Android Application Programming

Tutorial 5: Changing the Orientation and Saving State

Introduction

Your tic-tac-toe game only works in portrait mode. If you were to flip the emulator into landscape mode by pressing Ctrl-F11, you would see the board is cleared of all the images, and the text beneath the board is not visible.

The goal of this tutorial is to show you how to develop an application that can work in portrait or landscape mode. You will also learn how to make the application's data persist when the orientation is changed and how to make data persist between application invocations.

Setup Your Project

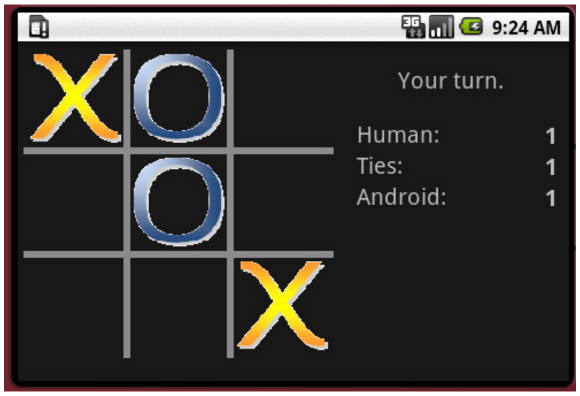
Download and import the Andriod project into Eclipse which implements the extra challenge from the previous tutorial:

1. Download the zip file **AndroidTicTacToeSolution3.zip** from Google Code University.
2. Unzip **AndroidTicTacToeSolution3.zip** in a directory of your choosing.
3. In Eclipse, choose **File → Import… → Existing Projects into Workspace…**
4. Select "Select root directory:" and click **Browse**.
5. Go to the **AndroidTicTacToeSolution3** directory that was created in step 2 and click OK.
6. Make sure "Copy projects into workspace" is checked, then click **Finish**.

Changing the Orientation

When you press Ctrl-F11, your emulator will switch from portrait mode to landscape mode. Pressing Crtl-F11 again switches it back. Your game was built for portrait mode, so it does not behave properly when switched to landscape mode.

You can force Android to only show your app in portrait mode: just add `android:screenOrientation="portrait"` (or `"landscape"` if you wish) to the `<activity>` element in the app's AndroidManifext.xml file. However, best practice is to allow the user to interact with the app in either orientation.



Your turn.

Human:	1
Ties:	1
Android:	1

Our goal is to create a landscaped orientation that looks like the screenshot on the right. The first thing we need to do is create a new layout for landscape mode:

1. Create a `res/layout-land` directory.

2. Copy the `res/layout/main.xml` file into `layout-land`. This is the file that Android will apply to the Activity's View when put in landscape mode.

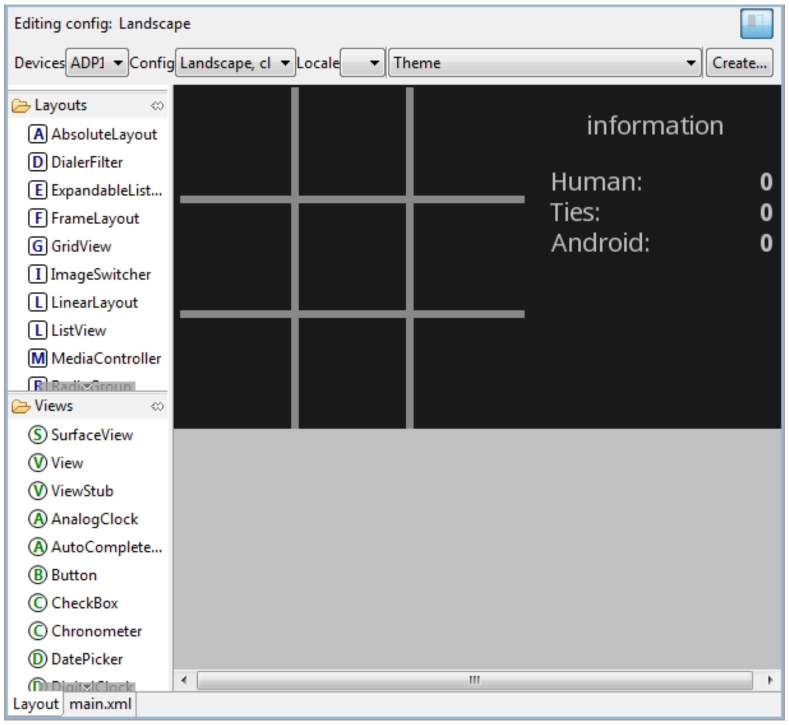
3. Change the alignment of the `BoardView` to the left of its parent, reduce the size so it is 270x270, give it some margin distance from the top of the screen:

```
<edu.harding.tictactoe.BoardView
     android:id="@+id/board"
     android:layout_width="270px"
     android:layout_height="270px"
     android:layout_marginLeft="5px"
     android:layout_marginTop="5px"
     android:layout_alignParentTop="true"
     android:layout_alignParentLeft="true"
/>
```

4. To create more screen real estate, remove the title bar from the application window by modifying the application element in AndroidManifest.xml:

```
<application android:icon="@drawable/icon" android:theme="@android:style/Theme.NoTitleBar">
```

5. You will also need to change the layout of the other widgets so they are displayed to the right of the game board. **This is left for you to do**. To test your layout without running your game in the emulator, you can switch to Layout mode and set the Config to Landscape (open or closed) as shown on the right.



6. We need to make some code changes so that the new width and height of the board are used when drawing the board and images. Override the `onSizeChanged()` function in the BoardView class which is called during the layout procedure. The `w` and `h` variables will be 300x300 when in portrait mode and 270x270 when in landscape mode.

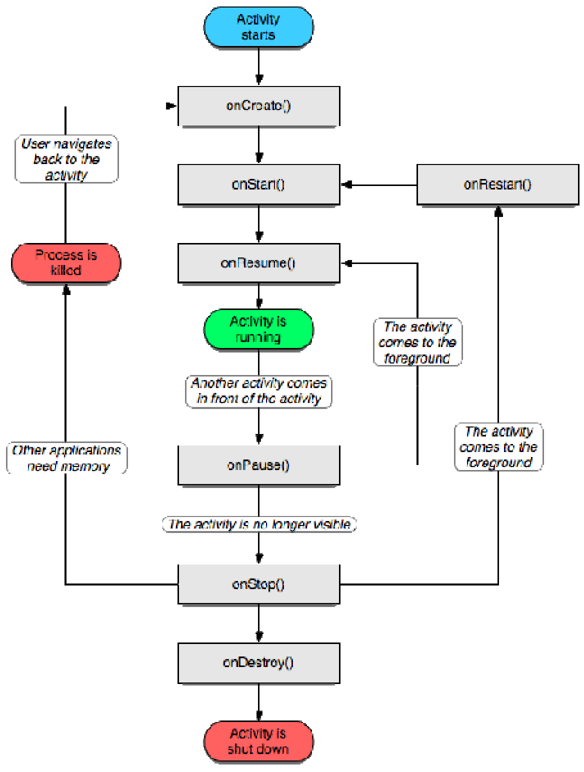
```
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {

   // w and h contain the new width and height
   mBoardHeight = h;
   mBoardWidth = w;

   mBoardCellWidth = mBoardWidth / 3;
   mBoardCellHeight = mBoardHeight / 3;
}
```

Run your program and press Ctrl-F11 to change its orientation. Your app should switch orientations and display all the screen elements correctly.

Try playing a game, and before the game is over, press Ctrl-F11. What happens? When the orientation is changed in the middle of the game, your game starts over from the beginning. If you play a few games and then change the orientation, you'll notice that the scores are reset to zero. Ideally we'd like our game to maintain its state when changing orientation, so we'll fix this problem next.

Saving Instance State

Before discussing how state information is stored, it is helpful to get a firm grasp on the life of an Activity. The Activity life cycle is illustrated in the diagram[1] on the next page. Each rectangle represents the Activity's callback methods which are called in response to different events. For example, after starting an Activity, the `onCreate()` method is called first followed by `onStart()` and `onResume()`. The application then runs until another Activity comes in front of the currently running Activity, causing the `onPause()` method to trigger. If the Activity is no longer visible, `onStop()` triggers, and if Android decides another process needs the memory, the Activity is killed. However, if the Activity is navigated to before being killed, `onRestart()` is triggered followed by `onStart()` and `onResume()`.


Activity starts
onCreate()
onStart()
onRestart()
onResume()
User navigates back to the activity
Process is killed
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Changing the orientation of an Android device will cause your Activity to terminate and restart, so all the app's variables are reset, and the game is started anew. This is just like what happens when when your Activity is no longer in the foreground, its memory is reclaimed, and you navigate back to the app.

Fortunately, Android provides a mechanism to remember whatever pieces of information we would like when the Activity is started again after changing the device's orientation. We first need to decide what information is necessary to start our game back in the same state. We'll bundle the information together before the Activity is killed, and Android will pass it back to us in the onCreate() method. We'll extract the bundled information and put it back into the necessary variables to pick up right where we left off.

1. To save the state of the game, you need to override the `onSaveInstanceState()` method for the `AndroidTicTacToe` class (the Activity). This method is called by Android before it pauses the application (before `onPause()` is triggered), and it passes the method a `Bundle` object which can be used to save key/value pairs that will be given back to the application (via another call to `onCreate()`) even if the Activity is dropped from memory.

[1] Figure is from *Application Fundamentals* at http://developer.android.com/guide/topics/fundamentals.html

```
@Override
protected void onSaveInstanceState(Bundle outState) {
   super.onSaveInstanceState(outState);

   outState.putCharArray("board", mGame.getBoardState());
   outState.putBoolean("mGameOver", mGameOver);
   outState.putInt("mHumanWins", Integer.valueOf(mHumanWins));
   outState.putInt("mComputerWins", Integer.valueOf(mComputerWins));
   outState.putInt("mTies", Integer.valueOf(mTies));
   outState.putCharSequence("info", mInfoTextView.getText());
   outState.putChar("mGoFirst", mGoFirst);
}
```

For the above code to work, you will need to create a `getBoardState()` function for the `TicTacToeGame` class which returns a character array representing the state of the board.

2. When `onCreate()` is called the first time, its parameter `savedInstanceState` will be `null`, and the game should be started like usual. But after changing the orientation, `onCreate`'s `savedInstanceState` will contain the key/value pairs placed in `onSaveInstanceState`'s `outState` bundle. You will need to extract the values from the `Bundle` and place them back into their associated variables like so:

```
@Override
public void onCreate(Bundle savedInstanceState) {

skip...

   if (savedInstanceState == null) {
      startNewGame();
   }
   else {
      // Restore the game's state
      mGame.setBoardState(savedInstanceState.getCharArray("board"));
      mGameOver = savedInstanceState.getBoolean("mGameOver");
      mInfoTextView.setText(savedInstanceState.getCharSequence("info"));
      mHumanWins = savedInstanceState.getInt("mHumanWins");
      mComputerWins = savedInstanceState.getInt("mComputerWins");
      mTies = savedInstanceState.getInt("mTies");
      mGoFirst = savedInstanceState.getChar("mGoFirst");
   }
   displayScores();
}
```

For the above code to work, you will need to create a `setBoardState()` function for the `TicTacToeGame` class that allows the board state to be set to the values in the given char array. You will likely find the array's `clone()` method to be helpful. **This is left to you to complete.**

3. The code above also calls a `displayScores()` function to show the scores. Declare it like so:

```
private void displayScores() {
   mHumanScoreTextView.setText(Integer.toString(mHumanWins));
   mAiScoreTextView.setText(Integer.toString(mAiWins));
   mTieScoreTextView.setText(Integer.toString(mTies));
}
```

4. Note that there is also an `onRestoreInstanceState()` method that can also be overridden for the Activity to restore the application's state. So instead of placing your restore code in the else statement in step 2, you could do this:

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
   super.onRestoreInstanceState(savedInstanceState);

   mGame.setBoardState(savedInstanceState.getCharArray("board"));
   mGameOver = savedInstanceState.getBoolean("mGameOver");
   mInfoTextView.setText(savedInstanceState.getCharSequence("info"));
   mHumanWins = savedInstanceState.getInt("mHumanWins");
   mComputerWins = savedInstanceState.getInt("mComputerWins");
   mTies = savedInstanceState.getInt("mTies");
   mGoFirst = savedInstanceState.getChar("mGoFirst");
}
```

5. Run your application and make some moves. When you press Ctrl-F11, the board should remain in the same condition it was before. However, you'll probably notice that the computer makes an additional move when it's your turn! This is because the variable that ensures it's the correct person's turn is not being saved. **It's left to you to fix this bug.**

Saving Persistent Information

If you were to click the Back button on the emulator and restart the application, either by clicking on the app's icon or using Eclipse to restart, your application's state is not saved; your game board will be cleared and your game scores (Human, Tie, Android) will be reset to 0. That's because Android only saves state information (by calling `onSaveInstanceState()`) if Android is responsible for killing the Activity. If you kill it by clicking the Back button, for example, then `onSaveInstanceState()` is not called.

In order for information to persist between application restarts, there are several mechanisms to choose from. You can store data to a file and read it back in, or you can save data to a SQLLite database. If you only need to store simple key/value pairs like we did before, the `SharedPreferences` class makes this easy to do.

Follow the instructions below to use the `SharedPreferences` class to make the game scores persist between application restarts:

1. Declare a `SharedPreferences` data member for the `AndroidTicTacToe` class:

```
private SharedPreferences mPrefs;
```

2. In the `onCreate()` method, use `Activity.getSharedPreferences()` to initialize `mPrefs`. The first argument is the name of the preference file, and the second argument indicates that the file should hold private values that we don't want to share with other applications.

```
mPrefs = getSharedPreferences("ttt_prefs", MODE_PRIVATE);
```

3. When your application is being terminated, the Activity's `onStop()` method will be called (refer back to the Activity life cycle figure shown earlier). This is your chance to save any persistent information. The `SharedPreferences` object to save the scores like so:

```
@Override
protected void onStop() {
   super.onStop();

   // Save the current scores
   SharedPreferences.Editor ed = mPrefs.edit();
   ed.putInt("mHumanWins", mHumanWins);
   ed.putInt("mAiWins", mAiWins);
   ed.putInt("mTies", mTies);
   ed.commit();
}
```

Note that the `SharedPreferences.Editor` has methods for storing any primitive type. Calling `commit()` replaces any of the data that was previously stored in `SharedPreferences` with the new data.

4. Now we need to restore the game scores from `mPrefs` when the `onCreate()` method is being called:

```
mPrefs = getSharedPreferences("ttt_prefs", MODE_PRIVATE);

// Restore the persistent scores
mHumanWins = mPrefs.getInt("mHumanWins", 0);
mAiWins = mPrefs.getInt("mAiWins", 0);
mTies = mPrefs.getInt("mTies", 0);
```

In the code above, `getInt()` is supplied 0 for the second argument which will be the value returned if the preferences have not been previously saved (like the first time you run your modified application).

5. Your app will now save the scores indefinitely. However, the user may not want to be reminded that Android beat him 20 times at tic-tac-toe. So let's give the user the ability to reset the scores. Add a new menu item to `res/menu/options_menu.xml` labeled "Reset Scores", and add an image for the menu item. To keep the menu from getting cluttered, remove the Quit option which really isn't necessary.

6. When the Reset Scores menu item is selected, set the `mHumanWins`, `mComputerWins`, and `mTies` variables to 0 and re-display the scores.

7. Since we are saving the values of `mHumanWins`, `mAiWins`, and `mTies` in the preferences, you can safely remove all the code you previously entered that saved their values with `savedInstanceState`.

Now run your application and play a few games. Take a look at the scores. Click the Back button on the emulator and restart your game by clicking on the TicTacToe icon or starting it from Eclipse. The scores should be the same as they were before you started a new game. Now test your Reset Scores menu item. The scores should immediately go back to 0.

Extra Challenge

If you press Ctrl-F11 immediately after making a move and before the computer has moved, the application will likely crash. You'll find the LogCat helpful for determining what is causing the crash. A simple try-catch block will solve this problem. However, the game will still not work correctly when changing the orientation immediately before the computer moves; the game will indicate it's still Android's turn, but the computer will never make a move. This problem has to do with the `Handler` executing code on old instances of an `Activity` which has been terminated. You need to fix this problem by making the computer make a move after the new `Activity` has started if it's the computer's turn to move.