CS 371M: Homework 1 Fling

Submission. All submissions should be done via git. Refer to the git setup and submission documents for the correct procedure. Please fill in the README file that you will find inside your Android Studio project directory (at the top level).

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, please note that in your README.

Overview. In this assignment, you will build two "games." The first, called "Jump" simply consists of a square (called the "puck") that, when touched, will teleport from the upper left corner of the game area to the upper right corner. Subsequent touches (or clicks) will move the puck to the lower right corner, the lower left corner, and back to the upper left corner. Recall that the upper left corner is the origin of our Android coordinate system.

- The game area has four borders, which remain black for the duration of the Jump game. The puck must be rendered to exactly abut the relevant borders. It should not overlap the border or be rendered any distance from the border. There is no scoring for Jump. The Jump game should remember which was the last position of the puck, because if the user plays a different game and then returns to Jump, the puck should resume its previous position.

Mode button. There is a mode button in the top right hand corner of the screen. It starts off with the label, "Jump," and when pressed toggles to "Fling." Fling is the other game you will code.

- To start a game of Fling, make sure the button indicates you are in Fling mode. Then click the button in the top middle of the screen to start. Clicking the button starts a countdown timer, and randomly places the puck within the borders. When the timer starts, one of the 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 (where two non-goal borders meet). Test every corner.

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. The XML files where you have to fill things in have XML comments.

- **MainActivity.kt** This file is complete and correct. Scoring is done here. You can change `durationMillis` to a larger value during testing to make life easier on yourself. You should also change `goalOrder` to do better testing.

  You should read this file, especially to see how the public methods in `jump` and `fling` are called.

  Any changes you make to this file will be disregarded when we test.

  This file does assume that you have added `ImageViews` to the layout with the following names: `frameT`, `frameB`, `frameS`, `frameE`. 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).

- **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.

- **Border.kt** This file is complete and correct. Lucky for you, it also contains a lot of fiddly details. Note that you have to initialize it by passing a list of ImageViews that are the borders for the puck area. You must pass these in the correct order, which is top, bottom, start, and end (you can see this in the code).

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

  Look at the public functions of this class. You will use them. Ask yourself, when and why would someone call the function, `nextGoal()`? The answer is that you will call it at the correct place in your code. You are welcome for such a wonderful hint.

- **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 the four borders and the right constraints for the puck. 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`.

- **Jump.kt** And here our journey really begins. I've only left the public methods (one with two lines of code) and one internal method definition. You can see how MainActivity initializes this object. Note, I don't need the border object as a property, so it doesn't say `private val` or just `val` before it.

  The functionality that this class implements isn't terribly complicated, so it is a good starting place. Have fun! Use a Kotlin `when` statement! Don't be afraid to call methods on the `border` object!

- **Fling.kt** This file requires the most work. You can see how MainActivity calls it. You can also see how it uses the border object to initialize some important constants.

  Ask yourself, "why is `puckMaxX` not equal to `border.maxX()`?" Where is the origin for the Android screen and Android objects? Why is `puckMinX` equal to `border.minX()`?

  Initializing `goalBorder` to `Border.Type.T` is arbitrary. We could have put any value there.

  All 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.

  Now let's skip to the end of the file. `playRound` takes a callback function that you call if the user wins (called `goalAchieved`). If time runs out, don't worry, the MainActivity will take care of cleanup. Look at the functions above, and think about the `border` object, and try to get a game started here.

  `listenPuck` sets up callback functions on the puck, so that when the user flings the puck, the Android framework will call your code. Most of `listenPuck` is written for you, but you must write the code for the fling callback. Hint: you don't need the MotionEvent objects, only the velocities.

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

  You can look at how we build a gestureDetector object from this gesture listener.

  The `deactivatePuck` function should make it so that the puck no longer responds to touch events. Hmm, there is code in `listenPuck` that makes the puck respond to touch events. I wonder how I could change that call to make it a deactivation.

  A 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. Think about the class constants we compute, like `puckMinY`. Where would that come in handy?

  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 verison of each function.

  The `success` function is called when the user succeeds (the puck hits the goal border). This function should cancel the animations, make the puck disappear and call `goalAchieved`.

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 will be in top level directory of the Android studio project. 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.