LAB 3

QR CODE

May 7, 2020

Name:Chenyu Yang
Student ID: 517030910386
Class: F1703015

Contents

1 INTRODUCTION

1.1 Bachground

A barcode is an optical machine-readable representation of data relating to the object to which it is attached. Barcodes originally were scanned by special optical scanners called barcode readers. Later applications software became available for devices that could read images, such as smartphones with cameras.

Nowadays the most applied barcode is QR code (abbreviated from Quick Response Code), integrated in applications like Wechat, Paypal, Alipay etc. A QR code consists of black modules (square dots) arranged in a square grid on a white background, which can be read by an imaging device (such as a camera, scanner, etc.) and processed using Reed-Solomon error correction until the image can be appropriately interpreted. The required data are then extracted from patterns that are present in both horizontal and vertical components of the image.

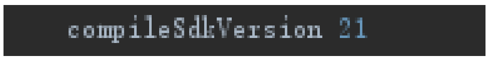
1.2 Environment

1. OpenJDK 64-bit;
2. Android 9.0 (pie);
3. Android Studio 3.3.2;
4. MI 6; (for test)

2 PROCEDURE

2.1 Preparation

The Project fails building at first with infomation ***java.lang.illegalstateexception: failed to find target with hash string 'android-21'***. I reviews the *build.gradle* file, which contains the required SDK information of this project.



```
compileSdkVersion 21
```

Figure 1: build.gradle

As it shows, the project requires SDK version 21 and our default setting is ver.26. After installing Android SDK ver.21 by following the notice of Android Studio, it works successfully.

2.2 Encoder

In this part, we translate the string into a QR code. The details are a little complicated but to realize it is rather easy. The class *MultiFormatWriter* can be directly used to translate it into a matrix. Each element of this matrix corresponds to a region in the QR code. And we just need to paint some certain region to black.

```java
String contentString = textContent.getText().toString();
if (!contentString.equals("")) {
  BitMatrix matrix = new MultiFormatWriter().encode(contentString,
  BarcodeFormat.QR_CODE, 300, 300);
  int width = matrix.getWidth();
  int height = matrix.getHeight();
  int[] pixels = new int[width * height];

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      if (matrix.get(x, y))
        pixels[y * width + x] = Color.BLACK;
    }
  }
  /*
   ......
  */
}
```

2.3 Decoder

The decoder needs to firstly open the camera, then to check each flame until finding a QR code. Finally it change the code to a string back. The usage of camera is a keypoint, however, it's not related to this project. So we only discuss the process from QR code to string. As it shown below, we mainly use three classes: *PlanarYUVLuminanceSource HybridBinarizer* and *QRCodeReader*.

1. *PlanarYUVLuminanceSource* is to abstract the bitmaps on different platforms and realize a standard interface for requesting the brightness value of gray level.

2. *HybridBinarizer* provides a set of methods for converting luminance data into one bit data. It can only receives a PlanarYUVLuminanceSource object.

3. *QRCodeReader* Finally, we use *QRCodeReader.decode()* to translate the bitmap back to the string.

At last, we stop the camera and put the string we get on the screen.

```java
PlanarYUVLuminanceSource source = new PlanarYUVLuminanceSource(
data, previewWidth, previewHeight, 0, 0, previewWidth,
previewHeight, false);
BinaryBitmap bitmap = new BinaryBitmap(new HybridBinarizer(source));
Reader reader = new QRCodeReader();

try {
    Result result = reader.decode(bitmap);
    String text = result.getText();

    Intent intent = new Intent();
    Bundle bundle = new Bundle();
    bundle.putString("result", result.toString());
    intent.putExtras(bundle);
    setResult(RESULT_OK, intent);
    finish();
  } catch (Exception e) {
    e.printStackTrace();
  }
  /*
  ......
  */
```

3 Result

3.1 Encoder

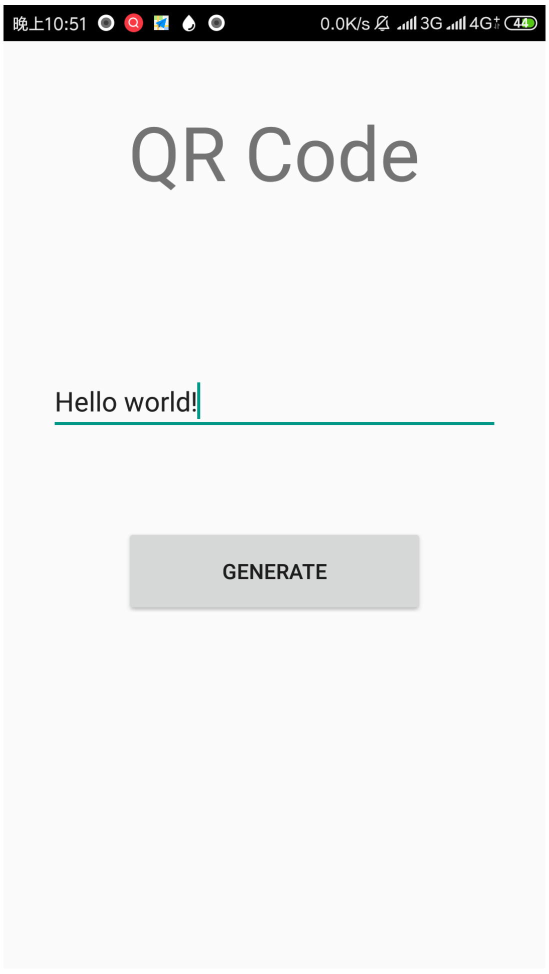


Figure 2: Input string

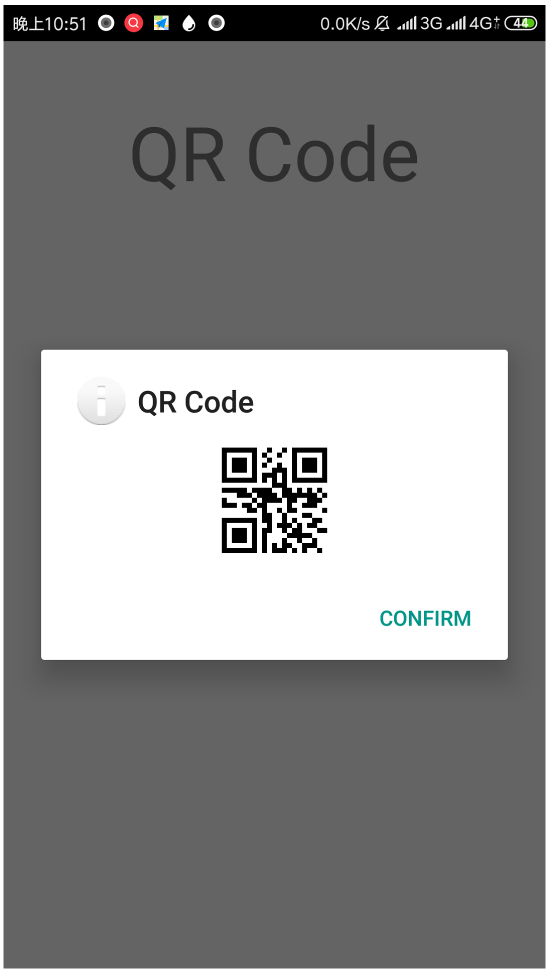
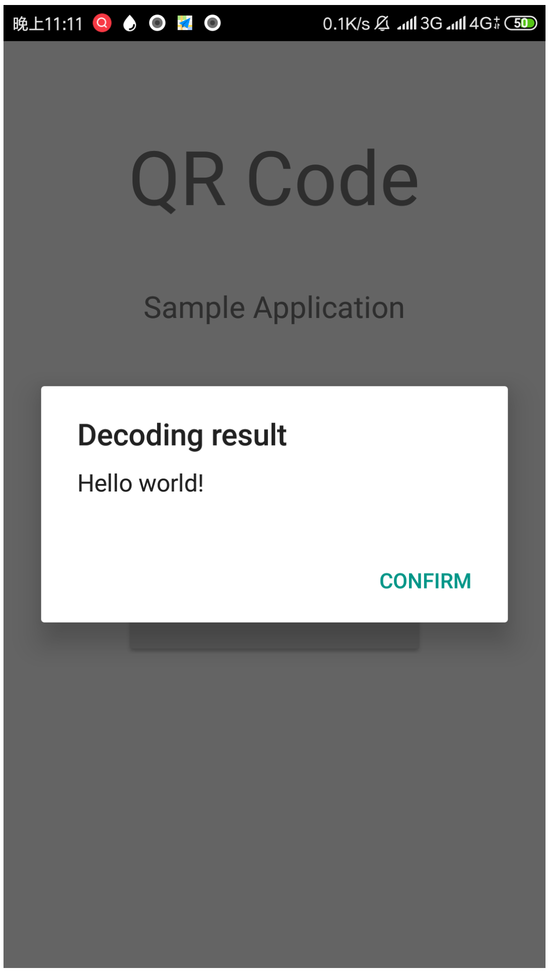


Figure 3: QR code

3.2 Decoder



Decoding result

Hello world!

Figure 4: Decoder result

4 Complete Code

`https://github.com/Achronferry/SJTU-EE447-Lab/tree/master`