SP-14-Red: Chess Game with AI Final Report

Section 4850-02

Spring 2024

Justin Gonzalez

Andrew Nguyen

Phillip Ngo

Professor Sharon Perry

4/27/2024

https://github.com/SP-14-Red/Chess-Game-AI

Project Components: 11

Lines of Code: 688

Website Link: https://sp-14-red.github.io/ChessGame/

1.0 Introduction

1.1 Purpose

This report covers the complete journey of creating an Artificial Intelligence that plays chess with a human player. Details of the final report include the requirements and design of the chessboard and chess AI, and how the project was expanded beyond the prototype phase.

1.2 Project Goals

The SP-14 RED Chess AI project aims to develop a sophisticated chess AI capable of challenging human players. The AI will be adjustable in complexity, matching the skill level

of the player, thus serving both educational and entertainment purposes. Players will be given the option to select between another human opponent or a chess AI will utilize the core mechanics to provide a baseline for players but will be able to adjust in level of skill according to the player's own skill level.

1.3 Overview of Chess

Chess is a strategic board game that dates to the Gupta Empire, where it was introduced as "chaturanga", which roughly translates to the "four branches of the military." These branches were represented by the Pawn, Rook, Knight, and Bishop. As time passed, the game spread to Persia, where it became known as Shatranj. Eventually, the Persian empire was conquered by Arabs, and Muslims spread the game across countries until it reached Europe via Spain and Italy. At this point, European leaders introduced the king and queen to the game, which lead to the basis of the chess we play today.

The game of chess was designed for two players, who sit across from each other and control their own "army" where each piece has a specific set of moves that they can make each turn. The goal for each player is to checkmate, or take out, the opponent's king, without losing their own. A significant part of chess is forming strategies and looking for openings, as the slightest of mistakes can be a huge advantage for an experienced player.

1.4 Definitions and Acronyms

- **Minimax**: A recursive algorithm designed to choose the next "best" move by selecting the option with the minimized loss. This option is based on the value that updates based on the state of the game.
- **Alpha-beta pruning:** An optimized method to the minimax algorithm that allows the program to ignore certain paths generated by the minimax algorithm.
- **AI:** Artificial Intelligence
- **GUI:** Graphical User Interface
- **Alpha-beta pruning:** an optimization technique for the Minimax algorithm that ignores fewer promising paths to improve decision-making speed.
- **Pygame:** A free and open-source cross-platform library for the development of multimedia applications like video games using Python.
- **Python:** The programming language chosen for the development of the chess game and its AI.

2.0 User Constraints

- Users must have access to a modern computer with a Windows operating system.
- Users must have a basic understanding of chess at the minimum.

3.0 Functional Requirements

3.1 Game Rules

- The game is expected to correctly implement all normal chess game rules.
- Rules such as castling and en-passant are coded in and conditional requirements are enforced.
- The rule of pawn promotion will be added, and pawns will automatically be promoted to a queen piece.

3.2 Chess Board

- A fully functional chess board featuring the corresponding grid pattern and default chess pieces.
- Chess board will have row and column titles enabled.
- Different chess board themes will be added to provide different experiences.

3.3 Skill Level Configuration

- Users should be able to choose the level of difficulty for the AI player (beginner, intermediate, expert).
- AI will analyze a user's skill level and adjust accordingly based on input data (if manageable).

3.4 Chess Piece Movement and Moves

- Show valid paths for a specific chess piece when clicked on.
- Appropriate checks are made to ensure only valid moves are applied (king is in check, castling requirements, etc.)
- Update the board according to the moves chosen by the player and AI.

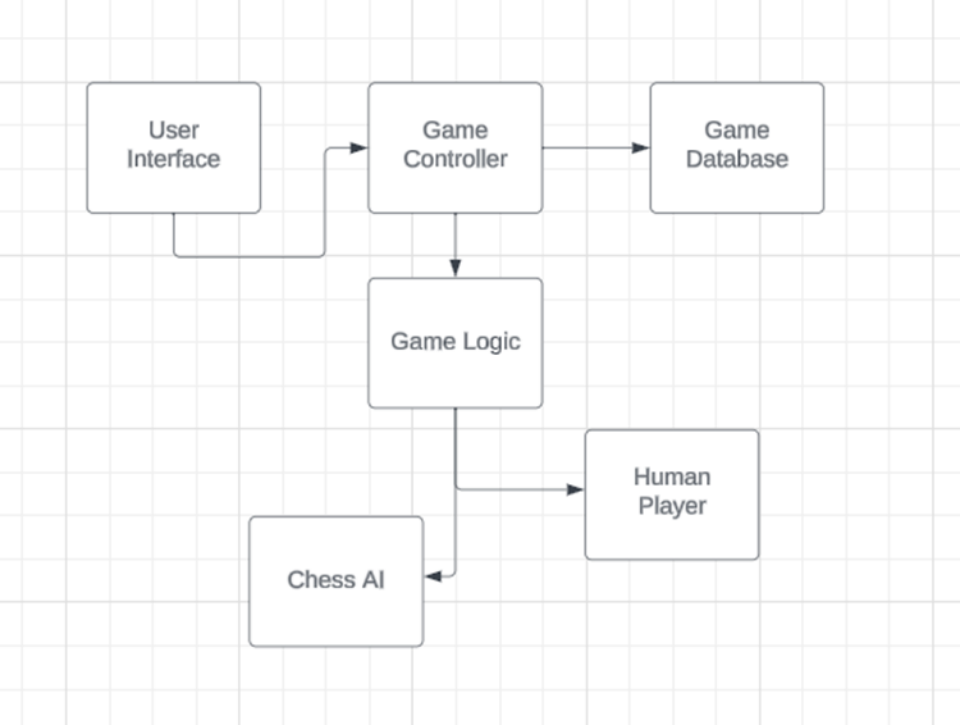
3.5 Opposing AI

- Implement algorithms such as Minimax and greedy algorithm (prototype).
- Implement alpha-beta pruning to further increase the skill level of the AI and plan for future moves.
- Adjust skill based on player's performance (if manageable).

3.6 Restart and Exit Functions

- The player should be allowed to either exit the game or restart the game at any given time.
- Restart will be bound to a keyboard input for ease of access.

3.7 Game Structure UML Diagram



4.0 Nonfunctional Requirements

4.1 Capacity

- The chess game should load on any device quickly without crashes or slow speeds.
- The AI should react to moves accordingly and in a quick manner.
- The AI should be able to hold a certain weight to its moves and start to plan for future moves and predict what the human opponent will do.

4.2 Capacity

- Ensure the program can allow for users to play versus another player or versus AI
- The program will store game history for statistic tracking.

4.3 Usability

- The UI should allow inexperienced players to gain a better understanding of how to navigate through the options/features.

5.0 External Interface Requirements

5.1 User Interface Requirements

The UI must be easy-to-understand and user-friendly while running without issue on various devices. The design must be simple to navigate while also remaining visually appealing to users. The UI will include highlighted paths on each piece to offer a sort of tutorial to those unfamiliar with the game.

6.0 Design

6.1 Purpose

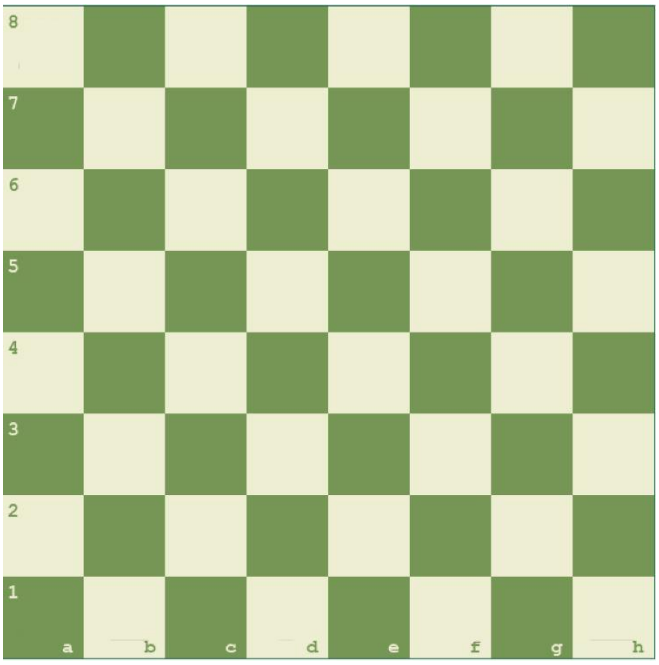
This section of the document provides an in-depth design overview of the Chess Game with AI System. This paper shows the components and methods used to design and compile a Python chess game with artificial intelligence capabilities. The paper highlights the system architecture, the design components and constraints, and the guidelines and goals for the project.

6.2 General Constraints

The design of the project will be limited by the Python programming language as well as the libraries within the language such as the Pygame library. Since the entire project is coded using Python, the constraints will be based on the capabilities and limitations of the coding language itself. External code and resources being integrated into the project will have to be originally in Python or converted to Python.

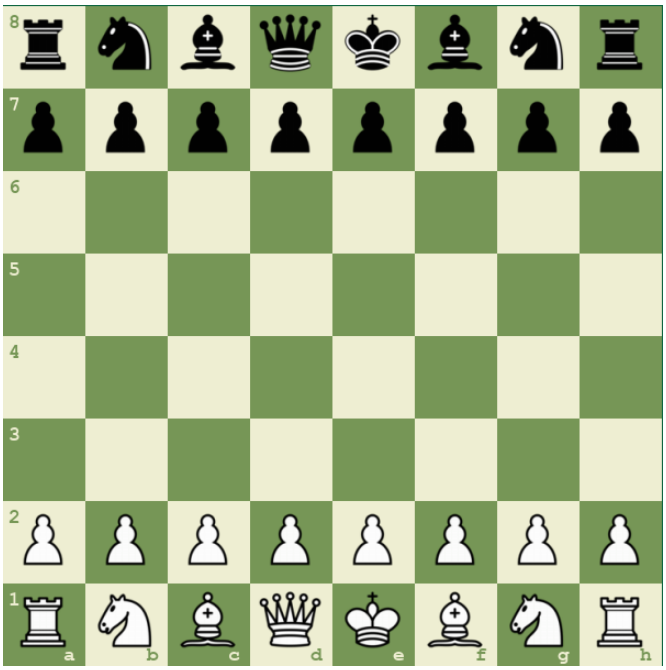
6.3 UI Design

The UI is represented by a standard 8x8 chessboard, with alternating squares of different shades of green. This color was chosen to offset the white and black pieces and minimize strain on the user's vision. The UI also includes alphanumeric codes for each piece of the board, effectively assigning each cell its own unique id to guide users with positioning of pieces. Below is the described design for the Chess board UI.



After the board is drawn, it is then adorned with the complete set of chess pieces in accordance with standard chess rules: 16 white chess pieces and 16 black chess pieces, totaling 32 pieces. As shown in the figure below, the top side of the chess board will

display the black pieces and the bottom side of the chess board will display the white pieces.



6.4 System Architecture

- Built with Pygame, the software provides an interactive and striking environment for players to view and play with the chessboard.
- The backend logic, including the AI, is scripted in Python, orchestrating the game flow, enforcing rules, and executing moves.

6.5 System Design

6.5.1 Chess Logic and Rules Engine

- General rules of chess
- Set of moves for each chess piece.
- Displaying/Validating legal chess moves
- Understanding game states and outcomes (current player's turn, check, checkmate)

6.5.2 AI Engine

The AI decision-making was conducted using the minimax algorithm alongside a greedy algorithm and alpha-beta pruning. This program allows the AI to make chess moves that have meaningful impact on the game instead of randomly generated moves.

6.6 System Guidelines

- The design consists of the standard 8x8 grid with a checkerboard pattern.
- The color scheme will include multiple options to provide user flexibility.
- The rules and capabilities of the program are confined to the standard rules of chess.
- Legal moves and previously made moves are highlighted and indicated appropriately.
- The option to reset the board/restart the game and change the theme will be highlighted along the borders.

7.0 Development

7.1 Project Overview

The creation of the SP-14 Red Chess AI project came due be due to the team's curiosity and appreciation for artificial intelligence and a shared passion for online gaming. The primary goal was to create a tool that not only served as a challenging opponent in chess but also aided in the educational development of its users by having multiple skill levels without the need for a human opponent. The objectives were clearly defined to develop a decent chess AI that can simulate human-like strategic thinking, providing both entertainment and learning opportunities for players of various skill levels.

The team consisted of three members: Justin Gonzalez, Andrew Nguyen, and Phillip Ngo, each bringing unique skills to the project. Justin took on the role of Project Manager, ensuring all major milestones were met, resources were allocated effectively, and documentation. Andrew, with a strong background in computer science technical writing, helped lead the documentation of the project and the design of the system architecture. Phillip focused on the user interface and the AI algorithms, designing a visually appealing GUI, and bringing the AI to life. This division of labor was essential for managing the project efficiently.

7.2 Design and Development Process

The project started with the team looking at existing chess AI frameworks to find out what was accessible in AI-driven chess games. We explored different game theories and AI techniques such as the minimax algorithm, working with the greedy algorithm originally before increasing optimization through alpha-beta pruning. Python also expanded into libraries, notably Pygame, known for its work in graphical network design and game performance management. This research helped connect our AI strategy with the best tools available.

One of the major design decisions was choosing Pygame to handle the graphical interface due to its simplicity and the community available to help. The prototyping phase involved creating initial versions of the chessboard and AI to test ideas and gather early feedback. The first prototype featured basic chess functionality, allowing players to move pieces according to traditional chess rules.

7.3 Implementation Steps

The project initially started with Java as the primary development language. However, after discovering the capabilities and support offered by the Pygame library, the team decided to switch to Python exclusively. This decision was driven by Python's extensive libraries and simpler syntax, which decreased the time needed for development and made integration of complex AI algorithms easier.

Coding the basic chess game mechanics was a simpler task which included applying all the standard chess rules, including piece moves, capture mechanics, castling, en-passant and other special moves. Challenges when creating the initial prototypes included enforcing accurate legal move generation, and game rules with certain threats such as checkmate. Problems were solved through intensive debugging sessions and peer review address this issue, helping to clean up the logic and increase the confidence level of the game.

The AI part was developed using a combination of minimax algorithms for basic decision making and combined with greedy algorithms for efficient move evaluation in less complex situations. Later versions used alpha-beta pruning instead of the greedy algorithm to reduce search space by trimming branches with less promising moves. This system was tested extensively through simulated games, putting the AI on its own against different players. Feedback from these meetings led to ongoing adjustments to the AI’s design and bug fixes.

The development of the user interface focused on providing a clean and intuitive experience for players. It was decided to use a shade of green for the chessboard to visually differentiate our game from traditional chess games and reduce visual strain during long games. Performance improvements include highlighting available paths when a piece was selected and providing visual cues for checks and checkmates.

7.4 Challenges and Understandings

Throughout the development of the chess AI, we encountered several technical challenges that tested our problem-solving skills and deepened our understanding of game development. The biggest problems we had to tackle included:

- AI Optimization: One of the greatest challenges was optimizing the chess AI to make decisions quickly without sacrificing the quality of gameplay. Initially, the AI's response time was quick, but the moves it chose did not always represent a winning strategy. Instead, with the greedy algorithm, the AI almost seemed to randomly act on the available moves given to it in that moment. Adding and implementing alpha-beta pruning significantly improved performance, without drastically affecting the speed of the decisions.
- Ensuring that our game ran smoothly on different devices was another technical obstacle. We faced issues with different operating systems handling Python and Pygame differently, although that can be due mostly to the age range of the devices running the program.

This project also helped us gain a deeper understanding of the software development life cycle, and the process of working in a team environment, where each member has different schedules and availabilities. Having a team environment with distinct roles assigned to each member required us to constantly be in communication, with clear and regular updates on all changes made to the project. We had to learn to be able to adapt to any technical hurdles that we encountered, and that ability to do that let us overcome most obstacles. Maintaining clear documentation helped us keep track of every change we made and let us clearly plan out where we wanted to take the AI from there.

7.5 Future Changes and Implementations

Future Changes:

- **AI Improvement:** Continue developing the AI to further improve its decision making. Improvements have been made by shifting from a greedy-algorithm to an alpha-beta pruning algorithm but the possibility of implementing other algorithms/techniques such as a Monte Carlo Tree Search (MCTS) algorithm or a powerful static evaluation function (as used by Stockfish, a popular chess algorithm) may allow for more advanced decision-making.
- **AI Difficulty Variation:** As the AI becomes more advanced and range of skill level the AI exhibits from different algorithms, the player will eventually be able to adjust the AI's skill level to match a desired level of play, whether it is an opponent on the same skill level as the player or a stronger opponent to provide a challenging experience.
- **Expanded Testing:** Shift test focus from AI versus human testers to our AI versus well-known AI. Allows the team to view how the AI we have developed contests with established chess AI such as Stockfish, Deep Blue, etc.
- **Implement more features into the UI:**
    - Option to select AI difficulty.
    - A clearer way of choosing which color piece AI controls.
    - A clearer way to choose between human or AI opponent.

If the project were to continue to be developed, we could possibly explore avenues to move the chess game onto a mobile app to be accessible to users without access to a laptop/PC. If unable to use mobile app, expand onto operating systems other than Windows as that is currently the only platform that can run the chess game with AI.

7.6 Conclusion

This work demonstrates the potential of AI to transform traditional gaming experiences. By integrating complex AI algorithms into well-known games like chess, we showed that AI can be used not only for competitive games but also as a tool for learning and improvement. The lessons learned from this project works towards a broader conversation about the potential of AI in games and shows that artificial intelligence can be used to improve gameplay capabilities, enhance user engagement, as well as be a great educational tool for students.

In conclusion, the SP-14 Red Chess AI project not only achieved its intended goals but also provided the team with valuable lessons in AI development and game design, communication and problem solving. The project stands as a testament to the possibilities

when technology and traditional games merge into something new and lays the groundwork for future research in more integrated and intelligent game systems.

8.0 Version Control

Originally, our goal was to implement the Java coding language to help build our AI and chessboard. But once our team discovered the Pygame library, the choice was made to switch completely to Python for all development and implementation. Version control was established on GitHub after some progress was already made on a standalone device. We used a feature-branch workflow for the project, where new features were developed in separate branches before being merged into the main branch. This process included pull requests and code reviews by team members to ensure code quality across all versions of the project. Version control gave the team a clear history of updates, which was extremely necessary for debugging current and future issues when implementing the AI. In the case that the program stopped working on any of the team members' devices, the code store in the GitHub organization acted as a perfect backup system.

9.0 Test Plan and Report

Testing was conducted on various Windows devices, all of which could run the program without issues. The main objectives of our chess AI design were to:

- Ensure that the AI decision-making process worked as intended throughout the game by making decisions that held some significance towards a proper strategy rather than a move selected at random.
- Ensure the AI's robustness against abnormal inputs and unpredictable user interactions.
- Validate AI integration with user interface and other system components.
- Ensure that the AI followed all rules and regulations without exception, when it comes to validating moves, legal moves, etc.

Key feedback that led to major changes included enhancing the AI level and improving the interface and UI. Test results showed the team that the AI was not challenging enough for more experienced players. This led to further enhancements of the AI's minimax algorithms and more implementation of alpha-beta pruning, which let it be more competitive and plan moves ahead of time instead of focusing on the present moment of the game. The UI also needed to be adjusted to highlight all moves that a player could make, especially if the testers were unfamiliar with special techniques such as castling and pawn promotion.

9.1 Test Results

9.1.1 Prototype Test Results

- When the AI won a round of chess, regardless of the color that won, upon restarting the prototype, the game would no longer respond to user interactions.
- If a certain amount of time passes without any moves being made, the program will sometimes cease to function and require a restart or a complete reboot of the program.
- When making an en-passant move, a visual bug would occur in which the pawn would render a space behind its original position before being taken.
- Due to a flaw in the minimax implementation, the AI would sometimes make moves that were illogical or did not help it secure a victory.
- Some chess pieces would not display correctly after pawn promotion was achieved in-game.

9.1.2 Final Test Results

- Player vs. AI: When switching the controls so that the user controls the black side and the AI controls the white side, attempting to castle kingside causes the program to crash with an unexpected error. When the user controls the white side and the AI controls the black side, this error doesn't seem to happen.
- The Pawn promotion and en-passant errors from the prototype have been adjusted and perform without fail.
- The AI's decision making has drastically improved from the prototype, and carefully chooses the best decision from the list of moves available at each turn.
- The player can switch between user control and AI control without error.
- Player vs. Player and AI vs. AI have no major bugs and perform without error.

9.2 Test Report

Requirements/Design	Pass	Fail
Highlight valid paths for moves	X	
Correctly implements greedy and alpha-beta pruning algorithms	X	
Restart/Exit functions to restart/close the program	X	
Implement different levels of AI difficulty		X
Update chessboard to reflect moves made	X	
Correctly implement the normal rules of chess	X	
Special moves, such as pawn promotion and en-passant	X	

10.0 Conclusion

In conclusion, this report has detailed the entire process and outcomes of making a chess AI capable of playing with humans in real time. Originally implementing the greedy algorithm and minimax, the implementation of alpha-beta pruning will improve the AI's thinking and planning exponentially. The implementation of a visually pleasing GUI allows for enhanced engagement with users. The encounters with various bugs throughout the project highlight the challenges of implementing and evolving an artificial intelligence system, without undermining the accomplishments of creating one. Overall, the completion of this Chess AI will be a great look into understanding how AI works and learns and will continue to be improved with future versions.