An Intelligent Cubic Player using the Minimax Algorithm, Alpha-Beta Pruning, and Heuristic Functions.

TEAM MEMBERS :

Shereen Mohamed 20230289 (IS)
Reham Ahmed 2023022 (IS)
Hend Adel 20230648 (CS)
Nadeen Tarek 20230621 (CS)
Ola Salama 20230347 (IS)
Nada Waleed 20230626 (IS)

1.Introduction and Overview

1.1 Project Idea and Overview

The main goal of this model is to develop an intelligent version of the classic Tic-Tac-Toe game, transforming it from a simple 2D board into a full 3D (4×4×4)environment. Winning in this extended version requires forming four consecutive pieces in any possible direction: horizontal, vertical, diagonal, or 3D space diagonal.

> The system is is implemented as a **desktop application** using Python and the Tkinter GUI library to enable gameplay between a human player and an AI agent, where the AI is capable of responding intelligently by selecting the optimal move within a limited search depth. The AI uses advanced adversarial search algorithms to evaluate board states and choose the best possible action

---

1.2 Similar Applications

Several existing applications and games use similar concepts, such as:

- **Classic Tic-Tac-Toe games using Minimax**
- **Chess and Checkers engines using Alpha-Beta pruning**
- **Connect-Four AI systems using heuristic-based evaluation**

Unlike standard implementations, this project extends the problem into a three-dimensional board, which dramatically increases the state space and computational complexity, requiring optimizations such as pruning and heuristic-based move selection.

1.3 Literature Review

1. Minimax Algorithm in Game Playing

The Minimax algorithm is a fundamental approach for solving two-player, zero-sum games with perfect information. As explained in the Minimax lecture notes provided by Concordia University, the algorithm models the game as a search tree in which players alternate between maximizing and minimizing utility values. Terminal game states are assigned utility values representing win, loss, or draw outcomes, and these values are propagated upward through the tree to determine the optimal move.

However, the notes emphasize that the computational complexity of Minimax grows exponentially with the branching factor and search depth. This makes the pure Minimax approach impractical for games with large state spaces, motivating the need for optimization techniques such as pruning and heuristic evaluation.

---

2. Alpha-Beta Pruning as an Optimization Technique

Alpha-Beta pruning was introduced to improve the efficiency of the Minimax algorithm without affecting the correctness of the final decision. The paper by Knuth and Moore provides a formal analysis of the Alpha-Beta algorithm and demonstrates that it can significantly reduce the number of nodes evaluated during search. The technique works by

maintaining two bounds: alpha, representing the best value found so far for the maximizing player, and beta, representing the best value for the minimizing player.

When the algorithm determines that a branch cannot influence the final decision, it prunes that branch and avoids further exploration. The analysis shows that, with good move ordering, Alpha-Beta pruning can reduce the effective complexity of the search dramatically, making deeper searches feasible in practice.

3. Heuristic Evaluation Functions

In many realistic game scenarios, searching the game tree until terminal states is computationally infeasible. The Stanford General Game Playing reading by Clune discusses the role of heuristic evaluation functions in estimating the desirability of non-terminal game states. A heuristic function assigns a numerical score to a state based on features such as piece alignment, control of important positions, or proximity to winning configurations.

The use of heuristics allows the search to be depth-limited while still producing strong decision-making behavior. The reading highlights that the quality of the heuristic function has a direct impact on both the performance and the playing strength of the AI agent.

---

4. Limitations of Existing Work and Motivation

The reviewed materials primarily focus on traditional board games and lower-dimensional environments, where the state space is relatively manageable. The MSc thesis reviewed from Skemman University further discusses the challenges of scaling search-based AI techniques as the complexity of the game environment increases. High-dimensional games introduce significantly larger state spaces and branching factors, which exacerbate the limitations of classical Minimax search.

These limitations motivate the integration of multiple optimization strategies, including Alpha-Beta pruning, heuristic evaluation, and move selection techniques, when dealing with complex environments.

---

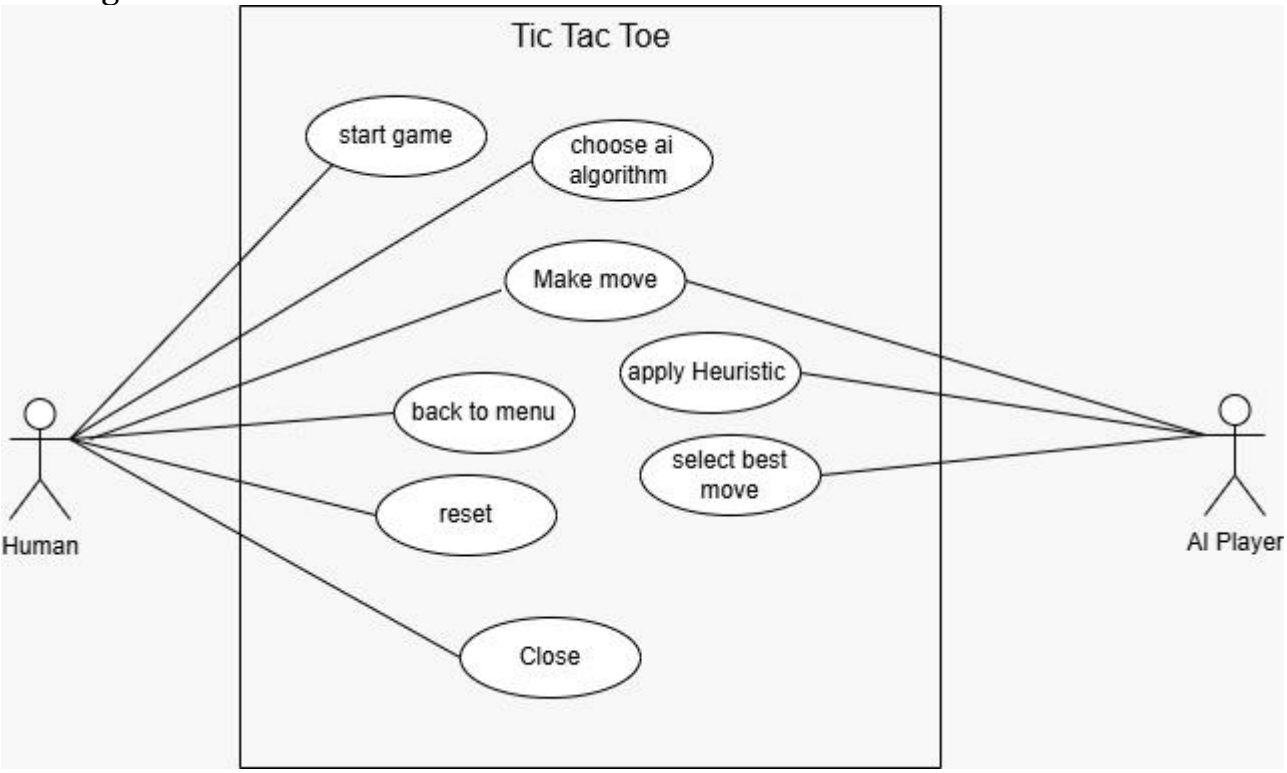
2. Proposed Solution

2.1 Main Functionalities (User Perspective)

From the user's point of view, the system provides the following features:

- Play 3D Tic-Tac-Toe against an AI opponent
- Select AI algorithm (Minimax or Alpha-Beta Pruning)
- Select heuristic evaluation function
- Visual representation of the 3D board
- Game timer and win highlighting
- Reset and replay functionality

The user interacts with the system via a graphical user interface (GUI), making moves by clicking on cells.



---

3. Representation of the States, Actions, and the State Space

3.1 State Representation

Each state is represented by a **4×4×4 three-dimensional array**, where each cell can be:

- Empty (None)
- Occupied by player X
- Occupied by player O

The current player is implicitly represented by the depth of the search and whether the algorithm is maximizing or minimizing.

3.2 Action Representation

An action corresponds to placing the player's symbol in an empty cell.
Actions are represented as tuples (x, y, z) such that:

$0 \leq x, y, z < 4$

To reduce the branching factor, only **promising moves near occupied cells** are explored.

3.3 State Space

The theoretical state space size is:

$$3^{64}$$

which is computationally infeasible.
Therefore, the effective search space is reduced using:

- Depth-limited search
- Heuristic evaluation
- Alpha-Beta pruning
- Move ordering and promising-move selection

---

4. Applied Algorithms

4.1 Minimax Algorithm

Minimax is an adversarial search algorithm that assumes:

- The AI tries to maximize the score
- The opponent tries to minimize it

The algorithm explores future game states up to a fixed depth and uses a heuristic function to evaluate non-terminal states.

Implementation Details

The core logic is encapsulated in the minimax function , which takes the current game state, the search depth, and a boolean flag **(maximizing_player)** as arguments.

•Base Cases : The recursion terminates when one of the following conditions is met:

1.A winning state is reached for the AI (returns a high positive score, e.g., +10000).

2.A winning state is reached for the opponent (returns a high negative score, -10000).

3.The maximum search depth is reached (depth == 0). In this case, the function returns the score from the selected Heuristic Evaluation Function.

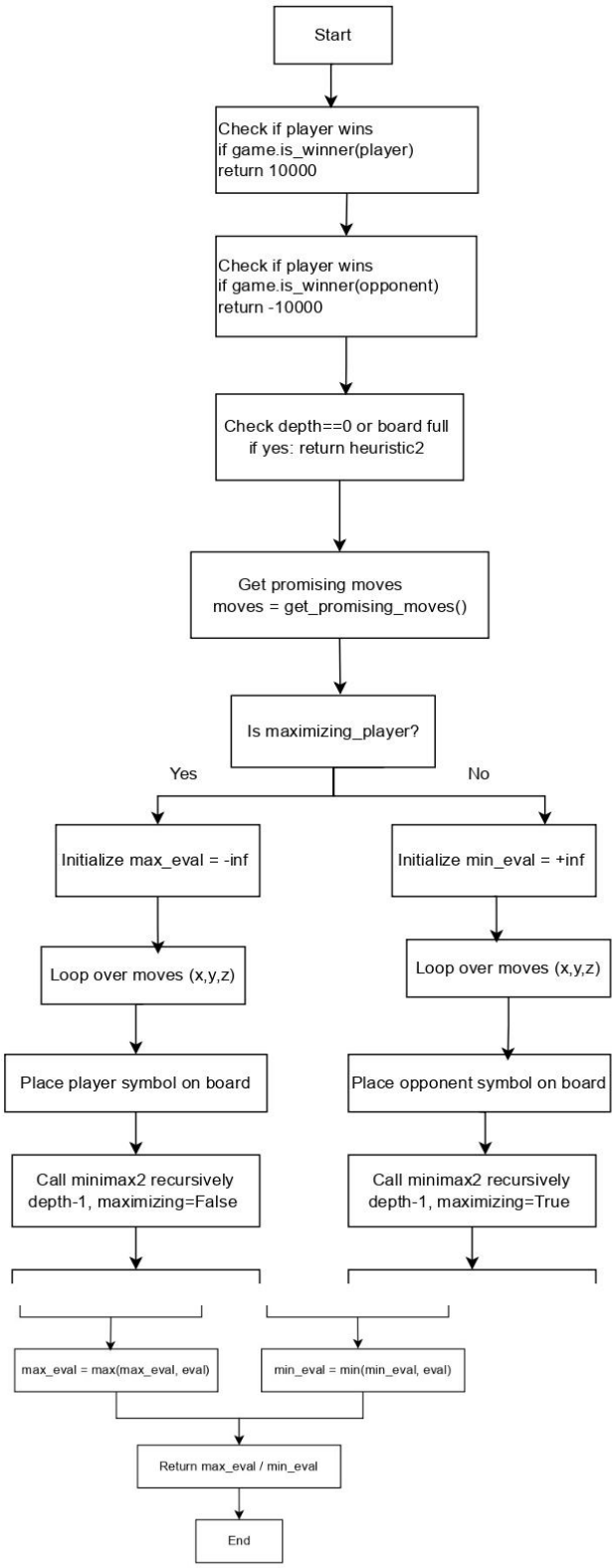
4.The board is full **(game.is_full())**, resulting in a draw (returns the heuristic score).

•Recursive Step:

•Maximizing Player (AI): The function iterates through all available moves, makes a move, recursively calls minimax for the minimizing player, and selects the move that yields the maximum score.

•Minimizing Player (Opponent): The function iterates through all available moves, makes a move, recursively calls minimax for the maximizing player, and selects the move that yields the minimum score.

The implementation utilizes the **get_promising_moves(game)** function to generate the list of moves to be explored, effectively integrating the Heuristic Reduction technique into the Minimax search.



block diagram:minimax algorithm

4.2 Alpha-Beta Pruning

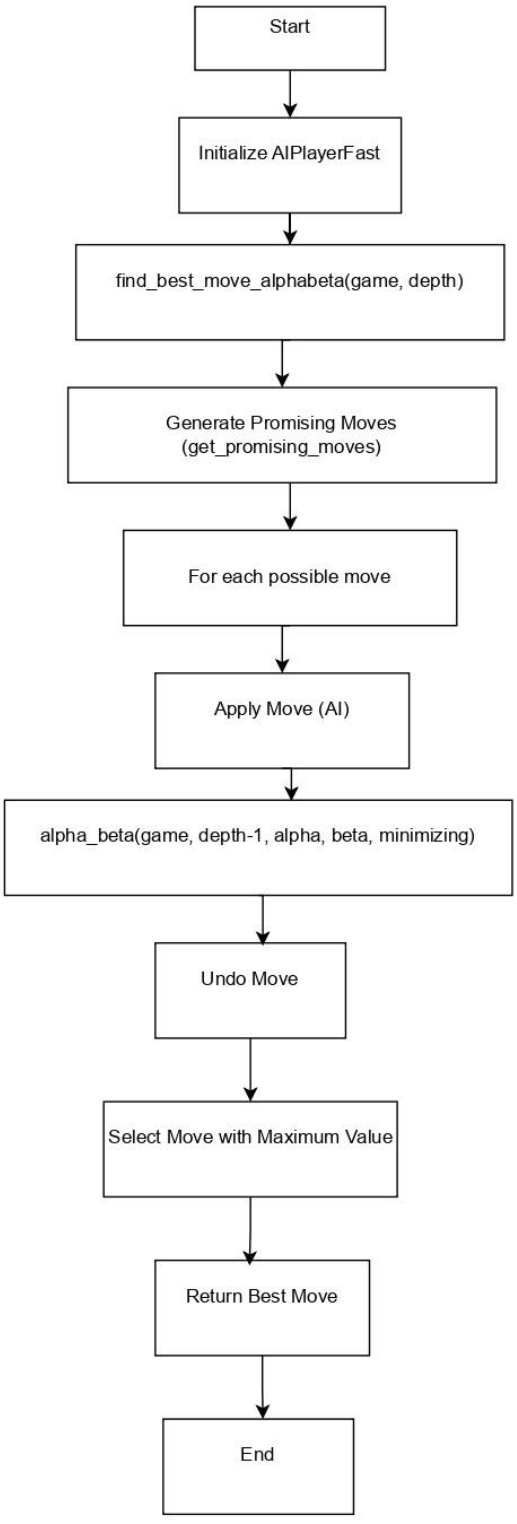
Alpha-Beta pruning is an optimization of Minimax that eliminates branches that cannot affect the final decision.

- **Alpha**: best value for the maximizing player
- **Beta**: best value for the minimizing player

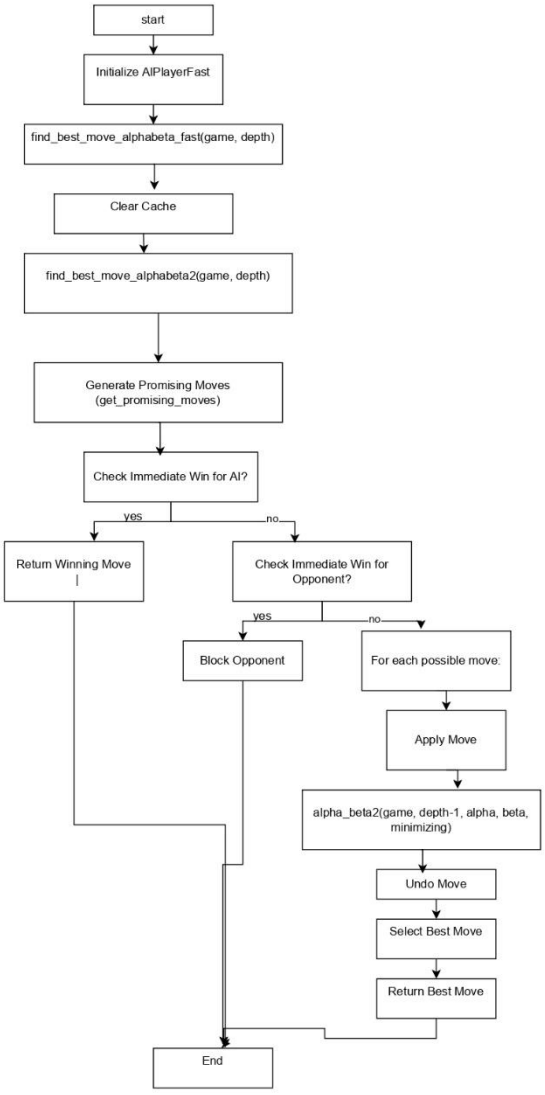
This significantly improves performance, allowing deeper searches.
Alpha-Beta Pruning is an optimization technique applied to the Minimax algorithm to reduce the number of nodes evaluated in the search tree without affecting the final result.

Implementation Details

The logic is implemented in the alpha_beta function , which adds two parameters: (alpha) and (beta).
Alpha : Represents the best value (highest score) found so far for the maximizing player along the current path. It is initialized to .
Beta : Represents the best value (lowest score) found so far for the minimizing player along the current path.
•Maximizing Player: If the current best score (value) is greater than or equal to beta, the remaining moves for this node are skipped , as the minimizing player will never allow the game to reach this state.
•Minimizing Player: If the current best score (value) is less than or equal to alpha, the remaining moves for this node are skipped, as the maximizing player will never allow the game to reach this state.
The Alpha-Beta implementation also includes a basic Transposition Table to store and retrieve the results of evaluated game states.



Block diagram:Alphabeta+h1



Alphabeta+h2

4.3 Heuristic Evaluation Functions

Two heuristic functions are used:

Heuristic 1 – Line-Based Evaluation

- Scores lines based on the number of symbols in a potential winning line
- Strongly rewards 3-in-a-line situations
- Penalizes opponent threats

Heuristic 2 – Positional & Aggressive Heuristic

- Extends heuristic 1
- Adds positional bonuses for:
  - Center cells
  - Corner cells
- Heavily rewards near-winning states
- Strongly penalizes opponent threats

This heuristic provides better performance in mid-game scenarios.

6.Experiments and Results

5.1 Experimental Setup

The AI was tested against:

- Human players
- Different depths
- Different heuristics

Metrics observed:

- Win/loss ratio
- Response time
- Quality of decisions

We evaluated the following approaches :

1. **Heuristic 1 only**
2. **Heuristic 2 only**
3. **Minimax + Heuristic 1**
4. **Minimax + Heuristic 2**
5. **Alpha-Beta + Heuristic 1**
6. **Alpha-Beta + Heuristic 2**

5.2 Results

The experimental results showed clear differences in both performance and gameplay quality among the tested approaches.

- **Greedy approaches (Heuristic 1 and Heuristic 2)**
  These methods were extremely fast and had minimal response time. However, they often made short-sighted decisions, especially in mid- and late-game situations. Heuristic 2 performed noticeably better than Heuristic 1 due to its positional awareness, but both lacked long-term planning.
- **Minimax-based approaches**
  Using Minimax significantly improved the AI's ability to anticipate future outcomes and block opponent strategies. Minimax combined with Heuristic 2 consistently produced better decisions than with Heuristic 1. However, the increased search depth resulted in higher response times, making deeper searches computationally expensive.
- **Alpha-Beta pruning approaches**
  Alpha-Beta pruning achieved the same decision quality as Minimax while exploring significantly fewer game states. When combined with Heuristic 1, it offered a strong balance between speed and accuracy. The best overall performance was achieved using **Alpha-Beta pruning with Heuristic 2**, which demonstrated strong offensive and defensive play while maintaining acceptable response times.

Overall, the experiments confirm that incorporating pruning techniques and advanced heuristics greatly enhances both the efficiency and strategic strength of the AI player.

---

6. Analysis, Discussion, and Future Work

6.1 Analysis and Insights

The experimental results clearly demonstrate that the choice of both the search algorithm and the heuristic evaluation function has a significant impact on the AI player's performance in 3D Tic-Tac-Toe.

A general observation from the experiments is that search-based approaches consistently outperform greedy strategies. While greedy heuristics can provide fast decisions, they lack the ability to anticipate future consequences, which is crucial in a game with a large branching factor such as 4×4×4 Tic-Tac-Toe.

Another key insight is that the quality of the heuristic function directly affects the effectiveness of the search algorithm. Even with the same search method, using a stronger heuristic led to better strategic decisions, improved win rates, and more human-like gameplay.

6.2 Advantages and Disadvantages

Detailed Comparison of the Approaches

1. Minimax + Heuristic 1

This approach combines the classic Minimax algorithm with a simple line-based heuristic.

Advantages:

- Able to look ahead and anticipate opponent moves.
- Stronger than greedy heuristics in blocking immediate threats.
- Conceptually simple and easy to implement and analyze.

Disadvantages:

- High computational cost due to exploring a large number of game states.
- Response time increases rapidly with deeper searches.
- Limited strategic depth because Heuristic 1 focuses mainly on line counts and ignores positional importance.

Discussion:

This approach performs reasonably well in early and mid-game scenarios but struggles in complex board states. The lack of positional awareness causes the AI to miss strategically strong moves, even though Minimax explores future possibilities.

---

2. Minimax + Heuristic 2

This approach enhances Minimax with a more sophisticated heuristic that includes positional and aggressive scoring.

Advantages:

- Significantly better move quality compared to Minimax + Heuristic 1.
- Better long-term planning due to positional awareness (centers and corners).
- More consistent performance against human players.

Disadvantages:

- Still computationally expensive at higher depths.
- Increased evaluation complexity slightly adds to response time.

Discussion:

The experiments showed that improving the heuristic alone can greatly enhance performance, even without changing the search algorithm. However, the high computational cost of Minimax remains a major limitation, especially for real-time gameplay.

---

3. Alpha-Beta Pruning + Heuristic 1

This configuration applies Alpha-Beta pruning to Minimax while keeping the simple heuristic.

Advantages:

- Significantly reduces the number of evaluated nodes.
- Faster response time compared to Minimax-based approaches.
- Maintains the same decision quality as Minimax at the same depth.

Disadvantages:

- Decision quality still limited by the simplicity of Heuristic 1.
- May overlook strong positional moves in complex scenarios.

Discussion:

Alpha-Beta pruning proved highly effective in improving efficiency. Even with a basic

heuristic, pruning allowed deeper searches within the same time constraints, making this approach more practical than pure Minimax.

---

4. Alpha-Beta Pruning + Heuristic 2

This is the most advanced and strongest configuration tested.

Advantages:

- Best overall performance in terms of win rate and decision quality.
- Efficient search due to pruning combined with a powerful heuristic.
- Produces strategic, human-like gameplay.
- Balanced offense and defense, especially in late-game situations.

Disadvantages:

- Most complex to implement and tune.
- Requires careful heuristic design to avoid bias or overfitting.

Discussion:

This approach consistently outperformed all others. The combination of pruning and a strong heuristic allowed the AI to explore deep game trees efficiently while making high-quality decisions. It represents the best trade-off between performance and computational efficiency.

Overall Insights

From the analysis, the following insights can be drawn:

- **Search depth alone is not sufficient**; the heuristic function plays a critical role in guiding the search.
- **Alpha-Beta pruning is essential** for making Minimax feasible in large state spaces.
- **Heuristic 2 consistently improves decision quality**, regardless of the underlying search algorithm.
- The combination of **efficient pruning and informed evaluation** produces the most robust AI behavior.

Future Work

Several extensions and improvements can be explored in future work:

- **Adaptive search depth**, where the AI dynamically adjusts depth based on game complexity.
- **Enhanced symmetry reduction**, allowing the AI to recognize equivalent board states more effectively and further reduce computation.
- **Learning-based heuristics**, using reinforcement learning or neural networks to automatically tune evaluation functions.
- **Monte Carlo Tree Search (MCTS)** as an alternative to Minimax-based approaches for large search spaces.
- **Parallel search optimization**, leveraging multi-threading to speed up decision-making.

References

https://kodu.ut.ee/~ahto/eio/2011.07.11/ab.pdf
https://users.encs.concordia.ca/~kharma/coen352/Project/Minimax-notes.pdf
https://skemman.is/bitstream/1946/23743/1/MSc.pdf
http://ggp.stanford.edu/readings/cluneplayer.pdf
https://www.cs.rochester.edu/~brown/242/assts/studprojs/ttt10.pdf

Project folder link : https://drive.google.com/file/d/11ioXne3DBDf-s9j8O6h4yqYql1olB2EW/view?usp=sharing