# 手势识别绑定快捷键

## 基于

### 原理简介

使用mediapipe识别手部21个点,

使用tensorflow训练模型,根据21个点分出给定的手势

使用opencv-python提供的窗口展示画面.

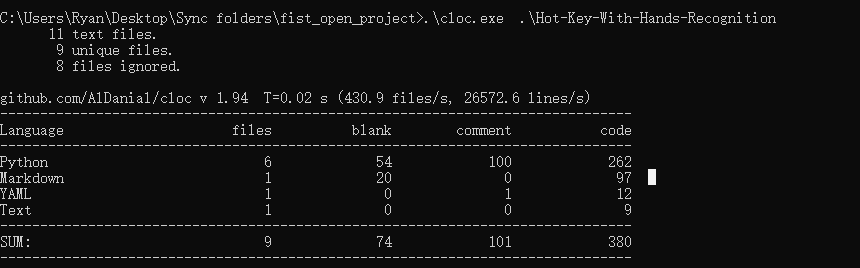
识别出手势后使用keyboard按照设定自动按下指定的快捷键

使用PyQt5制作GUI,用户可通过桌面应用程序控制识别开启,关闭.

使用pyyaml读取保存的参数文件.

最后使用auto-py-to-exe操作pyinstaller打包文件

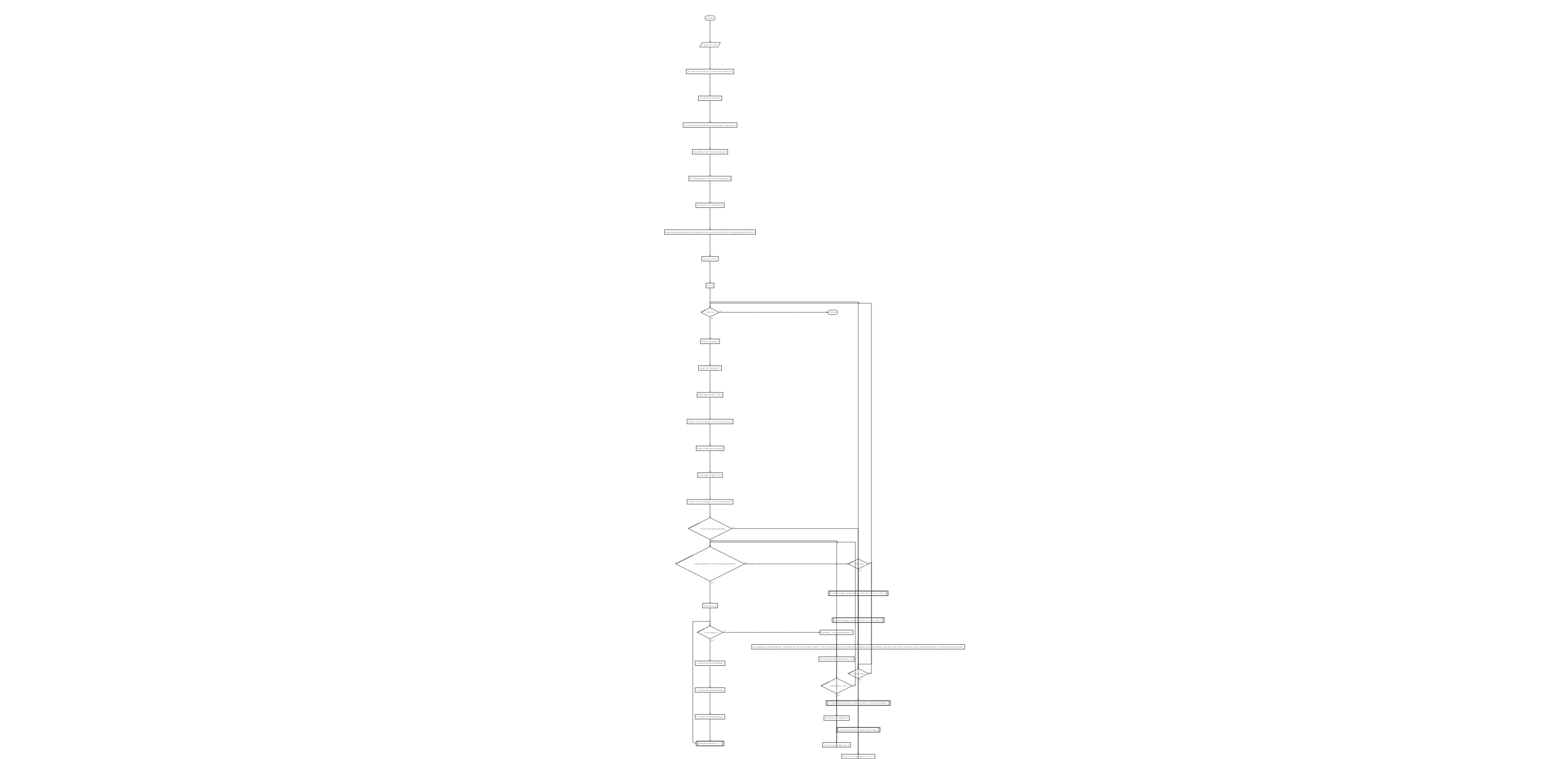
## 代码统计



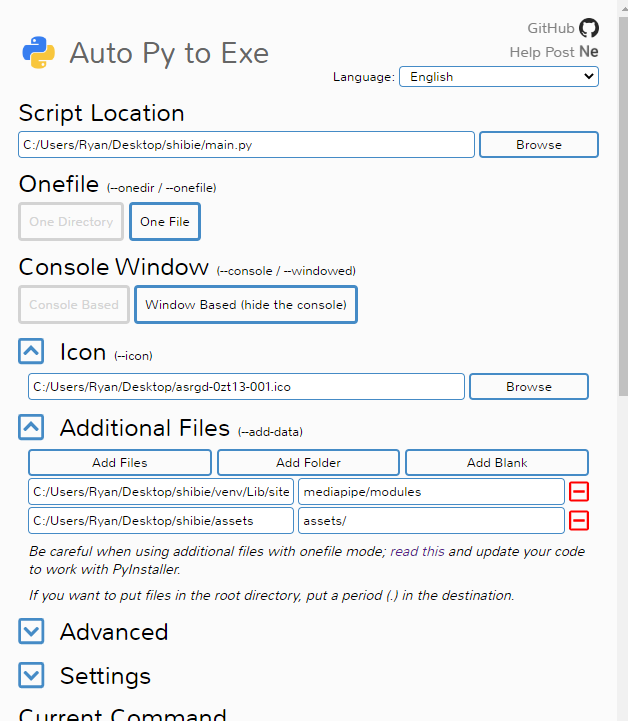
## 简介

这是一款基于手势识别绑定快捷键的应用.

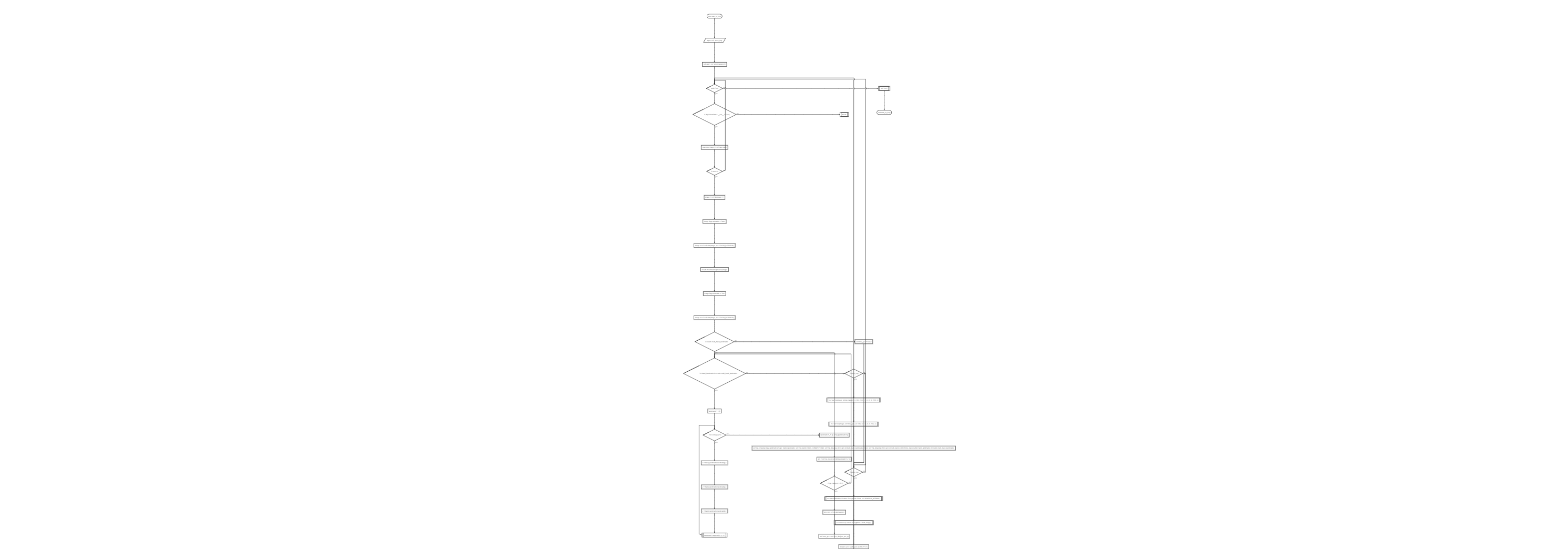
他可以识别出你的手势,然后根据识别的



## 打包







## 简介

### 目前最多可以识别出这些手势:



image-20221122185705967

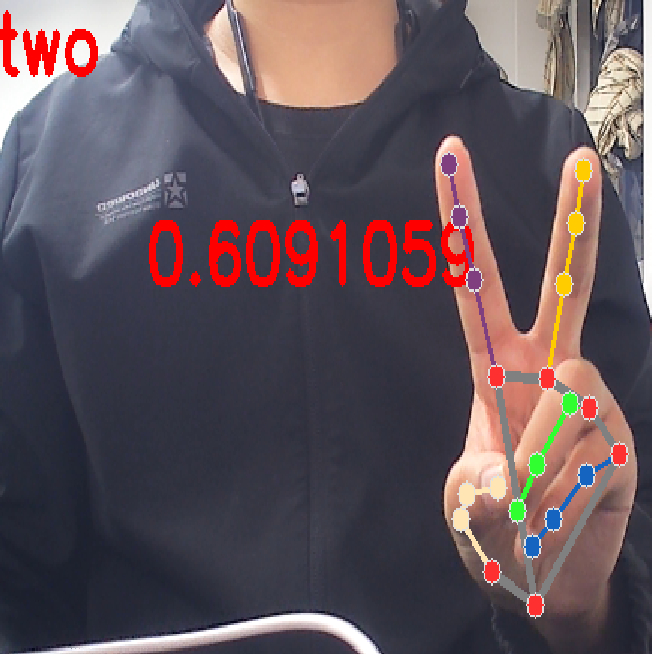


image-20221122185736690



image-20221122185754782

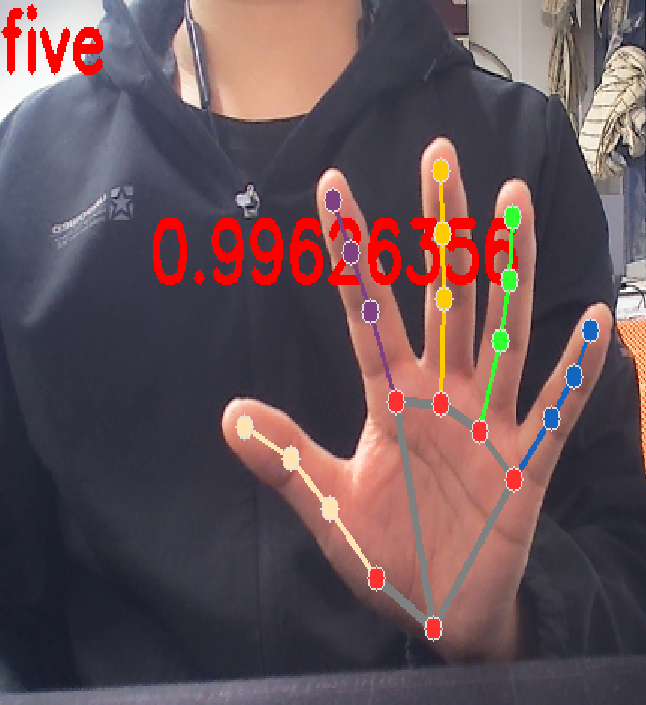


image-20221122185837820



image-20221122185858200



image-20221122185916507



image-20221122185934707

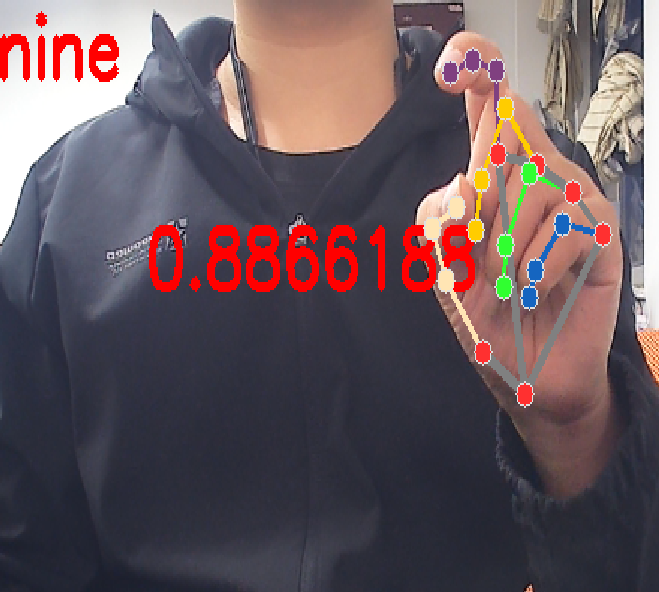


image-20221122185952538



image-20221122190004002

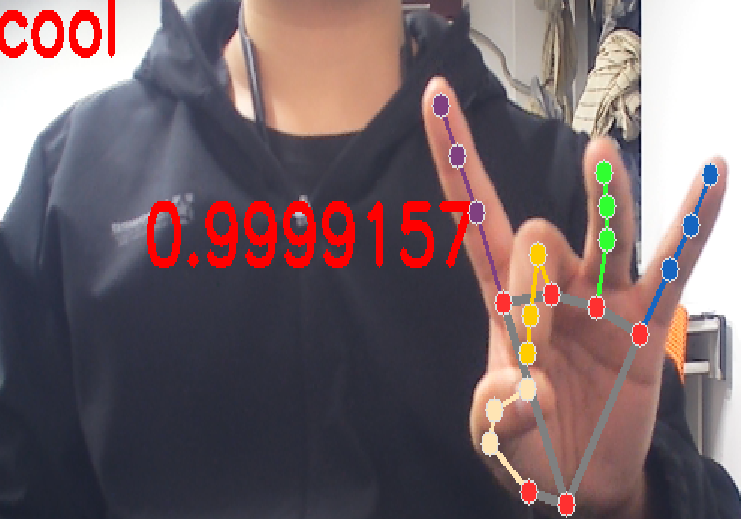


image-20221122190026476

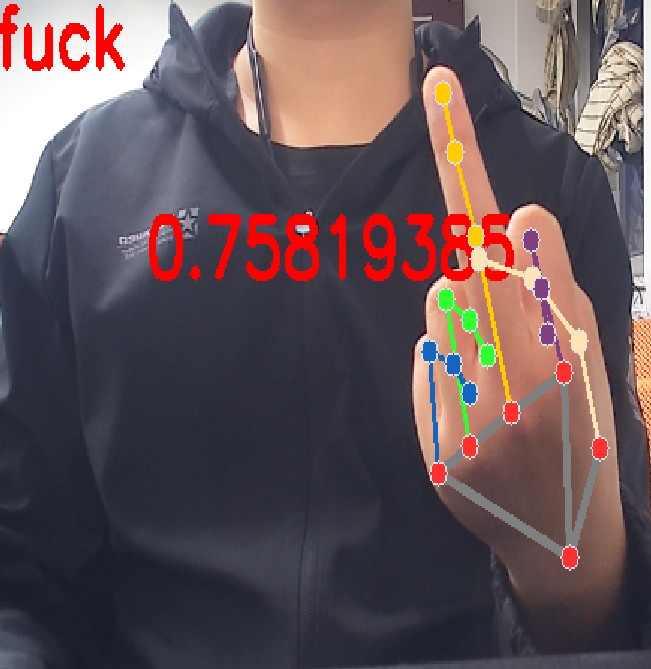


image-20221122190039590

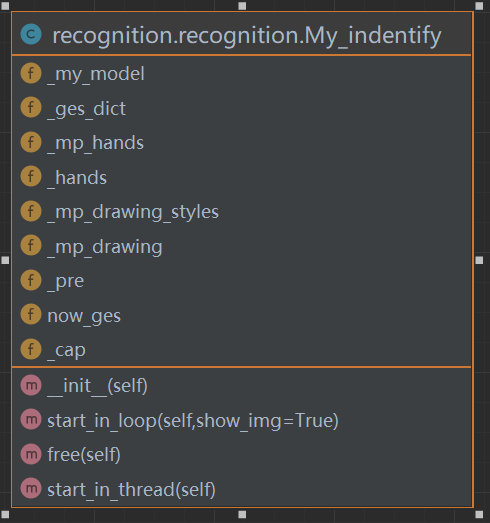
## 函数介绍

#### 类

其中now\_ges是共享给外部的变量,其它的是函数内部使用的变量

start\_in\_loop会在死循环中调用识别算法,直到全局控制位为False后退出.

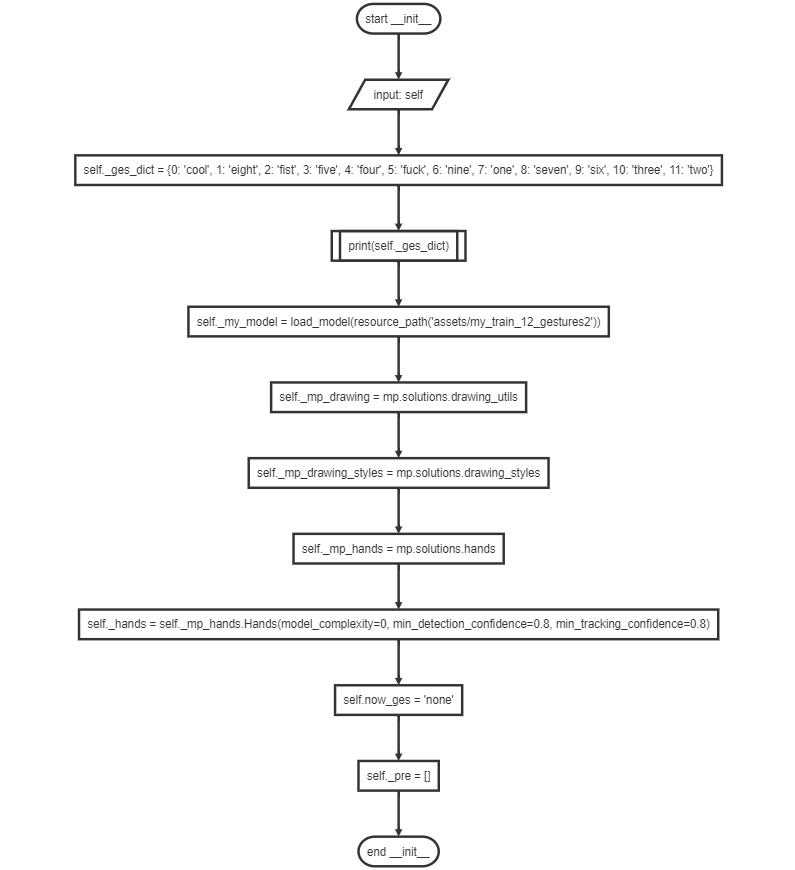
start\_in\_thread是共享给外部的调用方法,每次调用会新开启一个线程运行start\_in\_loop方法.



#### 初始化函数

def \_\_init\_\_(self):

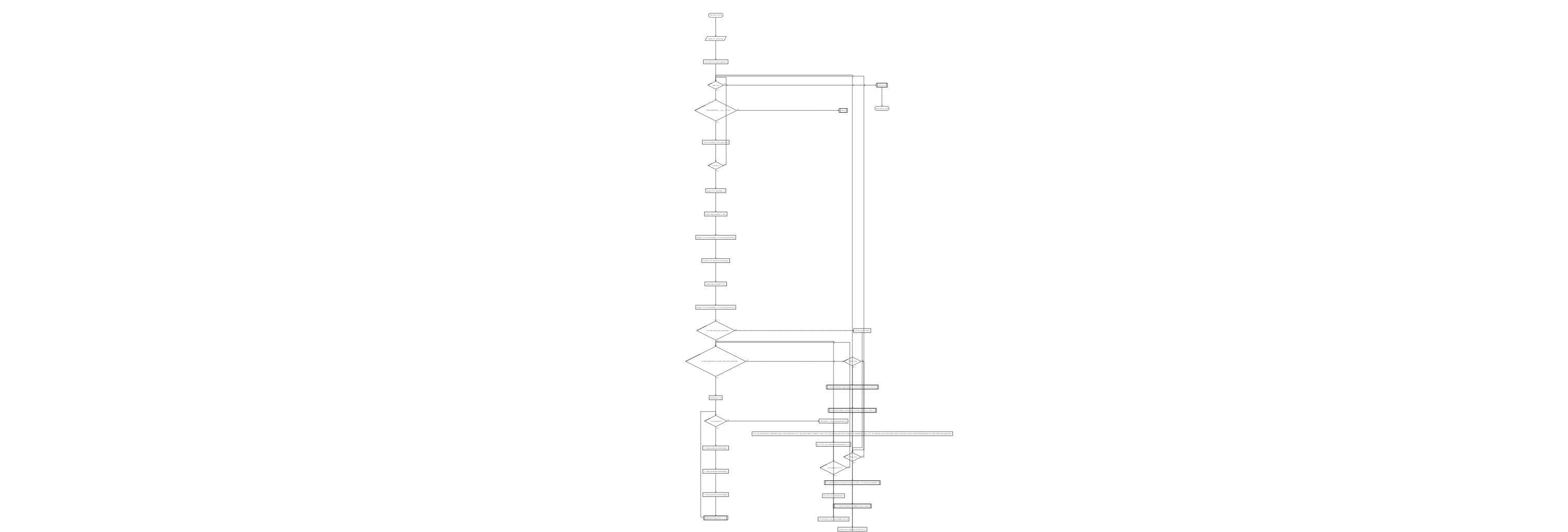
def \_\_init\_\_(self):
  
 # self.ges\_dict = np.load('assets/num\_to\_ss.npy', allow\_pickle=True) #识别出来的手势是数字,需要用字典把序号转化成字符串,加载保存的对照表
  
 # self.ges\_dict = self.ges\_dict.tolist()
  
 self.\_ges\_dict = {0: 'cool', 1: 'eight', 2: 'fist', 3: 'five',
  
 4: 'four', 5: 'fuck', 6: 'nine', 7: 'one', 8: 'seven',
  
 9: 'six', 10: 'three', 11: 'two'}
  
  
 print(self.\_ges\_dict)
  
 self.\_my\_model = load\_model(resource\_path('assets/my\_train\_12\_gestures2')) # 加载训练好的tensorflow模型
  
 self.\_mp\_drawing = mp.solutions.drawing\_utils # 创建一个绘图工具
  
 self.\_mp\_drawing\_styles = mp.solutions.drawing\_styles # 创建一个绘图样式
  
 self.\_mp\_hands = mp.solutions.hands # 创建mediapipe框架读取特征点的初步工具，需要输入一个视频流，后通过自定义的tenslrflow神经网络获得手势预测值
  
 self.\_hands = self.\_mp\_hands.Hands(model\_complexity=0, min\_detection\_confidence=0.8, min\_tracking\_confidence=0.8)
  
 self.now\_ges = 'none' #用于共享出去,给其他程序获得手势的标志位
  
 self.\_pre = [] # tensorflow模型预测输出为一个包含12个权值的list



#### 开启大循环函数

def start\_in\_loop(self,show\_img=True):

def start\_in\_loop(self,show\_img=True):
  
 self.\_cap=cv2.VideoCapture(0)
  
 while True:
  
 if MyGlobalStates.\_\_run\_\_ is True:
  
 # time.sleep(0.05)
  
 # 从摄像头读一帧画面
  
 success, image = self.\_cap.read()
  
 if success:
  
 image = cv2.flip(image, 1)
  
 # 将图片设置为不可写，提升性能
  
 image.flags.writeable = False
  
 # 转换为RGB格式
  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)
  
 # 调用mediapipe框架读取特征点，特征点为1个对象，保存在results中
  
 results = self.\_hands.process(image)
  
 # 打开图像可写的开关，将颜色由RGB转化成BGR,加速
  
 image.flags.writeable = True
  
 image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)
  
 # 获取手势预测值
  
 # 遍历results变量中的结果，将结果转化为nmupy数组格式输入神经网络进行预测
  
  
 if results.multi\_hand\_landmarks:
  
 # print(len(results.multi\_hand\_landmarks))
  
 for hand\_landmarks in results.multi\_hand\_landmarks:
  
 landmark21 = []
  
 for i in range(21):
  
 x = hand\_landmarks.landmark[i].x
  
 y = hand\_landmarks.landmark[i].y
  
 z = hand\_landmarks.landmark[i].z
  
 landmark21.append((x, y, z))
  
 landmark21\_ = np.array([landmark21])
  
 # 输入神经网络进行预测，得到预测结果为每一种手势对应的概率
  
 pre = self.\_my\_model.predict(landmark21\_[:1])
  
 # 预测结果大于0.5认为结果可信，进入下一步判断，可更改该值获得更高或较低的可信度
  
 if np.max(pre) > 0.5:
  
 # 得到预测结概率最大的手势的编号
  
 ges\_pre\_p = np.argmax(pre)
  
 # 将手势编号通过字典转化为我们给他起名的字符串
  
 self.now\_ges = self.\_ges\_dict[ges\_pre\_p]
  
 # 显示我们预测的手势与概率
  
 if show\_img:
  
 cv2.putText(image, str(np.max(pre)), (100, 200), 0, 1.3, (0, 0, 255), 3)
  
 cv2.putText(image, self.now\_ges, (0, 100), 0, 1.3, (0, 0, 255), 3)
  
 # 将手部的特征点连线显示
  
 for hand\_landmarks in results.multi\_hand\_landmarks:
  
 self.\_mp\_drawing.draw\_landmarks(
  
 image,
  
 hand\_landmarks,
  
 self.\_mp\_hands.HAND\_CONNECTIONS,
  
 self.\_mp\_drawing\_styles.get\_default\_hand\_landmarks\_style(),
  
 self.\_mp\_drawing\_styles.get\_default\_hand\_connections\_style())
  
 else:
  
 self.now\_ges='none'
  
 if show\_img:
  
 cv2.namedWindow('Gesture Recognition Client', cv2.WINDOW\_NORMAL)
  
 # 展示图片画面
  
 cv2.imshow('Gesture Recognition Client', image)
  
 # 按esc键退出
  
 if cv2.waitKey(5) & 0xFF == 27:
  
 break
  
 else:
  
 break
  
 self.free()

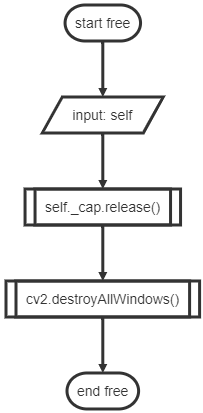


#### 在线程中开启

def start\_in\_thread(self):
  
 Thread(target=self.start\_in\_loop).start()#,name='recognizer')

#### 释放内存

def free(self):
  
 self.\_cap.release()
  
 cv2.destroyAllWindows()



#### 加载快捷键

def load(file=resource\_path("assets/bind.yml"))->dict:
  
 with open(file, 'r', encoding='utf-8') as f:
  
 binds = yaml.load(f.read(), Loader=yaml.FullLoader)
  
 f.close()
  
 return binds

#### 等待按键信号并按下快捷键

def waiting\_to\_press\_and\_release():
  
 global binds
  
 while True:
  
 if MyGlobalStates.\_\_run\_\_:
  
 if recognizer.now\_ges in binds.keys():
  
 if binds[recognizer.now\_ges]!='nothing':
  
 keyboard.press\_and\_release(binds[recognizer.now\_ges])
  
 time.sleep(0.2)
  
 else:
  
 break

#### 重新加载

def reloads():
  
 global binds
  
 binds=load()

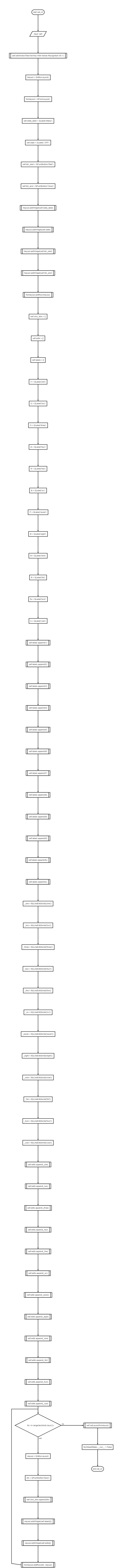
#### 在线程中开启

#等待开启的线程
  
def start\_in\_thread():
  
 Thread(target=waiting\_to\_press\_and\_release).start()

st3=>start: start start\_in\_thread
  
io5=>inputoutput: input: self
  
sub8=>subroutine: Thread(target=self.start\_in\_loop).start()
  
e10=>end: end start\_in\_thread
  
  
st3->io5
  
io5->sub8
  
sub8->e10

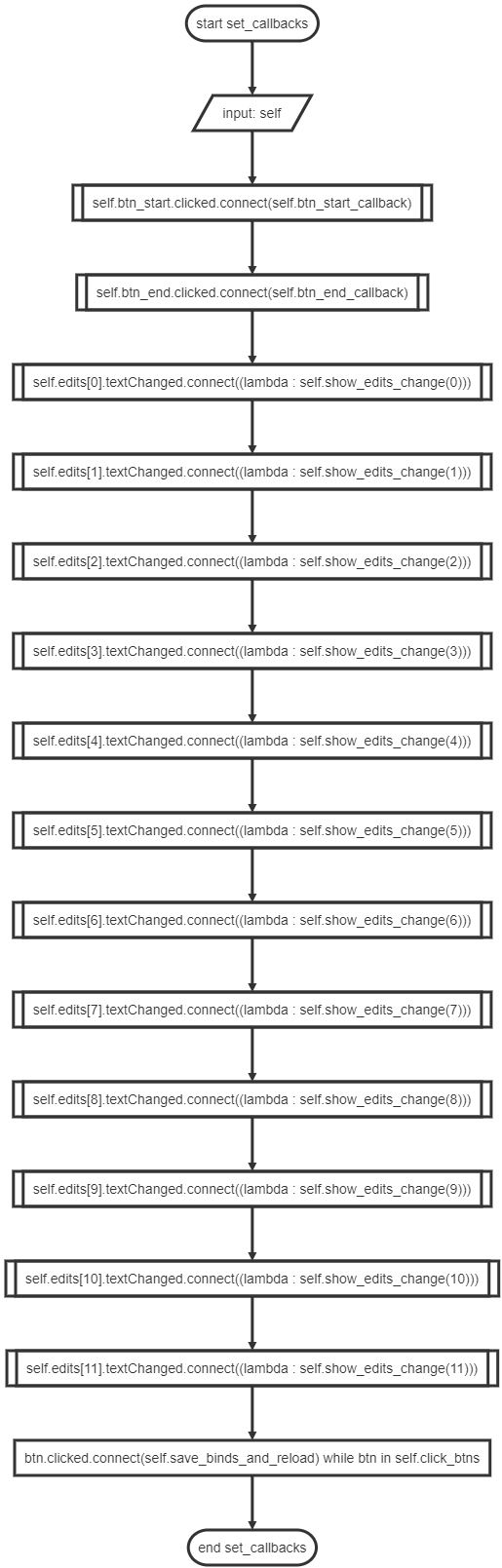
#### 加载UI

def set\_UI(self):
  
 self.setWindowTitle("Hot-Key-With-Hands-Recognition-V0.1")
  
 # self.resize(400, 100)
  
 hlayout = QHBoxLayout()
  
 fromlayout = QFormLayout()
  
  
 self.state\_label = QLabel('Status')
  
 self.state = QLabel(' OFF')
  
 self.btn\_start = QPushButton('Start')
  
 self.btn\_end = QPushButton('Close')
  
  
 hlayout.addWidget(self.state\_label)
  
 hlayout.addWidget(self.state)
  
 hlayout.addWidget(self.btn\_start)
  
 hlayout.addWidget(self.btn\_end)
  
  
 fromlayout.addRow(hlayout)
  
 self.click\_btns=[]
  
 self.edits=[]
  
 self.labels=[]
  
  
 l1=QLabel('one')
  
 l2=QLabel('two')
  
 l3=QLabel('three')
  
 l4=QLabel('four')
  
 l5=QLabel('five')
  
 l6=QLabel('six')
  
 l7=QLabel('seven')
  
 l8=QLabel('eight')
  
 l9=QLabel('nine')
  
 lfi=QLabel('fist')
  
 lfu=QLabel('fuck')
  
 lc=QLabel('cool')
  
  
 self.labels.append(l1)
  
 self.labels.append(l2)
  
 self.labels.append(l3)
  
 self.labels.append(l4)
  
 self.labels.append(l5)
  
 self.labels.append(l6)
  
 self.labels.append(l7)
  
 self.labels.append(l8)
  
 self.labels.append(l9)
  
 self.labels.append(lfi)
  
 self.labels.append(lfu)
  
 self.labels.append(lc)
  
  
 \_one= MyLineEdit(binds['one'])
  
 \_two = MyLineEdit(binds['two'])
  
 \_three = MyLineEdit(binds['three'])
  
 \_four = MyLineEdit(binds['four'])
  
 \_five = MyLineEdit(binds['five'])
  
 \_six = MyLineEdit(binds['six'])
  
 \_seven= MyLineEdit(binds['seven'])
  
 \_eight = MyLineEdit(binds['eight'])
  
 \_nine= MyLineEdit(binds['nine'])
  
 \_fist = MyLineEdit(binds['fist'])
  
 \_fuck = MyLineEdit(binds['fuck'])
  
 \_cool = MyLineEdit(binds['cool'])
  
  
 self.edits.append(\_one)
  
 self.edits.append(\_two)
  
 self.edits.append(\_three)
  
 self.edits.append(\_four)
  
 self.edits.append(\_five)
  
 self.edits.append(\_six)
  
 self.edits.append(\_seven)
  
 self.edits.append(\_eight)
  
 self.edits.append(\_nine)
  
 self.edits.append(\_fist)
  
 self.edits.append(\_fuck)
  
 self.edits.append(\_cool)
  
  
 # for e in self.edits:
  
 # e.setFocusProxy(Qt.NoFocus)
  
  
 for i in range(len(binds.keys())):
  
 vlayout = QHBoxLayout()
  
  
 btn=QPushButton('Save')
  
  
 self.click\_btns.append(btn)
  
 # PATH = QLineEdit(self.edits[i])
  
 # PATH.setFocusPolicy(Qt.NoFocus) # 设置不可编辑
  
  
 vlayout.addWidget(self.labels[i])
  
 vlayout.addWidget(self.edits[i])
  
 fromlayout.addRow(btn,vlayout)
  
  
 self.setLayout(fromlayout)
  
 MyGlobalStates.\_\_run\_\_ = False



#### 设置回调函数

def set\_callbacks(self):
  
 #绑定开始和结束按钮的按键,按下后修改全局状态标志位,修改开关状态的标签
  
 self.btn\_start.clicked.connect(self.btn\_start\_callback)#这里差了一个回调函数
  
 self.btn\_end.clicked.connect(self.btn\_end\_callback)
  
 self.edits[0].textChanged.connect(lambda: self.show\_edits\_change(0))
  
 self.edits[1].textChanged.connect(lambda: self.show\_edits\_change(1))
  
 self.edits[2].textChanged.connect(lambda: self.show\_edits\_change(2))
  
 self.edits[3].textChanged.connect(lambda: self.show\_edits\_change(3))
  
 self.edits[4].textChanged.connect(lambda: self.show\_edits\_change(4))
  
 self.edits[5].textChanged.connect(lambda: self.show\_edits\_change(5))
  
 self.edits[6].textChanged.connect(lambda: self.show\_edits\_change(6))
  
 self.edits[7].textChanged.connect(lambda: self.show\_edits\_change(7))
  
 self.edits[8].textChanged.connect(lambda: self.show\_edits\_change(8))
  
 self.edits[9].textChanged.connect(lambda: self.show\_edits\_change(9))
  
 self.edits[10].textChanged.connect(lambda: self.show\_edits\_change(10))
  
 self.edits[11].textChanged.connect(lambda: self.show\_edits\_change(11))
  
  
 for btn in self.click\_btns:
  
 btn.clicked.connect(self.save\_binds\_and\_reload)



#### 每个回调函数

def btn\_start\_callback(self):
  
 if MyGlobalStates.\_\_run\_\_ == False:
  
 try:
  
 self.state.setText('On')
  
 MyGlobalStates.\_\_run\_\_=True
  
 recognizer.start\_in\_thread()
  
 start\_in\_thread()
  
 except Exception as e:
  
 QMessageBox.information(self, 'Error', e, QMessageBox.Ok)

def btn\_end\_callback(self):
  
 self.state.setText('Close')
  
 MyGlobalStates.\_\_run\_\_=False

def show\_edits\_change(self,idx):
  
 binds[list(binds.keys())[idx]]=self.edits[idx].text()

def save\_binds\_and\_reload(self):
  
 with open(resource\_path('assets/bind.yml'),'w',encoding='utf-8') as f:
  
 f.write("#ALL\_Gesture: {0: 'cool', 1: 'eight', 2: 'fist', 3: 'five', 4: 'four', 5: 'fuck', 6: 'nine', 7: 'one', 8: 'seven', 9: 'six', 10: 'three', 11: 'two'}\n")
  
 #保存配置
  
 for k in binds.keys():
  
 f.write(f"{k}: {binds[k]}\n")
  
 f.close()
  
 reloads()
  
 QMessageBox.information(self, 'Saving Changes', 'Saved Successfully !', QMessageBox.Ok)

def closeEvent(self, a0: QtGui.QCloseEvent) -> None:
  
 MyGlobalStates.\_\_run\_\_=False

def start\_gui():
  
 app = QApplication(sys.argv)
  
 form = Winform()
  
 form.show()
  
 sys.exit(app.exec\_())