Get Down, Jump Up:

An interactive Music Visualizer

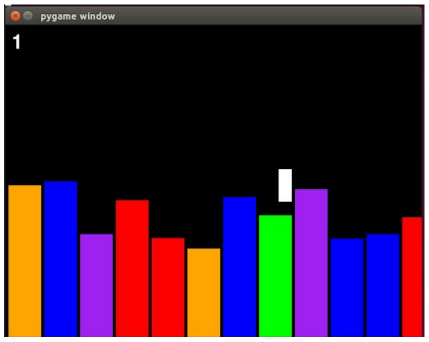
Project Overview

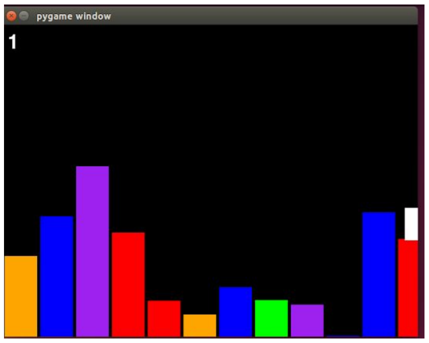
Write a short abstract describing your project.

Our project is a classic music visualizer that has bars which correspond to the amplitudes of different frequencies in the microphone. It also has a video game component - a small rectangle “character” that can traverse the bars and score points by going side-to-side.

Results

You can move the character back and forth, jumping across the bouncing, multicolored music bars and score points by touching the edges of the screen. The height of the bars is related to noise coming in from the microphone, but isn’t the best music visualizer ever. A score is tallied in the top left corner.



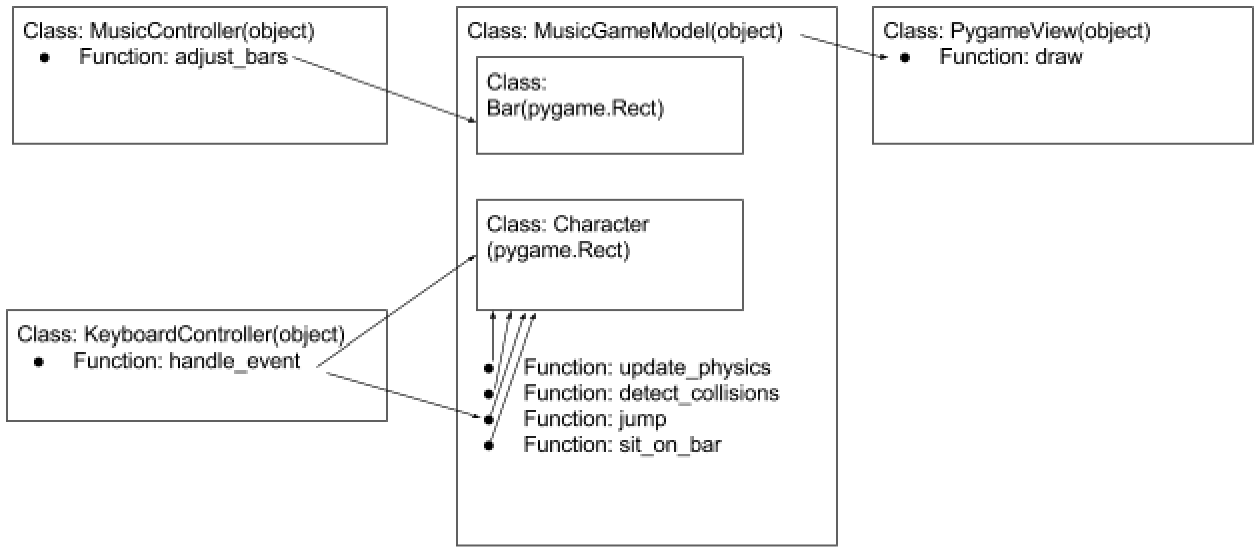


Implementation

The code operates through a loop with a delay of .1 seconds between iterations. Each loop, it checks if the user has quit the game, checks if they’ve manipulated the character through by pressing the arrow keys, updates the music visualizer bars to reflect what the microphone is hearing, checks if there are any new collisions and moves the main character out of the way if necessary, updates the character’s velocity and position, then draws everything on the screen.

One of the cooler decisions we made with the structure was that after we learned about inheritance, we made our music bars and character classes each a type of pygame Rectangle. Something else that changed mid-way was how we processed the music - we intended to process and pickle each song beforehand, but realized it was actually better to do in realtime as part of our running loop. That way, we could just use music coming in from the mic, instead of having to sync it up and play it with our code.

Diagram:



Class: MusicController(object)
Function: adjust_bars
Class: MusicGameModel(object)
Class: Bar(pygame.Rect)
Class: Character (pygame.Rect)
Function: update_physics
Function: detect_collisions
Function: jump
Function: sit_on_bar
Class: PygameView(object)
Function: draw
Class: KeyboardController(object)
Function: handle_event

Reflection *[~2 paragraphs]*

Neither of us ever felt overworked or panicked about the project. We worked at a reasonable pace and weren't scrambling to finish it at any point. The scope of the project was challenging but very doable. It was difficult to unit test in pygame so we ended up test running the program as we went and saw what worked and what didn't. Although we didn't end up using pickled music files in our final project, we using pickled music files worked as a unit test while we were implementing the audio portion of the project.
It would've been good to know how to set our music bar and character classes as pygame Rectangles in the first place as well as that we'd be taking live data from the microphone as opposed to a digital song. It took us awhile to figure out the code wouldn't run because the microphone was off; it would've been much better to know this sooner. We both got much more comfortable using classes, libraries, and methods

We worked together fairly well as partners but didn't really plan out how the work was going to be distributed so it ended up being pretty imbalanced. The one of us who did more of the work however never felt burdened and always like what she was doing. The one of us who did less wish she could've done more but personal circumstances made it difficult until later on. The further we got in the project the better idea we had of what we needed to do. This made allocating tasks easier and helped balance out the workload more. There were some communication issues throughout including divergences in our Floobits repository that were cumbersome to deal with, but we got better with that over time as well. If at all possible, next time we will try to do better at allocating tasks and making clear what work we want to do respectively from the get-go. Also we'll try to do better at communicating when major changes are being made and do more programming side-by-side as opposed to apart.