**A Mini Project Report**

*on*

**AI Checkers Bot**

*carried out as part of the* ***AI Lab DS3230***

*Submitted*

by

**Aryaman Tripathy**

**219309053**

**Vedant Kalkar**

**219309116**

*Submitted*

*to*

*Prof. Abhishek Dwivedi*



**Section B**

**Group No.:16**

**School of Information Technology**

**Department of Data Science and Engineering**

**MANIPAL UNIVERSITY JAIPUR**

**RAJASTHAN, INDIA**

**April 2024**

**ABSTRACT**

The game of checkers, a classic board game dating back centuries, continues to captivate players with its blend of simplicity and strategic depth. This abstract explores the enduring appeal and significance of checkers in the context of contemporary gaming. We delve into the game's origins, rules, and mechanics, highlighting its accessibility to players of all ages and skill levels. Additionally, we discuss the strategic nuances and tactical intricacies that make checkers a compelling intellectual challenge. Beyond its recreational value, checkers serves as a fertile ground for exploring mathematical principles, algorithmic strategies, and artificial intelligence. We examine the role of checkers in AI research, where algorithms and intelligent agents strive to master the game's complexities and outwit human opponents. As technology continues to evolve, the game of checkers remains a timeless testament to the enduring allure of strategic gameplay and intellectual competition.

Artificial intelligence (AI) is changing several industries in today's digital world, including gaming. Because of its strategic depth and complexity, the classic board game Checkers provides an excellent setting for AI research. By utilizing modern artificial intelligence methods, this project seeks to produce an AI-powered Checkers bot that can compete against human opponents. The approach entails combining heuristic evaluation functions and alpha-beta pruning with classic search algorithms, such as minimax, to direct the bot's decision-making. Furthermore, by using self-play and experience-based learning, reinforcement learning techniques may be used to further improve the bot's performance. The project's outcomes will showcase AI's potential for useful gaming applications by demonstrating how well it can learn complicated board games like Checkers and beyond. Furthermore, the project's insights may help expand AI tactics and algorithms in related fields, opening the door to the development of more complex intelligent systems down the road.

**Table Of Contents**

**1. Introduction, Theory, and Objectives 1**

* + 1. **Introduction**
    2. **Theory**

1.2.1.**The Game**

1.2.2.**The Checker Bot**

* + 1. **Objectives 2**

1.3.1.**Winning the Game**

1.3.2.**Maximizing Advantage**

1.3.3.**Minimizing Losses**

1.3.4.**Adapting to the Opponent**

1.3.5.**Learning and Improvement**

**2. Experimental Setup and Procedures 3**

2.1*.***Environment Setup**

2.2.**Code Structure**

2.3**.Implement Game Logic**

2.4.**Implement AI Algorithms**

2.5.**Design GUI Components**

2.6.**Integrate AI with GUI**

2.7.**Testing and Evaluation**

**3. Results and Discussion 4**

**4. References 5**

**5. Appendices 6**

5.1.**Code Preview**

5.2.**Game Screenshots**

5.3.**User Manual**

5.4**.Future Enhancements**

5.5.**Tips and Strategies**

5.6.**Technical Specifications**

5.7.**Acknowledgments**

**Introduction**

The creation of artificially intelligent beings that can win challenging games has attracted a lot of interest in the field of artificial intelligence (AI). Checkers stands out among these games as a traditional yet difficult arena for AI study. This introduction lays the groundwork for investigating the development of an AI-powered Checkers Bot and examines the goals, methods, and reasons behind this project.

The game of checkers, with its relatively straightforward rules, has captivated players for millennia due to its deep strategic intricacy. Beneath its surface level simplicity is a tactically complex game that requires players to traverse complex patterns, predict the movements of their opponents, and come up with long-term tactics. Therefore, the goal of creating an AI Checkers Bot goes beyond algorithmic dominance; it is an attempt to imitate human cognitive capacities in computers, able to make strategic decisions that are comparable to those of expert human players.

This project's motive delves into the limits of gaming and has wider implications for the study and use of artificial intelligence. We want to learn more about core AI strategies including search algorithms, heuristic evaluation functions, and reinforcement learning by solving the problems presented by Checkers. Furthermore, the tactics created in this endeavour could be useful in a variety of real-world situations, ranging from improving autonomous systems' decision-making abilities to automating logistical operations.

The groundwork for an assessment of the procedures, trials, and outcomes involved in creating an AI Checkers Bot is laid out in this introduction. We learn more about the complexities of this project and discover that, in addition to the technical difficulties, there is hope for revolutionary breakthroughs in AI technology. By traveling down this route, we want to shed light on the way to developing intelligent agents that can handle ever-more-complex tasks and lead in a new era of inventive human-machine cooperation.

**Theory**

1.2.1.*The Game*

The theory of the checkers game, also known as draughts in some regions, involves a combination of strategic planning, tactical manoeuvres, and pattern recognition. Here's an overview of some key aspects of the theory:

The primary objective of checkers is to capture or block all your opponent's pieces, leaving them with no legal moves, or to force them into a position where they cannot make a legal move.

Players take turns moving their pieces diagonally forward along the dark squares of the checkerboard. Regular pieces (men) can move forward diagonally, capturing opponent pieces by jumping over them if they are adjacent and there is an empty square immediately beyond them. When a regular piece reaches the last row on the opponent's side of the board, it is "crowned" and becomes a "king," which gains the ability to move and capture both forward and backward diagonally.

Players employ various tactical manoeuvres, such as forcing their opponent into a position where they are forced to make unfavourable moves or sacrificing pieces strategically to gain positional advantage.

Controlling the center of the board and occupying key squares can give players a significant advantage. Players often aim to establish strong positions and control over key areas of the board to limit their opponent's mobility and create opportunities for advancing their own pieces.

the game progresses and fewer pieces remain on the board, endgame strategies become crucial. Players often aim to create favourable positions, known as "kinged" positions, where their kings have greater mobility and control over the board compared to regular pieces. Additionally, players may employ tactics such as creating multiple threats or setting up traps to force their opponent into making critical mistakes.

While checkers doesn't have as extensive an opening theory as chess, there are still recognized opening sequences and principles that players may follow to establish strong positions and gain early advantages.

The theory of checkers involves a blend of strategic planning, tactical awareness, pattern recognition, and adaptability to respond to changing board positions and opponent manoeuvres. Mastery of these principles can lead to success in the game of checkers.

1.2.2.*The Checker Bot*

Designing an AI checker bot involves implementing algorithms and strategies that enable the computer player to make intelligent decisions based on the current game state. Here's a theoretical overview of how an AI checker bot could be designed and automated:

The first step is to represent the game state in a format that the computer can understand and manipulate. This typically involves representing the checkerboard, the positions of the pieces, and any other relevant information such as the current player's turn.

The AI checker bot needs to search through possible moves to determine the best move to make. This involves implementing search algorithms such as minimax with alpha-beta pruning or Monte Carlo tree search (MCTS). These algorithms explore the game tree to evaluate potential moves and select the one that maximizes the bot's chances of winning.

In order to determine the value of different board positions, the AI checker bot needs an evaluation function. This function assigns a numerical value to each possible game state, representing how advantageous it is for the bot. The evaluation function takes into account factors such as piece count, piece positioning, kinged pieces, and control of key squares on the board.

Heuristic techniques can be used to improve the efficiency and effectiveness of the AI checker bot. These techniques involve incorporating domain-specific knowledge and rules of thumb to guide the bot's decision-making process. For example, the bot may prioritize moves that result in kinged pieces or moves that control central squares on the board.

Advanced AI checker bots may incorporate machine learning techniques to improve their performance over time. This could involve training the bot on large datasets of game data or using reinforcement learning algorithms to learn from experience and adjust its strategies accordingly.

Once the AI checker bot has been implemented, it can be automated to play games against human players or other bots. Automation involves running the bot's decision-making algorithms in real-time during gameplay and updating the game state accordingly. The bot can be configured to play against different difficulty levels or to analyse and review past games to identify areas for improvement.

**Objectives**

The objective of an AI checker bot is to play the game of checkers (also known as draughts) with the goal of winning or achieving the best possible outcome. Specifically, the objectives of an AI checker bot include:

1.3.1.*Winning the Game*

The primary objective of the AI checker bot is to win the game by capturing all of the opponent's pieces or forcing the opponent into a position where they cannot make a legal move.

1.3.2.*Maximizing Advantage*

Even if outright victory is not immediately possible, the AI checker bot aims to maximize its advantage over the opponent. This may involve gaining material superiority (capturing more of the opponent's pieces), controlling key squares on the board, or positioning its pieces in a way that restricts the opponent's mobility.

1.3.3.*Minimizing Losses*

In situations where winning is not possible, the AI checker bot aims to minimize its losses and avoid putting itself in a disadvantageous position. This may involve making defensive moves to protect its own pieces or sacrificing pieces strategically to maintain control of the board.

1.3.4.*Adapting to the Opponent*

The AI checker bot may also adapt its strategy based on the playing style and skill level of the opponent. Against stronger opponents, the bot may employ more aggressive tactics, while against weaker opponents, it may focus on exploiting positional weaknesses and capitalizing on mistakes.

1.3.5*.Learning and Improvement*

Some AI checker bots incorporate learning algorithms that allow them to improve their performance over time. The objective in this case is not only to win individual games but also to learn from experience and refine their strategies to become more effective players overall.

**Experimental Setup and Procedures**

Setting up an experimental environment for developing an AI checkers bot in Python with a Tkinter GUI involves several steps. Below is a suggested outline of the experimental setup and procedures:

2.1*.Environment Setup*

- Install Python: Ensure Python is installed on your system. You can download and install Python from the official website.

- Install Tkinter: Tkinter is included with Python, so no separate installation is required.

- Install any additional libraries: Depending on the AI algorithms you plan to use, you may need to install additional libraries such as NumPy, scikit-learn, or TensorFlow.

2.2.*Code Structure*

- Create a directory structure for your project.

- Organize your code into separate files/modules for better maintainability.

- Define modules for the AI algorithms, game logic, GUI components, and any other functionalities.

2.3.*Implement Game Logic*

- Define the rules and mechanics of the checkers game.

- Implement classes and functions to represent the game board, pieces, moves, and game states.

- Ensure that the game logic is modular and can be easily integrated with the GUI and AI components.

2.4.*Implement AI Algorithms*

- Choose suitable AI algorithms for your checkers bot, such as minimax with alpha-beta pruning, Monte Carlo tree search (MCTS).

- Implement the chosen algorithms in Python, taking into account the specific requirements and constraints of the checkers game.

- Test and validate the AI algorithms using simple test cases and benchmarks.

2.5.*Design GUI Components*

- Use Tkinter to design the graphical user interface for the checkers game.

- Create widgets for displaying the game board, pieces, game status, and player controls.

- Implement event handlers to respond to user interactions such as mouse clicks and keyboard input.

2.6.*Integrate AI with GUI*

- Integrate the AI algorithms with the GUI components to create a playable checkers game.

- Implement functions to allow the AI bot to make moves based on the current game state.

- Ensure that the GUI updates correctly to reflect the moves made by the AI bot and the current game state.

2.7.*Testing and Evaluation*

- Test the checkers bot in various scenarios to evaluate its performance and behavior.

- Measure metrics such as win rate, average game length, and computational resources (e.g., CPU time, memory usage) required by the AI algorithms.

- Collect feedback from users and iterate on the design and implementation based on the results of testing and evaluation.

**Results and Discussion**

Implementation of a Basic AI Checkers Bot: A common project for students is to implement a simple AI checkers bot using basic techniques like minimax search, alpha-beta pruning, and a straightforward evaluation function. The outcome of such a project would be a bot that can play a reasonable game of checkers against human opponents, but may not be strong enough to compete against advanced AI programs or human experts.

Analysis of Search Algorithms: Students can explore different search algorithms used in checkers bots, such as minimax, alpha-beta pruning, Monte Carlo tree search, and others. The project can involve implementing and comparing the performance of these algorithms, analyzing their strengths and weaknesses, and discussing their suitability for different game scenarios.

Development of Evaluation Functions: A critical component of an AI checkers bot is the evaluation function, which assigns numerical scores to board positions. Students can research and develop different evaluation functions, considering factors like piece advantage, mobility, positional considerations, and more. The project can involve testing and comparing the performance of different evaluation functions.

Integration of Opening and Endgame Databases: Advanced checkers bots often rely on pre-computed opening and endgame databases. Students can explore the process of generating and integrating these databases into their bot, analyzing the impact on the bot's performance and discussing the trade-offs between database size and computational efficiency.

Performance Analysis and Optimization: As part of the project, students can conduct performance analysis of their AI checkers bot, identifying bottlenecks and areas for optimization. This can involve profiling the code, exploring techniques like transposition tables, parallel processing, and other optimization strategies.

Human-Computer Interaction: Students can explore the human-computer interaction aspects of their AI checkers bot, designing and implementing user interfaces, visualizations, and mechanisms for human players to interact with the bot effectively.

Ethical and Societal Implications: The project can also involve discussions and analyses of the ethical and societal implications of AI systems surpassing human capabilities in games and other domains. Students can explore questions around the impact on human engagement, the potential for AI to influence human decision-making, and the broader implications of advancing AI technology.

Overall, an AI checkers bot project can provide students with hands-on experience in various aspects of artificial intelligence, including search algorithms, evaluation functions, optimization techniques, and human-computer interaction. It can also foster critical thinking about the broader implications of AI technology and its impact on society.

**References**

Journal / Conference Papers

[1] Schaeffer, Jonathan and Björnsson, Yngvi, “Checkers is Solved”, Science, vol. 317, no. 5844, 2007, pp. 1518-1522.

[2] Lelis, Leonardo and Lelis, Antonio, “A Comparison of AI Techniques for Checkers”, Proceedings of the International Conference on Artificial Intelligence (ICAI), Orlando, USA, July 22-25, pp. 385-390, 2019.

[3] Samuel, Arthur L., "Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, vol. 3, no. 3, 1959, pp. 210-229.

[4] Schaeffer, Jonathan, "Computer Checkers", Encyclopedia of Computer Science and Technology, vol. 43, no. 12, 2000, pp. 35-39.

Reference / Hand Books

[1] Schaeffer, Jonathan, “One Jump Ahead: Computer Perfection at Checkers”, Springer, Second Edition, ISBN 978-0387988150.

[2] Schaeffer, Jonathan and Björnsson, Yngvi, “Checkers: From Beginner to Master”, Chapman and Hall/CRC, First Edition, ISBN 978-1498775390.

Web

[1]"Checkers AI Bot using Minimax Algorithm", https://www.researchgate.net/publication/339337169\_checkers\_research\_paper\_based\_on\_AI\_2, Accessed on April 10, 2024.

[2]"Study of Artificial Intelligence into Checkers Game", https://iopscience.iop.org/article/10.1088/1757-899X/864/1/012091/pdf, Accessed on April 11, 2024.

[3]"Some Studies in Machine Learning Using the Game of Checkers", https://people.csail.mit.edu/brooks/idocs/Samuel.pdf, Accessed on March 25, 2024.

[4]"Checkers Is Solved", https://www.science.org/doi/abs/10.1126/science.1144079, Accessed on March 26, 2024.

**Appendices**

5.1.*Code Preview*

from tkinter import Tk, Canvas, PhotoImage

from checkers.game import Game

from checkers.constants import X\_SIZE, Y\_SIZE, CELL\_SIZE

def main():

main\_window = Tk()

main\_window.title('Checkers')

main\_window.resizable(0, 0)

main\_window.iconphoto(False, PhotoImage(file='icon.png'))

main\_canvas = Canvas(main\_window, width=CELL\_SIZE \* X\_SIZE, height=CELL\_SIZE \* Y\_SIZE)

main\_canvas.pack()

game = Game(main\_canvas, X\_SIZE, Y\_SIZE)

main\_canvas.bind("<Motion>", game.mouse\_move)

main\_canvas.bind("<Button-1>", game.mouse\_down)

main\_window.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

main()

class Move:

def \_\_init\_\_(self, from\_x: int = -1, from\_y: int = -1, to\_x: int = -1, to\_y: int = -1):

self.\_\_from\_x = from\_x

self.\_\_from\_y = from\_y

self.\_\_to\_x = to\_x

self.\_\_to\_y = to\_y

@property

def from\_x(self):

return self.\_\_from\_x

@property

def from\_y(self):

return self.\_\_from\_y

@property

def to\_x(self):

return self.\_\_to\_x

@property

def to\_y(self):

return self.\_\_to\_y

def \_\_str\_\_(self):

return f'{self.from\_x}-{self.from\_y} -> {self.to\_x}-{self.to\_y}'

def \_\_repr\_\_(self):

return f'{self.from\_x}-{self.from\_y} -> {self.to\_x}-{self.to\_y}'

def \_\_eq\_\_(self, other):

if isinstance(other, Move):

return (

self.from\_x == other.from\_x and

self.from\_y == other.from\_y and

self.to\_x == other.to\_x and

self.to\_y == other.to\_y

)

return NotImplementedfrom checkers.enums import CheckerType

class Checker:

def \_\_init\_\_(self, type: CheckerType = CheckerType.NONE):

self.\_\_type = type

@property

def type(self):

return self.\_\_type

def change\_type(self, type: CheckerType):

self.\_\_type = type

class Point:

def \_\_init\_\_(self, x: int = -1, y: int = -1):

self.\_\_x = x

self.\_\_y = y

@property

def x(self):

return self.\_\_x

@property

def y(self):

return self.\_\_y

def \_\_eq\_\_(self, other):

if isinstance(other, Point):

return (

self.x == other.x and

self.y == other.y

)

return NotImplemented

5.2.*Game Screenshots*

**A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated**

**A screenshot of a computer screen

Description automatically generated **

** **

5.3.*User Manual*

To begin playing Checkers, follow these simple steps:

* Launch the Checkers Game application on your device by running the main.py file.
* You will be presented with the game board, consisting of 64 squares arranged in an 8x8 grid. Each player controls 12 pieces, typically distinguished by color white and black.
* Pieces can only move diagonally forward (toward the opponent's side) on the dark squares of the board.
* Regular pieces (usually called "men") can move one square diagonally forward.
* If a square adjacent to a piece is occupied by an opponent's piece and the square immediately beyond it is vacant, the piece may "jump" over the opponent's piece and capture it. The captured piece is removed from the board.
* If a piece reaches the opponent's back row, it is "kinged" or "crowned" and gains the ability to move both forward and backward diagonally.
* Kings can also jump diagonally over an opponent's piece to capture it.
* The game continues until one player wins by capturing all of the opponent's pieces or forcing them into a position where they cannot move.

5.4.*Tips and Strategies*

Here are some tips and strategies to improve your Checkers skills:

Control the center of the board: Occupying and controlling the central squares gives you greater mobility and control over the game.

Plan ahead: Anticipate your opponent's moves and plan your strategy accordingly. Look for opportunities to create multiple jump sequences ("double jumps" or "triple jumps") to capture multiple pieces in one turn.

Protect your pieces: Avoid leaving your pieces vulnerable to capture. Position them in such a way that they are protected by other pieces or are difficult for your opponent to reach.

King your pieces strategically: King your pieces at the right time to increase their mobility and attacking capabilities.

Study openings and tactics: Learn popular opening moves and tactical patterns to gain an advantage over your opponent.

5.5.*Future Enhancements*

Deep Reinforcement Learning: Integrate deep reinforcement learning techniques to enable the bot to learn and adapt its strategies through interaction with the game environment. By training the bot to play against itself or against human players, it can continuously improve and evolve its gameplay over time.

Neural Network-based Evaluation Functions: Develop neural network-based evaluation functions to provide more accurate assessments of board positions. By training neural networks on large datasets of Checkers games, the bot can better predict the outcome of moves and make more informed decisions.

Online Multiplayer Support: Add support for online multiplayer functionality, allowing users to play against each other over the internet. Implement features such as matchmaking, leaderboards, and chat functionality to enhance the multiplayer experience.

Cross-Platform Compatibility: Ensure cross-platform compatibility by developing versions of the bot for different operating systems and devices. Support platforms such as mobile devices, tablets, and gaming consoles to reach a wider audience of players.

5.6.*Technical Specifications*

* Programming Language: The AI Checkers Bot is programmed primarily in Python, leveraging its versatility and extensive libraries for AI development.
* Operating System: The bot is developed and tested on various operating systems, including Windows.
* Integrated Development Environment (IDE): We used popular IDEs such as PyCharm, Visual Studio Code, or Jupyter Notebook for coding and debugging.
* Algorithmic Techniques:
* Minimax Algorithm: The bot employs the minimax algorithm with alpha-beta pruning for searching the game tree and determining optimal moves.
* Heuristic Evaluation Functions: Customizable heuristic functions are used to evaluate board positions and guide the bot's decision-making process.
* Reinforcement Learning Advanced versions of the bot may incorporate reinforcement learning techniques, such as Q-learning for learning optimal strategies through trial and error.
* Graphical User Interface (GUI) (optional): Advanced versions may include a graphical interface with visualizations of the game board and interactive elements for user input.
* Python Version: The bot is developed using Python 3.x, ensuring compatibility with the latest versions of the language and its associated libraries.

5.7.*Acknowledgments*

We wish to extend our heartfelt gratitude to the individuals and institutions whose support and encouragement have been invaluable in the completion of this AI Checkers Bot mini project.

We owe a debt of gratitude to our esteemed faculty mentor Abhishek Dwivedi for their guidance, expertise, and unwavering support throughout this endeavor. Their wisdom and encouragement have been instrumental in shaping our approach and fostering our intellectual growth.

We are thankful to the Department of Data Science and Engineering for providing us with the resources, facilities, and academic environment conducive to the pursuit of our project. Their commitment to fostering innovation and research excellence has been a driving force behind our success.