

The Turing Game Development Report

1. Introduction

Background and Problem Statement

The aim of the system was to build an Android app that implements a variation of the Turing Game, to gain insight into the public perception of AI. The system is to enable users to play a mobile app game that tests their ability to differentiate a bot (AI) from a human player. More specifically, the user enters a chat conversation paired randomly with either a bot or a human player. The user is given a specific topic to talk about for a specific time period, after which the user must guess whether they were talking to a human or a bot.

Technical Approach

We approached the project by dividing the work into front-end tasks and back-end tasks. We each researched our respective sides of the project and did tutorials on areas we had to learn about.

When it came to backend and how we could implement the infrastructure needed for the app to function, we decided to use Google's Firebase for a number of reasons:

1. Firstly it has a free plan, which does have certain limitations, but for this app and the initial development we were required to do, these limitations had no impact whatsoever.
2. It's scalable so in the future if the client wants to continue using Firebase, he can upgrade to a different plan with no need to change any code on the app end.
3. The service is very reliable as a result of being hosted by Google. Throughout the course of development, we experienced no downtime. This is much more robust than other options we considered, such as hosting the backend ourselves.
4. Firebase offers a wide array of helpful FAQ's and a large community of users. When we arrived at an roadblocks, it was very easy to get help and guidance through forums and by looking through documentation. This will also make it easier for future developers of the app to understand the work we have done and also build upon it.
5. We also found example applications that used Firebase as a messaging service. This acted as a key point of reference and helped with the design of our own real time messaging service.
6. Firebase also provides authentication services to allow users to login. We were able to leverage this feature to implement anonymous login for the app. This is something that would have otherwise been outside our time budget.

For frontend we used Android Studio as the client didn't specify any preference about the platform the app needed to run on. After confirming that he was happy for the app to only run on Android, we setup a working environment in Android Studio, downloading all the necessary packages and plugins.

The third years provided us with a sample chat app that interacted with Watson AI, with this as our base project we started a Git repository. This allowed us to synchronise our work, log changes and also review each others code.

2. Requirements

Functional Requirements

- Create an app that allows users chat with either a bot or a human and guess which one they were chatting to correctly to win.
- Implement a timer to limit conversation time to avoid players leaving the chat for long periods of time and disrupting the flow of the game.
- Build the app for the Android platform using Android Studio.
- Program the app to ask users for feedback on what gave the bot away if they guessed correctly, post-game to collect data for further analysis.

Non-functional Requirements

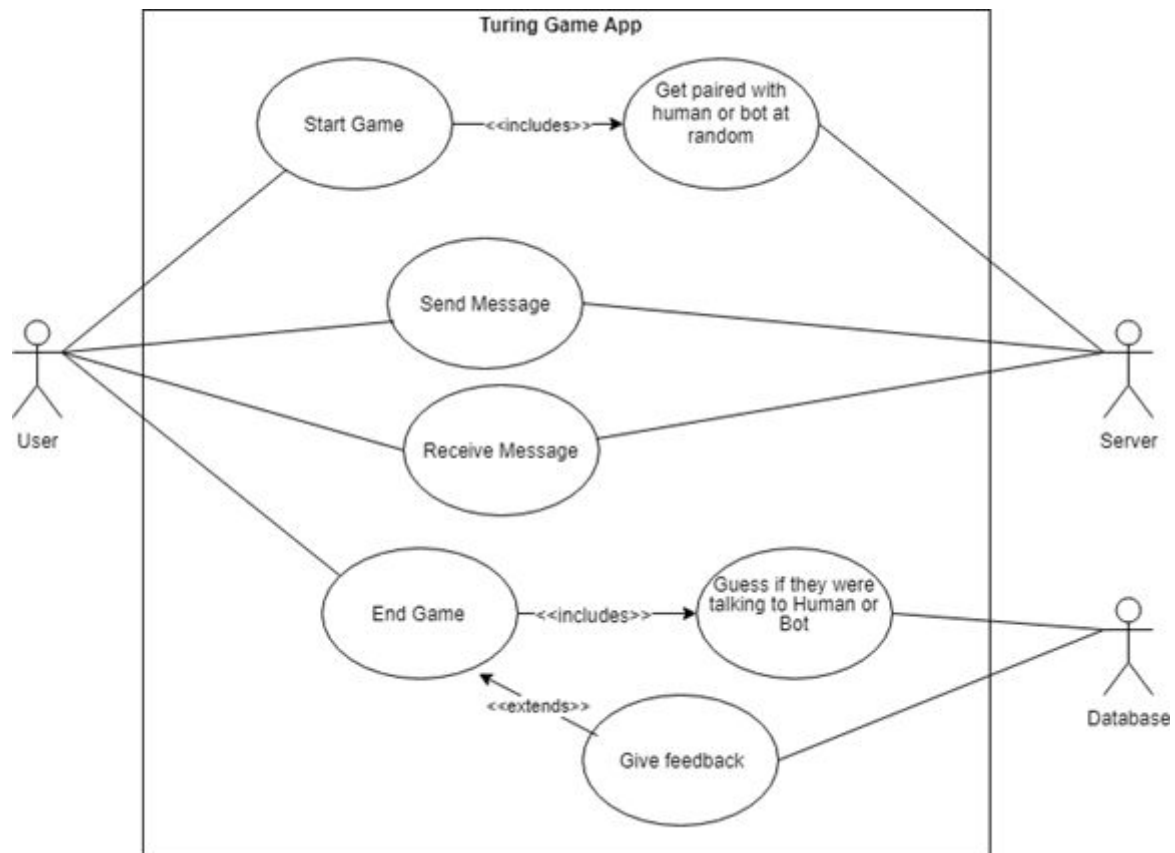
- The app to be user friendly and aesthetically pleasing, as usability is an important factor for this project.
- Server latency should be kept to minimum to facilitate a free-flowing conversation.
- The system to be scalable to allow for the client to be able to build upon our work in the future if he wishes to implement further improvements to the back-end server and train the bot with more dialog responses.
- The program should be easily accessible and easy to use by different software development teams that will be modifying the code.
- Option to contact the client through the app to provide further thoughts/feedback/general questions if a user wishes to do so

User Interaction Scenarios including Use Case diagrams

We implemented a server and an Android app.

Use Case for user to play the game

This use case shows the functionality of the system. The user is the primary actor as they control what actions they take within the app and navigate the game. Our server connecting users and facilitating human to human message exchange is a secondary actor, as well as the database storing guesses made. The feedback function is an improvement we would implement with further time, so it is included as something the user can do.



3. Design

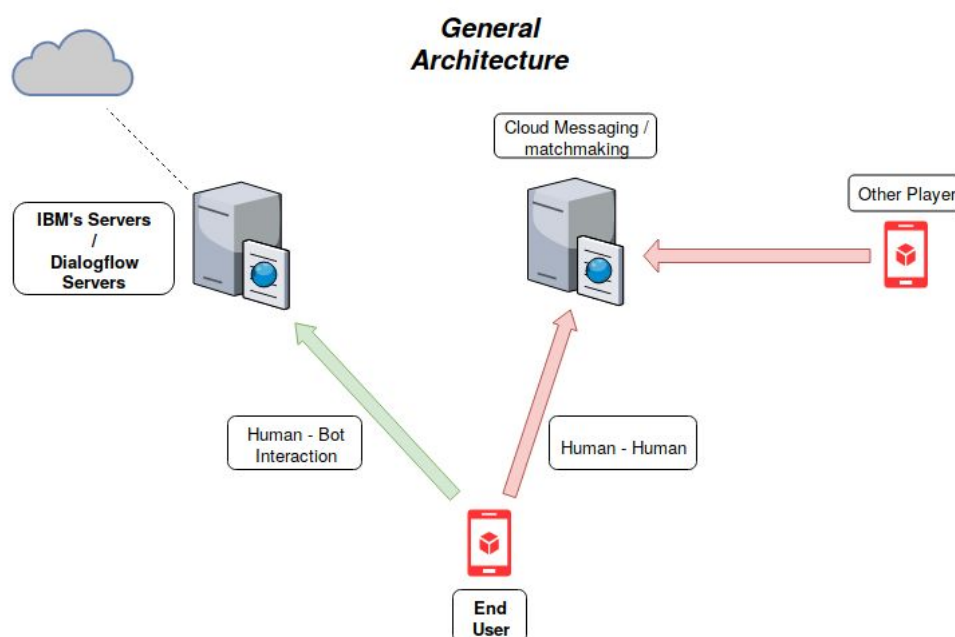
Architecture diagram

End users: The End users control all of the game parameters and logic. An End user interfaces with the other technologies used (firebase and IBM) and upon checking certain conditions play a game against either another human, or a chatbot.

Cloud Messaging / Matchmaking: The messaging service is provided by google's firebase. Firebase provides authentication for anonymous connection and also a realtime database where all the chat data is stored. Matchmaking is achieved using a combination of the app and database. The app queries the database to see if any other players are available, then based on the result makes a matchmaking decision.

Human - Human: After the matchmaking process, if two End users get paired together they are both given the same unique chatroom id. After this all of their messages will be sent to that specific chat room in the firebase database. Whenever a new message arrives in the chatroom, the app will update the players chat. This facilitates the instant messaging required for a flowing conversation.

Human - Bot: If at the end of the matchmaking process, the app finds that no other users are available to play at this time, the user will be paired with a chat bot. This involves sending the players messages not to firebase, but to the IBM DialogFlow Servers. Then a response is generated and sent from there back to the app and displayed on screen after a random delay. Once this conversation is complete, the messages exchanged get pushed to firebase.



<https://firebase.google.com/docs/cloud-messaging/>

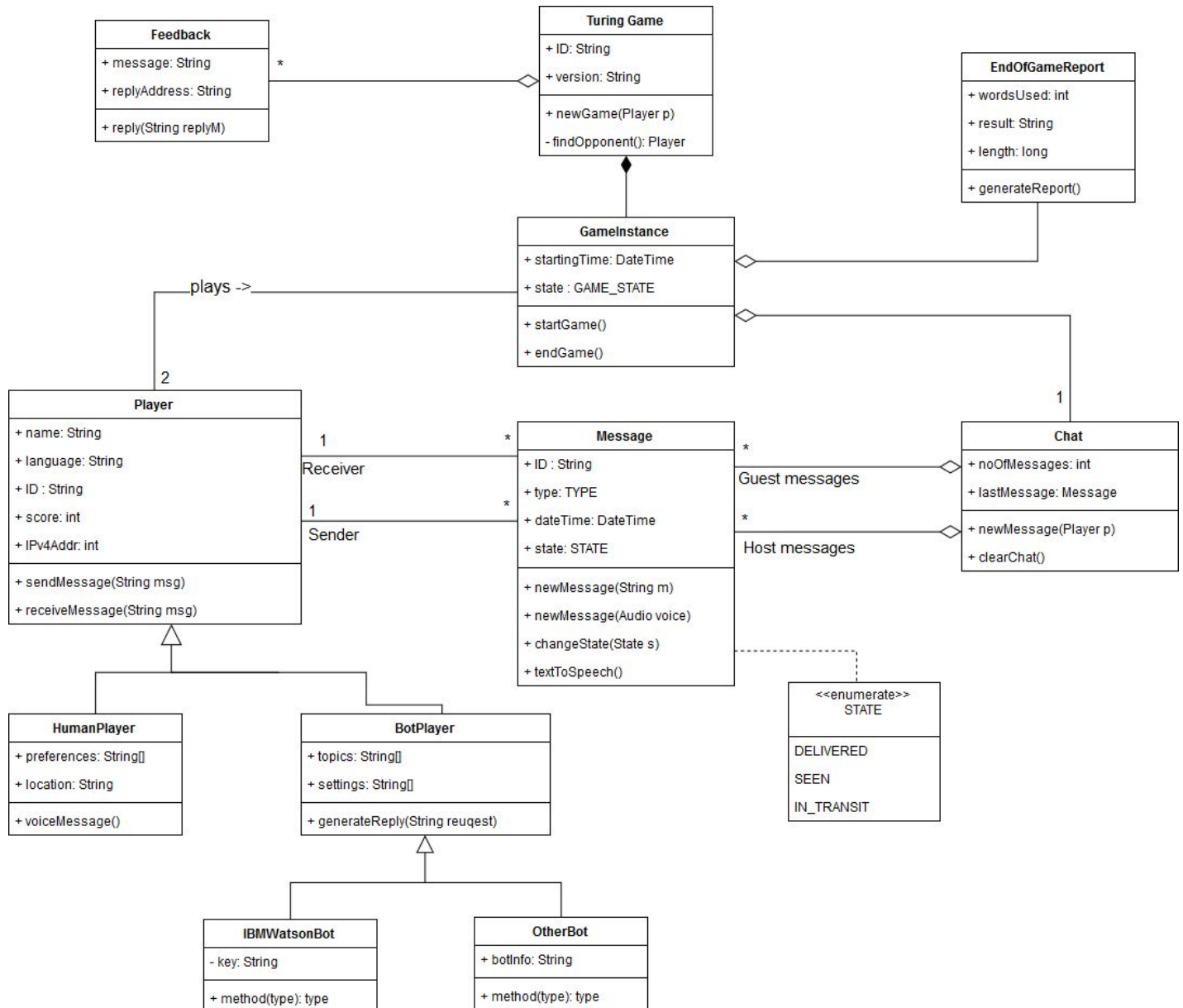
<https://www.ibm.com/watson/>

<https://firebase.google.com/products/>

<https://developers.google.com/games/services/>

UML Class Diagram

NOTE: The structure of the code does not fully follow the class diagram due to style of programming used in Android development. However, the data and the code still follow the logical structure of the class diagram.



Database structure:

```

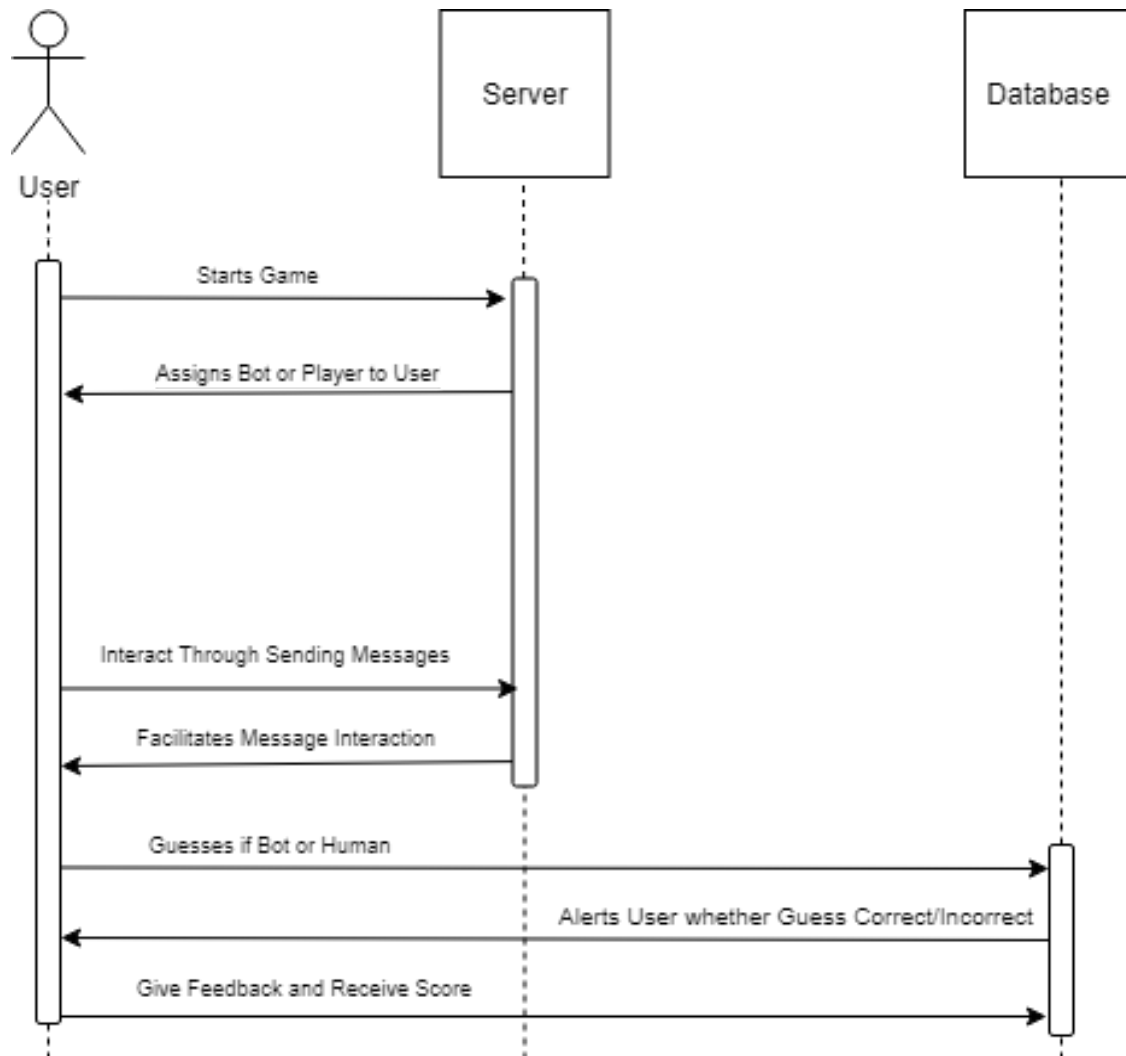
▼ object {3}
  ▼ availableGames {4}
    -LbOTnDpwfmTiOeqE4P_ : full
    -LbUXKqUMJEY148q_4hr : complete
    -LbUZs9H59Ad-kWoLIZ1 : full
    -LbUZtHZ6wT2S1bCnm7C : empty
  ▼ chatRooms {1}
    ▼ -LabXmBgGXxcMa3qyh-n {5}
      ▼ -LabXw2FdZhtWjaR9iW1 {4}
        id : -LabXw2FdZhtWjaR9iW1
        message : Hello, how are you?
        sender : watson bot
        type : bot
      ► -LabXw2M0VcN0te-Z-1U {4}
      ► -LabXw2OHwBzWsmMCyCd {4}
      ► -LabXw2TW0k8v-1mcXrH {4}
      ► -LabXw2V9xzJhJA3vCHA {4}
    ▼ feedback {1}
      ▼ -LbFTVdh18hVHY7ofZD3 {2}
        message : Really nice app!
        reply-email : example@email.com

```

UML Sequence Diagram

This diagram shows the sequence of events that occur when a user plays the game.

NOTE: The last action carried out by the user to “give feedback and receive score” is something we would add in the future, but was not doable within the time constraints so far. It has been left in for that purpose, to showcase how the full version will run when improved.



4. Implementation

Tools/Libraries/Platforms used

Android SDK:

We used Android Studio as our main IDE. Android studio includes the official Android ADK as well as many other tools required for successful development, testing and deployment of an Android app.

Google Firebase:

We used Google Firebase development platform. Services provided by this platform can be easily integrated into an Android application. Android Studio includes a native support for Firebase. In particular, we used Firebase Realtime Database as our main database for storing the dialogues, feedback and data needed for player matching. It is a noSQL database hosted by Google on their cloud servers. The key feature is that all data is synchronised in real-time. This allowed us to implement real-time messaging.

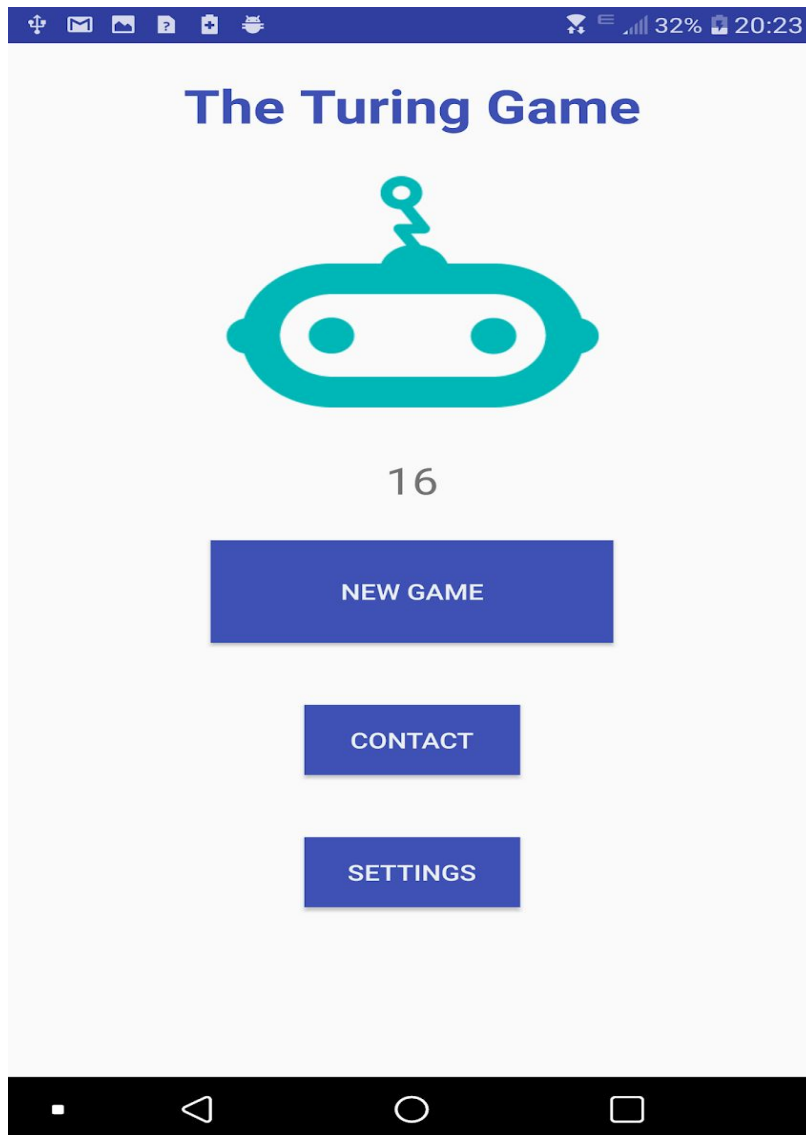
We also used Firebase anonymous authentication to identify the users, as it is required for the database.

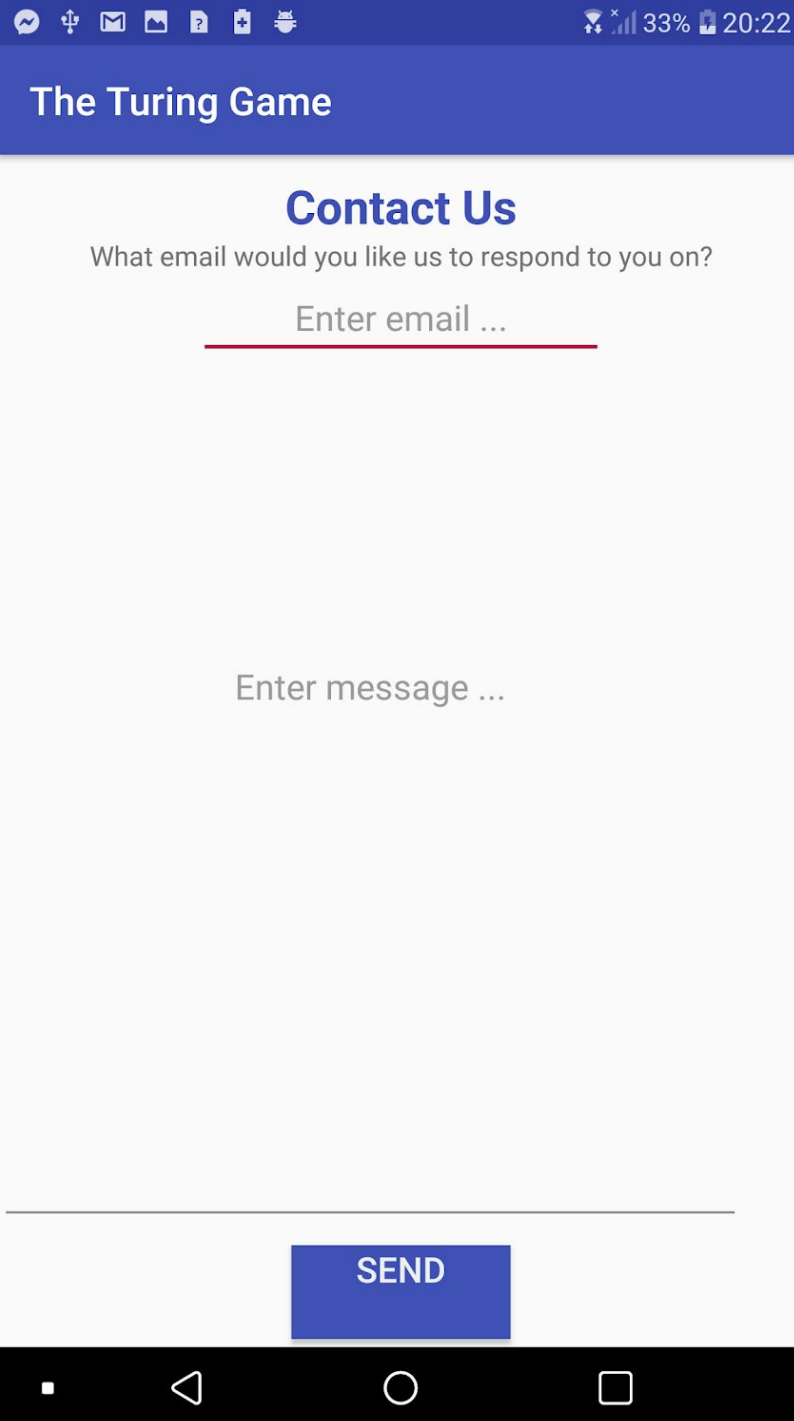
IBM Watson:

We used IBM Watson system. In particular, IBM Watson Assistant is a chatbot that we trained and used as an AI player in our game.

We also used Watson Text-to-Speech/Speech-to-text to allow for voice messaging.

User interface





The screenshot shows a mobile application interface. At the top is a dark blue header bar with the text "The Turing Game" in white. Below the header is a light gray area containing the title "Contact Us" in bold blue text. Underneath the title is the question "What email would you like us to respond to you on?". This is followed by a text input field with the placeholder "Enter email ..." and a red underline. Below this is another text input field with the placeholder "Enter message ...". At the bottom of the form is a blue button with the text "SEND" in white. The entire app is displayed on a black background that includes a status bar at the top with various icons and a time of 20:22, and an Android navigation bar at the bottom.

The Turing Game

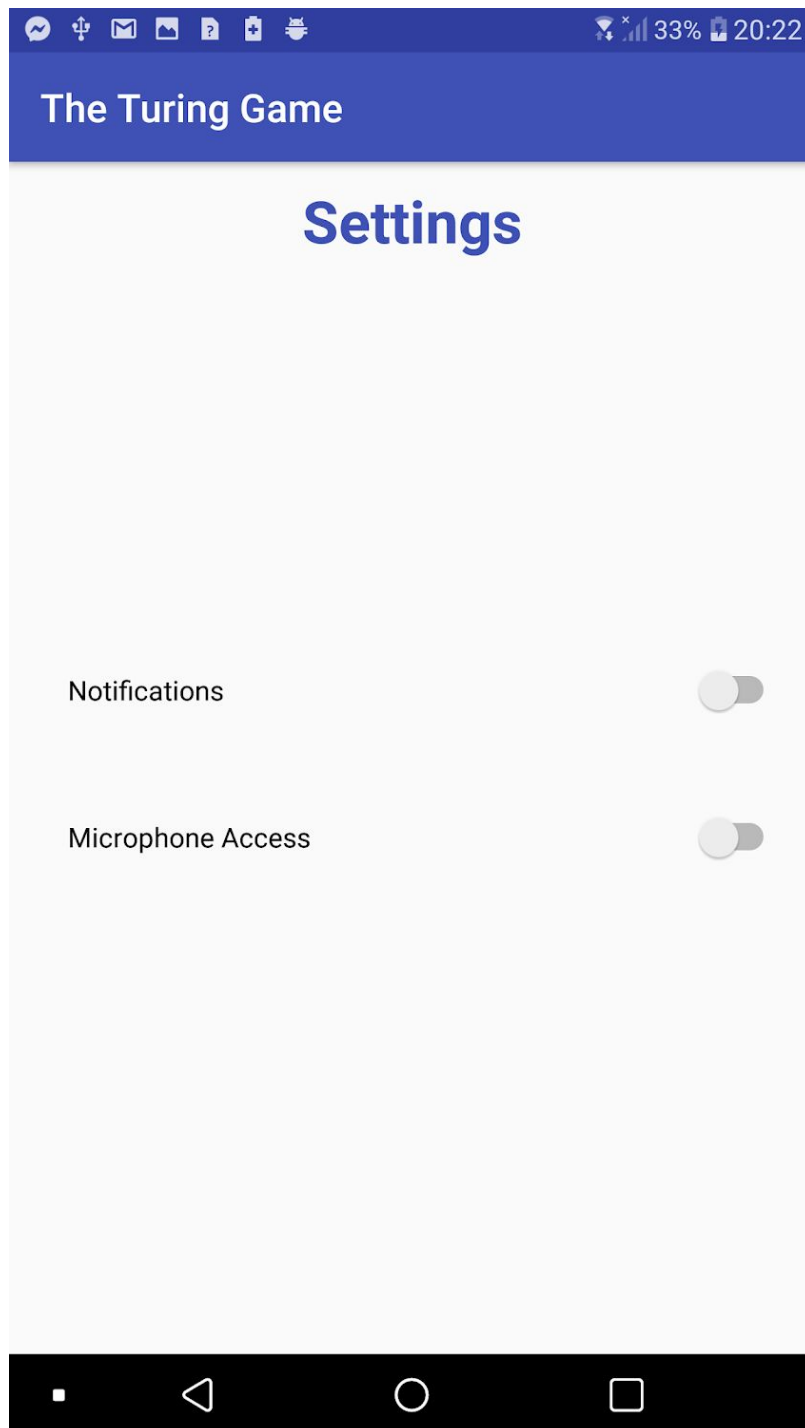
Contact Us

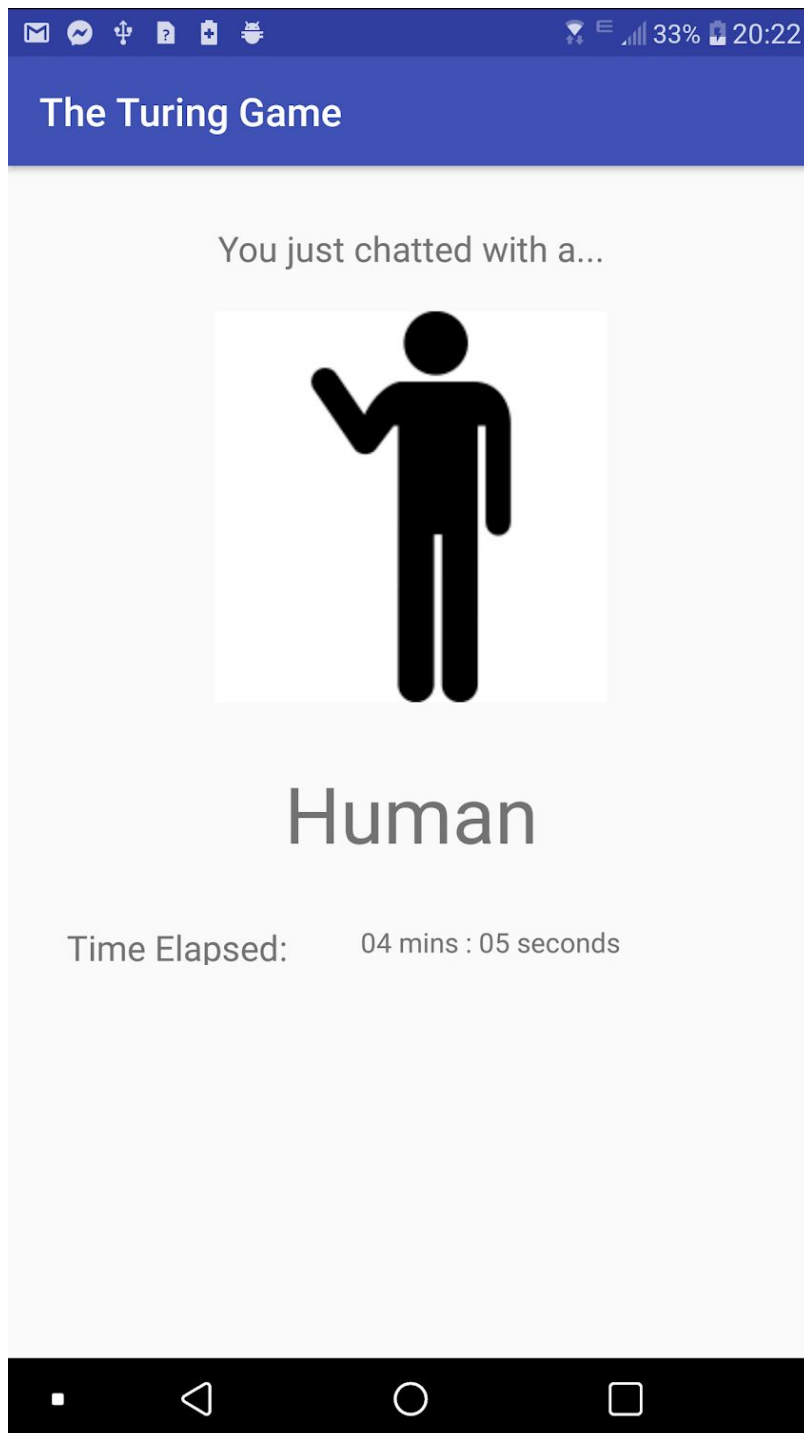
What email would you like us to respond to you on?

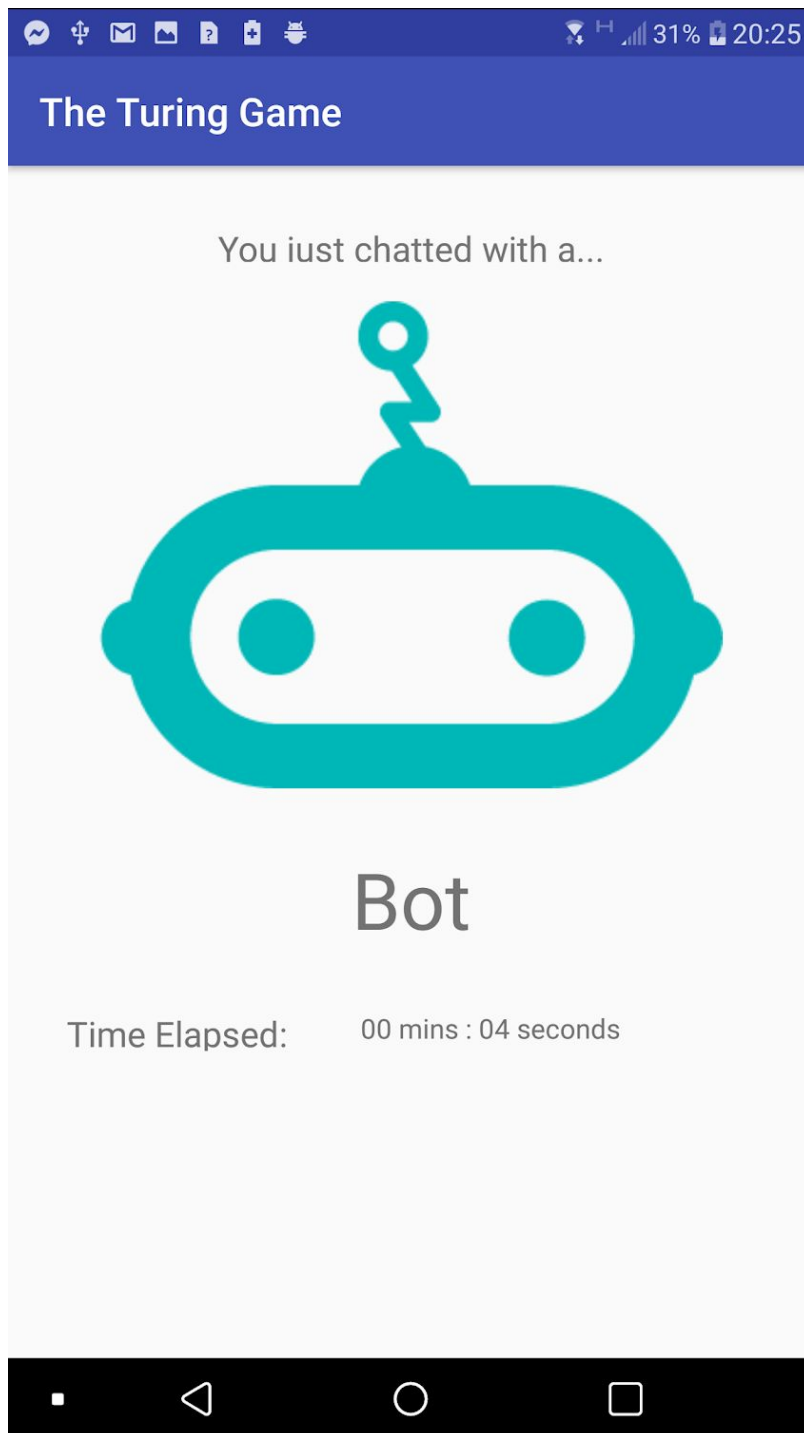
Enter email ...

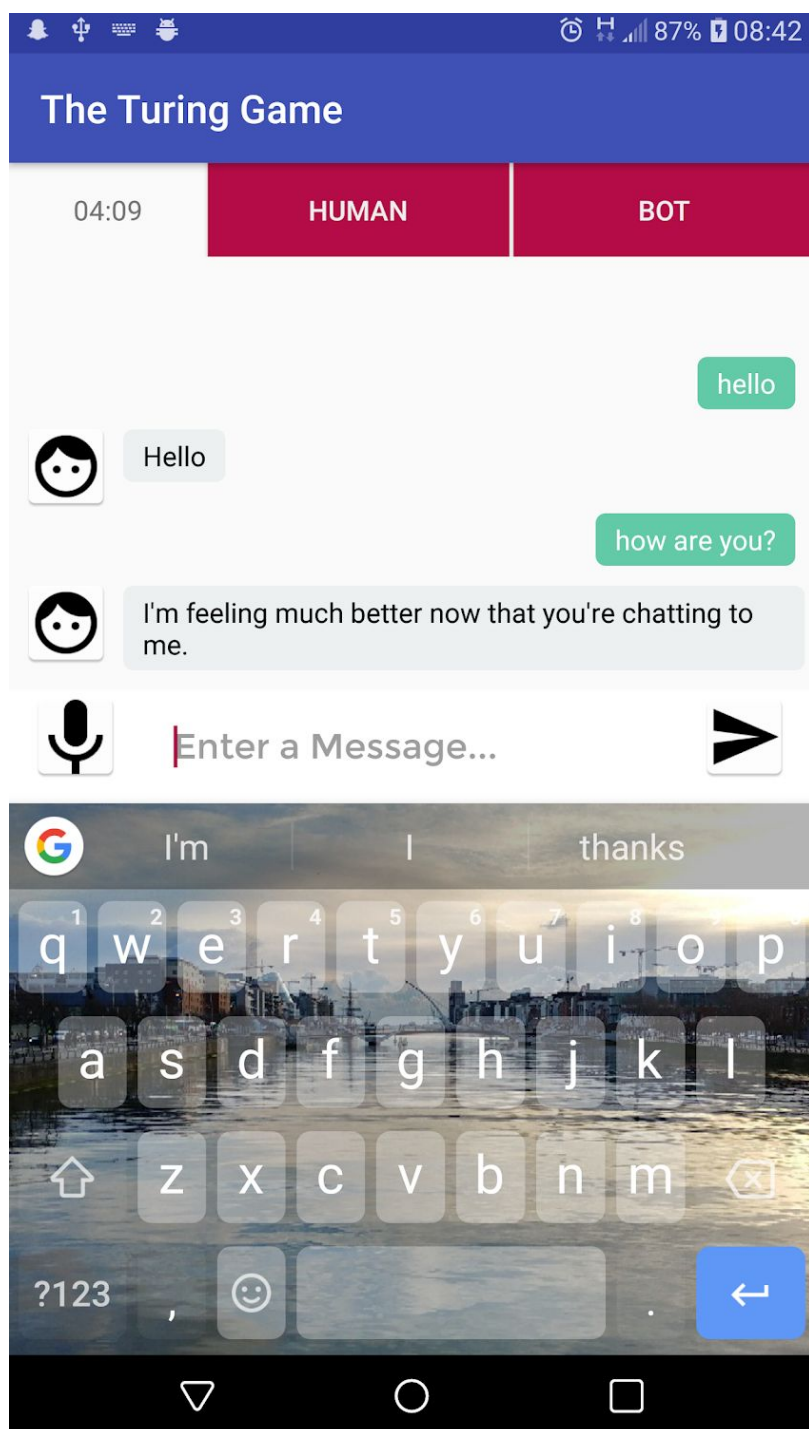
Enter message ...

SEND









Implementation of Algorithms

The Matchmaking Algorithm

Summary:

The matchmaking algorithm used by the app is rather simple, but works effectively for quick matchmaking and the storage of message data for further analysis.

When the app is started and the user presses on “New Game”, the app connects with the database and gets a reference to “availableGames” section of the database.

It scans through all the objects in this section of the database. These objects are key:value json objects. The key in this case is a “chatRoomId” (a randomly generated key from Firebase) and the value is the status of the chat room. “empty” is a chat room with one player waiting for a second player to join, “full” is a chat room that already has two players. “complete” is a chat room where the chat has ended and both players have either guessed/ are in the process of guessing.

Algorithm Steps:

1. When a game starts, the user will check to see if there are any 'empty' chat rooms.
2. If there are, the user will join (this is a human to human game) and a listener will be started on that chatroom to check for new messages.
3. If no chat rooms are available the user will create one and wait to see if someone joins.
4. If no one joins the user will be matched with a bot and the game will start.

Everything related to a single activity is contained within that activity class. This is a widely used design pattern in Android development.

Our human-to-human part of the game is implemented via Firebase Realtime database. The key to it is the ability for a programmer to assign a listener to the database that will be fired on the specified events. We use listeners in both messaging and matchmaking part. In messaging, both parties have listeners to detect new message objects. For example, a sender puts a new message object on a specific path in the database, which causes the listener of the receiver side to be run and display the new message. Similarly, listeners are set up to listen for game state changes, so the game is synchronised between two players.

5. Conclusions

Problems encountered during design/implementation

Android Studio was difficult to adjust to during the early stages which led to some setbacks. It took us some time to get used to Android environment and its coding style. Scaling the user interfaces to fit various phone screens was difficult to implement at first when still learning about this development environment.

Human to human chat interaction proved difficult to implement as a server was needed to connect two players. We decided to use Firebase platform, which allowed us to greatly speed up the development due to it being easy to use and well documented.

However, our decision also had some drawbacks. Since we did not have a separate server, we had to implement all our logic inside the app itself with some help from the realtime database.

Implementing a de-facto peer-to-peer matchmaking protocol with only a database proved to be challenging.

Self-Evaluation of how well the project objectives were reached

We think the fundamental requirements of the app were met. As this was a proof of concept we believe it is a good base to start at and can be further improved in the future. The client is satisfied with our work. We would ideally have liked to implement a feedback function, but overall project objectives were met accordingly.

Self-evaluation of working as a group

Work was divided well, with two people working on the back end as this had a heavier workload, then one person working on the front end. We communicated effectively through group meetings and a group chat for regular updates, as well as using GitHub to review and merge our work. There were no major issues working together cohesively and everyone contributed to the overall project outcome.

Suggestions for enhancements

- Further enhancement of the feedback system could be implemented with a limited number of answers so the data could be analysed more efficiently.
- The bot could be trained on more topics so its believability as being human would increase as topics can vary. Support for bots from other vendors could be added, as the client might want to create and train his own bot.
- The points system could be further worked on possibly scoring more points to users who guess faster, or exchange less messages.
- Matchmaking protocol/algorithm could be improved to be more scalable.