|  |  |
| --- | --- |
|  |  |
| Mobile Information Systems |  |
|  |  |
| Truth Estimation from Mobile Interaction |  |
|  |  |
|  | Written by  Jyotsna Gorle | Matr.Nr.: 120290  Christopher Heiden | Matr.Nr.: 120249 |
|  |  |
|  | Submitted to Prof. Echtler |
|  |  |
|  | Submission date: |

Contents

[1 Theoretical Part 1](#_Toc15659758)

[1.1 A Summary of the paper ”Veritaps: Truth Estimation from Mobile Interaction” 1](#_Toc15659759)

[2 Practical Part 2](#_Toc15659760)

[2.1 Differences between paper code 2](#_Toc15659761)

[2.2 How does the game work? 2](#_Toc15659762)

[2.3 How to calculate if someone lying or not? 3](#_Toc15659763)

[2.4 Important code snippets 3](#_Toc15659764)

[2.4.1 Transitions 3](#_Toc15659765)

[2.4.2 AI programming 4](#_Toc15659766)

[2.5 Code Structure 4](#_Toc15659767)

[3 Literature 5](#_Toc15659768)

[4 Annex 6](#_Toc15659769)

# Theoretical Part

## A Summary of the paper ”Veritaps: Truth Estimation from Mobile Interaction”

In the paper “Veritaps: Truth Estimation from Mobile Interaction” the authors Aske Mottelson, Jarrod Knibbe, and Kasper Hornbæk wrote about ways to find out if someone is lying or not using sensor data from commodity smartphones. They argue that people use the phone very often, so by change of the habit of typing or pushing, they can find out if someone if lying. To prove that they can do that, they programmed three applications that can be used by phone to show that it is possible to find out if someone is lying on mobile devices.

The first study, they made is called ”simple lies”. The app had the goal to find out if someone is lying through touch interaction. Therefore, participants have been asked to lie or to tell the truth. Lying takes more time than telling the truth. This is used in the application, so the app is calculating the response time for an answer. They proved it with buttons and sliders.

In the second study, they have shown that acceleration, rotation, and inter-key-press can be used in lie detection. For this app an artificial intelligence (short AI) has been programmed to communicate with the user of the app. However, this is not the only difference to the first study. “In this variant the participants are offered an incentive to lie.”[1]

The last app has been named as “Yatzy Game”. In this study, the authors tried to find out if it is possible to notice spontaneous lying by reference dishonesty. Therefore, the participants play a dice game on their phone. The actions in the app are randomized what is a difference to the other studies. Moreover, lying is rewarded relative to the outcomes, so lying is profitable.

In all of these games, it was possible to find out if a user has been lied or not after a certain time period. This means it is possible to find out if a user is lying or not.

# Practical Part

## Differences between paper code

The group of J. Gorle and C. Heiden has been focusing on the third study. The first and second study aren’t the focus of this project. The dice game[1] has been implemented with some slight changes in the design in design and functionality.

At first, the app has been restructured, so now the application has an opening screen, an information screen, and the game itself with its two screens (see Figure 2, 3, 4, 5, 6). Instead of using buttons to change the screen, the application is focused on touch gestures. Furthermore, the “Chance”-combination has been removed from the list. This also means that the user doesn’t have to play 12 rounds; only play 11.

In addition to that, an addition type of selecting the combination has been added. So, on the selection screen the user can do two different types of touch gestures. First, there is the long press what is used to find dices that show the same number as the dice the user is touching. A normal touch (“click”) is used to find combinations like “three of a kind”, ”four or a kind“, little straight”, ”large straight”, or ”house”.

## How does the game work?

Every user who is playing this game, has 12 rounds. Therefore, he/she can use five dices. In every round the user has to roll the dices at least one time. Still, it can be done two more times. If he/she thinks he/she has a good setup, the user can choose the best possible combination from a list by entering the points for this specific combination. In the app the points are the sum of the dice. This has to be done in every round. However, it isn’t possible to select the same combination twice. If there is no combination that can be selected, then the user has to enter 0.

The user gets “rewarded based on the sum of their reported scores, providing an incentive to lie”[1]:

* $0:50 : below 150 points
* $1:00 : between 150 and 200 points
* $2:00 : more than 200 points

## How to calculate if someone lying or not?

When the user enters the last screen on which he/she can tell what combination he/she got and how many points he will get with that combination, a timer is starting in the background and compute the time until the user is swiping the “done” slider. In this process, nearly everything is stored, like touching, scrolling through the list of combinations, and taps were when entering the amount of points on a num-pad. Furthermore, motions are recorded throughout the entry.

The outcome is the following:

* The participant reports their score accurately (Truth).
* The participant purposefully inflates their score (Lie)
* The participant unintentionally inflates the score (Truth - the participant does not intend to deceive)

## Important code snippets

In the next chapters important code snippets will be explained. The programming environment was the Android IDE. [2] In addition to that, the UML structure of the app will be presented. This has been made by Lucidchart.[3]

### Transitions

The most difficult part in programming the interface was to find out how to animate the transitions also and change the screens. Therefore, the group had to create an ”anim folder”. There, XML files had to be created with a set-tag and a translate-tag. In these tags programmers can implement transition aspects like the duration of the transition and the x- and y-coordinates where the transition starts and where it ends.

To use the transition, programmers have to define an FragmentTransaction. With that, you can replace the older screen and also set up a costume animation (see *Code 1*).

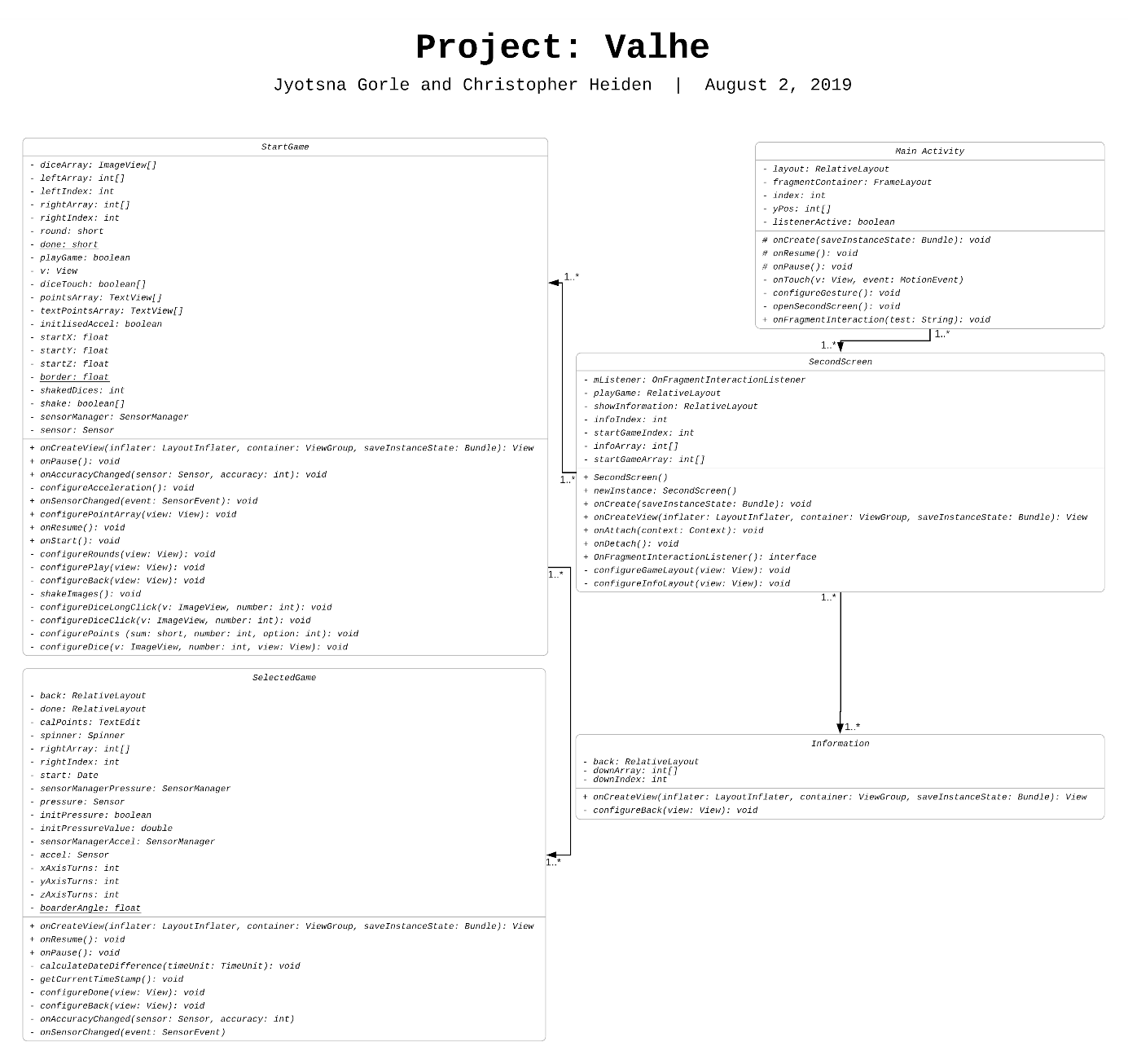
|  |  |  |
| --- | --- | --- |
|  | SecondScreen fragment = SecondScreen.newInstance(); FragmentManager fragmentManager = getSupportFragmentManager(); FragmentTransaction transaction = fragmentManager.beginTransaction(); transaction.setCustomAnimations(R.anim.enter\_from\_bottom, R.anim.exit\_from\_bottom); transaction.addToBackStack(null); transaction.add(R.id.fragment\_container, fragment, "SECOND\_SCREEN").commit(); |  |

Code 1 shows how to change screens in an app and also how to call the self-generated transitions.

### AI programming

## Code Structure

The code structure for the whole application can be found in Figure 1.

Figure 1 shows the code structure of the project.

# Literature

[1] A. Mottelson, J. Knibbe, and K. Hornbæk, “Veritaps: Truth Estimation from Mobile Interaction,” Copenhagen, 2018.

[2] “Android Studio,” *Google Developers*, 2019. [Online]. Available: https://developer.android.com/studio. [Accessed: 02-Aug-2019].

[3] “Professionelle Diagramme,” *© 2019 Lucid Software Inc.*, 2019. [Online]. Available: https://www.lucidchart.com/pages/de. [Accessed: 22-Jul-2019].

# Annex

|  |  |
| --- | --- |
|  |  |
| Figure 2 shows the first screen of the app which acts as an opening screen. | Figure 3 shows the second screen in which the user can either start the game or find out how the game works. |
|  |  |
| Figure 4 shows the information screen that the user can open to learning how the game works. | Figure 5 shows the rolling dice screen and the selecting dices part of the game. |
|  |  |
| Figure 6 shows the last screen of the app in which the user can choose the combination and the points he/she gets. |  |