Project 2: Tic-Tac-Toe Game

Ajith Kumar Panja Umasankar, Brendan Tith, Pranaya Bommana, Jahnavi Sri Sai Lavu, Gauri Milind Kadbane, Karthikeyan Sivasubramanian

***Abstract*—This project involves designing and developing a Tic-Tac-Toe game for Android, allowing players to compete against an AI using the Minimax algorithm with alpha-beta pruning. The goal is to implement an efficient game that optimally uses mobile resources. The AI opponent is designed to have different levels of difficulty, i.e. Easy, Medium, and Hard, to cater to a wide range of player skills. Additionally, the game records previous results, saving the winner and difficulty mode for each match, and providing users with a history of their performance. This project solidifies the understanding of AI algorithms in resource-constrained mobile environments.**

***Index Terms*—AI, Android, Minimax algorithm, Modular UI design**

I. INTRODUCTION

Tic-Tac-Toe is a classic game that provides an ideal foundation for exploring artificial intelligence and algorithmic efficiency in mobile application development. In this project, the primary goal is to create a mobile-based Tic-Tac-Toe game where a human can play against an AI that uses the Minimax algorithm with alpha-beta pruning. The Minimax algorithm is a decision-making process often used in game theory and AI to create an optimal opponent in two-player games. Alpha-beta pruning improves Minimax by cutting down the number of nodes evaluated, making it suitable for mobile devices with limited resources.

This game introduces three difficulty modes, where the AI adapts its level of "intelligence" by modifying its decision-making depth, allowing for a user experience suitable for different player skill levels. Furthermore, each game's results are stored locally on the device, tracking the winner and difficulty mode, providing players with insights into their progress.

Project Links

- Project Video: YouTube

II. TECHNICAL APPROACH

Minimax is an algorithm that focuses on decision-making, especially for two-player games like Tic-Tac-Toe. In Tic-Tac-Toe, there are a set of possible moves that guarantees a draw given optimal play from both sides, and a win if nonoptimal. This is relevant when creating a computer for a person to play against, as minimax identifies which moves are optimal. The goal of the computer, given it is going second, has the goal of trying to minimize the player's maximum possible gain. This is done through evaluating all the possible moves to be made from the current state to the end of the game. Moves are evaluated by trying to maximize from the human player's perspective after a possible minimizing move is played. For Tic-Tac-Toe, the value of a move is determined by how many moves it would take to either reach a win, loss, or a draw. For the case of the computer, it looks to have the highest value possible as these represent the fastest wins for the computer. The values of a win and loss for the computer are positive 10 minus depth and negative 10 plus depth respectively as each move adds 1 to the depth. However, given optimal play, the highest value is likely to be 0 due to drawing. A draw is given a value of 0 as this is the game state where neither player has won, thus the value cannot be positive as it means that the computer has failed to win but is not negative as the human has also failed. The algorithm for the computer is implemented through recursion, where it takes the current board, and attempts to find the largest value given all possible human player moves, recursively calling itself flipping between maximizer and minimizer for the human and computer, respectively. This creates several recursive calls, as each possible move and board state are analyzed.

The biggest problem of minimax is how it evaluates all moves possible, as when assuming optimal play, there is not a need to evaluate every single possible move. This is where alpha-beta pruning is applied, where alpha is the maximum value that can be guaranteed by the maximizer at this point in calculation, and beta is the lowest value that can be guaranteed by the minimizer at this point in the calculation. Thus, when maximizing, it is known that if it finds a value greater than beta for a move, then it stops because the minimizer will not allow for that option given optimal play. Similarly, when minimizing, if the value found is lower than alpha, it stops due to moves requiring the maximizer to have nonoptimal play. This means that with alpha-beta pruning, only optimal moves are considered. If a nonoptimal move is played by the human, then it simply calculates a new value which indicates a win given all moves were made using the minimax algorithm. With alpha-beta pruning, it has cut off possible branches of the future, shortening the amount of time it takes to calculate the optimal move in a minimax algorithm. This reduction in calculation time is incredibly important for hardware that has a challenging time supporting complex calculations like mobile devices.

III. DESIGN CHOICES

The UI design for the Tic-Tac-Toe game focuses on simplicity, usability, and modularity, with distinct screens and navigation. A `Screen` sealed class defines the main routes (Game, Settings, PastGames) to provide clear, type-safe navigation through the `NavController` within `MainApp()`, creating easy transitions between game functionalities.

A. Game Screen

It displays the current player or winner prominently, updating in real-time as players take turns. The game board is structured as a 3x3 grid, with cell colors dynamically changing based on the game state and winning positions and draws are highlighted. The screen also offers a "Restart Game" button and easy access to settings, past games, and data backup/restore options, enhancing the game flow.

B. Settings Screen

Here, users can choose a difficulty level via radio buttons (Easy, Medium, Hard), which directly influences the AI's behavior in the game. The interface also includes a "Back to Game" button, allowing users to resume gameplay seamlessly after adjusting settings.

C. Past Games Screen

It showcases previous games in a clean table format with Date, Winner, and Difficulty columns, allowing players to review game history easily. A LazyColumn is used for scrollable game records, making it space-efficient for viewing multiple records.

D. Backup and Restore

This functionality is accessible from the game screen, enabling players to back up and restore game data and settings. Dialog boxes provide feedback on completion, giving players reassurance on data handling and persistence.

E. Game Logic Handling

Finally, Game Logic Handling is managed within *GameScreen*, where core functionalities such as checking for a winner, resetting the game, and storing game records are implemented. The AI difficulty level, chosen in Settings, directly impacts gameplay, adding flexibility and replay value. For styling, the UI uses a consistent teal and white color scheme, which is applied across screens to create a cohesive look. Both the top and bottom app bars are styled with titles, author credits and navigation buttons enhancing the appearance of the application.

IV. IMPLICATIONS AND LIMITATIONS

This Tic-Tac-Toe game project demonstrates the use of AI algorithms, specifically Minimax with alpha-beta pruning, in enhancing gameplay experience on mobile platforms. While the project achieves its primary goal of offering a challenging and adaptable AI opponent, it has several limitations and challenges that provide insight into both technical constraints and areas for future development.

One significant limitation of the game is the absence of a redo/undo feature, which prevents players from experimenting with different moves or recovering from mistakes. This lack of flexibility may reduce replay ability and restrict strategic exploration. Additionally, the game only supports a single-player experience where the user plays against the AI, with no option for human versus human mode. Consequently, the app may not appeal to users interested in a multiplayer experience, potentially limiting its audience.

The team also encountered challenges in keeping the game's state consistent, especially since mobile apps often get interrupted or closed. Ensuring that the game could save and reload the exact board position, turn order, and difficulty level after reopening required careful handling of local storage, as mobile devices limit background activity. Another challenge was enabling a mid-game difficulty change. Adjusting the AI's difficulty dynamically required resetting the Minimax algorithm's depth and strategy, which posed technical obstacles in ensuring the smoothness of gameplay without recalculating previous moves.

Overall, these limitations underscore the technical complexities involved in developing an AI-driven mobile game, particularly with respect to maintaining seamless user experience across various scenarios. Future iterations of the app could address these limitations, potentially adding value and broadening the game's appeal.

REFERENCES

[1] B. Swaminathan, V. R. Vaishali, and T. S. R. Subashri, “Analysis of Minimax Algorithm Using Tic-Tac-Toe,” in *Intelligent Systems and Computer Technology*, 2020, doi: 10.3233/APC200197.

[2] M. Chakole, R. Nawathe, S. Dorle, G. Bhakre, and N. Nimbarte, “Optimal Strategy Formulation for Tic-Tac-Toe Using Minimax Algorithm for Interactive Gaming,” *Communications on Applied Nonlinear Analysis*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270505136.

[3] J. S. Jayabharathi and V. Ilango, “A Brief Revolution of Evolution and Resurgence on Machine Learning,” in *2021 Asian Conference on Innovation in Technology (ASIANCON)*, 2021, doi: 10.1109/ASIANCON51346.2021.9544706.