

**Tribhuvan University**

**Faculty of Humanities and Social Science**

**A Project Report on**

**CHESS WITH MINIMAX ALGORITHM**

**Submitted to**

Faculty of Humanities and Social Sciences

*In partial fulfillment of the requirements for the Bachelors in Computer Application*

**Submitted by:**

Arjal Shah

Dipendra Adhikari

Under the Supervision of

Govinda Gautam



**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Asian College of Higher Studies**

**Supervisor’s Recommendation**

I hereby recommend that this project prepared under my supervision by Govinda Gautam entitled **“CHESS with Minimax Algorithm”** in partial fulfillment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SIGNATURE**

Govinda Gautam

**SUPERVISOR**

Humanities and Social Science

Asian College of Higher Studies

A star with a symbol in the middle

Description automatically generated

**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

**Asian College of Higher Studies**

**LETTER OF APPROVAL**

This is to certify that this project prepared by Arjal Shah and Dipendra Adhikari entitled “CHESS with Minimax Algorithm” in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In our opinion

|  |  |
| --- | --- |
| ------------------------------------------- **SIGNATURE of Supervisor**  Govinda Gautam  Supervisor  Asian College of Higher Studies  Jawlakhel,Lalitpur | ------------------------------------------- **SIGNATURE of HOD/ Coordinator**  Coordinator  Asian College of Higher Studies  Jawlakhel,Lalitpur |
| -------------------------------------------- **SIGNATURE of Internal Examiner**  **Internal Examiner** | --------------------------------------------- **SIGNATURE of External Examiner**  **External Examiner** |

**Abstract**

CHESS is a classic two-player strategy game played on a square board divided into 64 alternating black and white squares, known as a chessboard. Each player begins with 16 pieces, including one king, one queen, two rooks, two knights, two bishops, and eight pawns. The objective is to checkmate the opponent's king, meaning it is placed in a position where it cannot escape capture. Each type of piece moves in specific ways: the king moves one square in any direction, the queen can move any number of squares vertically, horizontally, or diagonally, the rooks move along rows and columns, the knights move in an "L" shape, the bishops move diagonally, and the pawns advance forward but capture diagonally. In modern adaptations, chess can be played against another player or against an AI, which can be implemented using the minimax algorithm to simulate strategic decision-making.

***Keywords:*** *AI, CASE, OOP*

**Acknowledgments**

I would like to express my gratitude to supervisor, Mr. Govinda Gautam for assisting in learning about chess systems for this project “**CHESS** **with** **Minimax Algorithm**”. They provided us with the opportunity to work on this fantastic project, which has allowed me to learn a lot of new things, and I am grateful to my professors for that.

Apart from my supervisor, I would like to thank my professors and friends for allowing me to use the available equipment for this project.

**Project Members**

Arjal Shah

Dipendra Adhikari

**Table of Contents**

[Chapter 1 Introduction 1](#_Toc148273588)

[1.1 Introduction 1](#_Toc148273589)

[1.2 Problem Statement 1](#_Toc148273590)

[1.3 Objectives 2](#_Toc148273591)

[1.4 Scope and Limitation 2](#_Toc148273592)

[Chapter 2 Background Study and Literature Review 3](#_Toc148273593)

[2.1 Background Study 3](#_Toc148273594)

[2.2 Literature Review 3](#_Toc148273595)

[Chapter 3 System Analysis and Design 6](#_Toc148273596)

[3.1 System Analysis 6](#_Toc148273597)

[3.1.1 Requirement Analysis 6](#_Toc148273598)

[3.1.2 Feasibility Analysis 7](#_Toc148273599)

[3.1.3 Object Modelling 8](#_Toc148273600)

[3.2 Algorithm 9](#_Toc148273601)

[Chapter 4 Implementation and Testing 14](#_Toc148273602)

[4.1 Implementation 14](#_Toc148273603)

[4.1.1 Tools used 14](#_Toc148273604)

[4.1.2 Implementation Details of Modules 15](#_Toc148273605)

[4.2 Testing 17](#_Toc148273606)

[4.2.1 Test cases for Unit Testing 17](#_Toc148273607)

[4.2.2 Test cases for System Testing 18](#_Toc148273608)

[Chapter 5 Conclusion 20](#_Toc148273609)

[5.1 Conclusion 20](#_Toc148273610)

[5.2 Future Recommendation 20](#_Toc148273611)

[REFERENCE 22](#_Toc148273612)

[Appendicx 23](#_Toc148273613)

**List of abbreviations**

CASE : Computer Aided Software Engineering

UI : User Interface

OOP : Object Oriented Programming

**List of figures**

Figure 3.1: Use Case Diagram of CHESS…………………………………………….….6

Figure 3.2: Class Diagram of Chess………………………………………………………8

[Figure 3.3: Activity Diagram of Chess 9](#_Toc182127814)

**List of Tables**

Table4.2-1UniTesting........................................................................................................14

# CHAPTER I:INTRODUCTION

## 1.1 Introduction

## The "Chess" project is focused on developing a digital version of the classic board game, where players can compete either against another person or a computer (AI) opponent. This digital version includes all the essential rules and functions of chess, ensuring that players can move their pieces correctly, capture other pieces, and even use special moves like castling (a unique king and rook move) and en passant (a specific pawn capture). The game offers two playing options: one is to play against another human, and the other is to play against an AI opponent. The AI is built using the Minimax Algorithm, which helps the computer analyze possible moves and choose the best one, making the game challenging for the player. The main goal of chess is to "checkmate" the opponent's king, meaning the king is in a position to be captured and has no escape. The game can also end in a draw, either through a stalemate (when a player has no legal move but isn’t in check) or other conditions like insufficient pieces to win. [1]

## Chess is a game known for its complexity and strategic depth, allowing endless possibilities for players to learn, improve, and explore new strategies. It has a rich history, enjoyed by both casual players and serious competitors alike, from friendly games to professional tournaments and world championships. In the next parts of this report, we will dive deeper into how the game is designed, the key features it includes, and how it functions. We'll also discuss the technical details of how this digital chess game is being implemented and outline the overall goals and achievements of the project so far, along with plans for future improvements.

## 1.2 Problem Statement

The primary challenge is to develop an advanced AI-powered chess system aimed at enhancing the chess-playing experience for individuals across various skill levels. While existing chess programs and online platforms provide basic gameplay, they often fall short in several key areas that could improve user engagement and learning. For example, many platforms do not offer real-time feedback or suggestions, making it difficult for players to assess threats, strategize, and explore alternative moves effectively during gameplay. Additionally, the depth of analysis on chess moves and strategies is often limited, reducing the potential for players to understand different approaches and improve their skills. This project seeks to address these shortcomings by enabling players to start games with the standard board setup while providing features that offer real-time assistance and in-depth move analysis. These enhancements would help players make better decisions, learn faster, and enjoy a more competitive and strategic chess experience.

## 1.3 Objectives

The objectives of the CHESS game are as follows:

* To generate a working chess engine.
* To implement an AI system through which you can play with a computer.
* To insert chess rules into virtual chess to make it more feasible.

## 1.4 Scope and Limitation

The scope of the project includes:

* Implementing a graphical user interface (GUI) for the chess game.
* Allowing players to choose between playing against another human or an AI opponent.
* Implementing move validation based on the rules of chess.
* Supporting special moves such as en passant, and pawn promotion.

The limitation of this project includes:

* The system does not support two player to play in different devices.
* The system may not exhibit the same level of strategic thinking as human players of higher skill levels.
* The performance of the AI opponent can vary based on the hardware capabilities of the device running the game

# CHAPTER II:BACKGROUND STUDY AND LITERATURE REVIEW

## 2.1 Background Study

Chess is a game that has been played for hundreds of years, making it one of the most ancient and respected strategy games in history. Its origins trace back to northern India around the 6th century, where it was initially called "Chaturanga." The game spread westward through Persia, where it was known as "Shatranj," and eventually reached Europe, where it evolved into the modern chess we play today. As the game moved across different cultures, the rules were refined, and by the 15th century, chess became a more dynamic game, leading to its widespread popularity. Now, chess is played by millions of people worldwide, from casual players to professional competitors. The game is played on an 8x8 board made up of 64 squares, alternating in black and white. Each player starts with 16 pieces: one king, one queen, two rooks, two knights, two bishops, and eight pawns, all of which move in specific ways. The primary objective of chess is to checkmate the opponent’s king, which means putting the king in a position where it is under threat of capture and cannot escape. [2]

**C**hess is known for its intellectual depth and complex tactics, making it a game of strategy, critical thinking, and foresight. Players must think several moves ahead, anticipate their opponent’s responses, and adjust their plans based on the ever-changing position on the board. The game involves many elements such as piece development (moving your pieces into effective positions), managing pawn structure (arranging pawns in a way that strengthens your defense or attack), coordinating pieces to work together, making sacrifices (giving up pieces for a bigger advantage), and mastering endgame techniques where fewer pieces remain, and the goal is usually to force checkmate. Over the centuries, chess has attracted legendary players, such as Garry Kasparov and Bobby Fischer, whose strategies have shaped how the game is played. In modern times, computers and artificial intelligence have had a huge influence on chess, from AI-powered engines like Stockfish to famous matches between humans and machines. These advancements are key in developing digital chess games, where AI can simulate different levels of human-like play, providing a challenging experience for users. Understanding the rich history of chess, its strategic and tactical layers, and the role of AI is essential for creating well-rounded and engaging chess projects, especially in a digital format.

## 2.2 Literature Review

Susac et al. defined design and implementation of an autonomous digital chess board. Proposed approach combines low-cost elements with an array of Hall-effect sensors controlled by a microcontroller. Con to proposed approach is in possibility to detect only piece presence and not its type and color. Which is more complex in the 2023 era. [3]

Mahmood et al. defined the design of electronic chess board prototype which is able to recognize chess pieces and its position on the chessboard. Some improvements have been done such as adding the digital chess clock interfacing and using another conductive material to improve the contact between chessboard and chess pieces. It is also capable to display the move of chess pieces history on the chess program. The move can be saved for future reference. This chess program still needs an improvement in terms of connecting to the real digital chess clock instead of using a prototype of chess clock. The cost of this prototype is roughly about RM500 which is considered cheaper than the one in the existing market. [4]

Fogel et al. made this engine, where the flaws of machine learning are overcome. Since credit assignment system has its flaws, the evolutionary system is more efficient. The evolutionary algorithm uses three artificial neural networks to evaluate the worth of potential positions in the chessboard that are alternate. All three players start with neural networks and play in random variation to evaluate the fittest of them all through the quality of the moves. Then, the fittest players generate the off spring for the next generation. This algorithm is very useful as the evolution of a high-level chess program suggests broader applications to problem solving and also gives us the possibility to attempt solving problems to which known solutions do not exist. [5]

Tong Lai Yu et al used the single threaded chess engine Beowolf in this project. Platform independent SDL libraries are used inorder to make it more efficient so as to incorporate graphics and other media into the game. Beowulf searches a game tree using the Negamax Search algorithm, which is a simply a variation of the minimax search that accounts the zero sum game. The game tree is simplified by Alpha-Beta pruning. The 2D graphics and sound is added with the help of SDL libraries that are written in C/C++. The 3D graphics are implemented with the help of open source tools such as Blender 3D. Thus, this paper establishes the fact that a chess engine with amazing graphics, or any other grid game can be created with the help of open source tools. [6]

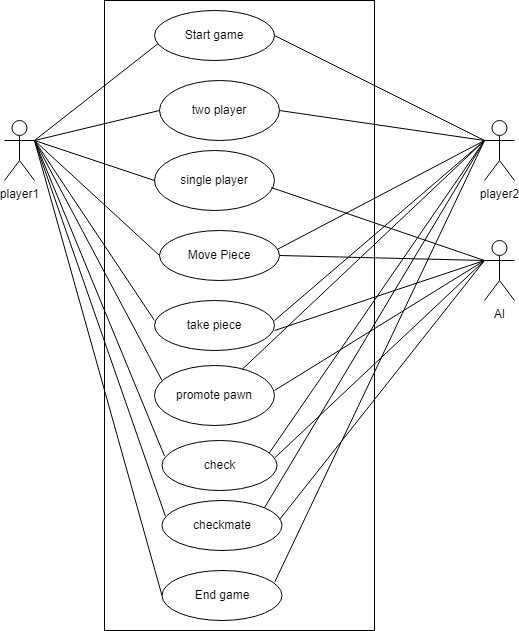
# CHAPTER III:SYSTEM ANALYSIS AND DESIGN

## 3.1 System Analysis

### 3.1.1 Requirement Analysis

Functional requirements

Figure 3.1 1Use Case Diagram of CHESS



Here with the above Use Case Diagram of CHESS we can understand that the user have two options after they start the game. The two options are single player and two player. The two player is for human vs. human and the single player is for human vs. AI.

Non-functional requirements

Non-functional requirement is a specification that describes the system’s operation capabilities and constraints that enhance its functionality. Some of the non-functional requirements are:

* Performance:   
  The app should provide a responsive user interface, with quick response times for user interactions.
* Usability:   
  The app should have an intuitive and user-friendly interface, with clear instructions and easy navigation.
* Reliability:   
  The app should be stable and not crash or freeze during normal usage scenarios. The app should handle errors gracefully, providing informative error messages to users when necessary.
* Error Handling and Logging:

The app should provide appropriate error messages to users, indicating the nature of the error.

### 3.1.2 Feasibility Analysis

Following feasibilities were studied before building the system to see if the system could be built with exact requirements in the expected time.

1. Technical feasibility

In order to design the system, it uses existing technologies, software and hardware hence there’s no technical problem that might arise to build this system.

1. Economic feasibility

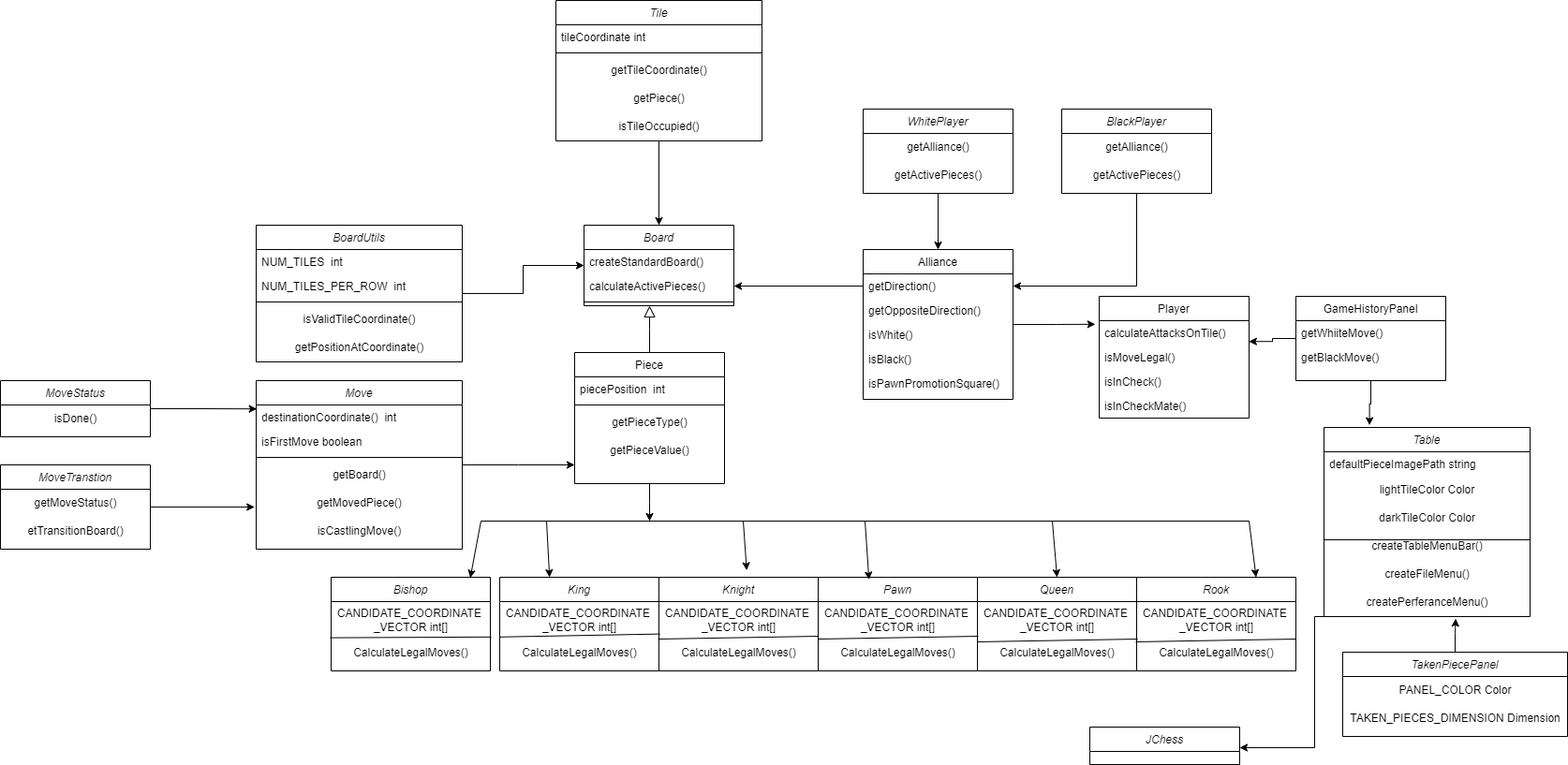
The system does not require extra software and hardware i.e it uses open-source technologies. So there is no recurring cost.

1. Operational feasibility
2. The system uses simple technologies to design. So it is user friendly.
3. Legal feasibility

The system is completely legal and should not face any problem regarding illegal issues.

### 3.1.3 Object Modelling

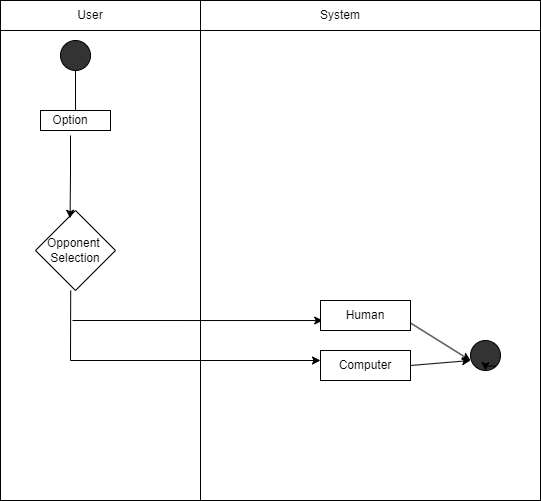
Figure 3.2 1Class Diagram of Chess



The above shown figure 3.2 represents class diagram of our Chess game. The box heading displays the class name whereas the other section are the variables declared and the methods implemented inside the class. The JChess class is the main class which shows the executed program. The classes like Piece, Player, Move and Tile are abstract classes which other classes extends to use their functions. For example the classes like king, Queen, Knight, Pawn, Rook, Bishop extend the class Piece. The GUI classes are GameHistoryPanel, Table and TakenPiecePanel.

### 3.1.4.Activity Diagram

Figure 3.3: 1 Activity Diagram of Chess



The figure 3.3 represents Activity Diagram of Chess game. In this figure we can see the activity process on the app .As user login, we can see the option to select the opponent .The opponent can be either human or computer.After selecting the opponent we can play game.

## 3.2 Algorithm

**Minimax Algorithm**

In the "CHESS" report, the minimax algorithm is utilized to explore the game tree, considering all possible moves and their consequences. The goal is to maximize the player’s own advantage while assuming that the opponent will also make optimal moves. It alternates between maximizing and minimizing positions to find the best move.

The Minimax Algorithm is a widely used approach in chess AI to create intelligent opponents. It is a recursive search algorithm that explores the game tree to evaluate different move options and select the best move for the AI player. The algorithm begins by generating a game tree, representing all possible moves and their resulting positions. Starting from the current position, it explores each legal move by applying it to the board and generating subsequent positions for the opponent. At the leaf nodes of the game tree (i.e., terminal positions or a specified depth), an evaluation function is used to assign a score or value to each position. The evaluation function measures the relative strength or desirability of a position for the AI player. It considers factors such as piece values, pawn structure, piece activity, king safety, control of the center, and other strategic and tactical elements. The Minimax principle forms the core of the algorithm. The AI player assumes that the opponent will make optimal moves, aiming to minimize the AI's outcome. The AI player maximizes its own score at each level, considering that the opponent will choose moves that minimize the AI's score. The algorithm recursively evaluates positions by alternating between maximizing and minimizing steps. It moves up the game tree, propagating the best move values from the leaf nodes to the root node. At each level, the algorithm selects the move with the highest score when it is the AI player's turn and the move with the lowest score when it is the opponent's turn.

By using the Minimax Algorithm, chess AI opponents can make informed decisions based on evaluating different move options and anticipating the opponent's responses. The algorithm allows the AI to consider multiple moves ahead, enabling strategic and tactical decision-making in the game of chess.

To understand how the algorithm has been implemented in the system an explanation is given below

Public Move execute(Board board) {  
 final long startTime = System.*currentTimeMillis*();  
 Move bestMove = null;  
 int highestSeenValue = Integer.*MIN\_VALUE*;  
 int lowestSeenValue = Integer.*MAX\_VALUE*;  
 int currentValue;  
 System.*out*.println(board.currentPlayer() + "THINKING with Depth = " +this.searchDepth);  
 int numMoves = board.currentPlayer().getLegalMoves().size();  
 for (final Move move : board.currentPlayer().getLegalMoves()){  
 final MoveTransition moveTransition = board.currentPlayer().makeMove(move);  
 if(moveTransition.getMoveStatus().isDone()){  
 currentValue = board.currentPlayer().getAlliance().isWhite() ?  
 min(moveTransition.getTransitionBoard(), this.searchDepth - 1) :  
 max(moveTransition.getTransitionBoard(), this.searchDepth - 1);  
  
 if(board.currentPlayer().getAlliance().isWhite() && currentValue >= highestSeenValue){  
 highestSeenValue = currentValue;  
 bestMove = move;  
 }else if(board.currentPlayer().getAlliance().isBlack() && currentValue <= lowestSeenValue){  
 lowestSeenValue = currentValue;  
 bestMove = move;  
 }  
 }  
 }  
 final long executionTime = System.*currentTimeMillis*() - startTime;  
 return bestMove;  
}

This is one of the method which is used to implement the algorithm. It is crucial for the performance of the Minimax algorithm. It should be designed to accurately reflect the game state and be computationally efficient. This method initiates the Minimax algorithm. It iterates through all legal moves of the current player, evaluates the moves using the min and max methods, and selects the best move based on the evaluation scores. The variable searchDepth represents the depth to which the algorithm explores the game tree. It also explains that white is the maximizing player while black is the minizing player. It further explains that if the you are white and the currentvalue is lower than highestSeenValue than the highestSeenValue will become the currentvalue and if the player is black and currentValue is higher than lowestSeenValue than the lowestseenValue becomes the current value.

public int min(final Board board, final int depth){  
 if (depth == 0 || *isEndGameScenario*(board)){  
 return this.boardEvaluator.evaluate(board, depth);  
 }  
 int lowestSeenValue = Integer.*MAX\_VALUE*;  
 for(final Move move : board.currentPlayer().getLegalMoves()){  
 final MoveTransition moveTransition = board.currentPlayer().makeMove(move);  
 if (moveTransition.getMoveStatus().isDone()){  
 final int currentValue = max(moveTransition.getTransitionBoard(), depth -1);  
 if(currentValue <= lowestSeenValue){  
 lowestSeenValue = currentValue;  
 }  
 }  
 }  
 return lowestSeenValue;  
}

This is the min method which is one of the two helping method of execute method. This method represents the minimizing part of the Minimax algorithm. It recursively explores the opponent's moves and returns the lowest score achievable for the current game state. If the depth parameter reaches 0 or an end-game scenario is detected, the board's evaluation score is returned. A recursive call is made to max method to explore the opponents move one level deeper in the game.

public int max(final Board board, final int depth){  
 if (depth == 0 || *isEndGameScenario*(board)){  
 return this.boardEvaluator.evaluate(board,depth);  
 }  
 int highestSeenValue = Integer.*MIN\_VALUE*;  
 for(final Move move : board.currentPlayer().getLegalMoves()){  
 final MoveTransition moveTransition = board.currentPlayer().makeMove(move);  
 if (moveTransition.getMoveStatus().isDone()){  
 final int currentValue = min(moveTransition.getTransitionBoard(), depth -1);  
 if(currentValue >= highestSeenValue){  
 highestSeenValue = currentValue;  
 }  
 }  
 }  
 return highestSeenValue;  
}

This is similar to the min method with the only difference that instead of lowestSeenValue there is highestSeenValue. Also this method recursive call min method to explore its opponents move.

# CHAPTER IV :IMPLEMENTATION AND TESTING

## 4.1 Implementation

### 4.1.1 Tools used

**i. CASE TOOLS**

For the designing and implementation of the system code the following CASE tools were used:

1. Intellij IDE

Intellij IDEA is a popular integrated development environment (IDE) developed by JetBrains. It is specifically designed for Java development, but it also supports a wide range of programming languages, including Kotlin, Groovy, Scala, and more.

b) Draw.io

Draw.io was utilized as a case tool for designing and creating visual representations, such as system flow diagrams, ER diagrams etc. during system analysis and design phase.

**ii. Programming languages**

For coding purpose, we used the following programming languages for our project:

a) Java

Java programming language was used for the backend development of Student Performance Analyzer web application. It provided the foundation for implementing core functionalities such as data processing, decision tree algorithm, web Servlets and JSP.

### 4.1.2 Implementation Details of Modules

Board class:

* whitePlayer(): Returns the white player object.
* blackPlayer(): Returns the black player object.
* currentPlayer(): Returns the current player object (either white or black) who has the next move.
* getEnPassantPawn(): Returns the pawn that can be captured en passant (if there is one). En passant is a special chess move.
* getBlackPieces(): Returns a collection of all active black pieces on the board.
* getWhitePieces(): Returns a collection of all active white pieces on the board.
* calculateLegalMoves(): Calculates and returns a collection of all legal moves for a given collection of pieces. It iterates through the pieces and calls the calculateLegalMoves method on each piece to obtain its legal moves.
* calculateActivePieces(): Calculates and returns a collection of active pieces for a given alliance (either white or black). It iterates through the tiles on the board and checks if each tile is occupied by a piece of the specified alliance.
* getTile(final int tileCoordinate): Retrieves and returns the tile at a specific board coordinate.
* createGameBoard(final Builder builder): Creates and initializes the chessboard based on the configuration provided by the Builder. It creates an array of Tile objects, populates them with pieces according to the board configuration, and returns the resulting list of tiles.
* createStandardBoard(): Creates a standard chessboard configuration with the starting positions of all pieces. It returns a Board object representing the initial state of a chess game.
* getAllLegalMoves(): Returns an iterable collection of all legal moves available to both white and black players on the current board. It combines the legal moves of both players into a single iterable.
* setEnPassantPawn(Pawn enPassantPawn): Sets the en passant pawn.

Move class:

* MajorMove: Represents major piece non-attack moves. Overrides methods for equality and provides a custom string representation.
* AttackMove: Represents generic attack moves (captures). Provides methods for calculating hash code, checking equality, and determining whether it's an attack move.
* PawnMove: Represents pawn non-attack moves. Provides a custom string representation.
* PawnAttackMove: Represents pawn attack moves (captures). Provides a custom string representation.
* PawnEnPassantAttackMove: Represents pawn en passant attack moves. Overrides the execute method to handle en passant captures.
* PawnJump: Represents pawn initial double-step moves. Overrides the execute method to handle en passant capture scenarios.
* CastleMove: An abstract class representing castling moves. Contains fields and methods common to both king-side and queen-side castling moves.
* KingSideCastleMove: Represents king-side castling moves. Overrides methods for equality and provides a custom string representation.
* QueenSideCastleMove: Represents queen-side castling moves. Overrides methods for equality and provides a custom string representation.
* PawnPromotion: Represents pawn promotion moves, which are pawn moves that result in promotion to another piece (usually a queen).
* MoveFactory: A factory class responsible for creating moves based on the current board configuration and coordinates.

Piece class:

* getPieceType(): Returns the type of the chess piece (e.g., PAWN, KNIGHT, BISHOP).
* getPiecePosition(): Returns the current position of the chess piece on the chessboard.
* getPieceAlliance(): Returns the alliance (color) of the chess piece (e.g., WHITE or BLACK).
* isFirstMove(): Indicates whether the chess piece has moved before. This is used for handling specific rules like pawn double-step moves and castling.
* getPieceValue(): Returns the relative value of the chess piece. This value is used to evaluate the strength of the piece in the context of chess tactics and strategies.
* movePiece(): It returns a new piece that represents the result of the move. Different pieces have different move logic, so this method is specific to each piece type.
* calculateLegalMoves(): An abstract method that should be implemented by subclasses. It calculates and returns a collection of legal moves that the piece can make on the given chessboard (Board).

Table class:

* createTableMenuBar():Creates the menu bar for the chess game interface, including options for new games, preferences, and exiting the game.
* createFileMenu():Creates the "File" menu, which contains options for starting a new game and exiting the application
* createPreferenceMenu():Creates the "Preferences" menu, which includes options for flipping the board and highlighting legal moves.
* createOptionsMenu():Creates the "Options" menu, which allows users to set up the game parameters.
* updateGameBoard(final Board board): Updates the game board with the provided board state.
* updateComputerMove(final Move move): Updates the computer's move.
* BoardPanel (): Represents the chessboard panel in the GUI. Handles the drawing of the board and its tiles.
* MoveLog (Inner Class): Manages the list of moves made during the game.

Provides methods to add, remove, clear, and retrieve moves.

## 4.2 Testing

### 4.2.1 Test cases for Unit Testing

*Table 4.2-1 Unit Testing*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case description** | **Test step** | Test Data | **Expected result** | **Actual result** |
| Valid Move | 1. open the chess game  2. check if the destination position is occupied or not.  3. if the position is not occupied, then make move.  4. if the position is occupied and the piece is opponent's piece, then make move. | piece position= c2  destination position = c4  player should be able to make a move. | As Expected | As Expected |
| Invalid Move | 1. Start the game.  2. Make a move that doesn't fulfill the valid move condition | piece position= c2  destination position = c6 | player shouldn't be able to perform the move | As expected |
| Check | 1. Start the game.  2. Make a move to make the opponent's king under threat of capture in the next move. | king position = c6  bishop position = e4 | player should be able to perform escape move | As expected |

### 4.2.2 Test cases for System Testing

Scenario 1: Check checkmate

* Make a move.
* Check if the performed move can put opponent's king under threat of capture in the next move.
* check if there is a chance for the king to escape from the threat
* if there is no escape move then you win by checkmate.

Scenario 2: Check special move

* + - Check if the move is one of the special move such as en passant and pawn promotion.
    - Perform those moves if the requirement is fulfilled.

# 

# CHAPTER V:CONCLUSION

## 5.1 Conclusion

In conclusion, the Chess Game Project represents a comprehensive and engaging implementation of the classic board game. With its inclusion of all the basic features expected in a chess game and the integration of artificial intelligence, it offers players a rich and challenging gaming experience.

The project's basic features, such as the chessboard layout, piece movement rules, and capture mechanics, provide a faithful representation of the traditional game, ensuring that both beginners and experienced players can enjoy it. Additionally, the user interface is intuitive and user-friendly, making it accessible to a wide audience.

The standout feature of this project is its AI capability. The inclusion of artificial intelligence not only allows for solo play against a computer opponent but also enhances the game's replay ability. This feature not only promotes skill development but also keeps the game engaging and enjoyable over time.

## 5.2 Future Recommendation

To enhance the Chess Game Project that uses the Minimax algorithm, consider the following future recommendations:

* Improved User Interface:

Enhance the user interface to provide a more visually appealing and user-friendly experience. Implement features such as customizable themes, piece animations, and interactive tutorials for beginners.

* AI Enhancements:

Optimize the Minimax algorithm by implementing alpha-beta pruning or other advanced techniques to increase the AI's efficiency and reduce computation time.

Introduce more sophisticated evaluation functions for the AI to make it even more challenging and capable of recognizing tactical and strategic nuances.

Allow players to customize the AI's playing style and difficulty level, making it adaptable to various skill levels.

* Online Multiplayer:

Implement an online multiplayer mode that allows players to compete with others over the internet. This feature would significantly expand the game's player base and provide a more dynamic and competitive experience.

**REFERENCE**

|  |  |
| --- | --- |
| [1] | R. Khan and .. I. J. Kesavan, " R. Design and Development of Autonomous Chess Playing Robot," Innov. Sci. Eng. Technol. , 2014. |
| [2] | R. Duda and P. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures. Commun. ACM," 1972. |
| [3] | F. H. Susac, "Digital Chess board based on array of Hall-Effect sensors," 2017. |
| [4] | N. Mohmood, "Low cost electronic chess set for chess tournament," 2011. |
| [5] | H. a. Q. Fogel, "A self Learning Evolutionary Chess Program," 2004. |
| [6] | T. L. Yi, "Chess Gaming and Graphics using Open-Source Tools," 2009. |

# Appendices

