Decision Tables (Testing)
(Currency & Object Detection Class)

brightness_test.dart

```dart
// PASSED
import 'package:flutter_test/flutter_test.dart';

class _CurrencyDenominationState {
  double calculateAverageBrightness(List<int>
brightnessValues) {
    double totalBrightness = 0;

    for (int brightness in brightnessValues) {
      totalBrightness += brightness;
    }

    return totalBrightness / brightnessValues.length;
  }
}

void main() {
  group('CurrencyDenominationStateTests', () {
    late _CurrencyDenominationState state;

    setUp(() {
      state = _CurrencyDenominationState();
    });

    test('calculateAverageBrightness - Average brightness
calculation', () {
      // Create test data with known brightness values
      final testBrightnessValues = [10, 20, 30, 40, 50];

      // Call the method
      final averageBrightness =

state.calculateAverageBrightness(testBrightnessValues);

      // Verify the result based on the known values
      expect(
          averageBrightness, equals(30.0)); // (10 + 20 +
30 + 40 + 50) / 5 = 30
    });

    // Add more test cases as needed

    tearDown(() {
      // Clean up any resources or reset global state
    });
  });
}
```

Decision Table

Test Cases	**Input Data**	**Expected Output**
Case 1	Brightness values: [10, 20, 30, 40, 50]	Average brightness: 30.0
Case 2	Brightness values: [0, 0, 0, 0, 0]	Average brightness: 0.0

Case 3	Brightness values: [255, 255, 255, 255]	Average brightness: 255.0
Case 4	Empty brightness values array	Average brightness: Error (undefined behavior)
Case 5	[100]	100.0
Case 6	Large Values	Average
Case 7	Negative Values	Average

Explanation:
Case 1: Testing with a typical set of brightness values.
Case 2: Testing with all values being zero.
Case 3: Testing with all values being maximum (255 for byte).
Case 4: Testing with an empty array, which could result in undefined behavior.
Case 5: Testing with a single value.
Case 6: Testing with large values to check overflow.
Case 7: Testing with negative values.

currency_camera_disposed_test.dart

```
// PASSED
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import
'package:sample/Screens/currencyDenomination/currencyDeno
minations.dart';

void main() {
  group('CurrencyDenominationWidgetTests', () {
    late List<CameraDescription> cameras;

    setUp(() {
      // Initialize any necessary data or dependencies

      cameras = [
        CameraDescription(
          name: "0",
          lensDirection: CameraLensDirection.front,
          sensorOrientation: 90,
        ),
      ];
    });

    testWidgets('Widget disposes correctly',
(WidgetTester tester) async {
      // Build our widget and trigger a frame.
      await tester.pumpWidget(
        MaterialApp(
          home: CurrencyDenomination(cameras: cameras),
        ),
      );

      // Wait for all asynchronous operations to
complete.
      await tester.pumpAndSettle();

      // Ensure that the camera stream is initially
running
      expect(find.byType(CameraPreview), findsOneWidget);

      // Dispose the widget
      await tester.pumpWidget(Container()); // Replace
with a new widget

      // Wait for all asynchronous operations to
complete.
      await tester.pumpAndSettle();

      // Ensure that the camera stream is stopped after
disposal
      expect(find.byType(CameraPreview), findsNothing);
    });

    // Add more tests as needed for various scenarios

    tearDown(() {
      // Clean up any resources or reset global state
    });
  });
}
```

Decision Table

Test Case	Input Data	Expected Output
Case 1	Widget Disposal Camera	stream stops after disposal
Case 2	Device Rotation Widget	handles device rotation correctly

Case 3	Camera Initialization	Widget initializes the camera stream correctly
Case 4	Recognition Logic	Widget correctly recognizes currency denominations
Case 5	User Interaction	Widget handles user interactions as expected
Case 6	Error Handling	Widget gracefully handles errors or unexpected situations

Explanation:

Widget Disposal (Case 1): This scenario tests if the widget disposes correctly and whether the camera stream stops after disposal.

Device Rotation (Case 2): Test how the widget handles changes in device orientation, ensuring it adapts and displays correctly.

Camera Initialization (Case 3): Ensure the widget initializes the camera stream correctly and starts streaming.

Recognition Logic (Case 4): If there's logic related to recognizing currency denominations, ensure the widget correctly identifies different denominations.

User Interaction (Case 5): If the widget involves user interactions, test those interactions, such as tapping, swiping, etc.

Error Handling (Case 6): Test how the widget handles errors or unexpected situations, ensuring it provides a graceful user experience.

currency_init_test.dart

```
// PASSED
import 'package:camera/camera.dart';
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import
'package:sample/Screens/currencyDenomination/currencyDeno
minations.dart';

void main() {
  group('CurrencyDenominationWidgetTests', () {
    late List<CameraDescription> cameras;

    setUp(() {
      // Initialize any necessary data or dependencies

      cameras = [
        CameraDescription(
            name: "0",
            lensDirection: CameraLensDirection.front,
            sensorOrientation: 90)
      ]; // Initialize with appropriate CameraDescription
objects
    });

    testWidgets('Widget initializes correctly',
(WidgetTester tester) async {
      // Build our widget and trigger a frame.
      await tester.pumpWidget(
        MaterialApp(
          home: CurrencyDenomination(cameras: cameras),
        ),
      );

      await tester.pumpAndSettle();

      // Verify that the widget has the expected initial
state
      expect(find.text('DETECTED'), findsNothing);
      expect(find.byType(CameraPreview), findsOneWidget);
      expect(find.text('NO'), findsOneWidget);
      expect(find.text('0.0'), findsOneWidget);
    });

    // Add more tests as needed for various scenarios

    tearDown(() {
      // Clean up any resources or reset global state
    });
  });
}
```

Decision Table

Test Case	**Input Data**	**Expected Output**
Case 1	Initial Widget Build	Camera stream is visible, 'DETECTED' is not displayed, 'NO' and '0.0' are displayed.

Case 2	Camera Initialization	Widget initializes the camera stream correctly.
Case 3	No Detection (Initial State)	'DETECTED' is not displayed, and initial labels and confidence values are displayed.

<u>Explanation:</u>

Initial Widget Build (Case 1): Ensure that the widget, upon the initial build, displays the camera stream and the initial state of labels and confidence values.

Camera Initialization (Case 2): Verify that the widget initializes the camera stream correctly, ensuring it is visible and ready.

No Detection (Initial State) (Case 3): Check that 'DETECTED' is not displayed initially, and the default labels ('NO') and confidence value ('0.0') are present.

<u>Overview</u>

<u>Test Cases</u>	**<u>Passed</u>**	**<u>Not Passed</u>**
brightness_test.dart	✅	
currency_camera_disposed_test.dart	✅	
currency_init_test.dart	✅	
backbutton_test.dart		❌

mocking_note_test.dart		❌
valid_note_test.dart		❌

Note: mocking_note_test.dart and **valid_note_test.dart** failed because they were trying to mock the model functionality using computational resources and in some cases used laptop's camera for detection.

But, as the model files are with .tflite extension they are made for mobile cameras hence it always failed on laptop's testing environment.