Quinn Roemer & Logan Hollmer

Dr. Daryl Posnett

CSC 133 MWF 9-9:50am

February 12, 2020

Assignment 0 – Documentation

While deciding on ways we wanted to modify the code to refactor and practice the OOP concepts we have been learning in class one of the first things we noticed was how the screen sizes X and Y values were being passed around to many functions making it hard to track and debug. As a result, one of the first changes we made was to encapsulate this data into a class of its own called "screenInfo." Inside this class, we have protected variables that hold values for the X and Y coordinates as well as calculated values for several other items linked to the screen (font size, margin size, debug size, etc). These values are calculated in the constructor. The reason we decided to use private variables and write "getters" for all of them is because we saw no reason for the screen information to change once created (in this use case). Therefore, we determined it to be wise to prevent the variables from changing after the object is first initialized. An item that was modified that is linked to this addition is the removal of redundant screen info saved inside objects. Instead, methods that require screen information to operate correctly are passed this object. Also, this method adds abstraction by removing the font size and margin calculations outside "PongGame" and into an object that automatically calculates them based on the initial screen size.

Another change that was made was having both the "Bat" and "Ball" classes extend the "RectF" class that is provided by Java. Our intent in doing this was to simplify the detect collision method and make it more apparent to what exactly is going on in that code. Originally we thought about creating an abstract shape class that extends "RectF" and having both "Bat" and "Ball" inherit from this class. However, with further investigation, we determined that because these two classes only had one method in common (update) that this was not the best course of action. By inheriting methods from "RectF" we enable our classes to use methods such as "intersect" without having to define a "RectF" object inside the classes. This allowed us to directly use the "intersect" method inside of detect collision without having to ask if the "RectF" object inside of "Bat" intersects with the "RectF" object inside of "Ball." Linked to this change is the modification of the variables in the "Bat" class that define the direction the bat is moving. Instead of using a 0 or 1 to denote the direction, a string literal is used to more clearly indicate the direction that the bat is moving. For example, instead of the state of the bat moving left being equal to 0, it now equals "left."

We observed that there was quite a bit of code and variables that were required to handle the audio. We found that this was a perfect place for an audio manager class that we called "AudioManager". An audio manager object is created upon initialization of the pong game. This audio manager object detects the current version of

Android and sets up the SoundPool while loading all of the audio files that we are using. From this point, when the game needs a sound to play, it simply calls a method to play the sound of choice. This change accomplishes several OOP principles. It accomplishes abstraction because the person using the class does not know all the inner details such as the detection of the Android version, audio file loading, etc. It also accomplishes encapsulation, because the data and the exact method calls, encapsulate the underlying code in just a few easy to understand public methods.

A series of smaller changes were also made, and while they don't deserve their own paragraph, we will detail these smaller changes here along with the reasoning behind them. The first change was to remove the boolean variable and method for debugging. This variable and method were only designed to show the frames per second (fps) counter while debug mode was on. However, we decided to keep the fps counter in the final version of the game because it was a feature that we liked. When displaying the background color and setting the text color, the original code used Color.argb() to convert an ARBG (alpha, red, blue, green) value into its integer representation in the method calls to Canvas and Paint. What we did is, create private final int variables to hold these values, the variables are named after the color they represent. Now we can simply pass the color variable “White” or “Blue” rather than having to use Color.argb() in the method call. This helps, because we can now easily see what color is being used, “Blue” is a lot easier to understand than “Color.argb(255, 26, 128, 182)”. Also, it promotes easy reusability, we can color something else blue in the future without having to retype it. Finally, we noticed that whenever the ball hit the left or the right side of the screen it would simply reverse its velocity in the X direction and play a sound. Since the same thing happens, we combined these IF statements. This makes the code clearer and easier to understand. Lastly, we performed a general clean-up of the code. Adding further documentation on the code we added, so as to explain the “how and why” behind each modification to the original code.

Image 1: Screenshot of the game

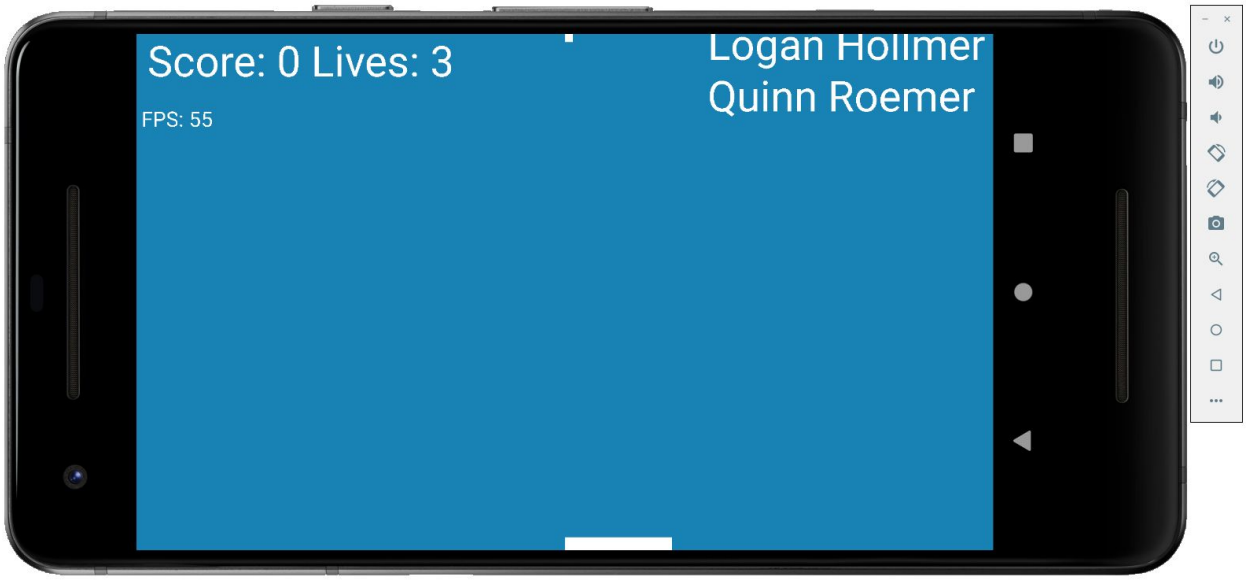

Image 1:
This depicts the game (with the modifications) running in an Android emulator

Image 2: UML diagram of the refactored pong game

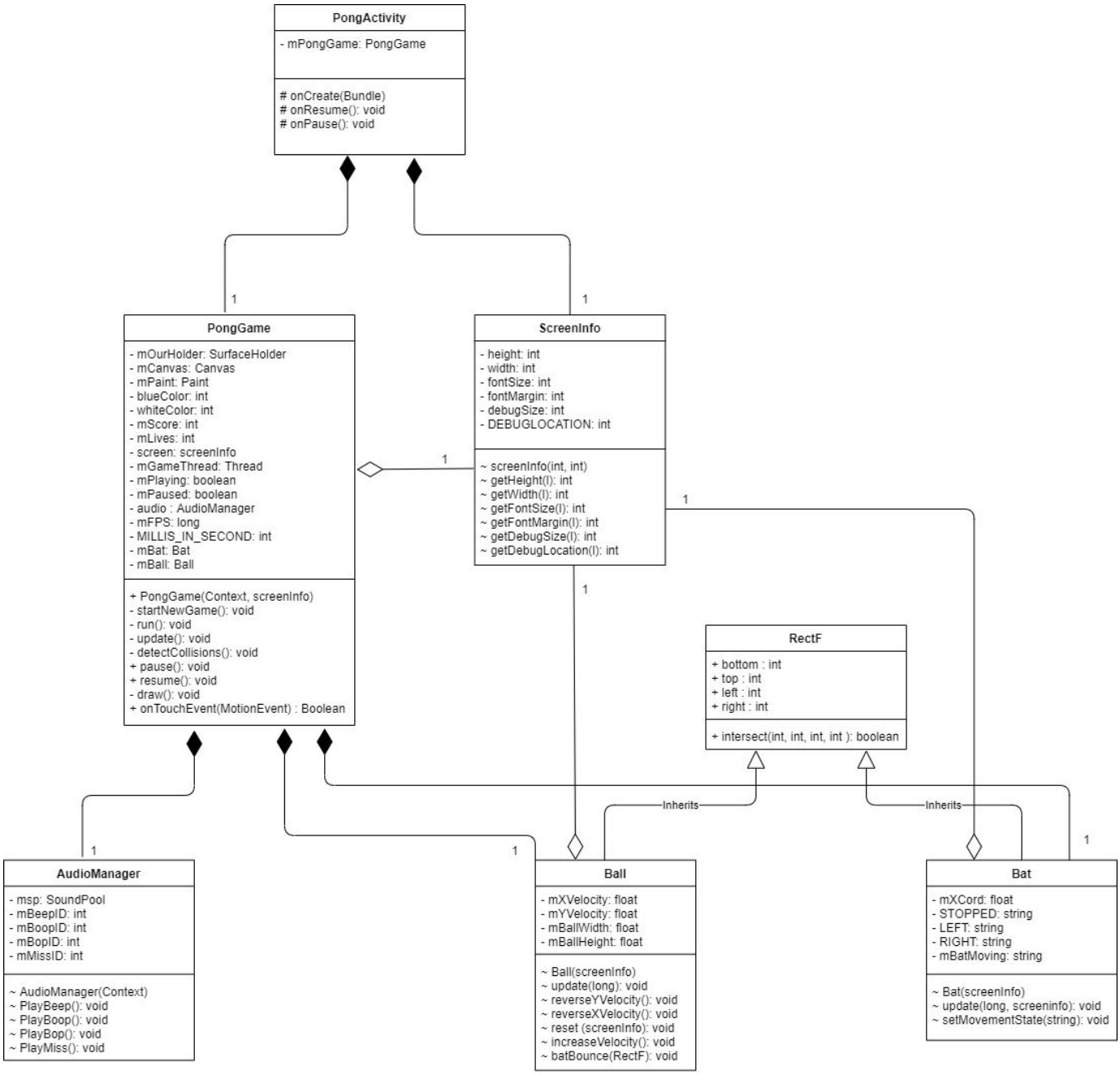

Image 2:

This depicts the UML diagram of the pong game after refactoring. Note that this is not a complete UML diagram as both PongGame and PongActivity inherit from other classes. However, this is meant to give a better understanding of how all of the individual classes interact with each other. Also, there are more functions in RectF then are shown, again this was done simply to visually demonstrate the refactoring changes. The variables and methods that are used from RectF were included in the diagram.