



**SIX WEEKS SUMMER TRAINING**

**REPORT**

On

**(C++ and Data Structures Algorithms)**

**Submitted by**

**Name: Penugonda Pranav**

**Registration Number: 12104850**

**Program Name: B.Tech. (Hons.) (CSE- Cyber Security and Block Chain)**

Under the Guidance of

**Bannuru Rohit Kumar Reddy**

**School of Computer Science & Engineering**

**Lovely Professional University, Phagwara**

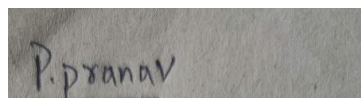
(June-July, 2023)

## DECLARATION

I hereby declare that I have completed my six weeks summer training at **Cipher Schools** from

**(11-06-2023) to (26-07-2023)** under the guidance Of **(Bannuru Rohit Kumar Reddy)**.

I have declare that I have worked with full dedication during these six weeks Of training and my learning outcomes fulfill the requirements of training for the award of degree of **B.Tech**, Lovely Professional University, Phagwara.

A rectangular box containing a handwritten signature in dark ink. The signature appears to be 'P. pranav' written in a cursive style.

Name: Penugonda Pranav

Registration No: 12104850

Date: 30-07-2023

## Summer Training certificate



### Project link:

<https://github.com/pranavpenugonda/DSA-cpp/tree/master/tic-tack-toe-project>

### Repository link:

<https://github.com/pranavpenugonda/DSA-cpp.git>

## Table of Contents

1.	Introduction
2.	Technology Learnt
3.	Reason for choosing this technology
4.	Profile of the problem
5.	Existing System
6.	Problem Analysis
7.	Software Requirement Analysis
8.	Design
9.	Implementation
10.	Learning outcome from training/technology learnt
11.	Gantt Chart
12.	Project Legacy
13.	Bibliography

## 1. Introduction

The name of the training course I opted is **C++ and Data Structures Algorithms**, the course started from **(11-06-2023)** and continued upto **(26-07-2023)**. In this course I've learned C++ programming language and then Data Structures concepts. During the training we went hands on different concepts of both C++ and DSA which really helped me to improve my coding skills and problem solving skills and at the end I've made a project on Tick-Tack-Toe game, It is terminal based UI which can be understandable and playable by any one.

## 2. Technology Learnt

During this six weeks summer training program I've learnt **C++ and Data Structures Algorithms**. This course covers all basics of C++ language like variables, data types, conditional statements, loops, arrays, functions, pointers, STL etc and also I learned Data Structures topics like array, stack, queue, linked list, trees, hash tables, time complexity, greedy algorithms and important algorithms like bubble sort, selection sort, quick sort, merge sort, binary search etc.

### **3. Reason For Choosing this Technology**

In this fast growing Technology era, Every day some new Technologies are coming but for any of them to learn the basic fundamental is to have any idea about any programming language and Data Structures and Algorithms in that. As C++ is very powerful language that supports object-oriented, procedural and generic programming. For DSA coding questions to solve easily we can use STL(Standard Template Library) in C++. It is a middle level language and it is object oriented, which makes it different from other programming languages. And it is some what easier than Java in terms of syntax. So, it could be easier to learn this language and gain programming skills which made me to take learn this technology.

## 4. Profile of the Problem

**Profile of the Problem:** Building a Tic-Tac-Toe Game using C++

### Introduction:

The Tic-Tac-Toe project involves implementing a classic two-player game using C++ programming language. The objective of the project is to create a functional and interactive game where two players take turns marking their symbol (either 'X' or 'O') on a 3x3 grid. The player who succeeds in placing three of their symbols in a horizontal, vertical, or diagonal row wins the game.

### Problem Statement:

Develop a console-based Tic-Tac-Toe game using C++ that allows two players to compete against each other. The program should handle user input, validate moves, display the current state of the game board, and determine the winner or a draw. The game should be user-friendly, providing clear instructions and visual feedback throughout.

### Scope:

- Designing the game logic for player turns and win conditions.
- Implementing user input mechanisms to accept player moves.
- Validating moves to ensure they are legal and the chosen cell is not already occupied.
- Displaying the game board and updating it after each move.
- Checking for win conditions after each move.
- Declaring a winner or a draw when appropriate.
- Providing a user-friendly interface with clear instructions and prompts.

### Methodology:

- Designing the game's data structures, such as a 2D array to represent the game board.
- Implementing functions to manage player input, validate moves, and update the game state.



- Incorporating control structures to handle player turns, win conditions, and game termination.
- Displaying the game board using console output and providing clear instructions.
- Testing the game thoroughly to identify and resolve any bugs or logical errors.

### **Expected Outcome:**

- A fully functional Tic-Tac-Toe game that can be played in a console environment.
- An executable program that allows two players to take turns, make moves, and determine a winner or a draw.
- Improved understanding of C++ programming concepts, data structures, and algorithm implementation.
- Practical experience in designing and implementing a simple interactive game.

### **Conclusion:**

Developing a Tic-Tac-Toe game using C++ provides a practical opportunity to apply the skills gained from the "C++ and Data Structures Algorithms" training course. The project encompasses various aspects of programming, data structures, and logical thinking, while also delivering an enjoyable and interactive gaming experience. Through this project, the student gains valuable experience in problem-solving, coding, and game development fundamentals.

## 5. Existing System

The concept of Tic-Tac-Toe has been a well-known game played using traditional pen and paper or even using physical game sets. These manual methods lack the convenience and automation that a computer program can provide. Players had to physically draw the game grid, mark X's and O's, and manually check for win conditions. And there are existing web applications and mobile applications on “tic-tack-toe” game and however, some users are facing issues related to UI, bugs related to game logic and sometimes the application is showing the wrong results. So, I have built an simple terminal based application on “tic-tack-toe” game which is user-interactive and simple to play, which can be understandable by anyone.

### Challenges of the Manual Approach:

**Human Errors:** In the manual approach, errors could occur due to miscalculations, missed moves, or misinterpretation of the game state. This could lead to incorrect game outcomes.

**Lack of Automation:** There was no built-in mechanism to validate moves, detect wins or draws, or manage the game flow automatically. Players had to rely on their own judgment.

**Limited Interaction:** Players could only play against each other, making it difficult to practice or experiment with different strategies alone.

### Advantages of the Proposed Computerized System:

**Automation:** The computerized system automates game mechanics, including move validation, win detection, and game progression. This eliminates the risk of human errors.

**User-Friendly Interface:** The program provides a clear and interactive interface, guiding players through the game and displaying the current state of the board.

**Saves Time:** The software saves time compared to drawing and updating a physical game board, making it quicker and more efficient to play multiple rounds.

**Debugging and Improvement:** Since the system is implemented using code, any issues can be identified and fixed, leading to a more reliable and consistent gaming experience.

## **6. Problem Analysis**

The development of a Tic-Tac-Toe game using C++ involves a comprehensive problem analysis to identify the key components, challenges, and requirements of the project. This analysis forms the foundation for designing a robust and functional solution.

### **Game Logic and Rules:**

- Defining the game board structure, often represented as a grid.
- Determining how player moves are made and registered on the grid.
- Identifying win conditions by checking rows, columns, and diagonals for matching symbols.
- Recognizing draw conditions when the board is fully occupied without a winner.

### **Data Structures:**

- Choosing appropriate data structures to represent the game board, such as a 2D array, vector and some programming concepts like loops, functions, classes, objects etc;
- Designing a structure to store player information, including their symbols ('X' or 'O').
- Creating mechanisms to track the state of the game, such as ongoing, win, or draw.

### **Input Handling and Validation:**

- Defining how players enter their moves, often using row and column coordinates.
- Implementing checks to ensure inputs are within valid ranges and the chosen cell is unoccupied.
- Handling erroneous inputs gracefully, providing user-friendly error messages.

### **Game Flow Control:**

- Designing a mechanism to alternate between players' turns.
- Ensuring the game terminates when a winner is determined or a draw occurs.
- Incorporating options to restart the game or exit at the end of each round.

## 7. Software Requirement Analysis

**Software requirements analysis** focuses on the tasks that determine the needs or conditions to meet the new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders *analyzing, documenting, validating and managing* software.

### **Functional Requirements:**

Think of functional requirements as the things a system or software must do. They describe specific actions, features, or functionalities that the system should be able to perform. These requirements outline the "what" of the system and focus on its behavior and capabilities.

### **Non-Functional Requirements:**

Non-functional requirements, on the other hand, are about how the system should perform or behave. They focus on the characteristics, qualities, and constraints of the system. These requirements are often not directly related to the features of the system but rather define the "how" of the system.

### **Functional Requirements:**

#### **Game Initialization:**

- The game should initialize with an empty 3x3 grid.
- Players should be assigned their respective symbols ('X' or 'O').

#### **User Input:**

- Players should be able to input their moves by specifying the row and column of the desired cell.
- The input mechanism should validate the correctness of input coordinates.

### **Move Validation:**

- The system should validate if the selected cell is unoccupied before allowing the move.
- Invalid moves should be rejected with appropriate error messages.

### **Game Logic:**

- The game should alternate between the two players' turns.
- The system should detect winning conditions (three matching symbols in a row, column, or diagonal).
- The system should recognize draw conditions when the grid is fully occupied with no winner.

### **Game Progression:**

- After a move is made, the game board should be updated to reflect the current state.
- The game should continue until a player wins or a draw is achieved.

### **User Interface:**

- The system should display the current state of the game board in the console.
- Clear instructions should be provided to guide players on how to input moves.

### **Game Outcome:**

- The system should determine the winner and display the result at the end of the game.
- If there is a draw, the system should communicate that no winner has been decided.

### **Non-Functional Requirements:**

#### **Performance:**

- The game should respond to user inputs promptly without noticeable delays.

- The AI opponent's moves (if applicable) should be reasonably quick and well-paced.

**Usability:**

- The user interface should be intuitive and easy to understand, even for individuals unfamiliar with the game.

**Error Handling:**

- The system should gracefully handle invalid inputs and errors, providing clear error messages.

**Portability:**

- The game should be compatible with various C++ environments and compilers.

**Robustness:**

- The system should be designed to handle unexpected scenarios and edge cases without crashing.

**Maintainability:**

- The codebase should be well-structured and documented to facilitate future maintenance and enhancements.

**Testing:**

- The game should undergo thorough testing to identify and fix bugs, ensuring a stable user experience.



## 8. Design

Design is a critical phase where the concepts and requirements identified in earlier stages are translated into a structured plan. In the case of developing a Tic-Tac-Toe game using C++, the design phase involves defining the architecture, data structures, and algorithms that will bring the game to life.

### High-Level Architecture:

- User Interface: Displays the game board, accepts player inputs, and provides feedback.
- Game Logic: Implements the rules of the game, move validation, and win detection.

### Data Structures:

- Game Board: A 2D array or vector to represent the grid, with each cell holding the player's symbol (X, 'O') or indicating an empty cell.
- Player Information: A structure or class to store player details, such as name and symbol.
- Game State: Variables to track the current player's turn, whether the game is ongoing, won, or drawn.

### Game Logic:

- Initialize Game: Initialize the game board and player information at the start.
- Alternate Turns: Implement logic to alternate between players' turns after each move.
- Input Handling: Accept player input for row and column, validate it, and update the board.
- Move Validation: Check if the chosen cell is empty and within valid range before making a move.
- Win Detection: After each move, check for win conditions in rows, columns, and diagonals.
- Draw Condition: If the board is fully occupied without a win, declare a draw.

- **Game Termination:** End the game when a win or draw is achieved, allowing players to restart or exit.

### **User Interface:**

- **Display Board:** Show the current state of the game board in the console.

**Prompt Input:** Prompt the current player to input their move (row and column).

- **Error Handling:** Display user-friendly messages for invalid inputs or errors.
- **Game Outcome:** Display the result when the game ends (win, draw, or exit option).

### **Error Handling:**

Implement mechanisms to handle unexpected scenarios and provide clear error messages to guide users and developers in diagnosing issues.

### **Testing:**

Regularly test the code as you develop to catch bugs and ensure each component is functioning as intended. Cover various scenarios like valid moves, wins, draws, and edge cases.

I've developed a simple UI which can be understandable by every body. Initially the game prompts the "Welcome" message to the user, and then displays the basic instructions to the user like player info, player input symbols and row, column numbering. User has to input the row and column number, as player switches for every input, so user has to input only the row and column number. For every input the game board should be updated and should be visible to the players. I've used 2D vector to store all these values. It should takes inputs and verify them and if any player wins , It displays "congratulations" text and if winner isn't declared then game continues and user enters any invalid input then it prompts "invalid" text, If draw the it displays "draw" text.

## Sample output:

```
C:\Users\LENOVO\Desktop\Ttk >
Welcome to Tic-Tac-Toe!
Player 1: X, Player 2: 0
To win the game make sure to enter symbol either in row/column/diagonal 3times and they should form a line
The board is numbered from 0 to 2 (row, col).
Current Board:
- - -
- - -
- - -

Player 1, enter your move (row col): 0 2
Current Board:
- - X
- - -
- - -

Player 2, enter your move (row col): 2 0
Current Board:
- - X
- - -
0 - -

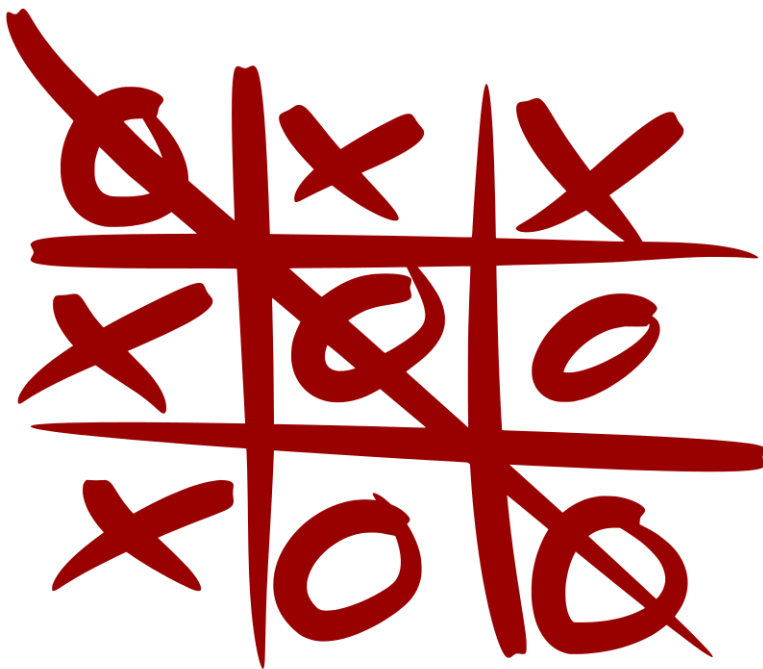
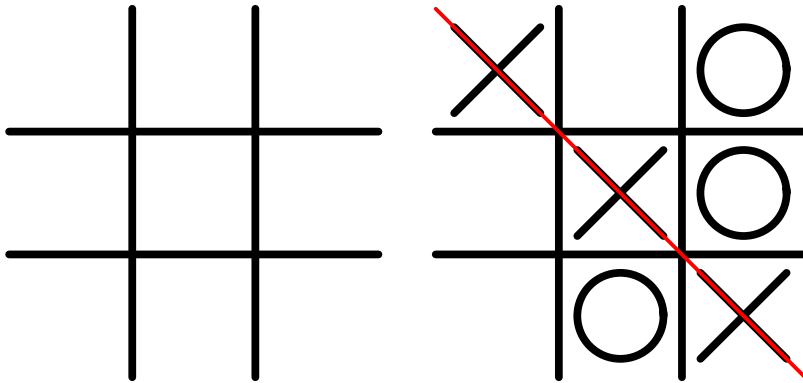
Player 1, enter your move (row col): 1 2
Current Board:
- - X
- - X
0 - -

Player 2, enter your move (row col): 1 0
Current Board:
- - X
0 - X
0 - -

Player 1, enter your move (row col): 2 2
Final Board:
- - X
0 - X
0 - X
Congratulations! Player 1 wins!

-----
Process exited after 333.2 seconds with return value 0
Press any key to continue . . .
```

## 9. Implementation



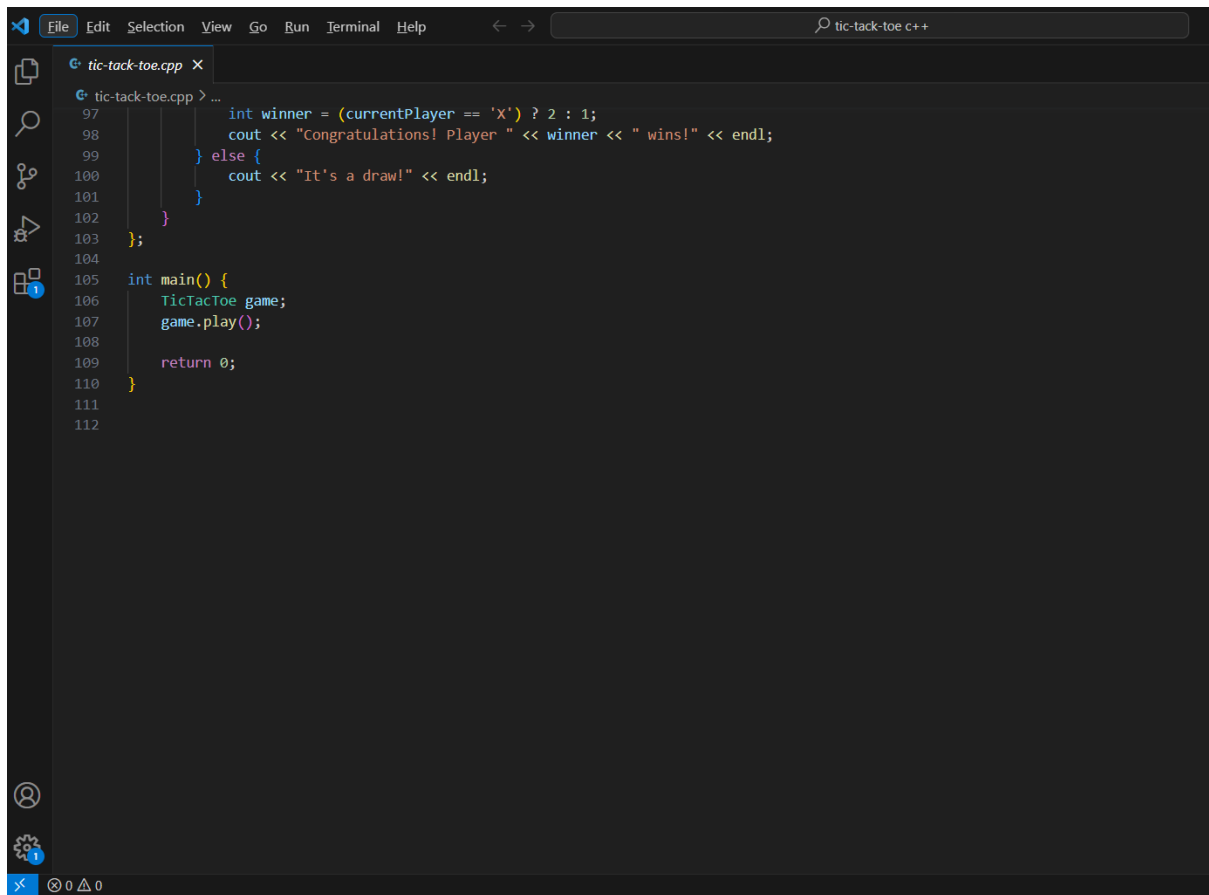
```
File Edit Selection View Go Run Terminal Help
tic-tack-toe c++

tic-tack-toe.cpp x
tic-tack-toe.cpp > ...
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 class TicTacToe {
6 private:
7     vector<vector<char>> > board;
8     char currentPlayer;
9
10 public:
11     TicTacToe() : board(3, vector<char>(3, '-')), currentPlayer('X') {}
12
13     // Print the current state of the board
14     void printBoard() const {
15         for (int i = 0; i < 3; i++) {
16             for (int j = 0; j < 3; j++) {
17                 cout << board[i][j] << " ";
18             }
19             cout << endl;
20         }
21     }
22
23     // Check if the board is full
24     bool isBoardFull() const {
25         for (int i = 0; i < 3; i++) {
26             for (int j = 0; j < 3; j++) {
27                 if (board[i][j] == '-')
28                     return false;
29             }
30         }
31         return true;
32     }
33
34     // Check if there is a winner
35     bool checkWinner() const {
36         // Check rows
37         for (int i = 0; i < 3; i++) {
```

```
File Edit Selection View Go Run Terminal Help
tic-tack-toe c++

tic-tack-toe.cpp x
tic-tack-toe.cpp > ...
31         return true;
32     }
33
34     // Check if there is a winner
35     bool checkWinner() const {
36         // Check rows
37         for (int i = 0; i < 3; i++) {
38             if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != '-')
39                 return true;
40         }
41
42         // Check columns
43         for (int j = 0; j < 3; j++) {
44             if (board[0][j] == board[1][j] && board[1][j] == board[2][j] && board[0][j] != '-')
45                 return true;
46         }
47
48         // Check diagonals
49         if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != '-')
50             return true;
51
52         if (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != '-')
53             return true;
54
55         return false;
56     }
57
58     // Make a move
59     void makeMove(int row, int col) {
60         if (row < 0 || row > 2 || col < 0 || col > 2) {
61             cout << "Invalid move. Try again." << endl;
62             return;
63         }
64
65         if (board[row][col] != '-') {
66             cout << "Cell already occupied. Try again." << endl;
67             return;
68         }
69     }
70 }
```

```
File Edit Selection View Go Run Terminal Help
tic-tack-toe.cpp X
tic-tack-toe.cpp > ...
64
65     if (board[row][col] != '-') {
66         cout << "Cell already occupied. Try again." << endl;
67         return;
68     }
69
70     board[row][col] = currentPlayer;
71     currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
72 }
73
74 // Play the game
75 void play() {
76     cout << "Welcome to Tic-Tac-Toe!" << endl;
77     cout << "Player 1: X, Player 2: O" << endl;
78     cout << "To win the game make sure to enter symbol either in row/column/diagonal 3times and they should form a line"<<endl;
79     cout << "The board is numbered from 0 to 2 (row, col)." << endl;
80
81     int row, col;
82     while (!checkWinner() && !isBoardFull()) {
83         cout << "Current Board:" << endl;
84         printBoard();
85
86         int playerNum = (currentPlayer == 'X') ? 1 : 2;
87         cout << "Player " << playerNum << ", enter your move (row col): ";
88         cin >> row >> col;
89
90         makeMove(row, col);
91     }
92
93     cout << "Final Board:" << endl;
94     printBoard();
95
96     if (checkWinner()) {
97         int winner = (currentPlayer == 'X') ? 2 : 1;
98         cout << "Congratulations! Player " << winner << " wins!" << endl;
99     } else {
100         cout << "It's a draw!" << endl;
101     }
}
```



```
File Edit Selection View Go Run Terminal Help
tic-tack-toe.cpp X
tic-tack-toe.cpp > ...
97         int winner = (currentPlayer == 'X') ? 2 : 1;
98         cout << "Congratulations! Player " << winner << " wins!" << endl;
99     } else {
100         cout << "It's a draw!" << endl;
101     }
102 }
103 };
104
105 int main() {
106     TicTacToe game;
107     game.play();
108
109     return 0;
110 }
111
112
```

## Output:

```
C:\Users\LENOVO\Desktop\TicTacToe >
Player 2, enter your move (row col): 2 0
Current Board:
- - -
- X -
0 - -
Player 1, enter your move (row col): 0 0
Current Board:
X - -
- X -
0 0 -
Player 2, enter your move (row col): 2 1
Current Board:
X - -
- X -
0 0 -
Player 1, enter your move (row col): 1 2
Current Board:
X - -
- X X
0 0 -
Player 2, enter your move (row col): 2 2
Final Board:
X - -
- X X
0 0 0
Congratulations! Player 2 wins!

-----
Process exited after 40.68 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\LENOVO\Desktop\TicTacToe >
Welcome to Tic-Tac-Toe!
Player 1: X, Player 2: O
To win the game make sure to enter symbol either in row/column/diagonal 3times and they should form a line
The board is numbered from 0 to 2 (row, col).
Current Board:
- - -
- - -
- - -
Player 1, enter your move (row col): 0 2
Current Board:
- - X
- - -
- - -
Player 2, enter your move (row col): 2 0
Current Board:
- - X
- - -
0 - -
Player 1, enter your move (row col): 1 2
Current Board:
- - X
- - X
0 - -
Player 2, enter your move (row col): 1 0
Current Board:
- - X
0 - X
0 - -
Player 1, enter your move (row col): 2 2
Final Board:
- - X
0 - X
0 - X
Congratulations! Player 1 wins!

-----
Process exited after 333.2 seconds with return value 0
Press any key to continue . . .
```



```
C:\Users\LENOVO\Desktop\Tik x + v
Welcome to Tic-Tac-Toe!
Player 1: X, Player 2: O
To win the game make sure to enter symbol either in row/column/diagonal 3 times and they should form a line
The board is numbered from 0 to 2 (row, col).
Current Board:
- - -
- - -
- - -
Player 1, enter your move (row col): 0 0
Current Board:
X - -
- - -
- - -
Player 2, enter your move (row col): 2 2
Current Board:
X - -
- - O
- - -
Player 1, enter your move (row col): 0 2
Current Board:
X - X
- - O
- - -
Player 2, enter your move (row col): 2 0
Current Board:
X - X
- - O
O - -
Player 1, enter your move (row col): 1 1
Current Board:
X - X
X - X
O - -
Player 2, enter your move (row col): 1 0
Current Board:
X - X
O X -
O - -
Player 1, enter your move (row col): 2 1
Current Board:
X - X
O X -
O X O
Player 2, enter your move (row col): 2 2
Cell already occupied. try again.
Current Board:
X - X
O X -
O X O
Player 2, enter your move (row col): 0 1
Current Board:
X O X
O X -
O X O
Player 1, enter your move (row col): 1 2
Final Board:
X O X
O X X
O X O
It's a draw!

Process exited after 92.75 seconds with return value 0
Press any key to continue . . .
```

## Code Explanation:

### Libraries:

<iostream>: Provides input and output functionality

<vector>: Allows the use of vectors to create a 2D grid board

`using namespace std`: Avoids the need to prepend `std: ` to standard library

Functions.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
```

### TicTacToe Class:

The 'TicTacToe' class encapsulates the game logic and functionality.

### Private Data Members:

```
6 private:
7     vector<vector<char> > board;
8     char currentPlayer;
```

`vector<vector<char> > board`: A 2D vector representing the game board.

`char currentPlayer`: Stores the current player's symbol ('X' or 'O').

### Constructor:

The constructor initializes the game board with empty cells and sets the starting player to 'X':

```
10 public:
11     TicTacToe() : board(3, vector<char>(3, '-')), currentPlayer('X') {}
12
```

### Member Functions:

**printBoard()**: Prints the current state of the game board to console

```
14     void printBoard() const {
15         for (int i = 0; i < 3; i++) {
16             for (int j = 0; j < 3; j++) {
17                 cout << board[i][j] << " ";
18             }
19             cout << endl;
20         }
21     }
```

**isBoardFull()**: Checks if the game board is fully occupied

```

23 // Check if the board is full
24 bool isBoardFull() const {
25     for (int i = 0; i < 3; i++) {
26         for (int j = 0; j < 3; j++) {
27             if (board[i][j] == '-')
28                 return false;
29         }
30     }
31     return true;
32 }

```

### checkWinner():

The checkWinner() function contains logic to check rows, columns, and diagonals for matching symbols and returns 'true' if a match is found.

```

35 bool checkWinner() const {
36     // Check rows
37     for (int i = 0; i < 3; i++) {
38         if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != '-')
39             return true;
40     }
41
42     // Check columns
43     for (int j = 0; j < 3; j++) {
44         if (board[0][j] == board[1][j] && board[1][j] == board[2][j] && board[0][j] != '-')
45             return true;
46     }
47
48     // Check diagonals
49     if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != '-')
50         return true;
51
52     if (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != '-')
53         return true;
54
55     return false;
56 }

```

### makeMove():

The makeMove() function validates the move based on the chosen cell's availability and player input. If the move is valid, it updates the board and switches players.

```

59     void makeMove(int row, int col) {
60         if (row < 0 || row > 2 || col < 0 || col > 2) {
61             cout << "Invalid move. Try again." << endl;
62             return;
63         }
64
65         if (board[row][col] != '-') {
66             cout << "Cell already occupied. Try again." << endl;
67             return;
68         }
69
70         board[row][col] = currentPlayer;
71         currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
72     }

```

### Play():

Implements the main game loop and player interactions:

```

75     void play() {
76         cout << "Welcome to Tic-Tac-Toe!" << endl;
77         cout << "Player 1: X, Player 2: O" << endl;
78         cout << "To win the game make sure to enter symbol either in row/column/diagonal 3times and they should form a line"<<endl;
79         cout << "The board is numbered from 0 to 2 (row, col)." << endl;
80
81         int row, col;
82         while (!checkWinner() && !isBoardFull()) {
83             cout << "Current Board:" << endl;
84             printBoard();
85
86             int playerNum = (currentPlayer == 'X') ? 1 : 2;
87             cout << "Player " << playerNum << ", enter your move (row col): ";
88             cin >> row >> col;
89
90             makeMove(row, col);
91         }
92
93         cout << "Final Board:" << endl;
94         printBoard();
95
96         if (checkWinner()) {
97             int winner = (currentPlayer == 'X') ? 2 : 1;
98             cout << "Congratulations! Player " << winner << " wins!" << endl;
99         } else {
100             cout << "It's a draw!" << endl;
101         }
102     }

```

### main():

In the main() function, an instance of the • TicTacToe• class is created, and the play() method is called to start the game:

```

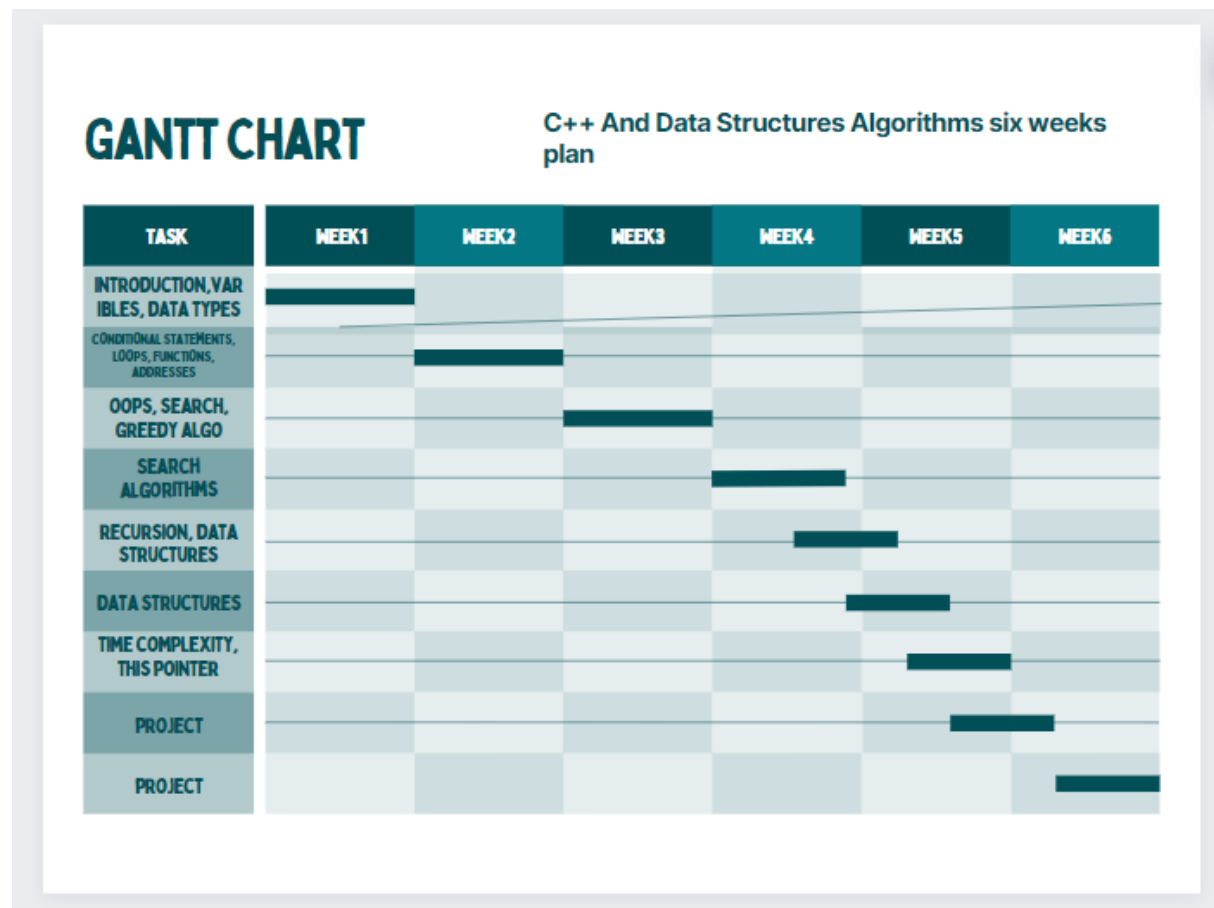
105     int main() {
106         TicTacToe game;
107         game.play();
108
109         return 0;
110     }
111
112

```

## **10. Learning outcome from training/technology learnt**

- Understanding the basic syntax, variables, data types, conditional statements, loops in c++.
- Understanding functions, address of elements, pointers in c++.
- Understanding data Structures like Arrays, Linked List, Stack, Queue, Hash tables etc.
- Important Algorithms like Linear Search, Binary Search, Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort etc.
- Understanding Object Oriented Programming concepts like class, objects, constructors, encapsulation, Inheritance, polymorphism, abstraction.
- Understanding recursion, greedy algorithms, time complexity, space complexity etc.
- Gaining Problem solving skills.

## 11. Gantt Chart



## 12. Project Legacy

Project legacy is the lasting mark a project leaves on individuals, organizations, and the field in which it operates. It reflects the project's broader impact on learning, growth, innovation, and the ongoing cycle of improvement.

Coding is not only about doing questions but also building upon real world projects that helps us to know about our current capability of understanding on concepts and implementing code integration is really important. This project really helped me to put all my theoretical skills to practical and then by gaining more in depth practical knowledge, And it helped me to grasp all the concepts at the time of project making, At that time I've known some C++ libraries and some problem solving approaches and I've built the problem solving mindset. It took me 7 to 9 days to complete the whole project. In this span of time I understood the importance of being consistent and patient about out work. As in my point of view I've done a descent "tic-tack-toe" game which can also be improved, so I've put it in **GitHub**, Questions and modifications are always allowed.

### **13. Bibliography**

**Resources I've used:**

<https://www.cipherschools.com/>

<https://cplusplus.com/>

<https://www.w3schools.com/>

<https://www.javatpoint.com/>