Ingegneria e scienze informatiche – Cesena

Programmazione ad oggetti

Relazione “Zelda Remake”

Di Angela Jelenkovich

Matricola 692234

Anno 2020/2021

Indice

Analisi

Il videogioco in questione è un remake di “Legend of Zelda”. L’obiettivo è sconfiggere i nemici che si incontrano nei vari livelli e arrivare così alla fine. I nemici sono stati programmati per andare contro al personaggio principale e ogni volta che lo raggiungono, gli tolgono una vita. Se l’eroe non ha più vite il gioco finisce. Per potersi difendere ha con sé una pistola.

Requisiti funzionali

- Dal primo livello all’ultimo, l’utente incontrerà difficoltà crescenti e diverse ambientazioni. Il numero di vite verrà resettato ad inizio di ogni livello.
- Al contatto con un nemico verrà sottratta una vita, al termine delle quali ci sarà il Game Over.
- Il gioco cercherà di offrire un’interfaccia il più chiara e fluida possibile.

Requisiti non funzionali

- L’applicazione dovrà avere un uso efficiente delle risorse per garantire fluidità e prestazioni ideali.

Analisi e modello del dominio

Le entità principali di *Zelda Remake* sono:

Hero: entità dinamica principale, che verrà controllata direttamente dal giocatore.

Enemy: entità dinamiche che dovranno cambiare direzione per andare contro il personaggio principale, ostacolandolo.

Static Object: entità statiche che rappresentano l'architettura dei livelli.

Immediatamente bisognerà porre l'attenzione sul controllo delle collisioni tra tutte le principali entità, e poi sulle restanti, come i proiettili. Altro aspetto centrale sarà il movimento degli Enemy che dovranno sempre cambiare direzione insieme all' Hero.

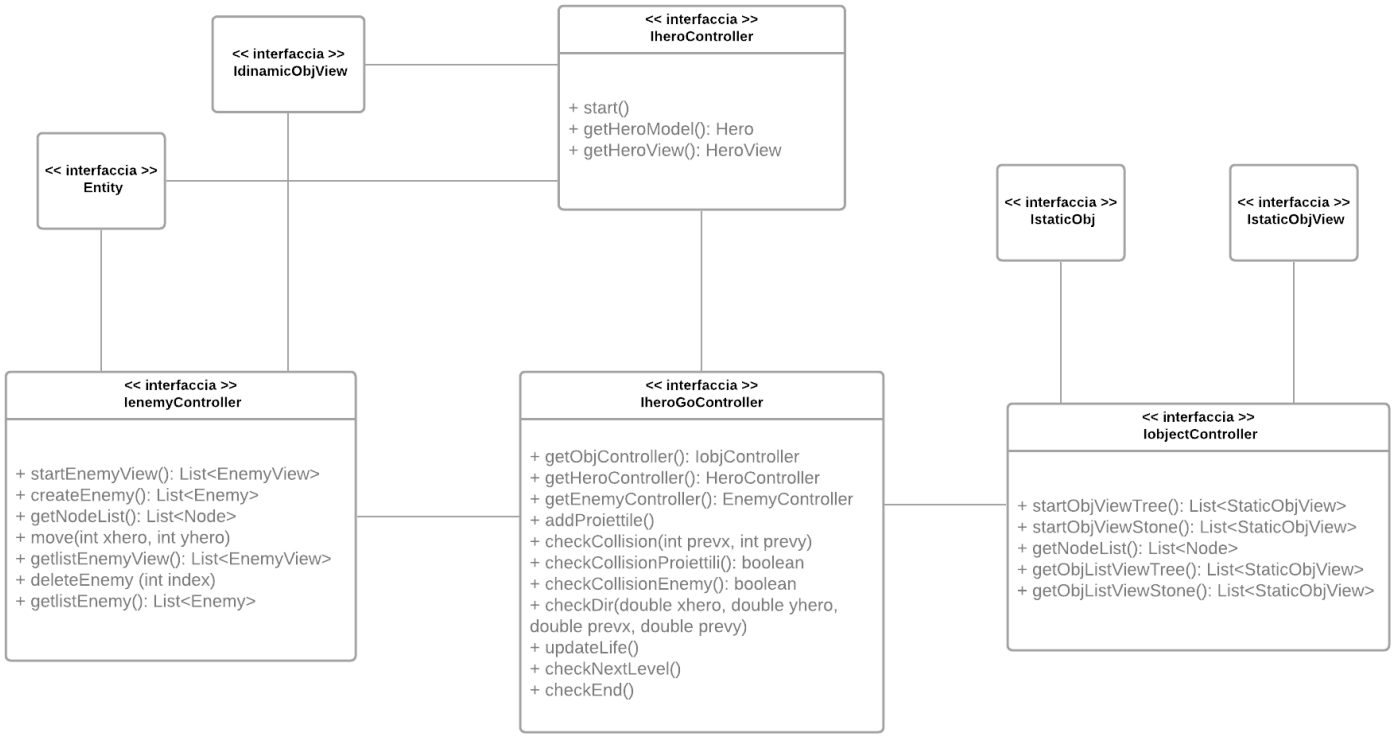

Fig.1: Schema UML delle entità principali

Design

Architettura

Per realizzare l'applicazione si è fatto uso del pattern architetturale MVC, con l'obiettivo di rendere ogni singola parte il più indipendente possibile dalle altre.

Tutte le entità vengono create nel Model, con le loro caratteristiche principali e non vi sono riferimenti né al View, né al Controller.

Nella View avviene l'interazione con l'utente attraverso la schermata del menu iniziale, successivamente attraverso le finestre dei livelli, inoltre ha il compito di far apparire le entità e intercettare gli input dell'utente.

La componente Controller è il mediatore tra Model e View, essa ha il compito di elaborare le informazioni fornite dal Model e di comunicarle alla View. Il centro del videogame è la classe HeroGoController cha ha accesso ha tutte le altre classi del gioco e si occupa insieme al GameLoop delle collisioni e dei movimenti.

Design dettagliato

MENU PRINCIPALE

Per tutti gli aspetti di grafica, il progetto fa uso della libreria JavaFX.

All'avvio del gioco si apre una finestra con un menu minimale che ha due bottoni: start ed exit.

HERO

Per la creazione dell'hero sono state implementate tre interfacce rispettivamente nei package Model, View e Controller.

L’interfaccia Entity è implementata dalla classe Hero che ne definisce le coordinate iniziali, le dimensioni, il percorso immagine e inoltre l’attributo *life*, con i rispettivi getter e setter. Per quel che riguarda la HeroView, implementa l’interfaccia IdinamicObjView che si occupa dell’impostazione delle dimensioni e posizioni del rettangolo che andrà a contenere Hero. Inoltre, ha il metodo setPosition che viene utilizzato dal controller, per aggiornare la posizione del rettangolo durante i movimenti.

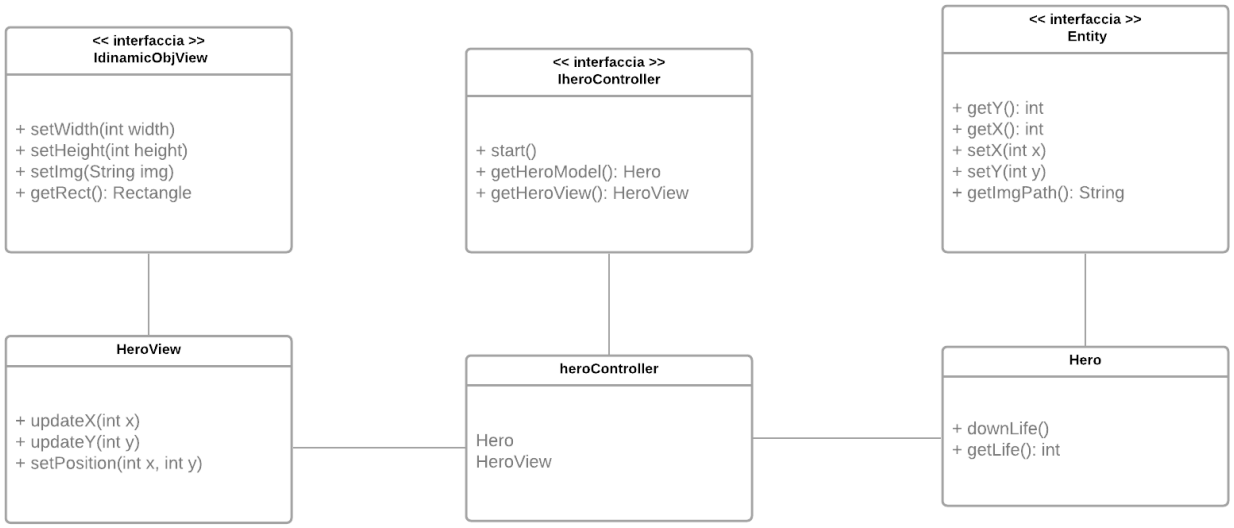

Fig.2: Schema UML Hero

ENEMY

In maniera similare all’Hero, anche la classe Enemy ha tre classi, ognuna per ogni logica MVC. Una delle differenze con l’eroe è il movimento automatico che viene implementato nell’EnemyController e il salvataggio della posizione precedente al movimento per evitare che le collisioni non facciano più muovere l’Enemy.

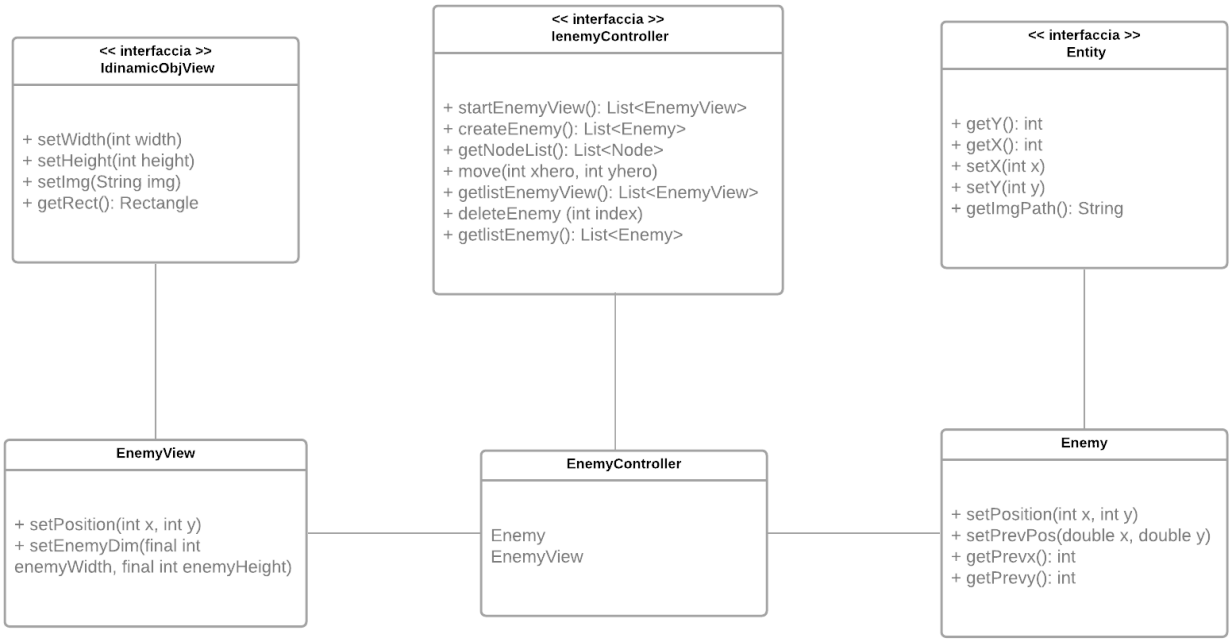

Fig.3: Schema UML Enemy

STATIC OBJECT

La classe staticObj del model è adibita alla creazione dell'oggetto, mentre nella staticObjView abbiamo i metodi necessari a capire la posizione dell'oggetto grazie al getRect, successivamente anche sfruttato per le collisioni. Per mostrare a schermo gli oggetti, così come per tutti gli altri componenti grafici il programma sfrutta la libreria JavaFX.

All'interno del package controller è stata creata IobjectController e le sue relative classi di implementazione (ciascuna per ciascun livello) la quale si occupa di generare gli oggetti (alberi/massi) in posizioni ben precise, per poi aggiungerli ad una list, la quale contiene tutti gli oggetti.

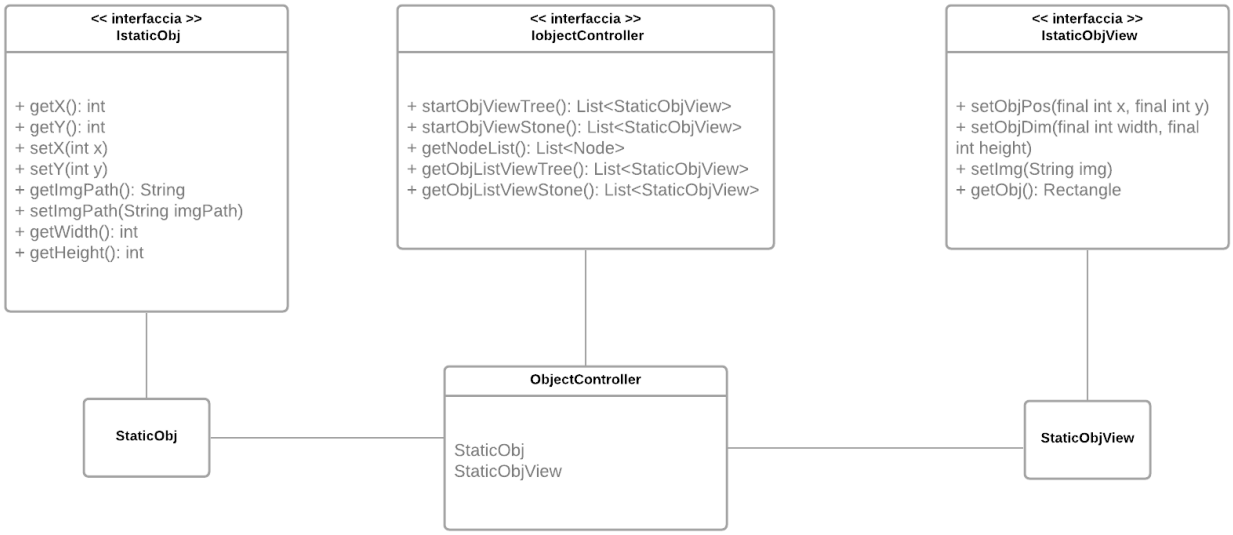

Fig.4: Schema UML Static Object

GAME LOOP E COLLISIONI

Nel GameLoop, abbiamo utilizzato un oggetto AnimationTimer, il quale richiama un timer per ogni frame, ha principalmente due metodi Start e Stop che ne permettono l'utilizzo. Il GameLoop in combinazione con il HeroGoController è responsabile della gestione delle collisioni, dell'aggiornamento della vita e del controllo sull'endGame. Inoltre, tramite l'implementazione del ViewObserver gestisce gli input da tastiera, per il movimento e il click del mouse, per sparare.

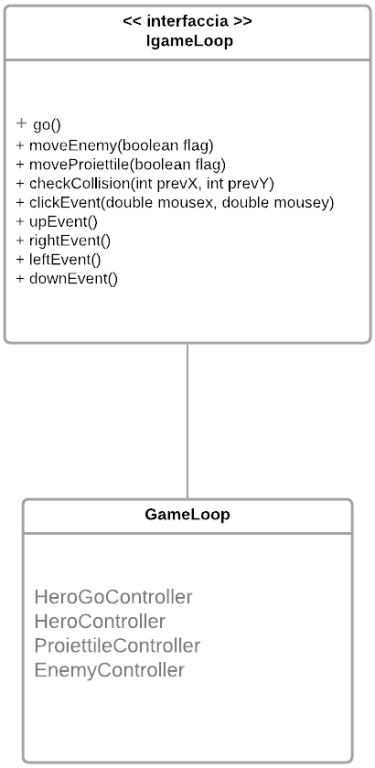

Fig.5: Schema UML GameLoop

Sviluppo

Note di sviluppo

- Utilizzo della libreria JavaFX: in particolare per l'utilizzo degli oggetti di tipo Rectangle, fondamentali nella gestione delle collisioni e per l'implementazione del GameLoop attraverso l'AnimationTimer.
- Utilizzo del FXML: solo per la Label della vita.

Commenti finali

Autovalutazione e lavori futuri

Lo sviluppo di tale progetto è stato impegnativo, soprattutto avendolo dovuto programmare interamente da sola, ma essere arrivata alla “fine” è stata una soddisfazione unica. Ho dovuto affrontare diverse problematiche, quindi ritengo di aver ampliato enormemente le mie competenze, ma sono comunque consapevole del fatto di avere tante altre cose da imparare. Ciò che mi ha lasciato questo progetto è la voglia di programmare e sperimentare, magari verso il mondo del gaming e del 3D.

Difficoltà incontrate e commenti per i docenti

Le maggiori difficoltà le ho incontrate nella gestione delle collisioni, infatti ho capito solo verso fine progetto che sarebbe stato meglio creare una classe che le gestisse senza dover duplicare codice, così come sarebbe stato meglio farlo nel GameLoop e nell’ HeroGoController.

Un altro aspetto su cui ho passato diverso tempo è stato il passaggio da un livello al successivo, probabilmente la soluzione che ho trovato non è del tutto corretta, ma è stato uno degli ultimi aspetti che ho curato quindi non ho avuto tempo per trovare un’altra soluzione.

Guida utente

All’avvio del gioco si apre il menu principale, con due bottoni:

Play: inizio del gioco

Exit: chiusura del gioco

Alla pressione del bottone play, inizierà immediatamente il primo livello.

Completati tutti i livelli o con la morte del personaggio, l’applicazione si chiuderà automaticamente.