МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”
КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Лабораторная работа №7
По дисциплине “**Современные платформы программирования**”
Тема: “**Оконные приложения**”

Выполнил:
студент группы ПО-11
Надежук А.Г.
Проверил:
Козик И. Д.

Брест 2025

Цель: освоить возможности языка программирования Python в разработке оконных приложений.

Вариант 3

Ход работы

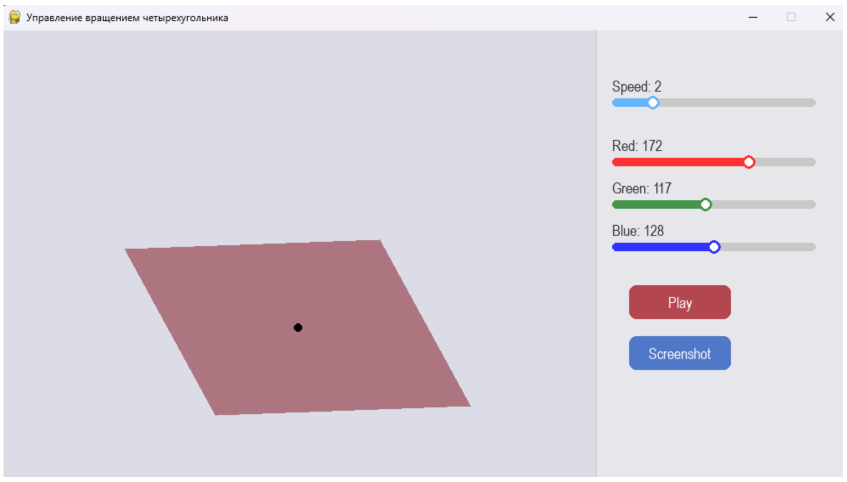
Задание 1. Построение графических примитивов и надписей. Изобразить четырехугольник, вращающийся в плоскости формы вокруг своего центра тяжести.

Требования к выполнению:

- Реализовать соответствующие классы, указанные в задании;
- Организовать ввод параметров для создания объектов (использовать экранные компоненты);
- Осуществить визуализацию графических примитивов.
- Должна быть предусмотрена возможность приостановки выполнения визуализации, изменения параметров «на лету» и снятия скриншотов с сохранением в текущую активную директорию. Для всех динамических сцен необходимо задавать параметр скорости!

Код программы:
```
class Quadrilateral:
    def __init__(self, vertices, color, rotation_speed):
        self.original_vertices = np.array(vertices, dtype=float)
        self.vertices = np.array(vertices, dtype=float)
        self.color = color
        self.rotation_speed = rotation_speed
        self.angle = 0
        self.center = self.calculate_center()
        self.paused = False
        self.dragging = False
        self.drag_offset = [0, 0]

    def calculate_center(self):
        return np.mean(self.original_vertices, axis=0)

    def rotate(self):
        if not self.paused and not self.dragging:
            self.angle += self.rotation_speed
            theta = np.radians(self.angle)
            rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta),
np.cos(theta)]])
            centered = self.original_vertices - self.center
            rotated = np.dot(centered, rotation_matrix)
            self.vertices = rotated + self.center

    def draw(self, surface):
        pygame.draw.polygon(surface, self.color, self.vertices)
        pygame.draw.circle(surface, (0, 0, 0), (int(self.center[0]), int(self.center[1])), 5)


class App:
    def __init__(self):
        pygame.init()
        self.width, self.height = 1000, 700
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption("Управление вращением четырехугольника")
        self.clock = pygame.time.Clock()
        self.font = pygame.font.SysFont('Arial', 18)
        self.title_font = pygame.font.SysFont('Arial', 24, bold=True)
        self.ui_panel_width = 300
        self.ui_panel_rect = pygame.Rect(self.width - self.ui_panel_width, 0,
self.ui_panel_width, self.height)
        quad_size = 200
        active_width = self.width - self.ui_panel_width
        center_x, center_y = active_width // 2, self.height // 2
        self.quad = Quadrilateral(
            vertices = [
                [center_x - quad_size, center_y - quad_size // 2],
                [center_x + quad_size // 2, center_y - quad_size // 2],
                [center_x + quad_size, center_y + quad_size // 2],
                [center_x - quad_size // 2, center_y + quad_size // 2]
            ],
            color = (172, 117, 128),
            rotation_speed = 2
        )
        self.speed_slider = {'rect': pygame.Rect(self.width - 280, 80, 240, 10), 'min': 0, 'max': 
10, 'value': 2, 'dragging': False}
        self.color_sliders = {
            'R': {'rect': pygame.Rect(self.width - 280, 150, 240, 10), 'min': 0, 'max': 255, 
'value': 172, 'dragging': False},
            'G': {'rect': pygame.Rect(self.width - 280, 200, 240, 10), 'min': 0, 'max': 255, 
'value': 117, 'dragging': False},
            'B': {'rect': pygame.Rect(self.width - 280, 250, 240, 10), 'min': 0, 'max': 255, 
'value': 128, 'dragging': False}
        }
        button_width = 120
        self.pause_button = pygame.Rect(self.width - 260, 300, button_width, 40)
        self.screenshot_button = pygame.Rect(self.width - 260, 360, button_width, 40)
        self.running = True
        self.background = (240, 240, 245)
        self.panel_color = (230, 230, 235)
        self.active_area_color = (220, 220, 230)

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if self.speed_slider['rect'].collidepoint(event.pos):
                        self.speed_slider['dragging'] = True
                        self.update_slider_value(self.speed_slider, event.pos[0])
                    for color, slider in self.color_sliders.items():
                        if slider['rect'].collidepoint(event.pos):
                            slider['dragging'] = True
                            self.update_slider_value(slider, event.pos[0])
                    if self.pause_button.collidepoint(event.pos):
                        self.quad.paused = not self.quad.paused
                    if self.screenshot_button.collidepoint(event.pos):
                        self.take_screenshot()
            elif event.type == pygame.MOUSEBUTTONUP:
                if event.button == 1:
                    self.speed_slider['dragging'] = False
                    for slider in self.color_sliders.values():
                        slider['dragging'] = False
            elif event.type == pygame.MOUSEMOTION:
                if self.speed_slider['dragging']:
                    self.update_slider_value(self.speed_slider, event.pos[0])
                    self.quad.rotation_speed = self.speed_slider['value']
                for color, slider in self.color_sliders.items():
                    if slider['dragging']:
                        self.update_slider_value(slider, event.pos[0])
                        self.quad.color = (
                            self.color_sliders['R']['value'],
                            self.color_sliders['G']['value'],
                            self.color_sliders['B']['value']
                        )

    def update_slider_value(self, slider, x_pos):
        slider_x = slider['rect'].x
        slider_width = slider['rect'].width
        relative_x = max(0, min(x_pos - slider_x, slider_width))
        slider['value'] = slider['min'] + (slider['max'] - slider['min']) * (relative_x / 
slider_width)

    def take_screenshot(self):
        screenshot_dir = "screenshots"
        if not os.path.exists(screenshot_dir):
            os.makedirs(screenshot_dir)
        i = 1
        while True:
            filename = os.path.join(screenshot_dir, f"screenshot_{i}.png")
            if not os.path.exists(filename):
                break
            i += 1
        pygame.image.save(self.screen, filename)
        print(f"Скриншот сохранен как {filename}")

    def draw_rounded_rect(self, surface, rect, color, radius=5):
        pygame.draw.rect(surface, color, rect, border_radius=radius)

    def draw_slider(self, surface, slider, label, color):
        pygame.draw.rect(surface, (200, 200, 200), slider['rect'], border_radius=5)
        active_width = (slider['value'] - slider['min']) / (slider['max'] - slider['min']) *
slider['rect'].width
        active_rect = pygame.Rect(slider['rect'].x, slider['rect'].y, active_width,
slider['rect'].height)
        pygame.draw.rect(surface, color, active_rect, border_radius=5)
        handle_pos = slider['rect'].x + active_width
        pygame.draw.circle(surface, color, (int(handle_pos), slider['rect'].centery), 8)
        pygame.draw.circle(surface, (255, 255, 255), (int(handle_pos), slider['rect'].centery),
5)
        label_text = self.font.render(f"{label}: {int(slider['value'])}", True, (50, 50, 50))
        surface.blit(label_text, (slider['rect'].x, slider['rect'].y - 25))

    def draw_button(self, surface, rect, text, color, hover_color):
        mouse_pos = pygame.mouse.get_pos()
        is_hovered = rect.collidepoint(mouse_pos)
        current_color = hover_color if is_hovered else color
        self.draw_rounded_rect(surface, rect, current_color, 8)
        text_surf = self.font.render(text, True, (255, 255, 255))
        text_rect = text_surf.get_rect(center=rect.center)
        surface.blit(text_surf, text_rect)
        return is_hovered
    def draw_ui(self):
        self.draw_rounded_rect(self.screen, self.ui_panel_rect, self.panel_color, 0)
        pygame.draw.line(self.screen, (210, 210, 210),(self.ui_panel_rect.x,
0),(self.ui_panel_rect.x, self.height), 2)
        self.draw_slider(self.screen, self.speed_slider, "Speed", (100, 180, 255))
        self.draw_slider(self.screen, self.color_sliders['R'], "Red", (255, 50, 50))
        self.draw_slider(self.screen, self.color_sliders['G'], "Green", (68, 148, 74))
        self.draw_slider(self.screen, self.color_sliders['B'], "Blue", (50, 50, 255))
        pause_text = "Pause" if not self.quad.paused else "Play"
        pause_color = (70, 180, 80) if not self.quad.paused else (180, 70, 80)
        self.draw_button(self.screen, self.pause_button, pause_text, pause_color, (90, 200, 100)
if not self.quad.paused else (200, 90, 100))
        self.draw_button(self.screen, self.screenshot_button, "Screenshot",(80, 120, 200), (100,
140, 220))
    def run(self):
        while self.running:
            self.handle_events()
            self.screen.fill(self.background)
            active_area = pygame.Rect(0, 0, self.width - self.ui_panel_width, self.height)
            pygame.draw.rect(self.screen, self.active_area_color, active_area)

self.quad.rotate()

self.quad.draw(self.screen)
            self.draw_ui()

pygame.display.flip()

self.clock.tick(60)
        pygame.quit()
        sys.exit()

if __name__ == "__main__":
    app = App()
    app.run()
```

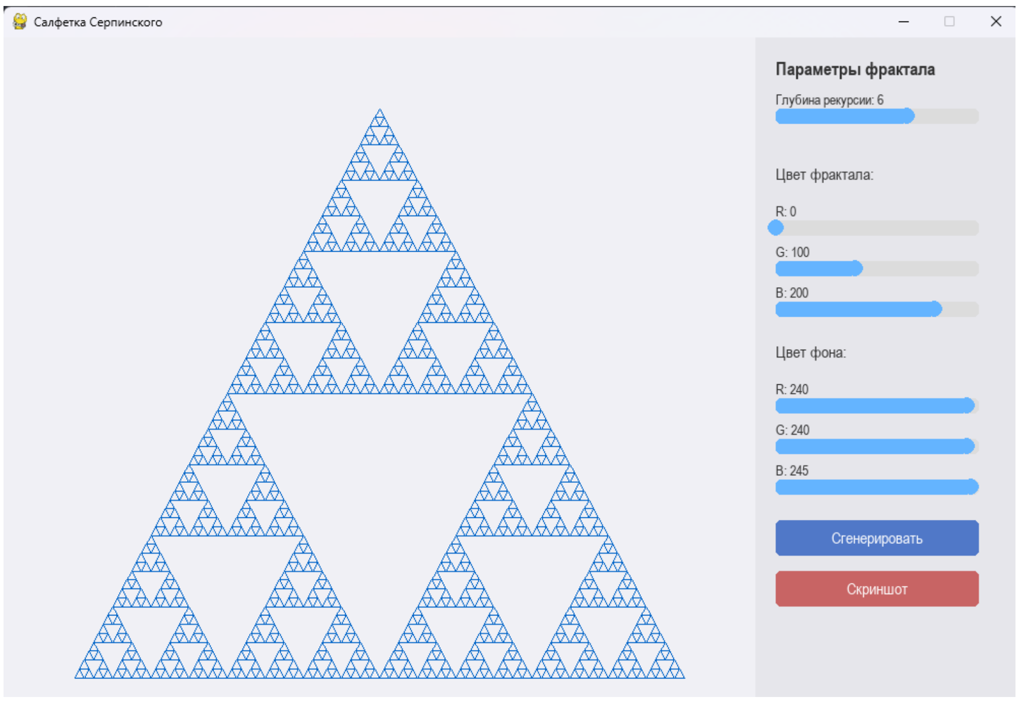
Результат выполнения:



Задание 2. Реализовать построение заданного типа фрактала. Везде, где это необходимо, предусмотреть ввод параметров, влияющих на внешний вид фрактала. Треугольная салфетка Серпинского.

Код программы:

```
class SierpinskiTriangle:
    def __init__(self, depth, color, bg_color):
        self.depth = depth
        self.color = color
        self.bg_color = bg_color
        self.points = []

    def generate(self, surface):
        width = surface.get_width() - 260
        height = surface.get_height()
        margin = min(width, height) * 0.1
        A = (width // 2, margin)
        B = (margin, height - margin)
        C = (width - margin, height - margin)
        self.points = []
        self._generate_fractal(A, B, C, self.depth)

    def _generate_fractal(self, A, B, C, depth):
        if depth == 0:
            self.points.extend([A, B, C])
        else:
            AB = ((A[0] + B[0]) / 2, (A[1] + B[1]) / 2)
            BC = ((B[0] + C[0]) / 2, (B[1] + C[1]) / 2)
            AC = ((A[0] + C[0]) / 2, (A[1] + C[1]) / 2)
            self._generate_fractal(A, AB, AC, depth - 1)
            self._generate_fractal(AB, B, BC, depth - 1)
            self._generate_fractal(AC, BC, C, depth - 1)

    def draw(self, surface):
        surface.fill(self.bg_color)
        for i in range(0, len(self.points), 3):
            if i + 2 < len(self.points):
                pygame.draw.polygon(surface, self.color,[self.points[i], self.points[i + 1],
self.points[i + 2]], 1)

class App:
    def __init__(self):
        pygame.init()
        self.width, self.height = 1000, 700
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption("Салфетка Серпинского")
        self.clock = pygame.time.Clock()
        self.small_font = pygame.font.SysFont('Arial', 14)
        self.font = pygame.font.SysFont('Arial', 16)
        self.title_font = pygame.font.SysFont('Arial', 18, bold=True)
        self.fractal = SierpinskiTriangle(
            depth=3,
            color=(0, 100, 200),
            bg_color=(240, 240, 245)
        )
        self.fractal.generate(self.screen)
        self.ui_panel_width = 260
        self.ui_panel_rect = pygame.Rect(self.width - self.ui_panel_width, 0,
self.ui_panel_width, self.height)
        self.elements = []
        y_pos = 20
        self.elements.append(("title", "Параметры фрактала", y_pos))
        y_pos += 50
        self.depth_slider = self.create_slider("Глубина рекурсии", 0, 10, 3, y_pos)
        y_pos += 55
        self.elements.append(("subtitle", "Цвет фрактала:", y_pos))
        y_pos += 55
        self.color_sliders = {
            'R': self.create_slider("R", 0, 255, 0, y_pos),
            'G': self.create_slider("G", 0, 255, 100, y_pos + 40),
            'B': self.create_slider("B", 0, 255, 200, y_pos + 80)
        }
        y_pos += 120
        self.elements.append(("subtitle", "Цвет фона:", y_pos))
        y_pos += 55
        self.bg_color_sliders = {
            'R': self.create_slider("R", 0, 255, 240, y_pos),
            'G': self.create_slider("G", 0, 255, 240, y_pos + 40),
            'B': self.create_slider("B", 0, 255, 245, y_pos + 80)
        }
        y_pos += 120
        self.generate_button = self.create_button("Сгенерировать", (80, 120, 200), y_pos)
        y_pos += 50
        self.screenshot_button = self.create_button("Скриншот", (200, 100, 100), y_pos)
        y_pos += 70
        self.running = True

    def create_slider(self, label, min_val, max_val, default_val, y_pos):
        return {
            'rect': pygame.Rect(self.width - 240, y_pos, 200, 15),
            'min': min_val,
            'max': max_val,
            'value': default_val,
            'dragging': False,
            'label': label
        }

    def create_button(self, text, color, y_pos):
        return {
            'rect': pygame.Rect(self.width - 240, y_pos, 200, 35),
            'color': color,
            'text': text
        }

    def handle_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if self.depth_slider['rect'].collidepoint(event.pos):
                        self.depth_slider['dragging'] = True
                        self.update_slider_value(self.depth_slider, event.pos[0])

                    for slider in [*self.color_sliders.values(),
*self.bg_color_sliders.values()]:
                        if slider['rect'].collidepoint(event.pos):
                            slider['dragging'] = True
                            self.update_slider_value(slider, event.pos[0])

                    if self.generate_button['rect'].collidepoint(event.pos):
                        self.generate_fractal()

                    if self.screenshot_button['rect'].collidepoint(event.pos):
                        self.take_screenshot()

            elif event.type == pygame.MOUSEBUTTONUP:
                if event.button == 1:
                    self.depth_slider['dragging'] = False
                    for slider in [*self.color_sliders.values(),
*self.bg_color_sliders.values()]:
                        slider['dragging'] = False

            elif event.type == pygame.MOUSEMOTION:
                if self.depth_slider['dragging']:
                    self.update_slider_value(self.depth_slider, event.pos[0])

                for slider in [*self.color_sliders.values(), *self.bg_color_sliders.values()]:
                    if slider['dragging']:
                        self.update_slider_value(slider, event.pos[0])
    def update_slider_value(self, slider, x_pos):
        slider_x = slider['rect'].x
        slider_width = slider['rect'].width
        relative_x = max(0, min(x_pos - slider_x, slider_width))
        slider['value'] = slider['min'] + (slider['max'] - slider['min']) * (relative_x /
slider_width)

    def generate_fractal(self):
        self.fractal.depth = int(self.depth_slider['value'])
        self.fractal.color = (
            self.color_sliders['R']['value'],
            self.color_sliders['G']['value'],
            self.color_sliders['B']['value']
        )
        self.fractal.bg_color = (
            self.bg_color_sliders['R']['value'],
            self.bg_color_sliders['G']['value'],
            self.bg_color_sliders['B']['value']
        )
        self.fractal.generate(self.screen)

    def take_screenshot(self):
        screenshot_dir = "screenshots"
        if not os.path.exists(screenshot_dir):
            os.makedirs(screenshot_dir)
        i = 1
        while True:
            filename = os.path.join(screenshot_dir, f"sierpinski_{i}.png")
            if not os.path.exists(filename):
                break
            i += 1
        pygame.image.save(self.screen, filename)
        print(f"Скриншот сохранен как {filename}")

    def draw_slider(self, slider):
        pygame.draw.rect(self.screen, (220, 220, 220), slider['rect'], border_radius=5)
        active_width = (slider['value'] - slider['min']) / (slider['max'] - slider['min']) *
slider['rect'].width
        active_rect = pygame.Rect(slider['rect'].x, slider['rect'].y, active_width,
slider['rect'].height)
        pygame.draw.rect(self.screen, (100, 180, 255), active_rect, border_radius=5)
        handle_pos = slider['rect'].x + active_width
        pygame.draw.circle(self.screen, (100, 180, 255), (int(handle_pos),
slider['rect'].centery), 8)
        value_text = self.small_font.render(f"{slider['label']}: {int(slider['value'])}", True,
(50, 50, 50))
        self.screen.blit(value_text, (slider['rect'].x, slider['rect'].y - 18))

    def draw_button(self, button):
        mouse_pos = pygame.mouse.get_pos()
        is_hovered = button['rect'].collidepoint(mouse_pos)
        color = button['color']
        hover_color = tuple(min(c + 30, 255) for c in color)
        current_color = hover_color if is_hovered else color
        pygame.draw.rect(self.screen, current_color, button['rect'], border_radius=5)
        text_surf = self.font.render(button['text'], True, (255, 255, 255))
        text_rect = text_surf.get_rect(center=button['rect'].center)
        self.screen.blit(text_surf, text_rect)

    def draw_ui(self):
        pygame.draw.rect(self.screen, (230, 230, 235), self.ui_panel_rect)
        for element in self.elements:
            elem_type, text, y_pos = element
            if elem_type == "title":
                text_surf = self.title_font.render(text, True, (50, 50, 50))
                self.screen.blit(text_surf, (self.width - self.ui_panel_width + 20, y_pos))
            elif elem_type == "subtitle":
                text_surf = self.font.render(text, True, (50, 50, 50))
                self.screen.blit(text_surf, (self.width - self.ui_panel_width + 20, y_pos))
            elif elem_type == "instruction":
                text_surf = self.small_font.render(text, True, (100, 100, 100))
                self.screen.blit(text_surf, (self.width - self.ui_panel_width + 20, y_pos))
        self.draw_slider(self.depth_slider)
        for slider in self.color_sliders.values():
            self.draw_slider(slider)
        for slider in self.bg_color_sliders.values():
            self.draw_slider(slider)
        self.draw_button(self.generate_button)
        self.draw_button(self.screenshot_button)

    def run(self):
        while self.running:
            self.handle_events()
            self.fractal.draw(self.screen)
            self.draw_ui()
            pygame.display.flip()
            self.clock.tick(60)
        pygame.quit()
        sys.exit()

if __name__ == "__main__":
    app = App()
    app.run()
```



Вывод: освоил возможности языка программирования Python в разработке оконных приложений.