МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**КАФЕДРА ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Лабораторная работа №7

По дисциплине “**Современные платформы программирования**”

Выполнил: студент группы ПО-11  
Сымоник И.А.  
Проверил:

Козик И. Д.

Брест 2025

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

Задание 1. Построение графических примитивов и надписей

6) Задать движение окружности по форме так, чтобы при касании границы

окружность отражалась от нее.

import pygame

import sys

import os

from pygame.locals import \*

class Circle:

    def \_\_init\_\_(self, x, y, radius, color, speed\_x, speed\_y):

        self.x = x

        self.y = y

        self.radius = radius

        self.color = color

        self.speed\_x = speed\_x

        self.speed\_y = speed\_y

        self.is\_paused = False

    def move(self, screen\_width, screen\_height):

        if not self.is\_paused:

            self.x += self.speed\_x

            self.y += self.speed\_y

            if self.x - self.radius <= 0 or self.x + self.radius >= screen\_width:

                self.speed\_x = -self.speed\_x

                self.x = max(self.radius, min(self.x, screen\_width - self.radius))

            if self.y - self.radius <= 0 or self.y + self.radius >= screen\_height:

                self.speed\_y = -self.speed\_y

                self.y = max(self.radius, min(self.y, screen\_height - self.radius))

    def draw(self, surface):

        pygame.draw.circle(surface, self.color, (int(self.x), int(self.y)), self.radius)

    def toggle\_pause(self):

        self.is\_paused = not self.is\_paused

    def set\_speed(self, speed\_x, speed\_y):

        if self.speed\_x != 0 and speed\_x != 0:

            self.speed\_x = abs(speed\_x) if self.speed\_x > 0 else -abs(speed\_x)

        else:

            self.speed\_x = speed\_x

        if self.speed\_y != 0 and speed\_y != 0:

            self.speed\_y = abs(speed\_y) if self.speed\_y > 0 else -abs(speed\_y)

        else:

            self.speed\_y = speed\_y

class App:

    def \_\_init\_\_(self):

        pygame.init()

        self.width, self.height = 800, 600

        self.screen = pygame.display.set\_mode((self.width, self.height))

        pygame.display.set\_caption("Task1")

        self.circle = Circle(

            x=self.width//2,

            y=self.height//2,

            radius=30,

            color=(255, 0, 0),

            speed\_x=5,

            speed\_y=5

        )

        self.font = pygame.font.SysFont('Arial', 18)

        self.ui\_elements = []

        self.create\_ui()

        self.speed\_input\_active = False

        self.speed\_input\_text = ""

        self.speed\_input\_rect = pygame.Rect(650, 50, 100, 30)

        self.current\_speed\_display = "5,5"

        self.screenshot\_counter = 1

    def create\_ui(self):

        pause\_button = {

            'rect': pygame.Rect(650, 10, 120, 30),

            'color': (100, 100, 100),

            'text': 'Пауза (P)',

            'action': self.circle.toggle\_pause

        }

        self.ui\_elements.append(pause\_button)

        screenshot\_button = {

            'rect': pygame.Rect(650, 90, 120, 30),

            'color': (100, 100, 100),

            'text': 'Скриншот (S)',

            'action': self.take\_screenshot

        }

        self.ui\_elements.append(screenshot\_button)

    def take\_screenshot(self):

        filename = f"screenshot\_{self.screenshot\_counter}.png"

        pygame.image.save(self.screen, filename)

        print(f"Скриншот сохранен как {filename}")

        self.screenshot\_counter += 1

    def format\_speed\_display(self, speed\_x, speed\_y):

        def format\_speed(s):

            return str(int(s)) if s.is\_integer() else str(s)

        return f"{format\_speed(abs(speed\_x))},{format\_speed(abs(speed\_y))}"

    def handle\_events(self):

        for event in pygame.event.get():

            if event.type == QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == KEYDOWN:

                if event.key == K\_p:

                    self.circle.toggle\_pause()

                elif event.key == K\_s:

                    self.take\_screenshot()

                elif event.key == K\_ESCAPE:

                    pygame.quit()

                    sys.exit()

                if self.speed\_input\_active:

                    if event.key == K\_RETURN:

                        try:

                            speeds = [float(s.strip()) for s in self.speed\_input\_text.split(',')]

                            if len(speeds) == 2:

                                self.circle.set\_speed(speeds[0], speeds[1])

                                self.current\_speed\_display = self.format\_speed\_display(speeds[0], speeds[1])

                        except ValueError:

                            print("Некорректный ввод скорости")

                        self.speed\_input\_active = False

                        self.speed\_input\_text = ""

                    elif event.key == K\_BACKSPACE:

                        self.speed\_input\_text = self.speed\_input\_text[:-1]

                    elif event.unicode.isdigit() or event.unicode in ',.-':

                        self.speed\_input\_text += event.unicode

            elif event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    for element in self.ui\_elements:

                        if element['rect'].collidepoint(event.pos):

                            element['action']()

                    if self.speed\_input\_rect.collidepoint(event.pos):

                        self.speed\_input\_active = True

                        self.speed\_input\_text = ""

                    else:

                        self.speed\_input\_active = False

    def draw\_ui(self):

        for element in self.ui\_elements:

            pygame.draw.rect(self.screen, element['color'], element['rect'])

            text\_surface = self.font.render(element['text'], True, (255, 255, 255))

            text\_rect = text\_surface.get\_rect(center=element['rect'].center)

            self.screen.blit(text\_surface, text\_rect)

        color = (255, 255, 255) if self.speed\_input\_active else (200, 200, 200)

        pygame.draw.rect(self.screen, color, self.speed\_input\_rect, 2)

        speed\_text = self.font.render("Скорость X,Y:", True, (255, 255, 255))

        self.screen.blit(speed\_text, (self.speed\_input\_rect.x - 120, self.speed\_input\_rect.y + 5))

        display\_text = self.speed\_input\_text if self.speed\_input\_active else self.current\_speed\_display

        text\_surface = self.font.render(display\_text, True, (255, 255, 255))

        self.screen.blit(text\_surface, (self.speed\_input\_rect.x + 5, self.speed\_input\_rect.y + 5))

        instructions = [

            "Управление:",

            "P - Пауза",

            "S - Скриншот",

            "ESC - Выход",

        ]

        for i, line in enumerate(instructions):

            text = self.font.render(line, True, (255, 255, 255))

            self.screen.blit(text, (10, 10 + i \* 25))

    def run(self):

        clock = pygame.time.Clock()

        while True:

            self.handle\_events()

            self.screen.fill((0, 0, 0))

            self.circle.move(self.width, self.height)

            self.circle.draw(self.screen)

            self.draw\_ui()

            pygame.display.flip()

            clock.tick(60)

if \_\_name\_\_ == "\_\_main\_\_":

    app = App()

    app.run()

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Задание 2. Реализовать построение заданного типа фрактала по варианту

6) Склоненное дерево Пифагора (обдуваемое ветром)

import pygame

import sys

import math

import os

from pygame.locals import \*

class Camera:

    def \_\_init\_\_(self, width, height):

        self.x = width // 2

        self.y = height // 2

        self.zoom = 1.0

        self.drag\_start = None

    def world\_to\_screen(self, point):

        return (

            (point[0] - self.x) \* self.zoom + self.x,

            (point[1] - self.y) \* self.zoom + self.y

        )

    def screen\_to\_world(self, point):

        return (

            (point[0] - self.x) / self.zoom + self.x,

            (point[1] - self.y) / self.zoom + self.y

        )

    def begin\_drag(self, pos):

        self.drag\_start = pos

    def update\_drag(self, pos):

        if self.drag\_start:

            dx = pos[0] - self.drag\_start[0]

            dy = pos[1] - self.drag\_start[1]

            self.x -= dx / self.zoom

            self.y -= dy / self.zoom

            self.drag\_start = pos

    def end\_drag(self):

        self.drag\_start = None

    def zoom\_in(self, factor=1.1):

        self.zoom \*= factor

    def zoom\_out(self, factor=1.1):

        self.zoom /= factor

    def reset(self, width, height):

        self.x = width // 2

        self.y = height // 2

        self.zoom = 1.0

class PythagorasTree:

    def \_\_init\_\_(self):

        pygame.init()

        self.width, self.height = 1200, 800

        self.screen = pygame.display.set\_mode((self.width, self.height), pygame.RESIZABLE)

        pygame.display.set\_caption("Дерево Пифагора с камерой")

        self.camera = Camera(self.width, self.height)

        self.params = {

            'angle\_left': 45,

            'angle\_right': 45,

            'length': 150,

            'depth': 10,

            'color\_mode': 0,

            'line\_width': 1,

            'size\_factor': 0.7,

            'wind\_effect': 0

        }

        self.base\_pos = [self.width // 2, self.height - 50]

        self.font = pygame.font.SysFont('Arial', 16)

        self.ui\_elements = []

        self.create\_ui()

        self.active\_input = None

    def create\_ui(self):

        self.ui\_elements = []

        groups = [

            {

                'title': "Форма дерева",

                'params': [

                    ('angle\_left', "Угол левых ветвей", 0, 90),

                    ('angle\_right', "Угол правых ветвей", 0, 90),

                    ('length', "Длина ствола", 50, 300),

                    ('depth', "Глубина ветвления", 1, 15),

                    ('size\_factor', "Уменьшение ветвей", 0.5, 0.9),

                    ('wind\_effect', "Наклон дерева", -30, 30)

                ],

                'pos': (20, 20)

            },

            {

                'title': "Внешний вид",

                'params': [

                    ('line\_width', "Толщина линий", 1, 5),

                    ('color\_mode', "Цветовой режим", 0, 1)

                ],

                'pos': (20, 300)

            }

        ]

        for group in groups:

            title = self.font.render(group['title'], True, (255, 255, 255))

            self.ui\_elements.append({

                'type': 'group\_title',

                'surface': title,

                'rect': pygame.Rect(\*group['pos'], 200, 20)

            })

            y\_offset = group['pos'][1] + 30

            for param in group['params']:

                name, label, min\_val, max\_val = param

                current\_val = self.params[name]

                param\_text = self.font.render(label, True, (220, 220, 220))

                self.ui\_elements.append({

                    'type': 'param\_label',

                    'surface': param\_text,

                    'rect': pygame.Rect(group['pos'][0], y\_offset, 150, 25)

                })

                value\_text = self.font.render(f"{current\_val:.1f}", True, (255, 255, 255))

                self.ui\_elements.append({

                    'type': 'param\_value',

                    'param': name,

                    'surface': value\_text,

                    'rect': pygame.Rect(group['pos'][0] + 160, y\_offset, 50, 25)

                })

                slider\_rect = pygame.Rect(group['pos'][0] + 220, y\_offset, 150, 20)

                self.ui\_elements.append({

                    'type': 'slider',

                    'param': name,

                    'rect': slider\_rect,

                    'min': min\_val,

                    'max': max\_val,

                    'value': current\_val

                })

                y\_offset += 35

        buttons = [

            ('reset', "Сброс", (20, 500), (100, 30)),

            ('screenshot', "Скриншот", (130, 500), (100, 30)),

            ('center', "Центрировать", (240, 500), (120, 30))

        ]

        for btn in buttons:

            name, text, pos, size = btn

            rect = pygame.Rect(\*pos, \*size)

            self.ui\_elements.append({

                'type': 'button',

                'name': name,

                'text': text,

                'rect': rect,

                'color': (70, 70, 70),

                'hover\_color': (100, 100, 100)

            })

    def draw\_tree(self, surface, start\_pos, angle, length, depth):

        if depth <= 0 or length < 1:

            return

        screen\_start = self.camera.world\_to\_screen(start\_pos)

        angle\_rad = math.radians(angle) + math.radians(self.params['wind\_effect']) \* (depth/self.params['depth'])

        world\_end = (

            start\_pos[0] + length \* math.cos(angle\_rad),

            start\_pos[1] - length \* math.sin(angle\_rad)

        )

        screen\_end = self.camera.world\_to\_screen(world\_end)

        if self.params['color\_mode'] == 0:

            greenness = min(1, (self.params['depth'] - depth) / 3)

            color = (

                50 + int(150 \* (1 - greenness)),

                50 + int(200 \* greenness),

                30 + int(20 \* greenness)

            )

        else:

            hue = (depth \* 30) % 360

            color = self.hsv\_to\_rgb(hue, 0.8, 0.7)

        thickness = max(1, int(self.params['line\_width'] \* (0.5 + depth/self.params['depth']) \* self.camera.zoom))

        pygame.draw.line(surface, color, screen\_start, screen\_end, thickness)

        if depth > 1:

            new\_length = length \* self.params['size\_factor']

            self.draw\_tree(

                surface, world\_end,

                angle + self.params['angle\_left'],

                new\_length, depth - 1)

            self.draw\_tree(

                surface, world\_end,

                angle - self.params['angle\_right'],

                new\_length, depth - 1)

    def hsv\_to\_rgb(self, h, s, v):

        h = h % 360

        c = v \* s

        x = c \* (1 - abs((h / 60) % 2 - 1))

        m = v - c

        if 0 <= h < 60:

            r, g, b = c, x, 0

        elif 60 <= h < 120:

            r, g, b = x, c, 0

        elif 120 <= h < 180:

            r, g, b = 0, c, x

        elif 180 <= h < 240:

            r, g, b = 0, x, c

        elif 240 <= h < 300:

            r, g, b = x, 0, c

        else:

            r, g, b = c, 0, x

        return (

            int((r + m) \* 255),

            int((g + m) \* 255),

            int((b + m) \* 255)

        )

    def update\_ui\_values(self):

        for element in self.ui\_elements:

            if element['type'] == 'param\_value':

                value = self.params[element['param']]

                text = f"{value:.1f}" if isinstance(value, float) else str(value)

                element['surface'] = self.font.render(text, True, (255, 255, 255))

    def handle\_events(self):

        mouse\_pos = pygame.mouse.get\_pos()

        mouse\_pressed = pygame.mouse.get\_pressed()

        for event in pygame.event.get():

            if event.type == QUIT:

                pygame.quit()

                sys.exit()

            elif event.type == KEYDOWN:

                if event.key == K\_s:

                    self.take\_screenshot()

                elif event.key == K\_r:

                    self.reset\_params()

                elif event.key == K\_c:

                    self.center\_camera()

                elif event.key == K\_ESCAPE:

                    pygame.quit()

                    sys.exit()

                elif event.key == K\_PLUS or event.key == K\_EQUALS:

                    self.camera.zoom\_in()

                elif event.key == K\_MINUS:

                    self.camera.zoom\_out()

            elif event.type == MOUSEBUTTONDOWN:

                if event.button == 1:

                    ui\_clicked = False

                    for element in self.ui\_elements:

                        if element['rect'].collidepoint(event.pos):

                            ui\_clicked = True

                            if element['type'] == 'button':

                                if element['name'] == 'screenshot':

                                    self.take\_screenshot()

                                elif element['name'] == 'reset':

                                    self.reset\_params()

                                elif element['name'] == 'center':

                                    self.center\_camera()

                            break

                    if not ui\_clicked:

                        self.camera.begin\_drag(event.pos)

                elif event.button == 4:

                    self.camera.zoom\_in()

                elif event.button == 5:

                    self.camera.zoom\_out()

            elif event.type == MOUSEBUTTONUP:

                if event.button == 1:

                    self.camera.end\_drag()

            elif event.type == MOUSEMOTION:

                if mouse\_pressed[0] and self.camera.drag\_start:

                    self.camera.update\_drag(event.pos)

            elif event.type == VIDEORESIZE:

                self.width, self.height = event.size

                self.screen = pygame.display.set\_mode((self.width, self.height), pygame.RESIZABLE)

                self.base\_pos = [self.width // 2, self.height - 50]

                self.camera.reset(self.width, self.height)

                self.create\_ui()

        if mouse\_pressed[0] and not self.camera.drag\_start:

            for element in self.ui\_elements:

                if element['type'] == 'slider' and element['rect'].collidepoint(mouse\_pos):

                    rel\_x = mouse\_pos[0] - element['rect'].x

                    percent = min(1, max(0, rel\_x / element['rect'].width))

                    new\_value = element['min'] + percent \* (element['max'] - element['min'])

                    if element['param'] in ['depth', 'color\_mode']:

                        new\_value = int(round(new\_value))

                    self.params[element['param']] = new\_value

                    self.update\_ui\_values()

    def take\_screenshot(self):

        tree\_surface = pygame.Surface((self.width, self.height), pygame.SRCALPHA)

        tree\_surface.fill((0, 0, 0, 0))

        self.draw\_tree(

            tree\_surface, self.base\_pos,

            90, self.params['length'],

            self.params['depth'])

        counter = 1

        while True:

            filename = f"pythagoras\_tree\_{counter}.png"

            if not os.path.exists(filename):

                break

            counter += 1

        pygame.image.save(tree\_surface, filename)

        print(f"Скриншот сохранен как {filename}")

    def reset\_params(self):

        self.params.update({

            'angle\_left': 45,

            'angle\_right': 45,

            'length': 150,

            'depth': 10,

            'color\_mode': 0,

            'line\_width': 1,

            'size\_factor': 0.7,

            'wind\_effect': 0

        })

        self.update\_ui\_values()

    def center\_camera(self):

        self.camera.reset(self.width, self.height)

    def draw\_ui(self):

        pygame.draw.rect(self.screen, (30, 30, 40, 220), (0, 0, 400, self.height))

        for element in self.ui\_elements:

            if element['type'] in ['group\_title', 'param\_label', 'param\_value']:

                self.screen.blit(element['surface'], element['rect'])

            elif element['type'] == 'slider':

                pygame.draw.rect(self.screen, (60, 60, 70), element['rect'])

                percent = (self.params[element['param']] - element['min']) / (element['max'] - element['min'])

                slider\_pos = element['rect'].x + int(percent \* element['rect'].width)

                pygame.draw.rect(

                    self.screen, (100, 150, 200),

                    (slider\_pos - 3, element['rect'].y, 6, element['rect'].height))

            elif element['type'] == 'button':

                is\_hovered = element['rect'].collidepoint(pygame.mouse.get\_pos())

                color = element['hover\_color'] if is\_hovered else element['color']

                pygame.draw.rect(self.screen, color, element['rect'])

                pygame.draw.rect(self.screen, (150, 150, 150), element['rect'], 1)

                text = self.font.render(element['text'], True, (255, 255, 255))

                text\_rect = text.get\_rect(center=element['rect'].center)

                self.screen.blit(text, text\_rect)

        help\_text = [

            "Управление камерой:",

            "ЛКМ + перемещение - двигать камеру",

            "Колесо мыши - масштабирование",

            "+/- - масштабирование",

            "C - центрировать камеру"

        ]

        for i, line in enumerate(help\_text):

            text = self.font.render(line, True, (200, 200, 200))

            self.screen.blit(text, (self.width - 250, 20 + i \* 25))

    def run(self):

        clock = pygame.time.Clock()

        while True:

            self.handle\_events()

            self.screen.fill((15, 20, 25))

            self.draw\_tree(

                self.screen, self.base\_pos,

                90, self.params['length'],

                self.params['depth'])

            self.draw\_ui()

            pygame.display.flip()

            clock.tick(60)

if \_\_name\_\_ == "\_\_main\_\_":

    app = PythagorasTree()

    app.run()

Изображение выглядит как снимок экрана, текст, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

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