Tic Tac Toe

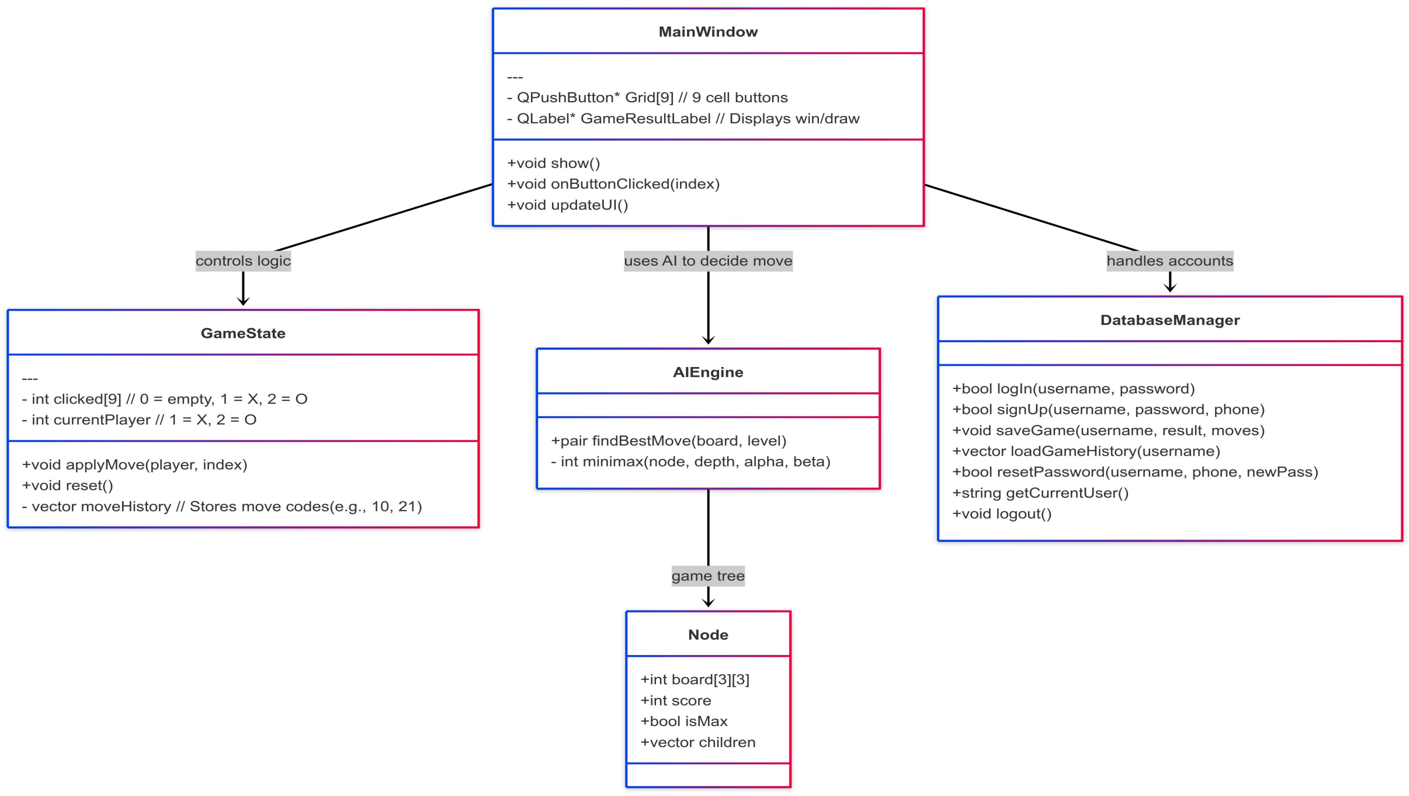
SDS report

Presented to: Prof. Omar Nasr

EECE27
ELECTRONICS & COMMUNICATIONS ENGINEERING

Ameer Ashraf Louly Abbass	9230242
Amir Sameh Fanous Attia	9230243
Mohamed abdullah farouk abdullah	9230787
Rola refaat bekhit	9230382
Nourhan Mohammed Fahmy	9230966

Software Design Specification (SDS)


MainWindow

- QPushButton* Grid[9] // 9 cell buttons
- QLabel* GameResultLabel // Displays win/draw
+void show()
+void onButtonClicked(index)
+void updateUI()
controls logic
uses AI to decide move
handles accounts
GameState

- int clicked[9] // 0 = empty, 1 = X, 2 = O
- int currentPlayer // 1 = X, 2 = O
+void applyMove(player, index)
+void reset()
- vector moveHistory // Stores move codes(e.g., 10, 21)
AIEngine
+pair findBestMove(board, level)
- int minimax(node, depth, alpha, beta)
DatabaseManager
+bool logIn(username, password)
+bool signUp(username, password, phone)
+void saveGame(username, result, moves)
+vector loadGameHistory(username)
+bool resetPassword(username, phone, newPass)
+string getCurrentUser()
+void logout()
game tree
Node
+int board[3][3]
+int score
+bool isMax
+vector children

Figure 1: full system main points

The full system diagram represents the modular design of the Tic Tac Toe application and3 how its key components interact.

At the center is the Main Window class, which serves as the **interface between the user and the game logic**. It connects directly to:

- **Game State**: Tracks the current state of the board, which player's turn it is, and a history of moves for replay. This component is responsible for updating internal logic, while Main Window reflects those updates on the screen.
- **AI Engine**: When in AI mode, Main Window passes control to AI Engine to compute the best move using the Minimax algorithm.
- **Database Manager**: Handles user login, registration, and stores/retrieves game history. It abstracts all SQLite operations so that Main Window doesn't need to manage database logic directly.

The **AI Engine** contains the logic for computing the best possible move. It constructs a **game tree** made of Node objects, each representing a possible board state. The AI recursively evaluates these nodes using minimax (), enhanced with alpha-beta pruning for speed.

The **Game State** class manages internal logic like marking cells as clicked, tracking move order, and resetting the board for new games or replays.

The **Database Manager** ensures persistent storage. It supports storing game results, full move histories, and user data such as usernames, passwords (hashed), and phone numbers. It also allows for login verification and password reset functionality.

This architecture ensures a **clear separation of concerns**:

- UI is handled by Main Window,
- Game logic by Game State,
- AI decision-making by AI Engine, and
- Data storage by Database Manager.

Each module communicates with others through clean interfaces, making the system maintainable, extendable, and efficient.

1. Purpose

This SDS outlines the design and architecture of the Tic Tac Toe game software. The goal is to define components, responsibilities, and interactions using diagrams and clear structural documentation to support maintainability, scalability, and correctness.

2. Scope

• A Qt-based GUI Tic Tac Toe game.
• Supports Player vs Player, Player vs AI (Minimax + Alpha-Beta), and Player vs AI (easy/medium).
• Tracks user login, game history, move history, and allows replay.
• Stores user data and match records in SQLite.

3. Software Architecture Overview

Main Components:

Module	Description
MainWindow	Manages game board UI and connects to logic.
AIEngine	Implements Minimax with Alpha-Beta pruning.
DatabaseManager	Handles user accounts, sessions, and history.
GameState	Manages move history, player turns, and resets.

4. Class Diagram

The system is organized into classes: MainWindow, AIEngine, Node, DatabaseManager, and GameState. Their relationships and roles are described above. Diagram tools like StarUML or draw.io can visualize these.

5. Sequence Diagram: Player vs AI Move

1. Player clicks a cell.
2. MainWindow applies the move and updates the board.
3. MainWindow calls AIEngine to find the best move.
4. AIEngine uses minimax with alpha-beta pruning.
5. AI move is applied and UI is updated.

6. Database Schema

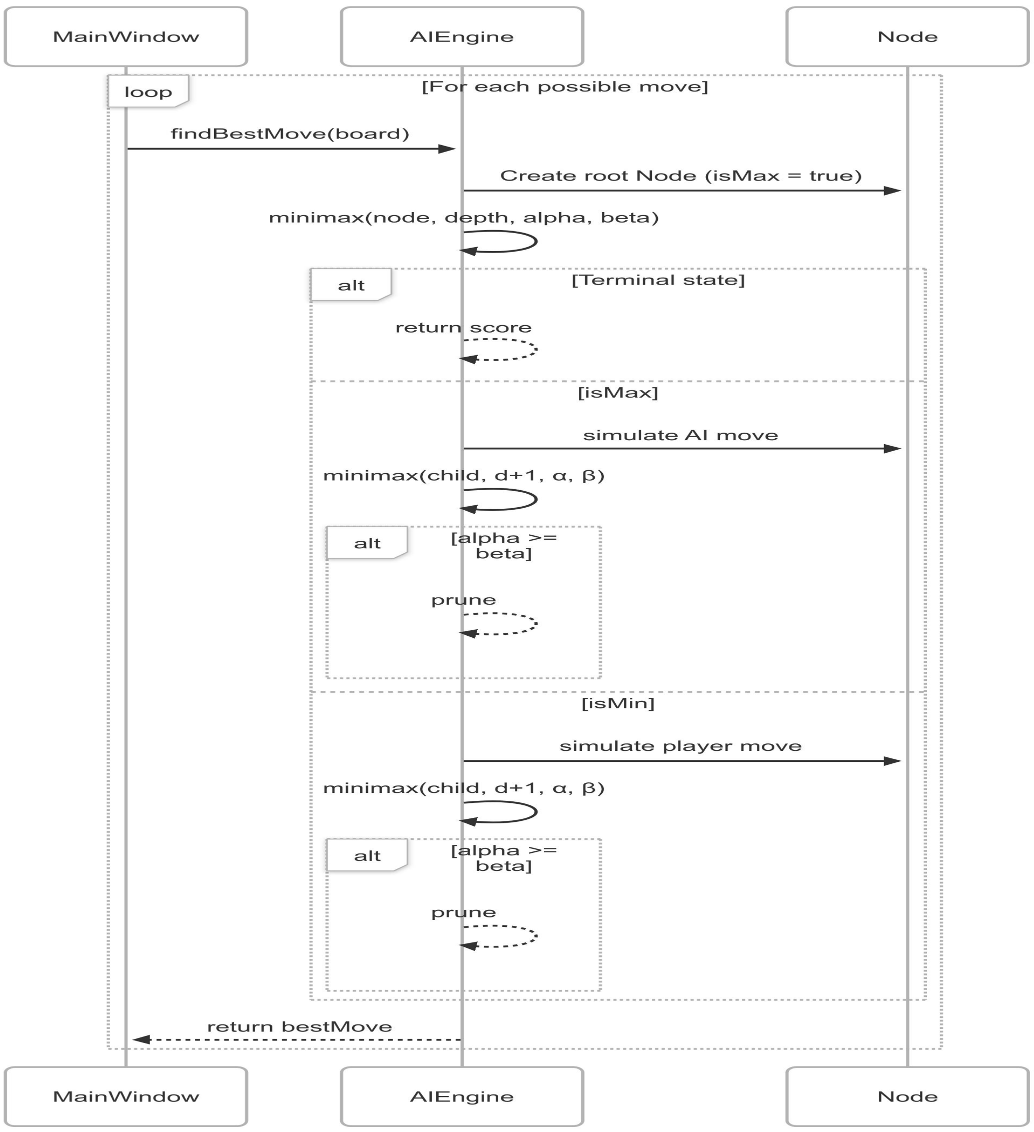
Users Table:

Field	Type	Notes
id	INTEGER	Primary Key
username	TEXT	Unique
password	TEXT	SHA-256 hash
phone	TEXT	Unique

Game history Table:

Field	Type	Notes
id	INTEGER	Primary Key
username	TEXT	Foreign Key
result	TEXT	"WIN", "LOSS", or "DRAW"
moves	TEXT	Comma-separated move codes
timestamp	DATETIME	Default: current timestamp

7. AI Design (Minimax with Alpha-Beta)



The AI constructs a game tree using the Node structure.
The minimax function evaluates all possible board outcomes.
Alpha-beta pruning skips branches that cannot influence the final decision.
The best move is chosen based on the maximum possible value for AI.

Figure 2 AI (Minimax with Alpha-Beta) Sequence Diagram

The AI component in this project uses a classic decision-making algorithm known as **Minimax**, enhanced with **alpha-beta pruning** to improve performance. The goal of the algorithm is to simulate all possible future moves in the game and determine the best next move for the AI opponent.

The diagram shows how this logic flows as a **sequence of recursive steps**:

- The Main Window initiates the process by calling Finestone() from the Engine after the human player makes a move.
- Inside findBestMove(), the game board is wrapped into a Node object representing the current state.
- The AIEngine begins evaluating each possible move by calling the minimax () function, which traverses down the game tree recursively.
- Each level of the tree alternates between the **AI (maximizing score)** and the **Player (minimizing score)**.
- As the recursion explores deeper, the algorithm tracks two values:
  - alpha: the best value the maximizer (AI) is currently guaranteed
  - beta: the best value the minimizer (Player) is currently guaranteed
- Whenever alpha >= beta, the algorithm **prunes** the remaining child nodes from consideration, avoiding unnecessary computation.

This pruning mechanism significantly improves performance, especially when the game tree becomes deeper. The AI then selects the move with the highest evaluated value and returns it to Main Window to apply it to the UI.

This approach guarantees optimal play from the AI and simulates a deep level of strategic foresight while keeping execution fast.

8. Replay System

• Moves are stored as integers (e.g., 10 for X on cell 0, 21 for O on cell 1).
• Move history is saved in the database as a comma-separated string.
• Replay button fetches the move list and applies them step by step.

9. Error Handling

• Validates database access and file permissions.
• Prevents duplicate usernames or phone numbers.
• Detects invalid moves.
• Handles corrupted game records safely during replay.

10. Future Enhancements

• Sound effects and animations.
• Online multiplayer with sockets.
• Avatar system and advanced statistics vithe delay fixes this problem by differing the saved timestamps.