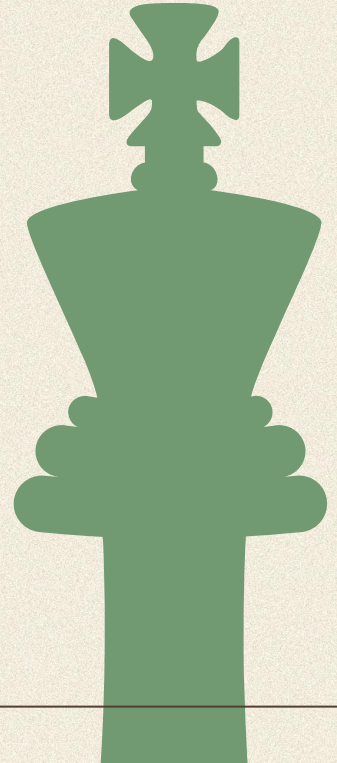
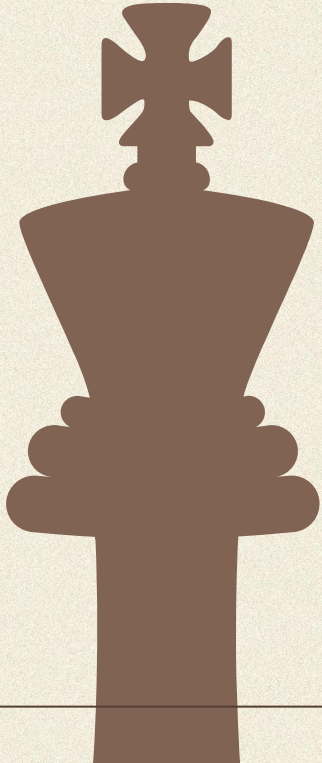




# Design Document

Phase2 - Group55







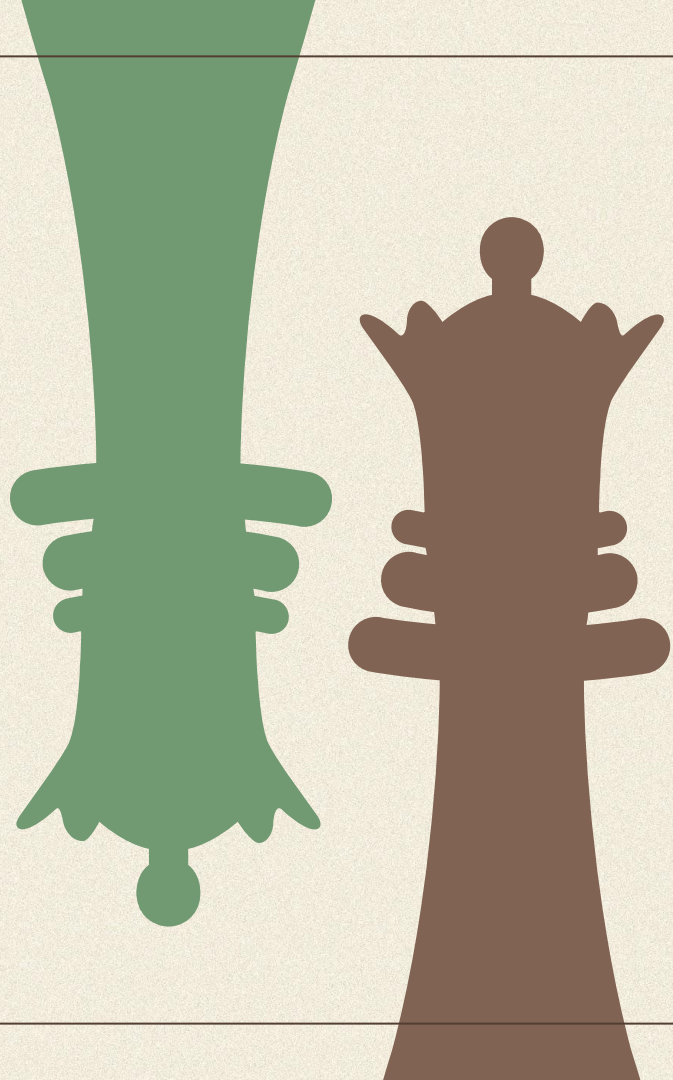


# CONTENTS



- Updated specification
  - Diagram + explanation of each component
  - Description of major design decision
  - Description of how the project consistent with SOLID design principles
  - Description of how the project adheres to Clean Architecture
  - Description of which packaging strategies used
  - Summary of design patterns implemented
  - GUI Demo
  - Progress report
  - Accessibility report
  - Future extensions
- 
- 





◆ 01 ◆

Specification



# In Short...



## Rating system

Implements Glicko2  
performance rating system



## LAN

Able to play 2 player  
chess on same internet



## Database

Able to keep user  
information in database



## 4 player chess

Able to play four player  
chess game (WIP)



## GUI

Able to work flawlessly  
with 2 player chess.



## More

Future Expandability



# The Full Specification

## Android App Starts With User Log In

Users can log in with their username and password, play as a guest, or sign up. This is logged in our database.

## After the User is Logged In

Once logged in, users can view profile statistics, including Glicko competitive rating, winrate, and a representation of the amount of each enemy piece they have taken. Users can also start a 1v1 game of Chess in hotseat locally or played over a LAN connection.

When the user starts the game, another player can log in with their username and password or play as a guest.

## Chess

Chess will follow the standard rule set. A turn-based 1v1 game between white and black pieces on an 8x8 board with 16 pieces per player. Each turn, a player can select and move 1 piece. There are 6 different pieces (Pawn, Bishop, Knight, Rook, Queen, King) which all move in different ways. A player can capture an opposing piece by moving into their tile, with exceptions for the pawn and the king.

- Pawn: Can move 1 tile forward. If it's the first time it's moving, can move forward 2 tiles. Can only taking pieces that are one tile away diagonally, or by the rules of *en passant*.
- Rook: Can move in all cardinal directions (NSEW)
- Bishop: Can move diagonally.
- Knight: Can move 2 tiles in one direction + 1 tile in the perpendicular direction (an L shape).
- Queen: Can move in all directions (NSEW + Diagonally)
- King: Can move one tile in any direction (NSEW + Diagonally). Cannot move if tile leaves king in line of sight of enemy Piece (check).

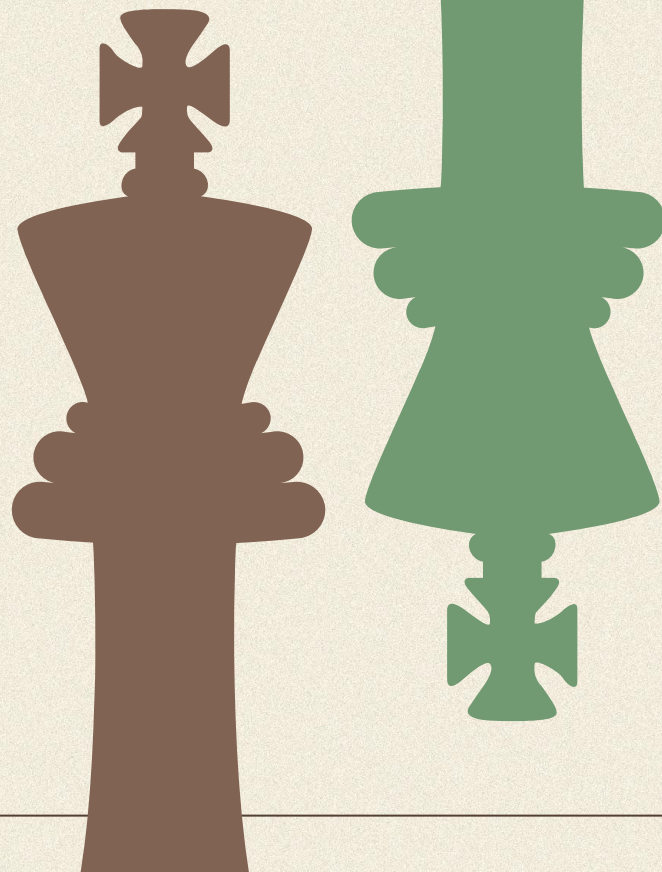
The game is won by putting the opposing King in a position where no matter what the opponent does on their next turn, the King can be taken. This is referred to as checkmate.





♦ 02 ♦

Diagrams  
+explanations

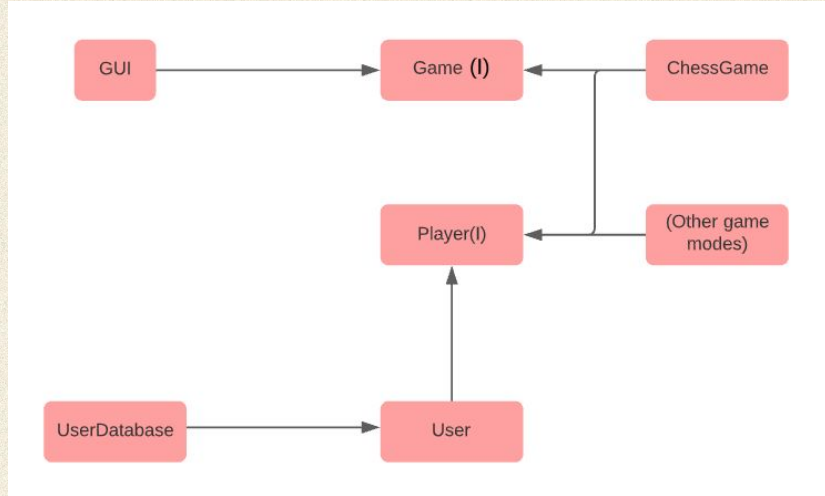




# ChessGame

## Key design features

- ISP: the Game(I) and Player(I) allows switching game modes cleaner.
- DIP: The interfaces also allow the development of code independent of the development of other parts of the code.

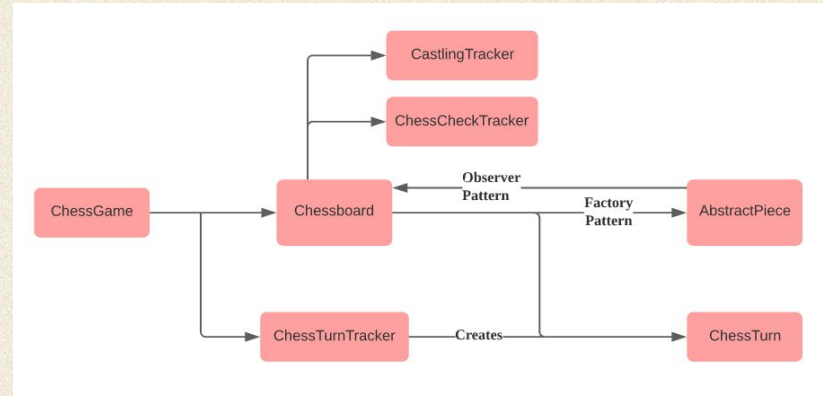




# ChessGame

## Key design features

- CastlingTracker and CheckTracker was split from Board, as Piece doesn't need access to it.
- Factory design pattern was used to create several subclasses cleanly
- Observer Pattern was used on Pieces such that pieces keep track of their coordinates
- Custom Coord class used ubiquitously to simplify vector math.

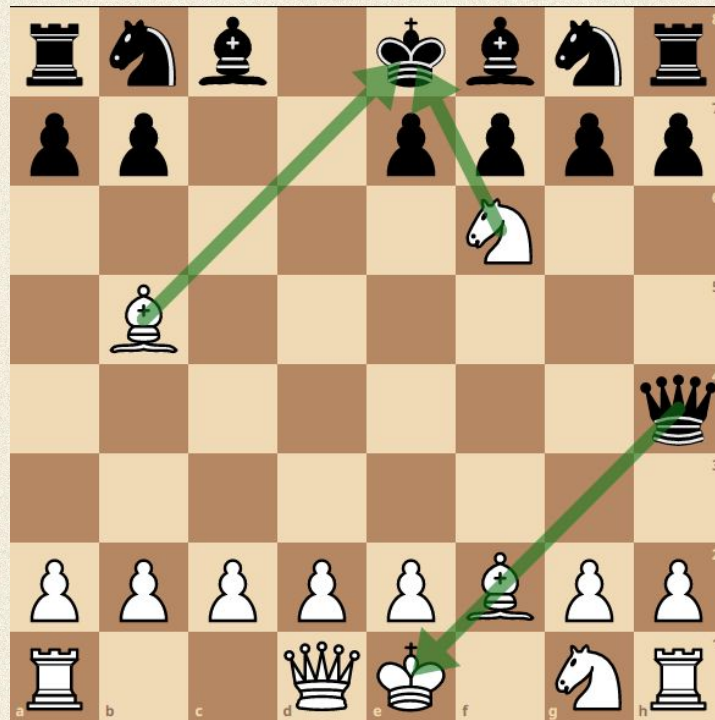




# ChessGame

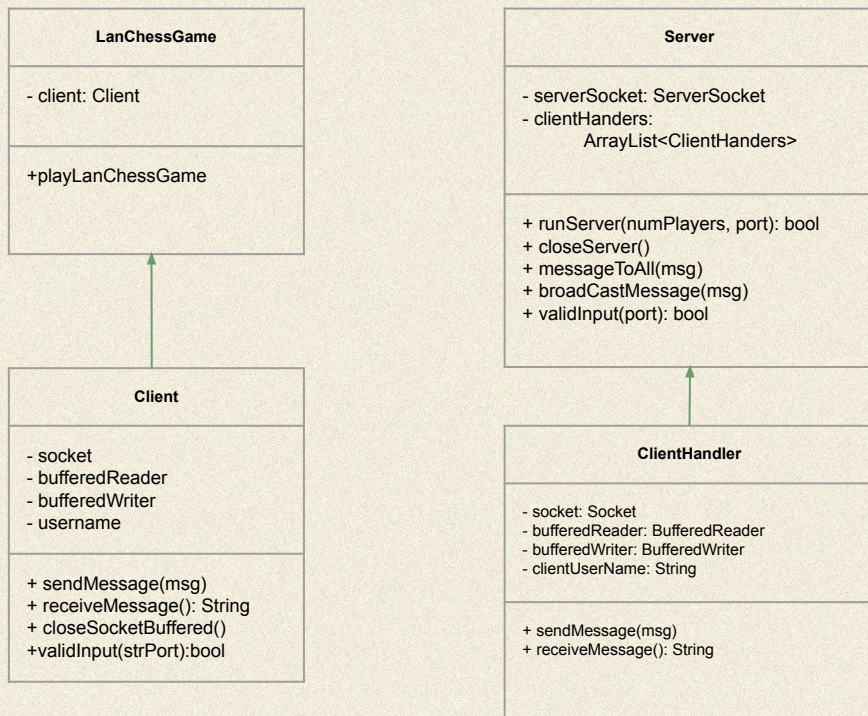
## Key features

- Castling (KQkq)
- En passant
- Checks (line of sight mechanism)
- Preview Legal moves
- Promote pawn





# LanChessGame extends ChessGame



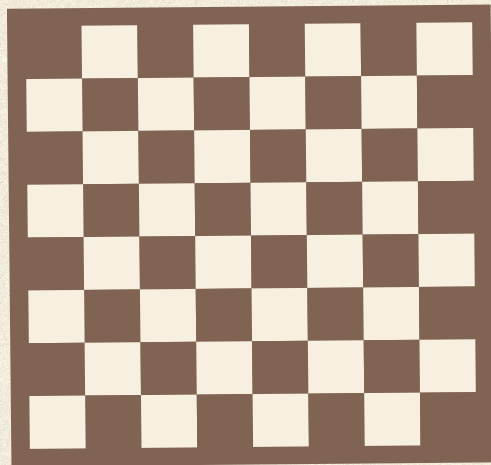
Client class helps LanChessGame to send and receive their moves to the server. Server class gives connection between two players. When client connects to the server, ClientHandler helps Server to keep informations needed to communicate with Client.

I've built methods that are needed to communicate between Client-Server and I named them straight forward to allow others to use it without guessing what each method does.



# OriginalChessGame

Take turns on one local device



Client1

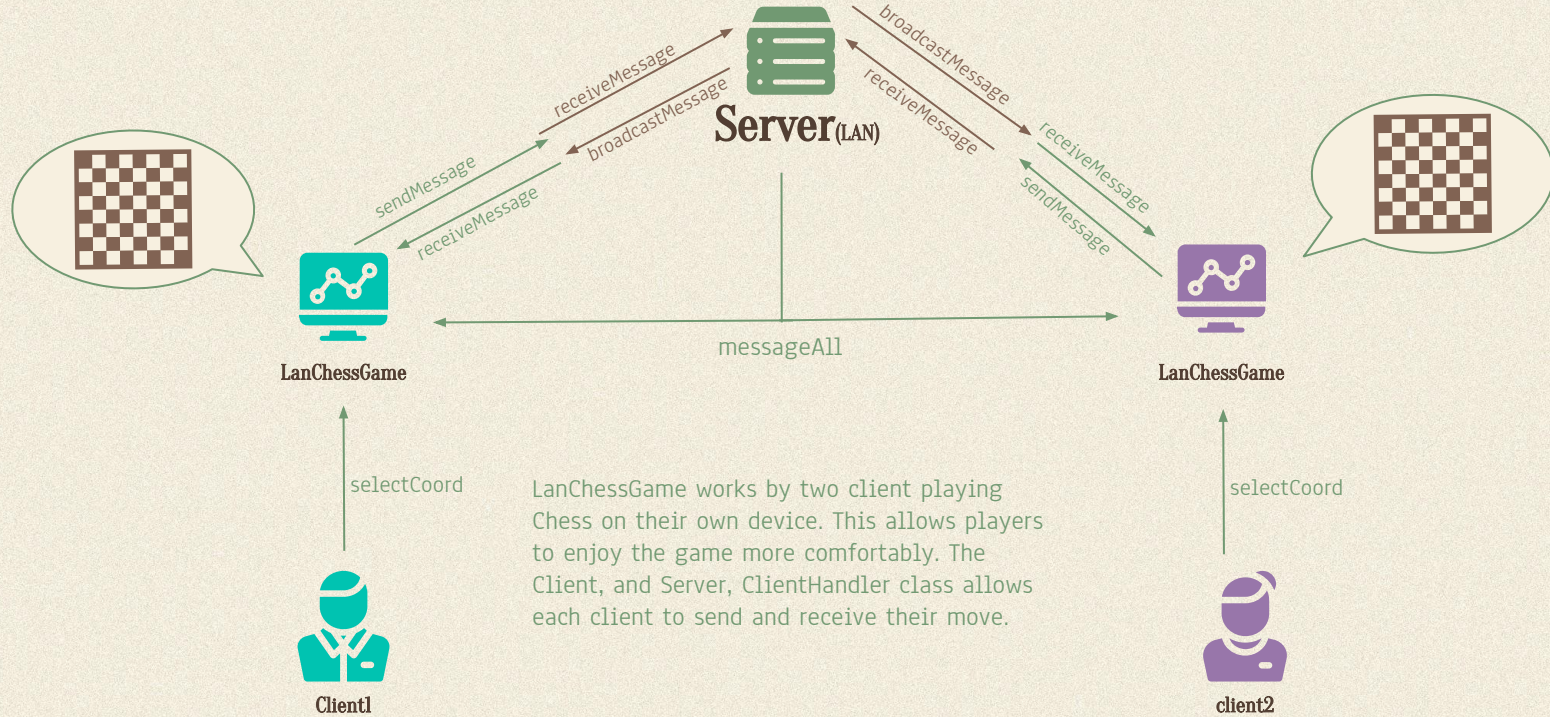


client2

ChessGame for hotseat works by two players paying on one device. Which is very inconvenient. That is why we've implemented LanChessGame.

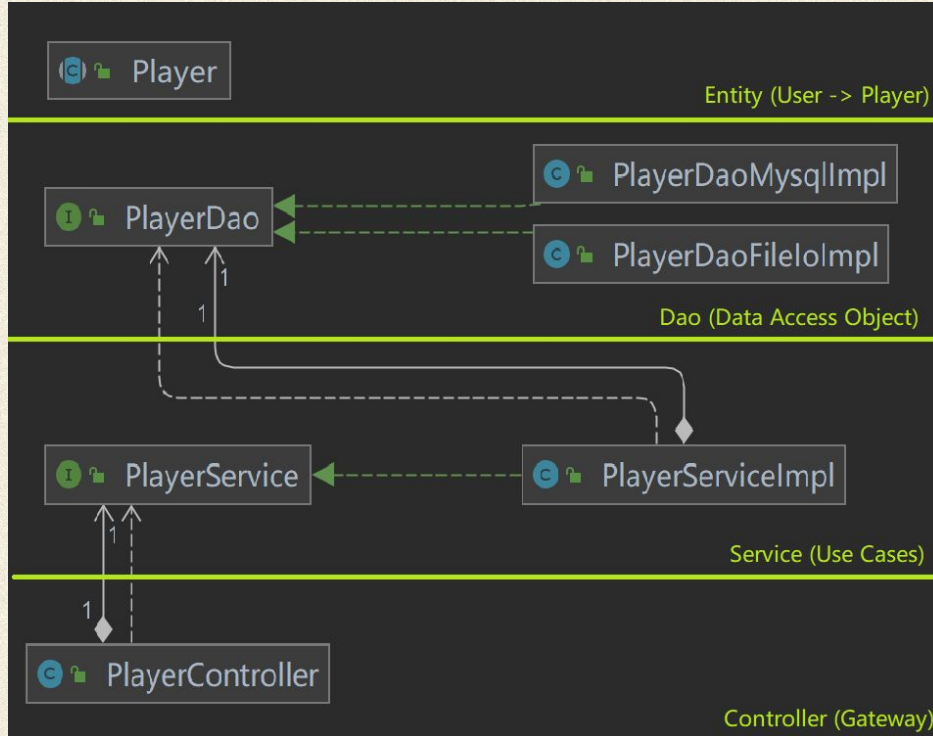


# LanChessGame-explanation





# DataBase--Clean Architecture



## PlayerController

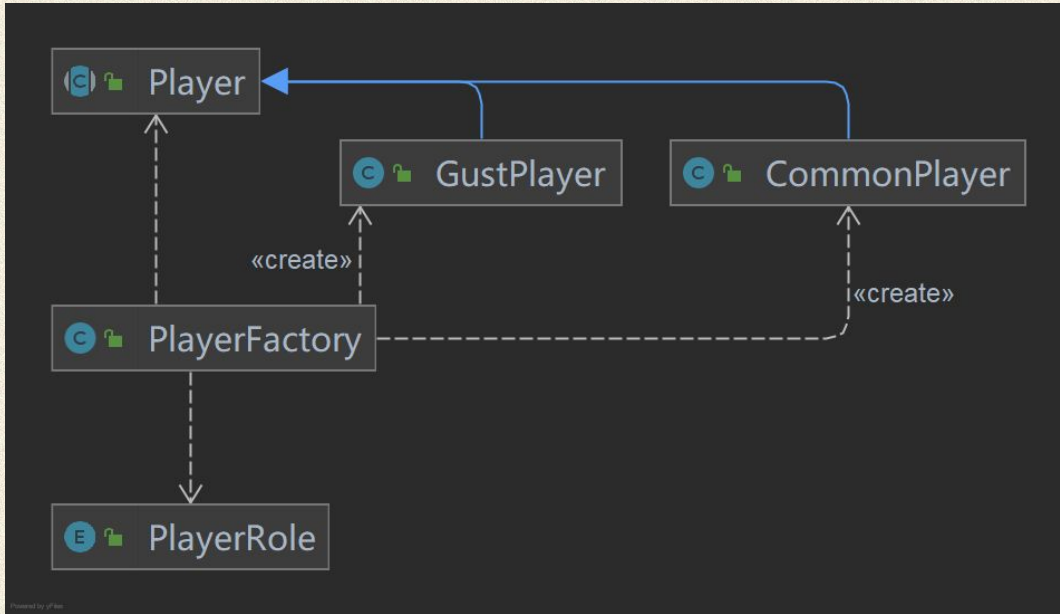
-PlayerService playerService

- + signIn(String, String): boolean
- + singUp(String, String): boolean
- + getAllPlayers(): List<Player>
- + update(Player): boolean
- + changeName(String, String): boolean
- + changePassword(String, String): boolean

PlayerDaoMysqlImpl: Persistence via MySQL  
PlayerDaoFileIoImpl: Persistence via Serialization



# DataBase--Design Pattern



Usage: Factory Pattern

Kind: Simple Factory Pattern

Instantiate different player subclasses



# DataBase--SOLID Principles

Single responsibility principle (SRP)	eg: Player, PlayerRole, PlayerFactory
	Each class is responsible for one single thing
Open/closed principle (OCP)	eg: Player(Abstract), CommonPlayer(Subclass) GustPlayer(Subclass)
	Use abstract class inheritance, open to extensions, closed to modifications
Liskov substitution principle (LSP)	eg: Player(Abstract), CommonPlayer(Subclass) GustPlayer(Subclass)
	Both subclasses can replace the parent Player, but not the reverse
Interface segregation principle (ISP)	eg: PlayerDao(Interface), PlayerService(Interface)
	Each for different functionality, no redundant need to be implemented
Dependency inversion principle (DIP)	eg: PlayerServiceImpl depend on PlayerDao(Interface) & Player(Abstract)
	High do not depend on lower, dependencies are on abstract or interface



# DataBase--Test

Framework : JUnit	
PlayerControllerImplTest	signIn, signUp, update, getAllPlayers, changePassword, changeName
	Test Passed: 6 of 6 tests
PlayerServiceImplTest	getAllPlayers, changePassword, changeName
	Test Passed: 3 of 3 tests
PlayerDaoImplTest	add, getByName, delete, update
	Test Passed: 4 of 4 tests



# The Rating System

<b>GlickoCalculator</b>	<b>Result</b>
- DEFAULT_RATING - DEFAULT_VOLATILITY - DEFAULT_DEVIATION - A few other default values for algorithm.	- WIN_POINTS - DRAW_POINTS - LOSS_POINTS - isDraw - player1 - player2
+ calculateRating(player, result list) + calculateNewDeviation() + ratingByOutcome + many conversion functions.	+ validPlayers(player1, player2) + participated(player) + getScore(player)
<b>GlickoRating</b>	<b>ResultsOverRatingPeriod</b>
- rating - deviation - volatility - numResults	- List<Result> results - Set<GlickoRating> participants
+ just a whole lot of getters and setters	+ addResult(winner, loser) + addDraw(player1, player2) + addParticipants() + clearResults()

- Originally wanted to use ELO system.
- Glicko system, designed by Harvard professor Mark Glickman, is better.
- Referenced from public domain document on mathematical rating model, see here: <http://www.glicko.net/glicko/glicko2.pdf>
- Glickman scales the scores for use in the algorithm, and also for comparison to more traditional rating metrics (i.e. ELO).
- **Lots** of lovely, not at all confusing, math.

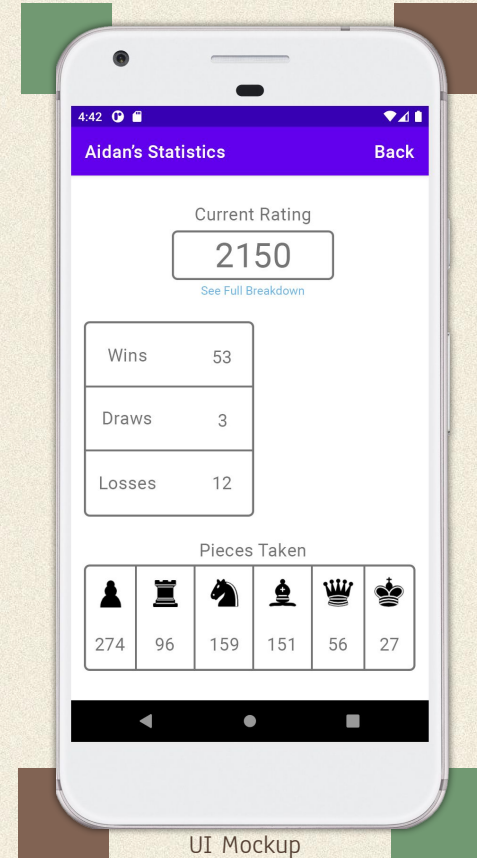
Let  $a = \ln(\sigma^2)$ , and define

$$f(x) = \frac{e^x(\Delta^2 - \phi^2 - v - e^x)}{2(\phi^2 + v + e^x)^2} - \frac{(x - a)}{\tau^2}$$



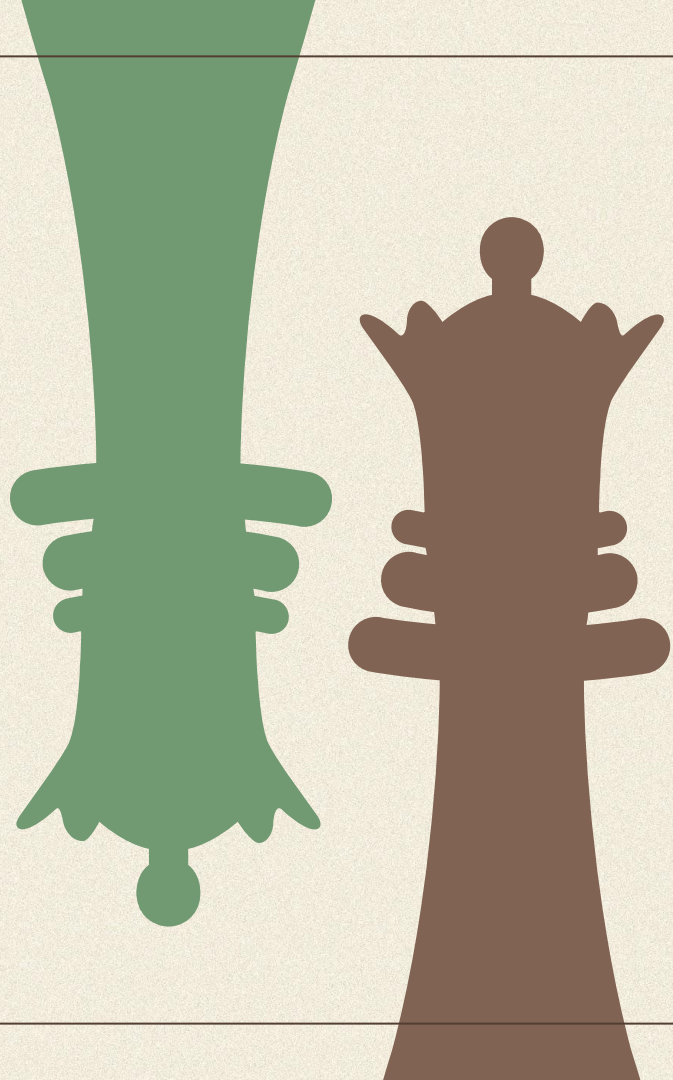
# The Rating System & Stats

- User will be able to view their profile's statistics, including wins, losses, draws, a count of all the pieces taken, and finally a breakdown of their competitive rating.
- This has *a lot* of potential in the future for interesting expansions to gameplay analysis, was too complicated to implement more crazy things right now.
- While the back-end implementation is largely complete, front-end development was not completed due to time restraints.



UI Mockup





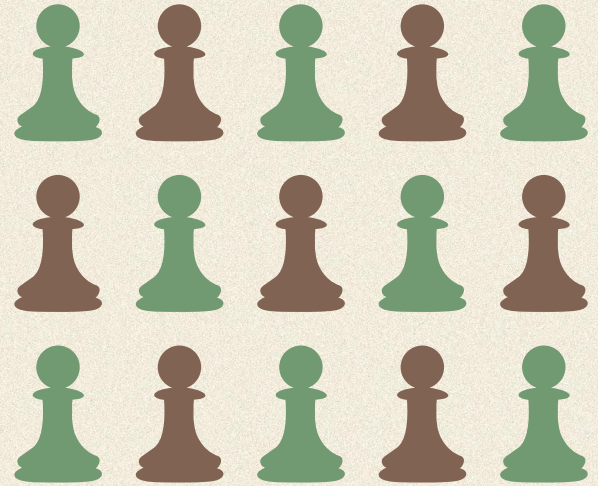
◆ 03 ◆

Design  
Patterns



# Design Decisions

We used the Factory Design method to create Piece objects, since there are 6 different kinds of pieces, that are part of the Chess Game.

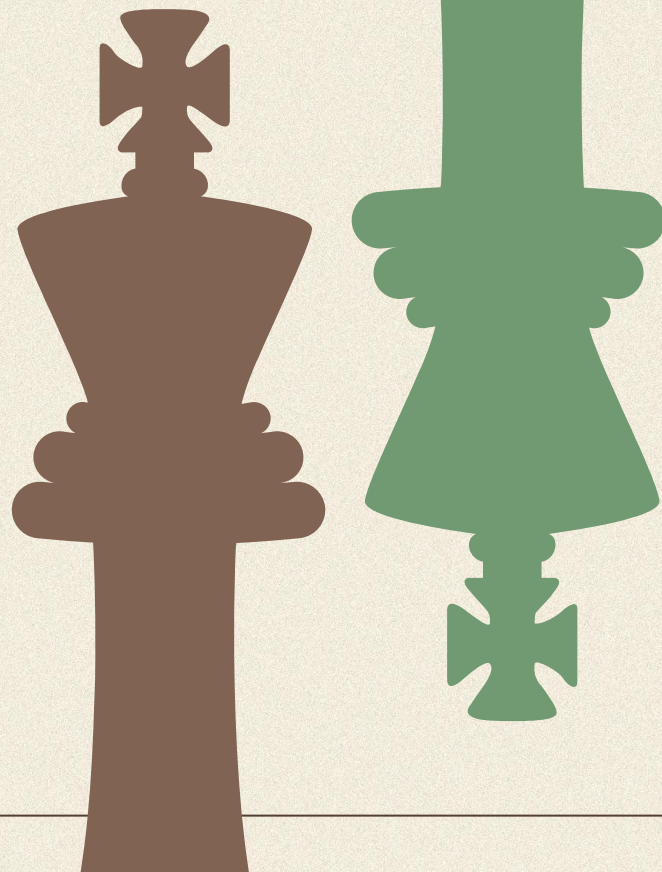




# 04

## SOLID

Design principle





# SOLID design principle



## Single Responsibility

Splitting the Chessboard into CastlingTracker, CheckTracker, and PieceTracker



## Interface Segregation

A player interface is used so that the Chessgame only accesses needed features



## Liskov Substitution

Using well placed Interfaces, we can easily swap one game mode with another. Allowing the reusing of code even if it has different rulesets.



## Open Closed

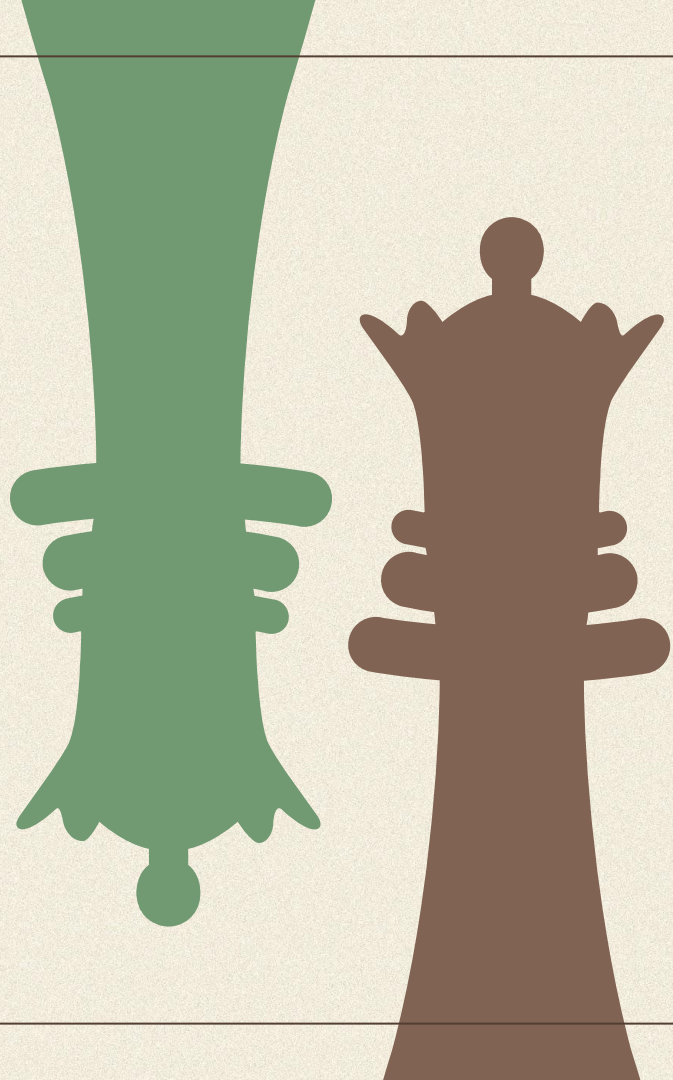
We implemented needed methods to modify each class's behavior. So open for easy extension, but closed for modification.



## Dependence Inversion

Well placed interfaces allow developers to work on separate parts of the code with reduced dependency to other parts of the code





◆ 05 ◆

Clean  
Architecture



# Clean Architecture



## Layers

Divided up code by each class's expectations, which allowed us to split up classes by their layer better.



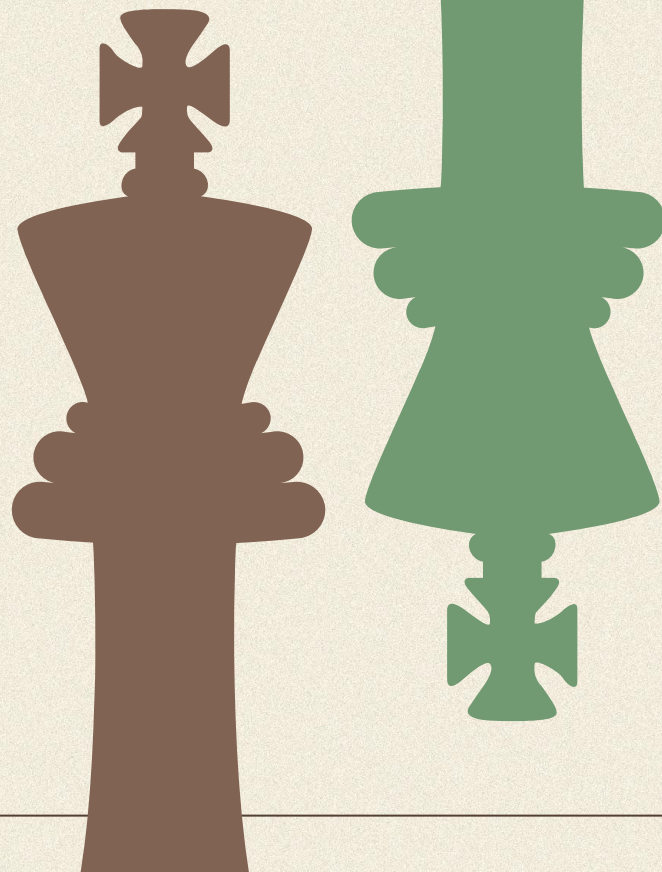
## Dependency Rule

When we were programming, we built class from inner layer to outer layer. This prevented inner layer have dependency of other layers.



◆ 06 ◆

Packaging

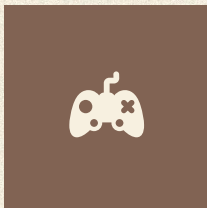




# Package by component

## Chess

ChessBoard  
ChessGame  
Piece

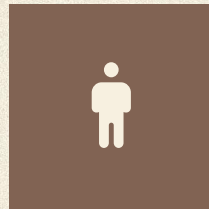


## Lan

Client  
Server  
ClientHandler

## Glicko(Ranking)

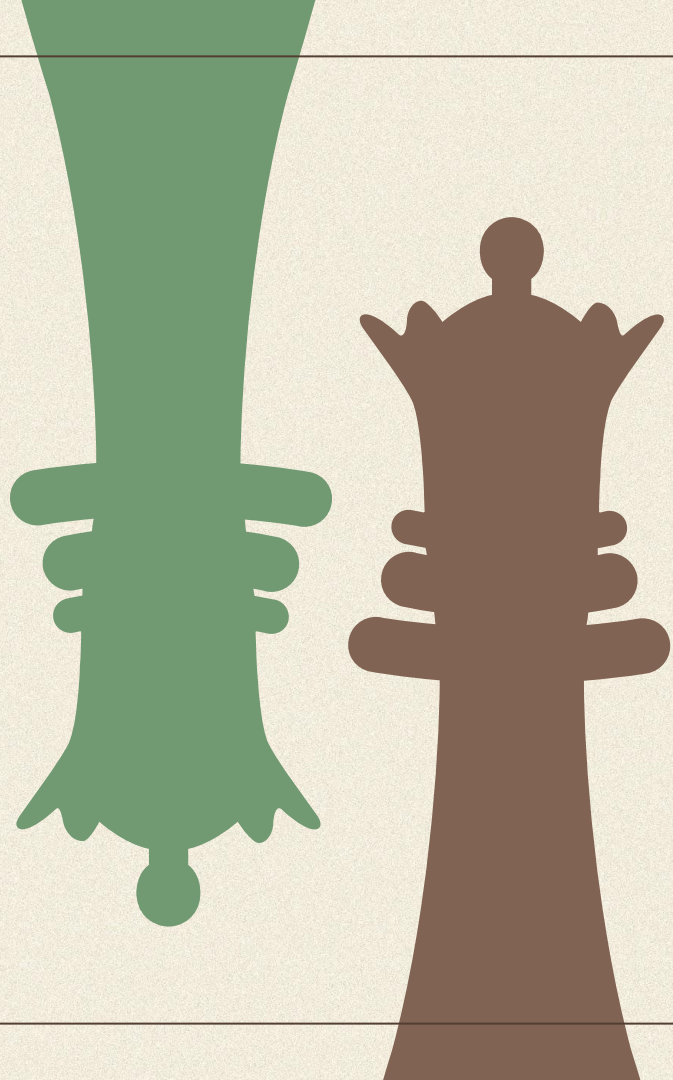
GlickoCalculator  
GlickoRating  
Result



## Player

Player  
GuestPlayer  
CommonPlayer





◆ 07 ◆

Design  
Patterns



# Factory Design Pattern



## PieceFactory

Instead of initializing each piece, such as King, Pawn, Bishop... use factory design pattern to simplify.



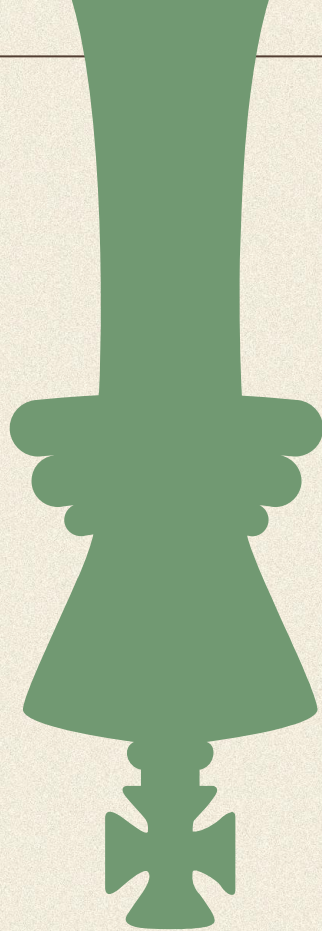
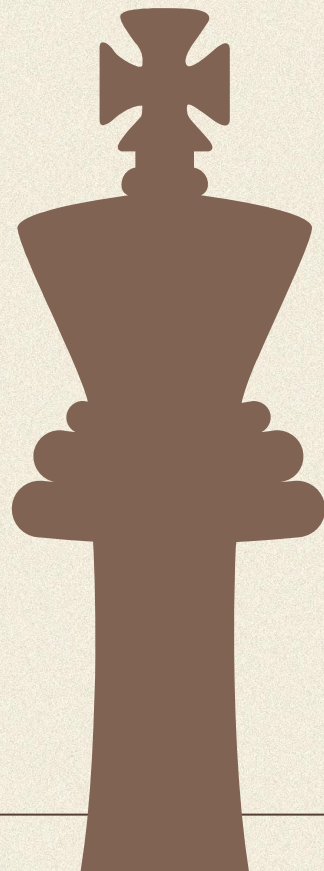
## PlayerFactory

Used to create different types of players using factory design pattern.



◆ 08 ◆

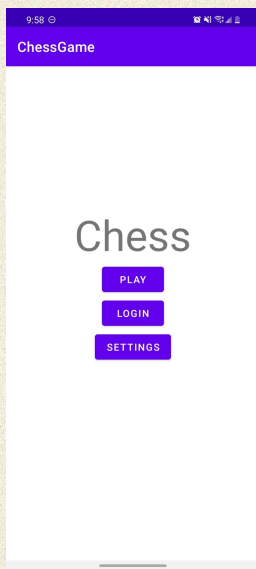
GUI



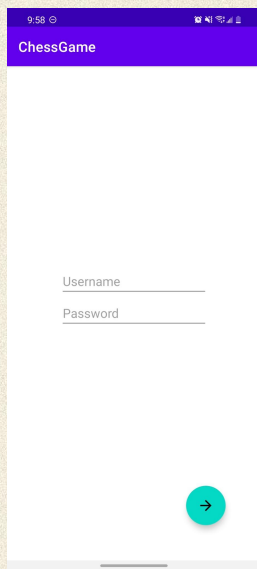


# GUI Screenshot

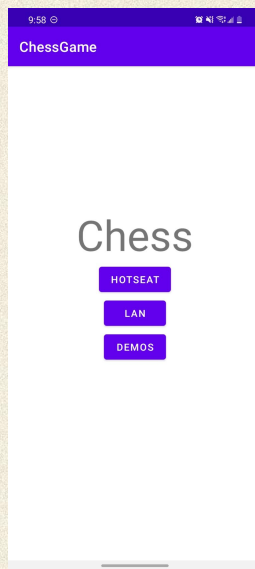
MainScreen



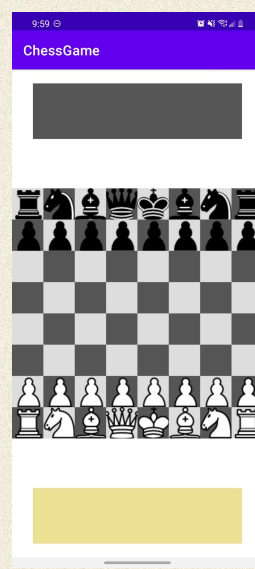
Login



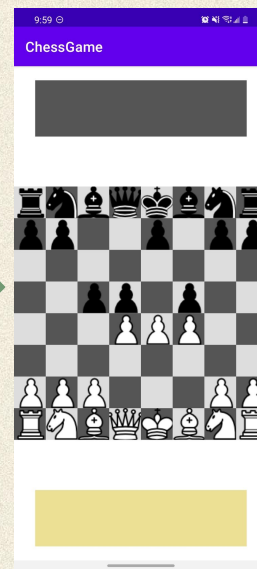
Menu



HotSeatChess



Yellow means that  
It's your turn.

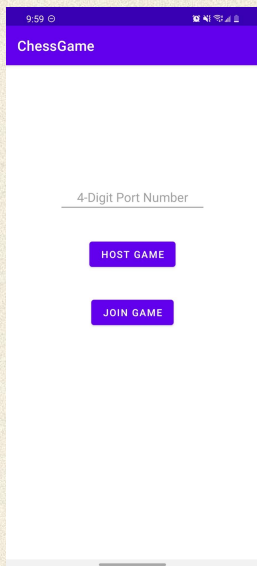




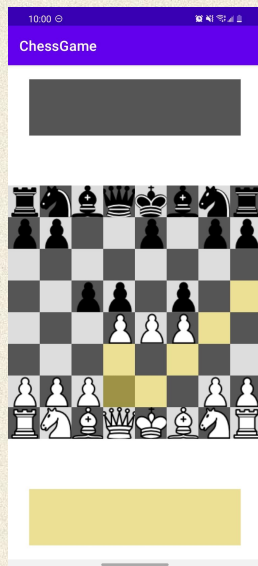
# GUI Screenshot

## LanChessGame

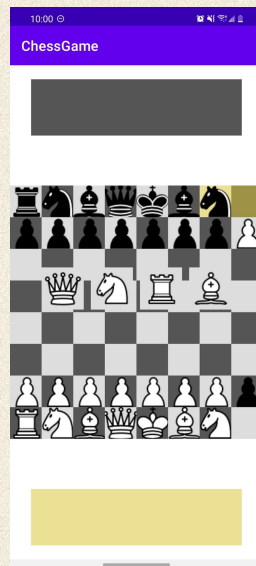
rest are the same



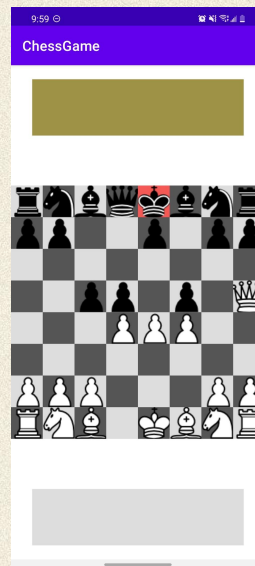
## LegalMoves



## Promoting



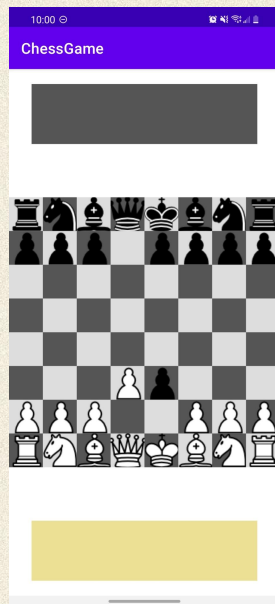
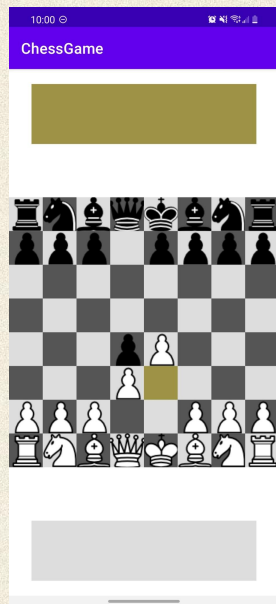
## CheckState



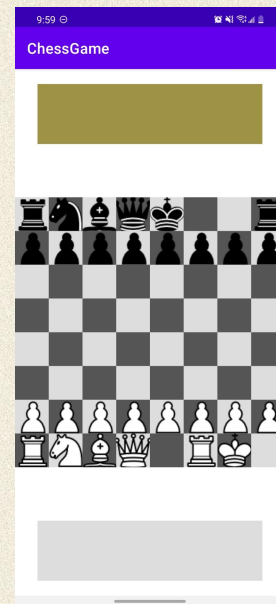
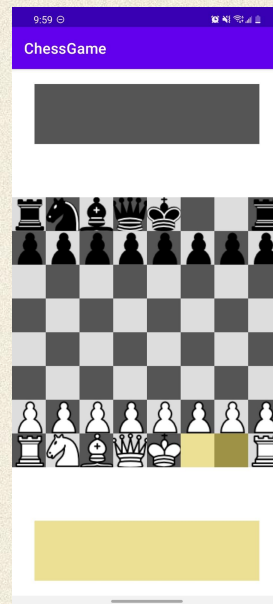


# GUI Screenshot

En passant

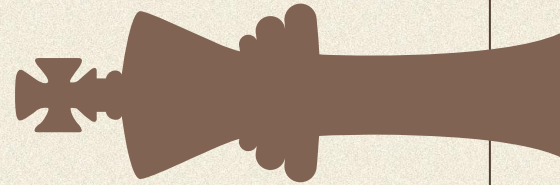
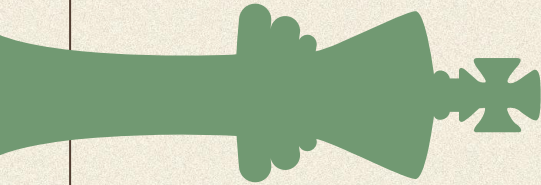
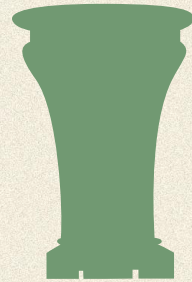
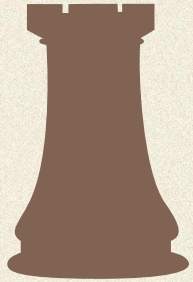


Castling





# Progress Report





# Since Last We Saw You...



## Giwon

Giwon worked on developing and finishing the core chess mechanics, developing the hotseat version of the game.



## Ang

Ang created and implemented the database, as well as creating a lot of unit testing for the program.



## Matthew

Matthew mainly worked on further cleaning up and developing the GUI, allowing for easier user interaction and a smoother overall experience.



## Jong Eun

Jong Eun was the main contributor to developing the Lan version of the Chess game, as well as the design document.



## Kole

Kole worked on his 4-player chess game mode.



## Aidan

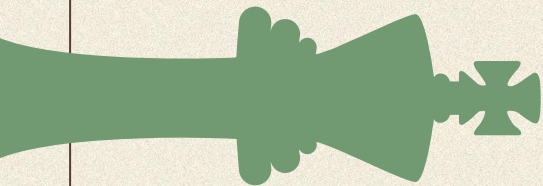
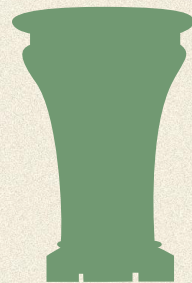
Aidan worked on implementing the Glicko rating system into the game, marrying it to the code base, as well as working on the design document.





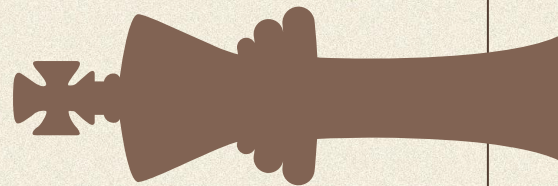
♦ ♦ ♦

# Accessibility



# Report

♦ ♦ ♦





# Principle of Universal Design

## 01 **Equitable Use**

Currently lacking here. Can work on introducing intuitive sound effects and narration for the hearing impaired. Also we can also add colour customization options to facilitate easy play for the visually impaired.

## 02 **Flexible in use**

Again, colour customization options would go a long way here. Text and image sizing, contrast, tap/drag options, as well as localizations would be great additions.

## 03 **Simple and Intuitive Use**

Currently our design GUI is very simple and very easy to understand for those who have played chess before. For those who haven't played chess it could be a bit confusing, especially the rating system.



# Principle of Universal Design

## 04 Perceptible Information

Currently, we do not have any informations about how our program works. We can add simple videos of how to play chess(how to move, what checkmate state is and etc).

## 05 Tolerance for Error

Chess, as a competitive game, doesn't allow for much tolerance of error; there can't be a redo button. We will investigate further how to improve this in other ways.

## 06 Low Physical Effort

Our GUI is built on android, so the game can be played on a mobile device, only requiring tapping of the screen to function. Investigating fun solutions for physically impaired users.

## 07 Size and Space for Approach and Use

The current focus was functionality, but in the future this will be taken further into consideration.



# The Future



## Expandability

- Online Multiplayer
- Matchmaking by Glicko Rating
- Enhanced Statistics Tracking
- User-Defined Gamemodes



## Refining

- Improve Accessibility
- Give app more personality with visuals and sound
- Make more robust tests, improve documentation
- Finish current feature implementations
- Fix stalemates





# Significant Pull Request<sup>(hyperlinked)</sup>

**01**

**Giwon**

Was able to figure out more efficient way for our program to work.

**02**

**JongEun**

This pull request notified group members that I got my Lan classes working.

**03**

**Ang**

Refactored Player with Clean Architecture which is main goal for our project.

**04**

**Kole**

4 player chess game...

**05**



**Matthew**

Was able to work on GUI, to help visualize chess in better way.

**06**

**Aidan**

Was able to implement rating system to our project (this pull is for the unfinished version).







THANKS

By Group55