Chess

Final Report

Crandall Captains
Sam Vanturennout, Alexander Nguyen, Erick Hernandez, Rohan Kanumuri

Course: CPSC 224 - Software Development

I. Introduction and Project Description

This document serves as a summary of our project's progress and technical details, providing an overview of the engineering efforts involved in creating an interactive Chess application. It is designed to present a clear outline of the project's scope, major features, technical implementation, and user interface design, making it easier for stakeholders to understand the development process and goals.

The project aims to develop a fully functional Chess program for two players. Following the standard rules of Chess, including special moves such as castling and pawn promotion, the application emphasizes user-friendly design with a graphical user interface (GUI). The game supports local play on the same device and includes features like move history tracking. The ultimate goal is to deliver an engaging Chess experience with added utility for players.

This document also includes references to relevant Chess rules and mechanics, detailed software architecture, and a timeline for project completion.

Project Description

Our Chess application provides an engaging platform for two players to enjoy a classic game of strategy. The application adheres to the official Chess rules and incorporates the following features:

1. **Game Initialization**: Players can start a new game with the standard 8x8 board setup or load a previously saved game.
2. **User Interface**: The game features a visually appealing and intuitive GUI, allowing players to interact with the board using mouse clicks or touch inputs. Valid moves are highlighted when a piece is selected.
3. **Game Mechanics**: The application enforces all standard rules of Chess, including movement constraints, special moves, conditions like check, checkmate, and stalemate. Turn-taking is automated between the two players.
4. **Error Handling**: The game gracefully handles invalid inputs and provides informative feedback to players.
5. **Additional Features**: Players can track the move history with the logboard.

Our design ensures a balance between functionality and usability, making the application accessible to players of all skill levels. The software architecture employs modular principles with classes such as Game, Board, Square, Move, and Piece (alongside its specific subclasses), facilitating maintainability and scalability.

II. Team Members - Bios and Project Roles

Rohan Kanumuri is a computer science and business administration student at Gonzaga University, with a strong interest in full-stack development, artificial intelligence, and data analytics. His technical expertise includes Python, Java, C++, and web technologies such as

React.js and Node.js. For this project, Rohan contributed to the implementation of the chess game's castling, piece display, and start screen features. He also played a role in debugging and testing check/checkmate logic.

Sam Vanturennout is a computer science student who is interested in AI and machine learning. Prior projects include modeling with large data sets and implementation of the game Farkle. Sam's skills include C++, Python, and Java. For this project his responsibilities included working on pawn promotion, castling, en passant, creating the abstract piece class as well as the in-class presentation.

Alexander Nguyen is a computer science student who is interested in game design and machine learning. Prior projects include computational modeling and implementing the game Farkle. Alex's skills include C++, Python, and Java. For this project, Alex's responsibilities included the board, check, bishop, promotion, and the in-class demo.

Erick Hernandez Najera is pursuing a degree in computer science with a strong interest in AI and data science. Erick's skills include Java, object-oriented programming, and UML diagramming. In this project, Erick's responsibilities include implementing piece promotion, adding check detection logic, and enhancing the `isSquareUnderAttack` method to support pawn promotion.

III. Project Requirements

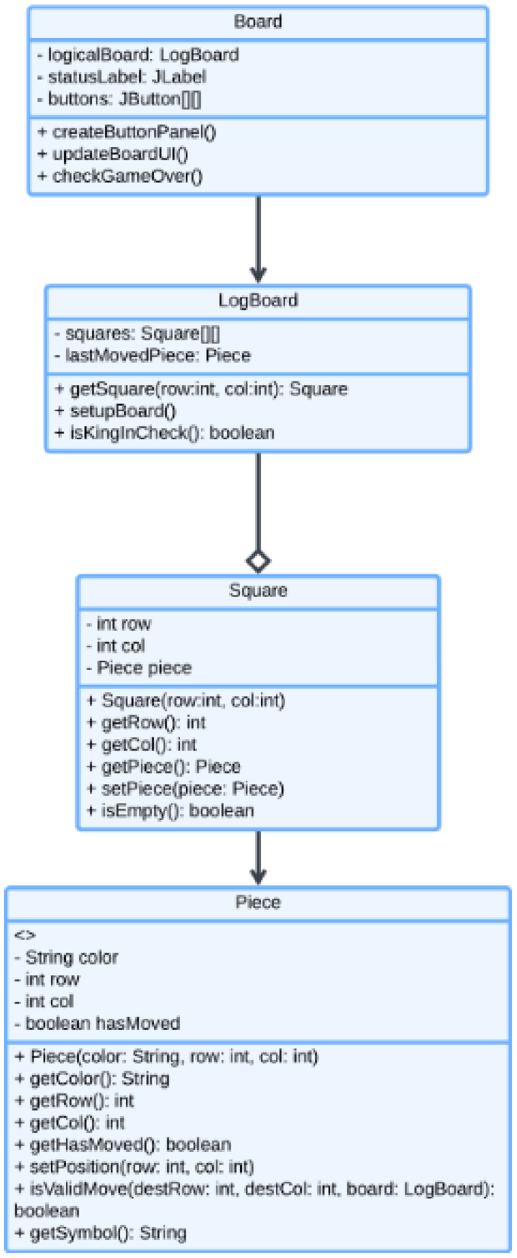
This section defines the requirements for our Chess application project, outlining the quantified expectations from the final design. These requirements were derived from the Project Plan Requirements document and provide an orientation for the reader on the objectives and deliverables of the project.

1. **Game Initialization**:
    a. A standard 8x8 chessboard setup with all pieces in their correct starting positions.
2. **User Interface**:
    a. Provide a clear and visually appealing chessboard and pieces.
    b. Support user interactions via mouse clicks or touch inputs.
    c. Highlight valid moves when a piece is selected to assist players.
3. **Game Mechanics**:
    a. Enforce standard Chess rules for all piece movements.
    b. Include special moves such as castling and pawn promotion.
    c. Detect and handle game states such as check, checkmate, and stalemate
    d. Alternate turns accurately between two players.
4. **Error Handling**:
    a. Ensure the application gracefully handles invalid inputs or unexpected actions.
    b. Provide informative error messages to guide the player.
5. **Additional Features**:
    a. Implement functionality to track move history.

These requirements ensure the Chess application achieves its goal of delivering a user-friendly, feature-rich experience while adhering to the official Chess rules. Additional implementation details are provided in subsequent sections.

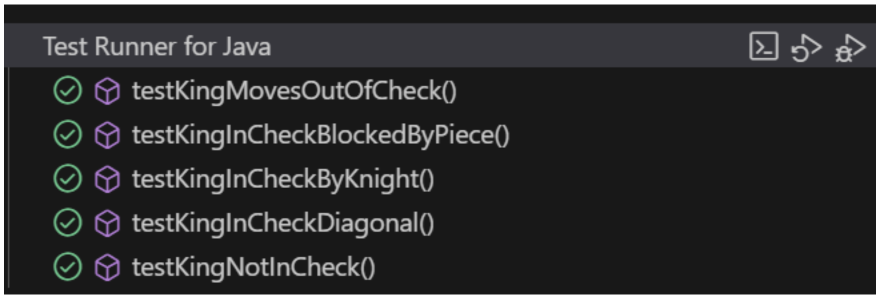
IV. Solution Approach

The game works by initializing a board. The board creates a LogBoard, buttons, labels. The LogBoard creates Squares. The Piece serves as the basis for the other pieces. The Piece is an abstract class. Every piece has a function that determines whether a move is valid or not. This is directly connected to the Board, which acts as the interface between the player and game. The Board will check if the move is valid; and if it is, the move will go through on the board. There is special logic added for promotion, castling, checkmate, and stalemate in the Board class. The LogBoard keeps track of specific variables such as if a specific piece has moved. This updates after every move from the Board. The Square class returns data for a specific square. This is needed for the LogBoard. The Pieces are initialized on the LogBoard, using the Squares. This project used three screens. The first screen appears when you run the program. The StartScreen class will create a screen that lets you click and start the game. It will lead into creating the Board screen, which is where the game takes place in. If you win or stalemate, it leads to the third screens which will allow you to quit or restart.



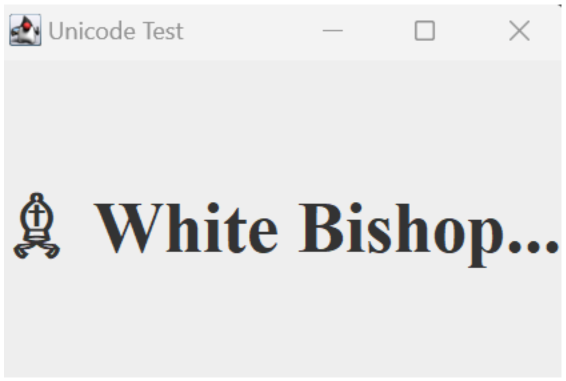
V. Test Plan

For this project we incorporated unit testing for most if not all components of our project. When creating the GUI our main form of testing was just trial and error as none of us had really worked with java swing before. We also had a variety of people play our game like our roommates to test the usability and clarity of the program.



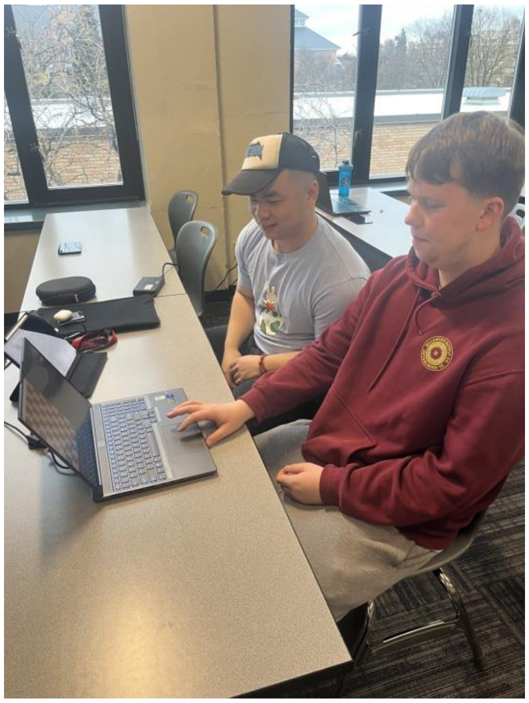
```
Test Runner for Java
✓ testKingMovesOutOfCheck()
✓ testKingInCheckBlockedByPiece()
✓ testKingInCheckByKnight()
✓ testKingInCheckDiagonal()
✓ testKingNotInCheck()
```

Some check unit tests^



Unicode Test

♗ White Bishop...



Icon Testing ^

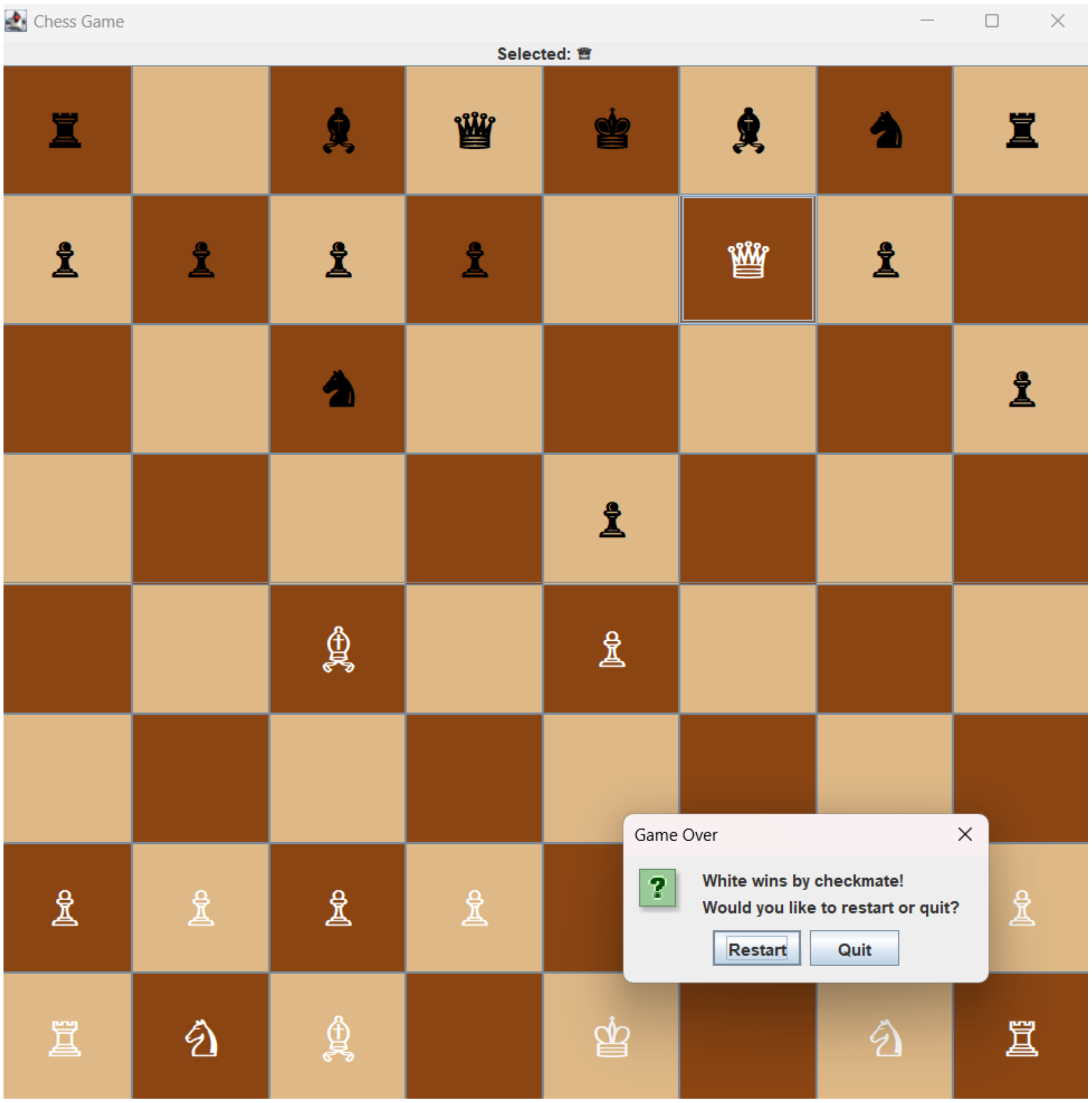
Roommates Testing^

Roommate Feedback: “Felt just like chess.com!”

“Exhilaring Gameplay!”

VI. Project Implementation Description

The project solution involves several major components. The LogBoard, acting as the logical board, is responsible for tracking all chess pieces and validating their moves. The Piece class encapsulates individual chess pieces like the king, queen, and bishop. The Board manages game logic, including turn handling, special rules, and win conditions. The graphical user interface (GUI) incorporates a chessboard (Board) and Squares to update the board state logically. The core gameplay focuses on piece movement and turn-based play, while special rules like castling, pawn promotion with user input, and check/checkmate detection are incorporated. Below are examples of the game in action, showing all three of the screens: the start (1st), the board(2nd), and the end screen (3rd).



Game Over

White wins by checkmate!
Would you like to restart or quit?

Figure 1: Checkmate in action (Third Screen)

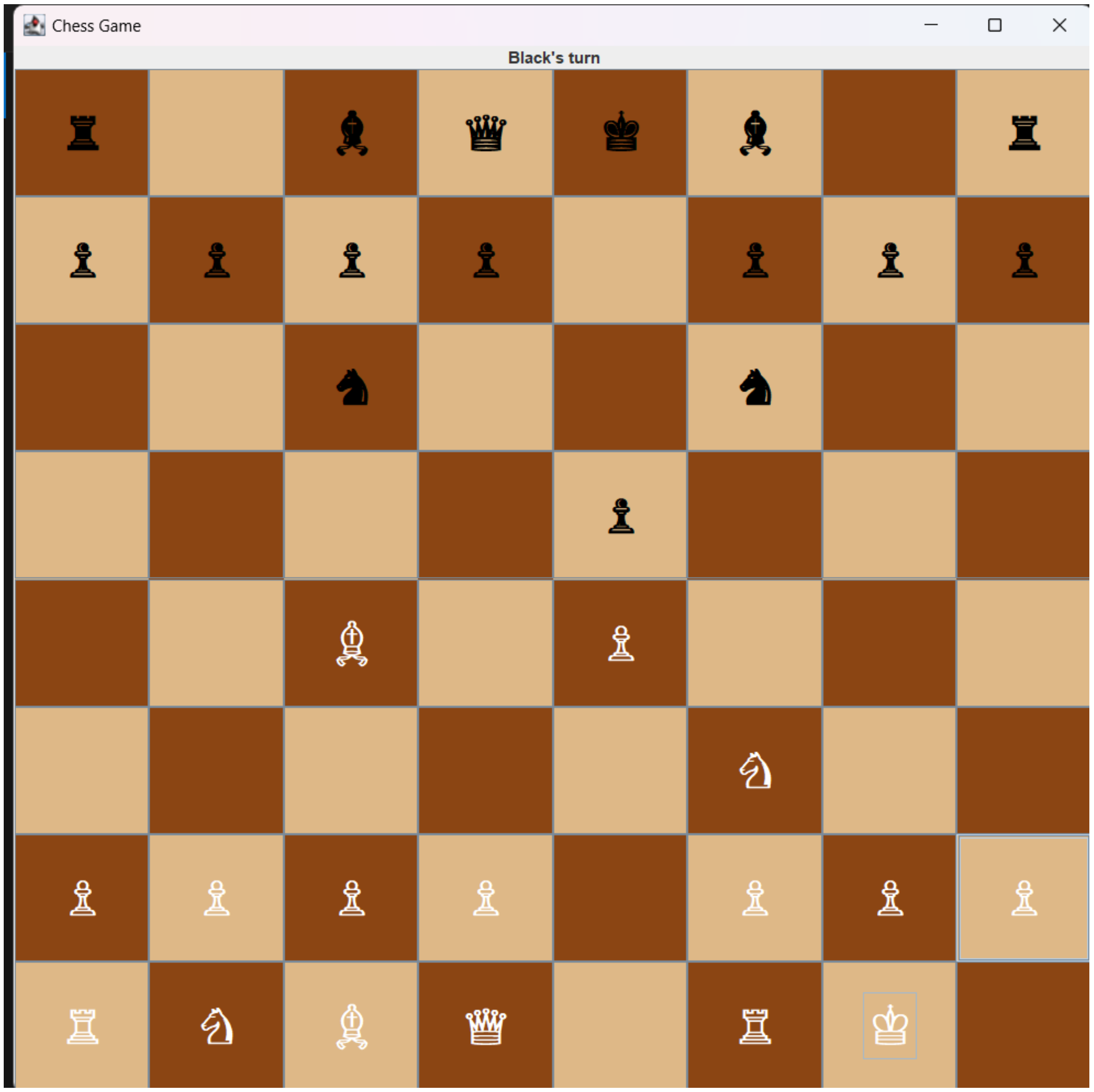


Figure 2: Castling in Action (Second Screen)

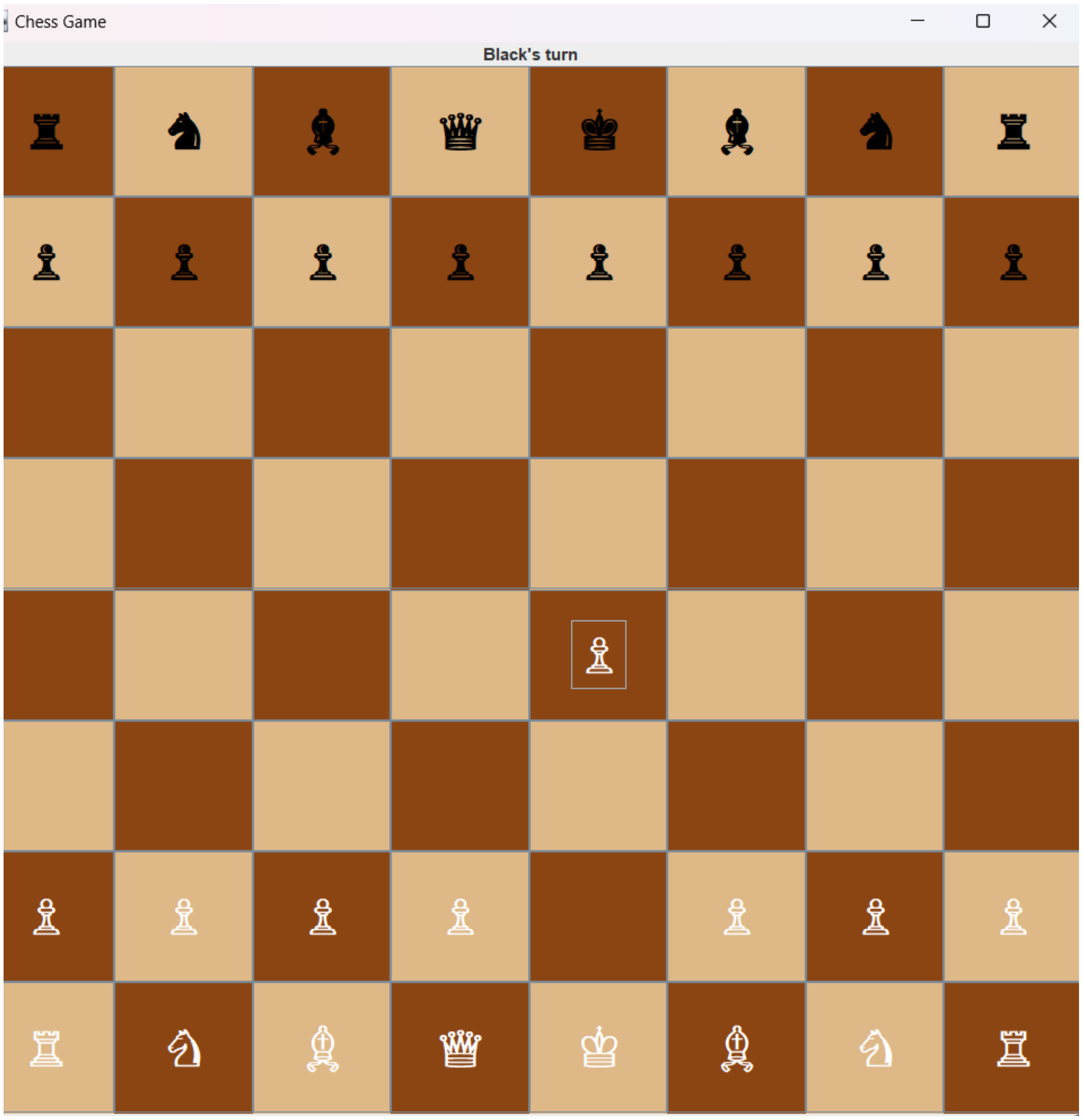


Figure 3: Pawn moves 2 squares on first move (Second Screen)

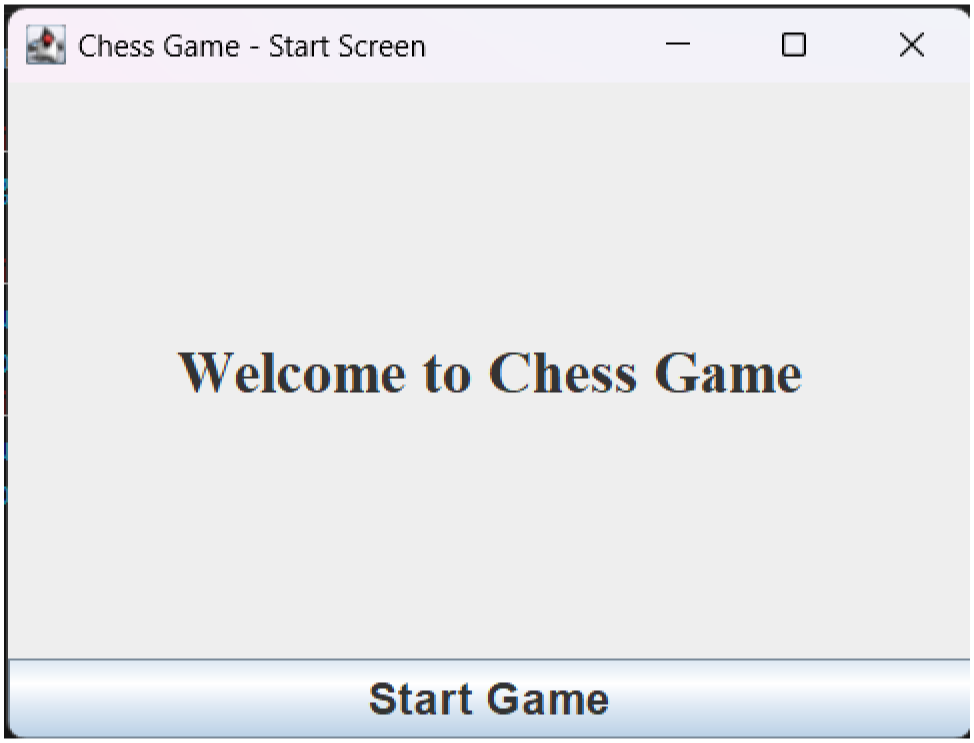
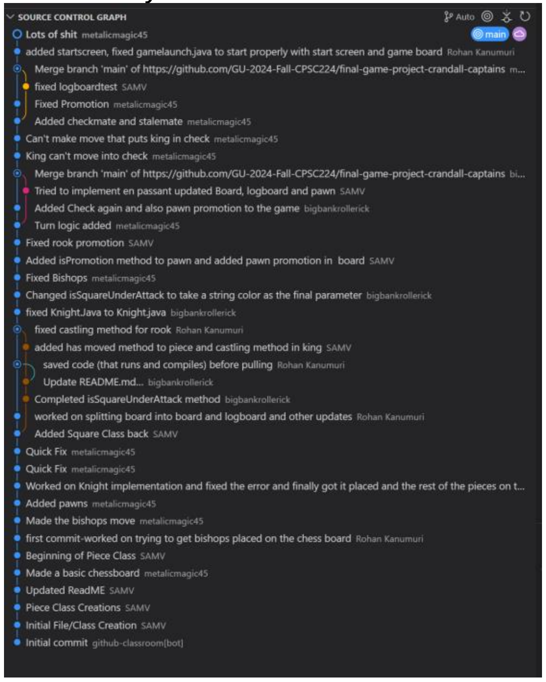


Figure 4: Start Screen (First Screen)

VI(a). Project Statistics

Codebase: Approximately 1000-1200 lines of Java code.
Classes Implemented: 10 major classes (LogBoard, Board, Square, and each Piece subclass).
Development Duration: 2 weeks
Commits: Over 30 commits with collaboration via GitHub.
Test Cases: Around 10 or more unit tests covering move validation, rule enforcement, and UI

functionality.



```
SOURCE CONTROL GRAPH
Lots of shit metalicmagic45
added startscreen, fixed gamelaunch.java to start properly with start screen and game board Rohan Kanumuri
  Merge branch 'main' of https://github.com/GU-2024-Fall-CPSC224/final-game-project-crandall-captains m...
  fixed logboardtest SAMV
  Fixed Promotion metalicmagic45
  Added checkmate and stalemate metalicmagic45
Can't make move that puts king in check metalicmagic45
King can't move into check metalicmagic45
  Merge branch 'main' of https://github.com/GU-2024-Fall-CPSC224/final-game-project-crandall-captains bi...
  Tried to implement en passant updated Board, logboard and pawn SAMV
  Added Check again and also pawn promotion to the game bigbankrollerick
  Turn logic added metalicmagic45
Fixed rook promotion SAMV
Added isPromotion method to pawn and added pawn promotion in  board SAMV
Fixed Bishops metalicmagic45
Changed isSquareUnderAttack to take a string color as the final parameter bigbankrollerick
fixed Knight.Java to Knight.java bigbankrollerick
  fixed castling method for rook Rohan Kanumuri
  added has moved method to piece and castling method in king SAMV
    saved code (that runs and compiles) before pulling Rohan Kanumuri
    Update README.md... bigbankrollerick
  Completed isSquareUnderAttack method bigbankrollerick
  worked on splitting board into board and logboard and other updates Rohan Kanumuri
  Added Square Class back SAMV
Quick Fix metalicmagic45
Quick Fix metalicmagic45
Worked on Knight implementation and fixed the error and finally got it placed and the rest of the pieces on t...
Added pawns metalicmagic45
Made the bishops move metalicmagic45
first commit-worked on trying to get bishops placed on the chess board Rohan Kanumuri
Beginning of Piece Class SAMV
Made a basic chessboard metalicmagic45
Updated ReadME SAMV
Piece Class Creations SAMV
Initial File/Class Creation SAMV
Initial commit github-classroom[bot]
```

https://github.com/GU-2024-Fall-CPSC224/final-game-project-crandall-captains

VII. Team Collaboration and Tools Used

For this project we mainly communicated through discord and imessage. We met 1 time before leaving for break to discuss basic stuff about our project. We also met after break to create the presentation and fix a couple last minute bugs. Most of the code for this project was done individually over break and we mainly just pulled and pushed to main. There were a couple branches but the majority of the work was done on main. We only did code reviews if people specifically asked for them because we trusted each other to get our parts done and ask for help if we needed it. Some of the most important lessons we learned about building software as a part of team is that delegation and time management are super important. It's also important that if you mess up you take accountability for your actions. We didn’t plan out our project; we would just text the group and see what still needed to be done and assign work based on what still needed to be completed. The main issue we had early on was that somebody forced pushed to main, deleting a decent chunk of code that had been completed. However, we were able to pull the deleted code from somebody who hadn’t pulled the force changes, so it was resolved easily.

VIII. Future Work

Some features that we would like to add for potential future releases are: the ability to offer a draw, better en passant functionality, more colors for the chessboard and maybe some different pieces with different abilities. We would also want to add an AI opponent so that single player games are possible. Online multiplayer could also be something that we could implement in the future. We would also like to make the game a little bit more visually pleasing as it's very barebone right now

IX. Glossary

Define technical terms used in the document.

Castling: A unique move involving the king and a rook; it allows the player to move the king two squares towards a rook while the rook jumps to the square next to the king, provided neither piece has moved before, there are no pieces between them, and the king isn't in, passing through, or moving into check.

En passant: A pawn capture that occurs when an opposing pawn moves two squares forward from its starting position and lands beside one of your pawns; your pawn can capture it as though it had only moved one square forward, but this must be done immediately on the next turn.

Pawn moves two squares: a pawn can move two squares forward on its first move instead of the usual one, provided there's no obstruction in its path.

X. References

Wikipedia Contributors. 2019. "Rules of Chess." Wikipedia. Wikimedia Foundation. November 19, 2019. https://en.wikipedia.org/wiki/Rules_of_chess.