Маланова Татьяна Денисовна

242080

Практикум по программированию

Лабораторная работа 13

ИД24-2

25.11.2025

Черная дыра

Реализуйте отрисовку стимуляции параллельного пучка фотонов, подходящего вблизи черной дыры и искривляющегося под действием ее гравитации.

1. Позвольте пользователю регулировать массу черной дыры при помощи слайдера.

Выполненные требования:

- Создание окна приложения , в котором будут фотоны и черная дыра

-Создание черной дыры

-Создание фотонов и их траектории

- Фотоны искривляются под действием гравитации и входят в ядро

-Окончательное создание объектов и параллельности фотонов

-Запуск цикла инициализации черной дыры ,фотонов, слайдера,FPS.

Дополнительная функциональность:

- Слайдер для регулировки массы и вывод FPS

- при изменении массы ядро меняется еще и визуально

Оригинальные расширения:

-Разделение логики и отрисовки через smooth_surface
-Модульная структура с четким разделением ответственности, то есть каждый класс имеет одну основную задачу.

Структура ОПП:

3 основных класса:

1. **BlackHole** - чёрная дыра

Управляет массой, радиусом
Отрисовывает ядро и кольца

2. **Photon** - фотон

Движется под гравитацией

ранит трек движения
Сам себя отрисовывает

3. **Slider** - слайдер управления

Обрабатывает ввод мыши
Отображает значение массы

Принципы:

Инкапсуляция - каждый класс сам управляет своими данными
Взаимодействие - фотоны получают черную дыру для расчетов
Отделение логики - физика, отрисовка и UI в разных классах

Организация кода:

1) Импорт библиотек
2) Загрузка файла json
3) Инициализация PyGame
4) Класс черной дыры
5) Класс фотонов
6) Класс слайдера
7) Создание обьектов
8) Главный цикл и запуск

Математическое исследование :

```
# радиус Шварцшильда пропорционален массе : R = 2GM/c²
# Отношение текущей массы к начальной
mass_ratio = self.mass / config['initial_mass']
# Радиус изменяется пропорционально массе
self.radius = int(self.base_radius * mass_ratio)
```

Физическая формула радиуса Шварцшильда: $R = 2GM/c^2$

Параметры:

- **self.mass** - масса черной дыры (M из формулы)
- **config['initial_mass']** - начальная масса (M_0 для масштабирования)
- **self.base_radius** - базовый радиус (R_0 при начальной массе)
- **scale** - коэффициент масштабирования = M/M_0
- **self.radius** - текущий радиус = $R_0 \times (M/M_0)$

В упрощенной версии:

- **G** (гравитационная постоянная) - скрыта в config['gravity_strength']
- **c** (скорость света) - не используется явно, так как мы работаем с относительными величинами

```
# Ньютоновская гравитация: F = G * M / r^2
force = (black_hole.mass * config['gravity_strength']) / (distance * distance)
```

Физическая формула: $F = G \times M / r^2$

Параметры:

- **black_hole.mass** - масса черной дыры (из слайдера)
- **config['gravity_strength']** - "гравитационная постоянная" в нашей симуляции
- **distance * distance** - квадрат расстояния (r^2)

```
# Обновляем скорость фотона - Ускорение = Сила × Направление
self.speed_x += force * nx
self.speed_y += force * ny
```

Физический закон: Второй закон Ньютона $F = m \times a$ (Сила = масса × ускорение)

Параметры:

- **force** - гравитационная сила (F)
- **nx, ny** - направление силы (единичный вектор)
- **force * nx** - компонента силы по оси X
- **force * ny** - компонента силы по оси Y

Управление настройками(json):

```
{
    "window_width": 1000,
    "window_height": 700,
    "background_color": [255, 255, 255],
    "core_radius": 40,
    "yellow_ring_width": 15,
    "gray_ring_width": 10,
    "photon_speed": 3,
    "gravity_strength": 0.5,
    "max_photon_speed": 7,
    "animation_speed": 60,
    "slider_x": 50,
    "slider_y": 630,
    "slider_width": 300,
    "slider_height": 15,
    "min_mass": 1000,
    "max_mass": 10000,
    "initial_mass": 3000,
    "base_radius_threshold": 15
}
```

```
with open('config.json', 'r') as f:
    config = json.load(f)
```

Нужен для параметров

Требование:

-Создание окна приложения , в котором будут фотоны и черная дыра

Решение:

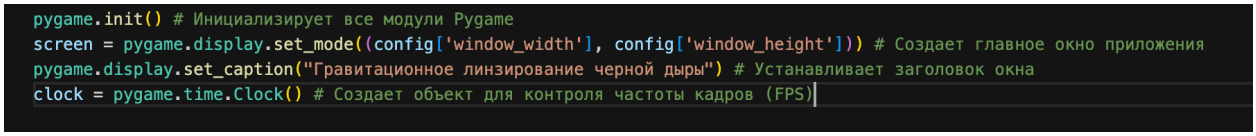
Код создает графическое окно Pygame с заданными размерами, заголовком и контролем FPS для отображения симуляции гравитационного линзирования.

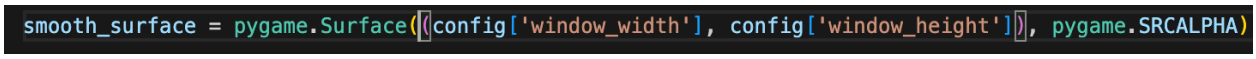
Фрагмент кода:

```
# pygame.init() # Инициализирует все модули Pygame
# screen = pygame.display.set_mode((config['window_width'],
config['window_height'])) # Создает главное окно приложения
# pygame.display.set_caption("Гравитационное линзирование черной
дыры") # Устанавливает заголовок окна
```

```
# clock = pygame.time.Clock() # Создает объект для контроля
частоты кадров (FPS)
# smooth_surface = pygame.Surface((config['window_width'],
config['window_height']), pygame.SRCALPHA)
```

Скриншот:





Требование:

- Создание черной дыры

Решение:

Пишем класс BlackHole, который рисует черное ядро , серое и желтое кольца ,расстояние между ними, а также серые параллельные линии ,в направлении которых сначала идут фотоны

Фрагмент кода:

```
# class BlackHole:
#     def __init__(self):
#         self.x = config['window_width'] // 2
#         self.y = config['window_height'] // 2 - 30
#         self.mass = config['initial_mass']
#         self.base_radius = config['core_radius']
#         self.radius = self.base_radius
#         self.gap_between_core_and_ring = 10  # расстояние
между ядром и желтым кольцом

#     def update_radius(self):
#         # Радиус ядра растёт пропорционально sqrt(массы)
#         scale = math.sqrt(self.mass / config['initial_mass'])
#         self.radius = int(self.base_radius * scale)

#     def draw_rings(self, surface):
#         # Серое кольцо — внешнее, с учётом зазора
#         pygame.draw.circle(surface, (100, 100, 100), (self.x,
self.y),
#                            self.radius +
self.gap_between_core_and_ring +
#                            config['yellow_ring_width'] * 2 +
config['gray_ring_width'],
#                            config['gray_ring_width'])
#         # Желтое кольцо — на расстоянии от ядра
```

```
#         pygame.draw.circle(surface, (255, 255, 0), (self.x,
self.y),
#                            self.radius +
self.gap_between_core_and_ring + config['yellow_ring_width'],
#                            config['yellow_ring_width'])

#     def draw_lines(self, surface):
#         # Центральная линия
#         pygame.draw.line(surface, (150, 150, 150),
#                          (0, self.y),
#                          (config['window_width'], self.y), 2)
#         # Верхняя линия
#         line_y = self.y - (self.radius +
self.gap_between_core_and_ring + config['yellow_ring_width'] *
1.5)
#         pygame.draw.line(surface, (150, 150, 150),
#                          (0, line_y),
#                          (config['window_width'], line_y), 2)

#     def draw_core(self, surface):
#         # Чёрное ядро
#         pygame.draw.circle(surface, (0, 0, 0), (self.x,
self.y), self.radius)
```

Скриншот:

```python
class BlackHole:
    def __init__(self):
        self.x = config['window_width'] // 2
        self.y = config['window_height'] // 2 - 30
        self.mass = config['initial_mass']
        self.base_radius = config['core_radius']
        self.radius = self.base_radius
        self.gap_between_core_and_ring = 10  # расстояние между ядром и желтым кольцом

    def update_radius(self):
        # Радиус ядра растёт пропорционально sqrt(массы)
        scale = math.sqrt(self.mass / config['initial_mass'])
        self.radius = int(self.base_radius * scale)

    def draw_rings(self, surface):
        # Серое кольцо — внешнее, с учётом зазора
        pygame.draw.circle(surface, (100, 100, 100), (self.x, self.y),
                           self.radius + self.gap_between_core_and_ring +
                           config['yellow_ring_width'] * 2 + config['gray_ring_width'],
                           config['gray_ring_width'])
        # Желтое кольцо — на расстоянии от ядра
        pygame.draw.circle(surface, (255, 255, 0), (self.x, self.y),
                           self.radius + self.gap_between_core_and_ring + config['yellow_ring_width'],
                           config['yellow_ring_width'])

    def draw_lines(self, surface):
        # Центральная линия
        pygame.draw.line(surface, (150, 150, 150),
                         (0, self.y),
                         (config['window_width'], self.y), 2)
        # Верхняя линия
        line_y = self.y - (self.radius + self.gap_between_core_and_ring + config['yellow_ring_width'] * 1.5)
        pygame.draw.line(surface, (150, 150, 150),
                         (0, line_y),
                         (config['window_width'], line_y), 2)

    def draw_core(self, surface):
        # Чёрное ядро
        pygame.draw.circle(surface, (0, 0, 0), (self.x, self.y), self.radius)
```

Требование:

- Параллельный пучок фотонов

Решение:

Класс Photon реализует модель фотона с гравитационным взаимодействием и визуализацией траектории

1. **__init__()** - создание фотона с заданной стартовой позицией
2. **reset()** - сброс в начальное состояние:
    - Возврат к стартовым координатам
    - Очистка и предзаполнение трейла для плавного старта
    - Установка начальной скорости (горизонтальное движение)
    - Активация фотона
3. **update()** - физическое обновление состояния(эта часть кода объяснена в следующем требовании)
4. **draw()** - визуализация(эта часть кода объяснена в следующем требовании)

Фрагмент кода:

```
#class Photon:
#     def __init__(self, start_x, start_y):
#         self.start_x = start_x
#         self.start_y = start_y
#         self.reset() # инициализирует все остальные параметры
фотона

#     def reset(self): # Сбрасывает фотон в начальное состояние
- возвращает к исходным параметрам.
#         self.x = self.start_x
#         self.y = self.start_y
#         self.trail = [] # Очистка трейла (следа)
#         self.trail_length = 80
#         self.speed_x = -config['photon_speed']
#         self.speed_y = 0
#         self.active = True # Пока active = True, фотон будет
обновляться и отрисовываться

#         # Заполняем трейл до старта чтобы Плавный старт - при
начале симуляции сразу видна траектория
#         for i in range(self.trail_length):
#             trail_x = self.x + (self.trail_length - i) *
config['photon_speed']
#             self.trail.append((trail_x, self.y))
```

Скриншот:

```python
class Photon:
    def __init__(self, start_x, start_y):
        self.start_x = start_x
        self.start_y = start_y
        self.reset() # инициализирует все остальные параметры фотона

    def reset(self): # Сбрасывает фотон в начальное состояние - возвращает к исходным параметрам.
        self.x = self.start_x
        self.y = self.start_y
        self.trail = [] # Очистка трейла (следа)
        self.trail_length = 80
        self.speed_x = -config['photon_speed']
        self.speed_y = 0
        self.active = True # Пока active = True, фотон будет обновляться и отрисовываться

        # Заполняем трейл до старта чтобы Плавный старт - при начале симуляции сразу видна траектория
        for i in range(self.trail_length):
            trail_x = self.x + (self.trail_length - i) * config['photon_speed']
            self.trail.append((trail_x, self.y))
```

Требование:

- фотоны искривляются под действием гравитации и входят в ядро

Решение:

Искривляются по формуле Ньютоновской гравитации в зависимости от массы ядра ,силы гравитации и радиуса

Фрагмент кода:

```
# def update(self, black_hole): # Обновляет состояние фотона на каждом кадре - движение, гравитация, траектория.
#     if not self.active:
#         return

#     # Добавляем позицию в трейл
#     self.trail.append((self.x, self.y)) # Добавление текущей позиции  в трейл
#     if len(self.trail) > self.trail_length: # Если точек больше 80, удаляется САМАЯ СТАРАЯ точка
#         self.trail.pop(0)

#     dx = black_hole.x - self.x   # разность по X между черной дырой и фотоном
#     dy = black_hole.y - self.y
#     distance = math.sqrt(dx*dx + dy*dy) # Расстояние = √(Δx² + Δy²) - теорема Пифагора

#     if distance > 0:
#         # Ньютоновская гравитация: F = G * M / r^2
#         force = (black_hole.mass * config['gravity_strength']) / (distance * distance)

#         # Нормализованный вектор к центру - Вектор длины 1, показывающий НАПРАВЛЕНИЕ к черной дыре
#         nx = dx / distance # X-компонента направления (от -1 до 1)
#         ny = dy / distance

#         # Обновляем скорость фотона - Ускорение = Сила × Направление
#         self.speed_x += force * nx
#         self.speed_y += force * ny

#         # Ограничиваем максимальную скорость
#         speed = math.sqrt(self.speed_x ** 2 + self.speed_y ** 2)
#         if speed > config['max_photon_speed']:
#             self.speed_x = self.speed_x / speed * config['max_photon_speed']
#             self.speed_y = self.speed_y / speed * config['max_photon_speed']

#     # Обновляем позицию
#     self.x += self.speed_x
#     self.y += self.speed_y
```

```
# def draw(self, surface):
#     if not self.active:
#         return

#     if len(self.trail) > 1:
#         # Рисуем сглаженную линию трека
#         pygame.draw.aalines(surface, (255, 0, 0), False,
self.trail, 1)  # Сглаженные линии

#     if len(self.trail) > 0:
#         # Рисуем сглаженный фотон
#         x, y = int(self.trail[-1][0]), int(self.trail[-1][1])
#         pygame.gfxdraw.aacircle(surface, x, y, 2, (255, 0, 0))
# Сглаженный контур
#         pygame.gfxdraw.filled_circle(surface, x, y, 2, (255, 0,
0))     # Заливка
```

Скриншот:

```
def update(self, black_hole): # Обновляет состояние фотона на каждом кадре — движение, гравитация, траектория.
    if not self.active:
        return

    # Добавляем позицию в трейл
    self.trail.append((self.x, self.y)) # Добавление текущей позиции  в трейл
    if len(self.trail) > self.trail_length: # Если точек больше 80, удаляется САМАЯ СТАРАЯ точка
        self.trail.pop(0)

    dx = black_hole.x - self.x   # разность по X между черной дырой и фотоном
    dy = black_hole.y - self.y
    distance = math.sqrt(dx*dx + dy*dy) # Расстояние = √(Δx² + Δy²) — теорема Пифагора

    if distance > 0:
        # Ньютоновская гравитация: F = G * M / r^2
        force = (black_hole.mass * config['gravity_strength']) / (distance * distance)

        # Нормализованный вектор к центру — Вектор длины 1, показывающий НАПРАВЛЕНИЕ к черной дыре
        nx = dx / distance # X-компонента направления (от -1 до 1)
        ny = dy / distance

        # Обновляем скорость фотона — Ускорение = Сила × Направление
        self.speed_x += force * nx
        self.speed_y += force * ny

        # Ограничиваем максимальную скорость
        speed = math.sqrt(self.speed_x ** 2 + self.speed_y ** 2)
        if speed > config['max_photon_speed']:
            self.speed_x = self.speed_x / speed * config['max_photon_speed']
            self.speed_y = self.speed_y / speed * config['max_photon_speed']

    # Обновляем позицию
    self.x += self.speed_x
    self.y += self.speed_y
```

```python
def draw(self, surface):
    if not self.active:
        return

    if len(self.trail) > 1:
        # Рисуем сглаженную линию трека
        pygame.draw.aalines(surface, (255, 0, 0), False, self.trail, 1)  # Сглаженные линии

    if len(self.trail) > 0:
        # Рисуем сглаженный фотон
        x, y = int(self.trail[-1][0]), int(self.trail[-1][1])
        pygame.gfxdraw.aacircle(surface, x, y, 2, (255, 0, 0))          # Сглаженный контур
        pygame.gfxdraw.filled_circle(surface, x, y, 2, (255, 0, 0))     # Заливка
```

0.0s

Требование:

- Слайдер для регулировки массы и вывод FPS

Решение:

Класс Slider реализует интерактивный элемент управления для динамического изменения массы черной дыры. def __init__ инициализирует геометрию слайдера и диапазон значений. def handle_event обрабатывает взаимодействие с мышью: захват ползунка, перетаскивание и расчет нового значения массы на основе позиции курсора. def draw отрисовывает визуальные компоненты: фоновую панель, заполненную область, подвижный ползунок и текстовую информацию с текущим значением массы и FPS

Фрагмент кода:

```
# class Slider:
#     def __init__(self, x, y, width, height, min_val, max_val,
initial):
#         self.rect = pygame.Rect(x, y, width, height)
#прямоугольная область слайдера с координатами (x,y) и размерами
(width,height)
#         self.min_val = min_val
#         self.max_val = max_val
#         self.value = initial #текущее значение массы
(начинается с initial)
#         self.dragging = False #флаг, отслеживающий
перетаскивается ли слайдер

#     def handle_event(self, event):
#         if event.type == pygame.MOUSEBUTTONDOWN and
self.rect.collidepoint(event.pos): #При нажатии кнопки мыши
проверяет, попадает ли курсор в область слайдера.
#             self.dragging = True # Если да – начинает
перетаскивание.
#         elif event.type == pygame.MOUSEBUTTONUP:
#             self.dragging = False #При отпускании кнопки мыши
прекращает перетаскивание.
#         elif event.type == pygame.MOUSEMOTION and
self.dragging:
```

```
#             relative_x = event.pos[0] - self.rect.x
#             self.value = self.min_val + (relative_x /
self.rect.width) * (self.max_val - self.min_val)
#             self.value = max(self.min_val, min(self.max_val,
self.value))
#             return True
#         return False

#     def draw(self, surface): #отрисовка слайдера
#         pygame.draw.rect(surface, (50, 50, 50), self.rect)
#         fill_width = int((self.value - self.min_val) /
(self.max_val - self.min_val) * self.rect.width) #вычисляет
ширину заполненной области пропорционально текущему значению
#         pygame.draw.rect(surface, (100, 100, 255),
(self.rect.x, self.rect.y, fill_width, self.rect.height))
#Отрисовка заполненной области

#         slider_x = self.rect.x + fill_width
#         pygame.draw.circle(surface, (200, 200, 255),
(slider_x, self.rect.centery), 8) #Рисует светло-синий ползунок
в позиции, соответствующей текущему значению.

#         font = pygame.font.SysFont(None, 24) #Создает и
отображает черный текст с текущим значением массы над слайдером.
#         text = font.render(f"Масса черной дыры:
{self.value:.0f}", True, (0, 0, 0))
#         surface.blit(text, (self.rect.x, self.rect.y - 25))

#         fps = clock.get_fps()
#         fps_text = font.render(f"FPS: {fps:.1f}", True, (0, 0,
0))
#         screen.blit(fps_text, (config['window_width'] - 100,
10))
```

Скриншот:

```python
class Slider:
    def __init__(self, x, y, width, height, min_val, max_val, initial):
        self.rect = pygame.Rect(x, y, width, height)
        self.min_val = min_val
        self.max_val = max_val
        self.value = initial
        self.dragging = False

    def handle_event(self, event):
        if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.pos):
            self.dragging = True
        elif event.type == pygame.MOUSEBUTTONUP:
            self.dragging = False
        elif event.type == pygame.MOUSEMOTION and self.dragging:
            relative_x = event.pos[0] - self.rect.x
            self.value = self.min_val + (relative_x / self.rect.width) * (self.max_val - self.min_val)
            self.value = max(self.min_val, min(self.max_val, self.value))
            return True
        return False

    def draw(self, surface):
        pygame.draw.rect(surface, (50, 50, 50), self.rect)
        fill_width = int((self.value - self.min_val) / (self.max_val - self.min_val) * self.rect.width)
        pygame.draw.rect(surface, (100, 100, 255), (self.rect.x, self.rect.y, fill_width, self.rect.height))
        slider_x = self.rect.x + fill_width
        pygame.draw.circle(surface, (200, 200, 255), (slider_x, self.rect.centery), 8)
        font = pygame.font.SysFont(None, 24)
        text = font.render(f"Масса черной дыры: {self.value:.0f}", True, (0, 0, 0))
        surface.blit(text, (self.rect.x, self.rect.y - 25))
        fps = clock.get_fps()
        fps_text = font.render(f"FPS: {fps:.1f}", True, (0, 0, 0))
        screen.blit(fps_text, (config['window_width'] - 100, 10))
```

Требование:

-Окончательное создание объектов и параллельности фотонов

Решение:

Созданы черная дыра , слайдер и параллельный пучок фотонов, который будет демонстрировать эффект гравитационного линзирования при приближении к черной дыре с помощью цикла, который вычисляет координату фотона .

Фрагмент кода:

```
# black_hole = BlackHole()
# slider = Slider(config['slider_x'], config['slider_y'],
#                 config['slider_width'], config['slider_height'],
#                 config['min_mass'], config['max_mass'],
config['initial_mass'])

# photons = []
# center_y = black_hole.y

# num_photons = 25
# spacing = 6            # расстояние между фотонами

# #Создание параллельного пучка фотонов
# for i in range(num_photons):
#     y = center_y - (i * spacing) # размещение фотонов ВЫШЕ
центра
```

```
#     start_x = config['window_width']
#     photons.append(Photon(start_x, y))
```

Скриншот:

```
#Создание черной дыры и слайдера
black_hole = BlackHole()
slider = Slider(config['slider_x'], config['slider_y'],
                config['slider_width'], config['slider_height'],
                config['min_mass'], config['max_mass'], config['initial_mass'])

photons = []
center_y = black_hole.y

num_photons = 25
spacing = 6           # расстояние между фотонами

#Создание параллельного пучка фотонов
for i in range(num_photons):
    y = center_y - (i * spacing) # размещение фотонов ВЫШЕ центра
    start_x = config['window_width']
    photons.append(Photon(start_x, y))
```

Требование:

-Запуск цикла инициализации черной дыры , фотонов, слайдера,FPS.

-при изменении массы ядро меняется еще и визуально

Решение:

Главный цикл программы обрабатывает пользовательский ввод, автоматически перезапускает фотоны каждые 5 секунд и обновляет физическую симуляцию. Он отрисовывает черную дыру с кольцами, траектории фотонов и интерфейс управления, поддерживая постоянную частоту кадров для плавной анимации гравитационного линзирования. Каждый кадр программа проверяет события мыши для слайдера, пересчитывает позиции фотонов под действием гравитации и визуализирует все объекты на экране, создавая непрерывную анимацию физического процесса.

Фрагмент кода:

```
# running = True
# restart_interval = 5000  # каждые 5 секунд
# last_restart_time = pygame.time.get_ticks()

# while running:
#     for event in pygame.event.get():
#         if event.type == pygame.QUIT: #проверяет событие
закрытия окна (крестик)
#             running = False
#         slider.handle_event(event)

#     # Автоматический перезапуск фотонов каждые 10 секунд
```

```
#     current_time = pygame.time.get_ticks()
#     if current_time - last_restart_time > restart_interval:
#ычисляет сколько прошло времени с последнего перезапуска
#         for photon in photons:
#             photon.reset()
#         last_restart_time = current_time

#     # Обновление массы и радиуса чёрной дыры
#     black_hole.mass = slider.value #обновляет массу черной
дыры из значения слайдера
#     black_hole.update_radius()

#     # Обновление фотонов
#     for photon in photons:
#         photon.update(black_hole)

#     smooth_surface.fill((0, 0, 0, 0))
#     black_hole.draw_rings(smooth_surface)
#     for photon in photons:
#         photon.draw(smooth_surface)
#     black_hole.draw_lines(smooth_surface)
#     black_hole.draw_core(smooth_surface)
#     # Отображение на экране
#     screen.fill(config['background_color'])
#     screen.blit(smooth_surface, (0, 0))
#     slider.draw(screen)

#     pygame.display.flip()
#     clock.tick(config['animation_speed']) #ограничение FPS (60
кадров/сек)

# pygame.quit()
```

Скриншот :

```
running = True
restart_interval = 5000  # каждые 5 секунд
last_restart_time = pygame.time.get_ticks()

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: #проверяет событие закрытия окна (крестик)
            running = False
        slider.handle_event(event)

    # Автоматический перезапуск фотонов каждые 10 секунд
    current_time = pygame.time.get_ticks()
    if current_time - last_restart_time > restart_interval: #ычисляет сколько прошло времени с последнего перезапуска
        for photon in photons:
            photon.reset()
        last_restart_time = current_time

    # Обновление массы и радиуса чёрной дыры
    black_hole.mass = slider.value #обновляет массу черной дыры из значения слайдера
    black_hole.update_radius()

    # Обновление фотонов
    for photon in photons:
        photon.update(black_hole)

    smooth_surface.fill((0, 0, 0, 0))
    black_hole.draw_rings(smooth_surface)
    for photon in photons:
        photon.draw(smooth_surface)
    black_hole.draw_lines(smooth_surface)
    black_hole.draw_core(smooth_surface)
    # Отображение на экране
    screen.fill(config['background_color'])
    screen.blit(smooth_surface, (0, 0))
    slider.draw(screen)

    pygame.display.flip()
    clock.tick(config['animation_speed']) #ограничение FPS (60 кадров/сек)

pygame.quit()
```

Полный код:

```
# import pygame
# import pygame.gfxdraw
# import json
# import math


# with open('config.json', 'r') as f:
#     config = json.load(f)

# pygame.init()
# screen = pygame.display.set_mode((config['window_width'],
config['window_height']))
# pygame.display.set_caption("Гравитационное линзирование черной
дыры")
# clock = pygame.time.Clock()


# font = pygame.font.SysFont(None, 24)

# # Поверхность для сглаживания
# smooth_surface = pygame.Surface((config['window_width'],
config['window_height']), pygame.SRCALPHA)
```

```
# class BlackHole:
#     def __init__(self):
#         self.x = config['window_width'] // 2
#         self.y = config['window_height'] // 2 - 30
#         self.mass = config['initial_mass']
#         self.base_radius = config['core_radius']
#         self.radius = self.base_radius
#         self.gap_between_core_and_ring = 10  # расстояние
между ядром и желтым кольцом

#     def update_radius(self):
#         # Радиус ядра растёт пропорционально sqrt(массы)
#         scale = math.sqrt(self.mass / config['initial_mass'])
#         self.radius = int(self.base_radius * scale)

#     def draw_rings(self, surface):
#         # Серое кольцо — внешнее, с учётом зазора
#         pygame.draw.circle(surface, (100, 100, 100), (self.x,
self.y),
#                            self.radius +
self.gap_between_core_and_ring +
#                            config['yellow_ring_width'] * 2 +
config['gray_ring_width'],
#                            config['gray_ring_width'])
#         # Желтое кольцо — на расстоянии от ядра
#         pygame.draw.circle(surface, (255, 255, 0), (self.x,
self.y),
#                            self.radius +
self.gap_between_core_and_ring + config['yellow_ring_width'],
#                            config['yellow_ring_width'])

#     def draw_lines(self, surface):
#         # Центральная линия
#         pygame.draw.line(surface, (150, 150, 150),
#                          (0, self.y),
#                          (config['window_width'], self.y), 2)
#         # Верхняя линия
#         line_y = self.y - (self.radius +
self.gap_between_core_and_ring + config['yellow_ring_width'] *
1.5)
#         pygame.draw.line(surface, (150, 150, 150),
#                          (0, line_y),
#                          (config['window_width'], line_y), 2)

#     def draw_core(self, surface):
#         # Чёрное ядро
#         pygame.draw.circle(surface, (0, 0, 0), (self.x,
self.y), self.radius)
```

```
# class Photon:
#     def __init__(self, start_x, start_y):
#         self.start_x = start_x
#         self.start_y = start_y
#         self.reset() # инициализирует все остальные параметры
фотона

#     def reset(self): # Сбрасывает фотон в начальное состояние
- возвращает к исходным параметрам.
#         self.x = self.start_x
#         self.y = self.start_y
#         self.trail = [] # Очистка трейла (следа)
#         self.trail_length = 80
#         self.speed_x = -config['photon_speed']
#         self.speed_y = 0
#         self.active = True # Пока active = True, фотон будет
обновляться и отрисовываться

#         # Заполняем трейл до старта чтобы Плавный старт - при
начале симуляции сразу видна траектория
#         for i in range(self.trail_length):
#             trail_x = self.x + (self.trail_length - i) *
config['photon_speed']
#             self.trail.append((trail_x, self.y))

#     def update(self, black_hole): # Обновляет состояние фотона
на каждом кадре - движение, гравитация, траектория.
#         if not self.active:
#             return

#         # Добавляем позицию в трейл
#         self.trail.append((self.x, self.y)) # Добавление
текущей позиции  в трейл
#         if len(self.trail) > self.trail_length: # Если точек
больше 80, удаляется САМАЯ СТАРАЯ точка
#             self.trail.pop(0)

#         dx = black_hole.x - self.x   # разность по X между
черной дырой и фотоном
#         dy = black_hole.y - self.y
#         distance = math.sqrt(dx*dx + dy*dy) # Расстояние =
√(Δx² + Δy²) - теорема Пифагора

#         if distance > 0:
#             # Ньютоновская гравитация: F = G * M / r^2
#             force = (black_hole.mass *
config['gravity_strength']) / (distance * distance)

#             # Нормализованный вектор к центру - Вектор длины
1, показывающий НАПРАВЛЕНИЕ к черной дыре
#             nx = dx / distance # X-компонента направления (от
-1 до 1)
```

```
#             ny = dy / distance

#             # Обновляем скорость фотона - Ускорение = Сила × Направление
#             self.speed_x += force * nx
#             self.speed_y += force * ny

#             # Ограничиваем максимальную скорость
#             speed = math.sqrt(self.speed_x ** 2 + self.speed_y ** 2)
#             if speed > config['max_photon_speed']:
#                 self.speed_x = self.speed_x / speed * config['max_photon_speed']
#                 self.speed_y = self.speed_y / speed * config['max_photon_speed']

#         # Обновляем позицию
#         self.x += self.speed_x
#         self.y += self.speed_y

#     def draw(self, surface):
#         if not self.active:
#             return

#         if len(self.trail) > 1:
#             # Рисуем сглаженную линию трека
#             pygame.draw.aalines(surface, (255, 0, 0), False, self.trail, 1)  # Сглаженные линии

#         if len(self.trail) > 0:
#             # Рисуем сглаженный фотон
#             x, y = int(self.trail[-1][0]), int(self.trail[-1][1])
#             pygame.gfxdraw.aacircle(surface, x, y, 2, (255, 0, 0))          # Сглаженный контур
#             pygame.gfxdraw.filled_circle(surface, x, y, 2, (255, 0, 0))     # Заливка



# class Slider:
#     def __init__(self, x, y, width, height, min_val, max_val, initial):
#         self.rect = pygame.Rect(x, y, width, height)
#         self.min_val = min_val
#         self.max_val = max_val
#         self.value = initial
#         self.dragging = False

#     def handle_event(self, event):
#         if event.type == pygame.MOUSEBUTTONDOWN and self.rect.collidepoint(event.pos):
```

```
#             self.dragging = True
#         elif event.type == pygame.MOUSEBUTTONUP:
#             self.dragging = False
#         elif event.type == pygame.MOUSEMOTION and
self.dragging:
#             relative_x = event.pos[0] - self.rect.x
#             self.value = self.min_val + (relative_x /
self.rect.width) * (self.max_val - self.min_val)
#             self.value = max(self.min_val, min(self.max_val,
self.value))
#             return True
#         return False

#     def draw(self, surface):
#         pygame.draw.rect(surface, (50, 50, 50), self.rect)
#         fill_width = int((self.value - self.min_val) /
(self.max_val - self.min_val) * self.rect.width)
#         pygame.draw.rect(surface, (100, 100, 255),
(self.rect.x, self.rect.y, fill_width, self.rect.height))
#         slider_x = self.rect.x + fill_width
#         pygame.draw.circle(surface, (200, 200, 255),
(slider_x, self.rect.centery), 8)
#         font = pygame.font.SysFont(None, 24)
#         text = font.render(f"Масса черной дыры:
{self.value:.0f}", True, (0, 0, 0))
#         surface.blit(text, (self.rect.x, self.rect.y - 25))
#         fps = clock.get_fps()
#         fps_text = font.render(f"FPS: {fps:.1f}", True, (0, 0,
0))
#         screen.blit(fps_text, (config['window_width'] - 100,
10))


# black_hole = BlackHole()
# slider = Slider(config['slider_x'], config['slider_y'],
#                 config['slider_width'], config['slider_height'],
#                 config['min_mass'], config['max_mass'],
config['initial_mass'])

# photons = []
# center_y = black_hole.y

# num_photons = 25       # было 12, стало 25
# spacing = 6            # расстояние между фотонами (уменьшено
для плотности)

# for i in range(num_photons):
#     y = center_y - (i * spacing)
#     start_x = config['window_width']
#     photons.append(Photon(start_x, y))
```

```
# running = True
# restart_interval = 5000  # каждые 5 секунд
# last_restart_time = pygame.time.get_ticks()

# while running:
#     for event in pygame.event.get():
#         if event.type == pygame.QUIT:
#             running = False
#         slider.handle_event(event)

#     # Автоматический перезапуск фотонов каждые 10 секунд
#     current_time = pygame.time.get_ticks()
#     if current_time - last_restart_time > restart_interval:
#         for photon in photons:
#             photon.reset()
#         last_restart_time = current_time

#     # Обновление массы и радиуса чёрной дыры
#     black_hole.mass = slider.value
#     black_hole.update_radius()

#     # Обновление фотонов
#     for photon in photons:
#         photon.update(black_hole)


#     smooth_surface.fill((0, 0, 0, 0))
#     black_hole.draw_rings(smooth_surface)
#     for photon in photons:
#         photon.draw(smooth_surface)
#     black_hole.draw_lines(smooth_surface)
#     black_hole.draw_core(smooth_surface)


#     # Отображение на экране
#     screen.fill(config['background_color'])
#     screen.blit(smooth_surface, (0, 0))
#     slider.draw(screen)

#     pygame.display.flip()
#     clock.tick(config['animation_speed'])

# pygame.quit()
```