Сложная анимация: "Фейерверки"

A. ТИТУЛЬНАЯ СТРАНИЦА

- **ФИО студентов:** Катиев Али, Абушинов Алексей
- **Номера студенческих билетов:** 245131, 242068
- **Название предмета и код:** Практикум по программированию
- **Номер лабораторной работы:** Лабораторная работа №3
- **Номер группы:** ИД24-1
- **Дата сдачи:** 25 ноября 2025 г.
- **Название назначенной задачи:** Сложная анимация: "Фейерверки"

B. ОПИСАНИЕ ЗАДАНИЯ

На экране случайным образом генерируются фейерверки. Как сами фейерверки, так и их частицы (искры) имеют **ограниченное время жизни**, после которого они должны исчезнуть.

Дополнительные требования:

1. Добавьте генерацию фейерверков **разных цветов** и **прозрачность**.
2. Поэкспериментируйте с **гравитацией** частиц.

C. ДОСТИГНУТАЯ СЛОЖНОСТЬ И РЕАЛИЗАЦИЯ

Выполненные основные требования

1. Реализована симуляция случайно появляющихся на экране фейерверков.
2. Частицы, образующиеся после взрыва, имеют ограниченное время жизни и плавно исчезают.
3. Программа спроектирована с использованием принципов **Объектно-Ориентированного Программирования (ООП)**.
4. Все ключевые параметры вынесены в конфигурационный файл `config.json`.

Выполненная дополнительная функциональность

1. Реализована генерация фейерверков ярких, случайных цветов.
2. Добавлен эффект **плавного затухания (прозрачности)** для частиц.
3. Реализовано **управление гравитацией** частиц.

Оригинальные расширения

1. **Интерактивное управление (HUD):** Реализована возможность **изменения основных параметров** симуляции (гравитация, частота запуска, разброс, количество искр) **в реальном времени** с помощью клавиш (`G/B`, `Q/E`, `R/F`, `PgUp/Dn`).
2. **Продвинутая Физика Частиц:**

    - Добавлено **сопротивление воздуха** (трение drag=0.95) для замедления частиц.
    - Реализован **режим ускоренного падения** после половины времени жизни для имитации быстрого «опадания» остывающих искр.
3. **Визуальный эффект "Шлейфа":** Каждая частица оставляет за собой **затухающий след** (`self.trail`), что делает анимацию более плавной и реалистичной.

D. ОБЪЯСНЕНИЕ ПРОЕКТИРОВАНИЯ ПРОГРАММЫ

Структура ООП: Классы и их обязанности

Класс	Обязанности
`Particle`	**Отдельная искра.** Управляет положением, вектором скорости, возрастом, прозрачностью. Отвечает за применение **гравитации, сопротивления воздуха** и отрисовку себя и своего **шлейфа**.
`Firework`	**Ракета и ее взрыв.** Управляет движением ракеты до взрыва. После взрыва генерирует **объекты `Particle`** и управляет их жизненным циклом (обновление, удаление).
`Simulation`	**Главный контроллер.** Управляет инициализацией Pygame, загрузкой настроек, основным циклом, **обработкой интерактивного ввода (HUD-управление)**, созданием фейерверков и общей отрисовкой.

Математическое исследование: Физика движения

1. **Проектильная траектория:** Положение частицы обновляется с учетом скорости ($x \leftarrow x+vx$, $y \leftarrow y+vy$).
2. **Гравитация и ускорение:** Вертикальная скорость изменяется под действием гравитации, заданной в конфиге:

   $$vy \leftarrow vy+gravity$$

3. **Сопротивление воздуха (Трение):** Для имитации замедления, скорости умножаются на коэффициент трения (`self.drag = 0.95`) на каждом кадре:

   $$vx \leftarrow vx \cdot 0.95$$

   $$vy \leftarrow vy \cdot 0.95$$

4. **Разлет частиц (Векторы):** В момент взрыва частицы разлетаются по случайному углу $angle \in [0, 2\pi]$:

   $$vx = \cos(angle) \cdot speed$$

   $$vy = \sin(angle) \cdot speed$$

⚙ Управление настройками: `config.json`

Файл `config.json` используется для хранения всех **ключевых параметров симуляции**, которые загружаются при старте. Это позволяет быстро менять характеристики фейерверков и физики без компиляции кода. В данной версии программы все эти параметры также **интерактивно изменяются во время работы** через функции `Simulation.handle_keys()`, обеспечивая динамическое тестирование.

Параметр	Описание	Значение по умолчанию
`"gravity"`	Сила гравитации, влияет на падение частиц.	`0.09`
`"particle_lifespan"`	Максимальное время жизни искры (в кадрах).	`75`
`"particle_spread"`	Множитель начальной скорости искр (разброс).	`0.75`
`"firework_spawn_rate"`	Вероятность появления новой ракеты.	`0.02`

E. ОСНОВНАЯ ЧАСТЬ (СТРУКТУРИРОВАННАЯ ПО ТРЕБОВАНИЯМ)

1. Требование: Генерация фейерверков и частиц с ограниченным временем жизни

- **Решение:** В классе `Simulation` с вероятностью `self.cfg["firework_spawn_rate"]` создается новый `Firework`. Каждая `Particle` имеет атрибут `self.age`. В `Particle.update()` возраст увеличивается до достижения `self.lifespan`. Удаление происходит в `Firework.update()` через list comprehension: `self.particles = [p for p in self.particles if p.alpha > 0]`.
- **Фрагмент кода (фрагмент `Firework.explode` и `Particle.update`):**

```
def explode(self):  1 usage
    # Создание множества искр при взрыве
    self.exploded = True
    color = random.choice(VIBRANT_COLORS)
    count = random.randint(self.cfg["particle_count_min"], self.cfg["particle_count_max"])
    power = 1.3                                    # дополнительная сила разлёта
    for _ in range(count):
        self.particles.append(Particle(
            self.x, self.y, color,
            self.gravity,
            self.cfg["particle_lifespan"],
            self.cfg["fade_speed"],
            self.cfg["particle_spread"],
            power
        ))
```

```python
def update(self):  4 usages (4 dynamic)
    # Обновление физики частицы каждый кадр
    self.age += 1
    if self.age > self.lifespan * 0.48:            # переключаемся в режим падения
        self.fall_mode = True

    if self.fall_mode:
        self.vx *= 0.965                           # торможение по горизонтали
        self.vy += self.gravity * 2.8              # ускоренное падение
        self.alpha = max(0, self.alpha - 2.8)      # быстрое угасание
    else:
        self.vy += self.gravity                    # обычное влияние гравитации
        self.vx *= self.drag                       # сопротивление воздуха
        self.vy *= self.drag
        self.alpha = max(0, self.alpha - self.fade_speed)

    self.x += self.vx                              # перемещение по оси X
    self.y += self.vy                              # перемещение по оси Y
    self.trail.append((self.x, self.y, self.alpha))
    if len(self.trail) > 28:                       # ограничиваем длину следа
        self.trail.pop(0)
```

2. Требование: Генерация разных цветов и прозрачность

- **Решение:** Цвет выбирается из палитры `VIBRANT_COLORS` в `Firework.__init__()`. Прозрачность (`self.alpha`) рассчитывается в `Particle.update()` и используется в `Particle.draw()`. Для корректной отрисовки прозрачных объектов в Pygame используется поверхность с альфа-каналом (`pygame.SRCALPHA`).
- **Фрагмент кода (фрагмент `Particle.draw`):**

```python
def draw(self, screen):  4 usages (4 dynamic)
    # Отрисовка следа частицы
    for i, (tx, ty, ta) in enumerate(self.trail):
        alpha = int(ta * (i / len(self.trail)) ** 1.3)
        if alpha <= 8: continue
        size = max(2, int(7 * (i / len(self.trail))))
        s = pygame.Surface((size * 2 + 6, size * 2 + 6), pygame.SRCALPHA)
        s.set_alpha(alpha)
        pygame.draw.circle(s, self.color, (size + 3, size + 3), size)
        screen.blit(s, (tx - size - 3, ty - size - 3))

    # Отрисовка самой искры с эффектом свечения
    if self.alpha > 10:
        s = pygame.Surface((28, 28), pygame.SRCALPHA)
        s.set_alpha(min(255, self.alpha))
        pygame.draw.circle(s, (*self.color, int(self.alpha * 0.5)), (14, 14), 12)  # полупрозрачный ореол
        pygame.draw.circle(s, self.color, (14, 14), 8)                             # основной круг
        bright = tuple(min(255, c + 120) for c in self.color)                      # яркая сердцевина
        pygame.draw.circle(s, bright, (14, 14), 4)
        screen.blit(s, (self.x - 14, self.y - 14))
```

3. Требование: Эксперименты с гравитацией и оригинальные расширения (Продвинутая Физика и HUD)

- **Решение:**
  - **Гравитация:** Значение `self.cfg["gravity"]` изменяется клавишами `G/B` через `Simulation.handle_keys()`.
  - **Продвинутая Физика:** Добавлено **сопротивление воздуха** (строки `self.vx *= self.drag`, `self.vy *= self.drag`) и **ускоренное падение** (`self.vy += self.gravity * 2.8`) после того, как частица преодолеет половину своего жизненного цикла.
- **Фрагмент кода (фрагмент `Simulation.handle_keys` и `Particle.update`):**

```python
def handle_keys(self):  1 usage
    # Обработка удержания клавиш для изменения параметров
    keys = pygame.key.get_pressed()
    changed = False

    if keys[pygame.K_g]: self.cfg["gravity"] += 0.02; changed = True
    if keys[pygame.K_b]: self.cfg["gravity"] -= 0.02; changed = True
    if keys[pygame.K_UP]:   self.cfg["firework_min_speed"] -= 0.5; self.cfg["firework_max_speed"] -= 0.5; changed = True
    if keys[pygame.K_DOWN]: self.cfg["firework_min_speed"] += 0.5; self.cfg["firework_max_speed"] += 0.5; changed = True
    if keys[pygame.K_q]: self.cfg["firework_spawn_rate"] += 0.002; changed = True
    if keys[pygame.K_e]: self.cfg["firework_spawn_rate"] -= 0.002; changed = True
    if keys[pygame.K_r]: self.cfg["particle_spread"] += 0.03; changed = True
    if keys[pygame.K_f]: self.cfg["particle_spread"] -= 0.03; changed = True
    if keys[pygame.K_PAGEUP]:   self.cfg["particle_count_min"] += 3; self.cfg["particle_count_max"] += 6; changed = True
    if keys[pygame.K_PAGEDOWN]: self.cfg["particle_count_min"] -= 3; self.cfg["particle_count_max"] -= 6; changed = True

    if changed:
        # Ограничение значений в допустимых пределах
        self.cfg["gravity"] = max(0.01, min(0.5, self.cfg["gravity"]))
        self.cfg["firework_spawn_rate"] = max(0.001, min(0.2, self.cfg["firework_spawn_rate"]))
        self.cfg["particle_spread"] = max(0.3, min(2.5, self.cfg["particle_spread"]))
        self.cfg["particle_count_min"] = max(20, self.cfg["particle_count_min"])
        self.cfg["particle_count_max"] = max(self.cfg["particle_count_min"] + 10, self.cfg["particle_count_max"])
        self.current_version += 1  # обновляем HUD
```

```python
def update(self):  4 usages (4 dynamic)
    # Обновление физики частицы каждый кадр
    self.age += 1
    if self.age > self.lifespan * 0.48:           # переключаемся в режим падения
        self.fall_mode = True

    if self.fall_mode:
        self.vx *= 0.965                          # торможение по горизонтали
        self.vy += self.gravity * 2.8             # ускоренное падение
        self.alpha = max(0, self.alpha - 2.8)     # быстрое угасание
    else:
        self.vy += self.gravity                   # обычное влияние гравитации
        self.vx *= self.drag                      # сопротивление воздуха
        self.vy *= self.drag
        self.alpha = max(0, self.alpha - self.fade_speed)

    self.x += self.vx                             # перемещение по оси X
    self.y += self.vy                             # перемещение по оси Y
    self.trail.append((self.x, self.y, self.alpha))
    if len(self.trail) > 28:                      # ограничиваем длину следа
        self.trail.pop(0)
```

F. ПОЛНЫЙ ИСХОДНЫЙ КОД|

```
import pygame
import json
import random
import math
# Список ярких цветов для взрывов фейерверков
VIBRANT_COLORS = [
   (255, 80, 80), (255, 180, 60), (255, 255, 120),
   (120, 255, 120), (120, 255, 255), (100, 180, 255),
   (160, 120, 255), (255, 120, 255), (255, 150, 200),
]
class Particle:
   # Класс отдельной искры после взрыва фейерверка
   def __init__(self, x, y, color, gravity, lifespan, fade_speed, spread, power=1.0):
      self.x = x + random.uniform(-10, 10)          # начальная позиция с
небольшим разбросом
      self.y = y + random.uniform(-10, 10)          # разброс по вертикали
      self.color = color                            # цвет частицы
      angle = random.uniform(0, 2 * math.pi)        # случайный угол вылета
      speed = random.uniform(9, 17) * spread * power # начальная скорость
      self.vx = math.cos(angle) * speed             # горизонтальная скорость
      self.vy = math.sin(angle) * speed             # вертикальная скорость
      self.gravity = gravity                        # сила гравитации
      self.lifespan = lifespan                      # максимальное время жизни
      self.age = 0                                  # текущий возраст частицы
      self.alpha = 255                              # начальная непрозрачность
      self.fade_speed = fade_speed                  # скорость угасания
      self.fall_mode = False                        # режим падения (после половины
жизни)
      self.drag = 0.95                              # сопротивление воздуха
      self.trail = []                               # координаты для отрисовки следа

   def update(self):
      # Обновление физики частицы каждый кадр
      self.age += 1
      if self.age > self.lifespan * 0.48:           # переключаемся в режим падения
         self.fall_mode = True

      if self.fall_mode:
         self.vx *= 0.965                           # торможение по горизонтали
         self.vy += self.gravity * 2.8              # ускоренное падение
         self.alpha = max(0, self.alpha - 2.8)      # быстрое угасание
      else:
         self.vy += self.gravity                    # обычное влияние гравитации
         self.vx *= self.drag                       # сопротивление воздуха
         self.vy *= self.drag
```

```
            self.alpha = max(0, self.alpha - self.fade_speed)

        self.x += self.vx                       # перемещение по оси X
        self.y += self.vy                       # перемещение по оси Y
        self.trail.append((self.x, self.y, self.alpha))
        if len(self.trail) > 28:                 # ограничиваем длину следа
            self.trail.pop(0)

    def draw(self, screen):
        # Отрисовка следа частицы
        for i, (tx, ty, ta) in enumerate(self.trail):
            alpha = int(ta * (i / len(self.trail)) ** 1.3)
            if alpha <= 8: continue
            size = max(2, int(7 * (i / len(self.trail))))
            s = pygame.Surface((size * 2 + 6, size * 2 + 6), pygame.SRCALPHA)
            s.set_alpha(alpha)
            pygame.draw.circle(s, self.color, (size + 3, size + 3), size)
            screen.blit(s, (tx - size - 3, ty - size - 3))

        # Отрисовка самой искры с эффектом свечения
        if self.alpha > 10:
            s = pygame.Surface((28, 28), pygame.SRCALPHA)
            s.set_alpha(min(255, self.alpha))
            pygame.draw.circle(s, (*self.color, int(self.alpha * 0.5)), (14, 14), 12)  #
полупрозрачный ореол
            pygame.draw.circle(s, self.color, (14, 14), 8)                    # основной
круг
            bright = tuple(min(255, c + 120) for c in self.color)              # яркая
сердцевина
            pygame.draw.circle(s, bright, (14, 14), 4)
            screen.blit(s, (self.x - 14, self.y - 14))

    def is_dead(self):
        #полностью исчезла ли частица
        return self.alpha <= 0


class Firework:
    # Класс одной ракеты: подъём → взрыв → искры
    def __init__(self, width, height, cfg):
        self.width = width
        self.height = height
        self.x = random.randint(200, width - 200)      # стартовая позиция по
горизонтали
        self.y = height                          # старт снизу экрана
        self.vy = random.uniform(cfg["firework_min_speed"],
```

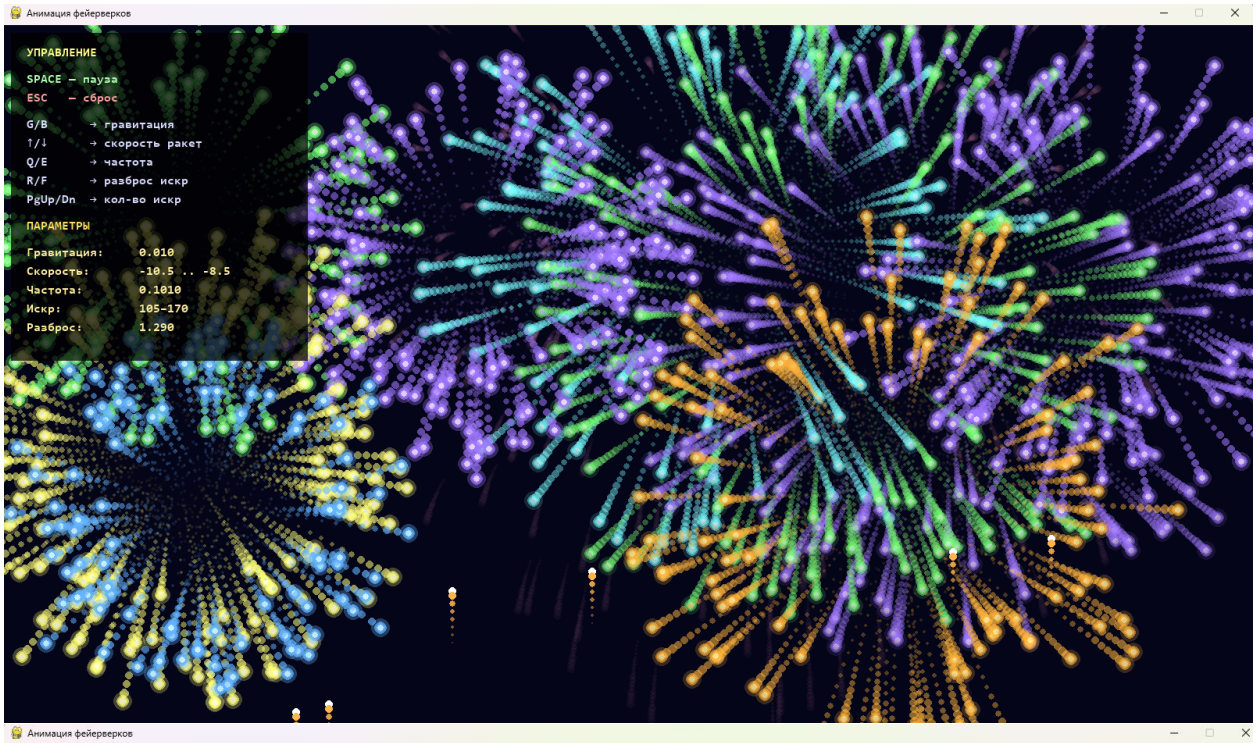
```
cfg["firework_max_speed"])  # начальная скорость вверх
        self.exploded = False                    # флаг взрыва
        self.gravity = cfg["gravity"]
        self.particles = []                      # список частиц после взрыва
        self.cfg = cfg                           # текущие настройки симуляции

    def update(self):
        # Обновление состояния ракеты или её осколков
        if not self.exploded:
            self.y += self.vy
            self.vy += 0.18                      # замедление при подъёме
            if self.vy >= -1.5:                  # взрыв
                self.explode()
        else:
            for p in self.particles[:]:
                p.update()
                if p.is_dead():
                    self.particles.remove(p)

    def explode(self):
        # Создание множества искр при взрыве
        self.exploded = True
        color = random.choice(VIBRANT_COLORS)
        count = random.randint(self.cfg["particle_count_min"],
self.cfg["particle_count_max"])
        power = 1.3                              # дополнительная сила разлёта
        for _ in range(count):
            self.particles.append(Particle(
                self.x, self.y, color,
                self.gravity,
                self.cfg["particle_lifespan"],
                self.cfg["fade_speed"],
                self.cfg["particle_spread"],
                power
            ))

    def draw(self, screen):
        # Отрисовка ракеты до взрыва или искр после
        if not self.exploded:
            pygame.draw.circle(screen, (255, 255, 255), (int(self.x), int(self.y)), 5)  #
белая точка - ракета
            for i in range(8):                   # след от ракеты
                py = self.y + i * 10
                if py > self.height: continue
                alpha = int(255 * (1 - i / 8))
                s = pygame.Surface((12, 12), pygame.SRCALPHA)
```

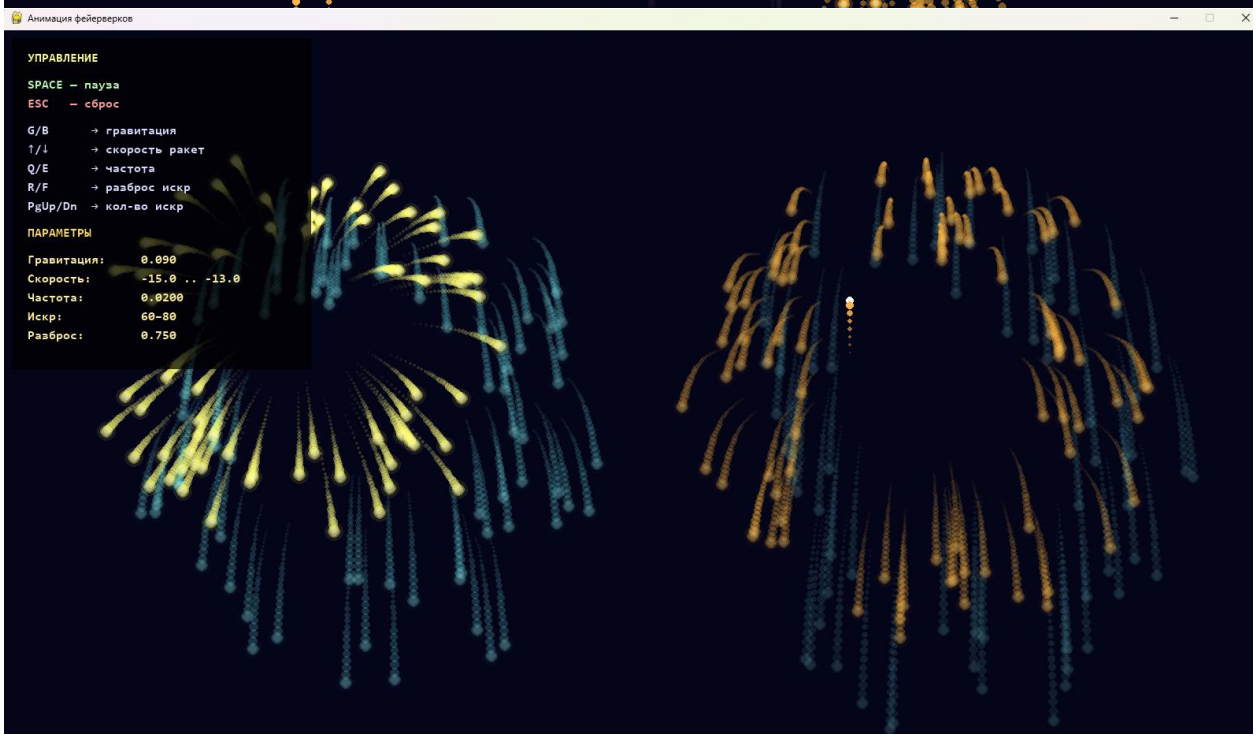
```
            s.set_alpha(alpha)
            pygame.draw.circle(s, (255, 180, 80), (6, 6), 5 - i * 0.6)
            screen.blit(s, (self.x - 6, py))
        else:
            for p in self.particles:
                p.draw(screen)

    def is_done(self):
        # можно ли удалить фейерверк из списка
        return self.exploded and len(self.particles) == 0


class Simulation:
    # Основной класс - управляет всей симуляцией
    def __init__(self):
        # Параметры по умолчанию
        self.default_cfg = {
            "gravity": 0.09,
            "particle_lifespan": 75,
            "particle_count_min": 60,
            "particle_count_max": 80,
            "particle_spread": 0.75,
            "fade_speed": 4,
            "firework_spawn_rate": 0.02,
            "firework_min_speed": -15,
            "firework_max_speed": -13
        }
        self.cfg = self.default_cfg.copy()

        # Загрузка пользовательских настроек из файла
        try:
            with open("config.json", "r", encoding="utf-8") as f:
                content = f.read().strip()
                if content:
                    user_cfg = json.loads(content)
                    self.cfg.update(user_cfg)
        except:
            pass  # если файл отсутствует или повреждён - используем дефолт

        pygame.init()
        self.screen = pygame.display.set_mode((1600, 900))  # окно
        pygame.display.set_caption("Анимация фейерверков")
        self.clock = pygame.time.Clock()
        self.fireworks = []                    # активные фейерверки
        self.running = True
        self.paused = False
```

```
        self.font = pygame.font.SysFont("consolas", 16, bold=True)  # шрифт для
HUD
        self.hud_cache = None                    # кэшированная панель управления
        self.hud_version = 0
        self.current_version = 0                 # счётчик изменений параметров

    def reset_to_default(self):
        # Полный сброс настроек и очистка экрана
        self.cfg = self.default_cfg.copy()
        self.fireworks.clear()
        self.current_version += 1

    def handle_keys(self):
        # Обработка удержания клавиш для изменения параметров
        keys = pygame.key.get_pressed()
        changed = False

        if keys[pygame.K_g]: self.cfg["gravity"] += 0.02; changed = True
        if keys[pygame.K_b]: self.cfg["gravity"] -= 0.02; changed = True
        if keys[pygame.K_UP]:   self.cfg["firework_min_speed"] -= 0.5;
self.cfg["firework_max_speed"] -= 0.5; changed = True
        if keys[pygame.K_DOWN]: self.cfg["firework_min_speed"] += 0.5;
self.cfg["firework_max_speed"] += 0.5; changed = True
        if keys[pygame.K_q]: self.cfg["firework_spawn_rate"] += 0.002; changed =
True
        if keys[pygame.K_e]: self.cfg["firework_spawn_rate"] -= 0.002; changed =
True
        if keys[pygame.K_r]: self.cfg["particle_spread"] += 0.03; changed = True
        if keys[pygame.K_f]: self.cfg["particle_spread"] -= 0.03; changed = True
        if keys[pygame.K_PAGEUP]:   self.cfg["particle_count_min"] += 3;
self.cfg["particle_count_max"] += 6; changed = True
        if keys[pygame.K_PAGEDOWN]: self.cfg["particle_count_min"] -= 3;
self.cfg["particle_count_max"] -= 6; changed = True

        if changed:
            # Ограничение значений в допустимых пределах
            self.cfg["gravity"] = max(0.01, min(0.5, self.cfg["gravity"]))
            self.cfg["firework_spawn_rate"] = max(0.001, min(0.2,
self.cfg["firework_spawn_rate"]))
            self.cfg["particle_spread"] = max(0.3, min(2.5,
self.cfg["particle_spread"]))
            self.cfg["particle_count_min"] = max(20, self.cfg["particle_count_min"])
            self.cfg["particle_count_max"] = max(self.cfg["particle_count_min"] +
10, self.cfg["particle_count_max"])
            self.current_version += 1  # обновляем HUD
```

```
    def draw_hud(self):
        # Отрисовка панели управления и текущих параметров
        if self.hud_version != self.current_version or self.hud_cache is None:
            self.hud_version = self.current_version
            overlay = pygame.Surface((380, 420), pygame.SRCALPHA)
            overlay.fill((0, 0, 0, 185))  # полупрозрачный тёмный фон

            lines = [
                ("УПРАВЛЕНИЕ", (255, 255, 140)),
                ("", None),
                ("SPACE — пауза", (180, 255, 180)),
                ("ESC   — сброс", (255, 160, 160)),
                ("", None),
                ("G/B     → гравитация", (220, 220, 255)),
                ("↑/↓     → скорость ракет", (220, 220, 255)),
                ("Q/E     → частота", (220, 220, 255)),
                ("R/F     → разброс искр", (220, 220, 255)),
                ("PgUp/Dn  → кол-во искр", (220, 220, 255)),
                ("", None),
                ("ПАРАМЕТРЫ", (255, 220, 100)),
                ("", None),
                (f"Гравитация:    {self.cfg['gravity']:.3f}", (255, 240, 160)),
                (f"Скорость:      {self.cfg['firework_min_speed']:.1f} ..
{self.cfg['firework_max_speed']:.1f}", (255, 240, 160)),
                (f"Частота:       {self.cfg['firework_spawn_rate']:.4f}", (255, 240, 160)),
                (f"Искр:          {self.cfg['particle_count_min']}–
{self.cfg['particle_count_max']}", (255, 240, 160)),
                (f"Разброс:       {self.cfg['particle_spread']:.3f}", (255, 240, 160)),
            ]

            y = 18
            for text, color in lines:
                if not text:
                    y += 10
                    continue
                surf = self.font.render(text, True, color)
                overlay.blit(surf, (20, y))
                y += 24

            if self.paused:
                pause = self.font.render("ПАУЗА", True, (255, 80, 80))
                overlay.blit(pause, (20, 380))

            self.hud_cache = overlay

        self.screen.blit(self.hud_cache, (10, 10))  # выводим панель на экран
```

```
    def run(self):
        # Главный цикл программы
        try:
            while self.running:
                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        self.running = False
                    if event.type == pygame.KEYDOWN:
                        if event.key == pygame.K_SPACE:
                            self.paused = not self.paused
                            self.current_version += 1
                        if event.key == pygame.K_ESCAPE:
                            self.reset_to_default()

                if not self.paused:
                    self.handle_keys()
                    if random.random() < self.cfg["firework_spawn_rate"]:
                        self.fireworks.append(Firework(1600, 900, self.cfg))

                    for fw in self.fireworks[:]:
                        fw.update()
                        if fw.is_done():
                            self.fireworks.remove(fw)

                self.screen.fill((5, 5, 25))          # заливка ночным небом
                for fw in self.fireworks:
                    fw.draw(self.screen)

                self.draw_hud()
                pygame.display.flip()                 # обновление экрана
                self.clock.tick(60)                   # ограничение 60 FPS

        finally:
            pygame.quit()                         # корректное завершение
            print("   мы счітаем это максімальный балл".center(56))  # финальное
сообщение

if __name__ == "__main__":
    Simulation().run()
```


Анимация фейерверков
УПРАВЛЕНИЕ
SPACE — пауза
ESC — сброс
G/B → гравитация
↑/↓ → скорость ракет
Q/E → частота
R/F → разброс искр
PgUp/Dn → кол-во искр
ПАРАМЕТРЫ
Гравитация: 0.010
Скорость: -10.5 .. -8.5
Частота: 0.1010
Искр: 105–170
Разброс: 1.290


Анимация фейерверков
УПРАВЛЕНИЕ
SPACE — пауза
ESC — сброс
G/B → гравитация
↑/↓ → скорость ракет
Q/E → частота
R/F → разброс искр
PgUp/Dn → кол-во искр
ПАРАМЕТРЫ
Гравитация: 0.090
Скорость: -15.0 .. -13.0
Частота: 0.0200
Искр: 60–80
Разброс: 0.750