Name: Tjun Jet Ong
Andrew ID: tjunjeto

15-112 Term Project Proposal:

<u>**Geometronome Dash**</u>

Project Description

Geometronome Dash is an adaptation of “Geometry Dash”, along with some additional features. Geometronome Dash works very similarly to Geometry Dash, in that you have a square that clears obstacles according to the rhythm of the music. Geometronome Dash aims to recreate that while still allowing the player to input any music file they like. After which, the game will use a beat-detection, pitch-detection, and smart map generation algorithm to generate the obstacles according to the rhythm and intensity of the music. The frequency of obstacle generation will utilize beat-detection algorithms while the complexity of the map will depend on the loudness and pitch of the music at that particular point of the music.

Competitive Analysis

After doing some research on similar projects online, I realized that there are quite a few projects on randomly-generated geometry dash maps. The most notable Geometry Dash randomizer online is the youtuber GD Colon's Geometry Dash randomizer. (https://www.youtube.com/watch?v=h2WygHkAyT4) There have been other projects that randomize geometry dash maps, but they are mainly very similar to GD Colon's. GD Colon's Geometry Dash randomizer makes use of a map generation algorithm that will randomly generate solvable maps in geometry dash and at the end of the gameplay, it outputs the map so the user can see what kind of random generation was going on. Furthermore, it also divides the game into sections via the use of "Portals". These portals are generated at specific time intervals of the music.

My project is similar to GD Colon's in that we both explore using smart, random map generation in order to generate solvable maps for geometry dash. The essence of our project is to remove the boring and mundane, pre-coded obstacles in Geometry Dash and add in a more "random" factor to spice up game play. However, from most geometry dash variants I've looked at online, including GD Colon's, none of them create maps based on beat detection and pitch detection algorithms. Hence, my project seeks to go a step further to explore random map generation based on whatever music the user inputs, so that anyone can have fun playing geometry dash while listening to their favorite song using music analysis tools like PyAudio and Aubio.

Structural Plan

This structural plan will give an overview of how the code is organized. We will first go through the different files used in the code, as well as the different game modes in the main.py file.

Folders and Files for Organization

- main.py
- bpm_detection.py
- pitch_detection.py
- shapes.py
- sound.py
- Images
- Music

Here, I will explain the intention of these different files.

main.py

This is the main code which is the crux of the project. Playing the game requires running from the main code. This main code includes a Model-View-Controller (MVC) framework that defines the different components of the game.

The most important component of main.py will be the game modes. The different game mode control functions are all stored within main.py. This includes the keyPressed, mousePressed, timerFired, and redrawAll functions that will define what the user will see in the different modes.

bpm_detection.py

This is a file that contains the algorithm of beat detection. It contains a two main functions required for audio analysis:

1. bpm_detector

This function uses Discrete Wavelet Transform using PyWavelet to detect possible high amplitudes of rhythm. It then filters the data and obtains the key features of the signal.

2. get_bpm

This samples the sound signal at specific intervals and tries to get the beats per minute (BPM) value within that specific time interval. It then creates a list that stores all the bpm values, and we will use numpy to compute the median of that list. We will then use the median bpm as the frequency of generation of obstacles for this project.

Pitch_detection.py

This python file takes in an audio file and makes use of the audio analysis libraries **Aubio** and **PyAudio** to perform signal analysis on the different pitches of music. It samples the sound signal at specific time intervals and will get the pitch of the music within that specific time interval. It then creates a list of pitches which we will eventually use to compute the intensity of obstacle generation later on within the game.

shapes.py

This is a file that contains all the classes of shapes that will be used in the project. This will include the main character of the game - the magic square - which is initialized as a class of its own and will be called upon as an object in the main code as app.magicSquare. It will also create classes of all the shapes of the obstacles so it is easier to draw and randomly initialize the obstacles coming towards the magicSquare during the game.

sound.py

This file contains the class object of Sound that was adapted from cmu_112_graphics.py that will be called upon in the main code in order to play the music during the game.

Other Folders

1. Music

This folder contains all the music files that have been loaded by the user so that it can be called upon and played during the game.

2. Images

This folder contains all the images that will be used in the game's User Interface.

Different Modes in the User Interface of the game

This structural plan will be organized into the different game modes, where the game mode changes depending on where the user clicks:

- splashScreenMode
- gameMode
- mapPackMode
- highScoreMode
- pauseMode
- gameoverMode

Here, I will explain the different modes that define the structure of the project:

Splash Screen Mode

The game starts off with a splash screen, where the users will get the pick between three buttons:

1. Play Button: For them to play the game (redirects to gameMode)
2. Map Packs: Allows users to load songs and access previously loaded songs (redirects to mapPackMode)
3. High Scores: Redirects users to see their high score for every song. (redirects to highScoreMode)

Game Mode

After the user clicks the “Play” button, the user will be directed to the actual game. This game will feature a smart randomly generated map based on the rhythm and intensity of the music, allowing the user to have fun playing geometry dash.

The frequency of obstacles appearing uses a beat detection algorithm whereas the difficulty of the obstacles (E.g. height) depends will utilize pitch and loudness detection.

Map Pack Mode

There are two main features to the map pack mode - allowing players to load songs that they like, as well as access the previous songs. The user will have to choose the song that they want to play in map pack mode, which will allow them to play on the play button mode.

1. **Loading songs**: Users can input the file of the song that they want. Using File I/O, the file will be stored and the random map generated will also be stored.
2. **Accessing songs**: Users will be able to access all the songs that they have loaded before.

High Score Mode

This will contain the user's highest score for every song that has been played. Using File I/O, there will be permanent storage of the song's high score on the device. Hence, every time the game is restarted, the user's high score will be there.

Pause Mode

During the game, the user can press pause at any time. This will allow the user to either go back to the home screen or continue playing.

Game Over Mode

Whenever the user crashes into an obstacle, the game is over. A game over screen is shown that will prompt the user to either restart the game or go back to the splash screen.

Algorithmic Plan

The trickiest part of the project will have to be to integrate the beat and pitch detection algorithm into Geometry Dash. This section aims to cover how I will approach the trickiest part of this project.

Beat Detection

In order to perform beat detection, we will use the Python File **bpm_detection.py**. This algorithm samples the signal at regular time intervals, and makes a prediction of the Beats Per Minute (BPM) within that time interval using Discrete Wavelet Transform (DWT). Then we will obtain a list of BPMs (variable list called bpms) and we will get the median of all those bpms (variable float called bpm).

This may be difficult because after getting the average bpm, we will have to integrate it with the game's timer to ensure that the obstacles come out at those specific time intervals. Since timerFired is expected to have certain delays, I will instead use time.time() to and use the modulo factor to compute the intervals at which the obstacles should appear.

Pitch Detection

After the beat detection is properly implemented, I will use pitch detection to decide how the map should look at different intervals of the song. If the song is at a very intense part (e.g. the chorus), then the pitch detection algorithm should be able to detect that the song is entering its climax. Thus, this will alter the map in ways such as increasing the frequency of obstacle generation and increasing the height of the obstacles. Thus, the pitch detection algorithm will be useful so that the map is not mundane and only relies on beat detection.

Using both Beat Detection and Pitch Detection for Smart Random Map Generation

Finally, we will combine both beat detection and pitch detection to obtain the final part of the project - the Smart Random Map Generation. The game should eventually be smart enough to come up with maps just by using beat and pitch detection, and maps can be generated effortlessly for the user to have a good experience playing geometronome dash with any song the user picks.

Timeline Plan

The following is the timeline for the project:

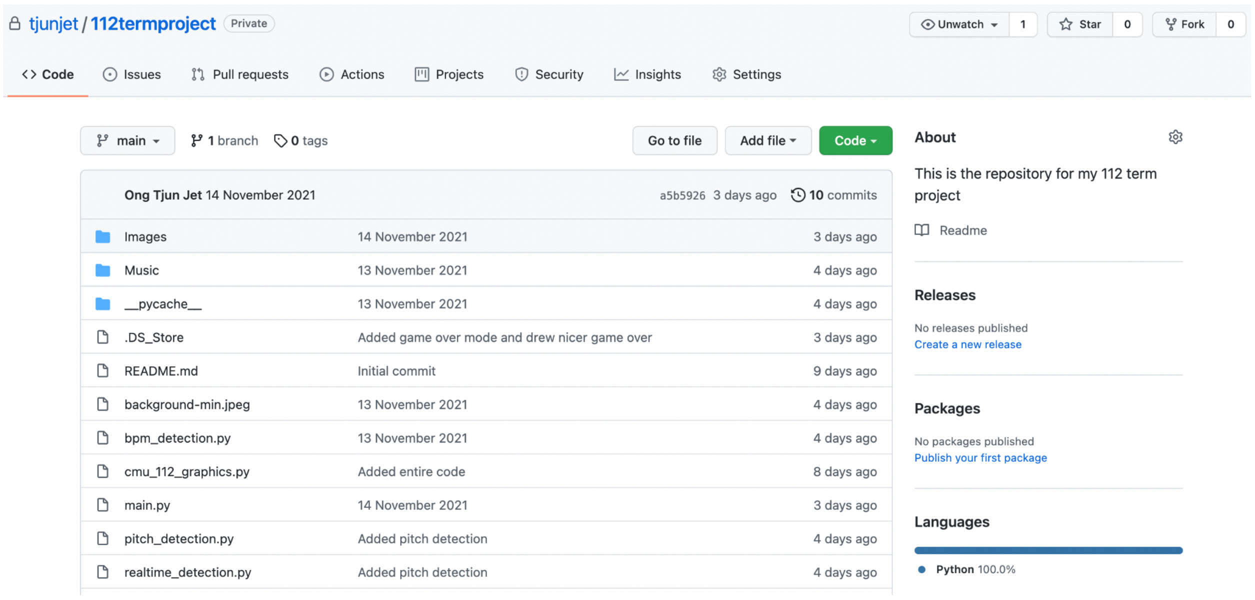
Date	Deliverable
18 November 2021 (TP1)	Briefly develop the UI of the project. Briefly implement bpm_detection in Geometronome Dash. Include Images and Sounds in the project. Debug any form of bugs that have occurred up to this point.
20 November 2021	1. Implement all the different game modes for the user to be able to navigate from one game mode to the other as outlined in the storyboard. 2. Create different classes of shapes. Two different shapes in Geometry Dash - Square and Triangles. If the magicSquare goes on the square, the game is not over. If the user comes into contact with a triangle, the game is over. 3. Improve the design of Geometronome Dash as a whole. This can include using images as the magicSquare and obstacles. 4. Figure out how to save high scores using File I/O. 5. Implement a percentage scoring system that calculates the total length of the song and prints the percentage of the song completed at the top for the user to keep track of their progress in the game.
23 November 2021 (TP2)	1. **Perform beat detection**. The obstacles should be generated based on the beat of the music in the game. This means that whatever the music file is, the game's algorithm should be able to generate the average beats per minute of the music and allow the users to have a fantastic experience playing Geometronome Dash as they can also experience the beat of the music. 2. Create two different game modes based on the portal. Ideally, ½ of the game in the duration. If the magicSquare goes through the portal, the ground will be gone and the square starts to fly.

26 November 2021	1. **Perform Pitch and Loudness Detection:** The map should now be able to be generated based on the loudness and the pitch. The louder the music, the more intense the music is. This will allow the map to become more intense and the game will become more intense. 2. **Integrate Smart Map Generation Algorithm:** With beat detection and pitch detection in place, the map can now be generated smartly, randomly. At the end of the day, the map must also be solvable. 3. **Users can upload any music they want and a map will be generated:** Users will now be able to place whatever music they want. Then, the algorithm will generate a random map based on the music file that is uploaded. Now, the user can play geometronome dash to any music they want!
29 November 2021	**Some other ideas that can potentially make this project more interesting and worth a try** 1. Using Deep Reinforcement Learning to allow the magicSquare to play any randomly generated map 2. Using Deep Learning to generate the maps based on the music detected instead of using a coded Smart Random Map Generation Algorithm. 3. Real Time Pitch to Map Detection that takes in microphone audio inputs and outputs the map almost immediately after. **Other things to finish** 1. TP3 Video 2. ReadME file on Git
1 December 2021 (TP3)	Final touch ups and edits

Version Control Plan

Github will be used for Version Control. Currently, I have already created a git repository to save the code. At every end of the day, I will use the commands “git add .” and “git commit -m” to add comments on any key alterations that I made for the project, as well as the “git push” command to push the entire code into my git repository. This ensures that in the event of a loss in the local file, I will still be able to pull the code from git.

This is a screenshot of my git repository that illustrates my version control plan.



Ong Tjun Jet 14 November 2021 — a5b5926 3 days ago — 10 commits

Images	14 November 2021	3 days ago
Music	13 November 2021	4 days ago
__pycache__	13 November 2021	4 days ago
.DS_Store	Added game over mode and drew nicer game over	3 days ago
README.md	Initial commit	9 days ago
background-min.jpeg	13 November 2021	4 days ago
bpm_detection.py	13 November 2021	4 days ago
cmu_112_graphics.py	Added entire code	8 days ago
main.py	14 November 2021	3 days ago
pitch_detection.py	Added pitch detection	4 days ago
realtime_detection.py	Added pitch detection	4 days ago

About

This is the repository for my 112 term project

Readme

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Python 100.0%

Module List

- Pygame (To play sound during the game)
- PyAudio (For audio analysis)
- Aubio (For audio analysis)
- PyWavelet (For Discrete Wavelet Transforms)
- Numpy (For easier computation of lists)
- Scipy (For signal analysis)
- Wave (To read .wav files)

Project Update

There are three main updates to the project ever since TP1. Firstly, there has been the addition of new game modes. Next, pitch detection was integrated into the actual code. Finally, moving forward, we will also try to integrate loudness detection to change the color of the background.

New Game Modes

The game can now be played in 3 different modes. At every 10% interval of the song, portals will be generated. Once the magicSquare goes through the portal, it will be able to reach a new game mode! The new game mode is randomly picked using a randomizer.

- **Default Geometry Dash Mode**

The default geometry dash mode features obstacles that are generated based on the beat of the music. The height of obstacles will vary based on the pitch of the music. The map is generated using a smart random-map generation algorithm which ensures that the map is always solvable. The magic square will have to jump across the obstacles in order to avoid game over!

- **Reverse Gravity Mode**

In Reverse Gravity Mode, every time the spacebar is pressed, the magicSquare teleports. Meaning that if it is currently on the ground, it will teleport to the ceiling and vice versa. The height of the obstacles and frequency of obstacle generation is dependent on the pitch of the music. The higher the pitch, the more frequent the obstacles are generated and the greater its height.

- **Zig Zag Mode**

In Zig Zag Mode, pressing the spacebar key will change the direction of “drop” or “fly”. Meaning to say that if the magicSquare is currently dropping, and spaceBar is pressed, the square will start flying. The square will move in a zigZag motion and the height of the mountains varies based on pitch. The smart generation algorithm ensures that there will always be a path for the magic square to go through.

Integrating Pitch Detection

A pitch detection algorithm was used to generate the obstacles. Apart from beat detection, where every bpm, the obstacles would be generated, the varying pitch would determine the frequency of obstacle generation as well as the height of the obstacles. The higher the pitch, the greater the frequency of obstacle generation (For example, generating it in quarter-notes rather than half-note intervals), and the height of the obstacles would increase. However, despite that, the map generation algorithm will detect if the height gets too high, and will vary the height accordingly so that the game can always be completed.

Loudness Detection

The final feature that I will eventually integrate is loudness detection. Using external modules like aubio and PyAudio, the loudness of the wav file at specific time intervals will be determined. The loudness of the sound will determine the background color of the music. (For example, the louder the sound, the brighter the color).

TP3 Update

Addition of mouseMoved features

In order to improve the user experience, whenever the user hovers their mouse over a button in the splashScreen, the image of the button that they are hovering over enlarges. This adds a fun part to the game as it makes the user experience more interactive.

Added Pause Button and Pause Screen

Now, users are able to pause the game at any time of the game. Whenever they press the pause button at the top left of the screen, they will enter pause mode. The user can either resume the game by pressing the resume button or the user can go back to the splash screen.

Displaying the score on the pause screen and gameover screen

Now, when the game is paused or when the game is over, the screen will display the current/ final score of the game. Hence, the user will now be able to see their score.

Stopping the game if the game has ended

If the song ends, the game now stops. A congratulations screen will be printed in order to show the user that they have completed the game with 100% score.

Ability to input music files

In mapPack mode, a new feature is added to the game. Users can now input the file path of their song into the input rectangle shown. Now, the song that is associated with the file path given will be the song that the user can play the game to.

Ability to save high scores

Using File I/O, high scores can now be saved in a file called “high_scores.txt”. The code will calculate the top three high scores and display them in highScore mode.