NTNU

Norwegian University of Science and Technology

TDT4240 — Software Architecture

Group 13 — Requirements

DrawGuess

Android

Sjøberg, Isak Olav
Weidel, Jacob
Færestrand, Benjamin
Trouchet, Ninon Lisa
Raeesi, Arya

Primary Quality Attribute: Modifiability

Secondary Quality Attribute: Usability and Performance

February 22, 2025

Contents

1 Introduction

1.1 Description of the Project

This document will outline a set of functional requirements and architectural attributes for a multiplayer game in Android. All documentation will be kept up to date, thus having a *Changes* section in section 6. The first iteration will set requirements (see section 2 and 3), from which the implementation phase can be started.

1.2 Description of the Game Concept

DrawGuess is a multiplayer Pictionary-style game in which players take turns drawing and guessing within set time limits. The game features a competitive point-based system and a real-time multiplayer experience in Android.

The game follows a structured sequence where players are assigned as either a *guesser(s)* or *drawer*. The *drawer* will sketch a given word, while the *guesser(s)* attempt to identify it. After each round, points are given based on response time and correctness. This process repeats for several rounds, until a final leaderboard appears. Figure 1 illustrates this workflow.

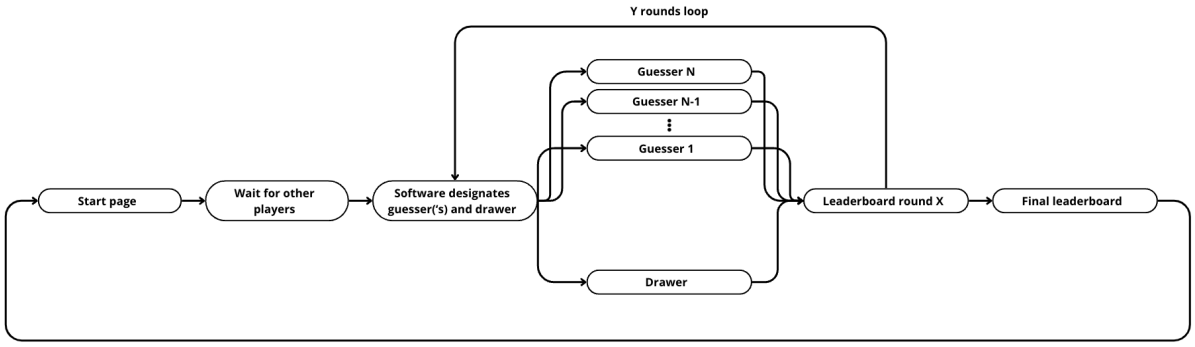

Figure 1: Game flowchart. The game consists of Y total rounds, looping until X (completed rounds) equals Y. In each round, one player is the *drawer* while N players take on the role of *guessers*.

With figure 1 in mind, figure 2 shows an illustration of the game's start page.

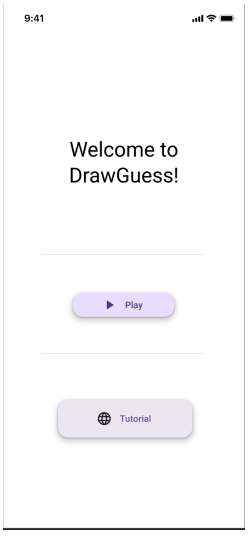

Figure 2: Illustration of the DrawGuess start page.

After clicking *Play*, the game will find opponents, and split up our user and opponents to one *drawer*, and one or more *guessers*.

Figure 3 shows the *guessers* screen during a round. The left screen shows the *guessers* screen before the *drawer* has finished drawing, while the right screen shows the *guessers* screen after finished drawing. When the drawing has appeared on the *guessers* screen, there's a timer to finish drawing. Faster correct guesses give more points, while wrong guesses give 0 points. If a *guesser* guesses correctly, the *drawer* will receive more points depending on how fast a *guesser* guessed correctly.

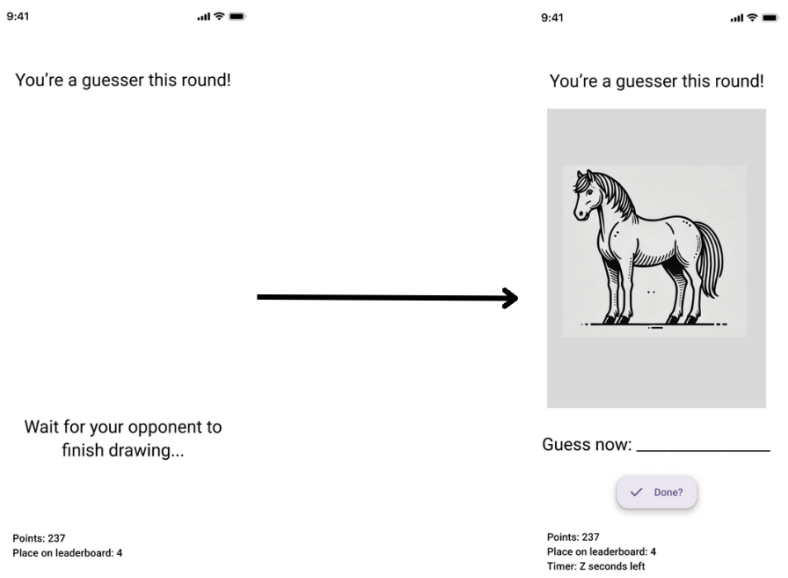

Figure 3: Illustration of one round for a *guesser*. Note that the horse on the right screen was created using AI [1].

Furthermore, figure 4, shows the *drawer's* screen during a round. The left screen shows the *drawer's* screen before drawing, and the right screen shows the screen after drawing. Faster drawing gives more points if a *guesser* is able to guess correctly. If a *guesser* isn't able to guess correctly, the drawer will receive 0 points. For mathematical expressions regarding point distribution, see appendix C.

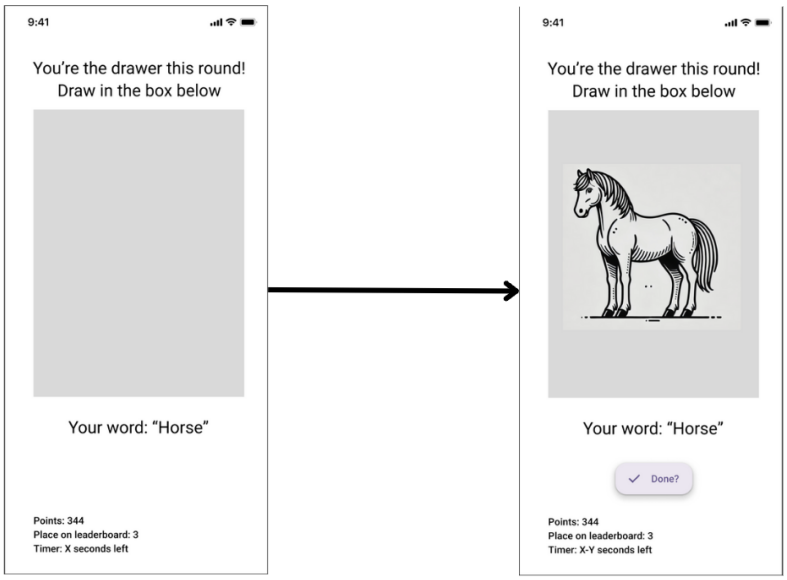

Figure 4: Illustration of one round for a *drawer*. The timer starts with X seconds left, and in the illustration the drawing is done with X-Y seconds left. Note that the horse on the right screen was created using AI [1].

After the *guessers* and *drawer* are done, the game goes to the round X leaderboard (see figure 5).

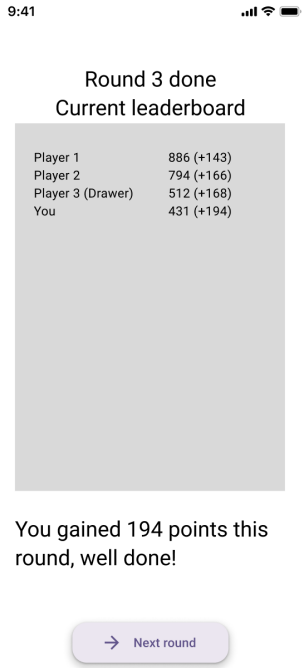

Figure 5: Illustration of the leaderboard after a round. In this example it's the third round, and our player is in last place.

Finally, after Y rounds (see figure 1), the final leaderboard appears (see figure 6). After the final leaderboard, our player can return to the start page when clicking *Homepage* in figure 6.

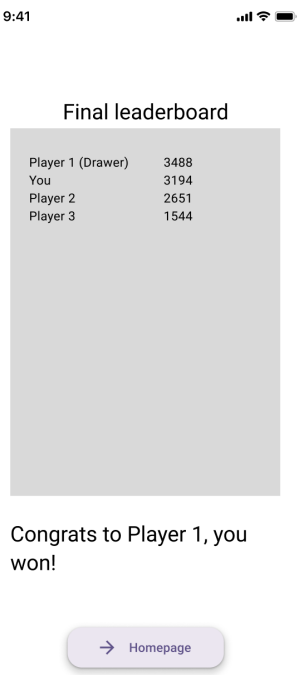

Figure 6: Illustration of the final leaderboard.

In short, in DrawGuess players start in the *start page*, play for Y rounds and return to the *start page* after the *Final leaderboard* appears (see figure 1).

1.3 Report Structure

This report entails several parts of DrawGuess. The report structure is as follows:

1. **Introduction:** an overview of DrawGuess, including its purpose and core gameplay mechanics.
2. **Functional Requirements:** the essential features and functionalities the game must implement.
3. **Quality Requirements:** the non-functional requirements such as modifiability, usability, and performance.
4. **COTS Components and Technical Constraints:** describes the external software, tools, and technologies used in the project.
5. **Issues:** highlights challenges encountered during development and their impact on the project.
6. **Changes:** documents modifications made to the report.
7. **Contributions:** details individual team members contributions to the project.
8. **Bibliography:** lists sources and references used throughout the project documentation.
9. **Appendix:** supplementary material such as diagrams, formulas, or additional explanations.

2 Functional Requirements

This section describes the main functional requirements of DrawGuess. Table 1 outlines our functional requirements.

Table 1: Functional requirements for DrawGuess.

ID	Requirement	Priority
FR1	The user should be able to start a new game session.	High
FR2	The system must assign one player as the *drawer* and the rest as *guessers*.	High
FR3	The *drawer* should be able to sketch on a canvas using touch gestures.	High
FR4	The *guessers* should see the drawing after the *drawer* has submitted it.	High
FR5	The *guessers* must be able to submit guesses through text input.	High
FR6	The system should validate guesses and determine correctness.	High
FR7	The system should award points based on correctness and response time of the *guessers* and *drawer*.	High
FR7.1	The system should award points based on the mathematical expressions outlined in appendix C.	Medium
FR8	The leaderboard should update after each round.	Medium
FR9	The game should continue for 10 rounds before displaying the final leaderboard.	High
FR10	The final leaderboard should be displayed at the end of the game session.	High
FR11	The user should be able to return to the main menu after the game ends.	Low
FR12	The system must allow users to reconnect if they get disconnected mid-game.	Medium
FR13	The system must ensure a maximum of 8 players per game session.	High
FR14	The system must notify all players when a new round begins.	Medium
FR15	The game should prevent spam guessing limiting guesses to one per round.	High
FR16	The drawer should receive points based on how quickly a correct guess is made.	High

3 Quality Requirements

We have chosen modifiability as our primary quality requirement, while usability and performance are our secondary requirements.

3.1 Modifiability

There are four quality requirements entailing modifiability, and are outlined in table 2, 3, 4 and 5 below.

Table 2: Add a new word.

ID	**M1**
Source	Developer
Stimulus	Wishes to add a new word.
Artifacts	Word list
Environment	Design time
Response	The new word is added without modifying the game logic and appears in the game.
Response measure	The word is fully functional within 30 minutes.

Table 3: Change scoring rules.

ID	M2
Source	Developer
Stimulus	Wishes to modify how points are awarded.
Artifacts	Scoring system
Environment	Design time
Response	Changes are applied through a configuration file without modifying core code.
Response measure	Scoring adjustments take effect in the next game session without requiring a restart.

Table 4: Change maximum player count.

ID	M3
Source	Developer
Stimulus	Wishes to change the maximum amount of players in one game.
Artifacts	Configuration file
Environment	Design time
Response	The player limit can be adjusted in the config file without modifying game logic.
Response measure	Changes take effect immediately after restarting the matchmaking server.

Table 5: Add a new drawing tool.

ID	M4
Source	Developer

Stimulus	Wishes to add a new drawing tool (e.g., "Eraser", "Fill Tool").
Artifacts	Drawing System
Environment	Design time
Response	The tool is implemented as a separate module and integrates with the existing drawing system.
Response measure	The tool is fully functional and accessible within 30 minutes.

3.2 Usability

Furthermore, there are two quality requirements entailing usability, and are outlined in table 6 and 7 below.

Table 6: First-time user.

ID	U1
Source	User
Stimulus	A new player enters the game for the first time.
Artifacts	System
Environment	Runtime
Response	User is introduced to game rules and gameplay through a README file in the DrawGuess GitHub repository.
Response measure	Finish the tutorial within 3 minutes and understands the basics of DrawGuess.

Table 7: Show remaining round time on the screen.

ID	U2
Source	User
Stimulus	A user wants to know how much time is left in the round.
Artifacts	UI components, timer system
Environment	Runtime
Response	A countdown timer is displayed on the screen during each round.
Response measure	The timer updates every second and remains visible throughout the round.

3.3 Performance

Lastly, there are three quality requirements entailing performance, and are outlined in table 8, 9 and 10 below.

Table 8: Low latency for drawing submission.

ID	P1
Source	User
Stimulus	A *drawer* sketches on the screen.

Artifacts	Networking system
Environment	Runtime
Response	The drawing is transmitted to all guessers immediately after submission.
Response measure	The delay between submission and visibility for guessers must not exceed 200ms.

Table 9: Game reconnection time.

ID	P2
Source	System
Stimulus	A player disconnects and attempts to reconnect.
Artifacts	Networking system
Environment	Runtime
Response	The system restores the player's session.
Response measure	Players must be able to reconnect within 20 seconds without losing progress.

Table 10: Low latency for score calculation and distribution.

ID	P3
Source	System
Stimulus	A round ends, and the system needs to calculate and distribute scores to players.
Artifacts	Scoring system
Environment	Runtime
Response	The system processes and updates scores for all players immediately after the round ends.
Response measure	The calculation and display of updated scores must complete within 500ms after the round ends.

4 COTS Components and Technical Constraints

This section entails the COTS (Commercial-off-the-shelf) components and technical constraints that impact the development of DrawGuess.

COTS components refer to third-party technologies integrated into the system, while technical constraints outline the requirements imposed to develop a functional system. These factors influence both the architecture and implementation of the game.

4.1 COTS Components and Their Technical Constraints

4.1.1 Firebase Firestore

Firebase Firestore is a cloud-hosted NoSQL database from Google that provides real-time synchronization between clients [2]. It offers automated scalability and integration with Android applications, making it well-suited for mobile development [3].

In DrawGuess, Firestore is used to store and synchronize game session data, including player states, round progress, and leaderboard scores, ensuring that all clients receive updates in near real-time. Firestore's real-time listeners allow the game to quickly reflect changes, such as when a new drawing is submitted or when a round ends. Additionally, Firestore manages matchmaking, helping to connect players to game sessions.

One technical constraint of Firestore is its lack of native SQL query support, as it follows a NoSQL document-based structure. This can introduce challenges in complex data retrieval and requires structured indexing for efficient queries. Despite this, Firestore's real-time capabilities and scalability make it a strong fit for the multiplayer architecture of DrawGuess.

4.1.2 Socket.IO

Socket.IO is a JavaScript library for real-time bidirectional communication between clients and a Node.js backend. It facilitates low-latency messaging by utilizing WebSockets as the primary transport mechanism, with automatic fallback to HTTP polling when necessary [4].

In DrawGuess, Socket.IO is used to synchronize the state of the game, including leaderboard updates and round coordination, ensuring a multi-player experience (see Figure 1).

An advantage of Socket.IO is its built-in handling of connection loss and automatic reconnection, enhancing robustness in network-fluctuating environments [5]. However, a technical drawback is the added latency compared to raw WebSockets, as the abstraction layer introduces overhead for managing fallback transport methods [6].

4.1.3 OkHttp WebSocket

OkHttp WebSocket is a WebSocket client library for Java and Android, used for real-time, low-latency communication [7]. It provides a robust and efficient way to establish WebSocket connections without needing manual handling of HTTP upgrades [7].

In DrawGuess, OkHttp will be used for real-time communication between the Android client and backend server, to establish message exchange with low overhead.

One advantage of OkHttp is its automatic connection handling and reconnection capabilities,

ensuring a stable multiplayer experience. Additionally, it optimizes data transfer efficiency by reducing redundant network requests compared to traditional HTTP polling [8]. However, a technical constraint is that OkHttp doesn't natively support advanced bidirectional messaging features like event-based communication, requiring additional logic to handle complex interactions between players [7].

There are more technical constraints to consider in this project that aren't from COTS components. These are specified in appendix A. In addition, the technologies used in this project and how they cover the technical and quality requirements from section 2 and 3 is explained in appendix B.

5 Issues

This section will be updated after future iterations.

6 Changes

Table 11 below outlines our change history.

Table 11: Change history.

Date	Change history	Comments
February 22, 2025	First released version.	None.

This section will be updated after future iterations.

7 Contributions

Table 12: Overview of contributions made by each group member.

Group member	Hours worked
Arya Raeesi	16
Isak Olav Sjøberg	4
Benjamin Færestrand	2.5
Jacob Weidel	1.5
Ninon Lisa Trouchet	0

Bibliography

[1] OpenAI, 2025. AI generated image of a horse using DALL-E.

[2] Firebase: *Understand real-time queries at scale*, 2025.

[3] Jacobsson, Per: *Building scalable real time applications with firestore*, 2022.

[4] Socket.IO: *How it works*, 2024.

[5] Diaconu, Alex: *Scaling socket.io - practical considerations*, 2023.

[6] VideoSDK: *Socket.io vs websocket: Comprehensive comparison and implementation guide*, 2025.

[7] Logic, NG: *Introduction to android websocket*, 2024.

[8] N, Leo: *Cracking the interview: Networking essentials for android engineers*, 2025.

[9] Developers, Android: *Set up android emulator networking*, 2025.

[10] Developers, Android: *Reduce your app size*, 2025.

[11] Olatunji, Damilola: *Dealing with cpu-bound tasks in node.js*, 2024.

A Other Components and Their Technical Constraints

While Android Studio and Node.js aren't strictly COTS components, they impose technical constraints on the development process and must be considered in the system design and implementation.

A.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Android application development, providing a toolset for building, testing, and deploying Android applications.

For DrawGuess, Android Studio serves as the primary development environment, enabling features such as real-time debugging, emulator testing, and performance profiling. It offers tight integration with Firebase, which simplifies database interactions and synchronization.

One technical constraint of Android Studio is network latency inconsistencies when using emulators, leading to limitations in real-time multiplayer games [9]. Additionally, the APK size limitations lead to limitations in dependency management, and must be carefully managed [10].

A.2 Node.js

Node.js is a JavaScript runtime environment built on Chrome's V8 engine, designed for building scalable, event-driven applications. It operates on a non-blocking, single-threaded event loop, making it well-suited for handling concurrent connections with low latency.

In DrawGuess, Node.js is used for handling game logic and multiplayer coordination in the backend. It facilitates communication between players using Socket.IO, ensuring synchronization of the leaderboard, game states, and round progress.

A technical constraint of Node.js is its single-threaded nature, which can become a bottleneck if the game scales to a large number of active users. While non-blocking I/O improves efficiency, CPU-intensive tasks like processing game data, may impact performance [11]. Another constraint is the WebSocket communication that relies on a stable internet connection, which could decrease the user experience in poor network conditions.

B How the Different Technologies Cover Technical and Quality Requirements

B.1 Firebase Firestore

The use of Firebase Firestore will help cover FR6, FR8 and FR12 from table 1. FR6 is covered because Firestore stores and gets data with low latency, due to its near real-time nature. FR8 is covered because of Firestore stores data. Lastly, FR12 gets partly covered by Firestore because DrawGuess must save the current game state if a disconnection occurs.

In addition, Firebase Firestore will partly cover quality requirement P3 from table 10. This is due to Firestore's near real-time nature and score data being stored there, helping with calculation and display being less than 500ms, adhering to the response measure from table 10.

B.2 Socket.IO

The use of Socket.IO will help cover FR7 and FR14 from Table 1. Firstly, Socket.IO handles communication between the clients and server when calculating and distributing points to the clients, therefore helping to cover FR7. Additionally, Socket.IO is responsible for the communication channel between the server and clients, giving DrawGuess the ability to notify all players when a new round begins, covering FR14.

In addition, Socket.IO helps cover quality requirement P3 from Table 10. This is because Socket.IO facilitates real-time communication between the server and clients, ensuring score updates have low latency after each round.

B.3 OkHttp WebSocket

The use of OkHttp WebSocket will help cover FR4 from Table 1. OkHttp is responsible for transmitting the *drawer's* submitted drawing to the guessers, therefore covering FR4. This ensures the drawing is received only after submission, preventing real-time visibility during the drawing phase.

Additionally, OkHttp helps cover quality requirement P1 (see Table 8) with low latency between the drawer's drawing submission and the guessers receiving the drawing. Since WebSockets provide persistent, low-overhead connections compared to HTTP polling, OkHttp ensures fast delivery of the drawing data with minimal delay.

B.4 Android Studio

The use of Android Studio will cover multiple functional requirements from Table 1. Primarily, Android Studio supports the development of the application, including FR3 (sketching on a canvas using touch gestures), FR5 (text input for guessing), and FR11 (returning to the main menu after a game session). Additionally, Android Studio provides built-in testing and debugging tools, which help meet FR6 (validating guesses) and FR15 (preventing spam guessing by limiting guesses per round).

Moreover, Android Studio contributes to meeting usability and modifiability quality requirements. Features like real-time debugging, emulator testing, and performance profiling assist in ensuring U1 (introducing new players through the README). Furthermore, its built-in devel-

opment tools simplify code modifications, which is essential for M1-M4 in Tables 2-5, supporting modifiability and future game improvements.

B.5 Node.js

The use of Node.js will cover multiple functional requirements from Table 1. Node.js serves as the backend for DrawGuess, handling multiplayer interactions. This helps cover FR7 (awarding points based on correctness and response time), FR8 (updating the leaderboard), and FR12 (allowing players to reconnect if they get disconnected mid-game).

For quality requirements, Node.js plays a crucial role in performance-related aspects, specifically P3 (low latency for score calculation and distribution). Since Node.js is single-threaded and event-driven, it efficiently manages concurrent players, ensuring quick response times.

C Mathematical Expressions

For the point distribution system, mathematical expressions are needed. We decided to cap the maximum amount of points per round to 500 points. In addition, points are to be given as integers.

Note that the points given in the illustrations in section 1 don't follow the expressions, but were put there for better understanding of the game system.

There are essentially two expressions, one for a *guesser* and one for the *drawer*.

C.1 Mathematical Expression for Point Distribution for a *Guesser*

The score for a *guesser* depends on how quickly they guess the correct answer. Let T_{max} be the total amount of time per round, T_{guess} be the guessing time, $S_{max} = 500$ be the maximum score per round, and $S_{guesser}$ be the *guessers* score per round. This leaves us with the expression in equation 1 below:

$$S_{guesser}(T_{guess}) = \begin{cases} \lfloor S_{max}(1 - \frac{\log(T_{guess}+1)}{\log(T_{max}+1)}) \rfloor, & 0 \leq T_{guess} \leq T_{max} \\ 0, & \text{Otherwise} \end{cases} \tag{1}$$

Note that the floor function is used to ensure integers being given as points.

C.2 Mathematical Expression for Point Distribution for the *Drawer*

Using the variables defined in section C.1, we can create a mathematical expression for the *drawer* too. Let S_{drawer} be the total points for the *drawer*, $N_{guessers}$ be the amount of *guessers*, and we can get the formula in equation 2 below:

$$S_{drawer} = \lfloor \frac{1}{N_{guessers}} \sum_{i=1}^{N_{guessers}} (S_{guesser,i}(T_{guess,i})) \rfloor, \quad 0 \leq T_{guess,i} \leq T_{max} \quad \forall i \tag{2}$$

where $S_{guesser,i}(T_{guess,i})$ uses equation 1 for all players, where i denotes each player in a sum.

Note that equation 2 uses the same methodology as equation 1, but adds an averaging and summation mechanism of all *guessers*. This ensures that the *drawer's* performance is evaluated based on the collective difficulty of the guessing process, making it a fair scoring system. By considering multiple *guessers'* response times, the game rewards clear and recognizable drawings rather than simply granting a fixed score per round.

Lastly, equation 1 and 2 use the $\log_{10}$-base, since it will lead to extreme point reductions for larger T_{guess}. This will give *guessers* incentive to answer as quick as possible, leading to a more exciting and fast-paced game environment.