



Tesina Finale di
Programmazione di Interfacce Grafiche e Dispositivi Mobili
Corso di Laurea in Ingegneria Informatica ed Elettronica, Curriculum Informatica
A.A. 2021-2022

DIPARTIMENTO DI INGEGNERIA

docente
Prof. Luca GRILLI

Fox On The Road

Applicazione Desktop JFC/SWING



studenti

329118 **Francesco Mancinelli** francesco.mancinelli3@studenti.unipg.it
329956 **Tommaso Cosimi** tommaso.cosimi@studenti.unipg.it

0. Indice

1 Descrizione del Problema	2
1.1 Il Videogioco Crossy Road	2
1.2 L'applicazione Fox On The Road	3
2 Specifica dei Requisiti	5
3 Progetto	7
3.1 Architettura del Sistema Software	7
3.2 Logic	8
3.3 View	12
3.4 Utilities	16
3.5 Problemi Riscontrati	17
4 Conclusioni e sviluppi futuri	19
4.1 Possibilità di estensione e personalizzazione	19
4.2 Sviluppi Futuri	19
5 Bibliografia	20

1. Descrizione del Problema

Questo progetto ha l'obiettivo di ricreare in maniera semplificata e con meccaniche ripensate il famoso gioco per Dispositivi Mobili *Crossy Road* [3, 4].

Per favorire un'ampia compatibilità su diverse piattaforme hardware e software, l'applicativo implementerà la tecnologia JFC/Swing.

L'applicazione è stata ampiamente testata su piattaforme Windows e *NIX-Like.

Nella realizzazione degli schemi *UML* per la tesina sono stati utilizzati il sito PlantTextUML [2] ed il software Inkscape [1] per la loro successiva conversione da SVG a PDF.

Sembra doveroso indicare i principali aspetti del videogioco originale *Crossy Road* [3, 4] nel prossimo paragrafo, così da poter esplicare in maniera più completa la nostra versione semplificata.

1.1 Il Videogioco Crossy Road

Il videogioco originale presenta una visualizzazione 2.5D, con meccaniche incentrate sull'utilizzo da parte di utenti di dispositivi mobili, e di conseguenza dispone di controlli basati su input touch-screen.

Crossy Road presenta un personaggio (inizialmente una gallina) il cui scopo è quello di attraversare la strada, senza ovviamente venire colpita dai veicoli.

Non esiste un'organizzazione a livelli, infatti il giocatore potrà continuare all'infinito finché non viene colpito da un veicolo o cade in una trappola secondaria.

Nel gioco originale, come prima anticipato, i comandi sono *touch-first* e l'utente dovrà fare un tap sullo schermo per muoversi in avanti, o fare uno swipe per muoversi lateralmente.

Il personaggio può muoversi liberamente nella mappa visualizzata, senza vincoli fuorché limiti laterali imposti dalla dimensione dello schermo del dispositivo mobile stesso.

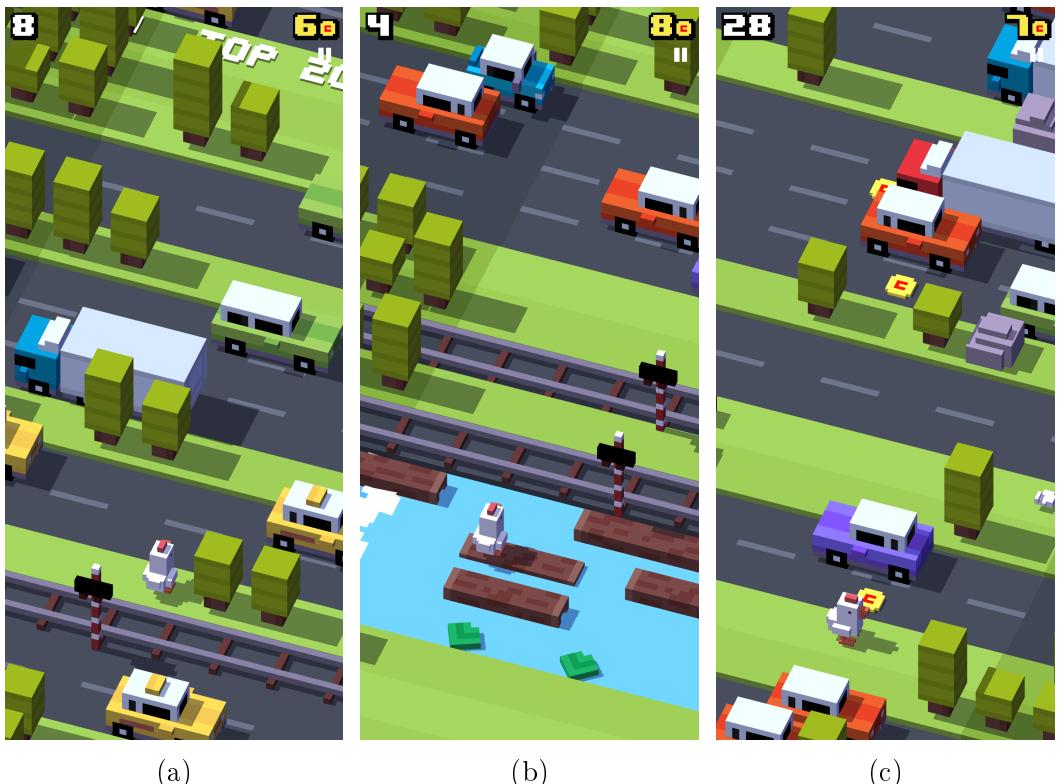


Figura 1.1: Tre screenshot del gioco *Crossy Road*. (a) rappresenta lo *stage* principale di gioco: la strada da attraversare. (b) rappresenta gli ostacoli secondari come il fiume. (c) invece rappresenta gli obiettivi nascosti come le monete da raccogliere.

Sarà possibile raccogliere monete durante il percorso, generalmente nascoste in giro per la mappa, che verranno poi utilizzate per sbloccare nuovi personaggi.

Oltre ad esse si potranno trovare invece ostacoli secondari, a partire da alberi e rocce che impediscono il passaggio del personaggio principale.

1.2 L'applicazione Fox On The Road

Fox On The Road è un applicativo Desktop che, come anticipato, vuole emulare un'esperienza semplificata di *Crossy Road*, ma con un particolare occhio all'utilizzo della tastiera.

L'applicazione presenta un'interfaccia che permette di visualizzare le statistiche di gioco oltre al gioco stesso, il tutto contenuto in un'unica finestra rappresentata tramite un *JFrame*.

L'utente vestirà i panni di una volpe e dovrà affrontare, inizialmente, un *tutorial* che consisterà in un livello guidato allo scopo di imparire i comandi e le meccaniche di gioco che dovrà affrontare. Dopo di esso, il *gameplay* è concepito come una serie di livelli sempre più impegnativi che metteranno alla prova la bravura dell'utente.

Ovviamente, per raggiungere la fine del livello la volpe non si dovrà far colpire dai veicoli che popolano le strade. L'obiettivo di ogni livello è arrivare alla fine del percorso senza perdere tutte le vite.



Figura 1.2: Screenshot dell'Applicazione *Fox On The Road*

2. Specifica dei Requisiti

L'applicativo *Fox On The Road* dovrà soddisfare la serie di requisiti sotto elencati:

1. Possibilità di muovere la volpe utilizzando due diverse combinazioni di tasti (WASD e frecce direzionali).
2. Presenza di un sottofondo musicale e dei suoni ambientali di gioco.
3. Presenza di più livelli di gioco con relativa percentuale di completamento.
4. Presenza di un livello di pratica che mette a disposizione dell'utente un'informatica sulle meccaniche di gioco (*Tutorial*).
5. Possibilità di estendere il gioco aggiungendo nuovi livelli semplicemente generando un nuovo file di testo che segua la convenzione indicata nel file README.txt.
6. Possibilità fermare l'avanzamento del timer, di fatto fermando le meccaniche di gioco principali tramite bottone grafico o input da tastiera (Implementazione di uno stato di pausa).
7. Possibilità di salvare i punteggi più alti.
8. Presenza di monete posizionate lungo il percorso che permettano di aumentare le vite disponibili.
9. Presenza di ostacoli statici quali alberi o rocce posizionati lungo il percorso che non permettano all'utente il passaggio.
10. Presenza di ostacoli mobili generati casualmente quali auto o camion.
11. Presenza di collisioni del personaggio principale con oggetti statici e dinamici con le relative conseguenze.

12. Impossibilità di sovrapposizione dei modelli degli ostacoli mobili.
13. Presenza di texture scelte casualmente per il tappeto erboso, gli ostacoli statici e gli ostacoli dinamici.
14. Presenza di texture animate per il personaggio principale e per le entità scelte.
15. Possibilità di visualizzare la mappa di gioco in maniera solidale al movimento della volpe.
16. Presenza di animazioni fluide per il movimento del personaggio principale e degli ostacoli mobili.
17. Possibilità di riconoscere l'applicativo aperto nella barra delle applicazioni in esecuzione tramite un'icona personalizzata.
18. Minimizzazione delle operazioni da terminale, privilegiando l'interfaccia grafica del gioco stesso e l'input da tastiera.

3. Progetto

Viene qua di seguito presentata in maniera esaustiva l'architettura del software e dei suoi singoli blocchi funzionali.

3.1 Architettura del Sistema Software

L'architettura software scelta per la realizzazione segue la filosofia Logic-View.

Il package Logic si occupa, come fa sottointendere il termine stesso, dell'implementazione della logica dell'applicazione. Complementarmente, il package View ha la responsabilità di gestire la realizzazione grafica del prodotto.

Per semplicità di implementazione, abbiamo deciso di utilizzare un altro package di supporto chiamato Utilities, che mette a disposizione metodi che fanno riferimento ad operazioni di I/O e metodi non categorizzabili come logici o visualizzativi.

Di seguito mettiamo a disposizione del lettore lo schema UML generale dell'applicativo correlato da una breve descrizione, dopodiché ci concentreremo sugli specifici package.

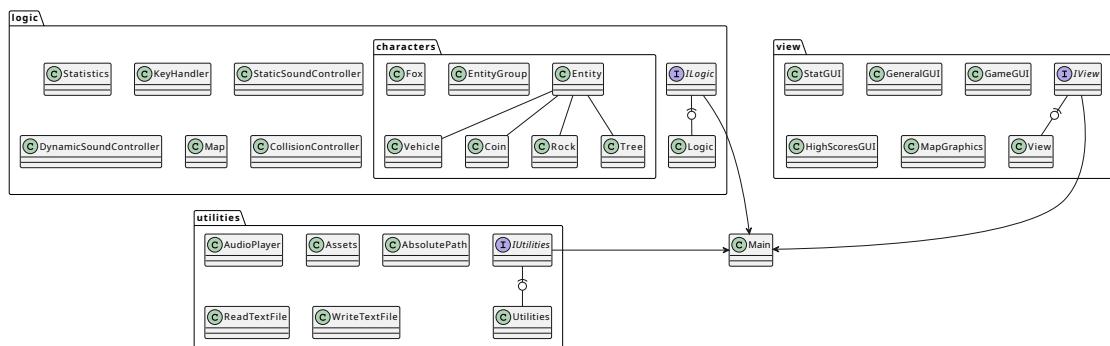


Figura 3.1: *Overview* dell'Architettura Software di *Fox On The Road*

Come ben si evince dallo schema in Fig. 3.1 ogni package ha a disposizione un'interfaccia per comunicare con le altre componenti senza avere necessità di condividere le proprie risorse.

Anche le interfacce sono tra loro collegate in quanto all'avvio dell'applicativo viene implementata una funzione di *Injecting* al livello del *Main*, che permetterà ai vari package di accedere a funzioni disponibili nelle altre strutture.

Tutte le classi - ad eccezione di casi particolari - all'interno di un package sono esplicitamente collegate alla classe che materializza l'interfaccia di comunicazione. Questa operazione viene svolta allo scopo di rendere modulare ogni parte del codice, così da renderlo sviluppabile da team indipendenti.

Come risultato di questa operazione, nessuna classe che non sia un'interfaccia (*ILogic*, *IView*, *IUtilities*), è collegata a nulla al di fuori del suo package, eccezione fatta per il collegamento con il *Main* di *Logic*, *View* ed *Utilities*, necessarie all'*Injecting*.

3.2 Logic

Viene riportata immediatamente sotto l'architettura del package *Logic* e del suo sotto-package *Characters*.

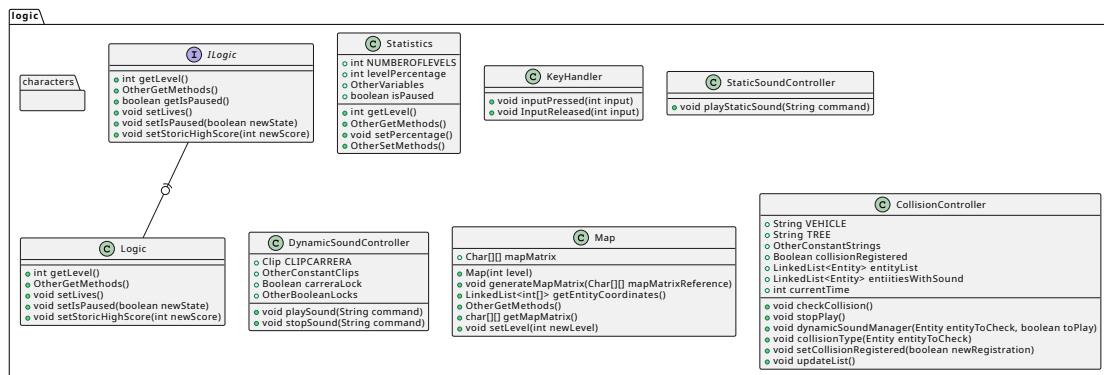


Figura 3.2: Package *Logic* nella sua interezza

• Logic

- Come accennato in precedenza, la classe *Logic* si occupa di collegare le classi interne al suo package con l'interfaccia di comunicazione *ILogic*. A tale collegamento sfuggono *DynamicSoundController* e *StaticSoundController*, che non hanno la necessità di essere richiamate da altri elementi al di fuori della logica.

- Statistics
 - La classe *Statistics* si occupa di monitorare e modificare le statistiche di gioco e controllare delle meccaniche, come ad esempio il cambio di livello con il metodo *setLevel(int newLevel)*, o la pausa tramite il metodo *setIsPaused(boolean newState)*.
- Map
 - La classe *Map* si occupa della costruzione logica della mappa di gioco, in base ai file di testo che rappresentano gli elementi e le entità, con la loro posizione all'interno di ciascun livello. Oltre ciò restituisce anche il numero di colonne giocabili per ogni livello.
- CollisionController
 - La classe *CollisionController* si occupa della gestione delle collisioni tra la volpe e le altre entità all'interno dell'applicativo. A seguire, una breve descrizione dei metodi principali:
 - * *checkCollision()* verifica che tra l'entità volpe e le altre entità contenute nella Lista *entityList* recuperata dalla classe *Map* non vi siano collisioni. Qualora ne fosse registrata una, il compito di gestirla verrà destinato alla funzione *collisionType()*. Nel caso la volpe invece fosse lontana da entità che generano rumore, il loro suono verrà interrotto da *stopPlay()*.
 - * *collisionType(Entity entityToCheck)* si occupa di controllare di che tipo sia l'entità che è stata colpita o che ha colpito la volpe, prendendo i provvedimenti necessari: ad esempio una collisione con un ostacolo statico come un albero fermerà la volpe, mentre la collisione con un veicolo porterà al reset del livello.
 - * *dynamicSoundManager(Entity entityToCheck, boolean toPlay)* controlla la riproduzione dei suoni delle entità in movimento e degli alberi.
 - * *stopPlay()* interrompe il suono proveniente da entità ora lontane precedentemente in riproduzione.
 - * *updateList()* si occupa di aggiornare la lista delle entità ad ogni cambio livello.
- StaticSoundController
 - La classe *StaticSoundController* si occupa di riprodurre suoni molto brevi di collisione con rocce e alberi, e raccolta di monete.

- DynamicSoundController

- La classe *DynamicSoundController* si occupa di riprodurre il suono ambientale degli alberi ed il rumore di ogni tipo di veicolo. Non è stato deciso di associare ad ogni singolo veicolo una *Clip* per motivi prestazionali e di sound design, perciò ci sarà una *Clip* per ogni tipologia di veicolo.

- KeyHandler

- La classe *KeyHandler* si occupa di gestire gli input da tastiera per poi restituirli alla classe *Fox* - contenuta nel sotto-package *Characters* - che li utilizzerà per muovere la volpe.

Viene riportata immediatamente sotto l'architettura del sotto-package *Characters*, contenuto nel package *Logic*.

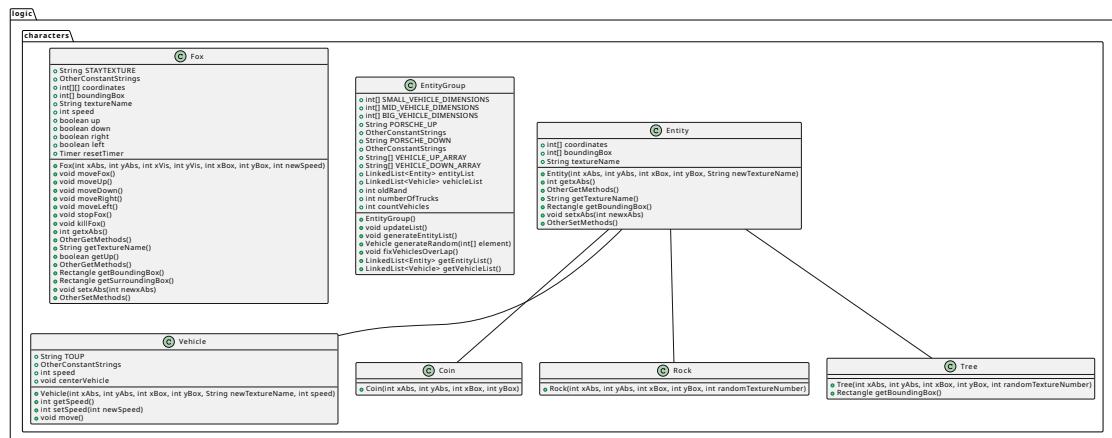


Figura 3.3: Sotto-package *Characters* di *Logic*

- Fox

- La classe *Fox* rappresenta il concetto logico di volpe all'interno dell'applicativo. Si occupa del movimento e degli stati della volpe. Il movimento è affidato ai metodi *moveFox()*, *moveUp()*, *moveDown()*, *moveRight()* e *moveLeft()*, invece gli stati della volpe sono affidati ai metodi *killFox()*, *stopFox()* e ai metodi di movimento che si appoggiano al metodo *setTextureName()*.

- Entity

- La classe *Entity* è una generalizzazione delle varie classi *Coin*, *Rock*, *Tree* e *Vehicle*.
- Coin
 - La classe *Coin* rappresenta il concetto logico di moneta all'interno dell'applicativo.
- Rock
 - La classe *Rock* rappresenta il concetto logico di roccia all'interno dell'applicativo.
- Tree
 - La classe *Tree* rappresenta il concetto logico di albero all'interno dell'applicativo.
- Vehicle
 - La classe *Vehicle* rappresenta il concetto logico di veicolo all'interno dell'applicativo. Essendo però elementi dinamici, hanno a disposizione un attributo che rappresenta la loro velocità ed un metodo *move()* che permetta il movimento.
- EntityGroup
 - La classe *EntityGroup* si occupa della generazione delle entità con *texture* e *hitbox* associate casuali, delle loro liste e della gestione delle collisioni tra entità dinamiche (veicoli). Di seguito vengono riportate le descrizioni di alcuni dei metodi principali:
 - * Il costruttore *EntityGroup()* va a generarsi immediatamente la lista di Entità tramite il metodo *generateEntityList()*.
 - * Il metodo *generateEntityList()* genera una lista collegata contenente tutte le entità - statiche e dinamiche - presenti nel livello. Provvede anche al riempimento di un'altra lista adibita solo ai veicoli.
 - * Il metodo *fixVehiclesOverlap()* ha il compito di controllare che le entità dinamiche non si sovrappongano ed eventualmente spostarne indietro una delle due.
 - * Il metodo *generateRandom(int[] element)* ha il compito di generare casualmente dei nomi di *texture* che si riferiranno - sul package *View* - a delle *texture* specifiche.

- * I metodi *flushList()* e *updateList()* hanno rispettivamente il compito di svuotare la lista ad ogni cambio di livello e di aggiornarla.

3.3 View

Viene riportata immediatamente sotto l'architettura del package *View*.

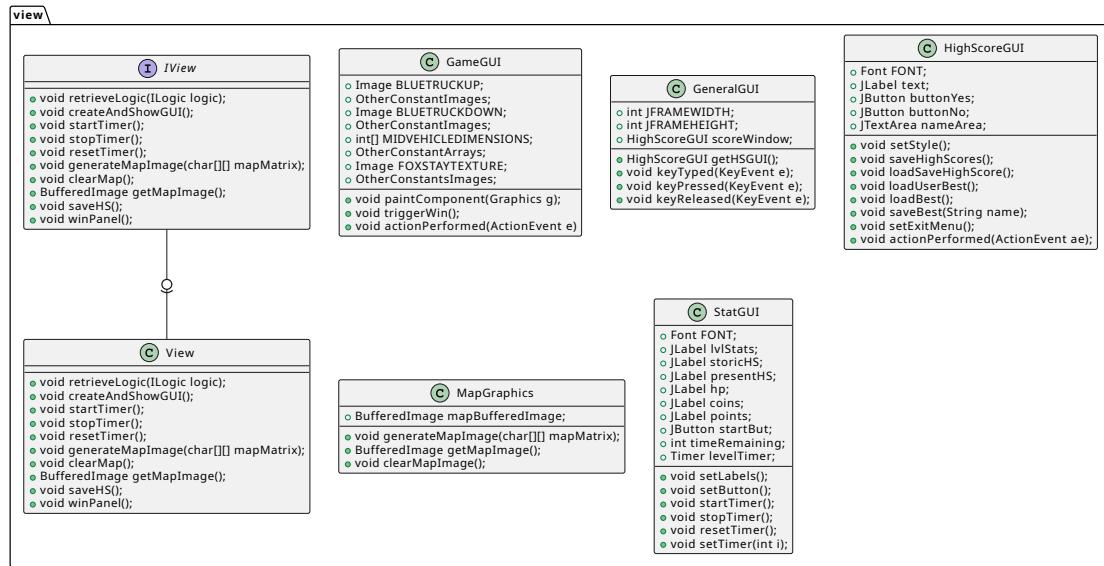


Figura 3.4: Package *View* nella sua interezza

- **GeneralGUI**

- La classe *GeneralGUI* si occupa di contenere i sottopannelli *GameGUI*, *StatGUI* e *HighScoreGUI* all'interno di un'unica finestra. Oltre ciò si occupa di estendere il *KeyListener* per catturare gli input dell'utente da trasferire successivamente alla classe *KeyHandler* nel package *Logic*.



Figura 3.5: *Screenshot* di *Fox On The Road*, viene mostrata la GeneralGUI

- GameGUI

- La classe *GameGUI* si occupa di visualizzare gli elementi grafici del gioco tramite il metodo *paintComponent()*, che li visualizza nel suo *JPanel*. Grazie ai metodi della classe astratta *ActionListener*, è stato impostato un timer di 16ms (62,5 Frame per Secondo) che richiama nuovamente il *paintComponent()*, permettendo il refresh della grafica. Alcuni elementi non sono sempre visualizzati, tra di loro spiccano l'*overlay* del tutorial e l'immagine di vittoria.



Figura 3.6: Dimostrazione del funzionamento della classe *GameGUI*

- StatGUI

- La classe *StatGUI* si occupa di visualizzare le statistiche della partita all'interno della finestra del gioco, accompagnandole ad un *JButton* che ha come *Label* un timer di livello che permetta di far andare il gioco in uno stato di pausa. Tra le statistiche visualizzate spiccano i punteggi migliori, le vite rimanenti e le monete raccolte.

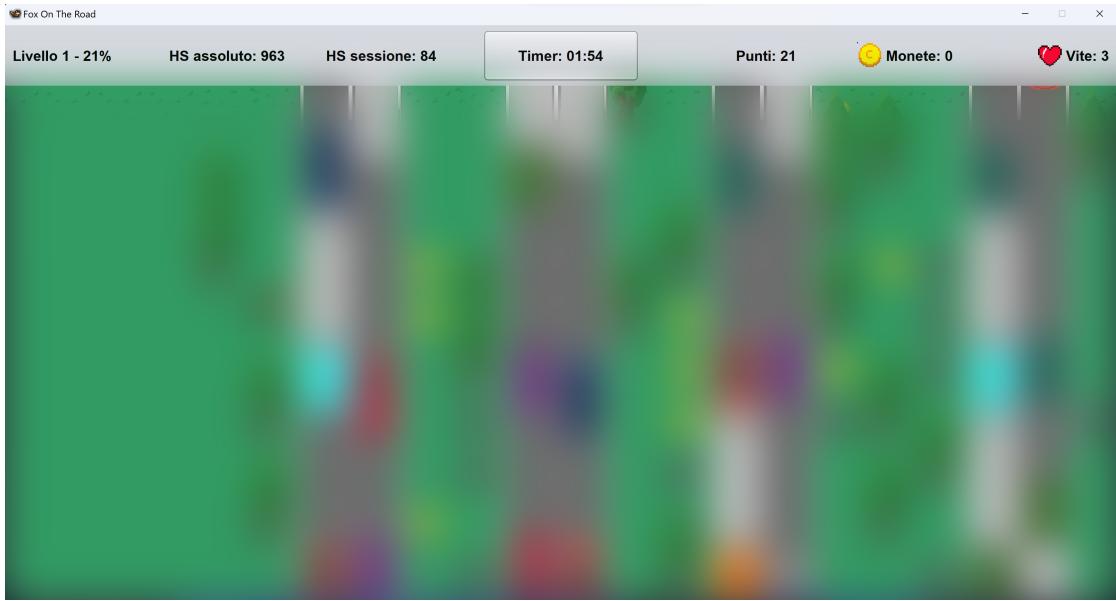
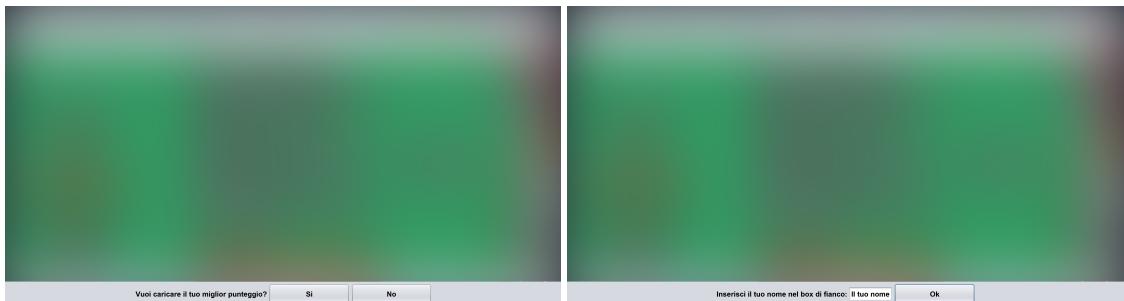


Figura 3.7: Dimostrazione del funzionamento della classe *StatGUI*

- HighScoreGUI
 - La classe *HighScoreGUI* implementa un *JPanel* che permette all’utente di salvare e/o caricare i migliori punteggi. Permette anche l’inserimento di nomi personalizzati, in modo da stilare una classifica consultabile tramite l’apposito file CSV nella cartella *Assets/Misc*.



- (a) Richiesta di caricamento del miglior punteggio personale (b) Richiesta del nome a seguito di una risposta affermativa

Figura 3.8: Dimostrazione del funzionamento della classe *HighScoreGUI*

- MapGraphics
 - La classe *MapGraphics* riceve in input la mappa logica generata dalla classe *Map* nel package *Logic* e produce in output le grafiche della

mappa, assegnando ad ogni elemento della matrice la *texture* appropriata scelta in maniera casuale. Oltre alla mappa pura e semplice, si occupa di piazzare nella locazione desiderata e di assegnare le *texture* corrispondenti alle entità statiche contenute nel file della mappa.

3.4 Utilities

Viene riportata immediatamente sotto l'architettura del package *Utilities*.

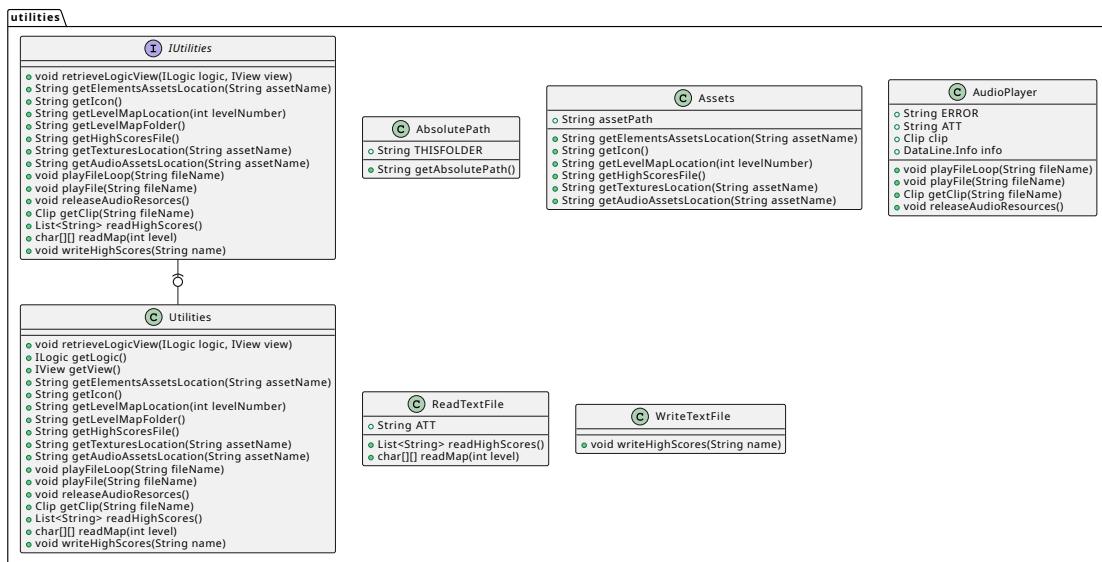


Figura 3.9: Package *Utilities* nella sua interezza

- *AbsolutePath*
 - La classe *AbsolutePath* si occupa esportare come stringa la posizione assoluta nel *File System*.
- *Assets*
 - La classe *Assets* si occupa di restituire le locazioni assolute delle risorse richieste facendo affidamento sulla posizione indicata da *AbsolutePath*.
- *ReadTextFile*
 - La classe *ReadTextFile* permette di leggere i file *CSV* e *TXT*, quindi si occupa di leggere il file con i punteggi migliori ed i file delle mappe.
- *WriteTextFile*

- La classe *WriteTextFile* permette di scrivere su file CSV, quindi si occupa di scrivere sul file con i punteggi migliori.
- **AudioPlayer**
 - La classe *AudioPlayer* permette di riprodurre file audio in formato WAV una sola volta o in loop, o di restituire le *Clip* per la loro riproduzione al di fuori di *Utilities*. Esempio eclatante sono quelle riprodotte all'interno di *DynamicSoundController* e *StaticSoundController*.

3.5 Problemi Riscontrati

L'implementazione della logica di *Fox On The Road* non ha sollevato problemi ingenti, tuttavia non si può dire che sia stata un'avventura senza imprevisti.

Dato che non avevamo un grande bagaglio culturale dal punto di vista della programmazione al di fuori dell'ambito universitario, i problemi più impegnativi si sono presentati nel primo momento di stesura e progettazione del codice del software: in particolare nel momento in cui dovevamo visualizzare le entità sopra al manto erboso che costituisce la base della visualizzazione grafica della *GameGUI*, ad esempio.

Si riportano di seguito alcuni dei più impegnativi (per noi) problemi riscontrati durante la stesura del codice:

- Gestione dell'accesso concorrenziale delle liste, che contengono le entità che devono essere visualizzate nella posizione allocata nella mappa di gioco, risolto copiando la lista e scorrendo la copia;
- Gestione delle risorse sonore per le entità, risolto aggiustando l'implementazione delle *Clip* da riprodurre;
- Gestione delle collisioni delle entità con il personaggio principale, in particolare riferiti alla collisione tra la texture della volpe e quelle delle entità statiche, che venivano inizialmente sovrapposte. Questo problema è stato risolto nella classe *CollisionController* spostando indietro la volpe ogni volta ci fosse un'interazione tra i due;
- Gestione della sovrapposizione dei veicoli, che essendo generati randomicamente e tutti nella stessa posizione iniziale, potevano sovrapporsi in maniera non consona, è stato risolto nella funzione *FixVehiclesOverlap()* nella classe *EntityGroup*;

- Gestione di JPanel atti all’interazione con l’utente ma che non sono fissi, in quanto presenti solamente nei casi in cui l’utente voglia caricare il suo miglior punteggio o debba uscire dall’applicazione salvando i progressi fatti, implementato nella classe *HighScoresGUI*;
- Generazione della mappa tramite *tile* di *BufferedImage* che vanno uniti in un’unica *BufferedImage*, problemi risolti nella classe *MapGraphics* dell’applicativo;
- Gestione del cambio dei livelli e lo svuotamento delle liste contenenti le entità del livello precedente, nonché il cambio di mappa, criticità risolte grazie al metodo *setLevel()* presente nella classe *Statistics*;
- Fluidità del movimento della volpe, che inizialmente era stata implementata tramite il semplice *KeyListener*, senza supporto dal *KeyHandler* ed anche grazie alla presenza di GIF che accompagnino il rispettivo movimento, creando una scena più dinamica.

4. Conclusioni e sviluppi futuri

4.1 Possibilità di estensione e personalizzazione

L'applicazione *Fox On The Road* è stata concepita come espandibile sin dall'inizio, con possibilità di aggiungere nuovi livelli semplicemente generando un file di testo da dieci righe con gli elementi indicati nella nostra guida "README.txt".

Viene data all'utente dunque la possibilità di personalizzare, estendere o addirittura eliminare i livelli già presenti e continuare il gameplay creandone altri, senza dover inserire una singola riga di codice nei file sorgente, per questo l'applicazione verrà distribuita esclusivamente nel formato *.jar*.

Oltre al rinnovamento nella struttura dei livelli, è disponibile la personalizzazione del lato estetico e/o sonoro, quindi delle *Texture* di gioco, e delle *SoundTrack*.

Ad esempio, sostituendo i file originali delle *Texture* con altri personalizzati che seguano i requisiti specificati nel file *README.txt* c'è la possibilità di avere un'esperienza di gioco medesima con i personaggi che più vi aggradano.

4.2 Sviluppi Futuri

Proprio per merito dell'espandibilità di cui si è parlato poc'anzi, l'utente finale - o chi per lui - ha la possibilità di espandere come vuole il gioco, rinnovandone il *gameplay*, la veste grafica e/o sonora, il tutto mantenendo le meccaniche di gioco previste in fase di progettazione, e senza toccare codice sorgente.

5. Bibliografia

- [1] Inkscape. Disegna la libertà. Sito web <https://inkscape.org/>.
- [2] PlantTextUML. The expert's design tool. Sito web <https://www.planttext.com/>.
- [3] Wikipedia. Crossy road — Wikipedia, l'enciclopedia libera, 2022. Online https://it.wikipedia.org/wiki/Crossy_Road, controllata il 16-agosto-2022.
- [4] Wikipedia. Crossy road — Wikipedia, the free encyclopedia, 2022. Online https://en.wikipedia.org/wiki/Crossy_Road, controllata il 09-settembre-2022.