie tramite Factory Method (usata in modo statico)

Problema: Una volta definita l’interfaccia **Zombie** è sorto il problema della creazione dei diversi tipi di zombie, i quali condividono l’implementazione dei metodi di interfaccia (a parte lo **NewspaperZombie**) ma si differenziano per valori associati ai campi di **ZombielImpl**.

Soluzione: Si è optato per il pattern Factory Method che risolve in modo ordinato ed elegante il problema, rispettando il principio DRY. Si è però deciso di rendere la classe **ZombieFactory** ”statica” per renderne più semplice lo sporadico utilizzo e poiché ne esiste solo un’implementazione.

Per la classe **NewspaperZombie** si è deciso di estendere la classe **ZombielImpl** (rispettando il principio OCP) per poter usare i metodi già correttamente implementati in **ZombielImpl**, modificandone altri (il sistema permette quindi una buona estendibilità).

Gestione delle informazioni delle entità

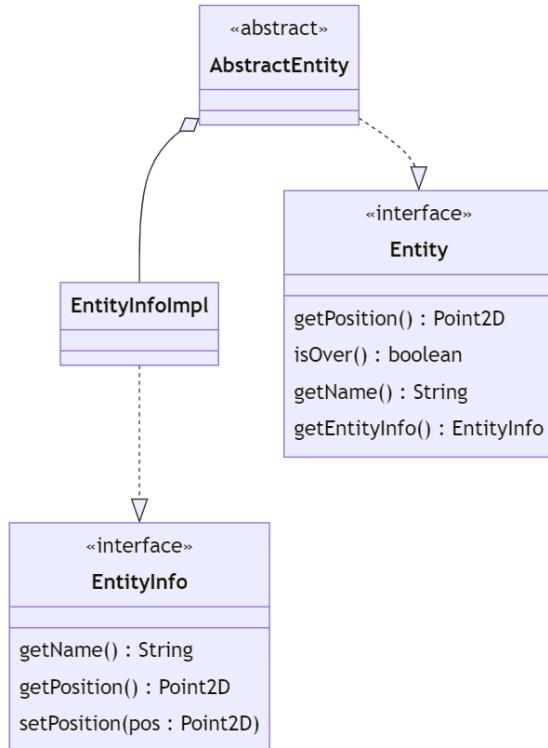


Figura 2.5: Gestione delle informazioni riguardo un'entità tramite EntityInfo

Problema: Passare in modo sicuro le informazioni di un'entità al controller (solo quelle strettamente necessarie, senza passare l'intera entità) e distinguere queste informazioni per poter risalire all'entità che le ha generate (in particolare utilizzando mappe).

Soluzione: Inizialmente il nome di un'entità (necessario per poter caricare le risorse associate ad essa) era ottenuto dal controller attraverso enum di tipi di piante, zombie e bullet, mentre la posizione attraverso l'entità stessa.

Si è deciso di sostituire l'enum con l'utilizzo di mappe (una soluzione più elegante ed estendibile) usando come rappresentanti delle entità (cioè come oggetti di passaggio tra model e controller) le classi `EntityInfo` e, per i tipi di piante, `PlantInfo`. In questo modo si sono ridotte le dipendenze tra model e controller e le informazioni sono state rese distinguibili dagli oggetti "info" che le wrappano in un oggetto unico e distinto.

Gestione scene

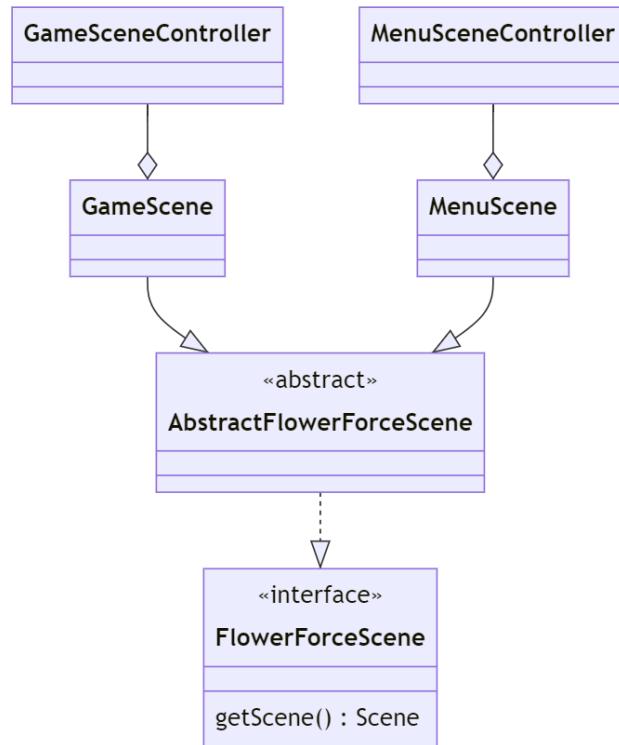


Figura 2.6: Gestione delle scene della view

Problema: Gestire correttamente le scene e il loro caricamento

Soluzione: Separazione da **FlowerForceApplication** di: creazione della scena, creazione del javafx controller e caricamento del file fxml. Utilizzo di interfaccia e classe astratta per evitare ripetizione di codice tra le varie scene (nello schema UML sono presenti solo le scene di game e menu per semplicità).

Nicolò Monaldini

Strutturazione delle entità Plant e Bullet

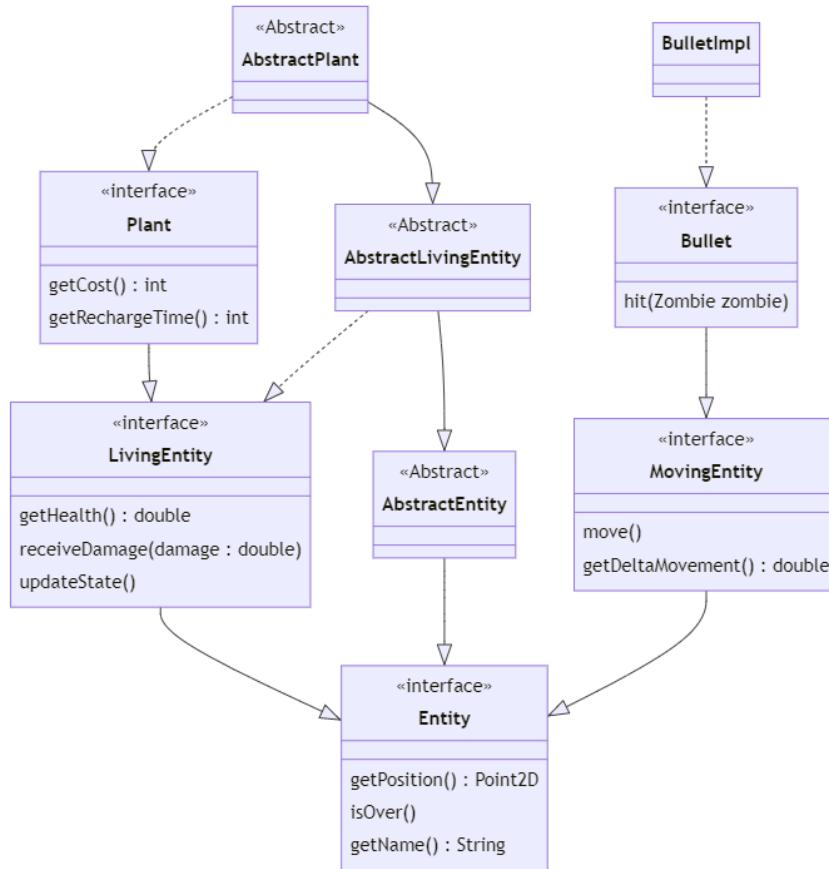


Figura 2.7: Struttura di alcuni elementi principali della parte entità

Problema: È necessario strutturare le entità in modo da evitare ripetizioni di codice

Soluzione: Al fine di rispettare il principio DRY e facilitare eventuali modifiche future alle entità si è scelto di fare largo uso di classi astratte, dove necessario e funzionale. In particolare, tutte le piante estendono la classe astratta `AbstractPlant`.

Creazione delle piante

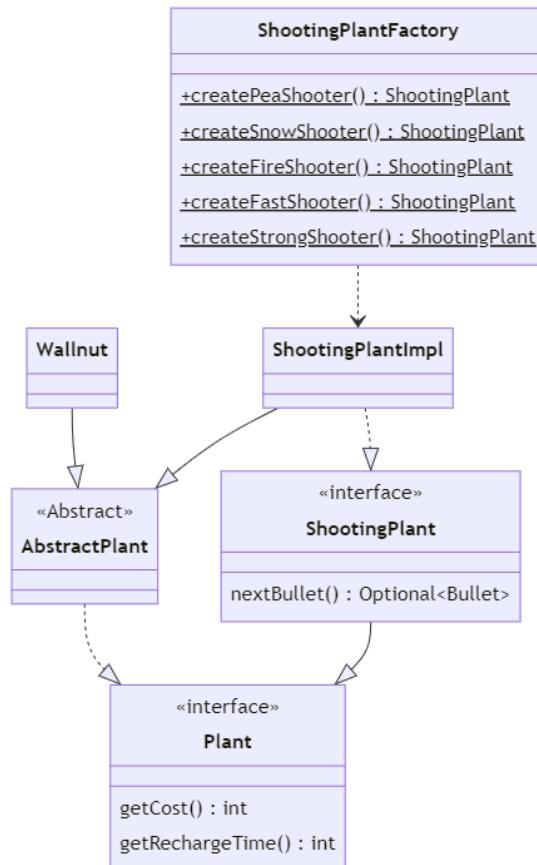


Figura 2.8: Logica di creazione di alcune Plant

Problema: Bisogna gestire la creazione delle varie tipologie di piante, possibilmente applicando i pattern studiati dove necessario.

Soluzione: Per gestire la creazione delle piante in modo efficace, ho scelto di utilizzare il pattern Factory, nel caso di implementazioni generali che possano essere usate per la creazione di più tipologie (ad esempio SunflowerImpl e ShootingPlantImpl). Invece, per la generazione di altre tipologie di piante, come ad esempio Wallnut che richiede l'override di un metodo, sono state create classi apposite.

Creazione dei rispettivi Bullet da parte di ogni pianta che spara

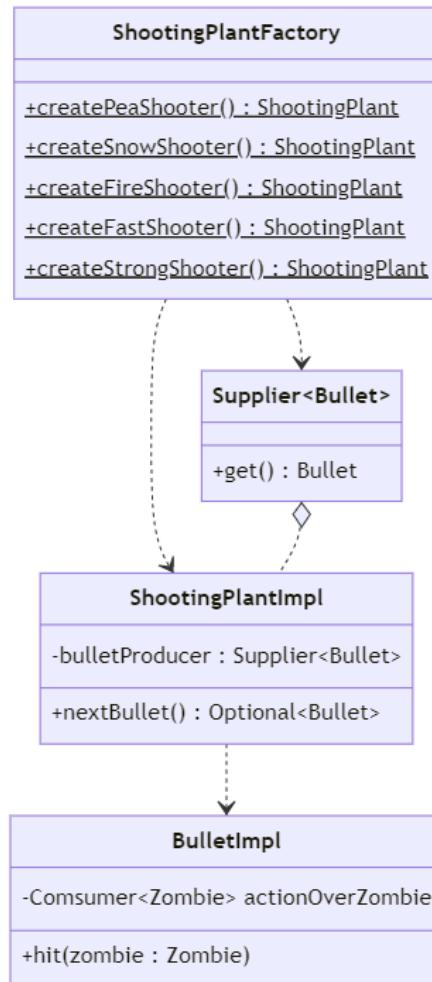


Figura 2.9: Rappresentazione della logica di creazione dei Bullet

Problema: Ogni pianta che spara proiettili (ShootingPlant) li spara di una certa tipologia, è quindi necessario che ogni pianta crei i proiettili (Bullet) di un determinato tipo, nonostante questi siano della stessa classe BulletImpl e non abbiano ognuno una classe specifica.

Soluzione: Inizialmente avevo optato per fare una classe specifica per ogni tipologia di Bullet (in modo da potere specificare l'azione da compiere sullo zombie colpito) e mantenere all'interno di ogni ShootingPlant un campo con la classe del Bullet da sparare. In questo modo ogni qualvolta che doveva

essere creato un nuovo Bullet veniva fatto tramite reflection. Poiché in questo caso l'utilizzo della reflection poteva essere evitato ho poi preferito creare un'unica implementazione di Bullet che potesse prendere nel costruttore un Consumer<Zombie>, in modo da poter differenziare l'azione compiuta da ogni tipo sullo zombie colpito. All'interno di ogni ShootingPlant ho poi inserito un Supplier<Bullet>, inizializzato nella factory, che verrà utilizzato per fornire il Bullet specifico all'occorrenza.

Aggiornamento dello stato di ogni entità

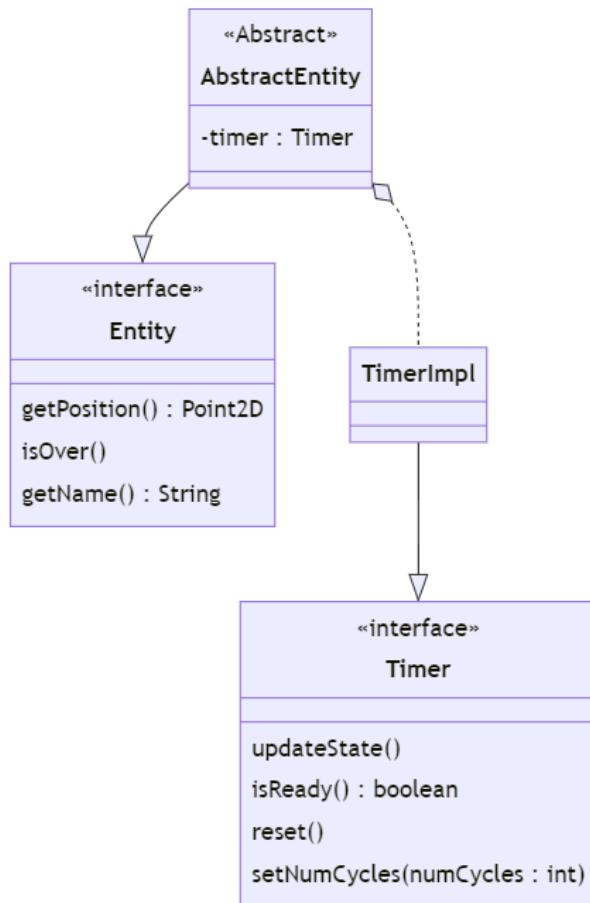


Figura 2.10: Utilizzo di oggetti Timer da parte di entità

Problema: Ogni entità ha bisogno di svolgere delle operazioni o restituire degli oggetti su richiesta ad intervalli regolari.

Soluzione: Per rispettare il Single Responsibility Principle, ogni entità delega il tracciamento del tempo (misurato in cicli del game loop) ad un oggetto esterno che implementa l'interfaccia Timer, al quale viene specificato ogni quanti cicli del game loop deve essere compiuta un'azione. Esso viene aggiornato ad ogni ciclo del game loop e viene interrogato per sapere se è il momento di compiere una determinata azione. Poiché ogni entità ha bisogno di questo strumento, seppur per gestire operazioni diverse, vi è un Timer in ogni Entity, che poi sarà utilizzato per scopi diversi.

Rendere la GUI scalabile

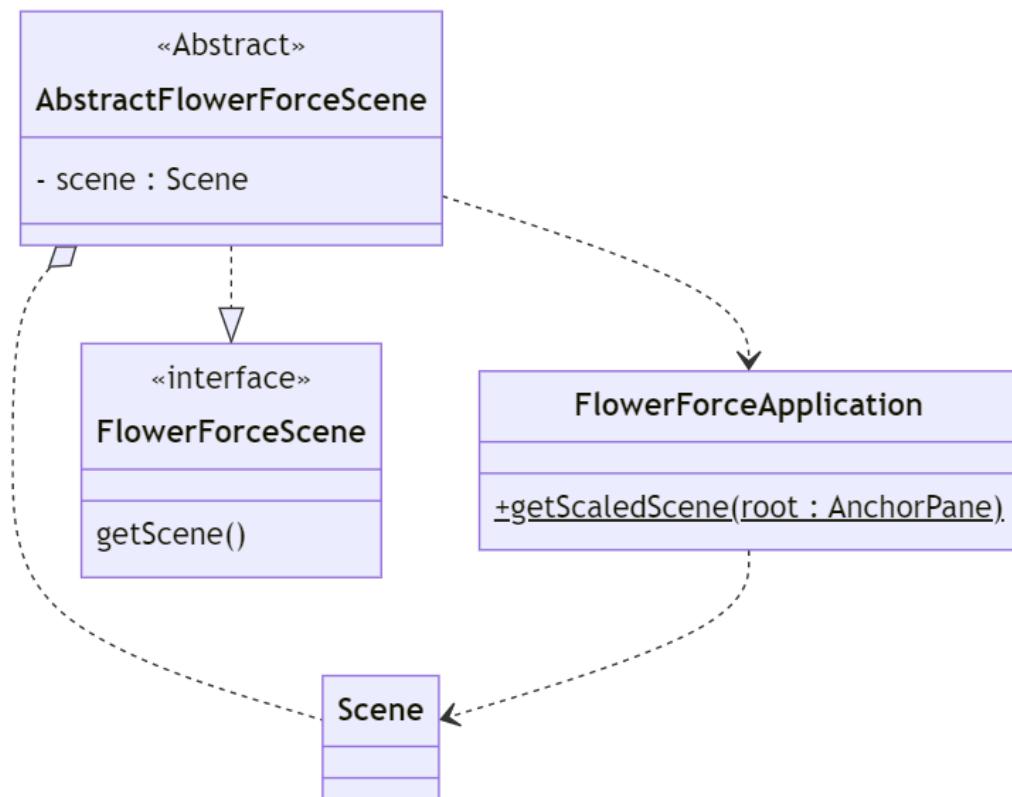


Figura 2.11: Logica di produzione delle scene correttamente scalate

Problema: È necessario rendere la GUI scalabile, ovvero far sì che si adatti a risoluzioni diverse e che, quindi, non abbia dimensione fissa.

Soluzione: Per fare ciò è stato creato un metodo in FlowerForceApplication che, prendendo come parametro un AnchorPane, ovvero il root element di ogni scena, restituisce la scena correttamente scalata attraverso l'uso di un oggetto Scale. Questo metodo viene chiamato da ogni FlowerForceScene (in particolare all'interno del costruttore di AbstractFlowerForceScene) per ottenere la scena scalata da mantenere come campo.

Juri Guglielmi

Struttura delle entità del Game

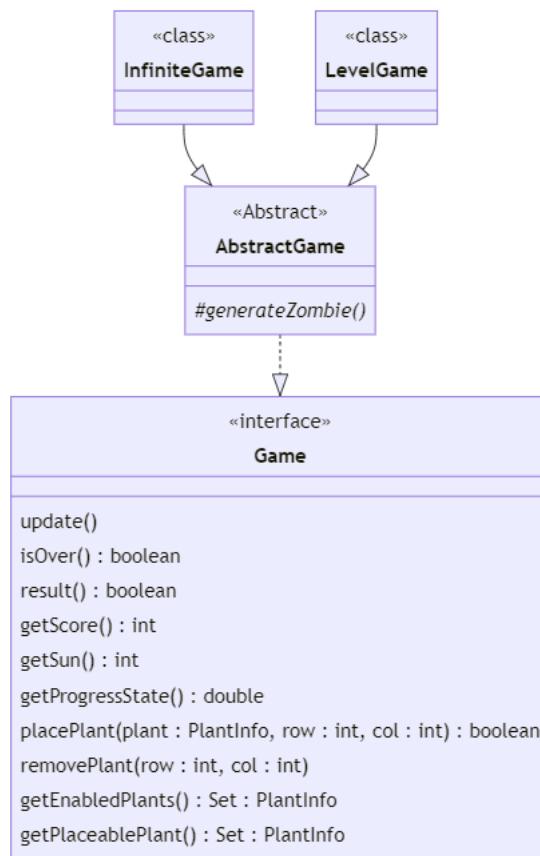


Figura 2.12: Struttura delle entità per definire l'interfaccia Game

Problema: Data la presenza di 2 modalità di gioco differenti ma che allo stesso tempo devono eseguire gli stessi compiti, è necessario strutturare correttamente le entità in modo da evitare ripetizioni di codice.

Soluzione: Per risolvere questo problema ho optato per una soluzione che utilizzasse Template Method. L’interfaccia iniziale è Game dalla quale si ottiene poi la classe astratta AbstractGame che contiene tutto il codice per la gestione di una partita. Mentre la gestione della generazione degli zombie, viene delegata a generateZombie() che verrà implementata dalle classi che la estendono. Nell’AbstractGame, vengono implementati tutti i metodi che sono in comune alle 2 classi(InfinteGame, LevelGame) che poi andranno ad estenderla. Infine le ultime 2 andranno ad implementare/sovrascrivere quei metodi che le rendono differenti.

Struttura delle entità ZombieGeneration

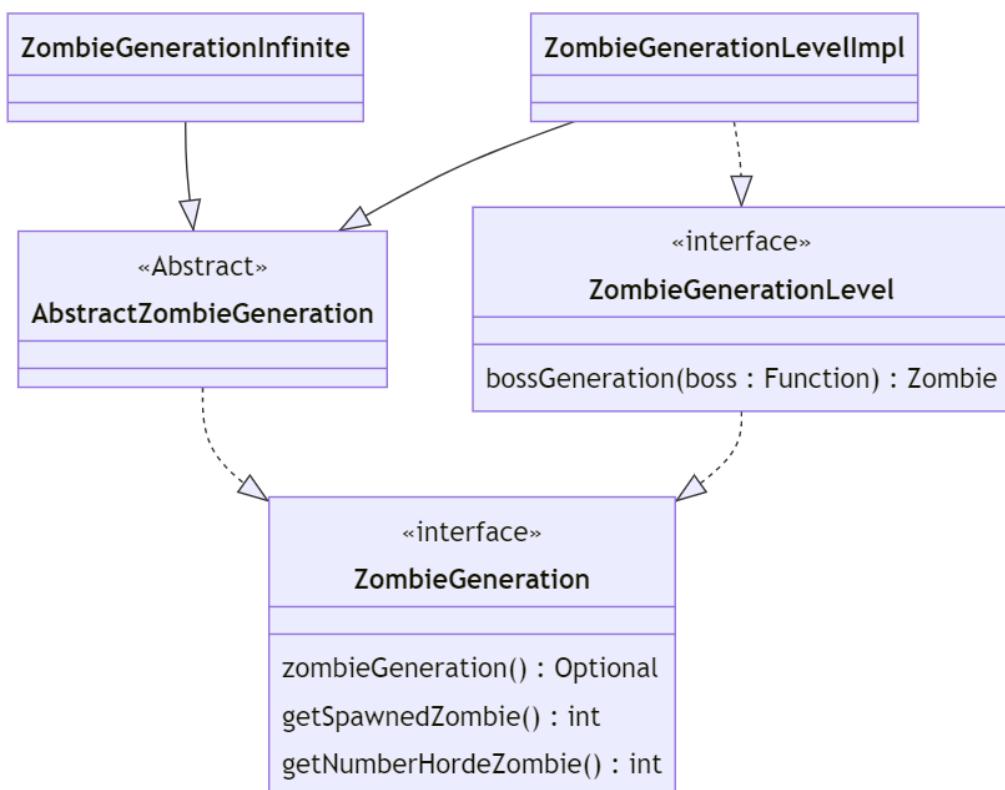


Figura 2.13: Struttura delle entità per definire ZombieGeneration

Problema: È necessario strutturare le entità per la generazione degli zombi in modo da evitare ripetizioni di codice.

Soluzione: Data la presenza di metodi in comune, ho optato per creare una classe astratta(AbstractZombieGeneration) che verrà estesa da 2 classi(ZombieGenerationInfinite, ZombieGenerationLevelImpl), in modo che entrambe possano utilizzare i metodi già implementati di AbstractZombieGeneration e di poterli sovrascrivere o implementarne altri. Per la classe ZombieGenerationLevelImpl oltre ad estendere la classe AbstractZombieGeneration, implementa anche l'interfaccia ZombieGenerationLevel poichè deve implementare il metodo per la creazione del boss finale se presente.

Struttura per la creazione degli zombie

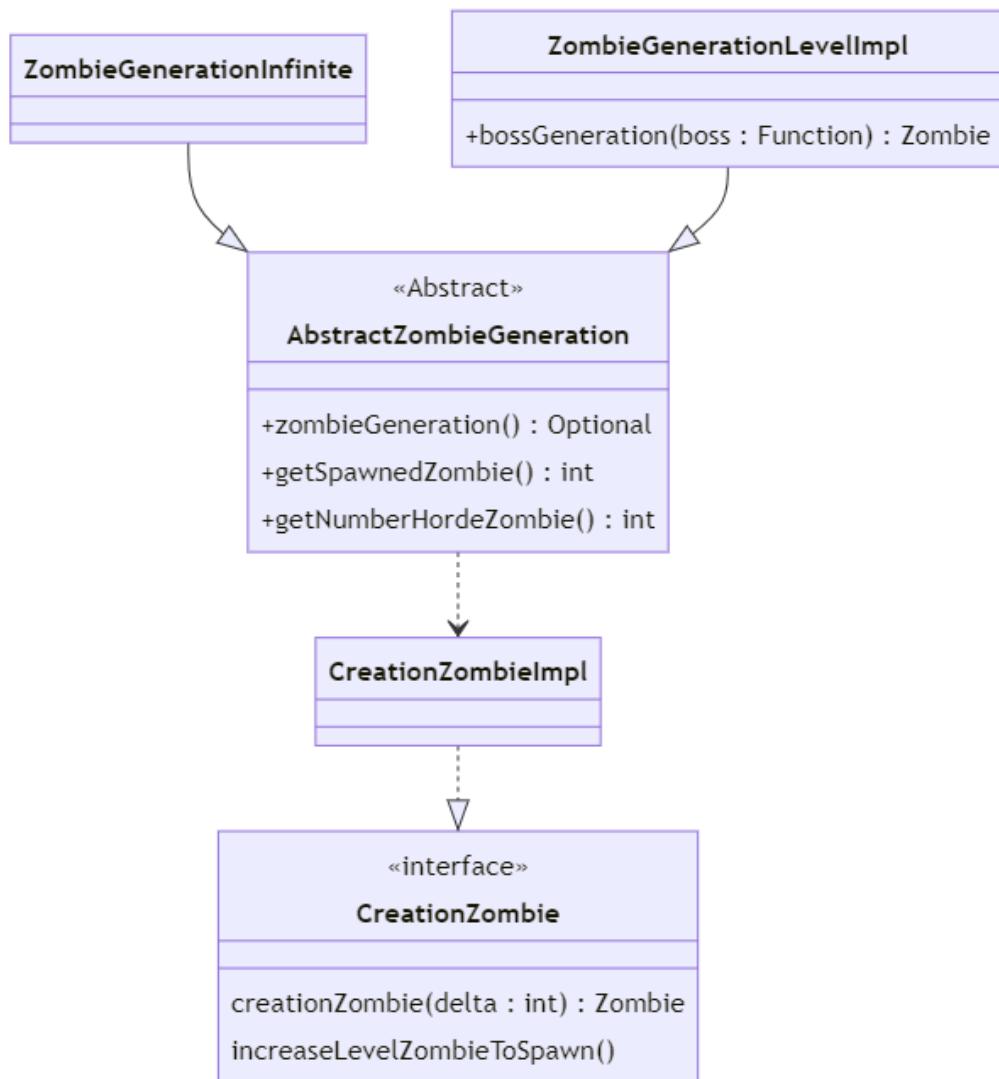


Figura 2.14: Struttura delle entità per definire creationZombie

Problema: È necessario strutturare le entità in modo da evitare che il codice per la creazione degli zombie sia ripetuto.

Soluzione: Ho modellato la gestione della creazione degli zombie tramite **CreationZombie** e **CreationZombieImpl**. Questa classe mi permette di generare degli zombie in modo casuale e che non superino un certo livello di diffi-

coltà. CreationZombie verrà utilizzata da AbstractZombieGeneration quando sarà ora di generare uno zombie. Delegando la creazione di uno zombie ad una classe, rendo il codice di AbstractZombieGeneration più pulito.

Nicolas Amadori

Collegamento tra Controller e GameLoop

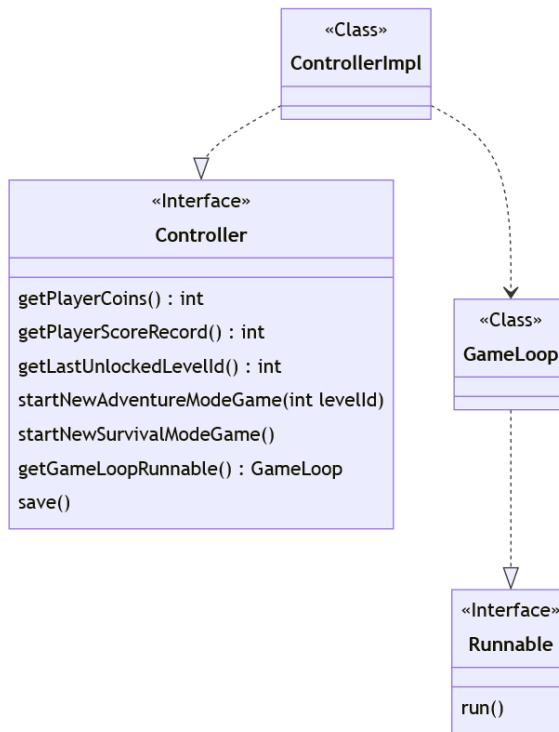


Figura 2.15: Relazioni tra Controller e GameLoop

Problema: È necessario definire le relazioni tra Controller e GameLoop

Soluzione: È stato utilizzato il pattern strategy per implementare il Controller. Il controller al momento dell'avvio di una partita instanzierà il GameLoop (che è un implementazione di Runnable) per poi restituirlo alla view. In questo modo il controller è completamente staccato dalla view, e non deve essere modificato nel caso in cui si cambiasse la libreria statica.

Collegamento tra Shop e Player

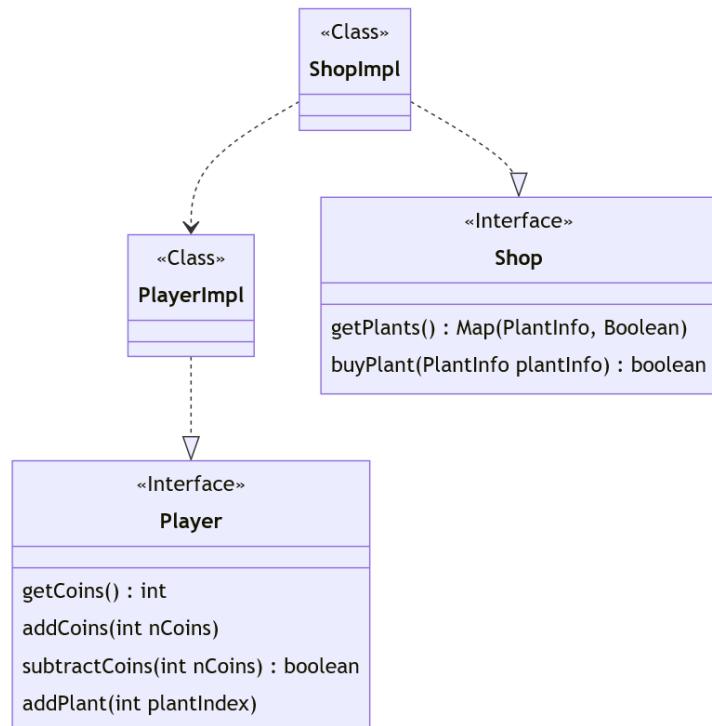


Figura 2.16: Relazioni Shop e Player

Problema: È necessario definire le relazioni tra lo Shop e il Player per effettuare gli acquisti di piante e poterle salvare

Soluzione: Sia Shop che Player vengono implementati da delle classi specifiche (utilizzando il pattern strategy), ma troviamo nel costruttore dello `ShopImpl` un'istanza di `Player`, che è l'istanza sulla quale verranno effettuate le operazioni di acquisto delle piante.

Classi di utilità

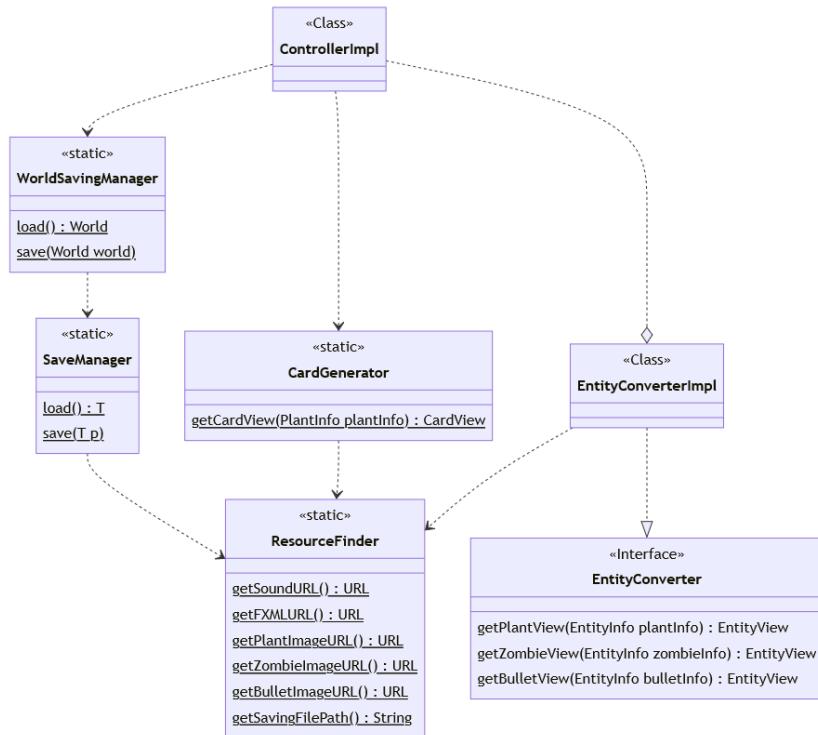


Figura 2.17: Classi di utilità

Problema: Sono necessarie delle classi che implementano delle funzionalità che possono servire in vari punti del progetto.

Soluzione: Sono state implementate varie classi d'utilità per avere sempre a disposizione certe operazioni. Sono state preferite le classi statiche rispetto ai singleton in quanto non abbiamo bisogno di salvare uno stato per avere queste funzionalità, e i metodi lavorano unicamente con ciò che gli è stato passato come parametri.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Abbiamo deciso di testare alcuni dei componenti principali del model, in particolare alcune Entity, il funzionamento di alcune loro componenti (come Timer ed EntityInfo), il Game e il funzionamento della logica di partita. Vengono anche testati le funzionalità del Player e dello Shop. Inoltre, fuori dal model, viene testato il salvataggio dei dati su disco, e la ricerca delle risorse (effettuata tramite una classe apposita) . I test sono automatizzati e sono basati sulla suite Junit.

3.2 Metodologia di lavoro

Per quanto riguarda l'uso del DVCS, abbiamo scelto la versione semplificata del *Git Flow* spiegata nelle lezioni di laboratorio.

Riccardo Mazzi

Mi sono occupato in autonomia di:

- Implementazione della logica degli zombie
- Implementazione delle "info" delle entità (EntityInfo e PlantInfo)
- Implementazione della classe di utilità del campo da gioco (YardInfo)
- Implementazione della view della partita, in particolare con le classi GameScene, GameSceneController e con l'interfaccia GameEngine.

- Gestione delle risorse della view della partita, come il file Game.fxml e le immagini di piante, zombie e proiettili.
- Realizzazione dei metodi necessari allo shop all'interno di Controller e ControllerImpl (*buyPlant* e *getPurchasablePlants*)

Mi sono occupato in collaborazione con il resto del gruppo di:

- Gestione della struttura delle entità (interfacce e classi astratte)
- Gestione della struttura delle scene all'interno della view (interfaccia e classe astratta)
- Caricamento delle scene in FlowerForceApplication
- Gestione dei suoni con la classe SoundManager
- Gestione delle view delle entità (interfaccia EntityView)

Nicolò Monaldini

Mi sono occupato in autonomia di:

- Implementazione della logica di piante e proiettili
- Implementazione del menu di gioco
- Implementazione dei meccanismi per rendere la GUI scalabile
- Implementazione del Timer usato dalle entità per svolgere azioni ad intervalli regolari

Mi sono invece occupato in collaborazione con il resto del gruppo di:

- Gestione delle entità e della loro struttura
- Gestione dei cambi di scena a livello di View
- Gestione generale delle scene

Juri Guglielmi

Mi sono occupato in autonomia di:

- Implementazione della logica della partita, generazione zombie e creazione zombie.
- Implementazione della view dello Shop, in particolare delle classi ShopScene e ShopSceneController.
- Gestione delle risorse della view dello shop, come il file Shop.fxml e le relative immagini.
- Implementazione della logica delle entità della View(EntityView e CardView).
- Implementazione della classe statica per i livelli (LevelInfo).

Mi sono invece occupato in collaborazione con il resto del gruppo di:

- Gestione delle entità e della loro struttura
- Gestione delle entità della View (EntityView e CardView)
- Gestione del campo di gioco

Nicolas Amadori

Mi sono occupato in autonomia di:

- Implementazione del controller che si occupa di mettere in collegamento view e model (*Controller* e *ControllerImpl*). Per questo è servita la collaborazione con i miei colleghi che si sono occupati dello sviluppo del World e della View essendo i 3 punti collegati strettamente tra di loro.
- Implementazione e avvio del GameLoop di gioco che si occupa di effettuare l'aggiornamento del model e il refresh della view (*Player*).
- Implementazione della classe che modella il player nel model (*Player* e *PlayerImpl*). Questa classe si è dovuta adattare per il salvataggio delle piante acquistate tramite il negozio.
- Implementazione della classe che modella il negozio nel model, e che interagisce con il player (*Shop* e *ShopImpl*).

- Implementazione della classe di utilità che si occupa della serializzazione in formato json su file degli oggetti (*SaveManager*).
- Implementazione della classe di utilità che si occupa della lettura o della scrittura dei salvataggi di gioco utilizzando sfruttando la classe *SaveManager* (*WorldSavingManager*).
- Implementazione della classe di utilità che si occupa di ottenere i riferimenti corretti alle risorse di gioco o ai file di salvataggio (*ResourceFinder*). Questa classe ha subito molteplici variazioni in quanto, in fase di testing, è sorto il problema per il quale con il jar bisogna attuare una gestione differente delle risorse, con i file di salvataggio salvati esternamente al jar, e le risorse interne ottenute con il metodo di metodo `getResource()`.
- Implementazione della classe di utilità che si occupa della riproduzione di effetti sonori e musica (*SoundManager*).
- Implementazione della classe che si occupa di convertire le informazioni delle entità ricevute dal model in entità disegnabili e gestibili dalla view (*EntityConverter* e *EntityConverterImpl*).
- Implementazione della classe di utilità che si occupa della generazione di oggetti CardView che la view utilizza per mostrare le card delle piante di gioco o le piante nel negozio (*CardGenerator*).
- Implementazione della scena di view contenente la schermata "HowToPlay" e il corrispettivo controller (*HowToPlayScene* e *HowToPlaySceneController*).

Mi sono invece occupato in collaborazione con il resto del gruppo di:

- Decisione di qualche scelta di design della schermata di gioco.
- Avvio dell'applicazione e scelte di organizzazione delle scene.

3.3 Note di sviluppo

Riccardo Mazzi

Uso di lambda expressions

Utilizzate per rendere il codice più pulito, nell'esempio qui riportato usate insieme al foreach delle collections: <https://github.com/NicolasAmadori/>

[Utilizzate anche in sintassi "method reference". Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/98523a76976d7ad990226a01d2c40ef13b39abf5/src/main/java/flowerforce/view/game/GameSceneController.java#L262-L272>](https://github.com/NicolasAmadori/OOP22-flower-force/blob/98523a76976d7ad990226a01d2c40ef13b39abf5/src/main/java/flowerforce/view/game/GameSceneController.java#L262-L272)

Uso di Stream

Utilizzati per rendere il codice più pulito e compatto, soprattutto insieme alle collections. Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/98523a76976d7ad990226a01d2c40ef13b39abf5/src/main/java/flowerforce/view/game/GameSceneController.java#L332-L340>

Uso di Optional

Utilizzato per definire oggetti che potrebbero essere nulli (vuoti). Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/98523a76976d7ad990226a01d2c40ef13b39abf5/src/main/java/flowerforce/view/game/GameSceneController.java#L82>

Uso della libreria JavaFX

Di largo utilizzo nelle classi di view, di seguito un esempio per trasformare correttamente immagine e posizione di un'entità in ImageView: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/98523a76976d7ad990226a01d2c40ef13b39abf5/src/main/java/flowerforce/view/game/GameSceneController.java#L383-L391>

Utilizzo di codice esterno per il corretto utilizzo della libreria JavaFx

- Tutorial per l'utilizzo di javafx, in particolare il caricamento del file fxml: https://www.youtube.com/watch?v=9XJicRt_FaI&t=3007s
- Tutorial per l'utilizzo degli alert in javafx: https://www.youtube.com/watch?v=61_QA_yEEtQ&t=288s

Nicolò Monaldini

Utilizzo di Optional

Utilizzato per evitare l'uso di valori null in più parti nel codice: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/ab7752cfecbdc246ad126bdf9706bc2>

```
src/main/java/flowerforce/model/entities/plants/ShootingPlantImpl.java#L46-L48
```

Utilizzo di lambda expressions

Utilizzato per implementare interfacce funzionali, qui riportato un esempio con Supplier. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/ab7752cfecbdc246ad126bdf9706bc2b98fe6ffb3/src/main/java/flowerforce/model/entities/plants/ShootingPlantFactory.java#L34-L44>

Utilizzo di altri costrutti funzionali

Utilizzo di forEach come alternativa funzionale all'uso di cicli. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/ab7752cfecbdc246ad126bdf9706bc2b98fe6ffb3/src/main/java/flowerforce/model/entities/plants/BaseExplodingPlant.java#L48>

Utilizzo della libreria JavaFX

Utilizzata in molteplici punti, qui un esempio di utilizzo per rendere la GUI scalabile. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/c49ae1dc0344d86a03b08f8e0dfadff275bfd4b0/src/main/java/flowerforce/view/game/FlowerForceApplication.java#L128-L137>

Juri Guglielmi

Uso di lambda expressions

Utilizzate all'interno degli stream e per rendere il codice più pulito: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/5c7af966669838da29f0102458364b/src/main/java/flowerforce/model/game/AbstractGame.java#L181-L182>

Inoltre, utilizzate anche come "method reference". Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/5c7af966669838da29f0102458364b/src/main/java/flowerforce/model/game/AbstractGame.java#L256>

Uso di Stream

Usati largamente nelle classi AbstractGame e in creationZombie. Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blob/5c7af966669838da29f0102458364b92594b313b/src/main/java/flowerforce/model/game/CreationZombieImpl.java#L30-L33>

Utilizzo di altri costrutti funzionali

Più volte utilizzo di forEach come alternativa funzionale all'uso di cicli.

<https://github.com/NicolasAmadori/OOP22-flower-force/blob/5c7af966669838da29f010src/main/java/flowerforce/model/game/AbstractGame.java#L321-L325>

Uso di Optional

Utilizzato per restituire se lo zombie è stato spawnato o meno. Permalink:

<https://github.com/NicolasAmadori/OOP22-flower-force/blame/5c7af966669838da29f010src/main/java/flowerforce/model/game/AbstractZombieGeneration.java#L75-L77>

Uso di Comparator

Quando un proiettile deve colpire uno zombie, utilizzo il Comparator per capire qual'è il primo zombie che gli è davanti. Permalink: <https://github.com/NicolasAmadori/OOP22-flower-force/blame/5c7af966669838da29f0102458364b92594bsrc/main/java/flowerforce/model/game/AbstractGame.java#L245-L255>

Uso della libreria JavaFX

Utilizzata in ShopSceneController, di seguito un esempio per mostrare gli oggetti nello shop: <https://github.com/NicolasAmadori/OOP22-flower-force/blame/5c7af966669838da29f0102458364b92594b313b/src/main/java/flowerforce/view/game/ShopSceneController.java#L106-L114>

Nicolas Amadori

Utilizzo di Generic

Utilizzati nella serializzazione di oggetti, per indicare il tipo di oggetto da serializzare o deserializzare <https://github.com/NicolasAmadori/OOP22-flower-force/blob/8e3983c744858f7748bad33312a2ab0c69f1380b/src/main/java/flowerforce/controller/utilities/SaveManager.java#L18-L81>

Utilizzo di Optional

Utilizzati nel caricamento del file di salvataggio del player, per indicare quando sono sorti errori in lettura (per esempio il file non era presente) <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889src/main/java/flowerforce/controller/utilities/WorldSavingManager.java#L22-L26>

Utilizzo di costrutti funzionali

metodi foreach, utilizzati per ottimizzare l'iterazione degli elementi all'interno di una lista rimuovendo i for classici <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/controller/ControllerImpl.java#L208-L212>

metodi ifPresent, utilizzati per utilizzare i metodi già proposti invece delle normali clausole per le condizioni <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/view/utilities/SoundManager.java#L133>

Utilizzo di lambda expression

Utilizzate per implementare varie interfacce funzionali in ottica di ottimizzazione del codice e del suo riutilizzo. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/controller/ControllerImpl.java#L191-L194>

Ove possibile, sono stati usati come "method reference". <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/controller/ControllerImpl.java#L213>

Utilizzo di interfacce funzionali come parametri

Utilizzate per riutilizzo massimo del codice <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/controller/ControllerImpl.java#L202-L223>

Utilizzo della libreria GSON

Utilizzato per la serializzazione degli oggetti in formato json. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/controller/utilities/SaveManager.java#L46>

Utilizzo della libreria JavaFX

Utilizzata per progettare il controller della scena HowToPlay, e per avviare l'AnimationTimer del gameloop. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/view/game/HowToPlaySceneController.java#L25-L28>

Codice reperito in rete

Codice per l'aggiunta di un eventhandler alle sound clip per effettuarne la chiusura una volta terminate. <https://github.com/NicolasAmadori/OOP22-flower-force/blob/e211e1232ea3666ce07cae16b69889446d736902/src/main/java/flowerforce/view/utilities/SoundManager.java#L140-L144>

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

Riccardo Mazzi

Mi reputo soddisfatto del lavoro all'interno di questo progetto. A mio parere, uno dei punti di debolezza era la divisione iniziale dei compiti su virtuale: ci siamo infatti accorti che alcuni elementi in precedenza pensati come "elementi di gioco" in realtà non erano propriamente tali ed erano quindi più semplici del previsto da implementare. D'altra parte sono stati invece sottovalutati altri elementi che sono risultati più complessi del previsto. Nonostante ciò abbiamo preso coscienza di questa debolezza e abbiamo agito per contrastarla, rivedendo la divisione di alcuni punti non definiti (come la view ad esempio) ed ottimizzando al meglio il lavoro di ognuno. Nella mia parte di progetto mi rammarica aver speso più tempo per l'implementazione della view del game (non avendo visto javafx a lezione quest'ultimo ha rappresentato una grossa difficoltà) piuttosto che per l'implementazione della logica degli zombie nel model (con più tempo sicuramente migliorabile, anche in relazione con le altre entità).

Nonostante tutto mi reputo soddisfatto di come abbiamo gestito la difficoltà (a mio avviso) maggiore: il team working. Tutti e quattro non avevamo molta esperienza con il lavoro di gruppo, ma siamo riusciti ad interfacciarcì in modo efficiente, proponendo idee diverse ma impegnandoci a raggiungere un'unica soluzione corretta, insieme.

Nicolò Monaldini

Nonostante il gioco sia molto migliorabile, mi ritengo abbastanza soddisfatto del lavoro svolto. Questo progetto mi ha insegnato molto sul lavorare in team

e sullo sviluppo di un progetto completamente da zero, in particolare sull'importanza dello svolgere una buona progettazione e sull'approfondimento in autonomia di librerie non viste a lezione. Il codice è sicuramente migliorabile e con una maggiore attenzione all'analisi alcune parti potevano essere strutturate meglio, tuttavia sono contento dell'impegno che c'è stato da parte di tutto il gruppo nello scrivere il più possibile codice pulito e riusabile.

Juri Guglielmi

Mi ritengo soddisfatto del lavoro che ho svolto, sia a livello singolo che a livello di gruppo. Questo progetto penso abbia migliorato le mie abilità a lavorare in team e sul come relazionarsi con gli altri componenti. Ho capito l'importanza della comunicazione e dell'ascolto, perché senza di esse non ci può essere coesione del gruppo. Mi ha fatto capire l'importanza di svolgere una buona progettazione prima di iniziare la parte di programmazione, e che spendere qualche ora in più in progettazione non è una cattiva idea. Infatti, ritengo che alcune parti del programma, con un'analisi più approfondita, potevano essere realizzate in modo più efficiente e inoltre ciò ci avrebbe anche permesso di evitare alcuni errori che ci eravamo portati dietro dalla prima analisi e che poi siamo andati a rimuovere solo in seguito. Per questo progetto penso di aver dato molto, ma ritengo di aver ricevuto molto indietro per via delle nozioni che ho imparato.

Nicolas Amadori

Mi ritengo soddisfatto del mio lavoro, soprattutto perchè ho imparato cose che prima non sapevo e sono riuscito a risolvere i vari problemi riscontrati durante lo sviluppo. Ho capito quanto sia importante il lavoro di squadra e la divisione dei compiti, e che le opinioni contrastanti non fanno altro che contribuire alla ricerca della soluzione migliore. Ho apprezzato molto la fase di progettazione iniziale solo una volta visto quanto complicata possa essere quella d'implementazione. Sento che forse avrei dovuto impostare il gameloop in maniera leggermente diversa, poichè non sono sicuro del fatto che vada crearlo come runnable che poi deve essere invocato in maniera continua, ma tra varie soluzioni che ho testato mi è sembrata la migliore.

Appendice A

Guida utente

Appena aperto il gioco ci si troverà nel menu di gioco, da qui si avrà accesso a diverse funzionalità, vediamole una ad una.

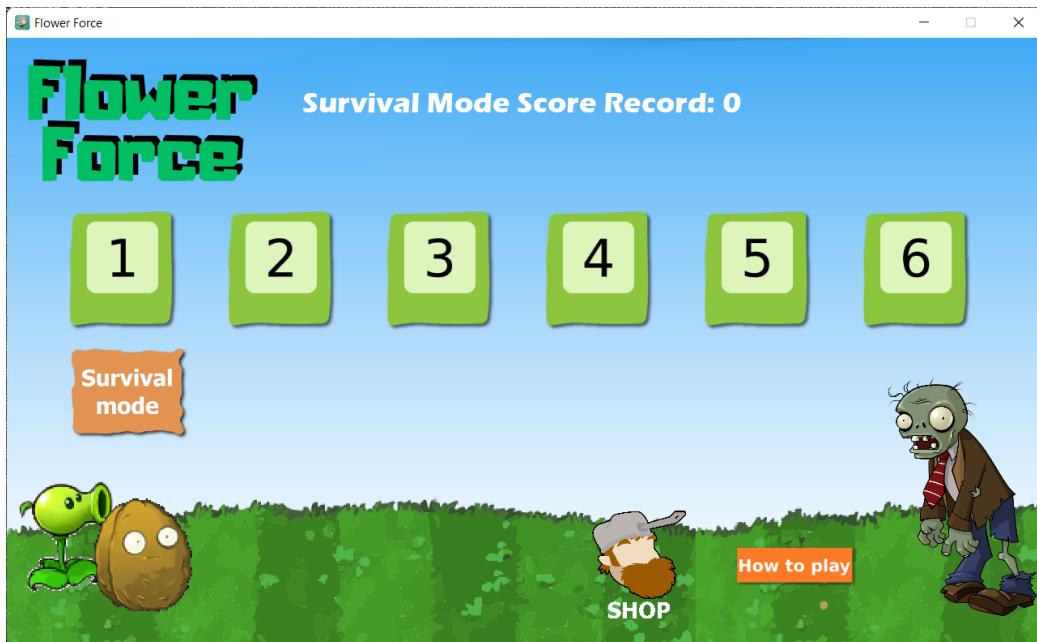


Figura A.1: Schermata menu

Si potrà accedere ai vari livelli a partire dall'uno (nel momento in cui si clicca su un livello non sbloccato compare un avviso che specifica l'ultimo livello sbloccato) e alla modalità sopravvivenza (sempre disponibile).

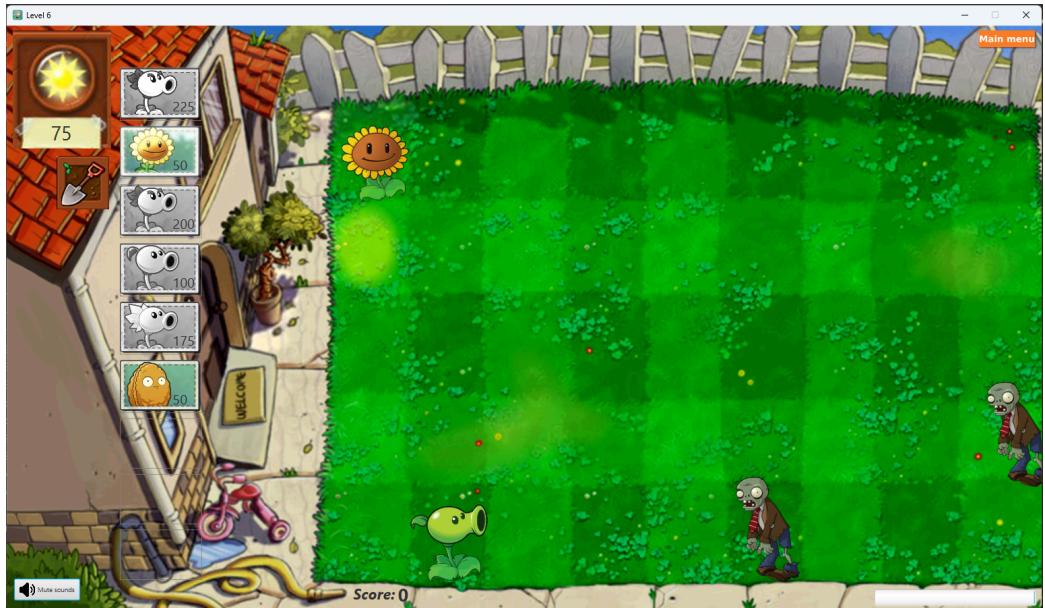


Figura A.2: Schermata in-game

All'interno della partita, cliccando su una delle carte disponibili si selezionerà una pianta e, cliccando in una cella libera del campo da gioco, verrà posizionata. Non sempre ogni pianta sarà selezionabile, in quanto è previsto un tempo di attesa per riutilizzare una pianta appena usata oppure perché non si hanno abbastanza soli in quel momento. Cliccando sull'icona della pala si potrà eliminare dal campo una pianta precedentemente piazzata. Cliccando sull'icona "Main Menu" in alto a destra si ritorna al menu principale, interrompendo la partita. Cliccando invece il bottone "Mute sounds" si potranno mutare o smutare gli effetti sonori.



Figura A.3: Schermata shop

All'interno dello shop, se si disporrà di abbastanza monete, si potrà acquistare la pianta mostrata sullo schermo cliccando sul pulsante "Buy now". Per cambiare la pianta visualizzata si potranno utilizzare le frecce "prev" e "next" mentre per ritornare al menu col pulsante apposito.

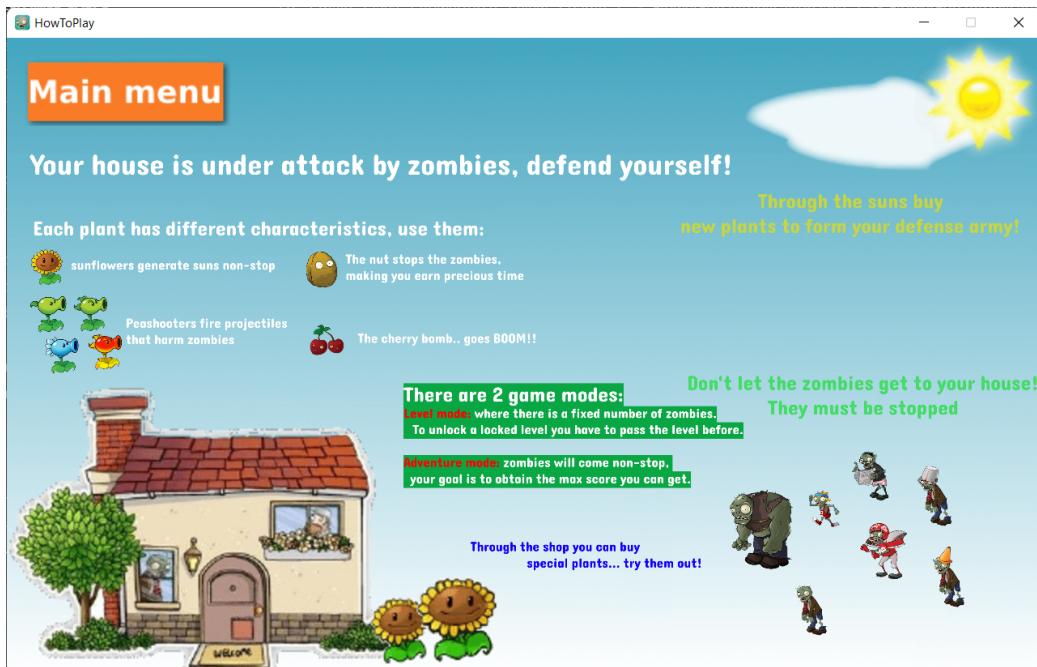


Figura A.4: Schermata How To Play

Infine, è possibile accedere alla schermata "How to play" per vedere una breve descrizione della logica di gioco e un'anteprima delle piante sbloccabili.

Appendice B

Esercitazioni di laboratorio

B.0.1 riccardo.mazzi@studio.unibo.it

- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p170263>

B.0.2 nicolo.monaldini@studio.unibo.it

- Laboratorio 03: <https://virtuale.unibo.it/mod/forum/discuss.php?d=112846#p168117>
- Laboratorio 04: <https://virtuale.unibo.it/mod/forum/discuss.php?d=113869#p169358>
- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p169861>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p169861>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p172540>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=117852#p173760>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p175253>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=119938#p176552>

- Laboratorio 11: <https://virtuale.unibo.it/mod/forum/discuss.php?d=121130#p177402>
- Laboratorio 12: <https://virtuale.unibo.it/mod/forum/discuss.php?d=121885#p178222>

B.0.3 juri.guglielmi@studio.unibo.it

- Laboratorio 03: <https://virtuale.unibo.it/mod/forum/discuss.php?d=112846#p168221>
- Laboratorio 05: <https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p170093>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=115548#p171421>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p173024>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p175688>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=119938#p176528>

B.0.4 nicolas.amadori@studio.unibo.it

- Laboratorio 03: <https://virtuale.unibo.it/mod/forum/discuss.php?d=112846#p168142>
- Laboratorio 06: <https://virtuale.unibo.it/mod/forum/discuss.php?d=115548#p171409>
- Laboratorio 07: <https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p173104>
- Laboratorio 08: <https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p173104>
- Laboratorio 09: <https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p174910>
- Laboratorio 10: <https://virtuale.unibo.it/mod/forum/discuss.php?d=119938#p176470>

- Laboratorio 11: <https://virtuale.unibo.it/mod/forum/discuss.php?d=121130#p177395>

Bibliografia

- Le immagini di sfondo, di alcune immagini delle scene e di alcuni pulsanti sono state reperite dalla seguente repository: <https://github.com/BhavyaC16/Plants-Vs-Zombies>
- Gli sprite delle entità, la musica e i suoni sono stati reperiti dal seguente sito: https://plantsvszombies.fandom.com/wiki/Main_Page