Final project report

J.D.D. Schefold
10421092

May 2015

1 Introduction

In this report a brief but thorough overview of the Android app Ghost, as created by Joris Schefold, will be presented. All screens and classes will be described as well as the underlying connections between them. A schematic overview of the app can be seen in figure 1. Per screen a short overview of the functionality will be given and a few of the most important functions will be explained in greater detail.

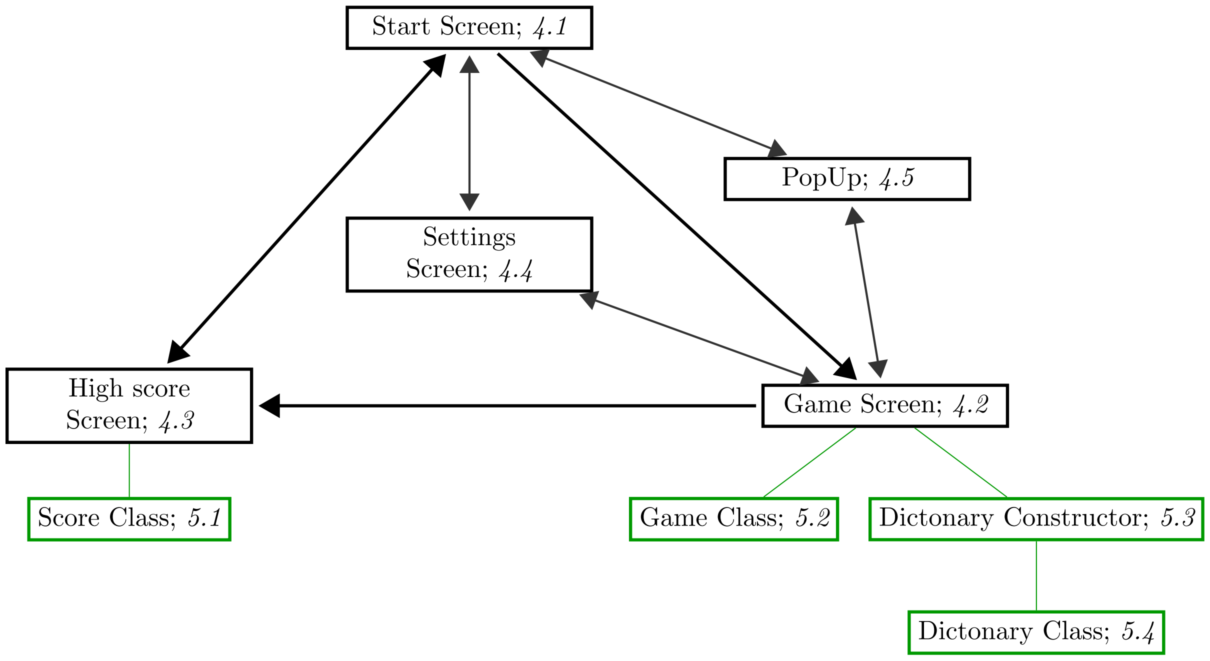

Figure 1: Schematics overview of the different screens (in black) and the classes (in green) that are used in the screens. The accessibility relations between the screens are shown by the arrows. A detailed description of every file is found in the section with number next to the screen.

2 Saving data

All the data used in this app is saved in three shared preferences:

gameDefaults
In game defaults the Id of the preferred language is saved. It also contains a string of all the previously chosen names separated by a semi-collon (;).

gameHighScores
In the high score shared preferences both the number of played games per player as well as the number of games won is saved.

gameStateStorage
Here all the information required to recreate the game is saved. It also contains a boolean to check whether the game was destroyed (and thus needs to be recreated).

3 Naming

In the app the following naming conventions are used:

- All function and variable names are in camelCase.
- All java files have CapitalizedCamelCase.
- All xml files implement snake_case

4 Activities/Screens

4.1 Start Screen

The start screen is the default screen that is loaded on starting the application. In this screen the user has the option to go to settings, to the high score screen, to start a new game and to input his/her name. To input a name the user can either enter the name in an EditText or select a previously chosen name from the drop down menu.
This possibility is created by placing an EditText on top of a spinner. To have the EditText change when a name is selected in the spinner the onOptionsItemSelected function overrides the EditText.

4.2 Game Screen

The game activity is started from the start screen. The names of the players are passed on by putting them in the Intent. The screen contains a button, two TextViews for the player names, a textView for the word and an EditText where the players can input a letter. At the start of the activity the player names are set and both a game class and a dictionary class are created.
In the drop down menu the players have the option to go to the settings screen or to restart the game. Most of the actual gameplay is handled by three OnClickListeners, the first one

is used to confirm the input, the second is used to go to the next round, and the third one is used to go to the high score screen.

OnClickListener listeners
The first OnClickListener, *confirmInput*, is used every time a player tries to input a character. The first thing it does is get the input from the edit text. It then checks if the input is a valid character (a letter) and acts accordingly.
If a player lost a round but not the game the second OnClickListener, called *nextRound*, is set to the button. This OnClickListener calls the game class and and, when pressed, sets the confirmInput listener on the Button.
If a player lost the round and the game the result is saved and a third OnClickListener, called *highScores*, is set on the button. This final listener closes the activity and moves to the high score screen.

Safe score
This function tries to load the previous scores of a player and update them, if this fails a new player score record for this player is created and the player is added to the list of previous players. This means that a name doesn't appear in the list of previous names in the start or settings screen until after a game has been completed.

4.3 High Score Screen

On start of this activity all the previous scores are loaded and for every person who ever played the game an instance of the score class is created. The resulting ArrayList of scores is sorted using a custom ranking system (your score is: $\sqrt{\#gamesplayed} \times win\ percentage$). Then the scores are added to a tableLayout row by row.

4.4 Settings Screen

This screen is accessible from both the high score screen and the game screen. In this screen it is possible to change the player names, the default language and to clear the score history. Just as in the start screen the player names are EditTexts superimposed above a spinner. The language choice is also done in the form of a spinner. The default value of the language spinner is the current selected default language.

4.5 popUp Screen

This activity consists of a screen with a message and a button to close the activity. The background of the activity is transparent which makes is possible to see the activity that called the popUp.

5 Classes

5.1 Score Class

To hold the scores and sort them a score class was made. Each instance of this class contains the number of games played, number of games won, the win percentage and the player name. This class also incorporates a custom Comparator. This comparator allows the players to be sorted according to a special score as detailed in the high score screen section.

5.2 Game Class

The game logic as well as the number of letters per player, the active player and the dictionary are contained in the game class. This class is also responsible for resetting the values when the next round starts. It has two different construction methods, one for starting a new game and one for recreating a game that was forcefully closed.

5.3 Dictionary Constructor

This class can open a file (by file id) and read it line by line. It then puts each line, if it doesn't contain any symbols, in a new HashSet. With this HashSet a new dictionary class is created and returned.

5.4 Dictionary Class

This class contains words in a HashSet and has multiple build-in operations on this set, most noticeably the function **deepCloneHashSet**. This function creates a new HashSet and adds the words from the dictionary to the new HashSet. At all times there are thus two HashSets contained in the dictionary, the original and an extra one that can be filtered without loss of information.