Relazione:
"DUKEMANIA"

Gianluca Migliarini,
Serafino Pandolfini,
Sofia Tosi,
Laura Leonardi,
Alessandro Brasini

25 Agosto 2021

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il gruppo si pone come obbiettivo quello di realizzare un clone del videogioco musicale "BeatManiaIIDX". Lo sviluppo mira alla completa funzionalità a livello di gameplay, e, in aggiunta all'opera originale dalla quale viene presa ispirazione, offrire all'utente la possibilità di importare i propri livelli giocabili tramite file Midi (Musical Instrument Digital Interface). Viene inoltre considerato come requisito la sostituzione della riproduzione di una traccia audio con la sintesi sonora in tempo reale.

Requisiti funzionali

- L'utente è in grado di scegliere quale canzone giocare a partire dalla scelta di un file Midi.
- L'applicativo deve riprodurre fedelmente tutte le tracce componenti la canzone, con un timbro musicale simile a quello di un GameBoy.
- L'applicativo deve offrire un gameplay su varie colonne, il giocatore dovrà premere le note sulla rispettiva colonna a ritmo di musica per ottenere punti.
- L'applicativo deve memorizzare una scoreboard con tutti i punteggi delle varie partite relative alla canzone selezionata.
- La traccia giocabile viene scelta dall'utente dalla lista delle tracce componenti la canzone.
- L'applicativo deve "semplificare" le traccie musicali complesse rendendole giocabili, segnalando un grado di difficoltà

Requisiti non funzionali

- L'applicativo deve funzionare correttamente sia su Windows sia su sistemi basati su Unix
- La fluidità del gameplay e la qualità dell'audio non devono essere compromesse dalla pesantezza del file midi (ovviamente se non troppo complesso)

1.2 Analisi e modello del dominio

Dukemania dovrà essere in grado di fare selezionare all'utente il proprio nickname e una canzone in formato Midi. Dovrà inoltre permettere la configurazione dei nomi di tracce e strumenti associati alla canzone selezionata. Inoltre, la traccia da giocare, sara' anch'essa selezionabile dall'utente. I vari strumenti saranno caricati a partire da un file di configurazione in locale all'avvio del gioco e saranno detti sintetizzatori. Ci sarà una entità responsabile della gestione e riproduzione del tono e del volume in entrata e in uscita. Le tracce della canzone, con le relative note, verranno individuate da un parser. A ogni traccia verrà associato un solo sintetizzatore che riprodurrà uno strumento. Il gameplay si svolgerà su varie colonne, in ognuna delle quali cadranno note relative alla traccia selezionata, opportunamente semplificata per garantirne la giocabilità. Le difficoltà primarie riguarderanno il calcolo del punteggio in base alla precisione del giocatore e la sincronizzazione tra la caduta delle note e il suono riprodotto. Alla fine della partita, dovrà essere visualizzata una schermata contenente il punteggio migliore di ogni singolo giocatore ottenuto sulla canzone selezionata in precedenza.

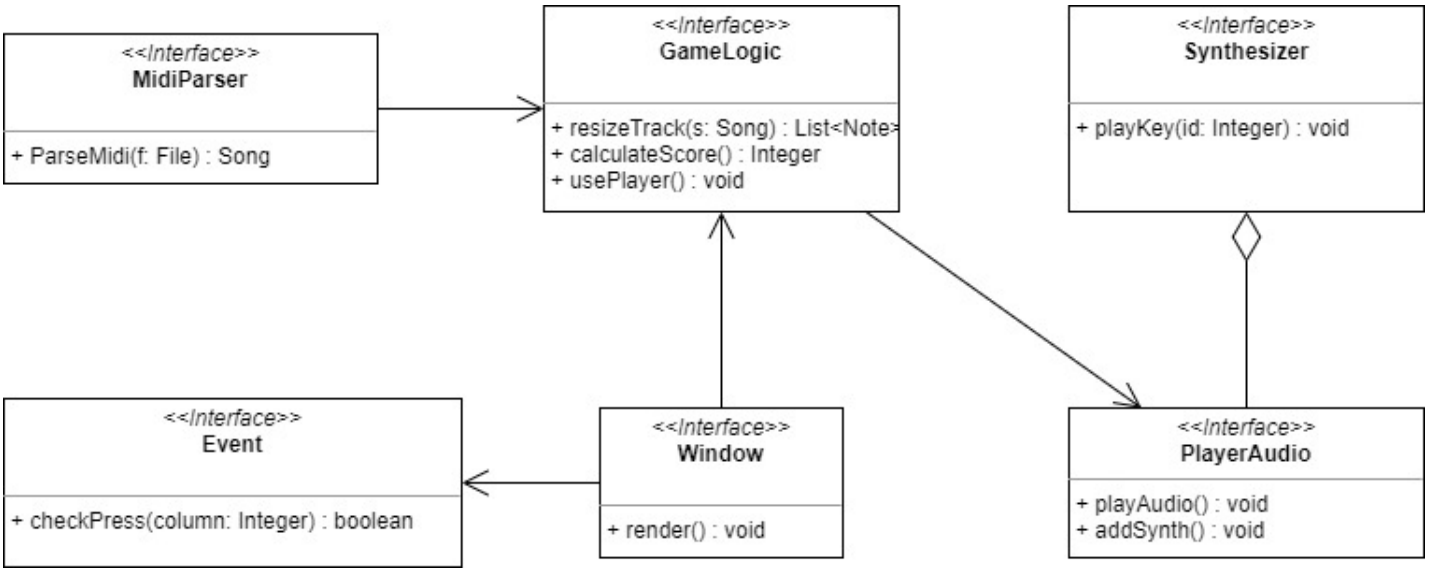

Figura 1.1: Schema UML della fase di analisi.

Capitolo 2

Design

2.1 Architettura

L'architettura di DukeMania è basata sul pattern MVC. La classe DukeMania è l'entry point dell'applicazione, al suo interno vengono inizializzate tutte le View e il WindowManager, la classe che fungerà da Controller principale. Il compito pricipale di WindowManager è quello di cambiare, tra le varie View, la finestra da renderizzare e visualizzare, occupandosi inoltre di passare il Model alle varie schermate, che a loro volta lo passeranno al relativo Controller. Il Model principale, rappresentato da GameModel, conterrà le configurazioni di gioco, rendendole così accessibili a tutte le View. La classe WindowManager implementa l'interfaccia SwitchWindowNotifier, il cui compito principale e' quello di notificare il WindowManager, segnalandogli che un controller di una View vuole sostituire la finestra attiva. Tutte le View implementano l'interfaccia Window, la quale le rende predisposte a ricevere i dati del Model, e ad utilizzare lo SwitchWindowNotifier tramite il proprio controller. Utilizzando questa strategia è semplice aggiungere nuove View e definirne il comportamento. Un sistema di window-switching è già implementato in LibGDX ma, al fine di rendere il più riutilizzabile possibile e platform-indipendent questa meccanica, è stato deciso di implementare una versione generale che può essere applicata a qualsiasi tipo di libreria grafica.

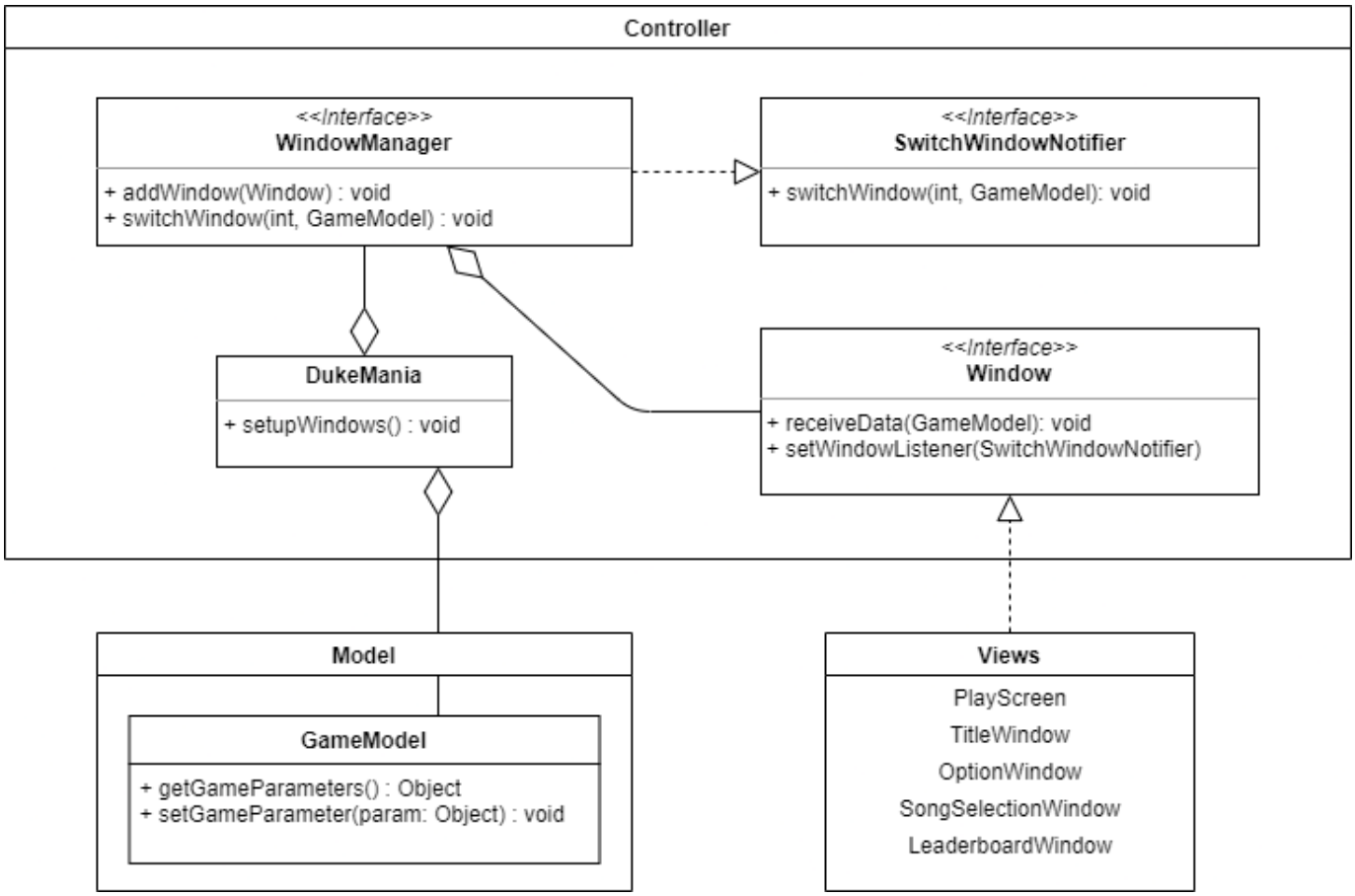

Figura 2.1: Schema UML dell'architettura MVC.

2.2 Design dettagliato

2.2.1 Gianluca Migliarini

La classe Engine si occupa di aggiornare e valutare lo stato dei vari sintetizzatori (tastiere simulate), ad ogni chiamata del metodo render viene riempito un buffer di samples (campioni audio). Ogni campione audio viene calcolato sommando i campioni audio di tutti i sintetizzatori al momento della chiamata. Una volta che il buffer e' pieno, viene mandato al dispositivo audio che si occupa di suonare i vari campioni. Ogni traccia della canzone viene affidata ad un sintetizzatore di tastiera o di percussioni. Per differenziarli, e' stata creata l' interfaccia Synth, contenente i metodi comuni tra i due, per calcolare quante note/percussioni stanno suonando e per ottenere un campione audio. Le variabili di impostazioni che garantiscono il corretto funzionamento dell'audio sono scritte nella classe Settings.java, come campi finali statici. Per evitare costi computazionali troppo alti, i vari campioni delle varie note di tutti i sintetizzatori vengono pre-caricati in istanze anonime dell'interfaccia BufferManager. Lo scopo di quest'ultima e' quello di rendere una nota risuonabile in ogni momento tramite il metodo refresh, e, tramite i meotodi di iterator, ottenere il prossimo campione suonabile e controllarne l'esistenza.

I sintetizzatori di tastiera e di percussioni si differenziano nel seguente modo:

KeyboardSynth

Per i vari sintetizzatori di tastiere, oltre ai metodi dell'interaccia Synth, e' presente il metodo playTimeNote, per suonare una nota ad una determinata frequenza per un determinato tempo espresso in microsecondi.
Per aumentare le performance, i vari campioni delle note sono caricati all'instanziamento della classe, a partire da una lista contenente gli indici delle note utilizzate e il relativo tempo di suonata.
I BufferManager vengono creati a partire da una classe Enveloper, che si occupa di gestire il volume della nota in entrata e in uscita. Per impostare correttamente tutti i parametri dei sintetizzatori di tastiera ho deciso di utilizzare il pattern "builder", in quanto alcune impostazioni non sono sempre necessarie, ad esempio gli LFO, visti in dettaglio piu' avanti.

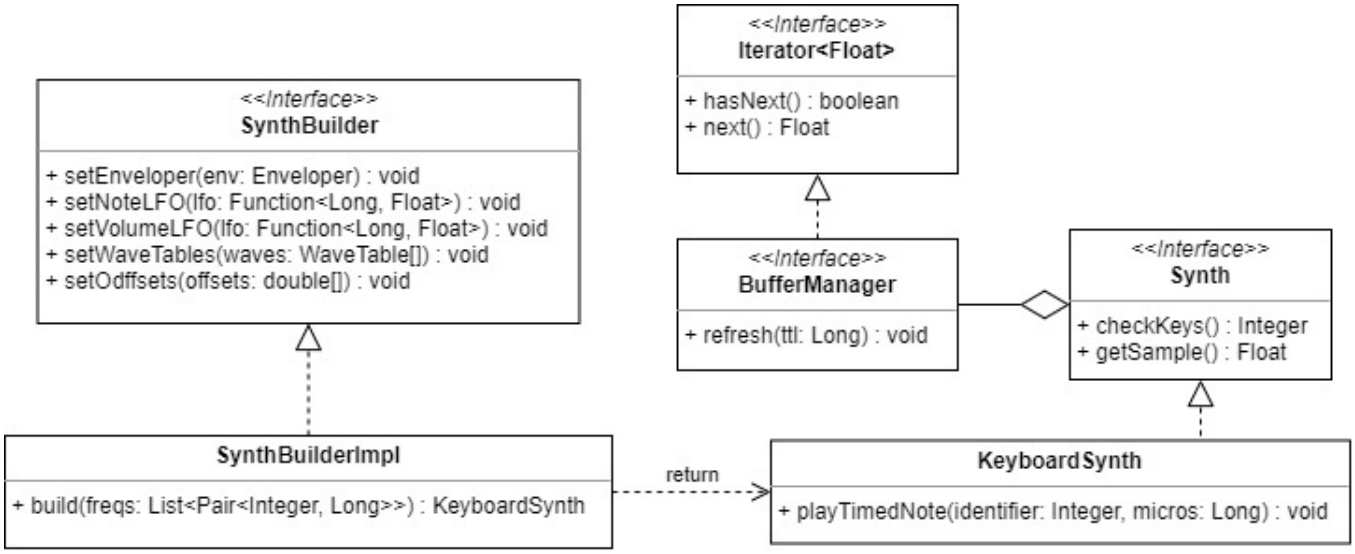

Figura 2.2: Schema UML del Builder dei sintetizzatori di tastiere.

DrumSynth

Nel caso del sintetizzatore per le percussioni, le varie istanze di BufferManager sono presenti come campo read only dell'enum DrumSamples. Ho scelto di utilizzare questo enum come pattern "singleton" in quanto non richiede l' istanziazione di oggetto, e perchè le percussioni non cambiano tra i diversi file Midi, a differenza delle note le quali hanno lunghezze diverse. L'unico metodo esterno all'interfaccia Synth e' playPercussion, il quale si occupa di ri-iniziare la riproduzione di un elemento della batteria.

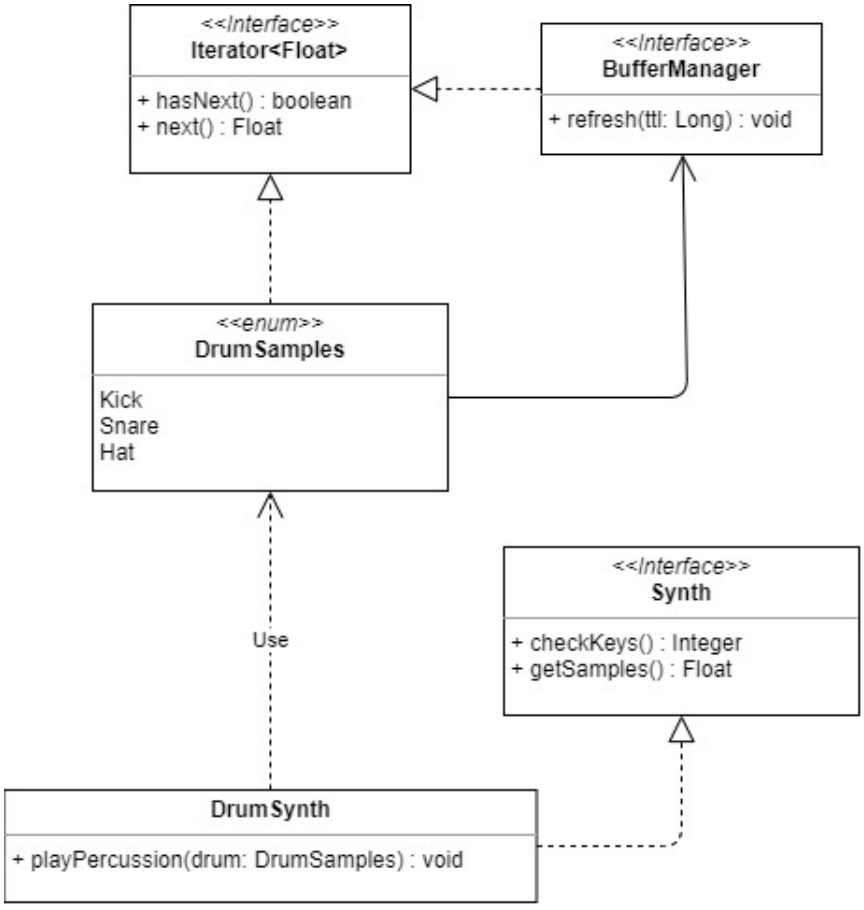

Figura 2.3: Schema UML del Singleton Drum Samples.

Creazione delle note e timbri sonori

Durante il caricamento, i sintetizzatori utilizzano l'enum Wavetables, contenente le varie forme audio utilizzabili assumibili dalle note; queste forme sono caricate all'avvio in array, i quali possono essere letti in intervalli più o meno distaccati per simulare le varie frequenze. Per differenziare i timbri sonori faccio uso degli LFO, funzioni adibite al cambio del tono e del volume nel tempo. Ho deciso di utilizzare il pattern "static factory" a causa della necessità di essere utilizzata senza istanze. Per aggiungere funzioni LFO in futuro, e avere ancora più varietà del suono, SynthBuilder utilizza una strategy con "Function Long,Float" come interfaccia funzionale, la funzione che rappresenta l'LFO.

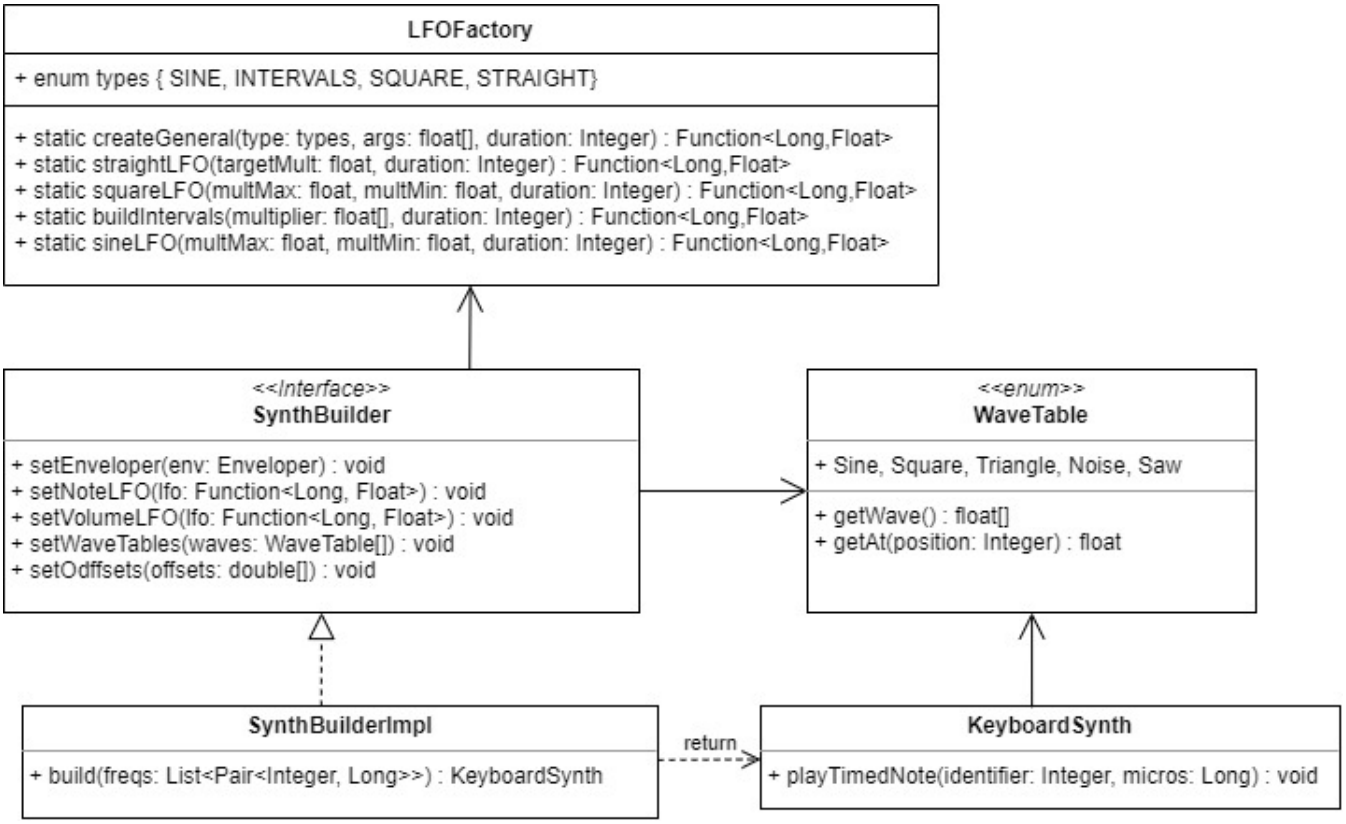

Figura 2.4: Schema UML dell'interazione tra LFOFactory, keyboardSynth e Wavetable.

2.2.2 membro 2

progetto

2.2.3 Sofia Tosi

Questo paragrafo illustra il design relativo alla gestione dell'input e degli assets. In seguito ad un'attenta valutazione, il gruppo ha stabilito di far interfacciare l'utente all'applicazione DukeMania tramite l'utilizzo di una tastiera per pc. Durante la progettazione della sezione concernente l'input, ho ritenuto oppurtuno utilizzare il pattern Adapter. La classe EventsFromKeyboardImpl rappresenta l'Adapter in sè che, implementando l'interfaccia omonima, riesce a semplificare e a permettere l'interazione tra la tastiera, che sfrutta la libreria com.badlogic.gdx.Input, e la classe relativa alla grafica PlayScreen. Per non superare eccessivamente l'ammontare di ore, l'applicazione attuale è stata implementata per consentire solo l'utilizzo della tastiera del pc. Laddove si volesse, in futuro, cambiare la periferica, l'applicazione non esclude questa possibilità. Si potrebbe infatti implementare EventsFromKeyboard e sostituire la classe dell'adapter con un altro adapter che interagisca con una diversa periferica (ad esempio un gamepad, una SG Guitar Hero o una tastiera midi).

singleton
Un'altra peculiare classe che merita di essere citata è l'AssetManager che, come dice il nome si dedica all'organizzazione degli assets, ovvero tutte le risorse grafiche necessarie per il corretto funzionamento dell'applicazione. Queste risorse, che possono essere, ad esempio, le texture degli elementi nella GUI o i file con i font utilizzati, vengono caricate una sola volta in memoria. Ho deciso di sfruttare il pattern Singleton per la realizzazione di tale classe. Difatti è importante che ci sia una sola istanza dell'AssetManager per evitare che le risorse grafiche vengano istanziate più volte in memoria sprecando spazio inutilmente.

2.2.4 membro 4

progetto

2.2.5 membro 5

progetto

Capitolo 3

Sviluppo

3.1 Testing

test

Le classi che sono state testate sono:

- **LFO:** per ogni funzione LFO è presente un test per garantire il corretto funzionamento della funzione di ritorno della factory
- **KeyboardSynth:** testato la correttezza del conto di quante note stanno suonando al momento, il metodo per ottenere il campione audio, il metodo per risuonare una nota.
- **DrumSynth:** testato la correttezza del conto di quante percussioni stanno suonando al momento, il metodo per ottenere il campione audio, il metodo per risuonare una percussione.
- **SynthBuilder:** testate le eccezioni

3.2 Metodologia di lavoro

3.2.1 Gestione del lavoro

Lo sviluppo del progetto è stato preceduto da una fase preliminare di analisi, durante la quale tutti insieme abbiamo lavorato per definire le basi dell'applicazione e, successivamente, per specificare i vari dettagli implementativi tramite UML. La suddivisione delle parti del progetto è avvenuta in modo equo assegnando a ciascuno una parte significativa e importante del progetto.

Gestione del DVCS

Per il DVCS è stato utilizzato git, in quanto è stato il Software spiegato a lezione. Abbiamo creato una repository su github e abbiamo deciso di creare un branch per ogni membro del team. In seguito, quando ritenuto che ognuno di noi avesse implementato il "core" della propria parte, abbiamo deciso di eseguire un merge in un nuovo branch "alpha", sul quale ogni membro ha continuato a salvare le proprie modifiche e sul quale sono stati effettuati i primi test di giocabilità.

Suddivisione del lavoro

3.2.2 Gianluca Migliarini

In questo progetto, mi sono dedicato principalmente della riproduzione audio. Gli oggetti principali del mio package sono i Synth, i quali sono in grado, alla chiamata di un metodo, di restituire un campione audio ottenuto dalla somma di tutte le note o percussioni che stanno suonando simultaneamente. La classe Engine, si occupa di ottenere i campioni dai vari sintetizzatori, comporre il buffer audio, regolarne il volume e riprodurlo in uscita. Mi sono inoltre dedicato allo sviluppo di delle classi LFOFactory ed Enveloper, per, rispettivamente aggiungere varietà ai timbri sonori e per regolare il volume delle singole note in entrata e in uscita al fine di eliminare i "click" audio, sgradevoli all'udito. Infine, per semplificare la gestione delle forme audio da caricare nelle note, ho sviluppato l'enum read-only WaveTable.

Classi sviluppate singolarmente:

- DrumSamples
- DrumSynth
- Engine
- Enveloper
- Filters
- KeyboardSynth
- LFOFactory
- Settings
- SynthBuilder
- WaveTable

• Problematiche riscontrate
problemi

3.2.3 membro 2

Per questo progetto mi sono focalizzato principalmente sugli aspetti di logica del gioco, comprendenti la gestione di tracce e note e della precisione, sia nell'aspetto audio-video che nel calcolo dei punteggi di gioco. Per quanto concerne l'aspetto delle tracce la classe fondamentale da me implementata è TrackFilterImpl che, fornita in input una Song, ottenuta dal midiParser, riduce il numero di note presenti in ogni sua traccia in modo da renderle giocabili. Ho provato diverse strategie per ridurre il numero delle note in modo da ottenere tracce con una distribuzione di note uniforme nel tempo e con quantitativi di note quanto più possibile differente per fornire più scelta al giocatore. Questa classe è stata particolarmente facile da integrare in quanto si occupa di elaborare dati già esistenti e a restituirli come era stato previsto in fase di progettazione. Per le note e il calcolo del punteggio ho realizzato un unica classe principale ColumnLogicImpl che divide le note nelle varie colonne grafiche eliminando le eventuali collisioni e, utilizzando la classe secondaria NoteRange, permette di fornire un punteggio per ogni nota premuta durante il gioco sulla base della configurazione del calcolo scelta tra le classi di calcolo del punteggio. Anche in questo caso l'obiettivo è stato quello di ottenere una distribuzione uniforme tra le varie colonne delle note e di fornire un attribuzione del punteggio che considerasse eventuali imprecisioni sulla pressione delle note durante il gioco. Questa classe è stata particolarmente difficile da finalizzare per una divergenza di classi/interfacce tra la parte logica e grafica, ed è stato quindi necessario creare una classe intermediaria LogicNoteImpl e dei metodi aggiuntivi in grado di fornire i dati mancanti per permettere un integrazione tra le due parti. In merito agli aspetti di precisione audio-video ho sviluppato la classe PlayerAudio che suona le note in base al tempo trascorso dall'inizio della traccia, ottimizzando la latenza tra ciò che appare a schermo e l'audio di gioco creando una relazione tra l'aspetto grafico e quello audio. L'integrazione di questa classe non ha generato particolari problemi se non la necessità di avere una conoscenza superficiale del funzionamento delle altre classi presenti nell'audioengine. L'ultima parte sviluppata singolarmente è quella relativa alla classe GameUtilitiesImpl, utilizzata per attribuire a ogni traccia restituita dal trackFilter una difficoltà basata sul numero di note.

Classi sviluppate singolarmente:

- classe 1
- classe 1
- classe 1

• Problematiche riscontrate

problemi

3.2.4 Sofia Tosi

In questa sezione si parlerà di grafica. La classe principale è PlayScreen, il cuore della View, attraverso la quale vengono richiamate tutte le altre classi implementate dalla sottoscritta e alcune classi dei miei compagni. Questa classe, alla quale ho dedicato più tempo, mi ha messo alla prova e mi ha stimolato a voler conoscere cose nuove. Infatti per implementarla mi sono servita della libreria grafica Libgdx, con la quale non avevo mai lavorato. Un'idea che mi è balenata in mente mentre la implementavo, che non si è manifestata durante l'analisi svolta all'inizio del progetto, è quella di creare una classe a parte, EventsFromKeyboard, per gestire i comandi di input. EventsFromKeyboard, come vuole suggerire il nome, è stata pensata per essere combatibile con una tastiera per pc. Nulla vieta di avvalersi di un'altra periferica. Infatti, ho cercato di limitare al massimo le dipendenze in PlayScreen in modo tale che risulti semplice sostituirla, avendo a disposizione un'altra classe compatibile con un'altra periferica. Ho ritenuto opportuno implementare AssetManager per gestire le risorse grafiche. Questa classe permette di caricare in memoria tutte gli asset grafici in una sola volta. Sarà poi il compito di PlayScreen richiamarli singolarmente, al momento opportuno, e mostrarli a video. Tra i miei compagni del gruppo, quello con cui ho dovuto interagire di più è Serafino Pandolfini, ed è stato proprio da quest'interazione che è scaturita la classe KeyImpl. Lo scopo di KeyImpl è proprio quello di fornirgli dei metodi per poter ottenere il tempo di inizio e di fine nel quale l'utente preme un tasto. Una classe fondamentale da utilizzare in PlayScreen è NoteImpl. NoteImpl rappresenta la nota, ma a differenza del Notelogic(la nota logica) e il, la rappresenta dal punto di vista grafico. In questa classe sono presenti i metodi per disegnare la nota, per disegnare le scintile che compaiono sulla nota quando l'utente la preme al momento giusto. Nonostante sia una classe che implementa metodi legati alla grafica si è cercato di limitare il più possibile l'uso di libgdx, per ridurre il più possibile le dipendenze. Un'ultima classe che merita di essere citata è senza dubbio SizeImpl, la quale adatta la GUI alla dimensione dello schermo del pc sul quale viene lanciata l'applicazione. Per collaborare io e i miei compagni abbiamo deciso di stabilire insieme come dovevano essere strutturate le interfacce e successivamente abbiamo implementato il codice basandoci sulle decisioni prese. Fortunatamente non ci sono stati particolari problemi, almeno per quanto riguarda la mia parte, e, in caso di dubbi, eravamo sempre disponibili a confrontarci.

Classi sviluppate singolarmente:

- PlayScreen
- AssetManager
- EventsFromKeyboard
- EventsFromKeyboardImpl
- Note
- NoteImpl

- Size
- SizeImpl
- Key
- KeyImpl
- ComputingShift
- ComputingShiftImpl

• **Problematiche riscontrate**
problemi

3.2.5 membro 4

ha fatto questo bla bla bla

Classi sviluppate singolarmente:

- classe 1
- classe 1
- classe 1

• **Problematiche riscontrate**

problemi

3.2.6 membro 5

ha fatto questo bla bla bla

Classi sviluppate singolarmente:

- classe 1
- classe 1
- classe 1

• **Problematiche riscontrate**

problemi

3.3 Note di sviluppo

schermata

3.3.1 Gianluca Migliarini

Feature avanzate del linguaggio:

- **lambda:** usate per caricare gli iteratori delle note per ogni traccia e i buffer dei campioni audio nei sintetizzatori di tastiere.
- **stream:** usati per controllare lo stato di tutti i sintetizzatori e sommare i vari campioni di questi ultimi.
- **optional:** usati per i campi opzionali nel builder dei sintetizzatori di tastiera.
- **function:** usate per gestire gli LFO

Librerie utilizzate:

- **libgdx:** utilizzata per Gdx.audio.newAudioDevice, necessario per riprodurre sul dispositivo audio hardware il buffer di campioni.

Classi riutilizzate:
Per comodità ho deciso di riutilizzare la classe generica Pair, introdotta dal prof. Viroli durante il corso.

3.3.2 membro 2

commenti

- **lambda:** tanta roba
- **stream:** tanta roba

librerie utilizzate:

- **libgdx:** gestione grafica

3.3.3 Sofia Tosi

commenti

- **lambda:** tanta roba
- **stream:** tanta roba

librerie utilizzate:

- **libgdx:** libreria grafica Libgdx: studio autonomo di tale libreria per realizzare l'interfaccia di gioco.

Classi riutilizzate:
Utilizzo della classe Pair fornita gentilmente dal professore Viroli

3.3.4 membro 4

commenti

- **lambda:** tanta roba
- **stream:** tanta roba

librerie utilizzate:

- **libgdx:** gestione grafica

3.3.5 membro 5

commenti

- **lambda:** tanta roba
- **stream:** tanta roba

librerie utilizzate:

- **libgdx:** gestione grafica

Capitolo 4

Commenti Finali

4.1 Autovalutazione e lavori futuri

schermata

4.2 Gianluca Migliarini

4.2.1 commenti

Sono molto soddisfatto del gruppo e del lavoro che siamo riusciti a portare a termine. Essendo una delle mie prime esperienze di lavoro in team ho trovato delle difficoltà principalmente riguardanti la gestione del tempo e a volte la coordinazione con gli altri membri, per questo ringrazio i miei compagni per la disponibilità e per aver tenuto sempre un clima sereno durante lo sviluppo; ritengo quindi che questo progetto mi abbia permesso di crescere notevolmente dal punto di vista organizzativo, in particolare per l'utilizzo di un software per la gestione del versioning. Inoltre sono molto felice di essere riuscito ad inserire in un progetto concreto il mio interesse per la sintesi audio.

4.2.2 difficoltà riscontrate

In conclusione, ritengo che il corso sia stato più che sufficiente per garantirmi uno sviluppo del progetto senza problematiche troppo grandi a livello implementativo, e che mi abbia incentivato ad utilizzare gli aspetto più avanzati del linguaggio, i quali avrei probabilmente evitato.

4.3 Serafino Pandolfini

4.3.1 commenti

Sono personalmente soddisfatto del lavoro che ho svolto; fin dai miei primi approcci alla programmazione ho sempre preferito, piuttosto che dedicarmi a particolari funzionalità di librerie, a porre il mio interesse sulle strutture dati e le loro organizzazioni e in questo progetto ho avuto modo di mettermi alla prova in un contesto che non fosse il mio solito programma giocattolo scoprendo le mie effettive conoscenze e capacità. Ho avuto modo di apprezzare in particolare gli stream e le loro funzionalità che riuscivano sempre a stupirmi. È stata ricorrente la situazione in cui emergeva un problema ed esisteva già un operazione in grado di risolverlo in modo quasi istantaneo. Non ho intenzione di continuare a sviluppare questo progetto ma se non mi fossi limitato al monte ore richiesto mi sarebbe piaciuto sviluppare altre modalità di gioco con calcoli dei punteggi e gameplay differenti prendendo ispirazione da altri giochi simili.

4.3.2 difficoltà riscontrate

Il progetto non è stato esente da difficoltà, principalmente la coordinazione tra i vari membri per quanto ottimale ha portato a incomprensioni sul funzionamento di certi aspetti e talvolta si è dovuto ricorrere a soluzioni non ottimali dal mio punto di vista. Dal mio punto di vista però l'emergere di queste difficoltà mi ha permesso di avere una comprensione più completa del progetto che non avrei avuto altrimenti.

4.4 Sofia Tosi

4.4.1 commenti

DukeMania è stato il mio primo grande progetto a cui ho preso parte. Non nego la mia preoccupazione iniziale, poi però pian piano durante l'analisi e implementando codice è svanita. Questo progetto mi ha permesso di crescere ampiamente, in vari ambiti. In primo luogo, ho imparato cosa significa scrivere codice di qualità e a quanto siano importanti i pattern. Al contempo, ho anche aumentato la mia conoscenza studiando una nuova libreria grafica e conoscendo meglio le potenzialità di Git e LaTex. Realizzare DukeMania non è stato affatto facile, soprattutto perchè si tratta di un gioco musicale e le mie conoscenze musicali sono esigue. Ma grazie alla collaborazione con i miei compagni, tutti molto disponibili e gentili siamo riusciti nel nostro intento. Indipendentemente dalla valutazione finale, mi ritengo molto soddisfatta del progetto.

4.4.2 difficoltà riscontrate

Ritengo il corso di Programmazione ad Oggetti un corso molto valido e interessante. Mi ha permesso di approfondire nuovi argomenti non soltanto da autodidatta, ma anche con le lezioni dei prof che le ho trovate chiare e stimolanti. L'unica modifica che apporterei è l'aggiunta di ore di laboratorio per imparare già da subito a prendere la mano con i pattern e Csharp. Nella mia modesta opinione avere più ore da dedicare a questi argomenti comporterebbe un notevole aiuto.

4.5 Laura Leonardi

4.5.1 commenti

progetto

4.5.2 difficoltà riscontrate

progetto

4.6 Alessandro Brasini

4.6.1 commenti

Tenendo in conto degli ostacoli incontrati nel corso dello sviluppo del progetto per arrivare a conoscere libgdx, mi sento molto soddisfatto del risultato finale ottenuto. Sono inoltre riconoscente ai miei compagni di team, i quali hanno mantenuto un approccio serio e professionale, nonostante la nostra comune inesperienza riguardante il lavoro in gruppo. Ho imparato l'importanza della fase di ideazione del progetto e l'utilità di suddividere equamente, tra compagni, il carico di lavoro, senza mai sottovalutare l'importanza di un DVCS, il quale si è rivelato fondamentale. In conclusione, la cosa che ho imparato di più da questo progetto, è il programmare in una modalità più responsabile e consapevole, in quanto i miei collaboratori riponevano fiducia nel mio lavoro come io nel loro.

4.6.2 difficoltà riscontrate

Durante il mio percorso formativo avevo già avuto occasione di programmare ad oggetti, ma è stato solamente dopo questo corso che ho capito la vera utilità di concetti come interfacce e classi astratte. Inoltre sono grato di aver utilizzato java, in quanto ho avuto la possibilità di imparare a programmare in un linguaggio del tutto nuovo, e per la prima volta ricco di sfaccettature e funzioni avanzate. Ritengo infine che le uniche problematiche inerenti al corso siano dovute dalla metodologia di lezione a distanza, problemi che purtroppo si sono riversati durante gli appelli d'esame e che mi hanno impedito di beneficiare degli insegnamenti al massimo.

Capitolo 5

Appendice

Guida Utente: