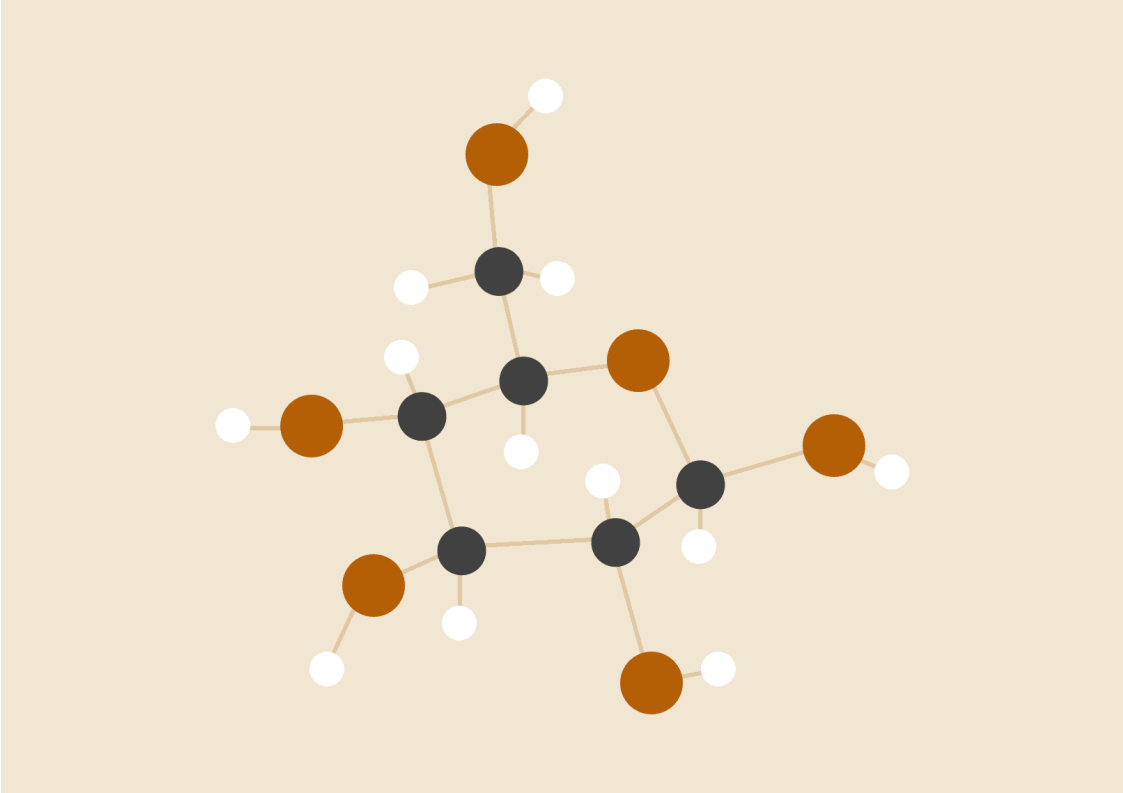
3SIS TIC-TAC-TOE

"Tic Tac Toe - Simple but extremely attractive game for all ages."



Game Developer

Dương Thùy Trang - ITITDK21074

Nguyễn Hà Khánh Vy - ITITDK21075

Đỗ Hoàng Mỹ Dung - ITITWE21040

Advisor

Prof. Nguyen Trung Nghia

CHAPTER 1: INTRODUCTION

1.1 Gaming in the Field

Welcome to the captivating world of Tic-Tac-Toe, a timeless classic that has entertained generations with its simple yet engaging gameplay. Known globally and cherished by people of all ages, Tic-Tac-Toe, or "Noughts and Crosses," is more than just a game; it's a strategic battle of wits and foresight.

In this report, we will delve into how the principles of Object-Oriented Programming (OOP) can be applied to build a Tic-Tac-Toe game. By utilizing OOP, we will create a version of Tic-Tac-Toe whose structure and source code are not only easy to understand but also easy to maintain and extend. We will explore the fundamental classes and objects required to implement the game, how to organize the source code according to OOP principles, and how design patterns can be applied to optimize gameplay and enhance the scalability of the game.

Join us as we explore how a classic game can be upgraded and improved through the power of Object-Oriented Programming, and discover why applying OOP principles is essential in modern software development.

1.2 About Tic-Tac-Toe game project

Operation for 3SIS Tic-Tac-Toe:

- Starting the Game.
- Player Turns.
- Marking Cells.
- Winning Condition.
- Scoring.
- Restarting or Quitting.

These operations form the basic flow of a Tic-Tac-Toe game. Depending on the implementation, additional features such as keeping track of scores, offering player options like undoing moves or choosing different grid sizes, and integrating multiplayer capabilities can enhance the gameplay experience.

1.3 About “3sis Tic-Tac-Toe”

Tic-tac-toe, a simple yet engaging puzzle game, offers several benefits, especially when it

comes to cognitive development and social interaction.

This game is typically played on a 3x3 square grid, where players take turns placing "X" or "O" on empty squares until one of the players wins or the game ends in a draw.

Interface

- **Main Screen:** The main screen will display the TicTacToe board with empty squares for players to place "X" or "O".
- **Color Scheme:** The pink-purple color will dominate the interface, creating a cute and friendly atmosphere. Empty squares can be highlighted with a brighter color to help players identify them easily.
- **Square Indices:** Each square on the board can be numbered from 1 to 9, aiding players in locating the square they want to place "X" or "O".

Gameplay Experience

- **Offline Play Mode:** Players can choose the offline play mode to play against the computer or with friends on the same device.
- **Computer Opponent:** The game can integrate artificial intelligence to act as an opponent, providing a challenging single-player experience.
- **Visual Effects:** Simple graphic effects like animations when "X" and "O" are placed on the board or when the board is refreshed can add to the game's appeal.

Overall, TicTacToe with a pink-purple interface and cute, friendly features will be an enjoyable entertainment experience for all ages.

1.4 References

Tutorial Android App Development for Beginner

1. https://www.youtube.com/watch?v=Nc77ymnm8Ss
2. https://www.youtube.com/watch?v=EQKzgTslxKo
3. https://www.youtube.com/watch?v=3T2NmMpFmJI
4. https://www.youtube.com/watch?v=rA7tfvpkw0I

Color

1. https://www.rapidtables.com/web/color/RGB_Color.html?fbclid=IwAR3h2QmmMZjbGZfq0Le_kIo7F2l0RXF0VIIaYHTGGrua9HPf0lGy61Oy3Nw

1.5 Developer team

NAME	ID	CONTRIBUTION
Dương Thùy Trang	ITITDK21074	Game Developer Write Report(chapter 1,2,5) Presente
Nguyễn Hà Khánh Vy	ITITDK21075	Game Developer Write Report(chapter 3, 5) Presenter
Đỗ Hoàng Mỹ Dung	ITITWE21040	Game Developer Write Report(chapter 4, 5) Presenter

CHAPTER 2: REQUIREMENTS

2.1 Possession

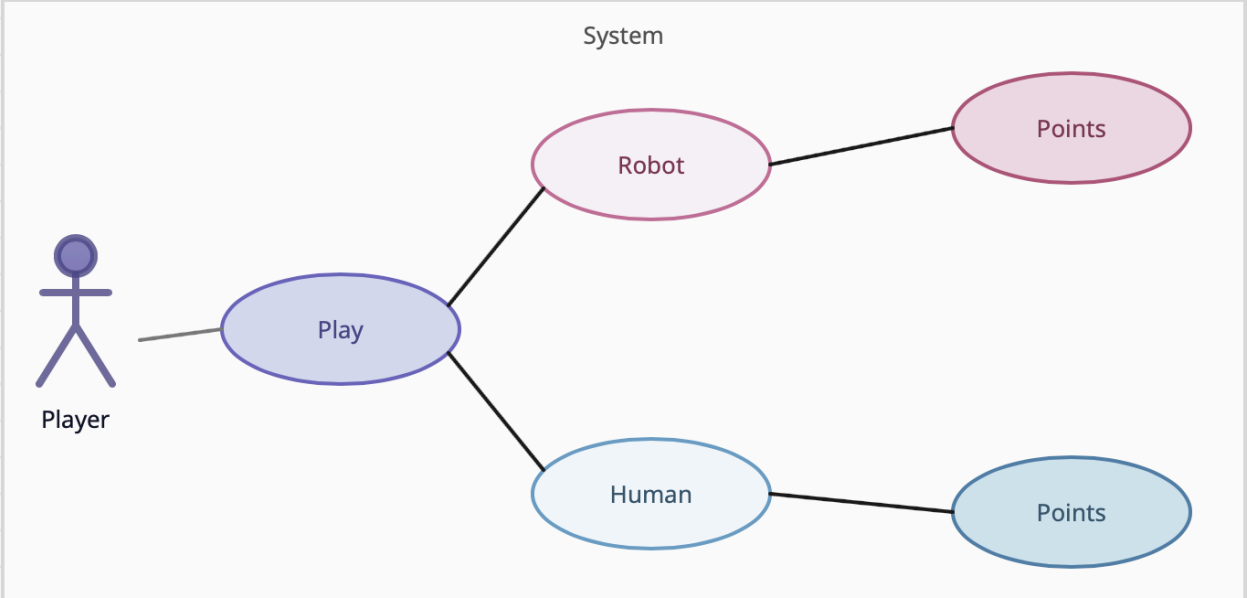
- Easy to use.
- Suitable for all ages.
- Beautiful and engaging graphics.
- Developed with measured and professional programming.
- Intelligent and intellectually stimulating game.

2.2 Desires

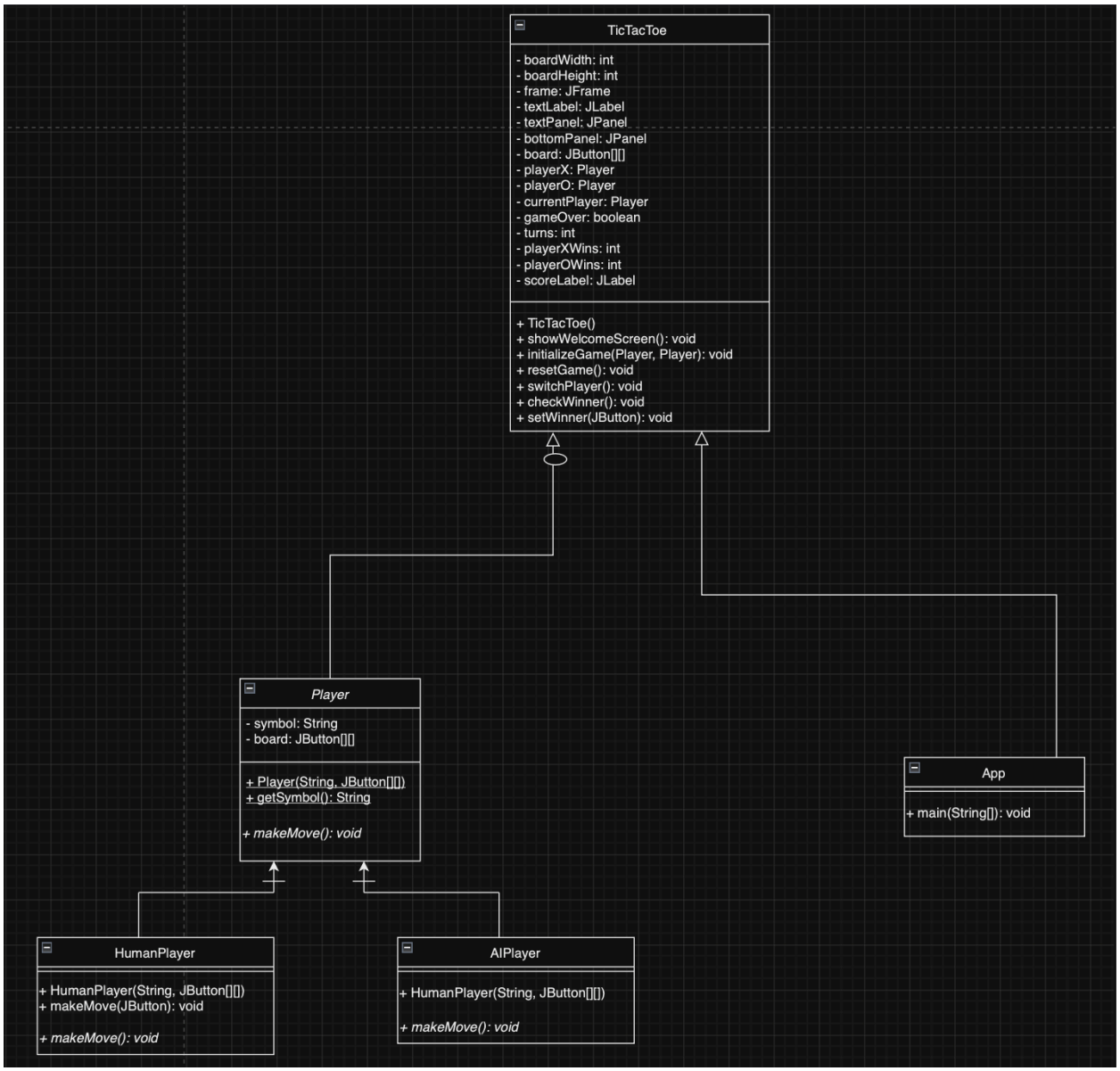
- **User Interface (UI)**: Create an intuitive and visually appealing interface that allows players to easily interact with the game.
- **Functionality**: Ensure that the game functions correctly according to the rules of Tic Tac Toe. This includes handling player moves, checking for a winner, and handling ties.
- **Platform Compatibility**: Make sure the game can run smoothly on various platforms, such as desktop computers, mobile devices, and web browsers.
- **Offline Enjoyment:** The game will be enjoyed without the need for an internet connection, allowing for offline play.
- **Efficient Design:** The game is designed to be lightweight in terms of storage requirements, ensuring it doesn't consume an excessive amount of disk space.

2.3 UML Diagram

Use Case Diagram



Class Diagram



CHAPTER 3: DESIGN & IMPLEMENTATION

3.1 TECHNIQUES & TOOLS

- **Github's Version Control System:** was used to control the system due to its robust features that facilitate collaboration, code management, and project tracking. Throughout the development process, GitHub provided a centralized repository to store all versions of the project files, enabling efficient management of code changes and history. Additionally, GitHub's issue tracking and project

boards helped in organizing tasks, monitoring progress, and ensuring that development milestones were met on time. By leveraging GitHub's branching and merging capabilities, we were able to experiment with new features and bug fixes without disrupting the main codebase.

- **Visual Studio Code for Coding:** was the primary integrated development environment (IDE) used for coding the Tic Tac Toe game. We selected it because of its lightweight nature, powerful features, and extensive ecosystem of extensions that cater to various programming needs. Furthermore, VSCode's seamless integration with GitHub allowed for easy synchronization of the local repository with the remote one, streamlining the workflow and ensuring that code changes were consistently tracked and updated.

3.2 CODING

Player class: This report examines the abstract Player class designed for use in a board game application, potentially similar to Tic-Tac-Toe. The Player class serves as a foundational blueprint for different types of players, providing essential properties and methods that all player types will share.

Abstract Class Player : An abstract class is a class that cannot be instantiated on its own and is meant to be subclassed. It can contain abstract methods (methods without a body) that must be implemented by subclasses.

```java
public abstract class Player {
```

- **protected:** These variables are accessible within the same package and by subclasses.
- **String symbol:** Stores the player's symbol (e.g., "X" or "O" in Tic-Tac-Toe).
- **JButton[][] board:** A 2D array of JButton objects representing the game board.

```java
import javax.swing.JButton;

public abstract class Player {
    protected String symbol;
    protected JButton[][] board;
```

The constructor initializes the symbol and board fields when a **Player** object is created. This ensures that each player has a defined symbol and a reference to the game board.

```
public Player(String symbol, JButton[][] board) {
    this.symbol = symbol;
    this.board = board;
}
```

- public Player(String symbol, JButton[][] board): Constructor that initializes the symbol and board fields with the provided values.
- this.symbol = symbol: Assigns the provided symbol to the instance's symbol field.
- this.board = board: Assigns the provided board to the instance's board field.

Getter Method and Abstract Method:

```
public String getSymbol() {
    return symbol;
}
```

- `getSymbol()`: This method provides access to the player's symbol, allowing other parts of the program to retrieve and use it as needed.

- `makeMove()`: This is an abstract method that subclasses must implement. It defines how the player makes a move on the board. The implementation will vary depending on the type of player (e.g., human player, computer player).

```
public abstract void makeMove();
}
```

- **HumanPlayer class:** the HumanPlayer class, a concrete subclass of the abstract

Player class. The HumanPlayer class defines how a human player interacts with the game board by implementing the makeMove method. The class uses the JButton component from the Swing library to represent the cells on the game board.

```java
import javax.swing.JButton;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class HumanPlayer extends Player {

    public HumanPlayer(String symbol, JButton[][] board) {
        super(symbol, board);
    }

    public void makeMove(JButton tile) {
        if (tile.getText().equals(anObject:"")) {
            tile.setText(symbol);
        }
    }
}
```

- The HumanPlayer class extends the abstract Player class, inheriting its fields and constructors. It provides a concrete implementation of the makeMove method, allowing a human player to make a move by interacting with a specific button on the game board.

Constructor: The constructor initializes the **HumanPlayer** with a symbol and a reference to the game board by invoking the constructor of the superclass (Player).

- symbol: The symbol assigned to the human player (e.g., "X" or "O").
- board: The 2D array of JButton objects representing the game board.

```java
public class HumanPlayer extends Player {

    public HumanPlayer(String symbol, JButton[][] board) {
        super(symbol, board);
    }
```

makeMove(JButton tile): This method allows the human player to make a move by setting their symbol on a selected button (tile) on the game board.

- **tile**: A **JButton** object representing the cell on the game board where the player wants to make a move.
- The method checks if the text of the **JButton** (tile) is an empty string, indicating that the cell is currently unoccupied.
- If the cell is unoccupied, it sets the text of the **JButton** to the player's symbol, effectively marking the player's move.

```
public void makeMove(JButton tile) {
    if (tile.getText().equals(anObject:"")) {
        tile.setText(symbol);
    }
}
```

The **makeMove** method directly modifies the text of a **JButton** on the game board, which is an intuitive way for a human player to interact with the game. When a player clicks on an unoccupied cell (button), the method checks if the cell is empty and then marks it with the player's symbol.

AIPlayer class: which extends an abstract Player class and implements an AI player for a Tic-Tac-Toe game. The key focus of this implementation is the use of the Minimax algorithm to determine the optimal move for the AI player.

```
public class AIPlayer extends Player {

    public AIPlayer(String symbol, JButton[][] board) {
        super(symbol, board);
    }

    @Override
    public void makeMove() {
        Random rand = new Random();
        int r, c;
        do {
            r = rand.nextInt(bound:3);
            c = rand.nextIn  int c - AIPlayer.makeMove()
        } while (!board[r][c].getText().equals(anObject:""));

        board[r][c].setText(symbol);
    }
}
```

Constructor: The constructor

```
public AIPlayer(String symbol, JButton[][] board) {
    super(symbol, board);
}
```

initializes the AIPlayer with a symbol and a reference to the game board by invoking the constructor of the superclass (Player).

makeMove(): This method allows the AI player to make a move by randomly selecting an empty cell on the game board.

- A `Random` object is used to generate random indices for the row (`r`) and column (`c`).
- A `do-while` loop ensures that the selected cell is empty by checking if the text of the `JButton` at the random indices is an empty string.
- Once an empty cell is found, the method sets the text of the `JButton` to the AI player's symbol, effectively marking the player's move.

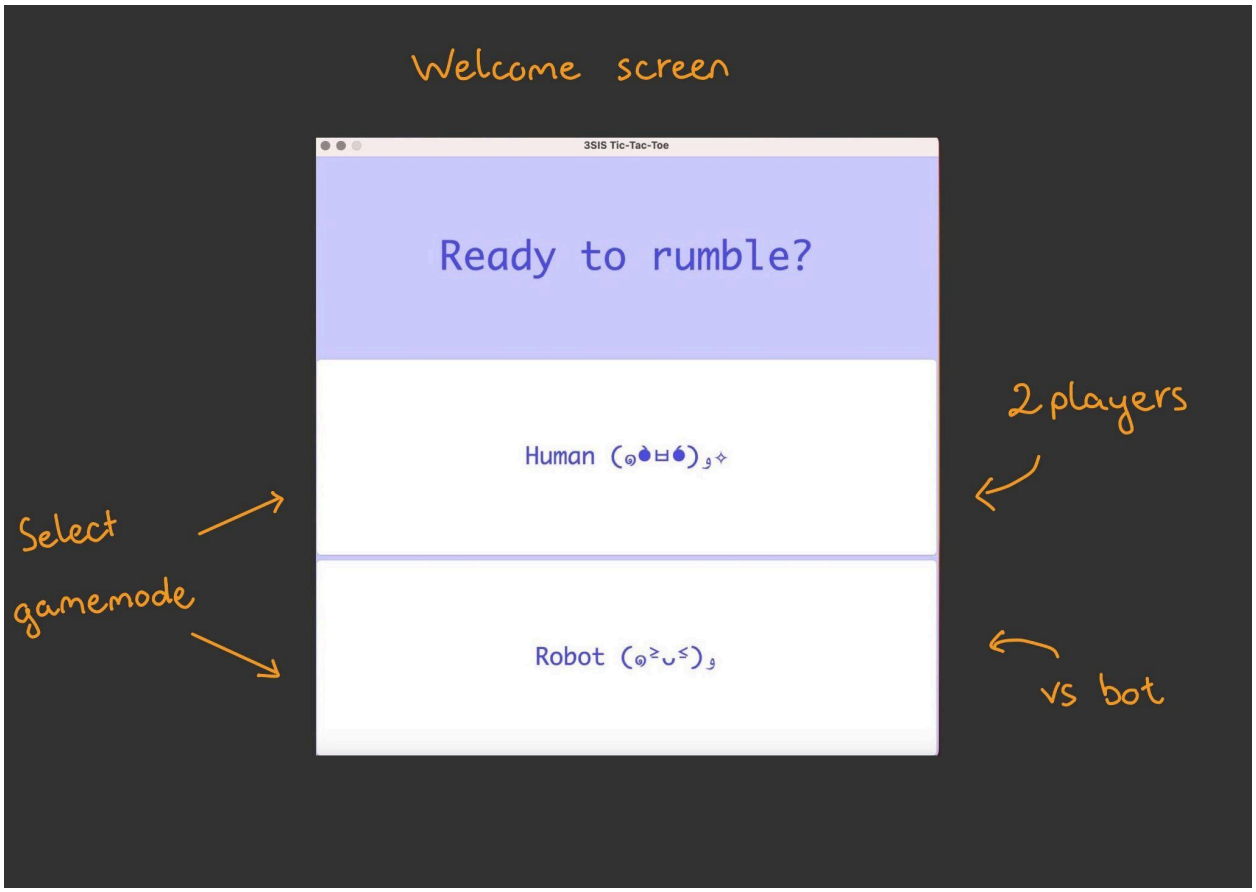
```
public void makeMove() {
    Random rand = new Random();
    int r, c;
    do {
        r = rand.nextInt(bound:3);
        c = rand.nextInt(bound:3);
    } while (!board[r][c].getText().equals(anObject:""));

    board[r][c].setText(symbol);
}
```

The current implementation of the `AIPlayer` class uses a simple random selection strategy to make moves. While this ensures that moves are made on empty cells, it does not provide any strategic advantage or consider the state of the game.

CHAPTER 4: GAMEPLAY

4.1 Welcome screen and game mode

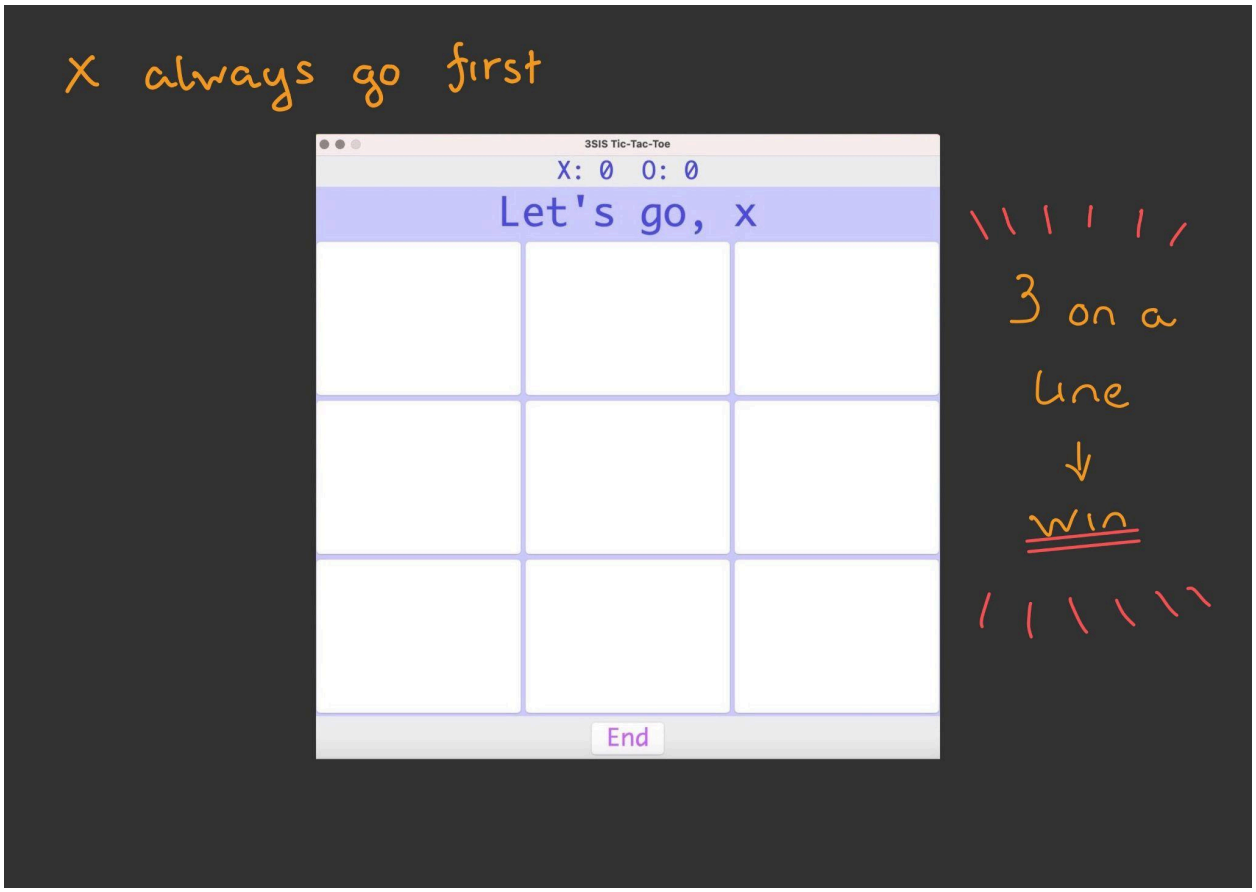
When you run the program, the game appears:



Select game mode to play against Human or Bot

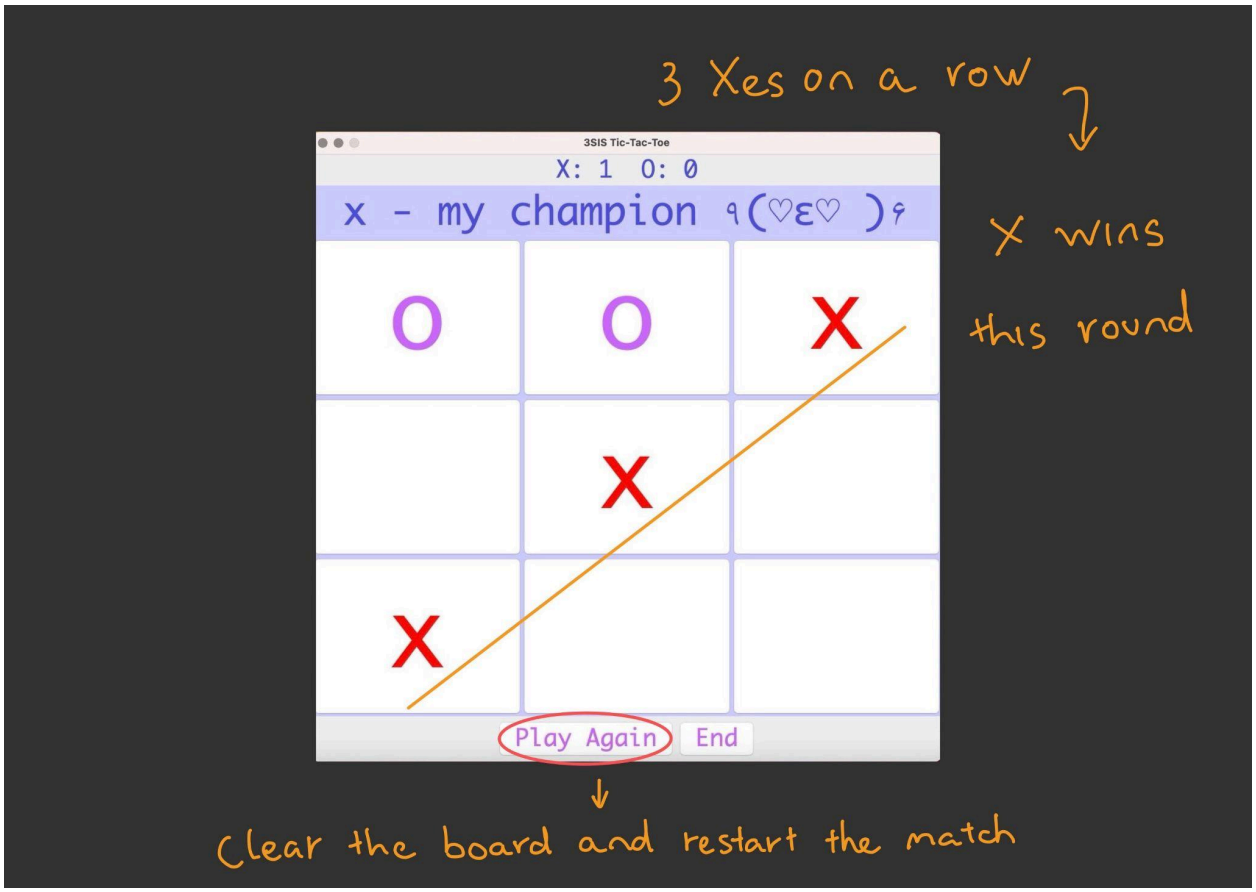
4.2 Gameplay and exit

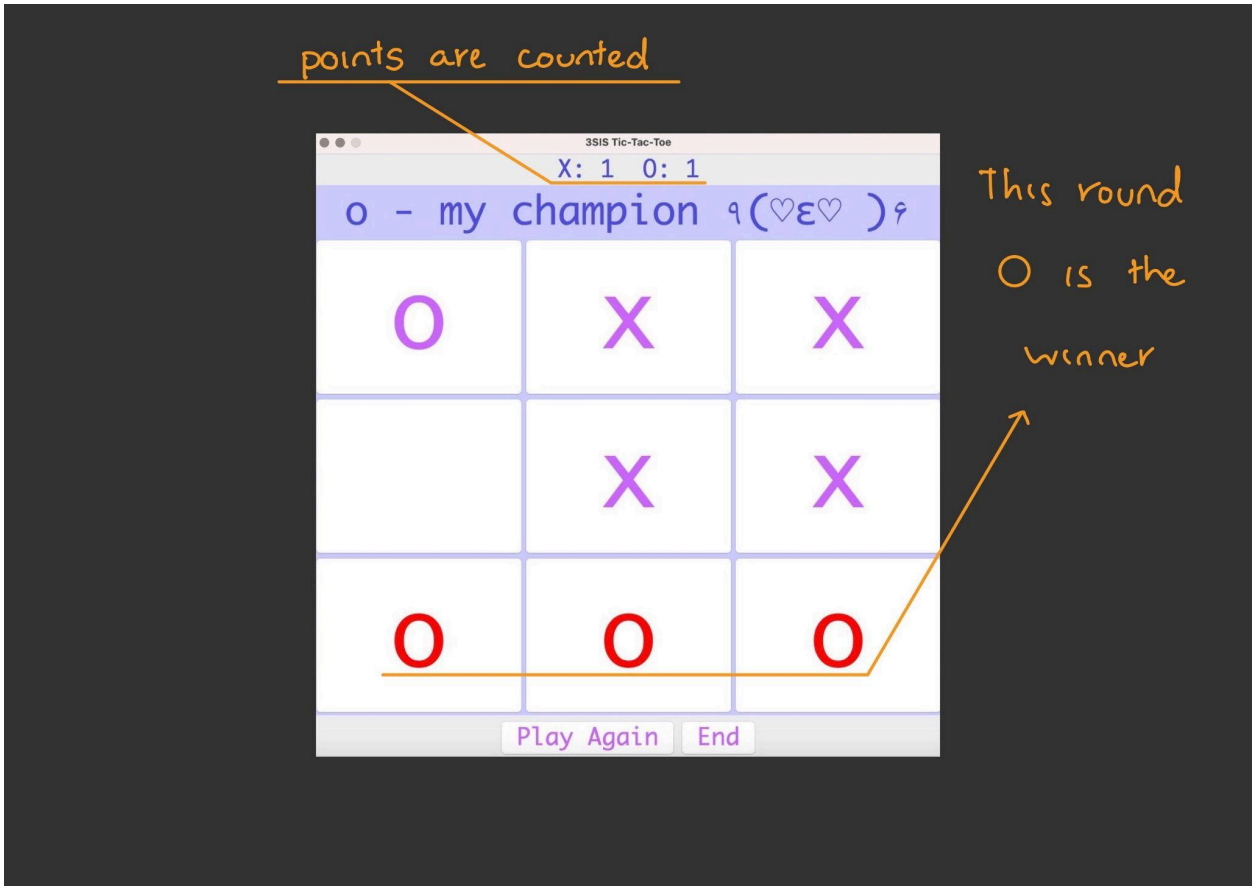
2 golden rules: X always go first & first to get 3 marks on a line wins



Point counting system helps to keep track of the game record

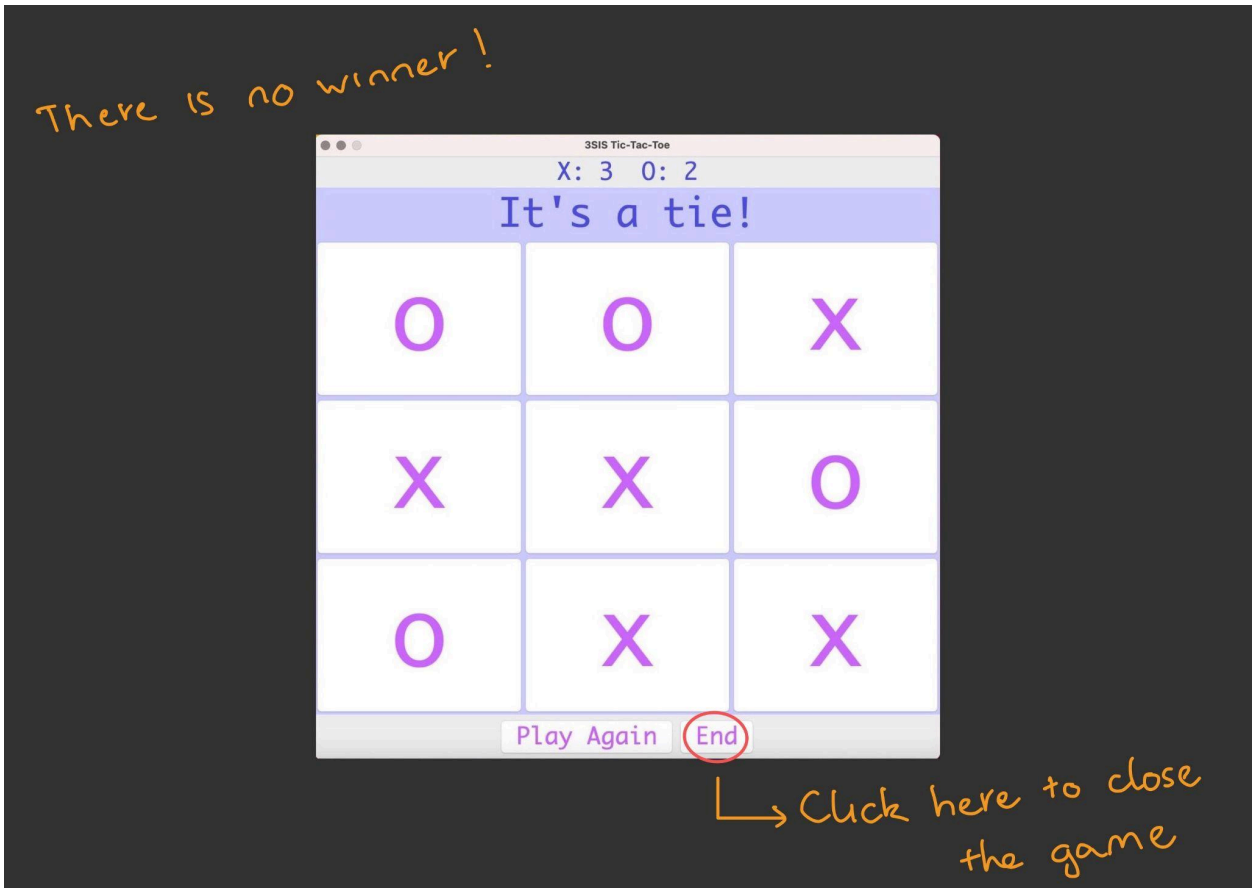
Click “Play again” to keep the score and start a new math


3 Xes on a row
3SIS Tic-Tac-Toe
X: 1 O: 0
x - my champion ٩(♡ε♡)۶
O
O
X
X
X
Play Again
End
X wins
this round
Clear the board and restart the match



If there is no winner, show “It’s a tie!”

Click on “Exit” to close the game



CHAPTER 5: EXPERIENCES

Throughout the development of this Tic-Tac-Toe game, we embarked on a journey filled with excitement and significance. Despite its simplicity and lack of sound, as well as the absence of an online play mode, this game still provided us with many memorable experiences.

Writing code for this game took us through a series of basic steps that we learned in our OOP class. From building the game's structure to implementing fundamental features, each part of the source code was an opportunity for us to apply and reinforce our knowledge.

Despite the absence of special effects, Tic-Tac-Toe remained a challenging and engaging endeavor for us. Creating a simple yet user-friendly interface was part of our journey.

We gained a deeper understanding of the importance of creating a game that is simple yet provides a good experience for players. Even without complex elements, this game

could still bring joy and challenge to players.

With this Tic-Tac-Toe game, we hope that players will enjoy the excitement and engagement it brings. Through this process, we learned that sometimes, simplicity is indeed the best approach.