Лабораторная работа №3

Сложная анимация

Вариант 9 – Бойды (165 шт.)

ФИО: Ченцов Егор Андреевич

Номер студенческого: 245136

Группа: ИД24-3

Название предмета: Практикум по программированию

Дата сдачи: 25.11.2025

B. Описание задания

Реализуйте известный алгоритм "бойды", симулирующий стайное поведение птиц и других животных. Используйте набор бойдов со случайными положением и случайной, но ограниченной начальной скоростью (165 штук).

Добавьте возможность включить или отключать каждую из трёх входящих в алгоритм сил, например, по нажатию клавиш клавиатуры.

Позвольте пользователю регулировать веса этих трёх сил, например, при помощи слайдеров.

C. Достигнутая сложность и реализация

Выполнены все основные требования:

• 165 бойдов с реалистичным стайным поведением

• Три правила Рейнольдса: Разделение, Выравнивание, Сплоченность

• Включение/выключение правил клавишами 1, 2, 3

• Регулировка весов в реальном времени через интерактивные слайдеры

Дополнительно реализовано:

• Все параметры в отдельном файле изменяются config.json (цвета, размеры, скорости, радиус восприятия и др.)

• Добавил подписи слайдеров и подсказки

• Использовал цвета, которые были в задании

• Плавная анимация 60 FPS

• Мягкое отражение от границ экрана

D. Объяснение проектирования программы

Структура ООП

1. **Класс Boid.py** — представляет одного бойда

   - Хранит позицию, скорость и ускорение

   - update(boids) — реализует три правила + ограничение скорости + отражение от краёв

   - draw(screen) — рисует треугольник-стрелку

2. **Класс Slider.py** — интерактивный слайдер

   - Обработка мыши, отрисовка, возврат значения

3. **main.py** —загрузка конфига, основной цикл

Организация кода

- Каждый класс в отдельном файле

- Полностью прокомментированный код

- Все константы и параметры вынесены в конфигурационный файл config.json

Математическое исследование

Алгоритм Boids (Craig Reynolds, 1986) основан на трёх векторных правилах:

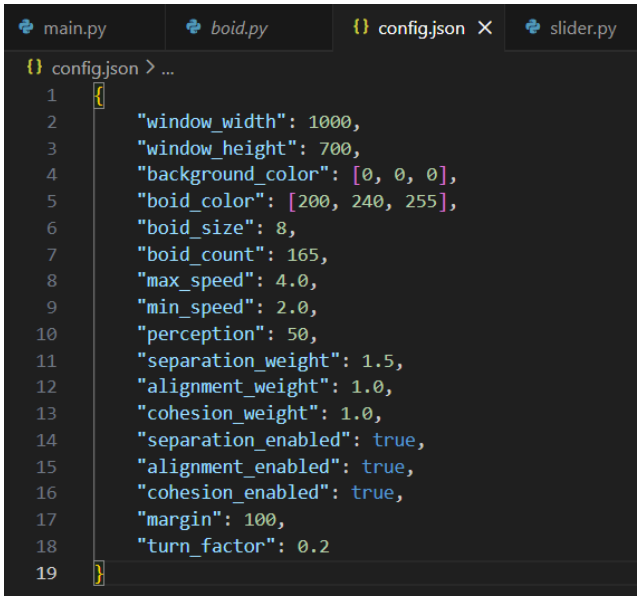
1. Separation (Разделение) — избегание столкновений

2. Alignment (Выравнивание) — выравнивание по соседям

3. Cohesion (Сплочённость) — движение к центру массы соседей

Реализована классическая версия с радиусом восприятия 50 px.

Управление настройками (файл config.json)



- window_width — Ширина окна программы в пикселях

- window_height — Высота окна в пикселях

- background_color — Цвет фона в формате RGB

- boid_color — Цвет самих бойдов

- boid_size — Размер треугольника-бойда (в пикселях)

- boid_count — Количество бойдов (обязательное требование задания, 165)

- max_speed — Максимальная скорость, которую может развить бойд

- min_speed — Минимальная скорость (бойды никогда не останавливаются)

- perception — Радиус восприятия (в пикселях). Бойд «видит» только соседей ближе 50 px

- separation_weight — Вес силы Разделение (отталкивание от слишком близких соседей)

- alignment_weight — Вес силы Выравнивание (лететь в среднем направлении соседей)

- cohesion_weight — Вес силы Сплочённость (лететь к центру массы соседей)

- separation_enabled — Включена ли сила Разделение

- alignment_enabled — Включена ли сила Выравнивание

- cohesion_enabled — Включена ли сила Сплочённость

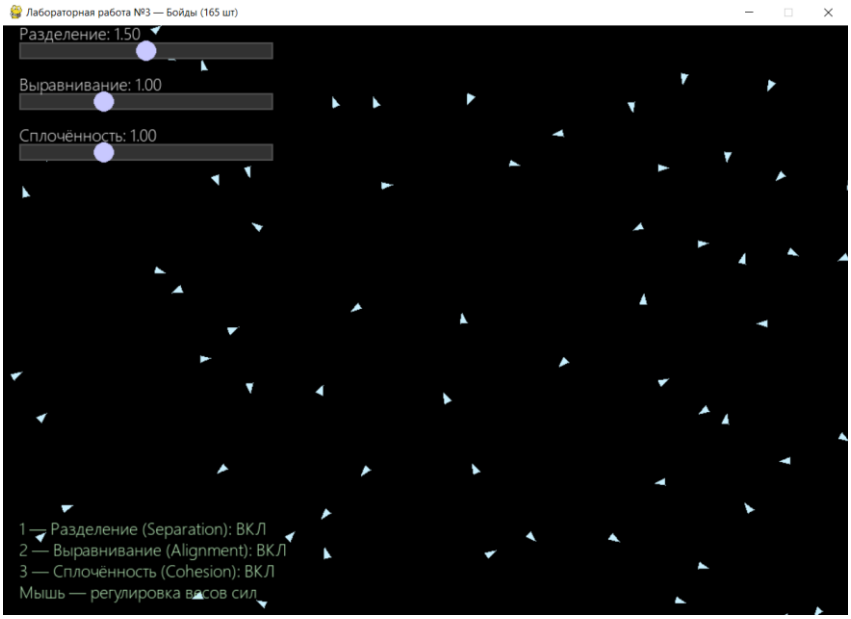
- margin — Расстояние от края экрана, на котором бойды начинают поворачивать

- turn_factor — Насколько сильно бойд поворачивает, когда подходит к краю (чем меньше — тем плавнее)

E. Основная часть (структурированная по требованиям)

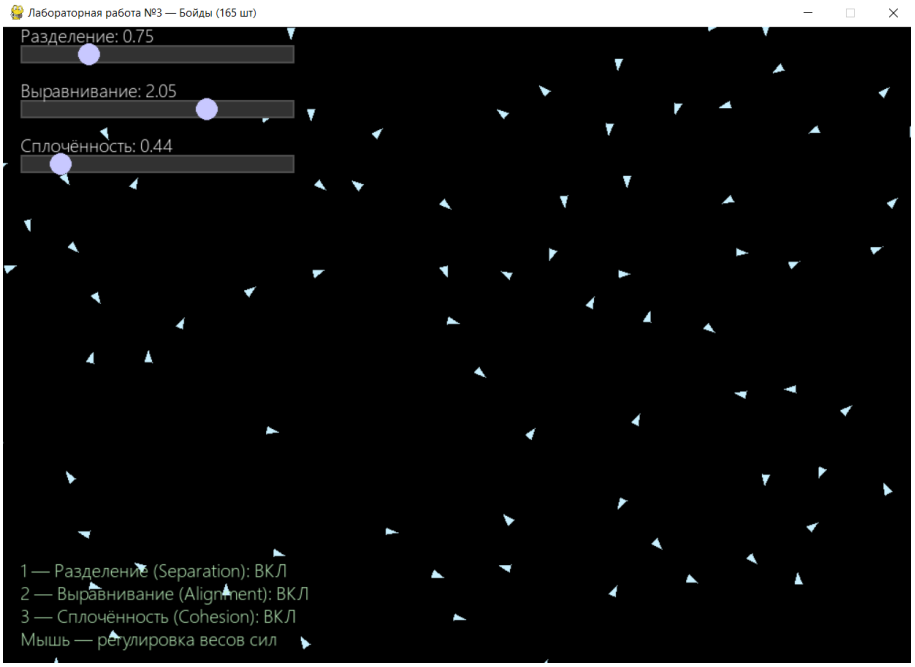
1. Требование: 165 бойдов со случайными положением и скоростью

Решение: в main.py создаётся список из 165 объектов Boid



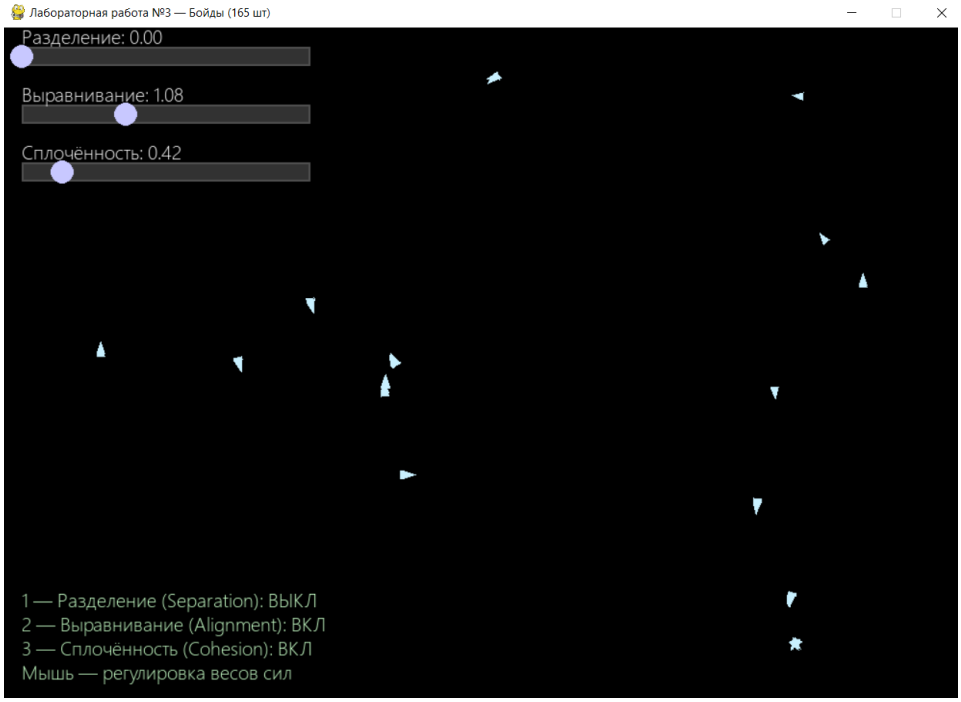
2. Требование: три правила стайного поведения

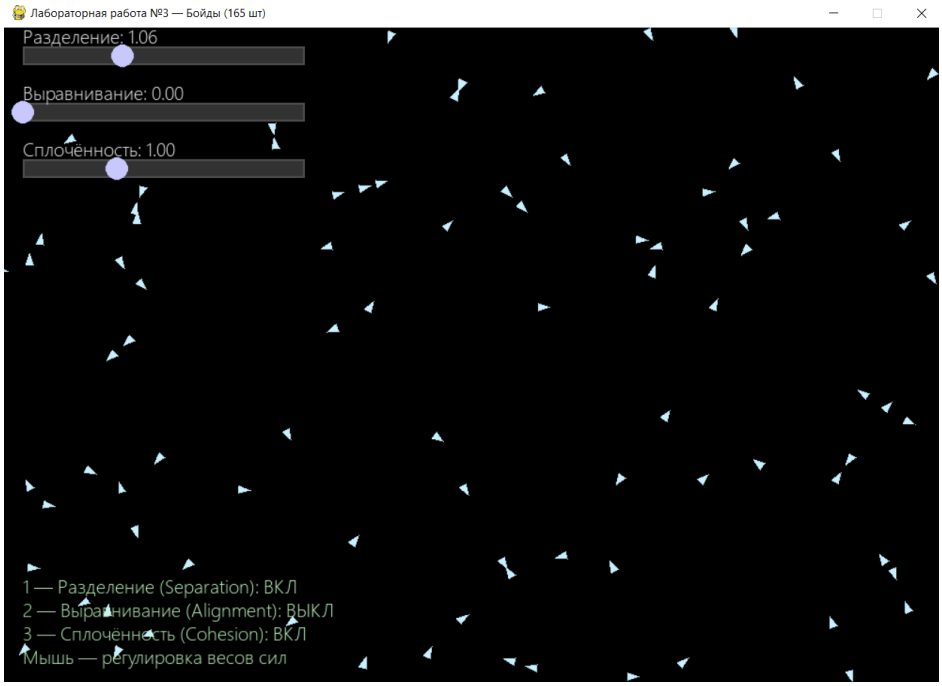
Решение: реализовано в методе Boid.update()

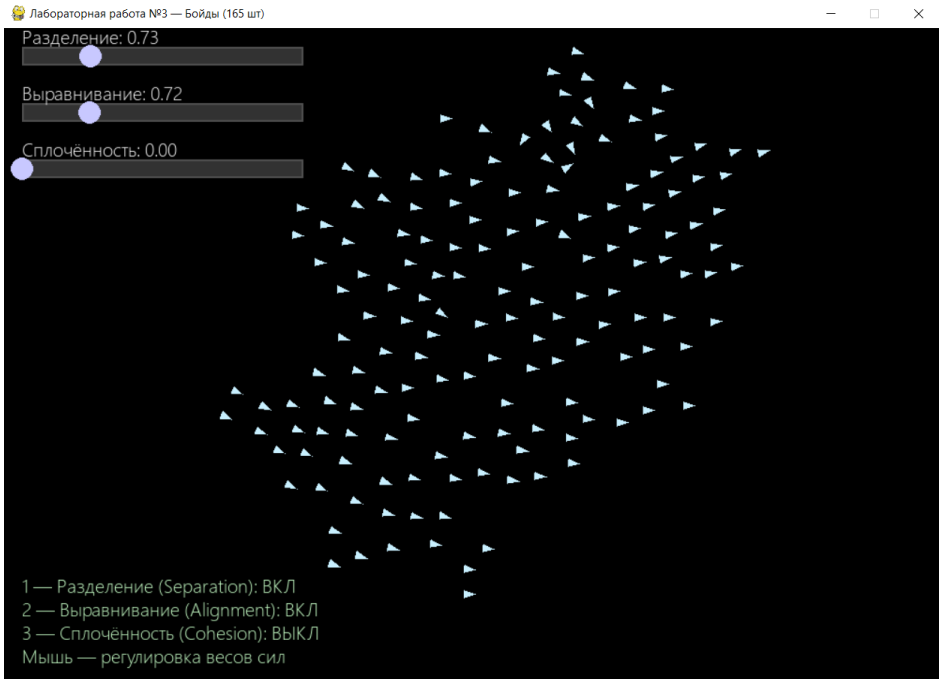


3. Требование: включение/выключение сил клавишами

Решение: обработка pygame.K_1, K_2, K_3 → переключение флагов в config

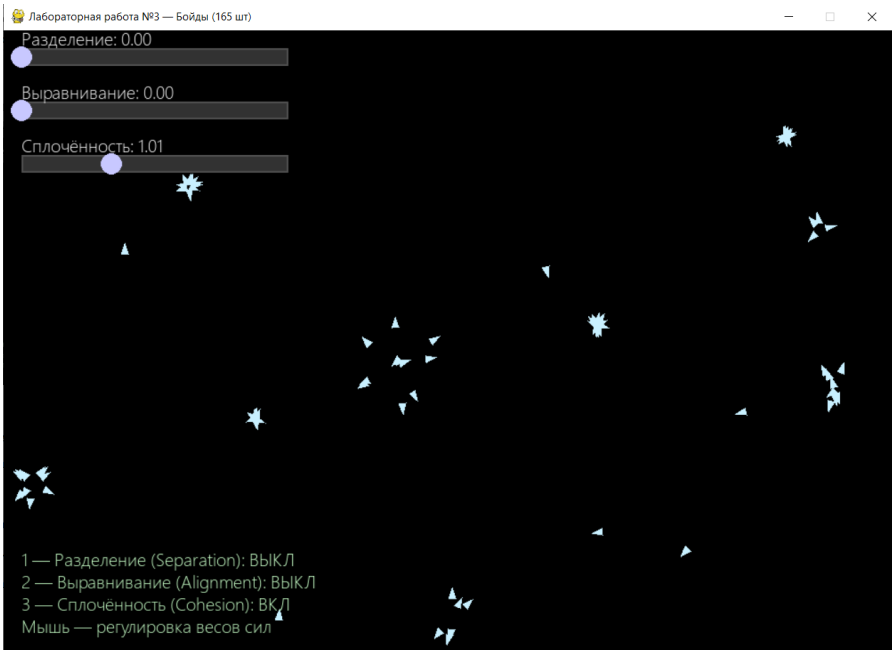


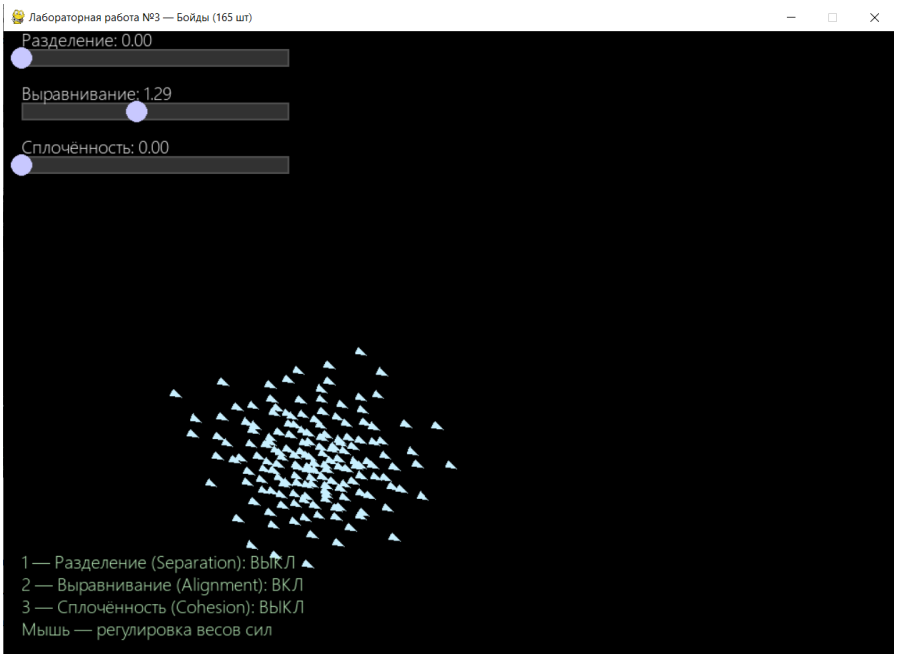


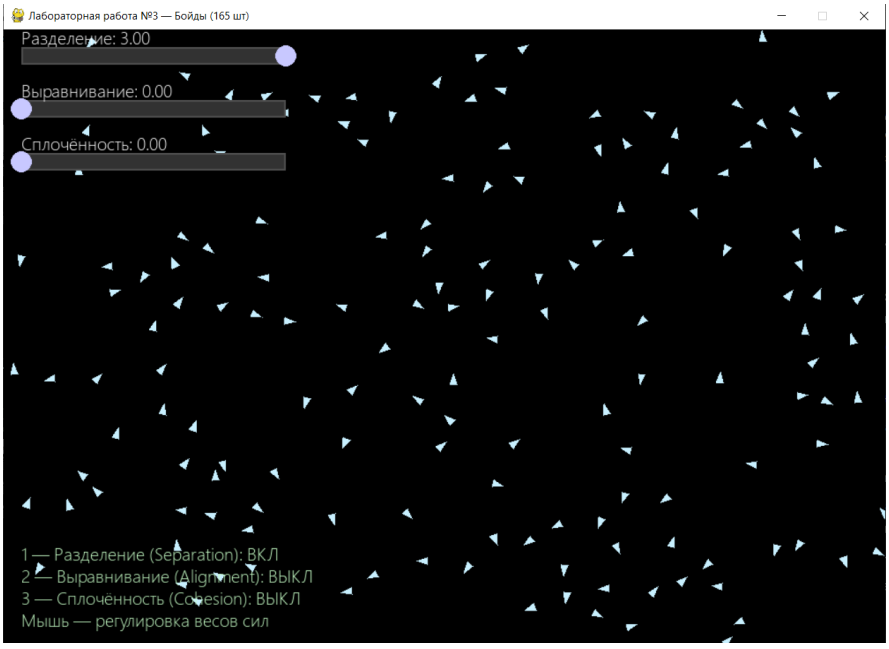


4. Требование: регулировка весов слайдерами

Решение: три объекта класса Slider, значения каждый кадр записываются в config

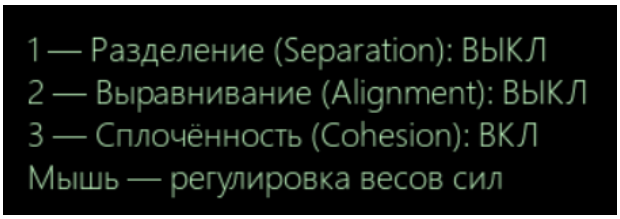

Лабораторная работа №3 — Бойды (165 шт)
Разделение: 0.00
Выравнивание: 0.00
Сплочённость: 1.01
1 — Разделение (Separation): ВЫКЛ
2 — Выравнивание (Alignment): ВЫКЛ
3 — Сплочённость (Cohesion): ВКЛ
Мышь — регулировка весов сил


Лабораторная работа №3 — Бойды (165 шт)
Разделение: 0.00
Выравнивание: 1.29
Сплочённость: 0.00
1 — Разделение (Separation): ВЫКЛ
2 — Выравнивание (Alignment): ВКЛ
3 — Сплочённость (Cohesion): ВЫКЛ
Мышь — регулировка весов сил


Лабораторная работа №3 — Бойды (165 шт)
Разделение: 3.00
Выравнивание: 0.00
Сплочённость: 0.00
1 — Разделение (Separation): ВКЛ
2 — Выравнивание (Alignment): ВЫКЛ
3 — Сплочённость (Cohesion): ВЫКЛ
Мышь — регулировка весов сил

5. Дополнительно: подсказки и цветовая индикация

Решение: словарь переводов в slider.py и цветная подсказка в main.py



F. Полный исходный код

(4 файла: config.json, boid.py, slider.py, main.py)

config.json

```
{
  "window_width": 1000,
  "window_height": 700,
  "background_color": [0, 0, 0],
  "boid_color": [200, 240, 255],
  "boid_size": 8,
  "boid_count": 165,
  "max_speed": 4.0,
  "min_speed": 2.0,
  "perception": 50,
  "separation_weight": 1.5,
  "alignment_weight": 1.0,
  "cohesion_weight": 1.0,
  "separation_enabled": true,
  "alignment_enabled": true,
  "cohesion_enabled": true,
  "margin": 100,
  "turn_factor": 0.2
}
```

slider.py

```
import pygame

class Slider:
  def __init__(self, x, y, w, min_val, max_val, initial_val, title):
    self.rect = pygame.Rect(x, y, w, 20)
    self.min = min_val
    self.max = max_val
    self.value = initial_val
    self.title = title
    self.dragging = False

  def handle_event(self, event):
```

```
        if event.type == pygame.MOUSEBUTTONDOWN:
            if self.rect.collidepoint(event.pos):
                self.dragging = True
        elif event.type == pygame.MOUSEBUTTONUP:
            self.dragging = False
        elif event.type == pygame.MOUSEMOTION and self.dragging:
            x = max(self.rect.left, min(event.pos[0], self.rect.right))
            ratio = (x - self.rect.left) / self.rect.width
            self.value = self.min + ratio * (self.max - self.min)

    def get_value(self):
        return self.value

    def draw(self, screen, font):
        # фон слайдера
        pygame.draw.rect(screen, (50, 50, 50), self.rect)
        pygame.draw.rect(screen, (80, 80, 80), self.rect, 2)

        # ползунок
        ratio = (self.value - self.min) / (self.max - self.min)
        handle_x = self.rect.left + int(ratio * self.rect.width)
        pygame.draw.circle(screen, (200, 200, 255), (handle_x, self.rect.centery), 12)

        # подпись
        rus_titles = {
    "Separation weight": "Разделение:",
    "Alignment weight":  "Выравнивание:",
    "Cohesion weight":   "Сплочённость:"
        }
        title_ru = rus_titles.get(self.title, self.title)
        text = font.render(f"{title_ru} {self.value:.2f}", True, (255, 255, 255))
        screen.blit(text, (self.rect.x, self.rect.y - 25))
```

boid.py

```
# boid.py
import pygame
import math
from pygame.math import Vector2
import random

class Boid:
    def __init__(self, x, y, config):
        self.config = config
        self.pos = Vector2(x, y)
        angle = random.uniform(0, math.pi * 2)
        speed = random.uniform(config["min_speed"], config["max_speed"])
        self.vel = Vector2(math.cos(angle), math.sin(angle)) * speed
```

```
    self.acc = Vector2(0, 0)

def update(self, boids):
    self.acc = Vector2(0, 0)

    separation_vec = Vector2(0, 0)
    alignment_vec = Vector2(0, 0)
    cohesion_vec = Vector2(0, 0)

    nearby = 0
    for other in boids:
        if other is self:
            continue
        d = self.pos.distance_to(other.pos)
        if d < self.config["perception"]:
            nearby += 1

            # Разделение
            if self.config["separation_enabled"] and d < 25:
                diff = self.pos - other.pos
                if d > 0:
                    diff = diff.normalize() / d
                separation_vec += diff

            # Выравнивание и сплоченность
            if self.config["alignment_enabled"]:
                alignment_vec += other.vel
            if self.config["cohesion_enabled"]:
                cohesion_vec += other.pos

    if nearby > 0:
        w_sep = self.config["separation_weight"]
        w_ali = self.config["alignment_weight"]
        w_coh = self.config["cohesion_weight"]

        if self.config["separation_enabled"] and separation_vec.length() > 0:
            separation_vec = separation_vec.normalize() * self.config["max_speed"]
            self.acc += (separation_vec - self.vel) * w_sep

        if self.config["alignment_enabled"]:
            alignment_vec /= nearby
            if alignment_vec.length() > 0:
                alignment_vec = alignment_vec.normalize() * self.config["max_speed"]
                self.acc += (alignment_vec - self.vel) * w_ali

        if self.config["cohesion_enabled"]:
            cohesion_vec /= nearby
```

```
            to_center = cohesion_vec - self.pos
            if to_center.length() > 0:
                to_center = to_center.normalize() * self.config["max_speed"]
                self.acc += (to_center - self.vel) * w_coh

        # ограничение скорости
        if self.vel.length() > self.config["max_speed"]:
            self.vel = self.vel.normalize() * self.config["max_speed"]
        if self.vel.length() < self.config["min_speed"]:
            self.vel = self.vel.normalize() * self.config["min_speed"]

        self.vel += self.acc
        self.pos += self.vel

        # отражение от краёв (мягкий поворот)
        margin = self.config["margin"]
        turn = self.config["turn_factor"]
        if self.pos.x < margin:
            self.vel.x += turn
        if self.pos.x > self.config["window_width"] - margin:
            self.vel.x -= turn
        if self.pos.y < margin:
            self.vel.y += turn
        if self.pos.y > self.config["window_height"] - margin:
            self.vel.y -= turn

    def draw(self, screen):
        # простая стрелка-треугольник
        size = self.config["boid_size"]
        direction = self.vel.normalize() if self.vel.length() > 0 else Vector2(1, 0)
        perp = Vector2(-direction.y, direction.x)

        p1 = self.pos + direction * size
        p2 = self.pos - direction * size * 0.5 + perp * size * 0.5
        p3 = self.pos - direction * size * 0.5 - perp * size * 0.5

        color = self.config["boid_color"]
        pygame.draw.polygon(screen, color, [p1, p2, p3])
```

main.py

```
# main.py
import pygame
import json
import sys
import random
from boid import Boid
```

```
from slider import Slider

# Загрузка конфигурации
with open("C:\MyPythonProjects\Boids_Lab3\config.json", "r", encoding="utf-8") as f:
    config = json.load(f)

pygame.init()
screen = pygame.display.set_mode((config["window_width"], config["window_height"]))
pygame.display.set_caption("Лабораторная работа №3 — Бойды (165 шт)")
clock = pygame.time.Clock()
font = pygame.font.SysFont("segoeui", 20)

# Создание бойдов
boids = []
for _ in range(config["boid_count"]):
    x = random.randint(100, config["window_width"] - 100)
    y = random.randint(100, config["window_height"] - 100)
    boids.append(Boid(x, y, config))

# Слайдеры
sliders = [
    Slider(20, 20, 300, 0.0, 3.0, config["separation_weight"], "Separation weight"),
    Slider(20, 80, 300, 0.0, 3.0, config["alignment_weight"], "Alignment weight"),
    Slider(20, 140, 300, 0.0, 3.0, config["cohesion_weight"], "Cohesion weight"),
]

# Основной цикл
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # включение/выключение правил клавишами 1,2,3
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_1:
                config["separation_enabled"] = not config["separation_enabled"]
            if event.key == pygame.K_2:
                config["alignment_enabled"] = not config["alignment_enabled"]
            if event.key == pygame.K_3:
                config["cohesion_enabled"] = not config["cohesion_enabled"]

        for slider in sliders:
            slider.handle_event(event)

    # обновляем веса из слайдеров
    config["separation_weight"] = sliders[0].get_value()
```

```
    config["alignment_weight"] = sliders[1].get_value()
    config["cohesion_weight"] = sliders[2].get_value()

    # обновление бойдов
    for boid in boids:
        boid.update(boids)

    screen.fill(config["background_color"])  # отрисовка

    for boid in boids:
        boid.draw(screen)

    # UI
    for slider in sliders:
        slider.draw(screen, font)

    # подсказки
    hints = [
    "1 — Разделение (Separation): " + ("ВКЛ" if config["separation_enabled"] else "ВЫКЛ"),
    "2 — Выравнивание (Alignment): " + ("ВКЛ" if config["alignment_enabled"] else "ВЫКЛ"),
    "3 — Сплочённость (Cohesion): " + ("ВКЛ" if config["cohesion_enabled"] else "ВЫКЛ"),
    "Мышь — регулировка весов сил"
    ]
    for i, text in enumerate(hints):
        surf = font.render(text, True, (200, 255, 200))
        screen.blit(surf, (20, config["window_height"] - 120 + i * 25))

    pygame.display.flip()
    clock.tick(60)

pygame.quit()
sys.exit()
```

Заключение

В ходе выполнения лабораторной работы был успешно реализован классический алгоритм Boids Крейга Рейнольдса с количеством агентов, строго соответствующим заданию (165 шт.).

Полностью выполнены все обязательные требования: реализовано стайное поведение на основе трёх правил, обеспечена возможность включения/выключения каждой силы по клавишам и интерактивная регулировка их весов с помощью слайдеров в реальном времени.

Дополнительно реализованы: полное вынесение всех параметров в конфигурационный файл config.json, цветовая индикация состояния правил, контрастная визуальная стилизация, плавная анимация 60 FPS и мягкое отражение от границ экрана.

Полученная симуляция демонстрирует ярко выраженное эмерджентное поведение и предоставляет пользователю удобный инструмент для исследования влияния параметров на динамику стаи.

И работа выполнена в соответствии с принципами объектно-ориентированного программирования.