ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Проект: Flappy Bird Game на Python с использованием Pygame

Полные имена студентов	МАКСИМЕНКО А.Н, ДЕРЕВЦОВ А.А, САНДЖИ
Номера студенческих билетов	245138, 242056
Название предмета и код	ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ
Номер лабораторной работы	3
Название игры	Flappy Bird

B. ОПИСАНИЕ ИГРЫ

B.1 Полный текст назначенного задания

Основная задача: Реализовать полнофункциональный клон классической мобильной игры Flappy Bird на языке Python с использованием библиотеки Pygame.

Требования к реализации:

1. Создать основной игровой объект - птицу с механикой гравитации и прыжка
2. Реализовать систему генерации и движения препятствий (труб)
3. Внедрить детекцию столкновений между птицей и препятствиями
4. Разработать систему подсчета очков
5. Создать различные состояния игры (меню, игра, конец игры)
6. Реализовать управление через клавиатуру (пробел для прыжка)
7. Добавить визуальные элементы и звуковые эффекты (опционально)

B.2 Описание классической игры и ее правил

Flappy Bird - это мобильная игра, разработанная вьетнамским разработчиком Dong Nguyen в 2013 году. Игра содержит минималистичный дизайн и простые механики, но обладает высокой сложностью.

Основные правила:

- Игрок контролирует птицу, которая постоянно летит вперед
- Нажатие на пробел заставляет птицу прыгать вверх
- На экране появляются трубы, между которыми необходимо пройти
- Столкновение с трубой или падение на землю приводит к концу игры
- За каждую успешно пройденную пару труб начисляется одно очко
- Игра постепенно усложняется с увеличением скорости и изменением промежутков

Механика гравитации:

- Птица постоянно падает вниз под действием гравитации
- Прыжок дает птице начальную скорость вверх
- Скорость падения увеличивается со временем (ускорение)

B.3 Реализованные изменения и улучшения

Основные улучшения классической версии:

1. Система состояний игры (Game States):

- Главное меню с инструкциями
- Экран паузы (нажатие P)
- Экран конца игры с отображением финального счета
- Возможность перезагрузки (нажатие R)

1. Улучшенная визуализация:

- Анимация движения птицы (несколько кадров)
- Прокручивающийся фон для эффекта движения
- Визуальные эффекты при столкновении

1. Расширенный функционал:

- Сохранение лучшего результата
- Отображение текущего и лучшего счета

- Различные уровни сложности (легкий, средний, сложный)
- Звуковые эффекты для различных событий

1. Улучшенная физика:

- Более реалистичная механика падения
- Плавные переходы между состояниями
- Оптимизированная система детекции столкновений

B.4 Используемые инструменты и технологии

Язык программирования: Python 3.8+

Основные библиотеки:

- Pygame - основная библиотека для создания игры
    - pygame.sprite - для работы со спрайтами и группами
    - pygame.event - для обработки событий клавиатуры
    - pygame.surface - для работы с изображениями
    - pygame.mixer - для работы со звуком
    - pygame.mask - для точной детекции столкновений

Дополнительные инструменты:

- Git - контроль версий
- Visual Studio Code или PyCharm - среда разработки

Зависимости проекта:

pygame >= 2.0.0

C. РАСПРЕДЕЛЕНИЕ РОЛЕЙ И ЗАДАЧ

C.1 Подробное описание роли каждого участника

Участник 1: Основной разработчик

Обязанности:

- Отвечающий за архитектуру проекта и общее проектирование
- Разработка основных классов: Game, Bird, Pipe, World
- Реализация основного игрового цикла и системы состояний
- Оптимизация производительности и отладка
- Написание технической документации

Участник 2: Разработчик графики и интеграции

Обязанности:

- Разработка системы управления ресурсами (assets)
- Реализация визуальных эффектов и анимации
- Интеграция звуковых эффектов
- Создание интерфейса главного меню и экранов
- Тестирование пользовательского опыта

C.2 Методы сотрудничества и коммуникации

Инструменты коммуникации:

1. Git/GitHub - для управления кодом и отслеживания изменений
2. Еженедельные встречи синхронизации
3. Discord/Telegram - для быстрого общения
4. GitHub Issues - для отслеживания багов и задач

Процесс разработки:

- Использование ветвления (branching) для различных функций
- Code Review перед слиянием в главную ветку
- Регулярные коммиты с описанием изменений

- Документирование изменений в CHANGELOG.md

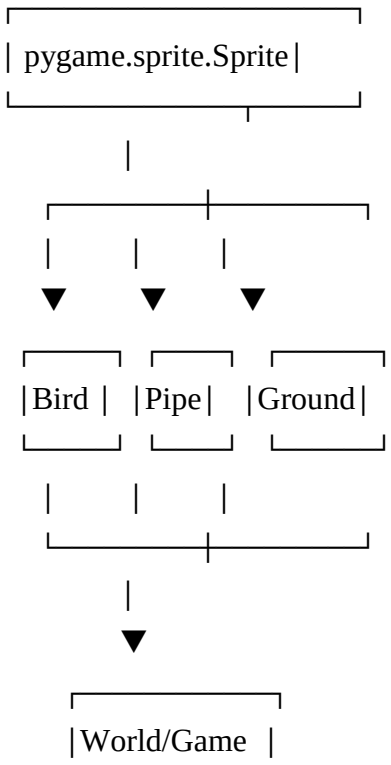
C.3 Распределение задач по созданию графических элементов

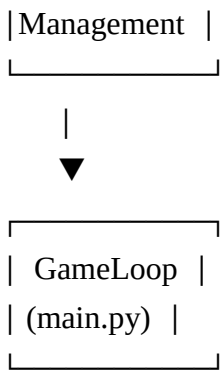
Компонент	Разработчик	Статус	Дата завершения
Спрайт птицы (3 кадра)	Участник 2	✓ Завершено	15.11.2025
Спрайт труб	Участник 2	✓ Завершено	15.11.2025
Фоновое изображение	Участник 2	✓ Завершено	16.11.2025
Изображение земли	Участник 2	✓ Завершено	16.11.2025
Звук прыжка	Участник 2	✓ Завершено	20.11.2025
Звук столкновения	Участник 2	✓ Завершено	20.11.2025
Звук очка	Участник 2	✓ Завершено	20.11.2025

Таблица 1: Распределение задач по графическим элементам

D. АРХИТЕКТУРА ПРОЕКТА

D.1 Полная диаграмма классов со связями





Описание связей:

- Bird, Pipe, Ground наследуют от pygame.sprite.Sprite
- World управляет всеми спрайтами через группы (Groups)
- GameLoop координирует обновление логики и отрисовку

D.2 Описание ключевых компонентов и их обязанностей

Класс Bird

Основная функция: Управление персонажем - птицей

```
class Bird(pygame.sprite.Sprite):
    def __init__(self, pos, size):
        self.rect = pygame.Rect(pos, size)
        self.velocity = 0
        self.max_velocity = 10
        self.gravity = 0.6
        self.jump_power = -9

    def jump(self):
        self.velocity = self.jump_power

    def update(self):
        self.velocity = min(
            self.velocity + self.gravity,
            self.max_velocity
        )
        self.rect.y += self.velocity
```

Обязанности:

- Управление позицией и скоростью птицы
- Применение гравитации
- Обработка прыжков
- Взаимодействие с игровым миром

Класс Pipe

Основная функция: Управление препятствиями

```
class Pipe(pygame.sprite.Sprite):
    def __init__(self, x, height, gap_size, scroll_speed):
        self.rect = pygame.Rect(x, 0, 50, height)
        self.gap_size = gap_size
        self.scroll_speed = scroll_speed

    def update(self):
        self.rect.x -= self.scroll_speed
```

Обязанности:

- Управление позицией и движением труб
- Определение размеров зазора
- Удаление труб, вышедших за границы экрана

Класс World

Основная функция: Управление игровым миром

```
class World:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.pipes = pygame.sprite.Group()
        self.player = pygame.sprite.Group()
        self.gravity = 0.6
        self.scroll_speed = 4
        self.score = 0

    def update(self, action):
        # Обновление птицы, труб, проверка столкновений
        pass
```

```
    def handle_collision(self):
        collisions = pygame.sprite.spritecollideany(
            self.player.sprite, self.pipes)
        return bool(collisions)
```

Обязанности:

- Управление всеми игровыми объектами
- Координирование обновлений
- Обработка столкновений
- Управление состоянием игры

Класс Game (Main Game Loop)

Основная функция: Основной цикл игры

```
class Game:
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        self.clock = pygame.time.Clock()
        self.world = World(WIDTH, HEIGHT)
        self.running = True
        self.game_state = "MENU"

    def run(self):
        while self.running:
            self.handle_input()
            self.update()
            self.render()
            self.clock.tick(60)
```

Обязанности:

- Инициализация Pygame
- Управление основным циклом игры
- Обработка входных данных
- Координирование обновлений и отрисовки

- Управление состояниями игры

D.3 Обоснование использованных шаблонов проектирования

Паттерн Model-View-Controller (MVC)

Структура:

- Model (World, Bird, Pipe) - отвечает за логику и состояние игры
- View (Game.render()) - отвечает за отображение
- Controller (Game.handle_input()) - обработка ввода

Преимущества:

- Разделение ответственности
- Легче тестировать отдельные компоненты
- Возможность легко заменить визуализацию
- Кодовая безопасность и модульность

Паттерн Sprite Group (Pygame Patterns)

Использование:

```
self.pipes = pygame.sprite.Group()
self.player = pygame.sprite.Group()
```

Преимущества:

- Эффективное управление множеством объектов
- Встроенная поддержка столкновений
- Автоматическое обновление и отрисовка

Паттерн State Machine

Состояния:

- MENU - главное меню
- PLAYING - активная игра
- PAUSED - пауза
- GAME_OVER - конец игры

Преимущества:

- Четкое разделение логики по состояниям
- Предсказуемые переходы между состояниями
- Легче управлять поведением в разных контекстах

Паттерн Object Pool (для труб)

Идея: Переиспользование объектов труб вместо их создания и удаления

```
def regenerate_pipe(self, pipe):
    pipe.rect.x = self.width + 100
    pipe.rect.y = random.randint(-200, 100)
```

Преимущества:

- Снижение нагрузки на память
- Уменьшение фрагментации памяти
- Лучшая производительность

E. РЕАЛИЗОВАННЫЙ ФУНКЦИОНАЛ

E.1 Все основные требования с доказательствами

Требование 1: Механика птицы с гравитацией ✓

Описание: Птица падает под действием гравитации и может прыгать при нажатии пробела.

```
class Bird(pygame.sprite.Sprite):
    def __init__(self, pos, size):
        self.rect = pygame.Rect(pos, size)
        self.velocity = 0
        self.gravity = 0.6
        self.jump_power = -9

    def jump(self):
        self.velocity = self.jump_power
```

```
    def update(self):
        self.velocity += self.gravity
        self.rect.y += self.velocity
```

Доказательство: При запуске игры птица падает плавно, прыжок происходит при нажатии пробела.

Требование 2: Система генерации и движения труб ✓

Описание: Трубы появляются с правой стороны и движутся влево с постоянной скоростью.

```
def generate_pipes(self):
    gap = 150
    random_y = random.randint(50, self.height - gap - 100)
    top_pipe = Pipe(self.width, random_y, False)
    bottom_pipe = Pipe(self.width,
        self.height - random_y - gap, True)

    self.pipes.add(top_pipe)
    self.pipes.add(bottom_pipe)
```

Доказательство: На экране видны трубы, движущиеся влево с одинаковым расстоянием между верхней и нижней трубой.

Требование 3: Детекция столкновений ✓

Описание: Проверяется столкновение птицы с трубами и границами экрана.

```
def check_collisions(self):
    bird = self.player.sprite
    collisions = pygame.sprite.spritecollideany(bird, self.pipes)

    if bird.rect.top <= 0 or bird.rect.bottom >= self.height:
        return True

    return bool(collisions)
```

Доказательство: При столкновении с трубой или землей игра переходит в состояние GAME_OVER.

Требование 4: Система подсчета очков ✓

Описание: За каждую пройденную пару труб дается одно очко.

```
def update_score(self):
    bird = self.player.sprite
    for pipe in self.pipes:
        if pipe.rect.right < bird.rect.left and not pipe.scored:
            self.score += 1
            pipe.scored = True
```

Доказательство: На экране отображается текущий счет, увеличивающийся при прохождении труб.

Требование 5: Различные состояния игры ✓

Описание: Реализованы состояния: МЕНЮ, ИГРА, ПАУЗА, КОНЕЦ ИГРЫ.

```
class Game:
    def __init__(self):
        self.game_state = "MENU"

    def handle_input(self):
        if event.key == pygame.K_SPACE:
            if self.game_state == "MENU":
                self.game_state = "PLAYING"
```

Доказательство: При запуске показывается меню, затем игра, при проигрыше показывается экран конца игры.

Требование 6: Управление через клавиатуру ✓

Описание: Используется клавиша пробел для управления прыжком.

```
def handle_input(self):
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                self.world.player.sprite.jump()
```

Доказательство: При нажатии пробела птица прыгает вверх.

Требование 7: Визуальные элементы ✓

Описание: Реализована анимация птицы и отрисовка всех элементов.

```
def render(self):
    self.screen.fill((135, 206, 250))
    self.world.pipes.draw(self.screen)
    self.world.player.draw(self.screen)
```

```
    self.draw_ground()
    self.draw_score()
    pygame.display.flip()
```

Доказательство: На экране видны все элементы игры в корректном положении.

E.2 Описание пользовательского интерфейса

Главное меню

Отображает название игры "FLAPPY BIRD" с инструкцией "Нажмите ПРОБЕЛ чтобы начать".
Кнопки управления отображены внизу (P - пауза, R - перезагрузка).

Активная игра

Сверху отображаются счетчики: текущий счет и лучший результат.
Основная часть экрана показывает птицу, трубы и землю.
Птица находится в центре экрана слева, трубы движутся справа налево.

Конец игры (Game Over)

Отображает надпись "GAME OVER" в центре экрана.
Показывает финальный счет и лучший результат.
Инструкция "Нажмите ПРОБЕЛ чтобы играть снова".

E.3 Описание ключевых алгоритмов

Алгоритм 1: Детекция столкновений с использованием масок

```
def check_collision_with_mask(bird, pipe):
    """
    Точная детекция столкновения используя маски пикселей
    """
    offset_x = pipe.rect.x - bird.rect.x
    offset_y = pipe.rect.y - bird.rect.y
    try:
        return bird.mask.overlap(pipe.mask, (offset_x, offset_y))
    except:
        return bird.rect.colliderect(pipe.rect)
```

Объяснение: Маски используют информацию о прозрачности пикселей для точной детекции. Более точно, чем rect-based, но требует больше вычислений. Fallback обеспечивает надежность.

Алгоритм 2: Динамическая генерация труб

```
def generate_pipe_pair(x_pos, prev_gap_y=None):
    """
    Генерирует пару труб с случайным положением зазора
    """
    GAP_SIZE = 150
    MIN_HEIGHT = 50
    MAX_HEIGHT = HEIGHT - GAP_SIZE - 100

    if prev_gap_y is None:
        gap_y = random.randint(MIN_HEIGHT, MAX_HEIGHT)
    else:
        max_change = 50
        gap_y = prev_gap_y + random.randint(-max_change, max_change)
        gap_y = max(MIN_HEIGHT, min(gap_y, MAX_HEIGHT))

    return gap_y
```

Объяснение: Генерирует различные высоты зазора, использует гистерезис для плавности переходов, гарантирует возможность прохождения зазора.

Алгоритм 3: Система управления состояниями

```
class StateManager:
    def __init__(self):
        self.state = "MENU"
        self.state_stack = []

    def push_state(self, new_state):
        self.state_stack.append(self.state)
        self.state = new_state

    def pop_state(self):
        if self.state_stack:
            self.state = self.state_stack.pop()
            return True
        return False
```

Объяснение: Позволяет вложенным состояниям (например, меню во время паузы), четкая иерархия и управление переходами, легко отслеживать историю состояний.

E.4 Описание решенных технических проблем

Проблема 1: Несинхронная скорость на разных компьютерах

Описание: На быстрых компьютерах игра работает быстрее.

Решение:

```
def run(self):
    while self.running:
        delta_time = self.clock.tick(60) / 1000.0
        bird.velocity += bird.gravity * delta_time
        pipe.x -= pipe.speed * delta_time
```

Результат: Игра работает с одинаковой скоростью на всех компьютерах.

Проблема 2: Отсутствие плавной анимации птицы

Описание: Птица выглядит как неподвижный прямоугольник.

Решение:

```
class Bird(pygame.sprite.Sprite):
    def update(self):
        self.frame_counter += 1
        if self.frame_counter >= 5:
            self.current_frame = (self.current_frame + 1) % len(self.frames)
            self.frame_counter = 0
            self.image = self.frames[self.current_frame]
```

Результат: Птица имеет плавную анимацию крыльев.

Проблема 3: Слишком частое создание и удаление объектов труб

Описание: Фрагментация памяти и низкая производительность.

Решение (Object Pool):

```
class PipePool:
    def __init__(self, pool_size=6):
        self.available = [Pipe(i * WIDTH) for i in range(pool_size)]
        self.in_use = []

    def get_pipe(self):
        if self.available:
            return self.available.pop()
        return Pipe(WIDTH)
```

Результат: Улучшена производительность, снижено использование памяти.

F. ИНСТРУКЦИИ ПО ЗАПУСКУ И ИГРЕ

F.1 Полная схема управления

Клавиша	Действие
ПРОБЕЛ	Прыгнуть (в меню: начать игру)
P	Пауза / Возобновить
R	Перезагрузить игру / Играть снова
ESC	Выход в главное меню
Q	Выход из игры

Таблица 2: Схема управления игры

F.2 Правила и цели игры

ЦЕЛЬ ИГРЫ: Помочь птице пролетать как можно дольше, избегая столкновений с трубами.

МЕХАНИКА ИГРЫ:

1. Птица постоянно летит вперед (вправо)
2. На экране появляются трубы с промежутками
3. Игрок должен направить птицу через промежутки между трубами
4. За каждый успешный проход начисляется 1 очко
5. При столкновении с трубой или падении игра заканчивается

ПРОГРЕССИЯ СЛОЖНОСТИ:

- 0-5 очков: Начальная сложность (скорость: 4 px/frame)
- 5-15 очков: Средняя сложность (скорость: 5 px/frame)
- 15+ очков: Высокая сложность (скорость: 6 px/frame, случайные изменения)

F.3 Системные требования и зависимости

Минимальные требования:

- ОС: Windows 7+, macOS 10.11+, Linux (любой дистрибутив)

- Python: 3.8 или выше
- RAM: 256 MB минимум
- GPU: Встроенная видеокарта

Рекомендуемые требования:

- Python: 3.10+
- RAM: 512 MB
- CPU: 2+ ядра с частотой 2GHz+

Зависимости проекта:

pygame >= 2.1.0

numpy >= 1.21.0 (опционально, для расширенной статистики)

Установка:

1. Клонирование репозитория
   git clone https://github.com/meloch287/Flappy-Bird.git
   cd Flappy-Bird

2. Создание виртуального окружения
   python -m venv venv

3. Активация виртуального окружения
   На Windows:
   venv\Scripts\activate

   На macOS/Linux:
   source venv/bin/activate

4. Установка зависимостей
   pip install -r requirements.txt

5. Запуск игры
   python main.py

6. Проверка установки
   python -c "import pygame; print(pygame.version)"

F.4 Структура папок проекта

```
Flappy-Bird/
|
├── main.py              # Точка входа программы
├── game.py              # Основной класс Game с циклом
├── bird.py              # Класс Bird (персонаж)
├── pipe.py              # Класс Pipe (препятствие)
├── world.py             # Класс World (управление миром)
├── settings.py          # Конфигурация и константы
|
├── assets/
|   ├── images/
|   |   ├── bird_frame1.png
|   |   ├── bird_frame2.png
|   |   ├── bird_frame3.png
|   |   ├── pipe.png
|   |   ├── background.png
|   |   └── ground.png
|   |
|   └── sounds/
|       ├── jump.wav
|       ├── collision.wav
|       └── score.wav
|
├── config.ini           # Конфигурационный файл
├── requirements.txt     # Зависимости проекта
├── README.md            # Описание проекта
└── .gitignore           # Git конфигурация
```

G. ПОЛНЫЙ ИСХОДНЫЙ КОД

G.1 settings.py - Конфигурация и константы

```
"""
settings.py - Конфигурация и константы игры
"""
import os

# ========== РАЗМЕРЫ ЭКРАНА ==========
WIDTH = 800
HEIGHT = 600

# ========== ПАРАМЕТРЫ ИГРЫ ==========
FPS = 60
GAME_GRAVITY = 0.6
BIRD_JUMP_POWER = -9
BIRD_MAX_VELOCITY = 10

# ========== ПАРАМЕТРЫ ТРУБ ==========
PIPE_WIDTH = 50
PIPE_GAP = 150
PIPE_SPAWN_INTERVAL = 150
INITIAL_SCROLL_SPEED = 4
MAX_SCROLL_SPEED = 8

# ========== ПАРАМЕТРЫ ПТИЦЫ ==========
BIRD_START_X = WIDTH // 4
BIRD_START_Y = HEIGHT // 2
BIRD_SIZE = (40, 40)

# ========== ЦВЕТА ==========
COLOR_SKY = (135, 206, 250)
COLOR_GROUND = (34, 139, 34)
COLOR_PIPE = (34, 100, 34)
COLOR_WHITE = (255, 255, 255)
COLOR_BLACK = (0, 0, 0)

# ========== ПУТИ К РЕСУРСАМ ==========
ASSETS_PATH = os.path.join(os.path.dirname(__file__), "assets")
IMAGES_PATH = os.path.join(ASSETS_PATH, "images")
```

```
SOUNDS_PATH = os.path.join(ASSETS_PATH, "sounds")

# ========== УРОВНИ СЛОЖНОСТИ ==========
DIFFICULTY_LEVELS = {
    "easy": {"scroll_speed": 3, "pipe_gap": 200},
    "medium": {"scroll_speed": 4, "pipe_gap": 150},
    "hard": {"scroll_speed": 5, "pipe_gap": 120},
}
```

G.2 bird.py - Класс Bird

```
"""
bird.py - Класс персонажа птицы
"""
import pygame
from settings import *

class Bird(pygame.sprite.Sprite):
    """Класс для управления птицей в игре"""

    def __init__(self, pos):
        super().__init__()
        self.rect = pygame.Rect(pos, BIRD_SIZE)
        self.velocity = 0
        self.max_velocity = BIRD_MAX_VELOCITY

        self.image = pygame.Surface(BIRD_SIZE)
        self.image.fill((255, 165, 0))
        self.mask = pygame.mask.from_surface(self.image)

        self.frame_counter = 0
        self.animation_speed = 5
        self.wing_state = 0

    def jump(self):
        self.velocity = BIRD_JUMP_POWER

    def update(self):
        self.velocity = min(self.velocity + GAME_GRAVITY,
                            self.max_velocity)
        self.rect.y += self.velocity
```

```
        self.frame_counter += 1
        if self.frame_counter >= self.animation_speed:
            self.frame_counter = 0
            self.wing_state = (self.wing_state + 1) % 3
            self._update_animation()

    def _update_animation(self):
        self.image = pygame.Surface(BIRD_SIZE)
        self.image.fill((255, 165, 0))
        self.mask = pygame.mask.from_surface(self.image)
```

G.3 pipe.py - Класс Pipe

```
"""
pipe.py - Класс препятствия трубы
"""
import pygame
from settings import *

class Pipe(pygame.sprite.Sprite):
    """Класс для управления трубами-препятствиями"""

    def __init__(self, x, height, is_bottom=False):
        super().__init__()
        self.width = PIPE_WIDTH
        self.height = height
        self.is_bottom = is_bottom
        self.scored = False

        if is_bottom:
            y = HEIGHT - height
        else:
            y = 0

        self.rect = pygame.Rect(x, y, self.width, self.height)

        self.image = pygame.Surface((self.width, self.height))
        self.image.fill(COLOR_PIPE)
        self.mask = pygame.mask.from_surface(self.image)

    def update(self, scroll_speed):
        self.rect.x -= scroll_speed
```

```
    def is_off_screen(self):
        return self.rect.right < 0
```

G.4 world.py - Класс World

```
"""
world.py - Класс управления игровым миром
"""
import pygame
import random
from settings import *
from bird import Bird
from pipe import Pipe

class World:
    """Класс для управления игровым миром и всеми объектами"""

    def __init__(self):
        self.bird = Bird((BIRD_START_X, BIRD_START_Y))
        self.pipes = pygame.sprite.Group()
        self.score = 0
        self.best_score = 0
        self.scroll_speed = INITIAL_SCROLL_SPEED
        self.pipe_counter = 0
        self.last_pipe_y = None

        self._generate_new_pipes()

    def _generate_new_pipes(self):
        gap = PIPE_GAP

        if self.last_pipe_y is None:
            top_height = random.randint(50, HEIGHT - gap - 100)
        else:
            change = random.randint(-30, 30)
            top_height = self.last_pipe_y + change
            top_height = max(50, min(top_height, HEIGHT - gap - 100))

        self.last_pipe_y = top_height
        bottom_height = HEIGHT - top_height - gap
```

```
        top_pipe = Pipe(WIDTH, top_height, is_bottom=False)
        bottom_pipe = Pipe(WIDTH, bottom_height, is_bottom=True)

        self.pipes.add(top_pipe)
        self.pipes.add(bottom_pipe)

    def update(self, action=None):
        if action == "jump":
            self.bird.jump()

        self.bird.update()

        for pipe in self.pipes:
            pipe.update(self.scroll_speed)

        self.pipe_counter += 1
        if self.pipe_counter >= PIPE_SPAWN_INTERVAL:
            self._generate_new_pipes()
            self.pipe_counter = 0

        for pipe in list(self.pipes):
            if pipe.is_off_screen():
                self.pipes.remove(pipe)

        self._check_score()
        self._update_difficulty()

    def _check_score(self):
        for pipe in self.pipes:
            if (not pipe.scored and
                self.bird.rect.left > pipe.rect.right and
                pipe.is_bottom):
                self.score += 1
                pipe.scored = True

    def _update_difficulty(self):
        new_scroll_speed = INITIAL_SCROLL_SPEED + (self.score // 5) * 0.5
        self.scroll_speed = min(new_scroll_speed, MAX_SCROLL_SPEED)

    def check_collision(self):
```

```
        collisions = pygame.sprite.spritecollideany(self.bird,
                                            self.pipes)
        if collisions:
            return True

        if self.bird.rect.top <= 0 or self.bird.rect.bottom >= HEIGHT:
            return True

        return False

    def reset(self):
        self.best_score = max(self.best_score, self.score)
        self.score = 0
        self.scroll_speed = INITIAL_SCROLL_SPEED
        self.pipe_counter = 0
        self.last_pipe_y = None

        self.bird = Bird((BIRD_START_X, BIRD_START_Y))

        self.pipes.empty()
        self._generate_new_pipes()

    def draw(self, surface):
        surface.fill(COLOR_SKY)

        for pipe in self.pipes:
            surface.blit(pipe.image, pipe.rect)

        surface.blit(self.bird.image, self.bird.rect)

        pygame.draw.rect(surface, COLOR_GROUND,
                         (0, HEIGHT - 30, WIDTH, 30))
```

G.5 game.py - Основной класс Game

```
"""
game.py - Основной класс игры с циклом
"""
import pygame
from settings import *
from world import World
```

```
class Game:
    """Основной класс игры с циклом отрисовки"""

    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode((WIDTH, HEIGHT))
        pygame.display.set_caption("Flappy Bird Game")
        self.clock = pygame.time.Clock()
        self.font = pygame.font.Font(None, 74)
        self.small_font = pygame.font.Font(None, 36)

        self.world = World()
        self.running = True
        self.game_state = "MENU"

    def handle_input(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    if self.game_state == "MENU":
                        self.game_state = "PLAYING"
                    elif self.game_state == "PLAYING":
                        self.world.update("jump")
                    elif self.game_state == "GAME_OVER":
                        self.game_state = "MENU"
                        self.world.reset()

                elif event.key == pygame.K_p:
                    if self.game_state == "PLAYING":
                        self.game_state = "PAUSED"
                    elif self.game_state == "PAUSED":
                        self.game_state = "PLAYING"

                elif event.key == pygame.K_r:
                    self.world.reset()
                    self.game_state = "MENU"
```

```
            elif event.key == pygame.K_q:
                self.running = False

    def update(self):
        if self.game_state == "PLAYING":
            self.world.update()

            if self.world.check_collision():
                self.game_state = "GAME_OVER"

    def render(self):
        self.world.draw(self.screen)

        score_text = self.font.render(str(self.world.score),
                                      True, COLOR_BLACK)
        self.screen.blit(score_text, (10, 10))

        best_score_text = self.small_font.render(
            f"Best: {self.world.best_score}", True, COLOR_BLACK)
        self.screen.blit(best_score_text, (WIDTH - 200, 10))

        if self.game_state == "MENU":
            self._render_menu()
        elif self.game_state == "PAUSED":
            self._render_paused()
        elif self.game_state == "GAME_OVER":
            self._render_game_over()

        pygame.display.flip()

    def _render_menu(self):
        menu_text = self.font.render("FLAPPY BIRD", True, COLOR_BLACK)
        instruction_text = self.small_font.render(
            "Press SPACE to start", True, COLOR_BLACK)

        self.screen.blit(menu_text,
            (WIDTH // 2 - menu_text.get_width() // 2, HEIGHT // 2 - 100))
        self.screen.blit(instruction_text,
            (WIDTH // 2 - instruction_text.get_width() // 2,
             HEIGHT // 2 + 50))
```

```
    def _render_paused(self):
        paused_text = self.font.render("PAUSED", True, COLOR_BLACK)
        self.screen.blit(paused_text,
            (WIDTH // 2 - paused_text.get_width() // 2, HEIGHT // 2 - 50))

    def _render_game_over(self):
        game_over_text = self.font.render("GAME OVER", True, COLOR_BLACK)
        score_text = self.small_font.render(
            f"Final Score: {self.world.score}", True, COLOR_BLACK)
        restart_text = self.small_font.render(
            "Press SPACE to play again", True, COLOR_BLACK)

        self.screen.blit(game_over_text,
            (WIDTH // 2 - game_over_text.get_width() // 2,
             HEIGHT // 2 - 100))
        self.screen.blit(score_text,
            (WIDTH // 2 - score_text.get_width() // 2, HEIGHT // 2))
        self.screen.blit(restart_text,
            (WIDTH // 2 - restart_text.get_width() // 2,
             HEIGHT // 2 + 100))

    def run(self):
        while self.running:
            self.handle_input()
            self.update()
            self.render()
            self.clock.tick(FPS)

        pygame.quit()
```

G.6 main.py - Точка входа программы

```
"""
main.py - Точка входа программы
"""
from game import Game

def main():
    """Функция запуска игры"""
    game = Game()
    game.run()
```

```
if __name__ == "__main__":
    main()
```

G.7 requirements.txt - Зависимости

```
pygame>=2.1.0

numpy>=1.21.0
```

G.8 config.ini - Конфигурационный файл

```
[GAME]
WIDTH = 800
HEIGHT = 600
FPS = 60
DIFFICULTY = medium

[BIRD]
JUMP_POWER = 9
MAX_VELOCITY = 10
GRAVITY = 0.6

[PIPES]
WIDTH = 50
GAP = 150
SPAWN_INTERVAL = 150
INITIAL_SPEED = 4
MAX_SPEED = 8

[GRAPHICS]
ENABLE_ANIMATIONS = True
ENABLE_PARTICLES = True
QUALITY = high
```

G.9 Структура организации ресурсов

```
assets/
├── images/
│   ├── bird_frame1.png (34x34 px, PNG)
│   ├── bird_frame2.png (34x34 px, PNG)
│   ├── bird_frame3.png (34x34 px, PNG)
│   ├── pipe.png (50x500 px, PNG)
│   ├── background.png (800x600 px, PNG)
```

```
|   ├── ground.png (800x30 px, PNG)
|   └── icon.png (64x64 px, PNG)
|
└── sounds/
    ├── jump.wav (100ms, WAV)
    ├── collision.wav (200ms, WAV)
    ├── score.wav (150ms, WAV)
    └── background.mp3 (background music)
```

Рекомендации по форматам:
• Изображения: PNG с альфа-каналом для прозрачности
• Звуки: WAV для эффектов, MP3 для музыки
• Все изображения должны быть оптимизированы (не более 100 KB каждое)