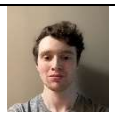
# SP-14 GREEN — Novel Chess Game

## CS 4850 - Section 02 – Fall 2024

## November 15, 2024

| | |
|---|---|
|  <br>Matthew Corvacchioli<br>**Website Management** |  <br>Joshua Peeples<br>**Lead Developer** |
|  <br>Ashton Miller<br>**Documentation & Design** |  <br>Dylan Luong<br>**Documentation** |
|  <br>Allen Smith<br>**Team Leader & Development** | |

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 is used to place the chessboard with differently colored squares onto the UI. Our UI implementation has evolved significantly over the course of the project.

To start, our UI was rather barebones, consisting of a home screen with our group with an option of navigating to a few other pages consisting of a rules page and a play screen. On the play screen, an active chessboard would be displayed with three buttons to choose between tutorial, PVP, and PVAI. We would end up scrapping the tutorial later.

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, and this version was initially light mode only. Throughout the semester, the idea of dark mode was discussed 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 game mode buttons turned blue. After finishing a game, a popup 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. Popup 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 home screen and the entire app were redesigned with a chessboard background image. The dark mode was fully implemented, now applying across the entire app and saving 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 who’s turn it is, if the game is over or not, and the current 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 that has 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.

Pieces – rules

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 normally only move 1 square forward, but if it has not been moved from its home row, it can instead optionally move 2 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 2 squares if they haven't 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.

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.

Knight: Knights move in an L shape, going 2 spaces in a cardinal direction and then 1 space in a direction that forms an L. Knights are the only piece 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 2 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

Rook: Rooks are powerful pieces that can move any number of unoccupied spaces in the cardinal directions. Rooks threaten any space they can move too. Rooks start 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 in that they can perform the Castle maneuver along with the King, see Castling for more detail.

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 is unlikely to survive till 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 that is opposite their team color.

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 1 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 by capturing the threatening piece). As a result of this, 2 kings cannot threaten each other directly. In traditional chess, the King cannot be captured. Instead, if the King were able to be captured after a turn of check, a checkmate is declared. The King can be one of the most powerful pieces on the board come late game where most of the minor and major pieces have been expended. Kings can perform the Castling maneuver with a valid rook.

Castling: Castling is the only move that allows you to move more than one piece in one turn. If the king has not moved yet, and the rook the king is trying castle with has not moved yet, the king can move 2 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.

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 occurs, 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.

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 it is also a stalemate due to insufficient material.

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.

## Modes of play

Player vs Player – 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 that was developed throughout the semester was the feature to show potential moves that each user can make in their turns. Using a combination of C# backend functions and HTML frontend interaction options, upon a selection of a piece, as the user begins to drag the piece the board displays the options on each square that the user can possibly move their piece.

Player vs Ai – Player vs AI functions by using one interface and one class. IAIplayer defines that every class that inherits from it must have 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 of the different AI models so much easier, because we just duplicate the simplest, change the decision-making logic, and it is ready to test.

Algorithms

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.

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, it then simulates every move, runs the move through the evaluation function and returns the move with what it deems to be 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 there is a move available for the AI that results in a checkmate it returns int.maxValue so that move is guaranteed to be the one that is selected.

Minimax – The MiniMax algorithm is a common AI model used in several two player games. Smarty AI and MiniMax both assign each piece type a weight and positional value, but the one used in MiniMax is a much higher scale allowing for weights to be more specific. Where MiniMax separates itself from Smarty AI is 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 a ton of restructuring to allow the player to choose. When it is black's turn the AI will choose the move that maximizes its evaluation score. 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 to win. This algorithm is by far the most competent AI model of the three we have developed, but because there are tons of 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 is trying to maximize and update the value of beta as the search tree is trying 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 player.

How to set it up

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. New UI will appear on the right side of the board that contains 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, there is a Resign button on the right side of the screen that will end the game.