Товстик Мария

244076

Практикум по программированию

Лабораторная работа 3

Flappy bird

Роль в проекте – разработчик

Состав команды: Бугарева Елизавета Ивановна, Товстик Мария

Описание игры

Разработать клон классической игры Flappy Bird с использованием языка Python и библиотеки PyGame. Игра должна включать:

1. Управляемого персонажа (птицу), который может совершать прыжки
2. Генерацию случайных препятствий (труб)
3. Систему подсчета очков
4. Механику столкновений
5. Экран начала игры и проигрыша
6. Возможность рестарта игры

Классическая игра:

1. Игрок управляет птицей, которая постоянно падает вниз под действием гравитации
2. Нажатие на пробел заставляет птицу совершить прыжок вверх
3. На пути птицы генерируются пары труб с промежутком
4. Задача — пролететь через как можно больше пар труб, не столкнувшись с ними
5. Столкновение с трубой или землей приводит к завершению игры
6. За каждую успешно пройденную пару труб начисляется 1 очко

Реализованные изменения и улучшения

1. Переработанная система классов — код организован по принципам ООП
2. Гибкая настройка параметров — легко изменять физику, скорость, размеры
3. Расширенная система состояний игры — разделение на "игра" и "меню"

Используемые инструменты и технологии

- Python 3.8+ — основной язык программирования
- PyGame 2.0+ — библиотека для создания игр
- Pygame Freetype — работа со шрифтами
- Random — генерация случайных значений
- Объектно-ориентированное программирование — архитектура проекта

АРХИТЕКТУРА ПРОЕКТА

1. GameObject (Абстрактный класс)

- Назначение: Базовый класс для всех объектов в игре
- Обязанности:

    1. Хранение позиции (x, y)
    2. Хранение изображения и прямоугольника для отрисовки
    3. Определение интерфейса для обновления и отрисовки

- Методы:

    1. `update()`: абстрактный метод обновления состояния
    2. `draw(surface)`: отрисовка объекта на поверхности

2. MovingObject (Дочерний класс GameObject)

- Назначение: Класс для движущихся объектов
- Обязанности:

    1. Добавление свойств скорости и направления
    2. Реализация базового движения

- Методы - `move()`: перемещение объекта на основе скорости

3. Bird (Дочерний класс MovingObject)

- Назначение: Игровой персонаж - птица
- Обязанности:

    1. Реализация физики прыжков и падения
    2. Обработка пользовательского ввода
    3. Проверка границ игрового поля

- Методы:

    1. `jump()`: выполнение прыжка
    2. `check_bounds()`: проверка выхода за границы

4. Wall (Наследник GameObject)

- Назначение: Препятствие - труба
- Обязанности:

    1. Автоматическое движение влево
    2. Обнаружение столкновений с птицей
    3. Отслеживание прохождения птицы

- Методы:

    1. `check_collision()`: проверка столкновения

2. check_pass(): проверка прохождения птицы
3. is_offscreen(): проверка ухода за экран

5. WallPair (Композитный класс)

- Назначение: Управление парой труб (верхняя и нижняя)
- Обязанности:

1. Координация двух стен как единого объекта
2. Централизованная проверка столкновений и прохождения
3. Синхронизация движения

6. Game (Основной класс)

- Назначение: Управление всем игровым процессом
- Обязанности:

1. Инициализация PyGame и ресурсов
2. Управление игровым циклом
3. Обработка событий
4. Координация всех игровых объектов
5. Управление состоянием игры

- Методы:

1. run(): главный игровой цикл
2. handle_events(): обработка ввода
3. update_game_logic(): обновление состояния игры
4. draw(): отрисовка всей сцены

7.Config

- Назначение: Взаимодействие с файлом json, в котором хранятся конфигурации

Обоснование использованных шаблонов проектирования

1. Шаблон "Наследование" (Inheritance)

Применение: Иерархия классов GameObject → MovingObject → Bird

```
class GameObject:  # Базовый класс для всех игровых объектов
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.image = None
        self.rect = None

class MovingObject(GameObject):  # Класс для движущихся объектов
    def __init__(self, x, y):
        super().__init__(x, y)


class Bird(MovingObject):  # Класс птицы
    def __init__(self, x, y, config):
        super().__init__(x, y)
```

Наследование позволяет избежать дублирования кода, создавая логическую иерархию объектов. Все игровые объекты имеют общие свойства (позиция, изображение), движущиеся объекты добавляют скорость, а птица специфичную физику

2. Шаблон “Принцип единственной ответственности (Single Responsibility Principle)”

Каждый класс отвечает за одну задачу:

- Bird — только за физику птицы
- Wall — только за поведение стены
- Game — только за управление игровым процессом
- Config — только за работу с конфигурацией

Упрощает тестирование, отладку и модификацию кода

3.Шаблон Компоновщик (composite)

Класс WallPair управляет парой стен как единым объектом

```
class WallPair:  # Класс пары стен (верхняя + нижняя)
    def __init__(self, x, screen_height, config):  # Управляет двумя стенами
как единым объектом
# Создание верхней стены
self.top_wall = Wall(x, self.wall_height, config, flip=True)

# Создание нижней стены
self.bottom_wall = Wall(x, self.wall_height + self.gap_height, config,
flip=False)

self.walls = [self.top_wall, self.bottom_wall]
def update(self):
    for wall in self.walls:
        wall.update()
```

Код работает с WallPair как с единым объектом, не заботясь о внутренней структуре.

Реализованный функционал

1. Управляемая птица с физикой - Класс Bird с методами update() и jump()

2. Генерация случайных препятствий – WallPair с random.randint()

3. Система подсчета очков – счетчик в классе Game

4. Обнаружение столкновений – метод check_collision()

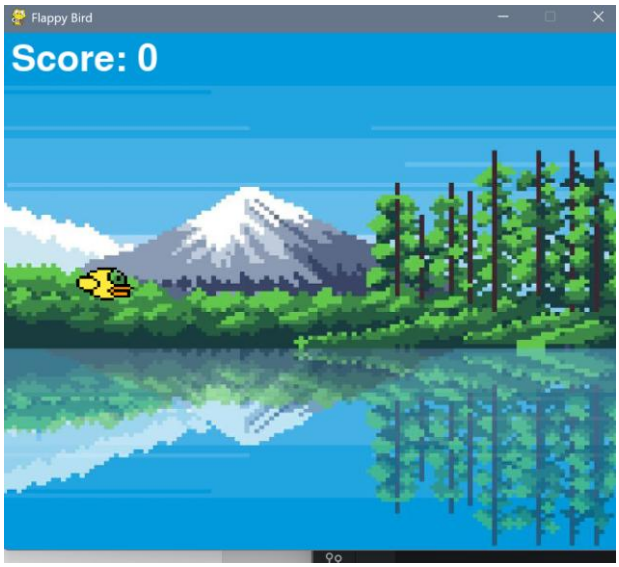
5. Состояние игры – переменная game_status

6. Перезапуск игры – метод restart_game()

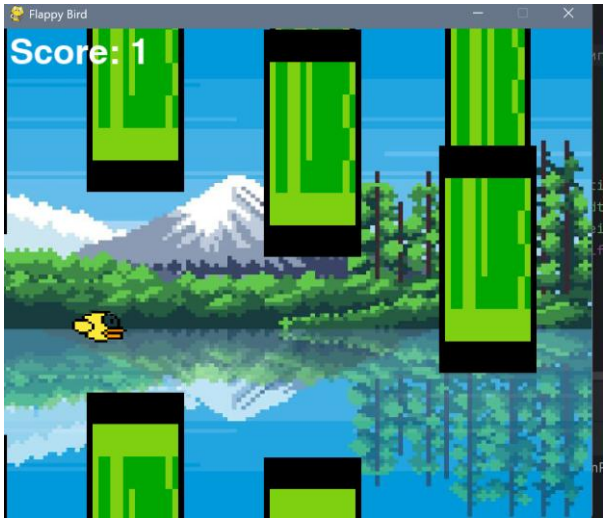
7. Файл с конфигурациями – bird_config.json

Скриншоты

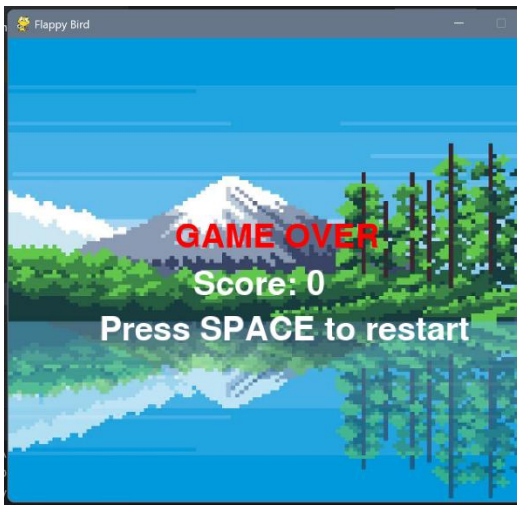
Начало игры



Изменение score после прохождения столба



Завершение игры после столкновения или падения за пределы экрана



Фрагменты кода для ключевых алгоритмов

1.Физика полета птицы

```
class Bird(MovingObject):
    def update(self):
        # Применение гравитации (ускорение вниз)
        self.speed_y += self.gravity
        # Обновление позиции
        self.y += self.speed_y
        self.rect.y = self.y

    def jump(self):
        # Мгновенное изменение скорости вверх
        self.speed_y = self.jump_strength
```

2.Генерация препятствий

```
class WallPair:
    def __init__(self, x, screen_height, config):
        # Случайная высота верхней трубы
        self.wall_height = random.randint(
            self.min_height,
            self.max_height
        )

        # Создание пары с фиксированным промежутком
        self.top_wall = Wall(x, self.wall_height, config, flip=True)
        self.bottom_wall = Wall(
            x,
            self.wall_height + self.gap_height,
            config,
            flip=False
        )
```

3.Обнаружение столкновения

```
class Wall:
    def check_collision(self, bird_rect):
        # Использование встроенного метода PyGame
        return self.rect.colliderect(bird_rect)

    def check_pass(self, bird_rect):
        # Проверка, что птица прошла стену
        if not self.passed and self.rect.right < bird_rect.left:
            self.passed = True
            return True
        return False
```

4.Игровой цикл

```
class Game:
    def run(self):
        while self.running:
            # 1. Обработка ввода
            self.handle_events()

            # 2. Обновление логики
            self.update_game_logic()

            # 3. Отрисовка
            self.draw()

            # 4. Контроль FPS
```

```
            pygame.display.flip()
            self.clock.tick(self.fps)
```

Описание решенных технических проблем

1. Удаление объектов во время итерации

Проблема: При удалении стен из списка во время итерации происходил пропуск элементов

Решение - итерация по копии списка

2. Конфликт систем координат

Проблема - верхняя труба должна "расти" вниз, нижняя – вверх

Решение- разные точки привязки для разных типов труб

F. Инструкции по запуску и игре

Space (пробел)	Прыжок птицы вверх или начало игры	Актиыная игра (game_status=’game’) Экран проигрыша (game_status = 'menu')
esc	Выход из игры	Любое состояние
F11	Полноэкранный режим	Любое состояние
P	Пауза/продолжение	Активная игра

Правила и цели игры

Цель игры

Набрать максимальное количество очков, пролетая через препятствия.

Основные правила

- Старт: Птица появляется в левой части экрана

- Управление: Нажмите ПРОБЕЛ для прыжка

- Физика:

Птица падает под действием гравитации, каждый прыжок дает импульс вверх, импульс прыжка фиксированный

- Препятствия:

Трубы появляются справа через равные промежутки времени, между верхней и нижней трубой есть зазор (проход)

- подсчет очков: +1 за каждое препятствие

- Проигрыш в случае столкновения с трубой или падения на землю

Стратегии игры:

1. Короткие прыжки лучше длинных
2. Лучше держаться ровно по центру между трубами
3. Плавные нажатия эффективнее резких

Системные требования и зависимости:

OS - Windows 7+, macOS 10.9+, Linux

RAM - 512 MB

Python – version 3.8+

PyGame – version 2.0+

Установка зависимостей:

```
pip install pygame==2.5.2 # Установка PyGame
```

Полный код

```
import random
import pygame
import pygame.freetype
import json


class Config: # Класс для работы с конфигурационным файлом
    def __init__(self, config_path='bird_config.json'):
        self.config_path = config_path
        self.data = self.load_config()

    def load_config(self):
        try:
            with open(self.config_path, 'r', encoding='utf-8') as f:
                return json.load(f)
        except FileNotFoundError:
            print('Конфигурационный файл не найден')
            return {}

    def get(self, key):
        return self.data.get(key, {})


class GameObject:  # Базовый класс для всех игровых объектов
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.image = None
        self.rect = None

    def update(self):
        pass

    def draw(self, surface):
        if self.image and self.rect:
            surface.blit(self.image, self.rect)
```

```python
class MovingObject(GameObject):  # Класс для движущихся объектов
    def __init__(self, x, y):
        super().__init__(x, y)
        self.speed = 0  # Добавляет свойства: скорость, направление
        self.dx = 0
        self.dy = 0

    def move(self):  # метод move() для перемещения
        self.x += self.dx * self.speed
        self.y += self.dy * self.speed
        if self.rect:
            self.rect.x = self.x
            self.rect.y = self.y


class Bird(MovingObject):  # Класс птицы
    def __init__(self, x, y, config):
        super().__init__(x, y)
        bird_config = config.get('bird_settings')
        self.gravity = bird_config.get('gravity', 0.4)
        self.jump_strength = bird_config.get("jump_strength", -6)
        image_path = bird_config.get("image_path", "bird.png")
        self.width = bird_config.get("width", 60)
        self.height = bird_config.get("height", 35)
        self.speed_y = 0

        # Загрузка и масштабирование изображения
        self.image = pygame.image.load(image_path).convert_alpha()
        self.image = pygame.transform.scale(self.image, (self.width,
self.height))
        self.rect = self.image.get_rect(center=(x, y))

    def update(self):
        # Применяем гравитацию
        self.speed_y += self.gravity
        self.y += self.speed_y
        self.rect.y = self.y

    def jump(self):  # делает прыжок
        self.speed_y = self.jump_strength

    def check_bounds(self, screen_height):
        # Проверка границ экрана
        if self.rect.top <= 0:
            self.rect.top = 0
            self.y = 0
            self.speed_y = 0
            return 'top'
        elif self.rect.bottom >= screen_height:
            self.rect.bottom = screen_height
            self.y = screen_height - self.rect.height
            return 'bottom'
        return None


class Wall(GameObject):  # Класс отдельной стены
    def __init__(self, x, y, config, flip=False):
        super().__init__(x, y)
        # Загрузка и подготовка изображения
        wall_config = config.get("wall_settings")
        self.speed = wall_config.get("speed", 2)
        width = wall_config.get("width", 100)
```

```
        height = wall_config.get("height", 500)

        image_path = wall_config.get("image_path", "wall.png")
        self.image = pygame.image.load(image_path).convert_alpha()
        self.image = pygame.transform.scale(self.image, (width, height))
        if flip:
            self.image = pygame.transform.flip(self.image, False, True)

        self.rect = self.image.get_rect()
        if flip:
            self.rect.bottomleft = (x, y)  # Верхняя стена
        else:
            self.rect.topleft = (x, y)  # Нижняя стена

        self.passed = False
        self.flip = flip

    def update(self):
        # Движение стены влево
        self.x -= self.speed
        self.rect.x = self.x

    def is_offscreen(self):  # проверка ухода за экран
        # Проверка, ушла ли стена за экран
        return self.rect.right < 0

    def check_collision(self, bird_rect):
        # Проверка столкновения с птицей
        return self.rect.colliderect(bird_rect)

    def check_pass(self, bird_rect):
        # Проверка, прошла ли птица стену
        if not self.passed and self.rect.right < bird_rect.left:
            self.passed = True
            return True
        return False


class WallPair:  # Класс пары стен (верхняя + нижняя)
    def __init__(self, x, screen_height, config):  # Управляет двумя стенами
как единым объектом
        wall_config = config.get('wall_settings')
        self.gap_height = wall_config.get("gap_height", 200)
        self.min_height = wall_config.get("min_height", 100)
        self.max_height = wall_config.get("max_height", 400)

        # Случайная высота для стен
        self.wall_height = random.randint(self.min_height, self.max_height)
        self.screen_height = screen_height

        # Создание верхней стены
        self.top_wall = Wall(x, self.wall_height, config, flip=True)

        # Создание нижней стены
        self.bottom_wall = Wall(x, self.wall_height + self.gap_height,
config, flip=False)

        self.walls = [self.top_wall, self.bottom_wall]
        self.scored = False

    def update(self):
        for wall in self.walls:
            wall.update()
```

```python
    def draw(self, surface):
        for wall in self.walls:
            wall.draw(surface)

    def check_collisions(self, bird_rect):
        for wall in self.walls:
            if wall.check_collision(bird_rect):
                return True
        return False

    def check_pass(self, bird_rect):
        if not self.scored:
            for wall in self.walls:
                if not wall.passed and wall.check_pass(bird_rect):
                    self.scored = True
                    return True
        return False

    def is_offscreen(self):
        return all(wall.is_offscreen() for wall in self.walls)


class Game:  # Главный класс, управляющий всей игрой
    def __init__(self):
        self.config = Config()
        pygame.init()

        # Настройки окна
        game_settings = self.config.get('game_settings')
        self.width = game_settings.get('window_width', 600)
        self.height = game_settings.get('window_height', 500)
        self.screen = pygame.display.set_mode((self.width, self.height))
        pygame.display.set_caption('Flappy Bird')

        # Частота кадров
        self.clock = pygame.time.Clock()
        self.fps = game_settings.get('fps', 60)

        # Загрузка фона
        self.bg = pygame.image.load('bg.jpg').convert()
        self.bg = pygame.transform.scale(self.bg, (self.width, self.height))

        # Игровые объекты
        bird_config = self.config.get('bird_settings')
        start_x = bird_config.get('start_x', 100)
        start_y = bird_config.get('start_y', self.height // 2)

        self.bird = Bird(start_x, start_y, self.config)
        self.wall_pairs = []

        # Таймер для создания стен
        wall_config = self.config.get('wall_settings')
        spawn_interval = wall_config.get('spawn_interval', 1500)

        self.spawn_wall_event = pygame.USEREVENT
        pygame.time.set_timer(self.spawn_wall_event, spawn_interval)

        # Состояние игры
        self.game_status = 'game'  # 'game' или 'menu'
        self.score = 0

        # Шрифт
        font_name = game_settings.get("font_name", None)
        font_size = game_settings.get("font_size", 36)
```

```
        self.font = pygame.freetype.Font(font_name, font_size)

        # Тексты
        texts = self.config.get("texts")
        self.score_text = texts.get("score_prefix", "Score: ")
        self.game_over_text = texts.get("game_over", "GAME OVER")
        self.restart_text = texts.get("restart_instruction", "Press SPACE to
restart")

        # Флаг работы игры
        self.running = True

    def handle_events(self):  # обработка ввода
        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:
                    if self.game_status == 'game':
                        self.bird.jump()
                    elif self.game_status == 'menu':
                        self.restart_game()

            if event.type == self.spawn_wall_event and self.game_status ==
'game':
                self.wall_pairs.append(
                    WallPair(self.width, self.height, self.config)
                )

    def update_game_logic(self):  # обновление логики
        if self.game_status == 'game':
            # Обновление птицы
            self.bird.update()

            # Проверка границ для птицы
            bounds_result = self.bird.check_bounds(self.height)
            if bounds_result == 'bottom':
                self.game_status = 'menu'

            # Обновление и проверка стен
            for wall_pair in self.wall_pairs[:]:
                wall_pair.update()

                # Проверка столкновений
                if wall_pair.check_collisions(self.bird.rect):
                    self.game_status = 'menu'

                # Проверка прохождения
                if wall_pair.check_pass(self.bird.rect):
                    self.score += 1  # +1 за пару стен

                # Удаление стен за экраном
                if wall_pair.is_offscreen():
                    self.wall_pairs.remove(wall_pair)

    def draw(self):
        # Рисование фона
        self.screen.blit(self.bg, (0, 0))

        if self.game_status == 'game':
            # Рисование стен
            for wall_pair in self.wall_pairs:
                wall_pair.draw(self.screen)
```

```
            # Рисование птицы
            self.bird.draw(self.screen)

            # Отображение счета
            self.font.render_to(
                self.screen,
                (10, 10),
                f'{self.score_text}{self.score}',
                (255, 255, 255)
            )
        else:
            # Экран проигрыша
            self.font.render_to(
                self.screen,
                (self.width // 2 - 120, self.height // 2 - 50),
                self.game_over_text,
                (255, 0, 0)
            )
            self.font.render_to(
                self.screen,
                (self.width // 2 - 100, self.height // 2),
                f'{self.score_text}{self.score}',
                (255, 255, 255)
            )
            self.font.render_to(
                self.screen,
                (self.width // 2 - 200, self.height // 2 + 50),
                self.restart_text,
                (255, 255, 255)
            )

    def restart_game(self):
        self.game_status = 'game'
        bird_config = self.config.get('bird_settings')
        start_x = bird_config.get('start_x', 100)
        start_y = bird_config.get('start_y', self.height // 2)
        self.bird = Bird(start_x, start_y, self.config)
        self.wall_pairs = []
        self.score = 0

    def run(self):
        while self.running:
            self.handle_events()
            self.update_game_logic()
            self.draw()

            pygame.display.flip()
            self.clock.tick(self.fps)

        pygame.quit()


# Запуск игры
if __name__ == "__main__":
    game = Game()
    game.run()
```

Конфигурационный файл bird_config.json

```
{
    "game_settings": {
        "window_width": 600,
        "window_height": 500,
        "window_title": "Flappy Bird",
        "fps": 60,
        "background_color": [135, 206, 235],
        "font_name": null,
        "font_size": 36,
        "font_color": [255, 255, 255]
    },

    "bird_settings": {
        "start_x": 100,
        "start_y": 250,
        "width": 60,
        "height": 35,
        "gravity": 0.4,
        "jump_strength": -6,
        "image_path": "bird.png",
        "min_y": 0,
        "max_y": 500
    },

    "wall_settings": {
        "width": 100,
        "height": 500,
        "speed": 2,
        "spawn_interval": 1500,
        "min_gap": 180,
        "max_gap": 220,
        "min_height": 100,
        "max_height": 400,
        "image_path": "wall.png"
    },

    "colors": {
        "white": [255, 255, 255],
        "black": [0, 0, 0],
        "red": [255, 0, 0],
        "green": [0, 255, 0],
        "blue": [0, 0, 255],
        "game_over": [255, 0, 0]
    },

    "texts": {
        "score_prefix": "Score: ",
        "game_over": "GAME OVER",
        "restart_instruction": "Press SPACE to restart"
    }
}
```

Структура организации ресурсов

```
flappy_bird_project/
│
├── main.py                 # Главный исполняемый файл
├── bird_config.json        # Конфигурация игры
├── requirements.txt        # Зависимости Python
├── README.md               # Документация проекта
│
├── assets/                 # Графические ресурсы
│   ├── bird.png            # Изображение птицы (60x35)
│   ├── wall.png            # Изображение трубы (100x500)
│   └── bg.jpg              # Фон игры (600x500)
```