NTNU

Norwegian University of Science and Technology

TDT4240 — Software Architecture

Group 13 — Implementation

Android

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

Chosen COTS: Firebase Firestore, Firebase Storage, Socket.IO, Render, LibGDX, AndroidSDK

Primary Quality Attribute: Modifiability

Secondary Quality Attribute: Usability and Performance

April 22, 2025

Contents

1 Introduction

1.1 Description of the Project and Implementation Phase

The goal of this project was to design and implement a software architecture for a multiplayer mobile game for Android, developed as part of the Software Architecture course at the Norwegian University of Science and Technology.

This report documents the full development process, including the system's design, implementation, and testing phases. The game was built based on a set of functional and quality requirements, with architectural decisions made accordingly. Testing was conducted to verify that these requirements were fulfilled. The report includes multiple class diagrams to illustrate the system's structure and also presents relevant test results.

In addition, the report features a user manual describing how to install, run, and play the game. It also contains an evaluation of how well the final implementation aligns with the original architecture. Finally, the report reflects on encountered challenges and issues.

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 turn-based multiplayer game with real-time interactions in the guessing phase.

The game follows a structured sequence where all players are first assigned as *drawers* and then as *guessers*. During the drawing phase, the players are given a word each and draw that word on their screens. Then they go over to the guessing phase, where players guess other players' drawings. If a player guesses correctly, they gain points. The guessing phase repeats until all players have guessed all drawings, excluding their own.

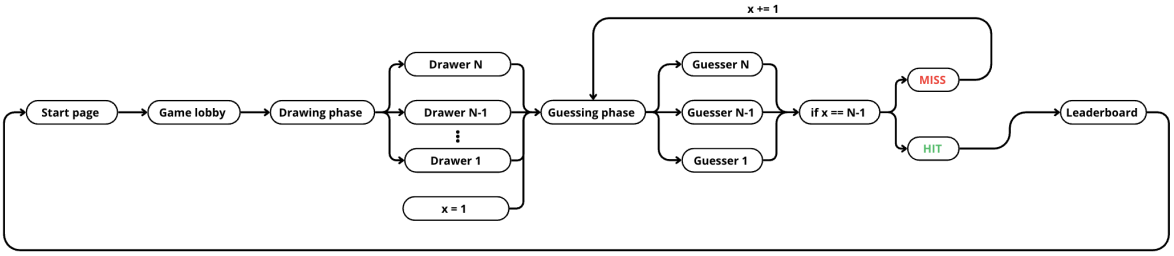

Figure 1: Game flowchart. The game consists of $N-1$ total rounds during the guessing phase, looping until x (completed rounds) equals $N-1$. In each round, a player gains points for correct guesses and receives none for wrong ones.

For a more comprehensive view of the UI and gameplay flow of DrawGuess, please see section 3.2.

2 Design & Implementation Details

This section presents the design and implementation details of the game DrawGuess and its underlying architecture. The game is developed using the LibGDX framework with Firebase as the backend. Several architectural patterns were adopted to support the development of a modular and maintainable system. Each subsection describes one of these patterns, how it was realized, and reflections on its applicability and effectiveness in our project.

2.1 Architectural Patterns

2.1.1 Model-View-Controller

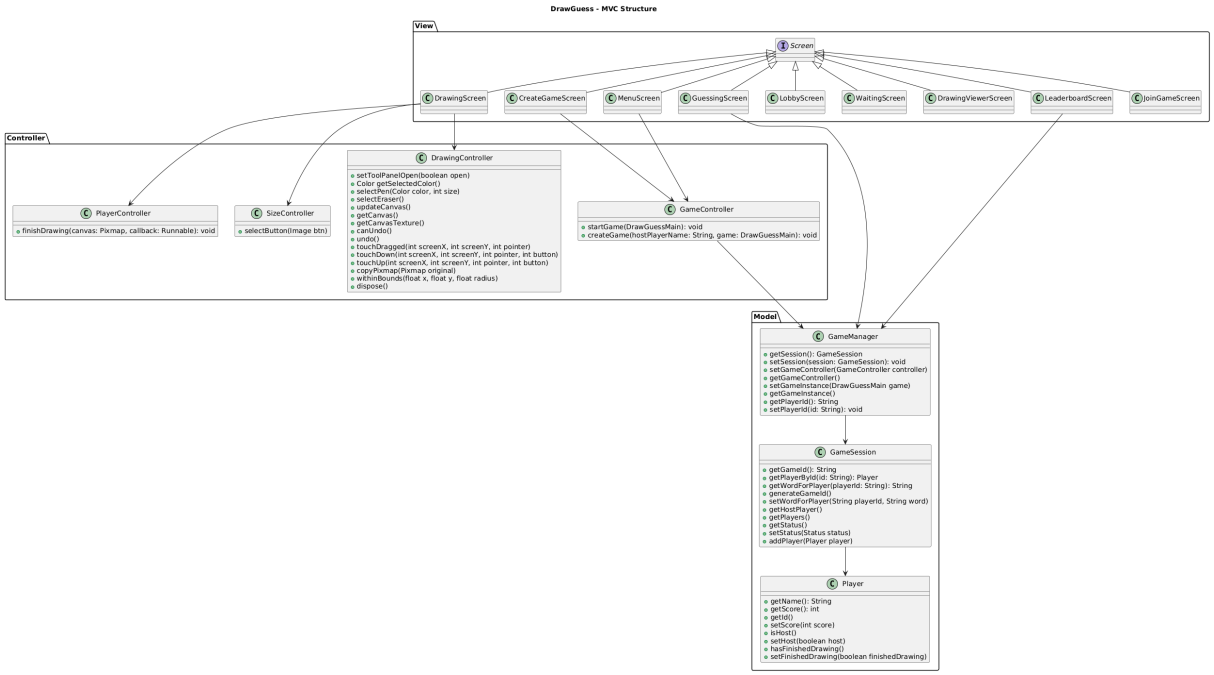

Figure 2: MVC diagram

Figure 2 illustrates the implementation of the Model-View-Controller (MVC) pattern in DrawGuess. In this architecture, views are realized by the various game screens, all implementing LibGDX `Screnn` interface, while controllers such as `GameController` handle user interactions and coordinate between views and the underlying data. The model includes all components not directly related to UI, such as `GameSession`, game logic, and drawing tools.

The MVC pattern was introduced to achieve separation of concerns and increase modifiability. However, during development, we observed that LibGDX's structure does not lend itself naturally to traditional MVC implementations. The GameStateManager-based view management made view abstraction more complex, and the benefits of MVC became less apparent. Despite these limitations, the MVC pattern helped separate responsibilities. However, given LibGDX's structure, a more reactive or observer-based approach might have better suited our needs.

2.1.2 Client-Server

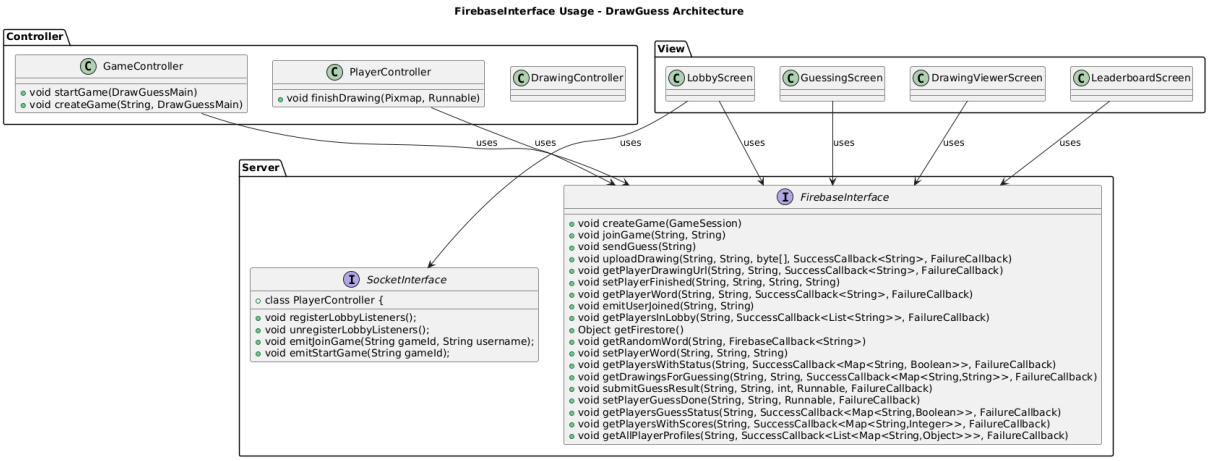

Figure 3: ClientServer diagram

Figure 3 demonstrates the realization of the Client-Server pattern in DrawGuess, with Firebase acting as the central server. Communication is handled through the `FirebaseInterface`, which defines an abstract interface for all interactions with Firebase, such as uploading drawings, retrieving player data, and managing game sessions. This interface is used throughout the controllers to seperate business logic from backend implementation details.

Although the game could benefit from a peer-to-peer architecture for real-time interactions, the Client-Server model was chosen due to its simpler implementation and Firebase integration. Firebase's real-time database allows for acceptable responsiveness in a turn-based drawing game. Nevertheless, its limitations in handling truly synchronous interactions led us to design the system with modifiability in mind. Future refactoring to enable peer-to-peer connections or a hybrid approach would primarily require replacing the `FirebaseInterface` implementation, without affecting core gameplay mechanics or UI components.

2.1.3 Entity-Component-System

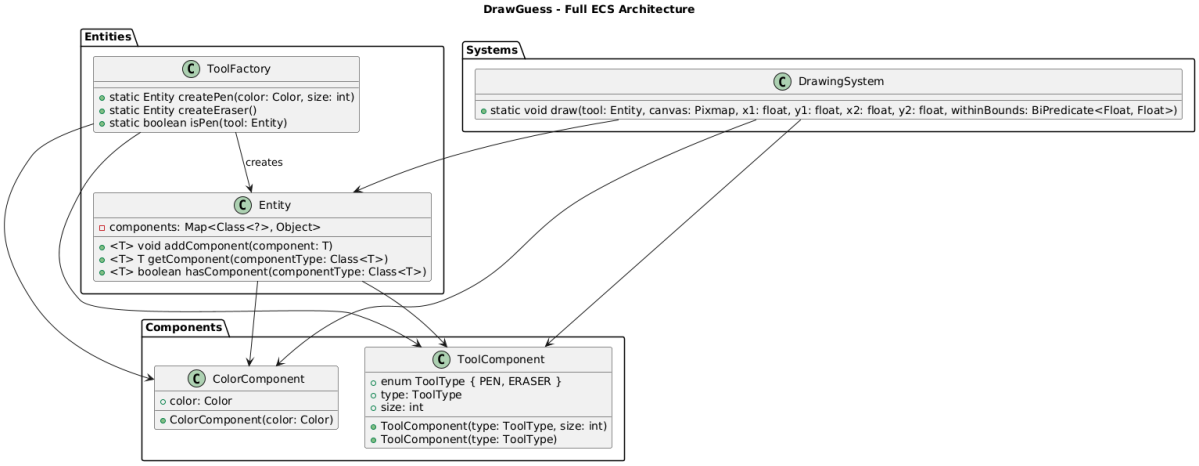

Figure 4: ECS diagram

Figure 4 illustrates the implementation of the Entity-Component-System (ECS) pattern. While LibGDX provides the Ashley ECS library, we developed a lightweight custom version tailored to

our needs. Entities such as pens and erasers are constructed using a combination of components, such as `ToolComponent` and `ColorComponent`, which encapsulate pure data without behavior. These entities are created through the `ToolFactory`, ensuring consistency and abstraction from manual instantiation.

Systems such as the `DrawingSystem` are responsible for applying behavior to entities. For instance, the drawing logic operates on entities that contain the appropriate tool and color components. This pattern promotes separation between data and behavior, reduces inheritance hierarchies, and allows for highly composable logic.

While the game does not use a full ECS engine, this partial implementation was beneficial for extensibility and clarity. Each component and system has a single responsibility, and additional drawing tools or logic changes can be implemented without altering existing code, adhering to the open/closed principle [1, s. 57].

2.2 Patterns

2.2.1 Singleton

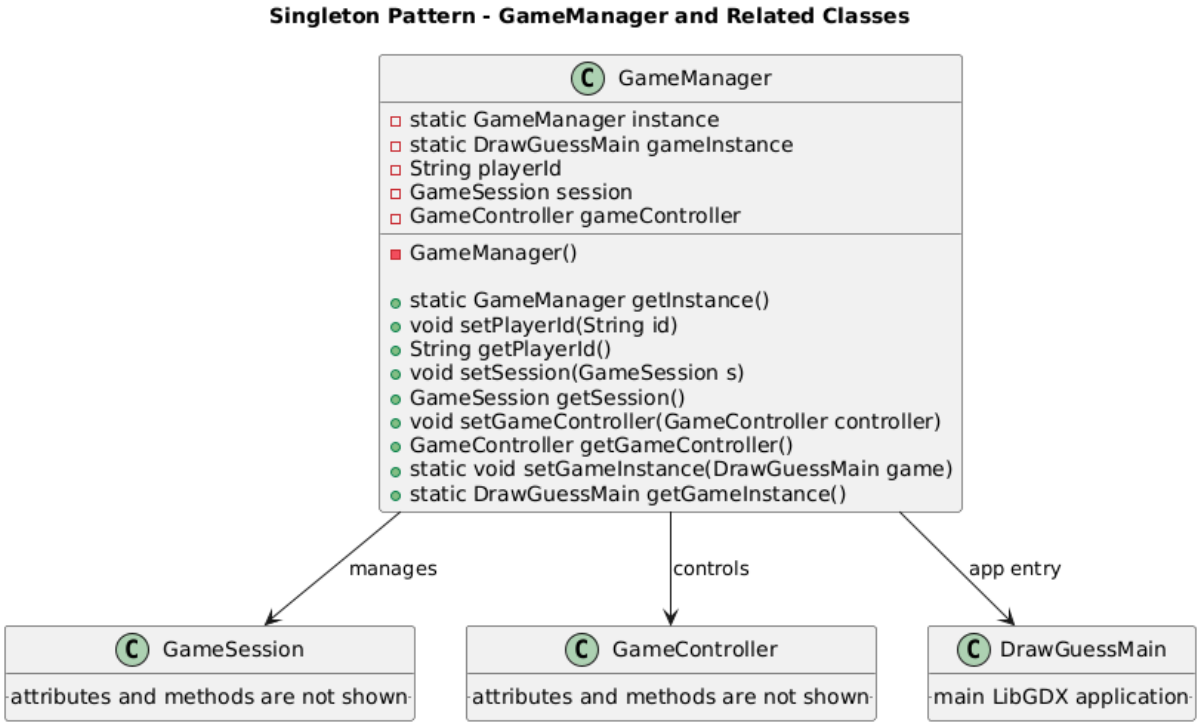

Figure 5: Singleton diagram

As shown in Figure 5, the `GameManager` class implements the Singleton pattern. This is done through a private static instance and a public static accessor method. The `GameManager` serves as a central access point for the current game session, player ID, and controller references.

By enforcing a single global instance, the game avoids inconsistencies across various states and controllers. This approach, although simple, is effective in a game with a linear flow where centralized control is required. In a more complex or multiplayer environment, this could be replaced by a dependency injection framework to improve testability and flexibility.

2.2.2 Factory

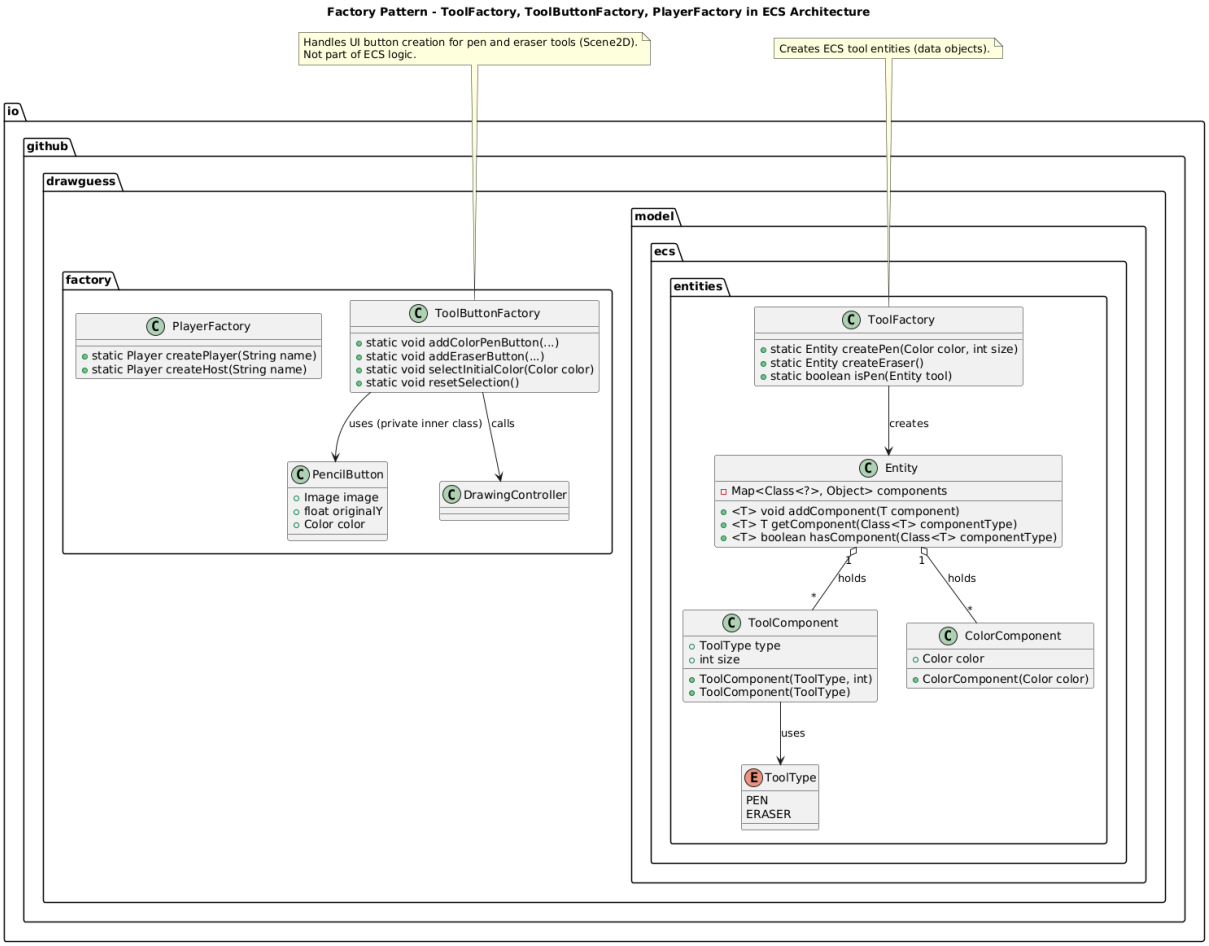

Figure 6: Factory diagram

Figure 6 illustrates the use of the Factory pattern across multiple components of the DrawGuess architecture. The diagram shows three factory classes. First is `ToolFactory`, which encapsulates the creation logic for ECS-based game tools like pens and erasers. Seocndly, `ToolButtonFactory`, responsible for generating the Scene2D UI buttons that allow players to select different drawing tools. Lastly, `PlayerFactory`, which provides methods for consistently creating player instances, including hosts and participants.

The factories abstract away the instantiation and configuration details of complex objects, providing a clear separation between object creation and usage. For example, the drawing system interacts with abstract `Entity` instances representing tools without knowing their internal component setup. Similarly, UI logic in the drawing screen uses `ToolButtonFactory` to manage tool selection visuals without needing to handle drawing logic directly.

This separation of concerns not only improves maintainability but also supports scalability: new tool types, drawing modes, or player roles can be introduced by extending the corresponding factory classes without modifying the code that uses these objects.

The Factory pattern has proven to be an effective design choice in this ECS-inspired architecture, contributing to cleaner, more modular, and testable code across both game logic and UI layers.

3 User Manual

This section will describe how you install and run DrawGuess. Section 3.1 outlines the steps you have to do before playing DrawGuess, while section 3.2 describes the gameplay itself.

3.1 How to Install, Compile and Run DrawGuess

The process for installing, compiling and running DrawGuess is described in section 3.1.1, 3.1.2 and 3.1.3.

3.1.1 Install

Firstly, you download DrawGuess by cloning the project (from the main branch) from this GitHub repository. The command for cloning the repository through the terminal is described below.

```
1 git clone https://github.com/isakosntnu/TDT4240.git
```

Code 1: Cloning the DrawGuess project from GitHub.

Once cloned, open the project folder in Android Studio. When prompted, allow Gradle to sync the project.

3.1.2 Compile

Once the project has been opened and Gradle has finished syncing, Android Studio will automatically index the project and make it ready for compilation. To compile the project click on Build > Make Project in the top menu. Android Studio will compile the code and check for any errors. No additional terminal commands are necessary, as all compilation is handled by the IDE.

3.1.3 Run

To run DrawGuess on an emulator open the Run menu and click Run 'android'. If you don't have an emulator configured, go to Tools > Device Manager to create one. After this, the app will be built, installed, and launched automatically on the selected emulator.

3.2 How to Play DrawGuess

This section describes step-by-step how to play DrawGuess. It contains the different screens a player can access, the different alternatives per screen, as well as how you play DrawGuess in practice. Note that the sequence itself is illustrated in Figure 1.

3.2.1 Home Menu

First and foremost, the home menu of DrawGuess (see Figure 7) has two alternatives, either you start a game or join a game.

Figure 7: The home menu of DrawGuess.

The two pencils in Figure 7 are both buttons. Section 3.2.2 goes through the start game sequence, while section 3.2.3 explains the join game sequence.

3.2.2 Start Game

When you reach the start game screen (Figure 8), you have two alternatives. Firstly, you can create a new game by writing your game name in the "Write your name" box, and then tap on the "Create New Game" button. In that case, you will reach the game lobby (see Figure 10). Alternatively, you can go back to the home menu by tapping on the button in the top left corner in Figure 8.

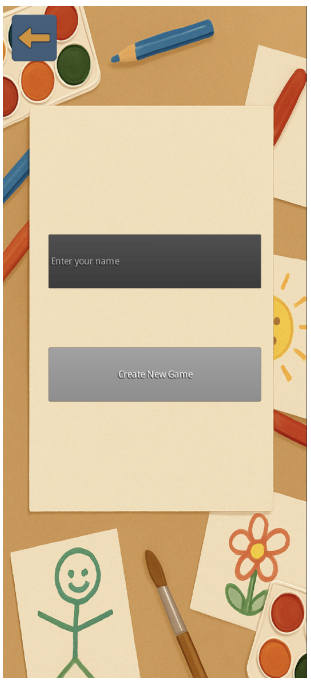

Figure 8: Start game screen of DrawGuess.

3.2.3 Join Game

In the join game screen (see Figure 9), you have two text input boxes and two buttons. Firstly, the top text input box ("Enter Game PIN") is where you write the game PIN of the game you want to join. The second text input box ("Enter Your Name") is where you write your chosen game name. Secondly, the button "Join Game" can be tapped at when you have filled the two text input boxes, which will lead you to a game lobby (see Figure 10). Lastly, the button on the top left corner leads you back to the home menu (see figure 7).

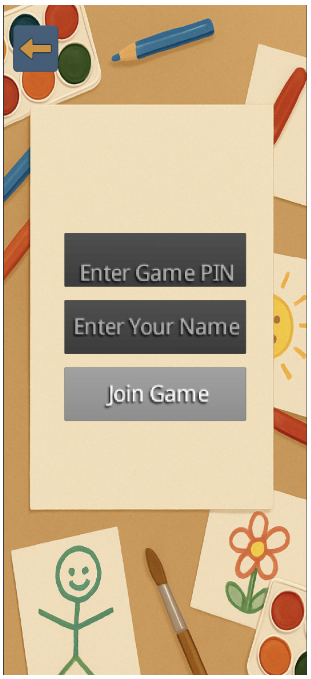

Figure 9: Join game screen of DrawGuess.

3.2.4 Game Lobby

In the game lobby, all players in that game pin will be given.

Figure 10: Game lobby of DrawGuess. Note that "host" has joined the game through the start game screen, while "player 1" has joined through the join game screen. The player names in this figure were chosen for anonymity sake, you can choose your own player name in the start game and join game screen.

When all players have joined, you can tap the "Start Game" button bottom of the screen. This will take you to the drawing phase of DrawGuess (see Figure 11).

3.2.5 Drawing Phase

When you've reached the drawing phase, you will have a canvas to draw on (see Figure 11). You can choose different colours (black, red, blue, green, yellow, purple, orange, cyan) and an eraser bottom of the screen. In addition, when tapping the button on the top right corner you can choose the size of drawing and erasing (see Figure 12). In addition, the button on the bottom left corner will remove your "latest drawing". Lastly, the button on the top left corner ("FINISH DRAWING") will submit your drawing, and you will wait for the drawing phase to end. The drawing phase goes on for 1 minute.

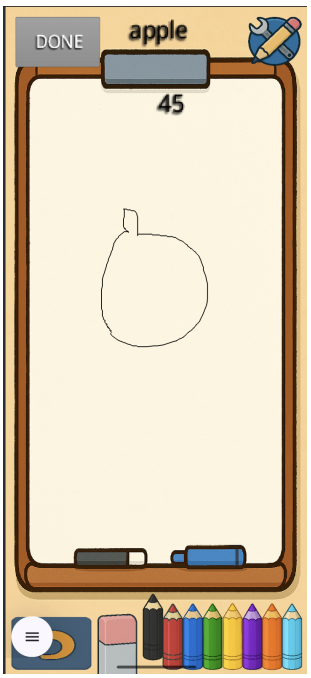

Figure 11: Drawing phase in DrawGuess.

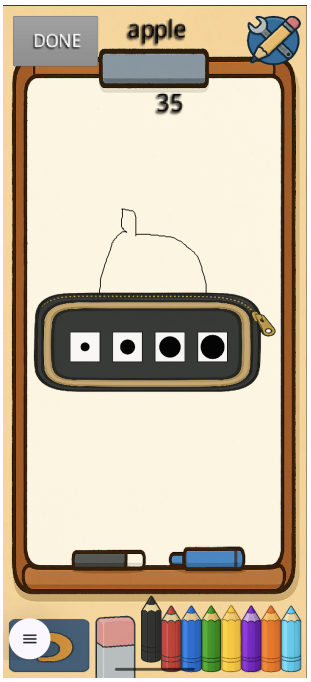

Figure 12: Drawing phase in DrawGuess. Choosing drawing and erasing size.

3.2.6 Guessing phase

The guessing screen contains an input text box for your guess ("Your guess..."), and a button for guess submission ("Guess") (see Figure 13). After submission you will get visual feedback if your guess was correct or wrong. In the case of a correct guess, you will also see how many points you gained from that guess. The point distribution follows the mathematical expression stated in appendix A. The guessing phase repeats until each player have guessed all the other players drawings, following the sequence from Figure 1. When all guesses have been submitted, the game finishes and players are sent to the leaderboard (see Figure 14).

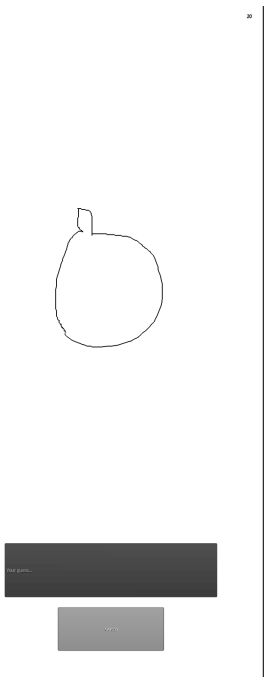

Figure 13: Guessing phase in DrawGuess.

3.2.7 Leaderboard

The leaderboard contains all players and their respective scores (see Figure 14). In addition, you can get back to the home menu (see Figure 7) by tapping the button in the top left corner.

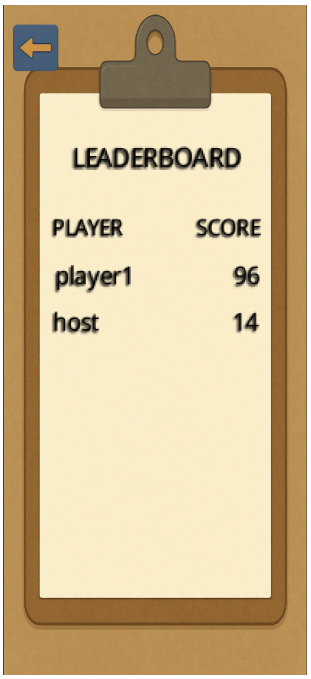

Figure 14: The final leaderboard in DrawGuess.

4 Test Report

This section outlines the tests done for all functional and quality requirements. For better context behind each requirement, please read the requirements document of DrawGuess.

4.1 Functional Requirements Tests

ID	FR1
Requirement description	The user should be able to start a new game session.
Executor	Jacob
Test date	April 21, 2025
Time used	30 seconds
Evaluation result	Success
Comments	Took 30 seconds from opening DrawGuess to starting a new game session where other players can join.

ID	FR2
Requirement description	The system must first assign all players as *drawers*, and after finishing drawing, assigning them as *guessers*.
Executor	Jacob
Test date	April 21, 2025
Time used	60 seconds
Evaluation result	Success
Comments	The drawing phase has a time limit of 1 minute, and only after that time limit, the game assigns all players as *guessers*.

ID	FR3
Requirement description	The *drawer* should be able to sketch on a canvas using touch gestures.
Executor	Isak
Test date	April 21, 2025
Time used	45 seconds
Evaluation result	Success
Comments	The drawing phase only starts after all players have joined the game lobby. Therefore, it took 45 seconds to wait for several players, and starting the drawing phase with touch gestures.

ID	FR4
Requirement description	The drawings should be visible to the players during the guessing phase (meaning after they have submitted it during the drawing phase).
Executor	Arya
Test date	April 21, 2025

Time used	10 seconds
Evaluation result	Success
Comments	The drawings only appear after all game players have submitted their drawings. During the drawing phase, you have 1 minute, leading to 1 minute for the time usage of this test.

ID	FR5
Requirement description	The *guessers* must be able to submit guesses through text input.
Executor	Arya
Test date	April 21, 2025
Time used	10 seconds
Evaluation result	Success
Comments	After a drawing has appeared on the player's screen, the system successfully gets the guesses through text input.

ID	FR6
Requirement description	The system should validate guesses and determine correctness.
Executor	Arya
Test date	April 22, 2025
Time used	10 seconds
Evaluation result	Success
Comments	The game successfully validates the guesses given by text input.

ID	FR7
Requirement description	The system should award points based on correctness and response time.
Executor	Arya
Test date	April 22, 2025
Time used	10 seconds
Evaluation result	Success
Comments	Took 10 seconds from receiving a drawing during the guessing phase, writing in a suggestion, and getting a score.

ID	FR7.1
Requirement description	The system should award points based on the mathematical expressions outlined in appendix A.
Executor	Benjamin
Test date	April 22, 2025
Time used	60 seconds
Evaluation result	Success

Comments	Took 60 seconds from receiving a score based on time usage, and validating that score with a calculator.

ID	FR8
Requirement description	The game's rounds should be the number of drawings submitted, before displaying the final leaderboard.
Executor	Benjamin
Test date	April 22, 2025
Time used	10 minutes
Evaluation result	Success
Comments	From opening DrawGuess to finishing a game took approximately 10 minutes. The leaderboard appeared after the final round.

ID	FR9
Requirement description	The final leaderboard should be displayed at the end of the game session.
Executor	Jacob
Test date	April 22, 2025
Time used	30 seconds
Evaluation result	Success
Comments	Took 30 seconds from the start of the last guessing phase to the final leaderboard appearing.

ID	FR10
Requirement description	The user should be able to return to the main menu after the game ends.
Executor	Isak
Test date	April 22, 2025
Time used	5 seconds
Evaluation result	Success
Comments	Button appears at the final leaderboard, which takes you back to the homepage. Worked as intended. The time used tracks from when the final leaderboard appeared to the executor getting back to the homepage.

ID	FR11
Requirement description	The system must allow users to reconnect if they get disconnected mid-game.
Executor	Isak
Test date	April 22, 2025
Time used	2 minutes
Evaluation result	Failure

Comments	Test conducted by the executor switching on and off WIFI mid-game. Logic was not implemented for handling this, resulting in the evaluation being a failure.

ID	FR12
Requirement description	The system must ensure a maximum of 8 players per game session.
Executor	Jacob
Test date	April 22, 2025
Time used	5 minutes
Evaluation result	Failure
Comments	Tried joining a game with 9 different emulators. The 9th emulator got access because code wasn't implemented for a max amount of players in a game session. Therefore, the evaluation was a failure.

ID	FR13
Requirement description	The system must notify all players when a new round begins.
Executor	Isak
Test date	April 22, 2025
Time used	30 seconds
Evaluation result	Failure
Comments	If the player keeps attention to the game, they will most likely notice that a new round has begun. At the same time, there isn't any specific notification that has been implemented when a new round begins. Therefore, the evaluation failed.

ID	FR14
Requirement description	The game should prevent spam guessing limiting guesses to one per drawing.
Executor	Isak
Test date	April 22, 2025
Time used	10 seconds
Evaluation result	Success
Comments	When trying to give a second guess in one round, the game prevents you from submitting it.

ID	FR15
Requirement description	Players should be able to send short messages to each other before the game starts.
Executor	Jacob
Test date	April 22, 2025
Time used	1 minute

Evaluation result	Failure
Comments	Code for a chat in the game lobby wasn't implemented.

ID	FR16
Requirement description	The game should offer an accessibility mode with larger buttons, high-contrast color schemes, and readable fonts to support visually impaired players.
Executor	Benjamin
Test date	April 22, 2025
Time used	1 minute
Evaluation result	Failure
Comments	Code wasn't implemented for different modes.

ID	FR17
Requirement description	The system should automatically flag and block offensive or inappropriate text input using a local profanity filter.
Executor	Isak
Test date	April 22, 2025
Time used	1 minute
Evaluation result	Failure
Comments	Code for a profanity filter wasn't implemented.

4.2 Quality Requirements Tests

ID	M1
Requirement description	Add a new word
Executor	Arya
Test date	April 21, 2025
Stimuli	Wishes to add a new word
Expected response measure	5 minutes
Observed response measure	2 minutes
Evaluation result	Success
Comments	Added a new word in the code.

ID	M2
Requirement description	Change scoring rules
Executor	Arya
Test date	April 22, 2025
Stimuli	Wishes to modify how points are awarded
Expected response measure	5 minutes
Observed response measure	3.4 minutes
Evaluation result	Success
Comments	Changed the formula for the scoring system in the code.

ID	M3
Requirement description	Add a new drawing tool
Executor	Isak
Test date	April 19, 2025
Stimuli	Wishes to add a new drawing tool (e.g., "Eraser", "Fill Tool")
Expected response measure	30 minutes
Observed response measure	27 minutes
Evaluation result	Success
Comments	Added the colour yellow in the drawing table. Took 27 minutes to find a suitable pen image and implement the code logic.

ID	U1
Requirement description	First-time user gameplay understanding
Executor	Internal: Isak. External: Nikolai Nore, Eivind Lillefosse, Martin Brekke Nilsen [1]
Test date	April 22, 2025
Stimuli	A new player enters the game for the first time
Expected response measure	90%
Observed response measure	93%
Evaluation result	Success
Comments	The internal executor concluded that the external executors understood the game during first-time play, if they successfully finished a round without help. The internal executor had 14 points of interest during the gameplay per external executor. Out of the total 42 points of interest, 39 of them passed, while 3 failed. This gave an observed response measure of approximately 93%.

ID	U2
Requirement description	Clear understanding of button functions
Executor	Internal: Isak. External: Nikolai Nore, Eivind Lillefosse, Martin Brekke Nilsen [1]
Test date	April 22, 2025
Stimuli	User navigates to a button in the UI.
Expected response measure	90% [2]
Observed response measure	97%
Evaluation result	Success
Comments	The internal executor concluded that the external executors understood the button functions based on testing them with all buttons in DrawGuess. The result of this testing gave an observed response measure of approximately 97%.

ID	P1
Requirement description	Low latency for a player to join an existing game lobby
Executor	Benjamin
Test date	April 22, 2025
Stimuli	A player joins a game
Expected response measure	3 seconds
Observed response measure	1.35 seconds
Evaluation result	Success
Comments	The executor tested the observed time measure by running two emulators simultaneously. The first emulator started a game, and established the game lobby. Then, when the second player tapped the button for joining that same lobby, the executor started a timer in his phone, and ended it when the second player appeared in the lobby. The timer ended with 1.35 seconds.

ID	P2
Requirement description	Low latency for score calculation and distribution
Executor	Jacob
Test date	April 22, 2025
Stimuli	A round ends, and the system needs to calculate and distribute scores to players.
Expected response measure	500 ms
Observed response measure	0.002375 ms
Evaluation result	Success
Comments	The executor got the observed response measure by using the function `calculatePointsWithTiming`. This function returned 0.002375 ms in the LogCat of Android Studio during testing.

[1]External contributors were needed for U1 and U2, because of them being dependent on users not being familiar with DrawGuess. Isak was present to document the external contributors performance.

[2]The 90% here refers to users' understanding each button's function within 3 seconds in DrawGuess.

5 Relationship with Architecture

5.1 ECS and Its Scope in the Final Implementation

The most notable inconsistency between our architecture and the final implementation lies in the scope of the ECS pattern. Initially, we did not plan to use ECS at all. However, after we started to implement code and build the project we realised the need for ECS. We then planned to apply ECS to manage multiple game components, including player interactions and scoring. However, during implementation, ECS was limited to the drawing system, focusing on components like `ToolComponent`, `ColorComponent`, and the `DrawingSystem` for handling drawing logic. This was becuase we felt the game components and such were all well seperated as it was, and we only felt the need for ECS in the drawing system. By concentrating ECS usage on drawing-related operations, we ensured a stable and efficient implementation for the most critical feature. Other parts of the game, such as guessing and scoring, were implemented using more straightforward object-oriented programming. This compromise allowed the team to prioritize delivering a functional product over follwing strictly the architectural blueprint.

5.2 Patterns: Planned vs. Implemented

The architecture document emphasized patterns such as Client-Server, MVC, and ECS. While the Client-Server model was implemented successfully using Firebase as the backend and WebSockets for real-time communication, the MVC pattern saw partial application.

- **Client-Server Pattern:** This was fully realized in the final implementation, with Firebase handling backend storage and WebSocket managing real-time interactions between the client and server. This decision aligned with the planned architecture and provided the required scalability and responsiveness.
- **MVC Pattern:** Although the UI (View) and game logic (Controller) were separated in most cases, some areas of the codebase blurred these lines. For instance, the guessing logic partially overlaps between UI and backend, resulting in tighter coupling than originally planned. This deviation occurred due to limited time and challenges in modularizing these interactions.
- **ECS Pattern:** As mentioned, ECS was implemented for the drawing system but not extended to other game features.

5.3 Tactics: Successes and Challenges

The architectural tactics from the document, including low coupling, high cohesion, and resource management, were implemented with varying degrees of success.

- **Low Coupling and High Cohesion:** Cohesion was achieved through the modular design of the `DrawingSystem`, where components like `ToolComponent` and `ColorComponent` handled specific responsibilities. However, certain areas, such as the scoring logic, became more tightly coupled as deadlines approached, resulting in less flexibility.
- **Efficient Resource Management:** This tactic was not very sucessfully implemented. The usage of Firebase and WebSockets ensured low latency, but we did not manage to compress the drawing data to vector graphics as this became too time consuming. Therefore, we could have imporved on the resource management.

5.4 Could These Inconsistencies Have Been Identified Earlier?

Some of these discrepancies could have been identified during the ATAM evaluation or earlier planning stages. Better team coordination and iterative reviews of the architecture during implementation could have reduced coupling in certain areas. With team members more synced and perhaps a more agile team approach to the project it would have been easier to have every team member aligned and reduce coupling. It also became clear to us that transmitting the drawing data as vectors were difficult to actually implement and this was something we could not achieve. At early stages we should have tried to focus on the most important aspects of our project and wait with features like this. As the ECS was not mentioned in out first architecture draft, it was impossible to discover the problems we faced with it in the first ATAM evaluation.

5.5 Conclusion

Overall, the implemented game aligns closely with the planned architecture, particularly in its use of the Client-Server model and modular design. However, some deviations, such as the limited scope of ECS and tighter coupling in specific modules, were necessary to accommodate time constraints and team dynamics. These changes highlight the importance of iterative architectural evaluations and flexible planning in future projects.

6 Challenges, Issues, and Lessons Learned

This section outlines the challenges, issues and lessons learned during the planning and development of DrawGuess.

6.1 Challenges

6.1.1 Model View Controller

The majority of our codebase adheres to the MVC pattern as we've chosen to apply it. When a game begins, the view is responsible for displaying the interface, the controller orchestrates the game's logic and interactions, and the model contains the core data structures such as players, current game state, and drawing content. Incorporating the MVC architecture introduced added complexity, mainly because libGDX was not originally designed with MVC in mind, making it difficult to build clearly separated and modular Views.

Although most parts of the system follow the MVC structure, there are still a few inconsistencies. For instance, the view typically fetches information directly from the model, but sometimes it also communicates with the controller, which breaks the strict boundaries of the pattern. Still, after adopting MVC, we quickly noticed its advantages. It significantly improved debugging, made it easier to expand functionality, and allowed multiple developers to work on different components without conflicts.

6.1.2 Server Setup

Another challenge we encountered was integrating the client-server architecture into our MVC-based system. In theory, combining these two patterns should be straightforward, but in practice it turned out to be more complex than expected. The MVC pattern emphasizes a strict separation of concerns, while client-server introduces asynchronous communication, shared state, and external control flow, particularly in multiplayer environments.

Hosting the server on Render added additional constraints, such as maintaining persistent state across connections and ensuring real-time synchronization between clients. It was not immediately clear where to place networking responsibilities within the MVC layers, whether game state updates from the server should flow directly into the model, be handled by the controller, or require an intermediate layer. Ensuring a clean separation while keeping the code maintainable and responsive required multiple architectural iterations. Despite the challenges, integrating the server properly allowed us to scale the multiplayer experience while maintaining a relatively clean structure.

6.2 Issues

6.2.1 Time Estimation and Architectural Planning

One issue we encountered was that our time usage significantly exceeded initial expectations. Although we anticipated the need to learn new tools and technologies, the process of designing, analyzing, and discussing the overall architecture took far longer than we had planned. A major reason for this was that it was our first time approaching architectural planning this thoroughly.

We underestimated the time required for aligning the client-server structure with the MVC pat-

tern, as well as coordinating how different components like libGDX and Firebase would interact. This experience taught us the importance of allocating sufficient time for the architectural phase in future projects. Having gone through this process once, we are now better equipped to realistically estimate the effort involved when defining technical structure at the start of a project.

Lastly, differences in team members' schedules, combined with one member not contributing at all without telling the other team members, created obstacles in balancing game implementation and producing well-documented deliverables. These factors impacted our ability to meet both coding and documentation expectations within the available time, but in return all the team members had to take on huge tasks which resulted in a steep learning curve over a short period.

6.2.2 Refactoring Became Risky Over Time

As the project evolved and the codebase expanded, refactoring naturally became a more delicate process. In the early stages, adjustments to the controller or model structures were straightforward, with limited dependencies to consider. However, as the system grew and more components, such as Firebase logic, UI updates, and multiplayer synchronization, became tightly integrated, even small changes could have far-reaching effects. A seemingly minor refactor in one area could unintentionally impact the flow of data or break synchronization across clients.

Because of this, we took a more cautious approach to structural changes later in development. Rather than risking instability in key features, we focused on keeping the system stable and predictable while prioritizing targeted improvements in isolated areas. This experience highlighted the importance of designing for modularity early on and reinforced how valuable automated testing can be in enabling safer refactoring in future projects.

6.3 What We Would Do Differently if We Started Again

6.3.1 Keep Feature Scope Slightly Smaller

Looking back, one of the key lessons we learned was the importance of maintaining a more focused and minimal feature scope in the early stages of development. While our ambition drove us to implement a wide range of features, such as real-time drawing synchronization, role-based multiplayer logic, and leaderboard tracking, each of these introduced considerable architectural and technical complexity. Although these features contributed to the final experience, they also required substantial effort to stabilize and made debugging and integration more difficult than necessary.

If we were to start over, we would adopt a stricter MVP-first approach, focusing solely on the essential gameplay loop and delaying non-critical features until the core mechanics were fully functional and tested. By narrowing the initial scope, we would have been able to iterate faster, reach a stable version earlier, and use the remaining time more effectively to polish and optimize the experience. This would also have made it easier to detect architectural flaws earlier in the process.

6.3.2 Align Roles and Workflow from the Beginning

In the early phases of development, responsibilities between team members were not clearly defined, which led to some overlap in implementation and confusion about ownership of various components. For instance, backend integration, UI updates, and Firebase structure were at times worked on simultaneously by multiple people without full alignment, causing merge conflicts and

duplicated logic. These coordination challenges cost valuable time and occasionally introduce avoidable bugs.

If we started again, we would establish clearer task ownership from day one, supported by a more structured workflow for version control and pull requests. The standard approach would be to follow agile methods such as Kanban or Scrum. With these methods we would ensure that all team members would meet often and always be up to date with each others work. It would also be a lot easier for the team to improve during the project as agile approaches take time to learn from retrospectives of what works and what is not working for the team. The problems we faced in the middle of the developing period could have been discovered then rather than now in retrospective. This would not only have improved efficiency, but also made it easier for each team member to work independently while still contributing cohesively to the larger system. Furthermore, having scoped roles would have helped us identify integration challenges earlier, allowing for more parallel development and less last-minute refactoring.

7 Contributions

Table 26 outlines the amount of hours put in by each group member during this project.

Table 26: Overview of the number of hours put in by each group member.

Group member	Hours worked
Arya Raeesi	105
Isak Olav Sjøberg	105
Benjamin Færestrand	110
Jacob Weidel	95
Ninon Lisa Trouchet	0

In addition, section 7.1, 7.2, 7.3, 7.4 and 7.5 outline the specific roles each group member had during the project.

7.1 Arya Raeesi

Arya contributed to the overall backend infrastructure by setting up the Node.js code and deploying it using Render. He also played a central role in report writing, structuring key documentation sections.

7.2 Isak Olav Sjøberg

Isak was responsible for developing the core game logic, including the sequence management for rounds and roles in DrawGuess. He also assisted in deploying the server infrastructure and contributed to writing and organizing the reports.

7.3 Benjamin Færestrand

Benjamin focused on the frontend and UI development, designing key user interfaces and handling Firebase integration. He also contributed to backend functionality and was responsible for setting up and configuring Firebase Firestore and Firebase Storage.

7.4 Jacob Weidel

Jacob worked on improving the user experience across screens and transitions, focusing on intuitive design and in-game interactions. He also developed and maintained key gameplay sequences and handled logic related to player actions during the game.

7.5 Ninon Lisa Trouchet

Ninon didn't contribute to the project.

8 Last words

We want to express our sincere gratitude to all external executors for their participation during our testing phase. Their involvement was important in validating the U1 and U2 quality requirements.

Overall, this project has been a challenging yet highly rewarding learning experience. We are genuinely thankful to the course coordinators for the opportunity to work on such an engaging and practical assignment.

Bibliography

[1] Meyer, Bertrand: *Object-Oriented Software Construction.* Prentice Hall, 1988.

A 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.

A.1 Mathematical Expression for Point Distribution During the Guessing Phase

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:

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

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

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