SPL-1 Project Report

Man vs Machine : A Checkers showdown With Minimax

Submitted by

Md. Kibria Hossen Roni

BSSE Roll No. : 1430

BSSE Session: 2021-2022

Submitted to

Dr. Zerina Begum

Professor

Institute of Information Technology

University of Dhaka

Institute of Information Technology

University of Dhaka

17-12-2023

Project : Man vs Machine :

A Checkers showdown With Minimax

Author : Md. Kibria Hossen Roni

Submission Date: 17 December, 2023

Supervised by: Dr. Zerina Begum

Professor

Institute of Information Technology

University of Dhaka

Supervisor's Signature : _____

Acknowledgement:

I extend my heartfelt gratitude to Dr. Zerina Begum, my esteemed professor, for her invaluable guidance and support throughout the development of the Checkers Game Project. I am thankful for the collaborative and intellectually stimulating environment she cultivated, fostering both creativity and academic growth. Her mentorship has been instrumental, and I appreciate the opportunity to contribute to the fascinating realm of AI and gaming under her tutelage.

Table of Contents:

1. Introduction

In the realm of artificial intelligence and gaming, the development of intelligent agents capable of competing against human players has been an intriguing and challenging pursuit. This project undertakes the creation of a checkers game with a dual focus—providing a platform for human vs. human interaction and implementing varying levels of artificial intelligence (AI) for human vs. AI gameplay. The aim of this endeavor is to explore different AI strategies and algorithms, ranging from basic random move policies to more sophisticated approaches like game tree analysis and the minimax algorithm.

The project is divided into two primary modules: human vs. human gameplay and human vs. AI gameplay. The latter module further breaks down into three distinct sub-modules, each representing a different level of AI difficulty—Beginner, Intermediate, and Expert. This tiered approach enables a nuanced exploration of AI capabilities and strategies, providing players with a diverse and challenging gaming experience.

In the Beginner level of AI, a straightforward random move policy is employed, allowing for a simple yet unpredictable opponent. Moving on to the Intermediate level, the implementation of a game tree analysis becomes evident, with a focus on optimizing jump moves—a critical aspect of strategic play in checkers. Finally, in the Expert level, the project delves into the complexities of the minimax algorithm, reaching depths of 5-8 layers to simulate a more intelligent and strategic AI opponent.

This report delves into the design, implementation, and evaluation of each module, offering insights into the decision-making processes behind the AI models and the overall gaming experience. Through this exploration, it is aimed to contribute to the understanding of AI in gaming, shedding light on the challenges, successes, and future possibilities within this dynamic and evolving field.

2. Background study

Some terminologies come with the project to describe the project and its working principles. Some of these terminologies and their little introduction given below:

- Checkers Game:

  Checkers, a classic board game known for its strategic depth, has captivated players for centuries. The game involves moving pieces diagonally across the board, capturing opponent pieces through jumps, and ultimately aiming to reach the opponent's back row for making king and then get control over opponent. Understanding the rules and dynamics of Checkers is fundamental to appreciating the complexities involved in designing an AI capable of competing at different skill levels.

- Artificial Intelligence in Games:

The integration of artificial intelligence into gaming has been a transformative force, enhancing player experiences and challenging human intellect. Various AI techniques have been applied to board games, with a focus on decision-making processes, strategic planning, and adaptability. Key concepts such as heuristic evaluation, game tree analysis, and the minimax algorithm have played pivotal roles in creating intelligent game agents.

❖ Random Move Policies:

The utilization of random move policies in AI, especially in beginner-level opponents, introduces an element of unpredictability. This approach serves as a foundation for understanding basic AI behavior and provides a benchmark for more advanced strategies.

❖ Game Tree Analysis:

Game tree analysis is a method employed in intermediate-level AI opponents. It involves examining potential moves and their consequences, creating a branching structure that represents possible game states. By focusing on jump moves, the AI can better navigate the intricate decision tree, enhancing its strategic capabilities.

❖ Minimax Algorithm:

The minimax algorithm is a cornerstone in the development of expert-level AI opponents. Rooted in decision theory, this algorithm aims to minimize the possible loss for a worst-case scenario while maximizing potential gains. Implementing the minimax algorithm in Checkers involves simulating multiple future moves, evaluating the resulting game states, and choosing the optimal move.

- Evaluation Function:

  The heart of the minimax algorithm lies in the evaluation function. This submodule assigns a numerical value to each potential game state, reflecting the desirability of that state for the AI player. Factors considered may include piece count, kinged pieces, board position, and control of the center. A well-crafted evaluation function is pivotal for the algorithm to make informed decisions.

- Depth-Limited Search:

  Given the exponential nature of the game tree, exploring all possibilities to the end of the game is computationally expensive. The depth-limited search submodule restricts the algorithm to a certain depth, determining how many moves ahead the AI should simulate. Balancing depth and computational resources is crucial to achieving a reasonable compromise between accuracy and efficiency.

- Iterative Deepening:

Iterative deepening is a submodule that involves repeatedly applying the minimax algorithm with increasing depths. This approach allows the AI to explore the most promising branches first, gradually refining its understanding of the game. Iterative deepening is particularly valuable in dynamic environments, providing adaptability to varying game complexities.

- Depth-First Search (DFS):

  The minimax algorithm inherently relies on a depth-first search strategy to traverse the game tree. DFS explores each branch to its deepest level before backtracking, allowing the algorithm to systematically analyze potential moves and outcomes.

- Backtracking:

  Backtracking is a crucial aspect of the minimax algorithm, enabling the AI to reconsider its choices and explore alternative paths. When traversing the game tree, backtracking occurs when the algorithm returns to a previous decision point to explore other possibilities, facilitating a comprehensive search for the optimal move.

- Alpha-Beta Pruning:

  Alpha-beta pruning is a sophisticated technique employed to further optimize the minimax algorithm. This submodule reduces the number of nodes evaluated in the game tree by eliminating branches that cannot possibly affect the final decision. By maintaining alpha and beta bounds, the algorithm intelligently prunes the search space, significantly speeding up the decision-making process.

Understanding these submodules, including depth-first search and backtracking, within the minimax algorithm provides a holistic view of the intricacies involved in creating an expert-level AI for Checkers. The combination of these components enables the AI to make strategic decisions effectively, navigating the vast decision space inherent in complex board games.

3. Project Overview

The implementation of the Checkers Game Project involved a systematic and iterative process, integrating both human vs. human and human vs. AI gameplay modules. Below is a detailed description of the key components and methodologies employed in bringing the project to fruition:

- ❖ Game Structure and Rules:

The project commenced with the establishment of the fundamental game structure and rules. The Checkers board, comprising an 8x8 grid, was implemented along with the standard checkers pieces. The rules governing piece movement, captures, and king promotions were carefully coded to ensure adherence to the traditional Checkers gameplay.

★ Firstly, initiating the board with appropriate pieces and empty spaces

```
void initiateBoard()
{
    for (int row = 0; row < BOARD_SIZE; row++)
    {
        for (int col = 0; col < BOARD_SIZE; col++)
        {
            if ((row + col) % 2 && row < 3)
            {
                checkerBoard[row][col] = BLUE_PIECE;
            }
            else if ((row + col) % 2 && row > 4)
            {
                checkerBoard[row][col] = RED_PIECE;
            }
            else
            {
                checkerBoard[row][col] = EMPTY;
            }

        }
    }
}
```

Fig-01: (Board initiation)

★ Getting move by mouse click from user

```
Move getMove()
{
    Move move;
    int x, y;
    getMouseClick(x, y);
    move.fromRow = y / SQUARE_SIZE;
    move.fromCol = x / SQUARE_SIZE;
    getMouseClick(x, y);
    move.toRow = y / SQUARE_SIZE;
    move.toCol = x / SQUARE_SIZE;

    return move;
}
```

Fig-02: (Getting move as input)

★ Then checking is taken move is valid or not. If the move is not valid then give some more chances. Here is a screenshot of depicting partial of the function

```
bool isValidMove(Move move)
{
    int fromRow = move.fromRow, fromCol = move.fromCol, toRow = move.toRow, toCol = move.toCol;
    if (toRow < 0 || toRow >= BOARD_SIZE || toCol < 0 || toCol >= BOARD_SIZE)
    {
        return false;
    }

    char piece = checkerBoard[fromRow][fromCol];
    if (piece == EMPTY || (redTurn && piece != RED_PIECE && piece != RED_KING) || (!redTurn && p
    {
        return false;
    }

    if (checkerBoard[toRow][toCol] != EMPTY)
    {
        return false;
    }

    int rowDiff = abs(toRow - fromRow), colDiff = abs(toCol - fromCol);
    if (rowDiff == 1 && colDiff == 1)
```

Fig-03: (Partial view from move validation checking)

★ Implementing a valid move

```
void makeMove(Move move)
{
    int fromRow = move.fromRow, fromCol = move.fromCol, toRow = move.toRow, toCol = move.toCol;

    char piece = checkerBoard[fromRow][fromCol];
    checkerBoard[fromRow][fromCol] = EMPTY;
    checkerBoard[toRow][toCol] = piece;

    if (abs(toRow - fromRow) == 2)
    {
        int jumpedRow = (fromRow + toRow) / 2;
        int jumpedCol = (fromCol + toCol) / 2;

        eattenPieces.push_back(checkerBoard[jumpedRow][jumpedCol]);
        checkerBoard[jumpedRow][jumpedCol] = EMPTY;
    }

    if ((piece == RED_PIECE || piece == BLUE_PIECE) && ((redTurn && toRow == 0) || (!redTurn && toR
```

Fig-04: (Making move)

★ Promoting the piece into king if the piece arrive to the last row of opponent's side

```
void promote(int row, int col)
{
    if (checkerBoard[row][col] == BLUE_PIECE)
    {
        checkerBoard[row][col] = BLUE_KING;
    }
    else if (checkerBoard[row][col] == RED_PIECE)
    {
        checkerBoard[row][col] = RED_KING;
    }

}
```

Fig-05: (Promotion to king)

❖ Human vs. Human Module:

The human vs. human module served as the foundation, providing a platform for two players to engage in a traditional game of Checkers. The implementation involved designing a user-friendly interface to facilitate player moves and maintain game state. User inputs were validated to ensure compliance with the game rules, and the game loop iteratively updated the board until a winner emerged or the game ended in a draw.

```
if(redTurn)
{
    cout<<"Red turn : "<<endl;
}
else
{
    cout<<"Blue turn: "<<endl;
}

Move move;
do
{
    move = getMove();
} while (!isValidMove(move));
makeMove(move);
```

Fig-06: (Human vs. Human)

❖ AI Integration - Beginner Level:

The initial foray into AI implementation focused on the Beginner level, employing a simple random move policy. The AI randomly selected legal moves, creating an unpredictable opponent for human players. This stage allowed for the seamless integration of AI into the existing framework, laying the groundwork for more sophisticated levels.

```
int random;
vector<Move> blueValidMoves;
collectBlueValidMoves(blueValidMoves);
srand(time(0));
random = (int)(rand() % blueValidMoves.size());
move = blueValidMoves[random];
makeMove(move);
printBoard();
printGraphics();
```

Fig-07: (Human vs Beginner AI)

❖ AI Integration - Intermediate Level:

Advancing to the Intermediate level, the project delved into game tree analysis to enhance the AI's decision-making capabilities. Special emphasis was placed on identifying and prioritizing jump moves, as these are integral to strategic play in Checkers. The AI's ability to analyze

potential future game states was refined, providing a more challenging experience for human players.

❖ AI Integration - Expert Level with Minimax Algorithm:

The pinnacle of AI implementation involved incorporating the minimax algorithm into the Expert level. The evaluation function was carefully crafted to consider various factors influencing the desirability of a game state. Depth-limited search, alpha-beta pruning, transposition tables, quiescence search, and iterative deepening were sequentially integrated to optimize the algorithm's efficiency and decision-making accuracy.

```
vector<Move> blueValidMoves;
collectBlueValidMoves(blueValidMoves);

int maxVal = -INF;
int value;
for (int i = 0; i < blueValidMoves.size(); i++)
{
    makeMove(blueValidMoves[i]);
    value = minimax(depth + 1, height, alpha, beta, !isBlue);
    undoMove(blueValidMoves[i]);
```

Fig-08: (Human vs Expert AI-Blue)

```
vector<Move> redValidMoves;
collectRedValidMoves(redValidMoves);

int minVal = INF;
int value;
for (int i = 0; i < redValidMoves.size(); i++)
{
    makeMove(redValidMoves[i]);
    value = minimax(depth + 1, height, alpha, beta, !isBlue);
    undoMove(redValidMoves[i]);
```

Fig-09: (Human vs Expert AI-Red)

4. User Interface

❖ Graphics.h Integration:

The User Interface (UI) of the Checkers Game Project was implemented using the graphics.h library, providing a graphical representation of the game board and pieces. The integration of graphics.h facilitated the creation of a visually appealing and interactive environment for players.

Fig-10: (Main menu using graphics.h)

Fig-11: (Menu Under Human vs. AI using graphics.h)

❖ Game Board Representation:

The game board, an 8x8 grid, was rendered on the graphical interface using graphics.h primitives. Each square on the board was visually distinguished, and the grid provided a clear spatial reference for players to make their moves. Colors were carefully chosen to enhance visibility and aesthetics.

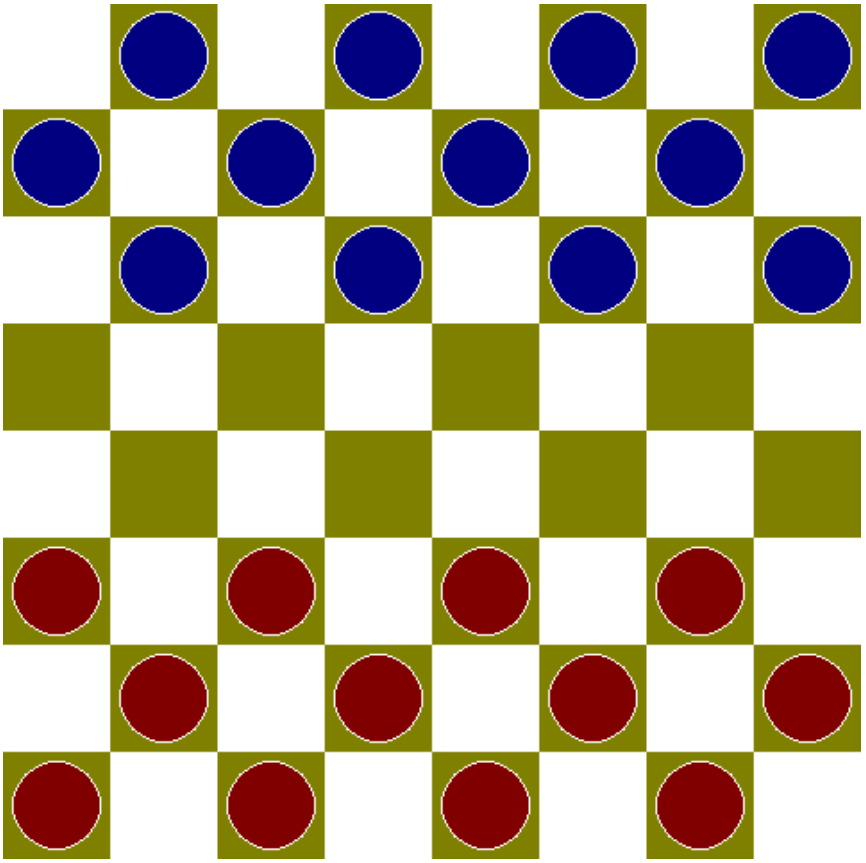

Fig-12: (Board and Piece representation using graphics.h)

- Piece Rendering:

  Checkers pieces, both regular and kinged, were visually represented on the board using graphics.h. Each type of piece was assigned a distinct color or shape, contributing to a visually intuitive representation of the game state. The dynamic rendering of pieces allowed players to track the progression of the game seamlessly.

- Player Input Handling:

  Player input was managed through graphics.h, capturing mouse clicks or to determine player moves. The graphical interface was responsive, providing real-time feedback to players as they interacted with the board. Input validation mechanisms ensured adherence to game rules, preventing illegal moves.

- Game State Display:

  The current state of the game, including turn information, captured pieces, and potential win scenarios, was dynamically displayed on the graphical interface. Clear and concise information ensured that players remained informed about the game's progress without causing cognitive overload.

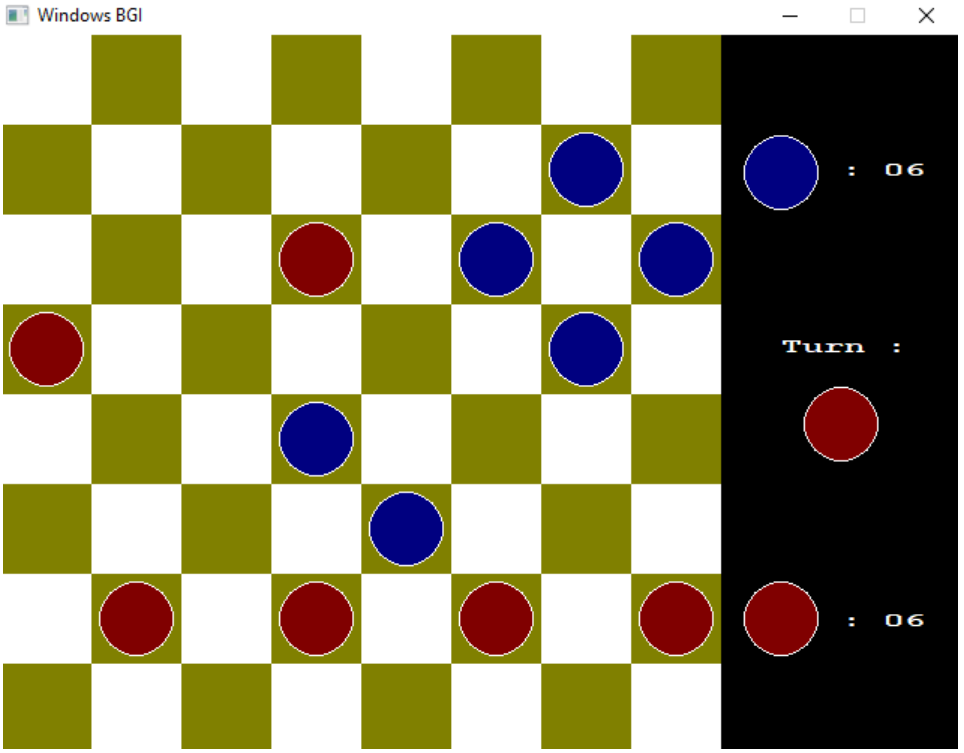

Fig-13: (Showing present game state)

- Iterative UI Refinement:

  The UI underwent iterative refinement based on user feedback and playtesting. Aesthetic enhancements, improvements in responsiveness, and adjustments to color schemes were implemented to optimize the overall user experience. The iterative approach ensured that the UI evolved in tandem with the project's development.

The integration of graphics.h into the User Interface of the Checkers Game Project was instrumental in providing players with a visually engaging and intuitive gaming experience. From rendering the game board to facilitating player interactions and visualizing AI decision-making, the UI played a pivotal role in shaping the overall appeal and accessibility of the project.

5. Challenges Faced

The development of the Checkers Game Project presented several challenges, each requiring thoughtful consideration and innovative solutions. The following outlines the key challenges encountered during the project's lifecycle:

- Graphics.h Limitations:

  Integrating graphics.h for the user interface posed challenges due to its limitations and lack of portability. The library's compatibility issues with modern compilers and operating systems required meticulous troubleshooting and adaptation to ensure a stable and functional graphical interface.

- AI Optimization and Performance:

  Implementing the minimax algorithm with various optimization techniques for the Expert-level AI introduced performance challenges. Balancing the depth of search for strategic decision-making with computational efficiency required fine-tuning to prevent undue delays in AI response time.

- Edge Cases in AI Decision-Making:

  The complexity of the Checkers game posed challenges in handling edge cases within the AI decision-making process. Situations involving kinged pieces, and potential forks in the game tree required specialized handling to ensure accurate and strategic AI moves.

- Iterative Deepening Complexity:

  Implementing iterative deepening in the minimax algorithm added a layer of complexity to the decision-making process. Coordinating the incremental deepening steps while maintaining an efficient search for the optimal move demanded careful synchronization within the algorithm.

- Alpha-Beta Pruning Precision:

  Although alpha-beta pruning significantly enhanced the AI's efficiency, ensuring its precision and correctness within the intricate decision tree was challenging. Debugging and refining the pruning logic were crucial to prevent unintended consequences on the final move selection.

- User Interface Responsiveness:

  Achieving a responsive and visually appealing user interface with graphics.h required addressing latency issues and optimizing rendering functions. Ensuring a seamless interaction experience for players, especially during AI computation, demanded continuous adjustments and performance enhancements.

Addressing these challenges required a combination of problem-solving skills, algorithmic refinement, and iterative development. The journey to overcome these obstacles contributed to a deeper understanding of both Checkers game dynamics and the nuances of integrating artificial intelligence into gaming environments.

6. Conclusion

The Checkers Game Project has been a dynamic exploration into the realms of traditional board gaming, artificial intelligence, and user interface design. Through meticulous planning, iterative development, and the integration of complex AI algorithms, the project has culminated in a versatile and engaging gaming experience.

The dual-module structure, catering to both human vs. human and human vs. AI gameplay, provided a comprehensive platform for players to enjoy the timeless game of Checkers. The incorporation of graphics.h for the user interface allowed for a visually appealing and interactive representation of the game board, enriching the overall gaming experience.

The AI implementation, ranging from a basic random move policy for beginners to the sophisticated minimax algorithm for expert-level opponents, showcased a progression of strategic complexity. Each level of AI brought its own set of challenges, necessitating innovative solutions to optimize decision-making, enhance performance, and balance difficulty levels.

The challenges faced throughout the project, from graphics.h limitations to the intricacies of AI optimization, became stepping stones for learning and improvement. Debugging complex algorithms, refining the user interface for responsiveness, and addressing compatibility concerns underscored the importance of adaptability and perseverance in software development.

In conclusion, the project successfully achieved its objectives of creating an interactive and challenging checkers game environment. The implementation choices and user-friendly interface contribute to a project that not only entertains but also serves as a valuable educational tool for understanding the intersection of gaming and artificial intelligence.

As technology continues to evolve, this project stands as a testament to the ongoing possibilities within the field of game development and AI integration. The lessons learned and challenges overcome serve as a foundation for future endeavors, encouraging a continued exploration of innovative solutions and advancements in the dynamic landscape of AI gaming.

7. Appendix

My github link: https://github.com/kibria30/SPL1_checkers_game

Reference

1. https://www.usatoday.com/story/graphics/2023/01/23/how-to-play-checkers-rules-strategy/

   - Name of the page: "How to Play Checkers: Rules and Strategy"

   - Last accessed on: 12 Nov 2023

2. https://en.wikipedia.org/wiki/Game_tree

   - Name of the page: "Game Tree - Wikipedia"

   - Last accessed on: 10 Nov 2023

3. https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/

   - Name of the page: "Minimax Algorithm with Alpha-Beta Pruning"

   - Last accessed on: 11 Nov 2023

4. https://www.youtube.com/watch?v=l-hh51ncgDI

   - Name of the page: "YouTube - Minimax Algorithm and Alpha-Beta Pruning Tutorial"

   - Last accessed on: 13 Nov 2023

5. https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/

   - Name of the page: "Minimax Algorithm with Alpha-Beta Pruning"

   - Last accessed on: 10 Dec 2023

6. https://www.youtube.com/watch?v=l-hh51ncgDI

   - Name of the page: "YouTube - Minimax Algorithm and Alpha-Beta Pruning Tutorial"

   - Last accessed on: 09 Dec 2023

7. https://brilliant.org/wiki/depth-first-search-dfs/

   - Name of the page: "Depth-First Search (DFS) - Brilliant.org"

   - Last accessed on: 13 Nov 2023

8. https://www.programiz.com/dsa/graph-dfs

   - Name of the page: "Depth-First Search (DFS) in Graph - Programiz"

- Last accessed on: 10 Nov 2023

9. https://www.scaler.com/topics/data-structures/backtracking-algorithm/
   - Name of the page: "Backtracking Algorithm - Scaler"
   - Last accessed on: 11 Nov 2023

10. https://www.simplilearn.com/tutorials/data-structure-tutorial/backtracking-algorithm
    - Name of the page: "Backtracking Algorithm Tutorial - Simplilearn"
    - Last accessed on: 12 Nov 2023