Carleton University

Honours Project

Music as a Game Obstacle

By Sukhveer Matharu

Supervised by
Dr. Michel Barbeau
School of Computer Science

Submitted on
Date: April 21, 2008

Abstract:

Over the years in the video game industry, development of a game can either be innovative, iterative, or both. By innovative, it can be unique that a user has not experience to a point where it can become a new genre. By iteration, it takes the concept that's already been made and change the presentation, story, and/or design. By both, it takes the concept that's already been made and improve upon it. The development for this game is both, takes the old game concept and improve it by having it depend on music for game interaction. It's divided into two phases. The first phase describes the research and the implementation of the beat-detector. The second phase explains the game design and the implementation which uses the beat-detector.

Acknowledgments

Dr. Michel Barbeau for supervising the project

All the beta testers that made this project possible

Table of Contents

List of Figures

Introduction:

Video games and Music shares a common purpose: to provide entertainment. Music in the video game industry played a role as an audio representation for video game presentation. As the gaming industry evolves, music has slowly been becoming important in certain video game genre such as music simulation games which is the only genre that depends on music for interaction. With that purpose alone there wasn't any other use for music within a video game. Aside from music interaction, the AI are either dynamic from user input or static from the programmer's implementation. Dynamic AI from user input meaning that their behavior is determine by the environment and the user's input. AI that is statically implemented would result of an AI to behave the same regardless of the environment nor the user's input. Up until now, the majority of AI behavior in games is either dynamically or statically implemented and didn't consider other means of implementing their behavior, such as music.

Purpose of this project:

It is possible for music to play a greater role in video game interaction which leads to the reason of this project. The purpose of this project is to show the use of music in game features, interaction, and AI behavior alteration. This project is a simple 2D space-shooter game that demonstrates the use of music in every aspect of this game including enemy AI behavior.

Phase 1: Beat Detector development.

Research.

The first phase of the project involves research on the foundation of music beats and how it can be detected in the digital world followed by the implementation of the idea on how to achieve this goal.

In order to detect a beat, we have to figure out what causes a beat and how we can tell whether a beat is being created or not from noise called music. The definition of the term “beat” is when an energy of a sound is greater than it's energy history. In other words, if the newest sound energy is greater than its old sound energy, we would get a “beat”. The human ear intercepts an energy sound by which the brain interprets the difference of this energy and the previous energy heard. The idea is to have a program computes the sound energy value and compare it to its history of sound energy values. The way a program can intercept a sound energy is through the audio spectrum frequency.

The audio frequency is the sound energy that outputs up to 1024 frequency samples. Frequency samples are basically a representation of a sound in terms of vibration. Each frequency sample is measured in hertz (Hz).

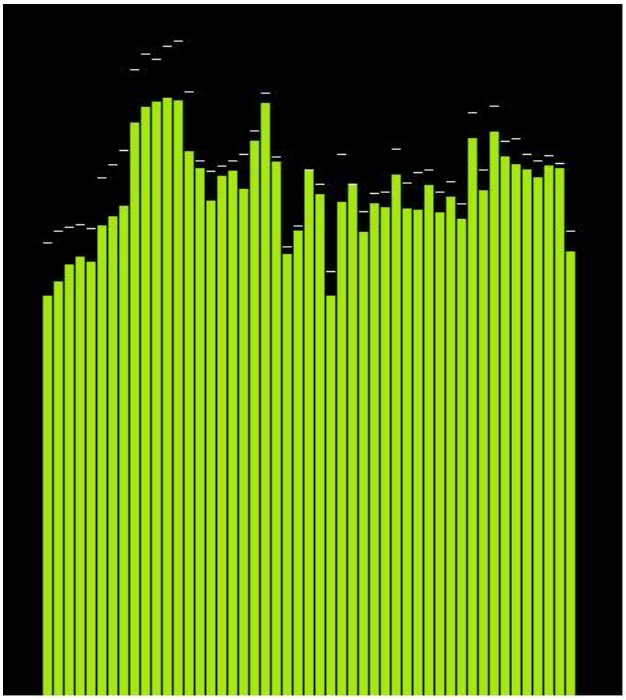

Figure 1. Frequency spectrum analysis bar

Once the audio-spectrum analysis is collected the next step is to transform this data into 1024 frequency-band spectrum which is done using Fast Fourier Transform(FFT) library. FFT is a fast library algorithm that takes in an arbitrary input and computes the Discrete Fourier Transform in one or more dimensions. From there we do the computation to get the results needed and apply the values to the following formula:

$$Es[i] = \frac{64}{1024} X \sum_{k=ix64}^{(i+1)x64} B[k]$$

where "Es" contains the sub-band; "i" and "B" is the spectrum computed by FFT. What this formula does is divide the frequency spectrum into 64 sub-bands and store the each sub-band into the "Es" variable.

Next, we add the computed value “Es” to the end of the history list up to 20 previously calculated Es sub bands. If the list is full with 20 “Es” values then we delete the oldest calculated Es and add a new one to the end of the list.

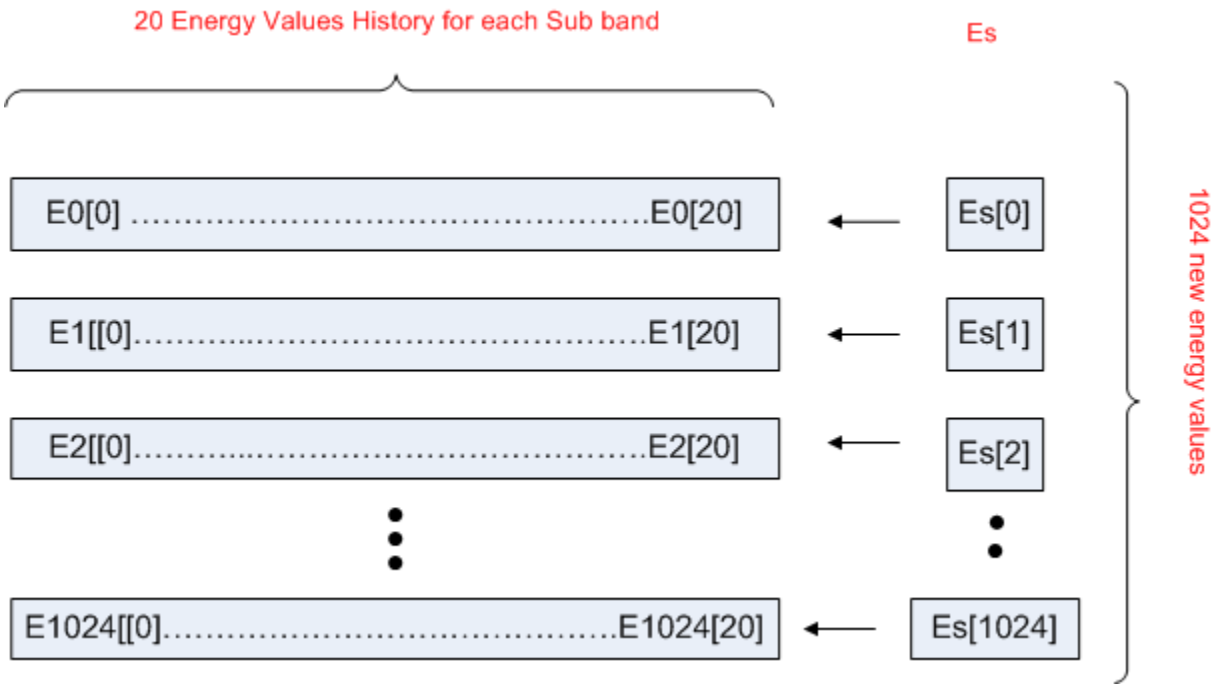

Figure 2: Sound Energy History

Next we compute the average energy from the history list using this formula:

$$< Ei >= \frac{1}{20} X \sum_{k=0}^{20} Ei[k]$$

Pick a threshold constant C where C>0, if Es[i] > (C* <Ei>) then we have a beat detected. To get a specific beat whether it's from high, mid, or low sound, we calculate the variance and apply each of the three threshold values against the variance.

Implementation of the beat-detector:

The language chosen to implement the beat-detector is C# and XNA framework for its simplicity over the other languages. For the implementation of the beat detector, there were two audio API to choose from. One is the build-in audio API provided by the XNA framework, the other is FMOD. After comparing their features, FMOD was chosen for this project.

The audio API used for the beat-detection algorithm is called FMOD. FMOD is a commercial API developed by Firelight Technologies that plays audio in various formats. It also contains features that are needed for developing the beat-detection algorithm. The reason this API is chosen is because it can compute without consuming a great amount of CPU resources and contains the necessary functions for this project.

For implementing the FFT calculations, a library was used and was able to integrate its functions into the program.

The implementation of the beat-detection involves creating 3 classes:

- Sound
- FMOD system
- AudioProcessor

The following explains the role of each class.

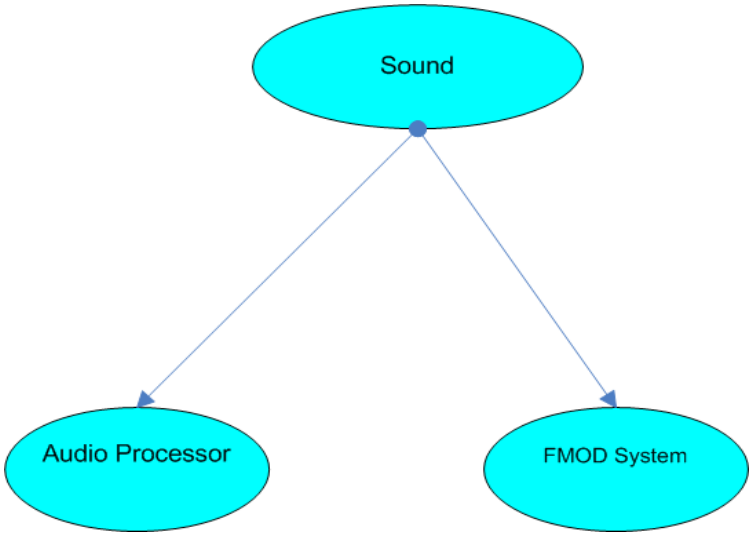

Figure 3: beat-detector hierarchy diagram

Sound Class:

This singleton class takes in the name of the audio file as a parameter and calls the AudioProcessor and FMODSystem classes to process the audio file. The reason for making this class a singleton is because in the program the only audio file being played is the music in order to not have duplicate beat-detectors and it can be accessed by any other class. This class handles basic functions such as playback, returning the position of the music time, and returning whether a beat is detected or not.

Audio Processor Class:

This is the most important out of all three classes. It takes in a FMODSystem parameter and uses it to fetch the sound frequency on every tick. It then follows the beat-detector procedure described in the research on analyzing the raw frequency data and outputs a signal of whether a bass, mid, or high beat is detected.

FMOD System Class:

This class takes in the audio file name and uses the FMOD API to process the audio file for playback. It contains the variables created that the AudioProcessor requires to process the audio data.

Testing:

A special GUI was developed solely for testing out the beat detector. While the music is being played, whenever a beat gets detected, the method from Sound Class sends a signal to the GUI and indicates the user that a beat has been detected. With this GUI many types of music were being used as test cases to debug the beat detector.

Difficulties Encountered:

One of the first difficulties encountered when implementing the beat detector was the FFT library. Originally the FFT library was designed for C, C++, and visual basic but not for C#. Fortunately after searching around the internet, a C# wrapper of the FFT library was found that contains the same function calls as the C++ version. The next problem involving FFT was implementing its functions which involves memory allocation. In the FFT processing function, it outputs the result in a memory that is required to be allocated by twice the size of the frequency spectrum. In C++ it wouldn't be a problem, but because C# is a memory-managed framework just like Java, allocating memory would have proven to be difficult. After doing research, a way was found to create pointers with memory allocated in C#.

```
    IntPtr pin, pout;
pin = fftwf.malloc((int)a_FModSystem.get_spectrum_size * 8);
pout = fftwf.malloc((int)a_FModSystem.get_spectrum_size * 8);
          //GETTING VALUES FROM FFTW
for (uint i = 0; i < a_FModSystem.get_spectrum_size; i++)
{
    fftw_in[i] = a_FModSystem.getSpectrum[i];
}
  Marshal.Copy(fftw_in, 0, pin, (int)a_FModSystem.get_spectrum_size * 2);
fftw.execute(fftw_p);
Marshal.Copy(pout, fftw_out, 0, (int)a_FModSystem.get_spectrum_size * 2);
```

Looking at the code snippet on how this problem was solved, the C# wrapper version of the FFT library contain their own malloc function. In the C# language there is a special pointer called "IntPr" that allows FFT to allocate memory and have it pointed by the IntPtr. Another interesting function from C# is called "Marshal.Copy" which allows us to copy the data from one pointer to the other.

Based on the test feedback using various types of music as test cases, it seems that there was some threshold problems: there are music genres that can have their beats detected fairly well whereas other genres tend to have highly inaccurate detection of its beats. The problem lies in the threshold balance so that the beat-detector can accommodate to all music genre. The only way to tackle this problem by trial and error which is tedious. Test every music and apply various thresholds until the beat detector can detect music while not being biased towards other music. There is one particular music genre it has problems with: rock music. Because of the nature of its bass in rock, the beat detector wasn't able to pick-up any bass beats in rock music unless the threshold for rock. If the threshold is set specifically for rock, the accuracy of detecting other genre would decrease. For the sake of balancing with all genre, consideration of rock genre had to be omitted.

Conclusion of Beat Detector:

After vigorous testing and debugging, the beat detector fairly accurate on most of the music genres with some minor setbacks such as the skipping of soft beats. While this beat detector can serve its purpose well for this project there is room for improvements in the detector. One of the improvements could be analyzing the music and perform more complex computation instead of computing it in real time. Perhaps with more research in the future the beat detector can be more accurate and hope to accommodate all genre including rock music.

Phase 2: Music Blaster development

Game Design:

This part of the project is developing a game that will integrate the beat-detector so that it will demonstrate the use of music in video game interaction and presentation. In the planning part of the design, the important aspect to consider is how well it can demonstrate the use of the beat-detector because if you design the game that can't show the use of music in the interaction then the game will just be another unoriginal concept.

The type of game designed for this project is a Space-Shooter game. It's a simple vertical side-scrolling game where the user controls a ship from left to right. The objective of the game is to shoot down enemies that spawns from the top of the screen and survive until the music ends. With this game design, the beat-detector can be utilized fairly well. For an example, for a space-shooter game, the enemy behavior can be controlled by the music beats. Since the 1970's, space-shooter games has been played by millions of people around the world and to this day there are games being developed based on this genre and therefore, it's the most recognized genre in the world.

When integrating the beat-detector in a space-shooter game, the first thing to consider is what do you want the game to do whenever it detects a music beat. Keeping in mind the overall importance of the game features such as the user interaction, difficulty, and visual presentation. The beat-detector data gives us the following values:

- Bass-beat signal
- Mid-beat signal
- High-beat signal
- Bass-beats per minute
- Mid-beats per minute
- High-beats per minute
- Length of the music.

There are 2 types of enemies: regular and turret. Each type of enemy have different behavior from the beat-detector. The types of enemy's behavior will be assigned as follows:

Regular Enemy:

- Movement pattern changes on every 3rd bass-beat signal.
- It spawns whenever the interval time between two bass-beats falls between 300 and 500 milliseconds.
- It fires a bullet whenever a mid-beat signal is received.
  The speed is determined by the bass-beats rate per minute.
- It has a shield of 10 hit-points.
- The number of units it can spawn is unlimited unless turrets spawn, then it has a limit of 1.

Turret Enemy:

- Movement pattern changes whenever the interval time between two bass-beats falls between 20 and 1500 milliseconds.
- It spans whenever all turrets got destroyed and the interval between two bass-beats is at least 1250 milliseconds.
- It fires a bullet whenever a bass-beat signal is received.
- The movement speed is determined by the bass-beats rate per minute.
- It has a shield of 50 hit-points.
- Whenever there are turrets, the regular enemy spawns at most 1 of its units until the turrets are destroyed.

The player only have 1 life with a shield of 100 hit-points with no way of regenerating it back. If the player gets hit with an enemy's bullet it will deduct the player's hit-points by one. If the

player has no more hit-points left it will explode and the game is over.

As for visual presentation, the game has stars in the background. The way the stars behave is that its scrolling speed is determined by the mid-beats rate per minute, and the colors of the stars changes whenever a bass-beat signal is received. There are 255^3 ways of selecting the color of the stars.

Implementation of the game:

For the implementation part, C# and XNA framework was chosen for it's quick and simplicity.

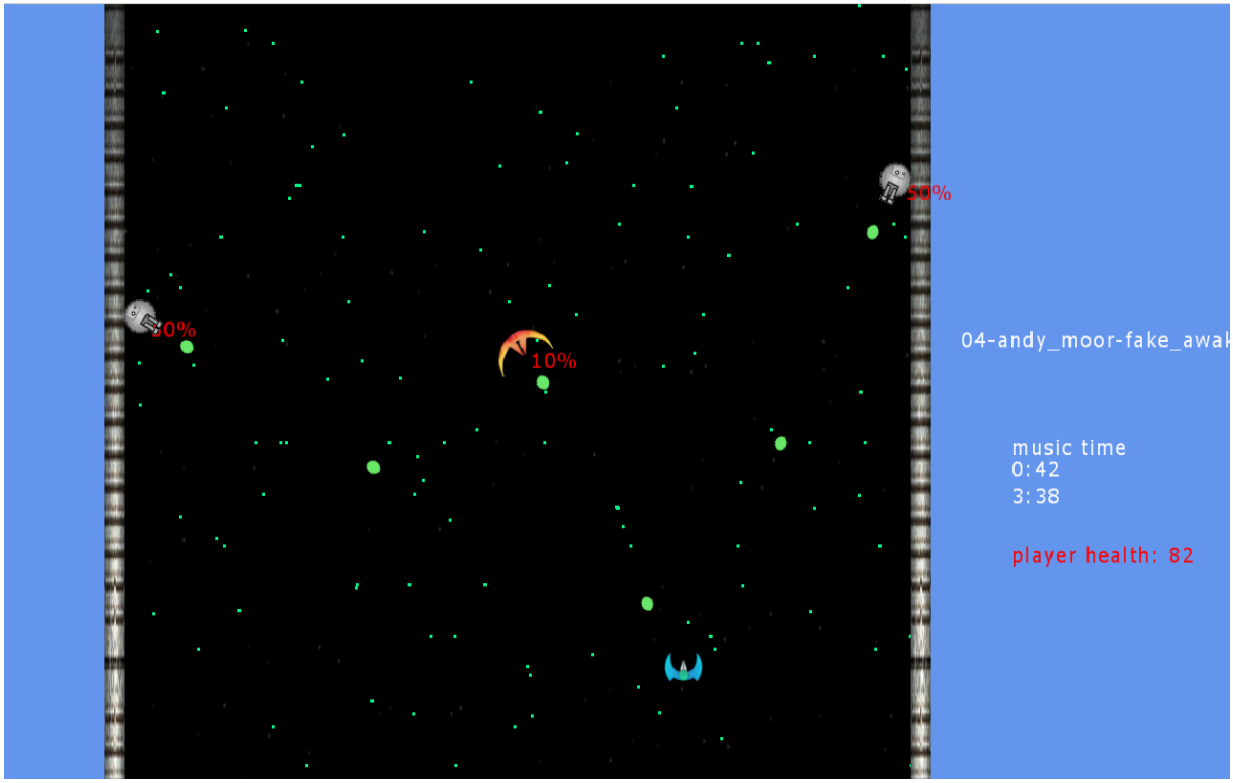

Figure 4: Game screen shot

The game-shell is taken from the phstudios site which contains the screen implementation as well as some of the game sprites. The pause-screen had a small bug which took a while to fix. The class hierarchy of screens is shown as follows:

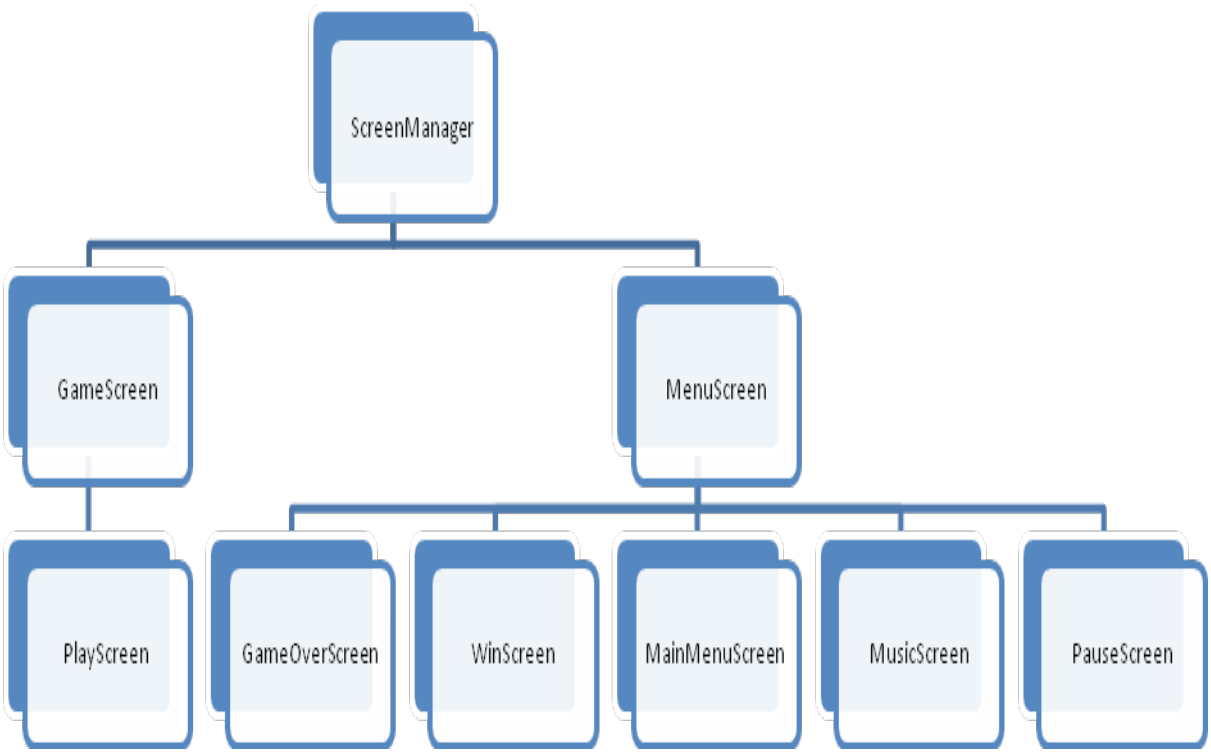

Figure 5: Screen class hierarchy diagram

The ScreenManager has a list that stores the priority of which screen gets to be displayed first. It can add a new screen or remove a screen from the list. The screen on the top of the list overlaps all other screen until it's either removed or have a new screen added to the list.

The following are explains the purpose of each screen:

- MainMenuScreen: Lets the user start the game, go to help screen, or exit the game.
- PauseScreen: Freezes the PlayScreen and lets the user to go to MainMenuScreen, continue with the game, or exits the game.
- Music Screen: Scans the music directory and lists the music files for the user to choose.
- WinScreen: Displays the score, and lets the user either play again or quit.
- GameOverScreen: Lets the user play again or quit.
- PlayScreen: Adds the GamePlayObject and displays the game play until the user either quits, wins, or loses.

EnemyControl class is a singleton class which is the “brain” by giving instructions to the enemies and the game overall features on how it should behave. On every update it reads the beat-detector data signals and gives each enemy and instructions on what do based on the results from

the beat-detector. Whenever an enemy is spawned or destroyed it lets the EnemyControl know who is still in the game.

InputState class is responsible for returning whether a specific key on the keyboard or a controller button is pressed. The controls for the game is simple as it uses the arrow keys to move the player and space button to fire. For the controller, the D-pad moves the player and the A button fires bullets.

Implementing the bullets, player, and enemies is quite straight forward. The following is the class hierarchy diagram of the game objects implemented.

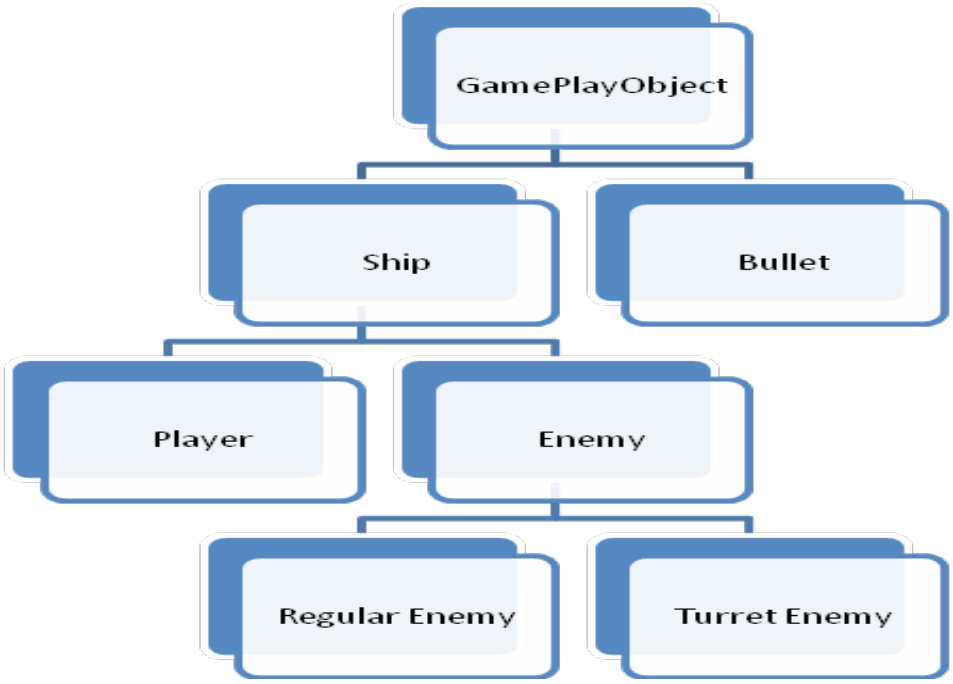

Figure 6 : Game Object class hierarchy diagram

Testing:

After the game been completed, it was given to various peers from different faculties for beta testing so that the game can be polished and bug-free. Each beta-testers were given a game and had the option of adding their music to give it a test. The main feedback that's needed is the

accuracy of the beat-detector, the balance of the game, and their overall impression. Here were some of the positive and negative feedbacks:

> "It's a creative idea. It's entertaining. There is a lot of things you can add to it."
> -Michael Rowe

> "The of idea of firing based on the sound is a novel idea"
> -John

> "It needs tuning to work well with different styles of music"
> -Calvin .

Based on the technical feedback received from the testers, the game had some input were fixed as well as the balancing issues from their submitted music cases.

Difficulties Encountered:

The first difficulty encountered was the frame rate. Usually the frame rate of the game clocks at 60FPS (frames per second) but at random times the frame rate would lower to 20FPS and go back up to 60FPS. This results in a huge lag in the game, music would not sync with the game, and the visual presentation that depends on the music and constant frame rate would be distorted. After carefully checking each line there was a function in the update method that was being called in a high running time which caused a drop in frame rate. It was a simple fix by applying restriction on how often it should be called and there seems to be no frame rate problems.

The 2nd difficulty was multi-language compatibility. When testing the music file that had a Japanese character, the game would crash. The reason it crashed because the SpriteFont in the XNA didn't include the character code for Japanese. An attempt to fix the problem by increasing the number of character code caused system processor couldn't handle the amount of characters because in a language like Japanese it contains at least 1000 characters. For this reason the only way to resolve the problem is to omit characters that's not in the range from 32 – 126.

Conclusion of Phase 2:

After thoroughly revise and debug the game based on the feedback results from the beta testers, the game runs smooth without any problems despite some of restriction that needed to be applied. In the end the development of the 2nd phase proved to be useful as it demonstrated the potential of integrating a beat-detector in a typical video game that was first created in the late 1970's.

Overall Conclusion of the Project

The purpose of the project was to see whether it's possible or not to have a video game which its behavior based on audio beats. In the beginning, whether having a game which its features and AI behavior controlled by music beats would be accepted by gamers was questionable, but the reception turned out surprisingly positive and it showed interest of their favorite music used as an obstacle. Although this is a small game, it helped paved a way and show the people in the video game industry that there are many ways of having AI behavior implements other than statically and dynamically. This project will be served as the base where in the future new features and improvements such as having 3D polygon models instead of sprites, and have more user interaction. As for the beat-detector, there are plenty of room for improvements to make have better accuracy and accommodate to all music genres. Having a beat-detector shouldn't be the only analyzer tool . Perhaps with more research on sound it's possible to analyze more on music characteristics and put that to use in video games. But more importantly it will inspire game developers to not just integrate music into already-made games, but create a whole new concept that's specifically made for music and will hopefully add a new genre to the video game industry.

References:

Beat Detection Algorithm Article
http://www.gamedev.net/reference/programming/features/beatdetection

http://www.rsblsb.com/

FFT library
http://www.fftw.org

XNA game shell:
http://www.phstudios.com