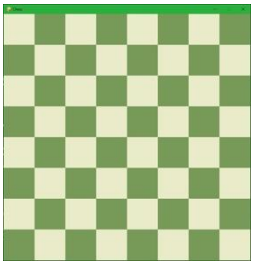
AI-Chess

Final Report

Version 3.0 (December 2nd)



Course: CS 4850

Section: 04, *Semester*: Fall 2024, *Professor*: Arthur Choi

Team: SP-14 (Blue)

Website: SeniorProjectWebsite

Github: https://github.com/brandoncantilang/ChessAI

Lines Of Code: 3287. Project Components: 8 classes of code, Unity, and AWS Beanstalk

Total man Hours = 334

Project Team

Role	Name	Major responsibility
Team leader	Brandon Cantilang	Delegate and communicate
Team members	Brandon Cantilang	Documentation
	Salem Adams	Developer
	Nick Rath	Developer
	Michael Barber	Documentation
Advisor / Instructor	Sharon Perry	Facilitate project progress; advise on project planning and management.

Table of Contents

Introduction & Overview

AI-Chess has been successfully completed and implemented to the liking of the group members for SP-14 (Blue). This project is implemented through the Unity platform and currently runs on a hosted webserver for anyone to enjoy. The implementation of a chess engine was challenging and took time to get right and is only further complicated when introduced to artificial intelligence and online webservices.

Communication between our team was important and allowed us to communicate the exact specification we desired. The Chess game is in good development and has been implemented onto a server hosting website “netlify”. The primary focus of Chess-AI was a fun and interactive experience for users to access directly and start having fun immediately.

The Unity engine allows for the capability of our code to be run across multiple platforms such as Mac and Windows. The capability for multiple operating systems to run Unity software was a major consideration for this project and successfully fulfills our group's goal of having an accessible chess game.

For Input/processing power, the game solely focuses on the user’s mouse inputs for dictating the state of the game and has a powerful AWS backend to allow the artificial intelligence to make calculated moves, depending on the user’s choice of difficulty.

The overarching goal of Chess-AI is the access of anyone to learn, play, and get better at chess, which has been accomplished through our website with a tab for meeting the development team, learning the basic rules of chess (rules tab), and finally the third tab where the user can start a match against another user on the PC or the artificial intelligence.

Requirements

This project's objective is to develop a fully functioning chess engine that offers users the ability to play versus another player or an AI. This AI will be capable of running on different difficulties, leaving it up to the user’s decision on what type of experience they wish to have.

An update to our website will release the experimentation of our group's artificial intelligence and the P2P gameplay logic is set. Although some movements may seem by mistake, many features of this game follow logic and does not allow players to have absolute control. The features that reduce control are limited to gameplay events such as “check” or “checkmate”, where the game will only allow the current player in control to try and win the game instead of doing a useless move that will result in a defeat, regardless.

Definitions & Acronyms

- Artificial Intelligence (AI)

    - Regarding our project, AI is the technology that lets the engine play the Chess game like a human. The AI will be responsible for analyzing and making decisions in the game as if they were their own person.
- User Interface (UI)
    - User interface is the visual element that the users will be interacting with to play the game.
- Web Graphics Library (WebGL)
    - WebGL is the API that we will be using to display the chess game onto the website.
- Frames Per Second (FPS)
    - A measure of the individual frames being show in on second of a game

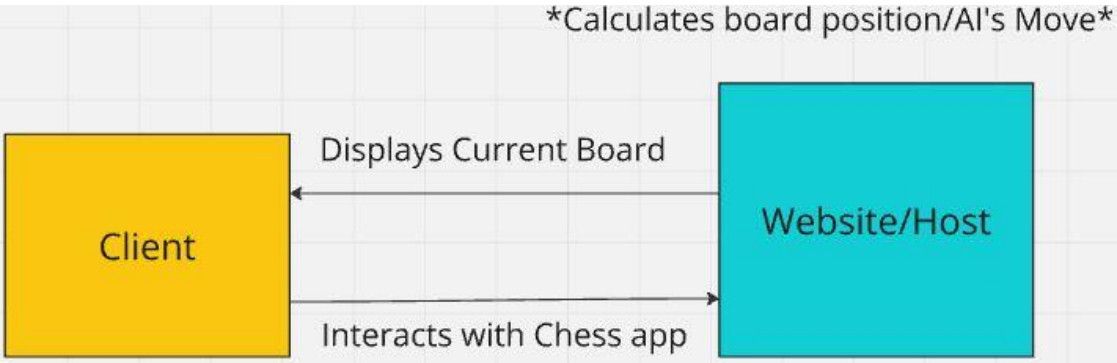
These definitions are useful in the understanding of our objective for this project. The artificial intelligence will mimic human interaction and is nearing completion for the scope of this project. A user interface has been developed and consists of areas on the screen that the user is directed to click on movable pieces, website tabs, and other UI areas such as choosing a difficulty. WebGL displays the game for the user to interact with and lastly, FPS is important for user retention, as low frames per second (possible with the large amount of the AI's calculations) would lead to a miserable experience for both new and returning players.

Design and Architectural Drawings

Chess-AI was constructed in Unity for the initial draft and implementation and then integrated into an online webservice environment. The website's host takes code uploaded from the chess application and runs it continuously for anyone to access. Web hosting takes place automatically without the intervention of any group members and should continue to run until either stopped or access to the server runs out. Our server hosts a main page that describes the group members and project, a rules section to explain the lesser-known aspects of chess for new players, and finally a third tab to play against a friend on the same PC.

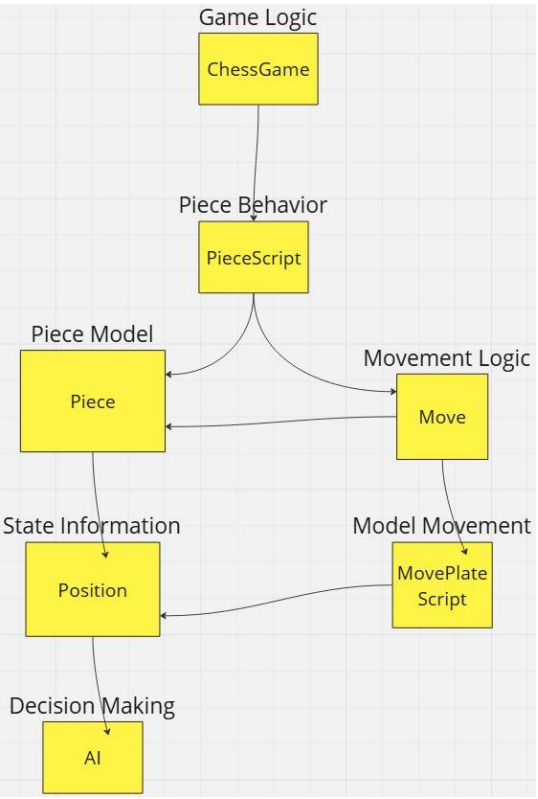
Diagrams will be shown below to visualize the implementation of our code and how they interact to repeatedly display the current board's state and the backend decision making of our Unity program.

Back-end logic is provided through our classes "PieceScript.cs", "piece.cs", "MovePlateScript.", "Move.cs", "AI.cs", "ChessGame.cs", "Position.cs" interacts to fulfil the interactions between the user, game, and AI.



(Figure 1, Hosting on AWS using Beanstalk)

AWS Elastic Beanstalk is provided through Amazon Web Services and gives deployment and management capabilities to developers through online hosting services. This service allows for the implementation and display of many coding languages like Python, Java, and containers for C# programs. Using AWS allows for a reliable service that ensures scalability for high traffic, application monitoring, and load balancing.



(Figure 2, Chess-AI's Model)

Game Management (ChessGame)

Classification: Type: Class | Subsystem: Game Management

Purpose: The core of the chess game, managing overall game state, board positions, piece initialization, and player turns.

Responsibilities:

- Initializes the board at the start of the game.
- Manages the current state of the game.
- Manages player turns and move validation.
- Maintains the internal representation of the chessboard as 2D array of chess piece objects.
- Interfaces with both the user and player to execute moves.

Interactions:

- Contains PieceScript to handle behavior and store positions.
- Interacts with MovePlateScript to handle potential move UI elements.
- Interacts with ChessAI to get the AI's move in the position.

Constraints:

- The board positions and turns must be maintained accurately to prevent illegal moves or state inconsistencies.

Resources:

- Manages a 2D array of game objects that represent the board, holding references to the pieces. It also manages an array for each player's pieces.
- Processing time includes move validation, AI interaction, and updating the game state.

Piece Behavior (PieceScript)

Classification: Type: Class | Subsystem: Piece Behavior

Purpose: Handles the logic related to the individual pieces like move generation and board interaction.

Responsibilities:

- Encapsulates the behavior and properties of each type of piece.
- Generates pseudo-legal moves based on the current board state and piece type.
- Validates moves considering rules like check.
- Manages UI interaction for piece drag/drop.

Interactions:

- Interacts with ChessGame directly to update board positions and validate moves. based on where a piece is moved.
- Utilizes MovePlateScript to display legal moves on the board.
- Provides legal move arrays for ChessAI to use.

Constraints:

- The class must maintain accurate board positions for each piece.
- Pieces should only move on their turn.
- Coordinates are integers with a 0-7 range for standard 8x8 chessboard.

Resources:

- Manages arrays for legal and pseudo-legal moves.
- Processing time includes move generation.
- Class must synchronize to ChessGame.

User Interface (MovePlateScript)

Classification: Type: Class | Subsystem: User Interface

Purpose: Manages the display and interaction of a piece's potential moves.

Responsibilities:

- Create object to represent possible move tiles for selected piece.
- Adjusts the visual appearance of move plates when they indicate an attack on another piece.

Interactions:

- ChessGame initializes move plates where moves are legal and destroys them when a piece is not being held.

Constraints:

- The move plates must appear and disappear quickly to maintain a smooth UI and user experience.
- The move plates must appear a different color (red) when representing a capture.
- The move plates must disappear after a move is made.

Resources:

- Instances of MovePlateScript are created and destroyed by player/AI interactions. The memory overhead must be considered.

Move Data (Move)

Classification: Type: Class | Subsystem: User Interface

Purpose: Represents a single move in the game. Encapsulates all necessary data including the piece making the move, target coordinates, and whether the move is an attack.

Responsibilities:

- Encapsulate move data.

Interactions:

- ChessGame initializes move plates where moves are legal and destroys them when a piece is not being held.

Constraints:

- A Move object must accurately represent a valid move. A move off the board should not exist.
- The Move class assumes that it will only be used in context of valid game states.

Resources:

- A Move object requires minimal memory, but in deep search algorithms large numbers of Move objects will be instantiated.

Exports: See page 13

Artificial Intelligence (ChessAI)

Classification: Type: Class | Subsystem: Artificial Intelligence

Purpose: Provides automated decision making, determining the best moves based on the current game state

Responsibilities:

- Analyzes the current board state to generate the best moves.
- Implements algorithms like Minimax and Alpha-Beta Pruning to evaluate moves.
- Adjusts its strategy based on the difficulty level set.

Interactions:

- Receives the current board state from ChessGame.
- Provides the chosen move to play to ChessGame.

- Uses PieceScript to evaluate possible moves.

Constraints:

- The AI must make moves in a reasonable time frame for a good user experience.
- The AI must only make legal moves including rules like check, checkmate, stalemate.

Resources:

- May require significant memory resources to store possible moves, board evaluations, and results of recursive algorithms like Minimax.
- Move generation may be processor-intensive, especially at higher difficulty where the searches are at higher depth.
- synchronized with ChessGame to avoid conflicts like AI trying to make a move out-of-turn.

VERSION CONTROL

Version control for our project was done through Unity Collaborate, which is a built-in feature of the Unity editor. This allowed our team to collaborate on the same project simultaneously while ensuring smooth development and integration.

Our team worked on this project together by uploading and synching changes directly through this platform, keeping all members updated with the newest version of the project. Unity Collaborate keeps track of all changes, whether it be changes to scripts, scenes, and more while also storing these changes in history in case the team needed to revert to any past versions.

When it comes to collaboration, Unity Collaborate flags the program when there are two developers on at the same time. If this occurs, the users must determine manually how to resolve the conflict by requiring one of them to choose which version should be kept or input the changes themselves. After changes have been made to the program, developers commit updates in order to keep the rest of the team up to date with any new changes. Overall, Unity Collaborate worked smoothly, providing easy version control and real-time collaboration within Unity, making it easier to manage our chess engine.

Development

Development for SP-14, Chess-AI has been interactive and fun. Our group met through the connection we all have with games and were talking about other possible ideas for projects, such as a budget app. The idea of building a Chess engine intrigued us, as all of us have a good bit of knowledge with chess. The initial problem we were facing was that the project also required our team to develop an artificial intelligence opponent, capable of playing at three different levels, beginner, intermediate, and advanced. Despite having little to no experience with AI, our team decided to accept the challenge and take on the chess project.

Our objectives for the project are simple. We are tasked with building a fully functional chess engine that offers users the ability to play player versus player and player versus AI. The AI should have three difficulties, beginner, intermediate, and advanced. Our goal is to not only give users an engaging and interactive experience, but to also provide users with a challenging experience where they can choose an AI's difficulty to match their skill level. Also, we decided to create our own website which we can host our game on. This way, users have the ability to access it on any web browser, to be used in a pass and play setting.

During the early stages of our project, we were left thinking how we wanted to tackle the project. Regarding the chess engine itself, the basic mechanics were pretty straightforward. We would be responsible for creating the chess pieces and establishing their move sets, adding complex moves such as en passant and castling, different game states such as check and checkmate, and establishing player turns.

With that being said, our decision for AI needed a bit more research before we continued to develop it. After some research, we decided it was best to go with the MiniMax algorithm to determine the decision making for our AI. For the chess pieces, each piece has its own designated value, the higher the value, the more important the piece is. Our job was to develop the AI to learn these values and make decisions based off this. For our different difficulty levels, we dove into the MiniMax tree and decision trees to further refine our AI. This gives our AI the ability to look past the initial moves and start looking deeper in the future to evaluate their decision making. The AI's available moves begin taking account for not only their moves, but the possible moves available for the users, giving more evaluation values for their decision making.

For users to access our game on any browser, we decided to host our own website and embed our game on it. We will be doing this with the use of a Unity plugin named WebGL. On our website, users will be able to find information about every team member, rules and details of the game, and finally the option to play the game. The goal for our website is to provide users with an appealing and organized user interface capable of hosting our game, giving users the ultimate experience.

When we first started developing a plan for our project, we decided to set specific milestones to act as due dates for our program. These milestones included setting up the board, getting a fully functioning game completed where checks and checkmate is established, creating an AI, creating different difficulties for the AI, embedding the game on our website, and lastly

add any refining touches before we publish our final product. These milestones have guided us throughout the whole semester as we've held weekly status meetings to give updates and address any challenges we've faced since the last meeting.

Risk Analysis

There are very minor risks involved with this project, there is no sensitive data that the team deems desirable enough that can be stolen from the website for any serious security concerns. The breadth of our program does not involve the scope of having millions of players, rigorous AI standards, downtime expectations, performance risks, or security risks. Our website could evolve into a multifaceted game with leaderboards, a community tab, and a community competitive scene. Later risk assessments would have been undertaken, but the only current risk is through the ddos of webhosting, which should be circumnavigated by AWS Beanstalk.

Test Plan and Test Report

To ensure a safe and reliable user experience, we will be using input validation to prevent wrong data from being processed. Error handling is also required in case any system issues or failures pop up and end up causing crashes, ruining the user's gaming experience.

Our Chess game will be designed to handle different aspects of system performance effectively. The memory usage needs to be optimized to make sure the game will run as efficiently as possible. Also, the processing power must also be under control to avoid system overload while also balancing the game logic and graphics. Lastly, the game should be properly managed as it should be fully capable of running different commands simultaneously such as user input and game updates without running into any issues.

We want all users to have an easy and enjoyable experience. To help with that, our UI will be accessible and have clear controls. To enhance the game, there will be helpful feedback, such as possible rules or hints to help players. The Chess engine should also be responsive, fitting to all screens and resolutions. The customizable options such as display and difficulty, should make for an amazing experience for users.

Possible Tests	Pass/Fail	Summary
Online Web Access Test	Pass	Users on mobile and desktop platforms have full access to the website and can play the game.
Gameplay Responsiveness Test	Pass	The game is playable and comes to an end
AI Intuitiveness Test	Pass	
Stable Connection Test	Pass	Network is reliable and consistent

Gameplay Stability	Pass	Game runs smoothly without interruptions
Consistent Turns	Pass	Turns are established correctly
Winning Move	Pass	Checkmates and stalemates
Win Screen	Pass	Winning screen displays correctly
Pawn Promotion	Pass	Piece promotion successful

Conclusion/Summary

The AI-Chess (Blue) team has come very far in the development of an online chess experience. The dedication to completing this project was quite impressive and ties this entire project into what our future will look like. Development is a team effort and requires knowledge and expertise in a wide range of fields and such, requires a diverse set of characters to fully encompass the required knowledge and the knowledge yet to be gained. Useful tools are at our disposal for either very cheap or likely free, these tools such as Unity, AWS, and Programming tools make programming a challenging but not impossible task. Chess-AI will not be perfect, and this can be said for many programs around the world, such as pre-released video games that may contain bugs but are still enjoyed by millions of people for what the game is. This project has given insights into the creation of websites, sharing of information on GitHub, collaboration with team members, and most importantly the knowledge that with time and effort, a team who believes in their abilities can produce a fun and memorable product.

Appendix

System Requirements

1.0 Introduction

- Creating an AI capable of interacting with a user to play a complete chess game on different difficulties creates an opportunity for a fun and engaging chess experience. Our

project aims to create an autonomous chess engine with different difficulty settings, making the experience enjoyable for anyone.

1.1 Overview

- This project's objective is to develop a fully functioning chess engine that offers users the ability to play versus another player or an AI. This AI will be capable of running on different difficulties, leaving it up to the user's decision on what type of experience they wish to have.

1.2 Project Goals

- The goals for our project are to develop an AI capable of playing a chess game, create a user interface which users can use to play the game, and implement player versus player and player vs Ai all while adhering to the official chess rules. We will also be aiming to embed this game onto a website, making it possible for users to access the game through any web browser.

1.3 Definitions and Acronyms

- Artificial Intelligence (AI)
    - Regarding our project, AI is the technology that lets the engine play the Chess game like a human. The AI will be responsible for analyzing and making decisions in the game as if they were their own person.
- User Interface (UI)
    - User interface is the visual element that the users will be interacting with to play the game.
- Web Graphics Library (WebGL)
    - WebGL is the API that we will be using to display the chess game onto the website.
- Frames Per Second (FPS)
    - A measure of the individual frames being show in on second of a game

1.4 Assumptions

- It is assumed that the engine will be fully compatible across all operating systems such as MacOS and Windows
- Users will have a stable internet connection to access and play the game.
- Users will be using a mouse as the input device to play the game.
- End users will be able to grab somewhat of an understanding of how to play chess.

2.0 Design Constraints

2.1 Environment

- The game must be compatible with different web browsers, such as Google Chrome or Safari.

- For users to access the website, they must have a stable internet connection.
- The user experience should be the same across all platforms such as MacOS and Windows.

2.2 User Characteristics

- The overall design for the engine should be user-friendly and easy to navigate. This way, user tech savvy or not will be able to properly navigate the menu and play the game without any confusion.
- Because of the AI development in the chess engine, the game will cater to users of all skill levels.

2.3 System

- The chess engine should be compatible with different operating systems.
- The game should have a short response time to make the game experience fluid.
- Memory and storage should be used most efficiently to refrain from any slowdowns involving processing the game.

3.0 Functional Requirements

3.1 Design

- User interface design – With the game being built on Unity, the game will have a user interface that users will use their mouse as input to click the different options of the game. First, there will be the welcome screen that gives the player the option to start a game. After selecting this option, they can select their opponent, either AI or another player. Once their opponent is selected, if AI was the chosen opponent, they'll be asked to select the level of difficulty and which color they would like to play as. After hitting start game, the creation of the chessboard and pieces will show up. Along the bottom of the screen also be 3 different tabs, one for settings, one for rules, and one showing the game status.
    - Play game
    - Game selection and settings
    - Interactive chessboard and pieces
    - Setting and rules
    - Game over

3.2 Graphic

- The most important aspect of an online chess game is the board itself. The avoid any confusion about which piece is which, the chess board and pieces must all be distinctive. Below are a few examples of the graphic representations that we will be using to help the user experience.
    - Interactive chessboard
    - Chess pieces
    - Move indicators
    - Game status indicator

3.3 Operating System

- Because this game will be operating on a website, it must be fully functional on different operating systems, and it must also be able to run smoothly on different browsers. We must also manage the system's resources to avoid high CPU and memory given the integration and size of the engine we're creating.

4.0 Non-Functional Requirements (use if applicable)

4.1 Security

- To ensure a safe and reliable user experience, we will be using input validation to prevent wrong data from being processed. Error handling is also required in case any system issues or failures pop up and end up causing crashes, ruining the user's gaming experience.

4.2 Capacity

- Our Chess game will be designed to handle different aspects of system performance effectively. The memory usage needs to be optimized to make sure the game will run as efficiently as possible. Also, the processing power must also be under control to avoid system overload while also balancing the game logic and graphics. Lastly, the game should be properly managed as it should be fully capable of running different commands simultaneously such as user input and game updates without running into any issues.

4.3 Usability

- We want all users to have an easy and enjoyable experience. To help with that, our UI will be accessible and have clear controls. To enhance the game, there will be helpful feedback, such as possible rules or hints to help players. The Chess engine should also be responsive, fitting to all screens and resolutions. The customizable options such as display and difficulty, should make for an amazing experience for users.

System Design

1. Introduction and Overview

SP-14 System Design Document details goals for the proposed project and attempts to encapsulate all possible details that contribute to the creation of AI-Chess. The SDD can be read and followed by developers to understand the concept of AI-Chess and how the project has been implemented. This document will be updated throughout the software development life cycle and will have higher and lower-level concepts explained.

2. Design Considerations

2.1. Assumptions and Dependencies

Unity Engine

This project assumes the use of Unity as the primary development platform for its rendering and input management capabilities. It is dependent on Unity's support for C# features and libraries for the game and AI logic. Changes in Unity's API or C# support could affect the project.

Cross-platform Compatible:
The project is built with the assumption that it will run on different operating systems like Windows and macOS. It is assumed Unity's cross-platform build tools will provide support for these operating systems. The project depends on the operating systems' stability.

Input/processing power
This project assumes the player will use a standard input device of a mouse. It assumes the system running the game has sufficient processing power to handle the AI calculations and graphics of the game.

End-user characteristics
It is assumed that end-users have a basic understanding of chess rules and gameplay. The project assumes that players will have varied skill levels, and the selection of difficulty will cater to a wide range of chess players. The project's success depends on user feedback for updates and enhancements.

Scaling
There may be time to implement AI learning from user behavior, online play, etc. The project depends on a codebase that can accommodate future changes without rewriting, as much as possible.

2.2. General Constraints

-Chess has a very large quantity of possible moves a player can make, so the AI must be able to calculate moves without major performance issues.
-The game will be played on multiple platforms; there must be consistent experiences between them.
-The chess AI may need to interact with other software such as external engines for performance testing.
-The user will not have time to let the AI think forever. It must make best judgement moves within a reasonable time frame.

2.3. Development Methods

AI-Chess is under Agile development, where collaboration between programming, research, and general development is vital for overall success. The agile approach uses iterative development to break the project into smaller, more manageable iterations (sprints). This is well-suited for chess AI because there are small, functional increments that can be continuously improved. It allows for incremental delivery of features, where each iteration builds upon the previous one.

We considered other development approaches but found agile to be the best suited for our project.

3. Architectural Strategies

Unity/C#

We chose to develop the project in Unity, which uses C# as the primary programming language. Unity was chosen for its 2D game development and cross-platform capabilities. C# is perfect for chess game development due to its object-oriented features and integration with the .NET framework. While Unity combined with C# is powerful and versatile, it introduces a learning curve for our team, which has little to no experience with it.

Reuse

The project leverages Unity's built-in components for handling game objects, physics, and user interface elements. This eliminates the need for custom implementation for basic functionalities, which will save lots of time in development.

Future plans

The game is designed with a modular architecture where different components are developed as different modules. It can introduce some complications with managing dependencies between modules, but the long-term maintenance and scalability is worth the trade-off.

UI Paradigms

The user interface is designed using Unity's Physics tools with a drag-and-drop functionality for moving chess pieces. This interaction model mimics physical chess play. Although the game is in 2D, the sprites selected for the pieces look 3D. The pieces always layer properly by row. Other UI plans include sounds upon making a move and shadows under the pieces when they are being dragged. Our chess engine will be integrated into a webpage where users can play just by navigating to our website. By embedding the game as a WebGL application, the website will be able to properly run the chess engine.

Error Handling

The system will include robust error detection to ensure that the game can recover from unexpected states or inputs. This will improve the user experience.

Memory Usage

The game will be playable on high-end and low-end devices. Memory usage needs to be carefully optimized, and memory leaks must be avoided. Object pooling, lazy initialization, and other techniques will optimize memory usage.

External Databases and Data Storage/Management

The game uses local storage to save game states and user preferences. We may implement an external database of openings to give the AI more variation in the beginning of the game.

4. System Architecture

The system was decomposed into five subsystems to achieve a clear separation of concerns, where each part of the system focuses on a specific aspect of the game functionality. Separating the game management and piece behavior from the UI system allows for easy updates without

affecting the game logic. The AI itself is encapsulated within ChessAI to build the AI logic without impacting other parts of the system. This system will also allow for different game modes, difficulty levels, etc.