

# Swipe A Selfie

Technical University of Denmark  
DTU Compute

02128 - Softwareproject  
Gruppe 4

Anna Ølgaard Nielsen - s144437      Martin Dariush R. Hansen - s144459  
Per Lange Laursen - s144456      Van Anh Thi Trinh - s144449

23 June 2015

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Gameplay</b>	<b>3</b>
<b>3</b>	<b>Analysis</b>	<b>3</b>
<b>4</b>	<b>Design</b>	<b>3</b>
<b>5</b>	<b>Implementation</b>	<b>4</b>
<b>6</b>	<b>Testing</b>	<b>7</b>
<b>7</b>	<b>Development process and discussion</b>	<b>7</b>
<b>8</b>	<b>Responsibilities</b>	<b>7</b>
<b>9</b>	<b>Sources</b>	<b>8</b>

## 1 Introduction

"Swipe a Selfie" is a card game that tests the speed of your reaction. The main purpose of the game is to match two photos shown on the screen to gain points. The application is thereby for entertainment and can be played by a person of any age. This report will briefly explain how the application is designed, implemented and tested.

## 2 Gameplay

The essence of the application is to match two cards, a reference card and a matching card. The user has to slide up to skip when the two cards don't match, and slide down to keep two matching cards. If the bonus card or "Rush Hour" card appears, the user has to double-tap on the card. The bonus card gives five extra points, and the "Rush Hour" card gives a 5 second "Rush Hour"-round.

## 3 Analysis

The game's main functionality consists of gestures, camera interaction, score, timer and image comparison. The specific functionalities and game features are shown below in the MoSCoW table.

Must:	Should:	Could:	Won't
Fling + tap gesture Image import Timer Score	Simple highscore Camera Instructions	Sound effects Bonus points More image-themes Graphics Highscore system	Online integration Difficulty Orientation

Table 2: MoSCoW table

## 4 Design

The application is made up of activities, retainable fragments, and different types of listeners. When starting the application, the first screen visible to the player is the main menu defined in MainActivity, which contains three buttons: Play, Instructions and Highscore. The buttons navigate from the MainActivity to the corresponding activities within the application.

When the instruction button is pressed, a slideshow of images appear in Instructions-Activity to inform the user about the rules. The user can navigate through the images by sliding left or right.

When the play button is pressed the user is led into the PhotoActivity where the user picks a bonus card, which can either be a default card or a personal photo taken with the device's default camera application. To be able to use this, the application uses the camera permission.

After the user has made his or her choice the GameActivity appears and the game begins. When the game is over, the user can either go to the main menu or stay in GameActivity to play again. If the user's score is high enough to make it into the high-score list, the user can submit the score and enter HighscoreActivity. This activity is as previously mentioned also accessible from the main menu.

To handle a part of the game logic, we have used a retainable fragment with an AsyncTask. This fragment is called VerifyFragment and it checks the type of interaction that the user has done and whether the two images are identical or not. Another fragment used in the game is DialogFragment called FinishDialogFragment, whose options change based on the player's score as previously mentioned in this section.

The buttons in PhotoActivity are ImageViews with added OnClickListener, while onTouchListener is mainly used for the fling gesture in the game and the instructions menu. Within the onTouchListener we have a custom gesture listener to have full control over the different types of gestures available within the GameActivity and the InstructionsActivity.

## 5 Implementation

Activities are opened and closed by starting and finishing intents. In certain activities where the user should be able to return to the main menu and it is not the previous activity on the stack, a new intent to the main menu is created with the added flag `FLAG_ACTIVITY_CLEAR_TOP`. This brings the very first activity (MainActivity) to the top and closes the others. Another used flag in the application is `FLAG_ACTIVITY_NO_ANIMATION`, which removes the animation during the transition between PhotoActivity and GameActivity. To remove the animation when an activity is closed, the 'finish' method is overridden with the added line "`overridePendingTransition(0, 0)`".

The only activity that is started with `startActivityForResult` is the camera application. When a photo is taken, `onActivityResult` is called with a request code, a result code and data. If the request code and result code are correct, a new intent to GameActivity is created with the data added. In GameActivity, the data can now be used to get the photo that the user has taken and apply it to the bonus card. The application checks whether the user has taken a photo or wants to use the default card with `getStringExtra`.

The key word is based on the type of gesture the user is done. A down swipe has the key "keep", an up swipe has the key "skip", and double tap has either the key "bonus"

or “rush” if it is a bonus or rush card. The `VerifyFragment` class creates an interface to be able to interact with the `GameActivity` class, in order to the reference photo and the current photo. These images are then sent through the `AsyncTask` where the photos id’s check to be same or not. If the images are the same and the key is keep, then the integer `addPoints` is set to 1 and the boolean `updateAllCards` is set to true. If the images are not the same and the key is keep `addPoints` is set to -1 and `updateAllCards` is set to false. After the `imageMatch` is completed `addPoints` and `updateAllCards` will be sent back to the `GameActivity` and the score and cards are updated accordingly. If the key is “skip” `addPoints` will be 0 and `updateAllCards` is set to false. If the key is “bonus” `addPoints` is set to 5 and `updateAllCards` to true, and if the key is “rush” the rush game mode starts. Within this game mode `addPoints` will set to 1 and `updateAllCards` to false no matter if it is a up or down swipe.

In the `FinishDialogFragment` we use a `AlertDialog.Builder` to build a dialog. The dialog have different text and buttons depending on the score that the user has gotten during the game. If the score is a high score the text is set to “Congratulations! You made your highscore“ or “Congratulations! You beat your highscore” if it is the highest high score. If the score is a high score an `editText` box is added to the dialog where the user can enter his or hers name and a submit button that will store the score and the name in a `Shared Preferences`. If the score is not a high score the message text is set to “You scored: XX score”. The `FinishDialogFragment` also uses an interface to interact with `GameActivity` if the user presses the start button.

The `Image` class extends the `ImageView` class, where we have overwritten the constructor to accept a custom ID, an image resource or bitmap, a bonus boolean and a rush hour boolean. The remaining functions within the `Image` class are getter methods for usage with the `GameActivity`, `GestureListener` and `VerifyFragment`. `getDrawImage()` returns the image resource stored within the object and `getBitmap()` returns the bitmap. In the `GameActivity` `getBitmap` is used as a check if an image has a bitmap and uses it for ressources otherwise it will uses `getDrawImage()`. `isBonus()` and `isRush()` is used to access the corresponding booleans for usage within the `GestureListener` to check is the card is a rush or bonus card. `getID()` is used with the `VerifyFragment` to get two `Images` in the `Image` array’s ID for the `idMatch` check.

The `GestureListener` class extends `GestureDetector.SimpleOnTouchListener` that implements two types of gestures fling and double tap. `onFling()` method checks what type of swipe the user has used. If it is a `topSwipe` then the `VerifyFragment` `starts()` method is called with the key “skip”. Else if it is a `bottomSwipe` then the `VerifyFragment` `starts` method is also called with the key “keep”. The `topSwipe` and `bottomSwipe` check is the coordinates from the two events difference is greater than 120 and the velocity is greater than 200. The only difference between the two methods is the subtraction of between the two events. `onDoubleTap` checks `GameActivity`’s current image bonus boolean and rush boolean. If the bonus boolean is true the `VerifyFragments` `start()` method is called

with the key “bonus” else if the rush boolean is true the start() method is called with the key “rush”.

In HighscoreActivity a top 5 list is shown with the five highest scores since the last reset. The highscore is kept in a SharedPreferences-class to make sure, they still appear even when destroying and creating the application. The SharedPreferences keep the information as key-value pairs. These pairs have a key to both the name and the points that indicates the place in the highscore and the type. For example will key “name1” be the name of the first place in the highscore, and key “point5” will be the points of the fifth place in the highscore. These keys match the id of each TextView shown in the `activity_highscore.xml` file. The methods in GameActivity involving the highscore saves the highscore gained from FinishDialogFragment, and decides if the score makes the highscore board. The method getMinKey() returns the key to the lowest score to replace, so that it only is the top 5 scores that is in the SharedPreferences. Every time the HighScoreActivity is called, the activity updates the names and points, so the values of the specific keys matches the ranking. For this an “insertion sort”-algorithm is used to sort the points, and both sorted names and points are saved in two separate arrays. These arrays are then shown as TextViews in a GridLayout in the `activity_highscore.xml` file. The reset button clears the Editor in the specific SharedPreference-class and calls onResume to update the view.

The InstructionsActivity displays a background image, a menu title, an image with the instructions, a text to the corresponding image, a page (image) number, and a back button. It has an array of eight different images and an array of the text corresponding to each of the images, displaying only one image and text section at a time. The initial image is the first image in the array. When sliding left, the next image in the array is displayed if the last image is not reached, and sliding right displays the previous image if the first image is not being displayed. The gesture detection is set up via gestureHandling. The slide is detected via the CustomGestureDetector class, which extends GestureDetector.SimpleOnGestureListener. When the detected horizontal slide distance and speed are greater than certain values, the corresponding methods increasing or decreasing the image number in the array and the update method is called, setting the correct image, the correct corresponding text, and page number.

The timer is implemented as a CountdownTimer set to 60 seconds. The text view showing the time is updated in the timer’s ‘OnTick’-method. When the timer reaches 0, the FinishDialogFragment appears. When the rush hour card is double tapped the boolean rushTime is set to true. If the boolean rushTime is true, the text view jumps to 5 seconds and counts down while the timer is still running as before. When the timer hits 0, the text view shows the actual time with 7 seconds in addition. This makes it look like two different timers where the first timer continues after the second time has ended. It will now look like the timer ends before it reaches 0. After the actual timer terminates, a new timer is created to show the remaining seconds. We have done it in

this way, because it is not possible to have two timers running at the same time.

## 6 Testing

The application has been tested on the emulators Galaxy Nexus API 18 and Nexus 5 API 22 and a real HTC One M7 API 21 to ensure that the applications runs fast and smoothly each time a new feature has been added.

## 7 Development process and discussion

Following the MoSCoW model from the analysis section, the first things implemented were the game logic, timer and score from the Must column. Since we had time left, the Should and Could functions were also added afterwards and additional game ideas such as rush hour mode.

## 8 Responsibilities

People	Main Responsibility	Other
Anna Ølgaard Nielsen (s144437)	HighscoreActivity	Highscore-methods (GameActivity + FinishDialogFragment) Rush Hour mode (GameActivity) Custom font
Martin Dariush R. Hansen (s144459)	InstructionsActivity	
Per Lange Laursen (s144456)	GameActivity VerifyFragment (part of GameActivity) FinishDialogFragment (part of GameActivity)	Danish Translation MainActivity  Custom icon
Van Anh Thi Trinh (s144449)	PhotoActivity (camera implementation)	Bonus card (GameActivity) Rush Hour mode (GameActivity) Sound effects (GameActivity) Graphics

Table 4: Responsibilities table

## 9 Sources

- Android Developer - 15-06-2015  
<http://developer.android.com/guide/index.html>
- Android Developer: "FragmentActivity" - 15-06-2015  
[http://developer.android.com/reference/android/support/v4/app/FragmentActivity.html#getSupportFragmentManager\(\)](http://developer.android.com/reference/android/support/v4/app/FragmentActivity.html#getSupportFragmentManager())
- Android Developer: "Camera" - 16-06-2015  
<http://developer.android.com/reference/android/hardware/Camera.html>
- Android Developer: "AlertDialog" - 16-06-2015  
<http://developer.android.com/reference/android/app/AlertDialog.html>
- Android Developer: "CountDown Timer" - 16-06-2015  
<http://developer.android.com/reference/android/os/CountDownTimer.html>
- Android Developer: "Dialog" - 16-06-2015  
<http://developer.android.com/reference/android/app/Dialog.html>
- Android Developer: "DialogFragment" - 16-06-2015  
<http://developer.android.com/reference/android/support/v4/app/DialogFragment.html>
- Android Developer: "Dialogs" - 15-06-2015  
<http://developer.android.com/guide/topics/ui/dialogs.html>
- Android Developer: "Taking Photos Simply" - 16-06-2015  
<http://developer.android.com/training/camera/photobasics.html>
- Flashkit - 18-06-2015  
<http://www.flashkit.com/>
- foobarpig - 18-06-2015  
<http://foobarpig.com/android-dev/how-to-disable-animation-on-startactivity-finish-a.html>
- Github: "aporter" - 17-06-2015  
<https://github.com/aporter/coursera-android-labs>
- Google Font - 19-06-2015  
<https://www.google.com/fonts#>
- SoundJay - 18-06-2015  
<http://www.soundjay.com/>
- Stack Overflow - 17-06-2015  
<http://stackoverflow.com/questions/8906269/alertdialogs-setcancelablefalse-method-n>



- Stack Overflow - 18-06-2015  
<http://stackoverflow.com/questions/27774414/add-bigger-margin-to-edittext-in-android>
- Stack Overflow: “Android - basic gesture detection” - 18-06-2015  
<http://stackoverflow.com/questions/3285412/limit-text-length-of-edittext-in-android>
- Stack Overflow - 19-06-2015  
<http://stackoverflow.com/questions/937313/android-basic-gesture-detection>
- Stack Overflow - 19-06-2015  
<http://stackoverflow.com/questions/2376250/custom-fonts-and-xml-layouts-android>
- Cormen, T. H. et al. - Massachusetts Institute of Technology (2009) - “Introduction to Algorithms” - 3. edition - page 18 (Insertion sort)