Android Strings, Arrays and StringBuilder – Hangman Project

The basic plan of hangman is to pick a word, hide it from the user and show a word of equal length made up of *s or ?s and ask the user to choose a letter. This can then be matched to a position within the real word and the placeholder character in the displayed word is replaced with that letter. If the letter is not present a penalty is built up and displayed.

The set of words is best stored in the strings.xml file within /res/values. Here the words are all of equal length. Later a difficulty setting could be used to choose words of different lengths.

```
<string-array name="words">
       <item>ASTRONOMER</item>
       <item>CAPSULATED</item>
       <item>COMESTIBLE</item>
       <item>FLATTENING</item>
       <item>IMPORTANCE</item>
       <item>LIKELIHOOD</item>
</string-array>
```

The xml array can be picked up in the Java code, a random position chosen and placed in a String and displayed in a TextView for testing.

```
TextView tvHidden = (TextView)findViewById(R.id.tSecret);
wordList = getResources().getStringArray(R.array.words);
int randPos =(int) (Math.random()* wordList.length);
tvHidden.setText(wordList[randPos]);
```

There is no point showing the actual word to the user so a new String of the same length is created by replacing the existing characters with a set of *s.

```
String solve =

secret.replaceAll(wordList[randPos], "**********");
```

An EditText can be used to grab the user input. Here it will only accept 1 character.

```
 <EditText
       android:id="@+id/eLetter"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:maxLength="1" >
```

The EditText needs an `OnKeyListener` Listener, this requires implementing `View.OnKeyListener` within the Activity. The Interface will require this method to be used.

```
public boolean onKey(View v, int keyCode, KeyEvent event){ }
```

Here the key entered will be picked up and compared to the hidden word.

```
if ((event.getAction() == KeyEvent.ACTION_DOWN)&& (keyCode ==
KeyEvent.KEYCODE_ENTER))
{
     //need the 1st char
     char letter =
     etLetter.getText().toString().toUpperCase().charAt(0);
     etLetter.setMaxLines(1);  //prevent enter adding a line
     etLetter.setText("");  //clear box
     etLetter.setSelection(0);
     //convert the secret word to an array of type char
     char []seq = secret.toCharArray();
     for(int x=0 ; x< seq.length;x++)
     {
          if(letter == seq[x])
          {
            StringBuilder temp = new StringBuilder(solve);
            temp.replace(x, x+1, String.valueOf(letter));
            solve = temp.toString();
            Toast.makeText(this, "match at " + x,
          Toast.LENGTH_LONG).show();
          }
          tvHidden.setText(solve);
     if(solve.equalsIgnoreCase(secret))
          Toast.makeText(this, "Word Found" ,
          Toast.LENGTH_LONG).show();

     }
```

There are 3 things holding the letters here.

- char is a simple scalar, it has no methods but we can convert a String to an array of type char and run through the array.
- String is the standard object for text. It has a number of methods for manipulating its contents but not a method to replace a char at a specific position. The replace method of String replaces all instances of 1 char with another in a String.
- StringBuilder is a class that allows some additional manipulation of its contents including the replace method needed above. Luckily a StringBuilder can be instantiated from a String and a StringBuilder can be converted to a String.

If a letter that is input matches a position in the hidden word then the * in that same position in the displayed word is replaced by the real letter.

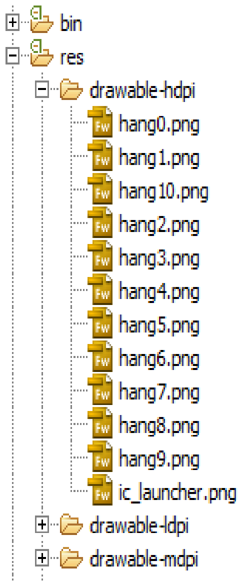
If the 2 words (secret and hidden) are the same then the user has matched the hidden word. Strings can be compared with == but the results are not reliable. A better plan is to use the `equalsIgnoreCase()` or `equals()` methods of String.

The number of wrong attempts taken to solve the hidden word can be held in an int variable.

```
boolean inWord = false;
for(int x=0 ; x< seq.length;x++)
{
     if(letter == seq[x])
     {
          StringBuilder temp = new StringBuilder(solve);
          temp.replace(x, x+1, String.valueOf(letter));
          solve = temp.toString();
          inWord = true;
     }
}
if(! inWord)
     attempts ++;
```

The Gallows

The number of attempts can be tested with an if condition and set to some limit. To improve the output the gallows will be displayed and built up. One solution is to have a set of images each for one part of the hanging.


```
bin
res
  drawable-hdpi
    hang0.png
    hang1.png
    hang10.png
    hang2.png
    hang3.png
    hang4.png
    hang5.png
    hang6.png
    hang7.png
    hang8.png
    hang9.png
    ic_launcher.png
  drawable-ldpi
  drawable-mdpi
```

The images need to be about the same size as the ImageView that will be used to display them.

Here all the png files have been given the same name except for the ending number. This will make them easier to identify in code.

An ImageBox will be used to display the images.

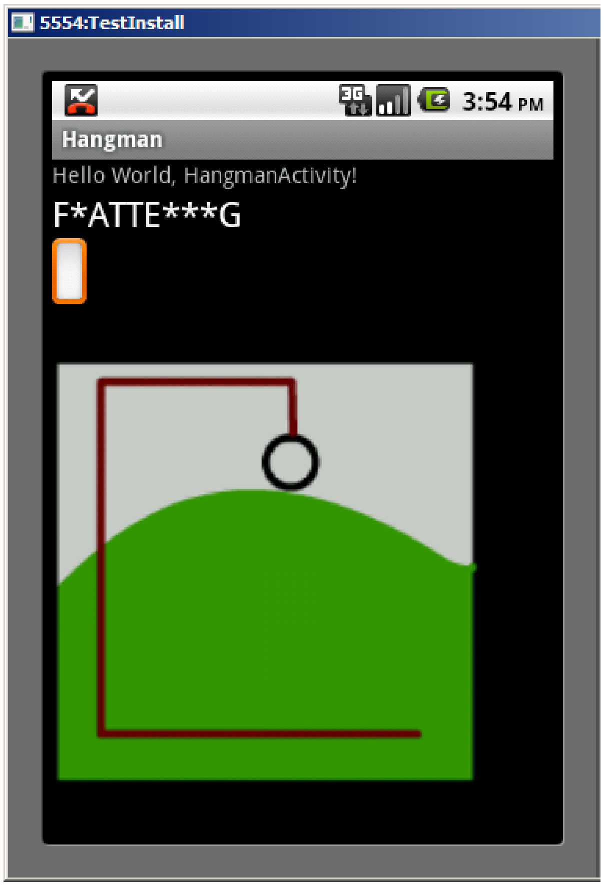
```
ivHang =
(ImageView)findViewById(R.id.imgHang);

ivHang.setImageResource(R.drawable.hang0);
```

The image will be changed when the user selects a letter that is not within the hidden word. To avoid needing to check for each image by name an array is set up and used to reference the next image to be loaded.

```
int []images =
{R.drawable.hang0,R.drawable.hang1,R.drawable.hang2,
R.drawable.hang3,R.drawable.hang4,
R.drawable.hang5,R.drawable.hang6,R.drawable.hang7,
R.drawable.hang8, R.drawable.hang9,R.drawable.hang10};
```

```
if(! inWord)
{
      attempts ++;
      if(attempts < images.length)
      ivHang.setImageResource(images[attempts]);
      else if (attempts == images.length)
            Toast.makeText(this, "Game Over",
            Toast.LENGTH_LONG).show();
}
```



Hello World, HangmanActivity!

F*ATTE***G

The code works but the EditText is inconvenient because when a letter is entered the control is cleared but backspace needs to be used to move back to the beginning of the control.

This will not occur if the `ACTION_DOWN` action is not picked up on the Listener event:

```
if (/*(event.getAction() == KeyEvent.ACTION_DOWN)&&*/ (keyCode
== KeyEvent.KEYCODE_ENTER))
```

Now there is a new problem as each entry of a letter is processed twice, in ACTION_DOWN and ACTION_UP. Even a correct entry results in 1 failed attempt.

This can be solved by only testing for a missed letter on ACTION_DOWN.

```
if((! inWord )&& event.getAction() == KeyEvent.ACTION_DOWN)
```