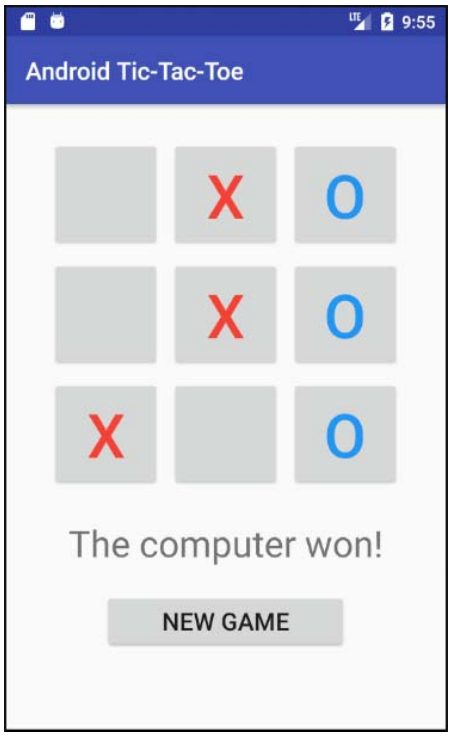
Tic-Tac-Toe
Android App Development
10 points

Create a Tic-Tac-Toe app that allows the user to play a game of tic-tac-toe against the computer. The app should use the MVC architecture.



The Model

The complete `TicTacToeGame` class below is the model. It defines a character array called `mBoard`, which represents the locations on the game board:

```
mBoard =  0  1  2
          3  4  5
          6  7  8
```

So if we want to place an X in the center of the board, then `mBoard[4]` would be set to 'X'.

Some important `TicTacTowGame` public methods:

- `newGame()` – Creates a new game by initializing mGame board
- `setPlayerMove()` – For setting the player's move
- `getComputerMove()` – For getting the computer's randomly chosen move
- `checkForWinner()` – Returns an integer indicating if there is a winner, tie game, or more moves to be made

```
package edu.harding.androidtictactoe;

import java.util.Random;


public class TicTacToeGame {

    public static final int BOARD_SIZE = 9;

    // Characters used to represent the human, computer, and open spots
    public static final char PLAYER_X = 'X';
    public static final char PLAYER_O = 'O';
    public static final char OPEN_SPOT = ' ';

    // Random number generator
    private Random mRand;

    // The game board
    private char mBoard[];

    public TicTacToeGame() {
        mBoard = new char[BOARD_SIZE];
```

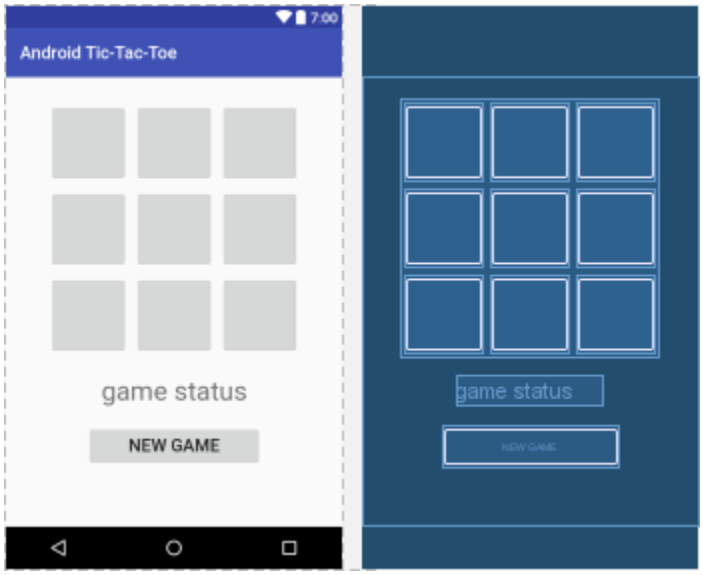
```
    mRand = new Random();
}

public boolean isGameOver() {
    return checkForWinner() != 0;
}

public void newGame() {
    // Reset all locations
    for (int i = 0; i < BOARD_SIZE; i++) {
        mBoard[i] = OPEN_SPOT;
    }
}

public boolean setPlayerMove(int move) {
    if (move >= 0 && move < BOARD_SIZE && mBoard[move] == OPEN_SPOT) {
        mBoard[move] = PLAYER_X;
        return true;
    }
    return false;
}

// 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() {
    if (checkForWinner(PLAYER_X)) {
        return 2;
    }

    if (checkForWinner(PLAYER_O)) {
        return 3;
    }

    // See if any spaces available
    if (spaceAvailable()) {
        return 0;
    }

    // Must be a tie
    return 1;
}

private boolean checkForWinner(char player) {
    // Check horizontal wins
    for (int i = 0; i <= 6; i += 3) {
        if (mBoard[i] == player && mBoard[i + 1] == player && mBoard[i + 2] == player) {
            return true;
        }
    }

    // Check vertical wins
    for (int i = 0; i <= 2; i++) {
        if (mBoard[i] == player && mBoard[i + 3] == player && mBoard[i + 6] == player) {
            return true;
        }
    }

    // Check for diagonal wins
    if ((mBoard[0] == player && mBoard[4] == player && mBoard[8] == player) ||
        (mBoard[2] == player && mBoard[4] == player && mBoard[6] == player)) {
        return true;
    }

    return false;
}

private boolean spaceAvailable() {

    // See if there is an open spot
    for (int i = 0; i < BOARD_SIZE; i++)
        if (mBoard[i] == OPEN_SPOT)
            return true;
```

```
        // All spaces must be occupied
        return false;
    }

    public int getComputerMove() {

        // Generate random move
        int tries = 0;
        int move;
        do {
            move = mRand.nextInt(BOARD_SIZE);
            tries++;
        } while (tries < 100 && mBoard[move] != OPEN_SPOT);

        if (tries == 100)
            throw new RuntimeException("getComputerMove() cannot find an unoccupied location.");

        mBoard[move] = PLAYER_O;
        return move;
    }
}
```

View

The activity_main.xml file defines the View. Use a ConstraintLayout to layout the screen. The game board should be defined in a GridLayout that centers itself in the ConstraintLayout.

```
ConstraintLayout
   GridLayout
      Button (x9)
   TextView
   Button
```



The 9 Buttons in the GridLayout should use a style called PlayButton (defined in styles.xml) that sets the buttons width and height to 90dp and the text size to 40sp. The TextView that displays the game status should be placed under the GridLayout, and a New Game button should appear under the TextView, as pictured.

Controller

Much of the Controller, the MainActivity class, is provided below. The onCreate() method initializes three class variables:

1. `mGameStatus` – The game status TextView

2. `mGameButtons` – An array of `Buttons` that reference the game board buttons

3. `mGame` – The `TicTacToeGame` object representing the Model

You are to complete the following methods for the game to work:

1. `onPlayButtonClick()` – This method is called when the user clicks on a game board button. Make the button read X, then make the computer move. Use the `isGameOver()` method to set the game status appropriately and to determine if the computer should move.

2. `getComputerMove()` – Use the appropriate `TicTacToeGame` method to get the computer's move, then make the corresponding button read O.

3. `onNewGameClick()` – This method should be called when the user clicks the New Game button. Call the appropriate method to start a new game.

```
package edu.harding.androidtictactoe;

import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.GridLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private TicTacToeGame mGame;
    private TextView mGameStatus;
    private Button[] mGameButtons;

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

        mGameStatus = (TextView) findViewById(R.id.game_status);
        mGameButtons = new Button[TicTacToeGame.BOARD_SIZE];

        GridLayout gridLayout = (GridLayout) findViewById(R.id.game_board);
        for (int i = 0; i < gridLayout.getChildCount(); i++) {
            mGameButtons[i] = (Button) gridLayout.getChildAt(i);
        }

        mGame = new TicTacToeGame();
        newGame();
    }

    private void newGame() {
        mGame.newGame();

        for (int i = 0; i < mGameButtons.length; i++) {
            mGameButtons[i].setText("");
        }

        mGameStatus.setText("Your turn");
    }

    public void onPlayButtonClick(View view) {
```

```
        Button button = (Button) view;

        // Determine which button was clicked
        int buttonNum = 0;
        for (int i = 0; i < mGameButtons.length; i++) {
            if (button == mGameButtons[i])
                buttonNum = i;
        }

        // TODO: Call the TicTacToeGame.setPlayerMove() method to update
        // the model, and change the button to X.  If the game isn't over then
        // make the computer move and update the status text to indicate it's the
        // computer's move.  After the computer moves, see if the game is over,
        // and if it isn't, update the status text to indicate it's the user's turn.
    }

    private boolean isGameOver() {
        int winnerStatus = mGame.checkForWinner();
        if (winnerStatus == 0) {
            return false;
        }
        else if (winnerStatus == 1) {
            mGameStatus.setText("Tie game!");
        }
        else if (winnerStatus == 2) {
            mGameStatus.setText("You won!");
        }
        else {
            mGameStatus.setText("The computer won!");
        }
        return true;
    }

    private void getComputerMove() {
        // TODO: Use TicTacToeGame method getComputerMove() to get the computer move
        // and change the appropriate button to O
    }

    public void onNewGameClicked(View view) {
        // TODO: Start a new game
    }
}
```

Extra Credit

1. Currently the X and O buttons are black. **1 bonus point** for programmatically changing the X and O button font color to a color resource in colors.xml.

2. Currently the computer moves too fast. **3 bonus points** for getting the computer to delay a second before making a move. Check out tronman's solution which shows how to use a Handler and Runnable to execute a method after a delay.

   https://stackoverflow.com/questions/1520887/how-to-pause-sleep-thread-or-process-in-android/3039718#3039718

Turn In

Verify that your game works correctly. The player always goes first. The player should not be able to make more moves after the game is over.

Clean your project (Build → Clean Project from the menu), then zip up your project. Then rebuild the APK file (Build → Build APK from the menu) for your project.

Submit the zip file the app\build\outputs\apk\app-debug.apk file to Easel. **Note:** The app-debug.apk file should be more than 1 MB. If it's smaller, then it's a partial APK meant for Instant Run. You must turn in the full APK file.