SWITCHEROO

Final project for Mobile/Wearable programming

Professors:

Mariusz Nowostawski

Christopher Frantz

***Team**:*

Dominik Huber
Jesús Herrera López
Dafina Marku
Sergio Gimeno Millán

May 06, 2018

INDEX

Introduction & Project Description

Switcheroo, a project created and designed by a group of students called "Too sober for this" or "Group 10", for the Mobile/Wearable programming course. The project is a mobile game application for Android phones only.

The app is a digital version of the classic phrase-drawing game. The original game is played in a piece of paper by an unlimited number of players. Its mechanic is really simple, the first player makes a drawing and gives the paper to the second player. The second player has to interpret what he/sh sees in the drawing and write a sentence about it. Once finished writing, he/she folds the paper so the next player only sees the sentence written. Next player draws again but this time the drawing has to match the sentence of the previous player. The game continues until the players decide to end it.

Our app is exactly this same game but instead of using a paper and passing it around, now every player gets to have his/her own paper on his/her own phone screen.

Regarding the name of the app, switcheroo is a sudden unexpected variation or reversal, often for a humorous purpose. It is colloquially used in reference to an act of intentionally or unintentionally swapping two objects. Since in this game the players "swap" the paper, or more precisely said, pass the paper to the following player, we thought that Switcheroo is an interesting and adequate name for it.

Features

-Firebase database: Uses the Firebase realtime database to ensure stable and fast game progression and also relatively clean code around the integration.

-User authentication via firebase: The first time any user uses the app, will be asked for a nickname. This nickname must be unique, therefore, it is compared in the database with the other registered users. If that nickname is already in use, it will be rejected and asked for another.

-QR reader/writer: The app uses QR codes to join or host a game lobby.

> -To **host** the game it is needed to create the QR image with the code of the room stored in firebase. This is done with the help of BitMap and Bit Matrix, that are turned into an image encoded in QR.

-To **join** a game, the phone's camera will be activated to caption the QR code. This action is performed by the library QRCodeReaderView. It extracts the information from the code, such as the key of the desired game.

-Drawing tools: This is a core feature for the app, as it one of the two parts that form the communication between players. It allows the user to draw with his/her finger on the screen. It has been programmed with the android Canvas. The user experience has been enhanced by adding an "Undo" button, which allows the player to delete the previous trace, and multiple color buttons that will change the color of the trace.

-Final Image: Once the game is over, all the players have the option to download the "piece of art" they all have created together. Both sentences and images are shown in a concatenated png file in the order they were performed.

Design

Our top priority were the functionality and the correct working of the app so we didn't allocate too many resources into the design part. It has a really basic design using some buttons, text views and a canvas.

The *Main Menu* activity contains a text with the instructions and two buttons depending if the user wants to host a game or join one already created. If the user wants to create a game, it will display a new activity with the QR coin for other players to scan and join the game. On the contrary if the user choose to join a game, a fragment with the camera will appear to scan the QR code.

Once all the players are inside the game, the host will start drawing something. We decided to place the color of drawing buttons in top and the action buttons below, especially the send button is placed in the right to imitate the chat apps send one, because in our opinion this way it is more intuitive for the user. Also the color choosing for the send (in green) and clear (in red) buttons

After a whole round is completed, the host will be asked if continue playing or end the game for good. This activity has the same design as the main activity, with the textview with some instructions and 2 buttons with the same colors.

Development Process

This project was developed in a short period of time, but with an intense amount of work. The main goal was to get a functional prototype in one weekend. We chose this Hackathon/Game Jam-like format because of positive experiences in another course that relies heavily on this format of developing new applications, especially games. Now we will explain the steps that were followed to reach this goal:

-Brainstorming for ideas: Before this “hackathon” took place, we needed to obtain an idea. We had a brainstorm for some hours in a meeting. After a while, we came up with a variety of viable ideas where this one shined among the rest.

-User interface design: Once the idea was settled, we knew what we wanted to achieve. The user interface is key in an app, as it is what the user experiences, so the project was built around a simple interface that fulfills the goals of the app´s idea. We thought that few activities and multiple fragments with different functionalities was the best option for this project.

-Protocol and database design: Due to the experience with the course’s lab 4, Firebase was chosen as the backing technology for handling the game’s background communication between host and clients. Using the Firebase realtime database, a protocol based on so-called *GameTransaction*s was developed, which enables easy communication between the host and the clients of the game. The *GameTransaction*s can have different types and payloads, a tabular overview over transaction types and associated payloads can be found in Appendix I.

There is no database design as such, the entire data model consists of Java classes which are automatically serialized and stored in the database by the Firebase integration libraries.

-Back-End implementation: Back-end implementation was done during the 48h Game Jam that led to the initial version of the app, namely by first drawing up the protocol on paper and then implementing all back-end functionality below a very basic GUI. The backing functionality (protocol, transactions, etc.) did since undergo no functionality changes, only refactorings.

-Play Testing: Once the debugging was finished, it was time to test our product. As it is a party game, we gathered together and played some rounds Laughs were assured and we had a great time.

Difficulty discussion

In this section, the difficulty around the app’s different components:

- The canvas in general was not a hard task to do, as there are multiple tutorials about it that explain in depth its behaviour and complexities. This makes it a really understandable feature of android. The difficulty increased when the functionality of “undo” was implemented, alongside with the different color changes. In other words, the whole structure of the canvas had to be changed to an array of traces. This way, we could track the last movement for the undo function, and also change the last

trace’s colour (if the user had chosen so) without any influence on the previous traces.
- Even though Firebase was a crucial technology in the course’s fourth lab, getting the asynchronous *Game Transaction*-based protocol running correctly was quite a challenge, in the sense of having the correct conditional branches for a multitude of cases, like the player being host or not, the current game state, etc. This was further complicated by the protocol being partly stateless, meaning that the host would only track whose turn it is, but not any other state about the players.
- Another challenging part was getting the *Bitmap* representation of the *DrawView* to work correctly, as sometimes data got corrupted in the process of being rendered, compressed, encoded as Base64, sent via *GameTransaction* and finally being reconstructed by reversing the process. Careful fine-tuning and referring to the documentation helped solving this problem in the end.

Future Work

In this section will be discussed all the work that due to the lack of time or other reasons we could not include into the delivered product. These features are not a must have for the app in order to work but will improve significantly the quality of life inside the app.

- **Add a timer for each turn:** At the current state of the app, the player who draws can spend drawing all the time he/she wants. The idea is implement a timer to make the game more fluid and, in a certain way, challenging.

- **Kick the players when the game is finished:** This feature is needed, in case the host closes the game or a bug happens, to kick out all the participants instead of having them waiting for a player to draw something.

- **Notifications:** A system to push notifications when the player drawing has finished or when it is the turn of the user to draw or answer.

- **Improve the design:** Overall the design is very rough and simple, so the idea of making it more appealing for the users is something that we have to take in mind. Creating a logo for the app, adding some transitions between activities, SFX(for example, the sound of a pencil while drawing) between others are improvements, in our opinion, much required.

What we have learned

- The importance of time management
- Fulfilling a project in a short amount of time
- QR libraries.
- Additional knowledge of Firebase.

Appendix I: Table of *GameTransaction* types

Type	**Payload desc.**	**Sender**	**Receiver**
TYPE_TEXT	A textual description of a previous image	any	any
TYPE_IMG	A Base64-encoded image, either from the start of the game, or based on a previous text	any	any
TYPE_END	The full image, meaning all images and texts stitched together, encoded in Base64	host	broadcast
TYPE_DONE	Empty string	any	host
TYPE_NEXT	User-ID of the next player	host	any

The protocol, for an exemplary game, would work as follows: At the beginning, the host is the first to draw an image to start the game. After the host player has finished drawing, the host will act as any other client, and sent a transaction to itself asking for the next player's ID. They will then reply to themselves, and send the finished image to the next player. Note that this essentially enables the host to play with themselves.

The next player will open the WritingFragment after receiving an image, in order for the player to describe it. After they are finished, the client will request the next ID from the host, and sent the text to the next player. Meanwhile, the host will log all image and text transactions for the final image.

If the host notices that they would be the next player in line, the hosting player is asked whether to continue or end the game. If they chose the latter, the host sends out a final transaction ending the game, which also contains the finished images for the clients to view and/or share. If they choose to continue, the host simply replies with their ID as the next player's id.