简化的PS

姓名：郭泽宇

学号：41802198

班级：通信1804

北京科技大学计通学院

|  |
| --- |
| 摘要：完成简易PS的实现，通过编写GUI界面，与用户进行交互，实现显示图片，调整图片亮度，缩放和旋转图片，添加水印等功能。GUI界面的实现，通过PyQt5来进行编写，对于图像的处理，则通过Pillow库对其进行处理，完成所需要的各项功能。布局采用上下格式，用QLabel来存放图片，之后下面用自己定义的Tab组件来实现各种操作，另外添加菜单栏进行文件的操作，如打开和保存、关闭等。 |
| 关键字：Python; Photoshop; PyQt5; Pillow; GUI |
| 代码行数：431行 |

# 项目背景和意义

## 项目背景

随着时代的发展，人们对于修图的需求越来越大，在生活中随处可见处理过的图片，原模原样的图片已经很少见了，在基本的手机自带的相机中，都有着一大堆对于图像进行处理的功能。在学习完本学期的Python课程之后，我对GUI界面编程十分感兴趣，所以打算用PyQt5结合Pillow库，制作一个简单的修图软件。

## 意义

通过搭建基本的软件GUI界面，和对Pillow库的使用，增强了我的编程能力。在项目开始之前，我对于PyQt5的GUI搭建和Pillow库还不是很了解，需要上网寻找资料，进行自学。这锻炼了我们收集信息的能力。在编程过程中，难免会出现一个又一个的bug，这时候就需要自己进行程序的调试。但有的时候是自己对于程序背后的理解有误，例如对于信号传递机制的理解不够到位等。在不断的处理bug的过程中，增强了我解决问题的能力。遇到问题，不断克服问题。看着一个项目从无到有，被自己一点点的构建起来，这极大的增强了我编程的自信和能力。经过此次开发，我对于项目开发的基本流程有了一定的了解，不再是盲目的开发，想到哪里就写哪里。而是在开始就对程序进行拆分，分成不同的部分，一步一步地分开进行实现。先实现最基本的一部分功能，之后为其添砖加瓦。先实现可用的软件，再继续实现其他。并在程序中注意响应变化，为未来程序的扩展进行准备。在真正的项目中，需求也是在一直变化的，所以我们要注意响应的变化。

# 需求分析

## 基本的主界面的分析

主界面主要是由一个QLabel标签和一个自定义的组件（继承于QTabWidget）来实现，主窗口继承于QMainWindow，QLabel用于显示图片的处理结果，QTabWidget用来存放对于图片的操作的选项，如调整大小和添加水印等。主页面继承于QMainWidget，可以实现对应的菜单栏和状态栏，来实现打开图片和保存图片等功能。

## 对于图片的各个参数的调整

用四个QSlider来实现对于照片的亮度，长宽和旋转角度的调整。为了让调节的参数更加的直观可见，加入QLabel来实现对QSlier数值的显示。可设置QSlier的范围为-100到100，这样就既可以使参数正向增加也可以负向减少了。对于图像的各个参数，直接传入这样的值是显然不行的，要通过一些基础的方法，来进行数值的转换，转成图片可以接受的参数。其中，亮度可以使用Pillow. ImageEnhance模块进行操作。大小可以使用PyQt5.QtGui的QPixmap.resize()进行操作。旋转可以使用Pillow的Image.rotate()进行操作。在调节过程中，难免会有调错，想要重新复原的需求。为了避免手动调整参数到初始值这种繁杂的操作，因此加入了3个重置按钮来重置照片的各个参数。在打开图片之前，同样需要调用这三个重置方法来实现各个数值和QSlider的复位。

## 水印

要实现图片的添加水印的功能，添加一个按钮来进行水印的添加和去除。使用Pillow.ImageDraw模块在图片的左上角添加一个“Watermark”的字样作为水印。并在添加之后，让按钮的显示内容变为“去除水印”，用于去除水印，来灵活地实现水印的添加和去除。

# 概要和详细设计

## 代码总框图



图 1代码总框架

其中，Photoshop.py是软件主体窗体，以及执行部分，Widget\_self.py是自定义控件部分，Variabel.py是全局变量与常量部分，ProcessPhoto.py是图片处理方法部分。

## 各部分框图

### 菜单栏以及其三个事件

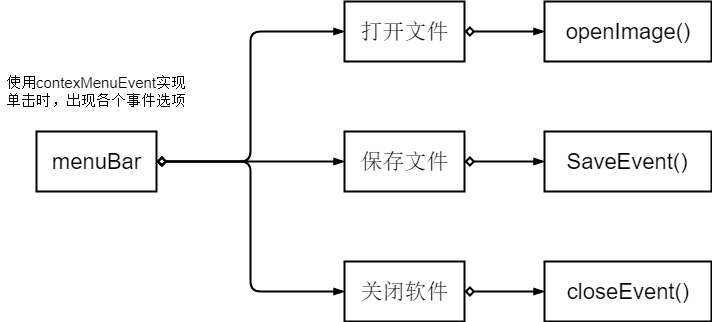


图 2菜单栏以及其三个事件

### 调整图片的各个参数

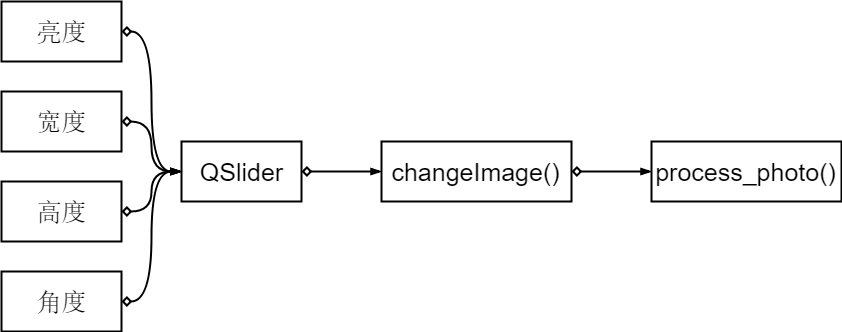


图 3调整图片的各个参数

### 水印



图 4水印

### 全局变量与常量

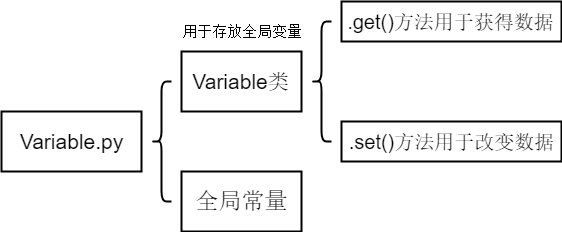


图 5全局变量与常量

### 处理图片的Process类

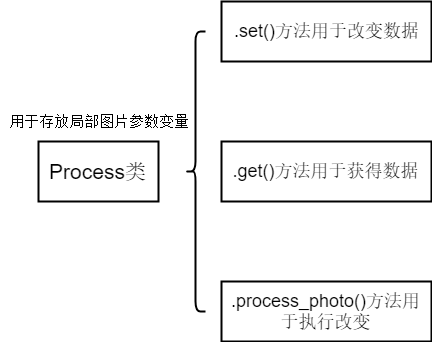


图 6处理图片的Process类

# 代码实现

## python版本以及库版本说明

Python: 3.9.5 64-bit

PyQt5: 5.15.4

PyQt5-stubs: 5.15.2.0

PyQt5-sip: 12.9.0

PyQt5-Qt5: 5.15.2

Pillow: 8.2.0

## 所使用的关键库

### 4.2.1 PyQt5

版本5.15.4，是主要的搭建界面所用的库，完成整个界面的搭建，在整个实验过程中使用了其中的多个组件，并完成事件的响应，响应鼠标的各种点击事件。

### 4.2.2 Pillow

版本7.1.1，PIL是python自带的处理图片的库，整个过程中的图片处理都通过这个库来实现，包括调整大小，水印滤镜等功能。

### 4.2.3 Sys

保证程序的正常运行所调用的库。

## 各个文件的说明

### 4.3.1 Photoshop.py

主要的程序文件，进行主要的界面搭建以及运行，这里是程序的入口文件。

### 4.3.2 Widget\_self.py

在该文件中，主要进行自定义组件，包括调整图片参数的QSlider，显示QSlider数值的QLabel，进行重置功能和加水印功能的QButton。并为每个组件链接响应事件的函数，来完成对于图片的不同处理。

### 4.3.3 Variable.py

用于存放在整个程序中用到的全局变量，各种常量，并为每种全局变量提供get和set方法，用于得到和设置全局变量。各种常量用于规定窗口的大小等。如果将来需求发生变化，需要改变初始窗口的大小，可以直接在这个文件中进行改变，增强了软件响应变化的能力。

### 4.3.4 ProcessPhoto.py

对图片进行处理的文件，每一次对图片进行处理，都需要进行调用其中的函数，包括改变亮度、调整大小、旋转图片和添加水印。并提供改变的接口，方便其他python文件进行调用。

## 关键代码说明

### Photoshop.py

#### 初始化自定义组件

1. **class** MyTab(QTabWidget):
2. **def** \_\_init\_\_(self, parent):
3. super().\_\_init\_\_()
4. self.parent = parent
6. # 添加自定义组件
7. self.adjust\_tab = AdjustTab()
8. self.addTab(self.adjust\_tab, "调整参数")    # 设置其显示名称
9. self.setMaximumHeight(300)      # 设置其最大高度

说明：自定义组件MyTab的实现，继承于QTabWidget类，为其添加了一个Tab用于操作图片。

#### 初始化软件主窗口

1. **def** initUI(self):
2. # 初始化界面
3. # 设置状态栏
4. self.statusbar = self.statusBar()
5. self.statusbar.showMessage('Ready')
6. # 设置菜单栏
7. # 新建打开动作
8. openAct = QAction('打开', self)  # 打开动作
9. openAct.setShortcut('Ctrl+O')   # 打开快捷键
10. openAct.setStatusTip('打开文件')    # 打开提示
11. openAct.triggered.connect(self.openImage)   # 连接打开事件
12. # 新建保存动作
13. saveAct = QAction('保存', self)  # 保存动作
14. saveAct.setShortcut('Ctrl+S')   # 保存快捷键
15. saveAct.setStatusTip('保存文件')    # 保存提示
16. saveAct.triggered.connect(self.SaveEvent)   # 连接保存事件
17. # 新建退出动作
18. exitAct = QAction('退出', self)  # 退出动作
19. exitAct.setShortcut('Ctrl+E')   # 退出快捷键
20. exitAct.setStatusTip('退出软件')    # 退出提示
21. exitAct.triggered.connect(self.close)   # 连接退出事件
23. # 新建一个菜单栏
24. menubar = self.menuBar()
25. fileMenu = menubar.addMenu('文件')  # 设置菜单栏显示的内容
26. # 加入上述三个事件
27. fileMenu.addAction(openAct)
28. fileMenu.addAction(saveAct)
29. fileMenu.addAction(exitAct)
31. # 布局
32. # 图片显示部分
33. imagelabel = QLabel("")  # 使用QLabel来显示图片
34. Variable.set\_imagelabel(imagelabel)  # 将QLabel放置到全局变量
35. imagelabel.setAlignment(Qt.AlignCenter)  # 中心对齐
37. # 修改操作部分
38. self.mytab = MyTab(self)    # 使用自定义控件来进行操作
40. # 设置框垂直布局
41. vbox = QVBoxLayout()
42. vbox.addWidget(imagelabel)
43. vbox.addWidget(self.mytab)
44. # 将布局页面设置到主窗口中间
45. main\_frame = QWidget()
46. main\_frame.setLayout(vbox)
47. self.setCentralWidget(main\_frame)
49. # 窗口设置
50. self.resize(Variable.WINDOW\_WIDTH, Variable.WINDOW\_HEIGHT)  # 调整大小
51. self.center()   # 让窗口出现在屏幕中间
52. self.setWindowTitle('简易PS')   # 窗口标题
53. self.setWindowIcon(QIcon('Mini Photoshop/ps.ico'))  # 窗口图标
54. self.show()  # 呈现窗口

说明：实现GUI组件的摆放，并添加菜单栏和状态栏，并为这些东西添加Action，并将其绑定到对应的函数上。

#### 打开文件的方法

1. **def** openImage(self):
2. # 打开文件事件
3. imagelabel = Variable.get\_imagelabel()  # 从全局变量中获取QLabel
5. fname, \_ = QFileDialog.getOpenFileName(
6. self, '打开图片', '/', "Image files (\*.jpg \*.png)")  # 打开对话框来进行文件选择，获得文件路径
8. **if** fname:   # 防止文件路径为空导致错误，设置一个条件来判断是否执行，之后的其他事件与此相同
9. image = Image.open(fname)   # 使用Pillow库来打开文件
10. Variable.set\_image(image)   # 将其保存到全局变量
12. # 将几个操作参数复位
13. self.mytab.adjust\_tab.reset()
14. self.mytab.adjust\_tab.resize()
15. self.mytab.adjust\_tab.rerotation()
17. # 呈现图片
18. qimg = ImageQt(image)   # 先将Image转成QImage
19. img\_pix = QPixmap.fromImage(
20. qimg, Qt.AutoColor)  # 再从QImage转成QPixmap
21. img\_pix = img\_pix.scaled(
22. Variable.DEFAULT\_WIDTH, Variable.DEFAULT\_HEIGHT, Qt.KeepAspectRatio)    # 将QPixmap按图片比例调整大小至可放入QLabel
23. imagelabel.setPixmap(img\_pix)   # 放入QLabel
24. # 将此时的长宽存入全局变量
25. Variable.set\_width(img\_pix.width())
26. Variable.set\_height(img\_pix.height())
27. # 记录图片的现时大小（注：之前只是修改了QPixmap的大小，这里指的是记录图片的目前大小，方便之后复用）
28. process.change\_width(0)
29. process.change\_height(0)
30. **else**:
31. **pass**

说明：该函数用于打开图片，当用户点击打开时，就会调用这个函数，用QFileDialog.getOpenFileName来获得图片的路径，之后调用Pillow来打开图片，同时更改用来存储图片的全局变量。为了保证多次打开之间不会相互影响，在每次打开图片之后，调用各个组件的重置函数。

#### 保存文件的方法

1. **def** SaveEvent(self):
2. # 保存文件事件
3. filename, \_ = QFileDialog.getSaveFileName(
4. self, "文件保存", '/', "Image files (\*.jpg \*.png)")
5. **if** filename:
6. image = Variable.get\_image()    # 获取全局变量中的image进行操作
7. process.change\_save(filename)   # 改变保存标志
8. process.process\_photo(image)    # 保存文件
9. **else**:
10. **pass**

说明：该函数用来响应保存事件，获得保存路径之后，改变保存标识，调用process\_photo()方法进行保存。

#### 关闭软件的方法

1. **def** closeEvent(self, event):
2. # 关闭软件事件
3. reply = QMessageBox.question(self, '温馨提示',
4. "你确定要退出吗？", QMessageBox.Yes |
5. QMessageBox.No, QMessageBox.No)    # 关闭弹出确认框
7. # 确认框的响应事件
8. **if** reply == QMessageBox.Yes:
9. event.accept()
10. **else**:
11. event.ignore()

说明：该函数用来响应窗口关闭事件，询问是否退出程序。

#### 设置菜单的功能

1. **def** contextMenuEvent(self, event):
2. # 菜单内容
3. cmenu = QMenu(self)
5. opnAct = cmenu.addAction("打开")
6. saveAct = cmenu.addAction("保存")
7. action = cmenu.exec\_(self.mapToGlobal(event.pos()))
9. **if** action == opnAct:
10. self.openImage()
11. **if** action == saveAct:
12. self.SaveEvent()

说明：用来响应用户在界面上的鼠标右击事件，显示一个菜单，可以进行图片的打开和保存。

### Widget\_self.py

#### 初始化自定义组件的界面和链接各个部件的功能

1. **class** AdjustTab(QWidget):
2. # 自定义的控件
3. **def** \_\_init\_\_(self):
4. # 初始化界面
5. super().\_\_init\_\_()
6. # 加入亮度调整部分
7. self.bright\_label = QLabel("亮度")  # 显示内容，下同
8. self.bright\_label\_value = QLabel('0')   # 显示数值，下同
9. self.bright\_slider = QSlider(Qt.Horizontal, self)   # 加入滑动条，下同
10. # 设置滑动条最大最小值，下同
11. self.bright\_slider.setMaximum(100)
12. self.bright\_slider.setMinimum(-100)
13. self.bright\_slider.setValue(0)  # 设置滑动条初始值，下同
14. self.bright\_slider.valueChanged[int].connect(
15. self.changeImage)  # 连接参数变更事件，下同
16. # 加入高度调整部分
17. self.high\_label = QLabel("高度")
18. self.high\_label\_value = QLabel('0')
19. self.high\_slider = QSlider(Qt.Horizontal, self)
20. self.high\_slider.setMaximum(100)
21. self.high\_slider.setMinimum(-100)
22. self.high\_slider.setValue(0)
23. self.high\_slider.valueChanged[int].connect(self.changeImage)
24. # 加入宽度调整部分
25. self.width\_label = QLabel("宽度")
26. self.width\_label\_value = QLabel('0')
27. self.width\_slider = QSlider(Qt.Horizontal, self)
28. self.width\_slider.setMaximum(100)
29. self.width\_slider.setMinimum(-100)
30. self.width\_slider.setValue(0)
31. self.width\_slider.valueChanged[int].connect(self.changeImage)
32. # 加入角度调整部分
33. self.rotation\_label = QLabel("角度")
34. self.rotation\_label\_value = QLabel('0')
35. self.rotation\_slider = QSlider(Qt.Horizontal, self)
36. self.rotation\_slider.setMaximum(180)
37. self.rotation\_slider.setMinimum(-180)
38. self.rotation\_slider.setValue(0)
39. self.rotation\_slider.valueChanged[int].connect(self.changeImage)
40. # 加入重置部分
41. self.reset\_button = QPushButton("重置亮度")  # 加入重置亮度按钮
42. self.reset\_button.clicked.connect(self.reset)   # 连接重置亮度事件
43. self.bind\_button = QPushButton("重置大小")  # 加入重置大小按钮
44. self.bind\_button.clicked.connect(self.resize)   # 连接重置大小事件
45. self.rerotation\_button = QPushButton("重置角度")    # 加入重置角度按钮
46. self.rerotation\_button.clicked.connect(self.rerotation)  # 连接重置角度事件
47. self.watermark\_button = QPushButton("添加水印")  # 加入添加水印按钮
48. self.watermark\_button.clicked.connect(self.add\_watermark)   # 加入添加水印事件
50. # 款式布局
51. # 水平部分
52. hbox1 = QHBoxLayout()
53. hbox1.addWidget(self.bright\_label)
54. hbox1.addWidget(self.bright\_label\_value)
55. hbox2 = QHBoxLayout()
56. hbox2.addWidget(self.high\_label)
57. hbox2.addWidget(self.high\_label\_value)
58. hbox3 = QHBoxLayout()
59. hbox3.addWidget(self.width\_label)
60. hbox3.addWidget(self.width\_label\_value)
61. hbox4 = QHBoxLayout()
62. hbox4.addWidget(self.rotation\_label)
63. hbox4.addWidget(self.rotation\_label\_value)
64. hbox5 = QHBoxLayout()
65. hbox5.addStretch()
66. hbox5.addWidget(self.reset\_button)
67. hbox5.addStretch()
68. hbox5.addWidget(self.bind\_button)
69. hbox5.addStretch()
70. hbox5.addWidget(self.rerotation\_button)
71. hbox5.addStretch()
72. hbox5.addWidget(self.watermark\_button)
73. hbox5.addStretch()
74. # 垂直部分
75. vbox = QVBoxLayout()
76. vbox.addLayout(hbox1)
77. vbox.addWidget(self.bright\_slider)
78. vbox.addLayout(hbox2)
79. vbox.addWidget(self.high\_slider)
80. vbox.addLayout(hbox3)
81. vbox.addWidget(self.width\_slider)
82. vbox.addLayout(hbox4)
83. vbox.addWidget(self.rotation\_slider)
84. vbox.addLayout(hbox5)
86. self.setLayout(vbox)

说明：实现GUI组件的摆放，并添加QSlider和QLabel还有QButton。为它们添加布局，并为这些东西添加Action，将其绑定到对应的函数上。

#### 改变图片的事件

1. **def** changeImage(self, value):
2. # 改变参数事件
3. # 从全局变量中获取image进行修改
4. image = Variable.get\_image()
6. # 获取修改参数
7. source = self.sender()  # 判断修改参数来源，获取具体修改内容
8. **if** source == self.bright\_slider:
9. # 修改亮度
10. self.bright\_label\_value.setText(str(value))  # 改变显示的值，下同
11. bright = (self.bright\_slider.value() + 100) / 100   # 获取参数，下同
12. process.change\_bright(bright)   # 改变图片相应属性值，下同
13. **elif** source == self.high\_slider:
14. # 修改高度
15. self.high\_label\_value.setText(str(value))
16. high = self.high\_slider.value()
17. process.change\_height(high)
18. **elif** source == self.width\_slider:
19. # 修改宽度
20. self.width\_label\_value.setText(str(value))
21. width = self.width\_slider.value()
22. process.change\_width(width)
23. **elif** source == self.rotation\_slider:
24. # 修改角度
25. self.rotation\_label\_value.setText(str(value))
26. angle = self.rotation\_slider.value()
27. process.change\_angle(angle)
29. # 应用修改
30. process.process\_photo(image)

说明：本部分响应QSlider的改变，通过使用Process中的.change()方法，来改变图片的属性值。最后通过process\_photo()方法执行这些改变。

#### 添加水印事件

1. **def** add\_watermark(self):
2. # 添加水印事件
3. image = Variable.get\_image()
4. process.change\_watermark()  # 改变水印标志
5. process.process\_photo(image)    # 应用修改
7. # 修改按钮显示内容
8. **if** process.get\_watermark():
9. self.watermark\_button.setText("取消水印")
10. **else**:
11. self.watermark\_button.setText("添加水印")

说明：本部分响应“增加水印”的QButton的响应，通过使用Process中的.change()方法改变水印标识，再通过process\_photo()方法执行改变。最后还需将QButton的内容进行更改。

#### 复位事件

1. **def** reset(self):
2. # 重置亮度事件
3. self.bright\_slider.setValue(0)  # 将相关值设置为初始值0，下同
5. **def** resize(self):
6. # 重置大小事件
7. self.width\_slider.setValue(0)
8. self.high\_slider.setValue(0)
10. **def** rerotation(self):
11. # 重置角度事件
12. self.rotation\_slider.setValue(0)

说明：本部分包括三个QSlider的复位事件，响应的是三个QButton的点击事件，以及图片打开事件。

### ProcessPhoto.py

#### 定义一个类来存储图片属性

1. **class** Process():
2. # 修改类
3. imagelabel = Variable.get\_imagelabel()
5. **def** \_\_init\_\_(self):
6. self.bright = 1
7. self.sharpness = 1
8. self.contrast = 1
9. self.angle = 0
10. self.height = 0
11. self.width = 0
12. self.watermark = False
13. self.save = ""

说明：定义Process类，专门用于对图像进行处理。因为如果只在其他地方对于图像进行处理，则会导致不同的属性处理的时候，另一属性的处理效果就会消失。所以要进行整体的封装，要将图片的各个属性进行封装，每次进行整体的处理。

#### 改变图片属性值的相关函数

1. # 改变属性值的函数
2. **def** change\_bright(self, bright):
3. self.bright = bright
5. **def** change\_width(self, value):
6. self.width = Variable.get\_width() + value
8. **def** change\_height(self, value):
9. self.height = Variable.get\_height() + value
11. **def** change\_angle(self, angle):
12. self.angle = angle
14. **def** change\_watermark(self):
15. self.watermark = **not** self.watermark
17. **def** get\_watermark(self):
18. **return** self.watermark
20. **def** change\_save(self, path):
21. self.save = path

说明：完成对于图片的不同属性的更改，方便之后进行处理

#### Process\_photo()函数

1. **def** process\_photo(self, image):
2. # 应用修改
3. **if** image **is** **not** None:
4. enhancer = ImageEnhance.Brightness(image)   # 获取图片亮度
5. image = enhancer.enhance(self.bright)   # 修改图片亮度
7. **if** self.watermark:
8. idraw = ImageDraw.Draw(image)   # 添加水印
9. # 设置水印内容
10. text = "Watermark"
11. font = ImageFont.truetype("arial.ttf", size=200)
12. idraw.text((10, 10), text, font=font)
14. # 应用图片旋转
15. image = image.rotate(self.angle)
17. # 显示图片修改效果
18. imagelabel = Variable.get\_imagelabel()
19. qimg = ImageQt(image)
20. img\_pix = QPixmap.fromImage(qimg, Qt.AutoColor)
21. img\_pix = img\_pix.scaled(self.width, self.height)   # 应用图片大小设置
22. imagelabel.setPixmap(img\_pix)
24. # 如果存储标准不为False，则进行保存，保存完将标志重新设置为False
25. **if** self.save:
26. img\_pix.save(self.save)
27. self.save = False

说明：对于图像的处理函数，由于Pillow库自带的函数性质，可以每次从头进行图片的处理，以达到和GUI界面更好的契合。本函数主要是根据图片的各种参数、标识进行执行修改。

### Variable.py

#### 全局常量

1. # 常量的定义
2. WINDOW\_WIDTH = 1000
3. WINDOW\_HEIGHT = 1000
4. DEFAULT\_WIDTH = 800
5. DEFAULT\_HEIGHT = 600

说明：对在整个程序中用到的常量进行定义

#### Variable类存储全局变量

1. **class** Variable:
2. # 全局变量
3. image = None
4. imagelabel = None
5. width = 0
6. height = 0

9. **def** set\_width(value):
10. Variable.width = value

13. **def** get\_width():
14. **return** Variable.width

17. **def** set\_height(value):
18. Variable.height = value

21. **def** get\_height():
22. **return** Variable.height

25. **def** set\_image(image):
26. Variable.image = image

29. **def** get\_image():
30. **return** Variable.image

33. **def** set\_imagelabel(imagelabel):
34. Variable.imagelabel = imagelabel

37. **def** get\_imagelabel():
38. **return** Variable.imagelabel

说明：保存用到的全局变量，并为其添加set和get函数用于设置和取得对应的全局变量的值。

# 代码测试

## 5.1程序运行主界面

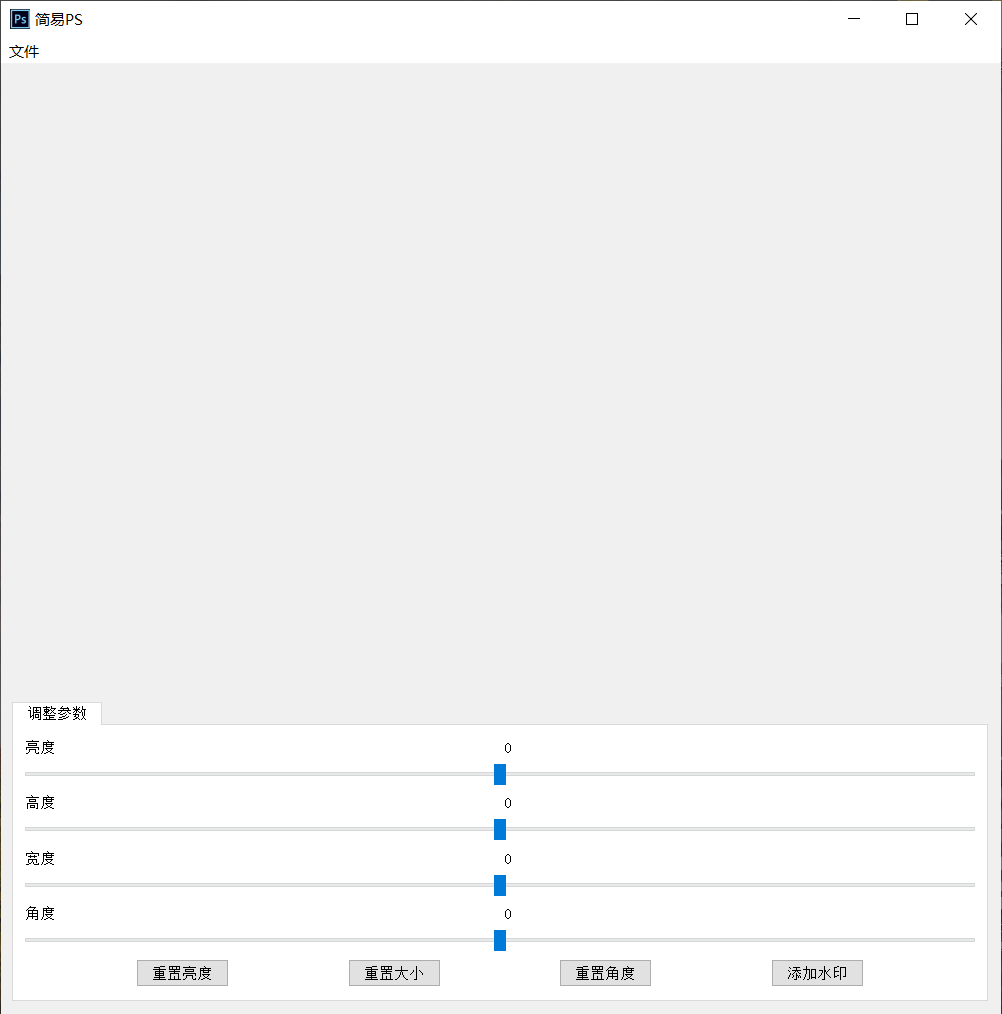


图 7程序运行主界面

## 5.2读取图片之后的界面

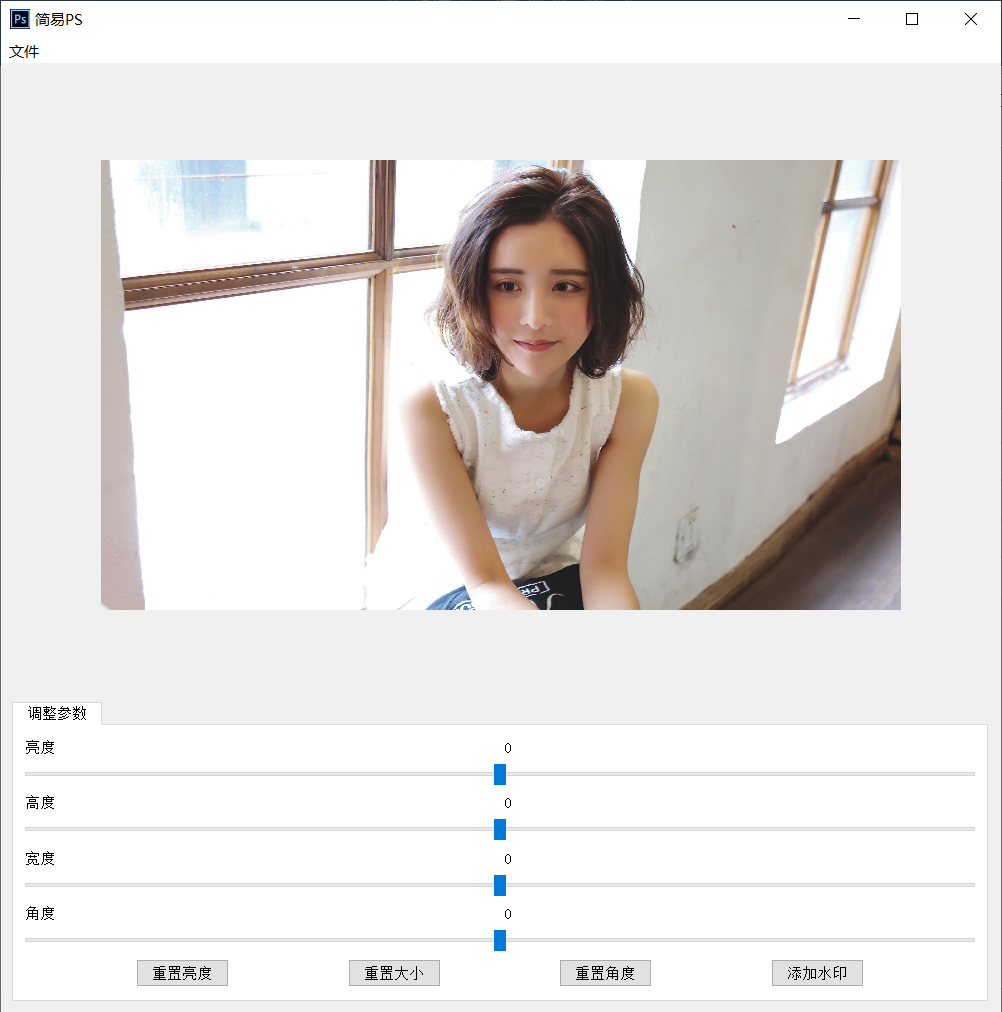


图 8读取图片之后的界面

## 5.3 使用各种操作调整图片

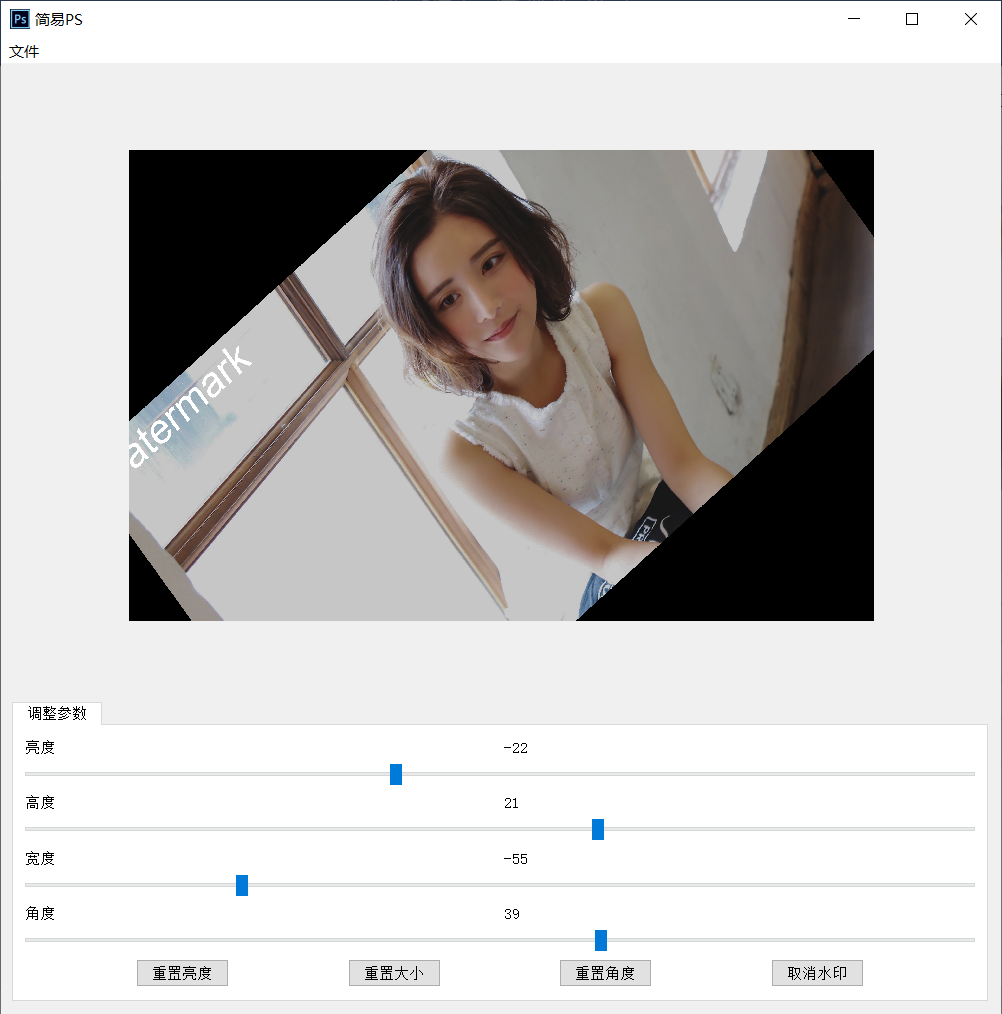


图 9使用各种操作调整图片

## 5.4 保存功能



图 10保存功能

## 5.5 重置功能和取消水印

本次使用了重置角度功能和取消水印功能



图 11重置功能和取消水印

## 5.6 退出功能

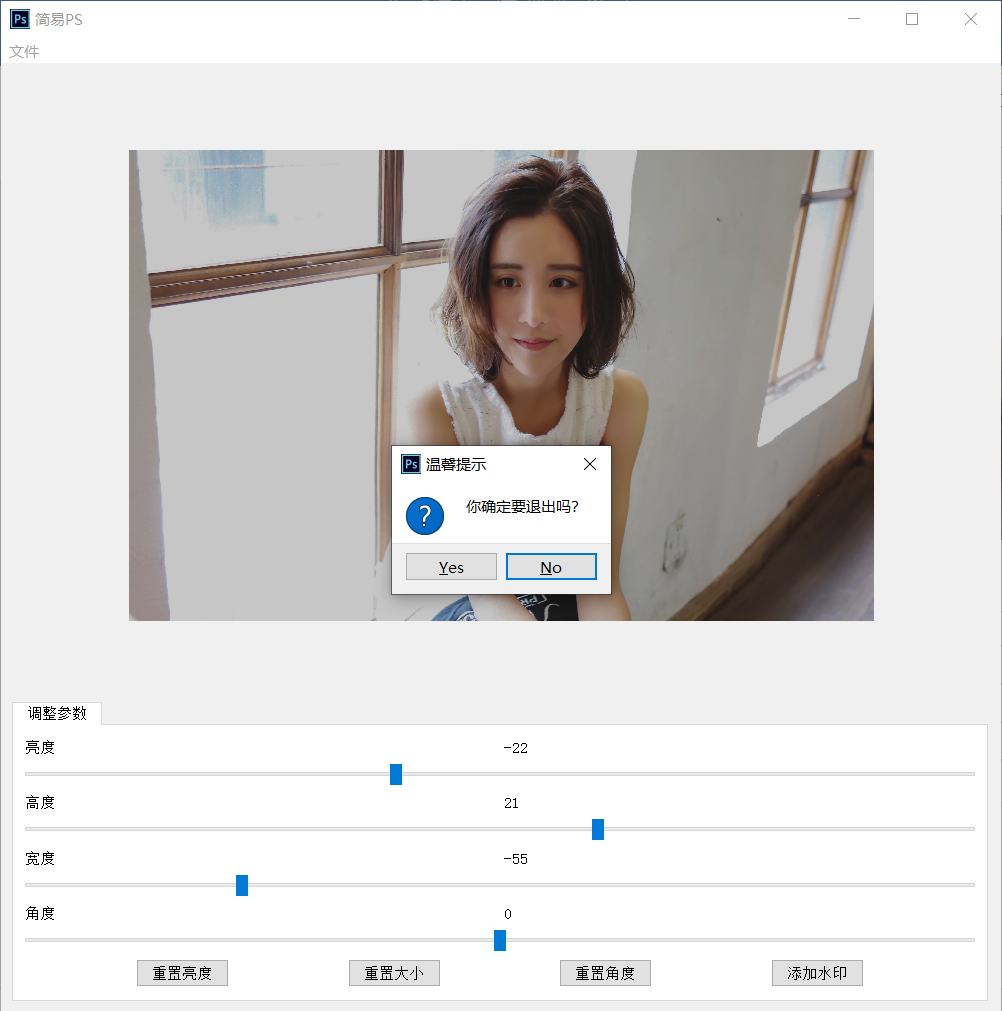


图 12退出功能

# 结论与未来方向

## 结论

该项目完成了一些图片处理的基本功能，加深了我对于PyQt5和Pillow库的了解。看着一个项目从无到有，自己的编程信心有了很大的提升。并对开发流程及注意事项有了一定的了解。最令我深刻的体会是，在开发中，要尽量解耦合，时时刻刻准备相应变化，让自己的代码在应对不同的需求的时候可以尽量少的改动。

## 未来方向

虽然已经完成了大部分的功能，但仍然还有很大的扩展空间，例如滤镜、裁剪等功能，这样对于图片的操作更加的自由地实现对于图片的编辑。但由于时间精力原因无法做到更好，十分遗憾。

# 致谢

感谢常争同学的督促与激励。

# 参考文献与链接

[1] [PyQt5 Reference Guide](https://www.riverbankcomputing.com/static/Docs/PyQt5/)

[2] [Pillow — Pillow (PIL Fork) 8.2.0 documentation](https://www.baidu.com/link?url=GcfQN3fUOddnL6zB2AIYUkssiK93ZZduqfUwyYclV6yoQrICSSykRgqgXY4eUAxr&wd=&eqid=a816220e0006db140000000360d47467)

[3] [PyQt5中文教程](http://code.py40.com/pyqt5/14.html)

# 9. 版本管理

## [Github](https://github.com/GLORYFeonix/Python_Learning_Homework.git)

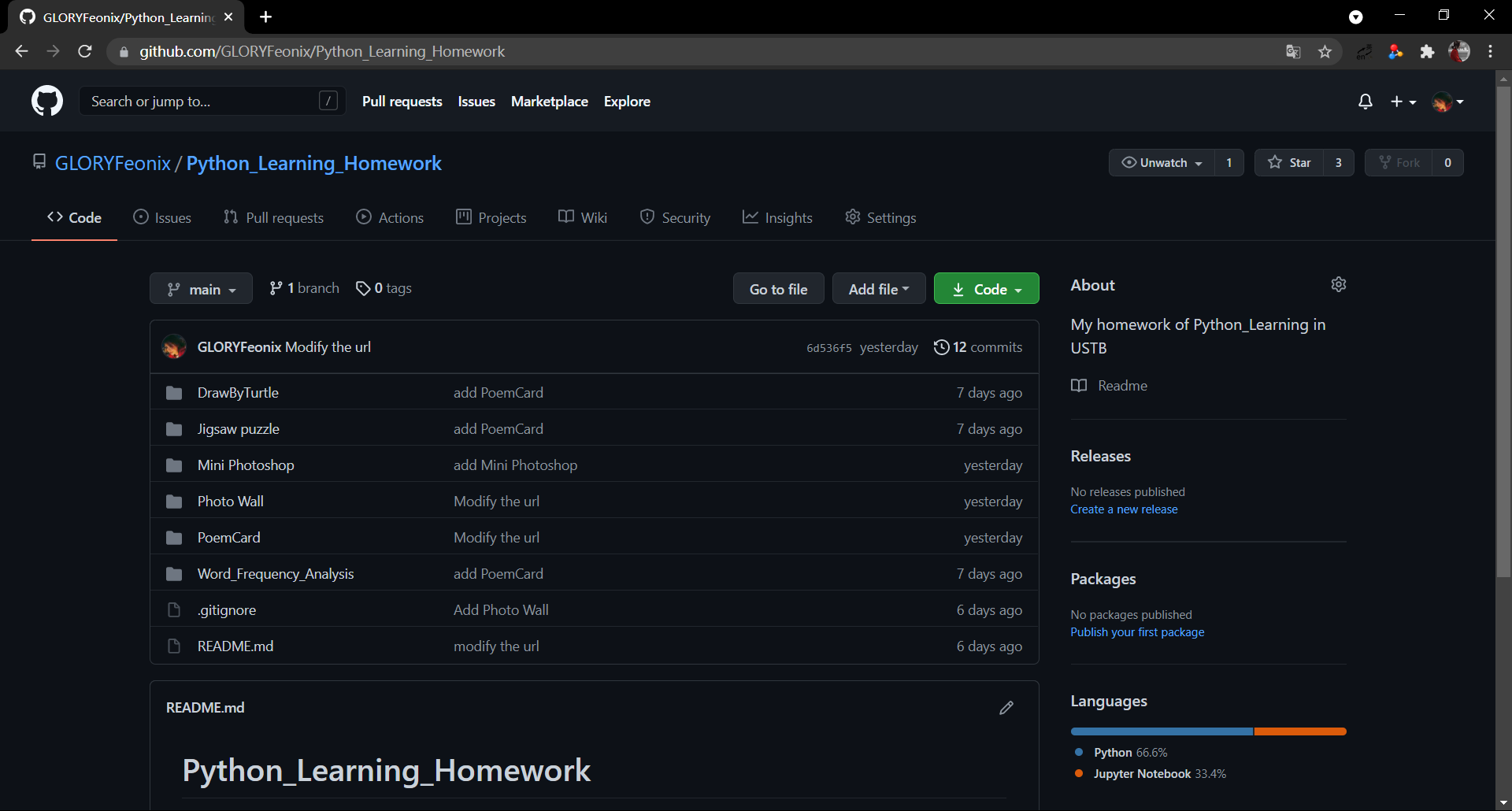


图 13Github

## [Gitee](https://gitee.com/gzy8810/Python_Learning_Homework.git)

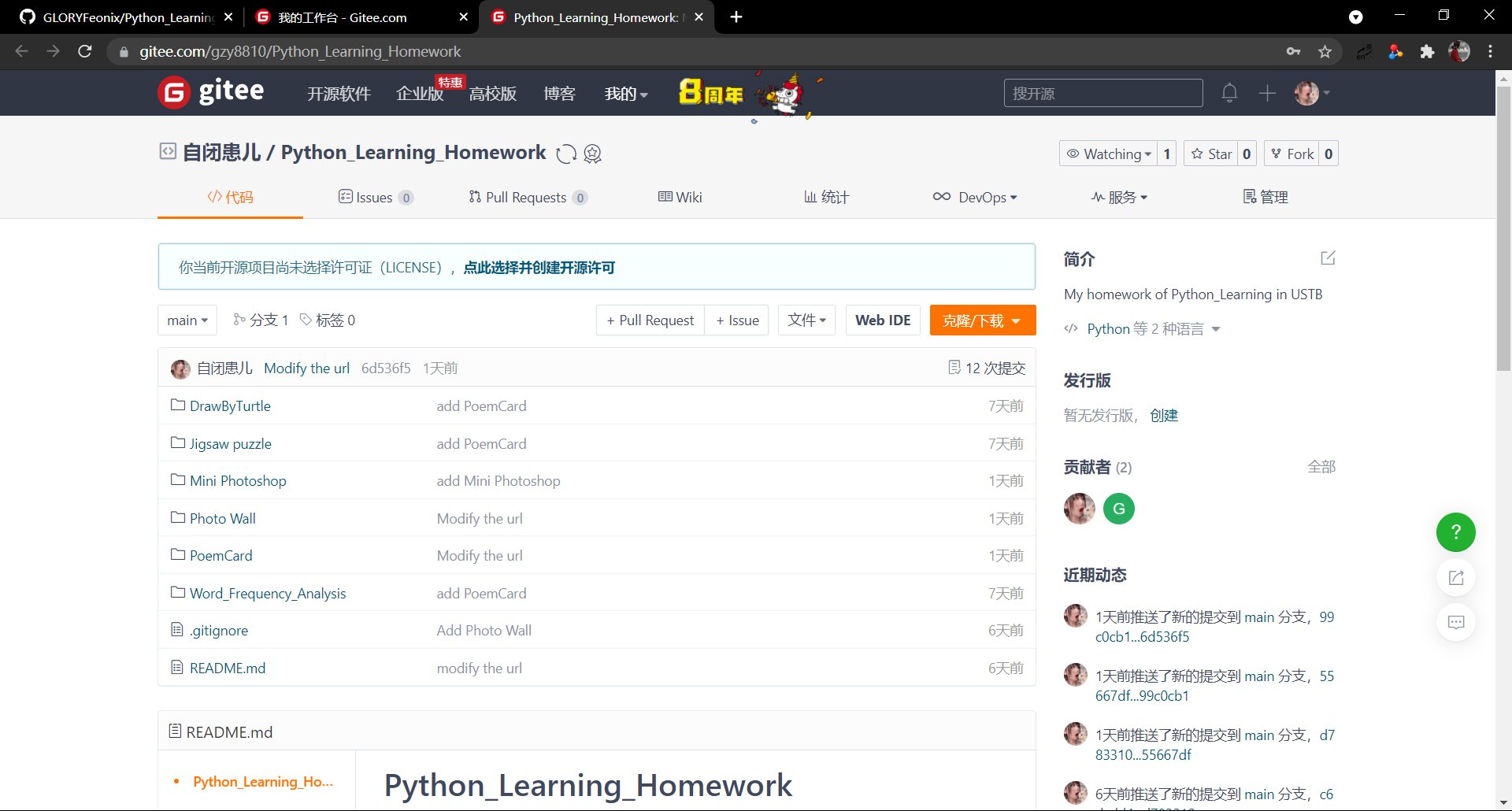


图 14Gitee