Steps to create a Tic-Tac-Toe game – Part 2

Let's fix a couple of things before moving on. The computer only makes one move, so correct that by updating the **Board.setSelectedCell** method as follows:

```
public void setSelectedCell(String symbol) {
	if (game.setCellIfValid(selX, selY, symbol)) {
		invalidate();
		if (!game.isGameOver())
			game.doComputerMove();
	} else {
		Log.d(getClass().getSimpleName(), "cell already filled");
		startAnimation(AnimationUtils.loadAnimation(game, R.anim.shake));
	}
}
```

That will also require you to change the **Game.doComputerMove** method from *private* to *public*. Edit the **Game.isGameOver** method and also add the supporting functions shown below:

```
public boolean isGameOver() {
	int[] winner = findWinner();

	if (winner != null) {
		if (cells[winner[0]] == playerSymbol) {
			showEndOfGame("Congratulations!  You won this game!");
			return true;
		} else {
			showEndOfGame("Opps, the computer won this game.");
			return true;
		}
	} else {
		boolean tie = true;
		for (int i = 0; i < cells.length; i++)
			if (cells[i] == SYMBOL_SPACE) {
				tie = false;
				break;
			}

		if (tie) {
			showEndOfGame("Nobody won!  Better luck next time.");
			return true;
		}
	}

	return false;
}

private int[][] winningCombos = new int[][] { { 0, 1, 2 }, { 0, 3, 6 },
		{ 0, 4, 8 }, { 1, 4, 7 }, { 2, 4, 6 }, { 2, 5, 8 }, { 3, 4, 5 },
		{ 6, 7, 8 } };

private int[] findWinner() {
	int[] winner = null;

	for (int i = 0; i < winningCombos.length; i++) {
		int[] combo = winningCombos[i];
```

```
            String s1 = cells[combo[0]];
            String s2 = cells[combo[1]];
            String s3 = cells[combo[2]];

            if (s1 != SYMBOL_SPACE && s2 != SYMBOL_SPACE && s3 != SYMBOL_SPACE)
                  if (s1 == s2 && s2 == s3) {
                        winner = combo;
                        break;
                  }
      }
      return winner;
}

private void showEndOfGame(String msg) {
      TextView message = new TextView(this);
      message.setText(msg);

      new AlertDialog.Builder(this)
                  .setTitle("Game Over")
                  .setView(message)
                  .setPositiveButton(android.R.string.ok,
                              new DialogInterface.OnClickListener() {

                                    @Override
                                    public void onClick(DialogInterface dialog,
                                                int which) {
                                          finish();
                                    }
                              }).create().show();
}
```

The game is now fully functional, although the computer is very easy to beat. If there is time, we should try to improve the **doComputerMove** algorithm.

Let's add support for background music next. Copy two suitable audio files into the project with the names ***res/raw/background.mp3*** and ***res/raw/intro.mp3***. Emulators usually only handle MP3, OGG and WAV formats well, but real Android devices also support WMA, AMR, and MIDI audio.

Now, update the ***res/xml/settings.xml*** file and add another **Checkbox** preference as shown here:

```
    <CheckBoxPreference
        android:defaultValue="true"
        android:key="music"
        android:summary="@string/music_summary"
        android:title="@string/music_title" />
```

Also add the following to ***res/values/strings.xml***:

```
    <string name="music_title">Music</string>
    <string name="music_summary">Play background music in the game.</string>
```

We need a way to access these preferences, so add the following to the **Prefs** class:

```
private static final String OPT_COLOR = "color";
private static final boolean OPT_COLOR_DEFAULT = true;
```

```
private static final String OPT_MUSIC = "music";
private static final boolean OPT_MUSIC_DEFAULT = true;

public static boolean getColor(Context context) {
	return PreferenceManager.getDefaultSharedPreferences(context)
			.getBoolean(OPT_COLOR, OPT_COLOR_DEFAULT);
}

public static boolean getMusic(Context context) {
	return PreferenceManager.getDefaultSharedPreferences(context)
			.getBoolean(OPT_MUSIC, OPT_MUSIC_DEFAULT);
}
```

Next, create a new **Music** (***src/pkg.name/Music.java***)class as shown below:

```
package com.bamafolks.android.games.tictactoe;

import android.content.Context;
import android.media.MediaPlayer;

public class Music {

	private static MediaPlayer mp = null;

	public static void play(Context context, int resource) {
		stop(context);
		if (Prefs.getMusic(context)) {
			mp = MediaPlayer.create(context, resource);
			mp.setLooping(true);
			mp.start();
		}
	}

	public static void stop(Context context) {
		if (mp != null) {
			mp.stop();
			mp.release();
			mp = null;
		}
	}

}
```

Finally, override the *onResume* and *onPause* methods in both the **Game** and **MainActivity** classes as shown here:

```
@Override
protected void onPause() {
	super.onPause();
	Music.stop(this);
}

@Override
protected void onResume() {
	super.onResume();
	Music.play(this, R.raw.background);
```

```
}
```

NOTE: Use **R.raw.intro** in ***MainActivity.onResume*** method.

Run the program and enjoy the music!

Let's wrap this up by adding support for resuming a game that was in progress. Start by adding the new constants and and updating **Game.onPause** as shown below:

```
public static final int CONTINUE = -1;

private static final String PREF_GAME = "tictactoe";
private static final String PLAYER_SYMBOL = "tictactoe";
private static final String COMPUTER_SYMBOL = "tictactoe";

@Override
protected void onPause() {
      super.onPause();
      Music.stop(this);

      SharedPreferences prefs = getPreferences(MODE_PRIVATE);
      Editor editor = prefs.edit();
      editor.putString(PREF_GAME, TextUtils.join(",", cells));
      editor.putString(PLAYER_SYMBOL, playerSymbol);
      editor.putString(COMPUTER_SYMBOL, computerSymbol);
      editor.commit();
}
```

Add the following to the **MainActivity.onClick** method:

```
      case R.id.continue_button:
            startGame(Game.CONTINUE);
            break;
```

Finally, update the **Game.getCells** method as shown below:

```
public String[] getCells(int first) {
      if (first == CONTINUE) {
            SharedPreferences prefs = getPreferences(MODE_PRIVATE);
            playerSymbol = prefs.getString(PLAYER_SYMBOL, SYMBOL_X);
            computerSymbol = prefs.getString(COMPUTER_SYMBOL, SYMBOL_O);
            return TextUtils.split(
                        getPreferences(MODE_PRIVATE).getString(PREF_GAME,
                                    " , , , , , , , , , "), ",");
      }

      String[] grid = new String[9];
      for (int i = 0; i < 9; i++)
            grid[i] = SYMBOL_SPACE;

      switch (first) {
      case PLAYER_FIRST:
            playerSymbol = SYMBOL_X;
            computerSymbol = SYMBOL_O;
            break;
```

```
		case COMPUTER_FIRST:
			playerSymbol = SYMBOL_O;
			computerSymbol = SYMBOL_X;
			break;
		case RANDOM_FIRST:
			if (random.nextBoolean()) {
				playerSymbol = SYMBOL_X;
				computerSymbol = SYMBOL_O;
			} else {
				playerSymbol = SYMBOL_O;
				computerSymbol = SYMBOL_X;
			}
		}
		return grid;
}
```

The game is now complete, however this is still one remaining problem. New games should work as expected, but when you continue a game, things may not work so well.

Class Discussion: Determine what the problem is.