Submission. All submissions should be done via git. Refer to the git setup and submission documents for the correct procedure. The root directory of your repository should contain your README file, and your Android Studio project directory.

There is NO code collaboration for homework. Each student must do their own coding and they must do all of their own coding. You can talk to other students about the problem, you can talk to the instructor or TA. If you discuss the homework deeply with someone, note that in your README.

Overview. For this assignment, you will help code up a simple game. You click the button in the top middle of the screen to start. Clicking the button starts a countdown timer, and creates a green square, affectionately referred to as a “puck.” This puck is in a rink, and when the timer starts, one of the rink’s borders turns green. Your goal is to fling the puck into the green border as fast as possible. The faster you do it, the more points you get. Points are measured in tenths of a second left on the countdown timer when the puck reaches its goal border.

The puck moves according to modeled physics. It has friction and if it impacts any non-goal border, it must bounce back as if it had a perfectly elastic collision (it does not lose any kinetic energy on impact with a border). Also test the case where your puck goes into a corner. Try every corner in the rink.

Your app has a single activity. The code as given has build warnings and then doesn’t do much when you run it.

Let’s go over the files in the project. All locations where you need to write code are marked with // XXX write me. You can trust me, there won’t be unmarked areas that you have to change.

- **MainActivity.kt** This is where your code first gets control from the Android framework which launched your app in response to the user click the icon. You can modify `durationMillis` during testing to make life easier on yourself.

  At the end of `onCreate`, you should initialize the two score indicators, the `score`, and the list of frames. I came up with an ordering for frames, which is arbitrary, but we should be consistent. The order is Top, Bottom, Start (Left), End (Right). You might want to name these frames `frameT`, `frameB`, `frameS`, `frameE`, but that involves some layout XML, so let’s not get ahead of ourselves.

  Leave `doScore` and `doRound` alone. You can check out the later for some sweet coroutine action.

  You need to write `newGame`, which initializes `Timer` and `Rink` objects and then passes them to `doRound`. The user plays a round, the score gets updated, then the user can play another round by hitting the big button again. If the user does not clear the puck in time, s/he gets no points, and the score stays the same. Why are there two score indicators that always have the same value? Good question. It is kind of silly, but that is how it ended up.

- **Timer.kt** Do nothing here. If you come up with a better visual effect than I did, post an animated gif to piazza and we will all marvel at your creativity.

  We will not use your version of this file when evaluating your code so DO NOT modify it.

- **AndroidManifest.xml** Do nothing here. The day for this file will come.
• **content_main.xml** What is here is correct (and should not be changed), but you need to add four borders and the rink area. Each border is 12dp thick, colored black (while the game is not active), and is visible at all times. I call the borders `frameT`, `frameB`, `frameS`, `frameE`, and the game area `gameFrame`. Please bear in mind that the game area should NOT overlap with the frame. It consists of the area **inside** the frames only.

I mark the area where you need to write XML like this `<!- - Need frames and rink (gameFrame) - ->` because that is how to write comments in XML.
• **Rink.kt** This is the part that requires the most work. The class takes five parameters. The first is a context object, which are used all over the place in Android. We could have gotten the context from the FrameLayout, but that isn’t best practices. I’ll give you a hint, pass this from your MainActivity for the context object. Don’t pass `applicationContext`, because it has a longer lifetime than your activity. We will talk more about contexts later in the course once you have more experience.

The frame is the rink area, and the borders are a list in order, Top, Bottom, Start, End. The frame and the borders do not overlap. We initialize a random object in MainActivity for you, please pass that to the Rink. Also pass the testing Boolean that we need in MainActitity.

All of the class member initialization within Rink is correct. You can test with different values (I encourage it), but hand in your code with a value of `3.0f` for friction. We will do all testing with that value.

The BorderType enum is a silly example I looked up of how to get a Kotlin enum that can be initialized with an integer.

`initPuck()` is correct, just leave it.

Now let’s skip to the end of the file. `playRound` takes a callback function that you call if the user wins (`goalAchieved`). If time runs out, don’t worry, the MainActivity will take care of cleanup.

The beginning of the function is correct. We remove any previous puck from the frame, call `initPuck()` and `listenPuck()`.

Then you need to do some things like set all of the borders to the color black (`Color.BLACK`), and choose a random integer from `0..3` to find the goalBorder for this round. Change the color of the goalBorder to goalBorderColor.

Then your `playRound` function can return. You have set up callbacks on the puck. So when the user flings the puck, the Android framework will call your code in Rink. That is what `listenPuck` does. Let’s look at that function, because you need to write part of it.

In `listenPuck` we provide code that defines a `gestureListener`, which provides the initial velocity for our fling events. You have to write the part of the function that instantiates a `GestureDetector`, using the `gestureListener`. Then call `puck.setOnTouchListener` and use your `gestureDetector`.

The `deactivatePuck` function is correct, leave it.

`flingAnimationX` and `flingAnimationY` are the two animations in the X and Y direction. Recall that in Android `(0,0)` is the upper left corner.

The most difficult part of the lab is correctly implementing `makeZFlingAnimation` (where Z stands in for both X and Y). These functions will make the fling animations so they bounce correctly and detect success. Read the documentation for `FlingAnimation`, and think about all four cases (hitting each border) separately, though the logic for all of them is similar. Take a look at some of the values we compute in Rink.kt. Why do we want to know `frameMaxX`? Also note that there is no problem calling `makeXFlingAnimation` recursively, so long as you don’t infinitely recurse. Just to give you a sense of scale, I eliminated about 20 lines from my version of each function.

When the user succeeds (the puck hits the goal border), cancel the animations, make the puck disappear and call `goalAchieved`. We give you this code in `success`. 

3
Please test your code thoroughly. You will probably do this by hand, but we have given you a fragment of an espresso test in ExampleInstrumentedTest.kt, in the AndroidTest directory. We don’t have class time to get into testing with espresso, at least not yet, but if you are interested in reading on your own and writing tests, please do so.

**README file.** Just modify the one we give you. It should be in plain text and named README (not README.txt). It should be in the root directory of your submitted files. It includes these items.

1. Your name.
2. Your eid.
3. Your email address.
4. How many hours you worked on this assignment.
5. Are you using any slip days.
6. The names of anyone you spoke with intensely for this assignment.
7. Any comments for the grader.