

CSE535 Project 2 – TicTacToe Game (Group 22)

Anusha Alangar, Gokul Vasudeva, Greha Shah, Manan Shah, Natasha Koli, Shashank Venkataramana

Abstract—The integration of artificial intelligence (AI) in mobile gaming presents unique challenges and opportunities for developers. This paper presents the design and implementation of a Tic-Tac-Toe game for Android platforms, featuring an AI opponent that utilizes the Minimax algorithm with alpha-beta pruning. The project addresses the complexities of implementing advanced algorithms within the constraints of mobile computing while providing an engaging user experience. Three distinct AI difficulty levels (Easy, Medium, Hard) are implemented to cater to various skill levels. Additional features include game history tracking and local as well as Bluetooth-enabled multiplayer modes, expanding the application's functionality beyond single-player gameplay. The findings of this study contribute to the growing body of knowledge on efficient AI integration in mobile applications and networked gameplay optimization.

I. INTRODUCTION

The rapid advancement of mobile technology has created new opportunities for the implementation of artificial intelligence in gaming applications. This project focuses on the development of a comprehensive Tic-Tac-Toe application for Android devices, serving as a case study for the integration of AI in mobile gaming. By combining classic gameplay with modern AI techniques, this project aims to explore the balance between computational efficiency and engaging user experience in mobile environments.

The primary objectives of this project include:

- Implementing an intuitive and responsive game interface optimized for mobile devices.
- Developing an AI opponent using the Minimax algorithm with alpha-beta pruning, capable of operating at three distinct difficulty levels.
- Incorporating game history tracking to enhance user engagement and provide insights into gameplay patterns.
- Implementation of same device as well as Bluetooth-enabled multiplayer functionality to extend the game's social aspects.

These objectives collectively address key challenges in mobile game development, including resource management, algorithm optimization, and networked gameplay. By tackling these challenges within the context of a familiar game, this project provides a practical framework for understanding the complexities of AI integration in mobile applications.

The significance of this work lies in its potential to advance understanding of how sophisticated AI algorithms can be effectively implemented in mobile games while maintaining performance and user engagement. This project contributes to the growing field of mobile AI gaming, offering insights into the development of intelligent, responsive, and socially interactive mobile applications.

II. TECHNICAL APPROACH

In this project, we implement a Tic-Tac-Toe AI using the Minimax algorithm with alpha-beta pruning. This approach allows the AI to make optimal moves while considering all possible future game states. Below is a detailed breakdown of the Minimax algorithm, alpha-beta pruning optimization, and how difficulty levels are integrated into the game.

A. Minimax Algorithm Overview

The Minimax algorithm is a recursive decision-making algorithm commonly used in two-player games like Tic-Tac-Toe. It works by simulating all possible moves for both players, assuming both are playing optimally. In the context of Tic-Tac-Toe, the AI is player 'O' (maximizing player), and the human is player 'X' (minimizing player). The algorithm's goal is to find the best possible move by maximizing the chances of winning for player 'O' while minimizing the chances for player 'X'.

Here's how the Minimax algorithm is structured for this project:

- **State Representation:** The Tic-Tac-Toe board is represented as a 3x3 matrix, where each cell contains 'X', 'O', or a space ' ' representing an empty spot. State refers to a specific configuration of the game board at a particular point in time, representing the current situation in the game.
- **Recursive Evaluation:** The algorithm recursively explores all possible game states from the current board configuration. At each step, the algorithm alternates between two players (maximizing and minimizing). It simulates the result of placing a mark (either 'X' or 'O') in each available spot and then evaluates the board's state to determine if the game is won, lost, or drawn.
- **Scoring Mechanism:** When a terminal state is reached (i.e., a win, loss, or draw), the algorithm assigns a score to that state:
 - AI Win (Player O): +10 points.
 - Human Win (Player X): -10 points.

- These scores propagate back up the recursive call stack, with the algorithm choosing the maximum score for player 'O' (maximizing) and the minimum score for player 'X' (minimizing). This ensures that the AI tries to maximize its chances of winning.

To enhance the efficiency of the Minimax algorithm, alpha-beta pruning is used.[1] Without pruning, the Minimax algorithm would evaluate every possible sequence of moves, which can be computationally expensive, especially in games with larger search spaces. Alpha-beta pruning reduces the number of nodes that need to be evaluated by eliminating moves that won't affect the final decision.

- Alpha: The best value the maximizing player (AI) can guarantee at that point in the tree.
- Beta: The best value the minimizing player (human) can guarantee at that point in the tree.

Pruning Example: If we are currently at a point where it's the minimizing player's (human's) turn, and we are evaluating two branches to decide the next move, let's say the left branch already provides a value of 5. When we start examining the right branch and encounter a value of 6, we can stop exploring further because the minimizing player is trying to find the lowest value. Since the value of 6 from the right branch is already higher than 5 (the value from the left branch), it will never be chosen by the minimizing player. Thus, further exploration of the right branch is unnecessary, as the player will prefer the smaller value of 5.

This optimization significantly reduces the computation time and ensures that the game remains responsive, even in "Hard" mode where the AI plays optimally.

To make the game more engaging, the AI operates at three different difficulty levels: Easy, Medium, and Hard. The implementation of these difficulty levels adjusts how the AI behaves, particularly in terms of how often it plays optimally.

Medium Mode: In Medium mode, the AI uses a mix of random moves and optimal moves. For example, 50% of the time, the AI will make random moves, and the remaining 50% of the time, it will use the Minimax algorithm to find the best possible move. This creates a balance between challenge and winnability.

The difficulty setting is managed through a simple conditional check in the AI's move selection function. Depending on the mode selected by the user, the game dynamically adjusts how the AI selects its moves.

The Minimax and alpha-beta pruning logic is integrated into the game's main loop in the following way:

- The Minimax function is called recursively until a terminal game state (win, loss, or draw) is reached, and the best possible move for the AI is returned. This move is then applied to the board.

In addition to the AI-based gameplay, human-vs-human gameplay was implemented using Bluetooth connectivity between two devices, allowing players to compete against each other in real time.

Bluetooth Setup and Connection: The app requests permissions for Bluetooth connect and scanning, as well as requests additional location permissions required for Bluetooth discovery. The user has the choice to be the lobby creator, which initiates Bluetooth discoverability and creates a server socket that waits for a connection, or join an existing lobby, which initiates Bluetooth discover which attempts to connect to a discovered device. The socket creation process requires a unique service UUID which has been generated for the app. Once the connections are successful, Bluetooth sockets are created.

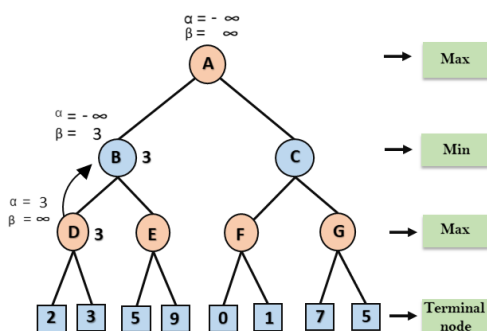


Fig. 1. Alpha-Beta Pruning in Minimax

Bluetooth sockets are used to handle the communication between devices. Both devices start a listening thread which infinitely reads from the socket, and converts read string messages to JSON and passes to the GameViewModel for processing.

Game State Transmission Using JSON: As and when one device needs to send a message, it will serialize a JSON to string and transmit it over the socket. Initially after communication, the devices transmit messages to identify their MAC addresses.

To maintain compatibility across platforms, the game state data is transmitted in JSON format. JSON was chosen due to its lightweight nature and ease of parsing, ensuring efficient real-time data exchange. The GameViewModel is responsible for constructing the JSONs based on the current state of the game on that user's device. The JSONs are constructed as defined in the project 2 guidelines, and so will not be repeated in this report.

The game data, including current board state, player turns, and move history, is serialized into a JSON object and transmitted to the other device whenever a user makes a move on their device and it was their turn.

Move Synchronization: On receiving this JSON, the game state is updated and synchronized between both devices. The transmitted data includes the player's move, current board configuration, and whose turn it is, as well as other control parameters.

Each device continuously listens for updates from the other, ensuring that both devices stay in sync throughout the game, such as reset requests, etc.

Player Turn Negotiation: A pre-determined choice by one of the players is used to decide which player goes first. This negotiation happens once the Bluetooth connection is established and a user selects ME or OPPONENT, and the result is shared between the devices.

The app has been implemented in such a way that disconnections are handled gracefully, with sockets being closed and terminated. This Bluetooth implementation enhances the game's engagement by allowing real-time multiplayer matches, further improving the interactive experience for users.

III. DESIGN CHOICES

The design of the Tic-Tac-Toe game app focuses on simplicity, usability, and performance optimization. Key elements include the user interface (UI), data persistence, and the overall gameplay experience.

A. User Interface (UI) Design

The UI was built using Jetpack Compose, a modern Android UI framework, ensuring a seamless and responsive user experience. The main game screen presents a clean 3x3 grid, where players can easily interact with the board. Player X and AI's O are displayed with distinct colors and symbols for easy recognition. The layout is minimalistic to focus the user's attention on the game itself.

A settings screen allows users to change the difficulty level (Easy, Medium, Hard) during gameplay without restarting the game.

A past games screen presents a list of previous games in a tabular format, including the winner, date, and difficulty level,

making it easy for users to track their progress.

A GameViewModel class is defined to maintain, provide and update the state of the game. This class defines mutable state variables, such as the board and control flags. The Composable Jetpack components read the state from the ViewModel, and request changes to the ViewModel on user interaction.

B. Data Persistence with Room

To store past game results and user settings, the Room database is used. Room offers an abstraction layer over SQLite, making data management more efficient and less error-prone. This choice ensures that game data, such as the outcome of previous games are stored locally and persist across sessions.

Room Database: The game's history and settings are stored using entities and DAOs (Data Access Objects), ensuring the app remains lightweight and efficient. Room provides compile-time verification of SQL queries and handles database migrations, which makes it more robust compared to raw SQLite.

C. Gameplay and Flow Optimization

The game's flow ensures smooth and responsive interaction between the human player and the AI. The Minimax algorithm with alpha-beta pruning is optimized for performance, allowing the AI to make decisions efficiently, even in "Hard" mode where it always plays optimally. Transitions between player turns are handled in real-time, with no visible delays, ensuring an engaging gameplay experience. The Minimax algorithm is invoked on a separate thread from the UI, and invokes a callback when done, ensuring the UI thread is not blocked.

Real-Time Difficulty Changes: The ability to change difficulty settings mid-game was designed to add flexibility to the gameplay. Players can switch between difficulty modes without resetting the board, making the game more dynamic.

These design choices—focusing on a modern UI, reliable data persistence using Room, and optimized game logic—contribute to a robust, user-friendly experience.

D. Bluetooth Human-vs-Human components & UI interactions (Extra-Credit)

For additional gamemodes, there is a gamemode switcher dialog Composable that the user can click to change gamemodes. For local multiplayer, the AI is replaced with another user turn, but that user is considered as Player 2.

When the user selects the Player vs Bluetooth Player gamemode, Bluetooth is enabled via an intent if not already. There are two buttons that allow a user to create a lobby, or join a lobby. When joining a lobby. Another dialog Composable is shown with a list of discovered devices, which are discovered and maintained in a mutable list, which is cleared every time the dialog is opened. Only named Bluetooth devices are listed, which the user can select to initiate a connection with that particular device.

The user is only permitted to make a move when their local game state thinks it is their turn to make the move. Once made, the move is transmitted to the other device. The other device, which is constantly listening for messages, receives the JSON, and GameViewModel will process the JSON to update the local state, allowing that device to now make the next move.

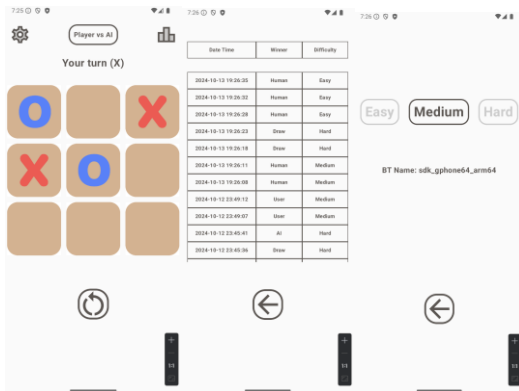


Fig. 2. User Interface of TicTacToe Game

IV. IMPLICATIONS AND LIMITATIONS

A. Implications

The implementation of the Minimax algorithm with alpha-beta pruning in the Tic-Tac-Toe game has several important implications for both gameplay and the technical complexity of the project:

- **Efficient AI Behavior:** The use of alpha-beta pruning ensures that the AI can make optimal moves without having to explore every possible game state. This makes the AI in "Hard" mode unbeatable, providing a challenging experience for users who want to test their skills.
- **Data Integrity with Room:** By using the Room database for storing past game results, the app ensures that the user's progress is maintained between sessions. This data persistence contributes to a better user experience, allowing players to revisit their previous games and adjust their strategies based on past results.
- **Scalable Design:** The use of Jetpack Compose for the UI design makes the app scalable, enabling future updates or expansions of the game (such as adding new game modes or features) to be implemented without significant changes to the core codebase.
- **Bluetooth gameplay (Extra-Credit):** Since the Bluetooth communication is via sockets and plain JSONs, we can potentially have our game perform multiplayer Bluetooth gameplay with a game developed by another team, as long as we attain their UUID, and they share connection establishment procedures if there are any.

B. Limitations

Despite these benefits, there are some limitations to the current design:

- **Performance on Older Devices:** While alpha-beta pruning optimizes the AI's decision-making process, the performance on older or low-powered devices may still experience lag, especially when running the AI in "Hard" mode. This is because the Minimax algorithm still explores a significant number of game states, and the

computational demands might cause slight delays on resource-constrained devices.

- **Limited Game Complexity:** Tic-Tac-Toe is a relatively simple game with a limited number of possible board states. While the use of Minimax is educational, the algorithm may not be as beneficial for more complex games where deeper search trees make the computational load significantly higher. This limits the educational value of the algorithm in games beyond Tic-Tac-Toe.
- **Fixed Game Flow:** The game currently follows a fixed flow where the human player always starts. This could lead to predictable gameplay patterns, which may become monotonous over time. Introducing more variety, such as allowing the AI to start, could enhance the game experience and keep it fresh for repeated plays.

Bluetooth Human-vs-Human Limitations (Extra-Credit):

- We found that for Bluetooth discovery, we were required to also provide location permissions. This could drive some users away as it seems like an invasive permission to provide.
- For security reasons, an app cannot get its own MAC address via the Bluetooth Adapter. This is a limitation of the Android platform, and requires us to send additional messages after the socket is created in order for each device to learn of its own MAC address^[4].
- Graceful disconnection was found to be buggy in some instances, this is a consequence of many moving parts and possible race conditions. This highlights the difficulty in maintaining synchronization between devices in multiplayer games.

IV. LINKS

Github Repository:

https://github.com/GokulVSD/CSE535_P2_TicTacToe_Group_2

Main Demo Video: <https://youtu.be/7Z2-RhYoIQc>

Extra-Credit Demo Video: <https://youtu.be/O1AZ4IyYz-U>

REFERENCES

- [1] D. Knuth and R. Moore, "An analysis of alpha-beta pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, Dec. 1975, doi: 10.1016/0004-3702(75)90019-3.
- [2] H. van den Herik, J. Uiterwijk, and J. van Rijswijk, "Games solved: Now and in the future," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 277–311, Jan. 2002, doi: 10.1016/S0004-3702(01)00152-7.
- [3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson, 2009, ISBN: 978-0-13-604259-4.
- [4] <https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id>

Project Group 22


Team Effectiveness Report

Name: Anusha Alangar

ASU ID: 1229511168

ASU email: aalanga1@asu.edu

Contribution: 1/6


Signature: 

Name: Gokul Vasudeva

ASU ID: 1229503862

ASU email: gvasude2@asu.edu

Contribution: 1/6


Signature: 

Name: Greha Chirag Shah

ASU ID: 1233576736

ASU email: gshah11@asu.edu

Contribution: 1/6


Signature: 

Name: Manan Sanjay Shah

ASU ID: 1233660573

ASU email: mshah131@asu.edu

Contribution: 1/6


Signature: 

Name: Natasha Moses Koli

ASU ID: 1229700552

ASU email: nkoli1@asu.edu

Contribution: 1/6

Signature: 

Name: Shashank Venkataramana

ASU ID: 1229385185

ASU email: svenk163@asu.edu

Contribution: 1/6

Signature: 