Design and Implementation of a Minimax-Based AI Agent in the Chain Reaction Game

Ahmad Syafiq - 13523135
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ahmad.syafiq2005@gmail.com , 13523135@std.stei.itb.ac.id

***Abstract*—This paper presents the design and implementation of a Minimax-based AI agent for the two-player Chain Reaction game, adapted from JindoBlu's mobile version. The game is modeled on a 5×5 grid with uniform critical mass and strict ownership rules, resulting in a reactive environment where a single move can trigger complex cascading explosions. The AI agent employs the Minimax algorithm with alpha-beta pruning, enhanced through game state cloning, transposition caching, and parallel move evaluation to ensure both correctness and efficiency. Two purely quantitative evaluation strategies are explored: one based on total orb count and another based on the number of owned cells . Although CellEvalStrategy is computationally simpler, it consistently enables more flexible and resilient gameplay by promoting spatial control and preserving legal move options. Experimental results across human, random, and AI-versus-AI matchups show that the agent performs competitively, even with a shallow search depth of five. Furthermore, the inherent advantage of playing second—due to the game's asymmetric opening rules—is shown to significantly affect outcomes, reinforcing the importance of turn order alongside evaluation design. These findings confirm that classical adversarial search, when paired with domain-aware heuristics, remains highly effective for dynamic, deterministic games like Chain Reaction.**

Keywords—Artificial Intelligence, Minimax, Alpha-Beta Pruning, Chain Reaction, Game AI, Evaluation Function, Heuristic Strategy, Turn-Based Game

I. Introduction

Artificial Intelligence (AI) has long played a pivotal role in games, both as a benchmark for AI research and in creating challenging opponents for human players. A landmark achievement in game AI was IBM's Deep Blue defeating world chess champion Garry Kasparov in 1997. This victory – the first time a computer beat a reigning world champion under tournament conditions – marked an inflection point in computing, heralding a future where machines could rival human experts in strategic thinking. Such milestones illustrate the impact of AI in games and validate game-playing as a fertile ground for developing and evaluating AI techniques.

Many classical games can be modeled as two-player zero-sum competitions, where one player's gain is the other's loss. Game-theoretic AI agents for these domains commonly rely on the minimax search algorithm, which assumes both players act optimally to minimize the opponent's payoff. The minimax approach exhaustively explores game state trees to identify optimal moves, but its brute-force application is computationally expensive due to the exponential growth of possibilities. A crucial enhancement is alpha-beta pruning, which eliminates branches that cannot influence the final decision, thereby significantly reducing the search space. Alpha-beta pruning is known to effectively double the achievable search depth compared to naive minimax search given the same computational resources. These techniques (minimax and alpha-beta) have become standard in AI for sequential perfect-information games such as Chess and Checkers [1]. We leverage this rich theoretical foundation as the basis for our AI agent's decision-making (detailed in Section II).

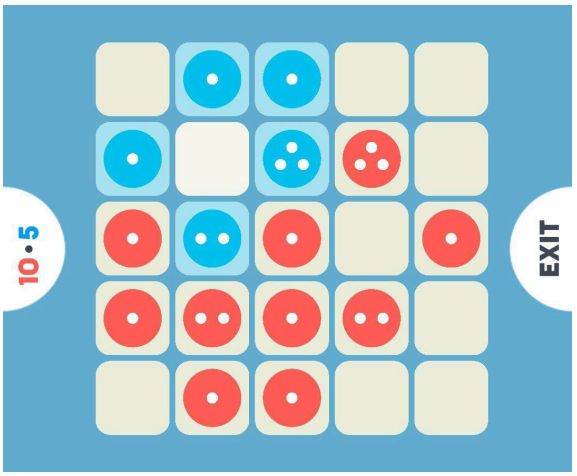

Fig 1.1 JindoBlu's Chain Reaction
(source: JindoBlu's 2 Player games)

One modern game that presents an interesting challenge for such AI methods is JindoBlu's 2 Player games' version of Chain Reaction, a strategic board game developed as an Android app. This version of Chain Reaction is a deterministic, turn-based game played on a 5x5 grid where players alternately place their colored "orbs" in cells they already control. When a cell's orb count reaches its critical mass, which is 4, it explodes, sending one orb into each adjacent cell and taking over those cells for the exploding

player. This chain reaction can cascade – an explosion may overload neighboring cells and trigger further explosions – and it continues until the board stabilizes. If an explosion hits an opponent's orb, that cell is captured (converted to the exploding player's color). The goal is to eliminate the opponent by eventually causing all of their orbs to be removed or converted. Despite its simple rules, Chain Reaction exhibits complex and unpredictable dynamics: a position that appears dominant can quickly collapse after a well-timed chain of explosions Naive heuristics (such as "having more orbs is always better") often fail, as dramatic comebacks are possible when a critical mass is reached. These properties make the game an excellent testbed for AI – it is a perfect-information zero-sum game like chess, but with explosive state transitions that pose a significant search and evaluation challenge.

In this paper, we present the design and implementation of a minimax-based AI agent for the two-player Chain Reaction game. The remainder of the paper is organized as follows: Section II provides background theory on zero-sum games and the minimax search algorithm with alpha-beta pruning. Section III describes the mapping of the Minimax Algorithm for Chain Reaction. Section IV details the experimental setup and results, evaluating the AI's performance against human players and other AI. Finally, Section V offers concluding remarks and discusses potential improvements and future work in extending the AI agent's capabilities.

II. Theoretical Foundations

A. Zero-Sum Game

Merriam-Webster defines zero-sum games as a situation in which one person or group can win something only by causing another person or group to lose it. In game theory, a *zero-sum game* is one in which one player's gain is exactly balanced by the other player's loss. Formally, the sum of payoffs to all players remains constant (often zero) for any outcome. In the context of two-player deterministic games (with no chance elements and perfect information), this means the interests of the two opponents are strictly opposed – a win for one player implies an equivalent loss for the other. Classic board games like chess or tic-tac-toe are well-modeled as zero-sum: both players have opposing goals and every advantage for one side comes at the expense of the other. Such two-player deterministic zero-sum settings provide a foundation for adversarial search algorithms, as optimal play by one player is directly aimed at minimizing the payoff of the opponent.

B. Minimax Algorithm

The minimax algorithm is a fundamental decision-making strategy for two-player zero-sum games under adversarial conditions. It assumes that one player (Max) attempts to maximize the outcome value while the opponent (Min) tries to minimize it. The algorithm explores the game's decision tree, evaluating terminal outcomes (or using a heuristic evaluation for non-terminal states at a depth limit) to propagate values backward. At a Max node (a state where the maximizing player is to move), the highest value among the child states is chosen; at a Min node (opponent's turn), the lowest child value is chosen. This backward induction models optimal play by both sides, effectively planning for the worst-case opponent response. Given enough time and space to search the full game tree, minimax will converge on the optimal move for the maximizing player. Formally, let $V(s)$ be the minimax value of a game state s (from Max's perspective). If s is terminal, $V(s) = U(s)$ is the utility (payoff) for that terminal outcome. Otherwise, one can define the minimax value recursively as:

$$V(s) = \begin{cases} \max_{a \in A(s)} V(s_a), & \text{if } s \text{ is a Max node;} \\ \min_{a \in A(s)} V(s_a), & \text{if } s \text{ is a Min node,} \end{cases}$$

where $A(s)$ is the set of legal actions in state s and s_a denotes the successor state after action a. This equation encapsulates the minimax principle that Max pursues the move with maximum guaranteed payoff while Min responds to minimize the payoff. The minimax procedure is provably optimal against an optimal adversary, making it a reliable baseline for two-player zero-sum games.

C. Alpha-Beta Pruning

While the minimax algorithm finds optimal moves, it can be computationally expensive as the game tree grows exponentially with depth. Alpha-beta pruning is an optimization that significantly reduces the number of nodes evaluated, without affecting the final result of minimax. The key idea is to "prune" away branches of the game tree that cannot possibly influence the final decision. As the search progresses depth-first, it keeps track of two thresholds: α, the best (highest) value found so far for Max (a lower bound on the outcome Max can guarantee), and β, the best (lowest) value found so far for Min (an upper bound on the outcome Min can guarantee). If at any point in the traversal these bounds overlap such that $\beta \leq \alpha$, it implies that the current node cannot lead to an outcome better than what has already been found along another path. In such a case, further exploration of that branch is futile and it is pruned (cut off), thereby skipping a large number of node evaluations that are irrelevant to the final minimax decision.

Alpha-beta pruning does not alter the result of the minimax algorithm; it simply avoids exploring moves that are suboptimal given the information already gathered. In the best-case scenario (when the game tree is traversed in an optimal move order such that the most promising moves are evaluated first), alpha-beta can reduce the effective branching factor dramatically. In fact, with ideal ordering, alpha-beta examines on the order of $O(b^{d/2})$ nodes instead of $O(b^d)$ for a game tree of branching factor b and depth d. This means the search can go roughly twice as deep in the same computation time compared to naive minimax. (By contrast, in the worst-case with very poor move ordering, alpha-beta reverts to examining $O(b^d)$ nodes, offering no improvement over basic minimax.) The significant pruning happens when high-value moves for Max or low-value moves for Min are discovered early, causing many sibling branches to be cut off. Thus, ordering moves from best to worst (for Max) and worst to best (for Min) yields the maximum pruning benefit. In summary, alpha-beta pruning exploits the minimax principle to ignore portions of the search tree that cannot affect the final choice,

thereby expediting the decision process under adversarial conditions.

D. Game Tree Representation

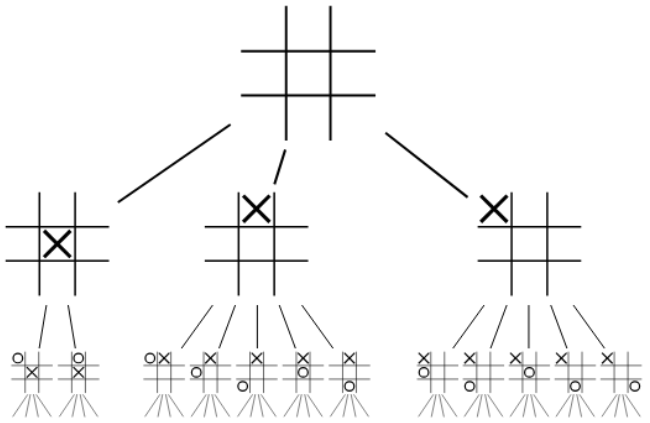

Fig. 2.1 Game Tree Representation of Tic-Tac-Toe (source: wikipedia.org)

Two-player deterministic games are naturally modeled as a game tree, where nodes represent game states and edges represent the players' moves (actions). The game begins at an initial state (the root of the tree), and each possible action leads to a child node representing the resulting state. Play alternates between the two players, so the tree's levels (plies) alternate between Max's turn and Min's turn. This yields a branching structure: from the root, one level down are all the states resulting from Max's possible first moves; the next level contains states resulting from each of Min's responses, and so on. This expansion continues until terminal states (leaves) are reached – these leaves correspond to end-of-game outcomes (win, lose, or draw positions, each with an assigned utility value). Each path from the root down to a leaf thus represents a complete sequence of moves (a possible play of the game from start to finish). This tree representation of game states and decisions is the foundation for minimax search: it provides the structure through which the algorithm can traverse all potential move sequences, evaluate their outcomes, and back up the results to inform the optimal move at the root. In implementing a minimax-based AI agent, the game tree is typically generated on the fly up to a certain depth (due to complexity constraints), but conceptually the agent is exploring this implicit tree of decisions and consequences to decide on the best move.

III. Mapping the Chain Reaction Game to Minimax and Alpha-Beta Pruning

To enable the application of the Minimax algorithm with alpha-beta pruning to the Chain Reaction game, the game must first be mapped into a formal problem representation suitable for adversarial search. In this mapping, the entire game board is modeled as a two-dimensional grid, where each cell holds information about the number of tokens it contains and which player currently controls it. The complete configuration of the board, along with the current player's turn, constitutes the "state" in the search tree.

A "move" is defined as the action of a player placing a token in a cell that is already under their control. When a cell reaches its critical capacity, it distributes its tokens to neighboring cells, potentially triggering further distributions in a chain reaction. This dynamic is captured in the state transition function, which takes a state and a move as input and produces a new state reflecting all resulting changes, including any cascading effects.

Terminal states are those in which one player has lost all their tokens on the board, signifying the end of the game. The Minimax algorithm explores possible sequences of moves by recursively simulating all legal actions for both players, alternating between maximizing and minimizing the evaluation score depending on whose turn it is. Alpha-beta pruning is integrated to eliminate branches of the search tree that cannot possibly influence the final decision, thereby improving computational efficiency.

A crucial component of this mapping is the evaluation function, which estimates the desirability of a given state when the search cannot proceed to a terminal state due to computational limits. In this work, the evaluation function is designed based on two main factors: the difference in the sum of cell values (orb) owned by the AI and its opponent, and the difference in the number of cells controlled by each player. The rationale behind this design is that having more tokens increases a player's potential to trigger chain reactions and exert influence on the board, while controlling more cells reduces the opponent's options and increases territorial advantage. By combining these two aspects, the evaluation function provides a balanced heuristic that guides the AI to both accumulate resources and expand its control, which are both critical for success in the game.

```java
public class CellEvalStrategy extends EvalStrategy {

    @Override
    public double evaluate(GameState state, Player player) {
        int myCells = state.countCells(player);
        int enemyCells = state.countCells(state.getOpponent(player));
        if (myCells == 0) return Double.NEGATIVE_INFINITY;
        if (enemyCells == 0) return Double.POSITIVE_INFINITY;
        return myCells - enemyCells * 5;
    }

}
```

Fig. 3.1 Evaluation Strategy by Counting Cells Code Snippet

```java
package com.iammadsfq.chainreaction.ai.strategy;

import com.iammadsfq.chainreaction.engine.GameState;
import com.iammadsfq.chainreaction.model.Player;

public class OrbEvalStrategy extends EvalStrategy {

    @Override
    public double evaluate(GameState state, Player player) {
        int myOrbs = state.countOrbs(player);
        int enemyOrbs = state.countOrbs(state.getOpponent(player));
        if (myOrbs == 0) return Double.NEGATIVE_INFINITY;
        if (enemyOrbs == 0) return Double.POSITIVE_INFINITY;
        return myOrbs - enemyOrbs * 5;
    }

}
```

Fig. 3.1 Evaluation Strategy by Counting Orbs Code Snippet

This mapping from the Chain Reaction game to the Minimax search framework, with a carefully designed evaluation function, allows the AI agent to simulate future possibilities, anticipate the opponent's responses, and select moves that maximize its chances of winning.

IV. Experiment and Analysis

To evaluate the effectiveness of the AI agent and the impact of different evaluation strategies, a series of experiments were conducted under controlled conditions. The experiments aimed to assess the performance of the Minimax-based agent using two heuristic variants—OrbEvalStrategy and CellEvalStrategy—across different types of opponents: human players, random-move agents, and head-to-head matches between the two AI configurations. All experiments were performed with a fixed maximum search depth of 5 plies, a value empirically selected to balance decision quality and computational feasibility.

The game was executed on a 5×5 board configuration, with uniform critical mass of 4 across all cells. Players were restricted to placing orbs only on cells they already owned, except during the first turn where an empty cell was permitted. These constraints closely followed the ruleset of JindoBlu's mobile version of the Chain Reaction game, and were enforced consistently during all tests.

The random agent used as a baseline opponent selects its moves uniformly at random from the list of legal actions available at each turn. No weighting or history-based bias was applied to its choices. For the human player experiments, a custom graphical user interface was used to allow real-time input from a human user, with game state updates and legal move constraints enforced by the engine. All human-versus-AI games were run interactively, and move times for the human player were not timed.

In each match, the following performance metrics were recorded: move count (total number of turns until game end), execution time for the AI agent per game (in nanoseconds), and final game outcome (winning player). To minimize bias due to turn advantage, experiments were distributed across both cases where the AI moves first and where it moves second. The agent's multithreaded execution was enabled in all tests, using Java's ExecutorService to parallelize top-level move evaluation.

Finally, to assess the comparative strength of the two heuristics, cross-strategy matches were performed between OrbEvalStrategy and CellEvalStrategy, alternating the first player in each game. This direct comparison enables analysis of each heuristic's strengths and weaknesses in dynamic adversarial settings, offering insight beyond random or human benchmarks.

TABLE I. Orb Counting Minimax Versus Human Player

Game	Measures				
	Orb Time (ns)	*Human Time (ns)*	*First Player*	*Move Count*	*Winner*
1	68582097	-	Orb	18	Orb
2	283712000	-	Orb	41	Orb
3	42858700	-	Orb	19	Human
4	97246501	-	Human	19	Orb
5	157979602	-	Human	27	Orb
6	48924398	-	Human	14	Orb

TABLE II. Orb Counting Minimax Versus Random Move

Game	Measures				
	Orb Time (ns)	*Random Time (ns)*	*First Player*	*Move Count*	*Winner*
1	109131000	340600	Orb	17	Orb
2	49832900	311600	Orb	10	Orb
3	29412900	267200	Orb	10	Orb
4	15187000	909000	Random	8	Orb
5	72671800	717600	Random	13	Orb
6	137820000	1197900	Random	16	Orb

TABLE III. Cell Counting Minimax Versus Human Player

Game	Measures				
	Orb Time (ns)	*Human Time (ns)*	*First Player*	*Move Count*	*Winner*
1	116105000	-	Cell	27	Cell
2	215258900	-	Cell	37	Cell
3	99218100	-	Cell	25	Cell
4	65110500	-	Human	18	Cell
5	59108500	-	Human	14	Cell

Game	Measures				
	Orb Time (ns)	***Human Time (ns)***	***First Player***	***Move Count***	***Winner***
6	105413500	-	Human	23	Cell

TABLE IV. CELL COUNTING MINIMAX VERSUS RANDOM MOVE

Game	Measures				
	Cell Time (ns)	***Random Time (ns)***	***First Player***	***Move Count***	***Winner***
1	102609100	571700	Cell	16	Cell
2	10514700	94900	Cell	4	Cell
3	267808500	478200	Cell	22	Cell
4	17112300	662000	Random	8	Cell
5	119889100	1353500	Random	16	Cell
6	55018700	1049100	Random	12	Cell

TABLE V. CELL COUNTING MINIMAX VERSUS ORB COUNTING MINIMAX

Game	Measures				
	Cell Time (ns)	***Orb Time (ns)***	***First Player***	***Move Count***	***Winner***
1	628518000	47857100	Orb	15	Cell
2	477355400	303234900	Cell	29	Orb

The results from the conducted experiments offer valuable insights into the performance of the AI agent designed using the Minimax algorithm with alpha-beta pruning, applied to the two-player Chain Reaction game. The primary objective of this work was to construct a functional and competitive AI capable of operating within the game's deterministic, chain-reaction-driven mechanics. The empirical results confirm that this objective has been achieved: the agent consistently selects valid and impactful moves, anticipates opponent responses through search depth up to five plies, and handles cascading state changes resulting from explosions and ownership shifts effectively. The success of the agent is evident across multiple test scenarios—against human players, random agents, and in direct AI-versus-AI matchups—demonstrating both the correctness and practical viability of the designed system.

The AI's decision-making is driven by two purely quantitative evaluation strategies: OrbEvalStrategy, which calculates the difference in total orb count between the player and their opponent, and CellEvalStrategy, which computes the difference in the number of controlled cells. Notably, CellEvalStrategy is the simpler of the two—it merely counts how many cells are owned by each player, while OrbEvalStrategy must aggregate orb quantities per cell, with each cell holding between zero and four orbs. Despite its simplicity, CellEvalStrategy often leads to more spatially diverse and stable play, as each owned cell represents a legal move and potential leverage point in future turns.

As shown in Tables I and II, OrbEvalStrategy performs reliably against random agents and secures multiple wins against human players, demonstrating its capacity to efficiently accumulate material advantage. However, this approach may cluster orbs in fewer positions, reducing tactical flexibility and increasing vulnerability to chain reactions—evident in cases like Game 4 of Table III, where OrbEvalStrategy loses despite initially leading in orb count.

In contrast, Tables III and IV demonstrate how CellEvalStrategy, though less sophisticated in metric, emphasizes distributed control and broader territorial access. A player owning four distinct cells, even with minimal orb presence, maintains more actionable options per turn than one who holds more orbs spread across fewer cells. This makes CellEvalStrategy naturally robust against board congestion and strategic confinement, especially in reactive environments where flexibility and reach matter more than raw material count.

One critical aspect affecting performance across all experiments is the advantage of the second player, due to the game's rule that only the first move may be played on an empty cell. This gives the second player the opportunity to respond adjacent to the first move and cause an early explosion, often leading to a 4-to-2 cell advantage after the initial turns. This asymmetry consistently benefits the player who moves second, often outweighing the influence of the heuristic itself.

This effect is clearly visible in Table V, which compares both strategies directly. In Game 1, CellEvalStrategy plays second and secures a quick 15-move victory. In Game 2, the roles are reversed—OrbEvalStrategy moves second and wins, although only after a longer and more complex 29-move struggle. These outcomes suggest that while CellEvalStrategy generally leads to stronger, more flexible play, the turn order remains the most decisive factor in early-game outcomes. Nevertheless, the fact that a simpler metric like CellEvalStrategy consistently enables efficient, winning behavior—even against a more detailed orb-based evaluation—highlights the importance of heuristic alignment over computational complexity.

Another significant takeaway is the agent's overall strength despite using a relatively shallow maximum search depth of five. In Chain Reaction, where each move can produce cascades of cell takeovers, the impact of a decision is often visible within a few plies. This makes long-term lookahead less critical than in classical games like chess. The AI benefits from the game's deterministic explosion mechanics, allowing effective tactical forecasting even with short-range planning. Combined with alpha-beta pruning and multithreaded evaluation, this enables fast and effective gameplay, confirming that a modest search depth coupled with a task-appropriate heuristic is sufficient to achieve competent, real-time AI performance in Chain Reaction.

In summary, the Minimax-based AI agent demonstrates reliable and strategically sound behavior across a variety of adversaries and scenarios. Although OrbEvalStrategy offers stronger material accumulation, CellEvalStrategy provides more balanced and consistent play despite being the simpler heuristic. The project demonstrates that with the right structural design, even classical search methods and lightweight evaluation functions can produce intelligent behavior in highly dynamic, non-trivial game environments. Furthermore, the results reinforce the influence of turn order in Chain Reaction, showing that game-specific asymmetries must be carefully considered when assessing AI effectiveness.

V. Conclusion

This paper presented the design and implementation of a Minimax-based AI agent for the two-player version of the Chain Reaction game. The agent was built upon a modular architecture, with a flexible evaluation system, alpha-beta pruning, and efficient simulation of game states through cloning. The AI was successfully adapted to the game's unique dynamics—such as cascading explosions, strict move constraints, and a compact 5×5 board—demonstrating that classical adversarial search techniques remain highly applicable in modern, reactive game environments.

Despite being constrained to a relatively shallow maximum search depth of five plies, the agent was able to perform effectively in a variety of competitive scenarios. This reflects the nature of Chain Reaction, where local moves can produce global consequences through chain reactions, reducing the need for deep foresight. The results affirm that even a limited-depth tree search, when combined with a domain-aligned evaluation function and pruning optimizations, can produce competent and strategic gameplay in real time.

Two purely quantitative evaluation strategies were explored and compared: OrbEvalStrategy, which computes the total number of orbs owned, and CellEvalStrategy, which simply counts the number of cells controlled. Although CellEvalStrategy is computationally simpler, it consistently led to more flexible, resilient play—especially in longer games—by promoting greater spatial distribution and preserving move options. Experiments revealed that while OrbEvalStrategy could still secure wins, especially when playing second, it typically required more moves and careful timing to overcome its limitations. The turn order advantage, favoring the second player due to the game's opening rules, was found to have a significant impact on outcomes, often outweighing the choice of heuristic.

Overall, this project demonstrates that traditional AI techniques such as Minimax search remain powerful when tailored to the structural and dynamic properties of a specific game. It highlights how even simple evaluation strategies, when thoughtfully aligned with game mechanics, can yield diverse and competitive AI behavior. Future work could explore hybrid heuristics, dynamic depth adjustments, or learning-based methods to further enhance strategic depth and adaptability. Ultimately, the success of this agent reaffirms the relevance of tree search–based AI design in dynamic, turn-based games with deterministic transitions.

VI. Appendix

The source code for this project is available at github: https://github.com/iammadsfq/Chain-Reaction-Game-13523135

Video Link at Youtube

https://www.youtube.com/watch?v=XUGgPqDXsuU

Acknowledgment

Author expresses gratitude to all parties who have assisted in the preparation of this paper, especially to:

1. The Almighty God, for His grace and guidance, allowing the smooth completion of this paper.
2. Both parents, for providing both moral and material support to the author.
3. Extended family and friends who have encouraged and aided in the completion of this paper.
4. Monterico Adrian, S.T., M.T. as the lecturer for the IF2211 Algorithm Strategy course and Dr. Ir. Rinaldi Munir, M.T. for kindly imparting additional knowledge and offering solutions to challenges encountered in writing this paper.
5. Colleagues from the School of Electrical Engineering, especially those in the IF K3 class, for their support and collaborative spirit that have inspired and motivated the author throughout his journey

The author deeply appreciates all the assistance, encouragement, and kindness received from these individuals and groups, without which the completion of this paper would not have been possible

References

[1] M. H. Winands, Y. Björnsson, and J. W. H. M. Uiterwijk, "Alpha-Beta Pruning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 13, pp. 11675–11682, Dec. 2020. [Online]. Available: https://cdn.aaai.org/ojs/8148/8148-13-11675-1-2-20201228.pdf

[2] D. Roth, "Lecture 32: Game Playing II," *CS 440: Artificial Intelligence*, University of Illinois at Urbana-Champaign, Fall 2018. [Online]. Available: https://courses.grainger.illinois.edu/cs440/fa2018/lectures/lect32.html

[3] C. Fry, C. Li, and M. Zheng, "Using Reinforcement Learning to Play Othello," *CS229 Final Project Report*, Stanford University, 2016. [Online]. Available: https://cs229.stanford.edu/proj2016/report/FryLiZheng-UsingReinforcementLearningToPlayOthello.pdf

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Ahmad Syafiq
13523135