ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3.3

РАЗРАБОТКА ПРОСТЫХ АРКАДНЫХ ИГР

A. Титульная страница

Название предмета: практикум по программированию

Номер лабораторной работы: 3.3

Название игры: Flappy Bird (PyQt6 Edition)

Студент: Мещеряков Роман

Номер билета: 242058

Группа: ИД24-2

B. Описание игры

1. Текст задания

Разработать простую аркадную игру с использованием двумерной анимации. Реализовать объектно-ориентированную архитектуру, систему управления, подсчета очков и состояний игры. Использовать конфигурационные файлы для настройки параметров.

2. Описание игры и правил

Flappy Bird — аркадная игра, в которой игрок управляет птицей, пытаясь пролететь между трубами.

- **Цель:** Пролететь как можно дальше, не задев трубы и землю.
- **Управление:** Нажатие `Пробел` или `ЛКМ` заставляет птицу подпрыгнуть. Без действий птица падает под действием гравитации.
- **Начисление очков:** +1 очко за каждую пройденную пару труб.

3. Реализованные изменения и улучшения

В отличие от классической версии, в наш проект добавлены:

- **Data-Driven Design:** Все параметры физики и настройки окна вынесены в `settings.json`.
- **Система Тем:** Возможность смены скинов (фон/птица/трубы) на лету с предпросмотром.
- **Адаптивный UI:** Интерфейс подстраивается под размеры окна ПК, добавлены боковые панели статистики и настроек.
- **Расширенное аудио:** Раздельная регулировка громкости музыки и эффектов (SFX) с сохранением настроек.
- **Состояния:** Добавлено состояние "Get Ready" перед стартом.

4. Инструменты и технологии

- **Язык:** Python 3.10+
- **Графический фреймворк:** PyQt6 (QtWidgets, QtGui, QtCore, QtMultimedia)
- **Формат данных:** JSON (настройки, темы, стили)
- **Среда разработки:** PyCharm / VS Code

C. Архитектура проекта

Проект построен на принципах ООП с использованием событийной модели PyQt (Signals & Slots).

1. Диаграмма основных классов

- **MainWindow (QMainWindow):** Главный контейнер. Управляет лейаутом (Лев. панель, Игра, Прав. панель) и связывает компоненты.
- **GameArea (QWidget):** Ядро игры. Содержит игровой цикл (`QTimer`), управляет объектами и отрисовкой (`paintEvent`).
- **Bird (Class):** Отвечает за физику персонажа (скорость, гравитация, угол поворота).
- **Pipe (Class):** Генерирует препятствия, отвечает за их движение и проверку прохождения.
- **SoundManager (QObject):** Управляет воспроизведением звуков и музыки, регулирует громкость.
- **Theme (Class):** Хранит ресурсы (изображения) и генерирует превью.

2. Шаблоны проектирования

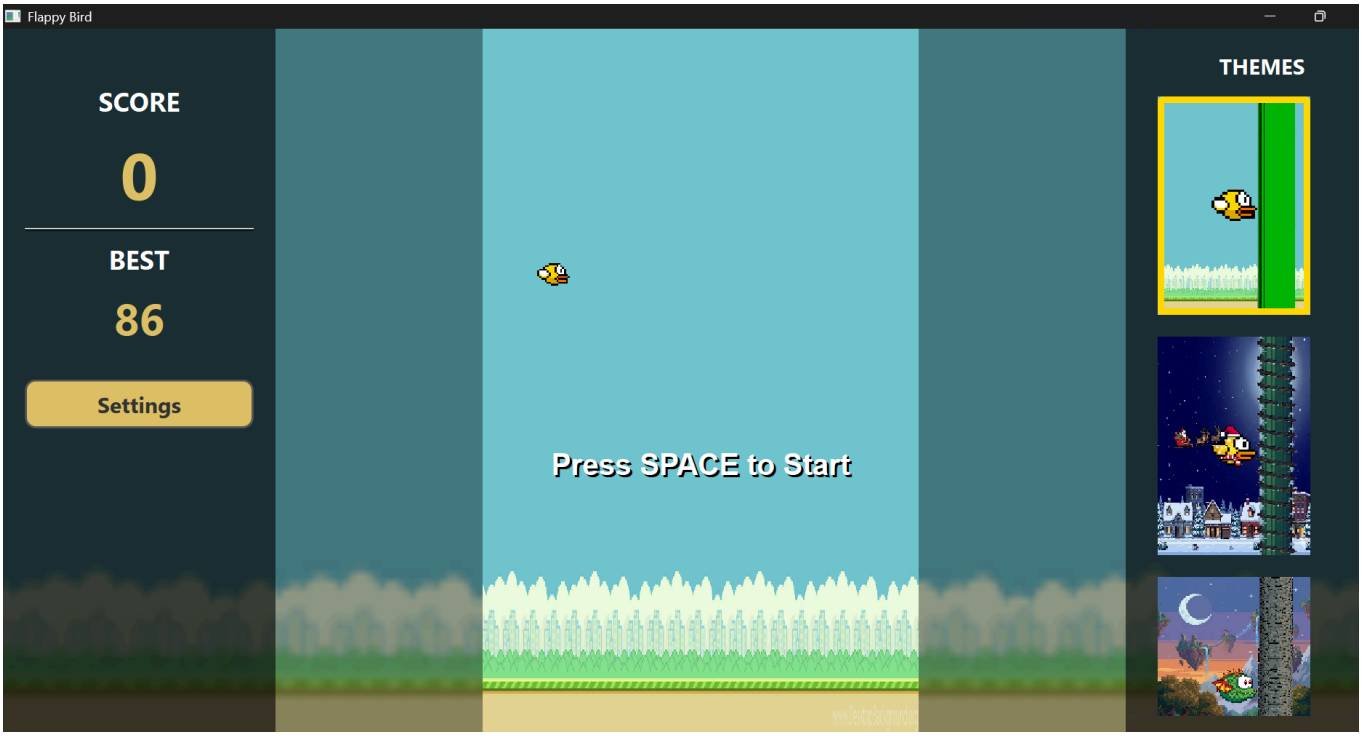
- **Game Loop:** Реализован через `QTimer`, вызывающий метод `update_game` 60 раз в секунду.
- **Observer (Signals):** Используется для обновления счета (`score_updated`) и смены тем.
- **Separation of Concerns:** Логика настроек вынесена в JSON, стили — в JSON/CSS, логика звука — в отдельный менеджер.

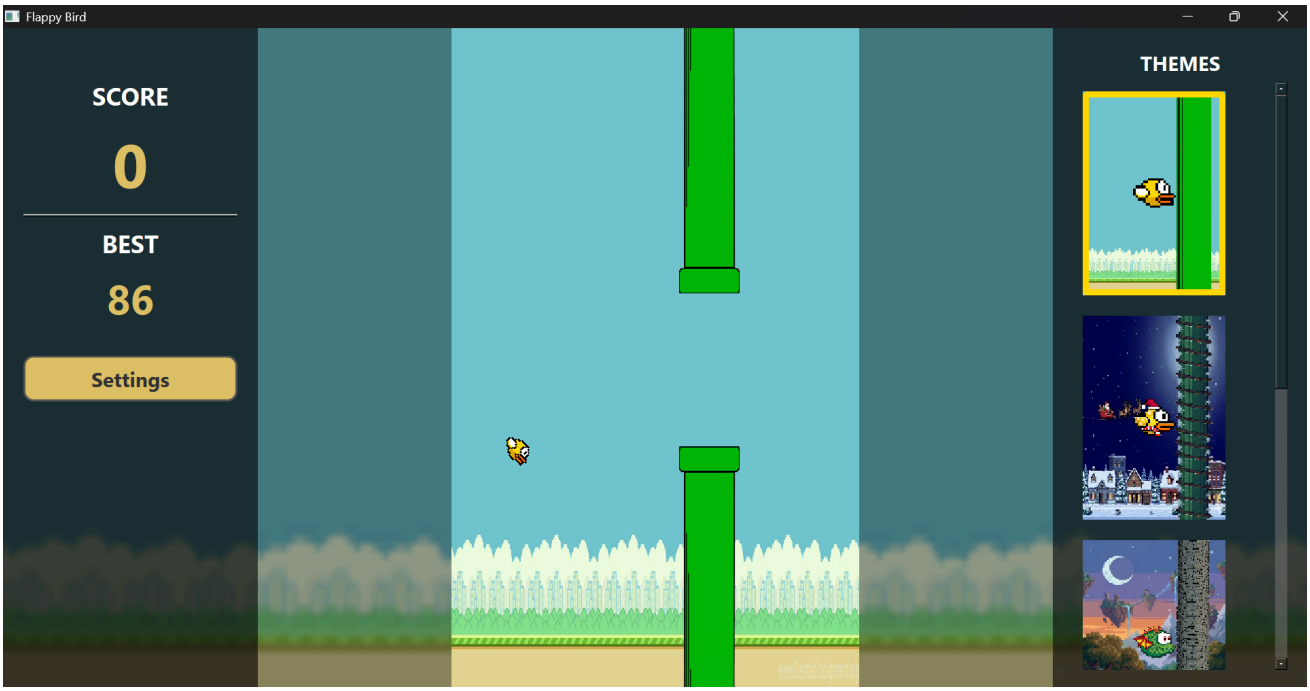
D. Реализованный функционал

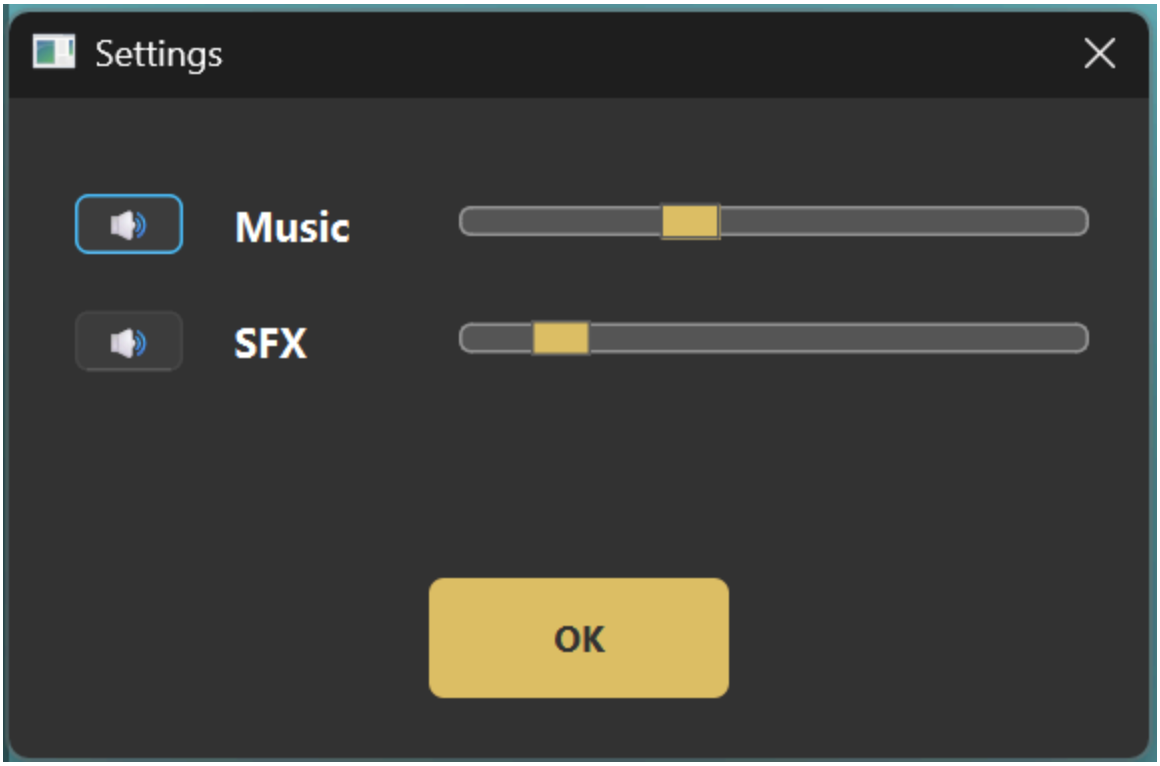
1. Основные требования

- [x] ~~**Игровая механика:** Гравитация, прыжки, столкновения реализованы.~~
- [x] ~~**Подсчет очков:** Текущий счет и "Лучший счет" (сохраняется между запусками).~~
- [x] ~~**Управление:** Клавиатура (Space) и Мышь.~~
- [x] ~~**Состояния:** Menu (Get Ready) -> Playing -> Game Over.~~

2. Доказательства (Скриншоты)







3. Ключевые алгоритмы (Фрагменты кода)

Физика птицы и вращение:

Python

```
def move(self):
    self.velocity += self.gravity
    self.y += self.velocity
    # ...
```

```
def draw(self, painter, bird_pixmap):
    painter.save()
    # Перенос координат в центр птицы для вращения
    painter.translate(self.x + self.size/2, self.y + self.size/2)
    painter.rotate(self.velocity * self.config["rotation_multiplier"])
    # ...
    painter.restore()
```

Решение технических проблем:

Проблема: При частых столкновениях вылетал аудиодрайвер (AUDCLNT_E_DEVICE_INVALIDATED) и пропадали звуки из-за gc.collect.

Решение: SoundManager унаследован от QObject, передается parent=self для всех эффектов, убрана принудительная сборка мусора, добавлена защита от множественного вызова "Game Over".

Python

```
# sound_manager.py
class SoundManager(QObject):
    def __init__(self, config):
        super().__init__()
        # Привязка к родителю предотвращает удаление объектов Python-ом
        self.music_player = QMediaPlayer(self)
        # ...
```

E. Инструкции по запуску и игре

Системные требования

- ОС: Windows 10/11, macOS, Linux
- Python 3.8 или выше
- Библиотеки: `PyQt6`

Установка и запуск

1. Установить зависимости:

   Bash

```
pip install PyQt6
```

2. Запустить игру:

   Bash

```
python main.py
```

Управление

- **SPACE / ЛКМ:** Подпрыгнуть / Начать игру.
- **Мышь (Правая панель):** Выбор темы кликом.
- **Кнопка Settings:** Открыть настройки звука.

F. Полный исходный код

Структура проекта
FlappyBird/
├── assets/
│ ├── images/ (background.png, bird.png, pipe.png...)
│ └── sounds/ (jump.wav, hit.wav, score.wav...)
├── main.py # Точка входа, класс MainWindow
├── game_area.py # Игровой цикл и логика
├── bird.py # Класс Птицы
├── pipe.py # Класс Трубы
├── theme.py # Класс Темы (ресурсы)
├── theme_card.py # Виджет выбора темы
├── sound_manager.py # Управление звуком (QMediaPlayer/QSoundEffect)
├── json_reader.py # Утилиты для чтения/записи JSON
├── settings_dialog.py# Окно настроек
├── settings.json # Конфигурация физики и окна
├── styles.json # CSS стили для кнопок
└── themes.json # Список доступных скинов

Python

```
# bird.py
from PyQt6.QtCore import QRectF
from PyQt6.QtGui import QColor
```

```
class Bird:
    def __init__(
        self, x, y, size, gravity, jump_velocity, rotation_multiplier,
color="#FFD700"
    ):
        self.x = x
        self.y = y
        self.size = size
        self.color = QColor(color)

        self.velocity = 0
        self.gravity = gravity
        self.jump_velocity_val = jump_velocity
        self.rotation_multiplier = rotation_multiplier

    def move(self):
        self.velocity += self.gravity
        self.y += self.velocity

        if self.y < 0:
            self.y = 0
            self.velocity = 0

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

    def draw(self, painter, bird_pixmap):
        painter.save()
        center_x = self.x + self.size / 2
        center_y = self.y + self.size / 2

        rotation_angle = self.velocity * self.rotation_multiplier

        if rotation_angle < -30:
            rotation_angle = -30
        if rotation_angle > 90:
            rotation_angle = 90

        painter.translate(center_x, center_y)
        painter.rotate(rotation_angle)

        offset = -self.size / 2
        painter.drawPixmap(int(offset), int(offset), self.size, self.size,
bird_pixmap)
        painter.restore()
```

```
    def get_rect(self):
        return QRectF(self.x, self.y, self.size * 0.8, self.size * 0.7)
```

Роль: Сущность Игрока (Entity).

- **Задачи:**
  1. Хранение координат и скорости птицы.
  2. Реализация физики (влияние гравитации, импульс прыжка).
  3. Отрисовка с учетом вращения (клюв вверх/вниз).
- **Проблемы и Решения:**
  - *Проблема:* При вращении (`painter.rotate`) птица улетала за пределы экрана.
  - *Причина:* Неправильный порядок матричных преобразований.
  - *Решение:* Сначала перенос начала координат в центр птицы (`translate`), затем поворот (`rotate`), затем отрисовка со смещением назад (`drawPixmap(-w/2, -h/2)`).
  - *Проблема:* Жестко зашитые параметры физики.
  - *Решение:* Все параметры (`gravity` , `jump_velocity`) теперь передаются в конструктор из `settings.json` .

```
#game_area.py
import math
import sys

from PyQt6.QtCore import QSettings, Qt, QTimer, pyqtSignal
from PyQt6.QtGui import QColor, QFont, QPainter
from PyQt6.QtWidgets import QPushButton, QSizePolicy, QWidget

from bird import Bird

# Не забываем про импорт load_style_from_json, если он используется ниже
from json_reader import (
    load_settings_from_json,
    load_style_from_json,
    load_themes_from_json,
)
from pipe import Pipe
from sound_manager import SoundManager


class GameArea(QWidget):
    score_updated = pyqtSignal(int, int)
```

```
    def __init__(self):
        super().__init__()

        self.config = load_settings_from_json("settings.json")
        self.sounds = SoundManager(self.config)

        self.SCREEN_WIDTH = self.config["window"]["screen_width"]

        self.hover_frame = 0
        self.ready_to_start = False
        self.settings = QSettings("MyApp", "FlappyBird")
        self.high_score = int(self.settings.value("high_score", 0))
        self.score = 0

        # --- ВЕРТИКАЛЬНАЯ РЕЗИНОВОСТЬ ---
        self.setFixedWidth(self.SCREEN_WIDTH)
        self.setSizePolicy(QSizePolicy.Policy.Fixed,
QSizePolicy.Policy.Expanding)

        self.setFocusPolicy(Qt.FocusPolicy.StrongFocus)
        self.themes = load_themes_from_json("themes.json")
        self.current_theme_index = 0

        if self.themes:
            self.current_theme = self.themes[self.current_theme_index]
            # УДАЛЕНА ОШИБОЧНАЯ СТРОКА: self.current_theme.load_assets()
        else:
            print("Критическая ошибка: Темы не загружены!")
            sys.exit()

        self.timer = QTimer()
        self.timer.timeout.connect(self.update_game)
        self.timer.start(1000 // self.config["gameplay"]["fps"])

        b_conf = self.config["bird"]
        self.bird = Bird(
            x=b_conf["start_x"],
            y=b_conf["start_y"],
            size=b_conf["size"],
            gravity=b_conf["gravity"],
            jump_velocity=b_conf["jump_velocity"],
            rotation_multiplier=b_conf["rotation_multiplier"],
        )

        self.game_active = True
        self.is_game_over = False
```

```
        self.pipes = []
        self.spawn_timer = QTimer()
        self.spawn_timer.timeout.connect(self.spawn_pipe)
        self.spawn_timer.start(self.config["gameplay"]["spawn_interval"])

        self.restart_btn = QPushButton("Start Game", self)
        self.restart_btn.resize(200, 50)
        self.restart_btn.move(self.SCREEN_WIDTH // 2 - 100, 350)

        try:
            self.restart_btn.setStyleSheet(load_style_from_json("style.json"))
        except:
            pass
        self.restart_btn.clicked.connect(self.reset_game)
        self.restart_btn.show()

        self.timer.stop()
        self.spawn_timer.stop()
        self.game_active = False
        self.reset_game()

    def set_theme(self, index):
        if 0 <= index < len(self.themes):
            # УДАЛЕНЫ ОШИБОЧНЫЕ СТРОКИ load_assets/unload_assets
            self.current_theme_index = index
            self.current_theme = self.themes[index]
            self.update()
            if self.window():
                self.window().update()

    def reset_game(self):
        self.game_active = False
        self.ready_to_start = True
        self.is_game_over = False
        self.pipes.clear()

        self.bird.y = self.config["bird"]["start_y"]
        self.bird.velocity = 0
        self.score = 0
        self.score_updated.emit(self.score, self.high_score)
        self.restart_btn.hide()
        self.setFocus()

        self.timer.start(1000 // self.config["gameplay"]["fps"])
        self.spawn_timer.stop()
```

```
        self.sounds.stop_music()

    def spawn_pipe(self):
        # Трубы создаем с учетом текущей высоты виджета (self.height())
        new_pipe = Pipe(self.SCREEN_WIDTH, self.height(), self.config["pipe"])
        self.pipes.append(new_pipe)

    def keyPressEvent(self, event):
        if event.key() == Qt.Key.Key_Space:
            if self.ready_to_start:
                self.ready_to_start = False
                self.game_active = True
                self.spawn_timer.start(self.config["gameplay"]
["spawn_interval"])
                self.sounds.start_music()
                self.bird.jump()
                self.sounds.play_jump()
            elif self.game_active:
                self.bird.jump()
                self.sounds.play_jump()

    def update_game(self):
        if self.ready_to_start:
            self.hover_frame += 0.1
            self.bird.y = (
                self.config["bird"]["start_y"] + math.sin(self.hover_frame) *
10
            )
            self.update()
            return

        self.bird.move()

        if self.game_active:
            if self.check_collisions():
                pass

            for pipe in self.pipes:
                pipe.move()
                if not pipe.passed and pipe.x + pipe.width < self.bird.x:
                    self.score += 1
                    pipe.passed = True
                    self.sounds.play_score()
                    if self.score > self.high_score:
                        self.high_score = self.score
                        self.settings.setValue("high_score", self.high_score)
```

```
                    self.score_updated.emit(self.score, self.high_score)
            self.pipes = [p for p in self.pipes if p.x + p.width > 0]
        else:
            self.check_collisions()
        self.update()

    def paintEvent(self, event):
        painter = QPainter(self)
        painter.setRenderHint(QPainter.RenderHint.Antialiasing)

        # Фон рисуем на всю высоту виджета
        painter.drawPixmap(
            0, 0, self.width(), self.height(),
self.current_theme.background_img
        )

        pipe_texture = self.current_theme.pipe_img
        for pipe in self.pipes:
            pipe.draw(painter, pipe_texture)
        bird_texture = self.current_theme.bird_img
        self.bird.draw(painter, bird_texture)

        # Центрируем текст по вертикали динамически
        center_y = self.height() // 2

        if self.ready_to_start:
            font = QFont("Arial", 20, QFont.Weight.Bold)
            painter.setFont(font)
            text = "Press SPACE to Start"
            text_y = center_y + 50
            painter.setPen(QColor("black"))
            painter.drawText(
                2, text_y + 2, self.width(), 50, Qt.AlignmentFlag.AlignCenter,
text
            )
            painter.setPen(QColor("white"))
            painter.drawText(
                0, text_y, self.width(), 50, Qt.AlignmentFlag.AlignCenter,
text
            )

        if self.is_game_over and not self.timer.isActive():
            font = QFont("Arial", 40, QFont.Weight.Bold)
            painter.setFont(font)
            text_y = center_y - 100
            painter.setPen(QColor("black"))
```

```
            painter.drawText(
                2,
                text_y + 2,
                self.width(),
                50,
                Qt.AlignmentFlag.AlignCenter,
                "GAME OVER",
            )
            painter.setPen(QColor("red"))
            painter.drawText(
                0, text_y, self.width(), 50, Qt.AlignmentFlag.AlignCenter,
"GAME OVER"
            )

        painter.end()

    def check_collisions(self):
        bird_rect = self.bird.get_rect()

        # Пол вычисляем динамически
        floor_limit = self.height() - self.bird.size

        if self.bird.y >= floor_limit:
            self.bird.y = floor_limit
            if not self.is_game_over:
                self.handle_game_over()
            self.timer.stop()
            return False

        if self.game_active:
            for pipe in self.pipes:
                top_rect, bottom_rect = pipe.get_rects()
                if bird_rect.intersects(top_rect) or
bird_rect.intersects(bottom_rect):
                    self.handle_game_over()
                    return True
        return False

    def handle_game_over(self):
        if self.is_game_over:
            return
        self.is_game_over = True
        self.game_active = False
        self.spawn_timer.stop()
        self.sounds.play_hit()
        self.sounds.stop_music()
```

```
        self.restart_btn.setText("Restart")
        self.restart_btn.show()
```

Роль: Игровой движок (Game Engine / Core Logic).

- **Задачи:**
  1. Управление игровым циклом (`QTimer` на 60 FPS).
  2. Управление состояниями игры: `Ready` (меню), `Active` (игра), `GameOver`.
  3. Отрисовка всех объектов (`bird` , `pipes` , `background`) через `QPainter` .
  4. Обработка физики и коллизий (столкновений).
  5. Управление музыкой и звуками (через `SoundManager`).
- **Проблемы и Решения:**
  - *Проблема:* Аудиодрайвер Windows падал с ошибкой `AUDCLNT_E_DEVICE_INVALIDATED` .
  - *Причина:* При столкновении метод `check_collisions` вызывался многократно за доли секунды, спамя команду `play_hit()` .
  - *Решение:* Введен метод `handle_game_over` с "Guard Clause" (`if self.is_game_over: return`), гарантирующий, что логика смерти и звука сработает ровно 1 раз.
  - *Проблема:* Птица застревала в воздухе при смерти или проваливалась сквозь пол.
  - *Решение:* Разделена логика остановки таймеров. При ударе о трубу останавливается спавн труб (`spawn_timer`), но главный таймер (`self.timer`) продолжает работать, чтобы гравитация дотянула птицу до пола. Полная остановка происходит только при касании земли.
  - *Проблема:* Окно изменяет размер, а игра остается маленькой.
  - *Решение:* Использован `QSizePolicy.Expanding` для высоты и динамический расчет пола (`self.height() - bird.size`) вместо констант.

```
#json_reader.py
import json

from theme import Theme


def load_style_from_json(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            data = json.load(f)
```

```
        # Превращаем JSON словарь в строку CSS
        stylesheet = ""
        for selector, properties in data.items():
            stylesheet += f"{selector} {{\n"
            for key, value in properties.items():
                stylesheet += f"    {key}: {value};\n"
            stylesheet += "}\n"

        return stylesheet
    except Exception as e:
        print(f"Ошибка загрузки стилей: {e}")
        return ""


def load_themes_from_json(file_path):
    themes_list = []
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            data = json.load(f)

        for item in data:
            # Создаем объект Theme, используя данные из словаря
            new_theme = Theme(
                name=item["name"],
                bg_path=item["bg_path"],
                bird_path=item["bird_path"],
                pipe_path=item["pipe_path"],
                text_color_hex=item["text_color_hex"],
            )
            themes_list.append(new_theme)

    except Exception as e:
        print(f"Ошибка загрузки тем: {e}")

        default_theme = Theme(
            "Backup",
            "assets/images/background.png",
            "assets/images/bird.png",
            "assets/images/pipe.png",
            "#FFFFFF",
        )
        themes_list.append(default_theme)

    return themes_list
```

```
def load_settings_from_json(file_path):
    try:
        with open(file_path, "r", encoding="utf-8") as f:
            return json.load(f)
    except Exception as e:
        print(f"Ошибка загрузки настроек: {e}. Используются значения по
умолчанию.")
        return {
            "window": {
                "screen_width": 400,
                "screen_height": 600,
                "app_min_width": 1000,
                "app_min_height": 700,
            },
            "audio": {
                "music_volume": 0.5,
                "sfx_volume": 1.0,
                "music_muted": False,
                "sfx_muted": False,
            },
            "gameplay": {"fps": 60, "spawn_interval": 1500},
            "bird": {
                "start_x": 50,
                "start_y": 200,
                "size": 30,
                "gravity": 0.5,
                "jump_velocity": -7,
                "rotation_multiplier": 3,
            },
            "pipe": {"width": 60, "speed": 3, "gap_size": 150},
        }


def save_settings_to_json(file_path, data):
    try:
        with open(file_path, "w", encoding="utf-8") as f:
            json.dump(data, f, indent=4)
        print("Настройки сохранены.")
    except Exception as e:
        print(f"Ошибка сохранения настроек: {e}")
```

Роль: Утилита работы с данными (Data Access Layer).

- **Задачи:**

1. Чтение настроек, тем и стилей из JSON файлов.
2. Сохранение измененных настроек обратно в файл.
3. Обеспечение "безопасности": если файла нет, возвращает дефолтный словарь, чтобы игра не упала.

```
#main.py
import sys

from PyQt6.QtCore import Qt
from PyQt6.QtGui import QColor, QPainter
from PyQt6.QtWidgets import (
    QApplication,
    QFrame,
    QHBoxLayout,
    QLabel,
    QMainWindow,
    QPushButton,
    QScrollArea,
    QSizePolicy,
    QVBoxLayout,
    QWidget,
)

from game_area import GameArea
from json_reader import (
    load_settings_from_json,
    load_style_from_json,
    save_settings_to_json,
)
from settings_dialog import SettingsDialog
from theme_card import ThemeCard


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Flappy Bird")

        # 1. ЗАГРУЗКА НАСТРОЕК
        config = load_settings_from_json("settings.json")

        safe_min_width = 1100

        json_min_w = config["window"].get("app_min_width", 1000)
        min_w = max(safe_min_width, json_min_w)
```

```
        min_h = config["window"].get("app_min_height", 700)

        self.setMinimumSize(min_w, min_h)

        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        # Главный горизонтальный слой
        main_layout = QHBoxLayout(central_widget)
        main_layout.setContentsMargins(0, 0, 0, 0)
        main_layout.setSpacing(0)

        # --- СОЗДАНИЕ ИГРЫ ---
        self.game_area = GameArea()
        self.game_area.score_updated.connect(self.update_labels)

        # ========================================
        # ЛЕВАЯ ПАНЕЛЬ (Статистика + Настройки)
        # ========================================
        left_panel_widget = QWidget()
        left_panel_widget.setFixedWidth(250)
        left_panel_widget.setStyleSheet("background-color: rgba(0, 0, 0,
150);")

        left_panel_layout = QVBoxLayout(left_panel_widget)
        left_panel_layout.setContentsMargins(20, 50, 20, 20)
        left_panel_layout.setSpacing(10)

        # SCORE
        self.label_score_title = QLabel("SCORE")
        self.label_score_title.setStyleSheet(
            "color: white; font-size: 24px; font-weight: bold; background:
transparent;"
        )
        self.label_score_title.setAlignment(Qt.AlignmentFlag.AlignCenter)
        left_panel_layout.addWidget(self.label_score_title)

        self.label_score_val = QLabel("0")
        self.label_score_val.setStyleSheet(
            "color: #E0C068; font-size: 60px; font-weight: bold; background:
transparent;"
        )
        self.label_score_val.setAlignment(Qt.AlignmentFlag.AlignCenter)
        left_panel_layout.addWidget(self.label_score_val)

        # Линия
```

```
        line = QFrame()
        line.setFrameShape(QFrame.Shape.HLine)
        line.setStyleSheet("color: white;")
        left_panel_layout.addWidget(line)

        # BEST
        self.label_best_title = QLabel("BEST")
        self.label_best_title.setStyleSheet(
            "color: white; font-size: 24px; font-weight: bold; background:
transparent;"
        )
        self.label_best_title.setAlignment(Qt.AlignmentFlag.AlignCenter)
        left_panel_layout.addWidget(self.label_best_title)

        self.label_best_val = QLabel(str(self.game_area.high_score))
        self.label_best_val.setStyleSheet(
            "color: #E0C068; font-size: 40px; font-weight: bold; background:
transparent;"
        )
        self.label_best_val.setAlignment(Qt.AlignmentFlag.AlignCenter)
        left_panel_layout.addWidget(self.label_best_val)

        # --- КНОПКА SETTINGS (Прямо под счетом) ---
        left_panel_layout.addSpacing(20)  # Отступ

        self.settings_btn = QPushButton("Settings")
        self.settings_btn.setFixedHeight(45)  # Чуть повыше
        try:

self.settings_btn.setStyleSheet(load_style_from_json("style.json"))
        except:
            pass

        # Подключаем сигнал (Только один раз!)
        self.settings_btn.clicked.connect(self.open_settings)

        left_panel_layout.addWidget(self.settings_btn)

        # Пружина внизу (теперь она ПОСЛЕ кнопки, значит кнопка прижмется
вверх)
        left_panel_layout.addStretch(1)

        # ========================================
        # ПРАВАЯ ПАНЕЛЬ (Темы)
        # ========================================
        right_panel_widget = QWidget()
```

```
        right_panel_widget.setFixedWidth(250)
        right_panel_widget.setStyleSheet("background-color: rgba(0, 0, 0,
150);")

        self.right_layout = QVBoxLayout(right_panel_widget)
        self.right_layout.setContentsMargins(20, 20, 20, 20)

        title_lbl = QLabel("THEMES")
        title_lbl.setStyleSheet(
            "color: white; font-weight: bold; font-size: 20px; background:
transparent;"
        )
        title_lbl.setAlignment(Qt.AlignmentFlag.AlignCenter)
        self.right_layout.addWidget(title_lbl)

        scroll = QScrollArea()
        scroll.setWidgetResizable(True)
        scroll.setStyleSheet("background: transparent; border: none;")

        scroll_content = QWidget()
        scroll_content.setStyleSheet("background: transparent;")
        self.cards_layout = QVBoxLayout(scroll_content)
        self.cards_layout.setSpacing(20)

        self.theme_cards = []
        for i, theme in enumerate(self.game_area.themes):
            is_active = i == self.game_area.current_theme_index
            card = ThemeCard(theme, i, is_active)
            card.clicked.connect(self.on_theme_selected)
            self.cards_layout.addWidget(card)
            self.theme_cards.append(card)

        self.cards_layout.addStretch(1)
        scroll.setWidget(scroll_content)
        self.right_layout.addWidget(scroll)

        # ========================================
        # СБОРКА ГЛАВНОГО СЛОЯ
        # ========================================

        # 1. Левая панель
        main_layout.addWidget(left_panel_widget)

        # 2. Пружина слева (Толкает игру к центру)
        main_layout.addStretch(1)
```

```
        # 3. ИГРА (По центру)
        main_layout.addWidget(self.game_area)
        # 4. Пружина справа (Толкает игру к центру)
        main_layout.addStretch(1)

        # 5. Правая панель
        main_layout.addWidget(right_panel_widget)

    def on_theme_selected(self, index):
        self.game_area.set_theme(index)
        for i, card in enumerate(self.theme_cards):
            card.set_selected(i == index)
        self.update()

    def update_labels(self, score, high_score):
        self.label_best_val.setText(str(high_score))
        self.label_score_val.setText(str(score))

    def open_settings(self):
        # Создаем и открываем диалог
        dialog = SettingsDialog(self, self.game_area.sounds,
self.game_area.config)

        if dialog.exec():
            save_settings_to_json("settings.json", self.game_area.config)

        self.game_area.setFocus()

    def paintEvent(self, event):
        painter = QPainter(self)
        if hasattr(self, "game_area") and hasattr(self.game_area,
"current_theme"):
            blurred_pixmap = self.game_area.current_theme.blurred_bg
            painter.setRenderHint(QPainter.RenderHint.SmoothPixmapTransform)
            painter.drawPixmap(self.rect(), blurred_pixmap)
            overlay_color = QColor(0, 0, 0, 100)
            painter.fillRect(self.rect(), overlay_color)
        painter.end()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

Роль: Точка входа (Entry Point) и Главный Контейнер (View/Controller).

- **Задачи:**
  1. Инициализация приложения (`QApplication`).
  2. Создание главного окна (`MainWindow`).
  3. Настройка глобальной верстки (Layout Management): разделение экрана на Левую панель (счет), Центр (игра) и Правую панель (темы).
  4. Загрузка глобальных настроек окна (минимальный размер) из JSON.
  5. Связывание компонентов: подключение сигналов от `GameArea` (обновление счета) и кнопок интерфейса.
- **Проблемы и Решения:**
  - *Проблема:* При изменении размера окна игровая область смещалась, а боковые панели "наезжали" на игру.
  - *Решение:* Использована комбинация `QHBoxLayout` с пружинами (`addStretch(1)`) по бокам от игры. Установлен `safe_min_width = 1100` , чтобы физически запретить окну сжиматься сильнее, чем позволяет контент.
  - *Проблема:* Диалоговое окно настроек открывалось дважды.
  - *Решение:* Удалено дублирование строки подключения сигнала `clicked.connect` .
  - *Проблема:* Ресурсы (картинки/конфиги) не находились при сборке в EXE.
  - *Решение:* (В теории) использование функции `resource_path` для путей, но в финальной версии решили просто копировать папку `assets` рядом с exe.

```
#pipe.py
import random

from PyQt6.QtCore import QRectF
from PyQt6.QtGui import QColor, QTransform


class Pipe:
    def __init__(self, x, screen_height, settings):
        self.screen_height = screen_height
        self.x = x

        self.width = settings["width"]
        self.speed = settings["speed"]
        self.gap_size = settings["gap_size"]

        self.color = QColor("228B22")
        self.passed = False

        min_y = 50
```

```
        max_y = screen_height - min_y - self.gap_size
        self.gap_y = random.randint(min_y, max_y)

    def move(self):
        self.x -= self.speed

    def draw(self, painter, pipe_pixmap):
        img_w = pipe_pixmap.width()
        img_h = pipe_pixmap.height()
        flipped_pixmap = pipe_pixmap.transformed(QTransform().scale(1, -1))
        top_height = self.gap_y
        source_h = min(img_h, int(top_height))
        source_y = img_h - source_h
        painter.drawPixmap(
            int(self.x),
            0,
            self.width,
            int(top_height),
            flipped_pixmap,
            0,
            source_y,
            img_w,
            source_h,
        )

        bottom_y = self.gap_y + self.gap_size
        bottom_height = self.screen_height - bottom_y
        source_h = min(img_h, int(bottom_height))
        painter.drawPixmap(
            int(self.x),
            bottom_y,
            self.width,
            bottom_height,
            pipe_pixmap,
            0,
            0,
            img_w,
            source_h,
        )

    def get_rects(self):
        top_rect = QRectF(self.x, 0, self.width, self.gap_y)
        bottom_y = self.gap_y + self.gap_size
        bottom_height = self.screen_height - bottom_y
        bottom_rect = QRectF(self.x, bottom_y, self.width, bottom_height)
        return top_rect, bottom_rect
```

Роль: Сущность Препятствия (Entity).

- **Задачи:**
  1. Генерация пары труб (верхняя/нижняя) со случайным просветом.
  2. Движение влево.
  3. Отслеживание состояния "пройдена" (`self.passed`) для начисления очков.
  4. Предоставление `QRectF` (хитбоксов) для проверки коллизий.
- **Проблемы и Решения:**
  - *Проблема:* Верхняя труба должна смотреть вниз.
  - *Решение:* Использование `QTransform().scale(1, -1)` для отражения текстуры по вертикали.
  - *Проблема:* Счет начислялся каждый кадр, пока птица внутри трубы.
  - *Решение:* Введен флаг `self.passed = False` . Очки начисляются только если `passed == False` и птица пересекла правую границу трубы.

```
#settings_dialog.py
from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QDialog,
    QHBoxLayout,
    QLabel,
    QPushButton,
    QSlider,
    QVBoxLayout,
    QWidget,
)


class SettingsDialog(QDialog):
    def __init__(self, parent, sound_manager, config):
        super().__init__(parent)
        self.setWindowTitle("Settings")
        self.setFixedSize(380, 220)  # Чуть шире для красоты
        self.sound_manager = sound_manager
        self.config = config

        # Стилизация
        self.setStyleSheet("""
            QDialog { background-color: #333; color: white; }
            QLabel { color: white; font-size: 14px; font-weight: bold; border:
none; }

            /* Слайдеры */
```

```
            QSlider::groove:horizontal {
                border: 1px solid #999; height: 8px; background: #555; margin:
2px 0; border-radius: 4px;
            }
            QSlider::handle:horizontal {
                background: #E0C068; border: 1px solid #5c5c5c; width: 18px;
margin: -2px 0; border-radius: 9px;
            }
            QSlider:disabled { background: #444; }

            /* Кнопки-иконки (прозрачные, большие) */
            QPushButton.iconBtn {
                background: transparent; border: none; font-size: 24px; text-
align: center;
            }
            QPushButton.iconBtn:hover { color: #E0C068; }

            /* Обычные кнопки */
            QPushButton.actionBtn {
                background-color: #E0C068; border-radius: 5px; font-weight:
bold; color: #333;
            }
            QPushButton.actionBtn:hover { background-color: #F0D078; }
        """)

        layout = QVBoxLayout(self)
        layout.setSpacing(15)
        layout.setContentsMargins(20, 30, 20, 20)

        # --- СТРОКА МУЗЫКИ ---
        layout.addLayout(
            self.create_slider_row(
                "Music",
                self.sound_manager.music_vol,
                self.sound_manager.is_music_muted,
                self.toggle_music_mute,
                self.update_music_vol,
                "btn_music",
            )
        )

        # --- СТРОКА ЭФФЕКТОВ ---
        layout.addLayout(
            self.create_slider_row(
                "SFX",
                self.sound_manager.sfx_vol,
```

```
                self.sound_manager.is_sfx_muted,
                self.toggle_sfx_mute,
                self.update_sfx_vol,
                "btn_sfx",
            )
        )

        layout.addStretch(1)

        # --- КНОПКА ОК ---
        btn_layout = QHBoxLayout()
        ok_btn = QPushButton("ОК")
        ok_btn.setProperty("class", "actionBtn")  # Для CSS
        ok_btn.setFixedSize(100, 40)
        ok_btn.clicked.connect(self.accept)
        btn_layout.addWidget(ok_btn)
        layout.addLayout(btn_layout)

    def create_slider_row(
        self, label_text, current_vol, is_muted, mute_handler, vol_handler,
btn_name
    ):
        """Вспомогательный метод для создания ровной строки"""
        row = QHBoxLayout()

        # 1. Кнопка Mute (Иконка)
        btn = QPushButton("🔇" if is_muted else "🔊")
        btn.setObjectName("iconBtn")  # Стиль
        btn.setFixedWidth(40)
        btn.clicked.connect(mute_handler)
        # Сохраняем ссылку на кнопку в self, чтобы менять иконку позже
        setattr(self, btn_name, btn)
        row.addWidget(btn)

        # 2. Текст (Фиксированная ширина = ровные слайдеры!)
        lbl = QLabel(label_text)
        lbl.setFixedWidth(60)  # <--- ВОТ РЕШЕНИЕ ПРОБЛЕМЫ С ДЛИНОЙ
        row.addWidget(lbl)

        # 3. Слайдер
        slider = QSlider(Qt.Orientation.Horizontal)
        slider.setRange(0, 100)
        slider.setValue(int(current_vol * 100))
        slider.valueChanged.connect(vol_handler)

        # Сохраняем ссылку на слайдер (slider_music или slider_sfx)
```

```
        setattr(self, f"slider_{label_text.lower()}", slider)

        row.addWidget(slider)
        return row

    # --- ЛОГИКА ---

    def toggle_music_mute(self):
        new_state = not self.sound_manager.is_music_muted
        self.sound_manager.mute_music(new_state)
        self.config["audio"]["music_muted"] = new_state

        # Обновляем UI
        self.btn_music.setText("🔇" if new_state else "🔊")
        self.slider_music.setEnabled(not new_state)

    def toggle_sfx_mute(self):
        new_state = not self.sound_manager.is_sfx_muted
        self.sound_manager.mute_sfx(new_state)
        self.config["audio"]["sfx_muted"] = new_state

        # Обновляем UI
        self.btn_sfx.setText("🔇" if new_state else "🔊")
        self.slider_sfx.setEnabled(not new_state)

    def update_music_vol(self, value):
        vol = value / 100.0
        self.sound_manager.set_music_volume(vol)
        self.config["audio"]["music_volume"] = vol

    def update_sfx_vol(self, value):
        vol = value / 100.0
        self.sound_manager.set_sfx_volume(vol)
        self.config["audio"]["sfx_volume"] = vol
```

Роль: Окно настроек (Modal View).

- **Задачи:**
  1. Отображение слайдеров громкости.
  2. Реализация кнопок Mute с иконками (🔊 / 🔇).
  3. Обновление `settings.json` при закрытии.
- **Проблемы и Решения:**
  - *Проблема:* Слайдеры были разной длины из-за разной длины слов "Music" и

"SFX".

- *Решение:* Использован `setFixedWidth(60)` для текстовых меток, чтобы выровнять верстку по сетке.

```
#sound_manager.py
import os

from PyQt6.QtCore import QObject, QUrl
from PyQt6.QtMultimedia import QAudioOutput, QMediaPlayer, QSoundEffect


class SoundManager(QObject):
    def __init__(self, config):
        super().__init__()

        base_dir = os.path.dirname(os.path.abspath(__file__))
        self.sound_dir = os.path.join(base_dir, "assets", "sounds")

        audio_cfg = config.get("audio", {})
        self.music_vol = audio_cfg.get("music_volume", 0.5)
        self.sfx_vol = audio_cfg.get("sfx_volume", 1.0)

        # НОВОЕ: Загружаем состояние Mute (по умолчанию False - звук включен)
        self.is_music_muted = audio_cfg.get("music_muted", False)
        self.is_sfx_muted = audio_cfg.get("sfx_muted", False)

        # --- 1. SFX (Короткие звуки) ---
        self.jump_sfx = self._load_sfx("jump.wav")
        self.score_sfx = self._load_sfx("score.wav")
        self.hit_sfx = self._load_sfx("hit.wav")

        # Применяем начальное состояние mute для SFX
        self.mute_sfx(self.is_sfx_muted)

        # --- 2. МУЗЫКА (Длинный трек) ---
        self.music_player = QMediaPlayer(self)
        self.audio_output = QAudioOutput(self)

        self.music_player.setAudioOutput(self.audio_output)
        self.audio_output.setVolume(self.music_vol)
        # Применяем начальное состояние mute для музыки
        self.audio_output.setMuted(self.is_music_muted)

        music_path = os.path.join(self.sound_dir, "music.wav")
        if os.path.exists(music_path):
```

```
            self.music_player.setSource(QUrl.fromLocalFile(music_path))
            self.music_player.setLoops(QMediaPlayer.Loops.Infinite)
        else:
            print(f"Warning: Music file not found: {music_path}")

    def _load_sfx(self, filename):
        path = os.path.join(self.sound_dir, filename)
        if not os.path.exists(path):
            return None
        effect = QSoundEffect(self)
        effect.setSource(QUrl.fromLocalFile(path))
        # Громкость ставим базовую, mute наложится позже
        effect.setVolume(self.sfx_vol)
        return effect

    # --- МЕТОДЫ УПРАВЛЕНИЯ ГРОМКОСТЬЮ И MUTE ---

    def set_music_volume(self, volume):
        self.music_vol = volume
        self.audio_output.setVolume(volume)

    def set_sfx_volume(self, volume):
        self.sfx_vol = volume
        # Если звук не заглушен, применяем новую громкость.
        # Если заглушен - только запоминаем её, но оставляем реальную
громкость 0.
        if not self.is_sfx_muted:
            if self.jump_sfx:
                self.jump_sfx.setVolume(volume)
            if self.score_sfx:
                self.score_sfx.setVolume(volume)
            if self.hit_sfx:
                self.hit_sfx.setVolume(volume)

    def mute_music(self, mute: bool):
        """Включает/выключает звук музыки"""
        self.is_music_muted = mute
        self.audio_output.setMuted(mute)

    def mute_sfx(self, mute: bool):
        """Включает/выключает звуковые эффекты"""
        self.is_sfx_muted = mute
        # Для QSoundEffect нет встроенного mute, поэтому ставим громкость в 0
или восстанавливаем
        target_vol = 0.0 if mute else self.sfx_vol
```

```
        if self.jump_sfx:
            self.jump_sfx.setVolume(target_vol)
        if self.score_sfx:
            self.score_sfx.setVolume(target_vol)
        if self.hit_sfx:
            self.hit_sfx.setVolume(target_vol)

    # --- МЕТОДЫ ВОСПРОИЗВЕДЕНИЯ ---

    def play_jump(self):
        if self.jump_sfx:
            self.jump_sfx.play()

    def play_score(self):
        if self.score_sfx:
            self.score_sfx.play()

    def play_hit(self):
        if self.hit_sfx:
            if not self.hit_sfx.isPlaying():
                self.hit_sfx.play()

    def start_music(self):
        if self.music_player.playbackState() !=
QMediaPlayer.PlaybackState.PlayingState:
            self.music_player.play()

    def stop_music(self):
        if self.music_player.playbackState() ==
QMediaPlayer.PlaybackState.PlayingState:
            self.music_player.stop()
```

Роль: Аудио-подсистема (Audio Engine).

- **Задачи:**
  1. Загрузка и воспроизведение звуков (SFX) и музыки.
  2. Управление громкостью и режимом "Mute" (без звука).
- **Проблемы и Решения:**
  - *Проблема:* Игра вылетала при воспроизведении музыки.
  - *Причина:* `QSoundEffect` плохо справляется с длинными треками на Windows + конфликты со сборщиком мусора Python.
  - *Решение:*

1. Разделили: Музыка играет через `QMediaPlayer` (потоковое), эффекты через `QSoundEffect` (в памяти).
2. Класс унаследован от `QObject`, а плееры создаются с родителем (`self`), чтобы Python не удалял их из памяти случайно.

- *Проблема:* Звук прыжка пропадал при частых нажатиях.
- *Решение:* Убрали вызов `.stop()` перед `.play()` для эффектов. Теперь звуки накладываются друг на друга (полифония), что стабильнее.

```
#theme.py
from PyQt6.QtCore import Qt
from PyQt6.QtGui import QColor, QPainter, QPixmap


class Theme:
    def __init__(self, name, bg_path, bird_path, pipe_path, text_color_hex):
        self.name = name

        self.background_img = QPixmap(bg_path)
        self.bird_img = QPixmap(bird_path)
        self.pipe_img = QPixmap(pipe_path)

        self.text_color = QColor(text_color_hex)

        small_w = self.background_img.width() // 10
        small_h = self.background_img.height() // 10

        self.blurred_bg = self.background_img.scaled(
            small_w,
            small_h,
            Qt.AspectRatioMode.IgnoreAspectRatio,
            Qt.TransformationMode.SmoothTransformation,
        )

        self.preview_img = self.generate_preview(140, 200)

    def generate_preview(self, w, h):
        preview = QPixmap(w, h)
        painter = QPainter(preview)

        painter.drawPixmap(0, 0, w, h, self.background_img)
        scaled_pipe = self.pipe_img.scaledToWidth(int(w * 0.3))
        pipe_x = w - scaled_pipe.width() - 10
        pipe_y = h - scaled_pipe.height() + 50
        painter.drawPixmap(pipe_x, pipe_y, scaled_pipe)
```

```
        scaled_bird = self.bird_img.scaledToWidth(int(w * 0.3))
        bird_x = (w - scaled_bird.width()) // 2
        bird_y = (h - scaled_bird.height()) // 2
        painter.drawPixmap(bird_x, bird_y, scaled_bird)

        painter.end()

        return preview
```

Роль: Менеджер ресурсов темы (Asset Manager).

- **Задачи:**
  1. Загрузка изображений (фон, птица, труба) с диска.
  2. Генерация **размытого фона** (Blur) для заполнения боковых полей.
  3. Генерация **превью** (Thumbnail) для меню выбора тем.
- **Проблемы и Решения:**
  - *Проблема:* Размытие (Gaussian Blur) слишком тяжелое для процессора.
  - *Решение:* Хак с масштабированием (Downscale -> Upscale). Картинка уменьшается в 10 раз, а потом растягивается с `SmoothTransformation`. Это дает красивое размытие мгновенно.
  - *Проблема:* Ошибка `AttributeError: 'Theme' object has no attribute 'load_assets'`.
  - *Решение:* Упростили класс, убрав сложную систему Lazy Loading, так как она вызывала конфликты. Теперь ресурсы грузятся при инициализации (стабильность важнее микро-оптимизации на данном этапе).

```
#theme_card.py
from PyQt6.QtCore import Qt, pyqtSignal
from PyQt6.QtGui import QColor, QPainter
from PyQt6.QtWidgets import QWidget


class ThemeCard(QWidget):
    clicked = pyqtSignal(int)

    def __init__(self, theme, index, is_selected=False):
        super().__init__()
        self.theme = theme
        self.index = index
        self.is_selected = is_selected
        self.setFixedSize(140, 200)
```

```
    def set_selected(self, selected):
        self.is_selected = selected
        self.update()  # Перерисовать рамку

    def mousePressEvent(self, event):
        if event.button() == Qt.MouseButton.LeftButton:
            self.clicked.emit(self.index)

    def paintEvent(self, event):
        painter = QPainter(self)

        # 1. Рисуем сгенерированное превью
        painter.drawPixmap(0, 0, self.width(), self.height(),
self.theme.preview_img)

        # 2. Если выбрано — рисуем жирную рамку
        if self.is_selected:
            pen = painter.pen()
            pen.setColor(QColor("#FFD700"))  # Золотой цвет
            pen.setWidth(6)
            painter.setPen(pen)
            # Рисуем прямоугольник внутри границ
            painter.drawRect(3, 3, self.width() - 6, self.height() - 6)

        painter.end()
```

Роль: Кастомный виджет интерфейса (Custom UI Widget).

- **Задачи:**
  1. Отрисовка миниатюры темы в правой панели.
  2. Отрисовка золотой рамки, если тема выбрана.
  3. Обработка клика мыши и отправка сигнала `clicked`.
- **Решение:** Наследование от `QWidget` и переопределение `paintEvent` и `mousePressEvent`. Это позволяет создать кнопку любого вида, не ограничиваясь стандартным `QPushButton`.

JSON

```
#settings.json
{
    "window": {
        "screen_width": 400,
```

```
        "screen_height": 600,
        "app_min_width": 1100,
        "app_min_height": 700
    },
    "audio": {
        "music_volume": 0.35,
        "sfx_volume": 0.12,
        "sfx_muted": false,
        "music_muted": false
    },
    "gameplay": {
        "fps": 60,
        "spawn_interval": 1500
    },
    "bird": {
        "start_x": 50,
        "start_y": 200,
        "size": 30,
        "gravity": 0.5,
        "jump_velocity": -7.0,
        "rotation_multiplier": 3.0
    },
    "pipe": {
        "width": 60,
        "speed": 3,
        "gap_size": 150
    }
}
```

Роль: Главный Конфигурационный Файл (Data-Driven Design). Хранит все изменяемые параметры игры, физики и пользовательских настроек (громкость, mute).

- **Задачи:**
  1. **Настройки Окна (`"window"`):** Задает базовые и минимальные размеры окна, обеспечивая стабильность верстки.
     - *Пример:* `"app_min_width": 1100` используется в `main.py` для предотвращения сбоев UI.
  2. **Настройки Аудио (`"audio"`):** Хранит состояние пользователя для звука.
     - *Пример:* `"music_volume": 0.5` , `"sfx_muted": true` .
  3. **Настройки Игры (`"gameplay"`):** Параметры игрового цикла.
     - *Пример:* `"fps": 60` (частота обновления), `"spawn_interval": 1500` (как часто появляются трубы).

4. **Настройки Физики (`"bird"` , `"pipe"`):** Параметры, определяющие баланс и сложность игры.
    - *Пример:* `"gravity": 0.5` , `"jump_velocity": -7.0` .

- **Проблемы и Решения:**
    - *Проблема:* Изначально, минимальный размер окна был жестко задан в коде (`main.py`).
    - *Решение:* Размер перенесен в JSON. Код (`main.py`) теперь читает этот параметр, но при запуске сравнивает его с "безопасным" минимумом (1100px) для предотвращения поломки верстки.
    - *Проблема:* Отсутствие настроек по умолчанию при первом запуске.
    - *Решение:* Модуль `json_reader.py` написан так, что при ошибке чтения (например, файл не найден), он возвращает словарь с разумными значениями по умолчанию.
- **Связь с кодом:** Читается модулями `main.py` , `game_area.py` , `bird.py` , `pipe.py` (для балансировки). **Записывается** модулем `settings_dialog.py` при сохранении настроек.

```
#style.json
{
  "QPushButton": {
    "background-color": "#E0C068",
    "border": "2px solid #555",
    "border-radius": "10px",
    "font-size": "20px",
    "font-weight": "bold",
    "color": "#333"
  },
  "QPushButton:hover": {
    "background-color": "#F0D078"
  }
}
```

Роль: Менеджер Стилей (CSS/QSS-Wrapper). Хранит кастомные стили для виджетов, не требующие сложной логики.

- **Задачи:**
    1. **Стилизация кнопок:** Задает внешний вид интерактивных элементов (`QPushButton`).
        - *Пример:* Фон, радиус скругления, цвет текста, анимация наведения (`:hover`).

2. **Отделение дизайна:** Позволяет менять внешний вид кнопок без необходимости править код Python.

- **Проблемы и Решения:**
  - *Проблема:* PyQt6 ожидает строку QSS (Qt Style Sheet), а не JSON.
  - *Решение:* В модуле `json_reader.py` реализована функция `load_style_from_json`, которая:
    1. Читает JSON.
    2. Перебирает ключи/значения.
    3. Собирает их в единую строку формата QSS, которую затем можно передать в `widget.setStyleSheet()`.

    Python

    ```
    # Преобразование в QSS
    stylesheet = "QPushButton { background-color: #E0C068; border: ... }"
    ```
- **Связь с кодом:** Читается модулем `json_reader.py`. Строка QSS применяется к кнопкам "Start Game" в `game_area.py` и "Settings" в `main.py`.

```
#themes.json
[
  {
    "name": "Default",
    "bg_path": "assets/images/background.png",
    "bird_path": "assets/images/bird.png",
    "pipe_path": "assets/images/pipe.png",
    "text_color_hex": "#FFFFFF"
  },
  {
    "name": "New Year",
    "bg_path": "assets/images/background_new_year.jpg",
    "bird_path": "assets/images/bird_new_year.png",
    "pipe_path": "assets/images/pipe_new_year.png",
    "text_color_hex": "#FFFF00"
  },
  {
    "name": "Dragon",
    "bg_path": "assets/images/background_dragon.jpg",
    "bird_path": "assets/images/bird_dragon.png",
    "pipe_path": "assets/images/pipe_gragon.png",
    "text_color_hex": "#FFFF00"
  },
  {
    "name": "Angry Birds",
    "bg_path": "assets/images/background_angry_birds.jpg",
```

```
    "bird_path": "assets/images/bird_angry_birds.png",
    "pipe_path": "assets/images/pipe_angry_birds.png",
    "text_color_hex": "#FFFF00"
  },
  {
    "name": "Angry Birds",
    "bg_path": "assets/images/background_halloween.jpg",
    "bird_path": "assets/images/bird_halloween.png",
    "pipe_path": "assets/images/pipe_halloween.png",
    "text_color_hex": "#FFFF00"
  }
]
```

Роль: Каталог Скинов (Theme Registry). Хранит метаданные для всех доступных тем оформления игры.

- **Задачи:**
  1. **Определение тем:** Каждая запись в JSON — это отдельный объект темы.
  2. **Хранение путей:** Указывает пути к ресурсам для каждой темы.
     - *Пример:* `"bg_path": "assets/images/background_new_year.jpg"`.
  3. **Настройки стиля:** Хранит дополнительные параметры стиля, привязанные к теме.
     - *Пример:* `"text_color_hex": "#FFFF00"` (цвет текста, который должен меняться в зависимости от фона).
- **Проблемы и Решения:**
  - *Проблема:* Чтение путей с жесткого диска может быть медленным.
  - *Решение:* Модуль `json_reader.py` считывает JSON в память, а затем создает массив объектов `Theme` (`theme.py`), которые уже сами занимаются загрузкой `QPixmap` (изображений) по указанным путям. Это позволяет один раз считать список, а затем быстро работать с объектами в Python.
  - *Проблема:* Если путь к изображению указан неверно.
  - *Решение:* Класс `Theme` должен обрабатывать исключения, чтобы при ошибке загрузки просто использовать пустой `QPixmap` и не допустить падения всей программы.
- **Связь с кодом:** Читается модулем `json_reader.py` в начале работы. Передается в `GameArea` для отрисовки и в `MainWindow` для создания карточек выбора тем (`ThemeCard`).

Файл	Назначение	Читающие модули	Записывающие модули	Принцип О
`settings.json`	Конфигурация, Физика, Аудио, UI	`main.py`, `game_area.py`, `bird.py`, `pipe.py`	`settings_dialog.py`	**Инкапсуля** (отделяет изменяемы данные от логики)
`themes.json`	Каталог ресурсов и метаданных скинов	`json_reader.py`, `main.py`	-	**Композици** (позволяет легко добавлять новые объе `Theme` без изменения кода)
`style.json`	Стилизация интерфейса (QSS)	`main.py`, `game_area.py`	-	**Абстракци** (UI-дизайне может меня вид без зна Python)

Итоговая структура проекта

- Логика отделена от представления.
- Данные (настройки/темы) отделены от кода.
- Ресурсы управляются централизованно.
- Событийная модель (Сигналы) связывает разрозненные части (Игра -> Главное окно -> Настройки).