Project Description

Another Platformer Project

By Jason Lam (jasonjlam)

A very difficult platformer based on I Wanna Be the Guy, I Wanna Be the Boshy, and other similar platformers. Features stages instead of the traditional side scroller format, very difficult platforming, varied level design, and pixel perfect movement.

Competitive Analysis

Platformers are a dime a dozen on the internet, and probably a dime a dozen when it comes to term projects. The platformer that I am taking inspiration from is “I Wanna Be the Guy” and its various derivatives (I Wanna Be the Boshy, etc). These are “kaizo” platformers, or very difficult games that often rely on trial-and-error to get through levels and to learn patterns. They feature some sort of weapon to get past enemies, a reset mechanic to account for thousands of deaths, and very difficult levels with some sort of gimmick. They are also famous for the difficulty of their boss fights, along with the hodge-podge of assets taken from other games.

My project will be similar in design, but not as large in scope. It will use the same style (patchwork of different games and their assets, difficult gameplay, and boss fights). However, due to time constraints my game will have a much smaller variety of environments, entities, and challenges.

Structural Plan

I am planning to split my project up into a lot of different files to organize it.

The game itself will be handled by game.py, which uses cmu_112_graphics.py and its various app/canvas functions to actually play the game. This includes handling all the drawing onto the canvas, creating the app, storing information, and other things for the game. It also handles controller input and translates that into movement.

The stages will be handled by stages.py, which has a base Stage class that imports from a .csv created using Tiled. Each stage has its own class that inherits all properties of Stage, along with any entities special to that stage. It also uses the .csv file to generate all of the tiles in the stage. These tiles are stored in the stage, separately from the entities and the moving tiles.

The player is handled by player.py, which deals with movement of the player. It handles all the complex calculations needed for collision detection, player movement, and other player functions.

The entities will be handled by entities.py, which has a base Entity class that has its own version of collision detection. This also counts things that can kill you but don't move, in order to consolidate code into one place. Each type of entity is a class that inherits the properties of Entity.

The tiles will be handled by tiles.py, which has a base Tile class that has its own version of collision detection. This not only has squares, but also slopes and some interesting shapes. This is what is imported from the .csv file.

The audio will be handled by audio.py, which uses Pygame to play sound and background music.

All assets will be included in the /assets folder. As of the moment assets are in the main folder, but once I finish up a sprite sheet I can use that to create a tileset, along with make animation for characters. This also includes sounds played.

All stages will be included in the /stages folder. These are .csv files that include the location and the types of tiles for a given stage.

All Tiled files will be included in the /Tiled folder. These are just the files used to edit stages, and then export as .csv.

Algorithmic Plan

By far the most difficult part of the project is collision detection. I used a combination of speculative contacts and bounding box detection to handle collisions with the player character. While this is relatively easy for enemies (contact results in death), for the player this requires being unable to pass through terrain. For each movement, I can move the player in the direction of the movement vector one unit at a time. If at any point the player intersects some geometry that it shouldn't, the movement vector is cut off. If I want to retain momentum in a certain direction, I can also slightly shift the vector in a certain direction rather than stop entirely. This process is called once for the x and once for the y axis, to check collisions separately. This code also handles moving platforms and slopes, based on behavior when contacting the bottom part of the character.

I also had difficulty with the jump mechanics of the character. By mapping all key presses to a dictionary representing whether or not the key is being held down, I can track whether or not a character wants to make a long jump or a short jump based on how long it is

being held down (holding the y velocity to a certain amount). I can also give the player the ability to double jump, that refills either in certain areas or when touching the ground.

Finally, I will work on enemy AI and make them interesting. While some enemies may be simple (move in a straight line toward the player), others may obey collision detection, or even pathfinder towards the player using some kind of pathfinding algorithm. I hope to implement some kind of AI for bosses that isn't just rotating through the same attacks every time, adapting to what the player is doing and acting accordingly.

Timeline Plan

Tuesday: Adding entities and different tile types

Wednesday: Implementing shooting and saving

Thursday: All glitches in movement ironed out

Saturday: Finalizing the first set of stages

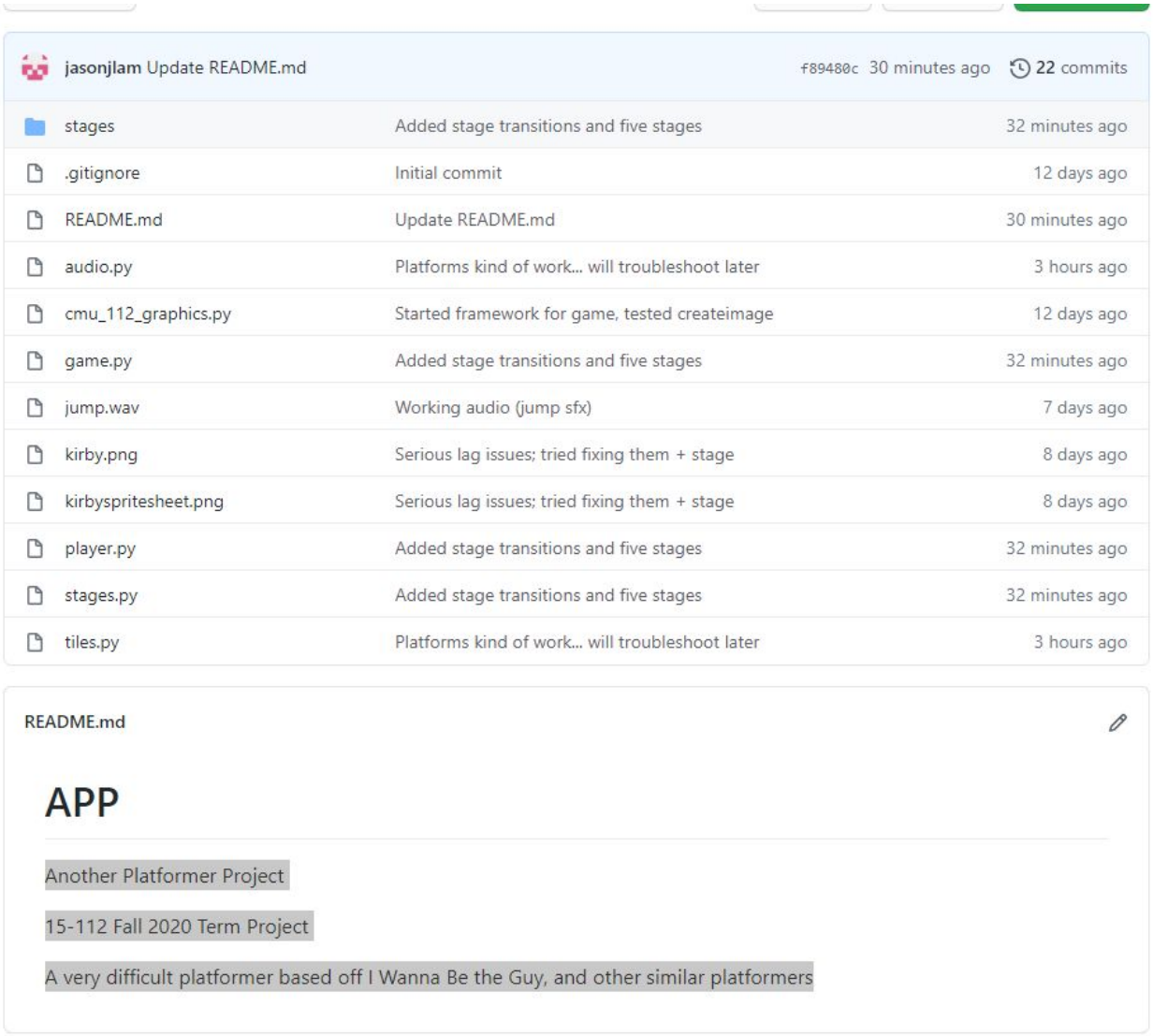
Monday: Boss fight finished

Tuesday: Implementing proper graphics

Version Control

I use Github to back up all of my code, along with a local save.

https://github.com/jasonjlam/APP



Module List

Cmu_112_graphics.py

Pygame (audio only)

Tiled (to create csv files for stages)