CS 4850 - Fall 2024

SP-14 GREEN - Novel Chess Game

Group Members:

Joshua Peeples, Matthew Corvacchioli, Allen Smith, Dylan Luong, Ashton Miller

Professor and Date:

Sharon Perry – December 2, 2024

Project Website:

https://sp-14-green.tiiny.site/

GitHub Repository:

https://github.com/S-14Chess-p/Senior-Project-ChessAI

General Project Stats:

Total Lines of Code: 3600

Number of Project Components and Tools: 2

Total Man Hours: 450

Table of Contents

1.0 Introduction

This report outlines our Chess AI project's essential functional, design, and performance requirements. It serves as a comprehensive guide, showing that the development process stayed on track to meet the necessary performance and design constraints. The report provides detailed specifications for the software, including the core functionality, user interface, and AI design. It also includes guidelines for the chess game's user experience (UI/UX), ensuring that developers and end-users clearly understand how the game should behave and interact. The project delivers a polished, well-functioning chess game that meets user expectations and technical requirements by adhering to these specifications.

The report begins with a brief overview of the project goals, followed by definitions of key terms such as Artificial Intelligence (AI) and Graphics User Interface (GUI), to ensure that all involved parties are aligned on the fundamental concepts. The core sections then dive into the functional requirements of the game, detailing the different modes of play, the game's pieces, and their movements, ensuring clarity on both the gameplay and design elements. Ultimately, this document will guide the development process, helping to create an engaging and functional chess game for various users.

2.0 Requirements Introduction

2.1 Overview

The software requirements highlight the project's functional, design, and performance requirements. It contains essential information regarding the performance and design constraints, which will be followed closely to ensure the project remains on track. The requirements also briefly touch on the project's UI/UX to provide a guideline.

2.2 Project Goals

1. Develop the game chess using the programming language C#

2. Create an AI that knows how to play the game but is not trained well; it will only sometimes win.

3. If there is time, add difficulty selecting options for the AI.

4. If necessary, create alternate game modes

2.3 Definitions

Artificial Intelligence (AI): In the context of this document, AI refers to the entity that controls the side not chosen by the Player and makes moves in adaptation to those taken by the Player.

Graphics User Interface (GUI): How the user will interface with the program in an easily accessible manner.

3.0 Functional Requirements

3.1 Overview

Like all HTML projects, this program can run in a browser. The user is presented with three pages labeled Home, Rules, and Play. The home page will give an overview of the project. The rules page will present the user with a read-only version of the ruleset. The play page will have three options: Online Player vs Player, Local Player vs Player, and Player vs AI. Each of them will do exactly what they sound like they do. Player vs Player will allow two players to play against each other locally on the same device or online using a room code. Player vs AI will allow a Single-player to select an AI model to play against.

3.2 Chess Game Modes

The chess game has different modes of play, and the modes are:

3.2.1 Player vs Player

Player vs Player (PVP): When two human players compete against each other in the game of chess.

3.2.2 Player vs Environment

Player vs Environment (PVE): When a human player competes against an AI in chess.

3.3 Chess Game Pieces

In chess, six different types of pieces can be moved.

On game start, each Player starts with a certain number of pieces on the board:

- Eight Pawns
- Two Knights
- Two Rooks
- Two Bishops
- One Queen
- One King

Each piece has a certain number of values called points, and some are worth more than others. Players use a system that keeps track of the chess pieces' worth.

3.3.1 Pawn (1 point)

Pawns are the most basic and least powerful pieces in the game. Each side starts with eight pawns, and they all start at the second row (or ranks) from the Player's side. If it is a pawn's first move, it can move forward one or two squares, but if a pawn has already moved, it can move forward one square at a time. A pawn threatens or captures the opponent's piece diagonally. A pawn has a special move when it encounters the opponent's Pawn's first move, the En Passant. See 3.4.14 for more details on En Passant.

3.3.2 Bishop (3 points)

Bishops can move diagonally in any number of unoccupied spaces if it is not blocked by its own pieces. An easy way to remember how the Bishop can move is it moves like an "X" shape. The Bishop can capture an opponent's piece by moving to the occupied square where that piece is located. Each Bishop starts on a colored square and can only move on that type of square for the rest of the game.

3.3.3 Knight (3 points)

Knights can move two spaces in a cardinal direction, then one space in a left or right manner, OR it can move one space in a cardinal direction, then two spaces in a left or right manner. An easy way to remember how the Knight can move is it moves in an "L-shape." The Knight is the only piece that can move over other pieces, whether ally or enemy. Knights can only capture the enemy pieces where it lands, not what jumps over. Since Knight can jump over pieces, it is the only piece that can be moved in the first two turns of the game along with the Pawn. Knights start between the Bishops and Rooks.

3.3.4 Rook (5 points)

Rooks are powerful pieces that can move any number of unoccupied spaces in the cardinal directions (if other pieces don't block it). An easy way to remember how a rook moves is that it moves like a "+" sign. The Rook can capture an enemy piece in their direction. The Rook starts in both corners of the Player's side. Rooks are powerful pieces whose main weakness is their lack of early-game mobility, as it is a weak move to move the pawns that block the Rooks early in the game. Rooks typically take multiple moves to get into relevant play. Rooks are unique in how they can perform the Castle maneuver along with the King. See 3.4.17 for more details on Castling.

3.3.5 Queen (9 points)

The Queen is the most powerful piece in chess. The Queen has the movement properties of the Bishop and Rook combined. The Queen can move horizontally, vertically, or diagonally in any number of squares. The Queen can capture any square it can move to. With proper positioning, it can threaten or capture any pieces without retaliation.

3.3.6 King (infinitely valuable)

The King is the most important piece in chess. Like the Queen, the King can move in any direction but only one space at a time. The King cannot move into any threatened squares, and when the King is in check, no other moves can be taken unless it brings the King out of check (whether by moving the King, blocking the threat with another piece, or capturing the threatening piece). As a result, the Player's and the opponent's kings cannot threaten each other directly. In traditional chess, the King cannot be captured. Instead, a checkmate is declared if the King can be captured after a turn of check. The King can perform the Castling maneuver with a rook. See 3.4.17 for more details on Castling.

3.4 Game Rule

When and where a piece can move.

3.4.1 Capture

The term for when a piece takes another piece off the board.

3.4.2 Check

Check is the term for when a King is threatened. When this occurs, the Player controlling the threatened King cannot make any moves that do not solve the threat. Valid options are to move the King, block the threat with another piece (which cannot be done against Knights), or capture the opposing piece. The King can capture pieces that threaten him, but only if another piece does not protect them.

3.4.3 Checkmate

The state of the game where a check would occur, but the Player in check cannot protect the King in 1 move. The game is over at this point, with the Player in Checkmate being the loser and the one who put them in Checkmate being the victor.

3.4.4 Draw by Consent

When neither Player can secure a victory, the outcome can be decided by an agreement among all players, which results in a stalemate

3.4.5 Draw by Stalemate

A Draw forced upon the players due to the game state occurs when a King is NOT in Check, no other piece can be moved, and the King can only move into threatened squares. Some rulesets declare this a victory for one Player, generally either the one with the most points remaining or the one who forced the Stalemate on another player.

3.4.6 Draw by Repetition

A Draw that can be claimed by either Player when the same game state repeats 3 times in a row (i.e. all pieces have been in their exact location at least 3 times in the game). The other Player does not have to consent to the draw.

3.4.7 Draw by Fifty-Move Rule

A Draw that either Player can claim occurs when two criteria have been met:

1. A pawn has not been moved in 50 consecutive moves
2. A piece has not been captured for 50 Consecutive moves.

3.4.8 Threatened

A piece is considered threatened when it is in such a position that it would be captured on the opponent's next turn if it were to remain there.

3.4.9 Protected

When a piece is 'Threatened' by an ally piece, that piece is considered Protected. If a Protected piece gets captured, the Player can retaliate.

3.4.10 Pin

Pin occurs when a less valuable piece is being threatened but cannot move without exposing the more valuable piece behind it.

3.4.11 Skewer

A skewer occurs when a more valuable piece is being threatened but cannot move without exposing the less valuable piece behind it.

3.4.12 Fork

A fork occurs when a single piece threatens two other pieces at the same time

3.4.13 Stuck piece

A piece that has no valid move.

3.4.14 En Passant

A unique capture mechanic exclusive to pawns. To perform this capture, you must take your opponent's Pawn as if it had moved just one square. You move your Pawn diagonally to an adjacent square, one rank farther from where it had been, on the same file where the enemy's Pawn is, and remove the opponent's Pawn from the board.

There are a few requirements for this move to be legal:

1. The Player's Pawn must advance exactly three squares (or ranks) to perform this move
2. The opponent's Pawn must move two squares in one move, landing right next to the Player's Pawn
3. The En Passant capture must be performed on the turn immediately after the Pawn being captured moves. If the Player does not perform En Passant on that turn, they can no longer do it on the next turn and the turn after.

3.4.15 Promotion

When a pawn reaches the opponent's back row, it is promoted into a major or minor piece.

3.4.16 Underpromotion

The term for when a pawn is promoted into a piece other than the Queen. This is done for strategic purposes.

3.4.17 Castling

A maneuver where a King can move two squares to the left or right towards the chosen Rook, and the chosen Rook jumps across to the other side of the King, all in one move.

This move can only be possible under these conditions:

1. Your King cannot have moved.

   Once your King moves, you can no longer castle, even if you return the King to its starting square.

2. Your Rook cannot have moved.

   If you move your Rook, you cannot castle on that side anymore, even if you return the Rook to its starting square.

3. Your King cannot be in check.

   You cannot castle while your King is threatened. Once you are out of check, you can castle. Being checked does not remove the ability to castle later.

4. The King cannot pass through check.

   If any square the King moves onto would put the King in a threatened position, you cannot castle. You will have to deal with the attacking piece first.

5. No pieces can be between the King and Rook.

   All the space between the King and Rook must be empty. Developing your pieces into the game as soon as possible is important.

3.5 Board Layout

Chess is played on a board of 8 x 8 squares arranged in vertical rows called "files" and horizontal rows called "ranks."These squares alternate between two colors in a checkered pattern. Each square is identified by its row and column coordinates. The files are listed from "a" to "h", and the ranks are listed from "1" to "8".

It's important to orient the board in the right direction so the chess pieces for each side are set up correctly. The board should be positioned so each Player has a light-colored square in their bottom-right corner.

Pawns are set up on the second row of the Player's side.

Rooks are placed in the corners.

Knights are placed next to the Player's rooks.

Bishops are placed next to the Player's knights.

The Player's Queen should be placed on the same square of color as her army. The white Queen should go in the light square, and the black Queen should go in the dark square.

Lastly, the King should be placed next to the Queen.

3.6 Chess Game Play

The Player opens the chess game and is allowed to choose which color (black or white). Once the Player or both players choose their color, the board layout will be set up with chess pieces on both sides, and both players are given a clock.

In chess, the Player with the white pieces always moves first. Once the white piece player moves, the turn is over for the white piece player, making them inactive at this point, and they can no longer move. Then, the timer starts for the Player with the black piece, and it becomes their turn to move. Once the black piece player makes the move, their turn is over, and the timer pauses for the black piece player, becoming inactive during the white piece player's turn.

This gameplay will continue until a player checkmates the opponent's King. It then becomes a Stalemate, meaning one Player can no longer take action, and neither side can win or make progress. One player surrenders or the timer runs out for one Player, resulting in a loss for one Player and a win for the other Player.

4.0 Analysis

We designed the webpage with the primary goal of creating a chess game with two game modes: Player vs. AI and Player vs. Player. Our initial focus was on building the core functionality of the chess game. First, using C# and HTML, we constructed an 8x8 chessboard as our display with different variances of chess pieces. Each piece has different names, designs, and functions, so we developed unique functions for each of them in C#. Then, we listed the rules of chess and implemented them into the chess game to make it function properly.

Once the basic core game was set up, we wanted to make a web application with interfaces for users to play the game. We wanted to build the app in a browser with a home page, a rule page, and a gameplay interface. For starters, an issue we considered was that C# cannot directly run in a browser, so we solved this issue by using the ASP.net framework to build a web application. Then, we use HTML files to set up the UI to help style our webpage. This allowed us to seamlessly integrate the different game modes and UI into a fully interactive experience. For Player vs. AI, we incorporated the Minimax algorithm to simulate an intelligent opponent, while for Player vs. Player, we included both local and online modes. In local mode, two players can play on the same device, while in online mode, players can join a game using a room code.

5.0 Design Introduction

The System Design describes how all the parts identified in the requirements interact at a high level. It orchestrates our design goals clearly and will provide an overview of the system architecture. Lastly, it describes the goals we had in mind when designing our product.

It guides the development team on how the system architecture and design should be set up. The intent is to provide the development team with documentation to refer to when they are stuck and documentation to serve as a structural piece that helps the whole team understand the project's architecture. This document is meant to be at a high level, and as such, anyone other than those the Project is intended for would have difficulty understanding it. As the project's development period continues, the team will grow much more of a low-level understanding of the project's ins and outs based on the expansion of the high-level knowledge from this document.

5.1 Design Overview

C# will be the primary language used to code the logic behind our project.

HTML files will be used to set up the user interface for our project. This HTML file will contain the necessary CSS stylings for the frontend design that the project aims to meet spec-wise.

Windows will be the primary operating system used for this project.

The user will utilize our program to play and enjoy chess, either with a friend in the player-vs-player mode or against a trained AI that we have programmed to learn the game to a playable metric.

5.2 Design Constraints

C# is server-side and, as such, is unable to run in a browser with an HTML file with a language like JavaScript would be able to, so we used the ASP.net framework that allows us to bypass this issue.

5.2.1 Environment

We have set our own small list of constraints

- C# will be used to program the logic
- HTML will be used to program the UI
- C# is server-side, so it can't directly run in a browser to work with an HTML file. We will use the ASP.net framework to bypass this issue.

5.2.2 User Characteristics

Each user needs a mouse to play our game. When the user clicks on a piece, it will highlight every valid move that piece can make. The user can then click on the highlighted square to move that piece or a different piece to show other possible moves.

5.2.3 System

Windows 10 Operating System (OS). An Internet connection is required.

5.3 Development Methods

We will begin by developing the game of chess as everyone knows it in the Player vs player form so that we know our program works. Once we are happy with how that is functioning, we will develop Player vs. AI and hopefully have a fully fleshed-out product by the end. We will expand beyond that if we have the time to do so.

Since the beginning of the Project, we have opted out of including the tutorial and opted for a Player vs Player online mode where two players on separate computers and either the same or separate network connections can play with one another instead.

6.0 Architectural Strategies

Our goal is to use the C# language for the overall design of the game and the functionality of the game's rules/AI, as well as HTML and CSS for the design of the front end of our user interface. The IDE of choice will be Microsoft Visual Studio through our KSU accounts.

We currently do not plan to use any existing software components and rely entirely on our own developed software, outside of using a webpage template that Visual Studio provides.

From the development phase of the project, we began the implementation phase. Once implemented, numerous testing methods were utilized to ensure the program ran according to our design and how we saw fit. During the testing phase, we took adequate measures to ensure the program was polished before the final submission.

The primary paradigm that we aimed to focus on is the Graphical User Interface that will be utilized for the front end of the project. We will have a home screen, which will have three buttons up on a toolbar for navigation. The first screen (the launch screen) will be a home screen providing a generic overview of the project. The second screen navigates the user to a page that has a list of extensive rules. The third page facilitates different versions of the game.

The first version of the game is a Player vs Player version, in which two players can simultaneously play against one another on the same device.

The game's second version is a Player vs AI version, where the user can play against an engineered AI with different characteristics, making each model a different experience.

Finally, the third version of the game features a Player vs Player online mode, in which two players can connect to enjoy the game on separate machines and Wi-Fi networks.

As mentioned before, each mode will be readily accessible from the Play screen, which the user can easily navigate from the home screen.

We developed the program using C # and intend to refine the design of its front end using HTML markup and CSS styling. The ASP.net framework will allow these two file types to work together.

7.0 System Architecture

We hope the system architecture will remain simplistic and will always be willing to expand it if necessary. C# will be the primary backend language responsible for running the AI and the logic for detecting valid moves. That C# code will then need to run through a web framework that will allow it to communicate with an HTML file, which will be responsible for the frontend UI and sending user input back to the C# program to detect valid moves.

Below, a diagram of the system architecture design can be found.

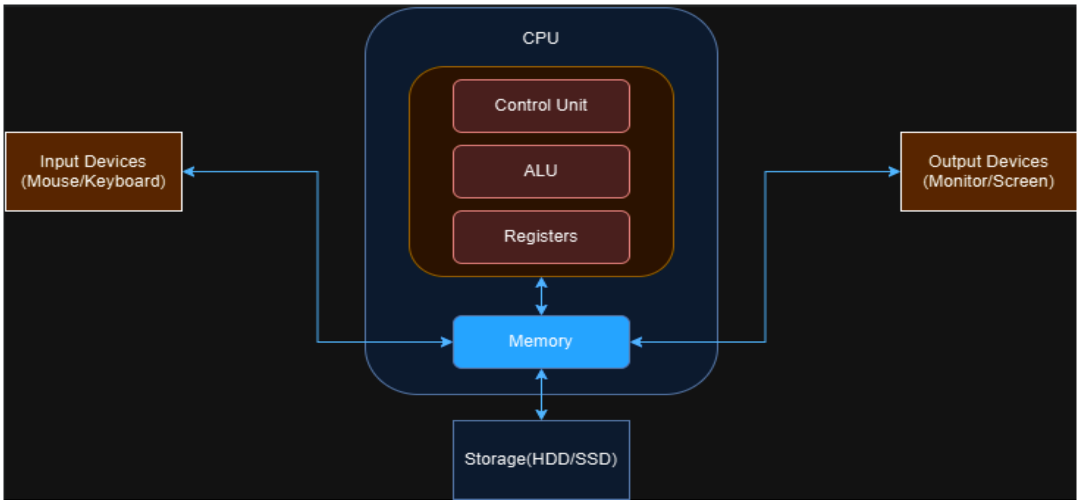

Figure 1: A system architecture design diagram that details the process from input to CPU to output.

8.0 Detailed System Design

8.1 Classification

The Minimax algorithm will be the primary component behind the AI; it utilizes a heuristic to make an informed decision on the best move available. This algorithm allows the programmer to limit how far in advance the AI can look, lowering its capabilities so it only sometimes makes the perfect move.

8.2 Definition

Minimax algorithm - Common in 2-player turn-based games, the Minimax algorithm represents the game in a partial heuristic, which uses the current state of the board as the root. The algorithm begins by generating all the possible moves for a defined depth, which limits how far ahead the AI can look. This depth value helps the AI's performance and prevents the AI from making the perfect choice every time. Once the heuristic is fully generated, the AI uses some logic to determine the best possible move with the generated move list.

8.3 Constraints

C# is server-side and requires a framework to work with HTML. We will be using ASP.net to handle this.

To prevent the AI from slowing the game down, we will have to test different depths within the Minimax algorithm to create an effective AI without hindering performance.

8.4 Interface/Exports

This component provides the set of services (resources, data, types, constants, subroutines, and exceptions). The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc.

For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

9.0 External Interface Requirements

9.1 User Interface Requirements

Three pages labeled Home, Rules, and Play. Home will give a general overview of the project. Rules will provide a non-visual written version of the ruleset. When the Play button is selected, the user is presented with three options: Online Player vs Player, Local Player vs Player, and Player vs AI. All three will present the user with a chess board to play with the option they select, which changes how the back end will function. Online Player vs Player will require a way to create a room code and a place to enter it so someone can join it. Player vs Player and Player vs AI will each have a UI element indicating to the user which mode has been selected, but there will be no additional UI elements other than that.

9.2 Hardware Interface Requirements

9.2.1 Display Interface

A screen or monitor displays the game board, pieces, and user interface elements. A basic LED display will suffice.

9.2.2 User Input Interface

A mouse and keyboard will be used to support the chess application on a computer. Input will be made through a mouse to select and move pieces on the board and a keyboard to enter settings.

9.2.3 Battery

The hardware should have enough battery life to support the game.

9.2.4 Processing Hardware

Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Memory are required to play the game.

10.0 Development

10.1 UI

Designing an effective user interface was our top priority for creating effective chess AI. Logically, we started by creating a representation of a chessboard using primarily HTML, though JavaScript and CSS were also used. A Module 2 function places the chessboard with different colored squares on the UI. Our UI implementation has evolved significantly throughout the project.

To start, our UI was rather barebones. It consisted of a home screen with our group and an option to navigate to a few other pages, including a rules page and a play screen. An active chessboard with three buttons is displayed on the play screen so you can choose between tutorial, PVP, and PVAI. We would end up scrapping the tutorial later and replacing it with Online Player vs Player.

The next iteration of the UI introduced a more advanced layout, moving the page option buttons from the previous version to a navigation bar. The pages were now more visually appealing; this version was initially in light mode only. The idea of dark mode was discussed throughout the semester and would later be implemented.

In this iteration, the chessboard was redesigned, and the game mode buttons were updated. The color scheme was refined, with the board adopting a green tint, and the game mode buttons received a visual pass. After finishing a game, a pop-up now replaces the individual Victory, Defeat, or Draw screens.

The third UI version is built on the previous design, with color updates and new features laying the foundation for the final version. The board's colors shifted from green and beige to purple and deep red, and the game mode buttons matched the board's purple for consistency. Buttons also gained shadows, with selected options highlighted in deeper purple. A grey box at the top of the page identified each game mode. The PvAI option allowed users to choose an AI opponent (Randy Random) from a dropdown and start the game. A Pop-up for results replaced dedicated victory screens and the dark mode checkbox was added as a first step toward full dark mode support.

In the final UI iteration, the entire app was redesigned with a chessboard background image. The dark mode was fully implemented. It now applied in the whole app and saved between pages. The play screen received the most updates: in dark mode, the board and interface elements shifted to a red theme, with dark red squares, white and black pieces, and red menus. The board also became rounded for a cleaner, more uniform design. In light mode, the previous color scheme kept distinguishing the two themes.

Functionality improvements included new AI opponents (Minimax AI and SmartyAI), a turn identifier, a resign feature for both PvP and PvAI modes, and a responsive design that allowed elements to resize based on the window size, ensuring a consistent experience across different screen sizes.

Chess game engine-The Chess game is built using three different C# files called Piece.cs, Game.cs, and Board.cs. Each of them has their own responsibilities and must work together to allow everything to function. Game.cs is responsible for keeping track of all the general rules of chess. It has variables that keep track of whose turn it is, whether the game is over, and the status (ongoing, Checkmate, Stalemate). Game.cs is also responsible for checking for Stalemate and Checkmate after every move and handles most of the special case rules like En Passant and Pawn Promotion. Game.cs also has a method that returns every valid move, which is used a ton in the AI models discussed later. Board.cs initializes the board as a 2D array and places each piece in the correct initial position. After initialization, it keeps track of where each piece is currently located and has a few methods that help with some of the special case moves and rules. Piece.cs defines a parent class with a few variables to help determine which team the piece belongs to and its position. Each piece type is defined as its own class that inherits from Piece. Each class defines movement logic, and in the case of pawns, there is a separate function to define capture logic and an attribute to determine if the Pawn is En Passant eligible. Rooks do not require additional capture logic, but they have a similar attribute to determine if the Rook is eligible for Castling.

10.2 Pieces – Rules

10.2.1 Pawn

The most basic piece in chess. Each Player has 8 of them, and they all start at the second row from the Player's perspective. A Pawn can typically only move one square forward, but if it has yet to be moved from its home row, it can instead optionally move two squares forward. A pawn threatens its forward diagonal squares and can capture an enemy piece in these squares. Pawns cannot move backward, nor can they move diagonally except to capture. Furthermore, Pawns can only move two squares if they have not moved yet. A pawn that moves to squares and bypasses the threat range of an enemy pawn can be captured by that Pawn as if it moved only 1 square. See En Passant for more details.

10.2.2 Bishop

Bishops can move any number of unoccupied spaces in a diagonal. The Bishop's threat range is the same as its movement range. Each Bishop starts on a colored square and can only move on that type of square for the rest of the game. Bishops are strongest in openings for claiming the center board and during the late game where most other Pieces have been captured.

10.2.3 Knight

Knights move in an L shape, going two spaces in a cardinal direction and then one space in a direction that forms an L. Knights are the only pieces that can move over other pieces, whether ally or enemy. Knights threaten the squares that they end their move on. Knights are the only pieces other than pawns that can be moved in the first two turns of the game. Knights are also the only piece that can threaten the Queen while not being threatened back. Knights start between the Bishops and Rooks

10.2.4 Rook

Rooks are powerful pieces that can move any number of unoccupied spaces in the cardinal directions. The Rook threatens any space they can move to. The Rook starts at the farthest corners of the board. Rooks are powerful pieces whose main weakness is their lack of early-game mobility, as it is a weak move to move the pawns that block the Rooks early in the game. Rooks typically take multiple moves to get into relevant play. Rooks are unique because they can perform the castle maneuver with the King; see Castling for more details.

10.2.5 queen

Queens are the most powerful pieces in chess. A Queen has the movement properties of a Bishop and Rook but cannot combine their movement in one turn. The Queen threatens any square it can move to and, with proper positioning, can threaten any piece without retaliation. The Queen is a powerful piece that is not to be squandered, but it is still ultimately expendable and unlikely to survive until the endgame during close games. Skilled players can offer up their Queen in exchange for potentially game-winning momentum and board control. Queen trades are common occurrences. The Queen starts adjacent to the King, on the square opposite their team color.

10.2.6 King

The King is the most important piece in chess. Like the Queen, the King can move in any direction, but unlike the Queen, it can only move one space at a time. The King cannot move into any threatened squares, and when the King is threatened, no other moves can be taken unless it brings the King out of threat (whether by moving the King, blocking the threat with another piece, or capturing the threatening piece). As a result of this, the two kings cannot threaten each other directly. In traditional chess, the King cannot be captured. Instead, a checkmate is declared if the King can be captured after a turn of check. The King can be one of the most powerful pieces on the board in the late game, where most minor and major pieces have been expended. Kings can perform the Castling maneuver with a valid rook.

10.2.7 Castling

Castling is the only move that allows you to move more than one piece in one turn. If the King has yet to move, and the Rook the King is trying to castle with has yet to move, the King can move two squares closer to the Rook's starting position, and the Rook will be moved to the other side of the King. This maneuver is not available if any of the squares traversed by the King are under threat.

10.2.8 En Passant

En Passant is a special pawn capture in chess that occurs when a pawn moves forward two squares as its first move and passes an adjacent opposing pawn. When this happens, the opposing Pawn can capture the first Pawn as if it only moved forward one square. This capture must be made on the next move, or the right to capture En Passant will be lost.

10.2.9 Stalemate

There are several forms of Stalemate. One of the most common occurs when one of the kings is not directly under threat, and the Player whose turn it currently is has no moves available. Threefold – Repetition occurs when the same board state occurs three times. If each side only has their King left, there is also a stalemate due to insufficient material.

10.2.10 Checkmate

To win a game of chess, you must put the opponent's King in a position where it cannot escape check in a single move, resulting in a checkmate.

10.3 Modes of Play

Player vs Player (Local)– For the Player versus player mode, our group sought to provide a local experience for two players of the game to interact on the same interface. This mode utilizes C#, HTML, and CSS to provide structurally tact design and functionality. A key feature developed throughout the semester showed potential moves that users can make in their turns. Using a combination of C# backend functions and HTML frontend interaction options, upon selecting a piece, as the user begins to drag the piece, the board displays the options on each square so that the user can move their piece.

Player vs Player (Online)- For the online version of the Player vs Player mode, two players can connect using a randomly generated game code from the host's side of the game. Once the code is generated, the host may share it with the other Player, and the other Player may then enter that code to play in a lobby. Most of the online play mode functionalities are the same as those in the local PvP mode.

Player vs AI – Player vs AI functions by using one interface and one class. IAIplayer defines every class that inherits from it as having a getNextMove Function. The class AIfactory keeps track of what AI is in use and assigns it to a team. This sounds simple on paper, but it has made developing all the different AI models so much easier because we duplicate the simplest, change the decision-making logic, and it is ready to test.

10.4 Algorithms

10.4.1 RandyRandom

The primary goal was to use a simple move selection algorithm to ensure our framework to set up the more advanced AI models was functioning. To fit these criteria, we utilized the random class built into C# to select each move. This gave us a nice baseline to work with, and when we want to make a more advanced model, we just duplicate RandyRandom and add decision-making logic.

10.4.2 SmartyAI

This was our attempt at making our own AI model that does not follow a preexisting algorithm. SmartyAI gets every valid move on the board, simulates every move, runs the move through the evaluation function, and returns the move with the best outcome. The evaluation function considers a few factors when evaluating a move. Firstly, each piece type has a positional value for every square on the board. Secondly, each piece type has a weight that is only added to the overall evaluation value if a piece of that type is captured. Furthermore, if a piece is captured in a move but doing so results in the piece that was moved being threatened, the weight of the piece now under threat is subtracted from the total evaluation of the move. This allows the AI to consider trades, so if it can take the Rook at the expense of its Queen, it will not make the move, but if it can take a queen at the expense of a knight, it will make that trade. Lastly, if a move is available for the AI that results in a checkmate, it returns int.maxValue so that move is guaranteed to be selected.

10.4.3 Minimax

The Minimax algorithm is a common AI model used in several two-player games. Smarty AI and Minimax assign each piece type a weight and positional value, but the one used in Minimax is a much larger scale, allowing for weights to be more specific. Minimax separates itself from Smarty AI because it can look at a specified number of moves in advance that can be altered simply by changing a variable. All our AI models are designed to always play as black because it would require significant restructuring to allow the Player to choose. The AI will choose the move that maximizes its evaluation score when it is black's turn. When looking ahead, the AI must predict the move that white will make, so it assumes white is playing optimally and goes down the path where white makes the move that minimizes black's odds of losing. This algorithm is by far the most competent AI model of the three we have developed, but because there are so many moves being evaluated every turn runtime can become a concern, so the Minimax algorithm has Alpha-Beta Pruning to help with this. This works by updating the values for a variable called Alpha as the search tries to maximize and update the value of Beta as the search tree tries to minimize. If the value of Beta ever exceeds the value of Alpha, we break out of that branch and avoid any further evaluation because that move will not help the AI.

10.5 Setting it up

Here is how to set up the Web Chess game:

1. On launch, the program will send you to the Home page, where you will see our names and a navigation bar with options labeled Home, Rules, and Play.
2. If you are unfamiliar with the rules of chess, click on the rules page to read up on them.
3. Once you feel comfortable with the rules, click on Play. You will be presented with an active chessboard. This page defaults to local PvP so you can play against your friends.
4. If you want to play against an AI, click the PvAI button on the right side of the board. This will shift the board to an inactive state and ask you to select an AI. A new UI will appear on the board's right side, containing a dropdown menu to select the AI model you want to play against. Click Start Game to begin playing against the AI.
5. If at any point you feel like you are going to lose and want to restart, the Resign button on the right side of the screen will end the game.

10.6 Monetization

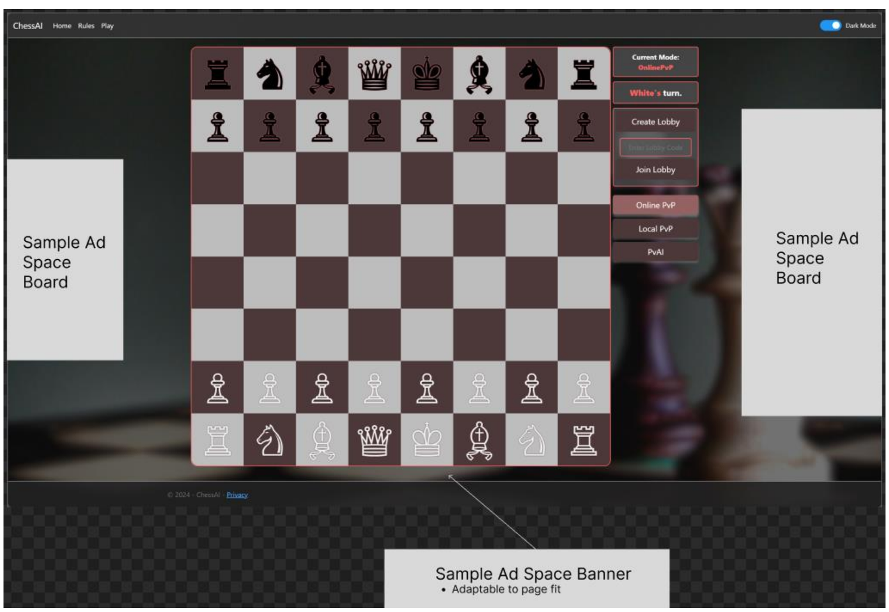

Figure 2: Sample ads display on the browser

We were responsible for creating a conceptual product centered around a monetization strategy. The above figure shows a mockup of the play screen PvP online mode with three separate ad spaces in specific positions on the page. For all the ad spaces displayed, there can be a multitude of opposing configurations in terms of number of spaces used, position of the spaces, and size of the spaces.

An important factor to note is that within the Banner implementation, the vision for this space specifically opposed that of the board spaces, where the boards remain theoretically static on the page.

The banner space would primarily be adaptable in size depending on the overall bottom margin length of the page to the board.

A few forms of monetization also considered for implementation were a paywall that would unlock new forms of AI opponents to play against using different algorithms. These new models would provide more variance in terms of difficulty to the Player. A couple of other conceptualized ideas focused on the overall cosmetic design realm, where a potential paywall would unlock different site themes that would change the design and colors of not only the board but also the pieces.

11.0 Test

11.1 Test Plan

Test scenario: Chess page functionality

Requirement	**Pass**	**Fail**
Run in browser	10	0
Home page displayed	10	0
Rules page displayed	10	0
Chess board displayed	10	0
Number of Chess pieces correctly displayed	10	0
Player vs Player function	10	0
Player vs AI function	10	0
Resign function	10	0

Test scenario: Chess piece functionality

Requirement	**Pass**	**Fail**
Pawn movement function	10	0
Rook movement function	10	0
Knight movement function	10	0
Bishop movement function	10	0
Queen movement function	10	0
King movement function	10	0

Test Scenario: Advanced rules

Requirement	Pass	Fail
Castling function	5	0
En passant function	10	0
Capture function	10	0
Check function	10	0
Checkmate function	10	0
King pinned function	10	0
Promotion function	5	0

Test Scenario: AI plays

Requirement	Pass	Fail
Randy Random Runs	10	0
Smarty AI Runs	10	0
Minimax Runs	9	1

Test Scenario: Online Functions

Requirement	Pass	Fail
Room Code Created	10	0
Successfully Joined	5	0
Move Reflected on Both Devices	10	0

11.2 Test Report

We ran each feature several times, and the number of tests varied based on how long each feature took to test. If anything went wrong, whether a crash or a bug, it was considered a failure; if no issues were found, it was considered a pass. In the case of the AI models, we thought it a pass if the AI played the game. We have been utilizing loggers throughout the development of our project so that the program can tell us what it is thinking as it runs, which has made keeping a low number of bugs accessible.

Whenever a major bug or crash has appeared, we have worked to resolve the issue quickly. We had one major bug causing captures to happen when they should not, but the issue was quickly diagnosed to be caused by the stalemate logic and a fix was pushed out within the week. We also found a crash that could occur in the Minimax AI when it moved a piece near the edge of the board. That one was more difficult to diagnose, but once found, a fix was pushed immediately. Unfortunately, there is a known issue with our final product that we were unable to fix. When the Minimax AI starts losing and only has a few moves available, it will sometimes attempt to make an invalid move, causing the program to crash.

Version Control

The purpose of version control within this project's scope was to keep track of all the many moving and static parts of the program that it comprises. For our version control tool of choice for this project throughout the semester, we used GitHub to account for and keep track of all changes from early program conception to draft stages, all the way to the near semester end finalized version. Currently, the development team has contributed nearly 70 total commitments to the project.

Conclusion/Summary

Chess has been played for centuries and is one of the most popular board games in history. Our version features AI opponents of varying difficulty levels, offering players different challenges. Users can also play against each other online or locally.

Throughout the semester, the project has evolved in design, functionality, and documentation. The most significant change has been to the User Interface (UI). Initially simple, the UI now includes features like dark mode with unique color palettes for each mode. Other design updates include improved layout, board aesthetics, and a more professional rules and index page.

Functionality also progressed significantly. What started as a simple app with one AI and local PvP grew into a fully functioning webpage with multiple AI models and new game modes like online PvP. The development team's primary focus was on enhancing app features and functionality.

Documentation has been crucial for tracking progress. Early documents like the Project Plan and Software Requirements Specification outlined the team's goals and design intentions. Later, the Development Document and Software Test Plan tracked updates, game modes, algorithms, and test benchmarks.

In conclusion, the team has made significant strides in development, design, and documentation, reflecting the time and effort invested in the project.

References

GeeksforGeeks. (2018, October 5). *Minimax Algorithm in Game Theory | Set 1 (Introduction) - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/Minimax-algorithm-in-game-theory-set-1-introduction/

Appendix