

Python 实现简单画板（二）

综合技术 2018-07-15 阅读原文



Python 实现简单画板

一、课程介绍

相信用过 Windows 的同学一定都对 Windows 自带的画板不陌生吧，虽然功能简单却也还实用。

今天我们的这门课程就是要利用 Pygame 模块来自己实现一个画板。

说明：本课程的项目来源于博客 <https://eyehere.net/2011/python-pygame-novice-professional-painter-1/>，其中博客给出的代码存在两处 BUG。

1. 使用铅笔的时候，如果调节画笔的大小会导致程序崩溃。
2. 初始的时候默认的画笔为铅笔，但是菜单栏中却显示的毛笔的笔刷

这里都已经进行了修复。

1.1 课程知识点

通过本次课程的学习，我们将学习到以下知识点：

- 使用 Pygame 制作画板

1.2 主要流程

本次课程的学习流程为：

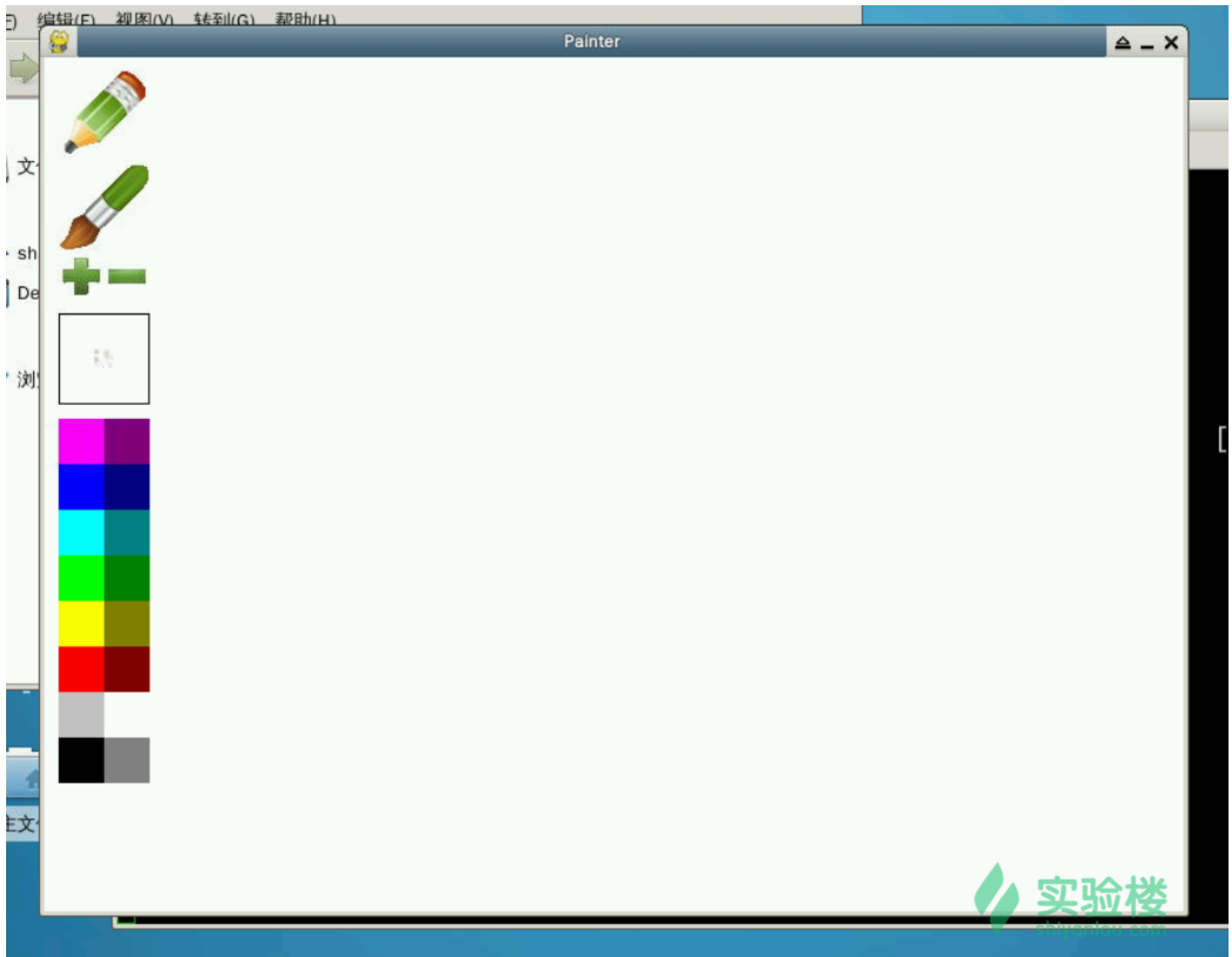
- 依赖模块安装
- 编程实现

1.3 实验环境

- 操作系统：Ubuntu 14.04.5
- Python 版本：3.5.1
- Pygame 版本：1.9.3

1.4 效果截图

最终，我们要实现的画板的效果图如下：



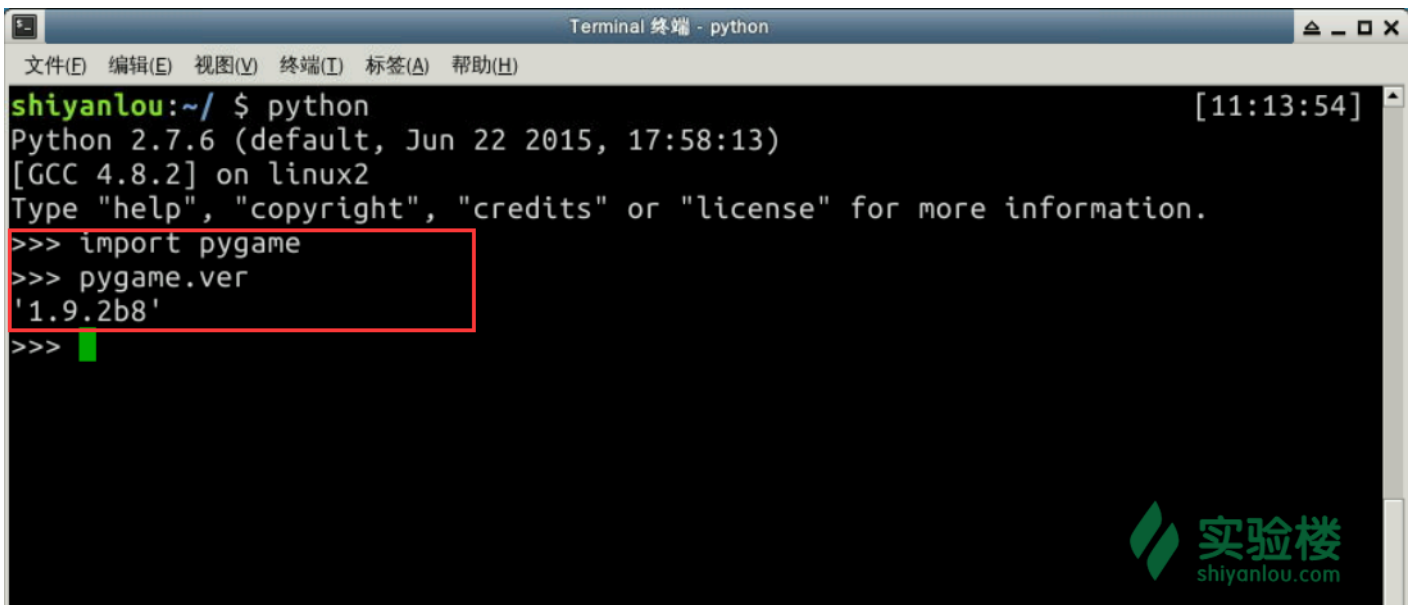
二、依赖模块安装

本次课程主要利用 Pygame 模块来进行开发，首先我们需要通过 pip 来安装 Pygame 模块。

```
$ sudo pip install pygame
```

安装完成之后可以进入 **Python 的交互界面**，通过以下命令查看安装的 Pygame 版本。

```
> import pygame  
> pygame.ver
```

A terminal window titled "Terminal 终端 - python" with a menu bar containing "文件(E)", "编辑(E)", "视图(V)", "终端(T)", "标签(A)", and "帮助(H)". The terminal output shows the user running 'python' in the directory '~/' at 11:13:54. It displays the Python 2.7.6 version, GCC 4.8.2 on linux2, and prompts for help. The user then enters a series of commands: '>>> import pygame', '>>> pygame.ver', and '>>>'. The output for 'pygame.ver' is '1.9.2b8'. A red box highlights the import and version check commands. In the bottom right corner, there is a logo for "实验楼 shiyanlou.com".

```
shiyanolou:~/ $ python [11:13:54]
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pygame
>>> pygame.ver
'1.9.2b8'
>>>
```

参考：维基百科——Pygame

三、编程实现

首先进入目录 Code 创建项目文件夹 Drawing_Board，之后的项目文件都位于该文件夹之下。最终，我们完整的项目目录树为：

```
/home/shiyanlou/Code/Drawing_Board/
|-- Board.py
`-- images/
    |-- big.png
    |-- brush.png
    |-- pen1.png
    |-- pen2.png
    `-- small.png
```

说明：

- Board.py：画板程序，可以通过 vim 命令或者 gedit 编辑器进行创建和编辑。
- images：图片资源文件夹

本次实验所需用到的图像文件资源压缩包均可通过 wget 指令进行下载。

```
$ wget http://labfile.oss.aliyuncs.com/courses/674/images.zip
```

解压缩至 /home/shiyanlou/Code/Drawing_Board/ 目录中。

```
$ unzip images.zip
```

3.1 程序框架

首先先介绍一下程序的主体框架，所涉及的类，后边再逐一深入实现。

```
# -*- coding: utf-8 -*-
import pygame
from pygame.locals import *
import math

# 画笔类
class Brush:
    pass

# 菜单类
class Menu:
    pass

# 画板类
class Painter:
    pass

# 主函数
def main():
    pass

if __name__ == '__main__':
    main()
```

说明：

- Brush 类是画笔类。负责管理加载画笔笔刷，调整画笔颜色、大小以及画笔绘制的工作。
- Menu 类是菜单类。负责管理和绘制菜单。菜单的功能包括允许切换画笔，调整画笔大小和选择画笔颜色。
- Painter 类是画板类。是整个程序最核心的类，负责统筹其它两个类对象的调度以及事件的监听。
- main 函数负责创建 Painter 对象，并且让画板运行。

3.2 Brush 类实现

```
class Brush:
    def __init__(self, screen):
        """
        初始化函数
        """

    def start_draw(self, pos):
        """
        开始绘制，并记录当前坐标
        """

    def end_draw(self):
        """
        结束绘制
        """

    def set_brush_style(self, style):
        """
        设置笔刷的样式
        """

    def get_brush_style(self):
        """
        获取笔刷的类型
        """

    def get_current_brush(self):
        """
        获取当前笔刷
        """

    def set_size(self, size):
        """
        设置笔刷大小
        """

    def get_size(self):
        """
        获取笔刷大小
        """
```

```
def set_color(self, color):
    """
    设定笔刷颜色
    """

def get_color(self):
    """
    获取笔刷颜色
    """

def draw(self, pos):
    """
    绘制
    """

def _get_points(self, pos):
    """
    为了绘制的线条更加平滑，我们需要获取前一个点与当前点之间的所有需要绘制的点
    """
```

Brush 类是笔刷类，通过上方的声明我们发现它需要定义的方法还比较多，但是事实上实现起来都非常简单。

```
def __init__(self, screen):
    # pygame.Surface 对象
    self.screen = screen
    self.color = (0, 0, 0)
    # 初始时候默认设置画笔大小为 1
    self.size = 1
    self.drawing = False
    self.last_pos = None
    # 如果 style 是 True，则采用 png 笔刷
    # 若是 style 为 False，则采用一般的铅笔画笔
    self.style = True
    # 加载刷子的样式
    self.brush = pygame.image.load("images/brush.png").convert_alpha()
    self.brush_now = self.brush.subsurface((0, 0), (1, 1))
```

我们在初始化函数中定义了许多变量，也加载了笔刷的图片资源。

(介绍 PNG 图片作为笔刷的原因。。。)

```
def start_draw(self, pos):
    self.drawing = True
    self.last_pos = pos

def end_draw(self):
    self.drawing = False

def set_brush_style(self, style):
    print("* set brush style to", style)
    self.style = style

def get_brush_style(self):
    return self.style

def get_current_brush(self):
    return self.brush_now

def set_size(self, size):
    if size < 32:
        size = 32
    print("* set brush size to", size)
    self.size = size
    self.brush_now = self.brush.subsurface((0, 0), (size*2, size*2))

def get_size(self):
    return self.size

# 设定笔刷颜色
def set_color(self, color):
    self.color = color
    for i in range(self.brush.get_width()):
        for j in range(self.brush.get_height()):
            self.brush.set_at((i, j),
                               color + (self.brush.get_at((i, j)).a,))

def get_color(self):
    return self.color
```

这些函数的实现都比较简单。

```

# 绘制
def draw(self, pos):
    if self.drawing:
        for p in self._get_points(pos):
            if self.style:
                self.screen.blit(self.brush_now, p)
            else:
                pygame.draw.circle(self.screen, self.color, p, self.size)
        self.last_pos = pos

# 获取前一个点与当前点之间的所有需要绘制的点
def _get_points(self, pos):
    points = [(self.last_pos[0], self.last_pos[1])]
    len_x = pos[0] - self.last_pos[0]
    len_y = pos[1] - self.last_pos[1]
    length = math.sqrt(len_x**2 + len_y**2)
    step_x = len_x / length
    step_y = len_y / length
    for i in range(int(length)):
        points.append((points[-1][0] + step_x, points[-1][1] + step_y))
    # 对 points 中的点坐标进行四舍五入取整
    points = map(lambda x: (int(0.5 + x[0]), int(0.5 + x[1])), points)
    # 去除坐标相同的点
    return list(set(points))

```

`_get_points` 是通过对鼠标坐标前一次记录点与当前记录点之间进行线性插值，从而获得一系列点的坐标，从而使得绘制出来的笔刷痕迹更加平滑自然。

`draw` 函数则是通过获取 `_get_points` 计算所得的每个点进行逐一绘制。

3.3 Menu 类实现

```

class Menu:
    def __init__(self, screen):
        """
        初始化函数
        """

    def set_brush(self, brush):
        """
        设置画笔

```



```

"""

def draw(self):
    """
    绘制菜单栏
    """

def click_button(self, pos):
    """
    定义菜单按钮的点击响应事件
    """

```

Menu 类负责菜单功能的实现，代码实现起来要简单一些。

```

def __init__(self, screen):
    self.screen = screen
    self.brush = None
    # 画板预定义的颜色值
    self.colors = [
        (0xff, 0x00, 0xff), (0x80, 0x00, 0x80),
        (0x00, 0x00, 0xff), (0x00, 0x00, 0x80),
        (0x00, 0xff, 0xff), (0x00, 0x80, 0x80),
        (0x00, 0xff, 0x00), (0x00, 0x80, 0x00),
        (0xff, 0xff, 0x00), (0x80, 0x80, 0x00),
        (0xff, 0x00, 0x00), (0x80, 0x00, 0x00),
        (0xc0, 0xc0, 0xc0), (0xff, 0xff, 0xff),
        (0x00, 0x00, 0x00), (0x80, 0x80, 0x80),
    ]
    # 计算每个色块在画板中的坐标值，便于绘制
    self.colors_rect = []
    for (i, rgb) in enumerate(self.colors):
        rect = pygame.Rect(10 + i % 2 * 32, 254 + i / 2 * 32, 32, 32)
        self.colors_rect.append(rect)
    # 两种笔刷的按钮图标
    self.pens = [
        pygame.image.load("images/pen1.png").convert_alpha(),
        pygame.image.load("images/pen2.png").convert_alpha(),
    ]
    # 计算坐标，便于绘制
    self.pens_rect = []
    for (i, img) in enumerate(self.pens):
        rect = pygame.Rect(10, 10 + i * 64, 64, 64)

```

```

        self.pens_rect.append(rect)

# 调整笔刷大小的按钮图标
self.sizes = [
    pygame.image.load("images/big.png").convert_alpha(),
    pygame.image.load("images/small.png").convert_alpha()
]
# 计算坐标, 便于绘制
self.sizes_rect = []
for (i, img) in enumerate(self.sizes):
    rect = pygame.Rect(10 + i * 32, 138, 32, 32)
    self.sizes_rect.append(rect)

```

在初始化函数中, 我们不仅加载了图片资源, 同时还为每个按钮设定各自的坐标并存储在 list 中, 以方便后边绘制。

```

def set_brush(self, brush):
    self.brush = brush

# 绘制菜单栏
def draw(self):
    # 绘制画笔样式按钮
    for (i, img) in enumerate(self.pens):
        self.screen.blit(img, self.pens_rect[i].topleft)
    # 绘制 + - 按钮
    for (i, img) in enumerate(self.sizes):
        self.screen.blit(img, self.sizes_rect[i].topleft)
    # 绘制用于实时展示笔刷的小窗口
    self.screen.fill((255, 255, 255), (10, 180, 64, 64))
    pygame.draw.rect(self.screen, (0, 0, 0), (10, 180, 64, 64), 1)
    size = self.brush.get_size()
    x = 10 + 32
    y = 180 + 32
    # 如果当前画笔为 png 笔刷, 则在窗口中展示笔刷
    # 如果为铅笔, 则在窗口中绘制原点
    if self.brush.get_brush_style():
        x = x - size
        y = y - size
        self.screen.blit(self.brush.get_current_brush(), (x, y))
    else:
        # BUG
        pygame.draw.circle(self.screen,

```

```

        self.brush.get_color(), (x, y), size)

    # 绘制色块
    for (i, rgb) in enumerate(self.colors):
        pygame.draw.rect(self.screen, rgb, self.colors_rect[i])

    # 定义菜单按钮的点击响应
    def click_button(self, pos):
        # 笔刷
        for (i, rect) in enumerate(self.pens_rect):
            if rect.collidepoint(pos):
                self.brush.set_brush_style(bool(i))
                return True
        # 笔刷大小
        for (i, rect) in enumerate(self.sizes_rect):
            if rect.collidepoint(pos):
                # 画笔大小的每次改变量为 1
                if i:
                    self.brush.set_size(self.brush.get_size() - 1)
                else:
                    self.brush.set_size(self.brush.get_size() + 1)
                return True
        # 颜色
        for (i, rect) in enumerate(self.colors_rect):
            if rect.collidepoint(pos):
                self.brush.set_color(self.colors[i])
                return True
        return False

```

这里值得注意的是，我们使用了 `pygame.Rect.collidepoint` 函数来检测是否触发了某个菜单按钮。

`pygame.Rect.collidepoint((x, y))` 函数的使用非常简单，只要传入一个坐标元组作为参数，如果该点位于 `pygame.Rect` 对象中的话，函数返回 `True`，否则返回 `False`。

3.4 Painter 类实现

```

class Painter:
    def __init__(self):
        # 设置了画板窗口的大小与标题
        self.screen = pygame.display.set_mode((800, 600))
        pygame.display.set_caption("Painter")
        # 创建 Clock 对象

```

```
self.clock = pygame.time.Clock()
# 创建 Brush 对象
self.brush = Brush(self.screen)
# 创建 Menu 对象, 并设置了默认笔刷
self.menu = Menu(self.screen)
self.menu.set_brush(self.brush)

def run(self):
    self.screen.fill((255, 255, 255))
    # 程序的主体是一个循环, 不断对界面进行重绘, 直到监听到结束事件才结束循环
    while True:
        # 设置帧率
        self.clock.tick(30)
        # 监听事件
        for event in pygame.event.get():
            # 结束事件
            if event.type == QUIT:
                return
            # 键盘按键事件
            elif event.type == KEYDOWN:
                # 按下 ESC 键, 清屏
                if event.key == K_ESCAPE:
                    self.screen.fill((255, 255, 255))
            # 鼠标按下事件
            elif event.type == MOUSEBUTTONDOWN:
                # 若是当前鼠标位于菜单中, 则忽略掉该事件
                # 否则调用 start_draw 设置画笔的 drawing 标志为 True
                if event.pos[0] <= 74 and self.menu.click_button(event.pos):
                    pass
                else:
                    self.brush.start_draw(event.pos)
            # 鼠标移动事件
            elif event.type == MOUSEMOTION:
                self.brush.draw(event.pos)
            # 松开鼠标按键事件
            elif event.type == MOUSEBUTTONUP:
                # 调用 end_draw 设置画笔的 drawing 标志为 False
                self.brush.end_draw()
        # 绘制菜单按钮
        self.menu.draw()
        # 刷新窗口
        pygame.display.update()
```

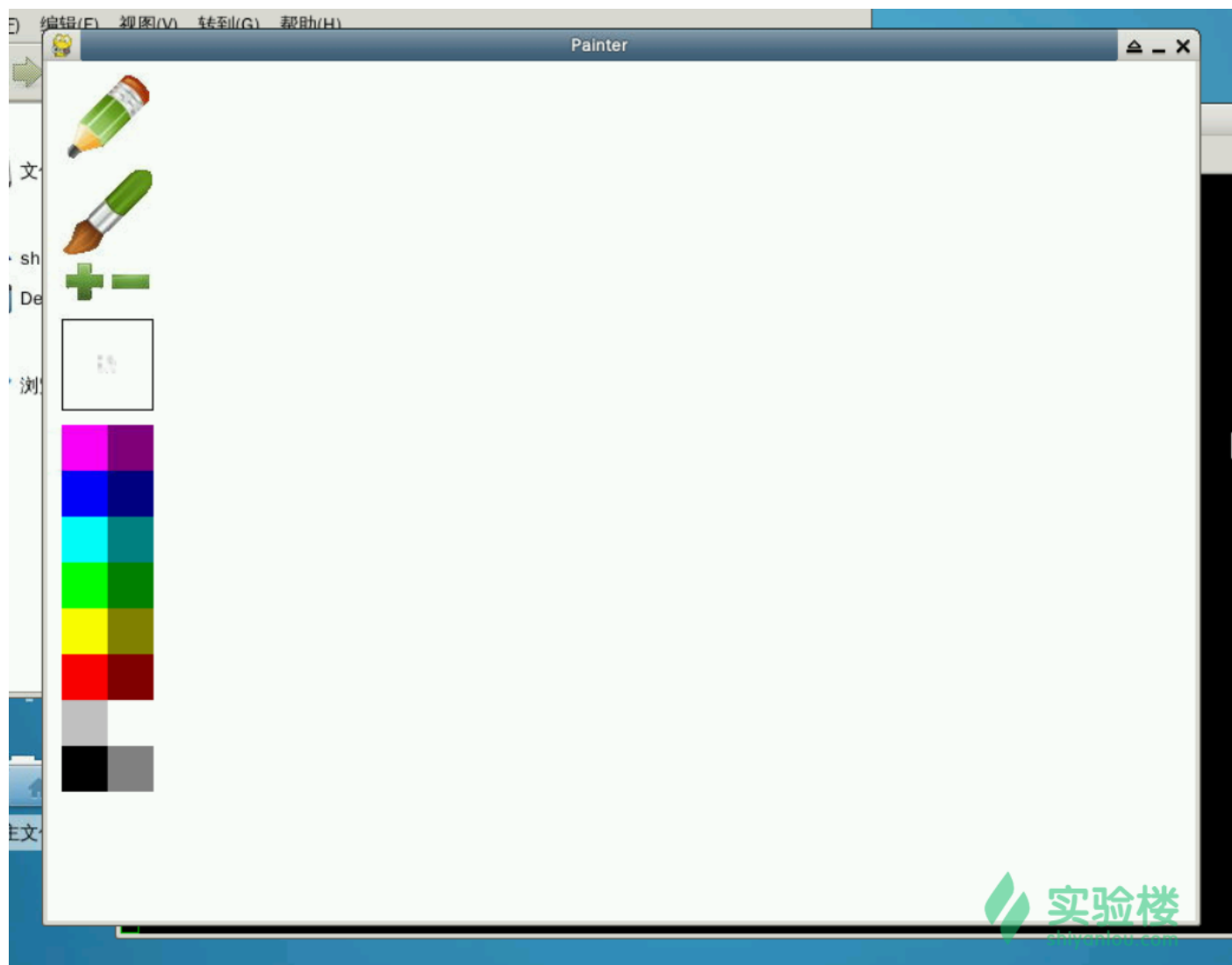
3.5 main 函数

```
# 主函数
def main():
    app = Painter()
    app.run()
```

四、运行

程序编写结束之后，执行以下语句来运行。

```
$ python Board.py
```



五、总结

通过这么课程我们实现了一个非常简单的画板，相信通过这门课程的学习之后大家对 Pygame 的使用也将更加熟悉。当然目前画板的功能还有待完善，大家可以试着在课后给画板添加些自己的功能。比如支持自定义颜色，支持更多的笔刷等。

另外本次实验的源代码可以通过以下方式获取：

```
$ wget http://labfile.oss.aliyuncs.com/courses/674/Board.py
```