SOFTWARE-HARDWARE CO-DESIGN

CEN - 4214

December 11, 2009

Design Project -2

By: Nanditha Kalluraya

Email: nkallura@fau.edu
Develop and Prototype the Hangman Application

Top-down design for mobile based application development using UML, Android Mobile Platform, Java and XML

Abstract

The main objective of this project is to develop and prototype the hangman application using Sudoku application as an example. Top-down design methodology is used to create UML sequence diagrams which aids in building the pseudo code for the application in Android platform using Java and XML. This approach of creating the design helps us to build an application which is scalable, user-centric (keeping the user requirements in mind and helps us build a better user interface) and also can be easily and quickly be modified in case additional features are needed. The end-product is a beautiful Hangman game developed using the top-down design methodology. This report will give us an overview of how the Hangman application is created emulating the Sudoku Application code given, with modifications.

Introduction

Hangman is a traditional game played by at least two players using pen and paper. The inception of computers has facilitated it to be a single-player game. Nowadays there are many applications being developed both on Pc’s as well as mobile phones. Hangman is a simple and good game to be played when an individual is bored.

There are a wide number of mobile development platforms on the market, some of them being Blackberry, iphone etc. In spite of such mobile environments, Android (an Open source software toolkit) has entered the competition and is likely to survive and gain importance in the future making Android a major platform for application developers.

A good starting point to learn about this platform would be to start from basic and create small applications. Hangman proved to be a good and simple game application to start with, mapping it to the already developed Sudoku application given in “Hello Android: Introducing Google’s Mobile Development Platform, The Pragmatic Bookshelf” was a boon in disguise.
Methods

Using the sequence diagrams that could be designed for Sudoku Application through the process of reverse-engineering, we created the sequence diagrams for the Hangman Application at a very low-level of detail. I then, identified the common pieces of the code for my design purpose and created the Main-Menu and About dialog box and using then using use case and class diagrams I created the Game Screen for the application.

The tools employed for this project are UML diagrams to create the sequence diagrams, both the top-level and mid-level ones and use case diagrams using Rational Rose and also software for developing Android applications- Java Runtime Environment, Java Development Kit, Android SDK, Eclipse Integrated Development Environment. Pseudo code for the application is written, on the Android platform using Java and XML. The screen-captures of how the User-Interface for the game screen, Main-Menu screen looks like is shown in this design project. They are also placed in the Appendix.

Results

The results being, state diagrams, class diagrams pseudo code, and screen captures, of the screens displayed on the emulator can all be found in the appendix.

Discussion

The result that came out from this version of the project was a beautiful Hangman Application with its functionality fully implemented. I had the Menu-screen with four buttons for About, Exit, Continue and New Game. All these buttons are functional and When New Game is entered the user can choose the difficulty level and Category of word to play from. I did the Game screen part of the implementation using Class diagrams and use cases built by using the Hangman requirements and Specifications of my application.

This design project, I was able to implement successfully. The target in the future would be to improvise the Hangman Application such that we could kind of create a mash-up technology and use location-based services of the Android phone and find out where the game is being played.
**Use Case Diagrams**

In all the use cases for this application there is only one initiation actor as well as receiving actor and Benefitting actor, the Game User.

1. **Use case: ClickHangmanIcon**

**Description:** Game User clicks on the Hangman Icon to enter into the application.

**Assumptions:** The Game User runs it from the Devices’ Home Screen.

**Pre-condition:** Game User wanting to play the game.

**Post-condition:** Main Menu screen User Interface of the Hangman Application being displayed.

**Steps:** User brings up the User interface where he can access applications on the device.

   Touches/Clicks the Hangman icon
2. Use case: Click About Button

**Description:** User clicks on the About button to know more about the application and game.

**Assumptions:** “About” button is present

Game user wants to view “About” the application.

**Pre-condition:** Player wanting to know more about the game

**Post-condition:** A dialog box with the necessary information about the game.

**Steps:** User is in the Main-Menu screen and touches the “About” button

The system displays the dialog box with information about the hangman application.

The User views the necessary information.
3. Use case: Click Continue Button

**Description:** Game User wants to continue a previously playing game.

**Assumptions:** There is a “Continue” button in the Main-Menu screen.

Game User exited from the game sometime ago in the history. Therefore there is a previously saved game.

**Pre-condition:** Player wants to continue a game that he exited from under necessary circumstance.

**Post-condition:** Previously saved game is displayed and user gets to continue his game.

**Steps:** Game User exits presses continue button.

The system displays the previously saved game.

User continues and plays the Game.
4. Use case: Click New Game Button

**Description:** Game User wants to play a New game.

**Assumptions:** There is a “New Game” button in the Main-Menu screen.

Game User wants to play a new game.

**Pre-condition:** Player wants to play a new game.

**Post-condition:** A difficulty list in the form of dialog box is shown.

**Steps:** Game User clicks on the New Game.

He clicks on the difficulty level which is asked to choose from. That is a different use case by itself.
5. Use case: Choose Difficulty level

**Description:** Game User wants to choose a difficulty level from the available options.

**Assumptions:** There is a “difficulty level” available to choose from.

**Pre-condition:** Player wants to choose difficulty level to play the game.

**Post-condition:** User selects the difficulty and plays the game.

   The category list is shown as a array list in the form of a dialog box.

**Steps:** Game User clicks on anyone of the available options.

   The user clicks on one of the difficulty levels.

   A category list is shown to choose from.
6. Use case: Choose Category

**Description:** Game User wants to choose a category among. Four available ones.

**Assumptions:** There is a “category” list.

**Pre-condition:** Player wants to choose a category

**Post-condition:** The category chosen and the Game screen displayed.

**Steps:** Game User chooses a category from the list of options available to him.

The system uses both the difficulty level and category level values and gets a word from the selected category.

User Plays the game.
7. Use case: To Exit the Game

**Description:** Game User wants to Exit the button.

**Assumptions:** There is a “Exit” button in the Main-Menu screen.
- Game User exits through by clicking the exit button.
- Game User exits through by pressing the back button of the device.
- Game User exits through by clicking the “home” button of the device.

**Pre-condition:** Player wants to exit the game.

**Post-condition:** Player exits the game.

**Steps:** Game User exits using the exit, back, home button.
- The system saves the present state of the game activity.
8. Use case: Play Game

**Description:** Game User is in the Game screen to play the game of his chosen difficulty level and category.

**Assumptions:** There is a “Game screen” user interface for the user to interact with.

**Pre-condition:** Player wants to play the game.

**Post-condition:** Game user plays the game and either wins it or loses it.

**Steps:** Game User is in the Game Screen user interface.

- The system displays the unknown word, the screen soft key board and the gallows.
- The user touches one of the characters of the soft keyboard.
- The system displays it at its respective positions on the screen.
- The third and fourth steps continue until the hangman is drawn or all the characters are filled.
- For each win or lose a Dialog box with appropriate message is displayed.
9. Use case: Click Play Again

**Description:** Game User wants to Play a Game again but with different mode and category.

**Assumptions:**
- There is a “Play Again” button after game over condition (either Game win or Game loses).
- Game User wants to play again.

**Pre-condition:** Player wants to play a game again.

**Post-condition:** Player gets a Main menu screen again and can select his options and play the game again.

**Steps:** Game User presses “Play Again” button.

- The system displays the Main Menu screen.
- The player chooses his options and plays the game again.

![Diagram](image.png)
10. Use case: Click on the Menu Of the Device

**Description:** Game User wants to Change the default Music settings.

**Assumptions:** There is a settings option implemented in the application. The user clicks this button from the Main-Menu screen.

**Pre-condition:** Game User wants to change the music settings

**Post-condition:** He gets the settings button.

**Steps:** Game User presses the Device “Menu” button.

The settings option comes

The user clicks on it and enters the Music settings screen and either enables or disables it.

![Diagram](image-url)
11. Use case: Click Settings

**Description:** Game User wants to go to Music settings.

**Assumptions:** There is a “Continue” button in the Main-Menu screen.

Game User exited from the game sometime ago in the history. Therefore there is a previously saved game.

**Pre-condition:** Game User wants to go to Music settings either to enable/disable it.

**Post-condition:** He reaches the Music Settings activity.

**Steps:** game user clicks on settings.

Enters into the Music settings screen and enables or disables it.
12. Use case: Selects Music

**Description:** Game User wants to either listen to / cancel the Music option.

**Assumptions:** There is a Music settings present in the Settings option.

- Game User wants to enable/disable the music settings

**Pre-condition:** Game user wants to listen to/ cancel the Music option

**Post-condition:** The music is enabled/ cancelled accordingly.

**Steps:** Game User clicks either enables or disables the checkbox and the music plays/stops accordingly.

- User plays his New Game.
References


Conclusion:

We have seen how through step wise top-down design methodology we could get to design a beautiful application which satisfies the needs of the user as well as makes our thinking process and hence, coding very systematic. This top-down type of analysis would be incomplete without UML diagrams (Activity diagrams, Use cases, Sequence diagrams and Class diagrams.)

I am very happy that I got the opportunity to take this course and learn how android functions. I wasn’t familiar as to, how to use UML diagrams for coding purposes. This course taught me that, as well as many other concepts that I was not well-versed with. All in all a good job!
Appendices

Appendix-A

(Hangman Requirements Document)

1. Overview

Hangman is a simple word guessing game. Traditionally it was played by two players but, since its inception as a computer and mobile phone game it has turned into a single player (that is, the player plays against a computer AI) and it is also used as a vocabulary improving game.

A game of hangman has the following game pattern:

♠ The program selects a word from a pool of words and displays a number of underscores equal to the word length.

♠ The user/player begins guessing letters. If the player guesses a letter that is in the word, then all instances of that letter are displayed at their corresponding positions in the word. Otherwise, the guess is incorrect. Every incorrect guess results in a body part being drawn. The number of incorrect guesses allowed, depends on the difficulty level chosen by the player. An easy level allows 6 incorrect guesses, a medium level allows 7 incorrect guesses and a difficult one allows 9 incorrect guesses.

♠ The game ends when either all the letters in the word has been guessed or when the player has used his number of incorrect guesses.

2. Functionality

When the user opens the Hangman application he/she has several options: one can learn more about the application, one can start a new game, and one can continue an existing game or exit an application.

If the user chooses to start a new game, he first has to make a choice regarding the difficulty of the game (easy, medium or difficult). Once the difficulty has been chosen, the player can choose his unknown word from four different categories. The categories that are available are places, animals, flowers and general. Once the difficulty level and category is chosen, the game starts. The application randomly displays the word depending on the difficulty level and category and for each letter in the word an underscore is displayed (underscores are separated by spaces.). A series of guesses follow. The user enters a letter. If that letter is part of the word, all the appearances of that letter are displayed at their corresponding position in the word, instead of the
underscore(s). If the guess is incorrect that letter is displayed under the word to be guessed and according to the difficult level the gallows and the body part is drawn. These actions are repeated until all letters are guessed or the hangman is drawn. If all letters are guessed the user wins. If the entire body is drawn the user looses the game. If the user chooses to leave a game that was not finished and later on continue it (through the continue option in the main menu), then the saved state of the game will be displayed (including the letters guessed, the incorrect guesses, and the body parts drawn so far). The state of the game must be saved every time the user leaves the current game screen. If the user chooses the about option, information about the game should be displayed.

3. Structure
The application should have two main screens: main menu and game. The first screen, the main menu, has to contain the four possible options displayed in a horizontally linear manner: Continue, New Game, About and Exit. Above the four buttons should be a text area which contains the title of the application, Android Hangman. The layout width and height of the text that contains the main title should wrap its content. For the four buttons that contain the possible options should have a layout width that fills the entire parent layout, and a layout height that wraps the entire content.

The About screen should contain a scrollable text area. The style of the About activity should be a dialog theme.

To start a new game, first the user needs to choose the difficulty level shown in the form of an alert dialog. This dialog will contain a title and the three difficulty levels: easy, medium and difficult. Secondly, the user has to choose a particular category to play with. This will also be shown in the form of an alert dialog, which will contain a title and the four categories: places, animals, flowers and general (the general category is good to test your vocabulary skills). The game will start after choosing the difficulty level and its associated category. Each difficulty level and its associated category will have a list of words associated with it. One word, from the list corresponding to the chosen difficulty level and its category, will be randomly selected. The game screen will contain the image (according to the difficulty level chosen) that will display the gallows or the body parts, a text area that displays the word to be guessed and another text area for displaying the incorrect guesses. There will also be icons to ‘PlayAgain’, ‘Main Menu’ or’quit game’. Also there would be an image in which ‘won’(number of winning games) and ‘kills’(number of games lost) will be displayed. When the game ends an alert dialog should appear, informing the user that he/she has won/lost the game. The last saved game screen will be shown when the user chooses to continue a game. If the previous game ended, then pressing the ‘Play again’ button should have the same effect as
starting a new game but with the same difficulty level and category previously chosen. To either change the difficulty level or category, we use the ‘Menu’ which takes us to the ‘Main Menu’. The ‘quit game’ option lets us to quit in the middle of the game and has same effect as the ‘Exit’ button in the Main Menu.

4. **Mapping Hangman to Android**

In order to achieve all the above you need to use the following:

- About screen should contain a TextView. Both the width and height for the layout should wrap their content around a ScrollView. The padding of the ScrollView should be 10dip.
- Difficulty screen should be an Alert Dialog.
- Categories screen should be an AlertDialog.
- Main menu screen layout should be a TextView.
- The game screen should contain an ImageView to display the body parts, an ImageView to display the games won and lost, a TextView for the word to be guessed and another TextView for the incorrect guesses. It should also contain icons or buttons, to go back to the Main menu, to quit the game and continue with the game (play again.)
- The game over dialog should be an AlertDialog.

5. **References**


Hangman Specification Document

1. Overview

Hangman is a simple word guessing game. Traditionally it was played by two players but, since its inception as a computer and mobile phone game it has turned into a single-player (that is, the player plays against a computer AI) and it is also used as a vocabulary improving game.

A game of hangman has the following game pattern:

♣ The program selects a word from a pool of English words and displays a number of underscores equal to the word length.

♣ The user/player begins guessing letters. The word extends from left to right. If the player guesses a letter that is in the word, then all instances of that letter are displayed at their corresponding positions in the word. Otherwise, the guess is incorrect. Every incorrect guess results in a body part being drawn. The number of incorrect guesses allowed, depends on the difficulty level chosen by the player. An easy level allows 9 incorrect guesses (which means in this case the game screen will not have any image and slowly as the number of incorrect guesses build up the gallows and the body parts will be drawn), a medium level allows 7 incorrect guesses (gallows but without the noose on the game screen, so in case of an incorrect guess: noose, face, body, 2 arms and 2 legs) and a difficult one allows 6 incorrect guesses (for this level of difficulty, gallows with the noose will be on the game screen and the face, body, 2 arms and 2 legs will be drawn). The order in which the arms and legs appear in any of the above three cases is left arm/leg first and then right arm/leg.

♣ The game ends when either all the letters in the word has been guessed or when the player has used his number of incorrect guesses.

The unknown word is hidden under the underscore and as the letters are guessed the word gets revealed. When the game starts some of the letters are not already filled in the blank (no clues are provided).
2. Functionality

When the user opens the Hangman application he/she has several options: one can start a New Game, one can Continue an existing game, one can learn more About the application, one can set settings or Exit the application. If the user chooses to start a New Game, s/he first has to make a choice regarding the difficulty of the game (easy, medium or difficult). Once the difficulty has been chosen, the player can choose his unknown word from four different categories. The categories that are available are Places, Animals, Flowers and General.

![Main Menu screen of Hangman Application](image)

Fig 1. Main Menu screen of Hangman Application
Fig 2. Difficulty prompt

Fig 3. Category prompt
Once the difficulty level and category is chosen, the game starts. The application randomly displays the word depending on the difficulty level and category and for each letter in the word an underscore is displayed (underscores are separated by spaces.). A series of guesses follow. The user enters a letter. If that letter is part of the word, all the appearances of that letter are displayed at their corresponding position in the word, instead of the underscore(s). If the guess is incorrect that letter is displayed below the ‘word to be guessed’ and according to the difficult level the gallows and the body part is drawn. If a letter is repeated a Textbox will appear with a caption “word repeated”. The letters once chosen cannot be erased or changed, so the player has to place his letters with caution. These actions are repeated until all letters are guessed or the hangman is drawn. If all letters are guessed the user wins. If the entire body is drawn the user looses the game. If the user chooses to leave a game that was not finished and later on Continue it (through the Continue button in the “Main Menu” screen), then the saved state of the game will be displayed (including the letters guessed, the incorrect guesses, and the body parts drawn so far). The state of the game must be saved every time the user leaves the current game screen.

Fig4. About Dialog Box
If the user chooses the About button, information about the game should be displayed in a dialog screen. One can set the settings by pressing the menu button on the device while the “Main Menu” screen is displayed. In this settings dialog will be a choice to disable or enable music playback. This game doesn’t provide any hints.

Choosing the Exit button should take the user back to the home screen. If the user repeats any letter while guessing and touches the “keypad” or touch screen a musical sound or a dialog displaying “Word Repeated” should appear.

3. Structure

The application should have two main screens: “Main Menu” and “Game”. The first screen, the “Main Menu”, has to contain the four possible options displayed in a horizontally linear manner in the center of the screen: Continue, New Game, About and Exit. Above the four buttons should be a text area which contains the title of the application, “Android Hangman”. This screen can be portrayed in two different orientations: vertical (default) or horizontal (landscape). In the default layout, under the menu title the buttons will be displayed one above the other and centered on the screen, vertically, creating a single row of four buttons. In the landscape layout, the buttons will be positioned in two columns each of which will have two buttons. The layout width and height of the text that contains the main title should wrap its content. For the four buttons that contain the possible options should have a layout width that fills the entire parent layout, and a layout height that wraps the entire content.

Pressing the About button brings up the “About” dialog. This screen should contain a scrollable text area. The style of the “About” dialog (activity) should be a dialog theme.

To start a New Game, first the user needs to choose the difficulty level (“Difficulty” alert dialog) shown in the form of an alert dialog. This dialog will contain a title and the three difficulty levels: easy, medium and difficult. Secondly, the user has to choose a particular “Category” to play with. This will also be shown in the form of an alert dialog, which will contain a title and the four categories: Places, Animals, Flowers and General (the general category is good to test your vocabulary skills). The game will start after choosing the difficulty level and its associated category. Each difficulty level and its associated category will have a list of words associated with it.
The “Game” screen has an image of the hangman corresponding to the level of difficulty that is chosen. One word, from the list corresponding to the chosen difficulty level and its category, will be randomly selected. The initial set of words is unique to each difficulty level and its associated category. The game screen will contain a text area that displays the word to be guessed in a rectangular box which is 2 characters more than the length of the word, the word will be hidden under the text area and as the player guesses the word the word will get revealed, another text area for displaying the incorrect guesses. An area for the keypad to display letters ‘A-Z’ will be present in the game screen also. Once a letter is guessed it would get unhighlighted or would be shown in a different color, so that only valid choices are displayed and can be used. If the word is already used and still the user happens to touch it and try to use it, then an animation to shake the screen will be invoked. Also there would be an image in which ‘wins’ (number of winning games) and ‘kills’ (number of games lost) will be displayed. When the game ends an alert dialog should appear, informing the user that he/she has won/lost the game. The user can enter the letter using the device’s keypad at any time during the process of playing the game. To select an underscore the user would have to press the center button or enter key on the device. Once a letter is entered it can’t be erased or changed. There will also be buttons to ‘Play Again’, ‘Main Menu’ or ‘Exit’. The ‘Main Menu’ and ‘Play Again’ will be in the top left corner of the game screen and the ‘Exit’ button will be on the Bottom right of the game screen.
The last saved game screen will be shown when the user chooses to Continue a game. If the previous game ended, then pressing the ‘Play Again’ button should have the same effect as starting a New Game but with the same difficulty level and category that was previously chosen. To either change the difficulty level or category, we have to use the ‘Menu’ which takes us to the ‘Main Menu’. The ‘Exit’ option allows us to quit in the middle of the game. Should the application have to suspend because another application on the device is starting then the music should stop playing. The user should be able to return to the application with everything---including the cursor position---preserved and music playback should resume as long as the game’s state should be saved. While the application is running music will be playing in a loop fashion. There will be one music file that will loop while the user is in the “Main Menu” screen and a different one for the “Game” screen.

Both the “Main Menu” screen and the “Game” screen should be activities which the user can return to as long as their processes have not been killed.

4. Mapping Hangman to Android

In order to achieve all the above you need to use the following:

♣ About screen should contain a TextView. Both the width and height for the layout should wrap their content around a ScrollView. The padding of the ScrollView should be 10dip.
♣ Difficulty screen should be an AlertDialog.
♣ Categories screen should be an AlertDialog.
♣ Main menu screen layout should be a TextView.
♣ The game screen should contain an ImageView to display the body parts, an ImageView to display the games won and lost, a TextView for the word to be guessed and another TextView for the incorrect guesses. It should also contain icons or buttons, to go back to the ‘Main Menu’, to ‘Exit’ the game and continue with the game(Play Again). Also a key pad with letters (A-Z) on the game screen should be made available.
♣ The game over dialog should be an AlertDialog.
Appendix-B (Activity diagrams)

The Activity diagrams used, for the Main menu and Game screen of the Hangman Application.

The “Main Menu” activity diagram
The “Game Screen” activity diagram

1. Application chooses a random word.
2. Application displays underscores in a rectangle box and according to the level of difficulty, sets the image in the game screen. Also, the “wins and loses” image will be displayed.
3. Application displays soft key board.
4. User selects a character from soft key board.
5. Application checks for the particular character and places the character at appropriate places. If not, displays under the word to be guessed and add a body part.
6. Number of guesses is greater than the specified incorrect guesses or word is completed?
   - Yes, Application displays dialog box “You Win”.
   - No, Application displays dialog box “You Lose”.
7. Application displays on game screen “Play Again” / “Exit” or “Main Menu” option.
8. Application performs the same operation as Exit of “Main Menu”, it takes the user to the home screen after setting.
9. Play Again
10. Main Menu
11. Exit
Appendix-C (Sequence diagrams)

This section contains the Top-level and Mid-level sequence diagrams of the Hangman Application.

Top-Level Sequence Diagrams

1. Sequence Diagram01 – Start Application:

![Sequence Diagram](image-url)
Sequence Diagram02 – Exit Game:

User > : HangmanUI > : EventHandler > : OperatingSystem

CurrentUI: Android Hangman Main Menu

1: press(Exit)
2: closeApplication(Game)
3: end(game)
3. Sequence Diagram

- **About:**

**Current UI: Android Hangman Main Menu**

1: press(About)

2: open("About" prompt)

3: draw("About" prompt)

4: display("About" prompt)

5: press(Back Button)

6: display("Main Menu" screen)

7: draw("Main Menu" screen)

8: display("Main Menu" screen)
4. Sequence Diagram

New Game:

User
: HangmanUI
: EventHandler
: GameDB

Current UI: ANDroid Hangman Main Menu

1: press(New Game)
2: open(Difficulty prompt)
3: draw(Difficulty prompt)
4: display(Difficulty prompt)
5: press(easy)
6: open(Category prompt)
7: draw(Category prompt)
8: display(Category prompt)
9 - 13 Same for medium and hard difficulty
10: open(Hangman board)
11: select(Hangman word)
12: draw(Hangman board)
13: display(Hangman board)
5. Sequence Diagram 05—Continue:

Current UI: Android Hangman Main Menu

1: press(Continue)

2: load(game)

3: open(game)

4: draw(game)

5: display(game)

Game ON

Game OFF

Continue Unhighlighted
6. Sequence Diagram

06 – Play Game:

Current UI: Android Hangman Game Screen

1: select(letter)

2: check(letter present)

3: Yes(letter present)

4: No (appropriately)

5: No

6: display(below) and add(body part)

7: Update(HangmanBoard)
7. Sequence Diagram07 – Pressing “Back” Button:

Middle-Level Sequence Diagrams (For the MainMenu screen, About dialog and exit)

The middle level sequence diagrams for the MainMenu screen activity, the About activity and pressing the exit_button is shown. The sequence diagrams for the Game screen and continue_button will be shown in the future Design-reports.
1. Sequence: Open Hangman and Display the Main Menu

- User
  - open(Hangman Application)
  - onFail()
  - onSplash()
  - onStart()
  - onStart()
  - onResum()
Sequence: Hitting the Exit Button from the Main Menu of Hangman. No music Enabled in Settings.
Sequence: Opening and Closing the About Dialog, Assume Music Enabled in Settings

1. true/ void button
2. new
3. onButton()
4. startActivity(intent)
5. onPause()
6. onBack()
7. drawAndDisplay(layout)
8. onRestart()
9. onRestore()
10. pass/place a background
11. onResume()
12. onDestroy()
Appendix-D *(Pseudo Code)*

This Appendix includes the screen captures of the main-menu screen and the About dialog box in both the portrait and landscape mode along with the code (Java and XML). In this section there is less description and many screen-captures placed, because it is said that “A picture can speak better than a thousand words.”

**Icon screen-capture:**
Main Menu screen in Portrait mode:
About screen in portrait mode:

Hangman is a simple word guessing game. It can also be used as a vocabulary improving game. The game is played as follows: An hidden word is given and the player has to guess the word before the body of a man is drawn and completed. In this game there is a twist, depending on the difficulty level the number of incorrect guesses are allowed.
Main Menu screen in Landscape mode:

![Main Menu screen in Landscape mode]

About screen in Landscape mode:

![About screen in Landscape mode]
Game Screen:

This is how Game Screen looks like. This is a screen capture of the dialog-box that appears when the user clicks on NewGame button from the main screen. The user is prompted to select a difficulty level to play from.

The next screen capture prompts the user to select from a particular category.

The screen capture of the Game Screen of the Hangman application with chosen category and difficulty level is shown below. I had chosen “medium level of difficulty” and category “Animals”.

© Nanditha Kalluraya. Florida Atlantic University, Boca Raton, FL.
When the Game is “Over” it displays a dialog box stating whether the Game is won or lost. My next two screen captures depict the same respectively.
This Game screen can be used in landscape mode as well. I’ll show one screen capture of the same.

The next screen capture shows how if we quit the game in between due to an emergency how we can continue the game.
The screen capture above shows the game being half-played, Suddenly the user exits the application or the Game when the user comes back and wants to play the game again. The game can be continued. I pressed the back-button of the device. In the next screen shot the saved state of the previous game should show-up.
When I pressed continue the game which was being played was saved and when continue is pressed it is retrieved for use.

My next few screen captures shows the usage of “settings option” which can be brought up by touching the “Menu” option of the mobile hand set when in the Main Menu screen of the application as shown below. I have implemented only settings for “Music” only.

![Android Hangman Menu Screen](image)

When “settings...” is touched the following screen comes up.
The “Music option” can be selected or deselected in the checkbox. By default it is selected. Similarly Hints option can be provided. I had provided the category of the word and some prefilled characters so I thought not to provide the hints in the settings option.

about.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ScrollView
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:padding="10dp" android:background="#6495ED">
 <TextView
 android:id="@+id/about_content"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/about_text" android:textColor="#000000"/>
</ScrollView>
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@color/background"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <ImageView
            android:id="@+id/icon"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:adjustViewBounds="true"
            android:src="@drawable/icon"
            android:maxHeight="100dip"
            android:maxWidth="100dip"/>
        <TextView
            android:text="@string/main_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginBottom="25dip"
            android:textSize="22.5sp"
            android:textColor="#8B2323"
            android:textStyle="bold"/>
        <Button
            android:id="@+id/continue_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/continue_label"
            android:textColor="#8B2323"
            android:textStyle="bold"/>
        <Button
            android:id="@+id/new_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/new_game_label"
            android:textColor="#8B2323"
            android:textStyle="bold"/>
    </LinearLayout>
</LinearLayout>
main.xml (Landscape mode)

<?xml version="1.0" encoding="UTF-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:background="@color/background"
android:layout_height="fill_parent"
android:layout_width="fill_parent"
android:padding="15dip"
android:orientation="horizontal">
  <LinearLayout android:orientation="vertical"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:layout_gravity="center"
android:paddingLeft="20dip"
android:paddingRight="20dip">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:adjustViewBounds="true"
        android:src="@drawable/icon"
        android:maxHeight="100dip"
        android:maxWidth="100dip"
        android:layout_gravity="center" />
    <TextView
        android:text="@string/main_title"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_gravity="center" />
  </LinearLayout>
</LinearLayout>

android:id="@+id/about_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/about_label"
android:textColor="#8B2323"
android:textStyle="bold" />

<Button
android:id="@+id/exit_button"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/exit_label"
android:textColor="#8B2323"
android:textStyle="bold" />
</LinearLayout>
</LinearLayout>
android:layout_marginBottom="15dip"
android:textSize="22.5sp"
android:textColor="#8B2323" />

<TableLayout
android:layout_height="wrap_content"
android:layout_width="wrap_content"
android:layout_gravity="center"
android:stretchColumns="*">
  <TableRow>
    <Button
      android:id="@+id/continue_button"
      android:text="@string/continue_label"
      android:textColor="#8B2323" android:textStyle="bold"/>
    <Button
      android:id="@+id/new_button"
      android:text="@string/new_game_label"
      android:textColor="#8B2323" android:textStyle="bold"/>
  </TableRow>
  <TableRow>
    <Button
      android:id="@+id/about_button"
      android:text="@string/about_label"
      android:textColor="#8B2323" android:textStyle="bold"/>
    <Button
      android:id="@+id/exit_button"
      android:text="@string/exit_label"
      android:textColor="#8B2323" android:textStyle="bold"/>
  </TableRow>
</TableLayout>
</LinearLayout>

Code for the Game Class

package org.kalluraya.Hangman;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.util.Log;
import java.util.Random;
import org.kalluraya.Hangman.WordList;
import org.kalluraya.Hangman.Mode;
public class Game extends Activity implements OnClickListener {

    /** Called when the activity is first created. */
    private static final String TAG = "Hangman";

    public static final String KEY_MODE = "org.kalluraya.Hangman.mode";
    public static final String KEY_CAT = "org.kalluraya.Hangman.cat";
    public static final String KEY_CONT = "org.kalluraya.Hangman.continue";

    public static final int DIFFICULTY_EASY = 0;
    public static final int DIFFICULTY_MEDIUM = 1;
    public static final int DIFFICULTY_HARD = 2;

    private static final int LENGTH = 9;

    public static final int CATEGORY_COUNTRIES = 0;
    public static final int CATEGORY_ANIMALS = 1;
    public static final int CATEGORY_FLOWERS = 2;
    public static final int CATEGORY_GREWORDS = 3;

    private String currentWord="";
    private char[] correctGuess = new char[LENGTH];
    private char[] incorrectGuess = new char[LENGTH];
    private int preFill = 0;
    private int numRetries = 0;
    private int currentRetry=0;
    private int numCorrect=0;
    private int[] hangmanSteps;
    private int mode=0;
private int category = 0;
private boolean gameOver = false;
private int wordMap[] = new int[LENGTH];
private int keyMap[] = new int[26];

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.gamescreen);
    Log.d(TAG, "onCreate");

    wordMap[0] = R.id.h1;
    wordMap[1] = R.id.h2;
    wordMap[2] = R.id.h3;
    wordMap[4] = R.id.h5;
    wordMap[6] = R.id.h7;
    wordMap[7] = R.id.h8;
    wordMap[8] = R.id.h9;
    keyMap[0] = R.id.k1;
    keyMap[1] = R.id.k2;
    keyMap[2] = R.id.k3;
    keyMap[3] = R.id.k4;
    keyMap[4] = R.id.k5;
    keyMap[5] = R.id.k6;
    keyMap[6] = R.id.k7;
    keyMap[7] = R.id.k8;
    keyMap[8] = R.id.k9;
    keyMap[9] = R.id.k10;
    keyMap[10] = R.id.k11;
    keyMap[12] = R.id.k13;
    keyMap[13] = R.id.k14;
    keyMap[14] = R.id.k15;
    keyMap[15] = R.id.k16;
    keyMap[16] = R.id.k17;
    keyMap[17] = R.id.k18;
    keyMap[18] = R.id.k19;
    keyMap[19] = R.id.k20;
    keyMap[20] = R.id.k21;
    keyMap[21] = R.id.k22;
    keyMap[22] = R.id.k23;
    keyMap[23] = R.id.k24;
    keyMap[24] = R.id.k25;
    keyMap[25] = R.id.k26;
View kButton = findViewById(R.id.k1); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k2); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k3); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k4); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k5); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k6); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k7); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k8); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k9); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k10); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k11); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k12); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k13); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k14); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k15); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k16); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k17); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k18); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k19); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k20); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k21); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k22); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k23); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k24); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k25); kButton.setOnClickListener(this);
kButton = findViewById(R.id.k26); kButton.setOnClickListener(this);

Log.d(TAG, "The CONTINUE status is " + getIntent().getStringExtra(Game.KEY_CONT) );

if ( getIntent().getStringExtra(Game.KEY_CONT) != null ) {
currentWord = getPreferences(0).getString("selected","junk");
String tmp1 = getPreferences(0).getString("correctGuess","junk");
correctGuess = tmp1.toCharArray();
String tmp2 = getPreferences(0).getString("incorrectGuess","junk");
incorrectGuess = tmp2.toCharArray();
mode = getPreferences(0).getInt("diffMode",0);

Mode diffLevel = new Mode();
numRetries = diffLevel.numRetries(mode);
hangmanSteps = diffLevel.hangmanSteps(mode);
int cwlength = currentWord.length();

Log.d("Game","The selected word is "+currentWord+ " and length is "+ cwlength);

//For handling words with length less than 9.. eg India
for(int i= cwlength; i<9; i++){
    Button hButton = (Button) findViewById( wordMap[i] );
    hButton.setVisibility(View.INVISIBLE);
}

//Now process the correct guesses from the saved instance;
int cglength = getPreferences(0).getInt("cglength",0);
int icglength = getPreferences(0).getInt("icglength",0);
Log.d("Game","correct is" + cglength + " incorrect is" +icglength);

for(int i=0; i < cglength; i++){
    int kn = (int)correctGuess[i] - 'A';

    processKey(correctGuess[i], keyMap[ kn ] );
}

//Now process the incorrect guesses from the saved instance
for(int i=0; i < icglength; i++){
int kn = (int)incorrectGuess[i] - 'A';

processKey(incorrectGuess[i], keyMap[kn]);
}
}

else if ( savedInstanceState != null ) {
    currentWord = savedInstanceState.getString("selected");
    String tmp1 = savedInstanceState.getString("correctGuess");
    correctGuess = tmp1.toCharArray();
    String tmp2 = savedInstanceState.getString("incorrectGuess");
    incorrectGuess = tmp2.toCharArray();
    mode = savedInstanceState.getInt("diffMode");

    Mode diffLevel = new Mode();
    numRetries = diffLevel.numRetries(mode);
    hangmanSteps = diffLevel.hangmanSteps(mode);

    int cwlength = currentWord.length();
    Log.d("Game","The selected word is "+currentWord+ " and length is " +
          cwlength);
    //For handling words with length less than 9.. eg India
    for(int i= cwlength; i<9; i++){
        Button hButton = (Button) findViewById(wordMap[i]);
        hButton.setVisibility(View.INVISIBLE);
    }

    //Now process the correct guesses from the saved instance;
    int cglength = savedInstanceState.getInt("cglength");
    int icglength = savedInstanceState.getInt("icglength");
    Log.d("Game","correct is" + cglength + " incorrect is" +icglength);

    for(int i=0; i < cglength; i++){
        int kn = (int)correctGuess[i] - 'A';
Log.d("GAME", "The char is "+ correctGuess[i] +" and the int value is "+ kn );

processKey(correctGuess[i], keyMap[kn]);
}

//Now process the incorrect guesses from the saved instance
for(int i=0; i < icglength; i++){
    int kn = (int)incorrectGuess[i] - 'A';
    Log.d("GAME", "The char is "+ incorrectGuess[i] +" and the int value is "+ kn );
    processKey(incorrectGuess[i], keyMap[kn]);
}

}else{

    mode = Integer.parseInt( getIntent().getStringExtra(Game.KEY_MODE) );
    Log.d(TAG,"The difficulty is " + mode);
    category = Integer.parseInt( getIntent().getStringExtra(Game.KEY_CAT) );
    Log.d(TAG,"The category is " + category);

    WordList wl = new WordList();
    Mode diffLevel = new Mode();

    currentWord = wl.getword(this, category);
    int cwlength = currentWord.length();
    Log.d("Game","The selected word is "+currentWord+" and length is "+cwlength);

    //For handling words with length less than 9.. eg India
    for(int i= cwlength; i<9; i++){
        Button hButton = (Button) findViewById( wordMap[i] );
        hButton.setVisibility(View.INVISIBLE);
    }

    numRetries = diffLevel.numRetries(mode);
hangmanSteps = diffLevel.hangmanSteps(mode);

preFill = diffLevel.getPreFill(mode);
Log.d("GAME", "prefill is"+ preFill );

if(preFill > 0){
    Random randIndex = new Random();
    for(int i=0; i<preFill; i++){
        int tmp = randIndex.nextInt(currentWord.length()) ;
        char c = currentWord.charAt(tmp);
        int kn = (int)c - 'A' ;
        Log.d("GAME", "The char is "+ c + " and the int value is "+ kn );
        processKey(c, keyMap[ kn ] );
    }
}

@Override

protected void onResume() {
    super.onResume();

    Log.d("Game","came to onResume");
}

protected void onPause(){
super.onPause();
Log.d("Game","came to onPause");
if(gameOver == false){
    getPreferences(0).edit().putString("selected", currentWord).commit();
    String cg = new String(correctGuess);
    String icg = new String(incorrectGuess);
    Log.d("Game","saving state " + cg + "incorrect " + icg);
    getPreferences(0).edit().putString("correctGuess", cg).commit();
    getPreferences(0).edit().putString("incorrectGuess", icg).commit();
    getPreferences(0).edit().putInt("diffMode", mode).commit();
    getPreferences(0).edit().putInt("cglength", numCorrect).commit();
    getPreferences(0).edit().putInt("icglength", currentRetry).commit();
}
}

@Override
protected void onSaveInstanceState(Bundle outState) {
    // Save away the original text, so we still have it if the activity
    // needs to be killed while paused.
    if(gameOver == false){
        outState.putString("selected", currentWord);
        String cg = new String(correctGuess);
        String icg = new String(incorrectGuess);
        Log.d("Game","saving state " + cg + "incorrect " + icg);
        outState.putString("correctGuess", cg);
        outState.putString("incorrectGuess", icg);
        outState.putInt("diffMode", mode);
        outState.putInt("cglength", numCorrect);
        outState.putInt("icglength", currentRetry);
    }
}
public void onClick(View v) {
    switch (v.getId()) {
    case R.id.k1 : processKey('A', R.id.k1); break;
    case R.id.k2 : processKey('B', R.id.k2); break;
    case R.id.k3 : processKey('C', R.id.k3); break;
    case R.id.k4 : processKey('D', R.id.k4); break;
    case R.id.k5 : processKey('E', R.id.k5); break;
    case R.id.k6 : processKey('F', R.id.k6); break;
    case R.id.k7 : processKey('G', R.id.k7); break;
    case R.id.k8 : processKey('H', R.id.k8); break;
    case R.id.k9 : processKey('I', R.id.k9); break;
    case R.id.k10 : processKey('J', R.id.k10); break;
    case R.id.k11 : processKey('K', R.id.k11); break;
    case R.id.k12 : processKey('L', R.id.k12); break;
    case R.id.k13 : processKey('M', R.id.k13); break;
    case R.id.k14 : processKey('N', R.id.k14); break;
    case R.id.k15 : processKey('O', R.id.k15); break;
    case R.id.k16 : processKey('P', R.id.k16); break;
    case R.id.k17 : processKey('Q', R.id.k17); break;
    case R.id.k18 : processKey('R', R.id.k18); break;
    case R.id.k19 : processKey('S', R.id.k19); break;
    case R.id.k20 : processKey('T', R.id.k20); break;
    case R.id.k21 : processKey('U', R.id.k21); break;
    case R.id.k22 : processKey('V', R.id.k22); break;
    case R.id.k23 : processKey('W', R.id.k23); break;
    case R.id.k24 : processKey('X', R.id.k24); break;
    case R.id.k25 : processKey('Y', R.id.k25); break;
    case R.id.k26 : processKey('Z', R.id.k26); break;
    }
}
public void processKey(char k, int kid) {
    Log.d("Hello", "pressed key is " + Character.toString(k));
    View kButton = findViewById(kid);
    if (kButton.getVisibility() == View.INVISIBLE) return;

    boolean found = false;
    for (int j = 0; j < currentWord.length(); j++)
        if (currentWord.charAt(j) == k) { //if guess matches
            Button hButton = (Button) findViewById(wordMap[j]);
            hButton.setText(Character.toString(k));
            found = true;
            correctGuess[numCorrect] = k;
            numCorrect++;
        }
}
if(numCorrect == currentWord.length()){
    //Dialog box for game win message
    gameOver = true;
    AlertDialog.Builder builder= new AlertDialog.Builder(this);
    builder.setTitle("CONGRATULATIONS!")
    .setMessage("You won the Game")
    .setPositiveButton(R.string.playagain_label,new 
    DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            Intent intent = new Intent( Game.this, Hangman.class);
            startActivity(intent);
        }
    })
    .setIcon(R.drawable.iconblue)
    .show();
}

if(found == false){
    incorrectGuess[currentRetry] = k;
    ImageView iv = (ImageView) findViewById( R.id.himage);
    iv.setImageResource( hangmanSteps[currentRetry] );
    currentRetry++;
    if(currentRetry >= numRetries){
        //Dialog box for game over message
        AlertDialog.Builder builder1= new AlertDialog.Builder(this);
        builder1.setTitle("Oh! Oh! YOU LOSE")
        .setMessage("The word was " + currentWord)
.setPositiveButton(R.string.playagain_label, new
    DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {

            Intent intent = new Intent(Game.this, Hangman.class);
            startActivity(intent);
        }
    }
    .setIcon(R.drawable.iconblue)
    .show();
}
}

kButton.setVisibility(View.INVISIBLE);

public boolean onKeyDown(int keyCode, KeyEvent event) {
    Log.d("Hello", "onKeyDown: keycode=" + keyCode + ", event=" + event);

    switch (keyCode) {
    case KeyEvent.KEYCODE_A:  processKey('A', R.id.k1); break;
    case KeyEvent.KEYCODE_B : processKey('B', R.id.k2); break;
    case KeyEvent.KEYCODE_C : processKey('C', R.id.k3); break;
    case KeyEvent.KEYCODE_D : processKey('D', R.id.k4); break;
    case KeyEvent.KEYCODE_E : processKey('E', R.id.k5); break;
    case KeyEvent.KEYCODE_F : processKey('F', R.id.k6); break;
    case KeyEvent.KEYCODE_G : processKey('G', R.id.k7); break;
    case KeyEvent.KEYCODE_H : processKey('H', R.id.k8); break;
    case KeyEvent.KEYCODE_I : processKey('I', R.id.k9); break;
    case KeyEvent.KEYCODE_J : processKey('J', R.id.k10); break;
    case KeyEvent.KEYCODE_K : processKey('K', R.id.k11); break;
    case KeyEvent.KEYCODE_L : processKey('L', R.id.k12); break;
    case KeyEvent.KEYCODE_M : processKey('M', R.id.k13); break;
    }
case KeyEvent.KEYCODE_N : processKey('N', R.id.k14); break;
case KeyEvent.KEYCODE_O:  processKey('O', R.id.k15); break;
case KeyEvent.KEYCODE_P:  processKey('P', R.id.k16); break;
case KeyEvent.KEYCODE_Q : processKey('Q', R.id.k17); break;
case KeyEvent.KEYCODE_R:  processKey('R', R.id.k18); break;
case KeyEvent.KEYCODE_S : processKey('S', R.id.k19); break;
case KeyEvent.KEYCODE_T : processKey('T', R.id.k20); break;
case KeyEvent.KEYCODE_U : processKey('U', R.id.k21); break;
case KeyEvent.KEYCODE_V : processKey('V', R.id.k22); break;
case KeyEvent.KEYCODE_W:  processKey('W', R.id.k23); break;
case KeyEvent.KEYCODE_X : processKey('X', R.id.k24); break;
case KeyEvent.KEYCODE_Y : processKey('Y', R.id.k25); break;
case KeyEvent.KEYCODE_Z : processKey('Z', R.id.k26); break;
default:
    return super.onKeyDown(keyCode, event);
} return true;
}
Using this class diagram as a model, I developed the code for the Hangman Application. This is just a foundation or a starting point for the design. It might have developed into something better when I did the actual coding.

The code for some of the other classes used in the application are shown below.
package org.kalluraya.Hangman;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;

public class Hangman extends Activity implements OnClickListener {

    private static final String TAG = "Hangman";
    private int mode=0;
    private int playedOnce =0;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        View continueButton = findViewById(R.id.continue_button);
        continueButton.setOnClickListener(this);
        View newButton = findViewById(R.id.new_button);
        newButton.setOnClickListener(this);
        View aboutButton = findViewById(R.id.about_button);
    }

    // Set up click listeners for all the buttons
    View continueButton = findViewById(R.id.continue_button);
    continueButton.setOnClickListener(this);
    View newButton = findViewById(R.id.new_button);
    newButton.setOnClickListener(this);
    View aboutButton = findViewById(R.id.about_button);
aboutButton.setOnClickListener(this);

View exitButton = findViewById(R.id.exit_button);
exitButton.setOnClickListener(this);

@Override
protected void onResume() {
    super.onResume();
    Music.play(this, R.raw.main);
}

@Override
protected void onPause() {
    super.onPause();
    Music.stop(this);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Log.d(TAG, "Sudoku -> onOptionsItemSelected(...)"));
    switch (item.getItemId()) {
    case R.id.settings:
startActivity(new Intent(this, Settings.class));

return true;

// More items go here (if any) ...
}

return false;
}

public void onClick(View v) {

switch (v.getId()) {

case R.id.continue_button:
    if(playedOnce ==1) {startGame(99);} else
    {playedOnce=1; openNewGameDialog();};
    break;

case R.id.about_button:
    Intent i = new Intent(this, About.class);
    startActivity(i);
    break;

case R.id.new_button:
    playedOnce=1;
    openNewGameDialog();
    break;

case R.id.exit_button:
    finish();
    break;
}
}

private void openNewGameDialog() {

    new AlertDialog.Builder(this)
setTitle(R.string.new_game_title)
.setItems(R.array.difficulty,
new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialoginterface, int i) {
        showCategoryDialog(i);
    }
});
.show();

private void showCategoryDialog(int selected_mode) {
    mode = selected_mode;
    new AlertDialog.Builder(this)
        .setTitle(R.string.new_title)
        .setItems(R.array.category,
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialoginterface, int j) {
                    startGame(j);
                }
            })
            .show();
}

/** Start a new game with the given category level */
private void startGame(int cat) {
    Log.d(TAG, "clicked on " + cat);
    Intent intent = new Intent(Hangman.this, Game.class);
    //If coming from continue button
if(cat != 99){
    intent.putExtra(Game.KEY_MODE,
    Integer.toString(mode));
    intent.putExtra(Game.KEY_CAT,
    Integer.toString(cat));
} else {
    intent.putExtra(Game.KEY_CONT, "1");
}
startActivity(intent);

Mode.java

package org.kalluraya.Hangman;

public class Mode {

    public int getPreFill(int type){
        int numfill=0;
        if(type==0){numfill=2;};
        if(type==1){numfill=1;};
        if(type==2){numfill=0;};
        return numfill;
    }

    public int numRetries(int type){
        int icAllowed=0;
        if(type==0){icAllowed=7;};
        if(type==1){icAllowed=6;};
        if(type==2){icAllowed=5;};
        return icAllowed;
    }

    /*
     * For different modes of complexity, we show different images
     * at each incorrect guess. The images that will be shown at each
     * incorrect guess is stored as an array, and that is returned below.
     *
     */
    public int[] hangmanSteps(int type){
        int[] hArr = new int[9];
        if(type ==0){
            hArr[0] = R.drawable.e2;
```java
public String getword(Context context, int type)
{
    String [] catArr ={};
    if(type==0){
        catArr = context.getResources().getStringArray(R.array.category1);
    }
    if(type==1){
        hArr[0] = R.drawable.e2;
        hArr[1] = R.drawable.e3;
    }
    return hArr;
}
}
```

WordList.java
package org.kalluraya.Hangman;
import java.util.Random;
import android.content.Context;

public class WordList {
    public String getword(Context context, int type)
    {
        String [] catArr ={};
        if(type==0){
            catArr = context.getResources().getStringArray(R.array.category1);
        }
        if(type==1){
            hArr[0] = R.drawable.e3;
        }
    }
    return hArr;
}
catArr = context.getResources().getStringArray(R.array.category2);

} 

if(type==2){ 
    catArr = context.getResources().getStringArray(R.array.category3);
} 

if(type==3){
    catArr = context.getResources().getStringArray(R.array.category4);
}

Random randIndex = new Random();
return catArr[ randIndex.nextInt(catArr.length) ].toUpperCase();