Software Requirements Specification (SRS)
Tic-Tac-Math

Team: 6
Authors: Jake Correnti, Scott Landry, Mildred Kumah, Al-Amin Muhammad, Paschal Ojatabu
Customer: Elementary School or Parent
Instructor: Dr. James Daly

Table of Contents

1 Introduction

This Software Requirements Specification (SRS) document provides a detailed description of the requirements for Tic-Tac-Math. The purpose of this document is to present a comprehensive outline of the functional and non-functional requirements, intended to guide the development team and stakeholders throughout the development process of the software application.

1.1 Purpose

The purpose of this document is to describe the software system that is being developed. Here, we aim to clearly define all the functional and non-functional requirements for Tic-Tac-Math. It stands as both a list of requirements and a guideline for development for all parties involved in the development of Tic-Tac-Math.

1.2 Scope

Tic-Tac-Math is an educational math-based Tic-Tac-Toe game designed specifically for students at a 4th grade math level. The game is 2 player and the player must answer a math problem correctly to claim a space. All problems are randomly generated and will focus on multiplication. The main benefits of the game include reinforcing classroom learning, enhancing students' problem-solving skills, and making learning a fun and interactive experience. This project aims to give teachers a fun option to utilize and give their students an engaging tool for students to learn their multiplication tables more fluently.

1.3 Definitions, acronyms, and abbreviations

- GUI: Graphical User Interface
- NIC: Network Interface Controller
- UI: User Interface
- HTML: HyperText Markup Language
- CSS: Cascade Style Sheet
- DNS: Domain Name System
- HTTP: HyperText Transfer Protocol
- TLS: Transport Layer Security
- TCP/IP: Transmission Control Protocol/Internet Protocol

- Actor: entities that interact with the system
- Use Case: represents the goals that an actor might want
- Composition: when an object "is made of" another object

- Tic-Tac-Toe: classic two-player strategy game in which participants take turns marking a 3x3 grid with their chosen symbol, typically "X" or "O," with the objective of forming a row, column, or diagonal of three matching symbols.
- Tic-Tac-Math: game inspired by tic-tac-toe, but before a player can mark the “X” or “O” symbol on their board, they need to correctly solve a math problem.

1.4 Organization

This first section of the SRS document provides an introduction to Tic-Tac-Math as well as provides an overview of this document. Section 2 details the overall description of the project by outlining the functionality of Tic-Tac-Math. Its purpose is to describe in detail how the game will look and behave. Section 3 focuses on specific requirements for the game and provides a list of all the game needs. Section 4 contains various models and diagrams that represent the game’s structure and behavior. These diagrams include use case diagrams, sequence diagrams, and class diagrams. Section 5 goes into detail on the prototype that was developed and also details some sample scenarios a user might encounter. Section 6 contains references to sources used for the game and surrounding system. Finally, section 7 contains the point of contact.

2 Overall Description

The information in this section of the document will provide a general description of the software. It will provide a Product Perspective, which puts the product in relation to other related products or projects. It then goes over Product Functions, which summarizes the primary functions of Tic-Tac-Math. Subsequently, this section discusses User Characteristics, which describes the characteristics of the eventual user of the product that will affect the specific requirements. Additionally, General Constraints will be specified, which includes general descriptions of any other item that will limit our options for designing the game. The last two aspects of this section include Assumptions and Dependencies as well as Apportioning of Requirements. Assumptions and Dependencies goes into detail regarding each factor that affects the requirements that are stated in this document. Finally, Apportioning of Requirements provides information on possible features that will be released in future versions of the software.

2.1 Product Perspective

Tic-Tac-Math is a website where two students can battle against each other in tic-tac-toe. However, they have to answer a math problem correctly in order for their symbol to be placed on the board. The students can play the game anywhere they can get access to a computer with an internet connection. Tic-Tac-Math provides a fun way for students to practice their math skills and be competitive with one another. Although it is a way for students to practice their math, it is not a formal way for instructors to test students on their skills.

The system's GUI, in the form of a website, should adhere to industry standards so the UI looks and behaves the same across multiple platforms and browsers. The system requires the underlying host hardware to have the ability to connect to the Internet through a NIC. The system also requires the host to have access to a keyboard and mouse. These may be integrated like those of a laptop, or externally connected via bluetooth or a cable. Additionally, the system requires the host machine to have a graphical display in order to display the game to the user. In regards to software constraints, the system requires that the host machine be running an operating system with a desktop environment installed. Within this operating system, it is required that an internet browser be installed. The internet browser must be able to render HTML and CSS while also running JavaScript code in the background. Finally, the system must also take advantage of protocols such as DNS, HTTP, TLS, and TCP/IP in order to interface with the Internet.

2.2 Product Functions

The initial major function the software will have is to present the users with a landing page when they first navigate to the website. This will provide the users with a description of Tic-Tac-Math and a way to start a new game.

Once the user starts a new game, the software displays the tic-tac-toe board to the users. This is the board the two players will be interacting with when they play against one another. Above the game board the software will display which player, X or O, will make the next move. Tic-Tac-Math will allow the player who's up next to select an open square on the board. Once the open square is selected, the software will generate a random math problem who's multiplicand and multiplier are randomly generated numbers between zero and ten. The software will provide an input box for the player to submit their answer to the question. If the user's provided answer is correct, the software will place the respective symbol on the board and the next player can make their move. However, if the player submitted the wrong answer, the software will generate new math problems until a correct answer is given. After every move, the game will determine if there is a winner or a tie. When the game is over, the software will provide the users the chance to reset the board and have a rematch or go back to the landing page.

Learning your multiplication tables can be an incredibly tedious process, often involving frequent repetition of the same problems. However, with Tic-Tac-Math, we aim to help educate students on their multiplication tables with a fun, interactive Tic-Tac-Toe style game that allows them to compete with their classmates in the process.

2.3 User Characteristics

Given that Tic-Tac-Math is a website, it is expected that the users of the software understand how to navigate the internet and interact with websites within an Internet Browser. Due to the nature of the game and how user input is recorded, it is expected that the user will know how to use a keyboard and mouse. Additionally, it is expected that users know how to play the game of Tic-Tac-Toe. Tic-Tac-Math also expects the user will have at least a third grade level education, preferably fourth grade or above. Specifically, the education level expects basic reading and multiplication skills in order to successfully use the software.

2.4 Constraints

Based on the possible constraints from the IEEE SRS document, there are no additional constraints with regards to the software. These constraints included regulatory policies, hardware limitations, interfaces to other applications, parallel operation, audit functions, control functions, higher-order language requirements, signal handshake protocols, criticality of the application, and safety and security considerations.

In regards to regulatory policies, there are no governmental restrictions that the developer would have to take into consideration when writing the software. There are also no additional hardware constraints the developer needs to take into consideration when implementing the system, such as a low amount of memory or disk space. Interfacing with other applications is not within the scope of Tic-Tac-Math and should not be a concern for the developers. Tic-Tac-Math, in its current state, is a synchronous piece of software and does not require parallel operation in order to perform as intended. Due to the nature of the game and the fact that there is effectively “no” backend with zero data persistence, there is nothing to log for a future audit. Any failures in the system would have nowhere to be logged, therefore, there is no central location where data could be accessed and therefore audited. Additionally, the developer does not need to be concerned with signal handshake protocols such as TLS, HTTP, or TCP/IP as that is all handled by the Web Browser. The fundamental structure of the software involves no forms of persistence or interaction with other applications on the host system. Since the software is completely enclosed and hosted via a third party, any issues regarding data a malicious actor might try to gain is futile. There is absolutely no data the actor can access that will prove useful, and additionally, if the actor were to gain access to the system, it would be through the fault of the Web Browser or other third party application that is outside the scope of the developers working on Tic-Tac-Math.

2.5 Assumptions and Dependencies

One dependency of the system is the device it's being run on has access to the Internet via a Web Browser. However, if this is not available then the current document will require changes.

An additional dependency of the system is the hardware running the game is using a mouse and keyboard. If a mouse and keyboard are not being used, the current document will also have to be adjusted to account for that. If there is no way for the user to provide input to the software, then the software is unusable.

Also, the system depends on the user's host operating system having a desktop environment installed. Without a desktop environment, the software is not usable.

2.6 Apportioning of Requirements

Based on negotiations with customers, there were a few requirements that were determined to be beyond the scope of the current project and they may be addressed in figure version/releases. One of the requirements that was discussed was the ability for the two players to be on different machines rather than the same one. This was determined out of scope because the game in its current state still completes the goal of having a fun way to practice your math skills. This addition would require a revamp of the backend, therefore it will be addressed in figure versions/releases.

An additional suggestion that was made by customers was to have a way to store students' math accuracy after each game so the instructors can go through and look at all of the statistics for their students. Although we felt that this was a good idea, it was not a requirement in order to get the main functionality of the software completed. Additionally, this new feature would require similar backend changes to the requirement above, therefore it was decided to address this requirement in a future version or release.

3 Specific Requirements

1. Game should have 4th grade (10 years old) math students who are currently studying basic multiplication tables be the core demographic
2. User Interface must provide the user the ability to start a new game
3. User interface must have a screen displaying the tic-tac-toe board
    a. The user interface must display who's turn it is
4. The user interface must display a simple math problem on the screen (two numbers between 0-10 multiplied together) and the user will be provided an input area to submit their answer to the problem
    a. Players who answer correctly are given the opportunity to make a single move on the tic-tac-toe board
    b. Players who fail to answer a question correctly will be given a new problem
    c. For every question a user answers correctly, the user interface will display their accuracy
5. Once a user has successfully won the game of tic-tac-toe, the user interface will display who the winning user is
6. The user interface must provide a way for the user to do a rematch or reset the board

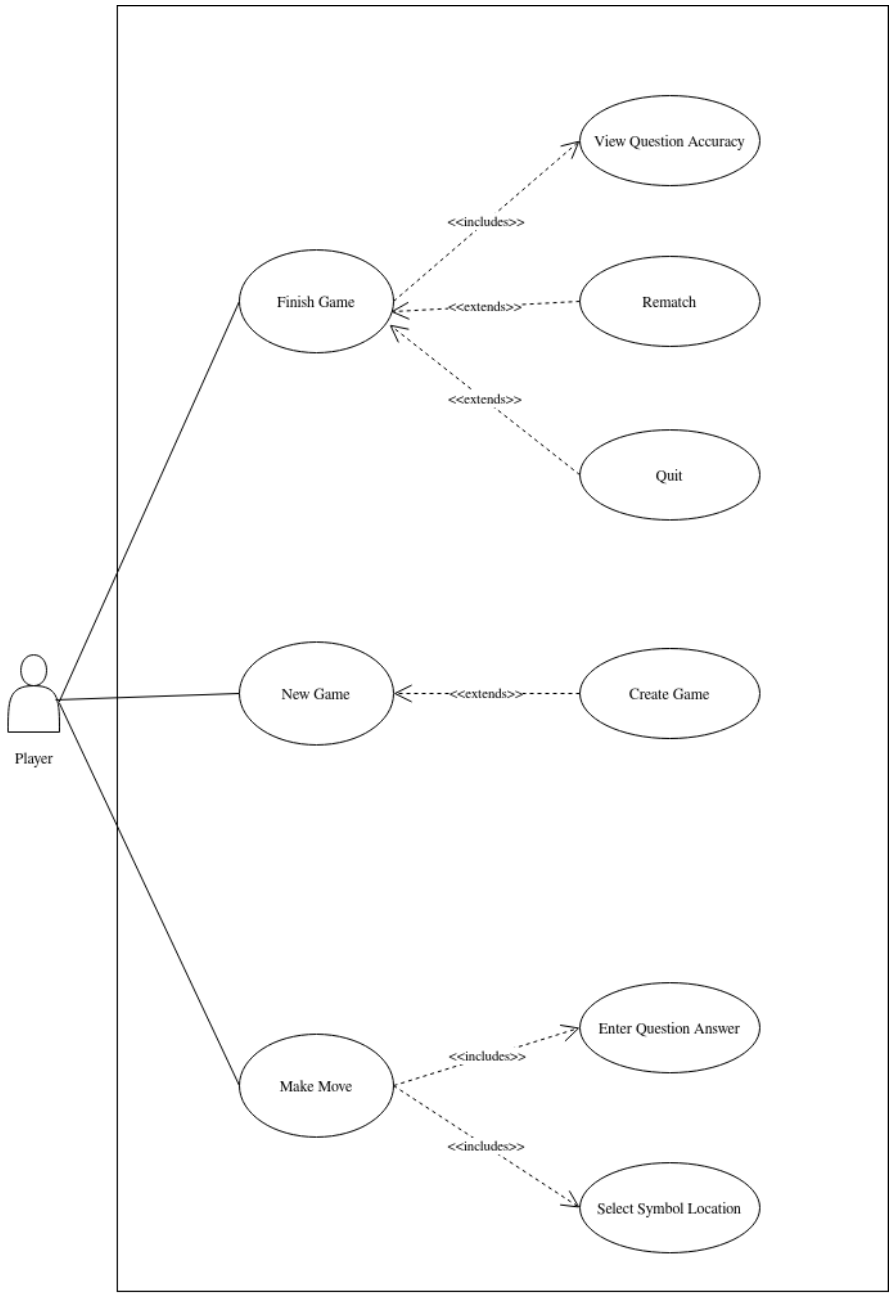
4 Modeling Requirements

Use Case Diagram

Use Case diagrams are a type of behavioral diagram that organizes behaviors of the system. Use cases are user goals, which are high-level services of the system. Use case diagrams are done from the point of view of an external actor.

The notation of a Use Case diagram is fairly straightforward. An actor is represented by a stick figure. A use case is represented by a labeled oval. A solid line is used to connect actors to the use cases that they want to achieve. Around the edge of the system, you can find a labeled box called the system boundary. We use <<includes>> and <<extends>> to signify that use cases are interacting with other use cases. An arrow marked with an <<includes>> means that there is a sub-goal that needs to get accomplished. On the other hand, an arrow marked with an <<extends>> means there is a special case that modifies the goal.

In regards to our specific Use Case diagram displayed below, we have a single actor referred to as Player. We have nine different use cases in the diagram, three primary and six secondary. The three primary use cases are Finish Game, New Game, and Make Move. The six secondary use cases are View Question Accuracy, Rematch, Quit, Create Game, Enter Question Answer, and Select Symbol Location. Surrounding the different use cases is the system boundary, which is labeled “Tic-Tac-Math Game”. Our Use Case diagram also uses <<includes>> and <<extends>> in order to show that use cases are interacting with one another. For example, Create Game is a use case that builds on top of, or extends, the functionality that is provided by New Game. Conversely, Enter Question Answer is functionality that Make Move consists of, or includes.



Use Case Name:	Make Move
Actors:	Player
Description:	The player selects to make a move on the tic-tac-toe board
Type:	Primary and Essential

Includes:	Enter Question Answer, Select Symbol Location
Extends:	None
Cross-refs:	Software requirements
Uses cases:	Enter Question Answer, Select Symbol Location

Use Case Name:	New Game
Actors:	Player
Description:	Player decides to start a new game of Tic-Tac-Math
Type:	Primary and Essential
Includes:	None
Extends:	None
Cross-refs:	None
Use cases:	Create Game

Use Case Name:	Finish Game
Actors:	Player
Description:	Player decides to no longer continue playing Tic-Tac-Math
Type:	Primary and Essential
Includes:	View Question Accuracy
Extends:	None
Cross-refs:	None
Use cases:	View Question Accuracy, Rematch, Quit

Use Case Name:	View Question Accuracy
Actors:	Player

Description:	The player's question accuracy is displayed
Type:	Secondary and Essential
Includes:	None
Extends:	None
Cross-refs:	None
Use cases:	None

Use Case Name:	Rematch
Actors:	Player
Description:	Player decides to continue playing Tic-Tac-Math against their opponent
Type:	Secondary and Essential
Includes:	None
Extends:	Finish Game
Cross-refs:	None
Use cases:	None

Use Case Name:	Quit
Actors:	Player
Description:	Player decides to no longer continue playing Tic-Tac-Math against their opponent
Type:	Secondary and Essential
Includes:	None
Extends:	Finish Game
Cross-refs:	None

Use cases:	None

Use Case Name:	Create Game
Actors:	Player
Description:	A new game is started, resetting all underlying data
Type:	Secondary and Essential
Includes:	None
Extends:	New Game
Cross-refs:	None
Use cases:	None

Use Case Name:	Enter Question Answer
Actors:	Player
Description:	Player enters their answer to the generated math question
Type:	Secondary and Essential
Includes:	None
Extends:	None
Cross-refs:	None
Use cases:	None

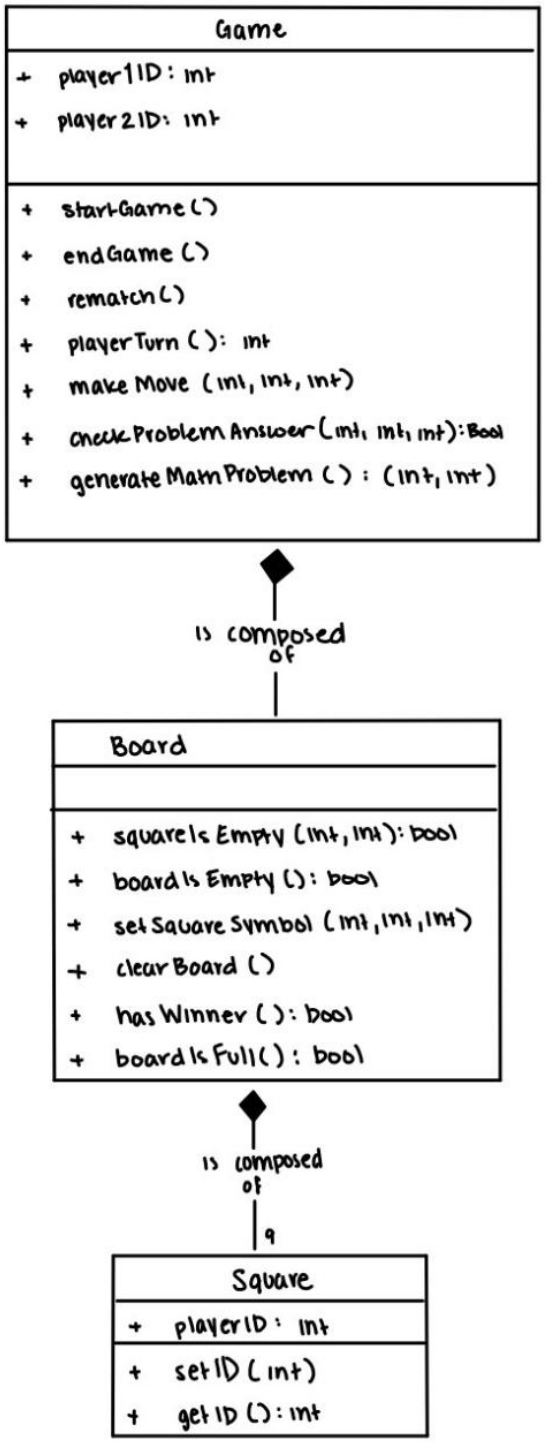
Use Case Name:	Select Symbol Location
Actors:	Player
Description:	Player selects the location on the board where they would like to place their symbol

Type:	Secondary and Essential
Includes:	None
Extends:	None
Cross-refs:	None
Use cases:	None

Class Diagram

Class diagrams are used to model attributes and methods of classes and the relationships between them. Each class is represented by a box with three components: the class name in the top box, attributes in the middle box, and methods in the bottom. It is important to note that attributes and methods are sometimes omitted. For the attributes and methods, the + prefix is used to indicate that it is a public attribute or method, meaning it can be accessed by anyone. It is also important to note that attributes of a class that are objects are represented by composition instead, which is signified by a closed black diamond head.

Our class diagram below begins with the Game class, which is responsible for maintaining the state of the entire game. The game class consists of Player 1's ID, Player 2's ID, and is composed of a Board object. The Board class is responsible for managing the state of the Tic-Tac-Toe board specifically. The Board class is composed of nine Square objects. The Square objects have a fairly simple job, and that is to manage the state of an individual square on the Tic-Tac-Toe board.



Tic-Tac-Math Class Diagram

Class Dictionary:

Game:

- Attributes:
  - player1ID: represents the identifier, “X”, for the first player
  - player2ID: represents the identifier, “O”, for the second player
  - Composed of one board object
- Methods:
  - startGame: starts a new game with a fresh game board
  - endGame: clears the game board and takes the users back to the landing page
  - rematch: clears the board and starts another game with the same two opponents
  - playerTurn: returns the identifier of the player whose turn it is
  - makeMove: given the player identifier and the x and y coordinates of the square, it will start the sequence of generating a new math problem, prompting the user for an answer, and setting the symbol on the board.
  - checkProblemAnswer: given the multiplicand, multiplier, and the user’s answer, it determines if the user’s answer is correct.
  - generateMathProblem: generates a random math problem and returns a pair of integers, the first being the multiplicand and the second being the multiplier. The two numbers are between zero and ten.

Board:

- Attributes:
  - Composed of nine Square objects
- Methods:
  - squareIsEmpty: given the x and y coordinates of the square, returns true if it is empty and false otherwise.
  - boardIsEmpty: returns true if no squares in the board have a symbol, false otherwise.
  - setSquareSymbol: given the x and y coordinates of the square and the player’s identifier, set’s the symbol based on the value of the player identifier.
  - clearBoard: clears all symbols from the board.
  - hasWinner: determines if there is currently a winner on the board based on the board’s current state.
  - boardIsFull: returns true if all squares in the board have a symbol associated with them, return false otherwise.

Square:

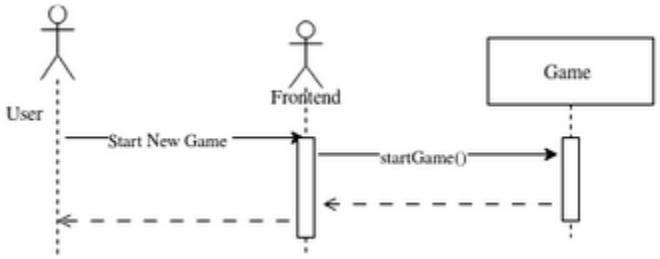
- Attributes:
  - playerID: integer representing the player who selected this square. The identifier is what is used to determine the symbol

- Methods:
    - setID: sets the identifier for the square, which in turn sets the square's symbol
    - getID: gets the identifier for the square, which is effectively getting the symbol for the square

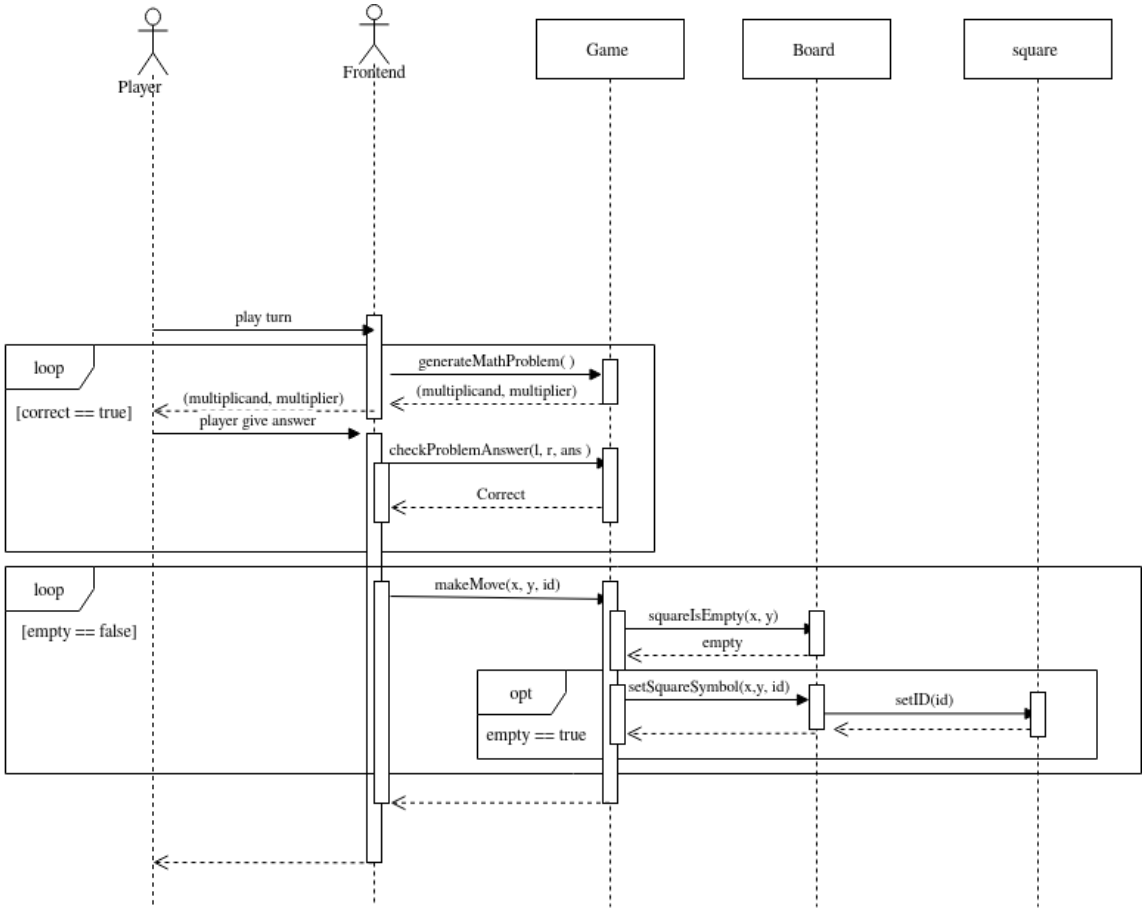
Representative Scenarios

The following scenarios contain sequence diagrams. A sequence diagram is a class and object diagram that describes the structure of the system. Specifically, it models how objects send messages to each other. A lifeline in a sequence diagram consists of two parts: a box representing an object labeled with the type of the object, and a vertical dashed line representing the lifetime of the object. In sequence diagrams, sometimes we have alternate roles, which are roles that are not covered by software objects. Alternate roles are represented by stick figures instead of a box. An activation bar is a rectangle on a lifeline, and it represents an object performing an operation. The bottom of an activation bar represents the end of the operation. Messages passed between objects are called call messages, and these are represented by arrows. Synchronous messages, the ones that the following diagrams will use, have a solid arrow head and it means the caller waits for a response. The following diagrams also take advantage of sequence fragments, which are a frame that encloses part of the diagram. An opt frame only runs certain conditions, an alt frame means there are multiple versions of an operation, but only one of which runs, and the loop frame runs a fragment possibly multiple times.

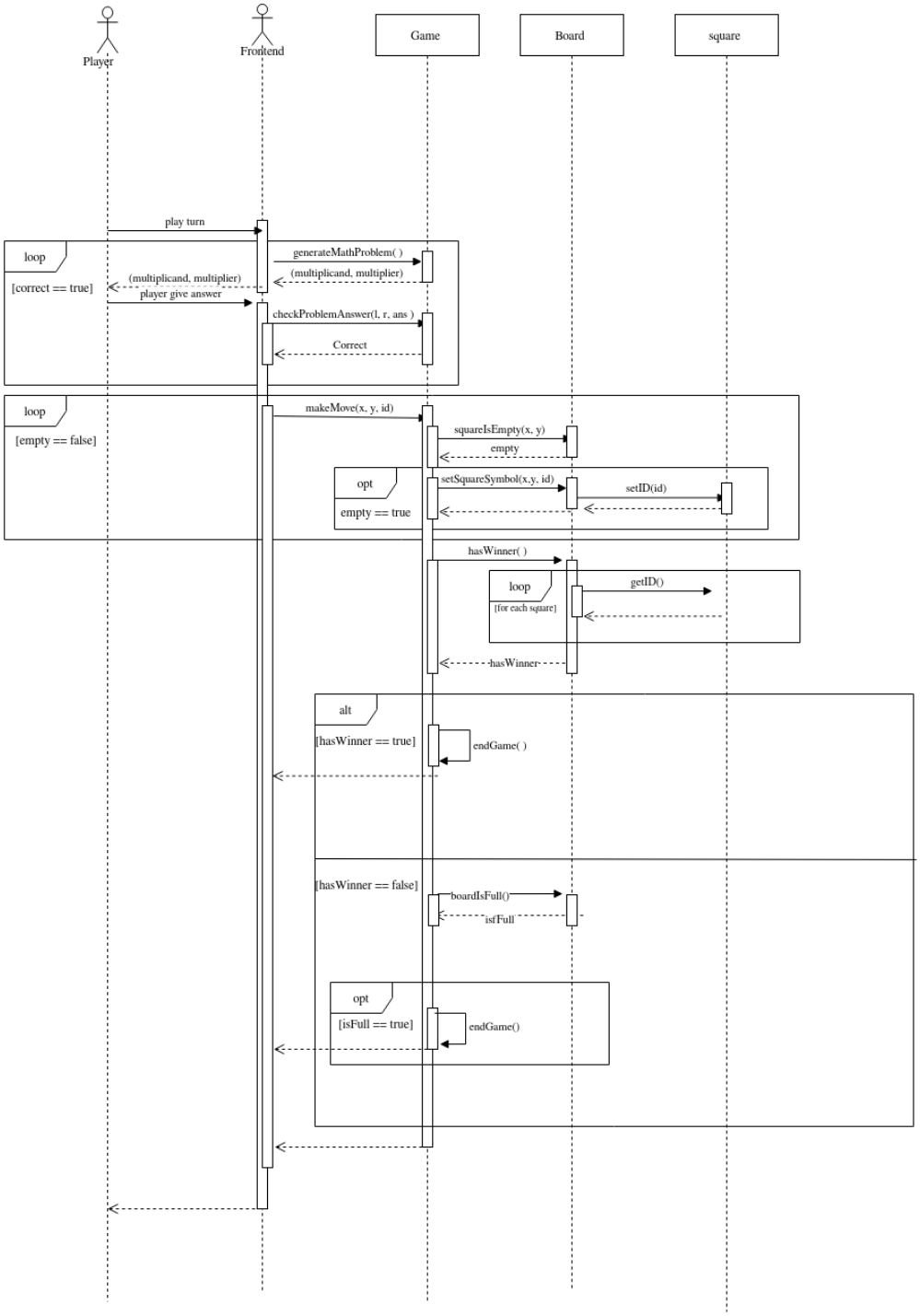
Scenario 1: A player creates and starts a new Tic-Tac-Math game by launching the webpage and startGame() is automatically triggered, making the game immediately available.



Scenario 2: A player makes a move on the board. The game generates a math problem with a multiplicand and a multiplier, then displays these values to the player in the form of Multiplicand x Multiplier. When the question is answered by the user, the game will check for the validity of the response and only allow the square to be claimed if the question was answered correctly.

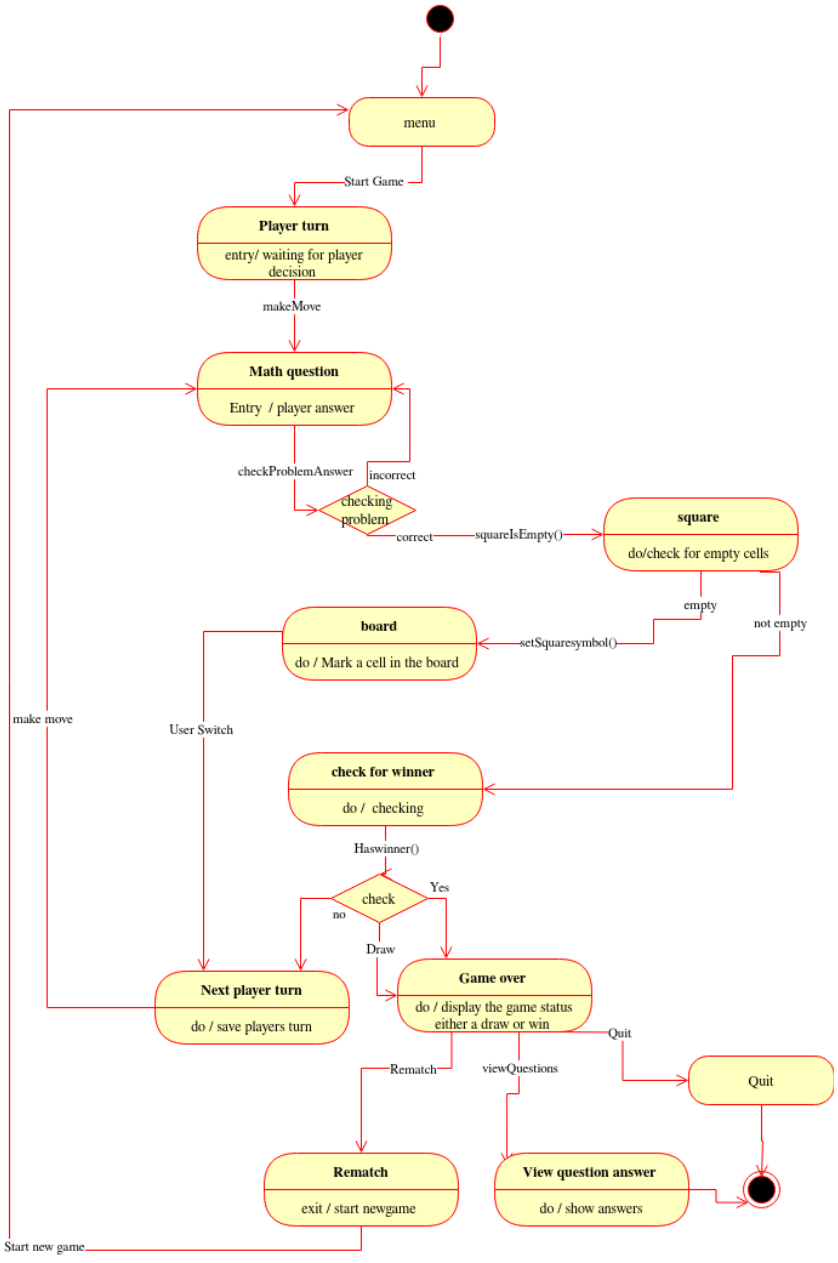


Scenario 3: A player claims the last square and ends the game. Continuing from the events shown in Scenario 2, the game will end if the last square is claimed. When the last square is claimed, the game will check if there are any winners or if the game was a tie.



State Diagram

A state diagram is a variation on finite automata. This means that the system has a state which can change if a certain event occurs. In this context, state is just an abstraction of the system's attributes. An event is just something that occurs at any point in time. State is represented by rounded rectangles and events are represented by labeled arrows. In a state diagram, the states can actually have actions which can be performed while in that state. There are three main types of actions: entry occurs when transitioning into the state, do is performed continuously while in the state, and exit occurs when transitioning out of the state. Actions are labeled with the format event/action.



5 Prototype

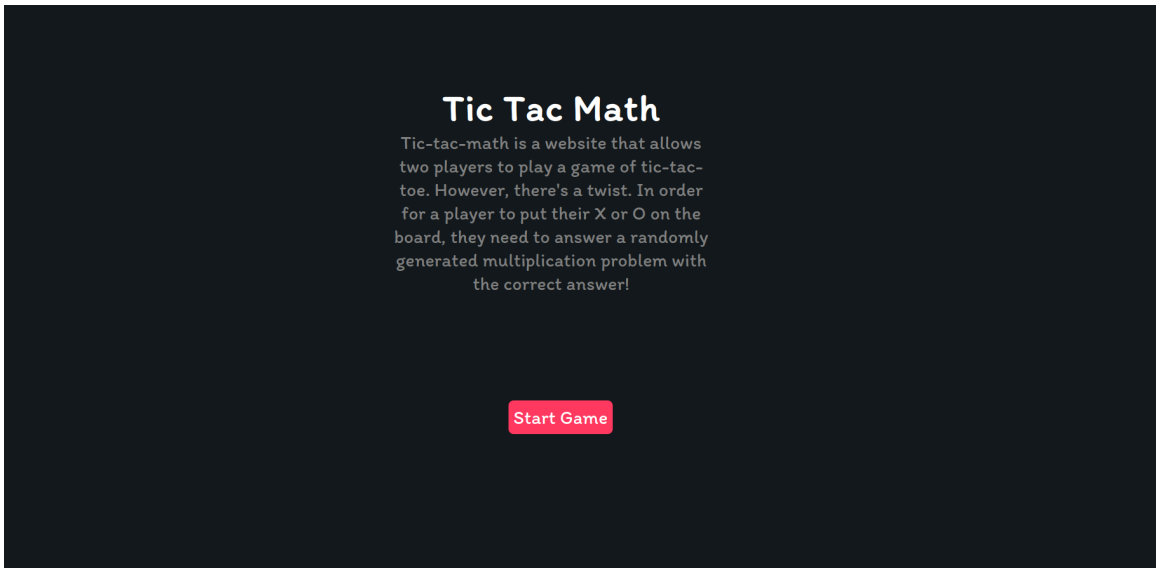
The Tic-Tac-Math prototype currently functions as intended. The prototype allows users to access a landing page, which when the “Start Game” button is pressed, will navigate them to a Tic-Tac-Toe board. The prototype allows Player 1, who is automatically given X, to make their first move on the board. When their move is selected, a text box will appear on the screen and prompt the user to enter their answer to the given math problem. From there, the game will place your respective symbol on the board. This repeats for both players until there is either a winner or there is a tie. When the game is over, the prototype displays either the winner or if it was a tie underneath the Tic-Tac-Toe board. At any point in the game, the users have the choice to clear the board and start the game over again. Players' statistics of correctly answered questions are tracked over the course of the round and return to the default 100% when the board is reset.

5.1 How to Run Prototype

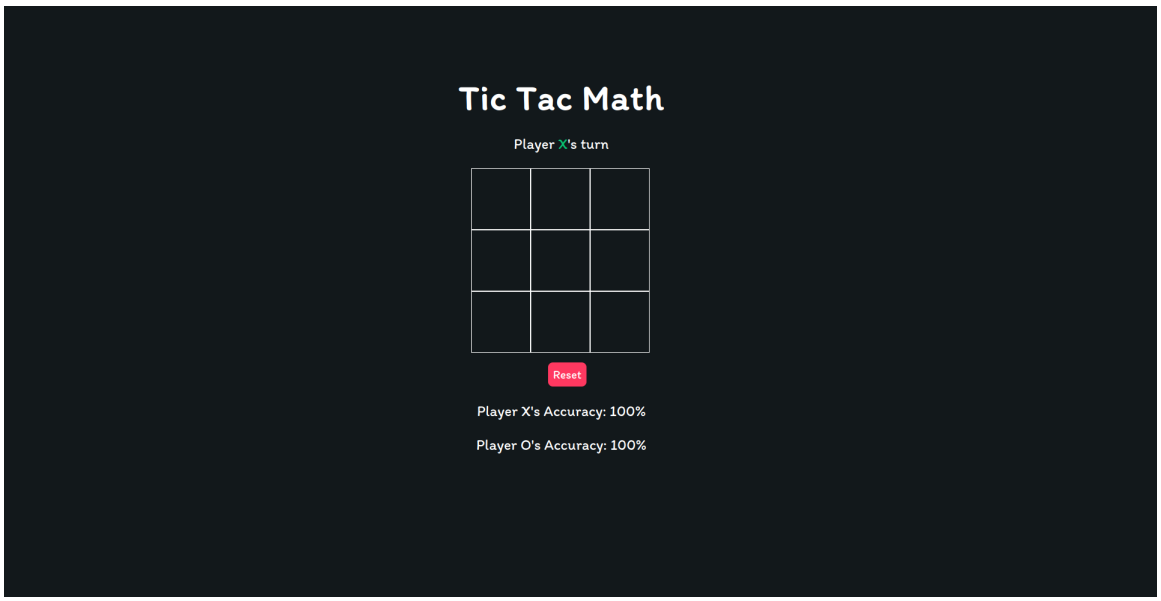
Running the Tic-Tac-Math prototype is incredibly simple. All the user needs to do is follow this link: https://jakecorrenti.github.io/tictacmath/. The link will take you to the landing page for Tic-Tac-Math, and from there all you have to do is have fun! There is no additional system configuration or plugins required in order to play Tic-Tac-Math. Additionally, assuming the user has access to a Web Browser and an internet connection, there are no specific OS or networking constraints.

5.2 Sample Scenarios

The following is a scenario where Player 1, who represents the symbol X, will be the winner of the game. First, the users navigate to the landing page. When they get to the landing page, they select the “Start Game” button, which takes them to the game board.



Tic-Tac-Math landing page

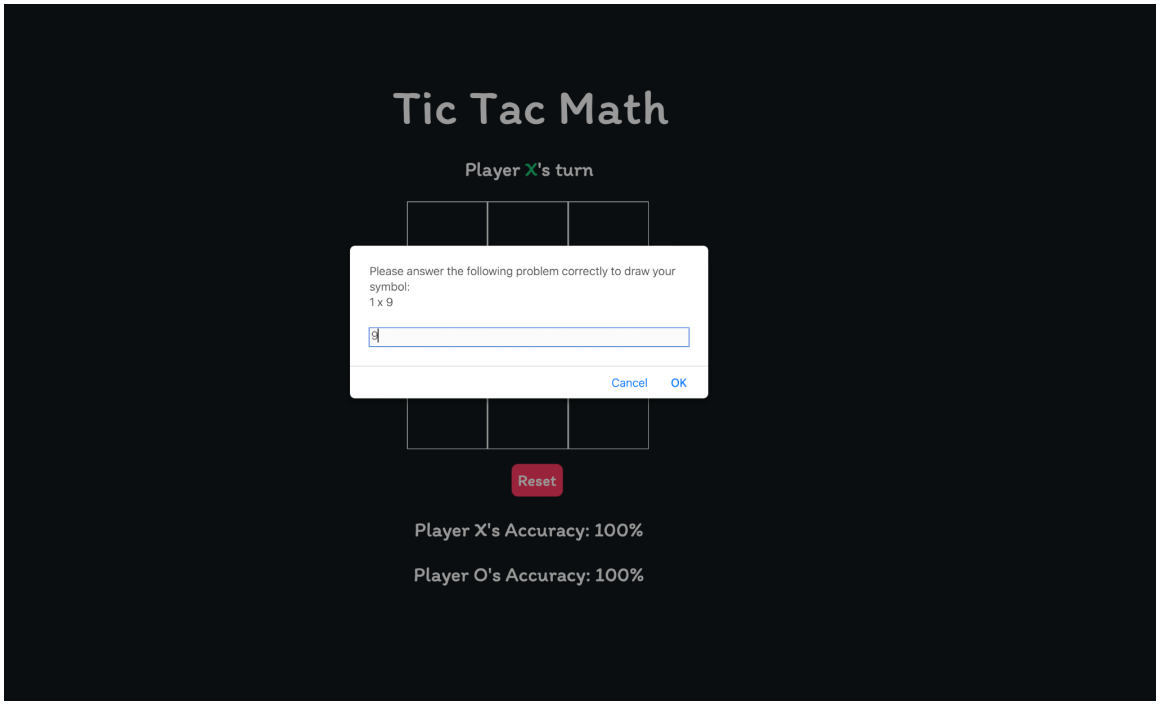


Navigated to the Tic-Tac-Math game board after pressing “Start Game”

Player 1 is now able to select where they want to put their X symbol. The player selects the square they want to put the symbol, and a pop-up will show up asking for

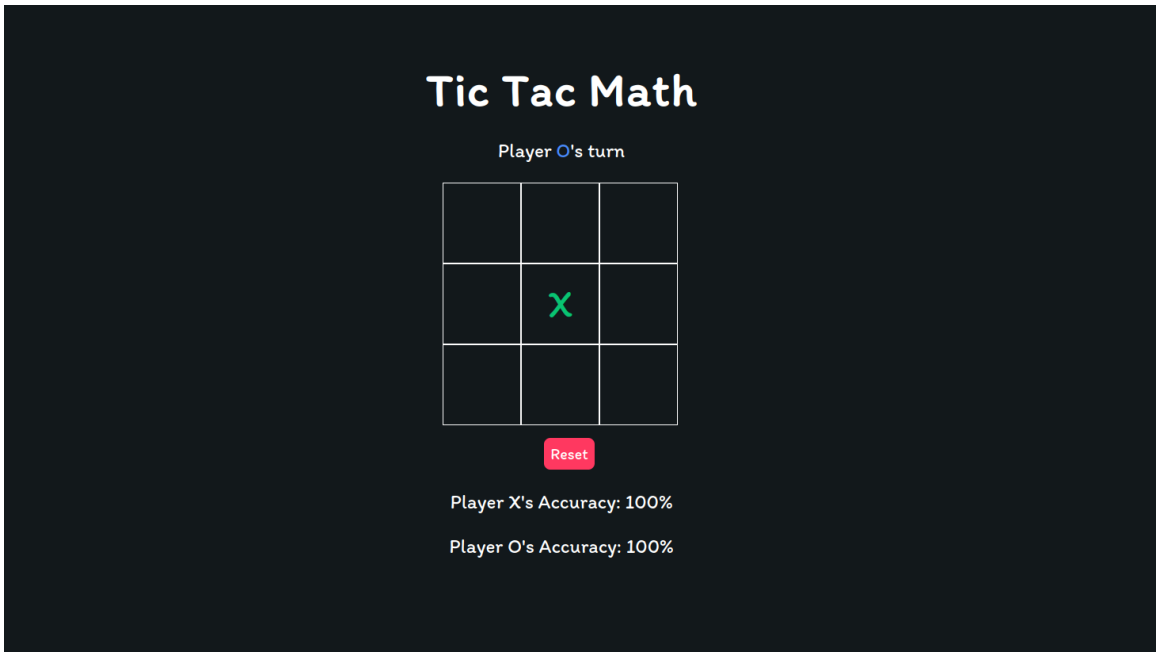
user input. The user will then input their answer for the problem.

Tic-Tac-Math prompting for the user's answer to the generated question



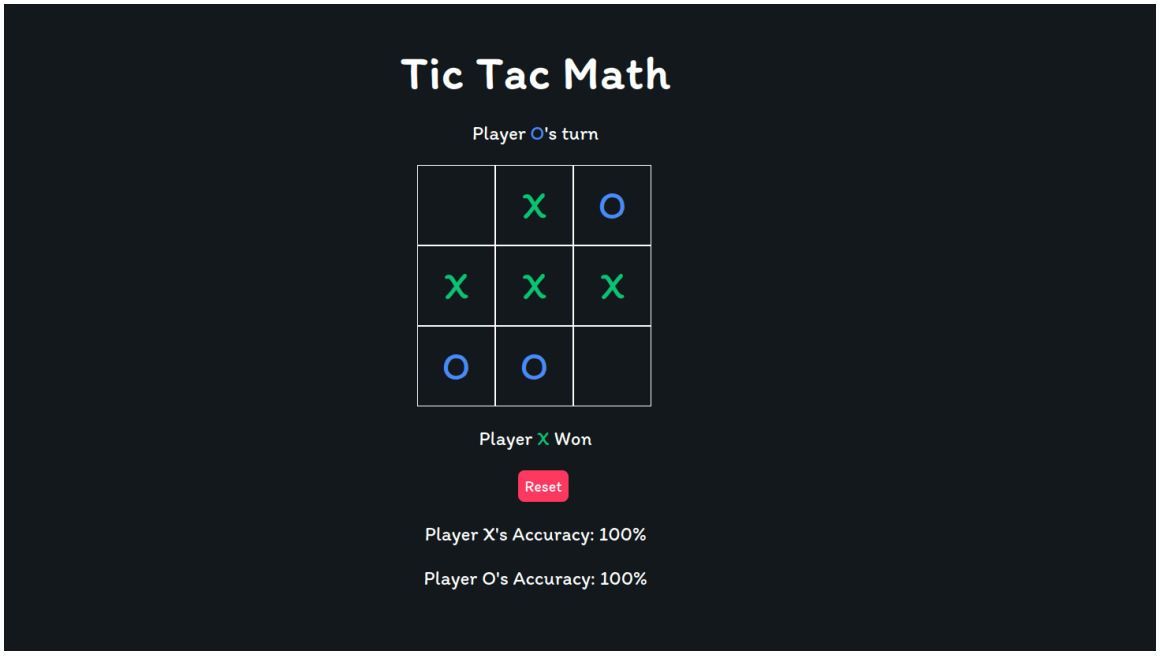
User inputs answer for problem

After the user inputs their answer to the problem, the game will place their symbol in the box they had selected. The game will then alternate to the next player's turn.



Player 1 input the correct answer to the problem. Now it is Player 2's turn.

Player 1 and Player 2 will alternate taking turns until either the game ends in a tie or a win. In this case, Player 1 is the successful opponent.



6 References

Start of your text.

[1] "IEEE Guide for Software Requirements Specifications," in IEEE Std 830-1984 , vol., no., pp.1-26, 10 Feb. 1984, doi: 10.1109/IEEESTD.1984.119205.

[2] “Tic-Tac-Math,” *jakecorrenti.github.io*. https://jakecorrenti.github.io/tictacmath/ (accessed Nov. 21, 2023).

[3] “Project Background,” *tic-tac-math*. https://jakecorrenti.github.io/tic-tac-math/ (accessed Nov. 21, 2023).

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.