EECS1022 (M,N,O) Winter 2021
Lab8 (Mobile Computing Part 2)
Building a Tic-Tac-Toe Game App using Android Studio

Chen-Wei Wang

Release Date: Friday, March 26
Due Date: 23:59 EST, Friday, April 9

- The format of this lab (to be completed in Android Studio) is the same as Lab7. Read the instructions carefully.
- You are required to **work on your own** for this lab. **No** group partners are allowed.

  Plagiarism checks will be run on all submissions, and suspiciously similar ones will be reported to Lassonde.
- To complete this lab, it is **strictly forbidden** for you to use any library class (e.g., `ArrayList`, `Arrays`). Violating this requirement will cause a **50% penalty** on your lab marks.
- Class(es) and method(s) derived from the given JUnit class **must** be added to the `model` package. You must **not** create any classes **not** indicated by the JUnit tests, otherwise your submitted model classes will **not** compile with the grading program.
- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- Your lab assignment is **not** graded during the weekly scheduled lab sessions.
  - Follow the instructions to submit (via eClass) the required file(s) for grading.
  - Emailing your solutions to the instructor or TAs will **not** be acceptable.
- Texts in blue are hyperlinks to the corresponding documents/recordings.

Policies

- **Your (submitted or un-submitted) solution to this lab exercise (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.**
- When you submit your lab, you claim that it is **solely** your work. Therefore, it is considered as **a violation of academic integrity** if you copy or share **any** parts of your Java code during **any** stages of your development.
- When assessing your submission, the instructor and TA may examine your code, and suspicious submissions will be reported to the department if necessary. **We do not tolerate academic dishonesty**, so please obey this policy strictly.
- You are entirely responsible for making your submission to the TA in time. Back up your work **periodically**, so as to minimize the damage should any sort of computer failures occur. Follow this tutorial series on setting up a **private** Github repository for your Java projects.
- The deadline is **strict** with no excuses. Refer to the course syllabus for the policy on a late submission.

Contents

Learning Outcomes

By completing the assigned exercises of this lab, you are expected to be able to:

1. Apply the Model-View-Controller (MVC) Pattern.
2. Design and Implement a GUI for a TicTacToe App.
3. Practice Event-Driven Programming:
   - Spinner (Drop-Down Menu List)
   - Text View
   - Edit Text Field
   - Button
4. Practice the pattern of controller: multiple buttons that access and modify a single, shared object (i.e., `TicTacToe` object).
5. Attach Controller Methods to GUI Components
6. Implement model classes:
   - attributes in a model class that are of array-referenced types (one-dimensional and/or two-dimensional)
   - local variable declarations
   - variable assignments
   - object creations
   - method calls using the dot notation

Assumptions

- You have already setup a Github account and stored work in a **private** repository: `EECS1022-W21-workspace`.

  Note. You only submit your lab through eClass, not Github. Though not required, it is highly recommended that you adapt to the practice of backing your work using a versioning tool like Github.

- You are able to use Android Studio to complete the tutorial videos and this lab assignment on either your own machine or the EECS remote labs.

Requirements of this Lab

- For this lab, you will be graded not only by JUnit tests given to you, **but also additional tests** covering some other input values. This is to encourage you to take more responsibility for the correctness of your code, by writing your own tests.
- **Programming Requirements**
  - You are **only allowed** to use (one-dimensional and/or two-dimensional) primitive arrays (e.g., `int[]`, `String[]`, `Facility[]`, `char[][]`) for declaring attributes and implementing methods.
  - Any use of a Java library class or method is forbidden (that is, use selections and loops to build your solution from scratch instead):
    - Here are some examples of **forbidden** classes/methods: `Arrays` class (e.g., `Arrays.copyOf`), `System` class (e.g., `System.arrayCopy`), `ArrayList` class, `String` class (e.g., `substring`).
    - The use of some library classes does not require an `import` statement, but these classes are **also forbidden** to be used.
    - Here are the exceptions (library methods which you **are allowed** to use if needed):
      - `String` class (`equals`, `format`)

  Violating this requirement will cause a **50% penalty** on your lab marks.
- Your lab submission will **only** be graded using JUnit (for this lab, the tests supplied to you plus some additional tests).
- For the JUnit test class `Tests.java` given to you:
  - Do **not** modify the test methods given to you.
  - You are allowed to add new nest methods.
- For each method which you are required to implement, derived from the JUnit test methods:
  - No `System.out.println` statements should appear in it.
  - No Scanner operations (e.g., `input.nextInt()`) should appear in it.

  Instead, declare input parameters of the method as indicated by the JUnit tests.
- You are welcome to ask questions related to this lab or the assigned tutorial videos on the forum. However, please be **cautious**:
  - You can help your fellow students understand the requirements of tasks.
  - **Do not share the code you developed to ask, or to answer, questions.**
    - Questions specific to the code you write would be best and most effectively addressed by TAs (during scheduled labs) or your instructor (during office hours or appointments).
    - Week 9's tutorial videos shows to you **debugger in Android Studio** (which should be used in the same manner as Eclipse debugger). You are encouraged to **set breakpoints and launch the debugger** when you are stuck at your own program.
  - **Hints** on how the solution should look like are **left only to the instructors** who moderate the forum.

1 Task 1: Complete Weekly Java Tutorial Videos

- For Lab8, you are assigned to study **Weeks 9 – 11** of the Java tutorial series:

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_64z4oGXy0kHBO7_gPkjNmG` [Week 9 (Android Studio)]

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_4NaazeNVRESY1x1St3QeHc` [Week 10 (2D Array: I)]

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_45YrYlSwHmg34Ctf6Akrx2` [Week 11 (2D Array: II)]

  To reference tutorial videos from the previous weeks, see:

  `https://www.youtube.com/playlist?list=PL5dxAmCmjv_6wy2m0yq2wObIWPz4tAxW6` [All Weeks]

  These Java tutorial videos assigned to you are meant for you to:

  1. Obtain extra hands-on programming experience on Java, supplementing your weekly lectures.
  2. Complete the lab assignment with the necessary skills and background.

  Though we do **not** require the submission of the weekly Java tutorial project (like in Lab0), **examples and insights discussed in these tutorials will be covered in your (written and programming) tests and exam**: should you decide to skip the weekly tutorial videos, it would be your choice.

  As you study through the example Java classes in the tutorial videos, you are advised to **type them out** (but absolutely feel free to add new Java classes to experiment) on Android Studio.

- You can find the iPad notes of illustrations from the tutorial videos here:

  `https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/EECS1022%20Tutorial%20on%20Java.pdf`

Notes from Lab5 and Lab6

- See this notes on how to manipulate objects with reference-typed, multi-valued attributes:

  `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Tracing_PointCollectorTester.pdf`

- See this notes on how to infer classes and methods from given JUnit tests:

  `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit.pdf`

- You can find here the example covered in the notes for practice:

  - Starter: `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit.zip`
  - Solution: `https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Inferring_Classes_from_JUnit_Solution.zip`

2 Task 2: Setup an Android Studio Project

Starting Task 2 should mean that you have already completed the weekly Java tutorial videos (Section 1).

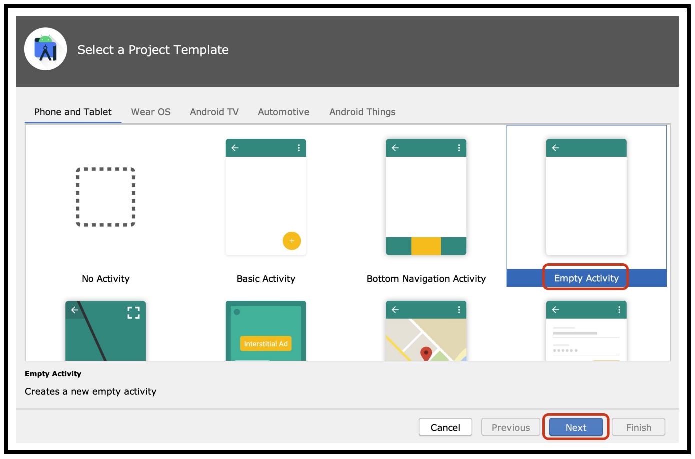
2.1 Step 1: Install Android Studio

See the detailed PDF instructions "**Install Android Studio 3.6.3 File**" on the M,N,O eClass site: `https://eclass.yorku.ca/eclass/pluginfile.php/2004743/mod_resource/content/2/Installing_Android_Studio_v3.6.3.pdf`.

2.2 Step 2: Set up an Empty Activity in Android Studio

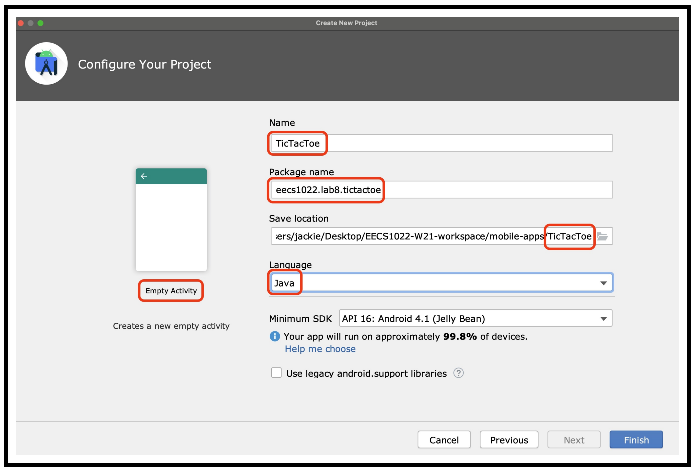
Unlike previous labs, there is no project archive file for you to import. Instead, you are asked to create an **Empty Activity** in Android Studio:

1. Launch Android Studio and click on `Create New Project`.
2. Under `Select a Project Template`, select `Empty Activity`.



Empty Activity
Creates a new empty activity

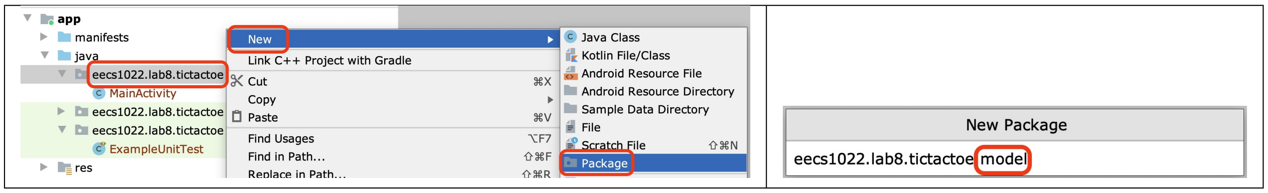
3. Under `Configure Your Project`:
   - Name: type `TicTacToe`
   - Package Name: type `eecs1022.lab8.tictactoe`
   - Save Location: make sure the last part of the path should match the app name `TicTacToe`
   - Language: choose `Java`



Name: TicTacToe
Package name: eecs1022.lab8.tictactoe
Save location: ːers/jackie/Desktop/EECS1022-W21-workspace/mobile-apps/TicTacToe
Language: Java
Minimum SDK: API 16: Android 4.1 (Jelly Bean)
Your app will run on approximately **99.8%** of devices.
☐ Use legacy android.support libraries

4. After the new project is successfully built:

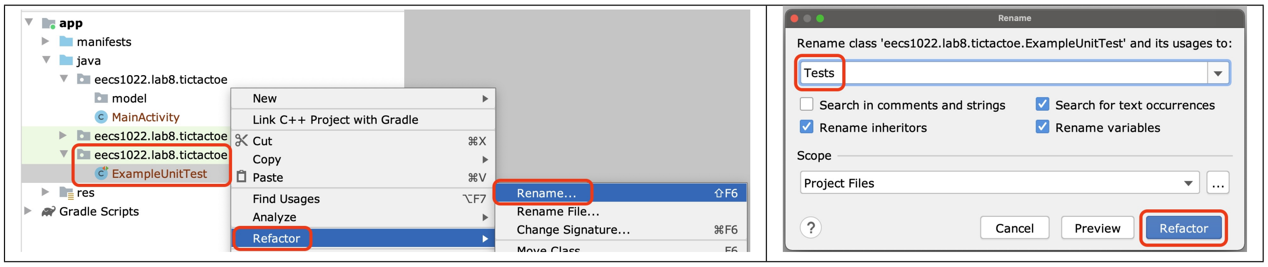
   4.1 Under `app/java/eecs1022.lab8.tictactoe`, create a new package called `model`.



New Package

eecs1022.lab8.tictactoe.model

   4.2 Under `app/java/eecs1022.lab8.tictactoe (test)`, rename the JUnit test class `ExampleUnitTest` to `Tests`.



Rename class 'eecs1022.lab8.tictactoe.ExampleUnitTest' and its usages to:

Tests

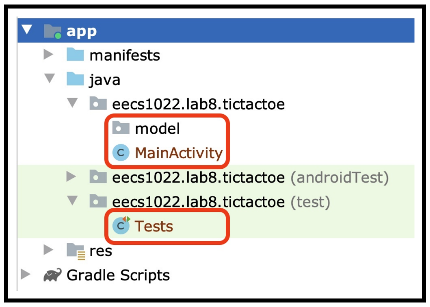
☐ Search in comments and strings ☑ Search for text occurrences

☑ Rename inheritors ☑ Rename variables

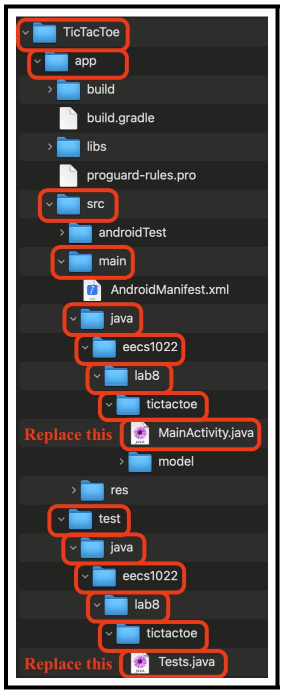
Scope

Project Files

   4.3 After performing the above steps, your project structure in Android Studio should look like:



2.3 Step 3: Download and Replace the Controller Class and JUnit Test Class



1. From eClass, download the archive file `EECS1022_W21_Lab8.zip`, containing:
   - The controller class: `MainActivity.java`
   - The JUnit test class: `Tests.java`
2. As shown in the left figure, do the following replacements:
   - Copy and replace the given `MainActivity.java` to your project folder: `TicTacToe/app/src/main/java/eecs1022/lab8/tictactoe`
   - Copy and replace the given `Tests.java` to your project folder: `TicTacToe/app/src/test/java/eecs1022/lab8/tictactoe`

     It is expected that **compilation errors** exist in both given files when first replaced into your project. See Section 3.2.1.

3 Task 3: Develop a Mobile App in Android Studio

3.1 Step 1: Study the TicTacToe Problem

You are required to develop an object-oriented program solving the problem of a tic-tac-toe game. Tic-tac-toe is a paper-and-pencil game for two players, `x` and `o`, who take turns marking the spaces in a 3×3 grid. The game may be started either by the `x` player or by the `o` player. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal line wins the game. For example, Figure 1 shows a tic-tac-toe game where player `o` wins through a diagonal line. However, it is not necessarily the case that every game has a winner (e.g., see Figure 2): when there is not a single free slot in the grid, and when neither player has managed to mark a winning line, then it is called a *tie*. That is, a game is considered as *over* either when there is a winner (e.g., Figure 1), or when there is a tie (e.g., Figure 2).

Note. Your app only needs to determine if there is a tie when all 9 slots on the board have been marked/occupied. That is, your app **need not** declare a tie when it is already known that no possible subsequent moves can result in a winner.

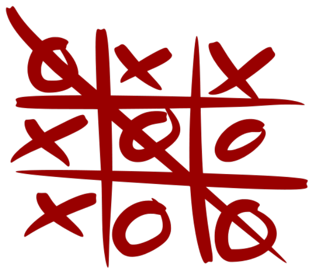


Figure 1: Tic-Tac-Toe: The `o` Player Wins

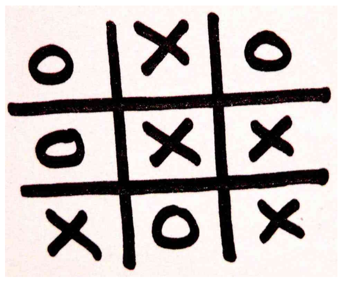
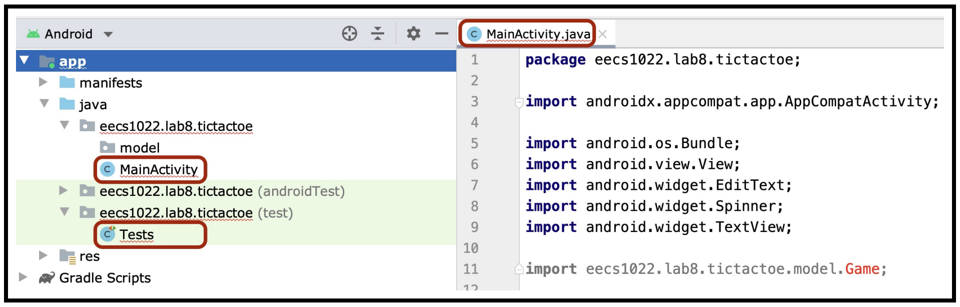


Figure 2: Tic-Tac-Toe: A Tie

3.2 Step 2: Using JUnit Tests to Develop Classes in Model Package

3.2.1 Inferring Model Classes from JUnit Tests

- It is expected that the `Tests` JUnit class (which you downloaded and copied from eClass) contains **compilation errors**. This is because that declarations and definitions of the required class(es) and method(s) it references are missing.



```
package eecs1022.lab8.tictactoe;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

import eecs1022.lab8.tictactoe.model.Game;
```

- The `model` package you created is empty. Class(es) and method(s) derived from the given JUnit class **must** be added to this package. You must **not** create any classes **not** indicated by the JUnit tests, otherwise your submitted model classes will **not** compile with the grading program.

  Therefore, your tasks are:

  1. Inferring from the given JUnit tests, add the missing class(es) and method(s) into the `model` package. Note a line from the `Tests` class (which specifies that all classes from the `model` package are referenced):

     ```
     import eecs1022.lab8.tictactoe.model.*;
     ```

  2. Pass **all** JUnit tests given to you (i.e., a **green bar**).

You must not modify these given JUnit tests, as they suggest how the intended class(es) and method(s) should be declared.

How to Deal with a Failed JUnit Test? Like the previous labs, in Android Studio, place breakpoints around assertions of the failed tests, launch the debugger, and then use the "step over", "step into", "step out", `Variables`, and `Watches` tools to help fix your code. See **Java Tutorials Week 9** for how to use all these.

3.2.2 Status of TicTacToe

A tic-tac-toe game's *initial* status reports the name of player due to move. Each subsequent status of the game is resulted from the **last** attempt to move (i.e., by entering a row number and a column number):

- When there is an error upon moving with the input row and/or column numbers, the game's status should be the corresponding error message.
- Otherwise, when there is no error, the game's status should indicate either the game is over (due to a winner or a tie being determined) or the name of player due to move (see the given JUnit tests).

When multiple errors occur simultaneously, exactly one message for the error with the higher priority (i.e., lower priority number) is displayed. For example, say the game is already over due to a tie, and the denoted (r, c) slot is already occupied, we only output the error message `Error: game is already over with a tie.`

Inputs: Row number r and column number c (which should be 1, 2, or 3 to be valid).

Rank	Error Condition	Error Message
1	There is already a winner.	`Error: game is already over with a winner.`
1	There is already a tie.	`Error: game is already over with a tie.`
2	The row number r is invalid.	`Error: row` r `is invalid.`
3	The column number c is invalid.	`Error: col` c `is invalid.`
4	Slot at position (r, c) is already marked as `x` or `o`.	`Error: slot @ (`r`,` c`) already occupied.`

In the above table, the two errors with the same priority **cannot** occur simultaneously.

3.2.3 Hints and Requirements

- See this notes on how to declare and manipulate reference-typed, multi-valued attributes:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_W21_Tracing_PointCollectorTester.pdf

- See this notes on how to infer classes and methods from given JUnit tests:

  https://www.eecs.yorku.ca/~jackie/teaching/lectures/2021/W/EECS1022/notes/EECS1022_Inferring_Classes_and_Methods_from_JUnit_Tests.pdf

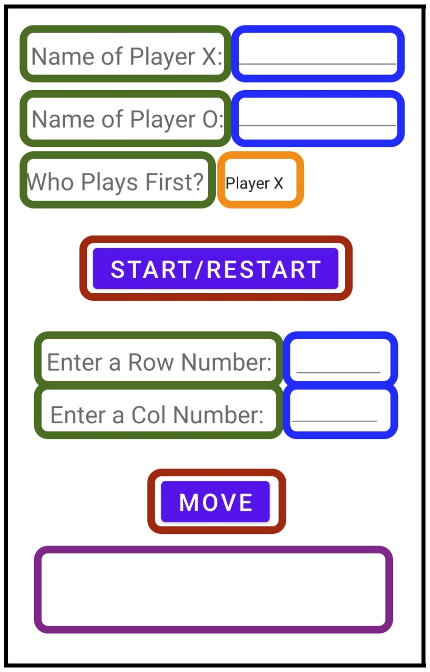
  Programming IDEs such as Eclipse and Android Studio are able to fix such compilation errors for you. **However, you are advised to follow the guidance as specified in the notes to fix these compilation errors manually, because: 1) it helps you better understand how the intended classes and methods work together; and 2) you may be tested in a written test or exam without the assistance of IDEs.**

- Any new class(es) you add must reside in the `model` package.
  - Once the necessary class(es) and method(s) are declared, you can add as many attributes as necessary to implement the body of each method.
  - Study carefully example instances as specified in `Tests.java`: they suggest the how the intended class(es) and method(s) should be declared and implemented.
  - Focus on *gradually* passing one test at a time.

3.3 Step 3: Design a GUI Layout

3.3.1 Given GUI Layout to Implement

You are then asked to implement the following GUI design:



In the left GUI design:

- Textviews (for displaying input prompts) are boxed in **green**.
- Plain Texts (or textfields for reading user-typed inputs) are boxed in **blue**.
- The spinner (for displaying a drop-down menu of options) is boxed in **orange**.
- Buttons (for invoking attached controller methods) are boxed in **red**.
- The (invisible) textview (for displaying computation results) is boxed in **purple**.

To implement the above GUI design, open `activity_main.xml` (under `app/src/main/res/layout`) in Android Studio.

Then, drag, drop, and organize the above specified GUI components. For each GUI component, be sure to set the appropriate `text`, as suggested above, and set its `id` to be used later in the controller (`MainActivity.java`).

3.3.2 Assumed Usage Pattern of the App

For the simplicity of this lab, you can assume that users (or testers) of your app behave in the following way:

1. Start/Restart

   1.1 Enter the name of player `x`.

   1.2 Enter the name of player `o`.

   1.3 Click on the button "START/RESTART"

   This step should cause your program to (re-)create a `Game` object. The bottom of the mobile screen must display the board and status (e.g., next player to make a move) of the current game.

   You can also assume that the very first button clicked by the user is "START/RESTART", whereas each subsequent click on "MOVE" is relative to the game started by the last-clicked "START/RESTART".

2. Move

   2.1 Enter a row number

   2.2 Enter a column number

   You can assume that the user's inputs can be converted into integers, but they may not denote valid rows/columns (in which case the game status should report such errors).

   From an external user's point of view, valid row/column numbers are 1, 2, or 3. On the other hand, the corresponding valid row/column indices (which only you as a programmer need worry about) are, respectively, 0, 1, and 2.

   2.3 Click on the button "MOVE"

   This step should cause your program to, if no errors, make the necessary changes to the board (e.g., mark `x` or `o` on the corresponding slot) and display an updated status of the game.

   See Section 3.2.2 for the possible error messages to be displayed.

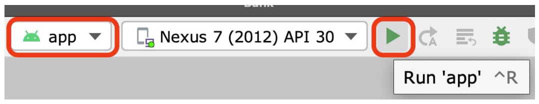
3. Repeat Step 2 until the game is over (i.e., a winner is determined or there is a tie).
4. Repeat from Step 1 if desired to start a new game.

3.4 Step 4: Develop the Controller Class

From Section 2.3, you already copied and replaced the given controller class `MainActivity`.

- Follow through the comments/hints in the given controller file to complete it.
- Refer to this short demo tutorial video on how your Android mobile app is supposed to work: https://www.youtube.com/watch?v=bSi5n6WwXsc&list=PL5dxAmCmjv_7YgI2LgcwjWTHiNZSR-aQX&index=3

  Tests shown in this demonstration are examples; you should test your app with many other inputs.
- Test the app by launching it using an emulator in Android Studio:



- The app name on the top of the emulator screen must read:

  `TicTacToe App - `*`FirstName LastName`*` (`*`StudentNumber`*`)`

  where your first name, last name, and student number are displayed in the suggested order.

  Hint: Go to the resource file `app/res/values/strings.xml` and modify the `app_name` attribute accordingly.

4 Submission

1. Before you submit, you must make sure that there are **no compilation errors** in the given JUnit test class and in all classes you have created in the `model` package.

   Submitting programs with errors (meaning that it cannot be run for grading) will result in possible partial, but low, marks.
2. Go to the eClass site for Sections M,N,O: https://eclass.yorku.ca/eclass/course/view.php?id=6214
3. Under the `Lab Submissions` section, click on `Lab8` to submit the following list of files:
   - All Java file(s) in the `model` package (as indicated by the given `Tests.java`):
     `TicTacToe/app/src/main/java/eecs1022/lab8/tictactoe/model`
   - The completed controller file: `TicTacToe/app/src/main/java/eecs1022/lab8/tictactoe/MainActivity.java`
   - The completed GUI layout file: `TicTacToe/app/src/main/res/layout/activity_main.xml`
   - The strings resource file: `TicTacToe/app/src/main/res/values/strings.xml`

   Note:
   - You may **upload** as many draft versions as you like before the deadline.
   - You must explicitly **submit** the draft version for grading before the deadline.
   - **Once you click on the `submit` button, you can no longer upload another draft version.**