Software Requirements Specification

for

TIC-TAC-TOE ANDROID APPLICATION

Prepared by

Bhuvana S	**202009008**	**bhuvana9902_ai@mepcoeng.ac.in**
Aksshaya B	**202009003**	**akksshaya_ai@mepcoeng.ac.in**

Instructor: ***Dr. P. Thendral, Asst. Professor (Sr. Grade), Department of AI & DS***

Course: **19AD691 – Principles of Software Engineering**

Date: **02/05/2023**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
Tic-Tac-Toe Android Application v1.0	Bhuvana S Aksshaya B	The First Version of this Application has been developed as an Interactive game between two Human Players.	22/03/2023
Tic-Tac-Toe Android Application v2.0	Bhuvana S Aksshaya B	The Revised Version comprises of a new module which allows the game to be played between a Human Player and an opponent Computer Player.	26/04/2023

1 Introduction

Tic-Tac-Toe is a popular two-player game played on a 3x3 grid. In this game, two players take turns marking Xs and Os on the grid until one player wins by placing three of their marks in a row, column, or diagonal, or the game ends in a tie if all the cells are filled without any player winning. The Tic-Tac-Toe Android Application project is an implementation of this game on the Android platform. The project aims to provide an enjoyable and user-friendly gaming experience to users, complete with a simple and intuitive user interface, customizable game options, and an AI opponent that adapts to the user's level of play.

In the Introduction section, the reader can expect to find a brief overview of the project, including its purpose, goals, and features. The section will also provide information on the project's scope, requirements, and intended audience. Additionally, the Introduction will introduce the project team and their roles, as well as the development process used to create Our Application. Overall, the Introduction section will provide a high-level view of the Tic-Tac-Toe Android Application project and set the stage for the rest of the documentation.

1.1 Document Purpose

The product whose software requirements are specified in this document is the Tic-Tac-Toe Android Application. The scope of this SRS covers the entire Tic-Tac-Toe Android Application, including its features, functions, and interfaces. This document provides a detailed description of the software requirements that Our Application should fulfill, such as the user interface, game rules, AI algorithms, and customization options. This SRS document describes the software requirements for the entire system and does not focus on a single subsystem or module. The document outlines the scope of the project and provides a clear understanding of what features and functionalities Our Application will have. It also describes the limitations and constraints that Our Application should follow, such as hardware and software compatibility requirements.

The purpose of this document is to serve as a reference for the project team and stakeholders to understand the software requirements of the Tic-Tac-Toe Android Application. This document provides a detailed description of Our Application's requirements, which can be used as a guide for the development, testing, and quality assurance processes. It also helps to ensure that Our Application meets the expectations of its users and stakeholders.

1.2 Product Scope

The Tic Tac Toe game is a game for two players, called "X" and "O", who take turns marking the spaces in a 3×3 grid. s. Both these players play on the client’s side. The server program evaluates the game and declares the winners or if the game is a draw. The player who succeeded in placing three respective marks in a horizontal, vertical, or diagonal row wins the game. The Tic Tac Toe is a great way to pass your free time whether you're standing in a line or spending time with your kids. Stop wasting paper and save trees, The Tic-Tac-Toe Game Application is changed from a pencil-paper game to a computer game where 2 players sit at a computer and play the same game. Because of the simplicity of Tic Tac Toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence. This is a single-player strategy game on the Windows platform. The player will progress through phases which require precise manipulation of the environment, though the

game Encourages creativity and daring via branching pathways. This Application is thus developed for the use of common people including the children above the age of 5.

1.3 Intended Audience and Document Overview

The Tic-Tac-Toe Android Application's Software Requirements Specification (SRS) is intended for different types of readers, including developers, testers, and the client. For Developers, This document provides a detailed description of the functional and non-functional requirements of the Tic-Tac-Toe Android Application.

- **Developers** can use this document as a guide to design, develop, and test Our Application. Testers can use this document to ensure that the Tic-Tac-Toe Android Application meets the specified requirements. They can also use this document as a guide to design and develop test cases.

- **The Client** can use this document to understand the features and functionality of the Tic-Tac-Toe Android Application.

- **Stakeholders** are people or organizations with an interest in the Tic-Tac-Toe Android Application's success, such as investors, customers, or end-users. The SRS document can be useful for stakeholders as it provides a clear understanding of the project scope, objectives, and timeline. By reviewing the document, stakeholders can ensure that the project aligns with their expectations and goals.

- **Professors or Academic Advisors** can use the SRS document to evaluate the quality of the project and provide feedback to the students. The document provides a comprehensive description of Our Application's functional and non-functional requirements, which can help professors assess the students' ability to analyze and design software systems.

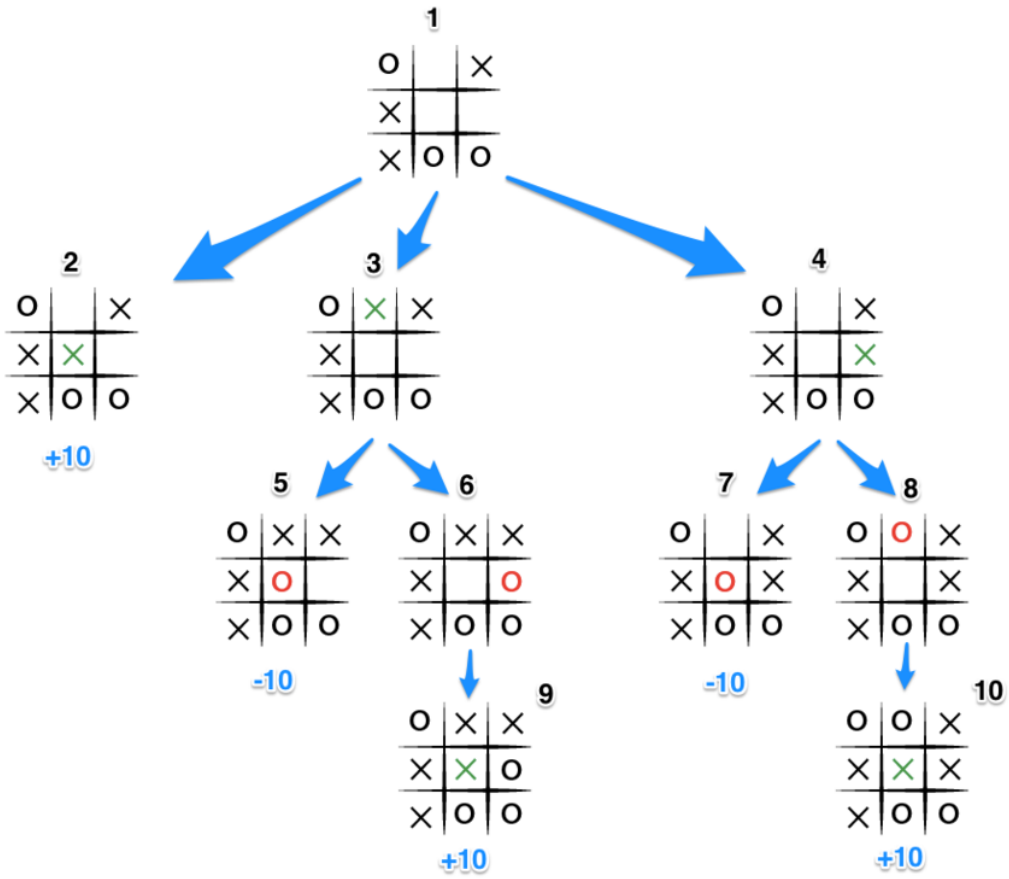
This document also provides a clear understanding of the project scope, timeline, and deliverables. The SRS is organized into several sections, including an overview, functional requirements, non-functional requirements, assumptions and dependencies, and appendices.

1.4 Definitions, Acronyms and Abbreviations

- **MiniMax**

  The MiniMax algorithm is a recursive algorithm used in decision-making and game theory. It delivers an optimal move for the player, considering that the competitor is also playing optimally. This algorithm is widely used for game playing in Artificial Intelligence, such as chess, tic-tac-toe, and myriad double players games. In this algorithm, two players play the game; one is called ‘MAX’, and the other is ‘MIN.’ The goal of players is to minimize the opponent’s benefit and maximize self-benefit. The MiniMax algorithm conducts a depth-first search to explore the complete game tree and then proceeds down to the leaf node of the tree, then backtracks the tree using recursive calls.

The key to the Minimax algorithm is a back and forth between the two players, where the player whose "turn it is" desires to pick the move with the maximum score. In turn, the scores for each of the available moves are determined by the opposing player deciding which of its available moves has the minimum score. And the scores for the opposing players moves are again determined by the turn-taking player trying to maximize its score and so on all the way down the move tree to an end state.



1.5 Document Conventions

This document follows the IEEE formatting requirements. Use Arial font size 11, or 12 throughout the document for text. Use italics for comments. Document text should be single spaced and maintain the 1” margins. For Section and Subsection titles we follow Arial font size 14 in bold format.

It is important to note that the specific formatting and naming conventions used in an SRS document may vary depending on the organization, industry, or project requirements. Therefore, it is important to follow any specific standards or conventions established for the project to ensure clear communication and understanding among all stakeholders.

1.6 References and Acknowledgments

The References which were very useful for the completion of this project are:

- Tic-Tac-Toe game in Android - GFG

  https://www.geeksforgeeks.org/how-to-build-a-tic-tac-toe-game-in-android/
- Tic-Tac-Toe Tutorials - CodwWithHarry

  https://www.codewithharry.com/videos/android-tutorials-in-hindi-5/
- Tic-Tac-Toe in Android - Practical Coding

  https://www.youtube.com/watch?v=Fa5egLurW5U
- Min-Max Algorithm in Game Theory - GFG

  https://www.geeksforgeeks.org/finding-optimal-move-in-tic-tac-toe-using-minimax-algorithm-in-game-theory/
- Tic-Tac-Toe with Min-Max - Tutorial

  https://www.simplifiedcoding.net/tic-tac-toe-android-app-tutorial/
- Min-Max Search Algorithm

  https://www.youtube.com/watch?v=l-hh51ncgDI

2 Overall Description

2.1 Product Overview

The Tic Tac Toe android application provides the user with interface to play the game in two modes – User vs User and User vs Computer. First the user has to select the mode in which he/she wants to play. The player makes the moves and the score will be updated based on the moves and the winner will be displayed at last. The player can either play again or quit the game.

2.2 Product Functionality

The major functionalities of Our Application include:

- Selecting the mode
- Making the moves
- Update the score
- Display the result

2.3 Design and Implementation Constraints

- The hardware device needs to be compatible with the android version to deploy the Tic Tac Toe application.
- Proper calculation of the score and validating the winner.
- The user must be able to navigate between the modes easily.
- This game can be played only by 2 players on a single system.
- Once a game is started, the user can only play the game or quit the game. He cannot open a new game.
- If the game is a win or a lose, the server should immediately display the winner, that is the server should be fast enough to display the winner without allowing the other player to make a move.

2.4 Assumptions and Dependencies

The Assumptions and Dependencies involved in our project are:

We follow Functional Point Analysis and COCOMO

2.4.1 Functional Point Analysis

Measurement Parameter	Count		Weighing Factor			
			Simple	Average	Complex	
1.Number of External Inputs (EI)	9		7	10	15	90
2.Number of External Outputs (EO)	1		5	7	10	7
3.Number of External Inquiries (EQ)	0		3	4	6	0
4.Number of Internal Files (ILF)	0		4	5	7	0
5.Number of External Interfaces (EIF)	0		3	4	6	0
Count-total						97

Functional Point = Count-total * [0.65+0.01*$\sum(f_i)$]

= Count-total * CAF

F = 14 * scale

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor(CAF). Our scale is 3 (Average).

Here, scale = 3

Therefore, F = 14 * 3 = 42

CAF = 0.65 + (0.01 * F)

= 0.65 + (0.01 * 42)

CAF = 1.07

Functional Point = 97 * 1.07

= 103.79

FP = 103.79

2.4.2 COCOMO Model Estimation

To use COCOMO for estimating the cost of developing our App, we need to consider the following parameters:

- **Size of the project:** The size of the project is measured in terms of lines of code (LOC) that need to be developed. For our Tic-Tac-Toe Android Application, we can estimate the size to be around **2,000 LOC**.

- **Complexity of the project:** The complexity of the project depends on factors such as the number of features, the degree of customization, and the level of integration required. For our Application, we can assume that the complexity is Less.
- **Team size:** The team size depends on the size and complexity of the project. For our project, the team size is **2 people**.

Our Project is identified as **Organic** as the requirements are well-understood and the team is reasonably small. The Effort is calculated as

E = a*(KDLOC)^b PM
Effort = 2.4 (2) ^1.05
= 4.97 PM

Tdev = c*(Effort)^d Months
Tdev = 2.5(4.97) ^ 0.38
= 4.597 Months

Thus the Effort and Time for development is calculated by using the Constructive Cost Estimation Model.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

User Interface (UI) design refers to the design of the visual elements of an application or website, including layout, color scheme, typography, and interactive elements. Effective UI design is critical to creating a positive user experience and achieving the goals of Our Application or website. A well-designed UI is critical to providing a positive user experience. When users find it easy to navigate through a website or an application, they are more likely to use it regularly and recommend it to others. An intuitive and user-friendly interface can also reduce the learning curve for new users.

UI design is an important aspect of brand identity. A consistent and visually appealing UI design helps establish a brand's image and enhances its reputation. A well-designed UI can also help differentiate a brand from its competitors. A UI design that is aligned with the project's functionality is important.

It is important that the UI is designed to meet the needs and goals of the users. A well-designed UI helps users understand the features and functionalities of the project and enables them to use it effectively. It should consider accessibility for people with disabilities, such as color blindness, low vision, or limited mobility.

A good UI design takes into account the needs of all users, ensuring that everyone can use the project. A well-designed UI can increase the conversion rates of a website or application. A clear and concise UI design can help users complete tasks easily and quickly, leading to a higher conversion rate.

Some key considerations for UI design for a Tic-Tac-Toe Android Application:

- Simple and clear layout: The layout should be simple and easy to understand, with clear sections for the game board, player information, and game status.

- Use appropriate colors: The colors used for the game board and player information should be easy on the eyes and not too distracting. Using different colors to distinguish between X and O is a good idea.

- Responsive and intuitive controls: The controls for the game board should be responsive to user input, and easy to use. For example, tapping on an empty cell should place the appropriate X or O mark.

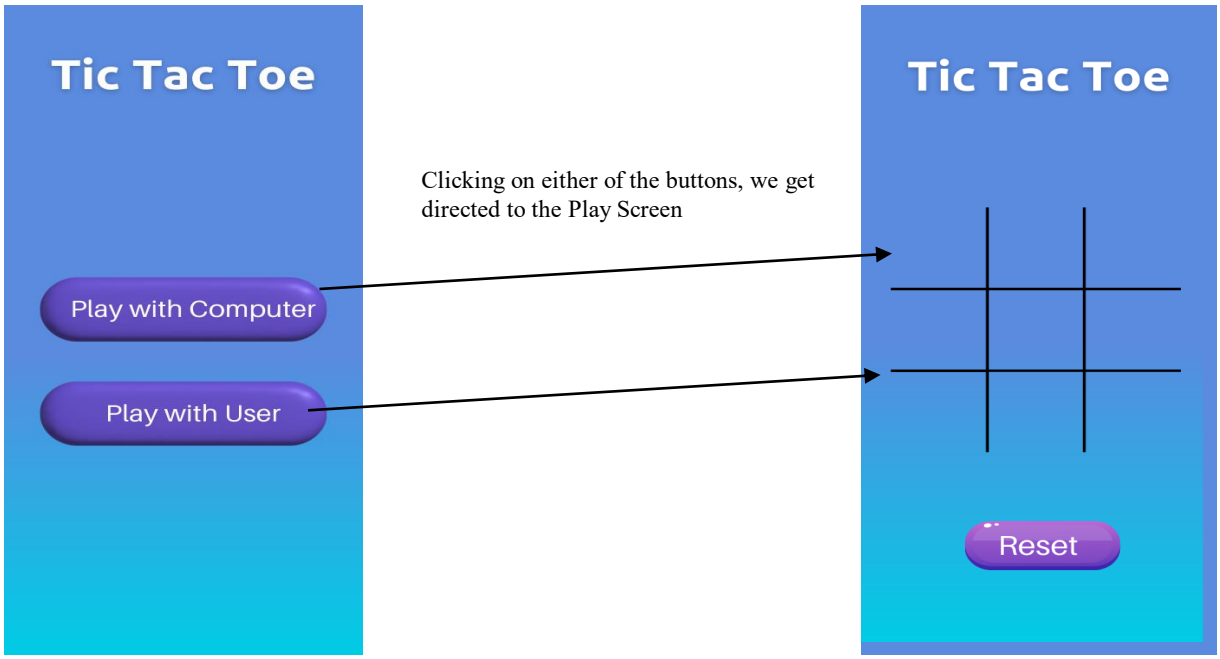
- Feedback: Our Application should provide feedback to users when they make a move, and when the game ends. This could include audio feedback or visual feedback such as flashing cells on the board.

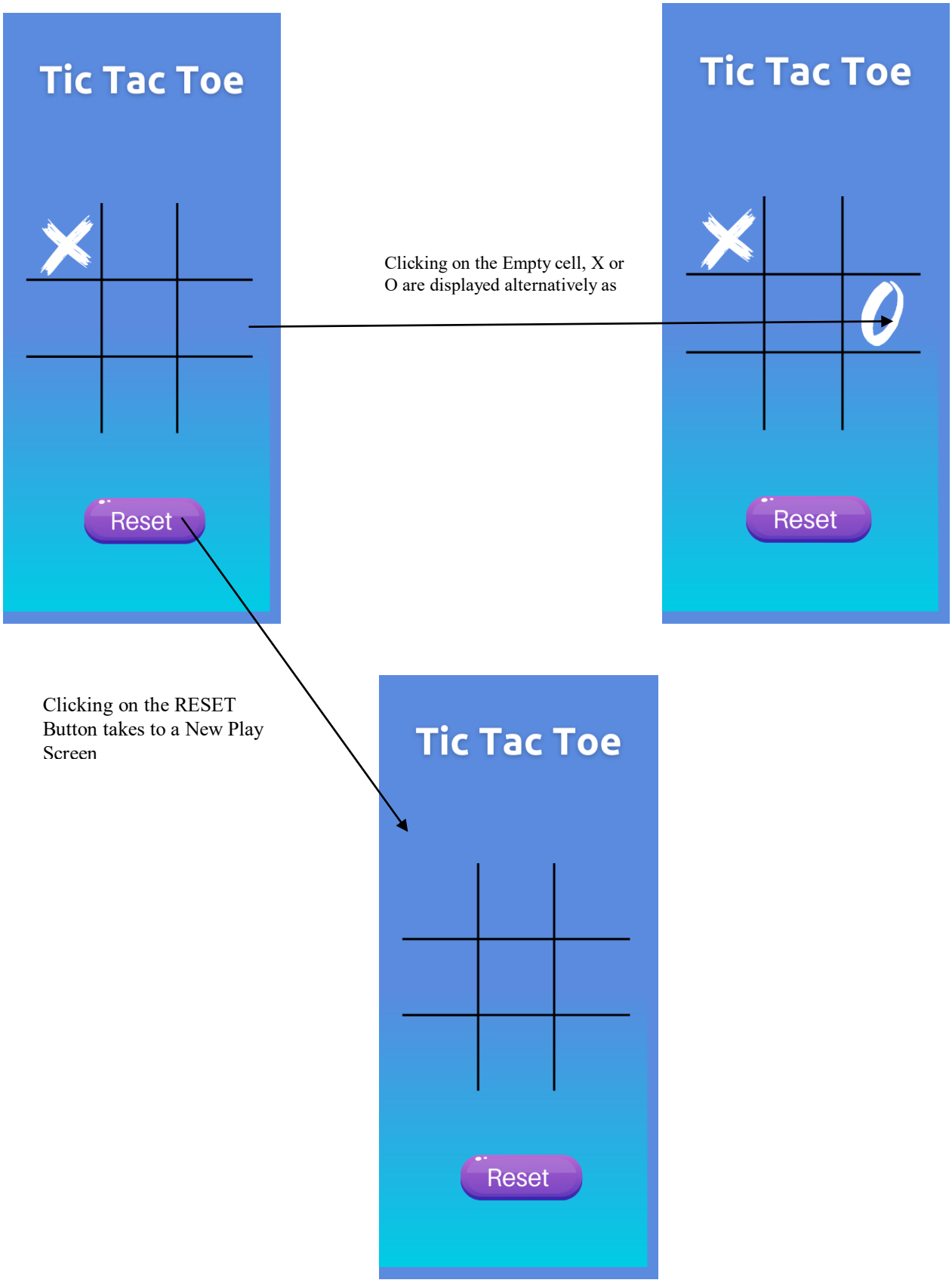
- Clear win/loss messaging: Our Application should provide clear messaging to users when they win, lose, or draw a game. For example, displaying a message that says "Player X Wins!" or "It's a Draw!" can be helpful.

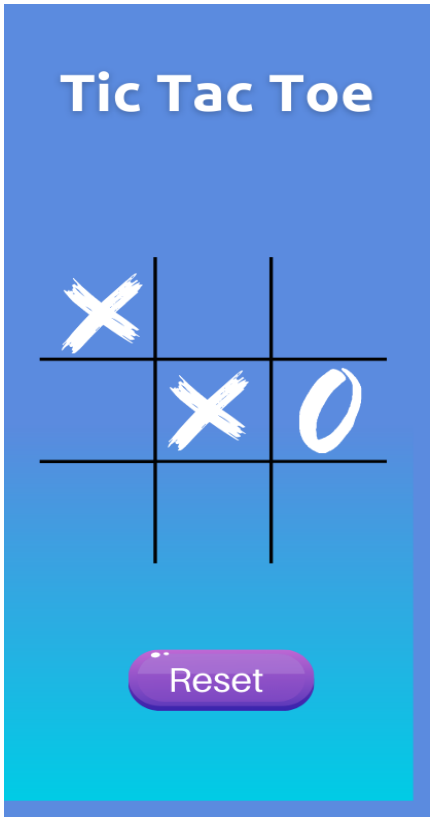
- Use of animations: The use of animations can make Our Application more engaging and fun to use. For example, an animation that shows the X or O mark being placed on the board can be a nice touch.

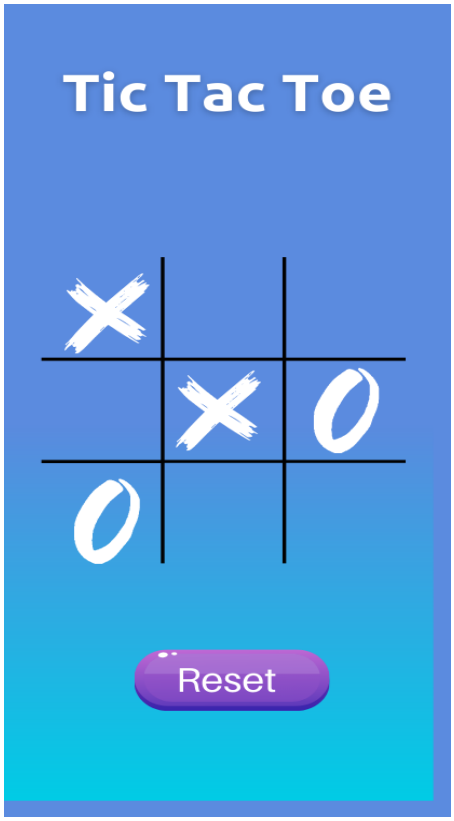
Overall, a good UI design for a Tic-Tac-Toe Android application should be simple, intuitive, and engaging. It should provide feedback to users, use appropriate colors, and be easy to navigate.

The Initial Layout:

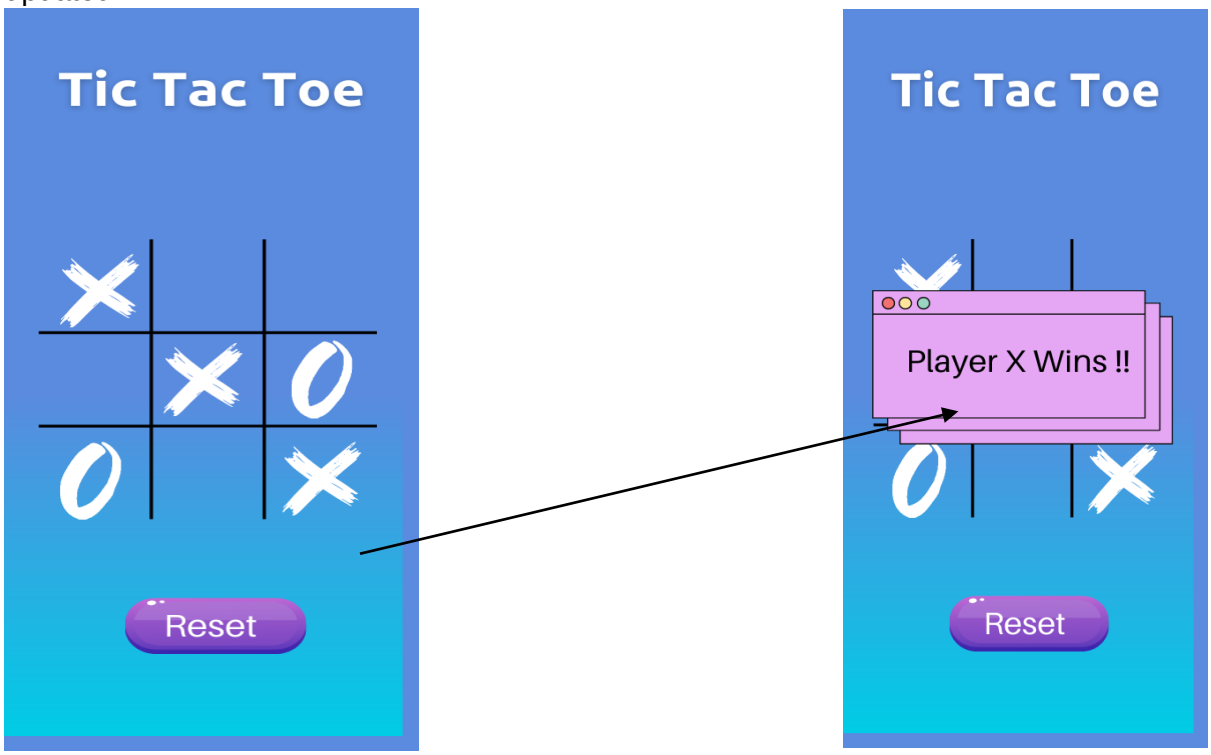



Tic Tac Toe
Reset
Clicking on the Empty cell, X or O are displayed alternatively as
Tic Tac Toe
Reset
Clicking on the RESET Button takes to a New Play Screen
Tic Tac Toe
Reset





When Winning Condition is satisfied, The Players are alerted with the Winner and the Score is updated.



3.1.2 Hardware Interfaces

Hardware interfaces refer to physical connections and protocols that enable data transfer between different hardware components, such as a motherboard, CPU, RAM, hard drive, graphics card, and other peripherals. Examples of hardware interfaces include USB, HDMI, Ethernet, SATA, PCIe, and Wi-Fi. These interfaces provide a standard way for hardware components to communicate with each other and for users to interact with the system.

The hardware interface needed is an android device which could support Our Application and provide a good gaming experience for the users. A typical mobile phone with Android OS 11 and greater is ideal for deployment.

3.1.3 Software Interfaces

Software interfaces, on the other hand, refer to the communication protocols and programming interfaces that enable different software components to interact with each other. Examples of software interfaces include APIs (Application Programming Interfaces), libraries, protocols, and file formats. These interfaces enable software components to share data, communicate with other software applications, and perform different tasks.

An interface game board containing blocks for x and o's for a player to play the game is provided.

3.2 Functional Requirements

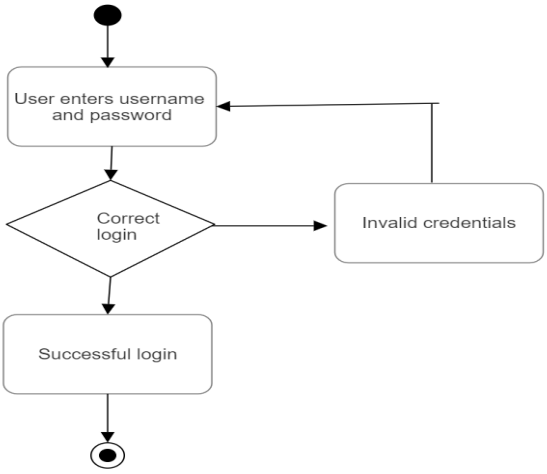
Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. Functional requirements refer to the specific actions and features that a software system or application must be able to perform in order to meet the needs of its users. These requirements typically describe the functionality of the system and the tasks it should be able to perform.

- **Basic Game Interface:** Our Application should have a user-friendly interface that allows players to easily interact with the game. This includes a playing board that displays the game's current state and allows players to make their moves.
- **Two-player mode:** Our Application should support a two-player mode where two users can take turns playing the game against each other.
- **Single-player mode:** Our Application should include a single-player mode where the user can play against an AI opponent.
- **Difficulty levels:** In single-player mode, Our Application should include multiple difficulty levels for the AI opponent, such as easy, medium, and hard.
- **Score tracking:** Our Application should keep track of the score for each game played in both two-player and single-player modes.
- **Undo move feature:** Our Application should allow users to undo their last move, either by shaking the device or through an undo button on the screen.

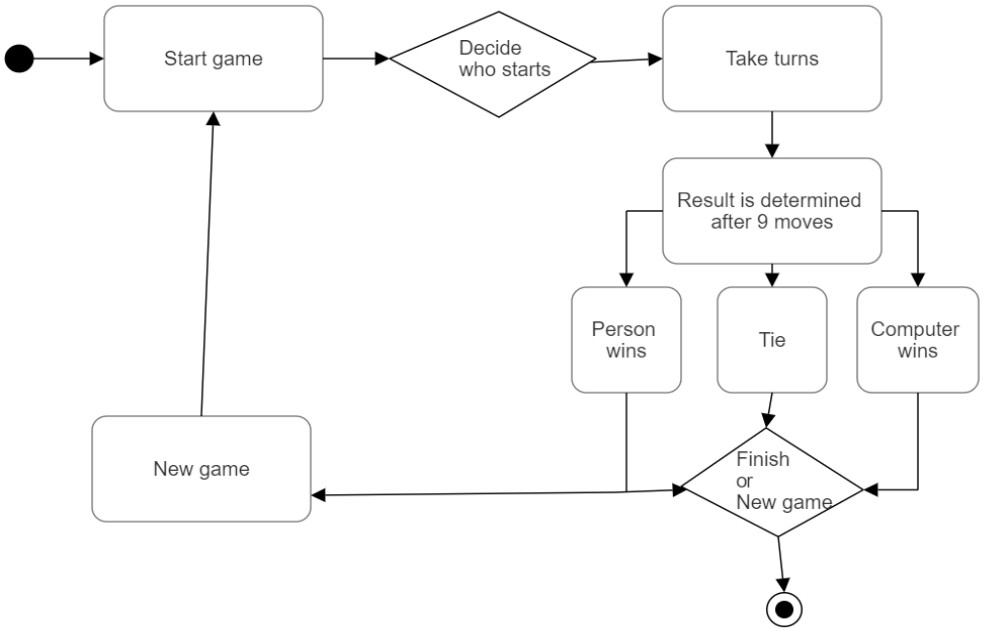
- **New game option:** Our Application should allow users to start a new game at any time, either during a game or at the end of a game.

3.2.1 Activity Diagram

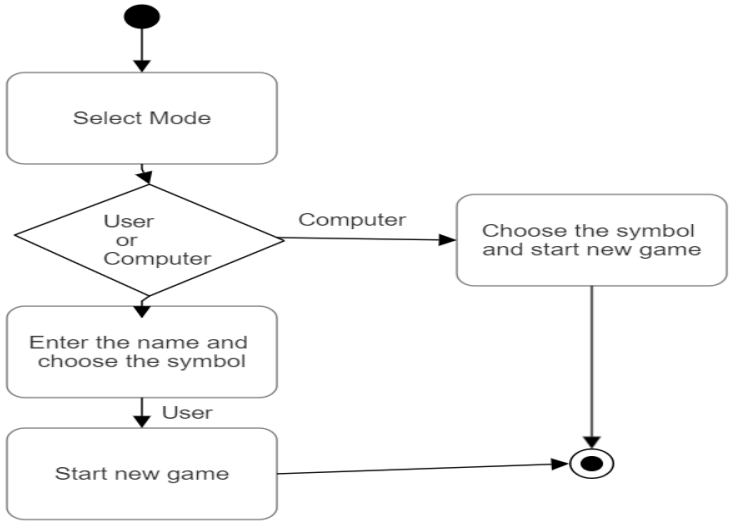
- **Login**



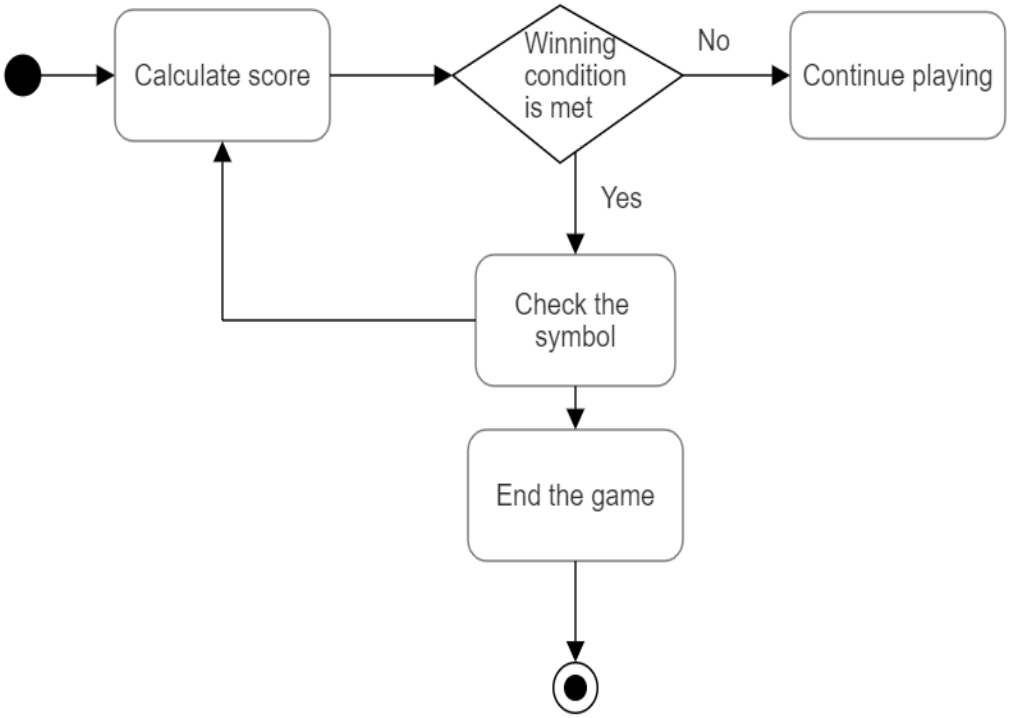
- **Start Game**



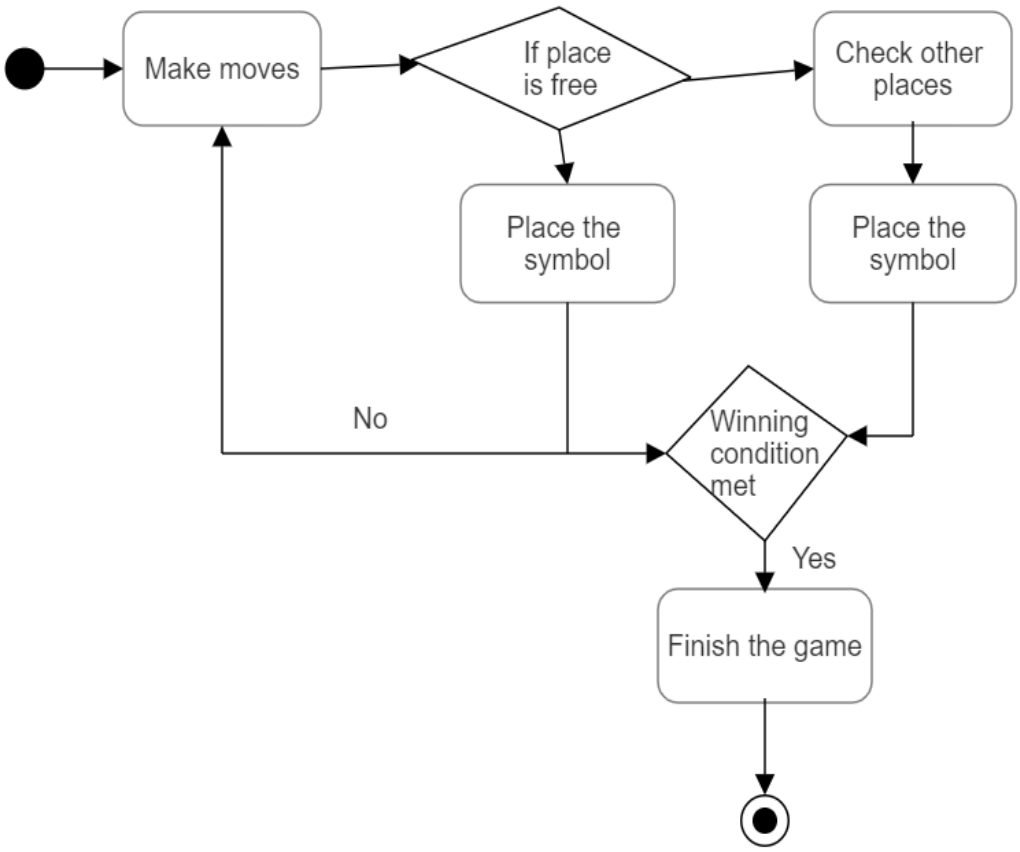
- **Select Mode**



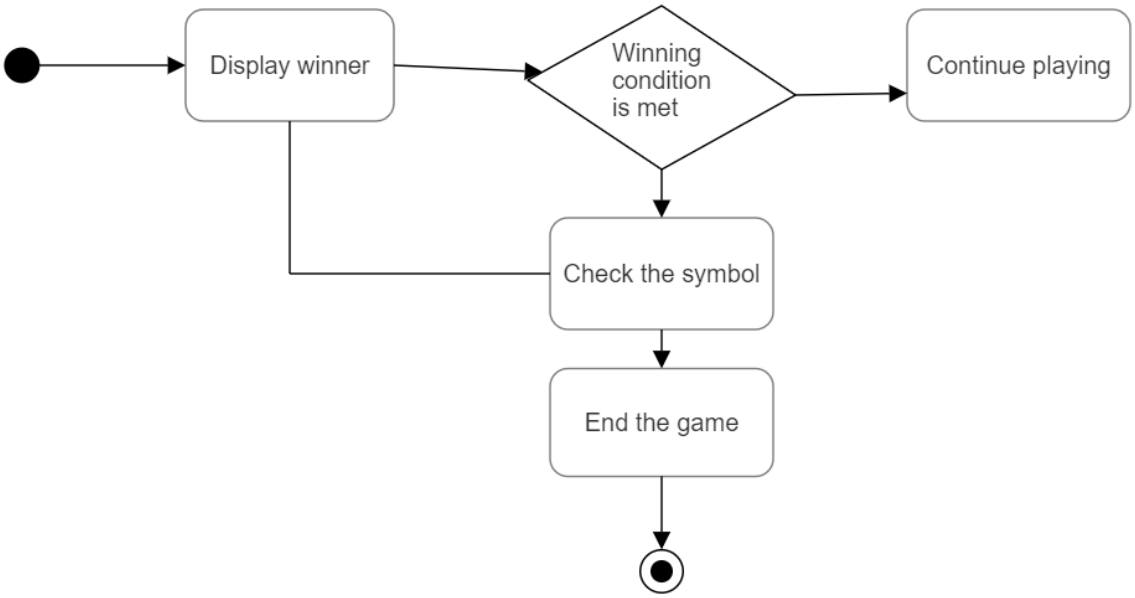
- **Calculate Score**



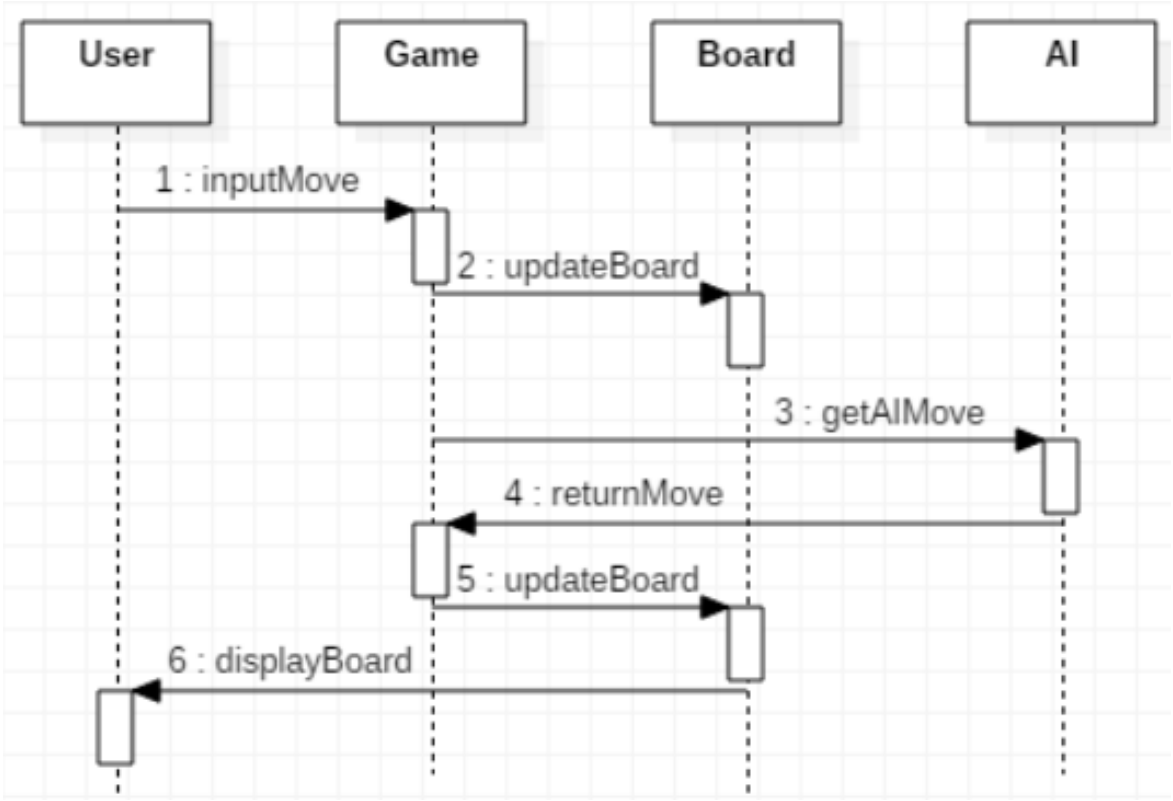
- **Make Move**



- **Display Winner**

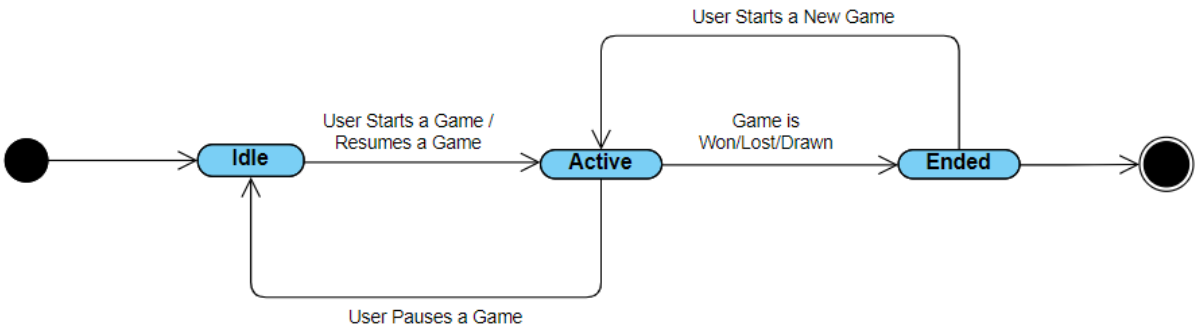


3.2.2 Sequence Diagram

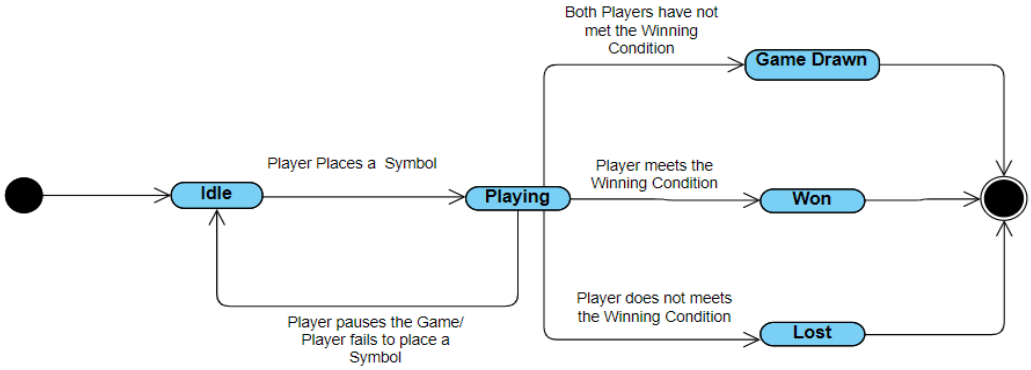


3.2.3 State Transition Diagram

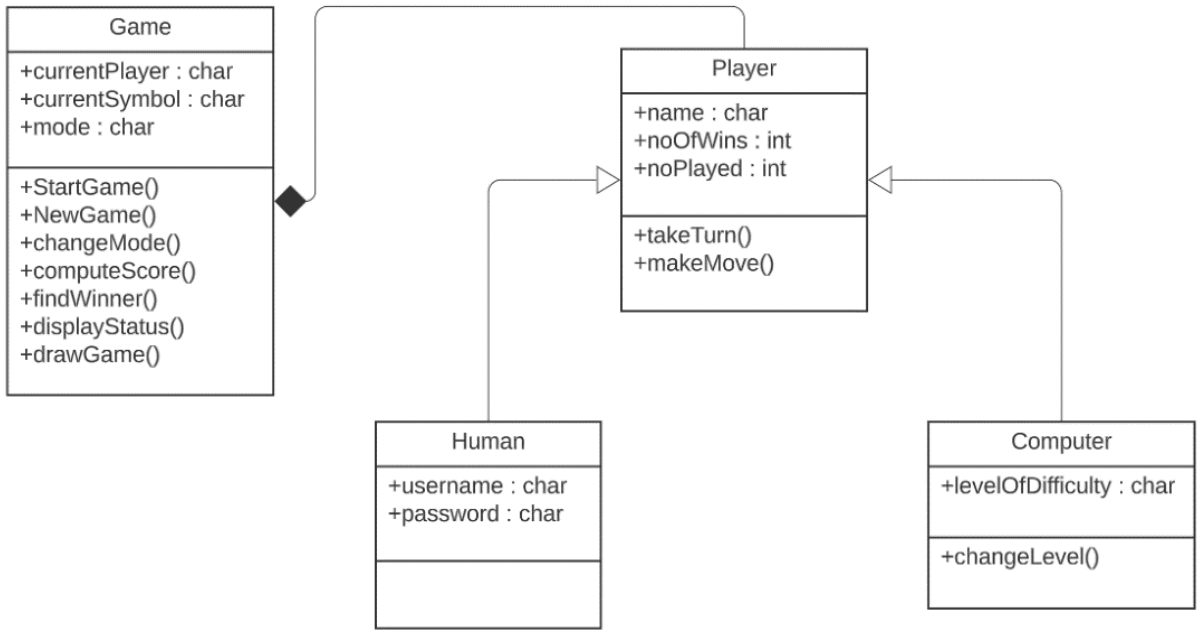
- **Game**



- **Player**



3.2.4 Class Diagram



3.2.5 Data Flow Diagram

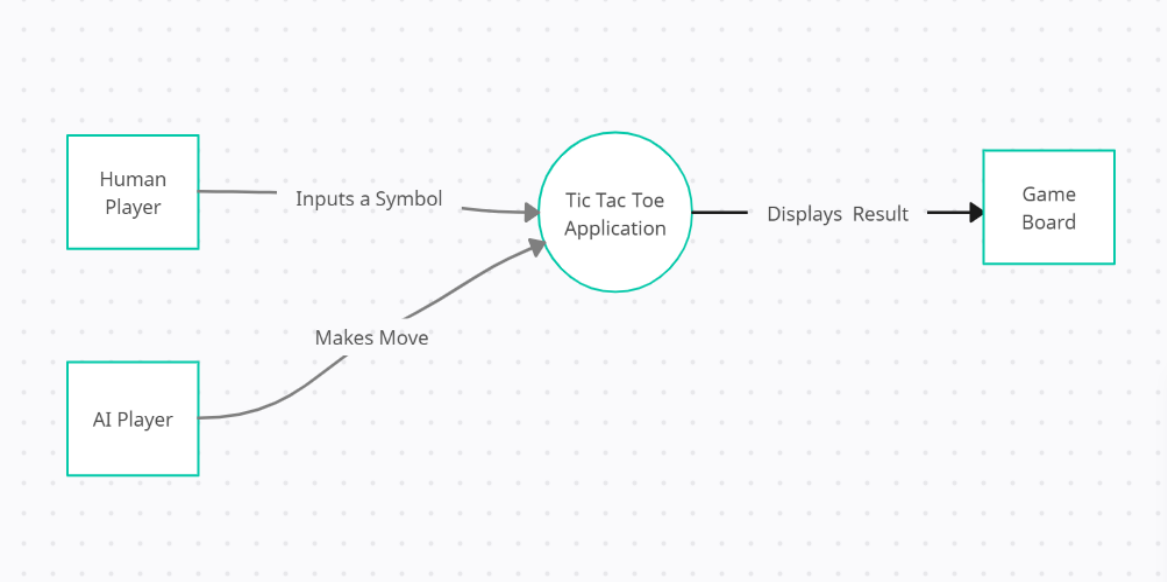
A Data Flow Diagram (DFD) is a graphical representation of a system or process, showing the flow of data and the processes that transform that data. In the case of a Tic-Tac-Toe Android application, a DFD can be used to illustrate how data flows between various components of the app.

Data Flow Diagrams (DFDs) are important because they provide a clear and visual representation of how data moves through a system or process. This makes them a useful tool for software development, system analysis, and project management. DFDs help to break down complex systems into smaller, more manageable parts. This allows developers, analysts, and stakeholders to better understand how the system works and how data flows through it. They use simple and standardized symbols to represent processes and data flows. This makes them an effective communication tool for stakeholders, as they can quickly and easily understand the system or process being depicted in the diagram. They can help to identify issues with a system or process. By visualizing the flow of data, it becomes easier to see where data may be getting lost, where bottlenecks may occur, or where errors might be introduced. DFDs can be used to plan and design systems or processes before they are built.

This can help to identify potential issues early in the development process, saving time and resources down the line. They provide a clear and concise way to document a system or process. They can be used to create manuals, training materials, or other forms of documentation that can be easily understood by stakeholders.

Level-0 DFD for Tic-Tac-Toe Android Application:

A Level-0 DFD provides a high-level overview of the system or process and shows the major processes and the flow of data between them. In the case of a Tic-Tac-Toe Android application, the Level-0 DFD might look something like the following image. This diagram shows that the user interacts with the Tic-Tac-Toe app, and the app interacts with the system. The arrows indicate the flow of data between these components. In this case, the data is the game state, which is passed between the user, the app, and the system.

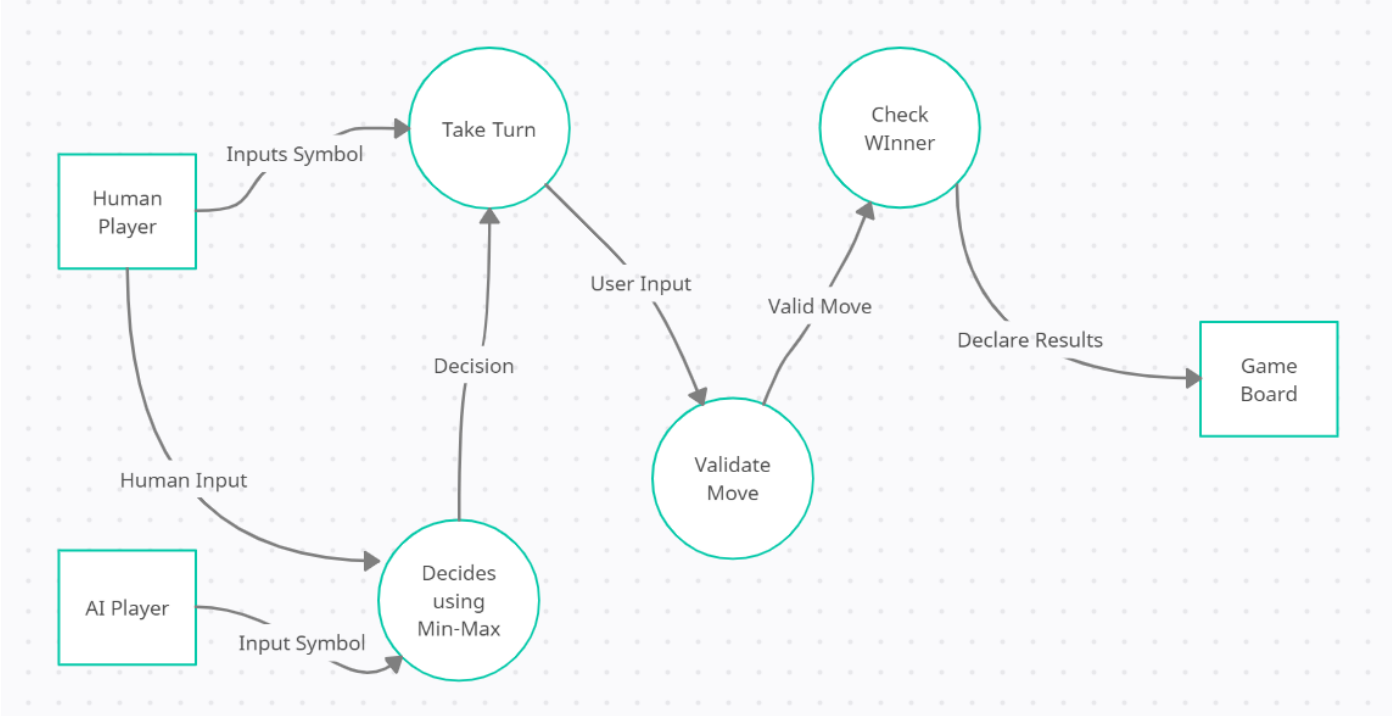


Level – 0 DFD

Level-1 DFD for Tic-Tac-Toe Android Application:

A Level-1 DFD provides more detail than the Level-0 DFD and shows the sub-processes involved in a major process. In the case of a Tic-Tac-Toe Android application, the Level-1 DFD might look something like the following image. This diagram shows that the Tic-Tac-Toe app consists of three sub-processes: Display, Game Logic, and Input Handling. The Display process is responsible for showing the game board to the user, the Game Logic process is responsible for determining the winner and updating the game state, and the Input Handling process is responsible for handling user input. These sub-processes communicate with each other and with the user and the system to ensure that the game is played correctly.

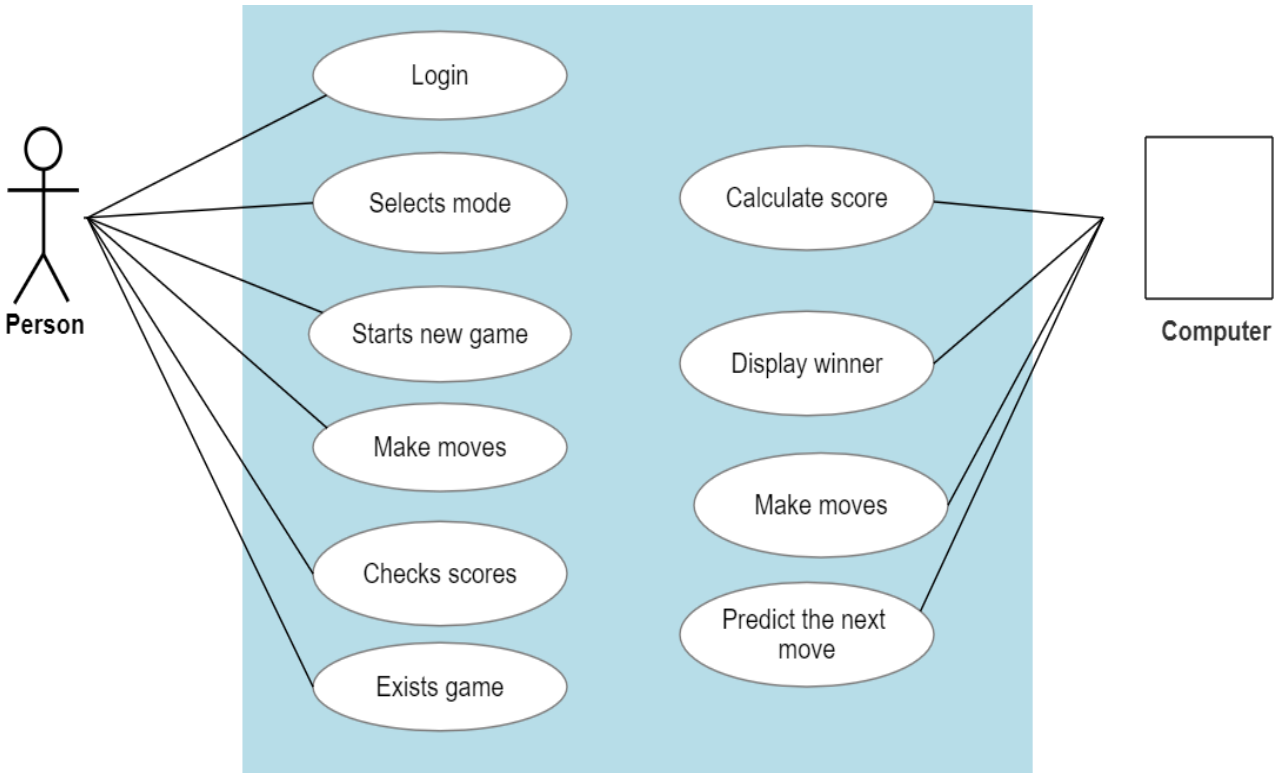
Overall, these data flow diagrams provide a high-level understanding of how data flows between the various components of a Tic-Tac-Toe Android application. They can be used to identify potential issues and to optimize the system for better performance. This is an important tool for understanding, communicating, and improving systems and processes. They help to identify issues, improve planning and design, and provide clear documentation for stakeholders.



Level - 1 DFD

3.3 Use Case Model

- **Actors:**
  - Person
  - Computer
- **Major use cases:**
  - Login
  - Selects mode
  - Starts new game
  - Make moves
  - Checks score
  - Exists game
  - Calculate score
  - Display winner
  - Predicts Next move



3.3.1 Modes of the Game

- **Human Player vs Human Player**

  In this game, two players take turns marking Xs and Os on the grid until one player wins by placing three of their marks in a row, column, or diagonal, or the game ends in a tie if all the cells are filled without any player winning. The Tic-Tac-Toe Android Application project is an implementation of this game on the Android platform. The project aims to provide an enjoyable and user-friendly gaming experience to users, complete with a simple and intuitive user interface, customizable game options, and an AI opponent that adapts to the user's level of play.

  Both these players play on the client's side. The server program evaluates the game and declares the winners or if the game is a draw. The player who succeeded in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

- **Human Player vs Computer Player**

  Human vs AI Tic Tac Toe is a classic example of a game that can be solved using the Minimax Algorithm. The Minimax Algorithm is a decision-making algorithm that is commonly used in game theory to determine the best move for a player, given the current state of the game. It works by considering all possible moves that a player can make, and then recursively evaluates the best possible move for each player, assuming that the other player will make their best move in response.

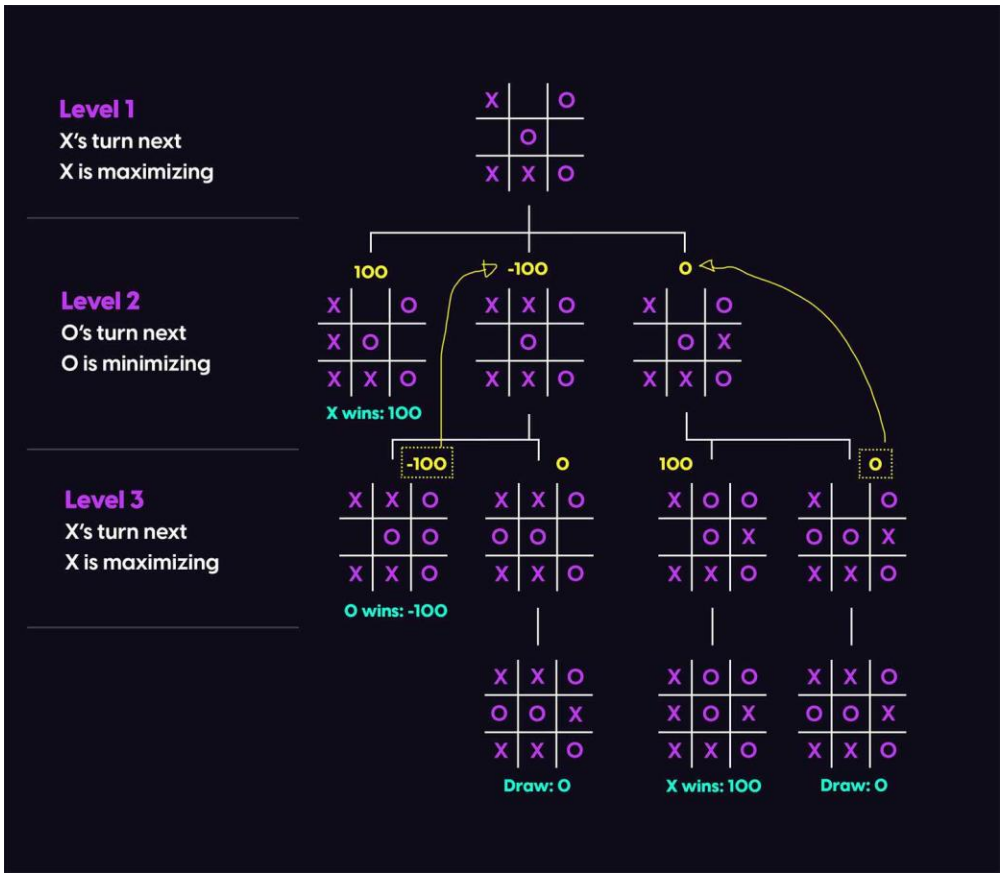
In the case of Tic Tac Toe, the Minimax Algorithm can be used to determine the optimal moves for both the human player and the AI player. Here's how it works:

- First, the AI player determines the current state of the game, i.e., the positions of all X's and O's on the board.
- Next, the AI player generates all possible moves it can make, i.e., all possible positions where it can place an X on the board.
- For each of these possible moves, the AI player assumes that the human player will make their best possible move in response, i.e., the move that will lead to the worst possible outcome for the AI player.
- The AI player then recursively evaluates the best possible move for the human player, assuming that the AI player will make its best possible move in response.
- This process continues until the game reaches a terminal state, i.e., either the AI player wins, the human player wins, or the game ends in a tie.
- Once the algorithm has evaluated all possible moves, the AI player chooses the move that leads to the best possible outcome for itself.
- The AI player then makes its move, and the process repeats until the game reaches a terminal state.

By using the Minimax Algorithm, the AI player can always make the optimal move, assuming that the human player also plays optimally. Of course, in practice, it is impossible to predict the human player's moves with certainty, so the AI player can only make an educated guess about what the human player's best move might be. However, by using the Minimax Algorithm, the AI player can still make a very good guess, and in most cases, it will be able to beat or tie the human player.

To check whether or not the current move is better than the best move we take the help of minimax() function which will consider all the possible ways the game can go and returns the best value for that move, assuming the opponent also plays optimally.

In order to make the game unbeatable, it was necessary to create an algorithm that could calculate all the possible moves available for the computer player and use some metric to determine the best possible move. After extensive research it became clear that the Minimax algorithm was right for the job.



The Minimax algorithm is a well-known algorithm used in game theory and decision-making that is commonly applied in games like Tic-Tac-Toe. In Tic Tac Toe, the algorithm determines the best possible move for a given player by looking ahead to all possible future moves and evaluating the outcome of each move.

In its simplest form, the Minimax algorithm can be applied to a game of Tic-Tac-Toe played between two human players. The algorithm evaluates each possible move and assigns it a score based on the likelihood of winning the game. If a move results in a win, it is assigned a score of 1. If it results in a loss, it is assigned a score of -1. If it results in a tie, it is assigned a score of 0. The algorithm then chooses the move with the highest score.

When playing against an AI, the difficulty level of the game can be adjusted by changing the depth of the search. In other words, the algorithm can be made to look ahead a certain number of moves to evaluate the outcome of each possible move. The higher the depth of the search, the more difficult the game becomes for the human player, as the AI is able to consider more moves and make more informed decisions.

For example, at a low difficulty level, the AI might only look ahead one or two moves, making it easier for the human player to find a winning strategy. At a higher difficulty level, the AI might look ahead three or four moves, making it much more difficult for the human

player to find a winning strategy. In extreme cases, the AI might be able to consider all possible moves and find the optimal strategy for winning the game every time.

Overall, the Minimax algorithm is a powerful tool for designing AI opponents in games like Tic-Tac-Toe, as it allows the AI to make informed decisions and provide a challenging and engaging gameplay experience for the human player.

4 Other Non-functional Requirements

4.1 Performance Requirements

Performance requirements for a Tic-Tac-Toe Android Application can vary depending on the specific needs of Our Application. Some possible requirements and their rationale are:

- Response Time: Our Application should respond to user actions within 0.5 seconds for a smooth and responsive user experience.
- Game Load Time: The game should load within 2 seconds to ensure that users do not lose interest in the game while waiting for it to load.
- CPU and Memory Usage: Our Application should not consume excessive CPU and memory resources as it can impact the overall performance of the device and potentially slow down other applications running simultaneously.
- Sound effects: Our Application should include sound effects for different events during the game, such as a player making a move, winning, or losing.
- Settings options: Our Application should provide various settings options, such as sound volume, difficulty level, and background theme.
- Difficulty Level: We can provide different difficulty levels like Easy, Medium, Hard and allow the user to switch between levels.
- User Interface: The user interface should be intuitive, easy to navigate, and visually appealing to enhance the user experience.
- Payment Options: We can further add payment options to play multiple levels of a game
- Score Calculation: The Total score of the user in different levels can be displayed to the user.
- Play History: By using Database, we could store the total games played by the user in different modules and the score in the game.

For real-time multiplayer games, the timing relationship is critical. Our Application should be designed to ensure that there is no noticeable delay between a user's action and the game's response. Our Application should also ensure that the game state is synchronized between all players in real-time to avoid inconsistencies. Additionally, for real-time systems, it is crucial to ensure that the performance requirements are met consistently to avoid any potential lag or delay that can impact the gameplay experience.

4.2 Safety and Security Requirements

Based on the Tic-Tac-Toe Android Application, some of the safety and security requirements that should be considered are:

- Our Tic-Tac-Toe application must prevent any potential harm or injury that could arise from physical actions associated with the use of the app, such as accidentally dropping or throwing the device.
- Our Tic-Tac-Toe app must be designed in a way that minimizes the risk of unauthorized access to the user's personal data, such as name, email, or other sensitive information.
- Our app must prevent any unauthorized use or modification of the game data, such as score, game progress, or other game-related information.
- Our app must have secure user authentication mechanisms to ensure that only authorized users can access the app's features and functionalities.
- Our Application must encrypt all data transmissions over the internet to prevent interception or eavesdropping of sensitive information, such as login credentials or user data.
- Our app must prevent any malicious attacks, such as hacking or malware, by implementing robust security measures, such as firewalls, intrusion detection, and prevention systems.

To meet these requirements, the app should implement various safeguards, such as encryption, user authentication, and secure data storage, and should be regularly tested and audited for potential security vulnerabilities. Additionally, any relevant external policies and regulations should be followed to ensure compliance with safety and security standards.

4.3 Software Quality Attributes

The following are the additional quality characteristics for the Tic-Tac-Toe Android Application that will be important to both the customers and the developers:

4.3.1 Usability

The usability of Our Application refers to the ease with which users can interact with Our Application and perform the intended tasks. To ensure high usability, Our Application must meet the following requirements:

- Our application must have a user-friendly interface that is easy to navigate and understand, with clear instructions and appropriate feedback.
- Our application must be intuitive and easy to use, even for users who are not familiar with the game of Tic-Tac-Toe.

- Our application must be responsive and fast, with minimal lag or delay in displaying the game board and accepting user inputs.

- Our application must provide visual cues and prompts to guide users through the game, such as highlighting the winning move or indicating the next turn.

- Our application must allow users to customize the game settings, such as the difficulty level, sound effects, and language.

4.3.2 Reliability

The reliability of Our Application refers to the ability of Our Application to perform its intended function without failure or errors. To ensure high reliability, Our Application must meet the following requirements:

- Our Application must have error handling mechanisms to detect and handle any runtime errors or exceptions.

- Our Application must be tested thoroughly to ensure that it functions correctly under all expected conditions and user scenarios.

- Our Application must be scalable and able to handle a large number of users and game instances without performance degradation.

4.3.3 Performance

Our Application should be responsive and perform well, with minimal lag or delay during gameplay.

4.3.4 Compatibility

Our Application should be compatible with a range of Android devices and screen sizes, and support multiple orientations (portrait and landscape).

4.3.5 Security

Our Application should be secure, protecting user data and preventing unauthorized access or malicious activity.

4.3.6 Maintenance

Our Application should be easy to maintain, with clear documentation and modular code that allows for easy updates and bug fixes. To achieve high usability and reliability, Our Application should be designed with a user-centered approach, with regular usability testing and feedback from users. Additionally, Our Application should be built using best practices for software design, such as modular architecture, clean code, and appropriate documentation. Regular testing and

maintenance should be performed to ensure that Our Application remains reliable and functional over time.

5 Other Requirements

5.1 Non-Functional Requirements

- Usability: Our Application should be easy to use and navigate, with intuitive controls and clear feedback for user actions.
- Performance: Our Application should be responsive and perform well, with minimal lag or delay during gameplay.
- Compatibility: Our Application should be compatible with a range of Android devices and screen sizes, and support multiple orientations (portrait and landscape).
- Reliability: Our Application should be stable and reliable, with minimal crashes or errors during gameplay.
- Security: Our Application should be secure, protecting user data and preventing unauthorized access or malicious activity.
- Internationalization: Our Application should support multiple languages and cultural preferences, such as by allowing users to select their preferred language and displaying appropriate date and time formats.
- Maintenance: Our Application should be easy to maintain, with clear documentation and modular code that allows for easy updates and bug fixes.
- Sound effects: Our Application should include sound effects for different events during the game, such as a player making a move, winning, or losing.
- Settings options: Our Application should provide various settings options, such as sound volume, difficulty level, and background theme.

5.2 Prioritized Use Cases

5.2.1 High Priority:

- Play the game in two-player mode.
- Play the game in single-player mode against an AI opponent.
- Display the current state of the game board and allow players to make moves.
- Keep track of the score for each game played.

5.2.2 Medium Priority:

- Provide multiple difficulty levels for the AI opponent in single-player mode.
- Allow users to undo their last move during gameplay.
- Allow users to start a new game at any time.

5.2.3 Low Priority:

- Include sound effects for events during gameplay.
- Provide various settings options, such as sound volume, difficulty level, and background theme.
- Support multiple languages and cultural preferences.

Appendix A – Data Dictionary

Source Code

MainActivity.kt:

```
package com.example.tictactoekotlin

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.GridLayout
import android.widget.ImageView
import android.widget.TextView
import androidx.core.content.ContextCompat
import org.w3c.dom.Text
import kotlin.random.Random

class MainActivity : AppCompatActivity() {

    //Creating a 2D Array of ImageViews
    private val boardCells = Array(3) { arrayOfNulls<ImageView>(3) }

    var board = Board()
    val text_view_result = findViewById<TextView>(R.id.text_view_result)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //calling the function to load our tic tac toe board
        loadBoard()

        //val button_restart = null
        val button_restart = findViewById<Button>(R.id.button_restart)
        button_restart.setOnClickListener {
            board = Board()
            text_view_result.text = ""
            mapBoardTOUi()
        }

    }

    private fun mapBoardTOUi() {
        for (i in board.board.indices) {
            for (j in board.board.indices) {
                when (board.board[i][j]) {
                    Board.PLAYER -> {
                        boardCells[i][j]?.setImageResource(R.drawable.circle)
```

```
                        boardCells[i][j]?.isEnabled = false
                    }
                    Board.COMPUTER -> {
                        boardCells[i][j]?.setImageResource(R.drawable.cross)
                        boardCells[i][j]?.isEnabled = false
                    }
                    else -> {
                        boardCells[i][j]?.setImageResource(0)
                        boardCells[i][j]?.isEnabled = true
                    }
                }
            }
        }
    }

    /*
    * This function is generating the tic tac toe board
    * */
    private fun loadBoard() {

        for (i in boardCells.indices) {
            for (j in boardCells.indices) {
                boardCells[i][j] = ImageView(this)
                boardCells[i][j]?.layoutParams = GridLayout.LayoutParams().apply {
                    rowSpec = GridLayout.spec(i)
                    columnSpec = GridLayout.spec(j)
                    width = 250
                    height = 230
                    bottomMargin = 5
                    topMargin = 5
                    leftMargin = 5
                    rightMargin = 5
                }
                boardCells[i][j]?.setBackgroundColor(
                    ContextCompat.getColor(
                        this,
                        R.color.colorPrimary
                    )
                )
                boardCells[i][j]?.setOnClickListener(CellClickListener(i, j))
                //val layout_board = null
                val layout_board = findViewById<GridLayout>(R.id.layout_board)
                layout_board.addView(boardCells[i][j])
            }
        }
    }

    inner class CellClickListener(
        private val i: Int,
        private val j: Int
    ) : View.OnClickListener {

        override fun onClick(p0: View?) {
            if (!board.isGameOver) {
                val cell = Cell(i, j)
                board.placeMove(cell, Board.PLAYER)
                board.minimax(0, Board.COMPUTER)
                board.computersMove?.let {
                    board.placeMove(it, Board.COMPUTER)
                }
```

```
                //val cCell = board.availableCells[Random.nextInt(0,
board.availableCells.size)]
                mapBoardTOUi()
            }
            when {
                board.hasComputerWon() -> text_view_result.text = "Computer Won"
                board.hasPlayerWon() -> text_view_result.text = "Player Won"
                board.isGameOver -> text_view_result.text = "Game Draw"
            }
        }

    }

/*private fun Nothing?.addView(imageView: ImageView?) {
    TODO("Not yet implemented")
}*/
}
```

MainActivity.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:background="@color/colorBackground"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/image_view_name"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"
        android:layout_alignParentTop="true"
        android:background="@drawable/app_name"
        android:layout_width="280dp"
        android:layout_height="55dp"/>

    <TextView
        android:layout_marginBottom="25dp"
        android:layout_above="@id/layout_board"
        android:layout_centerHorizontal="true"
        android:textAppearance="@style/TextAppearance.AppCompat.Headline"
        android:textColor="@android:color/white"
        tools:text="Computer Won"
        android:id="@+id/text_view_result"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" app:fontFamily="casual" android:textStyle="bold"
        android:textAllCaps="false" android:textSize="30sp"/>

    <GridLayout
        android:id="@+id/layout_board"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:alignmentMode="alignBounds"
```

```
        android:columnCount="3"
        android:padding="16dp"
        android:rowCount="3"
        android:useDefaultMargins="true" />

    <Button
        android:layout_marginBottom="15dp"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:id="@+id/button_restart"
        android:textAllCaps="false"
        android:text="Restart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RelativeLayout>
```

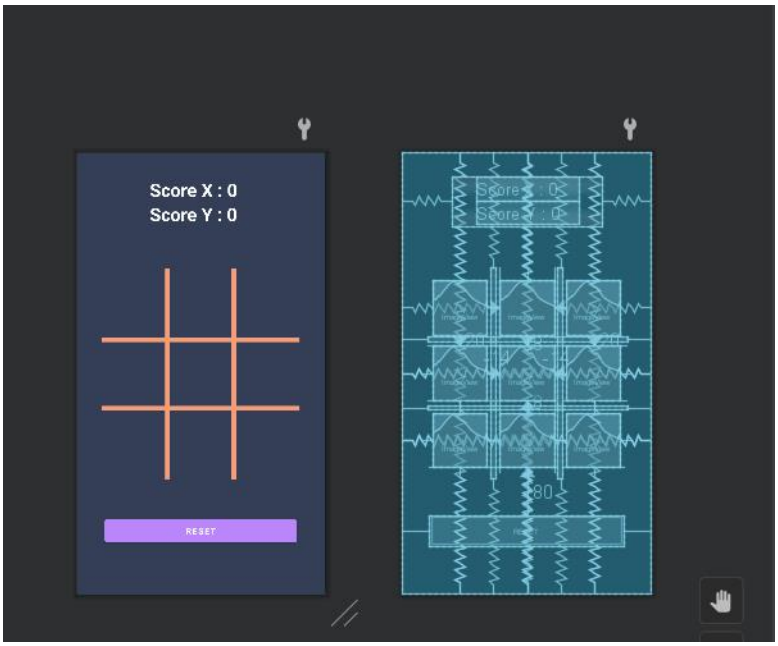
Board.kt

```
package com.example.tictactoekotlin

class Board {
    companion object {
        const val PLAYER = "O"
        const val COMPUTER = "X"
    }
    val board = Array(3) { arrayOfNulls<String>(3)}
    val availableCells : List<Cell>
        get(){
            val cells = mutableListOf<Cell>()
            for(i in board.indices){
                for(j in board.indices){
                    if(board[i][j].isNullOrEmpty()) {
                        cells.add(Cell(i, j))
                    }
                }
            }
            return cells
        }

    val isGameOver : Boolean
        get() = hasComputerWon() || hasPlayerWon() || availableCells.isEmpty()


    fun hasComputerWon() : Boolean{
        if(board[0][0] == board[1][1] &&
                board[0][0] == board[2][2] &&
                board[0][0] == COMPUTER ||
                board[0][2] == board[1][1] &&
                board[0][2] == board[2][0] &&
                board[0][2] == COMPUTER){
            return true
        }
        for(i in board.indices){
            if(board[i][0] == board[i][1] &&
                    board[i][0] == board[i][2] &&
                    board[i][0] == COMPUTER ||
                board[0][i] == board[1][i] &&
                board[0][i] == board[2][i] &&
```

```
                board[0][i] == COMPUTER){
                return true
            }
        }
        return false
    }

    fun hasPlayerWon() : Boolean{
        if(board[0][0] == board[1][1] &&
            board[0][0] == board[2][2] &&
            board[0][0] == PLAYER ||
            board[0][2] == board[1][1] &&
            board[0][2] == board[2][0] &&
            board[0][2] == PLAYER){
            return true
        }
        for(i in board.indices){
            if(board[i][0] == board[i][1] &&
                board[i][0] == board[i][2] &&
                board[i][0] == PLAYER ||
                board[0][i] == board[1][i] &&
                board[0][i] == board[2][i] &&
                board[0][i] == PLAYER){
                return true
            }
        }
        return false
    }

    var computersMove : Cell? = null
    fun minimax(depth: Int, player: String) : Int{
        if(hasComputerWon()) return +1
        if(hasPlayerWon()) return -1
        if(availableCells.isEmpty()) return 0

        var min = Integer.MAX_VALUE
        var max = Integer.MIN_VALUE

        for(i in availableCells.indices){
            val cell = availableCells[i]
            if(player == COMPUTER){
                placeMove(cell, COMPUTER)
                val currentScore = minimax(depth+1, PLAYER)
                max = Math.max(currentScore,max)
                if(currentScore >= 0){
                    if(depth == 0) computersMove = cell
                }
                if(currentScore == 1){
                    board[cell.i][cell.j] == ""
                    break
                }
                if(i==availableCells.size - 1 && max < 0){
                    if(depth == 0) computersMove = cell
                }
            }
            else if(player == PLAYER){
                placeMove(cell, PLAYER)
                val currentScore = minimax(depth+1, COMPUTER)
                min = Math.min(currentScore, min)
                if(min == -1){
```

```
                    board[cell.i][cell.j] = ""
                    break
                }
            }
            board[cell.i][cell.j] = ""
        }
        return if(player == COMPUTER) max else min
    }

fun placeMove(cell: Cell, player: String){
    board[cell.i][cell.j] = player
}
}
```

Appendix B – Screenshots



FigureA.1: Initial Layout

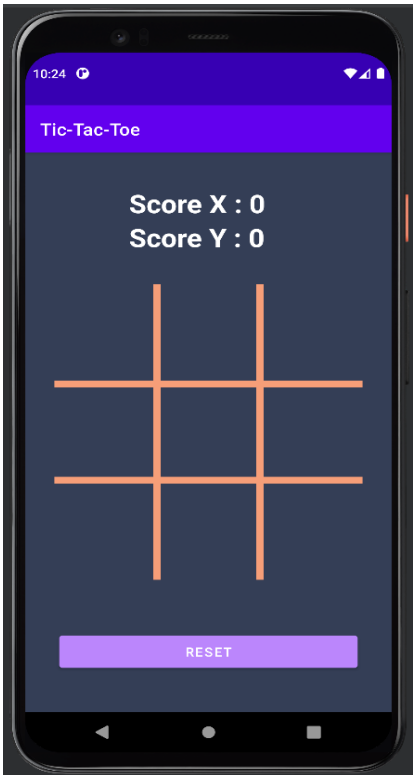


Figure A.2: Initial Layout

Figure A.3: Players Making Turns

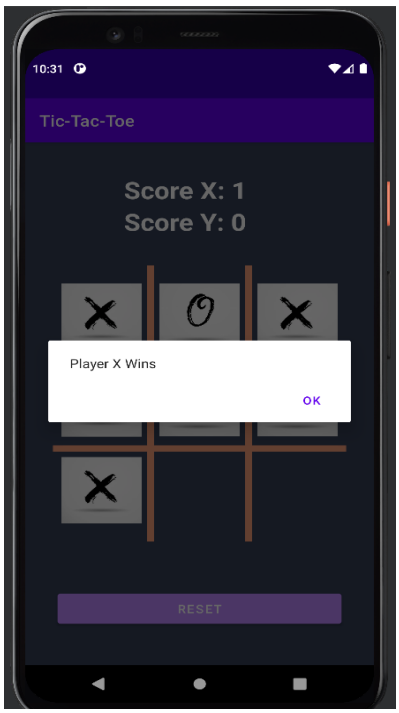


Figure A.4: Player X Wins

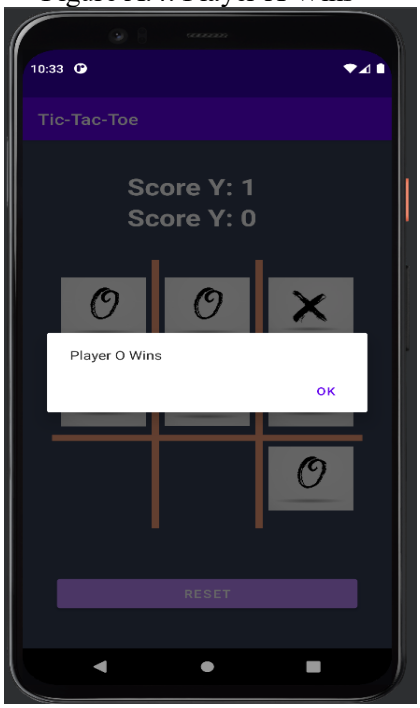


Figure A.5: Player O Wins