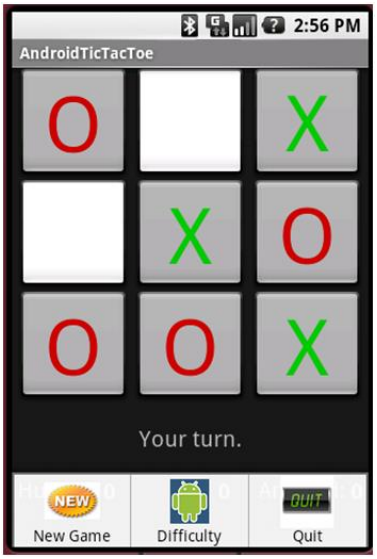
Android Application Programming

Challenge: Menus and Dialog Boxes

Introduction

In the previous tutorial, you created a simple tic-tac-toe game for the Android. The goal of this tutorial is to improve your game by creating a more sofisticated menu that uses a few dialog boxes. In this tutorial you will create an options menu (as shown on the right) with three options:



1. **New Game** – to start a new game
2. **Difficulty** – to set the AI difficulty level to Easy, Harder, or Expert
3. **Quit** – to quit the app

You should familiarize yourself with the official Menu Design Guidelines when designing your own menus. Android also supports *context menus*, floating lists of menu items that appear when holding your finger down on the screen (a long press), but they will not be covered in this tutorial.

Changes to the Game Logic

We are going to create a menu option in this tutorial to change the difficulty level of the game. Right now the game only has one difficulty level which we'll call "Expert". You will need to modify the `TicTacToeGame` class so the difficulty level can be set. When set to the "Easy" level, the computer will always just make random moves. When set to the "Harder" level, the computer will make a winning move if possible, otherwise it will make a random move.

1. Open **TicTacToeGame.java**, and add an enumeration for the difficulty level and a variable for keeping track of the current difficulty level setting:

```
public class TicTacToeGame {

    // The computer's difficulty levels
    public enum DifficultyLevel {Easy, Harder, Expert};

    // Current difficulty level
    private DifficultyLevel mDifficultyLevel = DifficultyLevel.Expert;
```

2. Create getters and setters for the difficulty level:

```
public DifficultyLevel getDifficultyLevel() {
    return mDifficultyLevel;
}

public void setDifficultyLevel(DifficultyLevel difficultyLevel) {
    mDifficultyLevel = difficultyLevel;
}
```

(cc) BY-NC

3. Finally, modify the `getComputerMove()` method to call the appropriate function depending on the difficulty level. **It's left to you** to implement `getRandomMove()`, `getWinningMove()`, and `getBlockingMove()` based on the pre-existing code. Note that the functions might need to *temporarily* modify the board array, but they should leave the array in the same state it was in before the functions were called.

```
public int getComputerMove() {

   int move = -1;

   if (mDifficultyLevel == DifficultyLevel.Easy)
   move = getRandomMove();
   else if (mDifficultyLevel == DifficultyLevel.Harder) {
   move = getWinningMove();
   if (move == -1)
      move = getRandomMove();
   }
   else if (mDifficultyLevel == DifficultyLevel.Expert) {

   // Try to win, but if that's not possible, block.
   // If that's not possible, move anywhere.
   move = getWinningMove();
   if (move == -1)
      move = getBlockingMove();
   if (move == -1)
      move = getRandomMove();
   }

   return move;
}
```

Create the Menu in XML

Now we will add a menu to the Activity that will be used to start a new game, set the TicTacToeGame's difficulty level, and quit. Android can display up to six menu options at once, and a More option makes an *expanded menu* available. We'll only have three options in our menu. Instead of creating the menu options inside of your code like we did in the previous assignment, it's best-practice to create your menu via XML instead so changes can be made to the menu without modifying any code.

The first thing you need to do is create the menu in XML:

1. Right-click on the **res** directory and select New → Folder
2. Name the folder **menu**.
3. Right-click on the new **menu** directory and select New → File
4. Name the file **options_menu.xml**.
5. Enter the following XML into the new file:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:title="New Game"
          android:icon="@drawable/new_game" />
    <item android:id="@+id/ai_difficulty"
          android:title="Difficulty"
          android:icon="@drawable/difficulty_level"  />
    <item android:id="@+id/quit"
          android:title="Quit"
          android:icon="@drawable/quit_game" />
</menu>
```

This creates three menu options with text and images. You will add the menu images to your project in the next step.

Add Menu Items Images

Now you'll need to add the images that are used in the menu items to your project:

1. Create **new_game.png**, **difficulty_level.png**, and **quit_game.png** images using your favorite image editor. Feel free to use images you find on the web (AndroidIcons.com has some free icons) or from the SDK (e.g., C:\ANDROID_SDK\platforms\android-2.1\data\res\drawable-mdpi). Make sure the images are not taller than 42 pixels, or the images will obstruct the text in the menu items. If you want to create icons that conform strictly to the Android platform, take a look at the official Android Icon Design Guidelines which give very specific instructions for creating menu item icons, including instructions for creating three different sizes of images for the various screen sizes Android may be running on.

2. Your project will have three folders under res called drawable-hdpi, drawable-ldpi, and drawable-mdpi. These folders are for holding images that are best for different resolutions. We will not concern ourselves with this issue right now. So create a folder called **res/drawable**, just like you did earlier when you created res/menu. The images in this folder will be used regardless of the Android device's screen resolution.

3. Drag the image files you created in step 1 on top of the **drawable** folder in Eclipse. This will require you to have the file explorer window open in front of Eclipse drag and drop the files. Another way to add your images to the project is to copy the images into the **drawable** folder using the file explorer and then select **File → Refresh** in Eclipse.

4. You should now see your png images in the **drawable** folder. If you used different file names than were specified in step 1, you should correct your options_menu.xml file so it uses the same names as your files.

Display the Menu and Respond to Menu Selections

Now you need to add code to your program to create the menu and respond to menu items selections:

1. Modify your code to load the options menu by modifying your `onCreateOptionsMenu()` function. Note how `R.menu.options_menu` equates to the ID in options_menu.xml you created earlier.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
     super.onCreateOptionsMenu(menu);

     MenuInflater inflater = getMenuInflater();
     inflater.inflate(R.menu.options_menu, menu);
     return true;
}
```

   Next you need to modify the `onOptionsItemSelected()` function which is called when a menu option is selected. You can tell which menu item was selected by examining the menu item's ID which was given in the `options_menu.xml` file. (showDialog is deprecated).

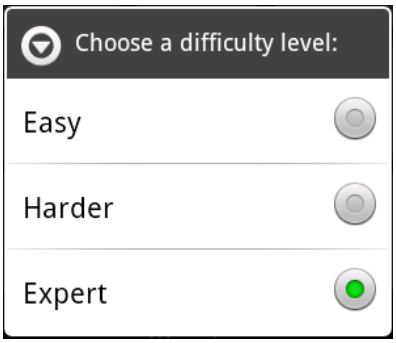
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.new_game:
          startNewGame();
          return true;
    case R.id.ai_difficulty:
         showDialog(DIALOG_DIFFICULTY_ID);
          return true;
    case R.id.quit:
          showDialog(DIALOG_QUIT_ID);
         return true;
    }
    return false;
}
```

The menu you just added will display dialog boxes that correspond to some constants which we'll define next.

2. Create two constants which will be used to identify the two dialog boxes:

```
static final int DIALOG_DIFFICULTY_ID = 0;
static final int DIALOG_QUIT_ID = 1;
```

3. You will next need to override the `Activity`'s `onCreateDialog()` method which is called the first time `showDialog()` is called on a dialog box. Its `id` parameter indicates which dialog box should be created.



We will make use of the `AlertDialog` class which is useful for creating simple dialog boxes with 1-3 buttons.

In the code below, there are two places marked "TODO" where the code is incomplete. **You will need** to set the `selected` variable so the `setSingleChoiceItems()` function will properly show radio buttons labeled Easy, Harder, and Expert as shown on the right. The currently selected difficulty level should initially be selected.

When one of the radio buttons is selected, the dialog will close, and the `item` parameter of `onClick` will indicate which radio button was selected. **You will need to write the code** to set the game's internal difficulty level. You will also need to add the appropriate string constants to **strings.xml**.

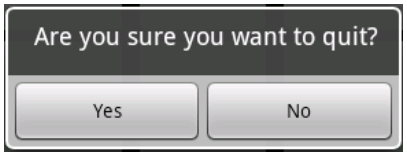
Note that each call to the builder object returns the reference to the builder object so method-chaining becomes possible.

(cc) BY-NC

```
@Override
protected Dialog onCreateDialog(int id) {
   Dialog dialog = null;
    AlertDialog.Builder builder = new AlertDialog.Builder(this);

   switch(id) {
   case DIALOG_DIFFICULTY_ID:

      builder.setTitle(R.string.difficulty_choose);

      final CharSequence[] levels = {
                getResources().getString(R.string.difficulty_easy),
                getResources().getString(R.string.difficulty_harder),
                getResources().getString(R.string.difficulty_expert)};

      // TODO: Set selected, an integer (0 to n-1), for the Difficulty dialog.
   // selected is the radio button that should be selected.

       builder.setSingleChoiceItems(levels, selected,
            new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int item) {
                  dialog.dismiss();   // Close dialog

                  // TODO: Set the diff level of mGame based on which item was selected.


                  // Display the selected difficulty level
                  Toast.makeText(getApplicationContext(), levels[item],
                      Toast.LENGTH_SHORT).show();
            }
       });
       dialog = builder.create();

      break;
   }

   return dialog;
}
```

The code above displays a Toast message (see `Android.widget.Toast`) that indicates which difficulty level was selected from the dialog box. Toast messages appear for just a few seconds and are helpful for presenting short, brief information to the user.

4. Next add to the same `switch` statement the code to display a Yes/No confirmation dialog when the user clicks on Quit. You'll need to add the appropriate strings to **strings.xml**. When the user clicks on Yes, the `onClick()` method executes and calls the Activity's `finish()` method which causes the activity to terminate.



(cc) BY-NC

```
   switch(id) {
...
   case DIALOG_QUIT_ID:
      // Create the quit confirmation dialog

       builder.setMessage(R.string.quit_question)
              .setCancelable(false)
              .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                        AndroidTicTacToeActivity.this.finish();
                  }
              })
              .setNegativeButton(R.string.no, null);
      dialog = builder.create();

      break;
   }
```

Note that most Android apps do not have a Quit option since clicking on the device's Home button essentially does the same thing.

Run your app and verify that the menu options work as expected.

Extra Challenge

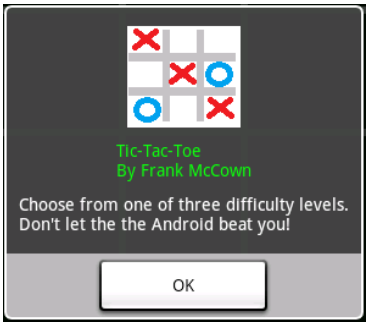
There are two extra challenges:

1. Create your own custom icon for your application. Name it `icon.png`, and place it in your `res/drawable` folder. Open your `AndroidManifest.xml` file, and alter `android:icon` to point to your new `icon.png`:


```
<application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
```

Now run your app on the emulator. Press the Home key and press the App Drawer button on the screen to list all the apps installed. This is where you will see the icon you have created for your app. Mine is shown above.

2. Create a menu option that displays an About dialog box, similar the one shown on the right, which identifies you as the programmer.



You can create your own custom dialog box by first creating a `res/layout/about_dialog.xml` file with the dialog box components (just like you did in `main.xml`). You'll need to reference an image you create from the `res/drawable` folder in your XML file. In the `onCreateDialog()` method, use a `LayoutInflater` to expand `about_dialog.xml` into a `View`, attach the `View` to an `AlertDialog.Builder`, and create the dialog. The code that does all this is shown below.

(cc) BY-NC

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
Context context = getApplicationContext();
LayoutInflater inflater = (LayoutInflater) context.getSystemService(LAYOUT_INFLATER_SERVICE);
View layout = inflater.inflate(R.layout.about_dialog, null);
builder.setView(layout);
builder.setPositiveButton("OK", null);
Dialog dialog = builder.create();
```

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