

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and
Engineering

By

NAME : Shyam Anand S

(Reg.No. 40731094)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

NOV 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Shyam Anand S** who carried out the project entitled "**Tic-Tac-Toe Game**" under my supervision from Aug 2022 to Oct 2022.

Internal Guide

Ms.S.Pothumani.,M.E(Ph.D)

Head of the Department

Dr.L.Lakshmanan.M.E.,Ph.D

Submitted for Viva voce Examination held on 12-09-2022

Internal Examiner

External Examiner

DECLARATION

I **Shyam Anand S** hereby declare that the Project Report entitled **Tic-Tac-Toe Game** by using Flutter was Done by me under the guidance of **Ms.S.Pothumani.,M.E(Ph.D)** at **AlgoxFusion (Pride)** (Company name and address) is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 12-09-2022

PLACE: Chennai

A handwritten signature in blue ink, appearing to read 'Balbir', with a horizontal line drawn underneath it.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D**, Dean, School of Computing, **Dr. S. Vigneshwari, M.E., Ph.D.** and **Dr. L. Lakshmanan, M.E., Ph.D.**, Heads of the Department of **Computer Science and Engineering** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.S.Pothumani M.E(Ph.D)** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

CERTIFICATE



ABSTRACT

Tic-Tac-Toe game can be played by two players where the square block (3 x 3) can be filled with a cross (X) or a circle (O). The game will toggle between the players by giving the chance for each player to mark their move. When one of the players make a combination of 3 same markers in a horizontal, vertical or diagonal line the circuit will display which player has won, whether X or O. In this paper, we implement a 3x3 tic-tac-toe game on the breadboard using gates. The game is designed so that two players can play tic-tac-toe using button on the board that make the LED ON when respective button is pressed. The button will contain a display function and a select function to place the symbol as well as toggle between the symbols allowing each player a turn to play the game. The circuit will update after each player makes their move and check for the conditions of game as it goes on. Overall circuit works without any bugs.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
1.	ABSTRACT	1
	LIST OF FIGURES	7
	INTRODUCTION	10
	1.1 Tic-Tac-Toe	10
	1.2 History	10
	1.3 Gameplay	11
	1.4 Strategies	12
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	13
	2.1 Aim	13
	2.2 Objective	13
	2.3 Goal	13
	2.4 Need Of The Application	13
3.	FUNCTIONS & TOOLS	14
	3.1 Applications & Frameworks	14
	3.1.1 Flutter	14
	3.1.1.1 Framework Architecture	14
	3.1.2 Dart	14
	3.1.2.1 Usage	15
	3.1.3 Android Studio	18
	3.1.3.1 Features	18
	3.1.4 Visual Studio Code	19
	3.1.4.1 Features	19
	3.1.5 Hardware And Software Requirements	21
	3.1.5.1 Hardware Requirements	21
	3.1.5.2 Software Requirements	21

	3.1.6 Types of Widgets in Flutter	22
	3.1.7 Properties in Flutter	25
	3.1.8 Types of Packages	27
	3.1.8.1 Packages used in Tic-Tac-Toe	27
4.	Results & Screenshots	28
5.	SUMMARY AND CONCLUSION	33
	REFERENCES	34
	APPENDIX	35
	SOURCE CODE	36

LIST OF FIGURES

Fig no	Fig title	Page no
1.1	Tic-Tac-Toe	10
1.2	Game of Tic-Tac-Toe, Won by X	11
3.1	Flutter Logo	14
3.2	Dart Logo	16
3.3	Android Studio Logo	19
3.4	Visual Studio Code Logo	20
3.5	System Requirements	21
4.1	IDE View	28
4.2	IDE View with Android Emulator	29
4.3	Start of the Game	30
4.4	Middle of the Game	30
4.5	X is the Winner	31
4.6	O is the Winner	31
4.7	It's a Draw	32
4.8	Android Emulator	33
5.1	Visual Studio Coding Environment	36
5.2	Frameworks used in App Development	36

CHAPTER-1 : INTRODUCTION

1.1 Tic-Tac-Toe

Tic-Tac-Toe is a 2 Player game where the players get to place X & O in a 3X3 grid shaped space. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. It is a solved game, with a forced draw assuming best play from both players. This game is souley made for entertainment and it helps with thinking process.

1.2 History

- Egypt, where such game boards have been found on roofing tiles dating from around 1300 BC.
- An early variation of tic-tac-toe was played in the Roman Empire, around the first century BC. It was called terni lapilli (three pebbles at a time) and instead of having any number of pieces, each player had only three; thus, they had to move them around to empty spaces to keep playing. The game's grid markings have been found chalked all over Rome. Another closely related ancient game is three men's morris which is also played on a simple grid and requires three pieces in a row to finish, and Picaria, a game of the Pueblos.



Fig 1.1 Tic-Tac-Toe

1.3 Gameplay

- Tic-tac-toe is played on a three-by-three grid by two players, who alternately place the marks X and O in one of the nine spaces in the grid.
- In the following example, the first player (X) wins the game in seven steps:

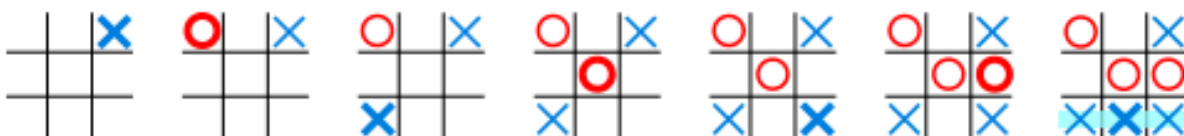


Fig 1.2 Game of Tic-Tac-Toe, Won by X

- There is no universally-agreed rule as to who plays first, but in this article the convention that X plays first is used.
- Players soon discover that the best play from both parties leads to a draw. Hence, tic-tac-toe is often played by young children who may not have discovered the optimal strategy.
- 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 that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space. If played optimally by both players, the game always ends in a draw, making tic-tac-toe a futile game.

1.4 Strategies

A player can play a perfect game of tic-tac-toe (to win or at least draw) if, each time it is their turn to play, they choose the first available move from the following list, as used in Newell and Simon's 1972 tic-tac-toe program.

1. **Win:** If the player has two in a row, they can place a third to get three in a row.
2. **Block:** If the opponent has two in a row, the player must play the third themselves to block the opponent.
3. **Fork:** Cause a scenario where the player has two ways to win (two non-blocked lines of 2).
4. **Blocking an opponent's fork:** If there is only one possible fork for the opponent, the player should block it. Otherwise, the player should block all forks in any way that simultaneously allows them to make two in a row. Otherwise, the player should make a two in a row to force the opponent into defending, as long as it does not result in them producing a fork. For example, if "X" has two opposite corners and "O" has the center, "O" must not play a corner move to win. (Playing a corner move in this scenario produces a fork for "X" to win.)
5. **Center:** A player marks the center. (If it is the first move of the game, playing a corner move gives the second player more opportunities to make a mistake and may therefore be the better choice; however, it makes no difference between perfect players.)
6. **Opposite corner:** If the opponent is in the corner, the player plays the opposite corner.
7. **Empty corner:** The player plays in a corner square.
8. **Empty side:** The player plays in a middle square on any of the four sides.

CHAPTER-2 :

AIM AND SCOPE OF THE PRESENT INVESTIGATION

2.1 Aim

To create an application (Tic-Tac-Toe) using Flutter and Dart.

2.2 Objective

This application helps to refresh and focus our mind. Main ideology of this app is to entertain ourself in a way where our mind takes part in it. Helps with decision making and improving brains task process.

2.3 Goal

The project is basically targeted at those people who would like to entertain themselves in a productive way, enjoy their leisure time with their friends and family in a competitive way as well as in a fun way possible.

2.4 Need Of the Application

Main need of the application is to relieve stress and spend our leisure time usefully with our friend and families.

CHAPTER-3 : FUNCTIONS & TOOLS

3.1 Applications & Frameworks

3.1.1 Flutter

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase.



Fig 3.1 Flutter Logo

3.1.1.1 Framework architecture

The major components of Flutter include:

- Dart platform
- Flutter engine
- Foundation library
- Design-specific widgets
- Flutter Development Tools (DevTools)

3.1.2 Dart

Dart is a programming language designed for client development, such as for the web and mobile apps. It is developed by Google and can also be used to build server and desktop applications.

It is an object-oriented, class-based, garbage-collected language with C-style syntax. It can compile to either native code or JavaScript, and supports

interfaces, mixins, abstract classes, reified generics and type inference.

3.1.2.1 Usage

There are four ways to run Dart code:

Web

To run in mainstream web browsers, Dart relies on a source-to-source compiler to JavaScript. According to the project site, Dart was "designed to be easy to write development tools for, well-suited to modern app development, and capable of high-performance implementations. "In a web browser, the code is precompiled into JavaScript using the dart2js compiler, making it compatible with all major browsers with no need for browsers to adopt it. By optimizing the compiled JavaScript output to avoid expensive checks and operations, code written in Dart can, in some cases, run faster than equivalent code handwritten in JavaScript idioms.

Stand-alone

The Dart software development kit (SDK) ships with a stand-alone Dart VM, allowing Dart code to run in a command-line interface environment. As the language tools included in the SDK are written mostly in Dart, the Dart VM is a critical part of it. These tools include the dart2js compiler and a package manager called pub. Dart ships with a complete standard library allowing users to write fully working system apps, such as custom web servers.

Ahead-of-time compiled

Dart code can be AOT-compiled into machine code (native instruction sets). Apps built with Flutter, a mobile app SDK built with Dart, are deployed to app stores as AOT-compiled Dart code.

Native

Dart 2.6 includes the dart2native compiler to compile to self-contained, native executable code. Before Dart 2.6, this feature exposed this capability only on iOS and Android mobile devices via Flutter.

Snapshots

Snapshots, a core part of the Dart VM, are files that store objects and other runtime data.

Script snapshots

Dart programs can be compiled into snapshot files containing all of the program code and dependencies precompiled and ready to execute, allowing fast startups.

Full snapshots

The Dart core libraries can be compiled into a snapshot file that allows fast loading of the libraries. Most standard distributions of the main Dart VM have a prebuilt snapshot for the core libraries that is loaded at runtime.

Object snapshots

Dart is a very asynchronous language, using isolates for concurrency. Since these are workers that pass messages, it needs a way to serialize messages. This is done using a snapshot, which is generated from a given object, then transferred to another isolate for deserializing.

Native mobile apps

Google has introduced Flutter for native mobile app development on Android, iOS and Windows. Flutter is a mobile app SDK, complete with framework, widgets, and tools, that gives developers a way to build and deploy mobile apps, written in Dart. Flutter works with Firebase and other databases like sqflite provided by flutter at pub.dev and other mobile app SDKs, and is open source.



Fig 3.2 Dart Logo

Flutter engine

Flutter's engine, written primarily in C++, provides low-level rendering support

using either Google's Skia graphics library or the custom "Impeller" graphics layer.[16] Additionally, it interfaces with platform-specific SDKs such as those provided by Android and iOS to implement accessibility, file and network I/O, native plugin support, and more.

Foundation library

The Foundation library, written in Dart, provides basic classes and functions that are used to construct applications using Flutter, such as APIs to communicate with the engine.

Design-specific widgets

The Flutter framework contains two sets of widgets that conform to specific design languages: Material Design widgets implement Google's design language of the same name, and Cupertino widgets implement Apple's iOS Human interface guidelines.

IDE support

Flutter maintains official support for the following IDEs and editors via plugins:

- IntelliJ IDEA
- Android Studio
- Visual Studio Code
- Emacs

Other IDEs can be used with community-supported plugins, or by using Flutter tools from the command line.

Widgets

The basic component in a Flutter program is a "widget", which can in turn consist of other widgets. A widget describes the logic, interaction, and design of a UI element with an implementation similar to React. Unlike other cross-platform toolkits such as React Native and Xamarin which draw widgets using native platform components, Flutter renders widgets itself on a per-pixel basis. There are two fundamental types of widgets: stateless and stateful. Stateless widgets only update if their inputs change, making them very efficient, while stateful widgets can call the `setState()` method to update an internal state and redraw.

Although widgets are the primary method of constructing Flutter applications, they can also be bypassed in favor of directly drawing on a canvas. This feature has been occasionally used to implement game engines in Flutter.

3.1.3 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013, at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0. At the end of 2015, Google dropped support for Eclipse ADT, making Android Studio the only officially supported IDE for Android development.

On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development. Java is still supported, as is C++.

3.1.3.1 Features

The following features are provided in the current stable version:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations
- Support for building Android Wear apps
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud Messaging (Earlier 'Google Cloud Messaging') and Google App Engine.
- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

Android Studio supports all the same programming languages of IntelliJ (and CLion) e.g. Java, C++, and more with extensions, such as Go; and Android Studio 3.0 or later supports Kotlin and "all Java 7 language features and a subset of Java 8 language features that vary by platform version." External projects backport some Java 9 features. While IntelliJ states that Android Studio supports all released Java versions, and Java 12, it's not clear to what level Android Studio supports Java versions up to Java 12 (the documentation mentions partial Java 8 support). At least some new language features up to Java 12 are usable in Android.

Once an app has been compiled with Android Studio, it can be published on the Google Play Store. The application has to be in line with the Google Play Store developer content policy



Fig 3.3 Android Studio Logo

3.1.4 Visual Studio Code

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft with the Electron Framework, for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

In the Stack Overflow 2021 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool among 82,000 respondents, with 70% reporting that they use it.

3.1.4.1 Features

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js, Python, C++, C, Rust and Fortran. It is based on the Electron framework, which is used to develop Node.js web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

Out of the box, Visual Studio Code includes basic support for most common programming languages. This basic support includes syntax highlighting, bracket matching, code folding, and configurable snippets. Visual Studio Code also ships with IntelliSense for JavaScript, TypeScript, JSON, CSS, and HTML, as well as debugging support for Node.js. Support for additional languages can be provided by freely available extensions on the VS Code Marketplace.



Fig 3.4 Visual Studio Code Logo

Instead of a project system, it allows users to open one or more directories, which can then be saved in workspaces for future reuse. This allows it to operate as a language-agnostic code editor for any language. It supports many programming languages and a set of features that differs per language. Unwanted files and folders can be excluded from the project tree via the settings. Many Visual Studio Code features are not exposed through menus or the user interface but can be accessed via the command palette.

Visual Studio Code can be extended via extensions, available through a central repository. This includes additions to the editor and language support. A notable feature is the ability to create extensions that add support for new languages, themes, debuggers, time travel debuggers, perform static code analysis, and add code linters using the Language Server Protocol.

Source control is a built-in feature of Visual Studio Code. It has a dedicated tab inside of the menu bar where users can access version control settings and view changes made to the current project. To use the feature, Visual Studio Code must be linked to any supported version control system (Git, Apache Subversion, Perforce, etc.). This allows users to create repositories as well as to make push and pull requests directly from the Visual Studio Code program.

Visual Studio Code includes multiple extensions for FTP, allowing the software to be used as a free alternative for web development. Code can be synced between the editor and the server, without downloading any extra software.

Visual Studio Code allows users to set the code page in which the active document is saved, the newline character, and the programming language of the active document. This allows it to be used on any platform, in any locale, and for any given programming language.

Visual Studio Code collects usage data and sends it to Microsoft, although this can be disabled. Due to the open-source nature of the application, the telemetry code is accessible to the public, who can see exactly what is collected.

3.1.5 Hardware and Software requirements

3.1.5.1 Hardware Requirements:

- I3 Processor or above, 8th gen or above recommended
- 4 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

3.1.5.2 Software Requirements:

- Operating Systems: Windows 10 or later (64-bit), x86-64 based
- Disk Space: 1.64 GB (does not include disk space for IDE/tools)



Fig 3.5 System Requirements

3.1.6 Types of Widgets in Flutter

Align

A widget that aligns its child within itself and optionally sizes itself based on the child's size.

Image

It is a widget that displays an image.

Column

- A widget that displays its children in a vertical array.
- The Column widget does not scroll (and in general it is considered an error to have more children in a Column than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

Boxfit

Boxfit Scales and positions its child within itself according to fit.

Flatbutton

FlatButton is the material design widget in flutter. It is a text label material widget that performs an action when the button is tapped.

Alertdialog

- A Material Design alert dialog.
- An alert dialog (also known as a basic dialog) informs the user about situations that require acknowledgement. An alert dialog has an optional title and an optional list of actions. The title is displayed above the content and the actions are displayed below the content.
- If the content is too large to fit on the screen vertically, the dialog will display the title and the actions and let the content overflow, which is rarely desired.
- For dialogs that offer the user a choice between several options, consider using a SimpleDialog.

Text

Text widgets display and style text.

Textstyle

An immutable style describing how to format and paint text.

Row

- A widget that displays its children in a horizontal array.
- The Row widget does not scroll (and in general it is considered an error to have more children in a Row than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a ListView.

Navigator

- A widget that manages a set of child widgets with a stack discipline.
- Many apps have a navigator near the top of their widget hierarchy in order to display their logical history using an Overlay with the most recently visited pages visually on top of the older pages. Using this pattern lets the navigator visually transition from one page to another by moving the widgets around in the overlay. Similarly, the navigator can be used to show a dialog by positioning the dialog widget above the current page.

Scaffold

- Implements the basic Material Design visual layout structure.
- This class provides APIs for showing drawers and bottom sheets.
- To display a persistent bottom sheet, obtain the ScaffoldState for the current BuildContext via Scaffold.of and use the ScaffoldState.showBottomSheet function.

statefulwidget

- A widget that has mutable state.
- A stateful widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely
- It is immutable

Myappbar

- A Material Design app bar.
- App state
- the state of an app is everything that exists in memory when the app is running.

ListView

ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.

BuildContext

A handle to the location of a widget in the widget tree.

Container

A convenience widget that combines common painting, positioning, and sizing widgets.

Center

A widget that centers its child within itself.

Stack

A widget that positions its children relative to the edges of its box.

Globalkey

A key that is unique across the entire app, which is also a uniquely identify elements.

3.1.7 Properties in Flutter

alignment

Align the child within the container.

child

The child contained by the container.

clipBehavior

The clip behavior when Container.decoration is not null.

color

The color to paint behind the child.

constraints

Additional constraints to apply to the child.

decoration

The decoration to paint behind the child.

foregroundDecoration

The decoration to paint in front of the child.

hashCode

The hash code for this object.

key

Controls how one widget replaces another widget in the tree.

margin

Empty space to surround the decoration and child.

padding

Empty space to inscribe inside the decoration. The child, if any, is placed inside this padding.

transform

The transformation matrix to apply before painting the container.

transformAlignment

The alignment of the origin, relative to the size of the container, if transform is specified.

appBar

An app bar to display at the top of the scaffold.

backgroundColor

The color of the Material widget that underlies the entire Scaffold.

body

The primary content of the scaffold.

bottomNavigationBar

A bottom navigation bar to display at the bottom of the scaffold.

primary

Whether this scaffold is being displayed at the top of the screen.

3.1.8 Types of Packages

According to the functionality, we can categorize the package into two types:

1. Dart Package
2. Plugin Package

Dart Package

It is a general package, which is written in the dart language, such as a path package. This package can be used in both the environment, either it is a web or mobile platform. It also contains some Flutter specific functionality and thus has a dependency on the Flutter framework, such as fluro package.

Plugin Package

It is a specialized Dart package that includes an API written in Dart code and depends on the Flutter framework. It can be combined with a platform-specific implementation for an underlying platform such as Android (using Java or Kotlin), and iOS (using Objective C or Swift). The example of this package is the battery and image picker plugin package.

3.1.8 Packages used in Tic-Tac-Toe

Material library

Flutter widgets implementing Material Design.

Invert_color library

It detects whether the colors of the image are inverted when drawn.

Dart:Math library

Mathematical constants and functions, plus a random number generator.

CHAPTER 4 - Results & Screenshots

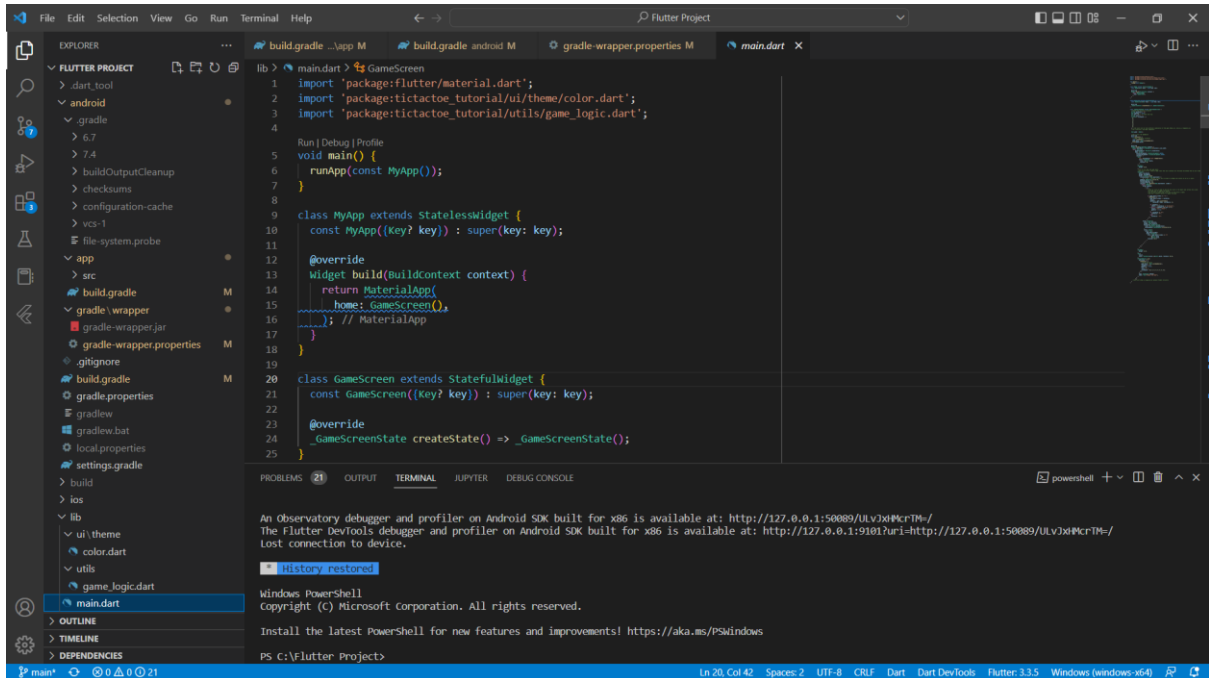


Fig 4.1 IDE View

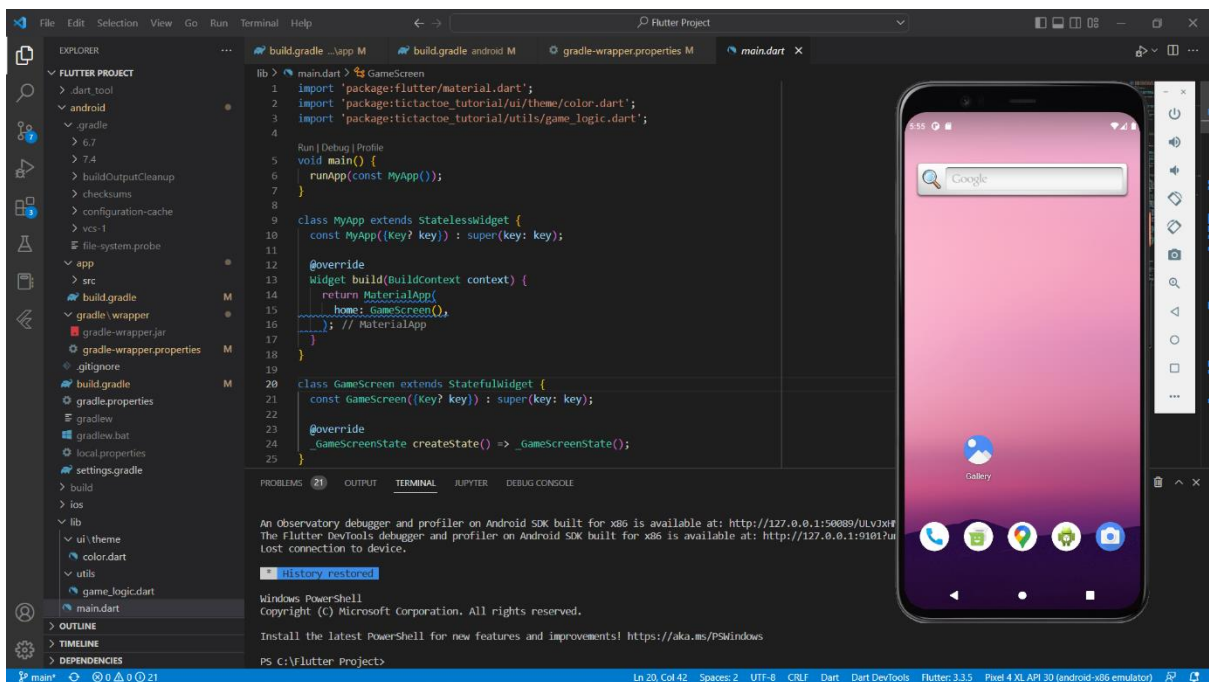


Fig 4.2 IDE View with Android Emulator

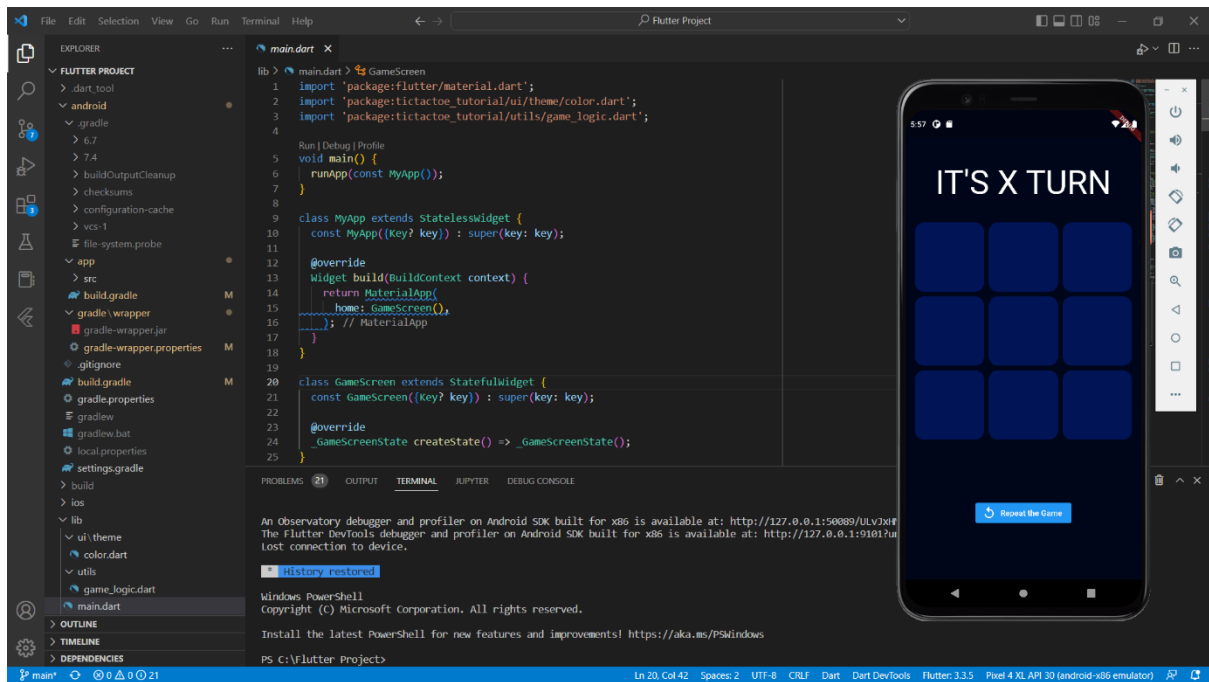


Fig 4.3 Start of the Game

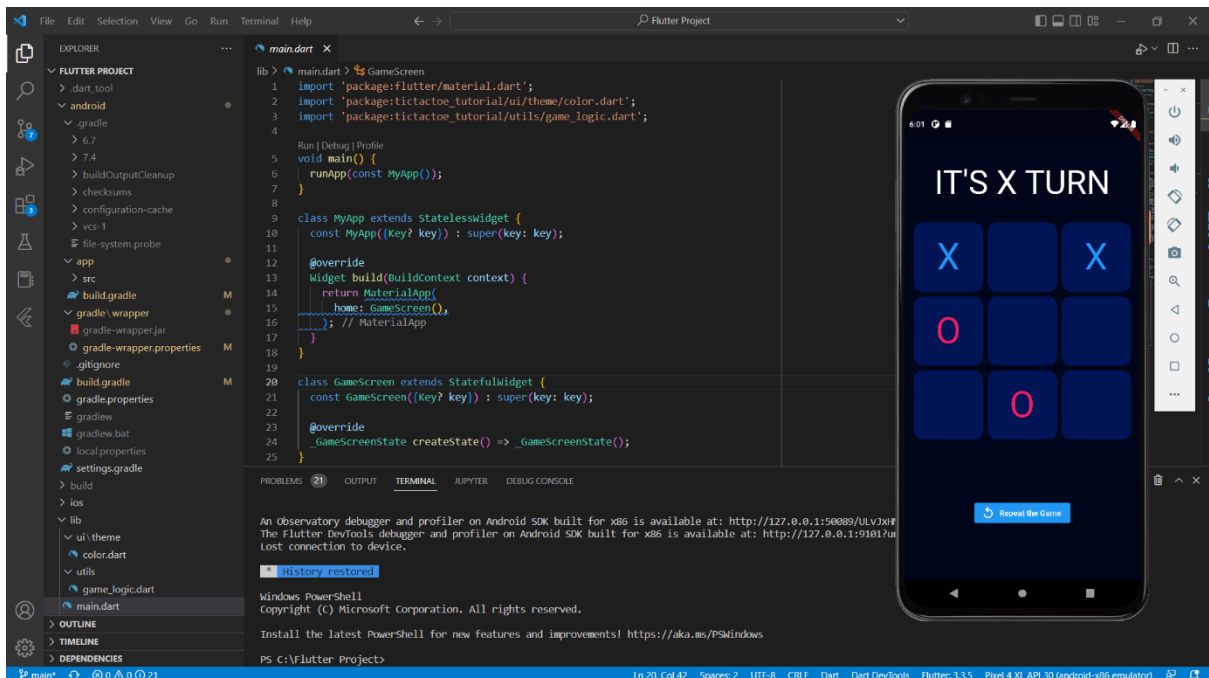


Fig 4.4 Middle of the Game

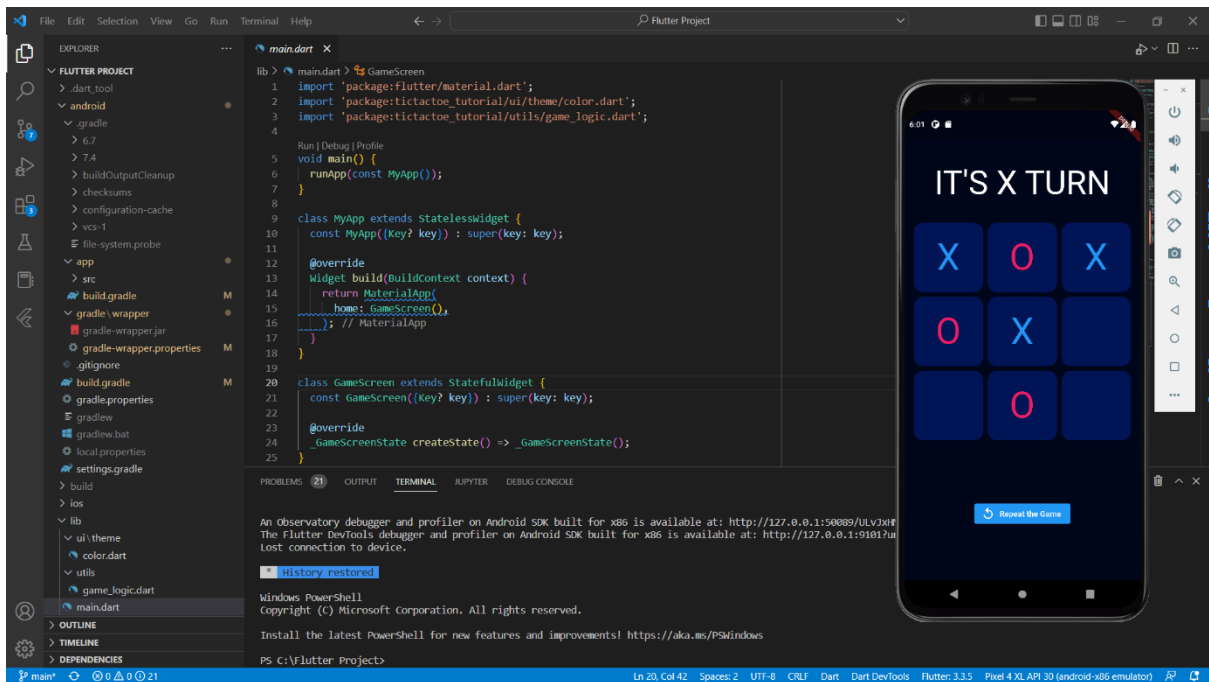


Fig 4.4 Middle of the Game

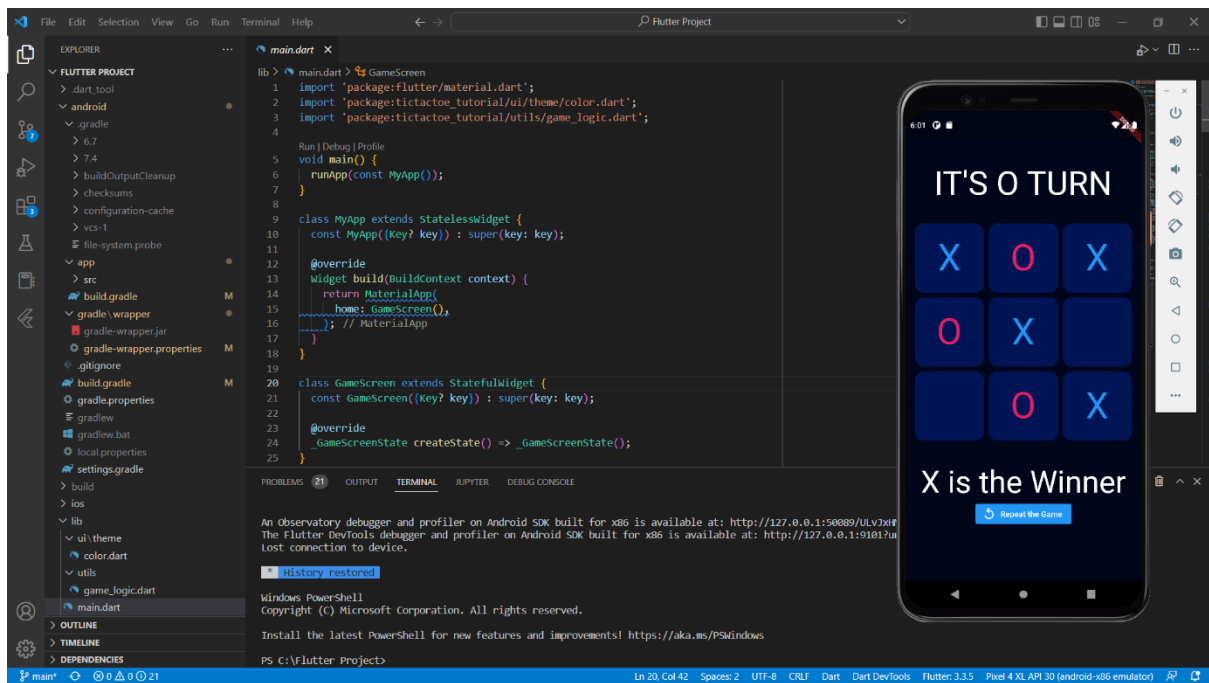


Fig 4.5 X is the Winner

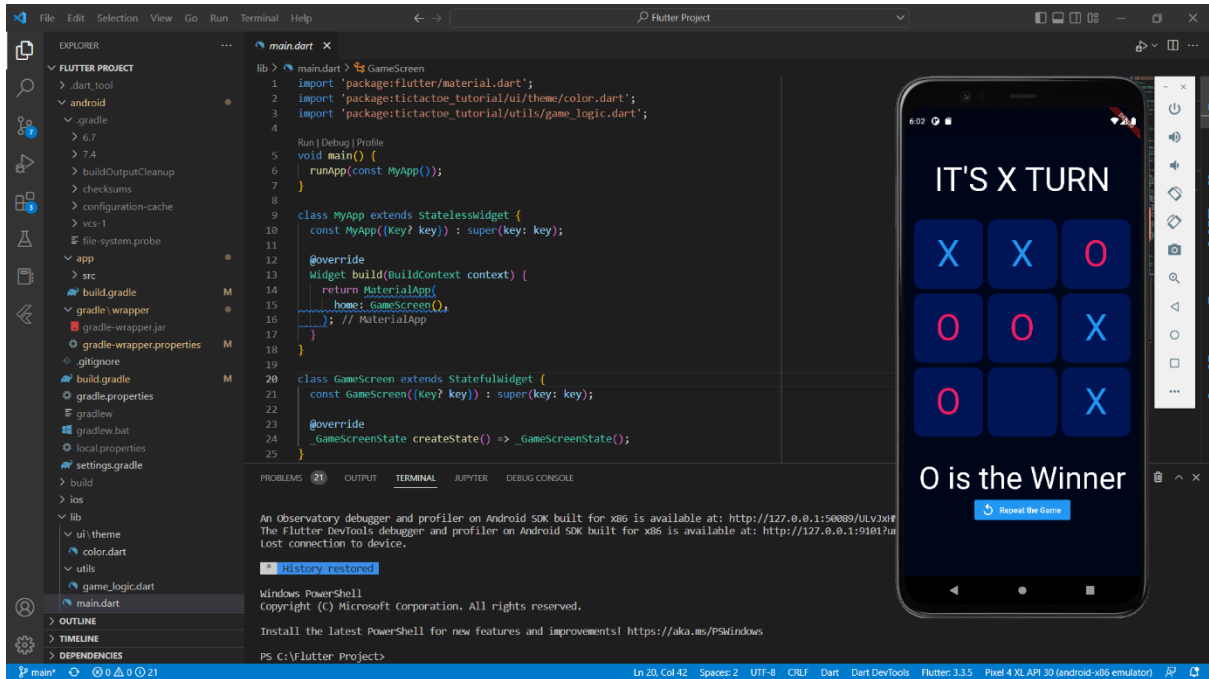


Fig 4.6 O is the Winner

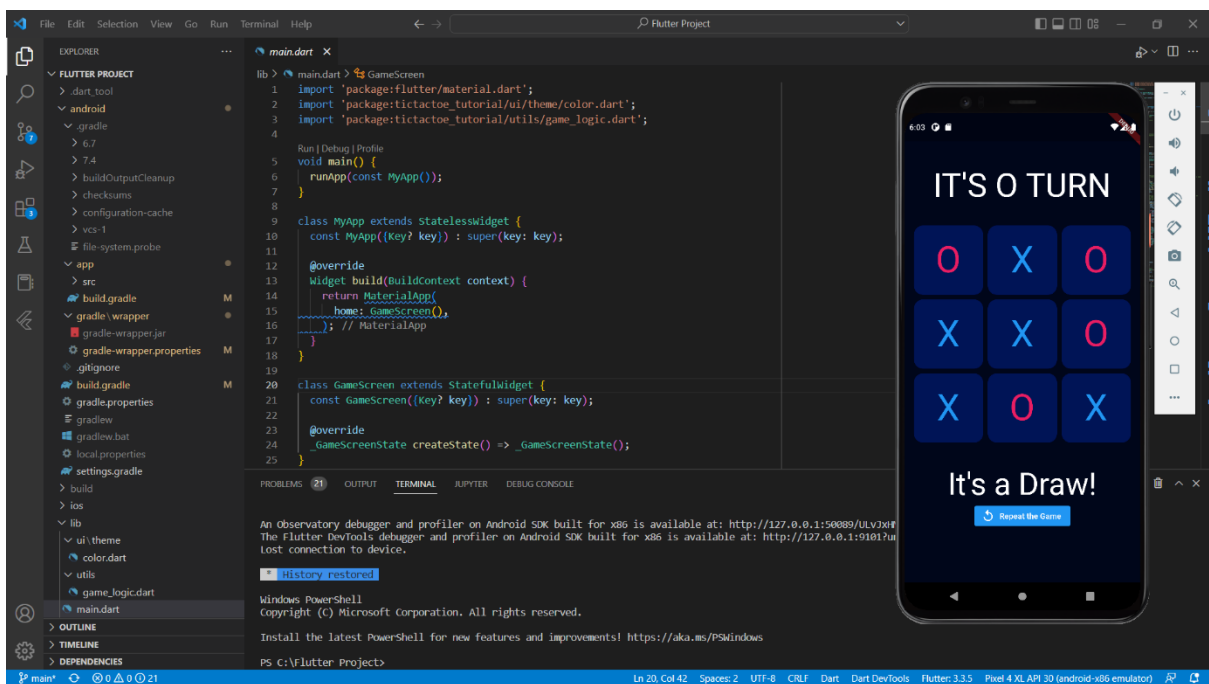


Fig 4.7 It's a Draw

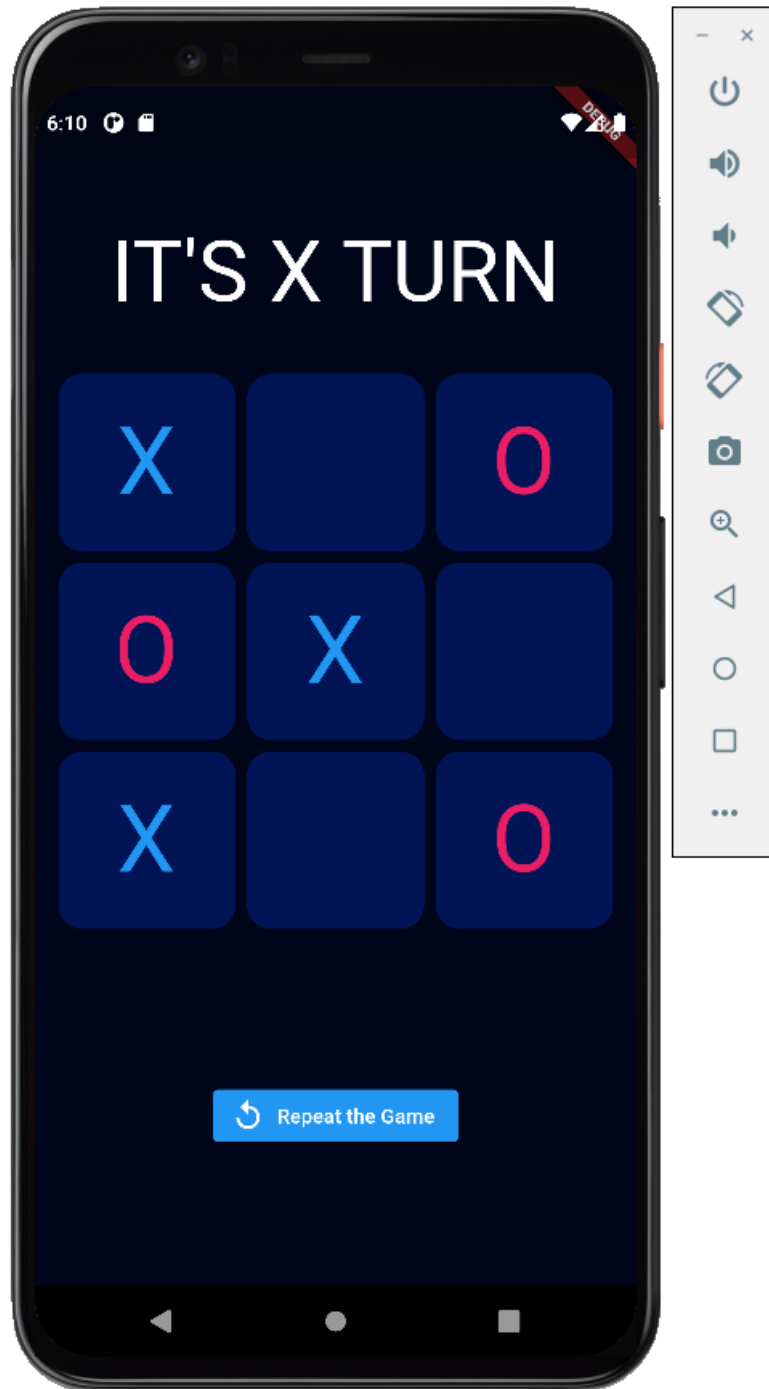


Fig 4.8 Android Emulator

CHAPTER-5 : Summary & Conclusion

- TIC TAC TOE is a commonly traditional game in Yemen, which consists of two players. One of the player chooses a cell of the game and puts an X mark in the chosen cell while the second player chooses another cell and puts an O mark in this second cell. The purpose of the paper is to transport this commonly traditional game to modern electronic game to get along with modern electronic world. The modern electronic game is important to countryside children's, so that they can play the game safely without incurring diseases. It also aims to offer children an opportunity to deal with cheap modern techniques. The game designed in this paper is based on the microcontroller.
- 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.
- This application helps to refresh and focus our mind. Main ideology of this app is to entertain ourself in a way where our mind takes part in it. Helps with decision making and improving brains task process.
- The project is basically targeted at those people who would like to entertain themselves in a productive way, enjoy their leisure time with their friends and family in a competitive way as well as in a fun way possible.
- Main need of the application is to relieve stress and spend our leisure time usefully with our friend and families.

References

<https://docs.flutter.dev/>

<https://en.wikipedia.org/wiki/Tic-tac-toe>

<https://stackoverflow.com/>

<https://dart.dev/guides/language/effective-dart/documentation>

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

<https://github.com/flutter/flutter>

Appendix

CODING ENVIRONMENT

Visual Studio Code is a coding IDE where we can write and execute our code. It has various customisations and extensions which makes programming easier and effective with various features pre-installed.



Fig 5.1 Visual Studio Coding Environment

FRAMEWORKS

Flutter and **Dart** are the language and programming language that we used for our project. The main advantage of this that we can design an app for both Android and ios at ease without any extra languages all in a single machine and at a single go.

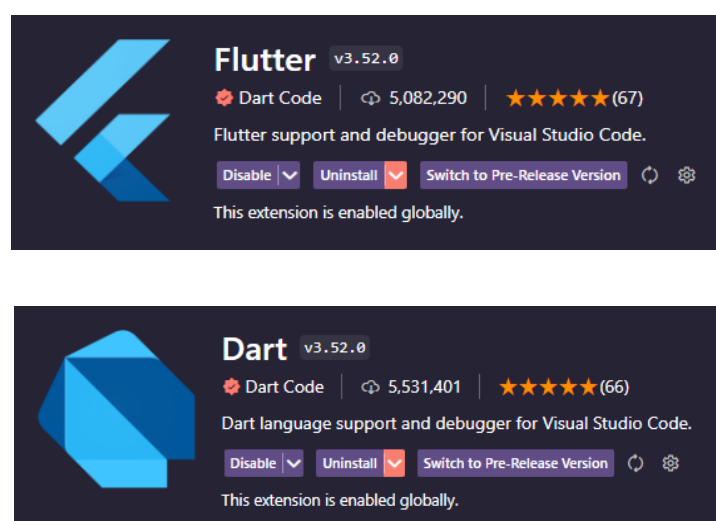


Fig 5.2 Frameworks used in App Development

Source Code

main.dart

```
import 'package:flutter/material.dart';
import 'package:tictactoe_tutorial/ui/theme/color.dart';
import 'package:tictactoe_tutorial/utils/game_logic.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: GameScreen(),
    );
  }
}

class GameScreen extends StatefulWidget {
  const GameScreen({Key? key}) : super(key: key);

  @override
  _GameScreenState createState() => _GameScreenState();
}

class _GameScreenState extends State<GameScreen> {
  //adding the necessary variables
  String lastValue = "X";
  bool gameOver = false;
  int turn = 0; // to check the draw
  String result = "";
  List<int> scoreboard = [
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
  ]
}
```

```

0
]; //the score are for the different combination of the game [Row1,2,3,
Col1,2,3, Diagonal1,2];
//let's declare a new Game components

Game game = Game();

//let's initi the GameBoard
@override
void initState() {
  // TODO: implement initState
  super.initState();
  game.board = Game.initGameBoard();
  print(game.board);
}

@override
Widget build(BuildContext context) {
  double boardWidth = MediaQuery.of(context).size.width;
  return Scaffold(
    backgroundColor: MainColor.primaryColor,
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        Text(
          "It's ${lastValue} turn".toUpperCase(),
          style: TextStyle(
            color: Colors.white,
            fontSize: 58,
          ),
        ),
        SizedBox(
          height: 20.0,
        ),
        //now we will make the game board
        //but first we will create a Game class that will contains all the data and
        method that we will need
        Container(
          width: boardWidth,
          height: boardWidth,
          child: GridView.count(
            crossAxisCount: Game.boardlenth ~/
              3, // the ~/ operator allows you to evide to integer and return an
            Int as a result
            padding: EdgeInsets.all(16.0),

```

```

mainAxisSpacing: 8.0,
crossAxisSpacing: 8.0,
children: List.generate(Game.boardlength, (index) {
  return InkWell(
    onTap: gameOver
      ? null
      : () {
          //when we click we need to add the new value to the board
and refrech the screen
          //we need also to toggle the player
          //now we need to apply the click only if the field is empty
          //now let's create a button to repeat the game

          if (game.board![index] == "") {
            setState() {
              game.board![index] = lastValue;
              turn++;
              gameOver = game.winnerCheck(
                lastValue, index, scoreboard, 3);

              if (gameOver) {
                result = "$lastValue is the Winner";
              } else if (!gameOver && turn == 9) {
                result = "It's a Draw!";
                gameOver = true;
              }
              if (lastValue == "X")
                lastValue = "O";
              else
                lastValue = "X";
            });
          }
        },
    child: Container(
      width: Game.blocSize,
      height: Game.blocSize,
      decoration: BoxDecoration(
        color: MainColor.secondaryColor,
        borderRadius: BorderRadius.circular(16.0),
      ),
      child: Center(
        child: Text(
          game.board![index],
          style: TextStyle(
            color: game.board![index] == "X"
              ? Colors.blue

```


game_logic.dart

```
class Player {
  static const x = "X";
  static const o = "O";
  static const empty = "";
}

class Game {
  static final boardlength = 9; // we will create a board of 3*3 blocks;
  static final blocSize = 100.0;

  //Creating the empty board
  List<String>? board;

  static List<String>? initGameBoard() =>
    List.generate(boardlength, (index) => Player.empty);

  //now we need to build the winner check algorithm
  //for this we need first to declare a scoreboard in our main file
  bool winnerCheck(
    String player, int index, List<int> scoreboard, int gridSize) {
    //first let's declare the row and col
    int row = index ~/ 3;
    int col = index % 3;
    int score = player == "X" ? 1 : -1;

    scoreboard[row] += score;
    scoreboard[gridSize + col] += score;
    if (row == col) scoreboard[2 * gridSize] += score;
    if (gridSize - 1 - col == row) scoreboard[2 * gridSize + 1] += score;

    //checking if we have 3 or -3 in the score board
    if (scoreboard.contains(3) || scoreboard.contains(-3)) {
      return true;
    }
    ;

    //by default it will return false
    return false;
  }
}
```