**15-112 Term Project Proposal**

**Project Description**

This project will be based on the rhythm game “A Dance of Fire and Ice” and my project is currently called “Never Ending Circles”. Note that the name of my project is a placeholder and could change in the future. In the game the player will guide two circles that are constantly orbiting one another, and the only input that the player needs to play is a keypress whenever a circle overlaps the path that has been set to be in time with the beat or melody of the music. If the player fails to press the key while the circle is on the path, they will be returned to the latest checkpoint passed. Here is a [YouTube video](https://www.youtube.com/watch?v=bwDm4ThMsao) showcasing the first set of stages of the original game. Gameplaywise, there will be modifiers in the path that change the speed of orbit, change the direction of orbit and that act as checkpoints when the circle moves onto them.

**Competitive Analysis**

There is a python project named [yuGen](https://tanwyhang.itch.io/yugen) by [SeniorOne/Tanwyhang](https://www.youtube.com/c/SeniorOne) which based on the popular rhythm game “Osu!” where the user clicks on circles that pop up on the screen in time with the music. It will be similar to my project in that it is a rhythm game where the player needs to input keypresses in time with the music and will likely implement similar functionalities but the design of the game itself will be different since in yuGen the player needs to use the mouse to move the pointer over circles and click while my project will only require the user to press a key in time with the music. Furthermore, the structure of the levels will be different as my game will have a path that will be followed while yuGen has circles that pop up on the screen.

Here is a [platformer game](https://www.youtube.com/playlist?list=PLjcN1EyupaQnHM1I9SmiXfbT6aG4ezUvu) made by [Coding With Russ](https://www.youtube.com/c/CodingWithRuss). The game takes a tile approach to creating the levels in the game which will likely be the same as my game’s approach to making levels since it fits that the aforementioned path the player will follow in my game would be created using tiles of the path connected to one another.

**Structural Plan**

Great care was taken during the design process to ensure that the game will be scalable, meaning that as more types of paths and gameplay features such as speed and rotation direction modifiers are added, no major changes to the structure of the code will be needed. This was done by taking a modular approach to the structure of the level (making levels by putting together tiled paths) and ensuring the properties of circular motion are easily editable (speed of rotation is stored as number of rotations per second)

**Classes and Files:**

\_\_main\_\_.py

* The python file that will import the main game class and will be run to start the game.

game.py

* The python file that will contain the main game class ‘NeverEndingCircles” which contains the main loop and the main logic of the game.

player.py

* This file will contain the ‘Player’ sprite class containing all attributes and methods that affect the player such as the image, mathematical formula for circular motion and snapping to tiles. In the main game file two instances of the ‘Player’ class will be created for the two circles.

tile.py

* This file will contain the ‘Tile’ sprite class which are 100-pixel side length squares that will make up the paths in the levels of the game using a modular approach.

camera.py

* This file will contain the ‘Camera’ class which will contain attributes and methods used to control the camera of the game.

levels.py

* This file will store the layout of the levels in the game. The game will take a “tiled” or “modular” approach to making and loading levels by using arrays to store the sequence of path tiles. The array will be read and using the information in the array, the path of the level will be set and displayed on the screen.

music.py

* This file will contain the ‘Music’ class which will run the music for the levels. This class will also play sound effects and manipulate the music such as stopping when the player fails and restarting at the appropriate point in the music when the player spawns at a checkpoint.

**Algorithmic Plan**

Regarding the circular motion of the circles, equations of uniform circular motion will be implemented. For the coordinates of circular motion, x = x centre of circular motion + (radius of circular motion \* sin(angle)) and y = y centre of circular motion + (radius of circular motion \* cos(angle)). Then, attributes of the circle will include the angle needed for circular motion equations, x and y coordinate of the centre of circular motion and rotations per second calculated by ((2\*pi)/Frames Per Second) \* Rotations Per Second.

For the structure of the levels, the layout of the paths will be stored in arrays in the levels.py file which will be imported and read by the ‘NeverEndingCircles’ class in the main game file and the sequence of tile paths will be set by reading the content of the array. Then, the information stored in the array will be passed to the tile class which will then determine the type of path tile (straight, right/left turn, 45-degree turn) that will be set in sequence. Instances of the Tile class will be stored in a pygame group class in the order that the path tiles are set and displayed in the window screen.

**Timeline Plan**

**TP 1**

October 17th – Have the core functionality of the game working which would be the circles orbiting and being able to move across a simple path.

**TP2**

October 30th – Have a level select stage and at least one playable level done that has features such as checkpoints, and modifiers. I should also have a score system that scores the player based on how accurate and consistent they are, and a save file system to save the user’s scores.

**TP3**

November 8th – Have multiple playable levels and add polish to the game such as smooth transitions, sound effects and interesting camera movement. Also implement functionalities such as “invincible” mode that makes the level play itself and a “practice” mode that allows the player to create their own checkpoints throughout the level.

**Version Control Plan**

I will be using GitHub. - <https://github.com/en0chC>

**Module List**

**Pygame**

This will be the main module I will be using to create the game.

**Pydub**

This library will be used to implement the music in the levels. It is useful as it allows me to splice the audio, start from any point in the music file, and to fade in and fade out the music which will be needed for the checkpoint feature in my game.

**TP2 Update**

**Files and Classes**

Added a gameStates.py file that contains the parent class “GameState” and child classes “Title”, “MainMenu”, “LevelTransition”, “Gameplay”, and “PauseMenu” which are the different game states of my game.

**Code Structure**

Restructured my game.py file to allow for different game states that can transition from one to another. The game states are stored in a stack data structure such that the bottom of the stack is the title screen and game states are linked to one another in a hierarchical structure. This allows transitions to game states by popping and pushing game state classes onto the stack and accessing the current game state by calling functions from the instance at the top of the stack.

**Libraries**

I am no longer using pydub since it requires multiprocessing to play music while allowing other functions to run which causes a seemingly random amount of delay when starting the music, causing the gameplay and music to be off time with one another. I then moved onto using pygame’s mixer module but that also had its problem that made it unsuitable for my project which is the inaccuracy when setting the starting position of the song. I am now using the vlc library to play the level music.

**TP2 Feature Changes**

I am no longer going to have a local file system to save the user’s scores and instead will be storing user information on a server which I will do for TP3.

**Added Features**

* Snap Camera.
* Level Class for storing level information in an array and loading a level by reading the level’s array and creating appropriate consecutive tiles encoded in the data from the array.
* Music Class to play the music for the levels.
* Speed Change, Reverse Direction and Checkpoint tiles.
* Stage fail condition and sound effect.
* Created all needed tile type image assets.
* Different game states that allow for game menus and level reloading.
* Score system that scores the player based on how consistent and accurate they are.