Università degli Studi di Padova

Facoltà di Ingegneria Informatica

Sviluppo software cross-platform per la composizione in real-time di brani musicali in funzione del proprio EEG semplificato

Laureando: Maistro Fabio

Relatore: Congiu Sergio

Data di Laurea: 30/09/2011
A.A. 2010/2011

Indice

RINGRAZIAMENTI

Si ringrazia Pub Company SRL per il tirocinio
e
il tutor aziendale Lorenzo Godina per il supporto

Si ringrazia il prof. Sergio Congiu per la disponibilità

DEDICA

Dedicato ai miei genitori Natalina e Maurizio
e
a mia sorella Katia

0. Sommario

Si da qui un breve riepilogo dell'attività svolta durante il tirocinio presso l'ente esterno.
Il periodo di tirocinio si è svolto presso l'azienda PubCompany SRL situata in Albignasego (PD) dedita ad attività di publishing e coordinamento dello sviluppo per varie piattaforme di gioco fra cui Nintendo Wii, Sony PlayStation 3 e Microsoft Xbox360.
Attualmente l'azienda è al lavoro su un progetto denominato 3i (Immersive Individual Interaction), che prevede l'utilizzo di un particolare cuffia bluetooth (chiamata 3i Mind Controller™, originariamente NeuroSky MindSet) prodotta dalla società NeuroSky® allo scopo di leggere alcuni valori delle onde celebrali della persona e trasformare tali dati in informazioni utili per provocare un certo comportamento all'interno di un videogioco.
A questo proposito è stato affidato al sottoscritto lo sviluppo di un'applicazione parallela cross-platform in Java che sia in grado di produrre pezzi musicali generando le note a partire da due particolari parametri letti in tempo reale dalle suddette cuffie: in pratica la persona che indossa le cuffie avrà la sensazione di essere lui, con il suo stato d'animo e il suo stress emotivo, a creare le diverse tonalità e sfumature della musica che sta ascoltando.
Tale applicazione potrà poi essere inserita in un gioco a sé stante di nome Brain Music (di cui si è già abbozzato un design e sviluppato parte della grafica preposta) oppure inserita nel principale videogame prodotto per la tecnologia 3i allo scopo di generarne le canzoni di sottofondo durante lo scorrimento dei vari menu.
L'applicazione non è stata purtroppo terminata, si è raggiunto lo scopo di riuscire a produrre musica dalle onde cerebrali, ma la musica generata risulta troppo artificiale e dunque dovrà essere poi migliorata utilizzando particolari tecniche di campionamento e sostituzione degli strumenti musicali, in modo da renderla più naturale possibile.
Anche dal punto di vista grafico e delle funzionalità l'implementazione si è fermata ad uno stadio prototipale, senza raggiungere il livello di dettaglio che sarebbe richiesto da un gioco completo.

1. Introduzione (raccolta dei requisiti)

Il progetto Brain Music completo prevede la creazione di tre interfacce principali di gioco raggiungibili dopo aver mostrato un'introduzione animata con i loghi dei produttori e un breve filmato d'introduzione al gioco (in gergo chiamato "trailer" o "intro").

1. **Tutorial & Trouble-shooting**: la prima interfaccia, non implementata nel prototipo, prevede di mostrare al giocatore come indossare il MindSet NeuroSky, hardware indispensabile per la lettura delle onde cerebrali dell'utente e quindi da considerare come un vero e proprio controller di gioco. Attraverso l'utilizzo di diverse schermate e guide animate si illustrano al giocatore le regole base per un corretto uso delle cuffie e si invita il giocatore stesso ad indossarle. Una volta che il giocatore ritiene di essere pronto, parte un'ulteriore schermata di trouble-shooting nel quale il sistema controlla di ricevere il segnale dalle cuffie: se così è allora il giocatore viene indirizzato alla prima vera schermata di gioco, altrimenti il software cerca di aiutare l'utente a capire cosa possa essere andato storto o cosa possa influire sulla mancata ricezione del segnale.
2. **Start Menu**: la seconda interfaccia è la prima schermata vera di gioco. In questa schermata devono essere disponibili le seguenti funzionalità:
    - Avviare la successiva schermata di gioco (chiamata Player Menu) che permetterà di iniziare il processo di lettura delle EEG e la relativa produzione di musica "controllata" dal cervello del giocatore.
    - Aprire una PlayList con tutte le canzoni già generate e salvate nella stessa sessione di gioco, permettendo di avviare la riproduzione di tali canzoni (con tutte le funzioni annesse presenti di solito in un player audio), conoscere il macrostile utilizzato per quella particolare canzone, nonché durata e data di creazione della stessa.
    - Salvare una determinata canzone su hard-disk o supporto rimovibile in modo da renderla disponibile per qualsiasi lettore che permetta di leggere files audio standard.

Schermata Start Menu

3. **Player Menu**: l'interfaccia principale di gioco. In questa schermata viene avviata la lettura dei parametri di attenzione e meditazione dalle cuffie e vengono conseguentemente generate le canzoni in base ai parametri letti. Tale schermata, completata solo in parte nel prototipo attuale, dovrebbe permettere una produzione continua di canzoni della durata standard o prefissata dall'utente in qualche opzione precedente e deve quindi dare un feedback visivo di qualche tipo all'utente per mantenere vivo il suo interesse per il gioco e per fargli sapere il tipo di musica che il suo cervello sta generando. Per questo motivo il progetto Brain Music prevede la possibilità di mostrare all'utente diversi tipi di feedback visivi che possono andare da un semplice spartito con delle note musicali che vengono via via colorate e messe in rilievo ad un visualizzatore audio con effetti di luce simile a quanto si può vedere in Microsoft Windows Media Player® oppure ad un'animazione che mostra gli strumenti che in quel momento stanno suonando la canzone. Purtroppo nessuno di questi feedback visivi è presente nell'attuale prototipo. Inoltre le funzioni che deve svolgere questa schermata sono:
    - Mostrare l'attuale macrostile ("genere musicale") dovuto al parametro iniziale di meditazione e gli strumenti che stanno suonando in questo momento;
    - Mostrare lo stato d'animo dell'utente durante la riproduzione per dare conto delle diverse variazioni di stile dovute alle variazioni del parametro di attenzione.

- Mostrare lo stato del segnale del MindSet con un'icona che indica visivamente la potenza del segnale ricevuto e quindi la sua affidabilità. In caso il segnale andasse perso è necessario interrompere la riproduzione e la generazione della canzone in tempo reale dando un qualche messaggio d'errore all'utente.
- Dare la possibilità al giocatore di mettere in pausa il sistema, togliersi le cuffie e riposarsi e poi tornare ad utilizzare il gioco, ovviamente generando una nuova canzone, mentre la vecchia, anche se interrotta, deve venire comunque salvata.
- Dare la possibilità al giocatore di ricominciare la generazione della canzone da 0, ripartendo dunque dalla scelta automatica del macrostile in base al parametro iniziale di meditazione, ecc.
- Dare la possibilità al giocatore di tornare alla schermata precedente per consultare eventualmente la PlayList o simili.
- Dare la possibilità al giocatore, premendo un tasto, di cambiare il feedback visivo attivo in quel momento (passare ad esempio dalla visualizzazione dello spartito colorato alla visualizzazione con effetti luminosi).
- Informare il giocatore quando la canzone sta volgendo al termine e mostrargli con leggero anticipo, durante la conclusione della canzone attuale, il nuovo macrostile che sarà usato come base per la canzone successiva.

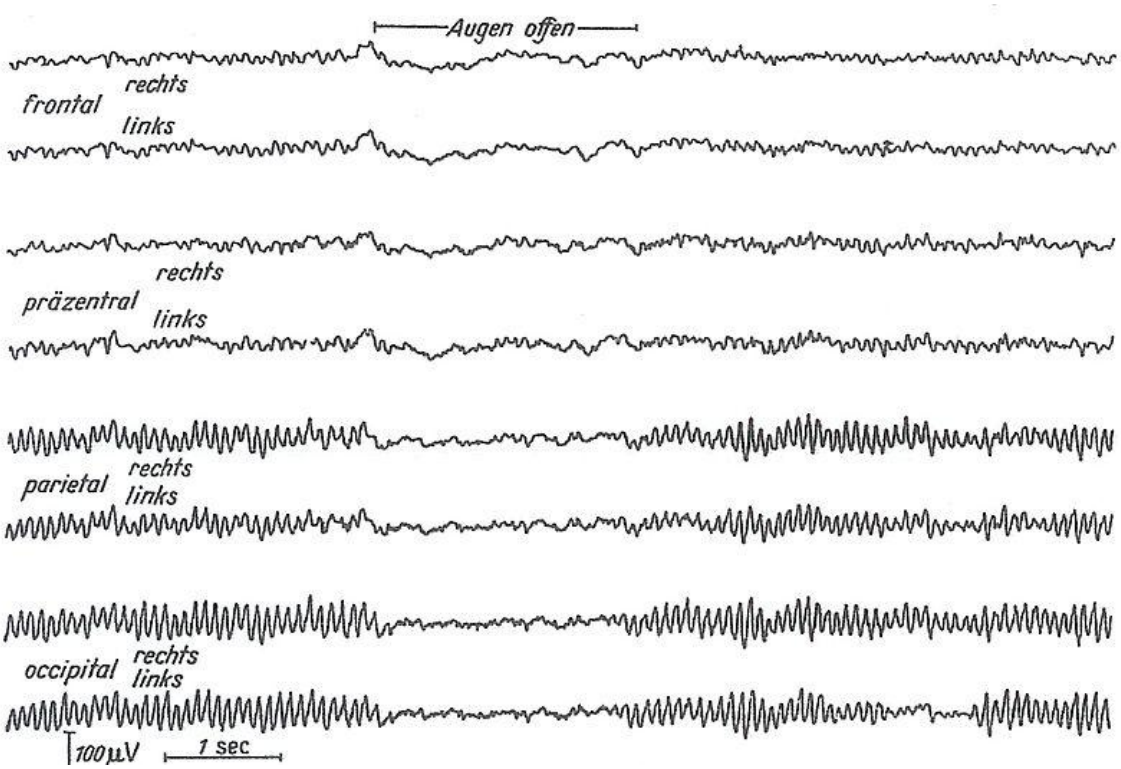
Progetto schermata Player Menu

2. Tecnologia utilizzata

2.1. Definizione formale EEG

L'elettroencefalogramma (EEG) registra l'attività elettrica cerebrale tramite elettrodi di superficie posizionati sulla testa. La continua fluttuazione della normale attività cerebrale induce tra vari punti del cuoio capelluto piccole differenze di potenziale elettrico (milionesimi di volt, microvolt) che vengono amplificate e registrate normalmente per alcuni minuti (in casi particolari fino a 24 ore). Si ottiene in questo modo un tracciato che segna per ciascun elettrodo le variazioni del voltaggio nel tempo. Normalmente gli elettrodi vengono montati secondo uno schema fisso (sistema 10-20) su tutte le parti del cuoio capelluto. Poiché ogni elettrodo riflette in prima linea l'attività della parte cerebrale più vicina, l'EEG è in grado di fornire informazioni non solo su attività elettriche anomale, ma anche sulla loro localizzazione.

Esempio di diagramma EEG



2.2. Sistema di lettura MindSet Neurosky®

Per il progetto Brain Music non vengono usati i tradizionali strumenti di lettura delle EEG utilizzati in neurologia o in ambiente medico, bensì uno strumento più accessibile, economico e veloce da utilizzare e cioè una paio di semplici cuffie con alcuni speciali sensori in grado di captare le onde elettromagnetiche, semplicemente indossando le stesse.
Un simile strumento viene prodotto da una società americana di nome NeuroSky che ha come scopo quello di rendere accessibile a tutto il mercato fuori dai laboratori la capacità di leggere le EEG attraverso dei sensori di onde cerebrali.
Per questo motivo NeuroSky ha dovuto superare le limitazioni presenti nei normali sistemi di lettura cerebrali usati nei laboratori, come ad esempio limitare il rumore distorsivo causato da altri sistemi elettrici in uso assieme al sensore o quello di utilizzare il gel conduttore direttamente sulle meningi per aumentare la capacità di lettura delle onde cerebrali da parte del sistema.
Per quanto riguarda il rumore viene utilizzata un'amplificazione del segnale che rende il signale cerebrale grezzo più forte, mentre sono stati utilizzati alcuni protocolli per eliminare le frequenze di rumore conosciute come quelle dovute ai muscoli, al battito cardiaco e agli altri strumenti elettronici. Attraverso l'uso di filtri notch dedicati, NeuroSky ha ottenuto un ottimo risultato in fatto di individuazione ed eliminazione del rumore e di sicuro rimane uno dei punti più importanti sui quali si concentrerà lo sviluppo futuro, per ottenere risultati sempre migliori.
Estrapolare il segnale EEG dal rumore richiede sia un punto di riferimento che un circuito elettrico di scarico. Lo scarico rende il voltaggio del corpo pari al voltaggio delle cuffie, mentre il punto di riferimento è utilizzato per sottrarre il normale rumore ambientale attraverso un processo conosciuto come "*common mode rejection*". Il lobo dell'orecchio in particolare è il luogo in cui si può registrare lo stesso rumore ambientale del sensore sulla fronte, ma con minima attività neuronale, per cui sottraendo il rumore ambientale registrato dal lobo dell'orecchio all'intero segnale registrato dal sensore frontale si può ottenere, con buona approssimazione, un segnale privo di rumore. Per questo motivo è importantissimo che la connessione all'orecchio sia perfettamente funzionante, cioè che le cuffie siano indossate nel modo giusto.
In definitiva, quello che fanno le cuffie NeuroSky è portare le capacità BCI (Brain computer interface, cioè sistema che traduce attività elettrofisiologica di un organismo vivente in segnali interpretabili da un sistema meccanico) fuori dal laboratorio e disponibile al pubblico in generale.

Foto MindSet NeuroSky®

2.2.1. Hardware & Software MindSet

La tecnologia che sta alla base del funzionamento del MindSet (nome dato da NeuroSky alle cuffie che permettono di leggere le EEG) è denominata ThinkGear e comprende:

- Il sensore che tocca la fronte
- Il contatto e i punti di riferimento posizionati sugli auricolari
- Il chip interno che processa tutti i dati

Sia le onde cerebrali grezze che i parametri eSense come quelli di attenzione e meditazione sono registrati e tradotti dal chip ThinkGear. I valori calcolati sono poi dati in uscita come output dal chip ThinkGear, attraverso le cuffie, al PC grazie ad una connessione Bluetooth. I tipi di dati fornito come output dal chip sono:

- Valori grezzi di campioni di onde cerebrali (128 Hz o 512 Hz, dipende dall'hardware)
- Scala di valori per monitorare la qualità del segnale (un segnale "Poor Quality = 0" è ottimale, significa che non ci sono problemi nella ricezione)
- Valori eSense di attenzione e meditazione
- Valori della potenza di banda del segnale EEG per le onde delta, theta, alfa, beta e gamma

Dal punto di vista software tali valori possono essere ricevuti da qualsiasi applicazione, grazie al ThinkGear Connector (TGC) che è un programma software – analogo ad un server socket – che viene avviato in background nel PC ed è responsabile di mandare i dati delle cuffie dalla porta seriale ad un socket di rete aperto. Ogni linguaggio o framework che contiene una libreria socket dovrebbe essere in grado di comunicare con il TGC che quindi è la scelta ideale per gli sviluppatori.
Tuttavia la migliore soluzione software per gli sviluppatori di programmi commerciali o con maggiori esigenze di integrazione, rimane il ThinkGear Communications Driver (TGCD) cioè un dispositivo driver con una semplice API che permette la comunicazione fra un'applicazione su un computer (o un dispositivo mobile) e un chip ThinkGear inserito ad esempio nel MindSet.
Tale driver è sviluppato come libreria .dll per Windows scritta in linguaggio C, ma è disponibile anche come classe/libreria Java, a patto che si usi attraverso la JNI (Java Native Interface) che permette di utilizzare in Java applicazioni e librerie scritte in altri linguaggi. Ed è proprio quest'ultima che è stata sfruttata per il progetto.

2.2.2. Protocollo di comunicazione software MindSet

Il protocollo di comunicazione del MindSet definisce, in dettaglio, come comunicare con le cuffie NeuroSky. In particolare si ha:

a. Interfaccia Bluetooth

Il MindSet trasmette i valori dati ThinkGear, decodificati all'interno dei pacchetti ThinkGear, come uno stream seriale di bytes lungo una Porta Seriale standard Bluetooth (Bluetooth SPP):
Profilo Bluetooth: Serial Port Profile (SPP)
Baud Rate: 57600
Chiave d'autenticazione (necessaria per tutti i dispositivi bluetooth): 0000

b. ThinkGear Data Values

I dati che il chip ThinkGear può trasmettere sono:

- *POOR_SIGNAL Quality*: questo valore intero di un byte descrive quanto può essere povero il segnale misurato dal MindSet. Il suo range va da 0 a 200 e ogni valore diverso da 0 indica che è stata rilevata una contaminazione del segnale causata dal rumore. Più alto è il numero, più è il rumore rilevato, mentre il valore 200 significa che i contatti ThinkGear non toccano la pelle dell'utente.
  Questo valore è tipicamente fornito ogni secondo e dunque tenta di essere il più

accurato possibile. Diverse possono essere le cause di un segnale povero e cioè, in ordine di gravità:

1) Mancanza di contatto fra sensore e pelle (ad esempio quando nessuno sta indossando le cuffie)
2) Contatto debole fra il sensore e la pelle della persona (ad esempio se ci sono capelli che ostruiscono oppure le cuffie sono indossate male)
3) Eccessivo movimento dell'indossatore (ad esempio se l'utente scuote troppo la testa)
4) Eccessivo rumore elettrostatico dell'ambiente circostante
5) Eccessivo rumore biometrico non-EGG (ad esempio EMG, EKG/ECG, ecc)

Una certa quantità di rumore è comunque impossibile da eliminare nel normale uso delle cuffie ed è per questo che la tecnologia è stata progettata per rilevare, correggere e compensare diversi tipi di rumore non-EEG. Si noti inoltre che se si è interessati soltanto ai parametri eSense di attenzione e meditazione, come nel caso in esame, il POOR_SIGNAL indica che i valori di attenzione e meditazione non vengono aggiornati finchè il POOR_SIGNAL rimane diverso da 0.
Di default l'output di questo Data Value è sempre attivo e fornito una volta al secondo.

- *eSense Meter*: per tutti i diversi tipi di eSense (es. Attenzione, Meditazione), il valore metrico è riportato in una scala eSense relativa da 0 a 100. Su questa scala un valore da 40 a 60 è considerato "neutrale", mentre un valore da 60 a 80 è considerato "leggermente elevato" e potrebbe essere interpretato come un livello più alto del normale. I valori da 80 a 100 sono considerati "elevati", nel senso che sono fortemente indicativi di livelli accresciuti di quel eSense.
  Allo stesso modo, un valore fra 20 a 40 indica livelli "ridotti" dell'eSense mentre un valore fra 1 a 20 indica livelli "fortemente abbassati" dell'eSense. Questi livelli potrebbero indicare stati di distrazione, agitazione o anormalità psichica, in base al significato di ogni eSense.
  Un valore eSense di 0 è un valore speciale che indica che il ThinkGear non riesce a calcolare il livello eSense a causa di diversi fattori, ma di solito è dovuto ad eccessivo rumore come visto nella sezione dedicata al POOR_SIGNAL.
  In particolare le metriche eSense disponibili sono:
  1) *ATTENZIONE*: un valore senza segno di un byte che indica l'intensità del livello di attenzione o concentrazione dell'utente, come quello che avviene durante un'intensa concentrazione o un'attività mentale stabilmente diretta verso un argomento preciso. Il suo range di valori va da 0 a 100 e distrazioni, ansia o mancanza di concentrazione possono abbassare il livello di

meditazione.
Di default questo Data Value è abilitato ed è tipicamente processato una volta al secondo.

2) *MEDITAZIONE*: un valore senza segno di un byte che indica l'intensità del livello di calma o rilassamento dell'utente. Il suo range di valori va da 0 a 100 e può essere considerato come una misura dei livelli mentali di rilassamento di una persona, intendendo con questo uno stato di relax solamente psichico e non fisico: un rilassamento dei muscoli del corpo potrebbe non modificare subito i livelli di meditazione. Tuttavia, per la maggior parte delle persone, il rilassamento del corpo è il principale aiuto per riuscire a rilassare anche la mente. La meditazione è relativa ad una ridotta attività dei processi mentali attivi del cervello ed è stato osservato che chiudere gli occhi ad una persona spegne le sue attività mentali preposte alla codifica delle immagini, dunque chiudere gli occhi può essere un buon sistema per aumentare il livello di Meditazione misurato. Distrazione, ansia, agitazione e stimoli sensoriali sono le cause più comuni di un abbassamento del livello di Meditazione.
Di default questo Data Value è abilitato ed è tipicamente processato una volta al secondo.

- *RAW Wave Value (16-bit):* questo Data Value consiste di due bytes e rappresenta un campione di una singola onda cerebrale grezza. Il suo valore è un intero con segno di 16-bit che quindi ha un range di valori da -32768 a 32767.
  Per ricostruire l'intero valore dell'onda è sufficiente fare uno shift del primo byte a sinistra di 8 bit e poi un or-bitwise con il secondo byte.
  Di default questo Data Value è abilitato ed è tipicamente processato 512 volte al secondo, o approssimativamente una volta ogni 2 ms.
- *ASIC_EEG_POWER*: questo Data Value rappresenta l'ampiezza attuale degli 8 tipi di EEG riconosciuti dal dispositivo. Il suo output è una serie di 8 interi da 3-byte senza segno in formato little-endian. Le 8 potenze delle onde EEG hanno quest'ordine di output:
  1) delta (0.5 - 2.75Hz)
  2) theta (3.5 - 6.75Hz)
  3) low-alpha (7.5 - 9.25Hz)
  4) high-alpha (10 - 11.75Hz)
  5) low-beta (13 - 16.75Hz)
  6) high-beta (18 - 29.75Hz)
  7) low-gamma (31 - 39.75Hz)
  8) mid-gamma (41 - 49.75Hz).

Tali valori non hanno unità di misura e sono per questo significativi solo se confrontati l'uno con l'altro per considerare le quantità relative e le fluttuazioni temporali.
Di default questo Data Value è abilitato ed è tipicamente processato una volta al secondo.

c. ThinkGear Packets

Il componente ThinkGear fornisce i suoi dati digitali come una sequenza seriale asincrona di bytes. La sequenza seriale deve essere poi analizzata ed interpretata come un pacchetto ThinkGear in modo tale da essere poi estratta ed interpretata come un ThinkGear Data Value descritto sopra.
Un pacchetto ThinkGear è costituito da 3 parti:

- Header (intestazione)
- Payload (carico utile)
- Checksum (controllo d'integrità dei dati)

Un pacchetto ThinkGear è usato per inviare i Data Values da un modulo ThinkGear ad un ricevitore arbitrario (un PC, un altro chip o qualsiasi cosa possa ricevere stream di bytes) ed il suo formato è progettato primariamente per essere robusto e flessibile: l'header ed il checksum, combinati, consentono una sincronizzazione veloce dei dati ed un controllo sull'integrità dei dati, mentre il Payload assicura che nuovi dati possano essere aggiunti in nuovi pacchetti in futuro senza che le applicazioni esistenti debbano cambiare gli interpreti dei pacchetti. Ciò significa che anche se venisse cambiato l'hardware ThinkGear o venissero inclusi nuovi Data Values o cambiato l'ordine degli stessi, questo non andrebbe ad influire sulle applicazioni che si occupano di interpretare i pacchetti ThinkGear e ricavarne i dati.

2.2.3. Uso del ThinkGear.java nel progetto

Per una migliore fruizione delle funzioni della libreria ThinkGear d'interesse per il progetto si è deciso di creare una classe ad hoc nel pacchetto *com.neurosky.thinkgear* e cioè la classe *HeadsetManager*. Attraverso i metodi della classe *HeadsetManager* è possibile stabilire più agevolmente la connessione con il MindSet, leggere i dati, tenere controllato il segnale e chiudere la connessione quando non più necessaria.
In particolare i passi standard da svolgere nel progetto per ottenere i valori di meditazione ed attenzione dell'utente attraverso il Mindset sono:

1. **Connessione al chip ThinkGear delle cuffie**: si utilizza il metodo *HeadsetConnect* specificando eventualmente il nome della porta COM che si vuole utilizzare come parametro del metodo. Se avviene qualche errore nella connessione viene stampato in

standard output l'errore e il programma viene interrotto, altrimenti si può procedere al passo successivo.

2. **Recupero dei valori medi di attenzione e meditazione**: è possibile utilizzare separatamente rispettivamente i metodi *GetMeanAttention* e *GetMeanMeditation*; tuttavia l'uso di uno dei due metodi precedenti prevede che in uno stesso lasso di tempo si possano ricevere o soltanto i parametri di attenzione o soltanto i parametri di meditazione, ma non entrambi.

```
public int GetMeanMeditation(int NumPackets)
{
    int i = 0;
    int[] meditations = new int[NumPackets+1];
    while( i < NumPackets  )
    {
    // Attempt to read a Packet of data from the connection
    int errCode = ThinkGear.ReadPackets( connId, 1 );

    // If ReadPackets() was able to read a complete Packet of data...
        if( errCode == 1 )
        {
        // If meditation param value has been updated by ReadPackets()...
            if( ThinkGear.GetValueStatus(connId, ThinkGear.DATA_MEDITATION) != 0 )
            {
            int med = (int)ThinkGear.GetValue(connId,ThinkGear.DATA_MEDITATION);
            // Get the updated meditation param value
            meditations[i] = med;
            i++;
            }
        }
    }
    int mean = 0;
    for (int z = 0; z < NumPackets; z++)
    {
        mean = mean + meditations[z];
    }
    meditation = (int) Math.round((double)mean/NumPackets);

    return meditation;
}
```

Per questo motivo è stata contemplata la possibilità di leggere entrambi i parametri attraverso il metodo *GetMeanBoth* e poi salvare entrambi i valori medi in due variabili intere, ritornate dal metodo sotto forma di un array di 2 celle.

```
public int[] getMeanBoth(int NumPackets)
{
    int i = 0;
    int[] attentions = new int[NumPackets+1];
    int[] meditations = new int[NumPackets+1];

    while( i < NumPackets  )
    {
    // Attempt to read a Packet of data from the connection
    int errCode = ThinkGear.ReadPackets( connId, 1 );

    // If ReadPackets() was able to read a complete Packet of data...
        if( errCode == 1 )
        {
        // If attention param value has been updated by ReadPackets()...
        if( ThinkGear.GetValueStatus(connId, ThinkGear.DATA_ATTENTION) != 0 )
            {
            int att = (int)ThinkGear.GetValue(connId,ThinkGear.DATA_ATTENTION);
            int med = (int)ThinkGear.GetValue(connId,ThinkGear.DATA_MEDITATION);

            attentions[i] = att;
            meditations[i] = med;
             i = i + 1;
            }
        }
    }
    int mean_att = 0;
    for (int z = 0; z < NumPackets; z++)
    {
        mean_att = mean_att + attentions[z];
    }

    int mean_med = 0;
    for (int z = 0; z < NumPackets; z++)
    {
        mean_med = mean_med + meditations[z];
    }

    attention = (int) Math.round((double)mean_att/NumPackets);
    meditation = (int) Math.round((double)mean_med/NumPackets);

    int[] to_ret = new int[2];
    to_ret[0] = meditation;
    to_ret[1] = attention;
    return to_ret;
}
```

Sia che si usino i primi due metodi, sia che si usi quest'ultimo, nel parametro formale del metodo va specificato il numero di pacchetti che si vogliono leggere dal MindSet prima di effettuare la media fra i valori trovati. Si ricordi che il sistema legge, di default, un valore di meditazione/attenzione al secondo e che quindi c'è una corrispondenza perfetta fra secondi di lettura e numero di pacchetti scelti: questo significa che, facendo la media dei valori registrati in un intervallo di tempo sufficientemente grande (ma non troppo ampio da rilevare grossi salti di range), si registrano delle variazioni di meditazione/attenzione generalmente fluide e si riesce ad evitare che variazioni brusche di attenzione/meditazione dovute ad imprecisioni del MindSet o a rumore di fondo, possano influire sul risultato finale della composizione musicale.
Usando un semplice esempio, se si decidesse di rilevare un solo parametro eSense per ogni pacchetto si avrebbe una situazione del genere:

0	80	14	70

in cui variazioni continue magari dovute alla mancanza di alcune registrazioni intermedie (perse a causa di rumore o POOR_SIGNAL) portano a differenze enormi in brevissimi lassi di tempo. Lo stesso esempio però, con due parametri per pacchetto, diventa una variazione media assolutamente normale (scarto di appena 2 punti):

0 80	14 70
Media: 40	Media: 42

Il trucco qui sta nel trovare il giusto compromesso fra l'eccessiva variabilità (che avviene se si esagera in difetto) e l'inaccuratezza dei dati (che avviene se si esagera in eccesso).

3. **Disconnessione dal dispositivo**: quando la rilevazione dei dati è finita si può chiudere la connessione fra MindSet e driver di lettura attraverso il metodo *HeadsetDisconnect()* che libera semplicemente la connessione stabilita precedentemente rendendo di nuovo la porta COM inutilizzata.

3. Introduzione al Concept del gioco

3.1. Introduzione software Impro-Visor

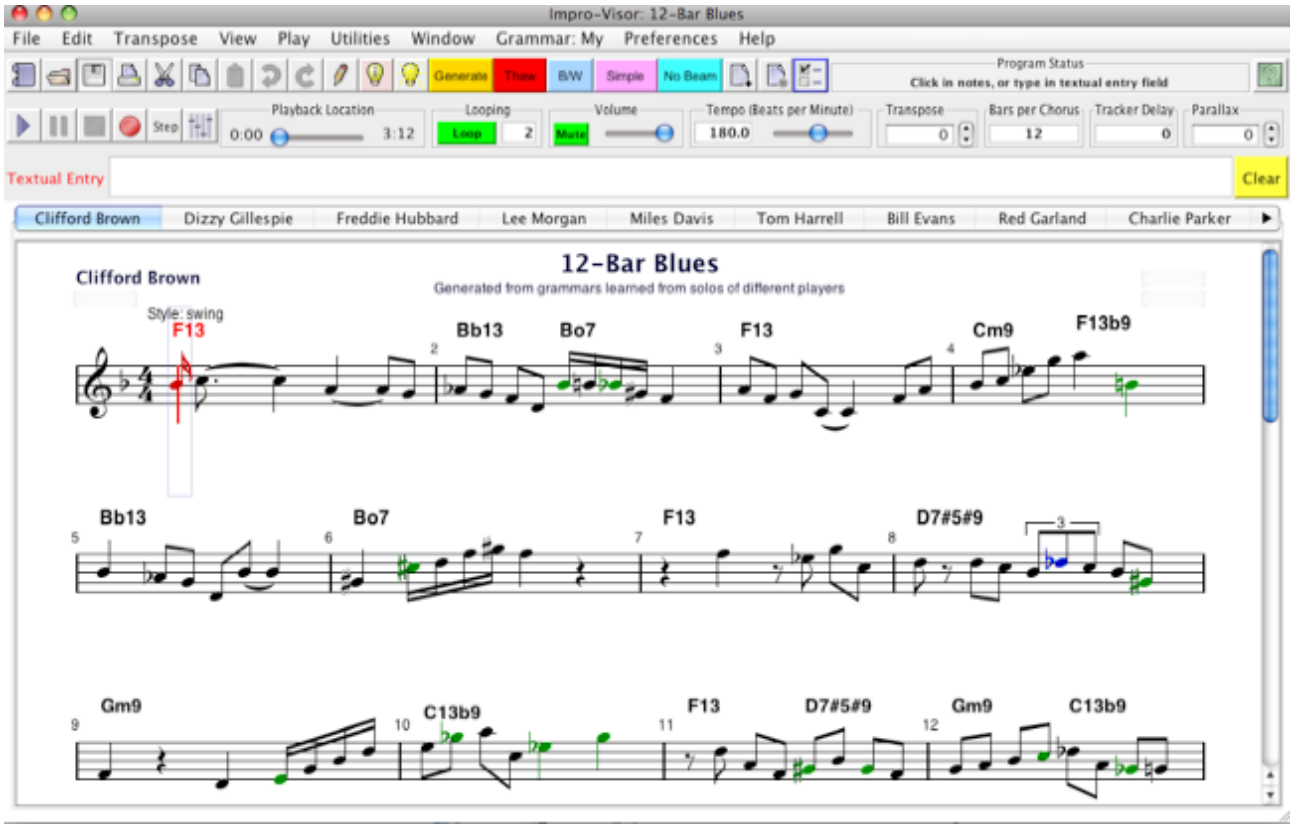
Per raggiungere l'obiettivo prefissato di generare musica improvvisata sulla base dei parametri cerebrali dell'utente, si è deciso di usare in parte il codice sorgente di un software freeware di nome Impro-Visor, ideato da Bob Keller (professore di Computer Science al College Harvey Mudd) e da alcuni suoi studenti e sviluppato in Java, utilizzando la libreria Java JMusic creata da Andrew Sorensen e Andrew Brown (Queensland University of Technology) e la libreria Java Polya creata dallo stesso Bob Keller.
Secondo le parole dello stesso autore del tool, "*Impro-Visor (abbreviazione di "Improvisation Advisor") è un programma di notazione musicale realizzato per aiutare i musicisti jazz a comporre e ascoltare degli assoli simili a quelli che potrebbero essere da loro improvvisati. L'obiettivo è di migliorare la comprensione della costruzione degli assoli e i cambi di tonalità degli accordi. Ci sono poi altre funzioni come ad esempio la possibilità di improvvisare un intero pezzo. Dato poi che la sezione ritmica di accompagnamento è generata automaticamente a partire dagli accordi, Impro-Visor può essere usato anche come uno strumento di esecuzione musicale. Avendo poi una gamma di stili di accompagnamento diversi, il suo uso non è limitato al solo jazz." (Da: homepage Impro-Visor: http://www.cs.hmc.edu/~keller/jazz/improvisor/)*

Il funzionamento standard di Impro-Visor per la creazione da zero di un pezzo musicale prevede:

1) La definizione di una "grammatica", cioè una specificazione linguistica che regola l'uso di lick (un corto frammento melodico allineato a uno o due accordi) semplici o complessi e può introdurre sfumature stilistiche.
2) La scelta di uno stile dominante (che ne indica anche gli strumenti da usare). Per stile si intende una specifica di come l'accompagnamento (accordi, bassi, percussioni) viene generato.
3) La regolazione del tempo (battute al minuto, di solito viene lasciato di default a 120 bpm).
4) La scelta del numero complessivo di battute (bars/chorus).
5) Generazione automatica tramite tasto "generate".

Tale procedimento è stato mantenuto, in background, nel progetto in esame, ma è stato cambiato il modo di generare del programma e di scelta di grammatica e stili, in modo tale che esso si rifaccia ai parametri mentali.

Foto software Impro-Visor



3.2. Qualità output Impro-Visor

L'output generato da Impro-Visor è una composizione musicale grezza, cioè un pezzo in midi che comprende tutte le note come se fossero suonate da uno strumento sintetizzato elettronicamente, dunque non di grande qualità sonora. E' possibile aggiungere più di un accordo, ma in generale il livello qualitativo rimane molto basso, troppo per quello che si vuole ottenere con Brain Music e cioè un sottofondo musicale di qualità. Per questo motivo, nelle fasi avanzate di progettazione, alle quali non si è arrivato con questo prototipo, sarà necessario procedere ad una ricampionatura degli strumenti in modo tale da renderli più simili alle loro controparti reali; andranno poi aggiunti degli effetti sonori o degli sfondi ad hoc per ottenere una composizione più omogenea e orecchiabile.

3.3. Lista operazioni concept di gioco

Lo sviluppo del prototipo di gioco è partito tenendo conto di quello che dovrebbe essere il risultato finale per quanto riguarda il corpo principale del gioco e cioè la generazione della canzone a partire dalle EEG dell'indossatore delle cuffie MindSet. Si prevede una sequenza di operazioni di questo tipo:

1) Il gioco parte scegliendo automaticamente un macrostile (si veda il paragrafo successivo per il significato del termine) in base al valore iniziale di meditazione (ogni valore di meditazione rilevato ha assegnato un particolare macrostile). In questo modo vengono automaticamente determinati il genere musicale del pezzo che si sta per improvvisare e gli strumenti musicali che verranno via via utilizzati.
2) A questo punto, il gioco processa i valori medi di meditazione/attenzione che vanno ad influenzare la modulazione musicale (meditazione), lo stile e la grammatica (attenzione) che vengono improvvisate in pseudo real-time. Quello che ne deriva è una canzone con un introduzione, una parte centrale e una conclusione, il tutto generato in tempo reale in base ai parametri di attenzione e meditazione che vengono continuamente letti dal MindSet.
3) Il processo del punto 2 continua finchè l'utente leva le cuffie, il segnale è perso oppure la canzone si considera finita (viene infatti definita una durata massima della canzone, oltre la quale il processo riparte dal punto 1, cioè dalla scelta del macrostile).
4) Durante la generazione della canzone viene dato un feedback visivo di qualche tipo all'utente (si mostrano gli strumenti musicali, si mostra un audio visualizer, ecc) che mostra anche i cambiamenti di stile dovuti alle modifiche di attenzione.
5) Alla fine di ogni singola canzone, l'utente è informato del nuovo macrostile (che viene calcolato a partire dal parametro di meditazione degli ultimi secondi della canzone precedente) tramite un feedback visivo.
6) E' necessario predisporre il gioco con una serie di pulsanti e funzioni per permettere il rewind, ma anche la possibilità di salvare la canzone "generata col proprio cervello", avere una playlist di quanto fatto, ecc.

3.4. Introduzione al concetto di "Macrostile"

Come già detto, all'inizio di ogni canzone viene scelto un macrostile, cioè una classe di stili di accompagnamento all'interno della quale verranno scelti gli stili da variare durante la canzone. Il macrostile dunque non è esso stesso uno stile, ma si può definire come un genere musicale, un raggruppamento di diversi stili secondo i canoni di divisione della musica in generi. Ad ogni macrostile, in particolare, vengono assegnati 5 diversi stili musicali e una banda di valori di

meditazione entro i quali viene scelto un macrostile piuttosto che un altro.
Il macrostile dunque è il concetto "costante" durante l'esecuzione di una singola canzone, mentre lo stile è il concetto "variabile" durante l'esecuzione della stessa.
Si tenga conto peraltro che Impro-Visor prevede la concezione di stile, ma non di macrostile, dunque nel progetto in esame sono stati raggruppati i diversi stili disponibili in Impro-Visor in 6 macrostili per introdurre il concetto di "genere musicale", sconosciuto ad Impro-Visor ma importante per il funzionamento di tutto il sistema di gioco.
Nella tabella sottostante è possibile vedere i macrostili che sono stati usati nel progetto:

Tabella macrostili e stili

Macrostyle	Threshold for macrostyles (meditation)	Style	Style Num-ber	Threshold for styles (attention)	Emotional label (ITA)	Emotional label (ENG)
Classic	80	No-style-but-swing	37	0	Calmo	Calm
		Shuffle	65	30	Rilassato	Relaxed
		March	35	45	Disteso	Atease
		Waltz-parisian	76	60	Contento	Content
		Pedal-bass-5	43	75	Sereno	Serene
Jazz	60	Bossa	7	0	Assonnato	Sleepy
		Cabaret	10	30	Disteso	Atease
		Song-for-my-father	67	45	Contento	Pleased
		Mambo	32	60	Felice	Happy
		Swing	70	75	Felicissimo	Delighted
Folk	50	Ballad	5	0	Annoiato	Bored
		Irish	24	30	Rilassato	Relaxed
		Polka	44	45	Contento	Glad
		Cha-cha-cha	13	60	Stupito	Astonished
		African	2	75	Eccitato	Excited
Latino	40	Latin	29	0	Felice	Happy
		Latin-enhanced	30	30	Rallegrato	Delighted
		Reggae	47	45	Eccitato	Excited
		Ska	66	60	Stupito	Astonished
		Reggae	47	75	Suscitato	Aroused
Pop	20	No-style	38	0	Calmo	Calm
		Pop-blues	45	30	Disteso	Atease
		Pedal-bass-1	39	45	Disteso	Atease
		Funk-disco	19	60	Teso	Tense
		Hits-0	21	75	Eccitato	Excited
Rock	0	Rock-50s	51	0	Assonnato	Sleepy
		Rock-slow	60	30	Impaurito	Afraid
		Rock-light	57	45	Suscitato	Aroused
		Rock-triplet	62	60	Allarmato	Alarmed
		Rock-heavy-even	55	75	Arrabbiato	Angry

Si noti inoltre che la scelta del macrostile definisce anche una gamma di strumenti assegnati a quel macrostile per far sì che la canzone sia il più possibile coerente con il genere musicale scelto, cercando di evitare dissonanze o bizzarre scelte strumentale (ad esempio non ci si aspetta di sentire suonare una chitarra elettrica se la canzone è di genere classico). Il nuovo set di strumenti è dunque univocamente determinato all'inizio della canzone in base al macrostile scelto ed, in particolare, la scelta del macrostile definisce uno strumento di basso, uno strumento d'accordo e uno strumento per la melodia. Dalla tabella sottostante si nota tuttavia come ad ogni macrostile siano assegnati più di uno strumento di basso, più di uno strumento d'accordo e più di uno strumento melodico: la scelta di uno strumento piuttosto di un altro all'interno dello stesso macrostile può essere arbitraria, casuale o dettata da particolari esigenze di assonanza.

Tabella strumenti per macrostili

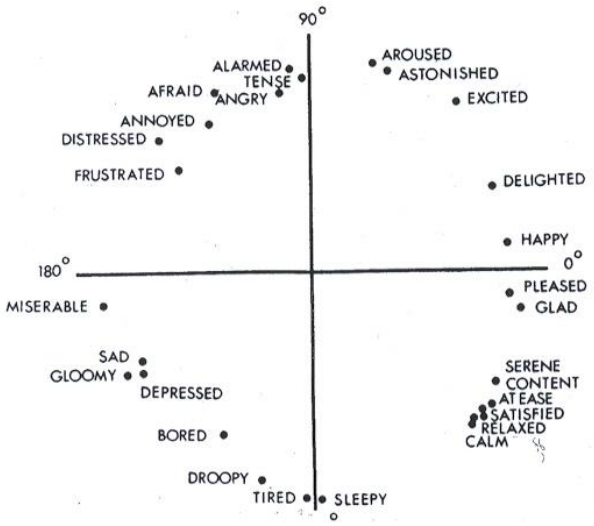
Macrostyle	Default bass instruments	Default chord instruments	Default melody instruments
Rock	34, 35, 37	19, 21	29, 30, 31
Pop	35, 39, 40	47, 51, 53	3, 65, 53
Latino	33, 34, 36	3, 24, 49	12, 13, 45
Jazz	33, 34	43, 45, 36	27, 57, 72
Classic	44, 42	42, 47, 52	7, 8, 49
Folk	33, 34, 36	21, 24, 43	23, 25, 41

3.5. Stili musicali ed etichette emozionali

Una volta selezionato un macrostile dunque, il programma comincia a valutare in continuazione i parametri di attenzione e meditazione dell'utente e, in particolare, in base ai parametri di attenzione, esso varia più volte durante la canzone lo stile interno (il numero di volte che lo stile viene cambiato può essere settato nel programma o essere deciso di default). Infatti, come si vede dalla tabella, ad ogni macrostile sono stati assegnati 5 stili interni, i quali hanno una soglia entro il quale il parametro di attenzione attiva quel determinato stile e non gli altri 4.
Ad ogni stile è poi associata una determinata etichetta emozionale, perché si presume che, una volta scelto il macrostile iniziale, il parametro di attenzione possa variare in base allo stato d'animo dell'utente. In sostanza si vuole creare un filo conduttore fra lo stato emozionale del giocatore e quello che sta ascoltando in modo da fargli capire come il suo stato d'animo interiore può influenzare i suoi parametri di attenzione e, quindi, anche il tipo di musica generata.
A tal proposito si potrebbe decidere, in futuro, di cambiare le etichette usando la concettualizzazione delle emozioni così come esposta dallo psicologo Russell che pone le diverse

etichette in un piano cartesiano x-y chiamato valence-arousal space (vedi *J. A. Russell, "A circumplex model of affect," J. Personality Social Psychol., vol. 39, pp. 1161–1178, 1980*).

Valence Arousal Space



4. Implementazione Core del prototipo

In questo prototipo di Brain Music, per motivi di debug si sono distinti due modi diversi di far funzionare il metodo principale che genera la canzone a partire dal proprio EEG semplificato e cioè un metodo in real-time (in realtà, come si vedrà, a causa delle limitazioni attuali di Impro-Visor, sarà solo un real-time simulato) e un metodo via log, in cui cioè non vengono utilizzate le cuffie ma i parametri di attenzione e meditazione vengono letti direttamente da un file di testo. Infatti, per motivi chiaramente pratici, si sono predisposti anche dei metodi particolari in grado di leggere da un file di testo dei valori di attenzione e meditazione come se si stesse leggendo in tempo reale dalle cuffie. In particolare un file di testo che simula il comportamento delle onde cerebrali risulta essere scritto in questo modo:

[timestamp]: attention, meditation, delta,theta,alpha1,alpha2,beta1,beta2,gamma1,gamma2

Ad esempio:
[11:02:38]: 70, 16, 6653.00, 4144.00, 504.00, 310.00, 1001.00, 474.00, 185.00, 21.00

4.1. Classi di definizione macrostili e strumenti

Per prima cosa si analizzano qui tutti i sistemi utilizzati per la definizione degli stili e dei macrostili dal punto di vista del codice Java. Tutto il codice preposto alla memorizzazione dei dati che riguardano le associazioni fra macrostili e stili e fra macrostili e strumenti nonché le routine di lettura del file di log sono inserite nel pacchetto chiamato brainmusic. In particolare si hanno le seguenti classi:

- ***BMStyle***: contiene tutti i metodi per creare un nuovo stile. Andando a vedere il costruttore più completo, questo assegna ad ogni stile:
    - STRING: nome dello stile
    - INT: soglia inferiore di attenzione per scegliere quel particolare stile
    - INT: soglia superiore di attenzione per scegliere quel particolare stile
    - STRING: etichetta emozionale in italiano
    - STRING: etichetta emozionale in inglese
    - INT: indice di Impro-Visor associato a quel particolare stile

  Inoltre, assieme ai normali metodi di manipolazione delle variabili per ogni stile, vi è il metodo *getStyleIndex()*, particolarmente importante perché restituisce l'indice dello stile

utilizzato da Impro-Visor associato a quel particolare stile da cui il metodo viene richiamato. Ad esempio se si era precedentemente creato lo stile March in questo modo:

BMStyle March = new BMStyle("march",45,59,"disteso","At ease",35);

allora la chiamata *March.getStyleIndex()* restituirà il valore intero 35.

- ***BMMacrostyle***: incorpora tutti i metodi per creare un nuovo macrostile. A parte i diversi costruttori che si differenziano per il numero di parametri che ricevono in ingresso, da notare soprattutto i seguenti metodi:
    - *CreateBMStyles*: metodo che riceve in ingresso i nomi dei 5 stili, le etichette emozionali a loro assegnate in italiano ed inglese (sotto forma di array) e l'array di indici di Impro-Visor relativi ai 5 stili. Dato che gli upperBound e i lowerBound sono predeterminati in ogni macrostile e non cambiano da macrostile a macrostile, possono essere inseriti direttamente nel corpo del metodo come valori interi costanti e non come parametri formali del metodo da ricevere ogni volta.
      Il metodo quindi si occupa di creare tutti gli stili per un determinato macrostile.
    - *GetBMStyle(int attention)*: metodo che, dato il parametro di attenzione ricevuto in ingresso, cerca all'interno del macrostile lo stile associato a quel particolare valore di attenzione, controllando i lowerBound e gli upperBound di ogni stile. Quello che viene dato in output è un oggetto BMStyle.

```
public BMStyle GetBMStyle(int attention)
    {
        BMStyle toRet = new BMStyle();
        for (int i = 0; i< style.size(); i++)
        {
        int foundlBound = style.elementAt(i).getlBound();
        int founduBound = style.elementAt(i).getuBound();

        if (attention <= founduBound && attention >= foundlBound)
            {
            toRet = (BMStyle)style.elementAt(i);
            break;
            }
        }
        return toRet;
    }
```

- *GetBMStyleIndex(int attention)*: come il metodo precedente, ma viene dato in output direttamente l'indice Impro-Visor associato allo stile cercato.
- *setInstruments()*: ricevuti in ingresso i 3 array degli indici degli strumenti di Impro-Visor (sono 3 perché riguardano strumento di basso, melodia e accordo), il metodo assegna tali strumenti al macrostile.
- *getRandomBassInst()*: metodo che restituisce l'indice di uno degli strumenti di basso assegnati a quel particolare macrostile, scegliendo in modo casuale.
- *getRandomChordInst()*: come il metodo sopra, ma restituisce l'indice di uno degli strumenti d'accordo.
- *getRandomMelodyInst()*: come il metodo sopra, ma restituisce l'indice di uno degli strumenti di melodia.

- ***BMMacrostyles_vect***: riempie i vettori dei macrostili con le seguenti informazioni, seguendo il pattern definito nel costruttore della classe precedente:
  - STRING: nome del macrostile
  - INT: soglia di meditazione superiore per scegliere quel macrostile
  - INT: soglia di meditazione inferiore per scegliere quel macrostile
  - Array di INT: codici di Impro-Visor relativi agli stili assegnati a quel macrostile
  - Array di STRING: nomi assegnati ai diversi stili del precedente array
  - 3 array di INT: codici di Impro-Visor relativi agli strumenti di basso, accordo e melodia assegnati a quel macrostile
  - 2 array di STRING: etichette emozionali in italiano e in inglese relative ai differenti stili definiti precedentemente per quel macrostile

Una volta creati i 6 macro-vettori, essi stessi vengono inseriti in un altro array "macros" che raggruppa tutti i macrostili.

Esempio di creazione di un macrostile

```
BMMacrostyle pop = new BMMacrostyle("Pop", 39, 20);
String[] lab_pop_IT = new String[]{"Calmo","Disteso","Disteso","Teso","Eccitato"};
String[] lab_pop_EN = new String[]{"Calm","At ease","At ease", "Tense","Excited"};
int[] ind_pop = new int[] {38,45,39,19,21};
Vector<BMStyle> vec_pop = pop.CreateBMStyles("No-style","Pop-blues","Pedal-bass-1","Funk-disco","Hits-0",
        lab_pop_EN,lab_pop_IT,ind_pop);
pop.assignStyleVector(vec_pop);
int[] pop_bass = new int[] {35,29,40};
int[] pop_chord = new int[] {47,51,53};
int[] pop_melody = new int[] {3,65,53};
pop.setInstruments(pop_bass,pop_chord,pop_melody);
```

Nella classe è inoltre presente anche il metodo *getBMMacro(int meditation)* che, dato un parametro di meditazione, restituisce il giusto vettore macrostile all'interno dell'array macros a seconda del parametro di meditazione dato nel metodo. Questo metodo è

ovviamente fondamentale in fase di recupero del giusto macrostile una volta che si è letta la meditazione dell'utente ad inizio canzone.

Codice metodo getBMMacro

```
public BMMacrostyle getBMMacro(int meditation)
{

    BMMacrostyle toRet = new BMMacrostyle();
    for (int i = 0; i< macros.size(); i++)
    {
        int founduBound = macros.elementAt(i).getuBound();
        int foundlBound = macros.elementAt(i).getlBound();
        if (meditation <= founduBound && meditation >= foundlBound)
        {
        toRet = (BMMacrostyle)macros.elementAt(i);
        break;
        }
    }
    toRet.print();
    return toRet;
}
```

- ***BMMng***: classe che incorpora i dettagli tecnici e i processi di lettura del file log. In particolare i seguenti metodi sono di particolare interesse:
    - *query(long T)*: metodo che salva il flusso d'onda ricevuto durante l'intervallo di tempo T. Tale metodo, ininfluente per la lettura real-time, risulta invece fondamentale per la lettura di debug da log in quanto, una volta definito l'intervallo di tempo entro il quale si vogliono leggere i valori, i valori vengono inseriti in un oggetto BMWaves dal quale sono di più facile lettura ed utilizzo.
    - *read(String filename)*: metodo fondamentale per la lettura del log in quanto apre lo scanner per leggere il file di testo.
    - *processLine(String aline)*: metodo che, data una riga del file di log, la processa dividendola nei diversi segnali che poi vengono inseriti nell'oggetto BMWaves.

4.2. Metodo di generazione canzoni da EEG

Prima di entrare nella trattazione del cuore pulsante di tutto il progetto, è necessario fare una premessa importante sul sistema di lettura in tempo reale dei parametri di attenzione e meditazione. Infatti, se la lettura delle onde cerebrali in tempo reale non dà nessun tipo di problema, diverso è il discorso quando si cerca di improvvisare parti di una canzone in tempo reale con Impro-Visor.

Infatti, a causa di una limitazione del codice sorgente del programma, lo stesso Impro-Visor non è in grado di costruire le note musicali e il pezzo musicale finchè riceve nuovi dati in input (come ad esempio dei nuovi stili da utilizzare nello spezzone successivo). Si era pensato perciò di procedere in uno pseudo-real-time in cui si generavano piccoli spezzoni di canzone di durata prefissata (ad esempio 5 secondi) e poi, finchè Impro-Visor generava il secondo spezzone, il primo veniva fatto ascoltare e poi gli veniva accodato il secondo finchè Impro-Visor generava il terzo e così via. Purtroppo anche quest'approccio è fallito perché Impro-Visor ha bisogno di conoscere in anticipo tutti gli stili adoperati nella canzone in modo da poter fare correzioni varie all'interno della canzone stessa, senza contare che la creazione di mini-canzoni provocherebbe spezzoni tutti diversi e senza nessuna consequenzialità.
Il problema fondamentale qui sta nel fatto che Impro-Visor è nato come software di supporto alla composizione, per cui incorpora senza problemi metodi di acquisizione in real-time della musica suonata da uno strumento collegato al PC, mentre incontra grosse difficoltà quando deve essere lui stesso a generare la musica finchè la stessa viene eseguita e non è ancora stata completata (per rendere l'idea, è come se si chiedesse ad uno scrittore di scrivere un libro per capitoli che vengono direttamente stampati via via che vengono completati, senza la possibilità per lo scrittore di revisionarli né tanto meno di rileggere tutto il libro a fine stesura e modificarlo).
In attesa dello sviluppo di una nuova versione di Impro-Visor che permetta ciò, si è deciso di ovviare a questo problema con un compromesso e cioè di generare tutta la canzone finchè un'altra viene ascoltata: in sostanza, in questo modo, si fa ascoltare all'utente la canzone generata dal suo cervello nei 2-3 minuti precedenti, mentre, nel frattempo, il programma sta raccogliendo i dati delle sue EEG e sta elaborando una nuova canzone da fargli ascoltare successivamente. Questo approccio prevede ovviamente che sia lasciato del tempo all'inizio del gioco per generare la prima canzone finchè l'utente non sente ancora nulla.
Seguendo dunque quest'approssimazione si vanno ora ad analizzare i due metodi principali di generazione della musica a partire dalle EEG semplificate dell'utente. Entrambi i metodi sono contenuti nella classe *notate.java* di Impro-Visor e non sono altro che una modifica del metodo *generate()* originale del programma.

- ***generateFromLog():*** questo è il metodo che gestisce la generazione di musica improvvisata a partire da un file di log che contiene i parametri di attenzione e meditazione registrati precedentemente dal MindSet ma che, tuttavia, non è attualmente in uso. I passaggi sono dunque i seguenti:
    1. Viene inizializzata la lettura del log tramite il metodo *initializeBrainMusic()* che crea un nuovo oggetto BMWaves e apre in lettura il file di log.
    2. Vengono inizializzati i necessari parametri (*beatinbar, slotsinbar*, ecc).
    3. Viene preso un intervallo di tempo ragionevole (in questo caso 200 millisecondi) entro il quale si va a scandagliare il timestamp all'interno del log per prelevare il

valore iniziale di meditazione per la scelta del macrostile. Infatti, a questo punto, viene richiamata la funzione *getSignal("meditation")* sull'oggetto BMWaves che restituisce i valori di meditazione all'interno del suddetto periodo di 200 millisecondi, se ne fa la media con la funzione *mean()* per ottenere un unico valore intero di meditazione *mean_MED* compreso ovviamente fra 0 e 100.

4. A questo punto viene inizializzato un vettore c che contiene tutte le specifiche dei vari macrostili presenti nel gioco e si ottiene il macrostile corrispondente al valore medio di meditazione attuale attraverso il metodo *getBMMacro(mean_MED)* assegnandolo alla variabile di classe *bm_Macrostyle*.
5. Viene richiamato il metodo *setPrefsDialog()* di Impro-Visor che permette la modifica delle preferenze per la canzone che si sta per generare; in questo caso vengono settate le preferenze relative ai diversi strumenti musicali del macrostile, scegliendo casualmente uno degli strumenti fra quelli disponibili per il macrostile dato.
6. Si inizializzano le tre grammatiche di base con i relativi valori di soglia.
7. Inizia il processo di generazione vero e proprio con un ciclo while che si conclude quando non ci sono più dati nel log o quando si è raggiunta la durata massima della canzone.
8. Si decide dunque di analizzare il log scandagliandolo per 2000 millisecondi (si è calcolato che, in questo modo, una canzone di 3 minuti contiene una dozzina circa di variazioni stilistiche) ottenendo ogni volta il valore medio di meditazione e attenzione.
9. Se si tratta del primo ciclo si aggiunge un'introduzione (intro) basata sul primo stile associato al parametro di attenzione corrente, se invece si tratta dell'ultimo ciclo si aggiunge una conclusione (outro) sempre basata sull'ultimo stile.
10. Ancora una volta si aprono le preferenze di Impro-Visor e si va a settare lo stile di questo spezzone attraverso il suo indice.
11. Si decide, in base al parametro di meditazione, se utilizzare la scala armonica in tonalità maggiore o in tonalità minore.
12. Si generano gli accordi attraverso una serie di operazioni complesse già utilizzate nel metodo originale *generate()* di Impro-Visor.
13. Se non si tratta dell'intro o dell'outro, si seleziona una grammatica in base al parametro di attenzione attuale.
14. Seguendo tutte le impostazioni dal punto 9 al punto 13 si può finalmente generare il primo spezzone di canzone relativo ai parametri di attenzione e meditazione trovati nel log per quel periodo di tempo.

15. Si aggiornano le varie variabili necessarie a tenere conto dell'avanzamento della canzone da una battuta all'altra e quindi, se le condizioni sono ancora vere, si ricomincia con un altro ciclo dall'operazione 8 all'operazione 15.
16. Quando la generazione è completata viene creato un file .midi contenente l'intera canzone generata e salvato in una specifica cartella del programma attraverso il comando *exportToMidiBM*, comando nativo di Impro-Visor, ma personalizzato per il progetto. Assieme al file .midi viene creato anche un file .bmi con lo stesso nome della traccia audio, un metadato contenente il nome del macrostile utilizzato (vedi spiegazioni nel paragrafo seguente).

- ***generateRealTime()***: questo è il metodo che gestisce la generazione di musica improvvisata a partire da un file di log che contiene i parametri di attenzione e meditazione registrati precedentemente dal MindSet che, tuttavia, non è attualmente in uso. I passaggi sono dunque i seguenti:
    1. Viene aperta la connessione con il MindSet attraverso il comando *HeadsetConnect*, in questo caso con la porta COM8 e con un baudrate di 57600.
    2. Vengono inizializzati vari i necessari parametri (*beatinbar, slotsinbar*, ecc).
    3. Viene preso il valore di meditazione medio fra i 10 valori che vengono letti dal MindSet nei primi 10 secondi, attraverso il metodo *GetMeanMeditation(10)*. Si ottiene così un unico valore intero di meditazione *mean_MED* compreso ovviamente fra 0 e 100.
    4. A questo punto viene inizializzato un vettore c che contiene tutte le specifiche dei vari macrostili presenti nel gioco e si ottiene il macrostile corrispondente al valore medio di meditazione attuale attraverso il metodo *getBMMacro(mean_MED)* assegnandolo alla variabile di classe *bm_Macrostyle*.
    5. Viene richiamato il metodo *setPrefsDialog()* di Impro-Visor che permette la modifica delle preferenze per la canzone che si sta per generare; in questo caso vengono settate le preferenze relative ai diversi strumenti musicali del macrostile, scegliendo casualmente uno degli strumenti fra quelli disponibili per il macrostile dato.
    6. Si inizializzano le tre grammatiche di base con i relativi valori di soglia.
    7. Inizia il processo di generazione vero e proprio con un ciclo while che si conclude quando si è raggiunta la durata massima della canzone (in questa implementazione prototipale non è qui prevista un'interruzione della generazione causata da malfunzionamento del MindSet).

8. Si procede quindi alla lettura contemporanea dei valori di meditazione e attenzione in intervalli di 5 secondi e si assegnano a due variabili di nome rispettivamente mean_MEDITATION e mean_ATTENTION.
9. Se si tratta del primo ciclo si aggiunge un'introduzione (intro) basata sul primo stile associato al parametro di attenzione corrente, se invece si tratta dell'ultimo ciclo si aggiunge una conclusione (outro) sempre basata sull'ultimo stile.
10. Ancora una volta si aprono le preferenze di Impro-Visor e si va a settare lo stile di questo spezzone attraverso il suo indice.
11. Si decide, in base al parametro di meditazione, se utilizzare la scala armonica in tonalità maggiore o in tonalità minore.
12. Si generano gli accordi attraverso una serie di operazioni complesse già utilizzate nel metodo originale *generate()* di Impro-Visor.
13. Se non si tratta dell'intro o dell'outro, si seleziona una grammatica in base al parametro di attenzione attuale.
14. Seguendo tutte le impostazioni dal punto 9 al punto 13 si può finalmente generare il primo spezzone di canzone relativo ai parametri di attenzione e meditazione rilevati per quel periodo di tempo.
15. Si aggiornano le varie variabili necessarie a tenere conto dell'avanzamento della canzone da una battuta all'altra e quindi, se le condizioni sono ancora vere, si ricomincia con un altro ciclo dall'operazione 8 all'operazione 15.
16. Quando la generazione è completata ci si disconnette dal MindSet e viene creato un file .midi contenente l'intera canzone generata e salvato in una specifica cartella del programma attraverso il comando *exportToMidiBM*, comando nativo di Impro-Visor, ma personalizzato per il progetto. Assieme al file .midi viene creato anche un file .bmi con lo stesso nome della traccia audio, un metadato contenente il nome del macrostile utilizzato (vedi spiegazioni nel paragrafo seguente).

Qui di seguito viene fornita una tabella riassuntiva delle operazioni svolte dai due metodi così che si possano anche confrontare fra di loro i sistemi di generazione da Log e di generazione Real Time.

Tabella confronto operazioni generazione

OPERAZIONE	generateFromLog	generateRealTime
OPERAZIONE 1	Inizializzazione BMWaves	Apertura connessione HeadSet
OPERAZIONE 2	Inizializzazione parametri	Inizializzazione parametri
OPERAZIONE 3	Recupero primo parametro meditazione dal log per scelta macrostile	Recupero primo parametro meditazione dai primi 10 secondi per scelta macrostile
OPERAZIONE 4	Assegnazione macrostile	Assegnazione macrostile
OPERAZIONE 5	Assegnazione strumenti musicali	Assegnazione strumenti musicali
OPERAZIONE 6	Inizializzazione grammatiche	Inizializzazione grammatiche
OPERAZIONE 7	Inizio generazione	Inizio generazione
OPERAZIONE 8	Recupero valori meditazione e attenzione medi dal log	Recupero valori meditazione e attenzione medi in intervalli di 5 secondi
OPERAZIONE 9	Assegnazione stile	Assegnazione stile
OPERAZIONE 10	Scelta scala armonica	Scelta scala armonica
OPERAZIONE 11	Generazione accordi	Generazione accordi
OPERAZIONE 12	Scelta grammatica	Scelta grammatica
OPERAZIONE 13	Aggiornamento variabili	Aggiornamento variabili
OPERAZIONE 14	Fine generazione	Fine generazione
OPERAZIONE 15	Salvataggio file .midi e .bmi	Salvataggio file .midi e .bmi

4.3. Gestione della Playlist delle canzoni

Come già specificato all'inizio, il progetto Brain Music prevede di poter salvare le canzoni generate con le proprie onde cerebrali e di riascoltarle in qualsiasi momento finchè il gioco rimane aperto, ma anche di salvarle in un supporto esterno o nell'hard-disk della piattaforma sul quale sta girando. Per questo motivo si è reso necessaria l'implementazione di tre classi specifiche per la gestione delle normali operazioni audio connesse alla possibilità di avviare/fermare una canzone, ecc. Inoltre una delle tre classi implementa tutte le funzioni di gestione della PlayList compresa la gestione di alcuni metadati con estensione .bmi che vengono creati assieme alla canzone e che contengono al loro interno l'indicazione sul macrostile che era stato associato a quella particolare canzone.
Purtroppo in questo prototipo la gestione si ferma a files di tipo .midi e dunque risulterà del tutto inutile nel progetto finale. In ogni caso, le tre classi in questione sono:

1. ***MainMusic_Player***: utilizzando i pacchetti *java.sound*, questa classe incorpora alcuni metodi per fornire una sorta di player per le canzoni. In particolare, una volta

inizializzato un file di tipo *MainMusic_Player* avendogli fornito come parametro la cartella in cui si trovano le canzoni, si possono usare i seguenti metodi:

a. *InizializeSong(String s)*: riceve il nome del file .midi della canzone, apre il sequencer midi di Java e avvia la riproduzione della canzone.

```
Sequence sequence;
Sequencer sequencer;
Synthesizer synth;


public void InitializeSong(String s)
{
    File f = new File(path + s);
    try
    {
    // Get sequence from file
    sequence = MidiSystem.getSequence(f);

    // Create a sequencer for the sequence
    sequencer = MidiSystem.getSequencer();
    sequencer.open();
    sequencer.setSequence(sequence);

    synth = MidiSystem.getSynthesizer( );
    synth.open( );

    // Start playing
    play();
    } catch (IOException e) {
    } catch (MidiUnavailableException e) {
    } catch (InvalidMidiDataException e) {
    }
}
```

b. *StopPlay()*: metodo che o ferma l'esecuzione della canzone se è già in riproduzione oppure avvia la canzone se era stata messa precedentemente in pausa.
c. *play()*: metodo che avvia la riproduzione della canzone.
d. *stop()*: metodo che interrompe la riproduzione della canzone.
e. *changeVolume(int midiVolume)*: metodo che cambia il volume del sintetizzatore midi portando ad abbassare il volume globale della canzone di un valore uguale al midiVolume fornito.

```
public void changeVolume(int midiVolume)
{
    if( synth.getDefaultSoundbank() == null )
    {
            // HARDWARE SYNTHESIZER
            try
            {
                ShortMessage volumeMessage = new ShortMessage();
                for( int i = 0; i < 16; i++ )
                {
                    volumeMessage.setMessage(ShortMessage.CONTROL_CHANGE,i,7,midiVolume);
                    MidiSystem.getReceiver().send( volumeMessage, -1 );
                }
            }
            catch( InvalidMidiDataException imde )
            {
                System.err.println( "Invalid MIDI data." );
                return;
            }
            catch( MidiUnavailableException mue )
            {
                System.err.println( "MIDI unavailable." );
                return;
            }
    }
    else
    {
          if (midiVolume != 0)
          {
                // SOFTWARE SYNTHESIZER:
                javax.sound.midi.Track[] tracks = sequence.getTracks();
                for(int i = 0; i < tracks.length; i++)
                {
                    sequencer.setTrackMute(i,false);
                }
                MidiChannel[] channels = synth.getChannels();

                for( int c = 0; channels != null && c < channels.length; c++ )
                {
                    channels[c].controlChange(7, midiVolume);
                }
          }
          else
          {
                javax.sound.midi.Track[] tracks = sequence.getTracks();
                for(int i = 0; i < tracks.length; i++)
                {
                    sequencer.setTrackMute(i,true);
                }
          }
    }
}
```

2. ***Notation_Functions***: classe che, oltre ad incorporare alcune funzioni di debug, implementa anche alcune funzioni di supporto che vengono utilizzate dal programma per conoscere i macrostili successivi o precedenti, gli strumenti successivi o precedenti da mostrare nella schermata principale di gioco. In particolare si possono notare i seguenti metodi:
    a. *getPreviousMacrostyle()*: metodo che fornisce il nome del macrostile che era stato usato per la canzone precedente a quella attualmente in riproduzione.
    b. *getPreviousInstruments()*: metodo che fornisce un array con i nomi degli strumenti che erano stati usati per la canzone precedente a quella attualmente in riproduzione.
    c. *VolChange(int volume)*: metodo che incorpora ad un livello superiore la funzionalità di cambio volume già presente nella classe MainMusic_Player.
    d. *PauseSong()*: metodo che incorpora ad un livello superiore il metodo *StopPlay()* già presente nella classe MainMusic_Player.
3. ***PlayList_Functions***: classe che incorpora tutti i metodi necessari alla riproduzione delle canzoni nella PlayList dello Start Menu, nonché i metodi per fornire tutte le informazioni sulle canzoni stesse. Ma prima di andare a vedere in dettaglio i metodi, è bene soffermarsi sul costruttore della classe stessa che, ricevuta la stringa con il percorso alla cartella in cui sono contenute le canzoni, esegue le seguenti operazioni:
        i. Inserisce in un vettore la lista di tutti i files .mid o .midi scartando tutti gli altri tipi di files.
        ii. Inizializza alcuni vettori di stringhe che conterranno tutte le informazioni sulle canzoni salvate nella cartella (vedi punto successivo).
        iii. Per ogni canzone recupera e salva in array dedicati:
            1. Nome del file
            2. Data di creazione del file
            3. Macrostile associato (tramite la lettura del file .bmi)
            4. Nome del file compreso il percorso
            5. Durata della canzone (formato: “MIN:SEC”)

    Venendo ai metodi della classe si possono trovare i seguenti metodi necessari per la popolazione della PlayList dal punto di vista grafico e d’interfaccia e per il funzionamento della stessa:

    - *Get_SongList()*: metodo che restituisce la lista con il nome delle canzoni presenti nella PlayList.
    - *Get_SongDurations()*: metodo che restituisce la lista con la durata delle canzoni presenti nella PlayList.

- *Get_SongDates():* metodo che restituisce la lista con la data di creazione delle canzoni presenti nella PlayList.
- *Get_SongMacros():* metodo che restituisce la lista con il macrostile delle canzoni presenti nella PlayList.
- *InizializeSong:* metodo che permette di avviare la riproduzione di una canzone fra quelle presenti nella PlayList e di mostrare il relativo progresso temporale della canzone in uno spazio dedicato.
- *StopPlay():* metodo che o interrompe l'esecuzione della canzone se sta già suonando oppure avvia la riproduzione se la canzone era stata messa precedentemente in pausa.
- *play():* metodo che avvia la riproduzione della canzone.
- *stop():* metodo che interrompe la riproduzione della canzone.
- *changeVolume(int midiVolume):* metodo che cambia il volume del sintetizzatore midi portando ad abbassare il volume globale della canzone di un valore uguale al midiVolume fornito.
- *getSongDuration(String name):* metodo che fornisce la durata della canzone il cui nome viene specificato nel parametro formale del metodo.
- *getSongFullName(String name):* metodo che fornisce il nome intero (compreso il percorso) della canzone il cui nome viene specificato nel parametro formale del metodo.

5. Implementazione grafica del prototipo

5.1. Introduzione concept grafico originale

Il gioco finale Brain Music prevede l'utilizzo di due schermate principali:

1. *Menu principale*
2. *Schermata di gioco*

Entrambe devono essere interamente navigabili con l'utilizzo degli strumenti a disposizione nelle varie piattaforme (joypad, mouse,ecc) e devono contenere pulsanti e strumenti utili allo svolgimento di diverse funzioni:

1. Il gioco deve salvare automaticamente un certo numero di canzoni generate all'interno della memoria interna della piattaforma
2. Il gioco deve permettere all'utente di interrompere l'acquisizione delle EEG e dunque la composizione della canzone in qualsiasi momento
3. Il gioco deve permettere all'utente di mettere in pausa l'acquisizione delle EEG e riprenderla in qualsiasi momento
4. Il gioco deve preoccuparsi di rendere noto all'utente quando le cuffie hanno problemi o non prendono il segnali
5. Il gioco deve dare tutti i feedback visivi e sonori necessari per il divertimento e lo svago connesso con il particolare modo di intrattenimento scelto
6. Il gioco deve permettere all'utente di riascoltare i brani generati precedentemente ed eventualmente esportarli su un diverso supporto
7. Il gioco deve mostrare lo stile successivo un po' prima che sia finita la canzone
8. Il gioco deve fornire i normali pulsanti relativi alle funzioni utilizzate e al sistema in uso (volume per l'audio, ecc)

Interfacce principali di gioco

5.2. Metodi di ridimensionamento della grafica

Uno dei problemi da affrontare nella realizzazione dell'interfaccia grafica del software riguarda la capacità dell'interfaccia stessa di adattarsi alle diverse risoluzioni video e ai diversi aspect ratio, cioè il rapporto matematico fra larghezza e altezza di un'immagine. In particolare il gioco deve essere in grado di supportare i seguenti rapporti:

- *4:3*: il formato più conosciuto ed utilizzato fin dalla nascita delle TV, nonché ormai soppiantato dai moderni formati widescreen
- *16:9*: il formato widescreen per eccellenza alla base dell'alta definizione (HDTV), permette di vedere il 33% di schermo in più rispetto al formato 4:3
- *16:10*: dal 2003 il formato widescreen più comune nei monitor LCD e laptop, ormai soppiantato completamente dal 16:9.

Si è scelto di affrontare il problema tenendo come punto di riferimento costante un'immagine di sfondo e cercando di ridimensionare tutti gli oggetti dell'interfaccia in relazione al rapporto di altezza e larghezza dello schermo utilizzato in quel momento. L'immagine di sfondo, invece, oltre a venire ridimensionata, viene anche eventualmente tagliata superiormente ed inferiormente per permettere una migliore visualizzazione globale.
Nella foto sottostante si può notare un raffronto visivo delle differenze nelle due modalità di visualizzazione nonché l'immagine di sfondo presa da riferimento di cui si è parlato sopra.

Differenze 4:3 e 16:9

In particolare, tutte le routines di ridimensionamento degli oggetti del gioco sono state inserite in un'unica classe denominata *ResizeManager*. Tale classe presenta un costruttore che recupera l'altezza e la larghezza in pixel dello schermo tramite le due funzioni standard della classe Toolkit del pacchetto java awt, mentre inizializza al valore 1 due variabili che saranno largamente utilizzate nel prosieguo ed andranno ad indicare un rapporto di altezza ed un rapporto di altezza (le specifiche su come vengono calcolati tali rapporti sarà chiarificato in seguito).

```
public ResizeManager()
{
        screen_width = Toolkit.getDefaultToolkit().getScreenSize().getWidth();
        screen_height = Toolkit.getDefaultToolkit().getScreenSize().getHeight();
        width_ratio = 1;
        height_ratio = 1;
}
```

Passando ai metodi si dà qui una panoramica generale soffermandosi sui metodi più particolari e cercando di raggruppare i metodi simili benchè con funzionalità o parametri leggermente diversi:

- *getWidthRatio()* e *getHeightRatio():* come dice il nome sono due metodi di semplice recupero delle due variabili che indicano il rapporto di larghezza ed altezza rispettivamente;
- *getScreenWidth()* e *getScreenHeight():* due metodi per il recupero della larghezza e dell'altezza dello schermo rispettivamente;
- *getScreenRatio():* metodo per conoscere la risoluzione dello schermo nel quale si sta giocando attualmente. Restituisce 0 se la risoluzione è 4:3, 1 se la risoluzione è 16:10 e 2 se la risoluzione è 16:9;
- *getWidthRatio(ImageIcon icon)* e *getHeightRatio(ImageIcon icon):* due metodi per calcolare il rapporto di larghezza e di altezza rispetto al file grafico che viene fornito; questi due metodi sono particolarmente importanti dato che ridefiniscono le variabili *width_ratio* ed *height_ratio*. In sostanza il metodo controlla le dimensioni del file grafico dato come parametro e se il file grafico ha una dimensione maggiore di quella dello schermo, viene restituito il rapporto fra la dimensione dell'icona e la dimensione dello schermo.
- *setWidthRatio(double d)* e *setHeightRatio(double d):* due metodi per assegnare alle variabili *width_ratio* e *height_ratio* un valore numerico definito nell'argomento del metodo.
- *resize(ImageIcon icon):* metodo standard per ridimensionare il file grafico dato in ingresso in base al *width_ratio*. In pratica viene estratta la dimensione in larghezza del file grafico dato, viene calcolata una nuova larghezza dividendo la larghezza attuale per il *width_ratio* (variabile di solito precedentemente calcolata tramite il metodo *getWidthRatio*) e infine viene creata una nuova immagina (icona) ridimensionata secondo la nuova larghezza utilizzando la funzione *getScaledInstance* di java.awt che garantisce un ottimo risultato di

ridimensionamento utilizzando il parametro *SCALE_SMOOTH* per non rovinare i contorni dell'icona stessa.

```
public ImageIcon resize(ImageIcon icon)
{
    double icon_width = (double) icon.getIconWidth();
    Image img = icon.getImage();
    int new_width = (int)(icon_width / width_ratio);
    Image new_img = img.getScaledInstance(new_width, -1,  java.awt.Image.SCALE_SMOOTH);
    icon = new ImageIcon(new_img);
    return icon;
}
```

- *setDimensions(…) e metodi successivi*: i metodi della classe che seguono servono tutti a ridimensionare i pulsanti-funzione del gioco e a posizionarli nel punto esatto richiesto dal gioco. In particolare ogni metodo richiede il file grafico del pulsante (*ImageIcon icon*), il componente su cui viene disegnato il file grafico (cioè l'oggetto, il pulsante vero e proprio) e alcune coordinate per indicare la posizione relativa dell'icona rispetto allo spazio in cui il componente è incorporato.

```
public void setDimensions(ImageIcon icon, JComponent comp, int hor, int ver)
{
    comp.setMaximumSize(new java.awt.Dimension(icon.getIconWidth(), icon.getIconHeight()));
    comp.setMinimumSize(new java.awt.Dimension(icon.getIconWidth(), icon.getIconHeight()));
    comp.setPreferredSize(new java.awt.Dimension(icon.getIconWidth(), icon.getIconHeight()));
    comp.setBounds((int)(hor / width_ratio),(int) (ver /width_ratio),
            icon.getIconWidth(), icon.getIconHeight());
}
```

Si noti come in tutti i metodi venga utilizzato il *width_ratio* o l'*height_ratio* per permettere all'algoritmo di ridimensionare icona e pulsante in modo tale da mantenere un'interfaccia organica all'interno dello schermo, di qualsiasi dimensione esso sia.
In alcuni metodi infine vengono utilizzate delle costanti numeriche che indicano dei particolari punti di ancoraggio determinanti per ottenere le animazioni (vedi paragrafo successivo) o per mantenere una proporzionalità esatta fra gli elementi durante il ridimensionamento.
A conclusione di questo paragrafo è giusto far notare che il ridimensionamento non coinvolge soltanto files grafici, ma anche le scritte delle schermate. Infatti tutte le scritte presenti, anche quelle nei pulsanti, devono essere localizzabili (traducibili nelle varie lingue) e perciò sono delle vere e proprie stringhe il cui font va ridimensionato di conseguenza, in modo tale da rimanere sempre leggibili e posizionati in modo corretto rispetto alla grafica.

5.3. Sistemi utilizzati per ottenere gli effetti grafici del prototipo

Per motivi di tempo insufficiente ad apprendere nuove tecnologie e di licenza non si è potuto, in fase di progettazione grafica, usufruire dei diversi script pensati per la grafica come ad esempio il JavaFx di cui si dà una breve definizione qui:

"JavaFX introduce un linguaggio di scripting dichiarativo ad oggetti, con molti riferimenti alla programmazione funzionale, estremamente pratico per lo sviluppo di applicazioni grafiche, con una sintassi simile a JavaScript e, per certi aspetti, ad ActionScript, che permette di gestire in modo semplice ed efficace le interazioni tra i controlli grafici, e rende del tutto banale la gestione delle animazioni più comuni (dissolvenze, ingrandimenti, spostamenti ecc.). JavaFX elimina quindi tutta la ridondanza tipica di Java, e rappresenta un naturale passaggio per i programmatori tradizionali dello storico linguaggio di Sun.

JavaFX permette anche di interagire senza alcuno sforzo con classi Java preesistenti. È inoltre possibile fare l'opposto: si può includere una applicazione JavaFX all'interno di un normale programma scritto in Java e Swing." (http://it.wikipedia.org/wiki/JavaFX)

Inoltre, per motivi di compatibilità e per la necessità di mantenere il progetto cross-platform si è stati costretti ad utilizzare Netbeans come IDE di sviluppo invece che, ad esempio, Eclipse, che avrebbe permesso di utilizzare le librerie grafiche del sistema operativo sul quale viene installato, rendendo però in questo modo, di fatto, impossibile mantenere la compatibilità fra le diverse piattaforme (si ricordi che il gioco deve funzionare sia in ambiente Windows che in ambiente Linux e su architetture molto diverse come quelle che possono essere le architetture di Ps3 e Xbox360). Dunque, su Netbeans, si sono potute soltanto utilizzare le classi standard di disegno fornite dal pacchetto Java e cioè *java.awt* e *javax.swing* che si distinguono per permettere l'aggiunta rispettivamente di componenti *heavyweight* (il comportamento dei componenti come rendering e triggers è affidato al sistema operativo) e componenti *lightweight* (componenti che non necessitano di allocazione di risorse native da parte della GUI del sistema operativo).
Sempre per motivi di portabilità, vista la necessità di mostrare dei video in background, si è deciso di adottare il plugin *Java Media Framework (JMF)* che permette di avviare diversi formati video ed audio all'interno di un progetto Java. Purtroppo tale plugin però supporta pochi formati video ad alta risoluzione e quindi, per questo progetto, ci si è accontentati di utilizzare il formato *MJPEG* che lavora bene per gli scopi del gioco (e cioè di mostrare un video in loop in background), ma la qualità del video stessa non è delle migliori.
Per spiegare a fondo tutti i procedimenti utilizzati per ottenere una grafica funzionale e piacevole per il gioco, si può fare riferimenti a due gruppi di classi Java ben precisi:

1. Classi di supporto grafico

Le classi di supporto grafico nel progetto sono 3:

1.1. *ResizeManager.java*

Questa classe è già stata spiegata nel precedente paragrafo: contiene tutti i metodi per ridimensionare i pulsanti in modo tale da rimanere nell'area visibile dello schermo, a qualsiasi risoluzione.

1.2. *GraphicsEffects.java*

Questa classe, utilizzando le librerie grafiche standard java chiamata *javax.swing*, si occupa di gestire alcuni effetti grafici del gioco. Attualmente ne gestisce soltanto uno e cioè la grafica dell'abbassamento o dell'innalzamento del volume audio.

In particolare viene tenuto conto del livello del volume con una variabile intera *volume_level* e viene poi disegnata la barra sul pulsante chiamato *vol_bar*. A questo punto i metodi principali da tenere in considerazione per questa classe sono *setGraphicsVolumeUp* e *setGraphicsVolumeDown* che servono a mostrare una tacca in più o una tacca in meno nella barra del volume che originariamente si trova al livello intermedio e consta di 7 tacche in totale.

Andando ad analizzare nel dettaglio il metodo *setGraphicsVolumeUp* (per il metodo *setGraphicsVolumeDown* valgono considerazioni analoghe con le dovute modifiche) si può notare come venga utilizzata una variabile booleana *can_increase* che permetta di capire se il volume è arrivato al livello massimo o possa ancora aumentare. Se non è al livello massimo l'algoritmo entra nell'*else statement* e lì viene incrementato di 1 il valore intero del volume e poi viene usato uno *switch* per scegliere il file grafico png esatto.

Chiuso il ciclo, se effettivamente c'è stato un innalzamento del volume, allora viene assegnato al pulsante il nuovo file grafico con *vol_bar.setIcon(ic)* e viene incrementato di 1 la variabile globale del volume chiamata *volume_level*.

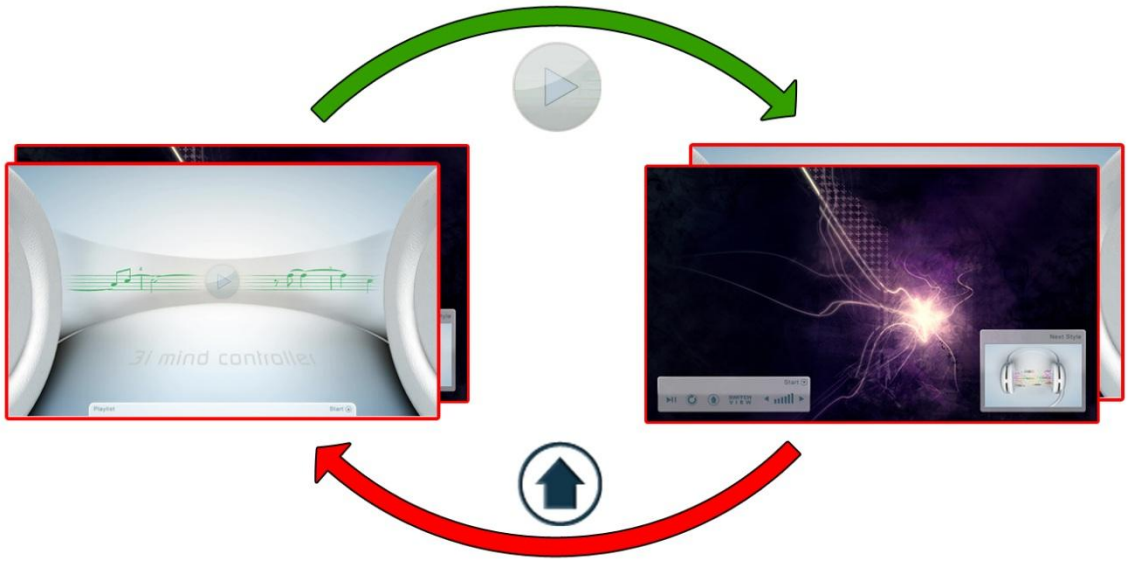
```
public void setGraphicsVolumeUp(ResizeManager man, String folder)
{
    int vol = volume_level;
    String path = "";
    boolean can_increase = false;
    if (vol >= 6)
    {
        can_increase = false;
    }
    else
    {
        vol = vol + 1;
        switch(vol)
        {
            case 1: path = "/graphics/" + folder + "/vol_01.png"; can_increase = true; break;
            case 2: path = "/graphics/" + folder + "/vol_02.png"; can_increase = true; break;
            case 3: path = "/graphics/" + folder + "/vol_03.png"; can_increase = true; break;
            case 4: path = "/graphics/" + folder + "/vol_04.png"; can_increase = true; break;
            case 5: path = "/graphics/" + folder + "/vol_05.png"; can_increase = true; break;
            case 6: path = "/graphics/" + folder + "/vol_06.png"; can_increase = true; break;
            default: path = ""; can_increase = false; break;
        }
    }

    if (can_increase)
    {
    ImageIcon ic = new javax.swing.ImageIcon(getClass().getResource(path));
    ic = man.resize(ic);
    vol_bar.setIcon(ic);
    volume_level = volume_level + 1;
    }
}
```

1.3. *Screens_Manager.java*

Prima di spiegare l'unico metodo importante di questa classe è necessario procedere alla spiegazione dell'infrastruttura che permette di passare da una schermata all'altra del gioco. Proprio infatti per le limitazioni software già spiegate sopra, si è scelto un compromesso particolare per permettere al gioco di passare agevolmente e velocemente da una schermata di gioco all'altra, evitando in questo modo fastidiosi stacchi in cui il gioco mostrerebbe schermate nere o un bruttissimo tearing. Tale compromesso prevede, in totale antitesi con la concezione standard dei linguaggi ad oggetti, di creare da subito, appena avviato il gioco, tutte le schermate necessarie (*frame*) e poi renderle invisibili o visibili a seconda del bisogno di quel momento. Ecco che dunque diventa necessaria una classe come questa per gestire agevolmente il passaggio da una schermata all'altra. In particolare, in questo prototipo di videogioco, ci si è preoccupati di creare il metodo per passare dalla schermata Player Menu alla schermata Start Menu, metodo chiamato *SwitchToStartMenu()*. Tale metodo recupera tutti i frame attivi in quel momento e cerca

il frame d'interesse, in questo caso il menu di inizio gioco. Se viene trovato allora quel frame viene reso visibile, mentre tutti gli altri vengono resi invisibili. In questo modo, con un semplice ciclo *for*, si ottiene una funzione di cambio di schermata che può essere agevolmente assegnata ad un tasto specifico del gioco.
Il motivo per cui questo approccio è stato preferito è perché creando tutti i frames all'inizio, tutta la fase di caricamento del gioco grava sull'inizio del gioco, mentre il programma non deve caricare nulla di grafico né deve creare nuovi oggetti nel passaggio da una schermata all'altra. Se si fosse deciso di creare ogni volta una nuova istanza di un frame per passare dalla schermata vecchia alla nuova, si sarebbe visto uno stacco dovuto ai tempi di caricamento della nuova istanza e non si sarebbero potute tenere in memoria agevolmente alcune impostazioni, come ad esempio il livello di audio o l'effetto grafico attualmente in uso nella schermata del Player, ecc.

Passaggio da una schermata all'altra



2. Classi dell'interfaccia di gioco

Queste sono le classi preposte alla creazione di tutta la grafica visibile nelle diverse schermate di gioco. Ogni schermata del gioco è creata da due classi, una che si occupa di creare lo sfondo e di posizionare gli oggetti su una griglia virtuale in modo tale da bloccare la loro posizione e un'altra classe che invece crea gli oggetti stessi e ne associa le relative

funzioni, nonché si preoccupa di gestire eventuali effetti grafici particolari per tali oggetti. In particolare nel prototipo attuale del gioco si possono distinguere due schermate:

2.1. Start Menu

E qui si trovano due classi:

2.1.1. *Start_Menu.java*

Classe di creazione dello sfondo e di posizionamento degli elementi. E' anche la classe che deve essere avviata (run) per far partire il gioco, dunque la classe che si preoccupa della creazione di entrambi i frame delle due schermate di questo prototipo, nonostante poi, all'atto pratico, a fine caricamento, venga mostrata soltanto la schermata principale.

Come si può notare la classe estende una struttura particolare di Java chiamata JFrame. Il JFrame è un contenitore di oggetti della classe grafica Swing (quella utilizzata per i lightweight objects) che ben si sposa con le esigenze del progetto di poter switchare velocemente da una schermata all'altra, ma soprattutto indispensabile per poter avviare un video in background. Infatti l'unico sistema conosciuto per mostrare un video in background utilizzando la JMF è quello di incorporare il lettore multimediale JMF all'interno di un JFrame e di rendere tale JFrame a tutto schermo. Non sarebbe stato possibile usando JMF, ad esempio, riuscire ad avviare un video in background nel pannello principale e poi sovrapporvi i pulsanti, cosa invece resa possibile dai JFrame che, in quanto componenti Swing, possono essere parzialmente sovrapposti da altri componenti Swing utilizzando il JLayeredPane, altro contenitore di oggetti della libreria Swing di Java.

Andando nel dettaglio e guardando il costruttore della classe si può notare che:

- viene dato un nome al Frame (indispensabile per lo switch delle schermate, come si è visto nel paragrafo precedente)
- viene creato un lettore video per la JMF
- viene ridimensionato il video in modo tale da prendere tutto lo schermo
- viene data la possibilità di chiudere il gioco premendo il tasto ESC della tastiera
- viene ottenuta la modalità Full Screen per l'intero frame
- si ricava il JLayeredPane standard dell'JFrame e vengono aggiunti tutti gli elementi necessari per la schermata Start Menu al JLayeredPane attraverso il metodo *add*
- Viene resa visibile tutta la schermata (finora tutto veniva caricato senza essere mostrato)
- Viene infine fatto partire il video (*p.start()*) dopo essersi assicurato che venga riprodotto indefinitamente, in loop, come video di background della schermata.

Si noti che il costruttore prende come argomento il percorso completo del video che si vuole avviare come video di background.
Infine il metodo *main* è quello che permette al gioco di essere avviato e crea al suo interno un thread che instanzia un nuovo oggetto Start_Menu con tutte le caratteristiche viste nel costruttore.

Interfaccia Start Menu

2.1.2. *Start_Menu_Elements*

Classe di creazione di tutti gli oggetti del menu principale che, si ricorda, deve contenere anche la possibilità di mostrare la playlist con tutte le canzoni generate dal gioco. Per questo motivo tale classe presenta anche alcuni metodi di supporto che servono a creare un'animazione a comparsa/scomparsa che permette di mostrare o nascondere la playlist premendo un tasto (tasto "Start") dell'interfaccia.
Si vanno dunque ora ad esaminare alcuni elementi del codice che si considerano i più importanti per comprendere alcuni meccanismi particolari. Si tralasceranno qui tutte le considerazioni sulle funzioni degli oggetti, concentrandosi invece solo sugli aspetti grafici degli stessi e si cercherà di dare una panoramica generale, senza entrare troppo nello specifico del codice, per non appesantire la lettura:

- **Costruttore della classe:** qui viene inizializzato l'oggetto per le funzioni della PlayList e viene creato il frame del PlayerMenu mantenendolo invisibile, in modo che stia tecnicamente in background rispetto alla schermata principale che invece viene visualizzata in quel momento.
  Viene peraltro utilizzata un'immagine particolare di default per ottenere il *width_ratio* e l'*height_ratio* relativi rispetto allo schermo utilizzato e necessari per tutte i ridimensionamenti successivi delle varie icone dei vari pulsanti.
  Viene infine caricato il file grafico dello sfondo della PlayList e inserito nella posizione standard ad inizio gioco e cioè in basso in modo tale che sia visibile soltanto il lembo superiore della finestra, quello dove si andrà a disegnare il pulsante "Start" che avrà la funzione di far comparire l'intera finestra.
- **Sistema di animazione della PlayList:** purtroppo, a causa delle limitazioni nel tipo di librerie utilizzabili, anche per rendere a video l'effetto "finestra a comparsa/scomparsa" si è stati costretti ad utilizzare un espediente grafico poco elegante, ma funzionante. In sostanza si utilizza un timer che viene fatto partire quando si vuole fare l'effetto di transizione dall'alto verso il basso (a scomparsa) o dal basso verso l'alto (a comparsa) e poi, per ottenere l'effetto, si è fatto un override del metodo *paint()* e si utilizza il metodo *translate* sulla grafica della playlist in modo tale da ridisegnare in continuazione, ad intervalli brevissimi (non percettibili dall'occhio umano), la playlist in posizioni leggermente diverse fino ad arrivare alla posizione voluta. Per questo motivo si fa uso di una variabile chiamata *translateY* che definisce la velocità di traslazione e permette di capire quando si è arrivati nel punto voluto e si può dunque stoppare il timer e, di conseguenza, anche la traslazione. Attraverso l'uso di alcuni cicli "*if…else*" e di alcune variabili booleane si riesce ad ottenere sia l'effetto a scomparsa che l'effetto a comparsa con lo stesso metodo e cioè il metodo *actionPerformed(…).* Durante l'utilizzo di questo metodo, in fase di progettazione, ci si è scontrati con un problema e cioè che, durante la traslazione, venivano mostrati alcuni bordi neri, anche se l'immagine era perfettamente contornata o comunque con sfondo trasparente. Tale problema era causato dal ridimensionamento degli oggetti necessario per rimanere all'interno delle diverse risoluzioni video dello schermo. Per ovviare a questo inconveniente si è scelto un sistema rudimentale e cioè di creare dei files grafici png con un'ampia area in trasparenza in modo tale che i bordi dell'immagine, a qualsiasi risoluzione e con qualsiasi ridimensionamento, risultino comunque fuori dallo schermo e non visibili durante il gioco (ci si basa sul fatto che l'animazione di una parte

trasparente non comporta nulla di visibile a schermo).
Si noti infine che i pulsanti presenti nella finestra della PlayList non si muovono assieme alla finestra e per questo è stato necessario renderli invisibili durante l'animazione e renderli nuovamente visibili (solo quelli necessari) ad animazione conclusa.

Schema meccanismo di animazione PlayList

- **Sistema pulsanti ordinamento categorie PlayList:** nella schermata della PlayList sono presenti tre pulsanti particolari, corrispondenti ai titoli delle colonne della PlayList e cioè Track, Genre e Date. Tali pulsanti hanno la funzione, se premuti, di ordinare le tracce in ordinamento crescente/decrescente a seconda del tasto premuto. Ad esempio, se si preme una volta sul tasto Track si ordineranno le tracce in ordine alfabetico rispetto alla colonna Track, cliccando una seconda volta si ordineranno in ordine alfabetico inverso (dalla Z alla A) sempre rispetto alla colonna Track. Per questo motivo vi è una gestione della grafica dei pulsanti che, negli stessi cicli *if…else* in cui si valuta come ordinare le tracce dal punto di vista tecnico,

viene fatto anche il necessario cambio di grafica, inserendo ad esempio una freccetta in su a fianco al nome Track quando si ordina in ordine alfabetico e una freccetta in giù quando si ordina in ordine inverso, proprio come avviene nelle cartelle di un sistema operativo.

- **Sistema di disegno della lista tracce nella PlayList:** quando nella PlayList si vogliono mostrare le diverse tracce disponibili, le informazioni di tali tracce vengono inserite in pulsanti singoli (per permettere che una traccia sia cliccabile e quindi eseguibile dal gioco) sopra i quali vengono apposte le scritte necessarie. Per questo motivo si è reso necessario l'uso di files grafici completamente trasparenti per mostrare i titoli delle tracce senza ingombrare il pannello PlayList di ulteriore grafica, bensì utilizzando quella già presente sullo sfondo. In sostanza si è scelto un approccio che nasconde all'utente finale l'implementazione effettiva e che permette, attraverso l'utilizzo di pulsanti con sfondo trasparente, di mostrare soltanto il testo del pulsante, mantenendo tuttavia il testo cliccabile per l'esecuzione.

- **Sistema di pulsanti per l'avvio o salvataggio delle canzoni nella PlayList:** il sistema di pulsanti di pausa, avvio, stop o salvataggio della canzone non presenta particolari complessità se non il fatto che viene utilizzata la possibilità di Java di cambiare l'immagine del pulsante a seconda che questo sia premuto o meno. Per questo motivo un pulsante premuto risulterà leggermente più scuro rispetto al pulsante non premuto.

- **Sistema di controllo del volume:** attraverso l'utilizzo di due pulsanti è possibile controllare il volume delle canzoni e viene mostrato il livello del volume come grafica di un altro pulsante. Per il disegno di tale pulsante e l'aggiornamento delle tacche del volume si è utilizzato il metodo preposto della classe *GraphicsEffects*.

2.2. *Player Menu*

Si possono distinguere due classi più una terza per la particolare finestra che deve mostrare il macrostile successivo a quello attualmente in uso:

2.2.1. Player_Menu.java

Classe di creazione dello sfondo e di posizionamento degli elementi. Analogamente alla classe *Start_Menu.java*, anche questa classe estende il JFrame e posiziona gli oggetti nella schermata grazie all'utilizzo del JLayeredPane.

Dunque, con le stesse convenzioni adottate nello Start_Menu nel costruttore:

- viene dato un nome al Frame (indispensabile per lo switch delle schermate)
- viene creato un lettore video per la JMF
- viene ridimensionato il video in modo tale da prendere tutto lo schermo
- viene data la possibilità di chiudere il gioco premendo il tasto ESC della tastiera
- viene ottenuta la modalità Full Screen per l'intero frame
- si ricava il JLayeredPane standard dell'JFrame e vengono aggiunti tutti gli elementi necessari per la schermata Start Menu al JLayeredPane attraverso il metodo *add*
- Viene infine fatto partire il video (*p.start()*) dopo essersi assicurato che venga riprodotto indefinitamente, in loop, come video di background della schermata.

Anche questa volta il costruttore prende come argomento il percorso completo del video che si vuole avviare come video di background.

Per quanto riguarda la grafica si può notare la presenza di altre due metodi, il metodo *VideoStart()* che permette di riavviare il video in caso fosse stato precedentemente messo in pausa per qualche motivo (ad esempio a causa dello switch della schermata) e il metodo *Show_Next_Style* che, come verrà spiegato in seguito, serve a far apparire, col giusto timing, la grafica preposta a mostrare il macrostile che verrà utilizzato nella canzone successiva.

2.2.2. *Player_Menu_Elements*

Classe di creazione di tutti gli oggetti della schermata di gioco che contiene anche un menu con alcune funzioni peculiari della schermata. Per questo motivo tale classe presenta anche alcuni metodi di supporto che servono a creare un'animazione a comparsa/scomparsa che permette di mostrare o nascondere tale riquadro premendo il tasto "Start" dell'interfaccia.

Nell'esaminare gli elementi della classe si andranno a considerare soltanto quelli più importanti per comprendere alcuni meccanismi particolari. Si tralasceranno inoltre tutte le considerazioni sulle funzioni degli oggetti, concentrandosi invece solo sugli aspetti grafici degli stessi:

- **Costruttore della classe:** qui viene inizializzato l'oggetto per le funzioni della barra menu assegnandogli i files grafici corrispondenti alle diverse situazioni possibili (menu in basso inattivo, menu visibile, files di transizione).

  Ancora una volta vengono fatte i dovuti ridimensionamenti e viene inserita la barra menu nella posizione standard appena si accede alla schermata e cioè si rende inizialmente pienamente visibile.
- **Sistema di animazione della barra menu:** il sistema di gestione dell'animazione della barra menu è completamente analogo a quanto già visto

per la PlayList dello *Start_Menu*, senza alcuna variazione e mantenendo valida l'approssimazione grafica necessaria ad evitare di vedere spiacevoli bordi neri durante la transizione. Come per la PlayList, anche in questo caso i pulsanti non si muovono assieme alla finestra e per questo è stato necessario renderli invisibili durante l'animazione e renderli nuovamente visibili (solo quelli necessari) ad animazione conclusa.

- **Pulsante SwitchView:** la funzione del pulsante SwitchView non è implementato nel prototipo, tuttavia il progetto prevedeva che questo pulsante potesse variare il tema che fa da sfondo all'intera schermata. In sostanza, premendo il pulsante "Switch View" doveva esserci la possibilità di passare da una visualizzatore musicale (come ad esempio quello presente su Windows Media Player) ad un visualizzatore delle note musicali suonate in quel momento o ad un visualizzatore di altri effetti di diverso tipo o che potesse mostrare gli strumenti musicali in uso in quel momento. Purtroppo non c'è stato il tempo materiale per implementare tale feature, cosa che peraltro risulta molto ostica senza utilizzare librerie esterne per la creazione di particolari effetti grafici.

- **Sistema di controllo del volume:** come nello Start Menu, anche qui attraverso l'utilizzo di due pulsanti è possibile controllare il volume della canzone che sta suonando in quel momento e viene mostrato il livello del volume come grafica di un altro pulsante. Per il disegno di tale pulsante e l'aggiornamento delle tacche del volume si è utilizzato il metodo preposto della classe *GraphicsEffects*.

- **Sistema di visualizzazione strumenti e stato d'animo:** ad ogni variazione stilistica del brano in esecuzione, che corrisponde a variazioni medie di attenzione e meditazione del giocatore, viene fatto corrispondere, come visto, un diverso stato d'animo. Tale stato d'animo, assieme agli strumenti scelti col macrostile iniziale, deve essere mostrato a video in questa schermata. Per questa funzione si è deciso di utilizzare una semplice scritta a video, anch'essa col un Font ridimensionato in funzione della risoluzione video.

- **Sistema di animazione menu a comparsa NextStyle:** in questa schermata di gioco è previsto l'utilizzo di una particolare finestra che deve essere mostrata in procinto della fine della canzone attuale in modo tale da informare il giocatore sul prossimo macrostile che verrà utilizzato per la canzone successiva, in base alle ultime indicazioni rilevate dal sistema di lettura delle onde cerebrali. Per gestire l'effetto ad entrata è stata utilizzata una classe particolare di nome *Player_Menu_Style_Element*. Tale classe gestisce l'animazione di scomparsa/comparsa nello stesso identico modo in cui vengono gestite le animazioni della PlayList e della barra menu con la differenza che si tratta di una traslazione orizzontale invece che verticale e che il tempo in cui la finestra rimane visibile non viene controllato dall'utente tramite un pulsante (pulsante "Start" nella PlayList), ma viene deciso a priori nel codice secondo il tempo che viene considerato sufficiente per permettere al giocatore di conoscere il nuovo macrostile.
  Rimangono valide invece le solite convenzioni utilizzate anche per le altre animazioni, compresa quella di creare files grafici nettamente più grandi e trasparenti per evitare i fastidiosi bordi neri.

Schema meccanismo di animazione NextStyle

5.4. Bug conosciuti

Si fornisce qui una lista dei bug grafici conosciuti:

- I due oggetti che prevedono l'animazione (PlayList e barra menu nel Player Menu) potrebbero avere un'animazione lenta con effetto tearing in computer dalle basse prestazioni.
- I video di background potrebbero avere un framerate basso in alcune situazioni.
- A causa del ridimensionamento dei pulsanti dovuto alle diverse risoluzioni, ci potrebbero essere dei bordini visibili per alcuni pulsanti delle schermate in alcune risoluzioni particolari (non molto frequenti fortunatamente).
- La pressione del tasto ESC potrebbe non chiudere immediatamente il gioco.

- Il video di background della schermata Player_Menu è un placeholder, quel video in realtà dovrebbe essere il trailer (o intro) da mostrare prima di arrivare alla schermata principale del gioco.
- Se si preme il tasto “Home” nella schermata Player Menu finchè è in corso la prima generazione della canzone (quella in cui non si sente ancora alcuna canzone), si attiverà una NullPointerException a causa della mancanza dell’istanza di MainMusic_Player che non è ancora stata creata.
- Nonostante vi siano dei metodi di ridimensionamento delle scritte, con alcune risoluzioni, i fonts usati per le scritte potrebbero essere troppo grandi e non entrare per intero nello schermo.

6. Conclusioni

Il lavoro svolto presso l'azienda si è articolato in diverse fasi:

- Analisi e raccolta dei requisiti e delle richieste dell'azienda per lo sviluppo del progetto, sia dal punto di vista del core del gioco che dal punto di vista grafico.
- Raccolta e comprensione del materiale già sviluppato precedentemente per la definizione dei macrostili e stili e la lettura da log.
- Comprensione dei meccanismi utilizzati da Impro-Visor e come poter integrare le sue funzionalità per gli scopi previsti dal progetto.
- Inizio dei lavori di ristrutturazione di Impro-Visor e del vecchio codice di Brain Music per adattarlo alle proprie esigenze.
- Risoluzione dei problemi di connessione con il dispositivo a causa di alcune incompatibilità di pacchetto con il software ThinkGear.java. In questa fase ha giocato un ruolo fondamentale il supporto tecnico da parte di NeuroSky che ha appresa le esigenze di BrainMusic e ha adattato il codice java già scritto in modo da scendere ad un compromesso accettabile che permettesse una facile comunicazione fra il programma e il dispositivo MindSet.
- Implementazione di tutti i metodi necessari allo svolgimento delle operazioni principali di definizione dei parametri da dare ad Impro-Visor, a seconda dell'EEG semplificato, affinchè possa improvvisare una canzone coerente con gli stili ricevuti.
- Implementazione dei metodi di gestione dei files .midi e di salvataggio degli stessi
- Testing del core di BrainMusic e dei metodi di gestione musicale
- Coordinamento con il grafico dell'azienda e ricerca di tutte le possibilità offerte da Java per avviare video in background e permettere le animazioni volute.
- Implementazione dei metodi grafici con continui raffinamenti e nuove limitazioni o compromessi (ad esempio quello di creare files grafici più grandi e trasparenti per evitare fastidiosi bordi neri durante un'animazione) richiesti al grafico.
- Testing finale del prototipo e redazione di documenti consuntivi che possano permettere, a chi prende in mano lo sviluppo del progetto per portarlo da un semplice prototipo a qualcosa di più concreto e definitivo, di capire il lavoro svolto e conoscere i motivi di alcuni compromessi o scelte fatte durante la programmazione.

In particolare, durante lo svolgimento dell'attività si sono incontrate le seguenti difficoltà:

- Conoscenza scarsa da parte del sottoscritto di tutti gli aspetti e le definizioni musicali utilizzate nel progetto.

- Difficoltà nel comprendere alcune delle meccaniche precedentemente scritte per BrainMusic.
- Problemi nell'integrazione Java della libreria ThinkGear per la comunicazione con il MindSet.
- Molte limitazioni dovute alla necessità di mantenere il gioco cross-platform e quindi compatibile con ogni tipo di hardware e soprattutto sistema operativo; da questo punto di vista l'impossibilità di usare alcune librerie di aiuto o alcuni framework particolari ha giocato un ruolo fondamentale nell'approssimazione del prototipo ad un livello poco soddisfacente sotto alcuni punti di vista.
- Nessun dipendente dell'azienda ha potuto fare una revisione del codice in quanto nessuno aveva le necessarie competenze informatiche.
- Mancanza di ulteriore tempo per poter continuare il progetto o per imparare nuovi script (es. JavaFX).
- Nessuna esperienza di programmazione Java professionale.

Per questi e altri motivi la progettazione del prototipo non ha seguito gli standard che erano stati prefissati all'inizio e non si è riusciti a completare il progetto che contiene ancora le seguenti mancanze o problematiche:

- Non esiste una schermata di Trouble-shooting né la possibilità di cambiare la porta COM assegnata al MindSet o il baudrate relativo.
- Come già spiegato, a causa di alcune limitazioni di Impro-Visor, il processo di generazione della canzone non avviene in real-time, ma soltanto con un ritardo di una canzone (mentre sta suonando la terza canzone, il gioco sta generando, leggendo le EEG semplificate del giocatore, la quarta canzone); questo significa che l'obiettivo iniziale del gioco stesso è stato in parte disatteso.
- La qualità grafica generale del progetto è molto bassa a causa delle limitazioni di Java e della JMF.
- Manca tutta la gestione delle animazioni di sfondo della schermata Player Menu.
- Manca la possibilità di settare, da qualche file di configurazione, la durata standard della canzone che, attualmente, è definito da una costante all'interno del codice.
- Vi sono alcuni problemi di ottimizzazione e di ridimensionamento causati dalle limitazioni di Java.
- L'output generato non è di qualità sufficiente per diventare commerciale essendo soltanto un .midi suonato da strumenti sintetizzati elettronicamente.
- La gestione delle canzoni si ferma al formato .midi, quando il progetto finale dovrà invece prevedere la gestione dei normali formati .wav o .mp3.

- Non esiste gestione della possibilità che l'utente si tolga le cuffie durante l'esecuzione di una canzone o, comunque, finchè sta giocando al gioco.

Fatte dunque queste premesse e tenendo conto che il progetto principale di 3i è sviluppato da un'altra azienda che scrive codice in linguaggio C, si presume che l'attuale prototipo darà soltanto il necessario bagaglio di informazioni e qualche dettagli di implementazione agli sviluppatori professionisti che poi tradurranno tutto il codice Java in codice C, eliminando, di fatto, con questo passaggio, anche la maggior parte delle limitazioni e delle difficoltà incontrate durante la programmazione in Java.

Rimane comunque un'esperienza assolutamente interessante in un campo innovativo e che promette di avere numerosi sviluppi in futuro, specialmente con l'avvento di tutte le nuove tecnologie di gaming che prevedono l'abbandono dei joypad tradizionali a favore di altri meccanismi più intuitivi e accessibili.

7. Bibliografia

Sito web Impro-Visor

http://www.cs.hmc.edu/~keller/jazz/improvisor/

Spiegazione generale tecnologia NeuroSky®

http://www.neurosky.com/AboutUs/BrainwaveTechnology.aspx

Pagina prodotto MindSet NeuroSky®

http://www.neurosky.com/Products/MindSet.aspx

Articolo Hi-Tech Italy sul progetto 3i

http://www.hi-techitaly.com/news/gaming/652-pub-company-inventa-i-giochi-3i-per-nintendo-wii-con-wii-qube-e-wii-relax.html

Pagina JMF

http://www.oracle.com/technetwork/java/javase/setup-138642.html