Assignment 3: *Fragments, Images, 2D Graphics, Motion sensors*

The purpose of this assignment is to get help you become experienced with Android application programming and using the Android Studio editor. In this assignment, you are going to practice what we saw in last two weeks of class: apps with multiple activities, activity lifecycle, fragments, camera and pictures, 2D graphics, and hardware sensors.

Instructions

Three different (fun!) project ideas are presented. You are required to choose one (and only one) of the proposed apps. However, the User Interface design is up to you. Time to work on your creativity! There is no minimum or maximum number of activities, just decide what you think it is more convenient for your app.

Submission

Submit your assignment using Canvas, under the Assignment section. The due date is shown in Canvas. You must submit a **.zip** file, containing the following items:

- Android Studio Project folder
- README.txt file (see instructions below)
- video file with a demo is optional for this assignment.

The file named README.txt should contain your name and email address along with the name of your app and a very brief description of it, along with any special instructions that the user might need to know in order to use it properly (if there are any). For example:

```
Laura Arjona <arjonal@uw.edu>
Sorting Hat - This app displays the sorting hat and a picture of the user. It returns the house
where the user belongs.
```

App 1: 2D CollidingBalls

In this app you are going to build a simple *Pinball* game. You will use the accelerometer to detect the user movement of the smartphone, and then use Canvas to generate an animation.

Game instructions:
A ball (referenced here as ball-A) will be moving randomly from top to bottom with a certain velocity. The user should be able to select the velocity. Ball-A should not exceed the horizontal dimensions of the screen. The ball should bounce whenever it reaches the horizontal margins of the screen. In the bottom of the screen, there will be a ball (referenced here as ball-B) constrained in the Y axis (always positioned in the bottom of the screen, and not moving the Y direction), and moving in the X axis. The movement in the X axis will be controlled by the user, by moving the phone. The movement of the phone will be tracked with the accelerometer. If ball-A reaches the bottom of the screen, the game is over, and the user loses the game. Every time the two balls collide (can be interpreted as position of ball-A intersects with position of ball-B), the user gets one point. When the two balls collide, ball-A will bounce upwards, and move from bottom to top, with the same velocity as before, and bouncing sideways. When ball-A reaches the top of the screen, the complete procedure repeats. The game should include an option to stop the game, so that the ball A stops moving around.

Additional notes: to track the movement of the phone using the accelerometer, refer to the slides from Week 4.

Grading

- Up to 80 points: minimum requirements (functionalities exposed above), and overall performance of the app.
- Up to 90 points: the game should have a background music, that the user could enable or disable. And remember to stop the music when the user leaves the app or the game! You don't want to annoy the user with the music when the game is stopped!
- Up to 95 points: show a bar graph with the user points per game (it is ok to lose the data when the app is closed). User GraphView (see example from class in Week 4).
- 95-100 points: use of at least one Fragment (we will investigate the source code for this).

App 2: BlurMyFace

In this app you are going to learn how to use the Google Face API for face detection that was introduced in class in week 4.

Your app should select a picture, detect the landmarks of the face, and then apply a blur filter to the face. The image could be taken with the camera and/or read from the project raw or drawable resource directory (/res/raw or /res/drawable). Only the face area should be blurred, and not the rest of the image. The result does not need to be perfect, but it is good enough to blur a certain area around the central part of the face (the goal is to make the face unrecognizable).

To create the blur effect, a help function is provided *bitmapBlur(sentBitmap: Bitmap, scale: Float, radius: Int): Bitmap?*. However, you can use a library to create this effect if you find it more convenient (for example, blurkit, or Picasso).

The app should also display if the user is smiling or not (based on a threshold given the probability of smile). You can use a Canvas with a Paint and Typeface object to draw text.

Grading

- Up to 80 points: minimum requirements (functionalities exposed above), and overall performance of the app.
- Up to 95 points: shake the phone to start the event of blurring the face (use the accelerometer).
- 95-100 points: use of at least one Fragment (we will investigate the source code for this).

Additional notes: for additional help, refer to the slides from Week 4. See also the supporting document: FaceDetectionSupport.pdf

App 3: SwapMyFace

In this app you are going to learn how to use the Google Face API for face detection that was introduced in class in week 4.

Your app should take a picture and detect the landmark positions of 2 faces in it. If a different number of 2 faces are detected, the app should warn the user. The app should only perform the "swapping" effect if 2 faces are detected. The image could be taken with the camera and/or read from the project raw or drawable resource directory (/res/raw or /res/drawable). Once the landmarks of the 2 faces are known, the app will "swap the faces". One possible way of implementing this is to detect the landmark positions of the face, select an area around it (for example, an oval around the nose), and exchange the pixel values. The result doesn't need to be neat! You could use these functions from the Bitmap class to set and get pixels values

```
setPixel(x: Int, y: Int, color: Int)
getPixel(x: Int, y: Int)
```

Here you can find a details about all the functions of Bitmap (link)

Grading

- Up to 80 points: minimum requirements (requirements exposed above), and overall performance of the app.
- Up to 95 points: shake the phone to start the event of swapping (use the accelerometer).
- 95-100 points: use of at least one Fragment (we will investigate the source code for this).

Additional notes: for additional help, refer to the slides from Week 4. See also the supporting document: FaceDetectionSupport.pdf