Programmazione di Interfacce Grafiche e Dispositivi Mobili

Corso di Laurea in Ingegneria Informatica ed Elettronica – A.A. 2019-2020

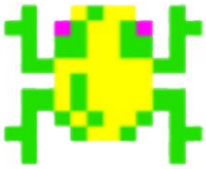
DIPARTIMENTO DI INGEGNERIA

FROG-ARCADE

PROGETTO DI:

LUCA TOMBESI

CHRISTIAN MONTECCHIANI



INDICE GENERALE

1) DESCRIZIONE DEL PROBLEMA

Questo progetto mira a ricreare Frogger, famoso gioco Arcade, nella maniera più fedele possibile mediante l'utilizzo delle componenti JavaFX.

IL GIOCO FROGGER

Frogger è un gioco nel quale il giocatore controlla una rana che deve riuscire, senza morire, a spostarsi dal basso verso l'alto dello schermo per raggiungere una delle 5 tane. La prima metà dello schermo consiste in una strada a cinque corsie, sulla quale passano nei due possibili sensi di marcia delle automobili che il giocatore deve evitare. La seconda metà invece consiste in un fiume, sul quale scorrono dei tronchi e delle tartarughe, le quali però dopo un certo periodo di tempo si immergono. I tronchi e le tartarughe presenti in ogni corsia si muovono nella stessa direzione. Mentre nella prima metà del livello la rana doveva evitare gli oggetti in movimento, in questa parte invece si deve servire dei tronchi e del guscio delle tartarughe per raggiungere, senza annegare, una tana in cima allo schermo. Per riuscire a vincere il giocatore deve portare una rana in ciascuna tana. Il giocatore dispone dalle cinque alle tre vite (a seconda del livello selezionato) e se le perde tutte prima di completare il livello perde. I modi con cui si possono perdere le vite oltre a quelli citati prima possono essere: il saltare in una tana già occupata; saltare sulle aree di spazio che delimitano le varie tane; il non raggiungere la tana in un tempo utile.

2) SPECIFICA DEI REQUISITI

1. Il programma prevede un menù iniziale e una schermata di gioco.
2. Il programma prevede la presenza di un sottofondo musicale.
3. Il programma prevede una serie di effetti sonori (clacson della macchina, rumore legato alla morte della rana, il gracidare della rana).
4. Il menù iniziale dà la possibilità di:
    a. Iniziare una nuova partita.
    b. Vedere la classifica dei punteggi totalizzati nelle partite precedenti.
    c. Togliere o rimettere la musica del gioco e i vari effetti sonori.
5. É possibile spostare la rana sia con le frecce direzionali sia con i tipici comandi W,A,S,D.
6. Ci sono diversi tipi di animazione a seconda se la rana è morta o ha raggiunto la tana.
7. Quando tutte le tane sono piene:
    a. Viene mostrata un'apposita scritta di vittoria.
    b. Il sistema mostra lo storico dei migliori risultati.
    c. Il sistema chiede al giocatore se iniziare una nuova partita o chiudere il programma.
8. Se il punteggio realizzato è tra i migliori 10 di sempre allora il sistema permetterà di salvarlo chiedendo al giocatore il suo nome.
9. In modo casuale in ogni livello compaiono uno o più bonus all'interno della tana disponibili solo per pochi secondi prima di scomparire.
10. Nella schermata di gioco c'è la possibilità di mettere in pausa la partita premendo il tasto pausa in alto a sinistra.
11. Il gioco presenta 3 livelli diversi di difficoltà crescente. Inoltre i punteggi assegnati con il raggiungimento delle tane e con i bonus presi aumenterà con la difficoltà del livello selezionato.
12. Il livello più semplice è quello principiante che presenta pochi ostacoli che si muovono lentamente.
13. Il secondo livello è quello medio che rispetto al primo presenta più ostacoli che si muovono più velocemente. In questo livello inoltre è presente il serpente che si muove lateralmente nell'area di prato che divide la strada dal fiume.
14. Il livello esperto rispetto agli altri 2 mette a disposizione un tempo minore per portare la rana in una delle tane. Inoltre nel livello è presente il coccodrillo. Esso analogamente alla tartaruga ed al tronco rappresenta un appoggio sicuro se la rana gli atterra sulla schiena, se invece gli salta sulla bocca la rana verrà mangiata e quindi morirà.

3) PROGETTO

In questa sezione si descrive la struttura dell'applicazione realizzata. Verrà prima illustrata l'architettura software e quindi verranno descritti i blocchi funzionali che la compongono.

3.1) ARCHITETTURA SOFTWARE

Per la realizzazione del gioco "Frogger" si è scelto di basarsi su un pattern Logic-View. Il seguente diagramma rappresenta l'architettura software implementata:

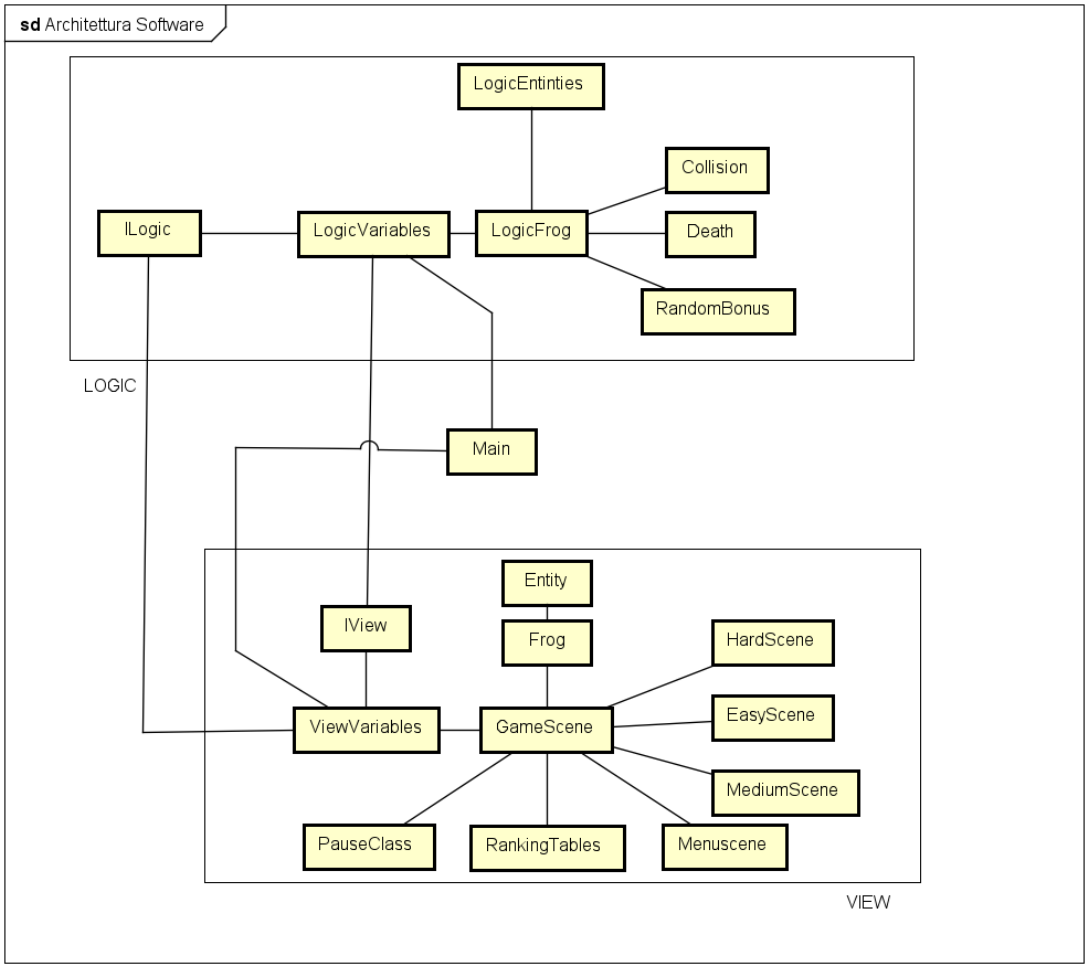


Figura 1: Architettura dell'applicazione.

Per semplicità sono state rappresentate solamente le classi principali.

VIEW

Viene ora descritto il package View e le classi fondamentali che lo compongono. La struttura del package è rappresentata dal seguente diagramma UML, in figura 2.

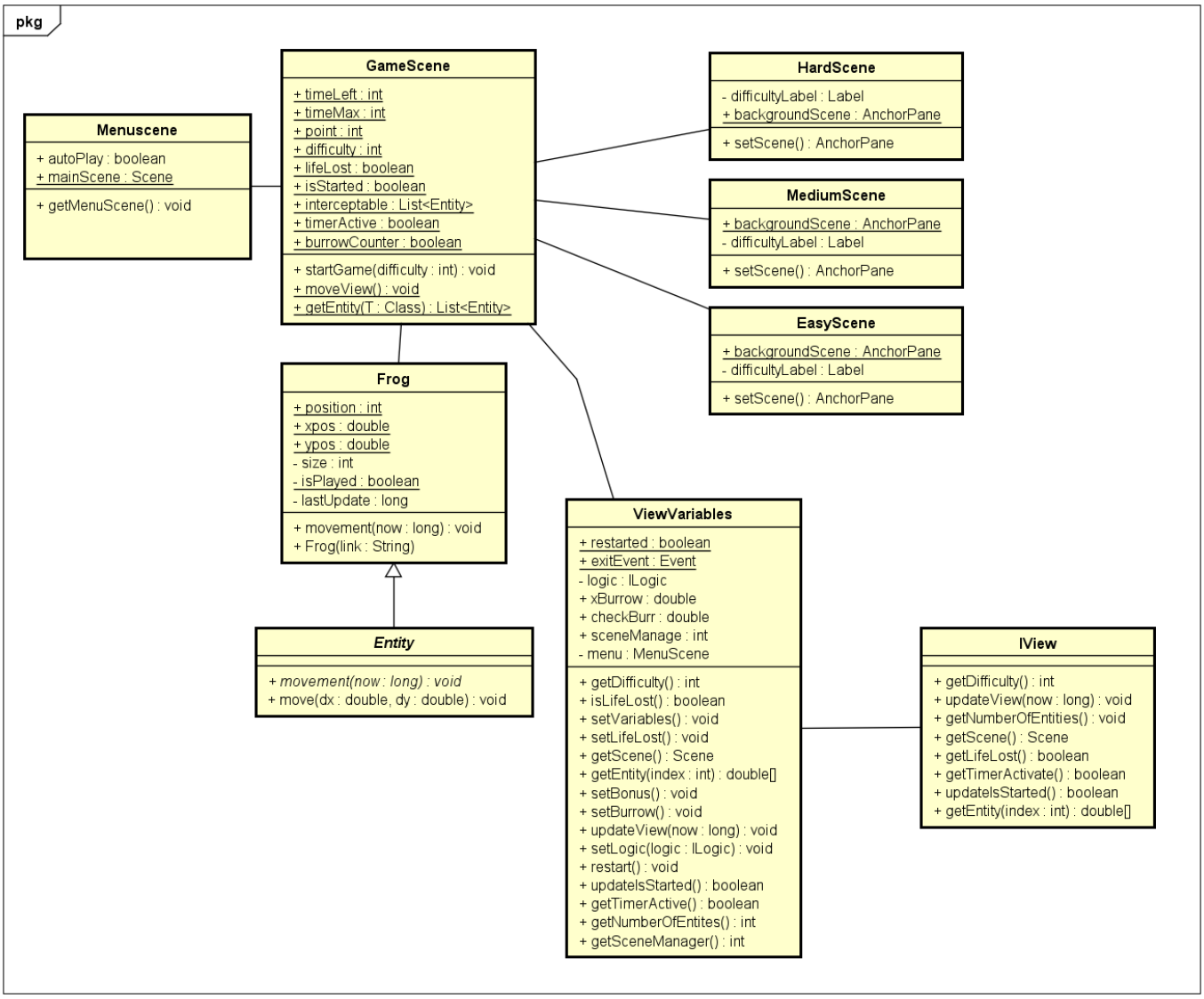


Figura2, Diagramma UML delle classi del View.

Nel View è presente la classe ***MenuScene*****,** che gestisce la visualizzazione della prima finestra di gioco ovvero il menu iniziale. Si è scelto di utilizzare l'AnchorPane come pannello dove sono stati posizionati i quattro pulsanti (Play, Audio, Ranking e Exit) e una ChoiceBox, che permette di scegliere una fra le tre difficoltà implementate (Easy, Medium, Hard). Ogni pulsante ha un ruolo ben preciso:

- **Exit:** chiude l'applicazione.
- **Audio:** silenzia la musica di sottofondo del menu.
- **Ranking:** visualizza la classifica dei giocatori migliori e il loro punteggio.
- **Play:** chiama il metodo **startGame()** della ***GameScene*** e varia il tema musicale di sottofondo.

La ***GameScene*** è la classe fondamentale del pacchetto View, perciò di seguito, vengono descritti in modo approfondito i metodi che la compongono. Questa classe contiene le informazioni per avviare la schermata principale del gioco e per gestirne lo svolgimento. I metodi che la compongono sono tre:

- **startGame()**: è il metodo chiamato quando viene premuto il pulsante "Play". Questo metodo crea la finestra di gioco principale, costituita da una scena che contiene un pannello nel quale vengono posizionati la rana, il pulsante pausa, le tre etichette relative al tempo mancante, alla difficoltà scelta e al punteggio. L'ambiente del gioco viene costruito da una delle tre classi **EasyScene()**, **MediumScene()** o **HardScene()** in base alla difficoltà selezionata. Ciascuna delle tre classi invoca il metodo **setScene()**.
- **startMoving():** gestisce le situazioni di vittoria o sconfitta mostrando la scritta "win" o "game over" a seconda del caso; decrementa il numero delle vite della rana, mostrate in grafica, quando la rana muore; mette in movimento ogni entità presente nella scena, usando il metodo **getEntity()**.
- **getEntity():** crea una lista contenente i soli nodi della scena che estendono la classe astratta Entity. Gli elementi di questa lista (ad esempio la rana, i tronchi, le tartarughe, le tane e i veicoli) sono i componenti principali del gioco.

Il metodo **setScene()** non fa parte della classe ***GameScene***, ma appartiene alle classi che distinguono i vari livelli. Questo metodo restituisce il pannello, dopo che, in base alla difficoltà selezionata, vengono posizionati i diversi elementi, con le relative dimensioni e velocità. Ogni elemento presente nella scena viene istanziato in questo metodo, a cui vengono passati la posizione dell'ascissa e dell'ordinata e altri parametri a seconda dell'entità.

Nel package View di fondamentale importanza è la classe ***ViewVariables***, che implementa l'Interfaccia ***IView***. Nella classe ***ViewVariables*** viene istanziato un oggetto ***ILogic***, che permette di chiamare tutti i metodi dell'interfaccia logica. I metodi fondamentali che la compongono sono:

- **updateView()**: viene richiamato continuamente nel ***Main*** per aggiornare le variabili e la visualizzazione.
- **getNumberOfEntities()**: restituisce la lunghezza dell'array delle entità presenti nella scena.
- **getScene()**: metodo richiamato dalla classe ***Main*** per acquisire la scena da visualizzare e dalla classe ***LogicVariables*** per acquisire la scena su cui richiamare i metodi che permettono il movimento della rana.
- **restart()**: ha il compito di resettare le variabili statiche nel momento in cui il giocatore, dopo aver vinto o dopo aver finito le vite, decide di ricominciare una nuova partita.
- **setVariables():** questo metodo imposta le variabili principali del gioco contenute nell'array di double *allVarr*, restituito dalla parte logica.
- **getEntity()**: restituisce un array di double dove sono contenute le informazioni principali delle entità del gioco.

La classe ***Frog***, che estende la classe ***Entity,*** è la classe più importante poiché imposta l'immagine della rana nell'animazione, attraverso l'intero *position* gestito dalla classe ***LogicFrog***.

LOGIC

Viene ora descritto il package Logic con il diagramma UML in figura 3.

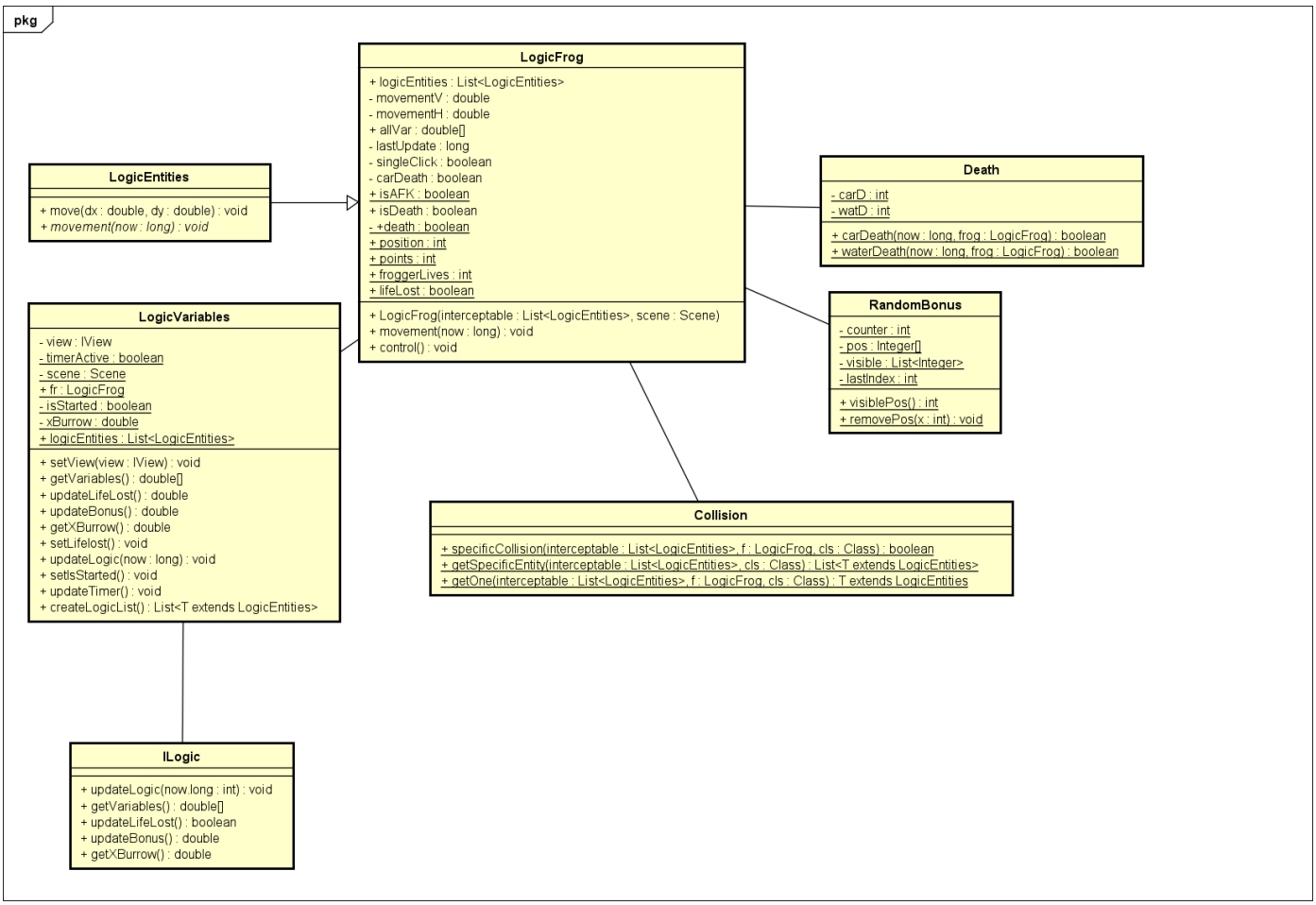


Figura 3: Diagramma UML delle classi del Logic.

Nel pacchetto Logic sono contenute tutte le informazioni relative alla geometria dei diversi oggetti presenti nella scena di gioco, tra i quali i veicoli, le tartarughe, le tane, il bonus, il serpente e i coccodrilli. La classe ***Death*****,** con i metodi **waterDeath()** e **carDeath()**, gestisce l'animazione di morte sulla base del modo in cui la rana ha perso una vita (la rana affoga o viene investita).

Le due classi fondamentali del pacchetto Logic sono le classi ***Collision*** e ***LogicFrog***.

La classe ***LogicFrog*** rappresenta logicamente la rana nel gioco e con il metodo **control()** gestisce il movimento della rana controllato dall'utente mediante i comandi W,A,S,D o le frecce direzionali. Tale metodo è richiamato nel metodo **movement()** ,nel quale vengono gestite sia le dinamiche dovute alle collisioni tra la rana e le altre entità sia le dinamiche dovute al tempo rimanente.

La classe ***Collision*** si occupa del rilevamento delle collisioni ed è costituita da tre metodi:

1. **speficCollision(**): restituisce vero se un oggetto di una classe specifica è entrato in contatto con la rana.
2. **getSpecificEntity()**: restituisce una lista di oggetti appartenenti ad una stessa classe che estende ***LogicEntity***.
3. **getOne()**: restituisce l'entità logica che collide con la rana.

In corrispondenza di ***ViewVariables*** nel pacchetto Logic è presente la classe ***LogicVariables***. In questa classe viene istanziato un oggetto dell'interfaccia ***IView*** per chiamare i metodi del ***ViewVariables***. Nella classe ***LogicVariables*** i metodi più importanti sono:

- **getVariables()**: serve a passare le variabili statiche aggiornate alla classe ***ViewVariables***.
- **updateLifeLost(**): restituisce il booleano che indica se la rana ha perso o meno una vita.
- **updateBonus()**: restituisce la posizione del bonus logico.
- **logicMovement()**: gestisce il movimento delle entità logiche presenti nella scena.
- **createLogicList(**): crea una lista di entità logiche che corrisponde alla lista di entità presenti nella scena di gioco.

3.2) PROBLEMI RISCONTRATI

COLLISIONE

Il problema della collisione, fra la rana e i diversi tipi di ostacoli che si incontrano lungo il percorso, è stato l'aspetto più difficile da implementare.

Leggendo la documentazione fornita da Oracle si è deciso di usare il metodo **intersects()**. Questo metodo restituisce vero se l'oggetto sul quale è stato chiamato interseca i contorni dell'oggetto tra le parentesi. Poiché gli ostacoli presenti nel gioco estendono la classe ***LogicEntities***, la quale a sua volta estende la classe rettangolo, si ha che ogni entità ha una forma geometrica ben precisa e l'intersezione viene gestita in maniera ottimale.

Nel metodo **movement()** della classe ***LogicFrog*** viene chiamato il metodo **specificCollision()** che serve a verificare se la rana è entrata in contatto con uno specifico tipo di entità logica. Se è necessario avere l'entità specifica con cui la rana è entrata in collisione (come per i tronchi o per le tane) viene chiamato il metodo **getOne()**.

RICOMPOSIZIONE DIVISIONE VIEW-LOGIC

Un altro problema presentatosi durante la realizzazione è stata la gestione dell'architettura.

Inizialmente non vi era una sostanziale divisione tra il pacchetto View e il pacchetto Logic, poiché vi era un forte accoppiamento tra le classi ed entrambi i package si occupavano in parte della parte grafica. Ovviamente, questo portava ad altri problemi, quali la poca flessibilità e la poca espandibilità del codice, per fare un esempio: per aggiungere o modificare una classe bisognava interagire con entrambi i package e fare cambiamenti drastici a tutto il codice.

In un primo momento si sono create le relative parti logiche di ogni entità e la classe **Variables**, la quale gestiva la comunicazione tra pacchetti. Essa gestiva molto bene il disaccoppiamento, ma solo in un'unica direzione: il pacchetto logico sfruttava solamente questa classe per ottenere le informazioni grafiche. Vi era, però, ancora una forte dipendenza poiché nella classe **Variables** venivano istanziati le entità logiche.

Il problema è stato risolto aggiungendo due interfacce e due classi che le implementassero in modo che la comunicazione tra i due package avvenisse solo all'interno di queste ultime due.

Nelle quattro figure successive sono rappresentati rispettivamente il menu inziale e i tre livelli in ordine crescente di difficoltà.

Figura 3: Menu iniziale, Frogger.

Figura 4: Livello Easy, Frogger.

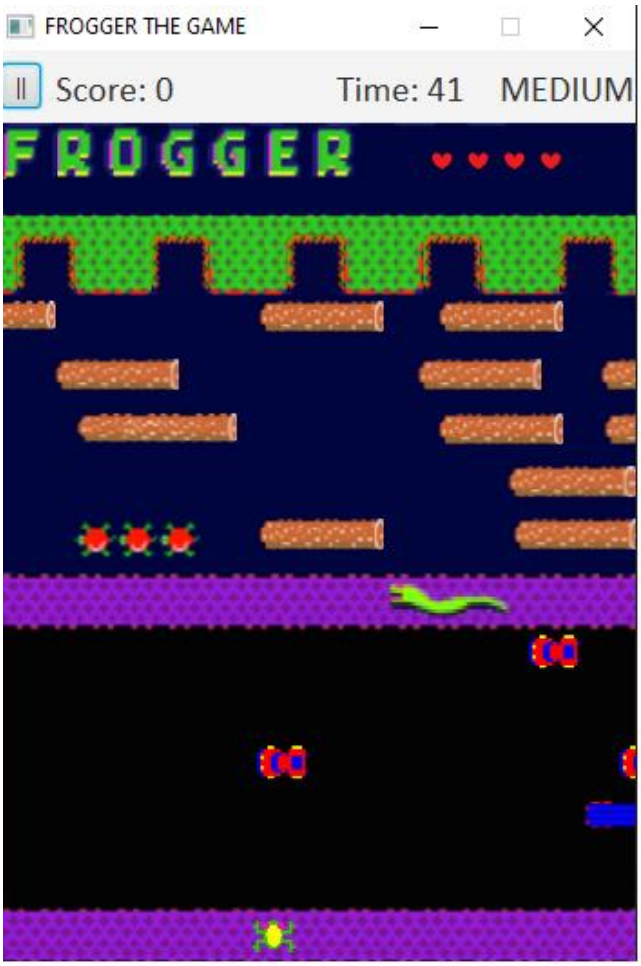


Figura 5: Livello Medium, Frogger.

Figura 6: Livello Hard,Frogger.

4) APPENDICE

Nel codice del progetto realizzato non vi sono parti di difficile comprensione, perciò in questo quarto punto di tesina si è deciso di esporre la possibilità di espansione del progetto.

Ricordando la vecchia architettura monolitica alla base del progetto, in cui aggiungere anche una sola entità comportava un gran rimaneggiamento del codice, adesso questo aspetto è di facile implementazione. Data la divisione Logic-View, una volta creata la classe grafica della nuova entità e la sua rispettiva classe logica risulta molto facile gestire sia l'intersezione con la rana sia le conseguenze annesse.
Inoltre è facile sia modificare la struttura dei singoli livelli (cambiando la posizione delle entità presenti o aggiungendone delle altre a piacere) sia aggiungere degli altri livelli in modo analogo con il quale sono stati realizzati i livelli già presenti.

La divisione dei compiti per la realizzazione del progetto: l'ingegnerizzazione del codice, quindi il cambio da architettura monolitica a Logic-View, la creazione delle classi riguardanti la classifica e il pacchetto Logic sono state curate da Tombesi Luca. Il resto è stato curato da Montecchiani Christian.

5) BIBLIOGRAFIA

- Wikipedia, “ Frogger “ , https://en.wikipedia.org/wiki/Frogger.
- Oracle Documentation, https://docs.oracle.com/javafx/2/.