Dr. Babasaheb Ambedkar Marathwada University,
Chhatrapati Sambhajinagar

Seminar Report

“Chess Simulator”

Submitted by

Tejas Kayande

Sammed Burse

Abhishek Apar

A project report submitted in fulfillment for the
Bachelor of Computer Science (BCS)
Academic Year 2023-2024

under the guidance of

P. N. Naik

Submitted to the

S.B.E.S. College of Science, Chhatrapati Sambhajinagar 431001
Department of Computer Science & IT

Department of Computer Science and IT

Shri Saraswati Bhuvan College of Science,
Chhatrapati Sambhaji Nagar – 431001

CERTIFICATE

This is to certify that the candidates of B.Sc. (Computer Science General) III year who have satisfactorily completed the project entitled "Chess Simulator" for the partial fulfillment of Bachelors Degree from Dr. Babasahed Ambedkar Marathwada University, Chhatrapati Sambhajinagar for the academic year **2024-2025**

Submitted by

Tejas Kayande

Sammed Burse

Abhishek Apar

Project Guide
P. N. Naik

HOD
Dr. D. T. Patil

Examinar's Signature

ACKNOWLEDGMENT

We extend our deepest gratitude to the Department of Computer Science for providing us with the opportunity to undertake this project. This journey has been one of immense learning, growth, and technical exploration, and it would not have been possible without the unwavering support of many individuals. First and foremost, we express our sincere appreciation to our esteemed project guide, P. N. Naik Mam, for their invaluable guidance, insightful feedback, and constant encouragement throughout the course of this project. Their expertise and mentorship have been instrumental in shaping our understanding and refining our work. We would also like to thank our faculty members for their support and for fostering an environment that nurtures innovation and problem-solving. Their lessons and insights have significantly contributed to the successful completion of this project.

Finally, we extend our heartfelt gratitude to our peers, friends, and family members for their continuous motivation, patience, and encouragement during this endeavor. Their belief in our abilities has been a source of inspiration throughout this journey. This project is a culmination of collaborative efforts, and we deeply appreciate everyone who played a role in its completion.

Thanking You

Tejas Kayande [22056]
Sammed Burse [22016]
Abhishek Apar [22003]

INDEX

1 User Interview

1.1 Questions

The following section contains an interview with the client conductucted on [DATE]. The purpose of this interview was to gather insights into the application's required features, system performance, and specific chess-related functionalities to help guide the development process.

The following set of questions are general questions about the applications and the client expectations:

Question 1:
What do you picture when you say a Chess-Simulator Application?

Answer:
I picture a clean, easy-to-use chess board where I can play against another person or possibly an AI. The pieces should move smoothly, following the rules of chess. Maybe some animations or effects to make it feel polished. It should look modern but not too flashy.

Question 2:
What are some of the features that must be in the application?

Answer:
The application must have a fully functional chessboard where pieces move according to standard chess rules, including castling and en passant. It should support local multiplayer so that two people can play on the same device. The interface should be simple and easy to understand, ensuring smooth gameplay. Move validation is essential to prevent illegal moves, and there should be an option to undo moves in case of mistakes. Additionally, the game should display important status updates like check, checkmate, and draws to keep players informed.

Question 3:
What would be the average skill level of the user in the game of chess that would use this application?

Answer:
The average user would likely be a casual to intermediate chess player, someone who knows the basic rules but may not be highly competitive. They might play for fun, practice, or to improve their skills. Some users could be beginners who are still

learning the game, while a smaller portion might be more advanced players looking for a simple simulator to test moves and strategies.

Question 4:
What are the expectations with the User Interface?

Answer:
The user interface should be simple and familiar, like a standard Windows application with a clean menu bar for essential options. The chessboard should be clear and easy to read, with well-defined pieces and smooth movement. There shouldn't be unnecessary animations or flashy effects—just a straightforward, functional design that focuses on usability. Everything should be easy to access without clutter or complicated navigation.

Question 5:
You talked about the addition of an Artificial Engine that plays against a human player, what are your expectations with this feature?

Answer:
The AI should be able to play at a reasonable level, offering a fair challenge to casual and intermediate players. It doesn't need to be overly advanced but should make legal and logical moves without obvious blunders. Ideally, there could be different difficulty levels so users can choose how challenging they want the game to be. The AI should play smoothly without long delays, making it feel responsive and natural.

Question 6:
Would you like any additional customization options?

Answer:
Basic customization would be nice, like the ability to choose between a few different board and piece styles. Options to toggle sound effects or adjust the window size would also be useful. However, customization doesn't need to be too complex—just enough to make the experience a little more personal while keeping the interface simple.

Question 7:
Would you like the ability to save and load games?

Answer:
Yes, it would be useful to have a way to save a game and continue it later. This would allow players to take breaks without losing progress. A simple save-and-load system would be enough—nothing too complex, just an option to resume where they left off.

The following set of questions are designed to understand the system requirements and performace expectations:

Question 1:
What is/are the target platform(s) that the user will be using the application on?

Answer:
The application will be used on Windows PCs, so it should run smoothly on different versions of Windows, preferably from Windows 10 onward. It should work well on both laptops and desktops without requiring high-end hardware.

Question 2:
Should the application support fullscreen mode?

Answer:
It's not necessary, but it would be a nice option. The game should primarily run in a windowed mode, but if fullscreen support is added, it should maintain clear visuals without stretching or distorting the chessboard and pieces.

Question 3:
What is of high priority: performance or appealing visuals?

Answer:
Performance is more important than appealing visuals. The game should run smoothly on a wide range of Windows PCs without lag or high resource usage. While a clean and readable design is necessary, unnecessary animations or flashy effects should be avoided if they impact performance. The priority is ensuring smooth gameplay, fast responsiveness, and minimal system resource usage.

Question 4:
Should the application have sound effects or music?

Answer:
Sound effects for piece movement, captures, and check notifications would be a nice addition, but they should be subtle and not distracting. There's no need for background music since chess is a game that requires focus. An option to toggle sound on or off would be useful for users who prefer a quiet experience. The sounds should be lightweight and not affect performance in any way.

Question 5:
Should the game include a move history or notation feature?

Answer:

Yes, a simple move history would be useful so players can review past moves. It doesn't need to be overly complex—just a list of moves in standard chess notation. This can help players track their game progress and learn from their mistakes. An option to export the move history as a text file or PGN format would be a nice addition but isn't essential.

The following set of questions are application specific (here chess specific) and more on the technical side:

Question 1:
What are some of the niche features would you like in the application, for instance the ability to load position from FEN[1] strings.

Answer:
A feature to load positions from FEN strings would be very useful, especially for analyzing specific board states or recreating famous games. Another niche feature could be a simple evaluation tool that gives basic feedback on the position, helping players understand their standing. A move suggestion or hint system could also be helpful for beginners. Additionally, the ability to step through move history or play through famous games would add extra value without making the application too complex.

Question 2:
Should the game support different chess variants, or just standard chess?

Answer:
For now, the focus should be on standard chess with all its rules properly implemented. Supporting chess variants like Chess960[2] or Three-Check[3] could be interesting in the future, but they are not a priority. The main goal is to ensure a smooth and accurate classic chess experience before considering additional modes.

Question 3:
How should illegal moves be handled in the application?

Answer:
Illegal moves should simply be impossible to make. If a player tries to move a piece incorrectly, it should remain in its original position without any error messages or alerts. This ensures a seamless and intuitive experience where players naturally learn the rules by only being able to make valid moves.

Question 4:
How should the application handle check, checkmate, and stalemate conditions?

Answer:
The game should automatically recognize when a player is in check, but without displaying any pop-ups or alerts—just enforcing the rule that the king must move out of check. When a checkmate or stalemate occurs, the game should end immediately with a clear indication of the result, such as displaying "Checkmate" or "Draw" on the screen. The user should then have the option to restart or exit the game.

Question 5:
What are your expectations for the AI player's strength and behavior?

Answer:
The AI should be smart enough to make reasonable moves without feeling repetitive or too easy to exploit. It doesn't have to be highly advanced, but it should provide a fair challenge, especially for casual players. Having different difficulty levels would be a good addition, allowing players to choose how tough they want the game to be. The AI should also play at a good pace, making its moves quickly to keep the game flowing smoothly.

Question 6:
Should the AI have a specific playstyle or just make the best possible moves?

Answer:
The AI should focus on making the best possible moves based on its difficulty level rather than following a specific playstyle. However, it would be interesting if higher difficulty levels played more strategically, prioritizing control of the center, piece activity, or certain tactical patterns. At lower levels, the AI could make simpler moves to give beginners a fair chance while still following basic chess principles.

Question 7:
How should the application handle threefold repetition[4]?

Answer:
The game should automatically detect when a position has repeated three times and declare a draw if the rule applies. The rules should be enforced without requiring player input, ensuring accurate chess mechanics.

Question 8:
Should the game allow takebacks or move rewinds?

Answer:
Yes, there should be an option to undo the last move, especially in local multiplayer or against the AI. This can help players learn and experiment with different strategies.

However, takebacks might not be allowed in competitive settings if a future online mode is considered.

Question 9:
How should pawn promotion be handled?

Answer:
By default, a pawn should automatically promote to a queen when it reaches the eighth rank. However, if the player holds a specific key while moving the pawn, they should be able to select a different piece. For example, holding the '2' key promotes to a rook, '3' to a bishop, and '4' to a knight. This method keeps the gameplay smooth without interrupting the flow while still allowing advanced players to choose their preferred promotion piece efficiently.

Question 10:
Should the application display possible legal moves for a selected piece, highlight the selected piece and highlight the latest move mode on the board?

Answer:
Yes, the application should highlight the selected piece so the player knows which piece they are moving. It should also highlight the last move made on the board, helping players keep track of the game flow. However, displaying possible legal moves should be optional—some players may prefer to have it for guidance, while others might want a more traditional experience without hints. A simple toggle in the menu would allow players to enable or disable this feature based on their preference.

Question 11:
How should the game handle resignation and draw offers?

Answer:
There should be an option for players to resign if they feel the game is lost. Additionally, players should be able to offer a draw, which the opponent can accept or decline.

Question 12:
Should the application support board flipping for Black's perspective?

Answer:
Yes, when playing as Black, the board should automatically flip so that the pieces are oriented correctly. There should also be an option to toggle this manually in case a player prefers to view the board from White’s perspective regardless of color.

1.2 Summary of the User Interview

The client envisions a clean and user-friendly chess simulator where users can play against either another human or an AI opponent. The application should prioritize smooth gameplay with basic animations but avoid unnecessary flashy effects. A modern yet simple interface, similar to standard Windows applications, is expected.

Core features must include a fully functional chessboard with standard chess rules, including special moves like castling and en passant. The interface should be intuitive, and the game should ensure move validation while allowing undo options. The target users are expected to be casual to intermediate chess players, with some beginners also using the application.

The UI should be simple and familiar, with a Windows-style menu for essential options. The chessboard must be clear, with smooth piece movements. A few customization options, such as board themes and piece styles, should be available, but extensive customization is unnecessary. Players should have an option to save and load games.

The application is targeted for Windows PCs, ensuring compatibility with Windows 10 and above, and should function well on both desktops and laptops. Performance is a priority over visuals, meaning the application should run smoothly without consuming excessive resources. Fullscreen mode is optional but should be implemented correctly without distortion. Subtle sound effects for piece movement and check notifications are preferred, with an option to toggle sound.

The application should support loading chess positions using FEN strings for analysis and study. While only standard chess is required initially, future updates could explore chess variants. Illegal moves should be impossible to execute without error messages. The game must automatically detect check, checkmate, stalemate, threefold repetition, and draws by insufficient material. Move history should be recorded in standard chess notation with an option to export.

The AI should offer a fair challenge to casual players, with difficulty levels available. It should play natural, logical moves without excessive delays. While the AI does not need a specific playstyle, higher levels could focus on strategic principles like center control and tactical patterns.

Pawn promotion should default to a queen, but holding a specific key (e.g., '2' for rook, '3' for bishop, '4' for knight) should allow alternative promotions. Players should have the option to resign or offer a draw, with the AI accepting draws when the position warrants it. Board flipping should be enabled for Black's perspective, with an option to toggle manually.

Selected pieces should be highlighted, and the last move should be visibly marked. An optional feature should allow players to see legal moves for a selected piece.

The Chess Simulator aims to be a lightweight yet robust application, focusing on an intuitive user experience, correct chess mechanics, and smooth AI interaction. While extra features such as move history and game-saving are valuable additions, the core development should emphasize gameplay accuracy and responsiveness.

2 Problem Defination Document (PDD)

2.1 Problem Statement

Chess is a timeless game of strategy and intellect, enjoyed by players of all skill levels. With the increasing shift towards digital platforms, there is a growing need for a lightweight yet fully functional chess simulator that provides an authentic chess-playing experience. Many existing applications either focus heavily on online multiplayer or introduce complex interfaces that may not appeal to casual players. This proposal outlines the need for a streamlined, intuitive, and efficient chess simulator designed to provide an engaging and accessible experience for users.

Many chess enthusiasts, learners, and casual players seek a reliable, lightweight, and functional chess application for Windows. Existing solutions often come with unnecessary complexity, online dependencies, or excessive resource consumption. This project aims to develop a Chess Simulator that provides a smooth, offline chess-playing experience with an intuitive interface, correct rule enforcement, and optional AI opponent.

Most available chess applications are either too simplistic, lacking core features like move validation and game-saving, or overly complex, making them difficult for casual players to navigate. Additionally, some chess programs prioritize online play, leaving users who prefer offline play with limited options. There is a need for a balanced chess application that offers a complete game experience while maintaining simplicity and ease of use.

2.2 Proposed Solution

The proposed Chess Simulator will be a standalone application designed for a smooth and user-friendly experience. It will provide:

- A standard chessboard interface that follows official chess rules, including special moves like castling and en passant.
- A local multiplayer mode where two players can take turns on the same device.
- An AI opponent with adjustable difficulty, allowing users to practice and improve their skills.
- A move validation system ensuring that only legal moves are possible, preventing errors.

- Game-saving and loading capabilities to allow users to resume previous matches.

- A clean and intuitive interface with essential visual indicators for moves, checks, and game results.

- A minimalist design approach that prioritizes functionality over unnecessary visual complexity.

2.3 Expected Impact

By developing a lightweight yet feature-rich chess simulator, this project aims to bridge the gap between overly simple and overly complex chess applications. It will provide a well-balanced platform for casual and intermediate players who want an offline, distraction-free chess experience. The simulator will support both learning and competitive play while maintaining a responsive and efficient interface.

This proposal sets the foundation for the Chess Simulator project, ensuring that the focus remains on usability, accessibility, and an authentic chess experience.

3 Software Requirement Specification (SRS) Documentation

The purpose of this document is to analyse all the software, hadware, functional and non-functional requirements of the application.

3.1 System Introduction

3.1.1 Application Scope

Application focuses on creating a user friendly chess application that follows the UI standards of other leading chess applications. The focus is also on creating a highly performant and memory efficent application for the low end computers. The application must run on Windows 10 Operating System and above. The application is a Windows-based application that allows users to play chess against a human opponent or AI. The goal is to provide an intuitive and smooth gameplay experience while adhering to official chess rules.

3.1.2 Intendent Audiance

The application is aimed at avid chess players that enjoy casual games of chess, and like to study opening theories and practice their stratergies by playing with stranger and friends. This application also involves an AI engine to further enhance the studies, which also makes for a reletively strong openent to practice opening theories and tactics.

3.1.3 Intended Use

The application is well suited for chess player that are familiar with some basic chess terminologies and gameplay. Its intended use involves analysis of games and position evaluation with the help of built-in AI engine.

3.2 System Requirements

3.2.1 Hardware Requirement

- Input Device – Mouse and Keyboard are required to interact with the application.
- Graphics – Integrated Graphics (Intel HD Graphics 4000 or equivalent)

3.2.2 Software Requirement

- Operating System – Windows 10 (64-bit) or above

3.2.4 Functional Requirements

Core Machanics

- Rendering – The application should display the 8x8 Chess Board on the screen on a window that is interactive and responds with very low latency.

- Piece Movement – The user should be able to move the Chess Pieces on the board on any square.

- Move Validation – The moves made by the user should be validated by checking them against the traditional rules of chess, and only allow the moves that are legal.

- Special Moves – The application must supports special moves like castling and en-passant.

- Game State Detection – The application should keep track of the position on the board and respect the traditional conclusions of a chess match, i.e, Checks, Checkmate and Stalemate.

User Interaction & Interface

- Piece Selection and Movement – The user should be to select the pieces on the board and then drag them to the square to move the piece to that square.

- Move Highlighting – The latest move on the board and valid moves for the currently selected piece should be visually indicated. Although the application should also have a way to turn this feature off in case any user finds in annoying.

- Move History Tracking – The application must maintain and display a history of moves made during the game. It should also have a feature to continue the game from the history into different moves.

- Visual and Audio Effects – The application must play sounds when a move is made and it should also have basic animation for dragging of piece for a smoother user experience.

Game Modes & Features

- Player-vs-Player Mode – The application must allow for 2 players to play each other on the same computer system where the turn is automatically altered after a move is made
- AI Opponent – The application should include an option to play againts the built in AI engine or any external engines like stockfist
- Multiplayer Support (optional)– The game should allow user to start a match online with other player and friends.

3.2.5 Non-Functional Requirements

Performance & Efficiency

- Smooth Graphics Rendering – The chessboard and pieces must be rendered efficiently using without noticeable lag or stuttering.
- Low Resource Usage – The application should run with minimal CPU and memory consumption to ensure smooth performance on mid-range systems
- Fast Move Processing – Moves should be validated and executed within milliseconds to maintain a seamless user experience.

Usability & Accessibility

- User-Friendly Interface – The game UI must be intuitive, allowing players to easily select and move pieces.
- Clear Visual Cues – Highlighting of legal moves, selected pieces, and check/checkmate conditions must be easily distinguishable.
- Configurable Settings – Players should be able to adjust board themes, piece styles, and other UI preferences.
- Keyboard Shortcuts – Key bindings for actions like undo, redo, and board flip should be available.

4 Software Design Documentation (SDD)

The purpose of this document is to provide a detailed design description of the application. This document defines the system architecture, key modules, data structures, algorithms, and UI design. It serves as a guide for developers to ensure consistency and maintainability.

4.1 Design Goals

- Modular Architecture – The system should be designed in seperate modules (e.g., Platform Layer, Game Logic, Chess Representation)
- Platform Layer Abstraction – The Operating Specific operations should be abstracted away.
- Performance Optimization – The game should run efficiently with minimal resource usage.
- User Interface – The User Interface should be fairly straightforward and intuitive

4.1.1 User Interface Design

- Main Chessboard Screen: Displays board, pieces, and move highlights. It will be almost the full size as the window except for the parts of menu
- Menu: The menu will be a Windows style menu at the top of the window just below the title bar.
- Indicators: The information such as checks, checkmates and draws will be displayed on the board, and the application will not have a dedicated onscreen console to display messages
- Sound Effects: The application will have minimal yet effective sound feedback for actions like piece movement, captures, check notifications, and game-end alerts (checkmate, stalemate, draw).
- Toggleable Sound Option: Users will have the ability to enable or disable sound effects from the menu settings.
- Subtle Audio Design: The sounds will be lightweight and non-intrusive

4.2 System Architecture

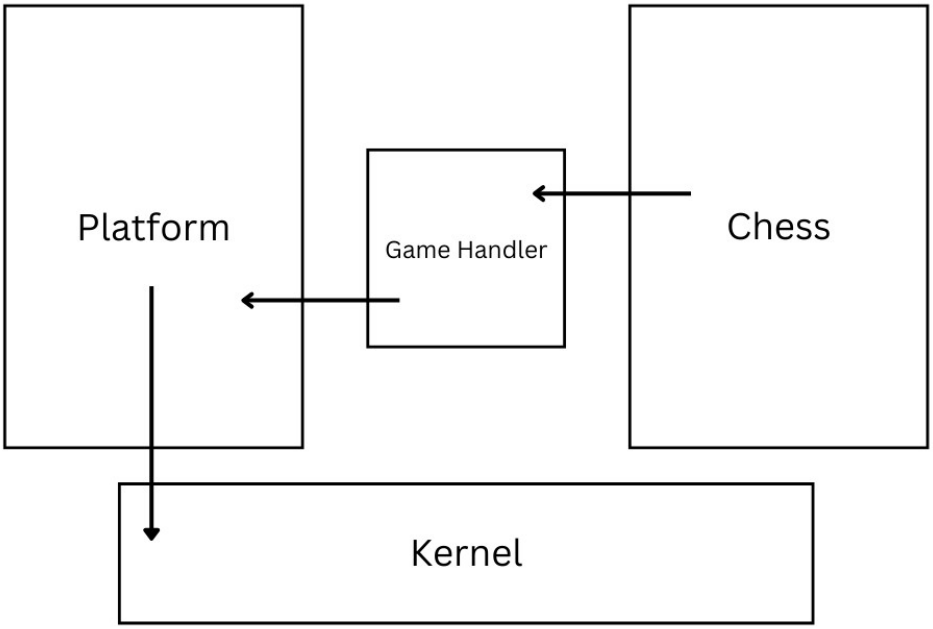

fig: Architectural digram of the application

The application will be divided into 3 layer:

4.2.1 Platform Layer

This layer provides all the Operating System-specific utilities, such as:

- Creating a window to display the game.
- Handling keyboard and mouse inputs.
- Providing functions to draw pixels on the screen.

This is the only layer that interacts with the kernel of the Operating System and makes OS calls. This approach helps maintain a clean codebase and reduces the number of kernel calls by batching them efficiently.

4.2.2 Chess Representation

This layer is responsible for handling the core chess logic, including:

- Board Representation: Manages the 8x8 chessboard state.

- Move Validation: Ensures that all moves adhere to standard chess rules.
- Game Rules Enforcement: Implements check, checkmate, stalemate, and special moves (castling, en passant, and pawn promotion).
- Move History & Tracking: Stores previous moves and allows for undo/redo functionality.

4.2.3 Game Handler

- The Game Handler is the bridge between the Platform and Chess layers, managing game flow and interactions. Key responsibilities include:
- Turn Management: Alternates moves between players.
- User Input Processing: Captures and translates player actions into game commands.
- Rendering Coordination: Requests the Platform layer to draw the board and pieces based on game state updates.
- Game State Management: Determines whether the game is in progress, paused, or has ended.
- UI Updates: Displays status messages (e.g., check, checkmate) and maintains a smooth user experience.

5 Technical Specification Document (TSD)

This document provides a detailed technical specification for the Chess Simulator application. It defines the core system architecture, data structures, algorithms, and key implementation details used in the development. The purpose is to establish a clear reference for developers, testers, and maintainers to ensure consistency, scalability, and performance throughout the software lifecycle.

Some developing choices and tools are as follows:

- Programming Language – C++ (without many object oriented features)
- Rendering API – Microsoft DirectX2D
- Compiler – Microsoft Visual Studio 2022 (64-bit)
- Platform Layer API – Microsoft Win32 API
- Sound – Microsoft Win32 API (PlaySound)
- Debugger – Microsoft Visual Debugger

5.1 Key Algorithms and Data Structures

5.1.1 BitBoard

A BitBoard is a 64-bit data structure used to represent the chessboard, where each bit corresponds to a square. This technique allows for efficient board representation and fast operations using bitwise manipulation. Each bitboard is a 64-bit integer where: ‘1’ represents an occupied square and ‘0’ represents an empty square.[5]

The decision to use BitBoard instead of an array was made considering the fact that array traversing is reletively slow compared to interger bit shifting, which makes a significant difference when calculating legal moves.

In this application a BitBoard is just a typedef 64-bit integer (uint64_t).

```
typedef uint8_t BitBoard;
```

A total of 12 BitBoard (6 per player) will be used to store the complete position on the board. The Data-Structure to store the board is given as follows:

```
struct Board {

    BitBoard wPawn, wKnight, wBishop, wRook, wQueen, wKing;
    BitBoard bPawn, bKnight, bBishop, bRook, bQueen, bKing;
```

```
    BitBoard wOccupied, bOccupied;

    struct PlayerInfo {
        bool king_moved;
        bool krook_moved;
        bool qrook_moved;
    };

    PlayerInfo white;
    PlayerInfo black;

    Player turn;
};
```

Here the PlayerInfo datastructure is to hold castling related information.

5.1.2 Square and Piece Representation

For convenience purposes we have defined stuctures that hold the location of a give Square on the board, and the type and color of a give piece. These structures are as follows:

```
struct Square {

    int rank;
    int file;

    inline bool operator==(const Square &square) const {
        return (rank == square.rank && file == square.file);
    }

    inline bool operator!=(const Square &square) const {
        return (rank != square.rank || file != square.file);
    }
};

struct Piece {

    enum Type {
        NO_PIECE,
```

```
        PAWN,
        KNIGHT,
        BISHOP,
        ROOK,
        QUEEN,
        KING
    } type;

    enum Color {
        NO_COLOR,
        WHITE,
        BLACK
    } color;

    inline bool operator==(const Piece& piece) const {
        return (type == piece.type && color == piece.color);
    }

    inline bool operator!=(const Piece& piece) const {
        return (type != piece.type || color != piece.color);
    }
};
```

6 Software Deployment and Compilation

This section outlines the process of compiling the software from source, packaging it for distribution, and deploying it on target Windows systems. The Chess Simulator application is designed for Windows PCs and should be easy to install and run with minimal system requirements.

6.1 Deployment Method

Once compiled, the Chess Simulator application is packaged into an installer or a portable executable that can be easily distributed. The deployment methods include:

- Portable Executable (.exe): The application can be packaged as a standalone executable that does not require installation.
- Installer-Based Deployment: A setup wizard can be provided for users who prefer a guided installation process. The installer ensures that all required dependencies are available on the system.
- Versioning and Updates: Future updates may be distributed as patches or new versions with improved functionality.

6.2 System Requirement for Deployment

To ensure smooth installation and execution, the following system requirements must be met:

- Operating System: Windows 10 or later (64-bit recommended)
- Processor: 64-bit processor with at least 2 GHz clock speed
- Memory: Minimum 4GB RAM (8GB recommended for optimal performance)
- Storage: At least 100MB of free disk space
- Graphics: Integrated or dedicated GPU supporting DirectX 11 or later

6.3 Future Deployment Considerations

Future deployment strategies may include:

- Automatic Updates: Implementing an update system that notifies users when a

new version is available.

- Cloud-Based Storage: Storing saved games or preferences in the cloud for cross-device accessibility.

- Cross-Platform Support: While currently designed for Windows, future versions could support other operating systems if demand arises.

References

[1] Chess.com, “Forsyth-Edwards Notation”, https://www.chess.com/terms/fen-chess
[2] Wikipedia, “Chess960”, https://en.wikipedia.org/wiki/Chess960
[3] Wikipedia, “Three-check chess”, https://en.wikipedia.org/wiki/Three-check_chess
[4] Chess.com, “Threefold Repetition”, https://www.chess.com/terms/threefold-repetition-chess
[5] Chess Programming Wiki, “Bitboards”, https://www.chessprogramming.org/Bitboards