Tracky Bird

SoftDes

Byron Wasti, Robert Siegel, Andrew Deaver

March 2015

1 Introduction

For our project, we decided to create a Flappy Bird clone using pygame in which we used OpenCV to track the user flapping their arms up and down to control the bird. We wanted this game to look smooth in regards to the bird's movements and consistent at tracking the user's arm flaps.

2 Results

We were successful in our implementation of the game. We ended up with a clone of Flappy Bird that requires both arms to control the flapping. The game does everything that you would expect the flappy bird game to do. This includes accurate falling physics, collisions with pipes, and start/end screens. With the start/end screens, we were able to implement the use of the flapping motion to start the game. We were also able to keep track of high scores locally. Below are screenshots of the game.

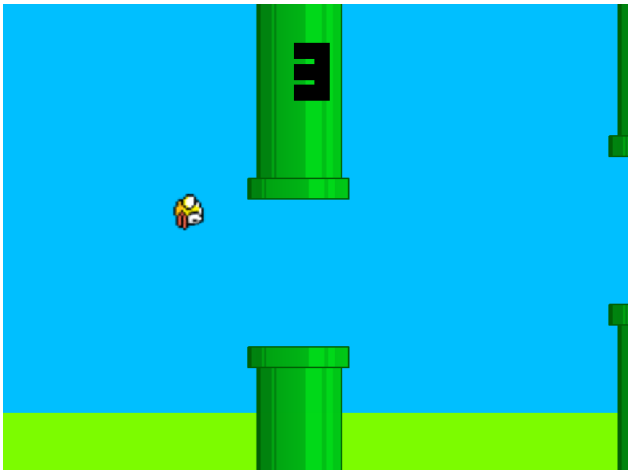

Figure 1: A screenshot of playing the game, this player has made it through 3 pipes succesfully.

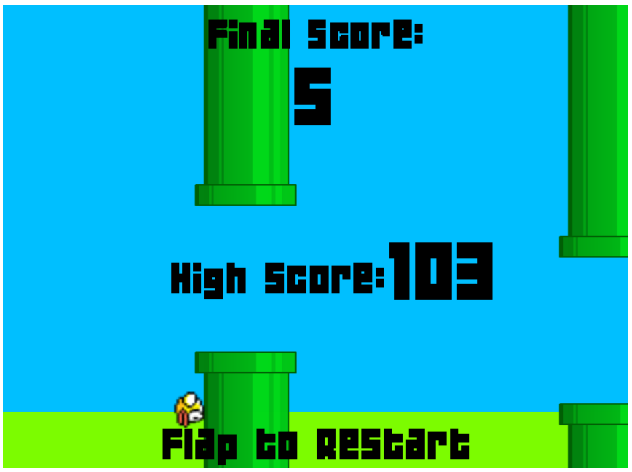

Figure 2: The death screen for Tracky Bird, showing persistent high score count. Yes, we got that good at the game.

3 Implementation

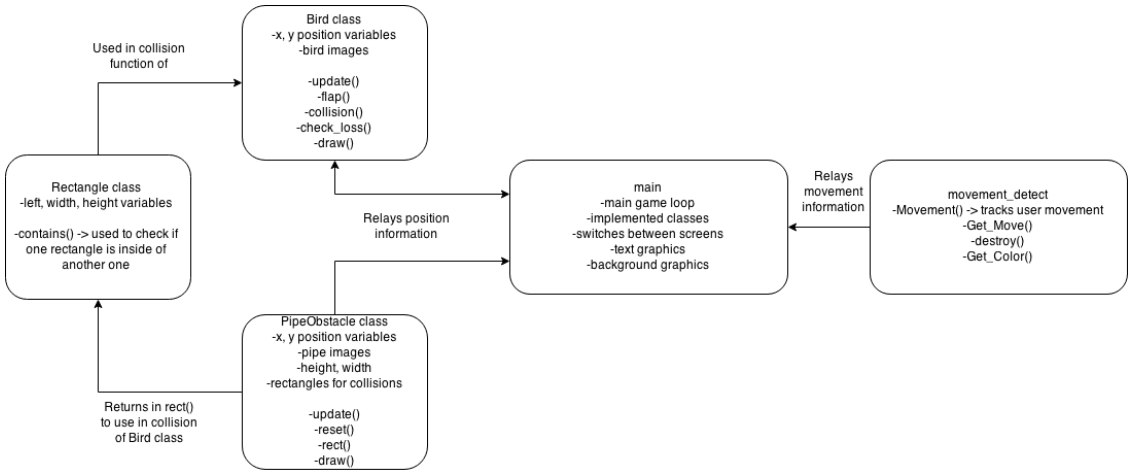

Figure 3: UML Diagram explaining the relationship between classes.

3.1 Game Classes

We used three classes in our implementation of Tracky Bird. The Bird class contains information about the bird, including its x and y coordinates as well its velocity, which is used to update the x and y coordinates. The Bird class also contains and displays proper images as it is falling through the air. The Bird class contains functions that update its position and vertical velocity, check for collisions with pipes and the ground, and draw the image. The PipeObstacle class contains information about the height and width of the the two sides of the pipe, as well as their x and y coordinates. The PipeObstacles have a constant gap size and a constant velocity. This class contains functions for updating the x position as the pipes move left across the window, resetting the position for when the screen changes, and drawing the pipe image in the appropriate location. Lastly, there is a Rectangle class used for collisions, which are checked in the Bird class. The only

function that the Rectangle class contains, other than its constructor, is a method called contains, which checks to see if a given rectangle intersects with the Rectangle that calls the function.

Although there is no inheritance from one class to the next, all of the classes transfer information to the other classes to update drawing and game information. This can be seen in collision checking. The PipeObstacle returns a list of rectangles which encompasses the top and bottom pipes. The Bird class then receives this information in the main game loop and generates a Rectangle around itself and uses the contains method to determine if its rectangle is contained in the rectangles of the Pipes that are on screen.

It is important to note that main is not a class, but rather just a game loop that calls the Bird and PipeObstacle class and draws the appropriate screens.

3.2 Motion Tracking

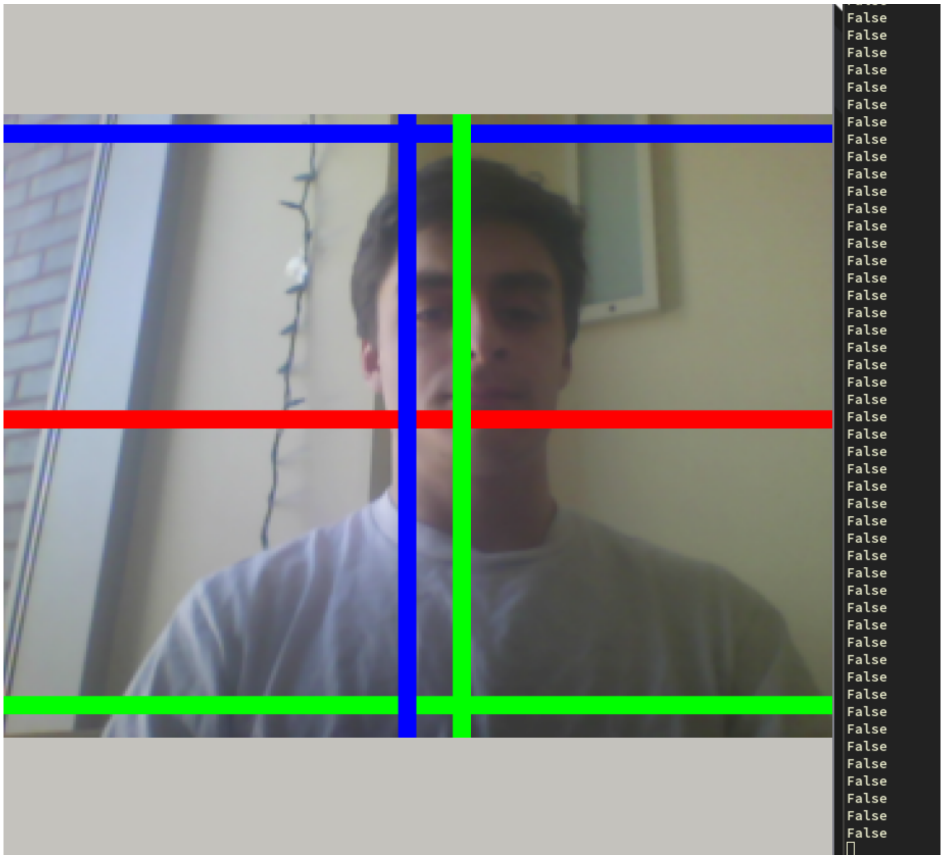

Figure 4: Testing out the motion tracking to detect flapping motion. Red horizontal line shows the cutoff point for when a flap is registered, and the vertical bands are to ensure that both arms are flapping.

A major part of our implementation of Tracky Bird was to accurately detect flapping motions of the user. There are a number of different techniques we could have used, ranging from the easiest being color tracking (and having the user wear a colored glove) to arm detection. Due to the time constraint, we decided to implement motion tracking.

Motion tracking was done by using OpenCV to take frames and subtracting the previous frame from the current frame using numpy subtraction. This gave white pixels and a lot of random noise, which we filtered for color. In order to speed up this and future processes, a pyramid algorithm was used to make the image files smaller.

Once a mask of only white pixels was available we swept through all of the points and summed up the average y position of the white pixels, and the max and min x positions of the white pixels. The output would register as a flap if the max and min x positions were farther apart than 50 pixels and on either side of the middle of the screen, and the average y position **drops** below the cutoff point (shown in red in Figure 4). The cutoff point is determined by being the midpoint between the maximum average y point found and the minimum average y point found in order to allow for it to self center around the user's motions.

4 Reflection

As a team of three, we made sure to split up the work very carefully. Byron worked mostly on motion tracking because he wanted to develop his OpenCV skills. Andrew created the structure of main and all of our classes. Robbie developed the Bird class, which included the game physics, and also focused on editing previously written code for bugs and readability. This division worked well, as we all worked on what we originally intended. Once each of us completed our own tasks, we helped the rest of the team move forward with their goals and improve what we had so far.

One of the main learning steps was how to use git more effectively. We also learned a lot about communication, which was crucial to our success as a team of three. Github was one great way for us to communicate well. By pushing our code frequently and checking the work of each other, we were able to work together and make our program more efficient.