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.

Overview:

For this assignment, you will be implementing the battle mechanic of the popular strategy board game Risk.

Battles in Risk are played with 3 red dice (for the attacker) and 2 white dice (for the defender). Both players roll their dice, and depending on the outcome you decide how many soldiers each player loses.

On the next page is a screenshot of an example app.

Remember, your app doesn't have to look exactly like this. Feel free to use a different layout, a different color scheme, etc. Use the screenshot as a suggestion, not a mandate.
Specifications:

Visual Layout and Behavior:

Your app should contain the following elements:

- In the action bar of your app include a button to exit the app.
- When your app is hidden, send the user a “welcome back” toast message when it becomes visible. This message should not be seen when the user opens the app initially.
- 3 red dice and 2 white dice (for a total of 5 dice)
  - The dice should be buttons and tapping on an enabled dice should roll the dice
  - Dice that are disabled should be visibly disabled. Disabled dice shouldn't be able to be tapped.
  - Before a battle starts for the first time the dice should all be disabled
  - The player should be able to repetitively tap on the dice, allowing the battle to be fought until one player is out of soldiers.
- Text specifying the number of remaining attacking and defending soldiers
- Some sort of status bar showing the number of attacking and defending soldiers
  - The “maximum” value on both status bars should be the maximum number of soldiers on a single side, e.g. in a game that starts with 55 attacking soldiers and 49 defending soldiers both bars should have a “maximum” value of 55.
  - The status bars should be proportional to each other. If the defender starts with 20 troops and the attacker starts with 10 troops then the colored portion of the status bar for the defender should be twice as long as the attacker’s.
- A Start button
- Text fields to enter the number of soldiers at the beginning of the battle
  - Be sure to pick an appropriate text field type. The user shouldn't be able to enter letters and symbols.
- Text that alerts the user when either the attacker or the defender has won
- Text that alerts the user when troops have been lost.
  - This text should be displayed after every dice roll
  - This text should specify which player lost the troops.

Initialization

When the app initially opens, all five dice should be disabled (or hidden). They will remain disabled until a battle actually starts.

Initially, the number of attacking and defending soldiers should be zero. There should be two editable text fields, one for the attacker and one for the defender. When the user presses start, the values in these text fields should be used for the number of attacking and defending soldiers.

If at any time the user presses start (even after starting a previous game), a new game should begin using the values that have been entered by the user.
The values entered in these fields must be positive nonzero integers. The text field should be of the proper type, i.e. I shouldn't be able to type letters and special characters into it. It should be able to gracefully handle invalid numbers (such as -2 or 0).

**Rules of Engagement:**

Every time the user taps on one of the dice all dice should be rolled.

Determine how many soldiers are lost in the following manner:

1. “Pair” the highest attacking (red) die with the highest defending (white) die. The highest die wins. The loser loses 1 soldier.
2. “Pair” the second highest attacking (red) die with the second highest defending (white) die. The highest die wins. The loser loses 1 soldier.
3. The defender (white) always wins in a tie.

In this manner, in each round of combat it is possible for either side to lose 2 soldiers, or for both sides to lose 1 soldier each.

Every time a soldier is lost the user should be alerted in some fashion.

Combat ends when one side runs out of soldiers. Be sure to alert the user when the game has ended.

**Examples:**

\[
\begin{array}{ccc}
\end{array}
\]
\[
\begin{array}{ccc}
\end{array}
\]

pair [5][5] and [4][2], white wins both, red (attacker) loses 2 soldiers

\[
\begin{array}{ccc}
\end{array}
\]
\[
\begin{array}{ccc}
\end{array}
\]

pair [4][5] and [1][2], red wins both, white (defender) loses 2 soldiers

\[
\begin{array}{ccc}
\end{array}
\]
\[
\begin{array}{ccc}
\end{array}
\]

pair [6][6] and [5][6], both the attacker and the defender lose 1 soldier

**Additional Rule:**

Players aren't allowed to roll more dice than they have soldiers. For example, if the attacker only has 2 troops remaining he isn't allowed to roll 3 dice, instead he may only roll 2 dice.

This holds true at all points in time during the battle. If a player starts with many soldiers but that number is reduced, the number of dice must also be reduced accordingly.

**Video**

If this is unclear, try watching the video below. It explains everything decently, but the guy in
the video uses the wrong color dice!! For our project (and in games of Risk between civilized
human beings) the attacker uses red dice and the defender uses white dice.

https://www.youtube.com/watch?v=2QbsRsSDMOE

Target Device:
Your app will be evaluated on the Pixel 2 API level 24 device emulator provided by Android
Studio.

Code Structure:

You will implement three dice classes.

1. An abstract class named Die. The class Die should inherit from ImageButton.
2. A class WhiteDie that inherits from Die.
3. A class RedDie that inherits from Die.

Use good programming standards here. You will lose points if there is unnecessary code
duplicated in WhiteDie and RedDie that should have been placed in the parent class.

All the dice rendered on the screen should be one of these classes.

Dice with the same color belong to the same group. When you click on one die, the other die
with the same color is rolling at the same time. You don’t want the die aware of the existence
of the other die. How to represent this composition relationship in code?

You can create a DiceGroup class that contains dice with the same color. Since the
DiceGroup knows all the die it can call other dice’s roll method when one die is clicked. But
how does it know when a die is clicked? You can pass this information using an interface! You
can create an interface that is implemented by DiceGroup. An interface looks like a class
declaration, but it only has methods, no data and no implementation. A class needs to
implement the interface which it declares with the java keyword implements. The class that
implements the interface can call other dice roll methods according to its own logic.

You will find this pattern handy when you put two dice groups into the Risk game.

README file:

1. Your README should be in plaintext.
2. Your README should be named “README”, not “README.txt”, not “ReadMePlease”,
etc.
3. Your README must be located in the root directory of your submitted files.
4. The following elements should be included inside your README:
   1. Your name
   2. Your EID
   3. Your email
4. How much time this project took you to complete
5. If you are using any slip days (if you are turning in the assignment late)
6. Any comments to the grader

Hints:

Some dice faces have been provided for you here. Feel free to use these or to find/create your own.

If you are confused by the rules of Risk come and see me ASAP.

One good method to display messages to the user is via toasts.

I will try to break your app by pressing buttons in orders you did not intend. Prepare for this.