

# Cairo University Faculty of Engineering Dept. of Electronics and Electrical Communications Second Year Embedded systems

**Advanced Tic Tac Toe Game  
AI-XO**

|  |  |  |
| --- | --- | --- |
| Student Name | Section | ID. |
| عمر احمد رجب بدير على | 3 | 9220513 |
| يوسف خالد عمر محمود | 4 | 9220984 |
| عمر ايمن امين محمد | 3 | 9220528 |
| محمد عصام عبدالعظيم ابراهيم سليمان | 3 | 9220720 |
| محمد احمد عيد الحكم | 3 | 9220647 |

Presented by

**Instructors:** **Dr.** Omar Nasr

Software Requirements Specification (SRS)

# Introduction:

## Purpose:

The purpose of this document is to detail the functional and non-functional requirements for the development of an advanced Tic Tac Toe game. The game will feature user authentication, personalized game history, and an intelligent AI opponent.

## Scope:

The project will follow best practices in software engineering, including secure user management, rigorous testing, and professional version control workflows.

It includes:

* Tic-Tac-Toe Gameplay
* AI Opponent
* User Authentication and Management
* Personalized Game History
* Graphical User Interface (GUI)
* Testing and Quality Assurance
* CI/CD Integration
* Performance Optimization

## Definitions, Acronyms, and Abbreviations:

* **AI**: Artificial Intelligence
* **GUI**: Graphical User Interface
* **SRS**: Software Requirements Specification
* **CI/CD**: Continuous Integration/Continuous Deployment
* **UML**: Unified Modeling Language

# Overall Description:

## Product Requirement:

|  |  |
| --- | --- |
| Software Requirements Specification | |
| Name | Advanced Tic Tac Toe Game |
| purpose | advanced Tic Tac Toe game with AI player and user profile and history |
| inputs | User name, Password, Player Moves (Mouse Clicks) |
| Output | Board Display, User Profile Information, Game History |

## Functions:

* Interactive gameplay for two players or player vs. AI
* User authentication and profile management
* Personalized game history tracking
* AI opponent using the Minimax algorithm with alpha-beta pruning
* GUI with game board, login, registration, and history views

## User Classes and Characteristics:

* **General Users**: Casual players looking to play Tic Tac Toe.
* **Registered Users**: Users with an account, who can log in and track their game history.
* **Developers**: Programmers and testers who will be involved in the development and maintenance of the game.

## Operating Environment

* **Hardware**: Any standard PC or laptop
* **Operating System**: Windows, macOS, Linux
* **Software**: C++ compiler, Qt Framework, QT Test, GitHub Disktop

# Specific Requirements:

## Functional Requirements:

### User Interface Requirements:

* **GUI Design**: The game will feature a graphical interface displaying the Tic Tac Toe board, user login and registration forms, and game history views.
* **Interaction**: Users can interact with the game through mouse clicks to place their marks on the board.
* **User Feedback**: The GUI will provide immediate feedback on the game state (e.g., win, loss, draw).

### Game Logic Requirements

* **Turn-Taking Mechanism**: The system will alternate turns between two players or between a player and the AI.
* **Win/Tie Detection**: The system will check for a win or a tie after each move.
* **Move Validation**: The system will ensure moves are made in valid, empty spots on the board.

### AI Requirements:

* **Minimax Algorithm**: The AI opponent will use the Minimax algorithm with alpha-beta pruning to make strategic moves.

### User Authentication and Management:

* **Registration**: Users can create an account with a unique username and password.
* **Login**: Users can log in to access personalized features.
* **Profile Management**: Users can update their profile information.
* **Password Security**: Passwords will be stored securely using hashing.

### Personalized Game History:

* **Save Game History**: The system will save details of each game session for logged-in users.
* **View Game History**: Users can view their past game history, including outcomes and moves.
* **Replay Games**: Users can replay past games from their history.

## Non-functional Requirements:

### Performance Requirements:

* **Response Time**: The system should respond to user inputs within 0.2 micro second.
* **AI Computation Time**: The AI should make a move within 1 micro seconds.

### Usability Requirements:

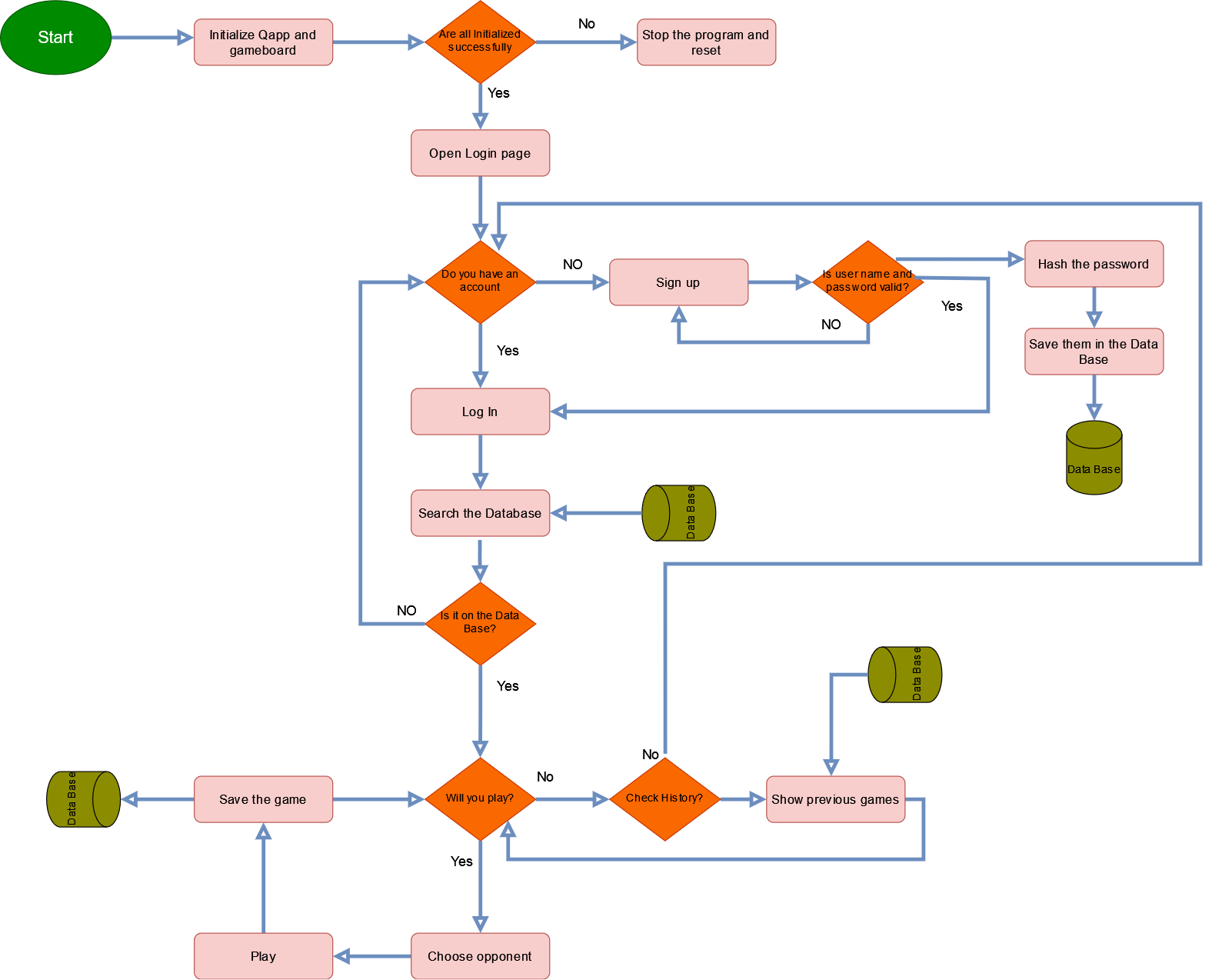
* **Ease of Use**: The interface should be intuitive and easy to navigate.

### Reliability Requirements:

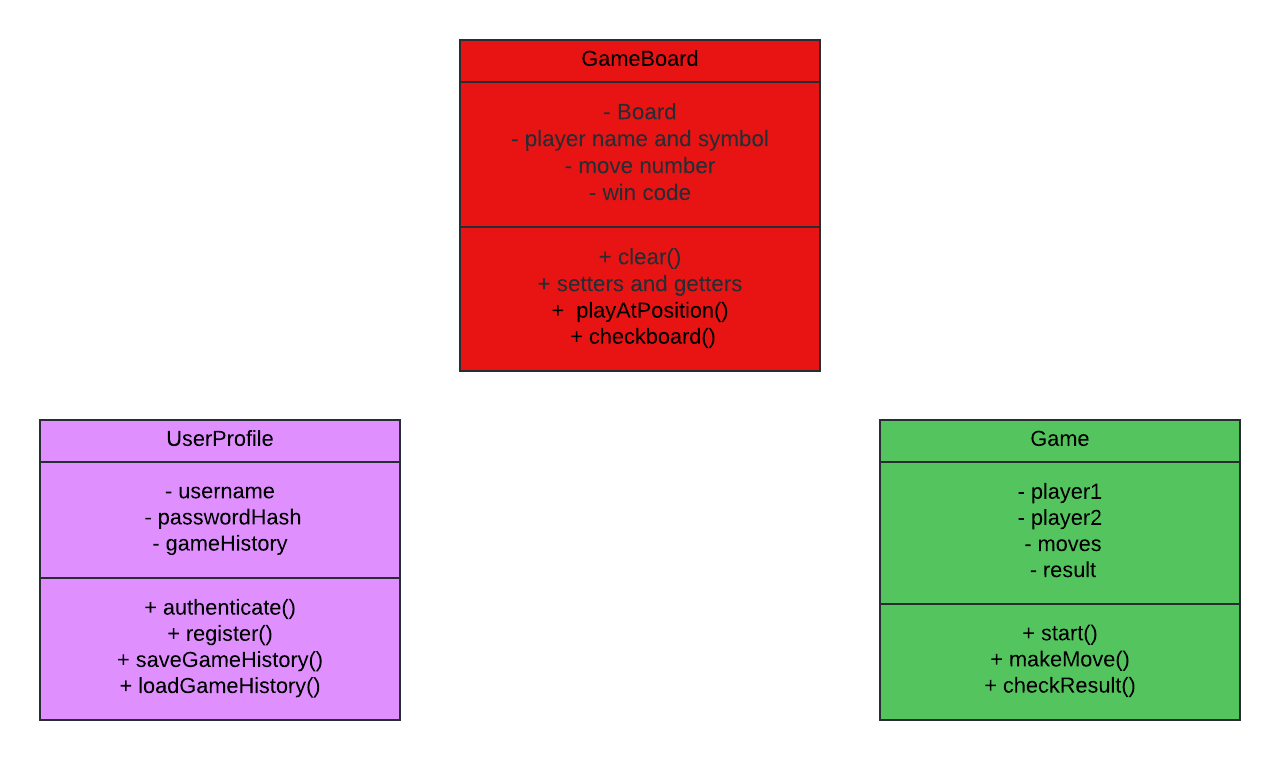
* **Error Handling**: The system should handle errors gracefully and provide informative messages to the user.
* **Cross-Platform Support**: The system should run on Windows, macOS, and Linux without modification.

Software Design Specification (SDS)

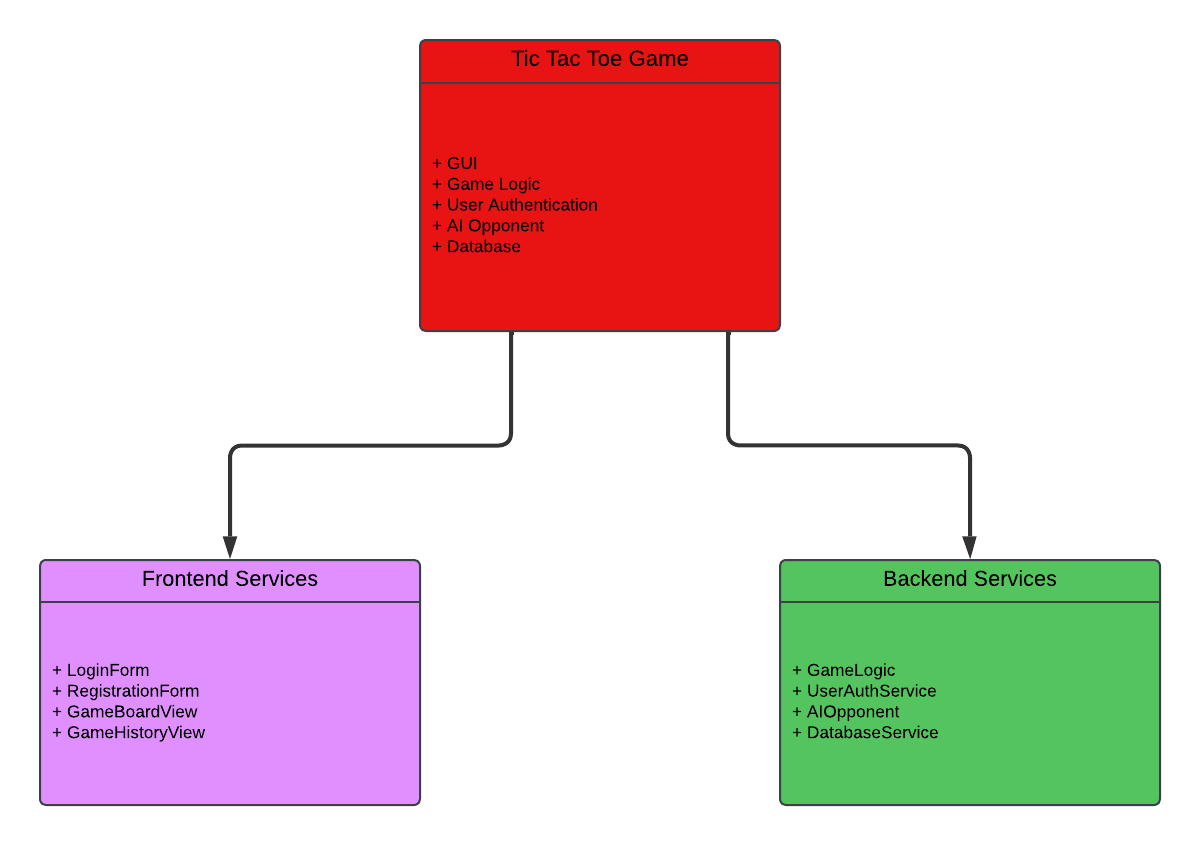
# Flow Chart:



# Class Diagram:



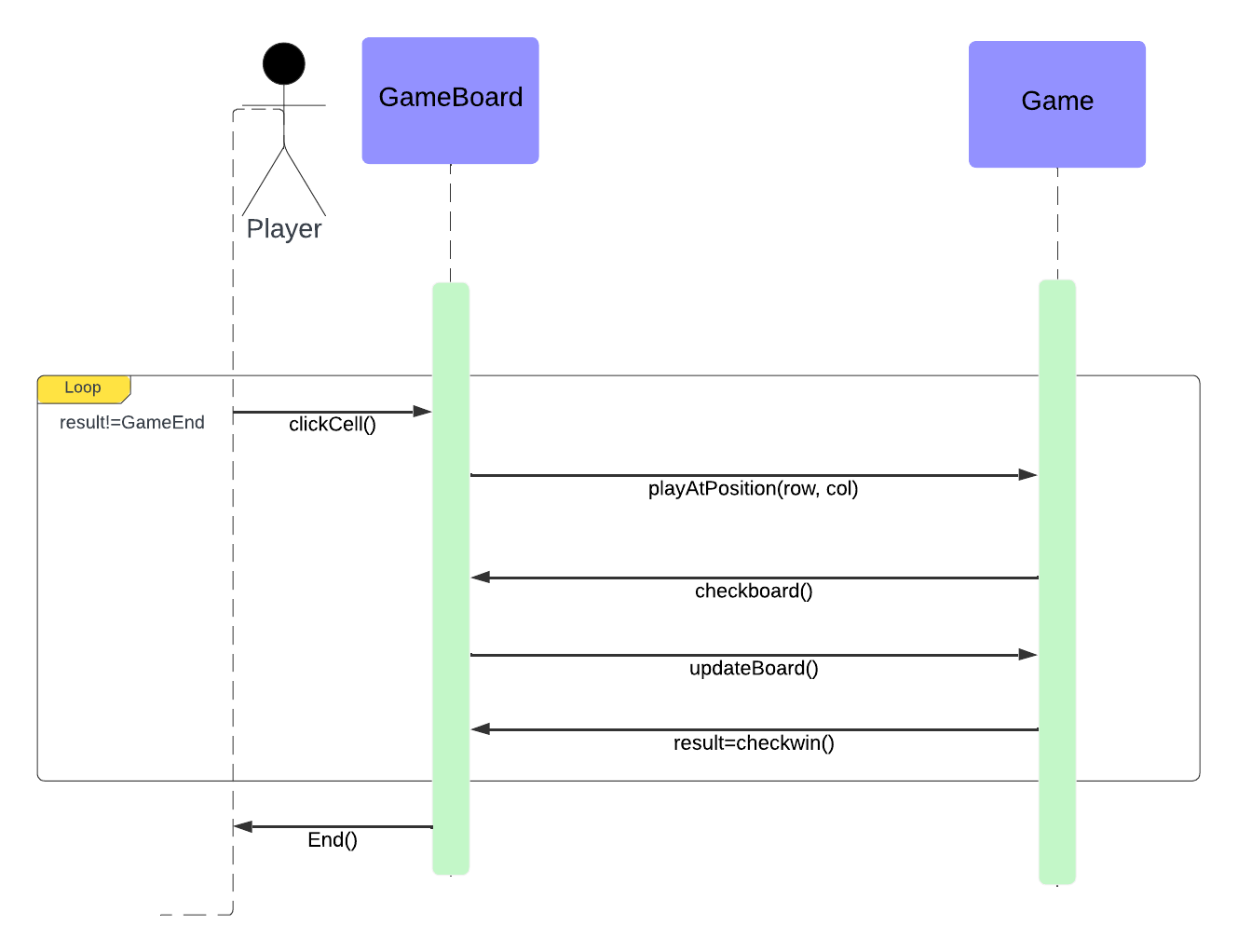
# Component Diagram:



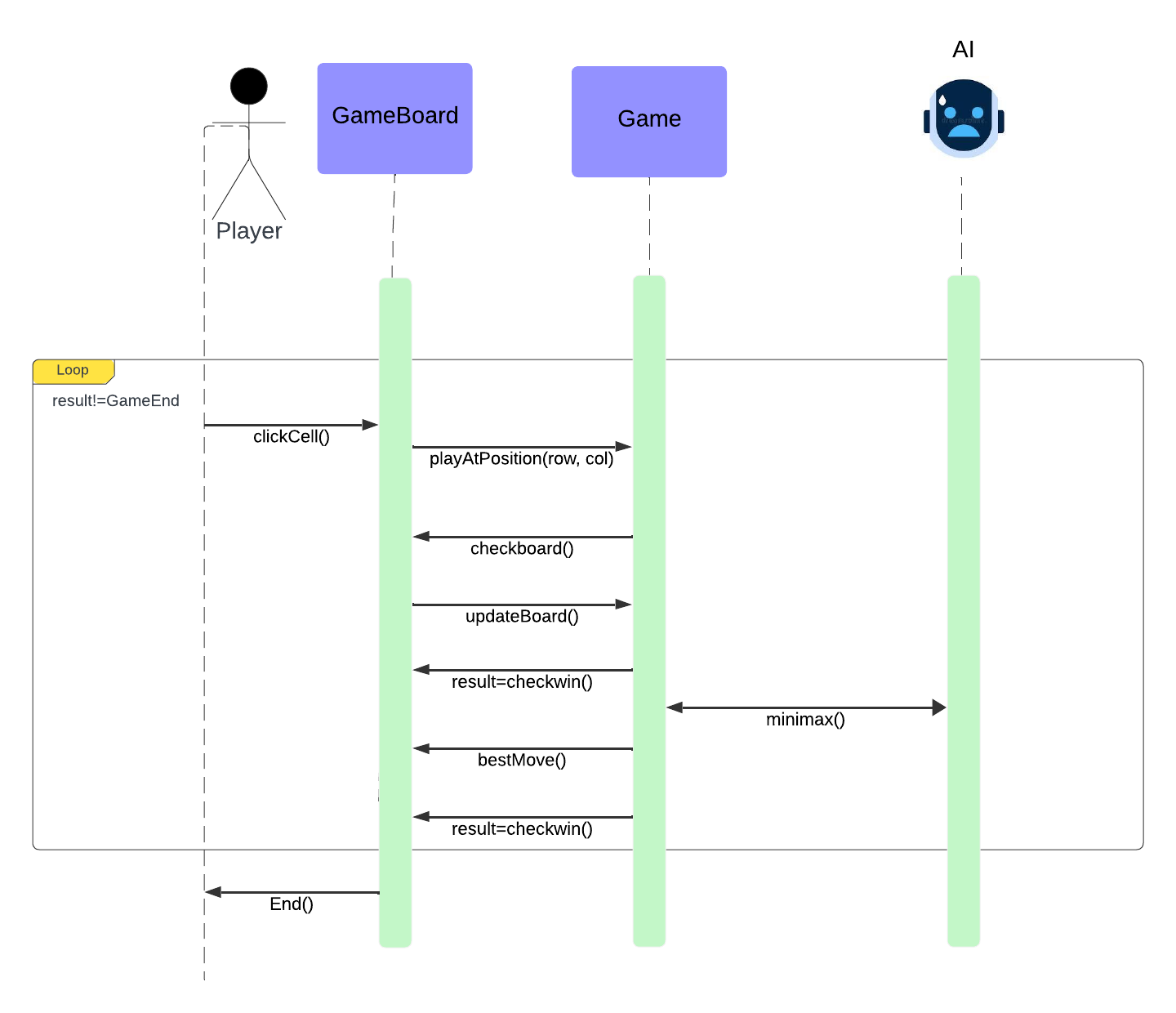
# Deployment Diagram:

# Sequence Diagram:

## 2 Players Sequence:



## AI Sequence:



Performance Measurement and Optimization

# AI vs CPU Response:

## AI Time Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AI** | Move | Game 1 (Draw) | Game 2 (Win fast) | Game 3 (Win Slow) |
| 1 | 0.1us | 0.1us | 0.1u |
| 2 | 0.1us | 0.1us | 0.1u |
| 3 | 0.2us | 0us | 0u |
| 4 | 0.1us |  | 0.1u |

As shown in the table above, The AI time response increases with the number of moves (As it process more possibilities), we can say that the average time is 0.15us

## CPU Time Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CPU** | Move | Game 1 (Draw) | Game 2 (Win) | Game 3 (Lose) |
| 1 | 0.1us | 0.1us | 0.1us |
| 2 | 0.1us | 0.1us | 0.1us |
| 3 | 0.1us | 0.2us | 0.1us |
| 4 | 0.1us | 0.3us |  |

As shown in the table above, The CPU sometimes after few moves takes more response time as it gives a random position that’s already played so we can say that the average time is 0.2us

# Log In vs Sign Up Response:

## Log In Time Response:

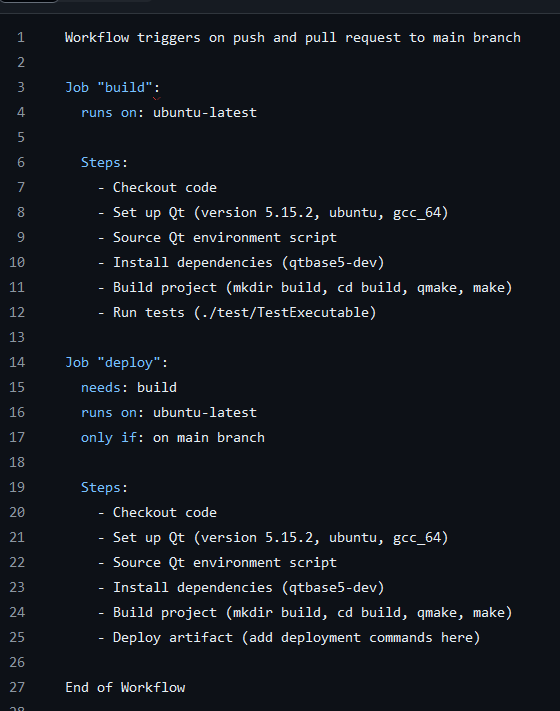
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Log In** | Accounts | Trial 1 (no Data Base) | Trial 2 (small Data Base) | Trial 3 (big Data Base) |
| 1 | 20.4us |  |  |
| 2 | 11.1us |  |  |
| 3 | 8.8us |  |  |
| 4 | 11.8us |  |  |

## Sign Up Time Response:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sign Up** | Accounts | Trial 1 (no Data Base) | Trial 2 (small Data Base) | Trial 3 (big Data Base) |
| 1 | 20.4us |  |  |
| 2 | 11.1us |  |  |
| 3 | 8.8us |  |  |
| 4 | 11.8us |  |  |

CI/CD Pipeline Configuration

# Pseudo code:



# Explanation:

The CI/CD pipeline automates the process of building, testing, and deploying the project. It triggers on every push to the `main` branch and on pull requests to `main`. The workflow includes two jobs: `build` and `deploy`. The `build` job compiles the project, runs tests, and uploads the test results. The `deploy` job is executed only when changes are made to the `main` branch and it deploys the built artifact.

Qt Test