Project Description:
Music Run, a game that turns music into a running game.
Competitive Analysis [5 pts]: A 1-2 paragraph analysis of similar projects you've seen online, and how your project will be similar or different to those.
Main (those found in app stores):
Temple Run: Utilizes similar graphics and is structurally similar in the sense that it is a run game with obstacles. Different in the obstacles occur on the beat of a song and does not have a demon chasing you. Also, no powerups.
Piano Tiles: Simiar in that the player does things to the beat of a song. Different in that Piano Tiles is not a run game, so it does not utilize physics.
Others (online games):
Upbeat: online game that uses keys to play the notes of a song. Similar controls, as I too will be using alphabet keys for notes and the Space Bar for rhythm. Different in that it does not utilize physics.
Overall, it seems that similar projects I've seen online use either physics (like a run game) or a rhythm, but not both.
Structural Plan [5 pts]: A structural plan for how the finalized project will be organized in different functions, files and/or objects.
Will be utilizing the animation framework provided by 112.
Splash Screen Mode will be one file, with Buttons being in the same file to enable easy calling.
Load File Mode will be one file. It will have the Splash Screen file imported into the file so as to utilize the path.
Game Mode will be its own file, with the possibility of helper files imported into the game.
Aubio will be utilized to detect beat and pitch, and all functions involving the module is in its own file.
All of this will be imported into the Main file, which is where the ModalApp gets called.
Algorithmic Plan [5 pts]: A detailed algorithmic plan for how you will approach the trickiest part of the project.
Jumping:
Breaks the physics down into x, y, and z coordinates. Since we're mainly just dealing with kinematics, we can assign some g (not neccesarily 9.81). We can have a variable in appStarted that keeps track if the player is on solid ground. If it is, then the z coordinate does not change. If it doesn't, then in timerfired, we can have the vertical velocity increase by g every second, and subtract the velocity from z. If z drops below a certain point, it's game over. If you land on solid ground again, onSolidGround becomes True and vertical velocity resets to 0. Just in case we missed something (and so the player doesn't glitch), we can reset z back to its default value.
Moving Forward:
Call timerFired to increment y by something. This should work as long as the path is straight.
If path is not straight:
Corners:
Use trigonometry to make it look like the path is rotating. Parallel assign x and y. Then continue as normal.
Curves:

Use Trigonometry, but use a mod in Timerfired so it goes slower. Don't parallel assign x and y
Obstacles: Get range of coordinates of obstacles. If at any point the coordinates of the player is inside the range, you bumped into it and it's game over.
Generating Obstacles: Ideally, use Aubio. Keep a timer at the beginning and at every beat, lay down a "jumping" obstacle or having a corner, which one would clear by using Space or using a letter key (Like H or L), respectively. We can calculate when the obstacle appears using amount y increments, which essentially equate to speed. We use d=vt to calculate where to lay the obstacles. Note that the faster the song, the closer the obstacles are.
Utilizing pitch: Can use pitch to generate point boosters and/or have it be "stars" flying by your head as you run (just trying to make it look pretty). Also, if time permits, get the collection of notes to find the key of the song. Use key to get color scheme of map. Low saturation is minor key, high saturation is major key. Atonal music (think Flight of the Bumblebee) get rainbow with mid saturation, etc.
Pyaudio (If time permits): Just play the song while the program's running.
Psuedo 3D graphics: Need more information. Hasn't quite gotten that figured out yet.
Timeline Plan [5 pts]: A timeline for when you intend to complete the major features of the project.
Should be Done:
SplashScreen: Should be Done
Load File Mode: redrawAll should be Done.
High Level Physics of GameMode
By Saturday 11/23:
Should have figured out how to load the file onto Load File Mode.
Should have the Physics framework done
Should know how to implement psuedo 3d graphics on a high level.
Should be able to load file into Aubio and return the results.
By TP2:
Should be able to have the psuedo 3d graphics framework done.
Should have a buggy rough draft of completed game.
TP2 and beyond:
Add cool stuff, like pitch visualization and whatever.
Version Control Plan [3 pts]: A short description and image demonstrating how you are using version control to back up your code.
Using Github to periodically upload files.
See Version Control.jpg
Module List:
Aubio
Pyaudio (Only after TP2)