Object-Oriented Implementation of Tic-Tac-Toe with AI Integration

Namrata Mali

[malinamratavijay@cityuniversity.edu](mailto:malinamratavijay@cityuniversity.edu)

Ayush Kumar Gupta

[guptaayushkumar@cityuniversity.edu](mailto:guptaayushkumar@cityuniversity.edu)

Sai Mani Ritish Upadhyayula

[upadhyayulasaimanir@cityuniversity.edu](mailto:upadhyayulasaimanir@cityuniversity.edu)

Vidhiben Ashokbhai Vanani

vananividhibenashok@cityuniversity.edu

School of Technology & Computing

City University of Seattle

Seattle, WA 98121, USA

**Abstract**

This project demonstrates how to develop and use a Tic-Tac-Toe game in Python by utilizing object-oriented programming (OOP) principles and including AI elements, focusing on modularity, efficiency, and interactive gameplay. The game features an unbeatable AI opponent implemented by the Minimax algorithm, ensuring that the AI always makes optimal moves to win, offering a challenging experience for the human player as the worst-case scenario would always be a tie and never a human win. Core OOP principles like encapsulation, abstraction, inheritance, and polymorphism are applied throughout the project to ensure clear structure and functionality and hence emphasizing the the OOP’s bebefits. Encapsulation is used to manage entire game logic using individual classes, while abstraction hides the complexities of tasks such as board rendering, move validation, and AI decision-making from the user. Inheritance allows shared behavior between human and AI players, while polymorphism ensures flexibility by enabling different implementations of the move-making process for each player type. Key components include the `Renderer` class for visualizing the game board with Pygame, the `Logger` class to record each move in a text file, and the `Minimax\_AI` class, which uses the Minimax algorithm to calculate the best possible move for the AI. The game’s structure is modular, with each class encapsulating specific functionality, ensuring that the program remains scalable and maintainable. This project showcases the power of combining algorithmic logic, like the Minimax algorithm, with OOP principles to create an engaging, interactive application. It shows how object-oriented programming can simplify complex problems, providing an organized, efficient solution that is user-friendly. Object-Oriented Programming (OOP) is utilized in developing Tic-Tac-Toe games using the Minimax Algorithm for game development.

**Keywords:** Object-Oriented Programming (OOP)**,** Tic-Tac-Toe**,** Minimax Algorithm**,** Game Development

1. **INTRODUCTION**

The timeless Tic-Tac-Toe game is a traditional strategy game for two players that has been created in multiple versions. This project applies the OOP principles in Python to create an operational, interactive version of the game. The game includes a user interface that enables two players to engage in gameplay on a 3x3 or 5x5 grid, incorporating AI integration for a superior gaming experience. The main goal is to offer a tough gaming experience while showcasing the application of OOP principles in developing flexible, manageable games.   
  
OOP principles like encapsulation, inheritance, abstraction, and polymorphism play a key role in the design of the games in this project. Through the use of distinct classes for various aspects such as game logic, player behavior, board rendering, move logging, and AI decision-making, the code is made modular and simple to expand upon. The Renderer class manages the visual display, the Logger class monitors actions, and the Minimax\_AI class ensures optimal gameplay for the AI. This method shows how OOP can help organize complex logic into manageable components such as classes, enhancing code readability and reusability. The project provides a game that can be played and demonstrates the connection between algorithmic logic and object-oriented design in creating successful software solutions.

1. **APPROACH**

The strategy used in this Tic-Tac-Toe game project merges object-oriented programming (OOP) concepts with sophisticated algorithmic design to guarantee a challenging gameplay. The central focus of the game is the grid structure, which clearly differentiates between HumanPlayer and AIPlayer roles. The AIPlayer uses the Minimax algorithm, which is a decision-making algorithm that considers all possible moves to predict outcomes in order to maximize the AI's chances of winning and select the best move to win while reducing the player's success. This enables the AI to perform at its best and creates a challenging experience for the human player.

OOP principles are central to the design, with various classes such as Renderer, Logger, Player, and Minimax\_AI dedicated to specific functionalities. This modular structure promotes clarity, maintainability, and scalability. For example, the Renderer class is responsible for drawing the game grid and updating the display, while the Logger class manages the recording of moves in a structured format. Encapsulation and abstraction are employed to hide complex operations, such as win-checking and rendering, from external components, ensuring that users only interact with high-level methods. The polymorphic behavior of the make\_move method allows both HumanPlayer and AIPlayer to implement their move logic differently while adhering to the same interface. Additionally, the move history is logged in a `tictactoe.txt` file for accountability, and the game includes a restart feature, providing an interactive and dynamic gaming experience.

1. **IMPLEMENTATION CHALLENGE**

**Problem:**

While implementing the Tic-Tac-Toe game for both 3x3 and 5x5 grids, we encountered a significant issue with the AI's move-making process when playing on the 5x5 board. Specifically, the AI would enter an infinite recursion loop during its turn. This happened because the recursive **Minimax algorithm**, which was designed to evaluate all possible moves by simulating the entire game tree, became computationally expensive as the board size increased. For the 5x5 board, the number of possible moves and states was so large that the recursion depth grew excessively, causing the program to exhaust system resources and resulting in the infinite recursion loop.

**Solution:**

To resolve this issue, we transitioned from a purely recursive Minimax approach to a more efficient **heuristic iterative** method. By introducing a heuristic evaluation function that estimates the value of a board state without needing to simulate every possible move to the end, we significantly reduced the time complexity. Instead of recursively evaluating all game states, the AI now uses an iterative process to assess the most promising moves based on a limited search depth, combined with a heuristic evaluation of the board.

This allowed the AI to make decisions much more efficiently, even for the larger 5x5 grid, and eliminated the infinite recursion issue. By restricting the depth of the recursive search and focusing on the most probable moves, the AI can play optimally without overloading the system.

1. **FORMULAE**

A black and white math equation

Description automatically generated with medium confidence

The Minimax algorithm analyzes possible game states through recursion to choose the best move for the AI, working with both 3x3 and 5x5 Tic-Tac-Toe boards. Points are awarded according to the game's results: AI victories earn +10 points, losses against the opponent lead to -10 points, and draws garner 0 points. The process works by continuously analyzing potential moves at every level. When it's the AI's chance, the program aims to increase the score by choosing the move with the greatest value (maximizing). When it's the opponent's chance to play, their goal is to reduce the score by selecting the move with the least value (minimizing). The algorithm considers all possible moves to ensure that the AI chooses the best move when assuming both players are playing at their best. In order to decrease latency, iterative deepening and heuristic evaluation are utilized, enabling quicker decision-making, especially for the larger 5x5 grid.

1. **ILLUSTRATIONS**

### **Class Diagram:**

The class diagram demonstrates the structure and relations between the core functional classes in the Tic-Tac-Toe game. At the center is the TicTacToe class, which executes the game flow. It interacts with the Player class, which is a parent class for both HumanPlayer and AIPlayer. The AIPlayer uses the Minimax\_AI class to determine optimal moves. The Renderer class handles the game board visualization, drawing the grid and pieces, while the Logger class logs each move to a file. The diagram emphasizes object-oriented principles like inheritance, encapsulation, and polymorphism to maintain modularity and clarity.

A diagram of a game

Description automatically generated with medium confidence

### **Illustration of Gameplay:**

The Minimax algorithm analyzes possible game states through recursion to choose the best move for the AI, working with both 3x3 and 5x5 Tic-Tac-Toe boards. Points are awarded according to the game's results: AI victories earn +10 points, losses against the opponent lead to -10 points, and draws garner 0 points. The process works by continuously analyzing potential moves at every level. When it's the AI's chance, the program aims to increase the score by choosing the move with the greatest value (maximizing). When it's the opponent's chance to play, their goal is to reduce the score by selecting the move with the least value (minimizing). The algorithm considers all possible moves to ensure that the AI chooses the best move when assuming both players are playing at their best. To decrease latency, iterative deepening and heuristic evaluation are utilized, enabling quicker decision-making, especially for the larger 5x5 grid.

A screenshot of a game

Description automatically generated

**Fig 1: 3x3 Tic Tac Toe**

**A screenshot of a game

Description automatically generated**

**Fig 2: 5x5 Tic Tac Toe**

### **Minimax Algorithm Flowchart:**

The flowchart illustrates the recursive process of the Minimax algorithm used by the AI to select the best move. It starts by evaluating the current game state. When the AI wins, it gets 10 points; if the opponent wins, it gets -10 points; and if it's a tie, it gets 0 points. The AI algorithm explores all possible future outcomes by switching between maximizing its own score and minimizing the opponent's score recursively.

A screenshot of a diagram

Description automatically generated

1. **CONCLUSION**

To sum up, the Tic-Tac-Toe game project demonstrates how object-oriented programming (OOP) principles are utilized in creating interactive applications. The project uses a organized class structure, with unique classes such as `Renderer`, `Logger`, `Player`, `HumanPlayer`, `AIPlayer`, and `Minimax\_AI`, each handling specific tasks to keep logic separated.

The integration of the Minimax algorithm ensures that the AI provides an unbeatable opponent which makes the best optimal move against the human oponent, offering a challenging and engaging experience for the player. The use of encapsulation hides complex operations such as win-checking and rendering, while abstraction simplifies user interactions. Polymorphism allows for interchangeable player types, whether human or AI, making the game flexible and scalable. Logging each move in a text file provides an added layer of transparency, and the option to restart the game if the user wishes to continue enhances user engagement. Overall, this project highlights how OOP concepts along with powerful algorithm design, can create an efficient, maintainable, and user-friendly application. By combining academic theory with practical implementation, the project showcases key software development techniques, offering developers and users a valuable chance to learn.

1. **REFERENCES**

GeeksforGeeks. (2023, February 20). *Finding optimal move in TicTacToe using Minimax Algorithm in Game Theory*. GeeksforGeeks. <https://www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/>

Pirildak, Y. (2022, January 29). Mastering Tic-Tac-Toe with Minimax Algorithm in Python. *Medium*. <https://levelup.gitconnected.com/mastering-tic-tac-toe-with-minimax-algorithm-3394d65fa88f>

Diagram source: *Napkin AI - The visual AI for business storytelling. (n.d.). Napkin AI.*[*https://www.napkin.ai/*](https://www.napkin.ai/)