CSC 413 Project Documentation

Summer 2024

Yuvraj Gupta

922933190

CSC 413-01

GitHub Repository Link

1. Introduction

1.1 Project Overview

The term project for CSC 413 focuses on the design and development of a unique Chess Game using JavaFX. This project demonstrates the application of object-oriented programming principles, game development techniques, and the use of JavaFX for building an interactive and visually appealing chess game.

The Chess Game developed in this project introduces several innovative elements that enhance the traditional chess experience. These include challenging chess variants and a sophisticated AI opponent, offering players a fresh and engaging gameplay experience.

1.2 Introduction of the Chess Game

The Chess Game reimagines the classic game of chess by incorporating new pieces and configurations, all while maintaining the core rules that make chess a timeless strategy game. Built using JavaFX, the game offers a visually appealing and smooth user interface, making it accessible to both casual players and chess enthusiasts.

Multiplayer Options: The game includes a multiplayer mode where players can compete against each other. This feature allows for direct competition between human players, bringing the social and strategic elements of chess to the forefront. The multiplayer mode supports real-time gameplay, making it easy for friends or family members to enjoy the game together.

Single-Player Against Bot: In addition to multiplayer, the game also offers a single-player mode where players can challenge an AI bot powered by Stockfish, one of the most powerful open-source chess engines. The bot is designed to simulate a real opponent, offering various levels of difficulty to

match the player's skill level. This ensures that the game remains challenging and rewarding, regardless of the player's experience.

Chess Variants: To further enrich the gameplay, several unique chess variants are included:

- Drook Variant: This variant introduces a new piece called the "Drook," which moves like a rook but is limited to moving only two squares at a time. This limitation forces players to adapt their strategies and rethink their approach to controlling the board.
- Amazon Piece: The Amazon is a special piece that combines the movements of both a queen and a knight. Its versatility makes it a powerful addition to the game, adding a new layer of strategy as players must account for its wide range of possible moves.
- Chess960: Also known as Fischer Random Chess, this variant randomizes the starting positions of the back-row pieces. By doing so, Chess960 encourages creative thinking from the very first move, as traditional opening strategies are no longer applicable.

These features collectively provide a new twist on the traditional game of chess, making it both intellectually stimulating and enjoyable. Whether you're playing against the Stockfish bot, competing in multiplayer mode, or exploring the new variants, the Chess Game offers a rich and engaging experience for all players.

2. Development Environment

- Version of Java Used: Java 21 SDK
- IDE Used: IntelliJ IDEA
- *Libraries Added:*

- JavaFX: After downloading the JavaFX SDK, it is important to configure your development environment (IDE) to include the JavaFX module path. This is done by setting the module path in your project settings and ensuring that JavaFX modules like `javafx.controls` and `javafx.fxml` are included in your runtime configuration.
- Stockfish: Integrating Stockfish requires downloading the engine and setting it up to communicate with your Java application. Typically, this involves executing Stockfish as a process and sending commands to it via standard input/output streams.

3. How to Build or Import Your Project

1. Clone the Repository:
    - Open your terminal or command prompt.
    - Navigate to the directory where you want to clone the repository.
    - Run the command: git clone [repository link].
2. Open the Project in IntelliJ IDEA:
    - Open IntelliJ IDEA.
    - Click on File > Open.
    - Navigate to the cloned repository directory and select it.
    - Click on Open to import the project.
3. Configure the Project:
    - Ensure that the JDK is set to Java 21.
    - If not set, go to File > Project Structure > Project and set the Project SDK to Java 21.
    - Click on Apply and then OK.

- Go to File -> Project Structure -> Modules.
- Select your module and click on the `Dependencies` tab.
- Click on the + icon, choose JARs or directories, and select the lib directory from your JavaFX SDK location
- Build the Project: Go to `Build` -> `Build Project` to compile all the source files and prepare for JAR packaging.

4. How to Run Your Project

1. Run from IntelliJ IDEA:
    - Open the Launcher class/ Jar run in Jar folder.
    - Edit Vm Options : --module-path /Path_to_lib_JavaFx --add-modules=javafx.controls,javafx.fxml,javafx.swing --add-exports javafx.base/com.sun.javafx.logging=ALL-UNNAMED
    - Click on the Run button or press Shift + F10.
2. Run from Terminal:
    - Open Jar folder.
    - Run command : java --module-path /Path_to_lib_JavaFx --add-modules=javafx.controls,javafx.fxml,javafx.swing --add-exports javafx.base/com.sun.javafx.logging=ALL-UNNAMED -jar csc413-tankgame-YuvrajGupta1808.jar
    - It should wore but didn't worked for me.

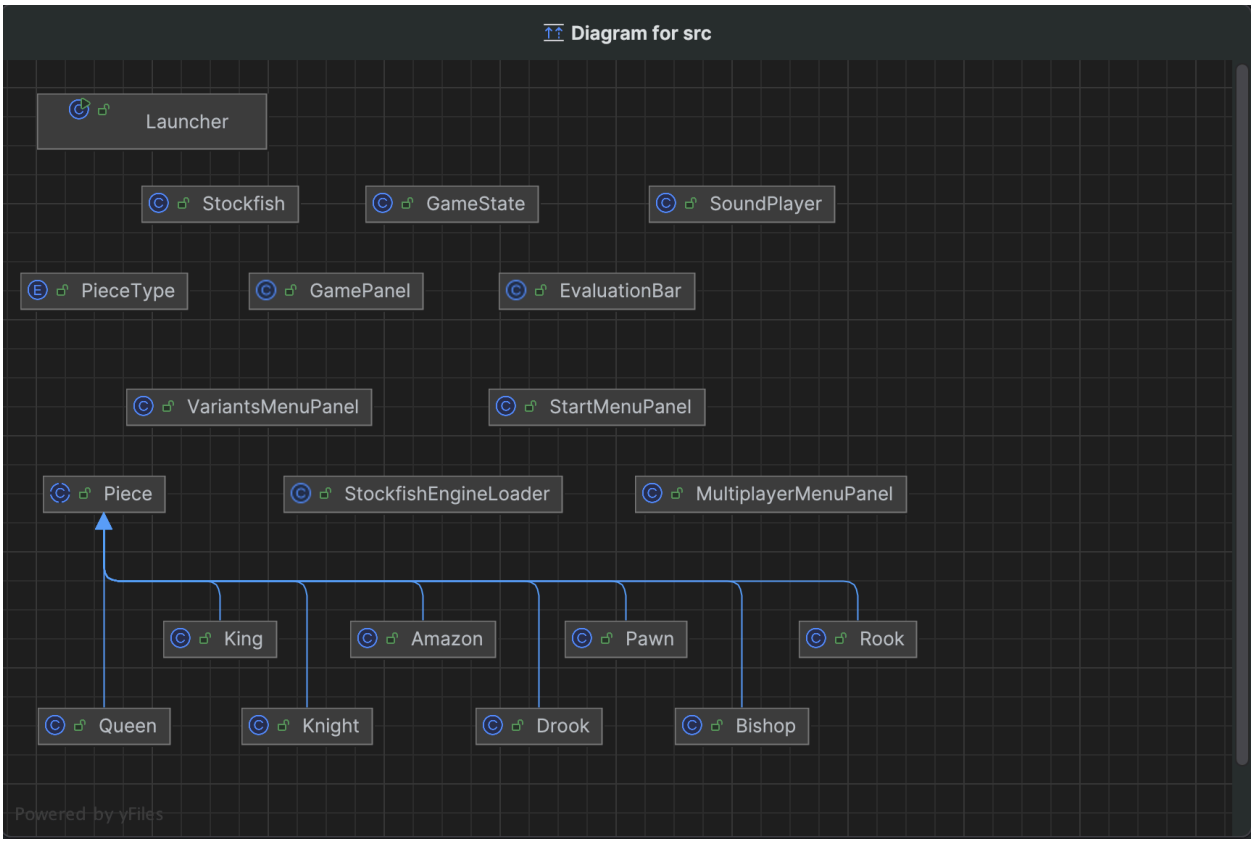
5. How to Play your game.

- Game Controls : The game is controlled entirely by mouse. Simply click on the pieces to select them, and click on the destination square to move them. All interactions, including selecting pieces, making moves, and navigating the game menus, are done using mouse controls.
- Game Rules : The rules of the Chess Game follow the standard chess rules, with additional variants like Drook, Amazon, and Chess960 providing unique challenges and strategic opportunities. The game also includes an AI opponent powered by Stockfish for single-player mode.

5. Assumptions Made

- The Board rotates every move in the variants.
- Player 1 is White and Player 2 is Black.
- Bots plays most accurate move.
- The pieces moves dynamically are animations.
- Gif on home page is a animation.

6. Class Diagram



7. Implementation Discussion

- Launcher: This class serves as the entry point for the application. It is responsible for initializing the game environment, loading resources, and launching the main game window. It essentially sets up everything required to start the game and directs the flow of control to the appropriate components.
- Stockfish: The `Stockfish` class handles the interaction with the Stockfish chess engine. It manages sending commands to the engine, receiving analysis and move suggestions, and integrating

the engine's output into the game. This class is crucial for enabling the AI opponent in the single-player mode

- GameState: The `GameState` class is responsible for tracking the current state of the chess game. This includes managing which player's turn it is, the positions of all pieces on the board, move history, and checking for game-ending conditions such as checkmate or stalemate. It acts as the central authority on the game's progress.
- SoundPlayer: The `SoundPlayer` class is designed to handle all audio-related tasks within the game. This includes playing sound effects for piece movements, captures, check alerts, and any other game events that require audio feedback. It enhances the player's experience by providing auditory cues that correspond to their actions in the game.
- PieceType: The `PieceType` enumeration defines the different types of chess pieces (King, Queen, Rook, Bishop, Knight, Pawn). This enum helps to categorize pieces, making it easier to handle piece-specific logic and behaviors in a clear and organized manner.
- GamePanel: The `GamePanel` class is the primary interface where the chessboard is displayed. It manages user interactions such as selecting pieces, making moves, and updating the visual representation of the board. It acts as the main stage where the game is played, rendering the board and pieces while processing user inputs.
- EvaluationBar: The `EvaluationBar` class provides a visual representation of the current evaluation of the board position, typically as calculated by the Stockfish engine. It shows how favorable the position is for either player, giving real-time feedback

on the game state. This can help players gauge their standing and inform their strategic decisions.

- VariantsMenuPanel: The `VariantsMenuPanel` class manages the user interface for selecting different chess variants before starting a game. It allows players to choose between standard chess and other variations like Drook, Amazon, or Chess960, setting the rules and initial conditions for the game based on the selected variant.
- StartMenuPanel: The `StartMenuPanel` class handles the initial game menu, where players can start a new game, access settings, view instructions, or exit the application. It provides the first point of interaction for users, guiding them to the desired mode of play or configuration options.
- Piece: The `Piece` class serves as the base class for all chess pieces, encapsulating common properties such as position, color, and movement logic. It provides the foundation for specific piece classes (e.g., King, Queen) by defining the essential behaviors and interactions that all pieces share, such as moving and capturing other pieces.
- Pieces: Each of these classes represents a specific type of chess piece, inheriting from the `Piece` class. They implement the unique movement patterns, interactions, and rules for their respective pieces. For example:
    - King: Implements the logic for moving one square in any direction and handling check/checkmate situations.
    - Queen: Combines the movement abilities of both the Rook and Bishop, making it the most powerful piece.

    - Knight: Handles the distinct “L” shaped movement and the ability to jump over other pieces.
    - Amazon: A special piece that combines the movements of the Queen and Knight, offering versatility in gameplay.
    - Drook: A variant piece that moves like a rook but is restricted to only two squares at a time.
    - Pawn: Manages the unique forward movement, diagonal captures, and promotion upon reaching the opposite side of the board.
    - Bishop: Allows diagonal movement across the board.
    - Rook: Handles horizontal and vertical movement, contributing to controlling ranks and files.
- StockfishEngineLoader: The `StockfishEngineLoader` class is responsible for loading and initializing the Stockfish engine, ensuring it is ready for use in the game. It manages the setup process, making sure that the engine can start analyzing positions and generating moves as needed for the AI opponent.
- MultiplayerMenuPanel: The `MultiplayerMenuPanel` class manages the interface for setting up multiplayer games. It allows players to configure multiplayer options, choose game variants, and connect with other players either locally or over a network. It facilitates the smooth initiation and management of multiplayer sessions.

8. Self-reflection on Development process during the term project

The development process of this Chess Game was both a challenging and enriching experience, filled with numerous learning opportunities and significant growth as a programmer and game developer. One of the most notable aspects of this project was the consistent failure to successfully implement a 3D version of the chess game, despite multiple attempts and a considerable amount of time and effort invested in the process.

Initially, the goal was to develop a 3D chess game that would offer an immersive and visually dynamic experience. I spent countless hours researching different 3D frameworks and engines, such as jMonkeyEngine and others, to see how they could be integrated with Java for the project. However, despite these efforts, I encountered numerous obstacles that prevented me from achieving the desired outcome.

The transition from 2D to 3D proved to be far more complex than anticipated. Challenges included managing 3D object rendering, handling camera angles, and creating a responsive and intuitive user interface in a 3D space. Additionally, the learning curve for 3D development tools and the complexities of working with 3D models and textures added significant pressure to the project timeline.

After several unsuccessful attempts and realizing the steep technical requirements and additional resources needed for 3D development, I made the strategic decision to focus on creating a polished and feature-rich 2D version using JavaFX. This decision was based on the realization that a well-executed 2D game with innovative features would be more impactful than a subpar 3D game. The hours spent on the 3D attempts were not wasted, though, as they provided invaluable lessons in project management, problem-solving, and the importance of recognizing and adapting to limitations.

I worked consistently on the project, often dedicating late nights and weekends to coding, debugging, and refining the game. The process was a test of patience and perseverance, particularly when facing bugs that were difficult to resolve or when attempting to implement the new chess variants like Drook and Amazon. The integration of the Stockfish engine also presented its own set of challenges, as it required careful handling of API interactions and performance optimization to ensure smooth gameplay.

Despite these challenges, the project ultimately succeeded in achieving its goals. The final product is a feature-rich 2D chess game with innovative variants and a strong AI opponent. This experience has significantly enhanced my understanding of game development, Java programming, and the importance of adaptability and resourcefulness in the face of obstacles.

9. Project Conclusion and Results

In conclusion, the Chess Game project was a deeply rewarding endeavor, despite the setbacks and challenges faced along the way. The initial ambition to create a 3D chess game was met with consistent failure, forcing a reevaluation of goals and a pivot to focus on a 2D implementation. This decision was crucial, as it allowed for the successful development of a polished and feature-rich 2D chess game that still offered players a unique and engaging experience.

The project demanded extensive hours of work, encompassing everything from learning new technologies, dealing with complex bugs, and iterating on design to ensure a high-quality final product. The inclusion of innovative chess variants and the integration of the Stockfish engine provided an additional layer of complexity, which was successfully navigated through persistent effort and a willingness to learn.

This project not only resulted in a functional and enjoyable chess game but also provided significant personal growth in terms of programming skills, problem-solving abilities, and project management. The lessons learned from this experience will undoubtedly be invaluable in future endeavors, both in game development and beyond.

The journey from the initial concept to the final product was marked by challenges, but each challenge served as an opportunity to learn and improve. The completed Chess Game stands as a testament to the power of perseverance, adaptability, and the willingness to push through difficulties to achieve a meaningful and successful outcome.