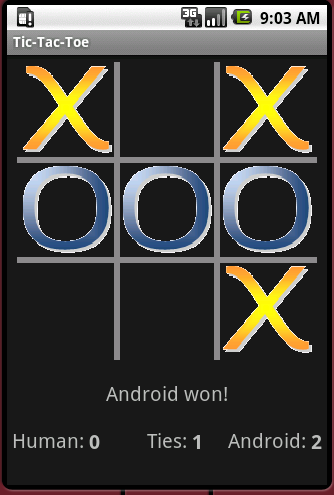
Android Application Programming

**Challenge: Graphics and Sound**

## Introduction

In the previous tic-tac-toe tutorial, your game used buttons for displaying the game board. In this tutorial, we’ll use a custom View for displaying the board which will give us more control over the board’s visual appearance. The View will draw a game board using rectangles and use bitmaps to represent X’s and O’s. We’ll also use the Android MediaPlayer to play sound effects when making moves.

## Creating a Custom View

We will first create a custom View control that will represent the game board. We’ll perform our own painting, drawing the game grid and images representing X and O.

1. Add a new Java class to your project, and name it **BoardView**. The class needs to extend android.view.View. This View will replace the buttons that are currently used to represent the game board.
2. In the BoardView class, create a constant to represent the thickness of the game board lines:

**public** **class** BoardView **extends** View {

// Width of the board grid lines

**public** **static** **final** **int** *GRID\_WIDTH* = 6;

1. Next, create two Bitmap data members which will be used to store the X and O images:

**private** Bitmap mHumanBitmap;

**private** Bitmap mComputerBitmap;

1. Create two images using your favorite graphic editor, or find two images off the Web that you’d like to use in your game. Images should be in an [Android supported format](http://developer.android.com/guide/appendix/media-formats.html): JPEG, GIF, PNG, or BMP. The size of the images should be no wider or taller than about 100 pixels. You will shrink the images down to the appropriate size when drawing them onto the View, so you don’t have to worry about the precise size of the images. Just make sure the two images have roughly the same width and height. Also make sure the name of the files follow all the rules for naming variables as the filenames will be used to create variables to access the images (no spaces, and only use lowercase alpha, numeric, and underscore characters).
2. Add the two images to your Eclipse project’s **res/drawable** folder by dragging and dropping the files onto the drawable folder. The res/drawable folder is where you previously placed the menu’s images.
3. Create an initialize() function that will load the two images. In the code below, the images are called x\_img and o\_img, so you will need to adjust the code to match your image names.

**public** **void** initialize() {

mHumanBitmap = BitmapFactory.*decodeResource*(getResources(), R.drawable.*x\_img*);

mComputerBitmap = BitmapFactory.*decodeResource*(getResources(), R.drawable.*o\_img*);

}

1. Call initialize() in the BoardView‘s three constructors:

**public** BoardView(Context context) {

**super**(context);

initialize();

}

**public** BoardView(Context context, AttributeSet attrs, **int** defStyle) {

**super**(context, attrs, defStyle);

initialize();

}

**public** BoardView(Context context, AttributeSet attrs) {

**super**(context, attrs);

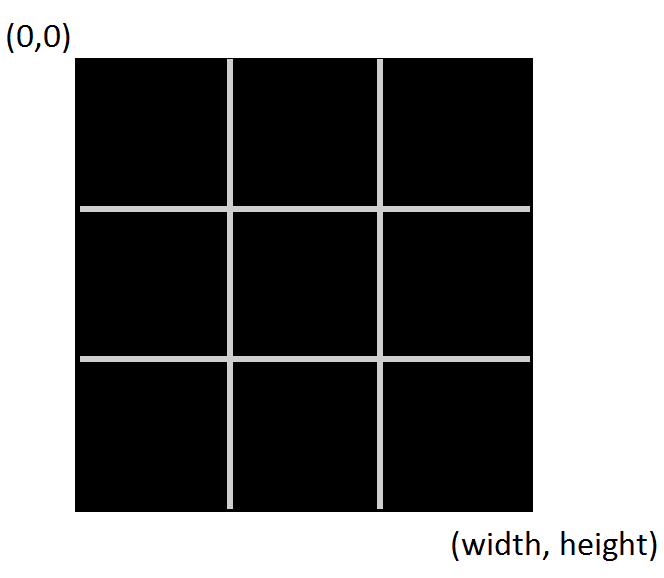
initialize();

}

1. Before we start drawing on the View, define a class-level variable Paint object:  
     
     
     
   and initialize it in the initialize() method:  
     
     
     
   We’ll use the mPaint object later to control the color and thickness of the lines we draw for the game board.

**private** Paint mPaint;

mPaint = **new** Paint(Paint.*ANTI\_ALIAS\_FLAG*);

1. Before we start drawing on the BoardView, it’s important to understand *what* we want to draw and *where* we want to draw it. Let’s first start with the game board. Below is a conceptual drawing of the BoardView as we want it to appear. The upper-left coordinate of the View is (0,0). The bottom right coordinate is (300,300) in dp units which will vary from device to device in terms of actual pixels. The actual width and height in pixels can be determined by calling View.getWidth() and View.getHeight(), which we will do momentarily.  
    
2. To draw the board, we will draw two vertical lines and two horizontal lines. You will now override the View’s onDraw() method which will use mPaint when drawing on the View’s canvas. The onDraw() function is called automatically any time the view needs to be painted, like when the View is first displayed. You should never call this method directly yourself.  
     
     
     
   The code above only draws the two vertical lines. **It is left to you** to draw the two horizontal lines.

@Override

**public** **void** onDraw(Canvas canvas) {

**super**.onDraw(canvas);

// Determine the width and height of the View

**int** boardWidth = getWidth();

**int** boardHeight = getHeight();

// Make thick, light gray lines

mPaint.setColor(Color.*LTGRAY*);

mPaint.setStrokeWidth(*GRID\_WIDTH*);

// Draw the two vertical board lines

**int** cellWidth = boardWidth / 3;

canvas.drawLine(cellWidth, 0, cellWidth, boardHeight, mPaint);

canvas.drawLine(cellWidth \* 2, 0, cellWidth \* 2, boardHeight, mPaint);

}

1. In order to test your code and see what it is actually drawing, we need to replace the previous game board which was made of buttons with our BoardView. Open the **layout/main.xml** file and **replace** the TableLayout XML containing the buttons with the following:  
     
     
     
   You may need to also adjust the relative placement of your other TextViews which relied on the name of the game board.

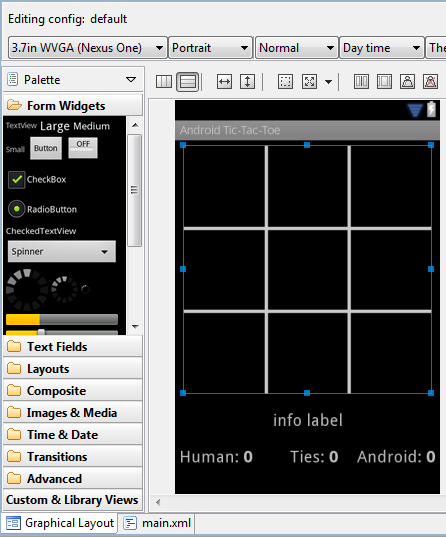
<edu.harding.tictactoe.BoardView

android:id=*"@+id/board"*

android:layout\_width=*"300dp"*

android:layout\_height=*"300dp"*

android:layout\_marginTop=*"5dp"*/>

1. Now click on the Graphical Layout tab of main.xml to see what the BoardView will look like. If you have successfully drawn the two horizontal lines in the correct location, the BoardView’s onDraw() method should make the View look like the image below:  
     
    
2. The BoardView must also display the X and O images as they appear on the board. Add the following code directly underneath the code which is drawing the board lines in the onDraw() method:   
     
     
     
   The code above uses canvas.drawBitmap() to draw the human/computer bitmaps in a rectangle that is just wide enough to fit in a board’s cell. **It’s left to you** to set the left, top, right, and bottom variables to define this destination rectangle appropriately. You’ll want to use the row, col, cellWidth, and GRID\_WIDTH variables in your calculations. To get you started on the right path, you could draw an X or O in the upper-left square of the board by setting left = 0, top = 0, right = cellWidth, and bottom = cellWidth.

// Draw all the X and O images

**for** (**int** i = 0; i < TicTacToeGame.*BOARD\_SIZE*; i++) {

**int** col = i % 3;

**int** row = i / 3;

// Define the boundaries of a destination rectangle for the image

**int** left = TODO

**int** top = TODO

**int** right = TODO

**int** bottom = TODO

**if** (mGame != **null** && mGame.getBoardOccupant(i) == TicTacToeGame.*HUMAN\_PLAYER*) {

canvas.drawBitmap(mHumanBitmap,

**null**, // src

**new** Rect(left, top, right, bottom), // dest

**null**);

}

**else** **if** (mGame != **null** && mGame.getBoardOccupant(i) == TicTacToeGame.*COMPUTER\_PLAYER*) {

canvas.drawBitmap(mComputerBitmap,

**null**, // src

**new** Rect(left, top, right, bottom), // dest

**null**);

}

}

1. The code above knows to draw the human or computer pieces based on the call to mGame.getBoardOccupant(i) which indicates if an X, O, or empty is present at that location. We have not yet declared the mGame variable, so you need to create an mGame data member of type TicTacToeGame for the BoardView class and create a setter for it that you will later call to give the BoardView access to the internal representation of the game board.

**private** TicTacToeGame mGame;

**public** **void** setGame(TicTacToeGame game) {

mGame = game;

}

1. Finally, write some accessors which will later be helpful to the AndroidTicTacToeActivity to determine which cell the user has touched on the BoardView:

**public** **int** getBoardCellWidth() {

**return** getWidth() / 3;

}

**public** **int** getBoardCellHeight() {

**return** getHeight() / 3;

}

## Using the BoardView

1. Earlier you added the BoardView to the main.xml file. Now go to the AndroidTicTacToeActivity class and declare a class-level variable for the BoardView. Inside onCreate(), set the mBoardView variable to the board you just created in main.xml, and call setGame() so the BoardView has access to the game board. You should also remove any code that is initializing buttons.

**private** BoardView mBoardView;

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

mGame = **new** TicTacToeGame();

mBoardView = (BoardView) findViewById(R.id.*board*);

mBoardView.setGame(mGame);

*snip...*

}

1. When starting a new game, we no longer have buttons that need to be cleared. Instead we need to just tell the BoardView to re-draw itself. This is done by “invalidating” the View. Modify the startNewGame() method by replacing the button code with a call to the View’s invalidate() method:  
     
     
     
   Now whenever startNewGame() is called, the BoardView’s onDraw() method will re-draw the game board.

**private** **void** startNewGame() {

mGame.clearBoard();

mBoardView.invalidate(); // Redraw the board

*snip...*

}

## Listen for Touches

If you run your application as it stands, you’ll be able to see the game board, but you will not be able to make a move. We need our app to sense touches on the game board to determine where the human is wanting to place his/her X. We’ll do this by adding a TouchListener to the BoardView.

1. First, create an inner class for AndroidTicTacToe called mTouchListener that implements the OnTouchListener interface as shown below. The onTouch() method will be called when the user first puts her finger on the device, when her finger moves locations while still touching the device, and when her finger is removed from the device. However, returning false from this method causes the move and up events not to be reported back to this event handler. Our app only needs to know when the finger is first placed on the device, so we’ll return false from onTouch().  
     
   mTouchListener will work much like the onClickListener used for the buttons in our last app except that we must now convert the touched (x,y) coordinate into a cell location to determine where the move should be placed. Note that some of the code is snipped out which handles checking for a win and making the computer move; all this code can be found in the existing onClickListener.

// Listen for touches on the board

**private** OnTouchListener mTouchListener = **new** OnTouchListener() {

**public** **boolean** onTouch(View v, MotionEvent event) {

// Determine which cell was touched

**int** col = (**int**) event.getX() / mBoardView.getBoardCellWidth();

**int** row = (**int**) event.getY() / mBoardView.getBoardCellHeight();

**int** pos = row \* 3 + col;

**if** (!mGameOver && setMove(TicTacToeGame.*HUMAN\_PLAYER*, pos)) {

// If no winner yet, let the computer make a move

***snip...***

}

// So we aren't notified of continued events when finger is moved

**return** **false**;

}

};

1. In the onCreate() method, attach the TouchListener to the BoardView:

// Listen for touches on the board

mBoardView.setOnTouchListener(mTouchListener);

1. Modify the setMove() function so it invalidates the BoardView when a legal move has been made; this causes the BoardView’s onDraw() method to fire which will display the user’s bitmap in the correct cell.

**private** **boolean** setMove(**char** player, **int** location) {

**if** (mGame.setMove(player, location)) {

mBoardView.invalidate(); // Redraw the board

**return** **true**;

}

**return** **false**;

}

1. Run your program and verify that your game works by touching the location on the game board where you’d like to move. As soon as you have touched an empty square in the board, your X image should appear in that square, and then the O image should appear in another square. You may want to click in all 9 location to verify that your earlier rectangle coordinates were calculated correctly. If you are not seeing your images anywhere on the screen, you may want to output to LogCat the left, top, right, and bottom variables after you have set them so you can debug your code. The right and bottom variables should be larger than the left and top variables.

## Adding Sound

Adding sound to a game can make it much more fun and appealing. Of course you should also allow users to turn off the sounds since not everyone likes to hear your game while in a public location.

1. You need to create or obtain two mp3 files. One will be played when the human makes a move and the other when the computer makes a move, so they should only be a second or two in duration. If you search Google for “free sound effects”, you’ll find many websites that offer free mp3 sound files to download. You can also use .wav files, but I have personally had problems getting them to work properly. Make sure that both filenames are composed only of lowercase letters, underscores, and digits since the file names will be converted into variable names.
2. In Eclipse, create a raw folder under res just like you did earlier in this assignment when creating a menu folder. The raw folder is where you should store sound files and other binary resources.
3. Drag and drop your mp3 files into the raw folder. You should see both files listed under raw in your project.
4. Crate two class-level MediaPlayer variables for the sound effects:

MediaPlayer mHumanMediaPlayer;

MediaPlayer mComputerMediaPlayer;

1. Media players consume resources that are shared by other Android processes, so special care should be taken to release the resources held by the media player when your application is not in use. We will load the sound effects into our media players when the Activity goes into the Resume state, and we will release the media players when the Activity goes into the Pause state. This could happen, for example, if another Activity comes to the foreground. We will discuss more about onResume and onPause in the next tutorial.

@Override

**protected** **void** onResume() {

**super**.onResume();

mHumanMediaPlayer = MediaPlayer.*create*(getApplicationContext(), R.raw.*sword*);

mComputerMediaPlayer = MediaPlayer.*create*(getApplicationContext(), R.raw.*swish*);

}

@Override

**protected** **void** onPause() {

**super**.onPause();

mHumanMediaPlayer.release();

mComputerMediaPlayer.release();

}

Note that the code above assumes the mp3 files were named sword.mp3 and swish.mp3; you will need to use the names of your mp3 files.

1. When the human makes a move (in the setMove() function), call the MediaPlayer’s start() method to play the human’s sound effect:

mHumanMediaPlayer.start(); // Play the sound effect

In a similar fashion, update setMove() to play the computer’s sound effect when the computer makes a move.

## Extra Challenge

Right now the computer moves immediately after the human which doesn’t leave enough time to see the “Android’s turn” message, and it causes the computer’s sound effect to be played immediately after the human’s sound effect, if at all. Fix these problems by making the computer wait one second before making its move.   
  
Although you could create a loop that blocks the UI thread for a second, this would make your app unresponsive during the delay. A better solution is to use the android.os.Handler’s postDelay() method. This method takes two arguments: 1) a Runnable (an interface representing an executable command), and 2) a delay in milliseconds before the Runnable is executed on the Handler’s thread. Here’s an example that outputs “Hello” to LogCat after 4 seconds has passed:

Handler handler = **new** Handler();   
handler.postDelayed(**new** Runnable() {

**public** **void** run() {

Log.*v*(*LOG\_TAG*, "Hello");

}

}, 4000);

To use the Handler correctly, you’ll need to re-organize the code that is responsible for making the computer move after the human moves. You’ll find it helpful to use a class-level variable to indicate whose turn it is because you do not want to allow the human to make a move when it’s the computer’s turn; right now this is not an issue because the computer moves immediately after the human moves.

**Except as otherwise noted, the content of this document is licensed under the Creative Commons Attribution 3.0 License**

<http://creativecommons.org/licenses/by/3.0>