ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Кафедра ВМиК

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

по предмету **«Объектно-ориентированное программирование»**

Выполнил: студент группы МО-203б

Ярошко Е. В.

Проверил:  
доцент каф. ВМиК

Котельников В. А.

**Уфа 2025 г.**

**Цель лабораторной работы**

Создать простейший визуальный редактор векторных объектов.

**Задание**

На основе Л.Р.3 реализовать простейший визуальный редактор векторных объектов (круг, квадрат, эллипс, прямоугольник, треугольник, отрезок, и т.д.) со следующей функциональностью:

* Создание графических объектов на экране: меню, панель инструментов с доступными классами объектов; добавление нового объекта в рабочую область; изменение пользователем размера формы
* Манипуляции объектами в рабочей области: выбор объекта для манипулирования (выделяется цветом или рамкой); несколько одновременно выбранных объектов; изменение цвета, размера, положения; удаление объекта из рабочей области; контроль выхода за рабочую область (при передвижении объект не должен выходить за границы ни одной своей частью)

Программные требования:

* манипуляции объектами выполняются с помощью клавиатуры, при этом выбор клавиш управления должен быть как можно ближе к используемым обычно для таких задач в среде Windows;
* «хорошая» иерархия классов;
* использование стандартных диалогов для выбора цвета;
* использование своего контейнера объектов из Л.Р.3;
* отделение взаимодействия с пользователем от логики работы классов.

**Ход выполнения лабораторной работы**

Выполнение данной лабораторной работы основано на уже существующей лабораторной работе №3.1, поэтому среда разработки: ядро и компилятор Python 3.13.2 + Customtkinter + Visual Studio Code.

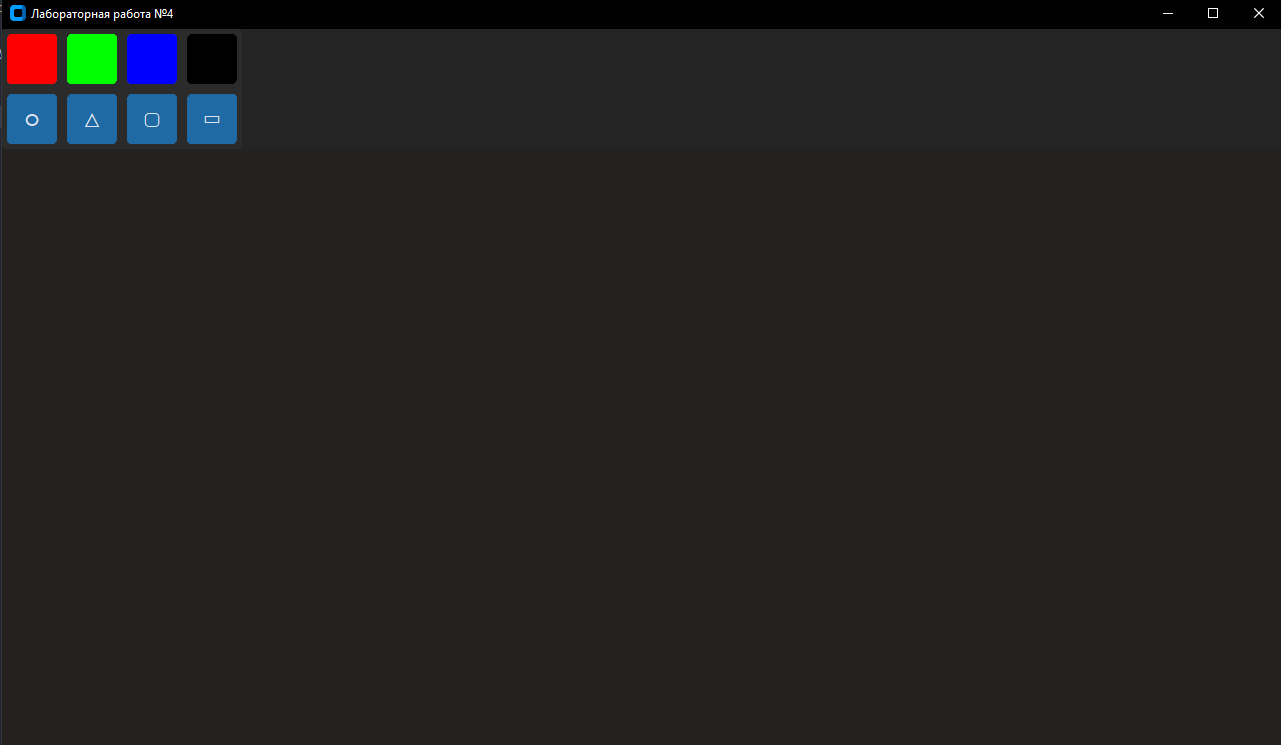
Чтобы подготовить код для расширения, был создан новый класс Figure и вынесен в отдельный файл BaseFigure. В него вынесен функционал уже существующего класса CCircle кроме отрисовки. Также в него добавлены новые методы определения своих границ (чтобы фигура не выходила за пределы экрана). От данного класса наследуются все классы фигур (кроме CSquare: он наследуется от CRectangle). Во всех классах переопределяется метод draw (т. к. каждая фигура отрисовывается по-разному) и метод mousecheck (т. к. у фигур разные границы отрисовки). Метод mousecheck необходимо переопределять для того, чтобы при клике мышкой корректно определять, попала ли мышка на фигуру или нет.

Создан новый класс кнопок BaseButton, от которого наследуются ещё два класса: ColorButton и FigureButton. Они отображаются в панели инструментов (класс EditorPanel в главном файле) и отрисовывают необходимые опознавательные знаки.

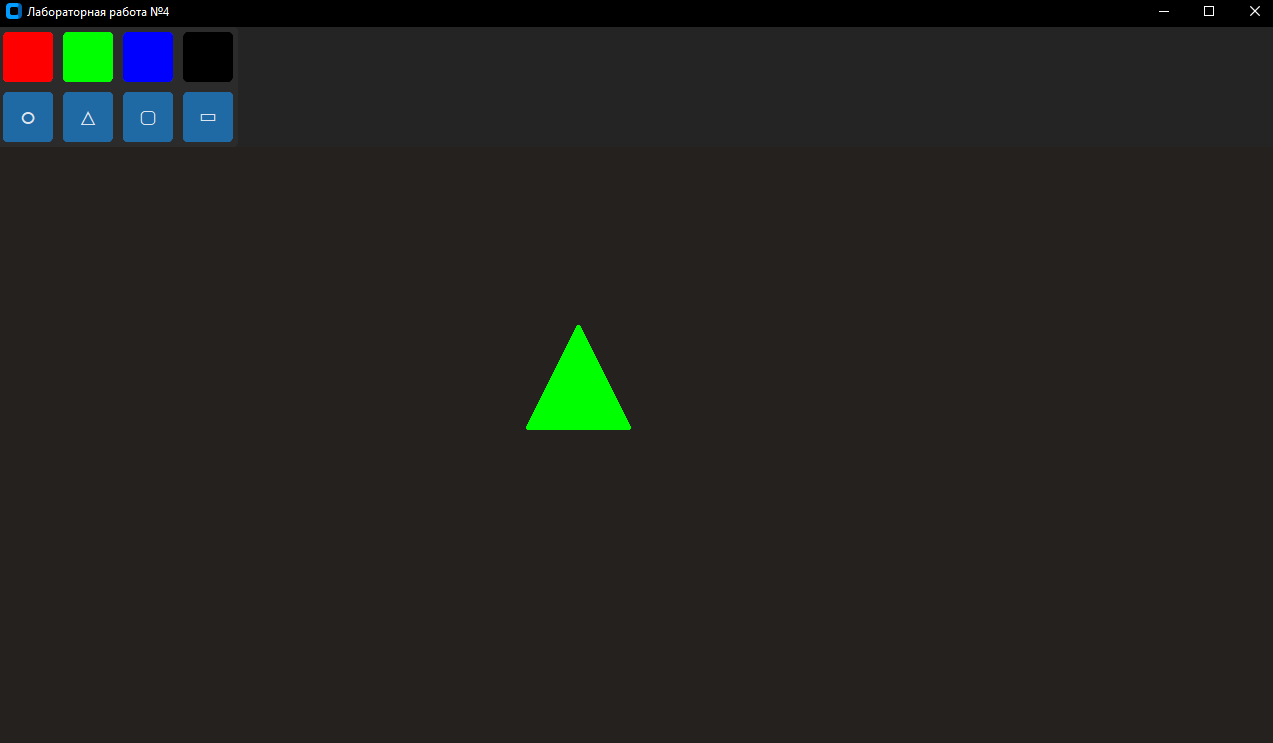
Т. к. по заданию лабораторной работы необходимо, чтобы объекты «не сбивались в кучу» при передвижении их к краю окна, функционал передвижения и изменения размера объектов реализован в классе Container.

В новом классе EditorPanel реализован функционал панели инструментов. С помощью неё становится возможным выбрать фигуру и её цвет (выбранные параметры передаются в Container, который создаёт объекты по этим параметрам). Также в нём хранятся массивы со значениями цветов и Unicode-символами фигур.

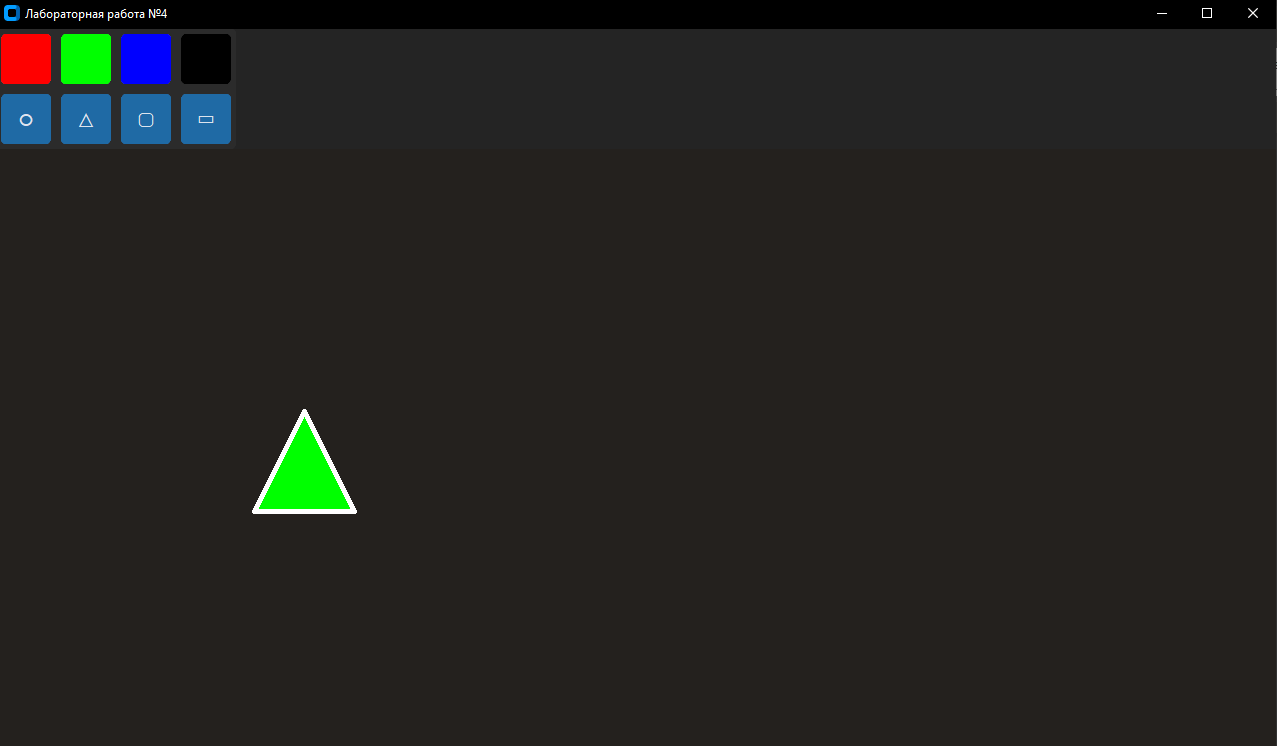
При компиляции приложения появляется главное окно (Рисунок 1):

Рисунок 1 — Главное окно приложения

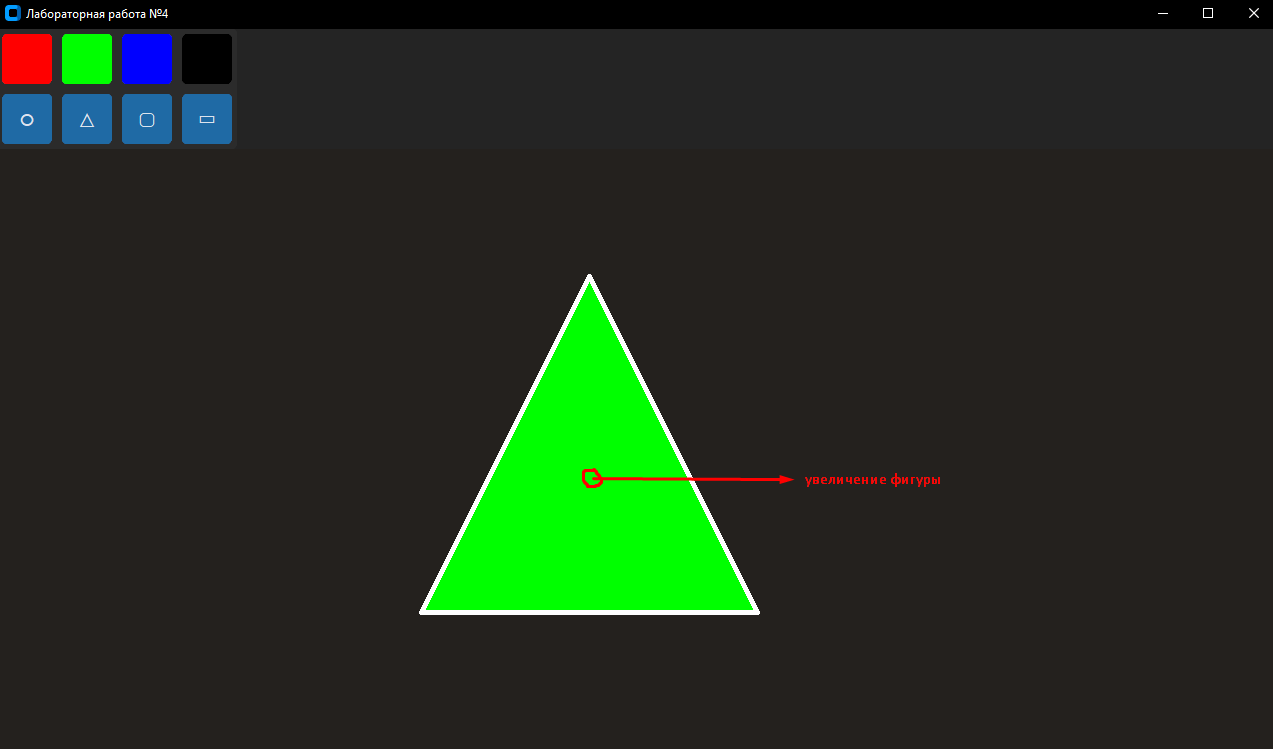
Слева вверху находится объект EditorPanel, который содержит в себе объекты классов ColorButton и FigureButton. При нажатии на разные кнопки (например, кнопку с изображением треугольника и зелёную кнопку) происходит изменение параметров создания фигуры, и при нажатии на рабочий холст (всё пространство между границами окна и границей EditorPanel) создаётся новый объект (Рисунок 2):

Рисунок 2 — Создание нового объекта

При нажатии ПКМ по данному объекту происходит его выделение (белая обводка), после чего, зажимая ЛКМ, его можно перемещать (Рисунок 3):

Рисунок 3 — Выделение и перемещение объекта

При зажатом Ctrl с помощью зажатой ЛКМ можно изменять размер объектов, перемещая мышку от центра фигуры (Рисунок 4):

Рисунок 4 — Увеличение фигуры

**Выводы по лабораторной работе**

Создали простейший визуальный редактор, позволяющий создавать, удалять и редактировать объекты в виде векторных фигур. Реализовали панель инструментов, создание объектов, их выделение, удаление, изменение размера, а также передвижение.

**Приложение 1**

Исходные коды программ по заданиям

**Задание 1**

Файл lab4.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/lab4.py>

Исходный код:

import customtkinter as ctk

from tkinter import Canvas

from math import sqrt

from figures.Circle import CCircle

from figures.Rectangle import CRectangle

from figures.Square import CSquare

from figures.Triangle import CTriangle

from buttons.ColorButton import CColorButton

from buttons.FigureButton import CFigureButton

class Container(): # Главный контейнер объектов

def \_\_init\_\_(self, master, canvas):

# Private

self.\_\_container = list()

self.\_\_selected\_container = list()

self.\_\_action = False

self.\_\_initial\_sizes = dict()

self.\_\_app = master

self.\_\_widget = None #виджет, на котором находится мышка

# Public

self.canvas = canvas

self.chosen\_color = "#000000"

self.chosen\_figure = "○"

def container\_append(self, object): # Добавляет в контейнер новый круг

print(f"Appending new object: {object}")

self.\_\_container.append(object)

self.\_\_initial\_sizes[object] = object.size

def safe\_move\_all(self, point): # Передвигает объекты и проверяет их границы, чтобы остановить их все

new\_positions = []

for obj in self.\_\_selected\_container:

new\_x = point.x + obj.\_offset\_x

new\_y = point.y + obj.\_offset\_y

if not obj.boundaries(new\_x, new\_y, obj.size[0], obj.size[1], self.canvas.winfo\_width(), self.canvas.winfo\_height()):

return # Один объект не может быть перемещён — отменяем перемещение всех

new\_positions.append((obj, new\_x, new\_y))

for obj, x, y in new\_positions:

obj.move(x, y)

def scale\_all(self, point): # Меняет размер объектов

new\_scales = []

weightpoint = [0, 0] # усред. точка между объектами (точка массы)

for obj in self.\_\_selected\_container: # считаем усреднённую точку

weightpoint[0] += obj.\_x

weightpoint[1] += obj.\_y

weightpoint = [x / len(self.\_\_selected\_container) for x in weightpoint] # среднее арифметическое

for obj in self.\_\_selected\_container:

new\_size\_coef = (self.measure\_prolongate(weightpoint[0], weightpoint[1], point.x, point.y) / 100.0) + 1.0

new\_size = [x \* new\_size\_coef for x in self.\_\_initial\_sizes[obj]]

if not obj.boundaries(obj.\_x, obj.\_y, new\_size[0], new\_size[1], self.canvas.winfo\_width(), self.canvas.winfo\_height()):

return

new\_scales.append((obj,new\_size))

for obj, new\_size in new\_scales:

obj.resize(new\_size)

def measure\_prolongate(self, start\_point\_x: int, start\_point\_y: int, end\_point\_x: int, end\_point\_y: int) -> float: # считает расстояние от точки до точки

return sqrt((end\_point\_x - start\_point\_x)\*\*2+(end\_point\_y - start\_point\_y)\*\*2)

def handle\_mouse\_click(self, point): # Создаёт новый круг и добавляет в список, если нет выделенных объектов

self.\_\_widget = self.\_\_app.winfo\_containing(point.x\_root, point.y\_root)

if self.\_\_widget == self.canvas:

if not self.\_\_selected\_container:

match self.chosen\_figure:

case "○":

self.container\_append(CCircle(master=self, x=point.x, y=point.y, canvas=self.canvas))

case "▢":

self.container\_append(CSquare(master=self, x=point.x, y=point.y, canvas=self.canvas))

case "▭":

self.container\_append(CRectangle(master=self, x=point.x, y=point.y, canvas=self.canvas))

case "△":

self.container\_append(CTriangle(master=self, x=point.x, y=point.y, canvas=self.canvas))

self.redraw()

else:

for object in self.\_\_selected\_container:

object.measure\_offsets(point.x, point.y)

def handle\_mouse\_down(self, point):

if self.\_\_selected\_container and self.\_\_widget == self.canvas:

self.safe\_move\_all(point) if not(self.\_\_action) else self.scale\_all(point)

self.redraw()

def select\_objects(self, point): # Выделяет круги (в зависимости от \_\_multiple\_selection меняется поведение)

if not(self.\_\_action):

for circle in self.\_\_selected\_container:

circle.deselect()

self.\_\_selected\_container.clear()

self.\_\_selected\_container.extend(list(filter(lambda x: x.mousecheck(point.x, point.y), self.\_\_container)))

for circle in self.\_\_selected\_container:

circle.select()

self.redraw()

def deselect\_objects(self, \*args): # Снимает выделение со всех кругов

for circle in self.\_\_selected\_container:

circle.deselect()

self.\_\_selected\_container.clear()

self.redraw()

def delete\_objects(self, \*args): # Удаляет выделенные объекты

for obj in self.\_\_selected\_container:

self.\_\_container.remove(obj)

del obj

self.\_\_selected\_container.clear()

self.redraw()

def initiate\_additional\_action(self, \*args): # Включает дополнительное действие

self.\_\_action = True

def stop\_additional\_action(self, \*args): # Выключает дополнительное действие

self.\_\_action = False

def redraw(self):

self.canvas.delete("all")

for circle in self.\_\_container:

circle.draw()

class EditorPanel(ctk.CTkFrame): # Главная панель с кнопками

def \_\_init\_\_(self, master, container):

super().\_\_init\_\_(master)

self.\_container = container

colors = ["#FF0000", "#00FF00", "#0000FF", "#000000"]

hover\_colors = ["#AA0000", "#00AA00", "#0000AA", "#444444"]

figures = ["○", "△", "▢", "▭"]

self.configure(height=200)

self.rowconfigure((0,1), weight=1)

self.columnconfigure((0,1,2,3), weight=1)

for b in range(4):

colorbutton = CColorButton(master=self)

colorbutton.configure(command=lambda col=colors[b]: self.change\_color(col))

colorbutton.set\_color(colors[b], hover\_colors[b])

colorbutton.grid(row=0, column=b, pady=5, padx=5)

figurebutton = CFigureButton(master=self)

figurebutton.configure(command=lambda figure=figures[b]: self.change\_figure(figure))

figurebutton.set\_figure(figures[b])

figurebutton.grid(row=1, column=b, pady=5, padx=5)

def change\_color(self, color: str):

self.\_container.chosen\_color = color

def change\_figure(self, figure: str):

self.\_container.chosen\_figure = figure

class App(ctk.CTk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.title("Лабораторная работа №4")

self.geometry("1280x720")

self.rowconfigure(0, weight=0)

self.rowconfigure(1, weight=1)

self.columnconfigure(0, weight=1)

self.canvas = Canvas(master=self, bg="#24211e", highlightbackground="#24211e")

self.canvas.grid(row=1, column=0, sticky="nsew", padx=0, pady=0)

self.container = Container(canvas=self.canvas, master=self)

self.editorpanel = EditorPanel(master=self, container=self.container)

self.editorpanel.grid(row=0, column=0, sticky="NW")

self.bind("<Button-1>", self.container.handle\_mouse\_click)

self.bind("<B1-Motion>", self.container.handle\_mouse\_down)

self.bind("<Button-3>", self.container.select\_objects)

self.bind\_all("<Escape>", self.container.deselect\_objects)

self.bind\_all("<Delete>", self.container.delete\_objects)

self.bind("<KeyPress-Control\_L>", self.container.initiate\_additional\_action)

self.bind("<KeyRelease-Control\_L>", self.container.stop\_additional\_action)

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

app = App()

app.mainloop()

Файл BaseFigure.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/figures/BaseFigure.py>

Исходный код:

class Figure(): # Общий класс фигуры

def \_\_init\_\_(self, master, canvas, x, y):

# Protected

self.\_container = master

self.\_size\_coef = 1.0

self.\_size\_x = 100.0

self.\_size\_y = 100.0

self.\_color =self.\_container.chosen\_color

self.\_border\_color = "#ffffff"

self.\_selected = False

self.\_x, self.\_y = x, y

# vvv Нужны для передвижения фигур мышкой

self.\_offset\_x, self.\_offset\_y = 0, 0

# vvv Нужны для скейлинга

self.\_startmouse\_x, self.\_startmouse\_y = x, y

# Public

self.canvas = canvas

self.size = [0, 0]

self.update\_size()

self.\_chosen\_border\_color = ""

print(f"New point: {x}, {y}")

def draw(self): # Переопред. в классе

self.\_chosen\_border\_color = self.\_border\_color if self.\_selected else self.\_color

def move(self, x: int, y: int):

self.\_x = x

self.\_y = y

def resize(self, new\_size):

self.size = new\_size

def select(self):

self.\_selected = True

def deselect(self):

self.\_selected = False

def mousecheck(self, x: int, y: int): # Переопред. в классе

pass

def update\_size(self):

self.size = [self.\_size\_x \* self.\_size\_coef, self.\_size\_y \* self.\_size\_coef]

def measure\_offsets(self, x: int, y: int): # Вычисляет относ. положение фигуры от мыши

self.\_offset\_x = self.\_x - x

self.\_offset\_y = self.\_y - y

def boundaries(self, x: int, y: int, size\_x: float, size\_y: float, canvas\_width: int, canvas\_height: int) -> bool: # опред. границ

return (size\_x // 2) <= x <= canvas\_width - (size\_x // 2) and \

(size\_y // 2) <= y <= canvas\_height - (size\_y // 2)

Файл Circle.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/figures/Circle.py>

Исходный код:

from figures.BaseFigure import Figure

class CCircle(Figure): # Базовый класс круга

def \_\_init\_\_(self, master, canvas, x, y):

super().\_\_init\_\_(master, canvas, x, y)

# Private

self.\_radius = self.size[0] // 2

def draw(self): # Отрисовывает круг на Canvas

super().draw()

self.\_radius = self.size[0] // 2

self.canvas.create\_oval(self.\_x - self.\_radius, self.\_y - self.\_radius,\

self.\_x + self.\_radius, self.\_y + self.\_radius, fill = self.\_color, width = 5, outline = self.\_chosen\_border\_color)

def mousecheck(self, x: int, y: int) -> bool: # Проверяет, наход. ли точка внутри круга

return ((x - self.\_x)\*\*2 + (y - self.\_y)\*\*2) <= self.\_radius\*\*2

Файл Rectangle.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/figures/Rectangle.py>

Исходный код:

from figures.BaseFigure import Figure

class CRectangle(Figure): # Базовый класс прямоугольника

def \_\_init\_\_(self, master, canvas, x, y):

super().\_\_init\_\_(master, canvas, x, y)

# Private

self.\_\_half\_size\_x, self.\_\_half\_size\_y = 0, 0

self.\_size\_x = 100.0

self.\_size\_y = 50.0

self.update\_size()

def draw(self):

super().draw()

self.\_\_half\_size\_x = self.size[0] // 2

self.\_\_half\_size\_y = self.size[1] // 2

self.canvas.create\_rectangle(self.\_x - self.\_\_half\_size\_x,

self.\_y - self.\_\_half\_size\_y,

self.\_x + self.\_\_half\_size\_x,

self.\_y + self.\_\_half\_size\_y,

fill = self.\_color, width = 5, outline = self.\_chosen\_border\_color)

def mousecheck(self, x: int, y: int) -> bool: # Проверяет, наход. ли точка внутри прямоугольника

return (self.\_x - self.\_\_half\_size\_x <= x <= self.\_x + self.\_\_half\_size\_x) and \

(self.\_y - self.\_\_half\_size\_y <= y <= self.\_y + self.\_\_half\_size\_y)

Файл Square.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/figures/Square.py>

Исходный код:

from figures.Rectangle import CRectangle

class CSquare(CRectangle): # Базовый класс квадрата

def \_\_init\_\_(self, master, canvas, x, y):

super().\_\_init\_\_(master, canvas, x, y)

self.\_size\_x = 100.0

self.\_size\_y = 100.0

self.update\_size()

Файл Triangle.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/figures/Triangle.py>

Исходный код:

from figures.BaseFigure import Figure

class CTriangle(Figure):

def \_\_init\_\_(self, master, canvas, x, y):

super().\_\_init\_\_(master, canvas, x, y)

# Private

self.\_\_triangle\_points = [[],[],[]]

self.update\_points()

def draw(self):

super().draw()

self.update\_points()

points = [coordinate for point in self.\_\_triangle\_points for coordinate in point] # раскрытие вложенности

self.canvas.create\_polygon(points, fill = self.\_color, width = 5, outline = self.\_chosen\_border\_color)

def update\_points(self):

self.\_\_triangle\_points = [[self.\_x - (self.size[0] // 2), self.\_y + (self.size[1] // 2)],\

[self.\_x, self.\_y - (self.size[1] // 2)],\

[self.\_x + (self.size[0] // 2), self.\_y + (self.size[1] // 2)]]

def sign(self, number: float) -> int: # проверяет знак

if number < 0:

return -1

elif number > 0:

return 1

return 0

def mousecheck(self, x: int, y: int) -> bool: # Проверяет, наход. ли точка внутри треугольника

# Укорочение переменных для удобства

x1 = self.\_\_triangle\_points[0][0]

y1 = self.\_\_triangle\_points[0][1]

x2 = self.\_\_triangle\_points[1][0]

y2 = self.\_\_triangle\_points[1][1]

x3 = self.\_\_triangle\_points[2][0]

y3 = self.\_\_triangle\_points[2][1]

# Считаются псевдоскалярные произведения

check\_first\_side = (x1 - x)\*(y2 - y1) - (x2 - x1)\*(y1 - y)

check\_second\_side = (x2 - x)\*(y3 - y2) - (x3 - x2)\*(y2 - y)

check\_third\_side = (x3 - x)\*(y1 - y3) - (x1 - x3)\*(y3 - y)

# True, если все знаки равны или хотя бы одно значение равно нулю

return ((self.sign(check\_first\_side) == self.sign(check\_second\_side) == self.sign(check\_third\_side))\

or (check\_first\_side\*check\_second\_side\*check\_third\_side == 0))

Файл BaseButton.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/buttons/BaseButton.py>

Исходный код:

import customtkinter as ctk

class CButton(ctk.CTkButton): # кнопка в панели

def \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.configure(width=50, height=50, text="")

Файл ColorButton.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/buttons/ColorButton.py>

Исходный код:

import customtkinter as ctk

from buttons.BaseButton import CButton

class CColorButton(CButton):

def \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.\_button\_color = "#000000"

self.\_hover\_color = "#111111"

def set\_color(self, color: str, hover\_color = str):

self.\_button\_color = color

self.\_hover\_color = hover\_color

self.configure(fg\_color = self.\_button\_color, hover\_color = self.\_hover\_color)

def get\_color(self) -> str:

return self.\_button\_color

def get\_hover\_color(self) -> str:

return self.\_hover\_color

Файл FigureButton.py: <https://github.com/qvap/OOP--Lab-4/blob/main/lab4/buttons/FigureButton.py>

Исходный код:

import customtkinter as ctk

from buttons.BaseButton import CButton

class CFigureButton(CButton):

def \_\_init\_\_(self, master):

super().\_\_init\_\_(master)

self.\_figure = ""

def set\_figure(self, figure: str):

self.\_figure = figure

self.configure(text=self.\_figure, font = ("Consolas", 25))