Software Design Specification (SDS)

By

The X Factor

Table of Contents:

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed software design specification for the Advanced Tic Tac Toe game. This document outlines the system architecture, design, and components required to build a Tic Tac Toe game with both Player vs Player and Player vs AI modes, including user authentication and game history tracking.

1.2 Scope

The Advanced Tic Tac Toe game will include the following features:

- **Two modes of play**: Player vs Player and Player vs AI.
- **Three levels of AI difficulty**: Easy, Medium, and Hard.
- User authentication with username and password storage.
- Game history tracking for each player.
- Graphical User Interface (GUI) for mode selection, login, and gameplay.
- Database integration for user management and game history storage.

1.3 Definitions, Acronyms, and Abbreviations

- **AI**: Artificial Intelligence
- **GUI**: Graphical User Interface
- **SDS**: Software Design Specification

2. System Overview

The system consists of the following major components:

- **User Interface**: Allows players to interact with the game.
- **Game Logic**: Contains the rules and logic for gameplay.
- **AI Module**: Provides different levels of AI difficulty.
- **Database**: Stores user credentials and game history.

3. System Architecture

The system architecture is divided into three layers:

- **Presentation Layer**: GUI for mode selection, login, and gameplay.
- **Application Layer**: Handles game logic, AI, and user authentication.
- **Data Layer**: Manages database interactions for user credentials and game history.

4. Detailed Design

4.1 User Interface

The user interface is designed to be intuitive and user-friendly, with the following screens:

- **Main Menu**: Options to play against another player or AI.
- **Difficulty Selection**: Choose AI difficulty level.
- **Authentication**: Login or sign up screen.
- **Game Screen**: Display the tic-tac-toe grid and handle user input.

4.2 Game Logic

The game logic will manage the core functionality of the game:

- **Grid**: Represents the Tic Tac Toe board. It allows placing symbols, checking for wins or draws, and resetting the board.
- **Player**: Manages player information, including name and symbol. It also handles player moves.
- **AI**: Extends the Player class to implement AI functionality with different difficulty levels. Uses algorithms like Minimax for decision-making.

4.3 AI Module

The AI module will provide three levels of difficulty:

- **Easy**: Random move selection.
- **Medium**: Minimax algorithm with limited depth.
- **Hard**: Full-depth Minimax algorithm.

4.4 Database

The database will store user credentials and game history:

- **Users Table**: Stores usernames and hashed passwords.
- **Passwords Table**: Stores hashed passwords separately for security.
- **Game History Table**: Stores game results including player names.

5. Implementation

5.1 Programming Languages and Tools

The project is implemented using:

- Programming Languages: C++
- Tools: Eclipse IDE, SQLite for database management, Qt for GUI.

5.2 Main Program

The main program initializes the system, handles user interactions, and coordinates between the UI, game logic, and database.

5.3 Player Class

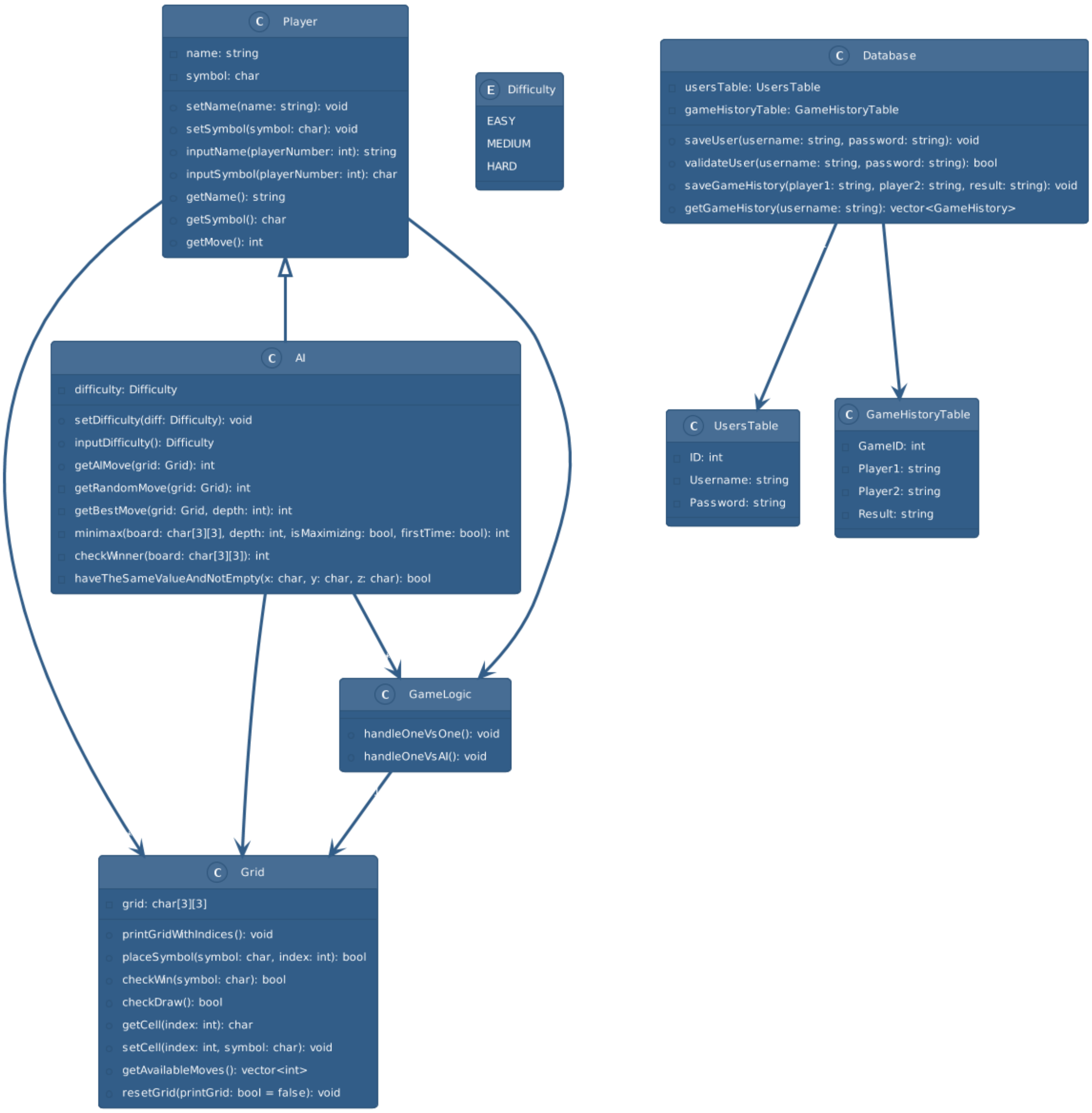
The Player class manages player details, including name, symbol, and moves.

5.4 AI Class

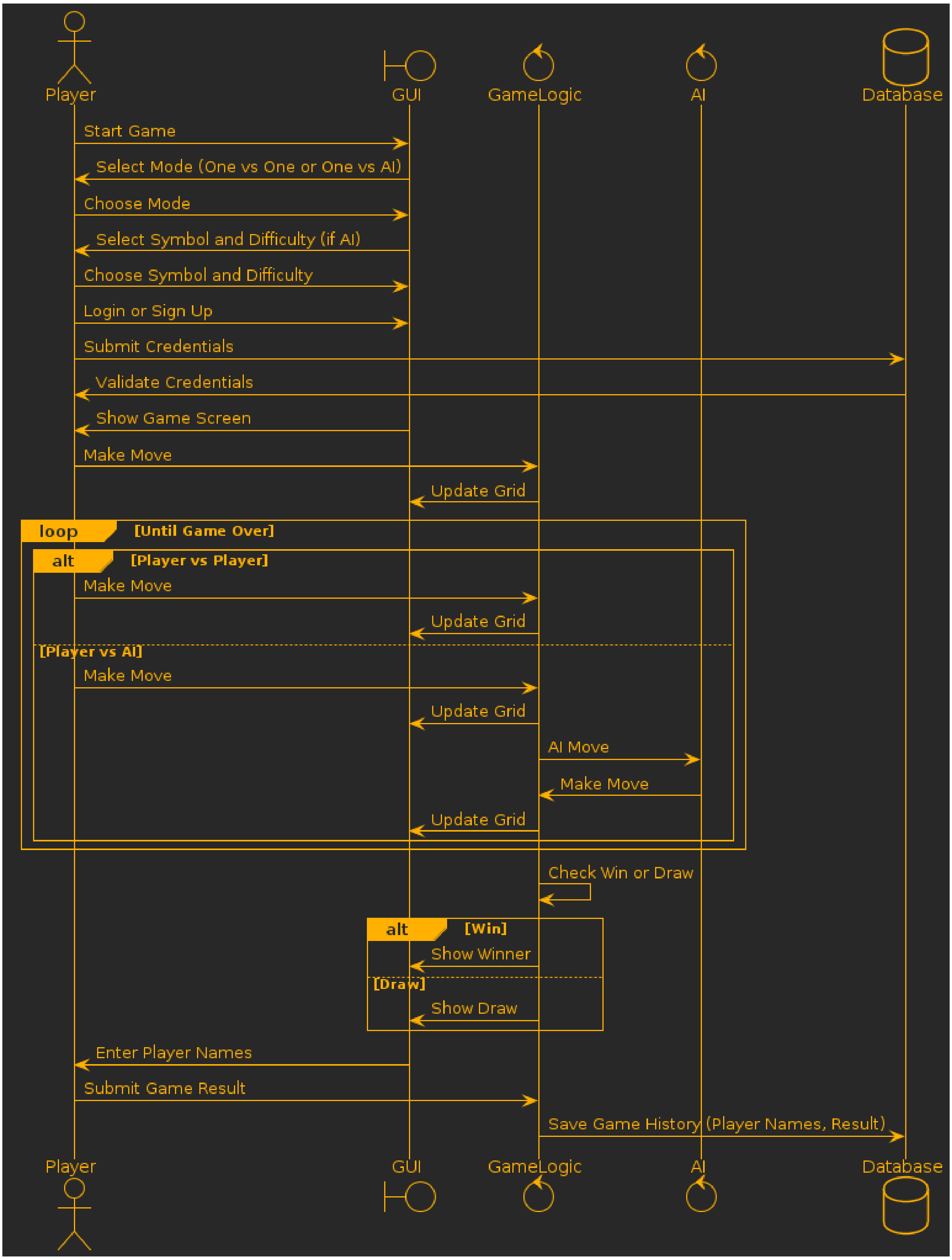
The AI class extends the Player class and includes methods to calculate the AI's moves based on the difficulty level.

6. Diagrams

6.1 Class Diagram


C Player
name: string
symbol: char
setName(name: string): void
setSymbol(symbol: char): void
inputName(playerNumber: int): string
inputSymbol(playerNumber: int): char
getName(): string
getSymbol(): char
getMove(): int
E Difficulty
EASY
MEDIUM
HARD
C Database
usersTable: UsersTable
gameHistoryTable: GameHistoryTable
saveUser(username: string, password: string): void
validateUser(username: string, password: string): bool
saveGameHistory(player1: string, player2: string, result: string): void
getGameHistory(username: string): vector<GameHistory>
C AI
difficulty: Difficulty
setDifficulty(diff: Difficulty): void
inputDifficulty(): Difficulty
getAIMove(grid: Grid): int
getRandomMove(grid: Grid): int
getBestMove(grid: Grid, depth: int): int
minimax(board: char[3][3], depth: int, isMaximizing: bool, firstTime: bool): int
checkWinner(board: char[3][3]): int
haveTheSameValueAndNotEmpty(x: char, y: char, z: char): bool
C UsersTable
ID: int
Username: string
Password: string
C GameHistoryTable
GameID: int
Player1: string
Player2: string
Result: string
C GameLogic
handleOneVsOne(): void
handleOneVsAI(): void
C Grid
grid: char[3][3]
printGridWithIndices(): void
placeSymbol(symbol: char, index: int): bool
checkWin(symbol: char): bool
checkDraw(): bool
getCell(index: int): char
setCell(index: int, symbol: char): void
getAvailableMoves(): vector<int>
resetGrid(printGrid: bool = false): void

6.2 Sequence Diagram



7. Database Design

Users Table:

- id (Primary Key)
- username (Unique)
- hashed password

Game History Table:

- id (Primary Key)
- player1_username
- player2_username
- result (Win/Loss/Draw)
- date

8. Testing

8.1 Unit Testing

Test individual components such as Player, AI, and Grid classes.

Use mock objects for database interactions.

8.2 Integration Testing

Test the interaction between the user interface and game logic.Ensure the AI module integrates correctly with the game logic.

8.3 User Acceptance Testing

Conduct testing sessions with actual users to ensure the game meets requirements and provides a satisfactory user experience.

9. Conclusion

This SDS provides a comprehensive overview of the design and implementation plan for the Advanced Tic Tac Toe game. The outlined architecture and components will ensure a robust and enjoyable game experience with features like AI opponents and user authentication.