NTNU

Kunnskap for en bedre verden

Assignment 1 - Autumn 2024

TDT4250 - Model-Driven Software Engineering

# Variability aspects to: Chess Evolution

*Author(s):*
Sondre Alfnes

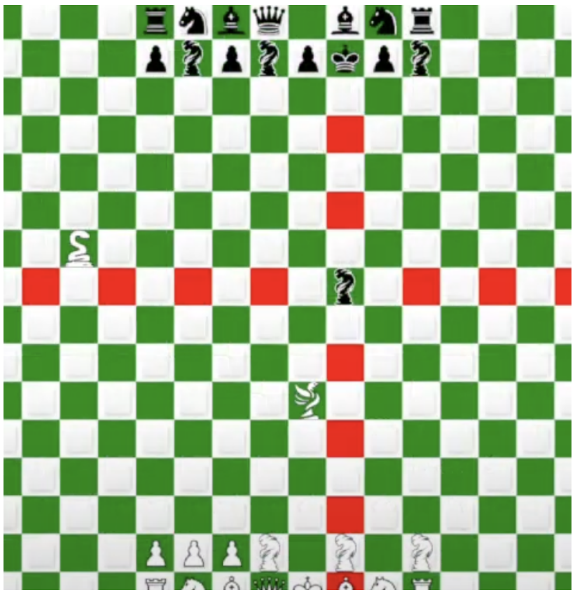
Date: 10.10.2024

# Innhold

# 1 Description of the System

## 1.1 Overview

The system that I am going to explain in this work is a mobile game, developed for android and is called “Chess evolution”. This is a re-imagination of the classic Chess game with its core principles of taking the king. But this game has customisable and experimental elements. This was added in to bring freshness and appeal for players. This project was my project for an assignment in TDT4240 - Software Architecture and builds upon various commercial off-the-shelf (COTS) components, e.g. Android SDK, Firebase and LibGDX to satisfy functional and quality requirements.



Figur 1: A example game of Chess Evolution

## 1.2 Game Concept

Chess Evolution keeps all the rules to chess, but also adds new ones. The game offers a selection of preconfigured loadouts, players can pick before entering a game, each possesses its own move sets with different movements. The player can also choose different board sizes, this introduces a new layer of strategy, enabling players to adopt alternative tactics and open up valuable strategies that can be used.

The game provides the local multiplayer mode so that players can play against each other on a single device. The game is designed in such a manner that it is completely intuitive to use, having touch-input controls and visual indicators pointing towards every move the player makes. For example, when a player choose a piece that can move, then the game will highlights all the possible moves for that piece to make it easier for players to plan their strategies.

## 1.3 Functional Requirements

We classified the system requirements under a few priorities - High, Medium, and Low basing on their functionality. The high-priority requirements are the baseline for the game, this is what defines the core functionalities, such as being able to take turns and to capture the opponent’s king. Expanding the user experience (Ex: in designing areas, contain simply can choose another loadouts and to check out a leaderboard) is medium-priority requirements. Ans low-priority requirements are kind of like ”add-ons”you would love to see in the game but are not necessary for the game play itself, i.e sound effects and an in-game menu.

## 1.4 Quality Requirements

The most important quality attribute of Chess Evolution is, Modifiability — This is so that it could be easily updated during the future. Usability, performance and testability are the secondary quality-attributes. This is pretty important for having a seamless and fun to use interface, great performance in the game as well as being able to test the system completely so that you find any issue and bug fixes.

## 1.5 COTS Components and Technical Constraints

Commercial off-the-shelf (COTS) components are prebuilt software modules, that can be quickly integrated into a system to offer specific features or capabilities without custom development efforts. Although COTS components provides a way to save development time and price to some extent, it could at the same time place limits on technical solutions that need to be handled.

The project utilizes three main COTS components:

- **Android SDK**: Provides tools and libraries for developing Android applications. Constraints include compatibility with various Android versions and devices.
- **Firebase**: Offers backend services such as real-time databases, authentication, and cloud storage. Constraints include data security, privacy compliance, and potential latency issues.
- **LibGDX**: A cross-platform game development framework supporting Android, iOS, and desktop platforms. Constraints include the learning curve and integration challenges with other libraries.

Using these COTS components, the project can benefit from established, reliable technologies, but it is essential to carefully manage integration and compatibility aspects to ensure a smooth and efficient development process.

## 1.6 Architectural Design

Chess Evolution design takes several models and architectural tactics related to its quality view, which we can call as a model in architectural space. Model-View-Control (MVC) design pattern, thus improving modifiablity of the code and its testability. It also uses the Singleton, Observer, Memento, Factory, Strategy and Dependency Injection design patterns to make it more modular and flexible.

## 1.7 Chess Evolution Conclusion

Chess Evolution is a different type of multiplayer online chess game, which combines the classic game of chess with a unique and customizable experience for players. Where the games infrastructure has been thought through. All the architecture and design of the system to fulfill all the requirements for functions and qualities is to create a stable and scalable solution for fun or serious players.

# 2 Current Features

## 2.1 High-Level Features

Chess Evolution is a game designed for players who want an enhanced chess experience that combines different high-level features to serve in entertainment & tournament needs or purely elo progression. These mechanics enhance the experience of chess in it's traditional form while adding new elements which offer a way for customization, and strategic depth. The Chess Evolution High Level Features used to achieve this are:

- **Local Multiplayer Mode:** This feature allows two players to play against each other on the same device, making it perfect for in-person social interactions and competitive play.
- **Customizable Loadouts:** Before starting a match, players can customize their loadouts, choosing from a variety of unique pieces with special abilities. This feature adds a layer of strategy and personalization to the game.
- **Turn-Based Gameplay:** Chess Evolution maintains the classic, turn-based nature of chess, supporting a thoughtful and strategic approach to each move. This feature is crucial for supporting asynchronous online play, allowing players to take their time and think about their moves.
- **User Profiles:** Players can create and customize their profiles, which store their game settings, progress, and Elo rating. This personalization enhances the user experience by tailoring the game to individual preferences.
- **Leaderboards:** The game features real-time leaderboards that display players' Elo ratings, fostering a sense of competition and community among players.
- **Tutorial Mode:** To accommodate players of all skill levels, Chess Evolution includes a tutorial mode that covers everything from the basic rules to advanced strategies.
- **Time Limits:** Players can set time limits for their matches, adding a sense of urgency and accommodating various play styles, from quick, casual games to longer, more thoughtful matches.

## 2.2 Variability Aspects

Chess Evolution incorporates several variability aspects that enhance its flexibility and adaptability. These aspects allow the game to cater to a wide range of player preferences and ensure that it can be easily modified or extended in the future. The key variability aspects include:

There are several different sources of variability that make Chess Evolution so versatile. These attributes make the game suitable for players of all skill levels and provide flexibility for possible future modification or addition. Some of the variability aspects we had in mind while making the game were:

- **Loadout Selection:** The ability to use different loadouts adds variability in gameplay resulting in distinctive games. This feature is designed to allow players to set up large numbers of pieces in various configurations, which may result in new dynamics not typically found in chess. It also forces players to change their strats and play styles for different opponents so every game is a new one.
- **Board Size Options:** Selectable board sizes accommodate different strategies and intensify gameplay dynamics. Larger boards offer a more complex and strategic gameplay, whereas smaller boards lead to quicker and more aggressive matches. This range of options lets players that are casual to extremely hardcore easy to find a board size that fits them.
- **Game Settings Customization:** Permits players to change game settings, such as the time limit and available features for more personalized matches. This high level of customization means that players can create a game which suits their time restrictions. This would then make Chess Evolution more appealing to a broader audience. Furthermore, this functionality has also applications for mimicking different types of chess (e.g., blitz, rapid, and classical), which will attract players in all skill levels.

- **Personalized User Profiles:** Allows multiple users per device each with their own settings and progress. Because each user can customize virtually all aspects of their experience whether it be the difficulty level, the look/feel or something more application specific, it's perfect for multi-user devices.
- **Sound Effects:** Sound effects can be toggled for a more immersive audio experience. It enables players to listen realistically if they want to fully dive into the world but can also bring a quieter experience of the game without any loss. This is very useful if you find yourself in a room where noise may be an issue, or when playing your own music on the background.
- **In-Match Menu:** Offers in-game changes including the option to forfeit, settings, tutorial. The biggest positive is that the in-match menu allows for a learning curve as players can visit the tutorial where they can learn how to perform complex moves or understand strategies which therefore makes Chess Evolution an educational tool next to being entertaining.

## 3 Current Variability Mechanisms

### 3.1 Parameters

Parameters are a trivial way to do variation. This includes the use of conditional statements to modify the control flow at runtime based on parameters which may be input through command lines, configuration files or preference dialogs. This follows a feature-based method where each feature correlates with a parameter. The complete functionality is bundled with the product when it is deployed, and certain features can then be switched on/off in a run-time manner. But this can create messy code overall as new feature are being made(Lecture 3, 2024). In Chess Evolutions case, its goes to the board size, each size is parameterizes in a json in the project, so this could cause a problem if features such like irregular board sizes should be implemented.

### 3.2 Design Patterns

Some design patterns of the Gang of Four work particularly well for implementing variability. For example, we used Observer pattern so that we can add or remove features to be turned on and off by a subject controlling some variability point. This makes also the Template Method design pattern to be suitable for use with both alternative and optional features, since it uses polymorphism and late binding. The Strategy pattern is also used to focus on the variability of algorithms by replacing ad-hoc conditionals with polymorphism. This takes me to the last advantage of Decorator which is that we can introduce optional features, or feature groups, by forwarding calls to a component and changing behavior accordingly (Lecture 3, 2024).

### 3.3 Frameworks

Frameworks provide an incomplete set of collaborating classes that can be extended and tailored for specific use cases, often through inversion of control. They offer explicit extension points, known as hot spots, where custom behavior can be injected. Frameworks can be either white-box, requiring knowledge of internals and typically extended by subclassing, or black-box, where developers interact mainly with interfaces. Plugins developed for frameworks can trace features to specific plugins, although maintaining and evolving these plugins can be challenging (Lecture 3, 2024). This was visible when I worked with LibGTX since it was too old and used old dependencies.

### 3.4 Components and Services

Components are independent units of composition with specified interfaces and context dependencies, suitable for deployment and composition by third parties. In the Chess Evolution project, the

Android SDK, Firebase, and LibGDX were used as components and services employed to create a comprehensive gaming experience. Android SDK enabled user interface development and device compatibility, Firebase provided essential backend services like user authentication and real-time databases for multiplayer functionality, and LibGDX facilitated game physics, graphics, and input handling across different devices. These tools collectively enhanced the game's modifiability, usability, performance, and testability, ensuring a seamless integration and operation.

Components are self-contained units of composition with declared interfaces and context dependentcies that exist at deployment time and can be composed into applications by third parties. Android SDK, Firebase, and LibGDX (as components and services inside the Chess Evolution project) were used to create a full-on gaming solution. The Android SDK for UI development and device compatibility, Firebase for the backend services like user authentication and real-time databases (for leaderboard), LibGDX handling physics, graphics rendering, all input on different devices etc. These tools enabled the ease of modification, usability, performance and testability to be implemented as an integrated whole.

## 3.5 Version-Control Systems

Version-control systems, like Git, manage changes to source code and other development artifacts that offers more temporal (revisions) and spatial (variants) variability. Branches in these systems can also be utilized for customer-specific variability or SPL development, although merging branches is still a more severe challenge. These are systems used to manage and track different versions, and sometimes product insight over time. In this project we used GitLab as out version control system for development and production. We got also into scrum as it went pretty good together with GitLab. This enabled faster delivery and high- quality update that are easily controlled.

## 3.6 Build Systems

Make, Ant and Maven manage dependencies and support multiple build targets, which should make them ideal for compile-time variability. They determine what to include in the final product and can also coordinate other variability mechanisms such as applying feature-specific updates, executing code generators, and including configuration files (Lecture 3, 2024). We opted to use Gradle as our build automation tool for the project, given its flexibility and extensive support for dependency management. Choosing this enabled us to better manage project dependencies, automate our build processes, and support multiple build variants so that we can easily make changes and test our game.

## 3.7 Preprocessors

Preprocessors operate on the source code before compilation, offering features such as include files, lexical macros, and conditional compilation. The C preprocessor (cpp) is a good example, which uses `#define`, `#if` and `#ifdef` directives to mark up code destined for various features. The preprocessor extracts these annotations and decides which source code will be included in the final product. Preprocessors are flexible and typically customizable, making them complex and hard to maintain (Lecture 3, 2024). In our project, we used Gradle's specific version to handle preprocessing tasks, streamlining our build process by dynamically including or excluding features based on the build configuration, although it caused complications, it was maintenance for the course.

## 3.8 Aspect-Oriented Programming (AOP)

Aspect Oriented Programming is a way of addressing cross-cutting concerns by adding aspects that realize these concerns and separate them from the base code. AOP workflows involve the use of join points to wire in aspects at certain locations within a program during its execution, with weaving then mapping these aspects into desired spots. Although specialized languages like AspectJ exist, AOP's practical application is limited due to difficulties in verifying correctness and the base program's obliviousness to the aspects (Lecture 3, 2024)

## 4 Possible Variability Extensions

### 4.1 Further Generalization

If we want to generalize the project even more, would move abstract behaviors & features and make them into components that are a throughly flexible & reusable as possible. One route is to more fully leverage the power of Model-Driven Engineering (MDE) By high level models, we can define the behavior and structure of systems in a way that yields more flexible and maintainable codebases. If your system is a software which has user interactions, so based on that example we could create some models representing the roles and permissions of users. We could then interpret or compile these models to executable code and make it easier for us to iterate on new ideas without compromising the stability of the original codebase (unless they do not have tests).

And finally, turning a more component-based architecture should make the project less specific. Yes, components could be written to clear interfaces and defined interaction protocols, which can be developed separately and coordinated as needed. This not only improves reusability, but it also allows for the easy addition of new features. For instance, in a web application handling authentication could be abstracted into a component that can accept alternative authentication providers (OAuth, LDAP. . . ) thus avoiding changes to the core of the application logic.

### 4.2 Envisioning a Software Product Line

A Software Product Line (SPL) based on the project can be envisioned by identifying the core features and potential variations and organizing them into a feature model. This SPL would allow for the systematic production of different variants of the software, tailored to specific customer needs or market segments.

#### 4.2.1 Core Features

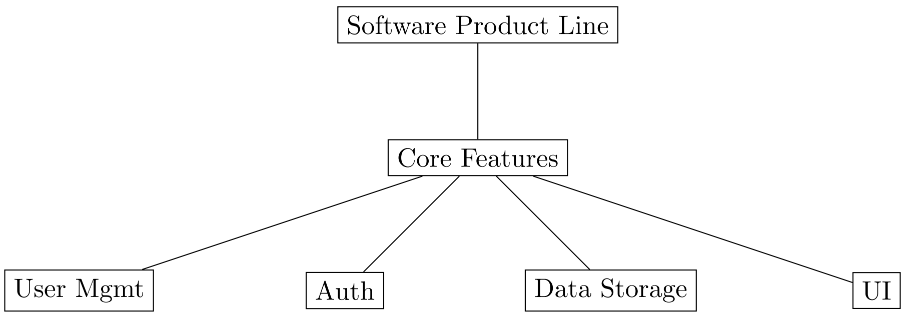
- **User Management**: Basic user registration, login, and profile management.
- **Authentication**: Support for multiple authentication methods (e.g. password-based, OAuth, biometric).
- **Data Storage**: Integration of databases to store user data and application-specific information.
- **User Interface**: A responsive and accessible user interface with customizable themes.

#### 4.2.2 Optional Features

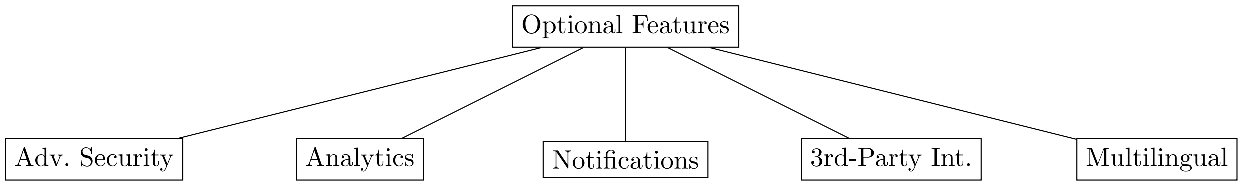
- **Advanced Security**: Two-factor authentication, encryption of sensitive data.
- **Analytics**: User activity tracking, reporting, and analytics dashboards.
- **Notifications**: Email, SMS, and push notifications for user engagement.
- **Third-Party Integrations**: Integration with external services and APIs.
- **Multilingual Support**: Localization and internationalization for different languages.

### 4.2.3 Feature Model

In figure 2 and 3 we can see the models for core and optional features:



Figur 2: Core Features of the Software Product Line



Figur 3: Optional Features of the Software Product Line

### 4.2.4 Additional Features

To enhance the SPL, additional features could be considered:

- **AI-Powered Recommendations**: Personalized recommendations based on user behavior and preferences.
- **Modular Plugins**: A plugin system allowing third-party developers to extend the functionality of the application.
- **Real-Time Collaboration**: Features that enable multiple users to collaborate in real time within the application.
- **Offline Mode**: Functionality for offline access and synchronization of data when the user reconnects.

By systematically organizing and managing these features within an SPL, the project can efficiently produce tailored software variants, meeting diverse customer requirements, while maintaining high quality and reducing time-to-market.

## 5 Sources

- Lecture 3. (2024). *Variability Implementation.* TDT4250 – H2024.
- NTNU. (2024). *Chess2 repository.* Retrieved from https://gitlab.stud.idi.ntnu.no/sandesl/chess2