Task Board

For (and including) monday 19/03 **Green marking = done**

Distribution of tasks:

(Tasks may be "invented" and solved without being entered on this board while working, but those should generally fall under the categories mentioned below. This is done to promote quick and dynamic workflow, but may also mean that this list may not be complete in terms of concrete tasks completed. Checking commits/content in the repository may serve that purpose better).

Eirin

- Implement castling methods in Rook and King
- Game class - create
- Implement pieces:
  - Rook
  - King
- Implement logic to capture
- General bugfixing

Marianne

- Create pawn tests
- Checkmate/natural ending implementation Game
- Pawn

Elise

- Fix knight and queen
- Draw GUI movehistory
- Implement pieces:
  - Knight
  - Queen
- Presentation
- Retrospective

Sofia

- Fix bishop
- Main menu - finish
- Learn libGdx
  create menu
- Implement pieces:
  - Bishop

Triki

- Learn libGdx
- Create gui for board

- Create listeners for clicked squares
- Create end-game scene
- Code quality control

*

Jonas M.W.

- Create simple AI
- Create intermediate AI
- Test AI

Jonas Trædal

- Create intermediate AI
- Connect AI to game/gui
- Test AI
- Update team plan

Stian

- Create code that writes to file with the respective users, with the new user and the default rating.
- Create a function that reads through the user file and check if the wanted username is already taken.
- Checking if the username is already taking
- Create tests for the RegisteredPlayer class
- Error handling taken username
- End game - calculate rating
- Win-loss statistics

Paraneetharan

- Sequence diagrams for the chess game
- Document control

Task 2 - requirements

Jonas Triki

- Keep track of ranking (winner statistics).

Elise Fiskeseth

- Chess Rules

Marianne Luengo Fuglestad

- Documentation (good practice)
- Finish task 2
- Add new requirements

Jonas Mossin Wagle

- Create account

- A human player must be able to play against a AI player

Plenum

- The simple machine player must make a move within 1 seconds.
- The intermediate machine player must make a move within 3 seconds.

Sofia Eika

- Photos and graphics open licence
- Source code graphic licence

Eirin Sognnes

- Easily expandable code
- Simple and easy to use GUI

Original task base and requirements:

(with some initial distribution of responsibilities)

Non-functional

Eirin:

Easily expandable code:

- Avoid code duplication
- Refactor code when necessary
- Descriptive documentation
- Good/descriptive variable and method names.
- Little to nothing of hard coding

GUI:

- Readable and comfortable text (contrast between background and text)
- Neutral and good-sized font
- Understandable terms (no obscure chess terms where it can be avoided)
- Visible markers on pieces and of valid moves.
- Good error messages when something goes wrong (i.e. illegal move)
- Distinct pieces (no confusion which piece is which)
- Simple menu, with good names and distinct choices (start game, view high score)
- Readable high score list, top on top, bottom on bottom.
- Regular chess board with regular markers for which spot on the board it belongs to.

Sofia:

- Find licence on everything that is not made by us.
- Create licence on everything we create.

Marianne:

Documentation

- Should give insight into why the code is written the way it is.
- Should describe what each API item (class, interface, package, method, etc.) does.
- Should be descriptive, but not redundant. For instance, a description of a method should give information beyond what the method name already does.
- Should be concise.
- The first sentence of a doc comment should be a summary sentence. It should contain a concise but complete description of the API item.
- Names and keywords should be written in <code> style in descriptions.
- Should be written in 3rd person. For instance, “gets the name” is preferred over “get the name”.
- Objects created from the current class should be referred to as “this”, not “the”.
- Should include all relevant tags (@return, @param, @author, @version, etc.).

Functional

Jonas:

Keep track of ranking:

- The system keeps the win and loss results records of the games played.
- The system should save the result of each game to a file.
- The system should load result of all games played when showing highscore.
- The system offers the ranking of players according to the level of intelligence.
- The system should save statistics about games lost, won (time spent).

Elise:

Rules of chess

- Enable the king to move exactly one square horizontally, vertically or diagonally.
- Enable the rooks to move any number of vacant squares horizontally or diagonally.
- Enable the bishops to move number of vacant squares diagonally.
- Enable the queen to move number of vacant squares horizontally, vertically or diagonally.
- Enable the knights to move to the nearest square not on the same rank, file or diagonal.
- Enable the pawns in their start positions to move two vacant squares straight forward.
- Enable the pawns in other positions than the start position to move one square forward.
- Enable the pawns to move one square diagonally if one of the opponents pieces are in the square.
- Enable all players to perform castling. Castling consists of moving the king two squares towards a rook and placing the rook on the other side of the king.
- Enable all players to perform en passant.
- Enable a pawn that has reached the eighth rank to promote to a queen, rook, bishop or knight.

Stian Fagerli:

Create account:

Functional requirements

- The system provides a registration function which enables a identification of a user.
- The system should uphold uniqueness of each user.
- The start-rating of every newly created user should be the same for everyone

Jonas Wagle:

A human player must be able to play against a AI player :

- The system offers the option to choose between a human player and a machine player.
- The system offers a difficulty option for the AI.

- The system should correctly start a game against an AI and not another human player.
- The system selects the correct difficulty of the AI

Plenum:
Simple AI within 1 second:
- Find all possible moves, pick random (within one second)

Intermediate AI within 3 seconds:
- Count current “points”
- Make a move, see if position is better, move if is, next if not.

Coding tasks:

Ranking:
- Save the result of each game to a file.
- Save statistics to each player.
- Show win/loss, game details, such as moves, time spent (?), etc.

Create account:
- Create code that writes to file with the respective users, with the new user and the default rating.
- Create a function that reads through the user file and check if the wanted username is already taken.
- Error handling if the username is already taking

Play against AI:
- Implement a function that start a game with an AI playing with given difficulty as an argument.

AI:
- Easy:
  - Get all possible moves
  - Make random move
- Intermediate:
  - Get all possible moves
  - Check if move is better, if is, move, else check next.
  - If none better, make random move.

Chess rules:
- Enable the king to move exactly one square horizontally, vertically or diagonally.

- Enable the rooks to move any number of vacant squares horizontally or diagonally.
- Enable the bishops to move number of vacant squares diagonally.
- Enable the queen to move number of vacant squares horizontally, vertically or diagonally.
- Enable the knights to move to the nearest square not on the same rank, file or diagonal.
- Enable the pawns in their start positions to move two vacant squares straight forward.
- Enable the pawns in other positions than the start position to move one square forward.
- Enable the pawns to move one square diagonally if one of the opponents pieces are in the square.
- Enable all players to perform castling. Castling consists of moving the king two squares towards a rook and placing the rook on the other side of the king.
- Enable all players to perform en passant.
- Enable a pawn that has reached the eighth rank to promote to a queen, rook, bishop or knight.

Board logic

- Create board
- Create pieces
- Pawn promotion
- Execute moves (check if king is in check after move - illegal)

Create GUI:

- Create startup menu
  - Login
  - Create new account
- Menu for start game
  - Start (AI (difficulty), regular)
  - Highscore
- Add images to all items
- Add markers for legal placements of pieces
- Markers for chosen piece
- High score list

Game:

- Start game
  - Single player
  - Multi player
- Save game (for later)
- Resign
- Checkmate (+stalemate (draw) + 50 move rule + threefold repetition)
- Connect GUI, AI, ranking og users
- End game <- calculate rating
- Error handling for taken usernames

Update documents from lab 2
Retrospect
Presentation

-