**Generate Game Contents based on Audio Spectrum Data:  
The Rotating Circle**

**Introduction**

There are many music games, like Guitar Heroes, Dance Dance Revolution, Geometry Dash, Osu, etc. where the gameplay simulates the playing of musical instrument or dance, which require players to perform action based on what they see on the screen. Even though these games are rhyme games, they have one thing in common: the music doesn’t generate and influent the game environment, which is the scene in this case, as what happen on scene is actually handmade. This project will focus on doing the same thing as those rhyme games do, but this time, the music will be the on who influences the game environment. To do that, this project will make use of the spectrum data of a given audio to create an object called “The Rotating Circle”, which will be the one which generates contents, using Unity Game Engine.

**Similar Works**

There are many games that attempts to use the actual music to create the player’s experience, such as AudioSurf, Beat Hazard, Melody’s Escape, Rhythm etc. The algorithms for each of these game is unknown, but there is one problem in these game: it doesn’t look or feel like the music is controlling the game environment, and each song doesn’t feel that much difference from one another based on what is shown on screen. This project will try its best to rectify this problem.

**How to get Spectrum Data**

The function in Unity, AudioSource.GetSpectrumData provides a block of the currently playing audio source's spectrum data. The data is then put in an array of size that is a power of 2 (64, 128, 256, etc.), with the minimum of 64 and the maximum of 8192. In order to do this, the audio must go through an Fft (fast Fourier transforms) window, which s. takes a signal (the audio signal in this case) and break it into sine waves of different amplitudes and frequencies. Each element in the array will contain an amplitude correspond to a frequency ban, which represent the frequency equal to this equation:

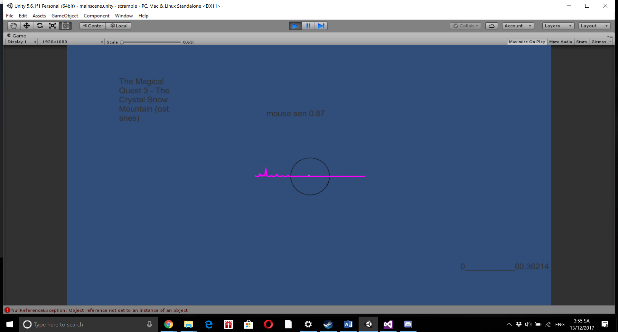
I \* Sampling Rate / (Fft size\*2) (1)

Where I is the index, the sampling rate is 48000 Hz for Unity. I use Blackman Harris because it gives the best quality.

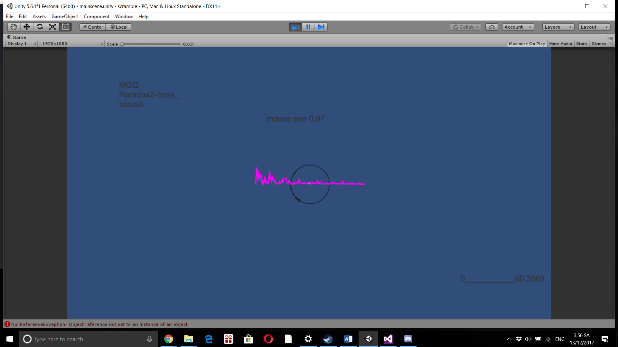
**Goals**

A good algorithm for using audio to generate contents need to accomplish these goals:

1) Each song should feel different from one another.  
2) The player should clearly see the change on scene as the music progresses.  
3) Create variety between each playthrough.

One of many things that make the song difference is it spectrum information, which will be demonstrate in these 2 pictures.

This picture shows the spectrum of the soundtrack named The Crystal Snow Mountain from The Magical Quest 3 OST, a peaceful and sad soundtrack in tone

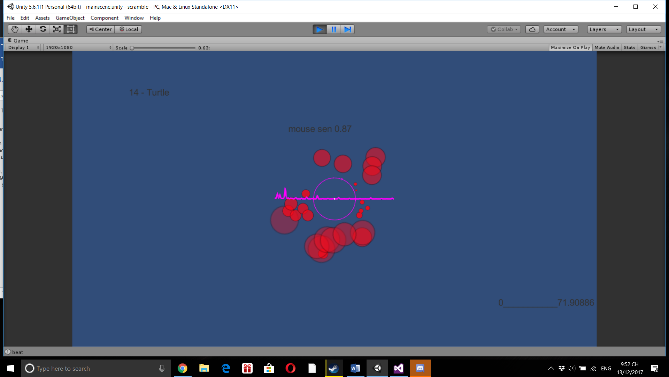
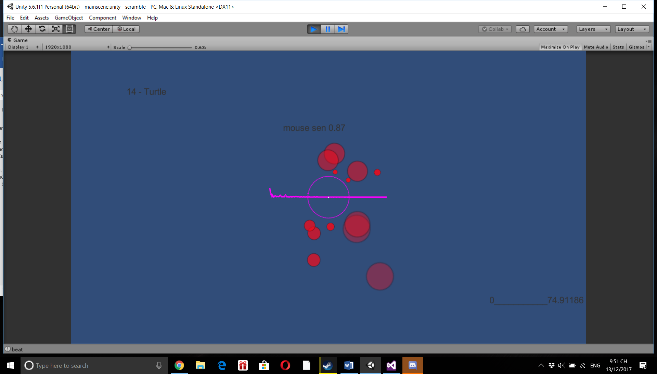


This picture shows the spectrum of the soundtrack named boss\_sonia3 from MGQ Paradox OST, a much more energetic soundtrack that resemble rock.

Therefore, I decide to use spectrum data of a song as the input for this project, sine each song has its own spectrum data and there are many possible input happen in every frame.

**Attempts, Problems and Final Decision**

There are several problems when trying to use frequency data. The first problem is that the frequency data is very erratic and change every frame. The first attempt was to make a platforming game where the player needs to move on platforms which change position based on the frequency bands in that platform. However, the change is too fast, making the platform moving up and down too quickly. There is option to smooth out the change, but then many fast and rapid beats won’t show up prominently enough. Also, putting the platforms in a line would make the environments too big for the player’s field of view, 2D or 3D, since in order to make each song different from each other, there has to be a lot of bands and the Fft window needs to be big. To counteract this problem, I decided to put these platform in a circle around a point making it easier to see, which is where the idea of “The Circle” came from. After that, however, it stills doesn’t solve the problem of the frequency change being too fast and chaotic. Therefore, I decide to abandon the idea of using the frequency bands to directly influence an object’s position, scale, rotation etc. and come up with another way for frequency data to work: instead of using the actual value of the frequency bands. Since this is still not good for directly influencing an object’s position, scale, rotation etc., I make it so that this will create objects instead (this will be elaborate further later). Another problem is choosing the best size of the Fft window and how many bands need to be used. One can make the number of band in the circle equal to the size of the Fft window for the most accurate frequency data. However, for the 16 song I have chosen, the lower frequency bands are the most active in most song and there are very little to no activity if you go higher. For the size of the Fft window, based on equation (1), making the value below 1024 will compress the band too much as if you lower the Fft window, the gap of each frequency band increase twice as much (the Fft window size still need to be a power of 2), which in turn also compressed most change in each frequency band to lowest ends of the spectrum. That is, for the Fft size of 64 on the same song, the first 10 bands or so will have the most movement, limiting the amount of useful inputs, while an Fft window of size 8192 (the maximum Fft window size Unity allow) will increase the amount of useful input since the frequency data is spread out to more bands and the frequency gap between each band is lower. Our goal is to make capture movements across as many bands as possible in the number of bands we put out while separate them wide enough to make a difference. However, if the Fft size is too large, the gap between each band will be lower, which requires the use of more bands in order to use capture meaning full movement.



These 2 pictures compare the different spectrum data in the same song with 2 different Fft window sizes (the left one is 256 while the right one is 1024). The right one has more movement across the bands, as the frequency gap between each of its band is 4 time smaller (the 2 picture has the same number of bands shown, which is 250. 250 bands for the left picture will represent almost all of the maximum sample rate of 48000 hz, which is unnecessary (mpst song can’t go that high), while those 250 would only represents a fraction of the maximum sample rate). Since a smaller Fft window size will make the gap between each band larger, the information each band show with a smaller Fft window size is not as accurate.

After many testing, I pick the Fft window size of 1024 and choose to use 250 of the 1024 bands as input.

**Creating a game**

Depend on the kind of game mode, there are many adjustments to be made to the original circle model to fit the mode. Right now, there are 2 possible modes. **1. Bullet Hell mode**

There are some problems in making this work. In this mode, each of the 250 bands will shoot out a projectile. After multiplying the value in each band with 60 (the actual value ranges from 0 to 1), I, at first, make it so that each time the value of a band is more than a certain value, it would shoot about a projectile forward. This was not a clever idea, as it will shoot a projectile every frame (which is about 60 per second), causing the screen to be overflow with projectile and severely affect performance. Simply increasing the value required for a spawn would not help, as song with lower amplitude (the Turtle mini boss from The Magical Quest 2, the Weasel mini boss from the same game etc.) will simply not register. The other option is to create a cooldown of sort to limit the amount of projectile spawn. i.e. for every 0.3 seconds, the game will check each band to see if the amplitude of that band reaches a threshold to see if it should shoot out a projectile at the time. This creates the problem of making every song feel the same, and the problem of a song either not being registered for having a low amplitude because the threshold is too high or simply make all the bands shoot because the threshold is too low, and no one know what is the lowest, highest or average amplitude of the bands to create the best value for the threshold in real time.

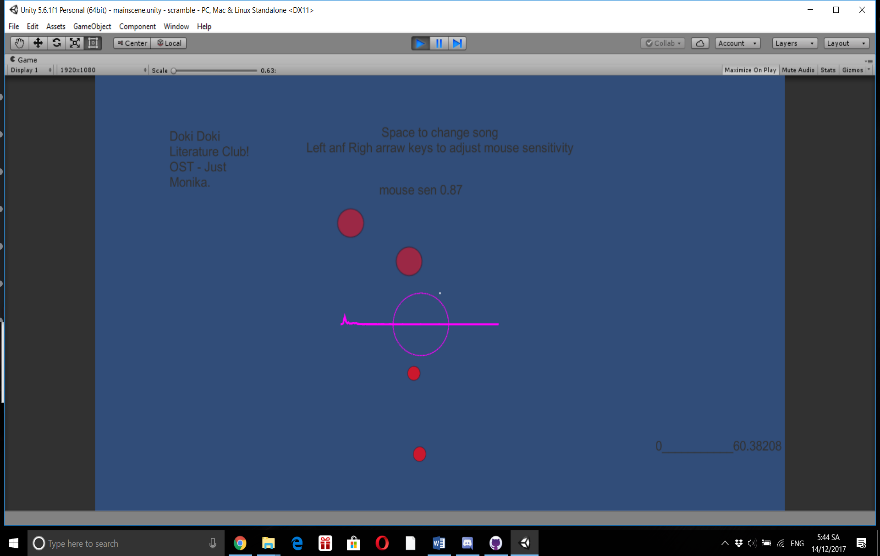
To solve the problems above, I have to rework the system a bit. Instead of making a global cooldown for all the band, I put an internal cooldown of 0.3 second on each band. Also, instead of focusing on finding the best threshold for every song, I instead make it so that every time a band’s value increase by 1 (after the actual value is multiply by 60) since the last frame, it will shoot out a projectile. This significantly reduces the amount of projectile on screen, while still preserve the speed of the song, If the song is slow enough, the player can still see changes when there is a beat happens, while if the song is fast enough, the player can still see changes with the cooldown of 0.3 seconds (base on human reaction time). The projectiles shooting out can be used in a bullet hell as either enemy to be shoot down, or bullets to be dodge.

**2. FPS mode**

This mode is an offshoot of the previous mode. The difference is the game will in in first person mode, where the player, in first person, need to shoot down the target that will come out of the bands like they are in bullet hell mode. However, the amounts of targets to shoot out is still too much for an Fps style target shooting system, even after many adjustments made in bullet hell mode. To fix this, I impose a limit on how many possible targets can be spawn in each frame to 6, and every time a band shoots, the code will ignore 7 bands ahead. This is still outputting too many targets, especially for faster song (Paradox, Red Raven 3, World of Alice etc.). Therefore, I use a beat detection algorithm so that each time a beat is detected in a frame, all bands in that frame is allowed to shoot out a target (provided that other requirements for that band is fulfilled). Still, there are songs that has extremely rapid beat (Paradox, Red Raven 3, Requiem, Dragon Pete etc.) To counteract this, I limit the amount of beat to be as most 2 beats per second (0.5 interval) in all circumstances. The reason behind this is the same as the reason for adding an internal cooldown in each band: if the song is slow enough, the player can still see changes when there is a beat happens, while if the song is fast enough, the player can still see changes with the cooldown of 0.5 seconds (because of the requirement of a target shooting mode, this cooldown need to be more than 0.3 seconds). Each target’s speed is changed to be influenced by the amplitude of the band it comes out and will slowly come to a stop. The goal is to shoot them out before they disappear on their own (a target last for 4 seconds), in which, if there is a scoring system, will incur a penalty.

**The Rotating**

The reason for introducing rotating object is because, in FPS mode. After many hours of play testing, I start to get used to each song’s pattern of spawning, and some songs like Just Monika, Turtle Mini Boss and TMNT, has most movement focus on 1 section of the spectrum array for a long time. Rotating the circle would cause targets appears in more spaces on the scene and the pattern to be harder to figure out. There is also an option to make it so that the rotation is random, to create a new challenge every time (one can do this by checking the Random box in the game actor named Rotating Point).



This picture shows what FPS mode look like. The circle will shoot out targets with varying speed based on the spectrum data of the song in real time. The player need to take them down faster than they can disappear on their own

**Result**

After finishing the prototype, the bullet hell mode is turn into FPS mode (since FPS mode came from modifying the code from bullet hell mode). The FPS mode is mostly finished (except for a scoring system). This mode manages to accomplish the 3 goals set out before.

**Future Work**

1) The modes are still needing to be polish in order for this to become a real game. Right now, this is just a prototype to show what this “Rotating Circle” is capable of.

2) Finish bullet hell mode and create a coherence game out of it.

2) Creating a scoring mechanics

3) Create more modes

4) A main menu for picking game modes, songs and settings

5) Allowing users to upload a song of their own

**Music from:**

[www.rengoky-teien.com](http://www.rengoky-teien.com)

The Magical Quest 2

The Magical Quest 3

Doki Doki Literature Club

Monster Girl Quest 1-2 Paradox

[www.soundcloud.com/kou-g](http://www.soundcloud.com/kou-g)

Bomberman for TurboGrafx-16

Borderlands 2

NES TMNT

**Beat detection code from:**

Allan Pichardo <https://github.com/allanpichardo/Unity-Beat-Detection>