Electrical and Computer Engineering
University of Toronto

Creative Applications for Mobile Devices

“Music Tapper”

Submitted by:

Ryne Yang – 998692954

Mani Golafra – 995111882

Word Count: 1947 words

Introduction

Mobile application industry is a very fast growing market and it is currently where general web development was back in the mid 90’s. Applications are being created daily and added to available platforms on a daily basis. This challenging market had made more and more people to become interested in having a share in this fast growing industry .Developers try to create creative, entertaining and useful applications to gain the most attention for the smartphone users.

We as developers decided to build an Android game called “Music Tapper” that we anticipate many users will be interested in. Music Tapper is a fast pace game that makes the players focus on the patterns of the playing song and based on those patterns it generates buttons on the screen for the user to tap on. This game is in some ways similar to the famous “Guitar Hero” game that is available on Android Market and iOS App Store. Guitar hero game analyses’ the song patterns locally on their own servers and then players can purchase these songs and the corresponding configuration files in order to play the song. In our implementations we chose to create a similar game that can analyze the song pattern locally on the phone and in real time. Our game can practically analyze any song stored on the phone and dynamically create buttons corresponding to the song pattern and then the player should tap on the screen at a particular moment when the button is passing the lower bar indicated on the screen.

Overall design

The main game engine consists of many concurrent parts that work together. Figure 1 illustrates the game engine’s block diagram. As it has been shown, the first step is receiving the MP3 data and decoding it to the MS wave format so that it can be manipulated in the next stages of the process. Furthermore we use Fast Fourier Transform function to extract the spectrum of the music based on the criteria that was used in the code. From there we analyze the spectrum based on the energy of the song at each sample. We sample the song in a 50 millisecond intervals and we extract the energy of each section as a guideline for generating buttons on the screen.

The core technical difficulty of this design is on the multi-threads concurrency and the spectrum analyzer. More than 5 threads are working simultaneously on the background as mentioned. The whole architecture of this application is based on a data stream pipe so that every different module can fetch the data from this pipe and then put back the processed data back to this pipe like the concept of the assembly line. By using this architecture, the application will be able to process the data at the same time the player is playing the game. However, there are some issues brought by this architecture as well. Concurrency is the first one, as well as the hardware resource allocation. Because this application is supposed to be running on the portable devices such as mobile phone or the tablet, these devices usually don't have a very powerful processor. This leads to the lack of processing ability which is actually in need of this application. In order to analyze the music data, MP3 decoding is inevitable, as well as FFT transforming. These two jobs are well-known for their heavy workload.

In order to solve this problem and eventually make it possible to run on the Android devices, we manually lower down the MP3 decoding quality. In other words, the output of the MS wave file will lose some information compared to the original MP3 file. We lower down the sampling frequency of the MP3 decoding and also the FFT transforming. And also, in order to use the FFT transforming, we have to truncation the data to fit the length of 2^n(FFT can only process the data length of 2^n). So that means, we have to truncate the original data which is 8820 bytes(50 milliseconds) to 8192 bytes. The rest 628 bytes can be ignored without affecting the result of the analysis in the later process. In order to reduce the workload in FFT, we blended the left channel and the right channel, so instead of processing twice on both channel, we only have to run FFT once to get the spectrum.

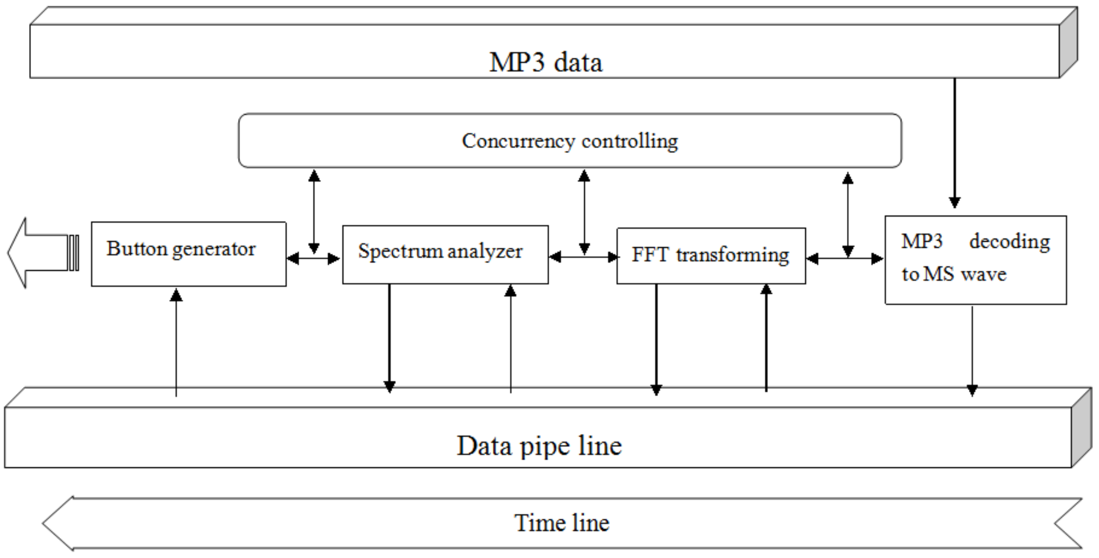

Figure 1: Game Engine block diagram

The concurrency between different threads is being controlled by a concurrency controller module so that we would make sure that the rate of the button generation directly corresponds to the songs flow and also make sure those threads execute the data flow in the right order. This is a very critical requirement for this application; either of them fails will directly result in crash.

In the module of the analyzer, we adopt two different ways to implement the spectrum analysis. In terms of extracting the beats of the music, in other words, the drums beats, we calculate the spectrum energy by using equation: $E(k) = Re^2(k) + Im^2(k)$. Notice that the drum beat usually stays within the frequency between 0Hz ~ 1200Hz and appears periodically, so we only calculate the energy between 0Hz ~ 1200Hz. And in order to sense the different beats from different styles of music, we did not use absolute threshold for the judgment. Instead, we use dynamic threshold which will automatically adjust itself to the best value when music proceeds. We also have a noise threshold which we use to filter out the noise so that those noises will not interfere the judgment especially when the music goes quiet.

Speaking of the solo detecting, we trace the solo by analyzing the peaks of the spectrum. In the spectrum, solos of the instruments usually appear to be the peaks in different frequency. And one single solo sound will eventually happens to be multiple peaks in the spectrum because all the acoustic instruments will make the sound consists of harmonic frequency. Noticing this feature, we designed a very simple solo tracing function which will make the judgment of whether there is a solo in a fragment of sound (50 milliseconds) within a very short time.

As for the user interface we created a fluid and dynamic game frontend that can adjust to the songs pattern of button generation. We used button objects that are created based on the game engines request at the time. The button object has its own class with its own unique specifications that can be easily modified in the code. Also there is another class named "speed" that takes care of the movement of the buttons which is also fully configurable. There is a "Maingamepanel" class that calls every other UI related class once a button generation request has been issued by the game engine. This class has a separate thread that keeps updating all of the button objects and judges whether a button has been touched or not.

Statement of functionality

Music Tapper game worked as required by this course and beyond. At the time of development a challenging part for us was to make the game engine and the analysis of the songs spectrum information. We successfully completed this task and then we were able to adopt the rest of the applications components with the game engine. The user interface also was a part that required extra attention since the game is fully dynamic and it requires adjusting to the rate of the button creation dynamically in real time.

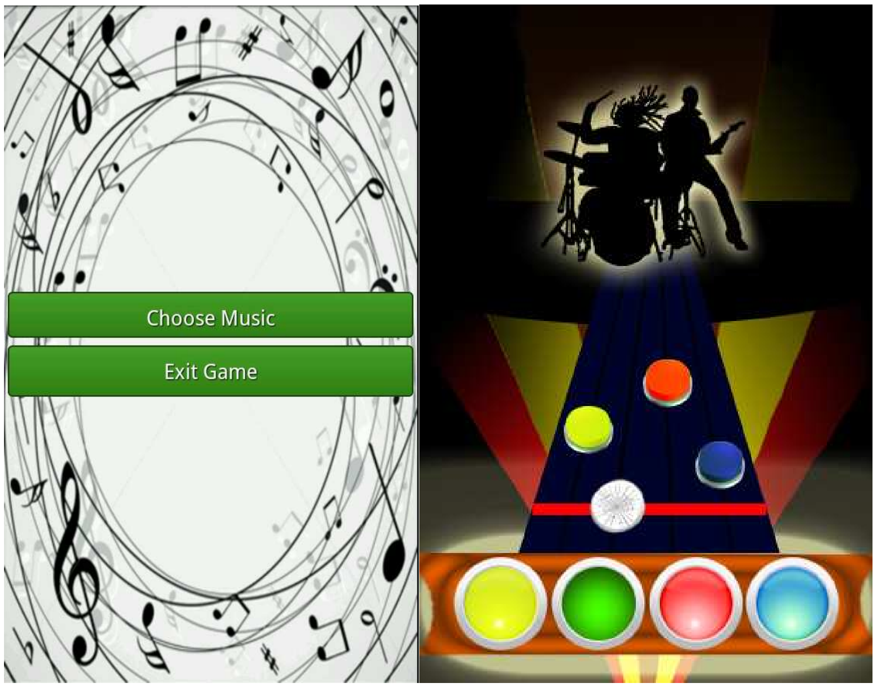

Figure 2: User Interface of Music Tapper

Figure 2 shows the user interface of the game. The left image is the first Activity's screen. From there when the user clicks on the "Choose Music" they get to choose any song that they have stored on the device. Upon selecting a song then image on the right will appear, which is the main user interface of the game. On this screen the buttons will fall from the top of the screen and the touch input screen is ready. Once the buttons reach the red line then the user is expected to press the corresponding button on the bottom of the screen. Upon an accurate press the user will gain score and the shape of the button will change and later on the link-list object will be removed from the queue. All the above worked as described and all the functions are working together concurrently and smoothly.

What did you learn?

Android platform was something new for both us and therefore the first important thing that we learned was the Android SDK and the methods of using it. Along learning we gained a huge amount of knowledge regarding the Android's View class and that how to create a dynamic surface view that can handle multi-touch input and at the same time produces fast moving objects.

Also, we learned a lot regarding analyzing the spectrum of the song. We learned how the energy of the song is being distributed in its spectrum and that how we can extract that information in a way that we can create patterns from it.

Another important learning take away for us was to manage the concurrency between all of our threads. We use seven threads concurrently and we had to make sure that all the threads are synchronized together. Specially button generation section that has to be very concurrent with the beats at a given time taught us how to use the system clock in a fast pace game like ours.

Contribution of group members:

We divided the work into two main parts; first the User Interface and second the Game Engine. Mani took care of the UI design and the fast pace touch inputs that had to be synchronized with the game engine and Ryne took care of extracting the song spectrum and analyzing it for the feed to the UI. We both contributed to the flow of the game and deciding on the best patterns for different styles of the music. We Listened and analyzed many different songs from different genres to be able to come up with a generic analysis method for virtually every type of music.

Future Work:

We think that one of the main selling points of any game is its interface and the quality of its design. Now that we have a strong game engine which is the backbone of our game, we would like to improve the graphics of it so that it could be more appealing to the players. Also, at the moment our game engine is fully customizable in regards to game difficulty level or game speed and etc. However, these customizations are only accessible through our java code and the user won’t be able to dynamically change the settings in the game environment. We would like to add such features to the interface so that the players can easily switch between game modes.

Another possible improvement to our game is to create some sort of incentive for the players to play it. We were thinking of implementing a central server that can keep track of all the players high score ratings. In this way competitive players can compare their results with one another and this can cause a more challenging environment for the players.

Having a “Business School class for marketing/entrepreneurship take it up” is such a great idea. We think that in this competitive current mobile application industry having some extra knowledge regarding the marketing side of the story would be a huge additional tool towards having a successful mobile application.