15-112 Term Project Proposal

Kevin Lin

Project Description

My term project, named Flappy Beat, is meant to be a game where the player moves in the same manner as they would Flappy Bird and with the goal of dodging obstacles generated by a music visualizer. It will feature the ability to play against an AI opponent for multiple song choices.

Competitive Analysis

Most of the popular EDM music accounts on YouTube feature very attractive visualizers, all of which stand out because of two aspects: color and spontaneity. In terms of color, the “coolest” looking visualizers feature most colors on the visible light spectrum. Furthermore, the fact that these colors are rarely connected as a continuous rainbow, along with the seemingly random movements of the visualizer, are what primarily provide spontaneity. These visualizers also usually feature a very attractive background image, usually of a landscape or cityscape that is both darkened and vignetted in order to better contrast with the brighter visualizer in the foreground. White particles also float in the background for added aesthetics.

Perhaps the most popular channel that frequently utilizes visualizers is Trap Nation, whose display is centered on a circular logo surrounded by the actual moving elements of the visualizer. This project will also feature a circular logo in the center, but the mechanics of the visualizer will be completely different. For videos released by Trap Nation (and pretty much any YouTube account), visualizers are directly connected to the center and feature fluid shapes. However, the visualizer for this project will feature only circles, whose collisions with the players will be very easy to detect. These circles are also hurled as obstacles outwards towards the players instead of remaining connected to the center. The vital component of spontaneity can be found in the random color, speed, and location of every obstacle generated. Finally, the player movements will be pretty much identical to that in Flappy Bird, with the only difference being that players can move up-left and up-right also. While interactive games with music visualizers exist (e.g. former top 112 term projects like Hextris), where the user must time their key presses with a certain action of the visualizer, it is likely that none of them have the user directly become a part of the “world” of the visualizer (like in Flappy Beat).

Structural Plan

The structure of the game will be broken up into 6 different .py files. They are as follows:

- **Main**
  - Consists of the game loop (using a subclass of PyGameGame), different game screens, and music visualization algorithm and frontend.
- **Player**
  - Consists of the Player class, which defines the flappy-bird style movement of both players in the game.
- **User**
  - Consists of the User class (subclass of Player), which defines the user's movement based on which keys they press.
- **Opponent**
  - Consists of the Opponent class (subclass of Player), which defines the opponent's (AI) movement based on the locations of nearby obstacles.
- **Obstacle**
  - Consists of the Obstacle class, which defines each obstacle spawned from the center of the visualizer.
- **PyGameGame**
  - Consists of the PyGameGame class, which completes all required initializations.

In addition to these files, there are two folders: one for all songs that are analyzed, and another for all images that are used.

Algorithmic Plan

The most difficult algorithm to write would be that of generating the music visualizer at the core of the project. For every song file, thousands of data samples will be collected at every second with aubio.pitch. This built-in method gives us information on both frequency and amplitude, which correspond to the pitch and volume measurements (respectively) that will be added as a tuple to a list. In order to reduce lag, this project will utilize the average data from every 5 samples, which will be stored in a new list. Within timerFired, each index of this new list will be read in perfect alignment with the timing of the song, which can be done by manipulating the rate at which data is sampled. The pitch and volume for each sample when then be used to determine the speed and size of each generated obstacle, which flies outwards in a straight line from two symmetric but randomized locations on the center circle.

Timeline Plan

Friday, November 22:

- Finish setting up structure and variables for all classes
- Finalize algorithm for data collection from audio file

Saturday, November 23

- Use audio data to generate moving obstacles for music visualizer
- Add user to game (along with responses to keys, collision detection, etc.)

Sunday, November 24

- Misc. debugging
- Finish Game AI

Monday, November 25

- Create end (win/loss) screen

Tuesday, November 26

- Create title screen, selection screen, and instructions screen (see storyboard for details)

Friday, November 29

- Enhance graphics for aesthetic appeal

Sunday, December 1

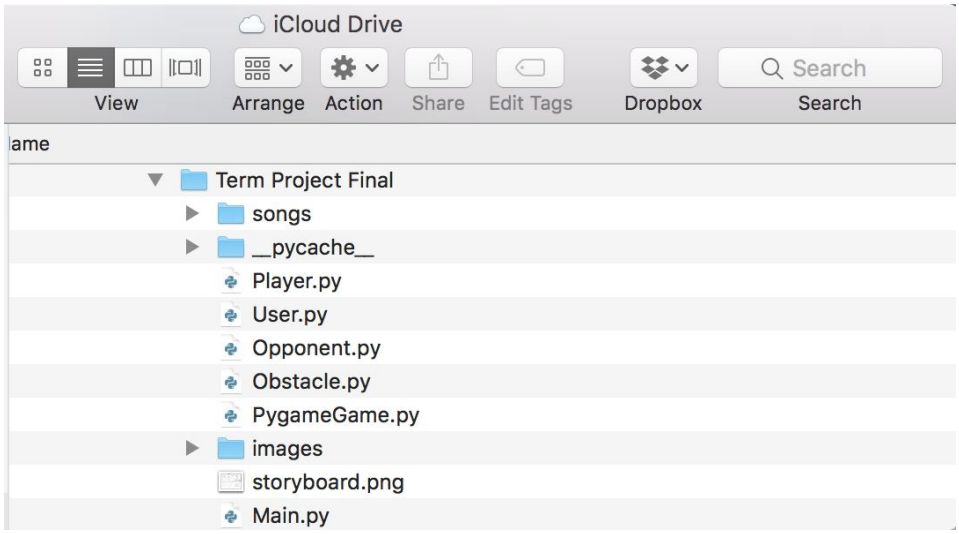
- Add multiplayer mode and user power-up.

Monday, December 2

- Misc. debugging
- Make final trailer

Version Control Plan

- All of my computer's files are backed up to my family's iCloud Drive:



- Since my family lives in Europe, the time zone difference may prevent me from contacting them to access these files immediately. Thus, I will also be emailing the same files above to myself every evening.

Module List

- **Math** for sin and cos (pretty much all circle calculations)
- **Random** for spontaneity of obstacle location, color, etc.
- **Pygame** for screen, mouse, and key control.
- **Pyaudio** to play audio
- **Aubio** to analyze audio

TP2 Update

- **Competitive Analysis**

Rather than aiming for a display that randomizes the color and speed of the obstacles, I decided to keep these two components constant, as too much variety for so many moving parts makes the visualizer look off beat. Thus, every obstacle will be white and travel at the same speed.

- **Structural Plan**

A new .py file named SuperObstacle (subclass of Obstacle) was added. After the MVP requirements were clarified, this class was added to implement large obstacles that will be shot out when the current beat is double the average volume. I also added another .py file named AIOpponent, which is separate from the Opponent class because the former is defined for an opponent (a second player) in multiplayer mode.

- **Algorithm Plan:**

While I still used aubio.pitch for data collection, the key change I made for TP2 was that the visualizer now responds to beats (also known as onsets) rather than changes in pitch. The data collected for beats is in the form of the time at which a new beat is detected, which facilitates data extraction from the new dictionaries I created for pitch and volume.

- **Module List**

I forgot to add Numpy to the list last time, which was a module I used (although only for one line) during my aubio tech demo. Also, instead of using pyaudio to play a song, I have decided instead to just use the built-in pyGame method for doing so.

TP3 Update

- **Algorithm Plan:**

A few additional complex features were added upon hearing suggestions from the TAs after reaching the Minimum Viable Project (MVP) stage. They are as follows:

- ➔ Additional game screens (title, help, setting/selection screens) beyond the actual gameplay setting were implemented. The selection screen was by far the most complex additional component for the following reasons:
  - ◆ The selection screen would display different sections depending on the user's progress in selecting game settings.
  - ◆ The selection screen is able to detect which future selections are valid based on previous ones. For example, if the user selected to play in a solo mode, then each of the 12 players would be open for selection. However, if the user changed their mind and instead selected a multiplayer mode, then the selection screen would ensure that two different players are selected before moving forward.
  - ◆ Gameplay is only allowed when all valid selections have been made.
- ➔ An "endless" mode was added to the gameplay. In this mode, the game doesn't end upon the end of one song, but instead only ends when the players score 0 points or leave their given boundaries. The algorithm in place for MVP was used, with the additional feature that if the user selected an endless mode game, then soon after the last note onset, the same song would play again and the same visualization would start over. The visualization uses a parameter that keeps track of elapsed time to actually visualize the music, so this re-setting is done easily by making the elapsed time 0 at the same time that music is played again.
- ➔ Lag was significantly reduced overall. This was done by collecting all data from all songs before the game actually ran. While the game now takes a few more seconds to load, this algorithmic change resulted in a huge improvement from a sometimes laggy game at the MVP stage.