Department of Electronic Engineering

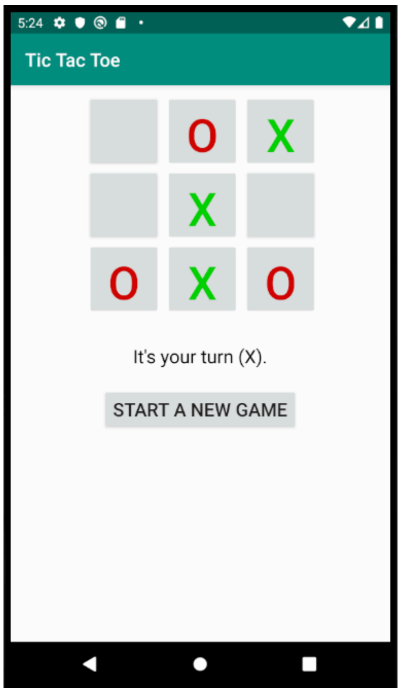
City University of Hong Kong

EE5415 Mobile Applications Design and Development

Tic Tac Toe Game Individual Project

I. Introduction

The aim of this lab is to create an Android **Tic Tac Toe** game as shown below.



Students will use buttons to represent the game board and a TextView to display the status of the game at the bottom of the screen. The buttons will have no text on them when the game starts, but when the user clicks on a button, it will display a green X. The app will them move and turn the appropriate button text to a red O. The TextView will indicate whose turn it is and when the game is over.

II. Game Design

In this lab, students will experience how to make use of a fully functional "Tic Tac Toe" game that runs in a console as game logic class for building an Android game app. Student may need to install JDK into your computer in order to run this demo game in the terminal. The JDK can be downloaded from

- http://java.sun.com/javase/downloads/index.jsp.

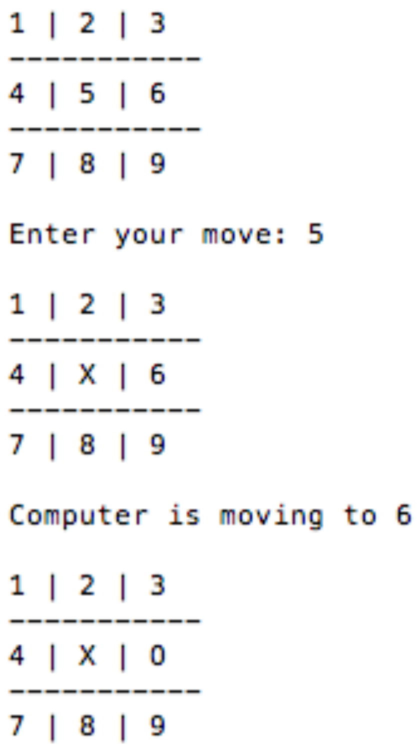
The Java source code of the console Tic Tac Toe game is available in the following link:

- http://cs.harding.edu/fmccown/android/TicTacToeConsole.java

Download this java program and use the following commands to compile and run it on your computer as console program:

```
javac TicTacToeConsole.java

java  TicTacToeConsole
```



```
1 | 2 | 3
-----------
4 | 5 | 6
-----------
7 | 8 | 9

Enter your move: 5

1 | 2 | 3
-----------
4 | X | 6
-----------
7 | 8 | 9

Computer is moving to 6

1 | 2 | 3
-----------
4 | X | O
-----------
7 | 8 | 9
```

In addition, students should take a look at the source code to familiarize how this program works. It uses a char array to represent the game board with X representing the human and O representing the computer. The program implements a simple AI for the computer: it will win the game if possible, block the human from winning, or make a random move. The game has all the logic to switch between the two players and to check for a winner.

The aim of this lab is to convert this console game into an Android game. We have two basic choices when porting this game to Android: we could merge the game logic with the user interface (UI) code, or **we could keep the two separated as much as possible**. The second approach is preferred since it allows us to make changes to the UI, like changing the layout of the board, without having to modify code that just deals with game logic. Also if we decide to port this game to another platform like the iPhone and MS-Windows 8, having the UI and game logic cleanly separated will minimize the amount of code needing to be modified and make the port much easier to perform.

III. Implementation Android Tic Tac Toe Game

The aim then is to place all the UI logic in the **`MainActivity.java`** and all the game logic in **`TicTacToeGame.java`**.

1. Create a new project in Android Studio with application name of "**Tic Tac Toe**".
2. Create a **`TicTacToeGame.java`** class file to this project by clicking on File -> New -> Class. Give it the name **TicTacToeGame** and click Finish. We will now need to copy and paste the tic-tac-toe code from the `TicTacToeConsole.java` file into `TicTacToeGame.java`.
3. There are some modifications students will need to make to the `TicTacToeGame` class in order to expose the game logic to the `MainActivity` activity. First, it's no longer necessary that we place the numbers 1-9 into the board character array to represent free positions, so let's **create a constant called `OPEN_SPOT` that uses a space character to represent a free location on the game board**. We'll continue to use X and O to represent the human and computer players. Add the following code into the `TicTacToeGame.jave` file.

```
public class TicTacToeGame {

    // Characters used to represent the human, computer, and open spots
    public static final char HUMAN_PLAYER = 'X';
    public static final char COMPUTER_PLAYER = 'O';
    public static final char OPEN_SPOT = ' ';
    public static final int BOARD_SIZE = 9;
    private char mBoard[] = { '1', '2', '3', '4', '5', '6', '7', '8', '9' };

    private Random mRand;

:
```

4. Students also need **to remove the `TicTacToeConsole()` , `displayBoard()`, and `main()` methods of TicTacToeConsole.java code.** The `main()` is at the end of the code. It is because we don't need these codes for a model class.

5. Students will need to **create several public methods** that can be used to manipulate the game board and determine if there is a winner. Below is a listing of the public methods **`TicTacToeModel()`**, **`clearBoard()`**, **`setMove()`**, and **`checkForWiner()`** , students are required to craft the existing code into. **All other methods should be private** since we do not want to expose them outside the class.

```
/** The constructor of the TicTacToeGame have to be remove some code as follows */
public TicTacToeGame() {

    // Seed the random number generator
    mRand = new Random();
```

```java
    char turn = HUMAN_PLAYER; // Human starts first
    int win = 0; // Set to 1, 2, or 3 when game is over

}


/** Clear the board of all X's and O's. */
public void clearBoard() {
    for (int i = 0; i < BOARD_SIZE; i++) {
        mBoard[i] = OPEN_SPOT;
    }
}


/** Set the given player at the given location on the game board * */
public void setMove(char player, int location) {
    mBoard[location] = player;
}


/**
 * Check for a winner and return a status value indicating who has won.
 * @return Return 0 if no winner or tie yet, 1 if it's a tie, 2 if X won,
 * or 3 if O won.
 */
public int checkForWinner() {

       :
       :

}
```

6. We also need to replace the original `getComputerMove()` as a **public** method and this method has a return `move` valuable.

```java
/** Return the best move for the computer to make. You must call setMove()
 * to actually make the computer move to that location.
 * @return The best move for the computer to make (0-8).
 */

public int getComputerMove()
{
    int move;
    // First see if there's a move O can make to win
    for (int i = 0; i < BOARD_SIZE; i++) {
        if (mBoard[i] != HUMAN_PLAYER && mBoard[i] != COMPUTER_PLAYER) {
            char curr = mBoard[i];
            mBoard[i] = COMPUTER_PLAYER;
            if (checkForWinner() == 3) {
                System.out.println("Computer is moving to " + (i + 1));
                return i;
            }
            else
                mBoard[i] = curr;
        }
    }
    // See if there's a move O can make to block X from winning
    for (int i = 0; i < BOARD_SIZE; i++) {
        if (mBoard[i] != HUMAN_PLAYER && mBoard[i] != COMPUTER_PLAYER) {
            char curr = mBoard[i];   // Save the current number
            mBoard[i] = HUMAN_PLAYER;
            if (checkForWinner() == 2) {
                mBoard[i] = COMPUTER_PLAYER;
```

```
                    System.out.println("Computer is moving to " + (i + 1));
                    return i;
                }
                else
                    mBoard[i] = curr;
            }
        }
        // Generate random move
        do
        {
            move = mRand.nextInt(BOARD_SIZE);
        } while (mBoard[move] == HUMAN_PLAYER || mBoard[move] == COMPUTER_PLAYER);

        mBoard[move] = COMPUTER_PLAYER;
        return move;

    }
```

7. For the game logic to be accessible to the Activity, create a class-level variable for the TicTacToeGame class in `MainActivity.java`:

```
public class MainActivity extends AppCompatActivity {

    // Represents the internal state of the game
    private TicTacToeGame mGame;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

This object will be instantiated in the **`onCreate()`** method. The Activity can now use **`mGame`** to start a new game, set moves, and check for a winner. Note that Android programming conventions use a lowercase “m” at the beginning of all member variables to distinguish themselves from local parameters and variables; we’ll use the same naming conventions throughout this lab.

III. Creating a Game Board

Students are recommended to represent the “Tic Tac Toe” game board using Button widgets. Based on the best-practices of Andriod, the game board will be created in the **`activity_main.xml`** layout file under the res/layout folder.

1. Use the following xml code to create the UI of the “Tic Tac Toe” game UI:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <TableLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" >

        <TableRow
            android:id="@+id/tableRow1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <Button
                android:id="@+id/button0"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:gravity="center"
                android:text="1"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button1"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="2"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button2"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_gravity="center_vertical"
                android:layout_marginLeft="5dp"
                android:text="3"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />
        </TableRow>

        <TableRow
            android:id="@+id/tableRow2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >
            <Button
                android:id="@+id/button3"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="4"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button4"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="5"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button5"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="6"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />
        </TableRow>

        <TableRow
            android:id="@+id/tableRow3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <Button
                android:id="@+id/button6"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="7"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button7"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="8"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />

            <Button
                android:id="@+id/button8"
                android:layout_width="80dp"
                android:layout_height="80dp"
                android:layout_marginLeft="5dp"
                android:text="9"
                android:onClick="onButtonClicked"
                android:textSize="50sp" />
        </TableRow>
    </TableLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:orientation="vertical"
        android:padding="10dp" >

        <TextView
            android:id="@+id/information"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:gravity="center_horizontal"
            android:text="info"
            android:textSize="20sp" >
        </TextView>

        <Button
            android:id="@+id/button_restart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dp"
            android:text="Start a New Game"
            android:textSize="20sp"
            android:onClick="newGame" >
        </Button>

    </LinearLayout>

</LinearLayout>
```

In this layout, a TableLayout is used to create the game board under a LinearLayout with vertical orientation for arranging a TextView for display the game statue and a Button for starting a new game.

2. To connect the buttons and text widgets defined in activity_main.xml with the MainActivity class, students are required to declare a Button array and TextView member variables in the **`MainActivity.java`** file as follows:

```
// Buttons making up the board
private Button mBoardButtons[];

// Various text displayed
private TextView mInfoTextView;

// Restart Button
private Button startButton;
// Game Over
Boolean mGameOver;
```

3. In the onCreate() method, instantiate the array and use the findViewById() method to attach each button in activity_main.xml with a slot in the array. Note that the ID's are found in R.java and match precisely the name assigned to each button in main.xml. You'll also need to instantiate mGame so the Activity will be able to access the "Tic Tac Toe" game logic.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    mGame = new TicTacToeGame();

    mBoardButtons = new Button[mGame.BOARD_SIZE];
    mBoardButtons[0] = (Button) findViewById(R.id.button0);
    mBoardButtons[1] = (Button) findViewById(R.id.button1);
    mBoardButtons[2] = (Button) findViewById(R.id.button2);
    mBoardButtons[3] = (Button) findViewById(R.id.button3);
    mBoardButtons[4] = (Button) findViewById(R.id.button4);
    mBoardButtons[5] = (Button) findViewById(R.id.button5);
    mBoardButtons[6] = (Button) findViewById(R.id.button6);
    mBoardButtons[7] = (Button) findViewById(R.id.button7);
    mBoardButtons[8] = (Button) findViewById(R.id.button8);
    mInfoTextView = (TextView) findViewById(R.id.information);

    mGame = new TicTacToeGame();
    startNewGame();

}
```

IV. Adding Game Logic

In order to make the user to play with the app, we have to add some logic to start a new game.

1. To start a new game, we need to clear the game board of any X's and O's from the previous game by calling `mGame.clearBoard()` method. This only clears the internal representation of the game board, not the game board we are seeing on the screen. In addition, we also need to clear the visible game board. Thus we need to loop through all the buttons representing the board and set their text to an empty string. We also need to enable each button (we will disable them later when a move is placed), and we to create an event listener for each button by setting each button's `OnClickListener` to a new `ButtonClickListener`, a class which we will define shortly. We pass to ButtonClickListener's constructor the button's number (0-8) which will be used to identify which button was actually clicked on. Add the code below to the startNewGame() function.

```
//--- Set up the game board.
private void startNewGame() {
    mGameOver = false;
    mGame.clearBoard();
    //---Reset all buttons
    for (int i = 0; i < mBoardButtons.length; i++) {
        mBoardButtons[i].setText("");
        mBoardButtons[i].setEnabled(true);
        mBoardButtons[i].setOnClickListener(new ButtonClickListener(i));
    }
    //---Human goes first
    mInfoTextView.setText("You go first.");
}
```

Finally, we will indicate that the human is to go first. Note that the "You go first." text should normally not be hard-coded into the Java source code for a number of reasons, but fix this later. Add these lines to the bottom of `startNewGame()` .

2. We need to call `startNewGame()` when the App first loads, so add this call on the last line of `onCreate()` :

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

        :
        :


    startNewGame();

}
```

3. Now let's add the `ButtonClickListener` that was used earlier when resetting the buttons. It needs to implement the `android.view.View.OnClickListener` interface which means it needs to implement an `onClick()` method. We also need to create a constructor that takes an integer parameter, the button number, since we need to know which button was clicked on. Without this information, we won't be able to easily determine which button was selected and where the X should be placed.
The `onClick()` method is called when the user clicks on a button. We should only allow the user to click on enabled buttons which represent valid locations the user may place an X. We will write a `setMove()` function shortly which will place the X or O at the given location and disable the button. After displaying the human's move, we need to see if the human has won or tied by calling `checkForWinner()`. If this function returns back 0, then the game isn't over, so the computer must now make its move. Once it moves, we must again check to see if the game is over and update the information text appropriately.

```
//---Handles clicks on the game board buttons
private class ButtonClickListener implements View.OnClickListener {
    int location;

    public ButtonClickListener(int location) {
        this.location = location;
    }

    @Override
    public void onClick(View v) {

        if (mGameOver == false) {

            if (mBoardButtons[location].isEnabled()) {
                setMove(TicTacToeGame.HUMAN_PLAYER, location);
                //--- If no winner yet, let the computer make a move
                int winner = mGame.checkForWinner();
                if (winner == 0) {
                    mInfoTextView.setText("It's Android's turn.");
                    int move = mGame.getComputerMove();
                    setMove(TicTacToeGame.COMPUTER_PLAYER, move);
                    winner = mGame.checkForWinner();
                }
                if (winner == 0) {
                    mInfoTextView.setTextColor(Color.rgb(0, 0, 0));
                    mInfoTextView.setText("It's your turn (X).");
                } else if (winner == 1) {
                    mInfoTextView.setTextColor(Color.rgb(0, 0, 200));
                    mInfoTextView.setText("It's a tie!");
                    mGameOver = true;
                } else if (winner == 2) {
                    mInfoTextView.setTextColor(Color.rgb(0, 200, 0));
                    mInfoTextView.setText("You won!");
                    mGameOver = true;
                } else {
                    mInfoTextView.setTextColor(Color.rgb(200, 0, 0));
                    mInfoTextView.setText("Android won!");
                    mGameOver = true;
                }
            }
        }

    }

}
```

4. Below is the `setMove()` function which is called from the `ButtonClickListener`. Place this function in the `MainActivity` class. It updates the board model, disables the button, sets the text of the button to X or O, and makes the X green and the O red.

```
private void setMove(char player, int location) {
    mGame.setMove(player, location);
    mBoardButtons[location].setEnabled(false);
    mBoardButtons[location].setText(String.valueOf(player));
    if (player == TicTacToeGame.HUMAN_PLAYER)
        mBoardButtons[location].setTextColor(Color.rgb(0, 200, 0));
    else
        mBoardButtons[location].setTextColor(Color.rgb(200, 0, 0));
}
```

5. Run the project and try to realize any problem of the app.

IV. Add a Button for Start a New Game

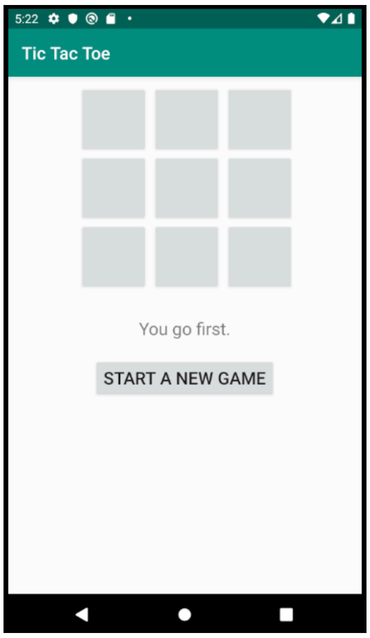
When students try to play the app in the last section, students should realize a message at the bottom of the board telling you when it's your turn, but the app moves so quickly that player are difficult to see the message indicating it's the device's turn. When the game is over, player could not be able to play another game unless to restart the app again.

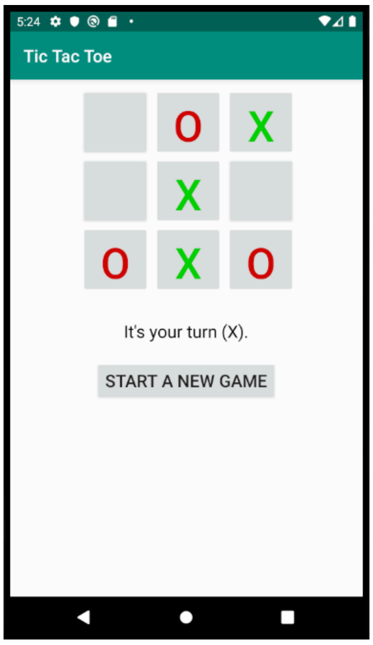
1. To fix this problem, students need to implement an `OnClickListener` method for the Restart a New Game Button of `newGame(View v)` method in the `MainActicity` class as follows:

```
//--- OnClickListener for Restart a New Game Button
public void newGame(View v) {
    startNewGame();
}
```

2. Another problem is that when the game is over, the human can continue making moves! We need to fix this problem by introducing class-level Boolean `mGameOver` variable in the `MainActivity` and set it to false in the `startNewGame()` method as shown above. When the user is clicking on a button, the variable should be checked so a move isn't made when the game is over. And `mGameOver` needs to be set to true when the game has come to an end. It's left to you to make the appropriate modifications.

3. Run the Tic Tac Toe project, and play a complete game. You should now be able to click the "Restart a New Game" button for starting a new game.





V. Marking Scheme of the Tic Tac Toe Game Individual Project

Basically, this lab is providing the basic structure for developing the course individual project with enhancing user interface and functionality design. The basic requirement of the individual project is applying the Android technology that introduced in this course to enhance the user interface. **For obtaining higher grade, students may consider adding basic rule-based AI algorithm to enhance the game.** For example, students may introduce a Level 1 that the computer just random select the position, Level 2 that the computer select the position based on the original java console program, and Level 3 that the computer select the position that it never lost.

Marking Scheme

- 30 marks for the basic functions of the Tic Tac Toe game that introduced in this lab.
- 10 marks for each of additional features or functions:
    - Making the app support multiple language with use of the strings.xml
    - Improve the game to support both portrait and landscape display mode. It is required to make the app's data persist when the orientation is changed.
    - In the basic game implementation, the human always goes first. Make the game fairer by alternating who gets to go first. Also keep track of how many games the user has won, the computer has won, and ties. You should display this information using TextView controls under the game status TextView.
    - To create options menu with two functions
        - Difficulty – to set the AI difficulty level to Easy (Level 1), Harder (Level 2), or Expert (Level 3)
        - Quit – to quit the app
    - Adding animation and sound effects to the game.
    - Total marks of this project assignment can be more than 100 with additional features.

Students are required to submit the source code of their developed Android projects in .zip format with a project report that similar to Lab02 format to describing the functionality design, software architecture, UI design and implementation detail in MS-Words (.doc) format.

All these materials are required to submit to ee5415@gmail.com account on or before **11:00pm of April 1, 2020**.

The subject of your email should satisfy the following format:

- EE5415 Individual Project – Student Name (Student Number)
- Example:

EE5415 Individual Project – Chan Chi Man (51234567)

*Note: Due to security problem, students cannot directly send a zip file to the course gmail account (ee5415@gmail.com). Students are required to rename the file extension of the zip to "zop". For example, if your compress project filename is "project.zip", you have to rename it to "project.**zop**". Then you can send this compressed project file as attachment to the gmail account.*

On-line storage of the source code submission is also accepted but it needs to keep available before the end of the semester.