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 your device into landscape mode, you would see the board is cleared of all the images, and the text beneath the board may also not be visible (depending if you changed the size of the board). While we could force the user to only use our app in portrait mode, we should make our app more flexible by allowing the user to play with it in either mode.

The goal of this tutorial is to show you how to develop an application that can work in portrait and 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.

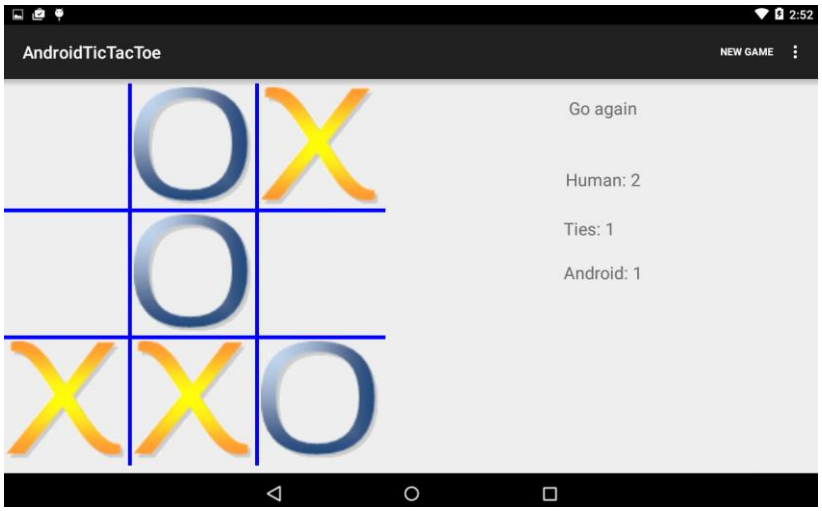
Changing the Orientation

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 `AndroidManifest.xml` file. However, best practice is to allow the user to interact with the app in either orientation.

Our goal is to create a landscaped orientation that looks like the screenshot below. Here we go!!

1. Create a `res/layout-land` directory. (for some reason if it does not appear…keep going)

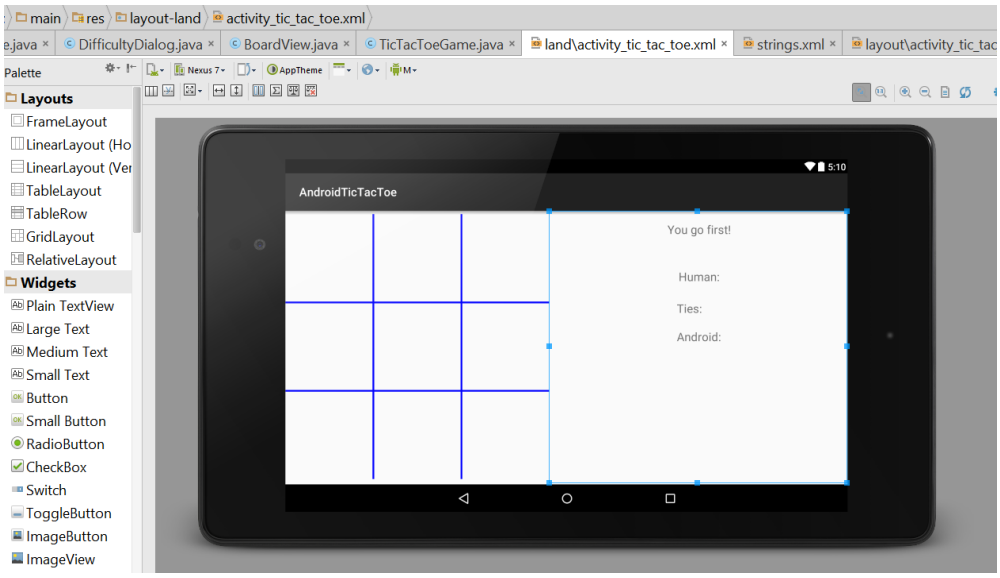
2. Create a new Layout resource file (right-click on layout folder) and name it `activity_tic_tac_toe`. Make sure that you specify the `layout-land` directory. This is the file that Android will apply to the Activity's View when the emulator is put in landscape mode.



 Frank McCown at Harding University

Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

3. Open the `layout/activity_tic_tac_toe.xml(land)` file and change the size of the `BoardView` to make it take up an appropriate amount of space (you may also do this for the normal layout to take up more space on the screen). Also align it on the left side of the screen. To do this, you can change the `LinearLayout`'s `orientation` property to `horizontal` or replace the `LinearLayout` entirely with a `RelativeLayout`. You will also need to change the layout of the `TextView` controls so they are displayed to the right of the game board. **It is left to you to make these edits.** To test your layout without running your game in the device, you can switch to Design Layout mode and set the orientation to Landscape as shown below.



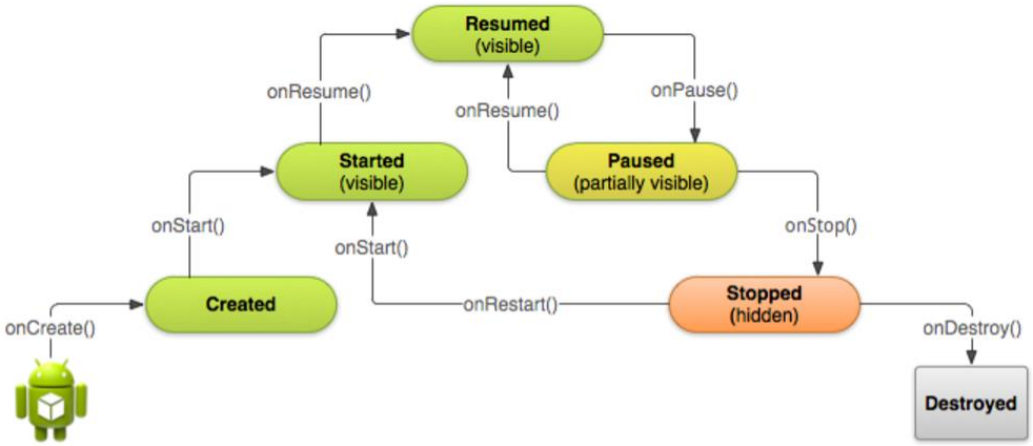
4. After you have the `BoardView` and `TextViews` aligned properly, run your program and change its orientation. Your app should switch orientations and display all the screen elements correctly.

   Try making a few moves, and before the game is over, change the orientation. What happens? When the orientation is changed in the middle of the game, the board is cleared and 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 also reset to zero. You can imagine how frustrated a user would get if they were about to win a game, they shifted their device a little which caused the orientation to change, and the game reset! 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 cycle of an Activity. The Activity life cycle is illustrated in the diagram[1] below. Each rectangle represents the Activity's 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()`.

[1] Figure is from *Application Fundamentals* at http://developer.android.com/training/basics/activity-lifecycle/starting.html



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 `TicTacToe` 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.

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

    outState.putCharArray("board", mGame.getBoardState());
    outState.putBoolean("mGameOver", your variable for game over);
    outState.putInt("mHumanWins", TODO);
    outState.putInt("mComputerWins", TODO);
    outState.putInt("mTies", TODO);
    outState.putCharSequence("info", TODO);
    outState.putChar("mGoFirst", mGoFirst);  // if you have implemented this is your game
}
```

   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. **It is left for you to implement this method. You also need to save the difficulty level.**

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:

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

snip...

   if (savedInstanceState == null) {
      startNewGame();
   }
   else {
      // Restore the game's state
      // Since you saved the state by using the Bundle and "put" methods, you now need
      // to "undo" all of the "puts". Figure out what Bundle methods you need to use.
   }

}
```

   For the above code to work, you will also 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. **This is left to you to complete.**

3. Note that there is also an onRestoreInstanceState() method that can 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 have placed it in an onRestoreInstanceState() method.

4. Run your application and make some moves. When you change the orientation the board should remain in the same condition it was before.

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 Android Studio 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 could store data to a file and read it back in, or you could save data to a SQLite database. If you only need to store simple key/value pairs like we have been doing, the `SharedPreferences` is the ideal class to use.

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 `TicTacToe` 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 is typically `MODE_PRIVATE` or 0.

```
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. Use 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);
   // save anything else you might need to retain here!!
   TODO
   ed.commit();
}
```

   Note that the `SharedPreferences.Editor` has methods for storing any primitive type. You need to figure out how to save the Board State (an array of char). Calling `commit()` replaces any of the data that was previously stored in `SharedPreferences` with the new data.

 Frank McCown at Harding University
Modified by Mike Efram at Healdsburg High School to suit needs of Advanced Programming Projects class.

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 scores
mHumanWins = mPrefs.getInt("mHumanWins", 0);
// and anything else that you saved above…
```

   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 labeled "Reset Scores".

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 various values 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 your device and restart your game by clicking on the TicTacToe icon. 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

1. The game's difficulty level is not being saved, so every time the game starts, the difficulty level is reset to your original setting. Fix this by saving the game difficulty level to `SharedPreferences`.