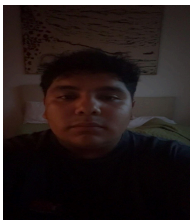
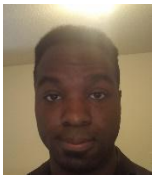
SP-14 Build Chess Game Using AI

CS 4850 - Section 01 – Fall 2024

Oct 15, 2024

 Giovanni Zavala Documentation	 Joshua Smith Team leader
 Yehong Huang Developer	 Mike Tokura Documentation

Team Members:

Name	Role	Cell Phone / Alt Email
Joshua Smith	**(Team Lead)** Developer	229-942-6880 jsmith09112000@gmail.com
Giovanni Zavala	Documentation	404.493.7402 gzavala1@students.kennesaw.edu
Mike Tokura	Documentation	678.485.9252 mtokura@students.kennesaw.edu
Yehong Huang	Developer	706-672-7330 hyh159263@gmail.com
Sharon Perry	Project Owner or Advisor	770.329.3895 Sperry46@kennesaw.edu

INDEX

1. INTRODUCTION

AI chess Game

This is a very basic chess game with an AI implementation using the GAN (Generative Adversarial Network), which was built using Python.

To begin the game, a player must run the AI_GAME.py file, providing the two game modes: “player versus player” and “player vs. computer,” in which the AI in the latter game mode implements the GAN against the player.

The computer will simulate every possible move it has available, and for every one of those moves, it will also play every possible move by its opponent. To run the simulation the computer always assumes that the player plays the move in their best interest. The return value contains (piece, move) tuples with the highest score.

Project modules:

Board:-

The Board class consists of a simple 2D list to store the chess pieces and several required methods like make_move(), unmake_move() (which is necessary for the minimax algorithm), methods to check if the game is finished, and an evaluation method that returns the total score of the caller’s piece minus the total score of the opponent’s piece (also required for AI).

Chess Piece:-

The Chess Piece class is abstract and is used as a parent for every piece. It consists of variables for the name of the piece, the initial starting position variable (Concerning the coordinates on the chessboard matrix), and abstract methods for each piece’s movement patterns and capture methods

some methods to keep the previous state of the chess piece intact (required by the AI when calling unmake_move()).

We will use a PGN file that contains over 3000 instances of online chess games. The file includes a detailed list of all the moves made in each game and metadata such as the date, time, and players involved.

The chess library will be designed to parse the PGN file, enabling the execution of all the moves on a chessboard and converting the information into a format that our AI opponent can process.

This data can then be transformed into an array with a shape of (8, 8, 12) using the following steps: First, a blank 8x8 matrix is created to represent the chessboard. Since there are 6 types of pieces on each side in chess, 12 different values are used to represent the pieces, with the array filled with zeros except for one value indicating the piece occupying each square. It's important to note that only Magnus Carlsen's moves are recorded, and this can be achieved using the metadata to filter the data. After collecting the data, it needs to be organized into a set of X and Y values for training the GAN. The X values will represent the initial state, and Y will represent the resulting board after Magnus Carlsen's move; the encoding of a move is more complex than encoding the board itself due to factors such as multiple pieces potentially moving to the same square. It is important to note that with regards

to the rules of the game, we designed the rules such that so long as the next movement position of Every chess piece that is close to the opposing team's king piece does not overlap with all of the next movement positions of said team's king, so the game will continue to run properly.

2. SYSTEM ANALYSIS

2.1 Existing System

1. board: This module defines the chess board and its properties. It contains the logic for placing pieces, checking for legal moves, and determining the game's outcome.
2. Graphics: This module displays the chess board and accepts user input. It also handles the game loop, updates the board, and accepts user input. It also handles the game loop and updates the board state based on user input.
5. Main: This is the program's entry point, which initializes the chess board and starts the game loop after the user chooses the game mode

The AI chess game uses object-oriented programming concepts to model the chess piece and board. The GAN is implemented to evaluate the best move for the AI player. The graphics module uses the Pygame library to display the chessboard and accept user input. The game loop updates the board state based on user input and AI's move.

2.2 Limitation of the Existing System

Computational complexity: The game of chess has an enormous number of possible moves, making it computationally expensive to explore all possible moves and their outcomes.

It's essential to assume that all the information required to make the best move is contained within the board itself. However, this assumption is slightly flawed as the order of the opponent's moves could provide insights into their strategy, potentially leading to missing information for the GAN to make accurate decisions. An example of this would be if the player were to make a move that was completely different from the AI's perceived movement patterns based on what it learned from the PGN file, which could ultimately lead to the AI taking longer amounts of time to decide the next best move to make.

Limited ability to learn: While AI chess systems can improve through self-play or by learning from human games, they still have limited ability to learn from experience and adapt to new situations.
Limited ability to understand the game: the AI chess system has limited ability to understand the game, the motivations of the players, and the subtle nuances of the game.

1. Chess Board: This component will represent the chess board, its squares, and pieces. The board will have a matrix representation and will be able to determine the legal moves for a piece.

2. AI Engine: This component will implement the Generative Adversarial Network to determine the best move for the AI player. The depth of the search
tree can be adjusted to control the difficulty level of the game.
3. User Interface: This component will provide an interactive interface for the user to
Play the game. The user will be able to move the pieces and the AI will respond with
its move.
4. Logging: A logging component will be added to record the moves made by both the
user and the AI, which will help understand the thought process of the AI
and in debugging the code.

2.3- Feasibility Methods

Technical Feasibility
The AI chess game includes the implementation of a chess board, game rules, and a GAN algorithm for AI decision-making.
The following libraries are required to run the code:
graphics: A library used to display the chess board and pieces on a graphical user
interface.
“Itertools” is a library used to group items in an iterable object.
Board: A custom library that implements the chess board and game rules.
Theoretically, the code should be feasible as it is written in Python, a widely-used, high-level programming language that is easy to learn and implement. The code uses object-oriented programming concepts and has been written in a structured and organized manner, making it easy to understand and modify.
However, the graphics library used in the code is not a standard Python library, and its installation and setup may require additional steps. Additionally, the code may require further optimization and fine-tuning to improve the performance and AI decision-making capabilities.
The operational feasibility of the AI chess project depends on
several factors, including the following:
1. Computing Resources: The proposed system requires a computer with sufficient
processing power to run the code and handle computation-intensive tasks
involved in playing a game of chess.
2. Availability of Required Libraries: The code requires the following libraries to run:
‘graphics’, ’Itertools’, ’numpy’, ‘logging’, and ‘Board’. These libraries must be
installed on the system and accessible to the code to run the game.
3. User Familiarity with the Code: The proposed system requires users to be familiar
with the code and the logic behind the AI's decision-making process to play
the game effectively.

4. Depth of AI General Adversarial Network: The code enables the AI to implement the GAN, which allows it to understand the best possible moves to make against the opponent with respect to the player’s choice and chess game data collected in the PGN file.

In conclusion, the operational feasibility of the AI chess project depends on the accessibility of the required resources, the user's familiarity with the code, and the chosen game mode.

3. SYSTEM DESIGN

3.1 Input Design

The proposed system satisfied the following input design objectives:
Initialization of the game: The game can be initialized in two modes, i.e., player versus player or player vs. AI. The game mode can be selected by passing the argument "game_mode" to the class "Board" and setting it to either “0” for player versus player or "1.” for player vs. AI.
Placing pieces on the board: The pieces can be placed on the board by calling the method "place_pieces()" on the board object. The method will initialize the starting positions of all the pieces for both players or AI on the board.

User input: The player can choose which piece to move and where they want it to go (with respect to the specific movement position said piece can move next) while playing the game. The graphics interface can be used to provide this input.
The user can engage with the game and perform motions with ease thanks to the input design. Additionally, the input is checked to make sure that only legitimate moves are accepted and that incorrect ones are not considered and ignored

3.2 Output Design

The outputs of the proposed AI chess game system would be the visual representation of the chess board, moves made by the players or AI, and the result of the game (checkmate, draw, or resign). The graphical representation of the chess board would be created using the graphics library. The moves made by the players or AI would be displayed on the chess board and updated in real time. The result of the game would be displayed in a message box or through some other means of notification.

4. SYSTEM TESTING

Software testing is the most important stage of specification design and coding review and is a crucial component of software quality assurance. Program modules are tested individually first, then "bundled" modules are tested as a whole. When a program module is interfaced with progressively larger programs up to the system test level, it may work flawlessly when it is isolated. The accuracy and dependability were ensured using the testing techniques listed below:

4.1 UNIT TESTING

When creating the classes for the chess piece objects, we created and tested methods for their specific movement patterns to make sure they moved across the board in the correct positions every time.
We made sure to test the capture methods for each chess piece object to make sure the pieces can capture the opposing team's chess pieces if the user and AI decide to do so. We made sure each piece's attack patterns corresponded with their specific movement patterns and also implemented special attack conditions for the pawn pieces.
We made sure to test the loop for the win condition of the game such that it checks for the "check" and "checkmate" conditions for both teams' king pieces based on their current position, their next available movement positions, and whether they can be captured on the next turn after each player and AI's respective turns.

4.2 Validation TESTING

We created "Barrier methods" for each chess piece so that any piece that the player and AI opponent want to move does not have an available position outside the matrix board's boundaries. We Tested the special conditions for the movement and attack pattern methods for the pawns; for the movement methods, on the pawn's first turn, it can move up to two spaces forward on the board and only one block forward every other turn. Its attack method can only capture the opponent's chess piece if they are either northwest or northeast of the pawn's current condition, and we kept in mind that the pawn can still have the option to move forward if it chooses not to attack.

5. SYSTEM IMPLEMENTATION

Once our chess AI is implemented, it‘s crucial to test its performance and analyze its strengths and weaknesses by:

Playing against it ourselves and observing its decision-making process

Pitting it against other chess engines or human players of varying skill levels

Measuring metrics such as search depth, evaluation accuracy, and move selection time

Analyzing its play style and identifying areas for improvement

By thoroughly testing and refining our chess AI, we can create a formidable opponent that challenges and engages players of all levels.

6. CONCLUSION

In conclusion, developing an AI chess game involves a complex combination of programming skills, game design, and artificial intelligence techniques. The game must be designed to be both challenging and entertaining, while also offering good replay value. We ensured the AI algorithms used were efficient, and the user interface was intuitive and visually appealing. Additionally, the game must be secure, scalable, and easily upgradeable over time to meet the evolving needs of users. By considering these factors and carefully planning the development process, it is possible to create a successful and enjoyable AI chess game that can provide hours of entertainment and challenge for players.