*Phase one page 2*

*Phase two page 11*

Tic-Tac-Toe (AI) player

BY:-  
ZIAD MOHAMED

*Abstract*—The goal of this work is to construct an artificial intelligence (AI) player for the popular game of Tic-Tac-Toe by integrating AI approaches, especially the Minimax algorithm enhanced with Alpha-Beta pruning and heuristic search. The straightforward but entertaining strategy game Tic-Tac-Toe makes a perfect testbed for assessing AI systems because of its clear rules and constrained search field.

The core idea of game theory, the Minimax algorithm, serves as the basis for our AI player's decision-making. The Minimax algorithm guarantees a strategic approach to games by iteratively analyzing possible game states and choosing the best move while taking the opponent's best reaction into consideration. Moreover, the search method is optimized by integrating Alpha-Beta pruning, which lowers computing overhead and boosts productivity.

*Heuristic search strategies are used in conjunction to the Minimax algorithm to improve the AI player's performance even more. Heuristic features expedite decision-making and enhance gameplay efficiency by directing the search towards promising moves. Within time restrictions, heuristic search helps the AI player make well-informed decisions by finding a compromise between computing complexity and precision.*

*We assess the efficiency of our AI player in playing Tic-Tac-Toe versus human opponents with a battery of tests and analysis. The outcomes show that the AI player can regularly provide competitive gameplay outcomes in a variety of game conditions, including wins, losses, and draws.*

*Overall, by demonstrating the possibilities of Minimax algorithms, Alpha-Beta pruning, and heuristic search in developing intelligent game-playing agents, this research advances AI methodologies in gaming applications. The results open up fresh opportunities for investigation and practical application of these methods in more intricate gaming contexts and real-world situations.*

Keywords—Tic-Tac-Toe, Minimax algorithm, Alpha-Beta Pruning, Parallel Alpha-Beta Pruning, Sequential Alpha-Beta Pruning , Heuristic functions

# Introduction

Since the beginning of time, humans and games have been closely linked and constitute an integral aspect of existence. Games were used as a relaxation means of social interaction and as a vehicle for creative expression. There are several methods for classifying, including Chance vs. Strategic, Cooperative vs. Non-Cooperative, and Outdoor vs. Indoor. Since games and technology have evolved simultaneously, software games have expanded to include both traditional indoor games and outdoor games that can be played without moving, Tic-Tac-Toe became the most popular fast paper games, game developers who have technological tool wanted to make a simulation version using artificial intelligence but they faced some obstacles that made them can’t make the full usage of this advanced technologies which are money and time constraints; What is Tic-Tac-Toe? : it a game of paper and pen/pencil, the players alternately draw X or O signs in the gaps of the 3x3 grid the winner should put his sign in a horizontal, diagonal or vertical pattern to win [1]; Computer programmers that can play tic tac toe flawlessly or that can list the 765 places that are fundamentally different or the 26,830 games that can be played up to rotations and reflections on this space are frequently developed [1]. A player in the game of tic tac toe attempts to guarantee two situations: Increase a player's personal winning potential. Cut down on a block failure. Reduce the likelihood of defeating the opponent by optimizing revenue. To increase earnings, apply the fork or win strategy. Block the opponent's fork or two "x" or "o" in a row if the opponent has them. A fork requires the player to first create an opportunity where he can succeed in two different ways; this is an algorithm named (Minimax Algorithm) [1] and the AI model will be optimized by Alpha Beta pruning and Heuristic functions .

A blue grid with black text

Description automatically generated

The Ai model can fully observe with the state of the environment, Strategic, Sequential, Static, Discrete, Multi-Agent, and the rules of the game are known.

# Literature Review

## Minimax algorithm

Minimax or the Minmax is a recursive or backtracking algorithm that helps in choosing the most optimal move, in a two-person turn-based game, presuming that the other player also plays as well as possible. The set of all potential movements in that game is called the game tree, and it looks like an inverted tree. All feasible motions are contained in the leaf node. The current condition of the board is indicated by the game state. The game tree is updated height-wise and the game state is modified with each move [5] This algorithm is a basic concept in game theory and involves working up from the leaf nodes [4] which can be applied on tic tac toe perfectly .

A diagram of a computer move

Description automatically generated

Some Terminologies in the (Minimax algorithm):

1-MAX (Maximizer): Player whose chances of winning are Maximized.

2-MIN (Minimizer): Player whose chances of losing are to be maximized.

3-Initial State: Denotes the initial state of the game board.

4-Terminal State: Denotes the final state of the game board. It can be a win, loss or draw

## Alpha beta pruning

## The minimax method can be optimized using the alpha-beta pruning strategy; It greatly aids in cutting down on computation times. It enables quicker and deeper tree traversal for the user. The maximizer is represented by the letter "Alpha" in alpha-beta pruning, and the minimizer is represented by the letter "Beta." It seeks to lower the quantity of nodes that are visited during computing. When at least one alternative is discovered demonstrating that the move is worse than the move that was previously looked at, the evaluation process comes to an end; A search technique called alpha-beta pruning aims to reduce the number of nodes in the search tree that the minimax algorithm evaluates. When at least one scenario is discovered that indicates a move is worse than one that has already been looked at, the evaluation of that action comes to an end [1].

A diagram of a diagram of a diagram

Description automatically generated with medium confidence

*Alpha beta pruning types:*

1- Sequential :

Compared to the standard Minimax technique seen in the majority of Artificial Intelligence games, the Alpha-Beta pruning algorithm demonstrated its superiority. While determining the optimal path for a certain player is the goal of both algorithms, Alpha-Beta Pruning executes noticeably faster. It is noted that the algorithm operates more quickly since it removes values that have no bearing on the outcome.

Take a look at the example game tree in Figure 2. The maximizing player's nodes are represented by a square, and the minimizing player's nodes are represented by a circle, much like in the Minimax method. The apex of the tree structure, A, is where the first step begins. The value of -Infinity is assigned to Alpha, whereas the value of +Infinity [3].

2- Parallel:

Parallelism has been used into the current Alpha-Beta algorithm to further boost its efficiency.

Prior to producing the remaining siblings in parallel, the first sibling node is searched. Similar to the Young Brothers Wait Concept, Dynamic Tree Splitting uses a peer-to-peer technique in place of the master-slave model. This approach makes use of many processors, each of which handles a node independently and sends the node's evaluation back to its parent.

An approach called beam search is used to locate paths quickly and efficiently while using less memory. In addition to conventional Alpha-Beta Pruning, the same technique has been applied to Graphics Processing Units, where different sized boards are employed to reduce the amount of time needed for computing. Parallel larger boards with GPUs operate at a quicker speed than serial smaller boards*.*

## Heuristics search:

In artificial intelligence, heuristic search is a strategy for finding an approximate solution when traditional approaches fail or for solving a problem more quickly than traditional methods. This is accomplished by sacrificing speed for optimality, completeness, correctness, or precision. The heuristic search method is capable of evaluating the information at hand and choosing which branch to pursue. The heuristic approach has the ability to generate a workable solution for the issue [2]; These days, numerous efficient heuristic techniques have been successfully used in a wide range of problem fields. It is possible to apply heuristic methods in game development since they can stimulate the imagination of the player. When playing computer games, players want to have fun, and programmers need to know how to use heuristics to direct the software so they can win. Therefore, for the user (player) to have a more engaging experience with the game, the program must be intelligent[3].

# Methodology

The details of the approaches are already discussed in the Literature Review section; This section will focus on the

pseudocodes , diagrams, and flowcharts of the algorithms.

Note: The Flowchart of the 3 approaches combined is at the end of this section.

## Minimax algorithm

Pseudocode:

The following Pseudocode illustrates the logic of the minimax algorithm implementing the tic-tac-toe.



Diagram:

The following diagram illustrates the game sequence between the computer and the player with the minimax algorithm.

A diagram of a computer move

Description automatically generated

## Alpha beta pruning

## In this project the parallel alpha beta pruning will be used to enhance the minimax algorithm type because it saves more calculations and computation.

## Pseudocode:

## This a code is optimizing the minimax algorithm using alpha beta pruning.

A screenshot of a computer program

Description automatically generated

Diagram:

This diagram shows how the alpha beta pruning cuts branches of the minimax tree that leads to loss and even the branches that leads to win of tie if the minimizer agent will have the choice.

A diagram of a person with red x and blue squares

Description automatically generated

## Heuristic Search

#### The heuristic search putted a strategy for tic tac toe called (Rule-based Strategy) which is:

#### The player of Tic-Tac-Toe must mark each of the nine spaces with two distinct symbols, often crosses ('X') and noughts ('O'). This table lists the five rule-based tactics that players employ to win games of tac-tac-toe.

A white text on a black background

Description automatically generated

The process of developing tic-tac-toe games using artificial intelligence heuristics passes by 3 development process; Firstly, we carry out pre-production tasks including rule-based research as shown in the figure above, game prototype testing, and prototype development. Second, we handle all aspects of the production process, such as game design and programming. Lastly, we carry out testing and maintenance during the post-production phase

A diagram of a process

Description automatically generated

Detailed explanation :

In the First phase (pre-production) : We create a functional prototype for the game UI, complete with gameplay concepts and features, during the pre-production stage. By changing some aspects of the game, it serves as a proof of concept and a test for concepts. We then use the results of the prototype testing to create the final game interface. Through this approach, we are able to learn more about the features and final interface of our game, which will increase user attraction[2].

In the second phase(production) : Our main goals were to incorporate the heuristic notion in the source code and to give the game a more intricate design. We give a suitable AI for ideal game in the programming section . For every potential solution, the computer (the opponent in the game) must determine the degree of difficulty . Next, we create a design that combines game rules with the program fix. We create a sound effect design. Next, we create a catchy game theme and select appropriate components to fit the theme. Additionally, we overlay background images to every scenario[2].

In the third phase(post-production) : Our post-production development process encompasses testing and maintenance. maintenance is crucial. After the game idea is implemented, programmers must review the source code again and correct any errors that arise. The procedure of testing the game's effectiveness is then covered in the section that follows[2].

# EXPERIMENTS

Experiment 1 [2]:

This experiment implements the minimax algorithm enhanced with the alpha beta pruning and using the heuristic search.

To test the effectiveness of the tic tac toe game we created, we recruited ten participants. There are three rounds in every game, and Table 2 contains the outcomes. Next, we create a bar chart in Figure 6 to show who won the game.

Experiment 2 [5]:

Overview:

The minimax algorithm, a recursive search technique used in games with perfect information, is described in the pseudocode. By examining all potential game states and choosing the move that maximizes the player's chances of winning while taking the opponent's optimal moves into consideration, it seeks to determine the best move for the player.

The algorithm employs three evaluation functions (Pseudocode):

A white text with black text

Description automatically generated

main loop:

The pseudocode iteratively explores potential movements by iterating over each vacant spot on the game board. It utilizes the evaluation functions to award a score to each potential move and switches between placing 'X' and 'O' based on a Boolean flag.

Updating Min/Max Values:

The algorithm updates (max\_val) and (min\_val) to track the optimal moves for the maximizing and minimizing players, respectively, after analyzing each potential move.

Choosing Move:

When it is the player who is maximizing their turn, the algorithm chooses the move with the highest score (max\_val), and when it is the player who is minimizing their turn, it chooses the move with the lowest value (min\_val).

A white text on a white background

Description automatically generated

# RESULTS

Experiment 1 [2]:

“We observed that there is no player won the game against the computer (opponent). The results showed the results of 30 games (three trials given to each player), and it was found that the players win, lose, and draw the games against the computer are 0, 5, and 25 respectively.”

A grid of numbers and symbols

Description automatically generated

A graph with blue and yellow squares

Description automatically generated

Figure 6.

Experiment 2 [5]:

A structured implementation of the minimax algorithm for two-player games is presented in the related pseudocode. Nevertheless, the thoroughness of the game state investigation and the correctness of the evaluation functions determine how effective it is. To confirm its effectiveness in various gaming settings and optimize it for real-world use in certain applications, more research and testing are required.

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer

Description automatically generated

# Analysis / Discussion

Minimax Algorithm:

Advantage:

Because it computes every possible combination of moves in a game, this method is faultless. As a result, the computer either wins or draws while competing against a minimax algorithm, so it is used for less complicated systems like tic tac toe to become an unbeatable opponent.

Neural networks are not as accurate as minimax algorithms.

Disadvantage:

The primary drawback of the minimax algorithm is that it gets slow while playing complicated games like go, chess, and so on. With a high branching factor, this game design offers the player an enormous number of alternatives.

By using alpha-beta pruning, the minimax algorithm's shortcoming can be solved.

Alpha-Beta pruning:

Advantage:

1- Efficiency:

Through the pruning of branches that are unlikely to have an impact on the outcome, alpha-beta pruning dramatically lowers the total number of nodes evaluated during the search process. This makes the search process more effective, particularly in games like Tic Tac Toe that have big search trees.

2-Faster Execution:

The Alpha-Beta method usually runs faster than the conventional minimax algorithm since it has fewer nodes to assess. In situations where response time is crucial, such as real-time applications, this acceleration is especially helpful.

3-Impove Scalability:

Alpha-Beta pruning makes the search space smaller, which helps the algorithm scale better to deeper search depths or more complex games. For games with a high branching factor, like Go or Chess, this scalability is crucial.

4-Optimal moves:

Even with pruning, the Alpha-Beta method ensures optimal moves identical to those of the standard minimax algorithm. As a result, it keeps up its ability to play games like Tic Tac Toe when all the information is accurate.

Disadvantages:  
1-implementation complexity:

The minimax method must incorporate alpha-beta pruning, which requires meticulous record-keeping of alpha and beta values. Particularly for inexperienced programmers, this additional complexity may result in implementation problems.

Limited Effect in Easy Games: The advantages of Alpha-Beta pruning might not be as noticeable in games with a small search space, such as Tic-Tac-Toe, as they would be in games with a larger search space, like Go or Chess. It's possible that the efficiency gains are not worth the cost of maintaining alpha and beta values.

2-Memory Overhead:

While alpha-beta pruning lowers computational overhead, the necessity to keep more data (alpha and beta values) for every node in the search tree may result in an increase in memory overhead. This expense may be an impediment in environments with limited memory.

3- Doesn't Address Heuristic Evaluation:

While alpha-beta pruning improves the search procedure, it doesn't address the heuristic evaluation function's quality, which is necessary for determining how desirable game states are. Suboptimal play can still result from poorly developed heuristics, even in the case of Alpha-Beta pruning.

Heuristic Functions:

Advantages:

1-Improved Performance:

Heuristic functions direct the search towards more promising steps, therefore substantially reducing the search space. A well-thought-out heuristic can assess board positions in Tic Tac Toe rapidly, enabling the algorithm to go further into the game tree within a constrained search depth.

2-Faster Decision Making:

The implementation of a heuristic function speeds up decision-making for the algorithm by eliminating the requirement for comprehensive exploration of the whole game tree. This is especially helpful in situations or applications that call for instantaneous reactions.

Disadvantages:

1-Risk of Suboptimality:

It is possible for heuristic functions to misjudge the desirability of game states, which could result in less-than-ideal moves. In Tic-Tac-Toe, an ineffective heuristic could prioritize less crucial elements of the game or miss out on winning possibilities.

2-Complexity:

It can be difficult to create a useful heuristic function for Tic-Tac-Toe that strikes a balance between simplicity and accuracy. It requires an in-depth understanding of the strategic factors and game dynamics, which is sometimes difficult to encode algorithmically.

3-Computation vs. Accuracy Tradeoff:

Heuristic functions must balance accuracy and computing efficiency. While simpler heuristics could compromise accuracy for speed, more computationally demanding heuristics might yield higher evaluations but slow down the program.

##### References

1. Swaminathan B a , Vaishali R b and subashri T S Rb

a Professor, Dept of CSE, Rajalakshmi Engineering College, Chennai b UG Student, Dept of CSE, Rajalakshmi Engineering College, Chennai

1. Zain AM1,2, Chai CW1, Goh CC1, Lim BJ1, Low CJ1, Tan SJ1

1School of Computing, Faculty of Engineering, Universiti Teknologi Malaysia, Skudai, Johor, Malaysia

2UTM Big Data Centre, Ibnu Sina Institute for Scientific and Industrial Research, Universiti Teknologi Malaysia, Skudai Johor, Malaysia

1. INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY [IJIERT]

ISSN: 2394-3696 Website: ijiert.org

VOLUME 7, ISSUE 8, Aug.-2020

1. Alpha-Beta Pruning Carl Felstiner May 9, 2019
2. R. Manimegalai , A. Sheeba

Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research

Coimbatore, India.

1. Faculty of Computer Science, Bialystok University of Technology, Białystok, Poland
2. Gregory Kuhlmann, Kurt Dresner and Peter Stone Department of Computer Sciences, The University of Texas at Austin 1 University Station C0500, Austin, Texas 78712-1188

[8] International Journal of Advanced and Innovative Research (2278-7844) / # 389 / Volume 2 Issue 11 Computer Engineering Department, Islamic University Gaza, Palestine

**Phase two**

**Abstract**

This project explores the development of artificial intelligence (AI) player for the game of Tic-Tac-Toe, focusing on the integration of the Minimax algorithm enhanced with Alpha-Beta pruning and heuristic strategies. We applied heuristic search techniques to optimize decision-making processes, thereby improving both performance and efficiency. The core objective was to create an AI capable of competing against human players under many conditions while understanding game dynamics and how to deal with them.

**Introduction**

Games have long been a fundamental part of human culture, offering avenues for social interaction and creativity. Tic-Tac-Toe, a classic paper-and-pencil game, has intrigued both players and developers alike.

In the digital era, creating an AI-powered Tic-Tac-Toe simulation posed a challenge due to resource constraints. However, with the Minimax Algorithm, integrated with Alpha-Beta pruning and heuristic functions, we aimed to overcome these hurdles.

Our project focuses on developing an AI player capable of strategic gameplay, aligning with the evolution of gaming technology. Through this project, we aim to explore the intersection of AI and traditional gaming, trying to make the algorithm and the techniques can make decisions and think.

**Methodology**

* **Minimax algorithm**

**Minimax Algorithm:** The Minimax algorithm is like the brain of our Tic-Tac-Toe AI. It carefully looks at all the possible moves it can make and predicts how the opponent might respond. Then, it chooses the move that gives it the best chance of winning while also trying to avoid losing. This helps our AI player play smart and competitive games against human opponents.

**How does the minimax think and make a decision?** The algorithm starts from the current board state and simulates all possible moves for both the AI and the player for a given depth. It calculates the outcome (win, lose, draw) for each scenario.

**Recursive Evaluation:**

Minimax performs a recursive search on the game tree. For each possible move, it recursively evaluates all subsequent moves until a terminal state (win, lose, or draw) or a specified search depth is reached.

**Maximizing and Minimizing Scores:** If it's the AI's turn, the algorithm tries to maximize the score, assuming that the AI makes the best possible move.

If it's the player's turn, the algorithm assumes that the player will make the move that minimizes the AI's score (best move for the player).

**Backpropagation of Scores:**

Once the terminal states or maximum depth is reached, scores are backpropagated to the upper levels of the game tree. Each node (game state) receives a score based on the outcomes of the possible moves it can make.

**Decision Making:**

The AI selects the move that leads to the best score at the current level of the tree.

* **Alpha-Beta Pruning**

**Alpha-Beta Pruning:** It helps the AI player make decisions faster by cutting off parts of the decision-making process that won't lead to good outcomes. By doing this, the AI can explore more options and make better choices in less time.

* **Heuristic search**

**Heuristic search:** it speeds up decision-making in each movement. These are like quick tricks that help the AI player decide how good a move is without thinking too much. By using these tricks, the AI player can make decisions faster and adjust to different game situations better. This makes the game more fun and challenging for human players. With these tricks, our Tic-Tac-Toe AI is ready to face any opponent and give players an exciting gaming experience, no matter their skill level.

**Testing**

* **Cells:**

**1 2 3**

**4 5 6**

**7 8 9**

* **But Numbers shows on the coming images are the order of game moves**

**Minimax algorithm**

We tested various scenarios to assess the effectiveness of the Minimax algorithm in guiding the AI agent's decision-making process. These scenarios included different starting positions, opponent strategies, and search depths.

**1st Scenario:**

A screenshot of a game

Description automatically generated

**Moves Order:**

1. Cell 9: X (My move)
2. Cell 1: O (AI's move)
3. Cell 8: X (My move)
4. Cell 7: O (AI's move)
5. Cell 4: X (My move)
6. Cell 5: O (AI's move)
7. Cell 6: X (My move)
8. Cell 3: O (AI's winning move)

**Analysis of AI's Moves:**

**Move 2 - AI places O in Cell 1:**

After my first move (X in Cell 9, the board had many open possibilities. The AI placed an "O" in Cell 1. As The AI doesn’t have immediate winning moves but chooses a position that opens multiple potential

**Move 4 - AI places O in Cell 7:**

I responded to the AI's first move by placing “X" in Cell 8. Now, the board shows my possible threat forming a horizontal line in the third row. Minimax algorithm puts "O" in Cell 7 directly and blocks my chance to win by completing a horizontal line in the third row. This move also begins to set up a potential diagonal from top right to bottom left.

**Move 6 - AI places O in Cell 5**

I placed another "X" in Cell 4, The AI puts “O” in call 5 to try to complete the ts diagonal line from Cell 3 (top right) to Cell 7 (bottom left), using Cell 3,5,7.

**Move 8 - AI places O in Cell 3**

I placed another "X" in Cell 6 to give the AI a chance to win as I want to test it, The AI puts “O” in call 3 as expected to complete the s diagonal line from Cell 3 (top right) to Cell 7 (bottom left), using Cell 3,5,7 and now AI made the winning move and AI: 1, Player: 0.

**2nd scenario:**

A screenshot of a game

Description automatically generated

**Moves Order:**

1. Cell 3: X (My move)
2. Cell 1: O (AI's move)
3. Cell 6: X (My move)
4. Cell 9: O (AI's move)
5. Cell 5: X (My move)
6. Cell 4: O (AI's move)
7. Cell 7: X (My winning move)

**Analysis of AI's Moves:**

**Move 2 - AI places O in Cell 1:**

After my first move (X in Cell 3, the board had many open possibilities. The AI placed an "O" in Cell 1. As The AI doesn’t have immediate winning moves but chooses a position that opens multiple potential wins.

**Move 4 - AI places O in Cell 9:**

I responded to the AI's first move by placing “X" in Cell 8. Now, the board shows my possible threat forming a vertical line in the third column. The Minimax algorithm puts "O" in Cell 9 directly to block my chance to win by completing a vertical line in the third column. This move also begins to set up a potential diagonal from top left to bottom right.

**Move 5 - I place X in Cell 5**

As I noticed that AI tries to create a chance to a diagonal line. I placed another "X" in Cell 5 to ruin the plan.

**Move 6 - AI places O in Cell 4**

The AI puts “O” in cell 4 to try to replace the plan from a diagonal line to a vertical line which is cell1,4,7.

**Move 7 - AI places O in Cell 4**

I ruined the AI’s plan and I put “X” in cell 7 and by luck I won a game. All I wanted to ruin AI’s plan.

**3rd scenario:**

A screenshot of a game

Description automatically generated

**Moves Order:**

1. Cell 5: X (My move)
2. Cell 1: O (AI's move)
3. Cell 3: X (My move)
4. Cell 7: O (AI's move)
5. Cell 4: X (My move)
6. Cell 6: O (AI's move)
7. Cell 9: X (My move)
8. Cell 2: O (AI's last move)
9. Cell 8: X (My last move)

**Analysis of AI's Moves:**

**Move 1 – I place X in Cell 5:**

I started the game by putting “X” in cell 5 which starting Tic Tac Toe with cell 5 is an intelligent move to win as I don’t give AI any chance to make diagonal lines.

**Move 2 - AI places O in Cell 1:**

After my first move “X” in Cell 5, the board had many open possibilities. The AI placed an "O" in Cell 1. As The AI doesn’t have immediate winning moves but chooses a position that opens multiple potential wins as AI notices that any horizontal lines are blocked. So, AI tries to make a chance to win by rows or columns.

**Move 3 – I place X in Cell 3:**

I put X in cell 3 to complete my plan which is the diagonal line from top right to bottom left.

**Move 4 - AI places O in Cell 7:**

The Minimax algorithm puts "O" in Cell 7 directly and blocks my chance to win by completing the diagonal. This move also begins to set up a potential vertical in the first column.

**Move 5 – I place X in Cell 5:**

I ruined the plan by putting “X” in cell 5

**Move 6 - AI places O in Cell 6**

The AI puts “O” in cell without any future plan, all it matters that AI prevent me to complete my plan.

**Move 7 – I place X in Cell 9:**

I put “X” in cell 9 (I know it will end without a winner).

**Move 8 - AI places O in Cell 3**

AI puts the last “O” in cell 2. (It knows it will be end without a winner)

**Move 7 – I place X in Cell 8:**

I finished the game by putting the last “X” in cell 9 and no one won.

**Minimax with alpha beta pruning**

**A screenshot of a game

Description automatically generated**

When we applied the same moves we made in minimax algorithm the Minimax algorithm with Alpha-Beta pruning, we observe that the results are identical, this is actually an expected. The reasons for this are the nature of the game and the purpose of each algorithm.

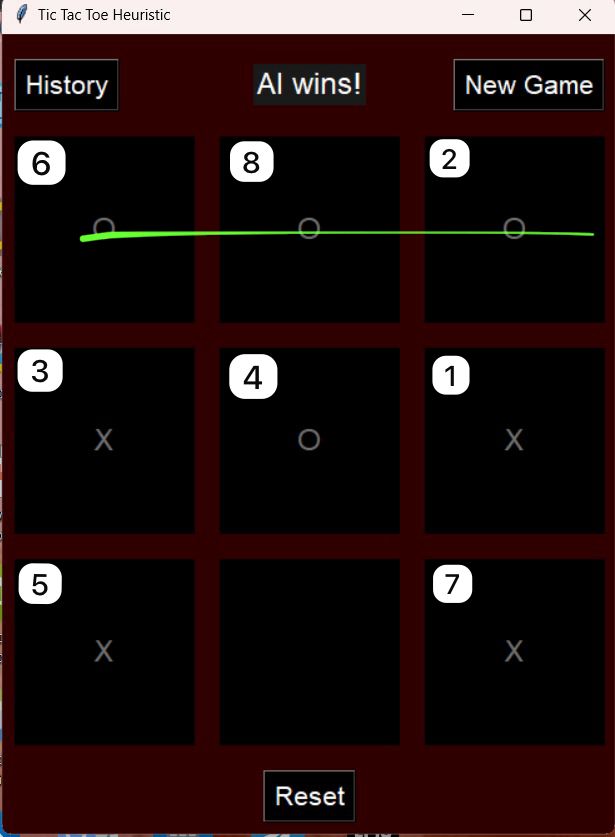
**But why did this happen?**

* Both Minimax and Alpha-Beta Pruning are designed to find the optimal strategy. Minimax evaluates all possible outcomes of all possible moves to decide the best move, Also, Alpha-Beta Pruning seeks to achieve the same by evaluating fewer nodes (Faster). The game’s moves and outcomes are based on the optimal moves identified by the search algorithm, which are the same whether all nodes are evaluated (Minimax) or only the necessary nodes (Alpha-Beta Pruning).
* where the game tree is relatively small and manageable, both algorithms are capable of exploring all relevant scenarios and will consequently reach the same decision regarding the best move.

**So, why alpha beta?**

While Alpha-Beta Pruning might not change the outcome in simple games like Tic Tac Toe, it offers a smarter way to apply Minimax, enabling the algorithm to handle the complexities of real-world applications much more efficiently and it speeds up the process. (Explained in details above)

**Minimax with heuristic search**



**Moves Order:**

1. Cell 6: X (My move)
2. Cell 3: O (AI's move)
3. Cell 4: X (My move)
4. Cell 5: O (AI's move)
5. Cell 7: X (My move)
6. Cell 1: O (AI's move)
7. Cell 9: X (My move)
8. Cell 2: O (AI's wining move)

**Analysis of AI's Moves:**

**Move 1 - Cell 6: X (My first Move)**

I started with “X” on a side middle position to make a chance to win by horizontal and vertical lines.

**Move 2 - Cell 3: O (AI's Move)**

The AI chooses a cell 3 corner, which is a strong response. Corners are valuable because they are part of multiple potential winning combinations (two diagonals and one row/column each).

**Move 3 - Cell 4: X (My Move)**

I placed “X” on cell 4. This move begins to set up a horizontal line in the middle row.

**Move 4 - Cell 5: O (AI's Move)**

The AI blocks my plan to win by a horizontal line in the middle row. This is a critical defensive move, as it prevents an immediate win setup and utilizes the central position to create multiple line threats.

**Move 5 - Cell 7: X (My Move)**

I continue to focus on the bottom row, trying to line up three X's horizontally. This also opens potential for a vertical line depending on the AI's response.

**Move 6 - Cell 1: O (AI's Move)**

The AI picks another corner, setting up for multiple potential wins and blocking my possible vertical line from Cell 7.

**Move 7 – Cell 9: X (My Move)**

I choose cell “9” to not make AI wins by the diagonal line (cells 1,5,9) and to create a second threat by aligning diagonally with Cell 5 and threatening the third row horizontally.

**Move 8 - Cell 2: O (AI's Winning Move)**

The AI completes a vertical line (Cells 2, 5, 8). This move shows the AI’s focus on central and strategic placement to block my moves.

**NOTE: No one can beat him. Now, I can say “him” not “it”. Every move makes sense**

**Evaluation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method / Feature** | **performance** | **efficiency** | **adaptability** |
| **Minimax algorithm** | The standard Minimax algorithm performs a complete search of the game tree, ensuring an optimal move selection at the cost of higher computation time, especially in games with a large number of possible moves​. | It is less efficient because it examines every possible move in the game tree, leading to a significant computational cost as the depth of the tree increases | While the algorithm is strong in ensuring optimal play, its adaptability is limited as it does not incorporate any mechanisms to adjust to varying game complexities or to optimize calculations dynamically |
| **Minimax with alpha beta pruning** | Alpha-beta pruning is using to eliminate branches of the tree that aren’t used, which can potentially lead to the same optimal decision but faster​ | More efficient than the solo Minimax as it cuts down on the number of nodes evaluated by pruning unnecessary branches, thus speeding up the decision-making process without changing the quality of the output​ | It offers some adaptability by reducing computational overhead, which makes it suitable for complex or larger game trees than what standard Minimax can handle effectively. However, its performance improvement is highly dependent on the order and depth of moves​ |
| **Minimax with heuristic function** | Integrates heuristic evaluations that approximate the game states, allowing the algorithm to focus on more promising branches of the game tree. This often results in faster decision times. | Heuristic function increases efficiency by reducing the breadth of the search needed. This makes the algorithm faster at the cost of potentially missing the absolute best move if the heuristic is not perfectly aligned with the game's strategy. | Highly adaptable to various game scenarios as it can adjust the search based on heuristic evaluations which consider the current state of the game. This adaptability makes it suitable for games with a high degree of complexity and varying states​. |

**Quantitative Results**

**Minimax:**

Nodes Processed: Processes every possible move in the game tree.

Time Taken: Longest, due to exhaustive searching.

**Minimax with Alpha-Beta Pruning:**

Nodes Processed: Fewer than standard Minimax due to pruning inefficient branches.

Time Taken: Reduced significantly compared to standard Minimax, dependent on move ordering.

**Minimax with Heuristic Function:**

Nodes Processed: Fewest among the three

Time Taken: Shortest, thanks to efficient heuristic evaluations leading to quick decisions.

**Qualitative Results**

**Minimax:**

Accuracy **(High)**: Finding the optimal move, leading to a win or a draw if possible.

**Minimax with Alpha-Beta Pruning:**

Accuracy **(High):** has the same optimal decision-making of standard Minimax.

**Minimax with Heuristic Function:**

Accuracy **(High)**: unbeatable, it always finds either a winning move or the game ends with a draw.

**Expected Outcomes**

**Minimax**: It’s expected to perform optimally but at the cost of efficiency, making it less ideal for real-time applications or more complex games.

**Minimax with Alpha-Beta Pruning**: improves upon the efficiency of the standard Minimax without losing any accuracy. It is better suited for games that have a moderate to high complexity, providing a balance between performance and resource usage.

**Minimax with Heuristic Function:** it provides high accuracy with minimal resource usage. Its ability to always do at least a draw, if not a win, makes it particularly valuable for real-time or complex scenarios where decision speed is critical. **(Can not be beaten)**

**Overview of the AI Agent**

**Game:** Tic-Tac-Toe

**Algorithm Base:** Minimax Algorithm

**Enhancements:** Alpha-Beta Pruning and Perfect Heuristic Function

**Key Features of the AI Agent:** Alpha-Beta Pruning, Advanced Heuristic Evaluation, Decision-Making Process, Adaptability and Learning Capability, Different levels of intelligence.

**Challenges:**

* Computational Complexity
* Decision Speed
* Optimality vs. Practicality
* Development of Heuristic Function which can’t be beaten was such a challenge!

**Conclusion**

In our project, we used the well-known Minimax algorithm in Tic Tac Toe game and made it better. We added Alpha-Beta pruning and heuristic functions to help it make decisions faster and smarter. We discovered the basic Minimax is really interesting, but it can be slow because it thinks about every possible move, even the ones that aren’t that important. With Alpha-Beta pruning, we helped it ignore the moves that don’t really help, which made our AI faster and sharper. Then, by using heuristics, we made our AI even better at adapting. Instead of just following rules, it started to look at the game in a different point of view and make smarter choices based on what was happening at the moment. As we finish up, we see that these improvements make a big difference. There’s a lot more we can do, too. We could try to make the AI learn while it’s playing, which could be really exciting.