

Relazione per
“Progetto Uso!”

Luzietti Manuel
Santoro Matteo

25 ottobre 2021

Indice

1 Analisi	2
1.1 Requisiti	2
1.2 Analisi e modello del dominio	3
2 Design	5
2.1 Architettura	5
2.2 Design dettagliato	6
2.2.1 Manuel Luzietti	7
2.2.2 Matteo Santoro	12
2.2.3 Shared Tasks	16
3 Sviluppo	17
3.1 Testing automatizzato	17
3.2 Metodologia di lavoro	17
3.2.1 Gestione del DVCS	17
3.2.2 Suddivisione del Lavoro	17
3.2.3 Manuel Luzietti	18
3.2.4 Matteo Santoro	19
3.2.5 In collaborazione	19
3.3 Note di sviluppo	20
3.3.1 Manuel Luietti	20
3.3.2 Matteo Santoro	21
4 Commenti finali	22
4.0.1 Manuel Luzietti	22
4.0.2 Matteo Santoro	22
A Guida utente	24

Capitolo 1

Analisi

Il software proposto si pone di realizzare un emulatore del videogioco musicale Osu! nel quale il giocatore deve colpire degli elementi chiamati HitCircle a tempo di musica senza andare fuori tempo e non fare quindi diminuire la barra della vita.

1.1 Requisiti

Il software dovrà permettere all'utente di poter compiere una partita, avendo la possibilità di scegliere la beatmap desiderata, delle opzioni nelle quali puo modificare cose come la risoluzioni o il volume. Il gioco si compone di alcune terminologie che abbiamo usato e che andremo ora ad elencare;

Beatmap

File particolari che possiedono un estensione di tipo .osu, all interno di questi file ci sono i settaggi per la partita quali difficolta, tempistiche, raggio di HitCirle con le quali modelliamo le diverse difficoltà.

HitCircle

Entità grafiche che compaiono a tempo di musica che dovranno essere cliccate per poter assegnare un punteggio se colpite in un determinato momento. La loro dimensione è contenuta all'interno della Beatmap, si compongono a loro volta di due componenti, un Circle ossia un cerchio colorato ed un Ring cioè un anello che indica al giocatore il momento esatto in cui va colpita l'intera figura per poter fare il punteggio più alto.

Approach Time

Una misurazione del tempo che indica il grado di precisione con il quale l'utente ha colpito un HitCircle rispetto al ClosureTime cioè il tempo di chiusura del Ring.

Requisiti funzionali

- Il fulcro dell'intero gioco si basa sull'entità HitCircle, il software dovrà quindi riuscire a gestirli generandoli in una precisa posizione in due dimensioni sullo schermo, in un preciso momento e facendoli poi scomparire se pigiati entro il tempo limite oppure se scaduto il suddetto tempo.
- Uso! dovrà essere in grado di leggere ed interpretare i file di tipo .osu per poter modellare il gioco correttamente.
- Essendo un gioco musicale è anche necessario che i file di tipo audio siano riprodotti correttamente e sincronizzati con la generazione/maturazione degli HitCircle affinché il gioco sia adatto.

Requisiti non funzionali

- Il tempismo con il quale gli HitCircle appaiono dipende esclusivamente dalla struttura della Beatmap che si può scaricare o creare.

1.2 Analisi e modello del dominio

Uso! all'avvio dovrà eseguire una specie di login, nel quale l'utente inserisce il proprio nick, successivamente si troverà davanti ad un menu nel quale potrà selezionare

- un'opzione play per essere indirizzato verso un menu nel quale potrà scegliere una canzone con la relativa difficoltà.
- tasto Option per poter diminuire/aumentare il volume del gioco, poter cambiare risoluzione
- tasto exit per poter uscire da Uso!

Successivamente il gioco inizierà, dalla beatmap verranno caricati i settaggi per la difficoltà tra i quali il raggio degli HitCircle che diventa via via minore con l'aumentare della difficoltà della beatmap, terminerà solo al compimento della beatmap ossia alla fine della canzone oppure nel caso la vita dell'utente indicata su una barra si esaurisca. Colpire con il giusto tempismo gli HitCircle non solo permette di guadagnare un numero di punti ma anche di alimentare la barra della vita che con il tempo diminuirà e subirà un ulteriore calo nel caso in cui si verifichi un miss di un HitCircle. Alla fine della partita ci sarà una piccola schermata di riepilogo nella quale si vedranno le statistiche della

partita che è appena stata conclusa, quali il punteggio totale, la serie di hit più lunga e la precisione con la quale è stato colpito ogni HitCircle.

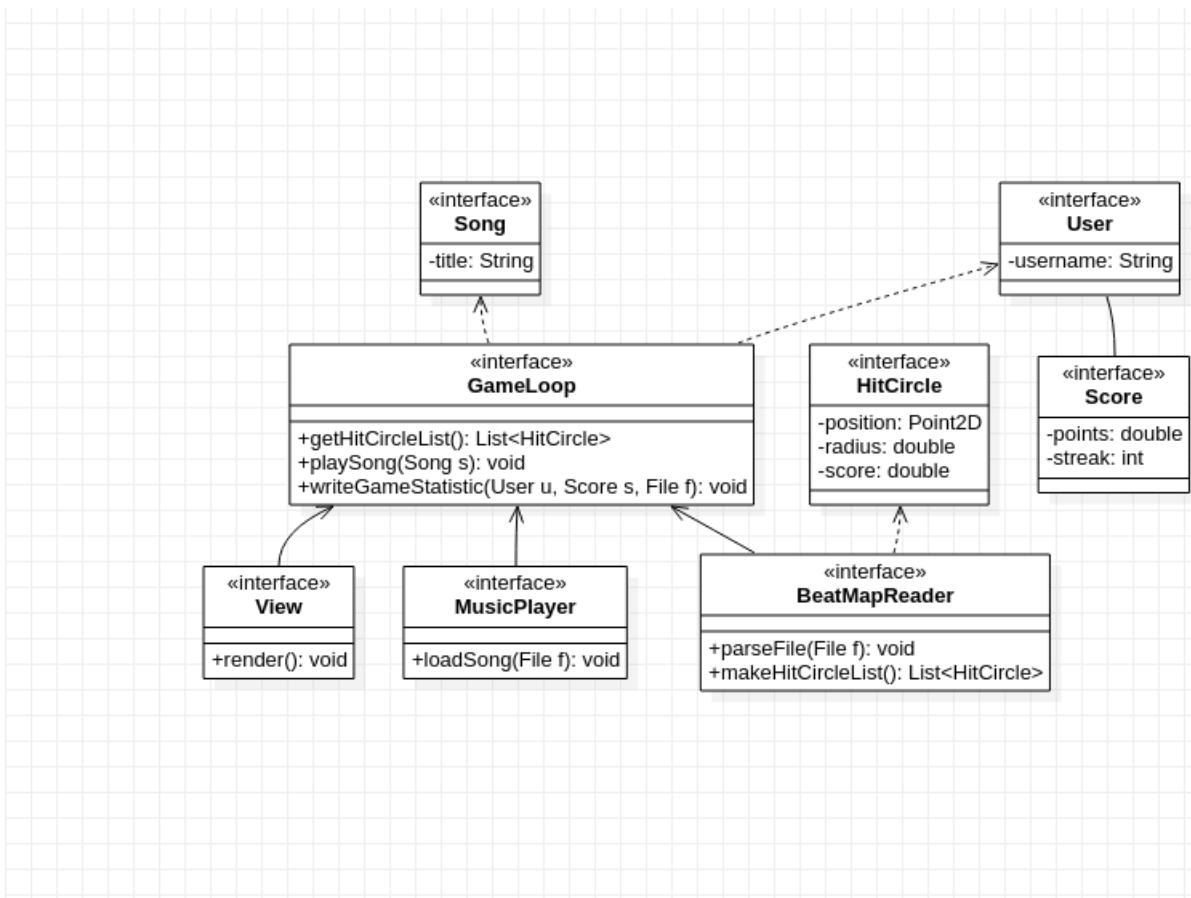


Figura 1.1: UML dell'analisi del problema

Capitolo 2

Design

2.1 Architettura

L’architettura di Uso! è basata interamente sul pattern architettonicale MVC. Il core del gioco risiede nel GameModel, nel quale sono più o meno contenuti tutti gli elementi necessari ad effettuare una partita. Per la View abbiamo scelto di utilizzare la libreria JavaFX, esistono diverse classi che rappresentano le View che astraggono concetti come il menu, la schermata di gioco e la schermata finale riepilogativa, non abbiamo deciso di creare multiple ”windows” piuttosto ci è sembrato esteticamente più bello utilizzare degli effetti di dissolvenza tra le Scene delle classi rimanendo fissi sempre sullo stesso Stage. Nel Model per la gestione dei dati abbiamo messo classi che ad esempio modellano il concetto delle statistiche di un gioco, e la logica che sta dietro a molte entità coinvolte in Uso! La parte di Controller invece funge da tramite tra view e model. Si occupa di manipolare il modello e di aggiornare la view. Prende gli eventi della view e aggiorna il model e allo stesso tempo prende informazioni dal model e li notifica alla view, aggiornandola di conseguenza.

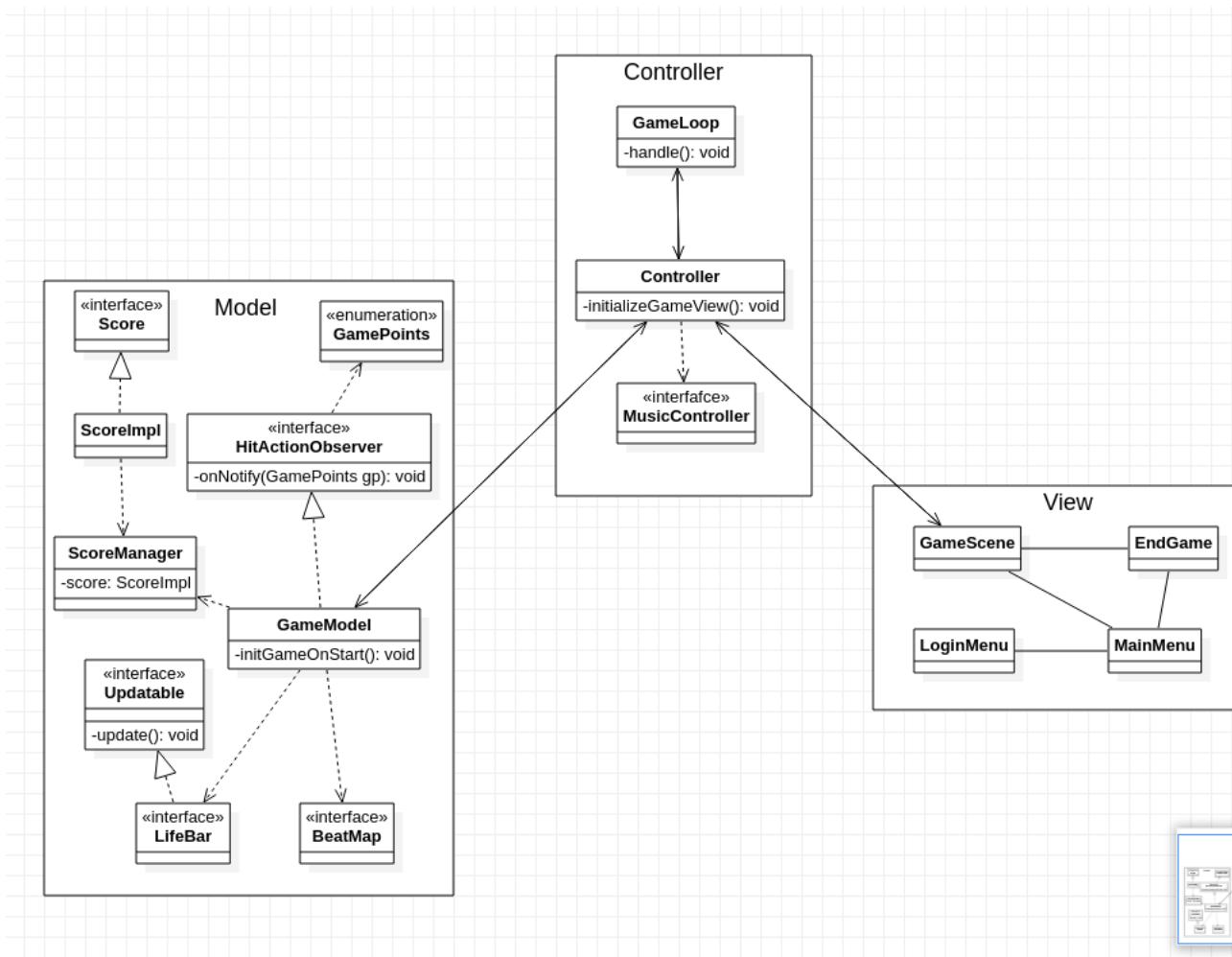


Figura 2.1: UML del design del problema

2.2 Design dettagliato

In principio il nostro gruppo era composto da 3 persone, a seguito della mancata collaborazione di un componente del gruppo abbiamo dovuto dividerci le parti mancanti decidendo di collaborare entrambi ad esse. Di seguito elenchiamo le principali features del gioco, divise per membri del gruppo:

2.2.1 Manuel Luzietti

BeatMapReader

La classe che si occupa di parsare i file .osu è BeatmapReader, la quale preleva alcuni elementi utili per definire una BeatMap e di conseguenza tutte le informazioni necessarie per aggiornare correttamente lo stato del gioco:

- velocità di chiusura del cerchio più esterno di un hitpoint
- velocità di drenaggio della vità
- velocità di comparsa del cerchio
- posizioni degli hitcircle
- tempo di spawn degli hitcircle

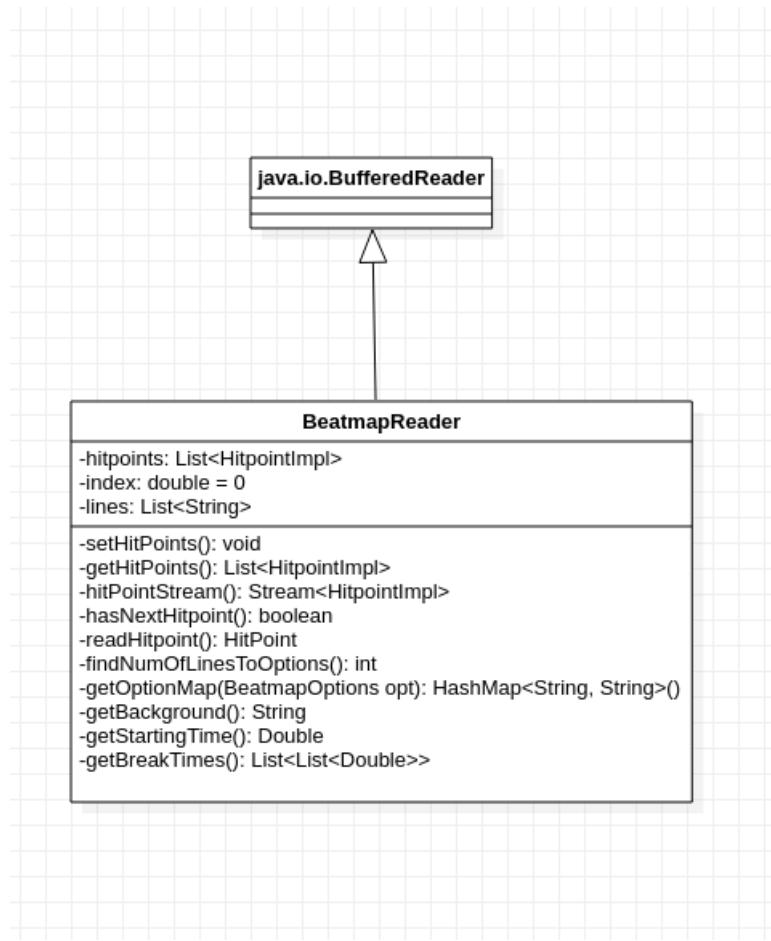


Figura 2.2: UML del BeatmapReader

GameModel e Update Method

E' il cuore del Model, composta degli elementi di gioco, rappresenta lo stato di una partita in corso relativo ad una particolare canzone selezionata dall'utente. Attraverso il metodo update si occupa di aggiornare il tempo di gioco, di popolare una lista contenenti i prossimi HitCircle da colpire, e di drenare la barra della vita. E' stato scelto di creare un'interfaccia updatable, questo perchè l' idea iniziale era quella di implementare il pattern **Update Method**, ovvero un metodo che ci permettesse di aggiornare in modo costante le entità che avevano bisogno di aggiornare il proprio stato in modo costante ad ogni frame di gioco. Questo è risultato utile solo per la barra della vita, ma è stato comunque lasciato in vista di una possibile espansione, come l' aggiunta di altre entità.

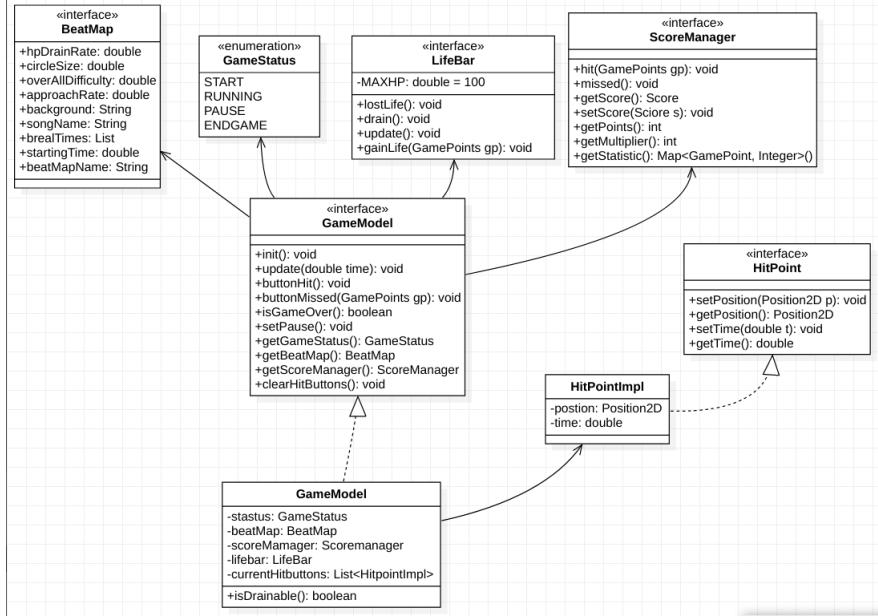


Figura 2.3: UML del GameModel

ScoreManager

Abbiamo scelto di implementare uno score Manager per operare indirettamente sul modello, così da tenere separati l' entità del punteggio dalla logica con cui esso verrà aggiornato. L'idea originale era quella di implementare lo *strategy pattern* in modo da poter creare diversi scoreManager ognuno con la propria logica di aggiornamento, è stato lasciato per poter permettere una futura estensione dell'applicativo.

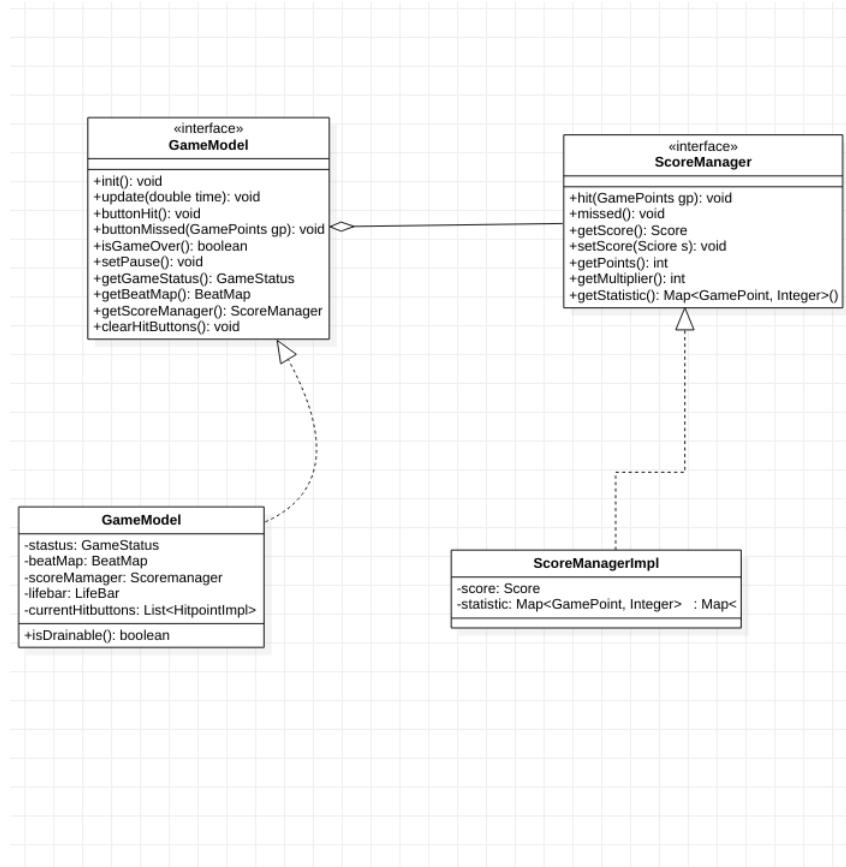


Figura 2.4: UML del ScoreManager

Factory HitPoint

Per visualizzare le concretizzazioni dei vari hitcircle su schermo ho deciso di utilizzare un Flyweight pattern, attraverso una factory di hitcircle. La factory di hitcircle viene istanziata una volta alla creazione della scena di gioco, inizializzando i parametri comuni a tutti gli hitcircle della beatmap in questione, così che i parametri comuni vengano passati e memorizzati una sola volta. Dopodichè per ogni hitcircle che dovrà essere visualizzato verrà chiamato il metodo `getHitcircleView()` che restituisce una istanza di `HitcircleViewImpl`, dalla quale vengono presi i due cerchi e la transizione relativa a questi. I cerchi vengono così aggiunti alla scena mentre la transizione viene aggiunta ad una lista in cui vengono mantenute per effettuare la possibile pausa del gioco. Attraverso la factory sarebbe anche possibile implementare diverse difficoltà della stessa beatmap senza andare a creare una beatmap per ogni difficoltà, semplicemente creando ulteriori metodi che restituiscono

la view dell' hitcircle, andandone ad alterare i parametri di un valore costante a seconda della difficoltà desiderata.

Gameloop/Controller

Al centro del Controller abbiamo la classe Controller. Questa viene istanziata ogni volta che viene eseguita una nuova canzone. Quando viene istanziata vengono settati i comandi di input e viene fatto partire il gameplay. Per realizzare il gameplay ho adoperato il pattern GameLoop. Per farlo ho deciso di usare un AnimationTimer, una particolare classe di JavaFx che permette di sincronizzarsi con il sistema di rendering proprio di javafx. Esso genera infatti un Pulse, evento che indica allo scene graph di sincronizzarsi con il motore grafico, 60 volte al secondo. Attraverso l' override del metodo handle è possibile eseguire codice ogni volta che viene ricevuto un Pulse. Anche se in media la pulsazione avviene ad un rate quasi costante, sono presenti spykes in cui il rate d' aggiornamento è superiore. Per tenere il gioco ad un rate fisso viene eseguita una sleep per rallentarne l' esecuzione, mantenendo il render ogni 16 ms circa. Attraverso questo rallentamento è garantito che il tempo di gioco scorra sempre alla stessa maniera anche in situazioni per cui l' hardware o aggiornamenti di javafx dovessero garantire maggiori prestazioni e maggiori rate d' aggiornamento. È stato deciso di rendere l' applicativo mono thread per via della sua forte natura interattiva con il giocatore. Il giocatore infatti opera in modo continuo con la view la quale deve essere in perfetto sincronismo col modello di gioco.

All'interno del gameloop viene eseguito l' update del modello e il render della view.

Observer

Per notificare il modello del gioco e il controller di effetti sonori relativi alla specifica beatmap in corso, ho deciso di utilizzare l' Observer Pattern. Ad ogni Hitcircle creato viene passata l' istanza del modello di gioco e del controller degli effetti sonori, che notificano ogni volta che il giocatore colpisce o manca il bersaglio. In questo modo non ho dovuto creare una nuova istanza di effetto sonoro per ogni hitcircle, è infatti il controller di effetti sonori che mantiene le istanze dei due possibili effetti sonori e si occupa di notificarli a seconda del punteggio effettuato. Per effettuare ciò ho leggermente alterato il pattern che non prevedeva un passaggio di parametri. Ho aggiunto il punteggio da notificare alle varie entità così che le entità poi possano alterare il proprio comportamento a seconda del punteggio notificato.

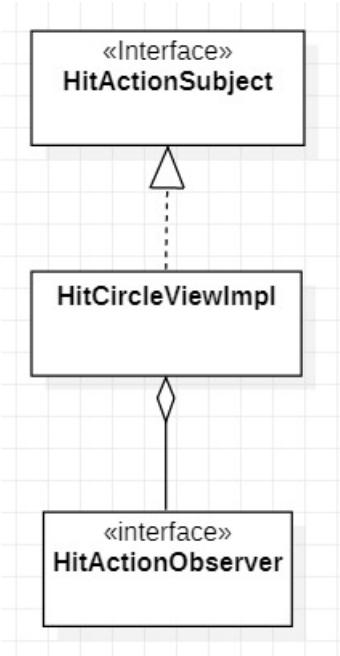


Figura 2.5: UML del HitCircleView

User

Ritenevamo fondamentale l'implementazione di una classe User in modo da poter gestirne i vari aspetti e preferenze. L'implementativo non prevede più utenti per una singola sessione quindi non abbiamo ritenuto fosse necessario la scrittura delle preferenze su file e abbiamo quindi adoperato una classe statica risolvendo il problema di dover mantenere molteplici preferenze accessibili da tutto l'applicativo. In una futura estensione sarebbe possibile scrivere le preferenze dei vari utenti su File così da non "perdere" le proprie preferenze una volta che si termina la sessione.

2.2.2 Matteo Santoro

HitCircle View

L' elemento grafico che andra ad identificare gli Hitpoint è costituito dalla classe HitcircleView. Questa classe è costituita di due Circle:

- cerchio esterno che rappresenta l' anello che si chiude verso il centro dell' Hitpoint
- cerchio interno che rappresenta la parte cliccabile dell' hitpoint.

Oltre ai due cerchi sono presenti vari parametri:

- velocità col quale dovrà chiudersi il cerchio esterno
- grandezza del cerchio
- valore generico che influenzera la difficoltà della partitâ.

Questi parametri vengono poi usati per creare le varie animazioni sui due cerchi. Le animazioni in particolare riguardano:

- comparsa dell' elemento su schermo (fade in transition).
- anello esterno che si stringe sul cerchio interno.

Nel caso il giocatore clicchi sull' hitcircle prima che il cerchio esterno si sia chiuso, verrà avviata un'altra animazione che scalerà il cerchio prima di farlo scomparire assegnando un punteggio positivo. Mentre se il cerchio si chiude prima che il giocatore ci clicchi sopra, verrà avviata un'animazione in cui il cerchio interno diventa nero per poi scomparire assegnando un punteggio negativo. Anche il tempismo col quale si clicca il cerchio determinerà una variazione sul punteggio.

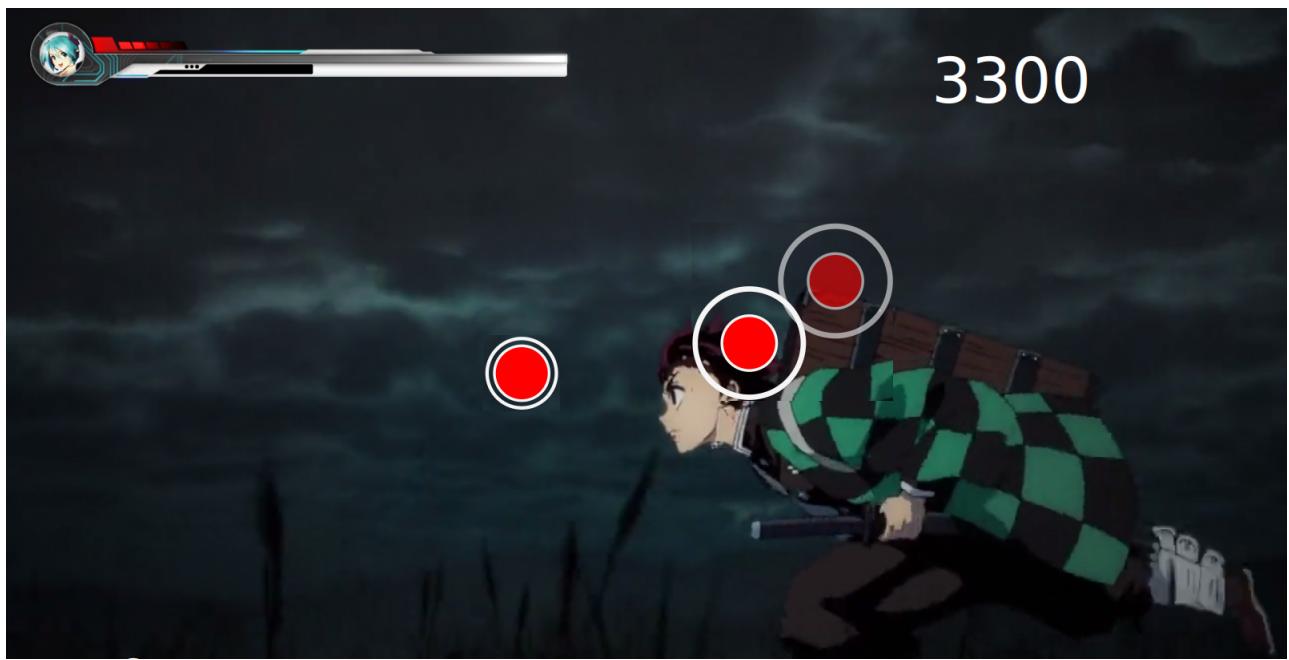


Figura 2.6: Screen del gioco in cui gli HitCircle appaiono

Statistic

Vogliamo che l'utente possa avere uno storico dei propri risultati in Uso! ma ci è sembrato estremamente inefficiente dover passare oggetti attraverso le varie classi per farlo e ciò avrebbe reso il codice poco mantenibile in quanto una modifica avrebbe influenzato varie interfacce. Abbiamo racchiuso perciò la logica delle statistiche in una sola classe che contiene un'istanza di sè stessa ed un `getInstance()` con il quale può essere chiamata in ogni parte del software, realizzando così il Singleton pattern. I punteggi vengono scritti in un file Json nel quale la chiave è il nick dell'utente e il valore è una lista dei punteggi eseguiti nel tempo.

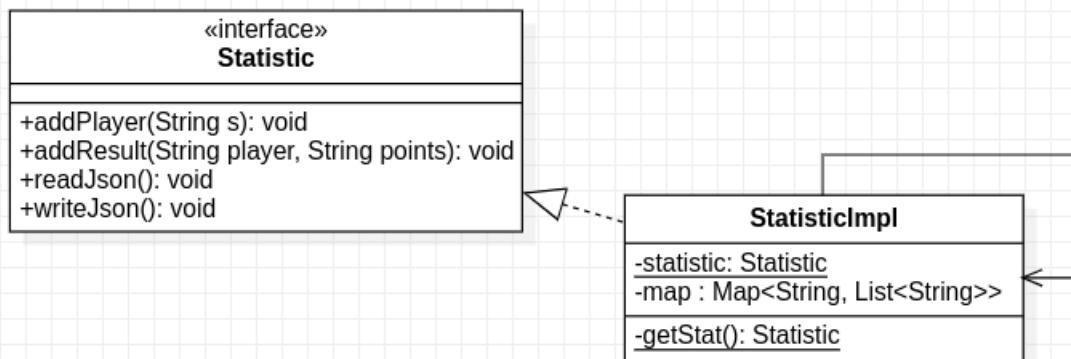


Figura 2.7: UML del pattern Singleton

MusicControllerFactory

Nell' applicativo avevamo bisogno di realizzare tre versioni di gestori musicali:

- una per musica semplice
- una per effetti sonori
- una per musica relativa alla canzone della beatmap da giocare.

Quindi abbiamo creato una factory statica attraverso la quale reperiamo la specifica estensione che ci serve. Visto che queste verranno istanziate solo attraverso la factory abbiamo deciso di rendere le classi innestate e private.

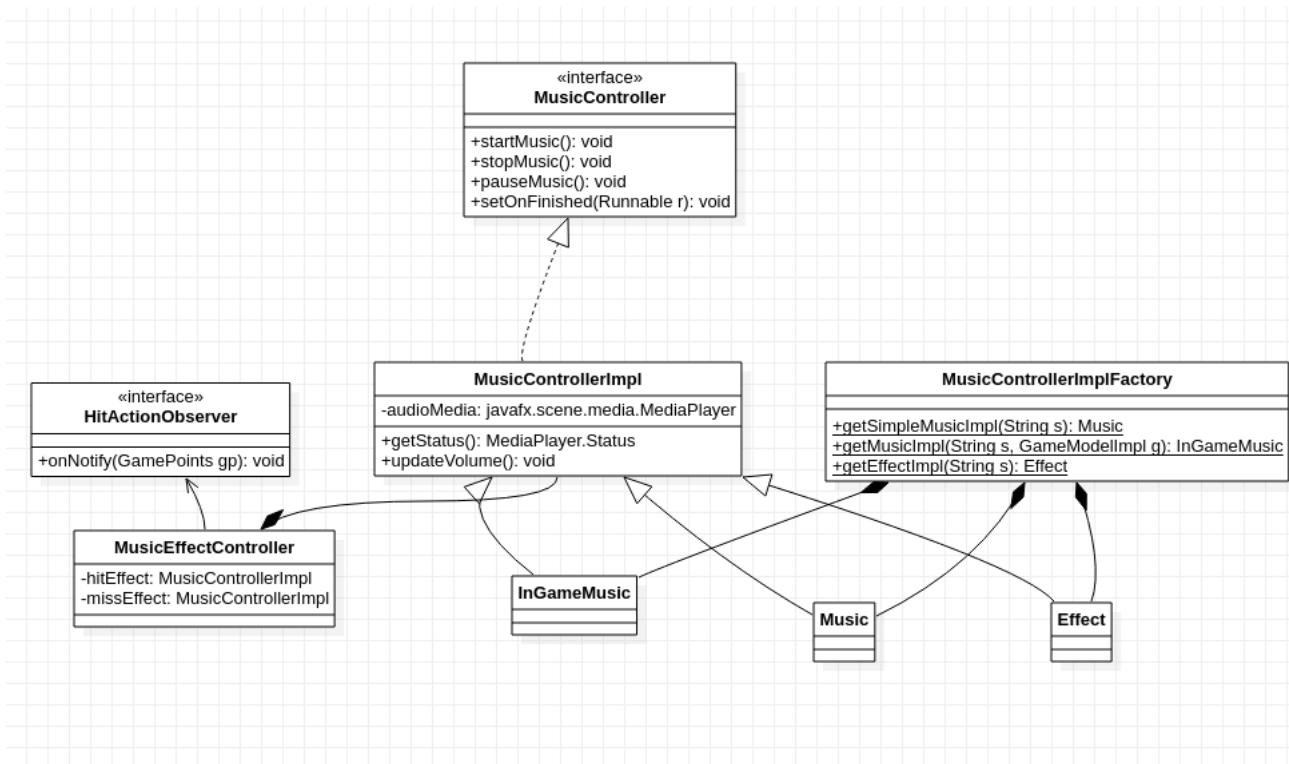


Figura 2.8: UML del MusicFactory

2.2.3 Shared Tasks

Gestione Scene e Scaling Risoluzione

Abbiamo deciso di creare la maggior parte degli elementi della view attraverso fxml, separando design da implementazione, infatti il comportamento di questi elementi è definito attraverso il relativo controller. Per effettuare i cambi di scena abbiamo deciso di implementare delle transizioni. Per farlo abbiamo usato una sola Scene con un nodo Root fisso per l' intera esecuzione. Quello che varia infatti durante l' esecuzione sono i figli del nodo Root, dando l' impressione che venga cambiata Scene. Per rendere tutto il più fluido possibile abbiamo usato varie Transition, classi di javaFx che definiscono un framework per realizzare animazioni. Per fare in modo che tutto l' applicativo fosse resizeable abbiamo prima creato l applicativo attenendoci ad un' unica risoluzione (1920x1080) per poi applicare degli Scale al bisogno, così che tutto fosse consistente. La scelta di non operare su Scene ma su Pane per creare le varie scene ci ha reso ancora più facile questo lavoro. Infatti applicando uno Scale ad un container, lo scale verrà applicato anche a tutti i nodi figli. Così facendo ci è bastato agire sul container più esterno, il fixed Pane che funge da Root all' intero applicativo, per poter scalare anche il container più interno contenente la scena, senza dover quindi dover cambiare risoluzione ad ogni scena.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Nell' applicativo è stato scelto di testare l' aspetto implementativo della classe BeatmapReader. Molti altri test invece sono stati effettuati in maniera visuale, come lo spawn degli hitcircle al momento opportuno, e tutte le questioni di tempistiche e coordinazione col sistema.

3.2 Metodologia di lavoro

Lo sviluppo di Uso! è stato preceduto da una fase di analisi all'interno della quale abbiamo estrapolato fin dall'inizio i concetti e le entità parte del gioco nelle parti del Model, View e Controller. Abbiamo cercato dove possibile di dividere equamente le parti di lavoro ma essendoci trovati in soli due componenti del gruppo è stato abbastanza difficile.

3.2.1 Gestione del DVCS

Per il DVCS abbiamo utilizzato git in quanto ne avessimo una certa familiarità entrambi e ne fosse stato spiegato il funzionamento a scuola, abbiamo creato diversi branch che si diversificavano per macro-argomenti sui quali lavoravamo in autonomia e che poi successivamente univamo.

3.2.2 Suddivisione del Lavoro

Inizialmente è stato creato il game loop del gioco. Una volta che il loop eseguiva correttamente abbiamo iniziato a sviluppare il modello, partendo dalle interfacce. Una volta che modello e controller sono stati integrati, e

ne abbiamo avuto conferma in maniera visuale, osservando la generazione sequenziale degli hitpoint a tempo con il loop, siamo passati a sviluppare la view, integrandola man mano con l' applicativo. Una volta giunti ad una prima versione funzionante del gioco, abbiamo proseguito aggiungendo dettagli fino ad arrivare ad una versione gratificante anche a livello estetico. In fine abbiamo conseguito una fase di alpha test in cui abbiamo eliminato i bug rilevati.

3.2.3 Manuel Luzietti

- Controller
- GameLoop
- HitActionObserver (interface)
- HitActionSubject (interface)
- MusicControllerImpl
- MusicEffectController
- MusicControllerImplFactory
- MusicControllerImpl
- ScoreManagerImpl
- BeatMapImpl
- GameModelImpl
- LifeBarImpl
- ScoreImpl
- BeatmapReader
- HitobjectSelector
- GameSceneController
- Resizeable
- SongButtonController
- HitecircleViewFactory

- Mapper
- Subject (interface)
- Observer
- User

3.2.4 Matteo Santoro

- MusicController (interface)
- MusicControllerImpl
- HitPointImpl
- Clock
- EndGameController
- HitcircleView (interface)
- HitcircleViewImpl
- NeonButton
- Statistic (interface)
- StatisticImpl

3.2.5 In collaborazione

- MusicController (interface)
- ScoreManager (interface)
- Beatmap (interface)
- GameModel (interface)
- HitPoint (interface)
- LifeBar (interface)
- Score (interface)
- LoginMenuController

- MainMenuController
- SongMenuController
- Updatable (interface)

3.3 Note di sviluppo

3.3.1 Manuel Luietti

Feature avanzate del linguaggio:

- **lambda** usate per settare tutti gli handler, e per eventuali istanze di oggetti con interfaccia funzionale.
- **Stream** tutte le volte che fosse necessario iterare su una collezione.
- **JavaFx a file fxml** per realizzazione di interfaccie grafiche

Liberie:

- **JavaFx**¹ libreria scelta per la grafica del progetto, utilizzate in tutte le classi della View.

Classi riutilizzate:

- **Pair** abbiamo riutilizzato la classe Pair{x,y} vista in classe del Prof. Viroli.

Note: per caricare dinamicamente dalle risorse le varie beatmap ho dovuto iterare su tutte gli elementi della cartella beatmaps nelle risorse. Questo ha evidenziato un problema nella creazione del jar. Ho dovuto quindi diversificare il procedimento di esecuzione da Ide da quello da Jar. Parte del codice per l' esecuzione da Jar è stato reperito da StackOverflow e poi riadattato alle mie esigenze. Il codice in questione istanziava un oggetto JarFile e poi iterava sulle entries, andando a cercare i file necessari. Le beatmap .osu, non sono state ideate da noi, sono state prelevate dal gioco originale e riadattate alle nostre esigenze, questo perchè l' idea originale era creare una copia fedele del gioco, infatti molti dei parametri usati (come velocità di chiusura del cerchio, ecc..) sono stati ricreati fedelmente. In più abbiamo ritenuto che ciò fosse un lavoro di puro design, che quindi non avrebbe comportato nessuna rilevanza a livello progettuale, focus del nostro progetto.

¹<https://openjfx.io/>

3.3.2 Matteo Santoro

Feature avanzate del linguaggio:

- **Stream** uso di stream per gestire comodamente le HashMap e aggiornarne i valori.
- **FXMLLoader** classe che carica un oggetto in java leggendolo da un file .fxml

Liberie:

- **Jackson**², libreria per la gestione di file Json, utilizzate in Statistic per poter leggere e scrivere i punteggi di un User ed avere uno storico.
- **JavaFx**³ libreria scelta per la grafica del progetto, utilizzate in tutte le classi della View.

Data la mia scarsa conoscenza di librerie esterne come Jackson o alcuni aspetti più avanzati di JavaFx ho utilizzato come supporto alcuni forum che mostravano come utilizzare alcune funzionalità delle librerie citate, principalmente ho preso degli snippet di codice per **Jackson-API** per i parser e la creazione delle JsonFactory poi ho usato **Stackoverflow** per la risoluzione di alcuni problemi legati alla libreria JavaFx e ad alcuni problemi di compatibilità con Linux.

Classi riutilizzate:

- **Pair** abbiamo riutilizzato la classe Pair{x,y} vista in classe del Prof. Viroli.

²<https://github.com/FasterXML/jackson-core>

³<https://openjfx.io/>

Capitolo 4

Commenti finali

4.0.1 Manuel Luzietti

Commenti

La parte più difficile del progetto a mio parere, è stato iniziare a sviluppare da zero. Non avevo assolutamente idea di dove partire. La parte di ricerca e progettazione iniziale prima di iniziare a scrivere qualsiasi cosa è durata molte ore. Una volta trovati i pattern strutturali su cui volevo basare il cuore dell' applicativo ho iniziato a scrivere e il livello di difficoltà percepito è sceso notevolmente. Mi sono serviti molto utili Gradle e JavaFx, cui cenni sono stati affrontati a lezione. Molto utili sono stati anche i cenni di linee di sviluppo, alle quali ci siamo ispirati, rispettandole in modo grossolano. Ritengo importante che tali cenni rimangano nel corso, e magari, che vengano approfonditi leggermente di più. Mi sarebbe piaciuto estendere ancora di più il progetto, aggiungendo gli Slider prima di tutto, feature che ritenevo importante per il gioco ma che a scapito di tempo non siamo riusciti a realizzare sebbene ci fosse già la struttura per farlo. Mi sono sforzato il più possibile per cercare di tenere il codice il più indipendente possibile, soprattutto per quanto riguarda la divisione di design da implementazione. Sicuramente rimane comunque molto migliorabile, ma mi ritengo soddisfatto del lavoro. é stato molto istruttivo realizzare un applicativo da 0, e soprattutto lavorare per la prima volta in team usando git.

4.0.2 Matteo Santoro

Commenti

Mi ritengo più che soddisfatto di questo progetto, è stata un esperienza in cui ho imparato moltissime cose che sicuramente non avrei approfondito, ho

trovato davvero utile l'uso di Git, del DVCS, non sapevo potesse essere così utile, avrei voluto avere ancora più tempo per poter implementare elementi che mancano come gli slider o schermate migliori, ho usato la classe Clock e creato una Map con il tempo e il punteggio, avrei voluto implementare un flowchart nella schermata finale.

Difficoltà

Questo progetto è stato difficile sotto molti punti di vista, stimare la quantità di lavoro all'inizio è stato praticamente impossibile, la realizzazione è stata più complicata del previsto dato che ci siamo trovati in soli due membri a dover svolgere l'intero applicativo. Ho avuto alcune difficoltà anche con lo sviluppo vero e proprio, ho lavorato utilizzando esclusivamente macchine Linux e i problemi con Eclipse non sono certo mancati, ma così facendo siamo abbastanza sicuri che questo software sia portaibile, nonostante tutto esistono alcuni errori che ho soltanto io che non riusciamo a spiegarci data la loro assenza sulle macchine Windows, in particolare un **com.sun.media.jfxmedia.MediaException: Could not create player!** che appare solo in alcune occasioni, inoltre la lettura di file audio come mp3 e immagini JPEG non ha funzionato sempre in modo corretto, anzi quasi mai bene.

Appendice A

Guida utente

All’apertura dell’applicazione apparirà una schermata a tutto schermo con un’icona. È possibile uscire dallo schermo intero in qualsiasi momento premendo il tasto Esc, facendo apparire il layout della finestra. Cliccando sul-



Figura A.1: Screen del Login

l’icona sarà possibile inserire il nickname per la sessione corrente, premendo invio una volta che è stato digitato. Alla pressione del tasto invio verrà avviata una transizione che porterà al Menù principale dove verranno visualizzati i pulsanti Play, Options, Exit.

Premendo sul tasto opzioni sarà possibile cambiare il volume degli effetti e della musica, sia del gameplay che del Menù principale, oltre che alla risoluzione (due sole disponibili, ma facilmente estendibile a qualsiasi risoluzione, in maniera semplice e immediata).

Cliccando sul pulsante Play, verrà avviata una transizione che porterà alla scelta della canzone da un Menù scrollabile.



Figura A.2: Screen del Menu



Figura A.3: Screen menu delle canzoni

Cliccando una volta sopra una canzone sarà possibile riprodurlne l’audio. Uscendo dal pulsante l’audio si interromperà. Cliccando due volte sopra il pulsante verrà fatta partire la schermata di gioco. Alla visualizzazione della schermata di gioco, vi è un lasso di tempo opzionale in cui la vita del giocatore non si esaurirà, questo lasso di tempo dipende dalla beatmap associata alla canzone. Al termine di questo la barra della vita inizierà a calare costantemente durante la canzone. Compito del giocate è colpire i cerchi che compariranno sullo schermo, nel modo più preciso possibile, come



Figura A.4: Screen menu delle canzoni

nel gioco originale è possibile colpire gli HitCircle sia con il click del mouse che semplicemente muovendo il cursore sopra e premendo a tempo i tasti **Z** e **X** della tastiera. I tempi col quale varia il tempismo col quale il giocatore dovrà cliccare varia da canzone a canzone a seconda della difficoltà di questa. Il giocatore dovrà quindi fare più punti possibili, ed evitare che la barra della vita scenda a zero. Sono possibili ulteriori momenti in cui la vita del giocatore non scenderà. È possibile mettere in pausa la canzone premendo spacebar. È possibile riprendere il gioco ripremendola. Al termine della canzone, o all'esaurirsi della vita, il gioco terminerà visualizzando una schermata con le statistiche relative alla partita appena effettuata. Premendo il tasto home è possibile tornare al menù principale per effettuare così un'altra partita.

Buon game!

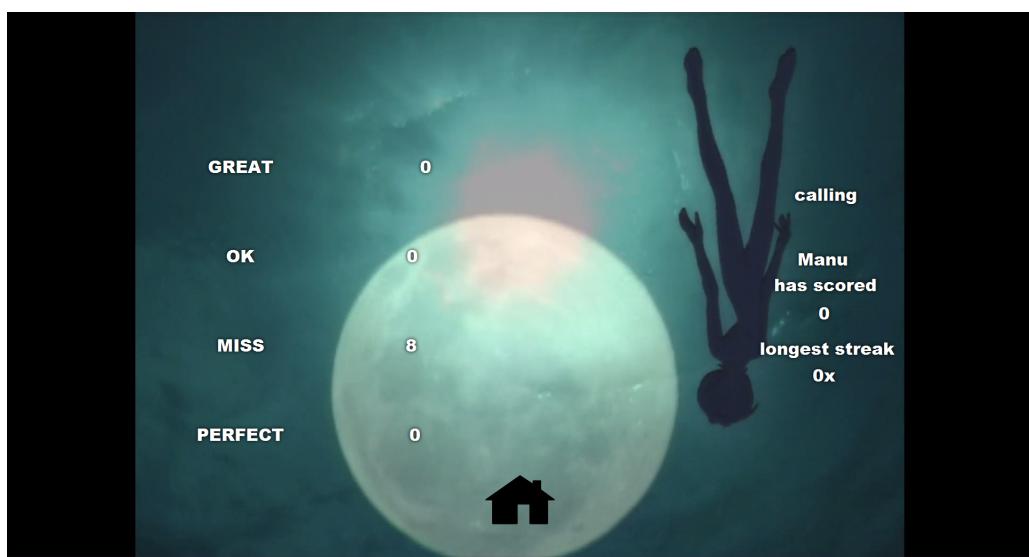


Figura A.5: Screen della schermata delle statistiche