1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

CORSO DI LAUREA IN
INFORMATICA APPLICATA
SCUOLA DI
SCIENZE TECNOLOGIE E FILOSOFIA DELL'INFORMAZIONE

Programmazione ad Oggetti ed Ingegneria del Software

Sessione autunnale 2020/2021

Matteo Pulcinelli
Matricola: 293473

Made with LaTeX

Indice

1 Specifica del problema

Si richiede di realizzare un software con interfaccia grafica per la gestione dei clienti. Il software in questione terrà traccia dei clienti e delle varie commissioni associate ad essi. Dei clienti ci interessano nome, cognome, il numero di telefono e email, mentre per ogni commissione ci interessano la data di scadenza e la descrizione.

Il software sarà composto da un menù principale con 4 tab

- **Home:** Questa è la schermata principale con cui si apre il software, all'interno troviamo la lista delle commissioni con scadenza in settimana e la possibilità di modificare e aggiungere commissioni.
- **Rubrica:** In questa schermata troviamo tutti i contatti salvati
- **Commissioni:** Qui possiamo visualizzare tutte le commissioni
- **Scadenze:** In quest'ultima tab verranno mostrate solo le commissioni non ancora completate

Una volta compilati opportunamente i campi della schermata di una nuova partita, verrà mostrata la schermata di gioco con la quale l'utente interagirà. Essa sarà composta da un rullante e due bacchette inizialmente alzate. Alla pressione del tasto **c** e **n** le bacchette (rispettivamente sinistra e destra) reagiranno all'input dell'utente colpendo il rullante. La sequenza di colpi da effettuare verrà rappresentata da immagini rappresentanti note che si muovono lungo due linee che congiungono i due angoli superiori dello schermo sino al rullante. Nel momento in cui il punto raggiunge il rullante e in contemporanea viene effettuato il colpo allora esso verrà conteggiato come corretto.

Il videogioco conterrà un semplice sistema di calcolo del punteggio totalizzato dall'utente. Tale punteggio sarà attribuito in base alla precisione del colpo.

Nel caso l'utente colpisca il rullante esattamente nell'istante in cui viene richiesto allora aumenterà il proprio punteggio di 100 punti (colpo perfetto). Nel caso in cui il colpo non sia precisamente nella posizione segnalata (colpo standard), ogni $2ms$ di ritardo o di anticipo (rispetto al colpo perfetto) risulterà in una penalità di 1 punto alla quantità che verrà aggiunta al punteggio totale. Nel caso l'utente colpisca il rullante quando il colpo non è ancora giunto sul rullante (colpo errato) non verrà attribuito alcun punteggio, dopo 20 errori di questo tipo la partita termina.

Durante la partita apparirà, in maniera casuale, al posto di un colpo standard un colpo speciale identificato diversamente dagli altri. Il punteggio assegnato nel caso in cui tale colpo venga eseguito in maniera corretta sarà doppio.

La difficoltà del gioco è determinata dal numero di BPM che si sta affrontando. Essi aumentano progressivamente ogni 5 colpi corretti (perfetti o standard) effettuati.

A fine partita il punteggio verrà mostrato all'utente e inserito in un file `csv` contenente tutti i punteggi effettuati con il gioco. Tali punteggi saranno visualizzabili a partire dalla schermata principale tramite un'apposita schermata raggiungibile premendo il tasto dedicato.

2 Specifica dei requisiti

2.1 Diagramma dei casi d'uso

2.2 Descrizione casi d'uso

Caso d'uso	Inizia partita
Id	1
Attori	Musicista
Pre-condizioni	N/A
Eventi base	L'utente avvia la partita cliccando sull'apposito pulsante
Post-condizioni	Viene mostrata la finestra di inserimento nome, inserimento BPM iniziali e selezione della modalità
Percorsi alternativi	N/A

Caso d'uso	Visualizza punteggi
Id	2
Attori	Musicista
Pre-condizioni	N/A
Eventi base	L'utente dopo aver cliccato sull'apposito pulsante, consulta i punteggi di gioco descritti dal punteggio numerico e dal nome dell'utente che ha totalizzato tale punteggio
Post-condizioni	N/A
Percorsi alternativi	N/A

Caso d'uso	Imposta nome, modalità e BPM iniziali
Id	3
Attori	Musicista
Pre-condizioni	L'utente deve aver avviato la partita
Eventi base	L'utente inserisce il proprio nome, i BPM iniziali e la modalità di gioco tra quelle proposte
Post-condizioni	Il gioco inizia
Percorsi alternativi	N/A

Caso d'uso	Effettua colpo
Id	4
Attori	Musicista
Pre-condizioni	La partita deve essere iniziata e devono essere state impostate le condizioni di gioco (nome, BPM iniziali e modalità)
Eventi base	L'utente effettua i colpi che appaiono sullo schermo tramite i due tasti impostati
Post-condizioni	N/A
Percorsi alternativi	N/A

Caso d'uso	Uscita dal gioco
Id	5
Attori	Musicista
Pre-condizioni	N/A
Eventi base	L'utente esce dal gioco premendo l'apposito pulsante proposto nel menù iniziale
Post-condizioni	Il gioco viene chiuso e il processo terminato
Percorsi alternativi	Abbandonare la partita in corso tramite il menù di pausa e poi uscire dalla partita con l'apposito pulsante

Caso d'uso	Metti in pausa partita
Id	6
Attori	Musicista
Pre-condizioni	La partita deve essere iniziata
Eventi base	L'utente, durante la partita, ha la possibilità di mettere in pausa il videogioco premendo il tasto *ESC* della tastiera. A quel punto avrà la possibilità di riprendere la partita oppure di abbandonarla, tornando al menù iniziale
Post-condizioni	Viene mostrato il menù di pausa del gioco
Percorsi alternativi	N/A

Caso d'uso	Riprendi partita
Id	7
Attori	Musicista
Pre-condizioni	Il gioco deve essere in pausa
Eventi base	L'utente, dopo aver messo in pausa la partita, premendo l'apposito tasto riprende la partita, che ricomincia nello stesso stato in cui era precedentemente
Post-condizioni	La partita riprende
Percorsi alternativi	N/A

Caso d'uso	Abbandona partita
Id	8
Attori	Musicista
Pre-condizioni	Il gioco deve essere in pausa
Eventi base	L'utente, dopo aver messo in pausa la partita, schiacciando l'apposito tasto abbandona la partita. In questo caso il punteggio della partita non viene salvato.
Post-condizioni	La partita termina e viene presentato il menù iniziale
Percorsi alternativi	N/A

3 Analisi e progettazione

3.1 Architettura

Per lo sviluppo del gioco è stato utilizzato il design pattern `MVC` che permette la separazione della logica del gioco dal codice che si occupa della presentazione all'utente.

Il pattern consiste in 3 componenti principali:

- **M**odel che implementa la logica di business del programma tramite le classi che espongono metodi utili ad accedere e manipolare i dati
- **V**iew che implementa l'interfaccia con cui l'utente interagirà. Ogni modifica che richiede l'intervento sui dati del `model` passerà attraverso il `controller`.
- **C**ontroller che si occupa di veicolare i messaggi inseriti dall'utente nella `view` per manipolare i dati contenuti nel `model`. Funge da tramite da `model` e `view` in quanto, sebbene la `view` sia in grado di leggere dati dal `model`, essa non è in grado di manipolarli.

Nell'architettura del progetto è stato fatto largo uso di quelle tecniche che contraddistinguono un linguaggio ad oggetti come `C#`: l'*incapsulamento*, il *polimorfismo*, l'*ereditarietà*, l'utilizzo di *eccezioni* e l'utilizzo di *classi generiche*.

3.1.1 Model

La struttura del `model` ruota interamente attorno alle *note* (ossi i colpi che deve effettuare l'utente), come esse vengono generate e il mantenimento dei punti quando esse vengono colpite.

Ogni nota estende la classe astratta `INote` così che, tramite il meccanismo di *upcasting* sia possibile riferirsi ad una generica classe che rappresenta una nota, la quale implementerà i metodi che la contraddistinguono. Ogni nota in particolare è dotata di un punteggio che viene assegnato all'utente qualora esso la colpisse in modo perfetto, un'immagine da mostrare a schermo all'utente e la posizione a schermo (*destra* o *sinistra*).

La sequenza di note che l'utente deve colpire viene generata da un *generatore di note*. Ogni generatore di note estende la classe astratta `INoteGenerator` che utilizza il design pattern `Observer`. Tale design pattern permette la comunicazione agli *observer* delle nuove note generate. La generazione di note

avviene tramite l'utilizzo del metodo astratto `NextNote` la cui implementazione viene delegata all'utente. Il metodo `NextNote` viene utilizzato da una routine interna che contiene il codice eseguito da un thread separato rispetto al flusso di esecuzione principale. Tale thread si occupa di generare le note con la giusta cadenza temporale, basandosi sui *BPM* raggiunti dall'utente.

Il punteggio dell'utente viene mantenuto nella classe `Game` che implementa l'interfaccia `IGame`. Tale classe espone un metodo utilizzato per comunicare che l'utente ha effettuato un colpo a vuoto ed un metodo per comunicare che l'utente ha colpito una certa nota con un determinato ritardo rispetto al colpo perfetto. Tale classe inoltre si occupa di calcolare il punteggio totalizzato dall'utente e di generare un eccezione di tipo `GameEndedException` nel caso l'utente giunga alla condizione di fine gioco (20 colpi effettuati troppo presto o troppo tardi). La classe `Game` si occupa inoltre di serializzare il punteggio dell'utente nell'apposito file `record.csv` e di deserializzare i record precedenti.

3.1.2 View

La struttura base della componente `View` è implementata interamente nella classe `MainView`. Tale classe, che implementa l'interfaccia `IMainView`, espone i metodi necessari per mostrare all'utente le schermate che compongono il gioco.

Ogni schermata di gioco è implementata tramite la classe `Panel` fornita dal framework `WinForms` e si occupa di gestire in modo autonomo i propri elementi grafici ed il proprio stato. Nel caso sia necessario, tale schermata può comunicare con `MainView` contenendone un riferimento e, nel caso di `PlayingPanel` anche con `MainController`.

Particolare enfasi va posta sulla classe `PlayingPanel`: tale classe infatti si occupa di implementare in modo vero e proprio la schermata di gioco vista dall'utente. Essa, implementando l'interfaccia `IObserver`, utilizza in modo diretto la classe `INoteGenerator` e mostra le nuove note generate all'utente inserendole in una lista generica di tipo `LinkedList` cosicchè sia possibile accedere all'elemento di testa e di coda.

Nel momento in cui un tasto viene premuto comunica in modo diretto a `MainController` dell'avvenuto colpimento di una nota o del colpo a vuoto.

Tale schermata inoltre mostra il punteggio raggiunto dall'utente, i colpi che l'utente può mancare prima che il gioco termini ed i BPM raggiunto.

3.1.3 Controller

La funzione di `controller` viene implementata dalla classe `MainController` che implementa l'interfaccia `IController`. Tale classe espone tutti i metodi necessari per conoscere le informazioni contenute nella classe `Game` ed interagire con i metodi presenti nell'interfaccia `IGame`.

3.1.4 Utils ed Eccezioni

Durante lo sviluppo del gioco si sono rese necessarie lo sviluppo di classi che svolgono funzioni che non sono direttamente imputabili a nessuna delle componenti del pattern `MVC`. Tali classi sono quelle necessarie allo sviluppo del pattern `Obervable`, la rotazione di immagini, l'utilizzo di una tupla di 3 elementi editabile ecc.

Discorso del tutto analogo per le eccezioni che sono state create *ad-hoc*.

3.1.5 Versioning

Lavorando in un team di 2 persone si è reso necessario l'utilizzo di un tool di versioning per permettere la sincronizzazione del codice ed evitare problemi di conflitti.

Si è fatto quindi largo uso del software di versioning `git` e della piattaforma github soprattutto grazie all'eccellente integrazione in *Visual Studio* tramite l'apposita estensione. L'intero software è reperibile all'indirizzo n28div/masterdrums.

3.1.6 Librerie esterne

Durante l'implementazione del gioco è stata utilizzata la libreria NAudio per riprodurre il suono del colpo sul rullante. Tale libreria è opensource ed installabile tramite `nuget`.

3.2 Diagramma di robustezza

Per illustrare il meccanismo di interazione tra i vari elementi del software, viene illustrato un diagramma di robustezza.

All'avvio del gioco l'utente potrà eseguire una delle seguenti operazioni:

- Visualizzare i record totalizzati dagli utenti nel gioco
- Chiudere il gioco
- Iniziare una nuova partita

Nel caso in cui l'utente decida di iniziare una nuova partita, dopo aver opportunamente inserito il nome, i BPM iniziali e scelto la modalità di gioco, verrà mostrata la schermata di gioco effettiva. Da tale schermata di gioco, durante l'esecuzione della partita premendo il tasto `ESC` della tastiera, l'utente avrà la possibilità di mettere in pausa la partita e di conseguenza di riprendere la partita o di abbandonarla, tornando al menù iniziale.

Se l'utente non abbandona la partita, ma essa termina tramite la condizione di fine gioco, il punteggio totalizzato viene memorizzato nell'apposito file `.csv`.

Nel caso in cui l'utente voglia visualizzare i record del gioco, verrà mostrata una schermata contenente i punteggi totalizzati dagli utenti, nel formato `Nome Punteggio`. Nel caso in cui non sia ancora stata effettuata la prima partita del gioco, la schermata non verrà mostrata e l'utente verrà opportunamente avvisato della mancanza di risultati.

3.3 Diagramma UML delle classi

3.4 Model

Nell'immagine posta sopra, viene descritta l'architettura mediante schema UML della parte riguardante le note e il generatore di note. Come visto nella specifica del problema, le modalità proposte dal gioco sono due, e dipendono dal modo in cui vengono generate le note. Nella modalità *combinazioni casuali* le note vengono generate in maniera puramente casuale, senza nessun controllo sulla loro posizione, mentre nella modalità *mani alternate* vengono generate note con posizione (destra e sinistra) alternate tra loro. Questo avviene grazie ai due generatori di note descritti nell'immagine sopra riportata: entrambi i generatori di note implementano la classe astratta `INoteGenerator`, ed eseguono l'override dei due metodi `ToString()` e `NextNote()`. Molta attenzione va posta in particolare nel metodo `NextNote()`, in quanto nella modalità *mani alternate* la prossima nota dovrà per forza essere nella posizione opposta rispetto all'ultima nota generata.

É importante inoltre notare come `INoteGenerator` estende la classe astratta `ISubject`, necessaria per rendere il generatore di note come l'oggetto osservato dai vari observer, come descritto nell'architettura del design pattern `Observer`.

Per quanto riguarda le note invece, si osservi come esse risultino tutte derivanti dalla classe astratta `INote`, della quale eseguono l'override dei metodi `HitPoint()` e `Image()`. Ciascuna tipologia di nota avrà infatti una sua caratterizzazione per quello che riguarda il punteggio e l'immagine che la rappresenterà nel pannello di gioco.

La classe `Game` è la classe che si occupa di mantenere il punteggio dell'utente, di aumentare i BPM e di salvare il punteggio nel file `csv`. Come si può notare dal diagramma UML tale classe implementa l'interfaccia `IGame` e ne effettua l'*override* dei metodi `Hit()`, che rappresenta il metodo che comunica al gioco un colpo vuoto, `Hit(note, deltaT)` che rappresenta il colpo andato a segno sulla nota `note` dopo un tempo di `deltaT` *ms* dal tempo di colpo perfetto ed infine il metodo `SerializeScore` che si occupa di salvare il punteggio nel file.

La classe `Game` inoltre espone il metodo statico `LoadBestResults` che si occupa di leggere dal file `csv` i punteggi effettuati precedentemente.

3.5 Controller

La struttura del controller è piuttosto semplice: la classe concreta `MainController` implementa l'interfaccia `IController` che espone metodi utili ad interagire con le classi che implementano l'interfaccia `IGame`. La classe `MainController` contiene inoltre al suo interno un riferimento alla *view principale* (`MainView`) così da poter comunicare di reagire a determinati eventi.

3.6 View

La componente *view* del progetto è principalmente gestita dalla classe concreta `MainView` che implementa l'interfaccia `IMainView` e funge da *"controller"* per le altre viste, implementate sotto forma di `Panel`. La classe `MainView` espone metodi per attivare o nascondere le viste che compongono il gioco (menù iniziale, schermata di pausa ecc.) e contiene metodi privati per creare ed inizializzare tali viste di modo che le dimensioni di esse siano dipendenti dalla dimensione dello schermo dell'utente (il gioco viene presentato a schermo intero).

Particolare enfasi va posta sulla classe concreta `PlayingPanel`: tale classe infatti rappresenta un *Observer* della classe `INoteGeneretor`. Al suo interno inoltre è presente il metodo `PaintObjects(object sender, PaintEventArgs e)` che viene chiamato periodicamente e si occupa di disegnare a schermo gli *sprite* che compongono il gioco. Il pannello di gioco infatti gestisce tutti gli elementi grafici disegnandoli in una `PictureBox` che copre l'intero schermo dell'utente. Così facendo è possibile gestire in maniere semplice ed efficiente la sovrapposizione di oggetti sullo schermo.

Di particolare importanza è anche il metodo `DrawNotes` che tramite trasformazioni geometriche e trigonometriche si occupa di disegnare a schermo le note e di calcolare la loro posizione futura.

4 Implementazione

4.1 Model

4.1.1 INote.cs

4.1.2 StandardNote.cs

4.1.3 SpecialNote.cs

4.1.4 PauseNote.cs

4.1.5 INoteGenerator.cs

4.1.6 AlternatedHandGenerator.cs

4.1.7 RandomNoteGenerator.cs

4.1.8 IGame.cs

4.1.9 Game.cs

4.2 Controller

4.2.1 IController.cs

4.2.2 MainController.cs

4.3 View

4.3.1 IMainView.cs

4.3.2 MainView.cs

4.3.3 IPlayingView.cs

4.3.4 PlayingPanel.cs

4.3.5 GamePausePanel.cs

4.3.6 HighscoresPanel.cs

4.3.7 MainMenuPanel.cs

4.3.8 NewGamePanel.cs

4.4 Exception

4.4.1 GameEndedException.cs

4.4.2 GameOptionException.cs

4.5 Utils

4.5.1 IObserver.cs

4.5.2 ISubject.cs

4.5.3 ImageUtils.cs

4.5.4 Triplet.cs

4.6 Program

4.6.1 Program.cs

5 Testing

Lo sviluppo del software è stato per gran parte effettuato utilizzando la tecnica del *pair-programming*. Tale tecnica ci ha permesso di analizzare ogni scelta implementativa a fondo senza effettuare nessuna scelta in modo approssimativo; il risultato è un software la cui probabilità di presenza di errori è piuttosto bassa.

Il testing di tale software è stato effettuato inizialmente in modalità `whitebox` durante la fase di sviluppo, testando ogni condizione limite che si sarebbe potuta presentare. Ciò ha permesso di arginare in fase iniziale la maggior parte degli errori che sarebbero potuti emergere a lavoro completato.

In seguito il software, giunti a quella che è la versione finale, è stato testato con le modalità `blackbox`. Nella sottosezioni seguenti verranno mostrati vari *screenshot* del software corrispondenti ai vari casi d'uso identificati nella sezione corrispondente.

5.1 Inizio di una partita

Cliccando sul pulsante *Nuova partita* viene mostrata la schermata seguente

5.2 Visualizzazione punteggi

Nel caso l'utente abbia già giocato almeno una partita vengono mostrati i punteggi totalizzati.

Nel caso invece l'utente non abbia ancora giocato nessuna partita viene mostrato un messaggio che comunica l'assenza di punteggi registrati.

5.3 Impostazione di nome, modalità di gioco e BPM iniziali

Nel caso l'utente tenti di procedere senza aver inserito il proprio nome la partita non viene iniziata.

5.4 Effettuazione colpo

5.5 Fine della partita

5.6 Partita in pausa

5.7 Partita ripresa da una pausa

5.8 Partita abbandonata da una pausa

6 Compilazione ed esecuzione

Gli strumenti utilizzati per la compilazione del programma sono i seguenti:

- **Ambiente di sviluppo**: Visual Studio Community 2017
- **Versione**: 15.9.28307.905
- **Framework**: .NET Framework 4.5.2

Per la compilazione della soluzione, è necessario recarsi nel seguente path dell'ambiente di sviluppo: nella barra dei menù selezionare *Compila* > *Compila soluzione*. In alternativa, se non si compila manualmente la soluzione, all'avvio del programma (tramite apposita icona su Visual Studio), la compilazione avviene in maniera automatica.

Per eseguire l'applicazione si può utilizzar l'ambiente di sviluppo oppure ricorrere all'eseguibile presente nella cartella del progetto al percorso `./MasterDrums/MasterDrums/bin/Debug/MasterDrums.exe`.

I requisiti minimi per per l'esecuzione del programma sono i seguenti:

- Sistema operativo: Windows 7 o successivi
- Architettura: 32 o 64 bit
- Framework: .NET Framework 4.0

Non vi sono requisiti di performance particolari, tuttavia si rimanda a consultare i requisiti minimi per il .NET Framework al sito Microsoft tenendo in considerazione che il programma utilizza al più 60 MB di memoria RAM.

Il software è stato testato su un computer con la seguente scheda tecnica:

- CPU: Intel(R) Core(TM) i5-8250U CPU @ 1.80 GHz
- RAM: 8GB DDR3

- GPU: Intel UHD Graphics 620
- SO: Windows 10 Home, Build 1903
- Architettura: 64 bit