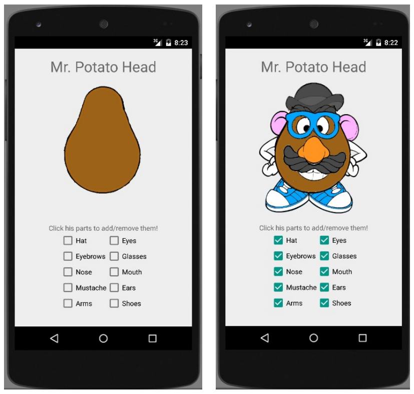
Stanford University, CS 193A
Homework 2: Layout

The purpose of this assignment is to practice more advanced graphical layouts than you did in Homework 1, as well as using more widgets/views in a graphical user interface. As with HW1, we'll suggest some ideas for programs you could write to practice layout. We will list several suggestions, but you **only need to write one app** to submit. If you have a different idea for a program you want to make that will still allow you to explore layout, please feel free to do something other than the suggestions provided.

Suggestion 1 (small): Mr. Potato Head *(thanks to Victoria Kirst for original assignment idea and images!)*

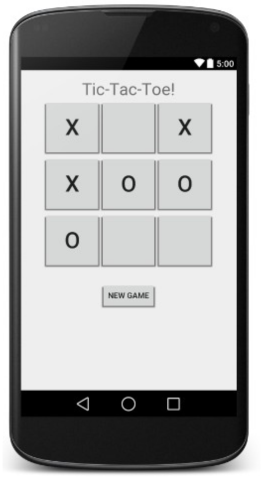


Write an app that displays a "Mr. Potato Head" toy on the screen as an `ImageView`. The toy has several accessories and body parts that can be placed on it, such as eyes, nose, mouth, ears, hat, shoes, and so on. We will provide you with image files for each body part and accessory, such as **body.png**, **ears.png**, **hat.png**, and so on. Initially your image view should display only the toy's body, but if the user checks/unchecks any of the check boxes below the toy, the corresponding body part or accessory should appear/disappear.

The way to display the various body parts is to create a separate `ImageView` for each part, and lay them out in the XML so that they are superimposed on top of each other. You can achieve this with a `RelativeLayout` in which you give every image the same position, though you should probably nest it in some other overall layout for the screen. The check boxes should align themselves into a grid of rows and columns.

You can set whether or not an image (or any other widget) is visible on the screen by setting its `android:visibility` property in the XML, and/or by calling its `setVisibility` method in your Java code. The `setVisibility` method accepts a parameter such as `View.VISIBLE` or `View.INVISIBLE`. There is also a `getVisibility` method if you need to check whether a widget is currently visible.

Suggestion 2 (medium): Tic-Tac-Toe



Write a basic game of tic-tac-toe, where two players take turns pressing buttons in a 3x3 grid to mark "X" or "O" characters on them respectively. If any player can place three of their letter in a row horizontally, vertically, or diagonally, that player wins the game.

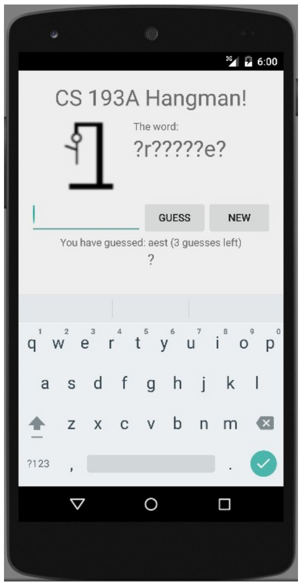
Setting up the buttons for the game is a good opportunity to practice using `GridLayout`. You'll probably want to set the buttons to have a large size so that they fill a large portion of the screen, as well as giving them a large font to make them easier to read and click.

If you want a simpler implementation, you can write your code as though two human players were playing it on the same screen; the first tap is an X move, the second is an O move, and so on. If you want more challenge, you could have the computer play as the second player. A simple strategy would be to just randomly move on any open square, but a more complex computer player would try to "block" the human player if the human has any two-in-a-rows and is one move away from winning the game.

Suggestion 3 (large): Hangman

Make a basic **Hangman** game that displays an image of a gallows and a hanging man, along with a word that the player is trying to guess. The word is chosen randomly from a provided dictionary. At all times the game displays a "**clue**" of the letters the player has guessed correctly; for example, if the word is `"apples"` and the player has guessed `e`, `k`, `p`, and `t`, the clue would be `"?pp?e?"`. The user can type single-letter guesses into an `EditText`. (The `EditText` allows the user to type multi-letter strings and non-letters; a robust game would handle such attempts gracefully, as well as other errors like trying to guess the same letter twice, etc.) You can display a message such as a `Toast` when the user guesses the word correctly or runs out of guesses and ends the game.

On the Homework page of the class web site you will find **images** named **hangman0.png** through **hangman6.png** to display when the user has 0-6 guesses remaining, respectively. Display these images using an `ImageView`. You can also download the dictionary of words and paste it into your **strings.xml** file in the **res/values** folder of your project. The XML data looks like this:



```
<string-array name="words">
    <item>the</item>
    <item>of</item>
    <item>apple</item>  ...
</string-array>
```

To access this array of words in your Java code, you'd write a line such as:

```
String[] words = getResources().getStringArray(R.array.words);
```

At right is a screenshot of our own implementation of this game. You don't have to match our app's appearance or behavior, but if you want to test our version, a demo **.APK** archive for it is available on the class web site.

Suggestion 4: Make Up Your Own

If you don't like our suggested assignment ideas or prefer to do something unique of your own, please feel free to do so. Whether you do our suggestions or your own, we'd prefer to see an app that has the following qualities:

- Your app should be set up as an **Android Studio** project, so it can easily be opened/run/graded by others.
- Your project should not always use the default names. *(For example, rather than calling your project the default name of `MyApplication` and your activity the default name of `MainActivity`, call your project something like `Hangman` and your activity something like `HangmanMainActivity`, etc.)*
- Your app should use at least **2 different layouts**. *(Examples: a `LinearLayout` with another `LinearLayout` inside it; or a `RelativeLayout` with a `GridLayout` inside it; etc.)*
- Your layout XML should adjust the **box model** properties of at least one widget or view. *(Example: setting the gravity, weight, padding, margins, etc. of a widget to adjust its appearance or spacing.)*
- Your app should respond to at least one **event**. *(Example: clicks on a button.)*
- Along with your app, please turn in a file named **`README.txt`** that contains your name and email address along with the name of your app and a very brief description of it, along with any special instructions that the user might need to know in order to use it properly (if there are any). For example:

```
Joe Student <jstudent@stanford.edu>
NumberGame 2.05 - This app shows two numbers on the screen and asks
the user to pick the larger number.  Perfect for Berkeley students!
```

As always, these assignments, as well as this class in general, are meant to be **low-stress** and fun. If you want help, please feel free to show your code to others or ask for help in our online **Piazza forum**. Feel free to make an app as simple or as complex as you like, relative to your familiarity level and time constraints. If you work on your solution for roughly 2 hours and are still not done, you can turn it in and we will award you credit.

Turnin and Grading:

Instructions for turning in this program can be found on the class web site. After programs are turned in, you will be asked to peer-evaluate another student's submission.

Your submission will be graded quickly by simply running it and evaluating its functionality. It does not need to be perfect or bug-free to receive credit. Your code will not be graded on style, but we still encourage you to follow good overall coding style for your own sake. If you want to see some good examples of proper Java coding style, consult the **Style Guide** linked from the Homework web page.