# IDATT2003 Project Report

Vetle Nilsen and William Holtsdalen

23. May 2025

**Abstract**

This report describes the design and development of a Java-based board game application. It's created to demonstrate proficiency in object-oriented programming, GUI development, and design principles. The application features two classic board games: Chutes and Ladders, and Ludo - implemented with a modular structure that allows for future extensions.

The project was developed using Java and JavaFX, in accordance with the constraints of the assignment. It follows a three-phase structure, with iterative improvements to code structure, file handling, and a graphical user interface. Furthermore, the application utilizes the Model-View-Controller (MVC) pattern and applies design principles such as low coupling, high cohesion, and the use of both Factory and Observer patterns.

Throughout the project, we focused on robust class design, unit testing, and easily maintainable architecture. Functional features such as custom game boards, event tiles, and bot players were implemented successfully. Extensions beyond the minimum viable product (MVP) include support for multiple board games and the ability to create new and edit existing boards.

The project highlights the importance of thoughtful design in building scalable applications and provided valuable experience in applying design principles to build a functional product.

# Contents

## Figures



## Tables

## Code Listings

# 1 Introduction

## 1.1 Background

This project is part of the course IDATT2003 - Programming 2 at NTNU Trondheim. The goal is to develop a board game application using Object-Oriented Programming (OOP). The developed application includes Chutes and Ladders and Ludo, which support different customized boards and local multiplayer functionality. While the functional goal of the project is to implement a board game application, the underlying academic objective is to improve the students' programming skills by applying theoretical concepts such as OOP, event-driven programming and layered architecture in a realistic development scenario.

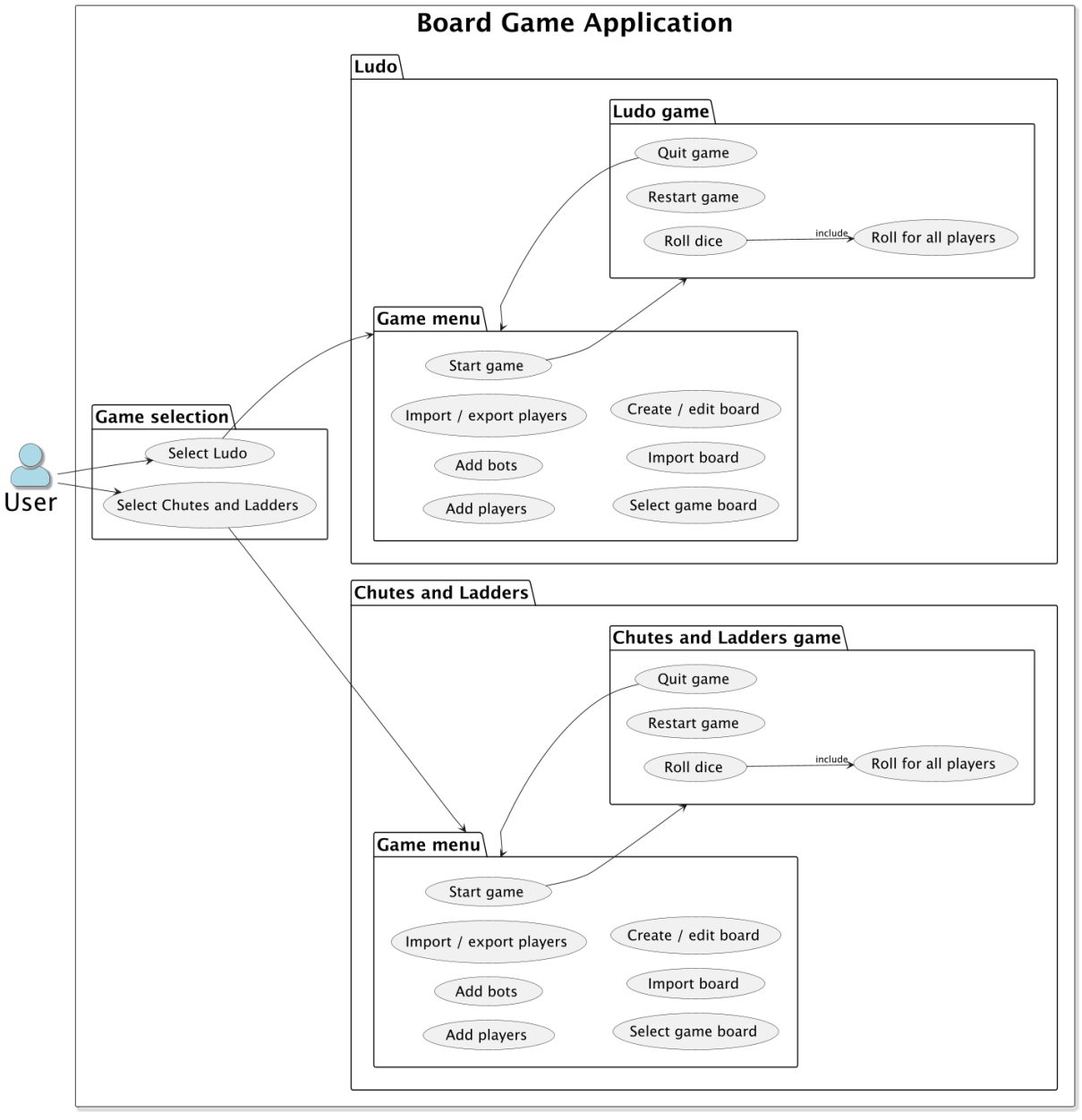



**Figure 1:** Use-case-diagram.

## 1.2 Requirements

### 1.2.1 Functional requirements

- The board game application should be able to be played by the same rules and mechanics as the original board games.
- Players must be able to move their game piece, view the state of the game, and restart the game.
- The application should provide a GUI.
- The application should provide local multiplayer functionality.
- The user should be able to add Bot-players to the game lobby.
- The gaming board should contain event tiles that trigger special action when landed on.
- There should be one or more random number generators, which should simulate die rolls.

### 1.2.2 Non-functional requirements

- The GUI should be user-friendly.
- The game should support 2–5 players, with at least 1 real-life player.
- The project should follow OOP principles to ensure maintainability and readability.
- The project should be able to support two different board games, and be scalable to add additional boardgames in the future.

## 1.3 Constraints

The project has several constraints that need to be followed. These can be divided into 2 main groups: development constraints and functionality constraints.

### 1.3.1 Development constraints

The development of the project must follow specific technological and structural constraints. Firstly, the application must be made in Java, with a focus on object-oriented design principles. The graphical user interface GUI must be built with JavaFX, and the use use of FXML is not allowed.

Version control is a mandatory aspect of the project. All code must be maintained in a Git repository, with regular commits. Additionally, the project is required to be set up as a Maven project.

### 1.3.2 Functionality constraints

The application is limited to local multiplayer and does not support network play or mobile platforms. It is designed for desktop use with a fixed layout. Bot-players are supported, but use simple logic with minimal strategy. All configuration, such as custom boards, is handled locally through the UI or external files.

### 1.4 Dictionary

Table 1 shows an overview of terms that represent the domain model of the project. It is essential to have a good grasp on these terms to understand the choices and solutions regarding the development of the application.

| Term | Description |
|---|---|
| Model | Contains the core game logic and data structures such as Board, Tile, and Player. |
| View | Handles user interface and rendering. In this project, views are implemented using JavaFX. |
| Controller | Coordinates application logic. Receives input, updates the model, and refreshes the view. |
| Token | Represents a player's movable piece on the board. In Ludo, players can have multiple tokens. |
| Observer | A design pattern that allows components to subscribe to events and be notified when changes occur. Used to decouple game logic and UI updates. |
| Navigation | Refers to moving between application views (e.g., main menu, game screen). |
| FileHandler | Handles serialization and deserialization of game data (e.g., boards or player data) using formats like JSON and CSV. |
| Serialization | Converts objects into a storable format (e.g., JSON) and restores them later. Used for board and player configuration. |
| Action | Behaviors triggered by user or game events, like TileAction or button clicks. |

**Table 1:** Dictionary of key software terms used in the project

## 2 Theory

The theoretical principles that form the foundation of the project's design and structure are shown in this section. These principles are referenced in the discussion and result section.

### 2.1 Coupling and Cohesion

Coupling refers to the interdependence between software modules, whereas cohesion refers to how closely related the functions within a module are. Low coupling and high cohesion promote modularity, testability, and maintainability. [1]

## 2.2 Separation of Concerns

Separation of Concerns (SoC) structures software into sections, each responsible for a separate functionality. This reduces complexity and allows individual parts of the system to be developed and maintained independently. A well-known architecture that applies this principle in practice is the Model-View-Controller (MVC) pattern [2], which is used extensively in this project and described in more detail throughout section 4.

## 2.3 SOLID Principles

The SOLID principles are five object-oriented design principles intended to improve software modularity and maintainability [3]:

- **Single Responsibility Principle (SRP)**: A class should have only one reason to change.
- **Open/Closed Principle (OCP)**: Classes should be open for extension, but closed for modification.
- **Liskov Substitution Principle (LSP)**: Subtypes should be substitutable for their base types without altering the correctness of the program.
- **Interface Segregation Principle (ISP)**: Clients should not be forced to depend on interfaces they do not use.
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules; both should depend on abstractions.

## 2.4 Design Patterns

Design patterns are reusable solutions to common problems in software design. They standardize architecture, improve efficiency by avoiding redundant problem-solving, and offer flexibility through abstract structures [4]. An example is the Observer pattern, where a *subject* notifies its *observers* whenever certain events happen.

## 2.5 Logging and Exception Handling

Logging tracks an application's runtime behavior for monitoring and debugging. Exception handling ensures stability by managing runtime errors gracefully.

## 2.6 Serialization

Serialization converts objects into a storeable or transmittable format, such as JSON or CSV. It enables the application to persist complex data structures and restore them later by deserializing the saved content. [5]

## 2.7 Testability and Extensibility

Testability refers to how easily components can be tested in isolation. Extensibility is the ability to add new functionality to a system with minimal changes to existing code.

# 3 Method

## 3.1 Development process

The project was released to the developers in three incremental parts, which naturally led to the use of an incremental and iterative development approach. Each part introduced new functional and technical requirements, guiding the project's progression from core logic to file handling and eventually to the full graphical user interface.

To manage the workflow, the team used an issue board in GitHub Projects, where tasks were created for each milestone and tackled iteratively. Weekly meetings were held to stay on top of progress, discuss architectural and design decisions, and plan upcoming work. Communication was done through Discord, allowing for continuous collaboration even during periods when a team member was absent.

## 3.2 Tools

A variety of tools were used throughout the project, each serving a specific purpose in the development process. Table [2] summarizes the tools, versions, and usage.

| Tool | Version | Description |
|---|---|---|
| IntelliJ IDEA | 2024.2.5 | IDE for Java-development and debugging, as well as for running tests. |
| Maven | 3.9.9 | Dependency management and building. |
| Overleaf | N/A | Writing and formatting the project report in LaTeX. |
| Git (GitHub) | 2.45.2 | Version control and task tracking with issue boards (GitHub Projects) |
| Figma | 2025.05.21 | Wireframe for designing the GUI |
| PlantUML | 1.2025.2 | Making UML diagrams |

**Table 2:** Tools used in the project

The project used Maven for dependency management, compilation, and test execution. This ensured a consistent build process across development machines and simplified version handling for libraries like JavaFX and Gson.

GitHub has been used for version control and collaborative development. For maintaining a clear and structured workflow, GitHub issues are created for new features and other tasks in an Issue board, illustrated in figure [2]. These issues are assigned to team members and used to track progress in GitHub Projects. Each task and feature is worked on in dedicated branches, named according to the corresponding issues (e.g. feat/110-create-ludo-board). Pull requests are created to integrate code from these feature branches into the 'dev' branch, as well as from the 'dev' branch into the 'main' branch for releases.

Initial GUI layouts were created in Figma. These wireframes were refined during the later development phases as new GUI components were implemented (see appendix E).

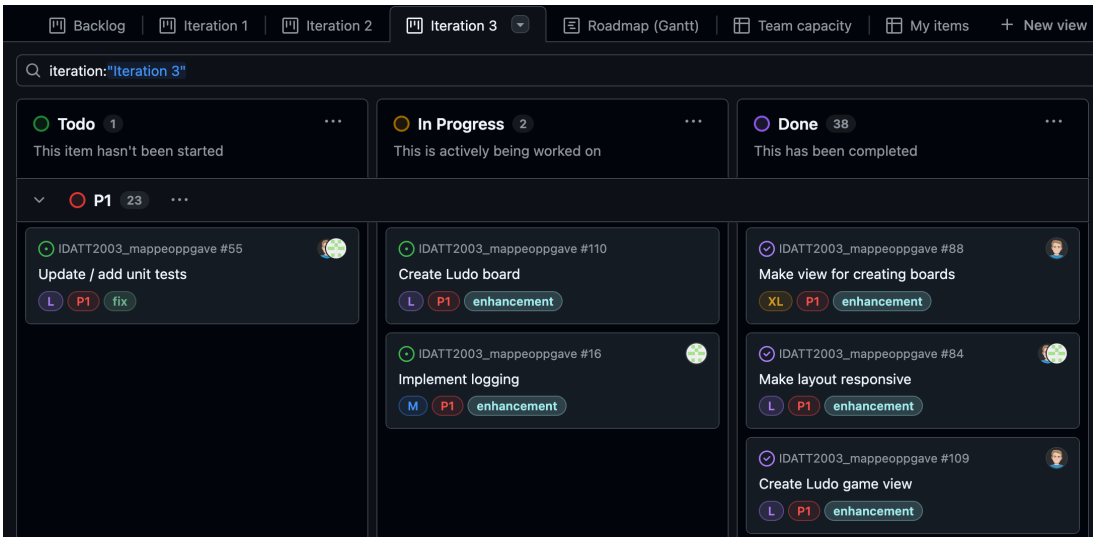



**Figure 2:** Part of GitHub issue board

### 3.3 Use of AI-tools

Attached to this report is a declaration covering the use of AI tools in this project, see appendix A. It describes which tools have been used and what they have been used for.

## 4 Results

### 4.1 Technical design

The application is built as a standalone JavaFX desktop application with no network connectivity. The architecture follows the MVC pattern, ensuring a separation of concerns (section 2.2) and simplifying maintenance and testing. The model layer encapsulates core entities of the system and rules for game states. The view layer handles graphical rendering of the different views/screens of the application using JavaFX, and the controller classes manage user input, view logic, and communication between the other layers.

A key design goal was flexibility and extensibility. To support this, the project is built around an abstract core of classes that define general board game functionality. Each supported game is implemented by extending these base classes with specific logic unique to that game. This modular structure allows new board games to be added with minimal impact on the rest of the codebase, aligning with the open/closed principle of SOLID outlined in section 2.3.

Figure 3 illustrates an example of how the core abstract classes are inherited by concrete implementations of views and controllers specific to each game type.

### 4.2 Implementation

The application is implemented in Java using the JavaFX library for the graphical user interface. All source code is organized in a modular package structure reflecting the MVC architecture, with ded-

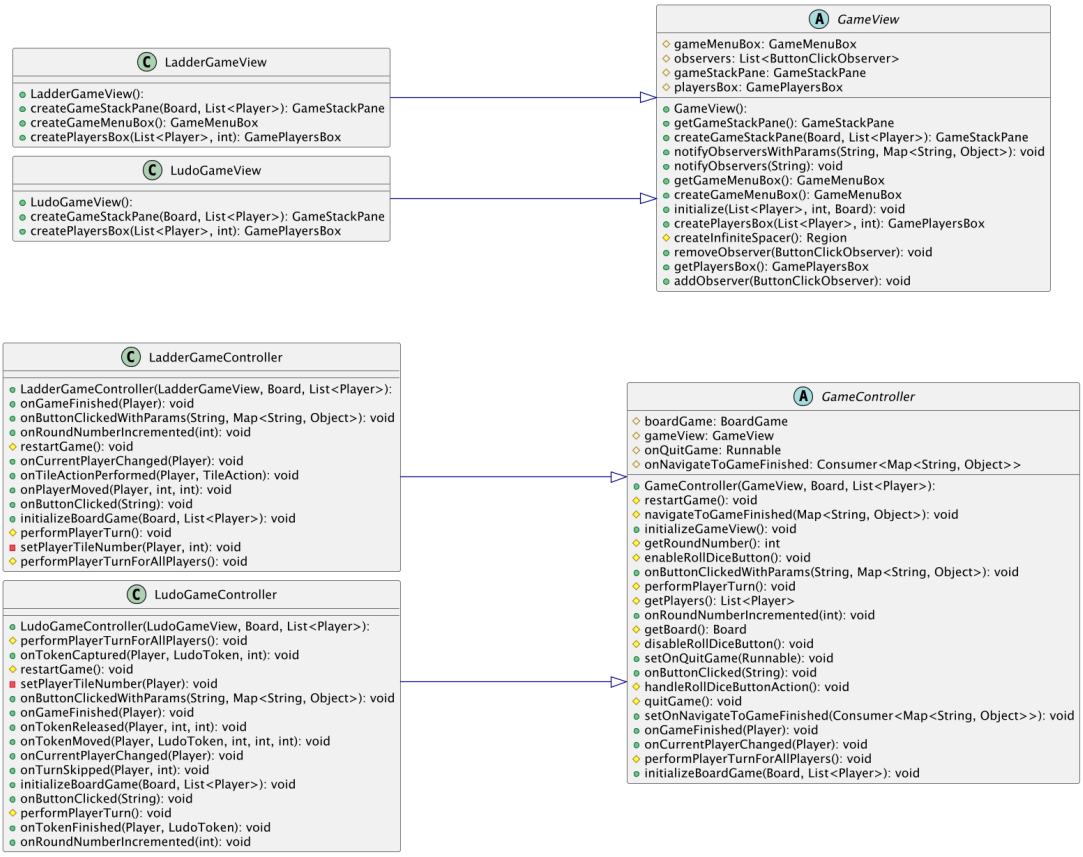



**Figure 3:** Example of abstract and concrete views and controllers

icated packages for model, view, and controller components, as well as for file handlers, navigation, and factories.

The model package contains abstract base classes such as Board, Tile, Player and BoardGame, which define shared functionality and serve as templates for specific game implementations. For example, LudoBoardGame extends the abstract BoardGame class and overrides relevant methods for Ludo-specific rules and behaviors like token captures and support for multiple tokens per player.

The controller layer includes both common and game-specific controller classes. The common controllers are abstract and define management of application-level state, while the game-specific controllers have additional logic to handle unique actions and transitions. The controllers interact with view components, updating UI-elements based on user input and changes in game state.

The views are implemented using pure JavaFX (without FXML), and are responsible for rendering interactive elements such as the game board, with animated player token movements, dynamic game menus, and more. Each view is built around reusable JavaFX components and styled using CSS. The GUI uses consistent color schemes and layout structures inspired by the visual identity of the original board games. Screenshots of the interface are included in Appendix B to illustrate the design choices.

The project uses Maven for build and dependency management, and includes several other dependencies besides JavaFX such as:

- **Gson** for handling JSON
- **Junit 5** for unit testing
- **Slf4j & Logback** for application logging (explained in section 2.5)

- **Ikonli** for graphical icons in JavaFX
- **Surefire** for running unit tests

## 4.3 Testing

The application has been tested through a combination of automated unit testing and manual end-user testing, with the common goal of ensuring correctness in core logic and robustness in user interaction. See figure 4 for an overview of test cases and priorities.

| **Software Development Test Specification** | **Team:** IDATT2003 Group 15<br>**Project:** Board Game App | | **Last revised:**<br>22/05/2025 | | | | | |
|---|---|---|---|---|---|---|---|---|
| Status | Test name | Description | (Pre conditions) | Test data | Priority | Expected results | Tester(s) | Comment(s) |
| Unit testing (UT) | | | | | | | | |
| Completed | UT model classes | Verify that model/entity classes function as expected by creating and executing unit tests. | Model classes must be implemented | N/A | HIGH | All unit tests, both positive and negative, should execute with no errors, with a high test coverage. | WH & VN | JUnit unit testing, preferrable executed with coverage. |
| Completed | UT factory classes | Verify that the factory classes for game boards, players and player tokens work as intended. | Model classes and file handler classes must be implemented | N/A | MEDIUM | Instances of models, such as game boards and players, should be created properly, given the appropriate parameters are provided. | WH | JUnit unit testing, preferrable executed with coverage. |
| Completed | UT file handler classes | Verify that the file handler classes work as intended, and are robust regarding file content and formatting. Should only test parsing, no file access. | Model classes must be implemented | Test data can be created in test class. | MEDIUM | All parsing (deserialization) and serializiation should work properly, and there should be sufficient error handling for content/format errors. | WH | JUnit unit testing, preferrable executed with coverage. |
| | | | | | | | | |
| End user testing (EUT) | | | | | | | | |
| Completed | End user test #1 (EUT#1) | Going through all the menus and testing all available functionality, as well as playing both board game types multiple times trying to trigger any unforeseen bugs. | Both the Chutes and Ladders game, and the Ludo game must be completed and playable. | Application v3.x.x. | HIGH | Feedback on design, features, game logic and potential bugs / areas for improvement. | VN | Feedback will be shared during in-person meeting. |
| Completed | End user test #2 (EUT#2) | Going through all the menus and testing all available functionality, as well as playing both board game types multiple times trying to trigger any unforeseen bugs. | Both the Chutes and Ladders game, and the Ludo game must be completed and playable. | Application v3.x.x. | HIGH | Feedback on design, features, game logic and potential bugs / areas for improvement. | WH | Feedback will be shared during in-person meeting. |

**Figure 4:** Test specification

### 4.3.1 Unit testing

Unit tests were written using JUnit5 for the most critical parts of the application. The focus was on business logic and data handling, especially within the model and file handler packages. The following areas were prioritized for testing:

- **Model classes**: Tested to ensure correct behavior of entity logic such as player movement, tile actions, and dice rolling. Both positive and negative tests were included to validate boundary conditions and rule enforcement.
- **Factory classes**: Validated to confirm that all predefined game configurations were instantiated correctly with valid parameters.
- **File handler classes**: Tested to ensure robust parsing and serialization of board and player data using JSON and CSV respectively. Tests verified that wrongly formatted data was correctly handled without breaking the application.

All unit tests execute successfully in the final release of the application, without errors, and coverage tools were used to ensure that all high-priority logic and paths are covered.

### 4.3.2 End-user Testing

Two rounds of manual user testing were performed to validate the functionality of the user interface and features of the games. Feedback was shared during in-person meetings and included suggestions like slowing down animations and improving menu clarity. Critical bugs discovered during testing were resolved prior to the final delivery.

## 4.4 Deployment to end user

The application is made available to end users through the GitHub repository located at `https://github.com/WilliamHoltsdalen/IDATT2003_mappeoppgave`

To run the application, users can either open the project in an IDE with Maven support (such as IntelliJ IDEA) or use a terminal with access to Maven commands. In addition to support for Maven, Java 21 needs to be installed.

Execution of the application is handled through Maven, and the following steps are used to build and launch the application.

1. Run `mvn clean package` to build the app and run all unit tests.
2. Launch the application using `mvn javafx:run`.

# 5 Discussion

## 5.1 Solution

One of the strongest aspects of the solution was the architectural design, and adherence to MVC. It was selected not only for its clarity and maintainability but also to reflect good object-oriented design principles, which were central to the learning goals of the project. By utilizing polymorphism and inheritance, by building around abstract base classes such as `BoardGame`, `Player`, and `Tile`, we ensured that the codebase was both flexible and easily scalable. This made it possible to implement multiple board games (Chutes and Ladders, and Ludo) by simply extending the abstract classes and adding game-specific logic. A key benefit of this structure is that new games can be added in the future with little to no refactoring of existing code. Code listing 1 illustrates how the abstract `BoardGame` class encapsulates core logic and shared components, such as player handling, dice rolling, and turn iteration, which can then be reused across games. Additionally, it shows how the BoardGame implements `BoardGameSubject` and contains a list of `BoardGameObserver` instances to utilize the observer design pattern, described in section 2.4.

**Code listing 1:** Abstract BoardGame class

```
1  public abstract class BoardGame implements Game, BoardGameSubject {
2    protected final List<BoardGameObserver> observers;
3    protected Board board;
4    protected List<Player> players;
5    protected Player currentPlayer;
6    protected Dice dice;
7    protected int roundNumber;
8
9    // [...] common methods and game logic, shared by all board games
10 }
```

In addition to a sound structure, the use of Maven for dependency management and GitHub for version control contributed to a well-organized development process. Unit testing, although not fully maintained throughout the entire project, proved valuable in later stages. After completing the test specification and aligning the tests with updates to the core classes, we identified and resolved several bugs, improving robustness and confidence in the solution greatly.

The deployment setup also contributed to accessibility. Since the project is distributed via GitHub and runs using simple Maven commands, it can be launched easily on any system with Java 21 and Maven installed, without any platform-specific dependencies.

## 5.2 Process

The project's incremental and iterative three-phase process worked well, ensuring stable progress from start to finish. However, the process was occasionally affected by alternating periods of absence due to illness, which led to periods of reduced progress. Communication through Discord mitigated some of these challenges, allowing for collaboration even when physical meetings were not possible. GitHub Projects and Issues were very useful throughout the whole project, especially when organizing remote work. The use of branches, pull requests and version tags ensured a clean and structured codebase at all times. Figure 5 shows the GitHub Project's burn-up chart, illustrating the gradual progress and completion of planned work.

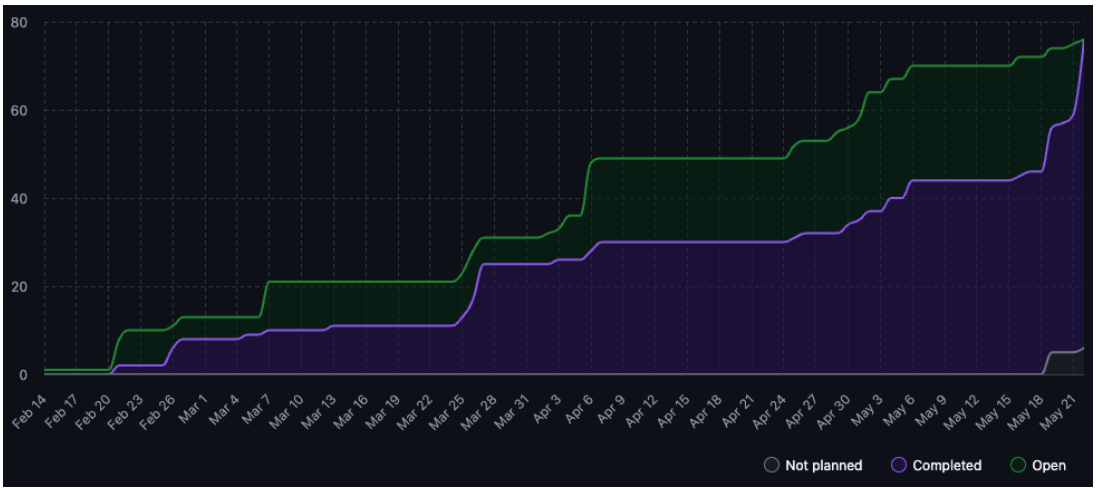



**Figure 5:** Burn up chart for GitHub issues

One area for improvement was the consistency of testing. During the middle phase of development, unit tests were not kept in sync with evolving models and business logic. This created a temporary gap in the test coverage, and in hindsight, aligning testing with development more consistently would have helped catch bugs and issues earlier.

## 5.3 AI-tools

The use of AI tools had a positive impact on both development and learning. GitHub Copilot and Supermaven provided efficient code completion, giving us more time to focus on design choices and implementation. While these tools improved productivity, they also required careful review to ensure the suggestions aligned with our intentions.

ChatGPT was used to understand complex errors during development, clarify Java concepts, and

propose suggestions to code structure and report text. It also generated image assets for Chutes and Ladders, making the graphical design process much easier.

Overall, AI tools contributed to increased productivity as well as improving technical solutions, and documentation quality.

# 6 Conclusion

The main goal of the project was to implement a flexible and extensible board game application following solid object-oriented design principles. This was achieved through a shared abstract architecture that enabled both Chutes and Ladders and Ludo, with minimal duplication. All functional and non-functional requirements (sections 1.2.1 & 1.2.2) were satisfied, and the final solution adheres to the assignment's constraints (section 1.3).

The implementation reflects the theoretical principles outlined in section 2, particularly in its modular design, extensibility, and use of patterns like MVC and Observer.

If starting over, we would maintain closer alignment between testing and code updates throughout the project. A testing gap in the middle phase reduced coverage and confidence temporarily.

The application does however have some limitations, such as lacking support for mobile platforms and online multiplayer. The layout and responsiveness of the GUI also assume a reasonably sized desktop screen and mouse input. In addition, gameplay aspects such as Bot-players are relatively basic and would benefit from further improvements.

For future work, the architecture is already well prepared to support additional board games. Extending the GUI to support theme customization or introducing new game types are all natural next steps.

Overall, the project provided valuable experience in modular architecture, the use of version control systems, and JavaFX-based UI design, and resulted in a complete and functional product aligned with the learning goals of the course IDATT2003.

# Terminology

**Bot-players** computer controlled players that perform their moves automatically. 6, 15

**FXML** an XML-based user interface markup language created by Oracle Corporation for defining the user interface of a JavaFX application [6]. 6, 11

**Gson** an open-source Java library that serializes Java objects to JSON. 9

**GUI** short for Graphical User Interface,when a user can view and interact with an application. 6, 9, 15

**Issue board** a Kanban-style table, typically with columns for "todo", "in progress", and "done". 9

**JavaFX** an open source, application development platform built on Java. 6, 9–12, 15

**MVC** short for the Model-View-Controller architectural pattern. 8, 10, 13, 15

**OOP** short for Object Oriented Programming. 5

**UI** short for user interface. 6, 11

## References

[1] geeksforgeeks. "Coupling and cohesion - software engineering." (), [Online]. Available: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/. (accessed: 22.05.2025).

[2] W. contributors. "Model-view-controller." (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1288991163. (accessed: 21.05.2025).

[3] geeksforgeeks. "Solid principles in programming: Understand with real life examples." (), [Online]. Available: https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/. (accessed: 23.05.2025).

[4] geeksforgeeks. "Software design patterns toturial." (), [Online]. Available: https://www.geeksforgeeks.org/solid-principle-in-programming-understand-with-real-life-examples/. (accessed: 22.05.2025).

[5] Wikipedia. "Serialization." (), [Online]. Available: https://en.wikipedia.org/wiki/Serialization. (accessed: 22.05.2025).

[6] W. contributors. "Fxml." (), [Online]. Available: https://en.wikipedia.org/w/index.php?title=FXML&oldid=1290047938. (accessed: 23.05.2025).

[7] geeksforgeeks. "Separation of concerns (soc)." (), [Online]. Available: https://www.geeksforgeeks.org/separation-of-concerns-soc/. (accessed: 22.05.2025).

# 7 Appendix

# A AI-declaration

# Deklarasjon om KI-hjelpemidler

Har det i utarbeidingen av denne rapporten blitt anvendt KI-baserte hjelpemidler?

[ ] Nei

[X] Ja

Hvis *ja*: spesifiser type av verktøy og bruksområde under.

---

**Tekst**

[ ] **Stavekontroll**. Er deler av teksten kontrollert av:
*Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai* eller lignende verktøy?

[ ] **Tekstgenerering**. Er deler av teksten generert av:
*ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr* eller lignende verktøy?

[X] **Skriveassistanse**. Er en eller flere av ideene eller fremgangsmåtene i oppgaven foreslått av:
*ChatGPT, Google Bard, Bing chat, YouChat* eller lignende verktøy?

Hvis ja til anvendelse av et tekstverktøy - spesifiser bruken her:

**Struktureringsforslag:** ChatGPT har blitt brukt til å danne forslag til strukturering av avsnitt.

**Tekstrevisjon:** ChatGPT har blitt brukt til kvalitetssikre deler av rapporten.

---

**Kode og algoritmer**

[X] **Programmeringsassistanse**. Er deler av koden/algoritmene som i) fremtrer direkte i rapporten eller ii) har blitt anvendt for produksjon av resultater slik som figurer, tabeller eller tallverdier blitt generert av: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

Hvis *ja t*il anvendelse av et programmeringsverktøy - spesifiser bruken her:

**Code completion**: GitHub Copilot og Supermaven er to KI-verktøy som har blitt brukt under utvikling av kode. Disse har fungert som «avansert autocomplete» med hensikten å øke effektivitet.

**Læring**: ChatGPT har blitt brukt som et hjelpemiddel for å forstå ukjente programmeringskonsepter, få forslag til struktur, samt for å forklare intrikate feilmeldinger som har oppstått under utviklingsprosessen.

**GitHub PR review**: 'GitHub Copilot code review' har blitt brukt til å oppdage småfeil og lage oppsummeringer av endringer i pull requests på GitHub.

---

**Bilder og figurer**

[X] **Bildegenerering**. Er ett eller flere av bildene/figurene i rapporten blitt generert av: *Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* eller lignende verktøy?

Hvis ja til anvendelse av et bildeverktøy - spesifiser bruken her:

Bildene til stiger, sklier og portaler til stigespillbrettene er generert av bildemodellen til ChatGPT 4o, og deretter redigert i Photoshop.

---

**Andre KI-verktøy**

| | |
|---|---|
| | **Andre KI-verktøy.** har andre typer av verktøy blitt anvendt? Hvis ja spesifiser bruken her: |

| |
|---|
| |

---

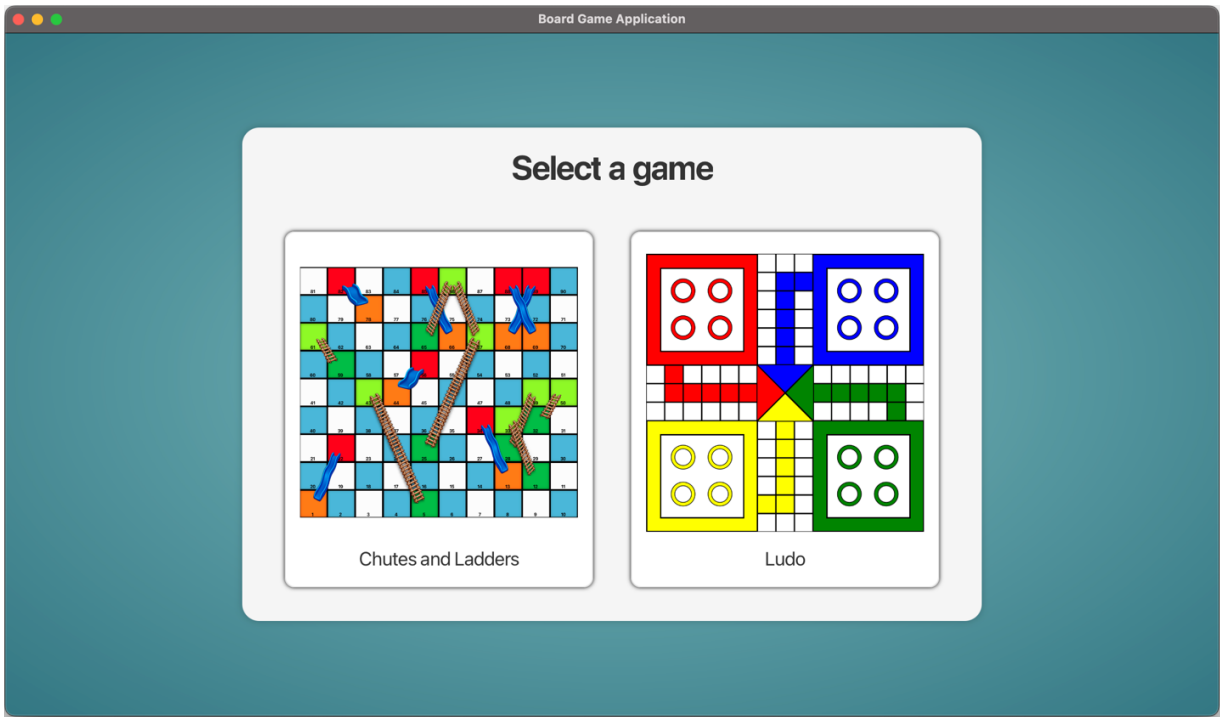
| | |
|---|---|
| **X** | Jeg er kjent med NTNUs regelverk: *Det er ikke tillatt å generere besvarelse ved hjelp av kunstig intelli- gens og levere den helt eller delvis som egen besvarelse.* Jeg har derfor redegjort for all anvendelse av kunstig intelligens enten i) direkte i rapporten eller ii) i dette skjemaet |

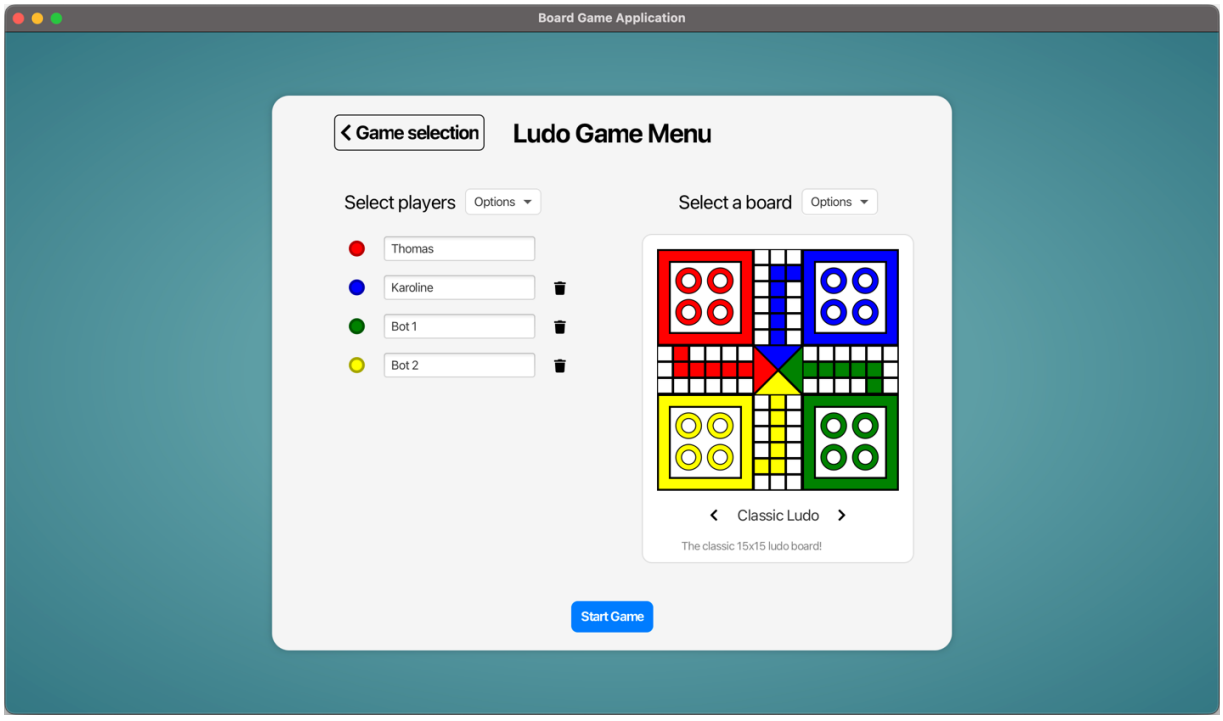
**IDATT2003 Gruppe 15 2025**

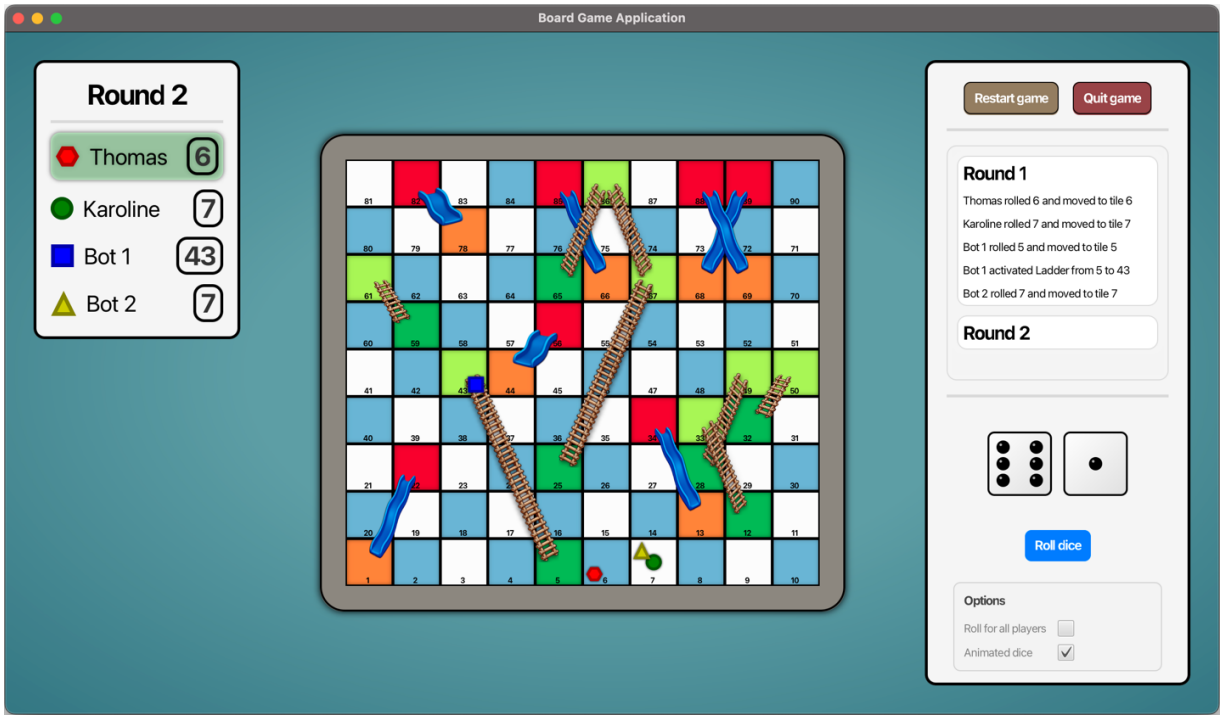
Signert av: Vetle Nilsen / 22.05.2025 / Trondheim
Signert av: William Laukvik Holtsdalen / 22.05.2025 / Trondheim

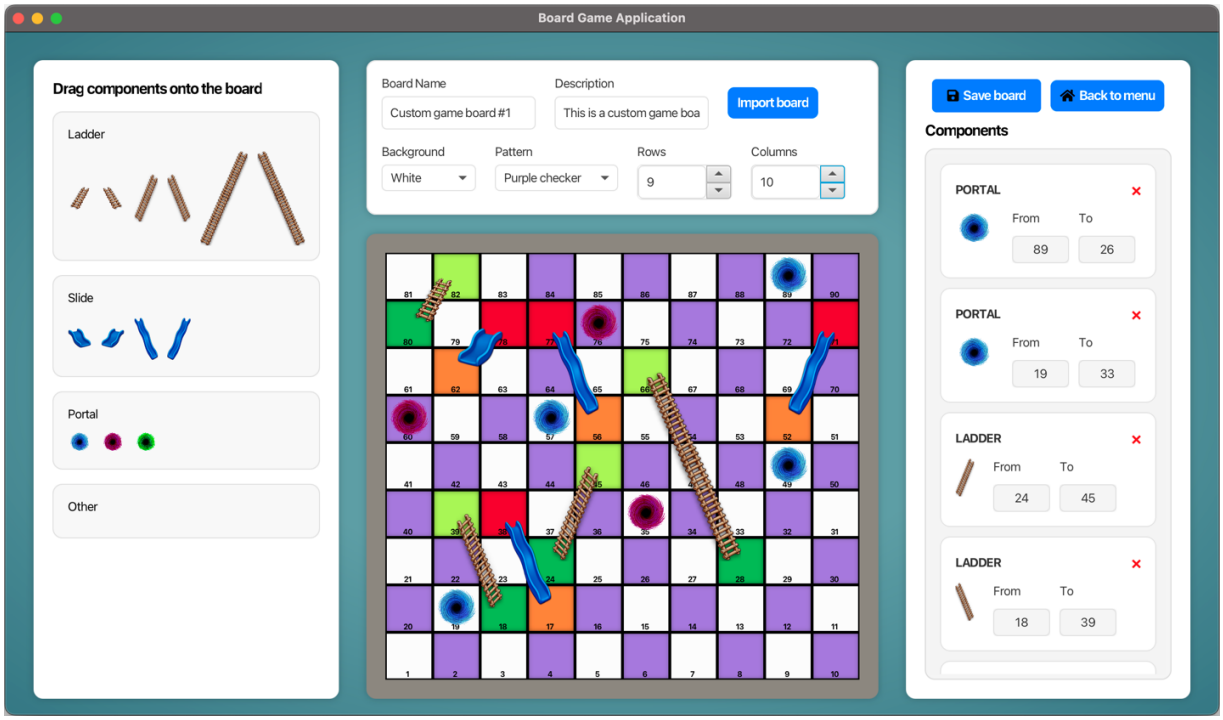
*Underskrift/Dato/Sted*

# B GUI screenshots

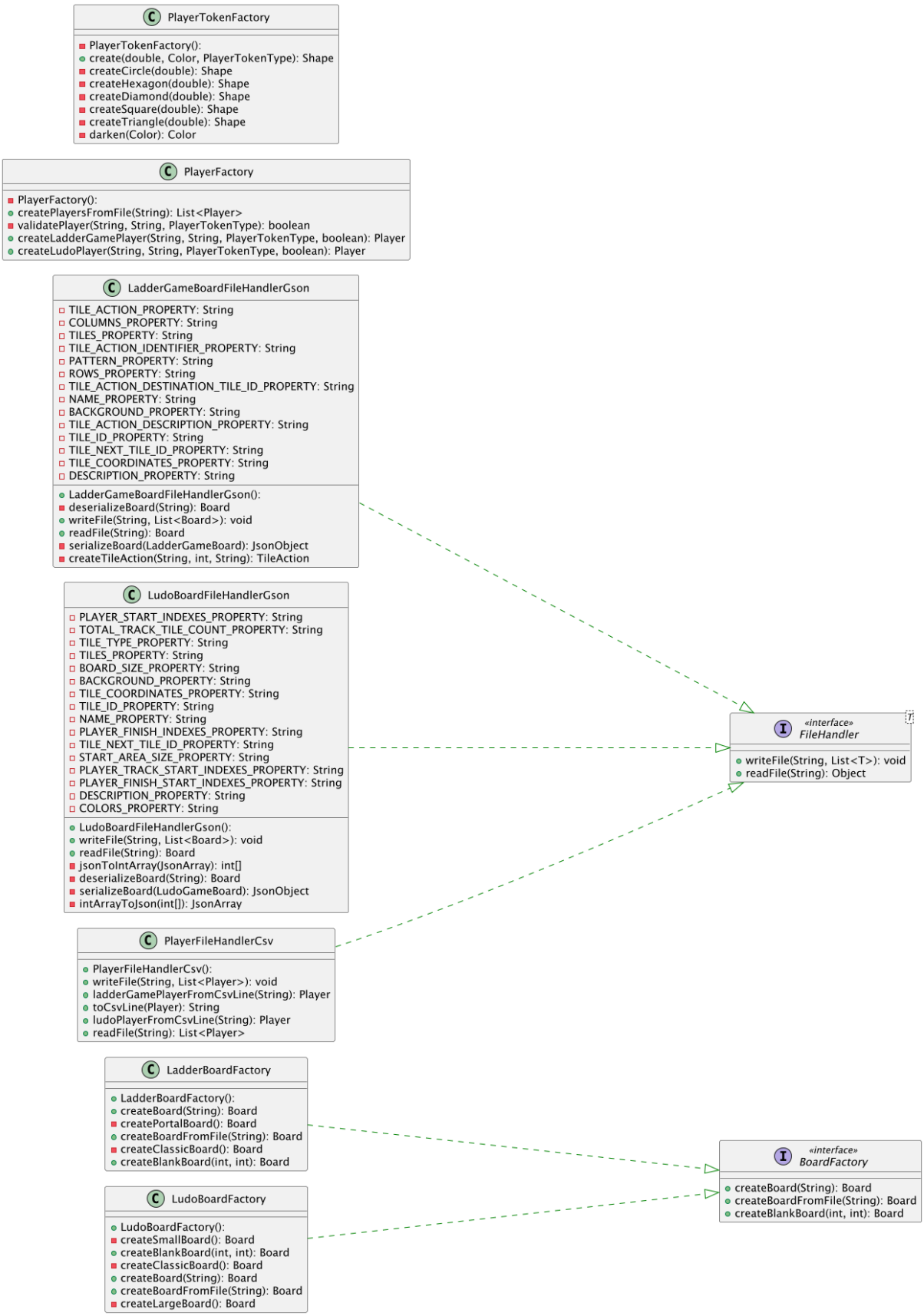

Board Game Application
Select a game
Chutes and Ladders
Ludo


Board Game Application
Game selection
Ludo Game Menu
Select players
Options
Thomas
Karoline
Bot 1
Bot 2
Select a board
Options
Classic Ludo
The classic 15x15 ludo board!
Start Game

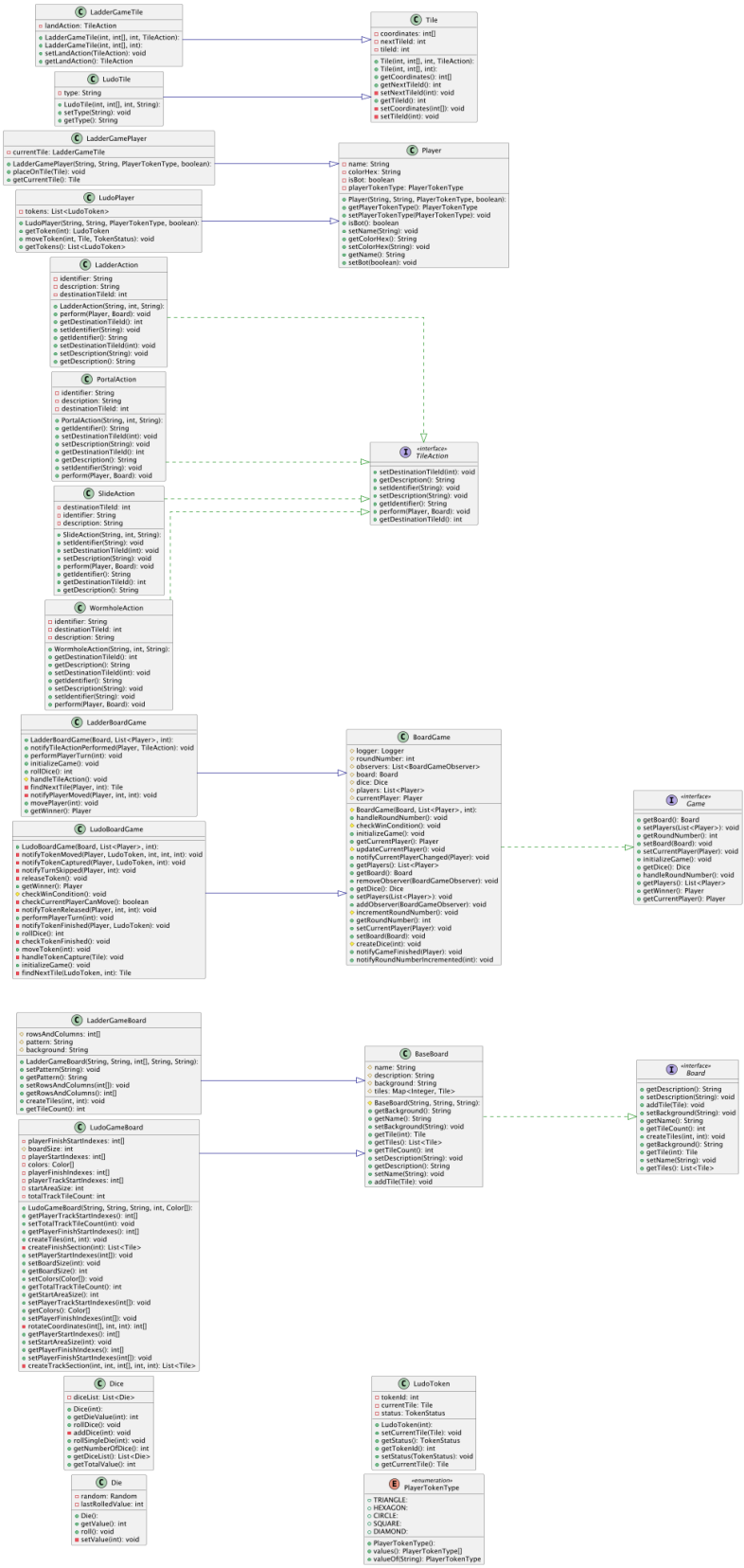

Board Game Application
Round 2
Thomas 6
Karoline 7
Bot 1 43
Bot 2 7
Restart game
Quit game
Round 1
Thomas rolled 6 and moved to tile 6
Karoline rolled 7 and moved to tile 7
Bot 1 rolled 5 and moved to tile 5
Bot 1 activated Ladder from 5 to 43
Bot 2 rolled 7 and moved to tile 7
Round 2
Roll dice
Options
Roll for all players
Animated dice


Board Game Application
Drag components onto the board
Ladder
Slide
Portal
Other
Board Name
Custom game board #1
Description
This is a custom game boa
Import board
Background
White
Pattern
Purple checker
Rows
9
Columns
10
Save board
Back to menu
Components
PORTAL
From
To
89
26
PORTAL
From
To
19
33
LADDER
From
To
24
45
LADDER
From
To
18
39

# C Class diagram for factories and file handlers



# D Class diagram for model classes



# E Figma Wireframes