V.1.1

University of Central Lancashire
UCLan

Taster Day Activity

Introduction

FlutLab is a modern online development environment for building Flutter applications. Flutter is a modern programming framework for building cross-platform applications – applications that run on multiple platforms, such as, mobile, tablet, desktop, web, large screens. Imagine you have an idea for a game, a social media app, or an app to track your progress of completing homework and revision. Instead of starting from scratch and writing all the code for different platforms like Android, iOS, or web separately, Flutter lets you write code once and use it across all these platforms. This means less work for you and more time to focus on making your app awesome!

Flutter uses a language called Dart, which is easy to learn and understand. With Dart, you can write code to create buttons, text boxes, animations, and pretty much anything you see in an app. Flutter also comes with a bunch of ready-to-use components called *widgets*, which are like building blocks that you can piece together to make your app look and work just the way you want.

One of the coolest things about Flutter is how fast you can see your changes. There's a feature called *Hot Reload*, which instantly updates your app as you make changes to the code. So, you can quickly try out different ideas and see what works best without waiting for your app to rebuild every time.

Flutter is also backed by Google, which means it's constantly improving and getting better with each update. And because it's open source, there's a huge community of developers who share tips, tricks, and ready-made components to help each other out.
So, whether you're interested in creating games, useful tools, or just want to bring your creative ideas to life, Flutter is a fun and powerful tool to explore and experiment with!

Dr Mark Lochrie

You can build Flutter applications using a range of your favourite IDEs, such as, Visual Studio Code, Android Studio, IntelliJ IDEA, Atom etc. For the purpose of today's activity, we will use a browser-based IDE to speed up our development time.

Activity

Today's activity will introduce the basics of Flutter, getting you to build your first Flutter mobile application right within the browser.

- Your PC should be logged in, if it isn't, please ask the tutor to log you in.
- Visit https://flutlab.io/ in Chrome.
- To get started you will need to create an account.
- You can continue with your Google account or provide an email address and password. You may need to verify your account by checking your email, but you can do this up to 24 hours later, to avoid your project being deleted.
- Click "Create Project" -> "Stateless Hello World"

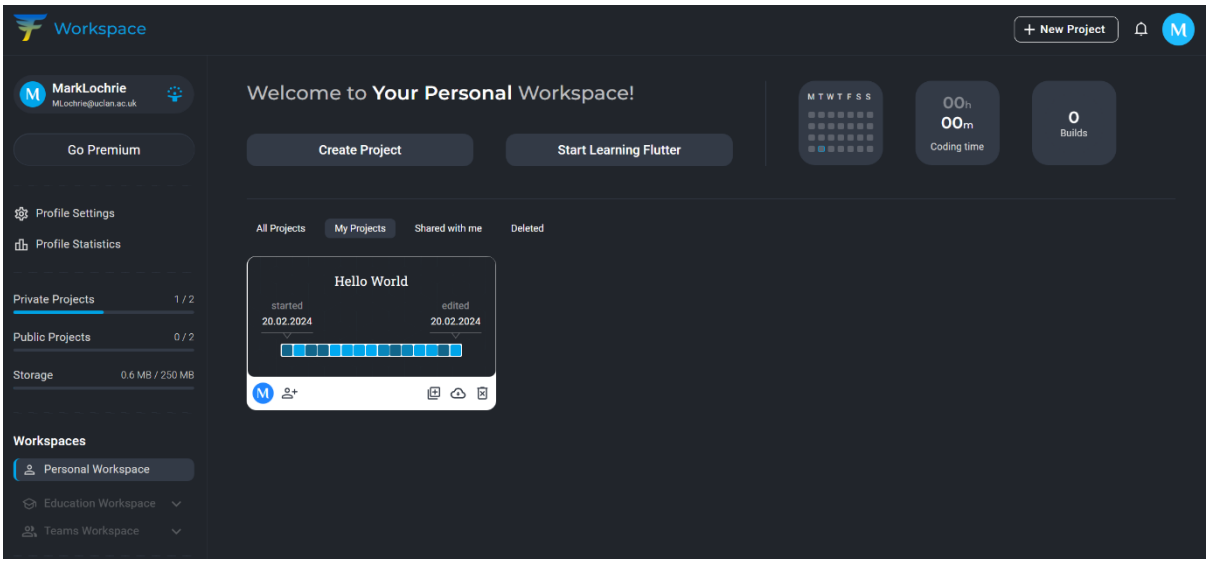


Figure 1. Getting started.

- Provide your project a name. i.e.,
    - Project name: UCLanTasterActivity
- Create.
- Click on the project to enter the editor – you will notice the project has some template code provided for you.

- Before we begin, let's see what the code does and how we run the application.

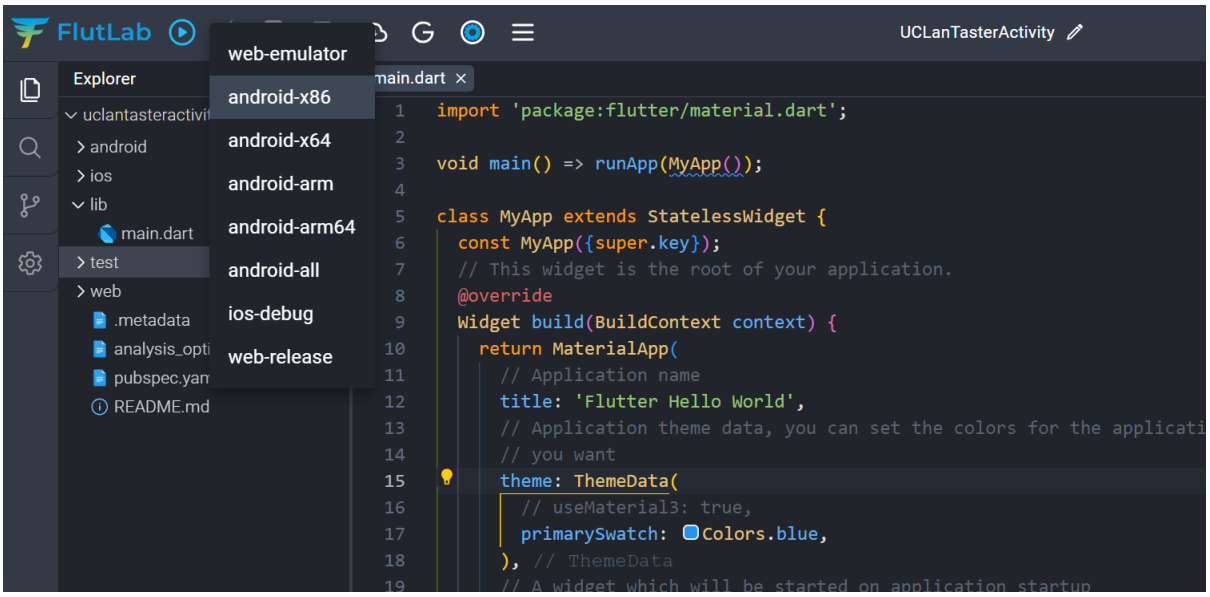


Figure 2. Running the code.

  - Using the toolbar at the top -> click the "*Build project*" icon (play button) and select "web-emulator" to build a Web Application. Note that if you have an Android device you can download the apk to run on your own device. This also applied to iOS.
  - It will take some time to compile the code and create the instance of the Web emulator.

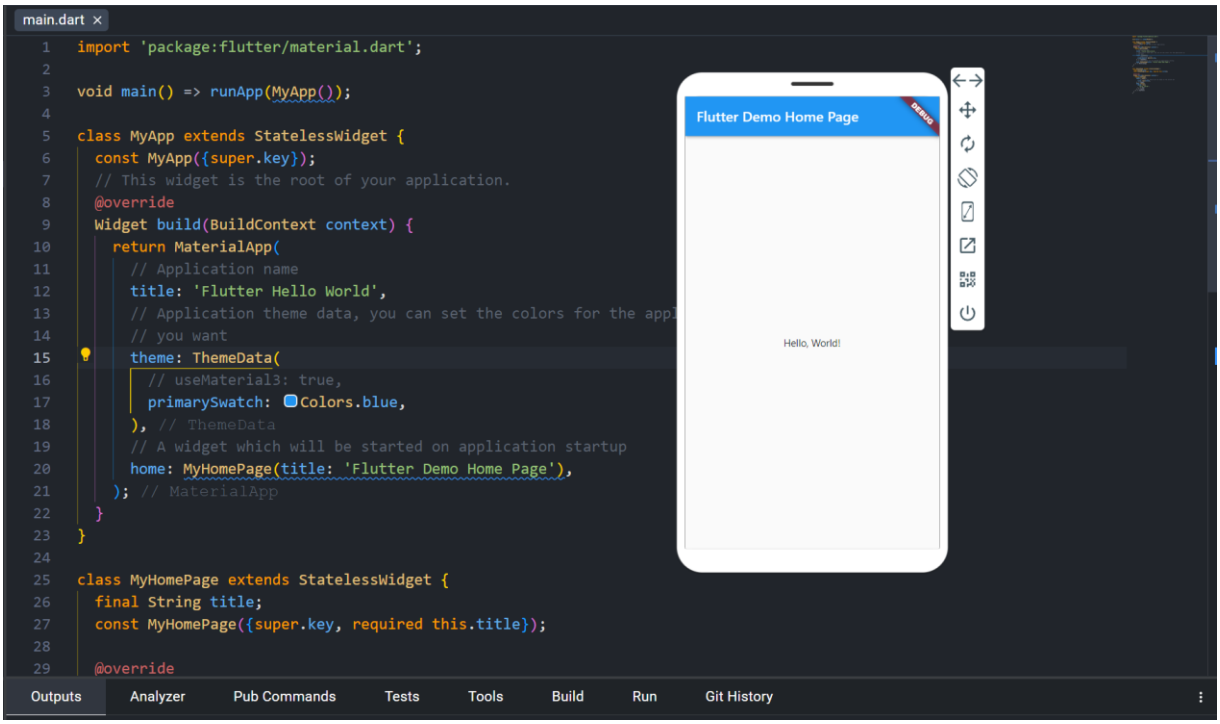


Figure 3. Using the emulator.

- You will notice a mobile device emulator will appear, looking like the above screenshot. Returning to your code, lets remove all the code so we can start to build our application.
-

Rock, Paper, Scissors: Technical Specification

You are to build the rock, paper, scissors game where you will need to think about the logic involved: how the game is won, lost, and drawn.

The player will start by typing their choice, the computer will randomly select theirs and the application will determine who wins!

You will learn about various programming constructs: statements, loops, variables etc.

Before you begin, Flutter uses the same Dart code for the presentation of your application (User Interface, UI) and the business logic.

Step 1: The Foundations

- Import the packages we need to support out development - As we are building an Android application to run as a Web Application, we need to import Google's design

library called *Material* and the *Math* library to provides the functionality for generating random numbers. Which will be used to simulate the computer's choice in the game.

```
import 'dart:math';
import 'package:flutter/material.dart';
```

- The void main() is the entry point of every Dart program. runApp(RockPaperScissorsApp());: This function call starts the Flutter application by running an instance of the RockPaperScissorsApp widget.

```
void main() {
      runApp(RockPaperScissorsApp());
}
```

- Building the Widget for RockPaperScissorsApp require you to inherit from a *StatelessWidget* (this is known as inheritance – a key characteristic in Object Oriented Programming), where you are accessing attributes and methods from the parent class. This is what makes your code more reusable and efficient.

```
class RockPaperScissorsApp extends StatelessWidget {
  const RockPaperScissorsApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rock Paper Scissors',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: RockPaperScissorsScreen(),
    );
  }
}
```

  - Widget build(BuildContext context): This method is required by all widgets and is responsible for building the UI of the widget.
  - return MaterialApp(...): This returns an instance of the MaterialApp widget, which represents the root of the Flutter application. It provides features like navigation and theme management.
  - title: 'Rock Paper Scissors': Sets the title of the application.
  - theme: ThemeData(...): Sets the theme of the application. Here, it sets the primary color theme to blue.
    - See if you can change the colour to your something you prefer.
  - home: RockPaperScissorsScreen(): Sets the home screen of the application to an instance of the RockPaperScissorsScreen widget.
- You have now built the main part of the application to get the UI drawn

```
class RockPaperScissorsScreen extends StatefulWidget {
  const RockPaperScissorsScreen({super.key});
  @override
  RockPaperScissorsScreenState createState() => RockPaperScissorsScreenState();
}
```

- Flutter uses either *Stateless* and *Stateful Widgets*. The difference between the two are, Stateful Widgets retain the State of the application – think about an application where you slide to change a value, the application needs to know the current state (value) and the new state (new value) which become the old state once changed.
- The *Stateful Widget* named RockPaperScissorsScreen. Stateful widgets can change their state during the lifetime of the *widget*.
    - createState(): This method creates and returns the state for the widget.

Step 2: Creating the UI for interaction

```
class RockPaperScissorsScreenState extends State<RockPaperScissorsScreen> {
//logic code here



//logic code here
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Rock Paper Scissors'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'Choose your move:',
              style: TextStyle(fontSize: 20),
            ),
            SizedBox(height: 20),
      //User input buttons here
            SizedBox(height: 20),
            Text(
              'Player: ', //add player choice variable inside the quotes
```

```
            style: TextStyle(fontSize: 18),
          ),
          Text(
            'Computer: ', //add computer choice variable inside the quotes
            style: TextStyle(fontSize: 18),
          ),
          SizedBox(height: 20),
          Text(
            'Result: ', //add result variable inside the quotes
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
        ],
      ),
    ),
  );
 }
}
```

- Run and test your code, you should see a UI with Center Column of Choose your move, player, computer, and result.
- We will now move onto building the business logic.

Step 3: Building the logic

- Lets build a List of three choices as *Strings* and declare three *String* variables to hold the choices. Notice these are declared as *late*. This means a value will be supplied later in the programme.

```
final List<String> choices = ['Rock', 'Paper', 'Scissors'];
late String playerChoice = '';
late String computerChoice = '';
late String result = '';
```

  - final List<String> choices = ['Rock', 'Paper', 'Scissors']: This defines a list of choices available in the game.
  - late String playerChoice = ''This variable holds the player's choice.
  - late String computerChoice = '' This variable holds the computer's choice.

- late String result = '' This variable holds the result of the game.

- Write a function called generateComputerChoice that doesn't return anything but uses the setState() method – which updates the state of the application.

```
void generateComputerChoice() {
    final random = Random();
    final index = random.nextInt(choices.length);
    setState(() {
      computerChoice = choices[index];
    });
  }
```

  - The function generates a random number and chooses this choice from the list of possible actions. See how the two variables are declared as *final* this means the variable cannot change – this is known as an immutable variable – so the value it's assigned will remain throughout the execution.
  - The *setState* method works by notifying the state of the widget has changed and the UI needs to be rebuilt. In this case the *state* changes to the new choice of the computer's selection. Managing the *state* is what makes Flutter efficient in building the UI as it will only update what it needs to rebuild.

The next step is over to you. You will be expected to recall the learning above of writing a function that doesn't return anything (*void*), creating *final* variables and using the *setState* method to update the UI.

- Create a function called determineWinner that doesn't return anything but accepts one parameter of playerChoice of type String.
- Create three final variables (playerIndex, computerIndex, resultMatrix)
- Assign the playerIndex variable to equal the choices.indexOf(playerChoice)
- Assign the computerIndex variable to equal the choices.indexOf(computerChoice)
- Assign the resultMatrix to equal a two-dimensional array [[]] and populate the array with three further arrays with three outcome *Strings for each possible outcome within the game.*

```
final resultMatrix = [
      ['Draw', 'You Lose!', 'You Win!'], //rock
```

```
      ['You Win!', 'Draw', 'You Lose!'], //paper
      ['You Lose!', 'You Win!', 'Draw'], //scissors
];
```

- Inside the setState method set the result to equal resultMatrix[playerIndex][computerIndex];
- Save and run your application. You will notice nothing really changes in terms of the UI. As we need to build the UI now with the buttons to allow for capturing user interactions.

Step 4: Updating the UI with the business logic

- Locate the line where the comment reads '// User input buttons here'
- Paste the following code.

```
Wrap(
              spacing: 10,
              children: choices.map((choice) {
                return ElevatedButton(
                  onPressed: () {
                    setState(() {
                      playerChoice = choice;
                      generateComputerChoice();
                      determineWinner(playerChoice);
                    });
                  },
                 child: Text(choice),
                );
              }).toList(),
            ),
```

- This code creates three buttons based on the choices List, created earlier.
- Each button is of an *ElevatedButton* type (from the Material library) and has an *onPressed* method that sets the *State* to the choice selected (Rock, Paper, Scissors) and performs the generateComputerChoice and determineWinner functions. The label for the button comes from the choice *List*.

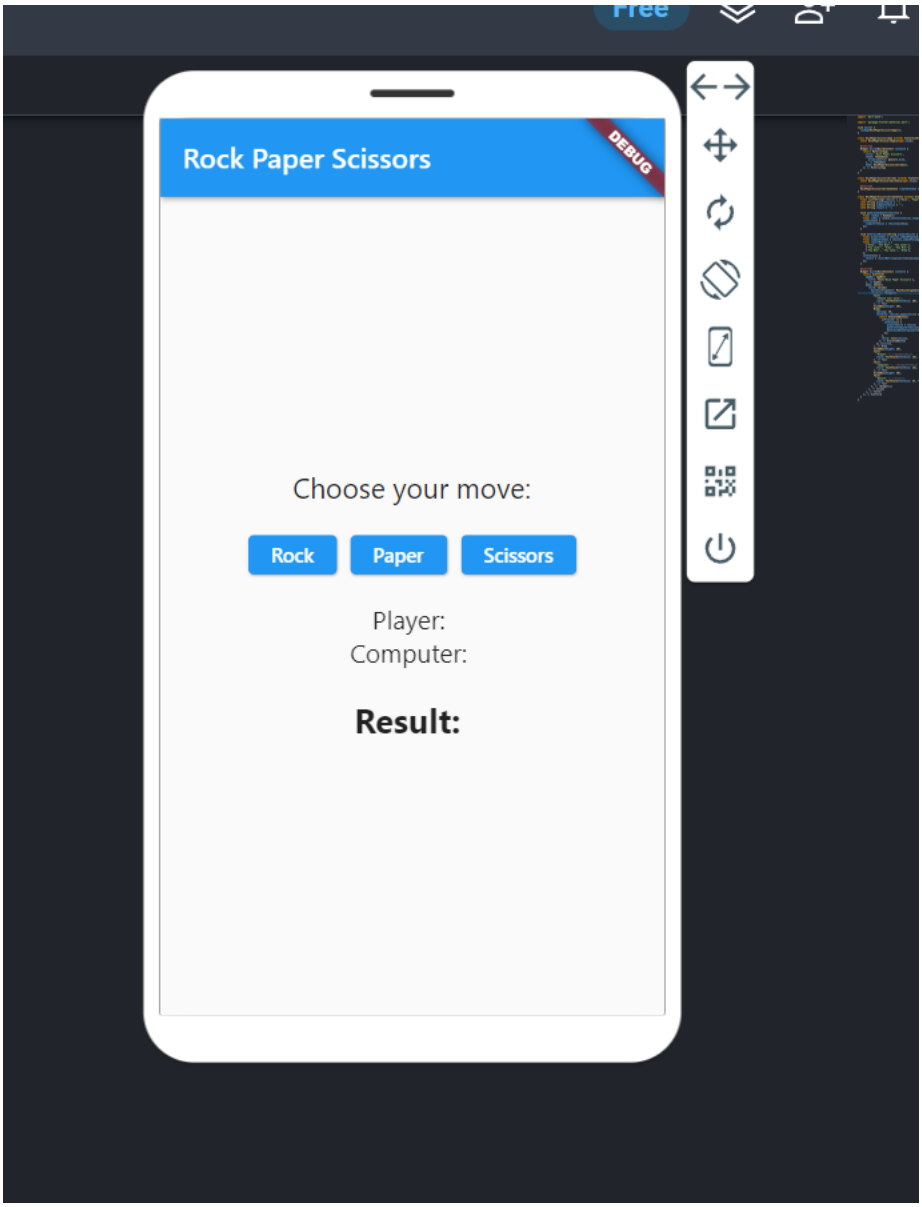


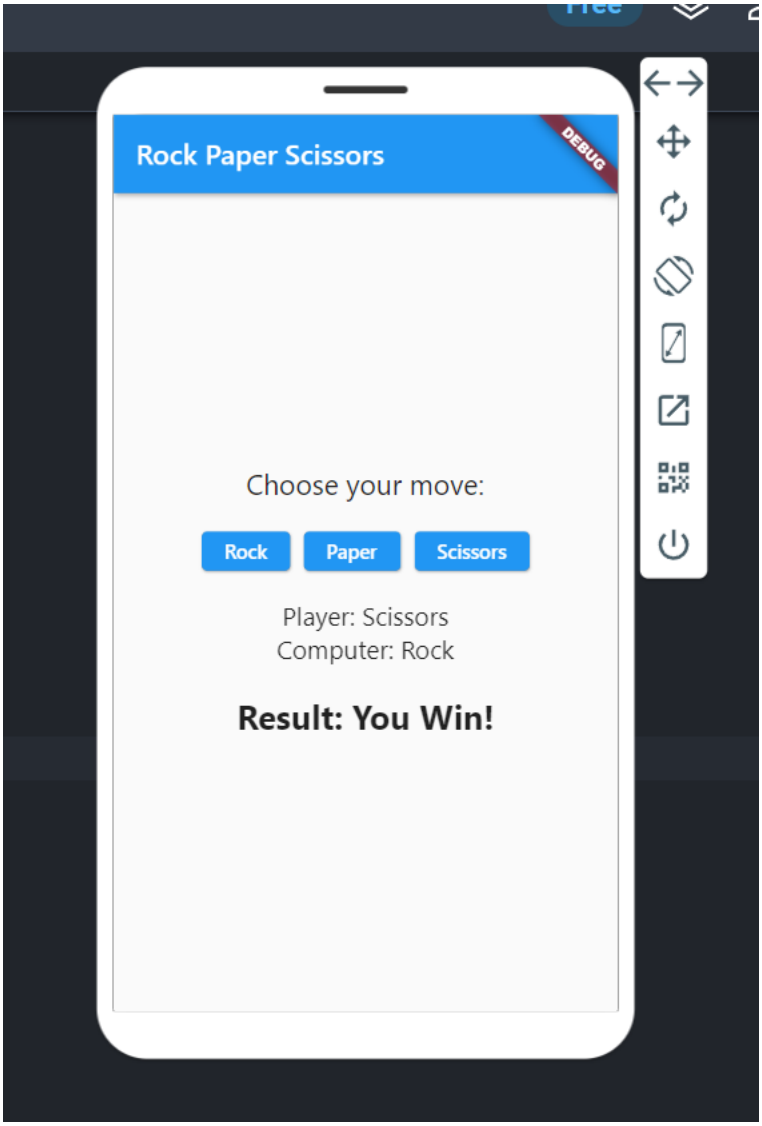
Figure 4. Screenshot of the three buttons added to the UI.

- Save and run your application and you should now see three buttons appear (Rock, Paper, Scissors). Try them out, everything is working, and the variables are being assigned. But we aren't presenting the data back to the UI, which is what we will do next.
- Find the comment '//add player choice variable inside the quotes' and add the playerChoice variable to the *Text Widget* inside the *String*. For this we are going to

use the concatenation ($) feature from Dart that allows us to merge a String with a variable inside the same quotes.

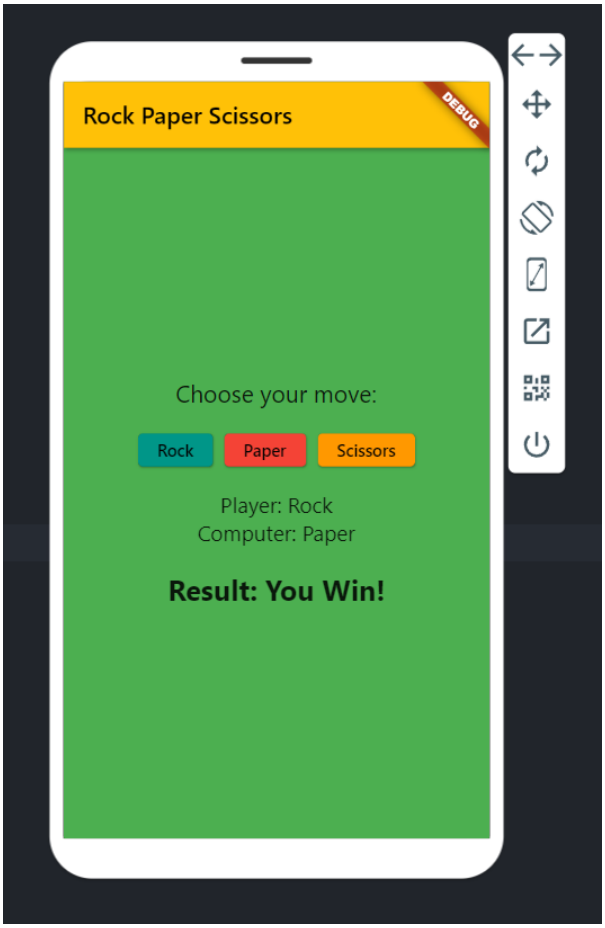
```
'Player: $playerChoice',
```

- Do the same for the computerChoice and result variables in the correct *Text Widgets* – **note** you will need to use the concatenation symbol $ in front of the variable.
- Save and run your application.
- Try out your game and see if you can beat the computer.



Step 5: Extending the game

- Try changing the colour of the buttons (you might need to use the Material API for the button class https://api.flutter.dev/flutter/material/ElevatedButton-class.html)
- Try change the colour of each button.
- Change the background colours of the body of the application to green if the user wins, red if the computer wins and orange if it's a draw



Well done! – if you want to continue your progress in your own time, we'd love to hear from you StudyComputing@uclan.ac.uk

Completed code https://mlochrie.github.io/recruitment/taster/flutter/rock-paper-scissors/main.dart

Code Explanation

Terminology:

Flutter is the framework -> ecosystem that encapsulates Dart (the programming language) and design systems (i.e., Material for Android and Cupertino for Apple) for building cross-platform applications.

Structure:

- MaterialApp: The main app widget that serves as the root of the application. It sets up the app's visual components and routing system.
- Scaffold: A fundamental layout widget that provides a visual frame for your app's screens, typically containing a top bar (AppBar), a central content area (body), and a bottom bar (BottomNavigationBar) (optional).
- Widgets: Reusable building blocks that represent UI elements like buttons, text, images, layouts, and more.
- State Management: Flutter uses a reactive approach to manage UI changes based on data updates. You can use the StatefulWidget class to create widgets that hold state and rebuild themselves when the state changes.

Key Elements:

- Imports: Statements at the beginning that bring in necessary modules and libraries for your code to function.
- Classes: Define the blueprints for your widgets, encapsulating their properties, methods (functions), and lifecycle.
- Methods: Functions associated with a class that perform specific actions or calculations.
- Properties: Attributes of a widget or class that determine its appearance or behaviour.
- Build method: The core method in a StatefulWidget that returns the widget tree, defining the UI structure.
- Once you provide the specific Flutter code you're interested in, I can break it down for you, explaining the meaning of each line, the components it creates, and how they work together to produce the desired UI.

Starting out in Flutter

- Start with the Basics: Get familiar with the fundamentals of Flutter development, such as widgets, layout, and state management.
- Use Official Documentation: Flutter's documentation https://docs.flutter.dev is a comprehensive resource for understanding concepts, APIs, and best practices. Watch out for the various Dart (programming language) versions. In version 3, Dart / Flutter utilises Null Safety.
- Experiment and Explore: Create small, self-contained projects to test your understanding and get hands-on experience. Look at code examples and tutorials to see how different features are implemented.

- Seek Help: If you encounter specific challenges, don't hesitate to seek assistance from online communities, forums, or stack overflow.
- Practice: The more you work with Flutter, the better you'll grasp its nuances and become proficient in writing and understanding its code.

Key Points for the activity:

- State management is used to update the UI dynamically based on player interactions and game state changes.
- Switch statements are used to set button background colours and handle game result logic.
- The code leverages the built-in Random class for computer choice generation.

V.1.1

Code

```dart
import 'dart:math';
import 'package:flutter/material.dart';



void main() {
  runApp(RockPaperScissorsApp());
}


class RockPaperScissorsApp extends StatelessWidget {
  const RockPaperScissorsApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Rock Paper Scissors',
      theme: ThemeData(
        primarySwatch: Colors.amber,
      ),
      home: RockPaperScissorsScreen(),
    );
  }
}
```

Explanation

Imports:

- dart:math: Provides the Random class for generating random numbers.
- package:flutter/material.dart: Imports the core Flutter widgets and functionalities.

main() function:

- Starts the app by calling runApp() with an instance of RockPaperScissorsApp. Every Flutter application calls the runApp from within the main().

RockPaperScissorsApp class:

- A StatelessWidget representing the overall app. StatelssWidgets are those that don't retain state. For example they don't have the ability to change like a counter application where you have a button the counter increases from 1, 2 … x the counter uses a StatefulWidget to remember the previous number.
- Keys are used in widget management and optimisation of painting the UI – it is an identifier to locate the Widget in the Widget tree. When the widget tree needs to be rebuilt (due to state changes), Flutter uses keys to compare the new widget tree with the previous one. If a widget has the same key in both the old and new tree, Flutter can efficiently reuse the widget's state, preventing unnecessary widget rebuilds and maintaining performance.

University of Central Lancashire
UCLan

V.1.1

```
class RockPaperScissorsScreen extends StatefulWidget {
  const RockPaperScissorsScreen({super.key});

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

class RockPaperScissorsScreenState extends
State<RockPaperScissorsScreen> {
  //extra work
  Color appBackgroundColor = Colors.white;
  //extra work
```

- Every Widget inherits from a Stateful or Stateless Widget and by doing do you need to override the main build method.
- Builds a MaterialApp that sets the app title and theme (amber primary colour). The MaterialApp is Google's Design System for building mobile application.
- Sets the home screen of the app to RockPaperScissorsScreen.

RockPaperScissorsScreen class:

- A StatefulWidget responsible for the game's state, such as player/computer choices and result.
- Creates a state object (RockPaperScissorsScreenState) that manages the state.

RockPaperScissorsScreenState class:

- Contains the game logic and UI build.
- It extends the State from RockPaperScissorsScreen

State Variables:

- appBackgroundColor: Tracks the background color based on the game result (green for win, red for lose, orange for draw).

V.1.1

```
  final List<String> choices = ['Rock', 'Paper',
'Scissors'];
  final List<Color> appBgColor = [Colors.green,
Colors.red, Colors.orange];

  late String playerChoice = '';
  late String computerChoice = '';
  late String result = '';


 void generateComputerChoice() {
    final random = Random();
    final index = random.nextInt(choices.length);
    setState(() {
      computerChoice = choices[index];
    });
  }

  void determineWinner(String playerChoice) {
    final playerIndex = choices.indexOf(playerChoice);
    final computerIndex =
choices.indexOf(computerChoice);
    final resultMatrix = [
      ['Draw', 'You Lose!', 'You Win
!'],
```

- choices: A list of possible game choices ("Rock", "Paper", "Scissors").
- appBgColor: A list of background colors corresponding to each choice.
- playerChoice: Stores the player's selected choice.
- computerChoice: Stores the computer's randomly generated choice.
- result: Holds the game result ("Draw", "You Win!", "You Lose!").

Methods:

- generateComputerChoice(): Generates a random choice for the computer using Random.nextInt() and updates computerChoice state.

- determineWinner(String playerChoice): Uses a predefined resultMatrix to determine the winner based on player and computer choices.

University of Central Lancashire
UCLan

V.1.1

```
    ['You Win!', 'Draw', 'You Lose!'],
    ['You Lose!', 'You Win!', 'Draw'],
  ];
  setState(() {
    result = resultMatrix[playerIndex][computerIndex];
    //extra work
    switch (result) {
      case 'You Win!':
        appBackgroundColor = Colors.green;
        break;
      case 'You Lose!':
        appBackgroundColor = Colors.red;
        break;
      case 'Draw':
        appBackgroundColor = Colors.orange;
        break;
    }
    //extra work
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: appBackgroundColor,
```

- Updates the result and appBackgroundColor states. Using the setState method which will automatically update any variable that is printed on screen to reflect this change.

- build(BuildContext context) method:
- Builds a Scaffold with the following structure:

University of Central Lancashire
UCLan

V.1.1

```
appBar: AppBar(
  title: Text('Rock Paper Scissors'),
),
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Text(
        'Choose your move:',
        style: TextStyle(fontSize: 20),
      ),
      SizedBox(height: 20),
      Wrap(
        spacing: 10,
        children: choices.map((choice) {
          //extra work
          Color bgColor = Colors.black;
          switch (choice) {
            case 'Rock':
              bgColor = Colors.teal;
              break;
            case 'Paper':
              bgColor = Colors.red;
              break;
            case 'Scissors':
```

- AppBar: Displays the app title ("Rock Paper Scissors"). The App bar is the banner at the top of most mobile applications, typically you have additional buttons in the app bar for navigation and confirming actions.
- body: A Center widget containing a Column layout. Which places all content in the centre of the app view.
- The column widget can accept multiple children i.e., you can have more than one Widget contained with that Widget. Notice “children” unlike the Centre Widget which can only accept one “child”.
- Text prompting the user to choose a move.
- A Wrap widget displaying choice buttons https://api.flutter.dev/flutter/widgets/Wrap-class.html A widget that displays its children in multiple horizontal or vertical runs. A Wrap lays out each child and attempts to place the child adjacent to the previous child in the main axis, given by direction, leaving spacing space in between. If there is not enough space to fit the child, Wrap creates a new run adjacent to the existing children in the cross axis.
- Iterates through choices list to create ElevatedButton widgets for each option.
- Each button has a background colour based on its choice using a switch statement.

```
            bgColor = Colors.orange;
            break;
        }
        //extra work
        return ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: bgColor,
          ),
          onPressed: () {
            setState(() {
              playerChoice = choice;
              generateComputerChoice();
              determineWinner(playerChoice);
            });
          },
          child: Text(choice),
        );
      }).toList(),
    ),
    SizedBox(height: 20),
    Text(
      'Player: $playerChoice',
      style: TextStyle(fontSize: 18),
    ),
    Text(
```

- Clicking a button sets playerChoice, generates computerChoice, and determines the result using related methods. Achieves this by updating the setState method.

- The $ symbol is used to concatenate the String with the variable.
- Text displaying the player's choices.

V.1.1

```
              'Computer: $computerChoice',
              style: TextStyle(fontSize: 18),
            ),
            SizedBox(height: 20),
            Text(
              'Result: $result',
              style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
            ),
          ],
        ),
      ),
    );
  }
}
```

- Text displaying the computer's choices.
- Text displaying the game result with bold font.