

Министерство науки и высшего образования РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа киберфизических систем и управления

Лабораторная работа
по дисциплине «Кроссплатформенное программирование»
«Графический редактор»

Выполнил:

студент гр. 3530902/70201

_____ Матченко Т.И.

Проверил:

доцент, к.т.н.

_____ Хлопин С.В.

Санкт-Петербург

2021

Оглавление

Задание	3
Ход работы	3
Описание интерфейса	3
Описание инструментов.....	4
Описание работы программы	7
Выводы	10
Приложение	11

Задание

Создать кроссплатформенный графический редактор, позволяющий рисовать примитивы, проводить их закрашивание, перемещение. Результат работы программы должен позволять возможность сохранять в общепринятый графический формат.

Ход работы

Инструмент разработки: программа написана на Python 3 с использованием библиотек Tkinter для интерфейса и Pillow для обработки изображений.

Описание интерфейса:

- Верхняя панель:
 - Выбор размера кисти/толщины линий фигур;
 - Кнопка Fill, отвечающая за заливку рисуемых фигур;
- Область для рисования фиксированного размера;
- Нижняя панель:
 - Карандаш;
 - Примитивы (линия, прямоугольник, эллипс);
 - Очищение области рисования;
 - Выбор цвета;
 - Заливка;
 - Выбор фрагмента и затем его перемещение;
 - Сохранение в формате PNG;
 - Окно «About»;

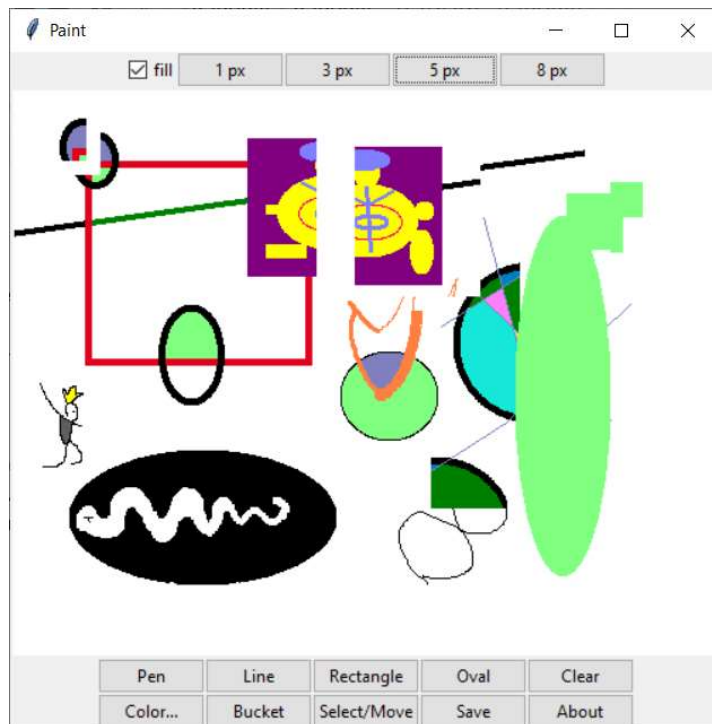
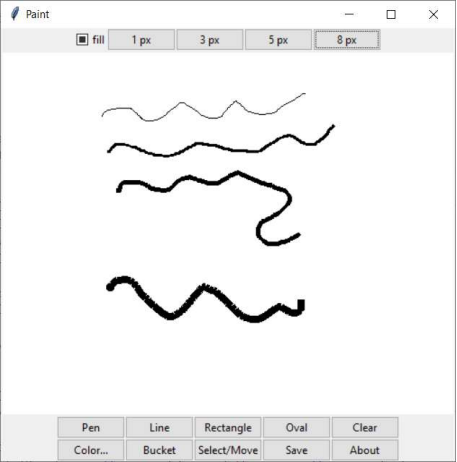
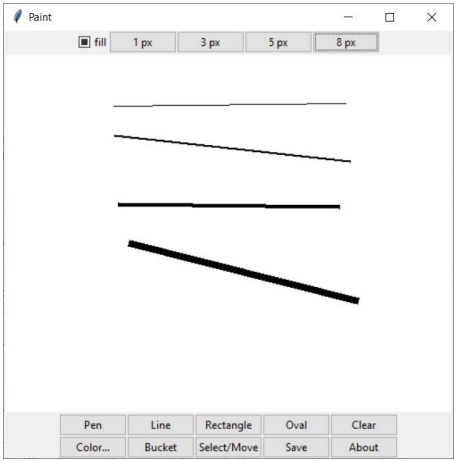
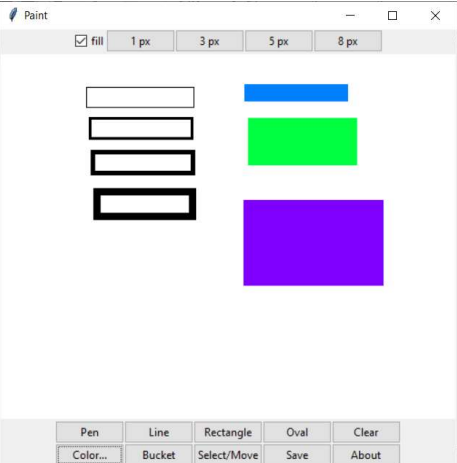


Рис. 1 – Пример работы программы

Описание инструментов:

Табл. 1 – Описание инструментов

<p>Карандаш.</p> <p>Зажать левую кнопку мыши и свободно рисовать.</p>	 <p>Рис. 2 – Пример работы карандаша</p>
<p>Линия.</p> <p>Проводит линию вслед за курсором.</p>	 <p>Рис. 3 – Пример работы линии</p>
<p>Прямоугольник.</p> <p>Если поставить галочку на Fill, будет залит выбранным цветом.</p>	 <p>Рис. 4 – Пример работы прямоугольника</p>

Овал.

Если поставить галочку на Fill, будет
залит активным цветом.

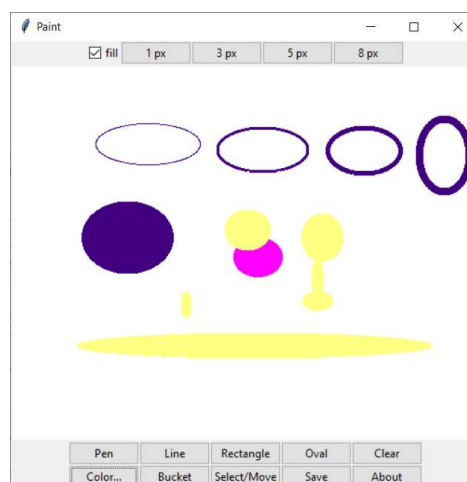


Рис. 5 – Пример работы овала

Заливка.

Заливает область выбранным цветом.

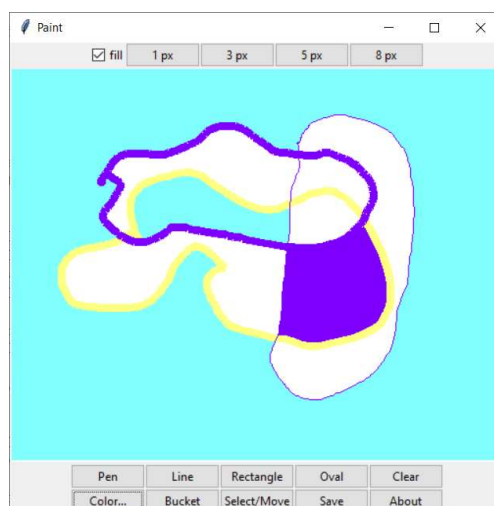


Рис. 6 – Пример работы заливки

Выбор цвета.

Здесь можно выбрать другой цвет.

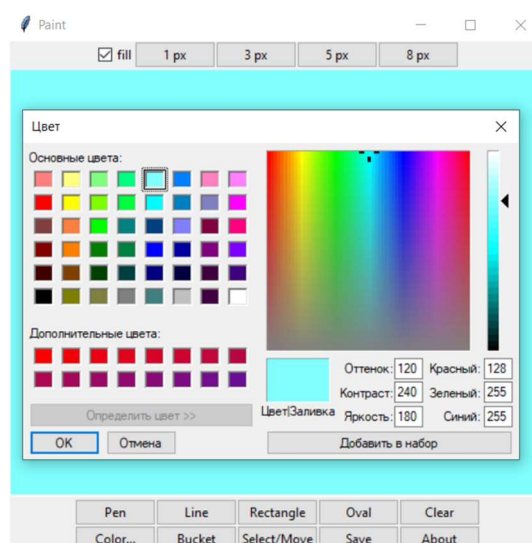


Рис. 7 – Пример выбора цвета

Выбор и перемещение.

Сначала с помощью зажатия левой кнопки мыши выбирается фрагмент. Затем с помощью кнопок «←», «→», «↑», «↓» осуществляется его перемещение.

До завершения перемещения фрагмента другие кнопки блокируются. Чтобы завершить перемещение, нужно нажать Enter.

На месте, где был фрагмент, останется белый прямоугольник.

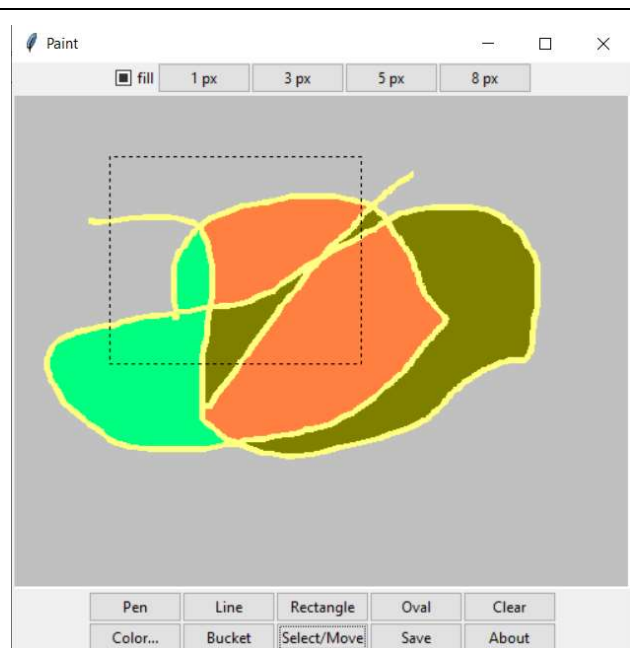


Рис. 8 – Пример выбора фрагмента

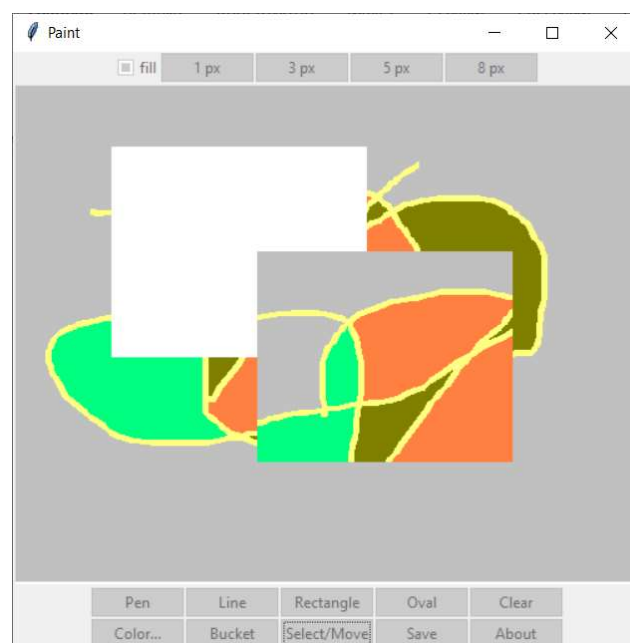


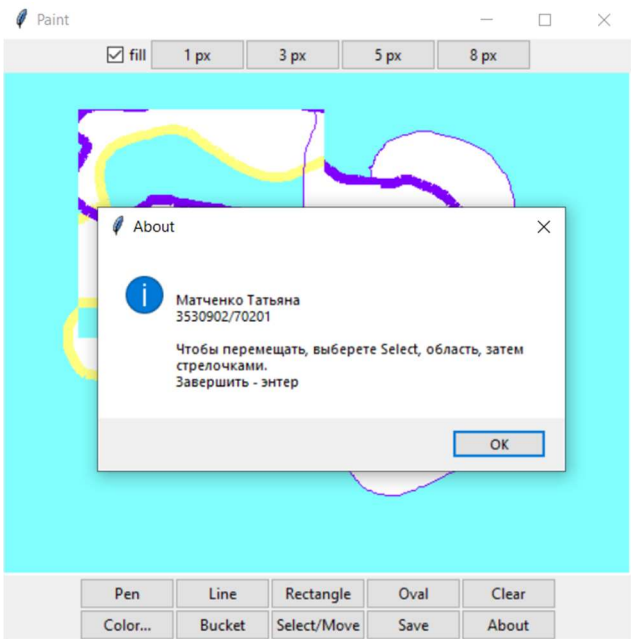
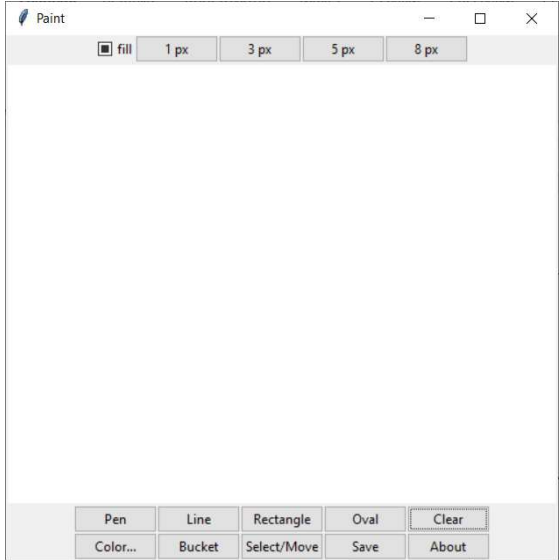
Рис. 9 – Перемещение фрагмента

Сохранение.

Сохраняет изображение в директорию с программой под именем image_N.png. N – число сделанных сохранений за сессию.

компьютер > Stuff (E:) > UniStuff > crossplatform		
Имя	Даты	Тип
image_0.png	12.04.2021 17:24	Файл "PNG"
paint.py	10.04.2021 19:11	Python File
отчет.docx	12.04.2021 17:19	Документ

Рис. 10 – Успешное сохранение рисунка

<p>About.</p> <p>Выводит сведения о приложении.</p>	 <p>Рис. 11 – Окно About</p>
<p>Очистить.</p> <p>Очищает область рисования.</p>	 <p>Рис. 12 – Чистый область рисования</p>

Описание работы программы

Библиотека tkinter обладает средствами построения примитивов (линия, прямоугольник, овал).

Однако область рисования tkinter'a оперирует объектами, а не растровым изображением, из-за чего затруднительно сделать заливку областей и выделение/перемещение целого фрагмента. Было решено одновременно работать и с Tkinter, и с библиотекой Pillow. Библиотека Pillow также поддерживает отрисовку примитивов, но не может менять их впоследствии, что нужно для красивой отрисовки

примитива «вслед за курсором», но позволяет легко сохранять рисунок и работать с пикселями.

Табл. 2 – Сравнение библиотек Tkinter и Pillow

Tkinter	Pillow
+ Может отрисовывать примитивы	+ Может отрисовывать примитивы
+ Может менять отрисованный примитив (изменять его координаты)	- Не может менять отрисованный примитив
- Не может сохранить изображение	+ Может сохранить изображение
- Не может работать с пикселями	+ Может работать с пикселями

Так как мы работаем одновременно с двумя библиотеками, рисовать придется на двух объектах одновременно: на области рисования библиотеки Tkinter и на изображении библиотеки Pillow.

Карандаш:

- Нажатие левой кнопки мыши: рисуем круг;
- Перемещение мыши (с зажатой левой кнопкой): рисуем короткие линии вслед за курсором.

Примитивы (линия, прямоугольник, овал):

- Нажатие левой кнопки мыши: рисуем выбранный примитив с началом и концом в одной точке, на данном этапе пользователь его не видит;
- Перемещение мыши (с зажатой левой кнопкой): меняем конечную точку примитива – рисуем примитив «вслед за курсором».

Заливка:

- Нажатие левой кнопки мыши: происходит заливка области выбранным цветом.

Выделить/переместить:

- Выделить фрагмент:
 - Нажатие левой кнопки мыши: рисуем пунктирный прямоугольник с началом и концом в одной точке, на данном этапе пользователь его не видит;
 - Перемещение мыши (с зажатой левой кнопкой): меняем конечную точку пунктирного прямоугольника – рисуем его «вслед за курсором»;
 - Отпустить левую кнопку мыши: завершение рисования пунктирного прямоугольника. Если снова зажать левую кнопку мыши и перемещать мышь, можно так же выбрать новую область. Если нажать одну из кнопок «←», «→», «↑», «↓», начнется перемещение фрагмента;
- Переместить фрагмент:
 - Нажатие кнопки «←», «→», «↑» или «↓»:
 - Первое нажатие кнопки:

- Стираем пунктирный прямоугольник;
- Блокируем остальные кнопки в приложении;
- Сохраняем фрагмент в отдельную переменную, на месте фрагмента на изображении рисуем белый прямоугольник;
- Рассчитываем новые координаты, исходя из направления;
- Перемещаем фрагмент по указанным координатам;
- Enter: Завершение перемещения.

Выводы

Результатом выполнения лабораторной работы стало создание графического редактора. Созданы возможности отрисовки свободной линии, примитивов, заливки областей, выделения и перемещения фрагмента.

В процессе работы получена практика работы с такими библиотеками Python, как Tkinter и Pillow.

Приложение является кроссплатформенным ввиду написания на Python и использования кроссплатформенных библиотек.

Приложение

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from tkinter import colorchooser
from PIL import Image, ImageDraw, ImageTk

class dFrame(ttk.Frame):
    #чтобы у фрейма была возможность отключить своих детей
    def enable(self, state='!disabled'):
        def cstate(widget):
            # Is this widget a container?
            if widget.winfo_children():
                # It's a container, so iterate through its children
                for w in widget.winfo_children():
                    w.state((state,)) # change its state
                    cstate(w) # and then recurse to process ITS children
        cstate(self)

    def disable(self):
        self.enable('disabled')

class PaintApp:

    def __init__(self, root):
        #переменные
        self.width = 500
        self.height = 400
        self.size = 5
        self.color_hex = '#000000'
        self.color_tuple = (0, 0, 0)
        self.action = 'pen'
        self.refresh_state = 1 # делать ли рефреш после отпускания мыши
        self.save_number = 0 # для нескольких сохранений за сессию
        self.filled = False # заполнять ли фигуру. контролируется галочкой
        self.movement_step = 3 # px
        self.pressed_keys = {}
        self.lockdown = False # are all buttons disabled? (for movement phase)
        self.drawing_select_rect = False

        #функции кнопок
        def save():
            # save_number increments by 1 at every save
            filename = f'image_{self.save_number}.png'
            self.canvas_image.save(filename)
            self.save_number += 1

        def set_action(var: str):
```

```

        self.action = var

    def set_size(var: int):
        self.size = var

    def choose_color():
        color_tmp = colorchooser.askcolor(color=self.color_hex)
        try:
            self.color_tuple = tuple(map(int, color_tmp[0]))
            self.color_hex = color_tmp[1]
        except TypeError:
            return

    def clear():
        self.canvas.delete('all')
        self.canvas_image = Image.new('RGB', (self.width, self.height), 'white')

        self.draw = ImageDraw.Draw(self.canvas_image)

    def refresh():
        #выпускаем имадж на свободу
        self.image_tk = ImageTk.PhotoImage(self.canvas_image)
        self.canvas.delete('all')
        self.canvas.create_image((0, 0), image=self.image_tk, anchor='nw')

    def switch_filled():
        self.filled = not self.filled

    #расставляем GUI
    #нижние кнопки (всяких разных действий)
    self.buttons_frame = dFrame(root) # фрейм для верхних кнопочек
    ttk.Button(self.buttons_frame, text="Pen", command=lambda: set_action('pen')).grid(row=0, column=0)
    ttk.Button(self.buttons_frame, text="Line", command=lambda: set_action('line')).grid(row=0, column=1)
    ttk.Button(self.buttons_frame, text="Rectangle", command=lambda: set_action('rect')).grid(row=0, column=2)
    ttk.Button(self.buttons_frame, text="Oval", command=lambda: set_action('oval')).grid(row=0, column=3)
    ttk.Button(self.buttons_frame, text="Clear", command=clear).grid(row=0, column=4)
    ttk.Button(self.buttons_frame, text="Color...", command=choose_color).grid(row=1, column=0)
    ttk.Button(self.buttons_frame, text="Bucket", command=lambda: set_action('bucket')).grid(row=1, column=1)
    ttk.Button(self.buttons_frame, text="Select/Move", command=lambda: set_action('select')).grid(row=1, column=2)
    ttk.Button(self.buttons_frame, text="Save", command=save).grid(row=1, column=3)

```

```

        ttk.Button(self.buttons_frame, text="About", command=lambda: messagebox
.showinfo(
    "About", "\nМатченко Татьяна\n3530902/70201\n\nЧтобы перемещать, вы
берете Select, область, затем стрелочками.\nЗавершить - энтер")
    ).grid(row=1, column=4)

    #верхние кнопки толщины и галочка заливки
    self.size_buttons_frame = dFrame(root) # фрейм для них
    ttk.Checkbutton(self.size_buttons_frame, text='fill', command=switch_fi
lled).grid(row=0, column=0) # галочка для заливки фигур
    ttk.Button(self.size_buttons_frame, text="1 px",command=lambda: set_siz
e(1)).grid(row=0, column=1)
    ttk.Button(self.size_buttons_frame, text="3 px",command=lambda: set_siz
e(3)).grid(row=0, column=2)
    ttk.Button(self.size_buttons_frame, text="5 px",command=lambda: set_siz
e(5)).grid(row=0, column=3)
    ttk.Button(self.size_buttons_frame, text="8 px",command=lambda: set_siz
e(8)).grid(row=0, column=4)
    self.size_buttons_frame.pack()

    #ставим канвас. канвас - это иллюзия и рабочая область. канвас для аним
ации.
    self.canvas = Canvas(root, width=self.width, height=self.height, bg='wh
ite')
    #создаем белый имадж. на нем рисуется то же, что и на канвасе. имадж не
видим.
    #имадж каждый раз копируется в канвас в рефреше
    self.canvas_image = Image.new('RGB', (self.width, self.height), 'white'
)

    #закрепляем за имаджем draw, рисуем с помощью него
    self.draw = ImageDraw.Draw(self.canvas_image)

    self.canvas.pack()
    self.buttons_frame.pack()

    #сохраненные координаты
    self.first_x, self.first_y = None, None
    self.last_x, self.last_y = None, None
    #для карандаша

    def create_circle(centre_x, centre_y):
        '''создает точку в указанном месте с диаметром self.size'''
        #высчитываем уголки
        topleft_x = centre_x - self.size/2
        topleft_y = centre_y - self.size/2
        bottomright_x = centre_x + self.size/2
        bottomright_y = centre_y + self.size/2
        #рисуем на имадже
        self.draw.ellipse((topleft_x, topleft_y, bottomright_x, bottomright
_y),

```

```

        fill=self.color_hex, outline=self.color_hex)

    #рисует на канвасе
    self.canvas.create_oval(topleft_x, topleft_y, bottomright_x, bottom
right_y,
                           fill=self.color_hex,outline=self.color_hex)

    #для заливки

    def is_in_bounds(x, y):
        '''Возвращает True, если выбранный пиксель находится в границах кан
васа'''
        return (x < self.width) and (x > 0) and (y < self.height) and (y >
0)

    def floodfill(x, y, desired_color: tuple):
        '''заливка'''
        #если пикнутый пиксель уже нужного цвета, нет смысла. выход
        source_color = self.canvas_image.getpixel((x, y))
        if source_color == desired_color:
            return
        to_check = set() # собираем сет пикселей, которые еще надо провери
ть

        checked = set() # и сет уже проверенных
        to_check.add((x, y)) # загоняем стартовый пиксель на проверку
        while len(to_check) != 0: # пока еще есть что проверять
            #перемещаем из непроверенных в проверенные
            (x, y) = to_check.pop()
            checked.add((x, y))

            if not is_in_bounds(x, y):
                continue # если проверяемый пиксель за границами канваса,
пропускаем

            current_color = self.canvas_image.getpixel((x, y)) # смотрим н
а цвет пикселя

            if not current_color == source_color:
                # если цвет пикселя отличен от источника, пропускаем. это г
раница.

                continue

            # красим пиксель в нужный цвет, если все ок
            self.canvas_image.putpixel((x, y), desired_color)

            #добавляем в сет соседей, если их нет в проверенных
            if not (x-1, y) in checked: to_check.add((x-1, y))
            if not (x+1, y) in checked: to_check.add((x+1, y))
            if not (x, y-1) in checked: to_check.add((x, y-1))
            if not (x, y+1) in checked: to_check.add((x, y+1))

    def on_press(event):
        if not self.action == 'move':
            refresh()

```

```

if self.action == 'pen':
    #при нажатии создаем точку и запоминаем координаты
    #запоминать нужно, потому что потом поведем от сюда микролинии
    self.last_x, self.last_y = event.x, event.y
    create_circle(self.last_x, self.last_y)

elif self.action == 'bucket': # запускаем покраску в пиксель
    floodfill(event.x, event.y, self.color_tuple)
    refresh()

elif self.action == 'line':
    self.first_x, self.first_y = event.x, event.y
    self.last_x, self.last_y = event.x, event.y
    self.line = self.canvas.create_line(self.first_x, self.first_y,
self.first_x, self.first_y,
fill=self.color_hex, width=
self.size)
    self.canvas.itemconfigure(self.line, state='hidden')

elif self.action == 'rect':
    self.first_x, self.first_y = event.x, event.y
    self.last_x, self.last_y = event.x, event.y
    filling = self.color_hex if self.filled else ''
    self.rect = self.canvas.create_rectangle(self.first_x, self.first_y,
self.last_x, self.last_y,
fill=filling, outline=
self.color_hex, width=self.size)
    self.canvas.itemconfigure(self.rect, state='hidden')

elif self.action == 'oval':
    self.first_x, self.first_y = event.x, event.y
    self.last_x, self.last_y = event.x, event.y
    filling = self.color_hex if self.filled else ''
    self.oval = self.canvas.create_oval(self.first_x, self.first_y,
self.last_x, self.last_y,
fill=filling, outline=self.
color_hex, width=self.size)
    self.canvas.itemconfigure(self.oval, state='hidden')

elif self.action == 'select':
    self.first_x, self.first_y = event.x, event.y
    self.last_x, self.last_y = event.x, event.y
    self.select_rect = self.canvas.create_rectangle(self.first_x, self.first_y,
self.last_x, self.last_y,
dash=(2, 2), fill=
'', outline='black')
    self.canvas.itemconfigure(self.select_rect, state='hidden')

def on_motion(event):
    if self.action == 'pen':

```

```

        #делаем кисть через микролинии
        current_x, current_y = event.x, event.y
        self.canvas.create_line((self.last_x, self.last_y, current_x, c
urrent_y),
                                width=self.size, fill=self.color_hex)
        self.draw.line((self.last_x, self.last_y, current_x, current_y)
,
                        fill=self.color_hex, width=self.size)
        self.last_x, self.last_y = current_x, current_y

    elif self.action == 'line':
        #анимируем отрисовку линии
        self.last_x, self.last_y = event.x, event.y
        self.canvas.itemconfigure(self.line, state='normal')
        self.canvas.coords(self.line, self.first_x,
                            self.first_y, self.last_x, self.last_y)

    elif self.action == 'rect':
        #анимируем отрисовку прямоугольника
        self.last_x, self.last_y = event.x, event.y
        self.canvas.itemconfigure(self.rect, state='normal')
        self.canvas.coords(self.rect, self.first_x,
                            self.first_y, self.last_x, self.last_y)

    elif self.action == 'oval':
        #анимируем отрисовку овала
        self.last_x, self.last_y = event.x, event.y
        self.canvas.itemconfigure(self.oval, state='normal')
        self.canvas.coords(self.oval, self.first_x,
                            self.first_y, self.last_x, self.last_y)

    elif self.action == 'select':
        #анимируем отрисовку прямоугольника выделения
        self.drawing_select_rect = True
        self.last_x, self.last_y = event.x, event.y
        self.canvas.itemconfigure(self.select_rect, state='normal')
        self.canvas.coords(self.select_rect, self.first_x,
                            self.first_y, self.last_x, self.last_y)

    def order_coordinates():
        #делаем так, чтобы first был topleft и last был bottomright
        if (self.first_x > self.last_x) and (self.first_y < self.last_y):
            self.first_x, self.last_x = self.last_x, self.first_x
        elif (self.first_x < self.last_x) and (self.first_y > self.last_y):
            self.first_y, self.last_y = self.last_y, self.first_y
        elif (self.first_x > self.last_x) and (self.first_y > self.last_y):
            self.first_x, self.last_x = self.last_x, self.first_x
            self.first_y, self.last_y = self.last_y, self.first_y

    def on_release(event):

```



```

        if self.refresh_state and self.action == 'pen':
            refresh()

        elif self.action == 'line':
            if self.canvas.itemcget(self.line, 'state') == 'normal':
                self.draw.line((self.first_x, self.first_y, self.last_x, se
lf.last_y),
                               fill=self.color_hex, width=self.size)
            refresh()

        elif self.action == 'rect':
            if self.canvas.itemcget(self.rect, 'state') == 'normal':
                filling = self.color_hex if self.filled else None
                self.draw.rectangle((self.first_x, self.first_y, self.last_
x, self.last_y),
                                   fill=filling, outline=self.color_hex, w
idth=self.size)
            refresh()

        elif self.action == 'oval':
            if self.canvas.itemcget(self.oval, 'state') == 'normal':
                order_coordinates()
                filling = self.color_hex if self.filled else None
                self.draw.ellipse((self.first_x, self.first_y, self.last_x,
self.last_y),
                                  fill=filling, outline=self.color_hex, wid
th=self.size)
            refresh()

        elif self.action == 'select':
            if self.canvas.itemcget(self.select_rect, 'state') == 'normal':
                order_coordinates()
                self.drawing_select_rect = False
            else:
                refresh()

        #для передвижения куска
        def cut_and_fake():
            '''вырезаем часть изображения, но пока оставляем ее в
            рабочей области - канвасе. иллюзия.\n\n
            на изображении ставим белый прямоугольник.'''
            self.image_crop = self.canvas_image.crop((self.first_x, self.first_
y, self.last_x+1, self.last_y+1))
            self.crop_height, self.crop_width = self.image_crop.size
            #рисует белый прямоугольник - место преступления
            self.draw.rectangle((self.first_x, self.first_y, self.last_x, self.
last_y),
                                fill='white', outline='white', width=self.size)
            #временные координаты (левый верхний край) куска - его самые первые
            self.current_x, self.current_y = self.first_x, self.first_y

```

```

def move_crop_on_canvas():
    '''перемещение" кусочка по канвасу.\n\n
    на самом деле он телепортируется - мы его стираем,
    чтобы поставить в новом месте.'''
    delta_x, delta_y = 0, 0
    #смотрим, какая клавиша зажата.в зависимости от этого меняем дельту
    if 'Right' in self.pressed_keys: delta_x += self.movement_step
    elif 'Left' in self.pressed_keys: delta_x -= self.movement_step
    elif 'Down' in self.pressed_keys: delta_y += self.movement_step
    elif 'Up' in self.pressed_keys: delta_y -= self.movement_step
    #смотрим, чтобы середина куска случайно не вылезла за рамки
    centre_x = self.current_x + self.crop_width/2 + delta_x
    centre_y = self.current_y + self.crop_height/2 + delta_y
    if is_in_bounds(centre_x, centre_y):
        #если не вылезет, ставим новые временные координаты
        self.current_x += delta_x
        self.current_y += delta_y
        #очищаем канвас, вставляем старый имадж (где вырезан кусок)
        refresh()
        #теперь на этот канвас вставляем наш "перемещающийся" кусок
        #канвас все еще иллюзия.мы его вообще юзаем чисто ради анимации
        self.tmp_im = ImageTk.PhotoImage(self.image_crop)
        self.canvas.create_image((self.current_x, self.current_y), image=self.tmp_im, anchor='nw')

def paste_crop_to_img():
    '''вставляем кусок на имадж в подобранные координаты'''
    self.canvas_image.paste(self.image_crop, (self.current_x, self.current_y))
    refresh()

def set_lockdown(mode: bool):
    '''(раз)блокирует все кнопочки.\n\n
    для завершения фазы перемещения нужно обязательно нажать энтер'''
    if mode:
        #блокируем!
        self.buttons_frame.disable()
        self.size_buttons_frame.disable()
        self.lockdown = True
    else:
        #разблокируем!
        self.buttons_frame.enable()
        self.size_buttons_frame.enable()
        self.lockdown = False

def key_press(event):
    if self.action == 'select' and not self.drawing_select_rect:
        self.action = 'move'
    if self.action == 'move':

```

```

        if not self.lockdown:
            #первое перемещение
            refresh() # стереть прямоугольник выделения
            set_lockdown(True) # нужно заблокировать кнопки
            cut_and_fake() # и вырезать кусок
            self.pressed_keys[event.keysym] = True
            move_crop_on_canvas()

def key_release(event):
    self.pressed_keys.pop(event.keysym, None)

def stop_movement(event):
    try:
        paste_crop_to_img()
    finally:
        set_lockdown(False)
        refresh()
        self.action = 'select'

#закрепляем функции за событиями
self.canvas.bind("<ButtonPress-1>", on_press)
self.canvas.bind("<B1-Motion>", on_motion)
self.canvas.bind('<ButtonRelease-1>', on_release)
root.bind("<KeyPress>", key_press)
root.bind("<KeyRelease>", key_release)
root.bind("<Return>", stop_movement)
self.buttons_frame.pack()

def main():
    root = Tk()
    root.title("Paint")
    app = PaintApp(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```