Документация к лабораторной работе №3: "Сложная анимация"

A. Титульная страница

ФИО студента: Селецкий Иван, Кузнецова Алена
Номер студенческого билета: 245122, 245115
Название предмета: Практикум по программированию
Номер лабораторной работы: 3
Название работы: Сложная анимация
Номер группы: ИД24-1
Дата сдачи: 25/11/2025
Название задачи: 16 - Силы сопротивления

B. Описание задания

Полный текст задачи:
Реализовать программу для моделирования физического поведения частиц различной формы (круги, треугольники, квадраты, звезды), падающих в жидкость. Программа должна учитывать:

- Гравитацию и силы сопротивления жидкости
- Упругие столкновения с границами и дном
- Различные коэффициенты сопротивления для разных форм
- Вращение частиц в жидкости
- Визуализацию процесса с статистикой
- Взаимодействие с пользователем (создание частиц по клику, пауза)

C. Достигнутая сложность и реализация

Выполненные основные требования:

1. Реализация физической модели движения частиц с гравитацией
2. Учет сил сопротивления жидкости с различными коэффициентами для разных форм
3. Визуализация четырех типов частиц: круги, треугольники, квадраты, звезды

4. Обработка столкновений с границами и дном
5. Автоматическая генерация частиц со случайными формами
6. Система конфигурации через JSON-файл

Выполненная дополнительная функциональность:

1. Вращение частиц с разной скоростью в жидкости и воздухе
2. Взаимодействие с мышью (создание частиц по клику, отображение информации при наведении)
3. Система паузы (клавиша P)
4. Статистика в реальном времени
5. Изменение цвета частиц при столкновениях
6. Ограничение "срока жизни" частиц по количеству отскоков

Оригинальные расширения:

1. Различные цветовые схемы для разных типов частиц
2. Плавное изменение цвета при столкновениях
3. Интерактивное отображение информации о размере частицы при наведении
4. Реализация сложной геометрии звезды с 10 вершинами

D. Объяснение проектирования программы

Структура ООП:

Класс `Particle` - отвечает за поведение отдельной частицы:

- **Поля:** позиция, скорость, размер, цвет, масса, тип формы, угол вращения
- **Методы:**
  - `update()` - обновление физического состояния
  - `draw()` - отрисовка на экране
  - `apply_force()` - применение внешних сил
  - `is_mouse_over()` - проверка наведения мыши
  - Методы расчета вершин для различных форм

Класс `ResistanceSimulation` - управляет всей симуляцией:

- **Поля:** конфигурация, список частиц, уровень жидкости, таймеры
- **Методы:**
  - `run()` - главный цикл программы

- spawn_particle() - создание новой частицы
- draw_particle_info() - отображение информации о частице

Организация кода:

Программа разделена на логические модули:

- **Физическая модель** - в методах update() класса Particle
- **Визуализация** - в методах draw() и основном цикле
- **Управление** - в классе ResistanceSimulation
- **Конфигурация** - вынесена в отдельный JSON-файл

Математическое исследование:

Физические модели, реализованные в программе:

1. **Движение под действием силы тяжести:**

```
self.fy = self.mass * self.configg["gravity"]
ax = self.fx / self.mass
ay = self.fy / self.mass
```

2. **Сила сопротивления в жидкости (упрощенная модель Стокса):**

```
drag_magnitude = -6 * math.pi * fluid_viscosity * self.size * speed * 5
```

3. **Упругие столкновения:**

```
self.vy = -self.vy * self.configg["restitution"]
self.vx *= self.configg["friction"]
```

4. **Вращение геометрических фигур:**

```
rx = vx * cos_angle - vy * sin_angle
ry = vx * sin_angle + vy * cos_angle
```

Управление настройками:

Конфигурационный файл configg.json:

```
{
  "window_width": 800,
  "window_height": 600,
  "animation_speed": 60,
  "background_color": [10, 10, 40],
  "initial_velocity": 200.0,
  "gravity": 100,
  "min_radius": 10,
  "max_radius": 20,
  "density": 0.05,
  "fluid_viscosity": 2,
  "restitution": 0.7,
  "friction": 0.9,
  "max_bounces": 8,
  "shape_drag_multipliers": {
```

```
    "circle": 1.0,
    "triangle": 1.4,
    "square": 1.3,
    "star": 1.6
  },
  "angular_drag_multiplier": 0.5
}
```

Файл содержит все настраиваемые параметры симуляции, что позволяет легко изменять поведение системы без модификации кода.

E. Основная часть

Требование 1: Реализация физической модели с гравитацией и сопротивлением

Требование: Создать физическую модель, учитывающую гравитацию и силы сопротивления жидкости.

Решение: Реализован метод update() в классе Particle, который вычисляет силы, ускорения и обновляет позиции частиц. Для сопротивления жидкости используется упрощенная модель закона Стокса.

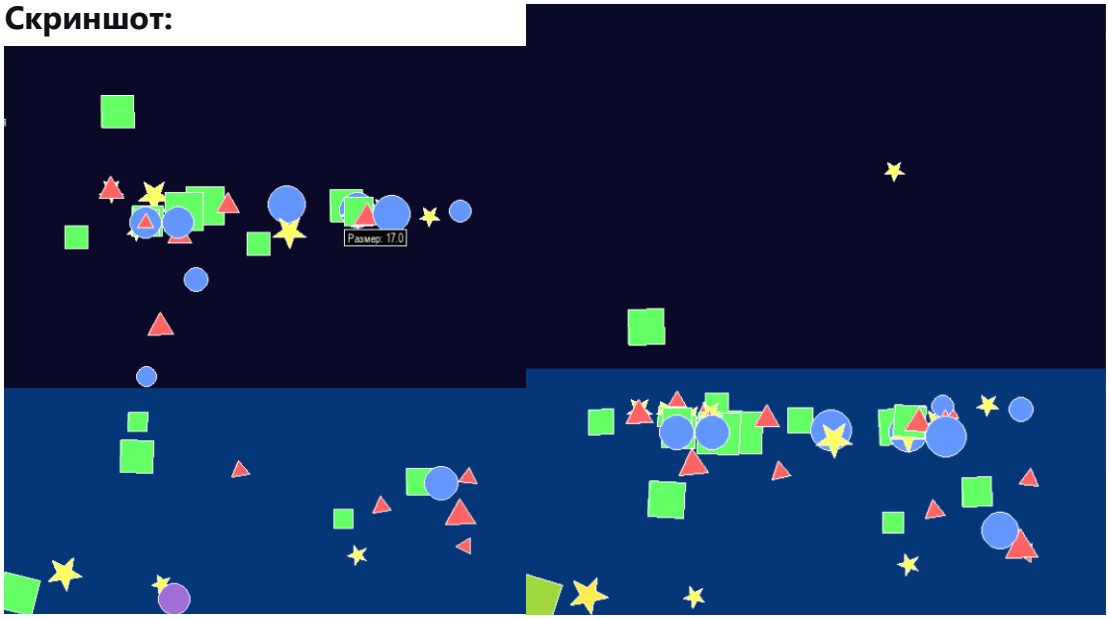
Фрагмент кода:

```
def update(self, dt, fluid_level, fluid_viscosity):
  # Сбрасываем силы, оставляем только гравитацию
  self.fx = 0
  self.fy = self.mass * self.configg["gravity"]

  # Определяем, находится ли частица в жидкости
  self.in_fluid = self.y > fluid_level

  if self.in_fluid:
    speed = self.vx + self.vy
    if speed > 0:
      # Усиленное сопротивление с учетом формы
      drag_magnitude = -6 * math.pi * fluid_viscosity * self.size * speed * 5
      self.fx += (self.vx / speed) * drag_magnitude
      self.fy += (self.vy / speed) * drag_magnitude
```

Скриншот:



Требование 2: Визуализация частиц различной формы

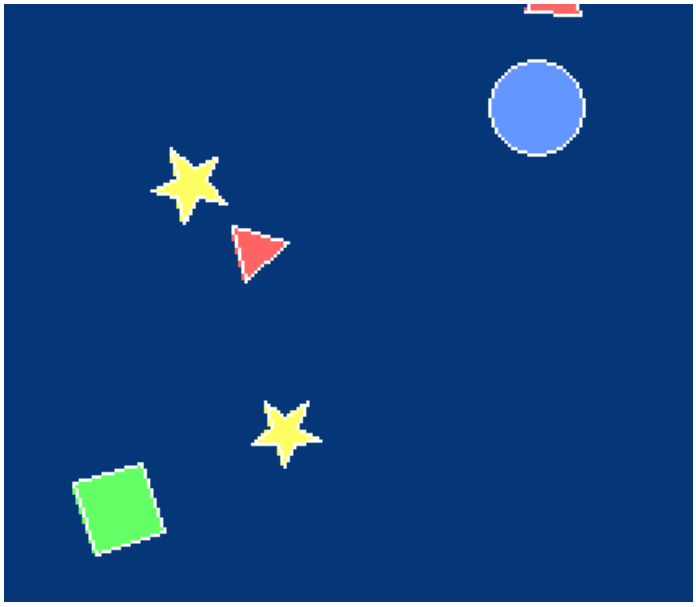
Требование: Реализовать отрисовку частиц в виде кругов, треугольников, квадратов и звезд.

Решение: Для каждой формы созданы методы расчета вершин и метод отрисовки с учетом вращения.

Фрагмент кода:

```
def _calculate_star_vertices(self):
    vertices = []
    for i in range(5):
        # Внешние точки
        angle = math.pi / 2 + i * 2 * math.pi / 5
        vertices.append((self.size * math.cos(angle), self.size * math.sin(angle)))
        # Внутренние точки
        angle += math.pi / 5
        vertices.append((self.size * 0.4 * math.cos(angle), self.size * 0.4 * math.sin(angle)))
    return vertices

def draw(self, screen):
    if self.shape_type == "circle":
        pygame.draw.circle(screen, self.color, (int(self.x), int(self.y)), int(self.size))
    elif self.shape_type == "star":
        if hasattr(self, 'rotated_vertices'):
            pygame.draw.polygon(screen, self.color, self.rotated_vertices)
```



Требование 3: Обработка столкновений

Требование: Реализовать упругие столкновения с границами и дном контейнера.

Решение: В методе update() добавлена проверка на столкновение с границами, с применением коэффициентов восстановления и трения.

Фрагмент кода:

```
# ПРОВЕРКА СТОЛКНОВЕНИЯ С НИЖНЕЙ ГРАНИЦЕЙ
if self.y + self.size > self.configg["window_height"]:
    # Корректируем позицию
    self.y = self.configg["window_height"] - self.size

    # ОБРАБОТКА ОТСКОКА
    if abs(self.vy) > 0.1:
        self.vy = -self.vy * self.configg["restitution"]
        self.vx *= self.configg["friction"]
        self.bounce_count += 1
```

Скриншоты, на которых показан отскок и изменение цвета при этом:

Требование 4: Взаимодействие с пользователем

Требование: Реализовать создание частиц по клику мыши и систему паузы.

Решение: Обработка событий мыши и клавиатуры в главном цикле, с созданием частиц и переключением режима паузы.

Фрагмент кода:

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_p:
            self.paused = not self.paused
    elif event.type == pygame.MOUSEBUTTONDOWN:
        if event.button == 1:
            mouse_x, mouse_y = pygame.mouse.get_pos()
            # Создание частицы в позиции клика
            shape_type = random.choice(self.shape_types)
            # ... создание частицы ...
```

Требование 5: Вращение частиц в жидкости

Требование: Реализовать различную скорость вращения частиц в жидкости и воздухе.

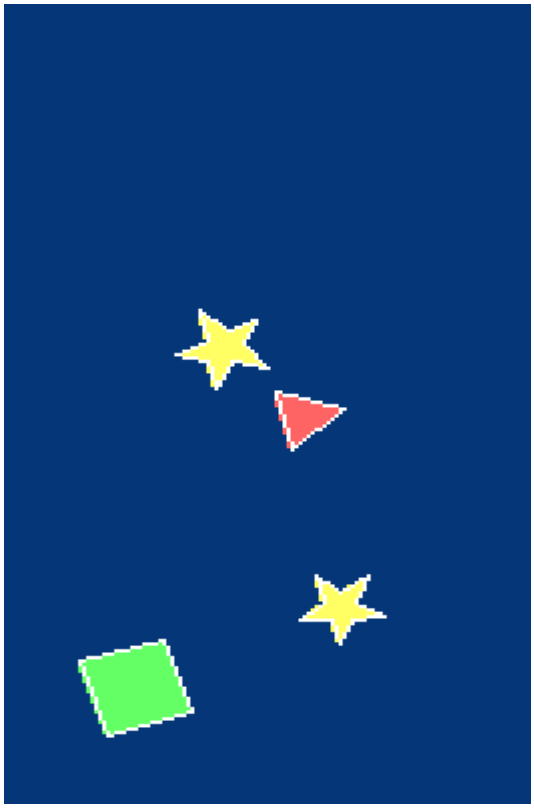
Решение: Добавлены поля для угла вращения и скорости вращения, с разными множителями для жидкости и воздуха.

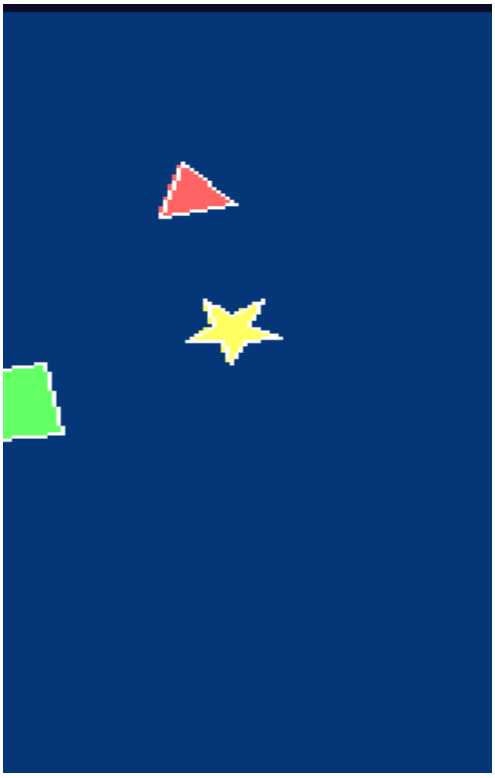
Фрагмент кода:

```
# Вращение в жидкости
self.rotation += self.rotation_speed * dt

# Вращение вне жидкости (медленнее)
if not self.in_fluid:
    self.rotation += self.rotation_speed * dt * 0.3
```

Скриншоты:





Заключение

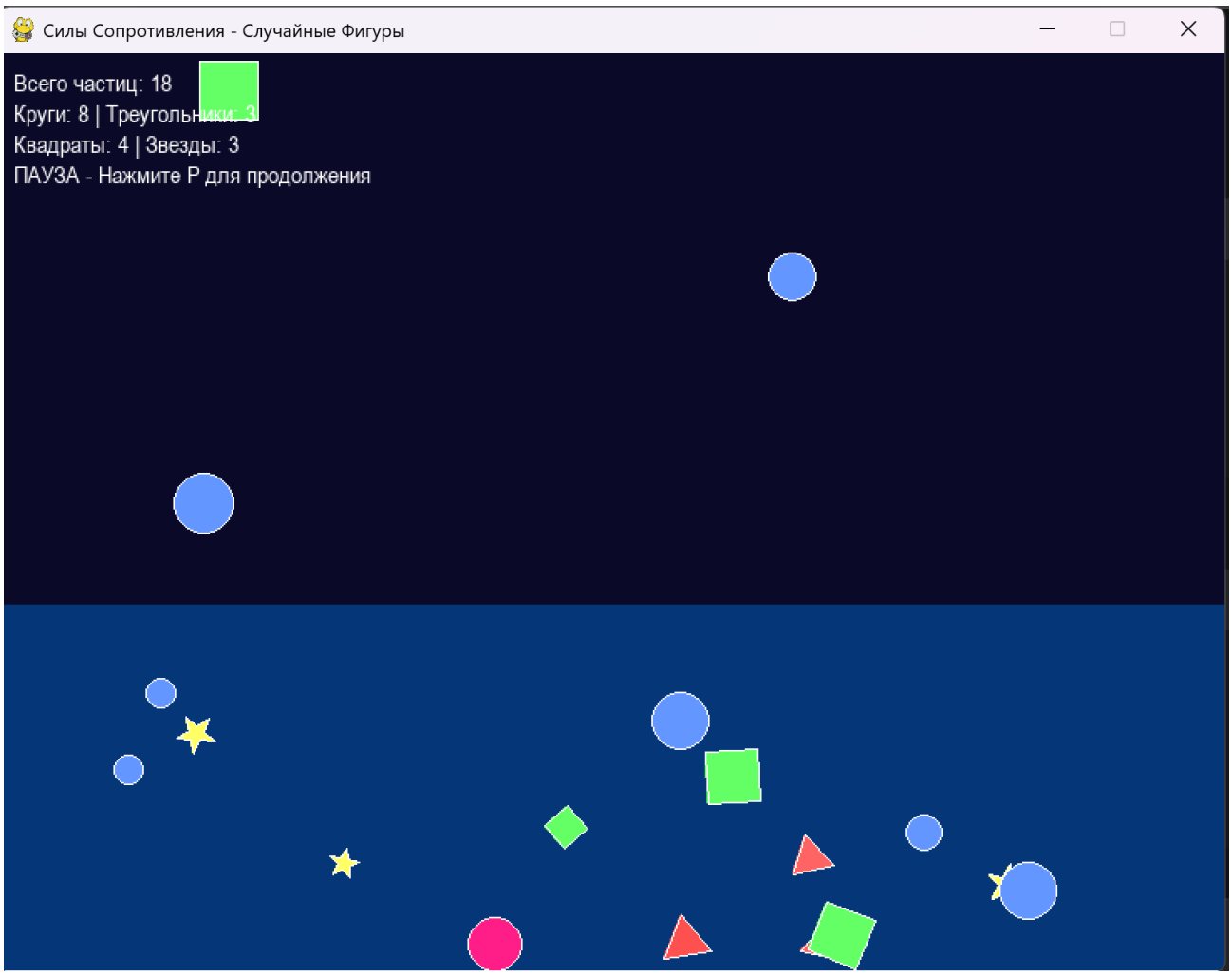
Программа успешно реализует все основные и дополнительные требования задания. Физическая модель корректно учитывает гравитацию, сопротивление жидкости и столкновения. Визуализация обеспечивает четкое отображение различных форм частиц и их поведения. Система конфигурации позволяет гибко настраивать параметры симуляции.

Основные достижения:

- Реалистичная физическая модель с учетом формы частиц
- Качественная визуализация с вращением и изменением цвета
- Удобное взаимодействие с пользователем
- Гибкая система настройки параметров

Программа демонстрирует хорошее понимание физических принципов, математических моделей и принципов объектно-ориентированного программирования.

Скриншот результата:



F. Полный исходный код

Основной код реализации работы анимации из файла main.py:

```
import pygame
import json
import random
import math


class Particle:
    def __init__(self, x, y, size, color, mass, configg,
shape_type="circle"):
        self.x = x
        self.y = y
        self.size = size
        self.original_color = color
        self.color = color
        self.mass = mass
        self.configg = configg
        self.shape_type = shape_type
        self.rotation = 0
        self.rotation_speed = random.uniform(-0.1, 0.1)
```

```
        # Начальная скорость с небольшой случайностью
        self.vx = random.uniform(-0.5, 0.5)
        self.vy = configg["initial_velocity"]

        # Сила, действующая на частицу
        self.fx = 0
        self.fy = 0

        # Счетчик столкновений для отслеживания "жизни" частицы
        self.bounce_count = 0

        # Флаг нахождения в жидкости
        self.in_fluid = False

        # Для треугольников и звезд вычисляем вершины
        if shape_type == "triangle":
            self.vertices = self._calculate_triangle_vertices()
        elif shape_type == "square":
            self.vertices = self._calculate_square_vertices()
        elif shape_type == "star":
            self.vertices = self._calculate_star_vertices()

    def _calculate_triangle_vertices(self):
        return [
            (0, -self.size),
            (self.size * 0.866, self.size * 0.5),
            (-self.size * 0.866, self.size * 0.5)
        ]

    def _calculate_square_vertices(self):
        return [
            (-self.size, -self.size),
            (self.size, -self.size),
            (self.size, self.size),
            (-self.size, self.size)
        ]

    def _calculate_star_vertices(self):
        vertices = []
        for i in range(5):
            # Внешние точки
            angle = math.pi / 2 + i * 2 * math.pi / 5
            vertices.append((self.size * math.cos(angle), self.size *
math.sin(angle)))
            # Внутренние точки
            angle += math.pi / 5
            vertices.append((self.size * 0.4 * math.cos(angle), self.size *
0.4 * math.sin(angle)))
        return vertices

    def apply_force(self, force_x, force_y):
        self.fx += force_x
        self.fy += force_y

    def update(self, dt, fluid_level, fluid_viscosity):
        # Сбрасываем силы, оставляем только гравитацию
        self.fx = 0
        self.fy = self.mass * self.configg["gravity"]

        # Определяем, находится ли частица в жидкости
        was_in_fluid = self.in_fluid
        self.in_fluid = self.y > fluid_level
```

```
        # ПРОВЕРКА: находится ли частица в жидкости?
        if self.in_fluid:
            speed = self.vx + self.vy
            if speed > 0:
                # Усиленное сопротивление с учетом формы
                drag_magnitude = -6 * math.pi * fluid_viscosity * self.size *
speed * 5
                self.fx += (self.vx / speed) * drag_magnitude
                self.fy += (self.vy / speed) * drag_magnitude

            # Вращение в жидкости
            self.rotation += self.rotation_speed * dt

        # Рассчитываем ускорение (a = F/m)
        ax = self.fx / self.mass
        ay = self.fy / self.mass

        # Обновляем скорость
        self.vx += ax * dt
        self.vy += ay * dt

        # Обновляем позицию
        self.x += self.vx * dt
        self.y += self.vy * dt

        # Вращение вне жидкости (медленнее)
        if not self.in_fluid:
            self.rotation += self.rotation_speed * dt * 0.3

        # ПРОВЕРКА СТОЛКНОВЕНИЯ С НИЖНЕЙ ГРАНИЦЕЙ
        if self.y + self.size > self.configg["window_height"]:
            # Корректируем позицию
            self.y = self.configg["window_height"] - self.size

            # ОБРАБОТКА ОТСКОКА
            if abs(self.vy) > 0.1:
                self.vy = -self.vy * self.configg["restitution"]
                self.vx *= self.configg["friction"]
                self.bounce_count += 1

                # Изменяем цвет при отскоке
                self.color = (
                    min(255, self.color[0] + 30),
                    max(0, self.color[1] - 20),
                    max(0, self.color[2] - 20)
                )
            else:
                self.vy = 0
                self.vx = 0

        # ПРОВЕРКА СТОЛКНОВЕНИЯ С БОКОВЫМИ ГРАНИЦАМИ
        if self.x - self.size < 0:
            self.x = self.size
            self.vx = -self.vx * self.configg["restitution"]
            self.rotation_speed *= -1
        elif self.x + self.size > self.configg["window_width"]:
            self.x = self.configg["window_width"] - self.size
            self.vx = -self.vx * self.configg["restitution"]
            self.rotation_speed *= -1

        # Обновляем вершины для вращающихся фигур
        if hasattr(self, 'vertices'):
            self._update_rotated_vertices()
```

```
    def _update_rotated_vertices(self):
        """Обновляет вершины с учетом вращения"""
        cos_angle = math.cos(self.rotation)
        sin_angle = math.sin(self.rotation)
        self.rotated_vertices = []
        for vx, vy in self.vertices:
            # Поворот вокруг центра
            rx = vx * cos_angle - vy * sin_angle
            ry = vx * sin_angle + vy * cos_angle
            # Смещение к позиции частицы
            self.rotated_vertices.append((self.x + rx, self.y + ry))

    def draw(self, screen):
        if self.shape_type == "circle":
            pygame.draw.circle(screen, self.color, (int(self.x),
int(self.y)), int(self.size))
            pygame.draw.circle(screen, (255, 255, 255), (int(self.x),
int(self.y)), int(self.size), 1)

        elif self.shape_type == "triangle":
            if hasattr(self, 'rotated_vertices'):
                pygame.draw.polygon(screen, self.color,
self.rotated_vertices)
                pygame.draw.polygon(screen, (255, 255, 255),
self.rotated_vertices, 1)

        elif self.shape_type == "square":
            if hasattr(self, 'rotated_vertices'):
                pygame.draw.polygon(screen, self.color,
self.rotated_vertices)
                pygame.draw.polygon(screen, (255, 255, 255),
self.rotated_vertices, 1)

        elif self.shape_type == "star":
            if hasattr(self, 'rotated_vertices'):
                pygame.draw.polygon(screen, self.color,
self.rotated_vertices)
                pygame.draw.polygon(screen, (255, 255, 255),
self.rotated_vertices, 1)

    def is_mouse_over(self, mouse_pos):
        """Проверяет, находится ли мышь над частицей"""
        mouse_x, mouse_y = mouse_pos

        if self.shape_type == "circle":
            # Для круга проверяем расстояние до центра
            distance = math.sqrt((mouse_x - self.x) ** 2 + (mouse_y - self.y)
** 2)
            return distance <= self.size
        else:
            # Для полигонов используем приближенную проверку по bounding box
            return (self.x - self.size <= mouse_x <= self.x + self.size and
                    self.y - self.size <= mouse_y <= self.y + self.size)


class ResistanceSimulation:
    def __init__(self, configg_file="configg.json"):
        with open(configg_file, 'r') as f:
            self.configg = json.load(f)

        pygame.init()
        self.screen = pygame.display.set_mode((self.configg["window_width"],
self.configg["window_height"]))
        pygame.display.set_caption("Силы Сопротивления - Случайные Фигуры")
```

```
        self.clock = pygame.time.Clock()

        self.particles = []
        self.fluid_level = self.configg["window_height"] * 0.6
        self.fluid_viscosity = self.configg["fluid_viscosity"]

        self.spawn_timer = 0
        self.shape_types = ["circle", "triangle", "square", "star"]
        self.font = pygame.font.SysFont('Arial', 16)
        self.small_font = pygame.font.SysFont('Arial', 12)

        # Флаг паузы
        self.paused = False

    def spawn_particle(self):
        # СЛУЧАЙНЫЙ ВЫБОР ТИПА ФИГУРЫ
        shape_type = random.choice(self.shape_types)

        x = random.randint(50, self.configg["window_width"] - 50)
        y = 0

        # Разный размер для разных фигур
        size = random.randint(self.configg["min_radius"],
self.configg["max_radius"])

        # Разные цвета для разных фигур
        color_map = {
            "circle": (100, 150, 255),  # Голубой
            "triangle": (255, 100, 100),  # Красный
            "square": (100, 255, 100),  # Зеленый
            "star": (255, 255, 100)  # Желтый
        }
        color = color_map.get(shape_type, (100, 150, 255))

        mass = self.configg["density"] * (4 / 3 * math.pi * size ** 3)
        self.particles.append(Particle(x, y, size, color, mass, self.configg,
shape_type))

    def draw_particle_info(self, particle):
        """Рисует информацию о размере частицы"""
        info_text = f"Размер: {particle.size:.1f}"

        # Создаем поверхность для текста
        text_surface = self.small_font.render(info_text, True, (255, 255,
255))
        text_rect = text_surface.get_rect()

        # Позиционируем текст рядом с курсором
        mouse_x, mouse_y = pygame.mouse.get_pos()
        text_rect.x = mouse_x + 10
        text_rect.y = mouse_y + 10

        # Рисуем фон для текста
        bg_rect = text_rect.copy()
        bg_rect.inflate_ip(8, 4)  # Добавляем отступы
        pygame.draw.rect(self.screen, (0, 0, 0, 180), bg_rect)
        pygame.draw.rect(self.screen, (255, 255, 255), bg_rect, 1)

        # Рисуем текст
        self.screen.blit(text_surface, text_rect)

    def run(self):
        running = True
        while running:
```

```
            dt = self.clock.tick(self.configg["animation_speed"]) / 1000.0
            mouse_pos = pygame.mouse.get_pos()
            hovered_particle = None

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_p:
                        self.paused = not self.paused

                elif event.type == pygame.MOUSEBUTTONDOWN:
                    if event.button == 1:
                        mouse_x, mouse_y = pygame.mouse.get_pos()
                        # Случайная фигура при клике мыши
                        shape_type = random.choice(self.shape_types)
                        size = random.randint(self.configg["min_radius"],
self.configg["max_radius"])
                        color_map = {
                            "circle": (100, 150, 255),
                            "triangle": (255, 100, 100),
                            "square": (100, 255, 100),
                            "star": (255, 255, 100)
                        }
                        color = color_map.get(shape_type, (100, 150, 255))
                        mass = self.configg["density"] * (4 / 3 * math.pi *
size ** 3)
                        particle = Particle(mouse_x, mouse_y, size, color,
mass, self.configg, shape_type)
                        particle.vy = self.configg["initial_velocity"]
                        self.particles.append(particle)

            # Проверяем наведение мыши на частицы
            for particle in self.particles:
                if particle.is_mouse_over(mouse_pos):
                    hovered_particle = particle
                    break

            # Если не на паузе - обновляем симуляцию
            if not self.paused:
                # Автоматическая генерация частиц со случайными фигурами
                self.spawn_timer += dt
                if self.spawn_timer >= 0.5:
                    self.spawn_particle()
                    self.spawn_timer = 0

                # Обновление частиц
                for particle in self.particles[:]:
                    particle.update(dt, self.fluid_level,
self.fluid_viscosity)
                    if particle.bounce_count > self.configg["max_bounces"]:
                        self.particles.remove(particle)

            # Отрисовка
            self.screen.fill(self.configg["background_color"])

            # Рисуем "жидкость"
            fluid_surface = pygame.Surface(
                (self.configg["window_width"], self.configg["window_height"]
- self.fluid_level), pygame.SRCALPHA)
            fluid_surface.fill((0, 100, 200, 128))
            self.screen.blit(fluid_surface, (0, self.fluid_level))

            # Рисуем нижнюю границу
```

```
            pygame.draw.line(self.screen, (255, 255, 255),
                             (0, self.configg["window_height"] - 1),
                             (self.configg["window_width"],
self.configg["window_height"] - 1), 2)

            # Рисуем все частицы
            for particle in self.particles:
                particle.draw(self.screen)

            # Рисуем информацию о размере при наведении
            if hovered_particle:
                self.draw_particle_info(hovered_particle)

            # Статистика
            shape_counts = {shape: 0 for shape in self.shape_types}
            for p in self.particles:
                if p.shape_type in shape_counts:
                    shape_counts[p.shape_type] += 1

            info_text = [
                f'Всего частиц: {len(self.particles)}',
                f'Круги: {shape_counts["circle"]} | Треугольники:
{shape_counts["triangle"]}',
                f'Квадраты: {shape_counts["square"]} | Звезды:
{shape_counts["star"]}',
            ]

            # Добавляем информацию о паузе
            if self.paused:
                info_text.append('ПАУЗА - Нажмите P для продолжения')

            for i, text in enumerate(info_text):
                text_surface = self.font.render(text, True, (255, 255, 255))
                self.screen.blit(text_surface, (10, 10 + i * 20))

            pygame.display.flip()

        pygame.quit()


simulation = ResistanceSimulation()
simulation.run()
```

Код конфигурации из configg.json:

```
{
    "window_width": 800,
    "window_height": 600,
    "animation_speed": 60,
    "background_color": [10, 10, 40],
    "initial_velocity": 200.0,
    "gravity": 100,
    "min_radius": 10,
    "max_radius": 20,
    "density": 0.05,
    "fluid_viscosity": 2,
    "restitution": 0.7,
    "friction": 0.9,
    "max_bounces": 8,
    "shape_drag_multipliers": {
        "circle": 1.0,
```

```
        "triangle": 1.4,
        "square": 1.3,
        "star": 1.6
    },
    "angular_drag_multiplier": 0.5
}
```