МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра геоинформатики и информационной безопасности

КУРСОВОЙ ПРОЕКТ ПО ДИСЦИПЛИНЕ

«Технологии и методы программирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

«Виртуальное пианино»

| Обучающийся группы | № 6311-100503DМ.К. Барнаева |
| --- | --- |
| Руководитель курсового проекта | Е.В. Мясников |
|  |  |
| Дата защиты \_\_\_\_\_\_\_\_\_\_\_\_\_ г.  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |

САМАРА 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра геоинформатики и информационной безопасности

**Индивидуальное задание на курсовой проект**

Студенту группы № 6311-100503DМ.К. Барнаева

Тема проекта: ***«Виртуальное пианино»***

| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты проекта | Содержание задания |
| --- | --- | --- |
| ПК-1 Способен выполнять интеграцию программных модулей и компонент, проводить верификацию программных продуктов:  ПК-1.2 Выполняет интеграцию программных модулей и компонентов, проводит верификацию выпусков программного продукта | Знать: основы и принципы интеграции программных систем и компонентов.  Уметь: проводить интеграцию различных модулей и компонентов программных систем.  Владеть: методологиями обеспечения качества программных продуктов. | Разработка архитектуры и интеграция модулей виртуального пианино  1. Изучение существующих методов организации интерактивных графических приложений и обработки звука.  2. Разработка архитектурного решения на основе системы сцен (Scene-based) для разделения режимов работы.  3. Изучение существующих библиотек Python для работы со звуком (Pygame.mixer) и графикой (Pygame).  4. Программная реализация интеграции модулей: главного управляющего класса (Game), сцен (PianoScene, FallingPianoScene, MelodyRecorderScene, LicenseScene) и вспомогательных утилит (класс кнопки, система ввода-вывода, конфигурация). |
| ПК-5 Способен осуществлять непосредственное руководство процессами разработки программного обеспечения:  ПК-5.1 Управляет конфигурациями и выпусками программного продукта;  ПК-5.3 Осуществляет руководство проверкой работоспособности программного обеспечения;  ПК-5.4 Осуществляет руководство разработкой проектной и технической документации | Знать: основные принципы управления конфигурациями и релизами программных продуктов.  Уметь: разрабатывать и внедрять стратегии управления конфигурациями.  Владеть: навыками настройки и использования систем управления конфигурациями.  Знать: основные принципы жизненного цикла разработки программного обеспечения.  Уметь: анализировать результаты тестирования и выявлять дефекты.  Владеть: навыками организации процесса тестирования.  Знать: требования к содержанию и структуре проектных и технических документов.  Уметь: анализировать технические задания и разрабатывать проектную документацию в соответствии с требованиями.  Владеть: современными программными средствами для разработки и оформления проектных и технических документов. | Управление разработкой, тестированием и документированием программного продукта  5. Разработка программы «Виртуальное пианино», реализующей функции музыкального инструмента, обучающей игры и студии записи.  6. Отладка и тестирование разработанной программы: модульное тестирование ключевых функций (воспроизведение звука, запись мелодий, игровая механика), интеграционное тестирование взаимодействия сцен, пользовательское приемочное тестирование.  7. Проведение экспериментов по проверке корректности конвертации мелодий в формат WAV.  8. Разработка отчетной документации: пояснительная записка к курсовому проекту, аннотация, исходный код программы с комментариями. |

Срок представления отчета 24 декабря 2025 г.

Руководитель курсового проекта

доцент каф. ГИиИБ, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Мясников

*(подпись)*

Задание принял к исполнению

студент группы № 6311-100503D \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.К. Барнаева

*(подпись)*

РЕФЕРАТ

**Пояснительная записка к курсовому проекту:** 61 с., 9 рисунков, 0 таблицы, 5 источников, 0 приложение.

ВИРТУАЛЬНОЕ ПИАНИНО, PYGAME, ОБУЧАЮЩАЯ ИГРА, ЗАПИСЬ МЕЛОДИЙ, АВТОРИЗАЦИЯ, JSON, WAV-ЭКСПОРТ

Цель работы — разработка интерактивного приложения-пианино с графическим интерфейсом, обучающими игровыми режимами и системой записи мелодий.

Краткое описание результатов работы: выполнен обзор существующих аналогов, предложена модульная архитектура на основе сцен, разработан алгоритм записи и воспроизведения мелодий с временными метками, реализована система авторизации по коду доступа. Разработана программа на языке Python с использованием библиотеки Pygame, включающая три основных режима: свободное пианино (3 звуковых тембра), игра с падающими нотами, запись и управление библиотекой мелодий. Программа экспортирует мелодии в форматы JSON и WAV.

Ключевые особенности: модульная сценарная архитектура, конвертация записанных последовательностей в аудиофайлы, игровая механика с подсчётом очков, система доступа по лицензионному коду.

Содержание

[ВВЕДЕНИЕ 8](#_Toc217443574)

[1 МЕТОДЫ И АЛГОРИТМЫ РЕАЛИЗАЦИИ ВИРТУАЛЬНОГО ПИАНИНО 10](#_Toc217443575)

[1.1 Анализ существующих решений 10](#_Toc217443576)

[1.2 Выбор технологий и библиотек 10](#_Toc217443577)

[1.3 Архитектурный подход 11](#_Toc217443578)

[1.4 Разработка алгоритмов 12](#_Toc217443579)

[1.5 Выводы по разделу 14](#_Toc217443580)

[2 ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15](#_Toc217443581)

[2.1 Выбор средств разработки и системных программных средств 15](#_Toc217443582)

[2.2 Требования к разрабатываемой программе 15](#_Toc217443583)

[2.3 Структура программы 17](#_Toc217443584)

[2.3.1 Основные модули 17](#_Toc217443585)

[2.3.2 Описание основных классов 17](#_Toc217443586)

[2.3.3 Описание основных структур данных 20](#_Toc217443587)

[2.4 Описание интерфейса пользователя 21](#_Toc217443588)

[2.5 Контрольный пример и результаты тестирования 23](#_Toc217443589)

[2.6 Выводы и результаты 26](#_Toc217443590)

[ЗАКЛЮЧЕНИЕ 27](#_Toc217443591)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc217443592)

[ПРИЛОЖЕНИЕ А 29](#_Toc217443593)

[А.1 Главный управляющий модуль game.py 29](#_Toc217443594)

[А.2 Модуль сцены основного пианино scenes/piano.py 31](#_Toc217443595)

[А.3 Модуль сцены игрового режима scenes/falling\_piano\_scene.py 34](#_Toc217443596)

[А.4 Модуль сцены записи мелодий scenes/save\_mode.py 39](#_Toc217443597)

[А.5 Модуль сцены авторизации scenes/license\_scene.py 50](#_Toc217443598)

[А.6 Модуль проверки лицензии utils/auth\_system.py 52](#_Toc217443599)

[А.7 Модуль конфигурации utils/config.py 53](#_Toc217443600)

[А.8 Модуль работы с файлами utils/io\_opertion.py 56](#_Toc217443601)

[А.9 Модуль кнопок utils/button.py 59](#_Toc217443602)

[А.10 Точка входа в программу main.py 60](#_Toc217443603)

# ВВЕДЕНИЕ

Актуальность разработки интерактивных музыкальных приложений обусловлена интересом к самостоятельному обучению игре на музыкальных инструментах, а также потребностью в доступных инструментах для творчества. Существующие решения часто либо ограничены по функционалу (простые синтезаторы), либо требуют подписки (обучающие платформы), либо не сочетают игровые и творческие возможности.

В настоящее время на рынке представлены такие продукты, как Synthesia (визуализация нот для обучения), Simply Piano (интерактивные уроки с подпиской) и Audacity (запись аудио). Однако комплексное решение, объединяющее свободное музицирование, игровое обучение и запись собственных композиций в едином интерфейсе, встречается редко.

Целью данного проекта является создание приложения «Виртуальное пианино», которое сочетает в себе функции музыкального инструмента, обучающей игры и студии записи.

В соответствии с поставленной целью были сформулированы и решены следующие задачи:

1. Анализ существующих аналогов и определение ключевых функций;
2. Изучение библиотек Python для работы со звуком и графикой;
3. Разработка архитектуры приложения на основе системы сцен;
4. Реализация основного режима пианино с поддержкой нескольких тембров;
5. Разработка игрового режима «Падающие ноты» с системой подсчёта очков;
6. Создание модуля записи, сохранения и экспорта мелодий (JSON, WAV);
7. Реализация системы авторизации для доступа;
8. Тестирование и отладка программы, проверка корректности работы всех режимов.

# МЕТОДЫ И АЛГОРИТМЫ РЕАЛИЗАЦИИ ВИРТУАЛЬНОГО ПИАНИНО

## Анализ существующих решений

В рамках подготовки к разработке проведён анализ трёх категорий программных продуктов:

* Обучающие программы (Synthesia, Simply Piano) – предоставляют интерактивные уроки с визуальной обратной связью, но, как правило, не позволяют записывать собственные композиции и требуют оплаты подписки;
* Аудиоредакторы (Audacity) – предлагают мощные инструменты записи и обработки звука, но не содержат игровых элементов и специализированного интерфейса для обучения игре на клавишах;
* Простые виртуальные синтезаторы – часто реализуют только базовое воспроизведение звуков по нажатию клавиш без дополнительных функций.

На основе анализа сделан вывод о необходимости создания гибридного приложения, объединяющего сильные стороны каждой категории: интерактивность и визуализацию из обучающих программ, возможность записи из аудиоредакторов и простоту использования из синтезаторов.

## Выбор технологий и библиотек

Для реализации выбраны следующие технологии:

* Язык программирования Python – обеспечивает простоту разработки и богатую экосистему библиотек;
* Библиотека Pygame – предоставляет готовые модули для работы с графикой, звуком и обработки событий ввода, что идеально подходит для создания интерактивных приложений с графическим интерфейсом;
* Библиотеки Wave и NumPy – используются для программной генерации WAV-файлов из записанных последовательностей нот;
* Модуль hashlib – применяется для реализации простой системы лицензирования путём сравнения хэшей SHA-256;
* Формат JSON – выбран для хранения данных о мелодиях (ноты и временные интервалы) благодаря своей структурированности и лёгкости использования.

## Архитектурный подход

В качестве базового архитектурного решения выбрана сценарная модель (Scene-based architecture). Каждая функциональная часть приложения (основное пианино, игра, запись, авторизация) выделена в отдельную сцену – самостоятельный класс, управляющий своим циклом событий, обновлением состояния и отрисовкой. Главный класс Game осуществляет перемещение между сценами на основе флагов их состояния (например, need\_switch\_to\_game, need\_switch\_to\_record). Такой подход обеспечивает:

* Низкую связность – сцены слабо зависят друг от друга.
* Лёгкую расширяемость – для добавления нового режима достаточно создать новый класс-сцену.
* Упрощённую навигацию – управление переходами централизовано.

## Разработка алгоритмов

* + 1. **Алгоритм записи и воспроизведения мелодий**

Входные данные: последовательность нажатий клавиш пианино (S, D, F, G, H, J, K) в реальном времени.

Выходные данные:

1. Структура данных в формате JSON, содержащая для каждой ноты её имя и задержку относительно предыдущей ноты (в миллисекундах).
2. Аудиофайл в формате WAV, представляющий собой сведённую последовательность звуков, воспроизведённых с учётом временных интервалов.

Алгоритм записи:

1. Фиксируется время начала записи (record\_start\_time).
2. При каждом нажатии клавиши сохраняется:
   * Имя ноты (например, 'S');
   * Абсолютное время нажатия относительно record\_start\_time;
   * Относительная задержка от предыдущей ноты.
3. По окончании записи массив нот преобразуется в список словарей и сохраняется в JSON-файл.

Алгоритм конвертации в WAV:

1. На основе данных JSON вычисляется общая длительность будущего аудиофайла;
2. Создаётся пустой аудиобуфер соответствующей длины;
3. Для каждой ноты в последовательности:
   * Загружается соответствующий звуковой файл (.wav);
   * Вычисляется момент начала звучания ноты в буфере (сумма задержек предыдущих нот);
   * Аудиоданные ноты накладываются на буфер в рассчитанной позиции.
4. Полученный буфер нормализуется и экспортируется в WAV-файл.
   * 1. **Алгоритм записи и воспроизведения мелодий**

Входные данные: выбранная пользователем мелодия из библиотеки (в виде JSON-структуры).

Выходные данные: статистика игровой сессии (количество набранных очков, количество промахов).

Описание алгоритма:

1. Инициализация. Выбранная мелодия загружается, формируется очередь нот для отображения;
2. Игровой цикл:
   * Генерация нот. Визуальные объекты нот появляются в верхней части игровой области с интервалами, соответствующими временным задержкам, указанным в JSON-файле;
   * Движение нот. В каждом кадре анимации координата Y каждого объекта ноты увеличивается на заданную величину (FALLING\_SPEED), создавая эффект падения;
   * Проверка попадания. Определяется момент, когда центр объекта ноты пересекает специальную область взаимодействия («зелёная зона»), расположенную напротив виртуальных клавиш. Если в этот момент пользователь нажал на клавиатуре клавишу, соответствующую падающей ноте, фиксируется успешное попадание: объект ноты удаляется, значение счётчика увеличивается;
   * Учёт промахов. Если объект ноты покидает область взаимодействия без соответствующего нажатия клавиши, фиксируется промах, и значение счётчика промахов увеличивается.
3. Завершение игры. Игровая сессия завершается после отображения и обработки всех нот выбранной мелодии.

## Выводы по разделу

В результате проведённого анализа и проектирования для реализации проекта выбран язык программирования Python в связке с библиотекой Pygame, что обеспечивает кроссплатформенность и скорость разработки интерактивных приложений. Определена и применена сценарная архитектура как оптимальная для организации различных режимов работы программы. Разработаны ключевые алгоритмы, включая запись и сериализацию мелодий, их конвертацию в аудиоформат, а также игровую механику с падающими нотами. Данные решения легли в основу программной реализации, описание которой приведено в следующем разделе.

# ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В разделе описываются теоретические основы используемых в проекте алгоритмических и программных средств, изучение или разработка которых производилась студентом в соответствии с заданием.

Ориентировочный объем раздела 8-10 страниц для работ, имеющих исследовательскую часть, и 13-17 страниц для работ без исследовательской части. Примерная структура раздела приведена ниже.

## Выбор средств разработки и системных программных средств

Для реализации проекта выбраны следующие средства разработки и системные программные средства:

* язык программирования Python;
* библиотека Pygame для создания графического интерфейса и работы со звуком;
* среда разработки PyCharm Community Edition;
* системные библиотеки Python: hashlib, json, wave, numpy.

Выбор языка Python обусловлен его простотой, читаемостью кода и наличием богатой экосистемы библиотек. Библиотека Pygame предоставляет необходимый инструментарий для работы с графикой, звуком и обработкой событий ввода, что соответствует задачам проекта.

## Требования к разрабатываемой программе

* + - 1. Требования к функциональным характеристикам.

- Воспроизведение звуков при нажатии клавиш клавиатуры;

- Поддержка как минимум одной октавы (7 белых клавиш);

- Ограничение доступа к функционалу для пользователей без активной подписки;

- Поддержка режимов работы (свободная игра/мини-игры);

- Переключение между звуковыми библиотеками.

* + - 1. Требования к структуре системы и организации взаимодействия между ее компонентами.

Программа должна быть построена по модульному принципу со следующими компонентами: модуль аутентификации, модуль звукового воспроизведения, модуль игровой логики, модуль пользовательского интерфейса, модуль работы с данными.

* + - 1. Требования к режимам функционирования.

- Режим обычного пианино - свободная игра на виртуальном инструменте, с возможностью выбора звуковой библиотеки;

- Режим мини-игры - игровой процесс обучения мелодиям.

* + - 1. Требования к аппаратному обеспечению.

Процессор: не менее 1 ГГц

Оперативная память: не менее 512 МБ

Поддержка воспроизведения звука (встроенная звуковая карта)

Устройства ввода: клавиатура и мышь

* + - 1. Требования к общесистемным программным средствам.

Операционная система Windows 7 и выше.

* + - 1. Требования к средствам разработки.

Язык программирования: Python 3.8+

Графическая библиотека: Pygame 2.0+

Среда разработки: PyCharm.

* + - 1. Требования к интерфейсу пользователя.

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователей средней квалификации.

* + - 1. Требования к надежности.

Программное обеспечение должно обрабатывать ошибки в процессе работы и своевременно оповещать пользователя.

* + - 1. Требования к безопасности и защите информации.

Хранение пароля в хэшированном виде

* + - 1. Требования к разрабатываемой документации.

Документация должна разрабатываться в соответствии с требованиями СТО.

## Структура программы

### Основные модули

Структура исходного кода программы включает следующие модули:

* main.py – модуль точки входа в программу;
* core/game.py – главный модуль, инициализирующий приложение и управляющий переключением сцен;
* scenes/piano.py – модуль сцены основного режима пианино;
* scenes/falling\_piano\_scene.py – модуль сцены игрового режима «Падающие ноты»;
* scenes/save\_mode.py – модуль сцены режима записи и управления мелодиями;
* scenes/license\_scene.py – модуль сцены окна авторизации;
* utils/config.py – модуль конфигурации (размеры, цвета, пути);
* utils/buttons.py – модуль универсального класса кнопок;
* utils/auth\_system.py – модуль проверки лицензионного кода;
* utils/io\_operation.py – модуль работы с файлами json и png;
* sounds – содержит банк звуков.

### Описание основных классов

В программе реализованы следующие ключевые классы:

1. Класс Game (модуль game.py) – главный управляющий класс приложения. Отвечает за инициализацию графической среды Pygame, управление жизненным циклом приложения и переключением между сценами. Основные методы класса:

* \_\_init\_\_ – конструктор класса, выполняет инициализацию Pygame, создание окна, установку начальной сцены (лицензионной);
* switch\_scene – осуществляет переход между сценами (например, с лицензионной на основную сцену пианино) на основе переданного режима;
* run – главный игровой цикл, обрабатывающий события, обновление состояния и отрисовку текущей сцены.

1. Класс PianoScene (модуль scenes/piano.py) – класс сцены основного режима пианино. Отвечает за воспроизведение звуков, визуализацию клавиатуры и обработку взаимодействия пользователя. Основные поля и методы:

* self.sound\_modes – словарь, загружающий звуковые файлы для трех режимов (рояль, пианино, ударные);
* self.current\_mode – атрибут, хранящий идентификатор текущего звукового режима;
* self.key\_glow\_time – список временных меток для управления подсветкой клавиш;
* switch\_mode – метод циклического переключения между звуковыми режимами;
* handle\_events – метод обработки событий клавиатуры и мыши для воспроизведения звуков и навигации.

1. Класс FallingPianoScene (модуль scenes/falling\_piano\_scene.py) – класс сцены игрового режима «Падающие ноты». Наследуется от PianoScene и расширяет его функциональность игровой механикой. Основные поля и методы:

* self.falling\_notes – список активных падающих нот, каждый элемент которого является словарем с координатами, типом ноты и состоянием;
* self.score, self.misses – атрибуты для хранения игровой статистики (количество попаданий и промахов);
* self.is\_playing – флаг, указывающий на активное состояние игрового процесса;
* add\_falling\_note – метод создания новой падающей ноты на основе данных мелодии;
* update\_falling\_notes – метод обновления позиций нот, проверки попаданий и удаления вышедших за пределы экрана;
* reset\_game – метод сброса игрового состояния (очистка нот, обнуление счета).

1. Класс MelodyRecorderScene (модуль scenes/save\_mode.py) – класс сцены режима записи мелодий. Наследуется от PianoScene и добавляет функциональность записи, сохранения и управления библиотекой композиций. Основные поля и методы:

* self.recording – флаг, указывающий на активность процесса записи;
* self.recorded\_notes – список записанных нот, где каждый элемент содержит идентификатор ноты и временную метку;
* self.all\_melodies – список всех сохраненных мелодий, загруженных из JSON-файла;
* start\_recording, stop\_recording – методы управления процессом записи;
* save\_melody – метод сохранения записанной последовательности в JSON-файл;
* convert\_json\_to\_wav – метод конвертации данных мелодии из JSON в аудиофайл формата WAV.

1. Класс LicenseScene (модуль scenes/license\_scene.py) – класс сцены лицензионной проверки. Реализует интерфейс для ввода и верификации кода доступа. Основные поля и методы:

* self.input\_code – атрибут, хранящий введенную пользователем строку кода;
* self.verified – флаг успешности проверки лицензии;
* self.message – строка для отображения статуса проверки (например, «Доступ разрешен»);
* check\_code – метод, выполняющий проверку введенного кода путем сравнения его хэша с эталонным значением.

1. Класс ImageButton (модуль utils/buttons.py) – универсальный класс для создания графических кнопок с поддержкой текста, иконок и интерактивности. Основные поля и методы:

* self.rect – объект прямоугольной области кнопки для обработки коллизий;
* self.text, self.icon – атрибуты для хранения текстовой метки и графической иконки кнопки;
* self.is\_hovered – флаг наведения курсора мыши на кнопку;
* draw – метод отрисовки кнопки на экране с учетом состояния (обычное, наведение);
* handle\_event – метод обработки событий мыши для определения нажатия на кнопку.

### Описание основных структур данных

Основные структуры данных, используемые в программе:

* Список falling\_notes (в классе FallingPianoScene). Хранит словари, описывающие каждую падающую ноту. Ключи словаря: note\_index, note\_name, x, y, color, hit;
* Список recorded\_notes (в классе MelodyRecorderScene). Хранит словари с данными о записанных нотах. Ключи словаря: note, time, delay;
* Словарь SOUND\_PATHS (в модуле config.py). Вложенная структура, сопоставляющая идентификаторы звуковых режимов и путей к аудиофайлам для каждой ноты;
* Список all\_melodies (в MelodyRecorderScene и FallingPianoScene). Содержит словари, каждый из которых представляет одну сохраненную мелодию с ключами: name, flag, notes (список нот с задержками).

## Описание интерфейса пользователя

Графический интерфейс пользователя реализован с помощью библиотеки Pygame. Интерфейс различных сцен выдержан в едином стиле с темной цветовой схемой.

Сцена авторизации (LicenseScene). Содержит заголовок «ПРОВЕРКА ДОСТУПА», текстовое поле для ввода кода (отображаемого символами •), кнопки «ПРОВЕРИТЬ» и «ВЫЙТИ», а также информационную строку внизу экрана;

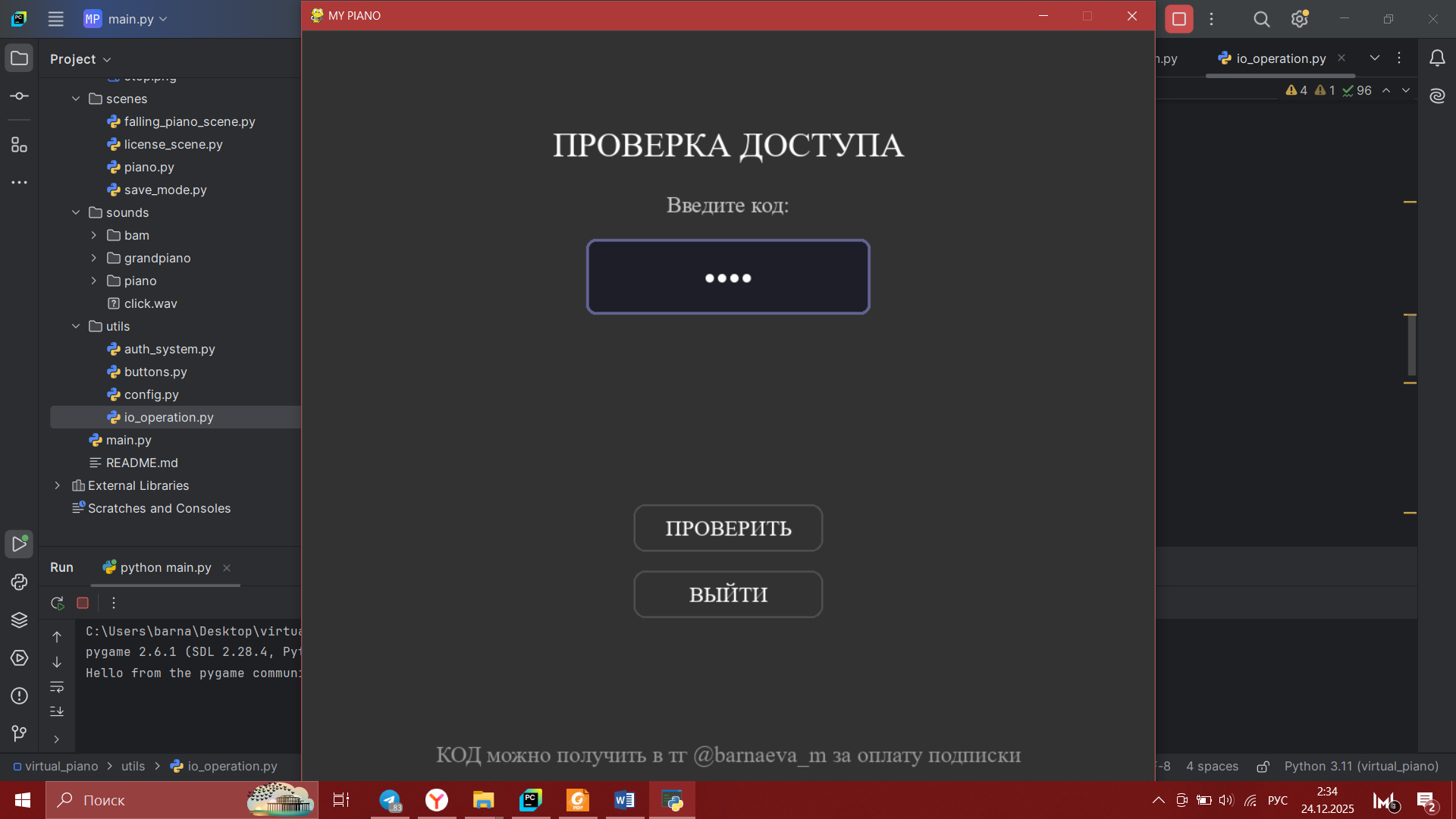


Рисунок 1 – Сцена авторизации

 Основная сцена пианино (PianoScene). В верхней части расположены кнопки управления: «РЕЖИМ ЗВУКА», «РЕЖИМ ИГРЫ», «ЗАПИСЬ МЕЛОДИИ», «ВЫХОД». Напротив кнопок «РЕЖИМ ЗВУКА», «РЕЖИМ ИГРЫ» отображается название текущего звукового и игрового режимов соответственно. В нижней части экрана визуализируются клавиши пианино из семи белых клавиш с подписями (S, D, F, G, H, J, K). При нажатии соответствующих клавиш на физической клавиатуре виртуальная клавиша подсвечивается.

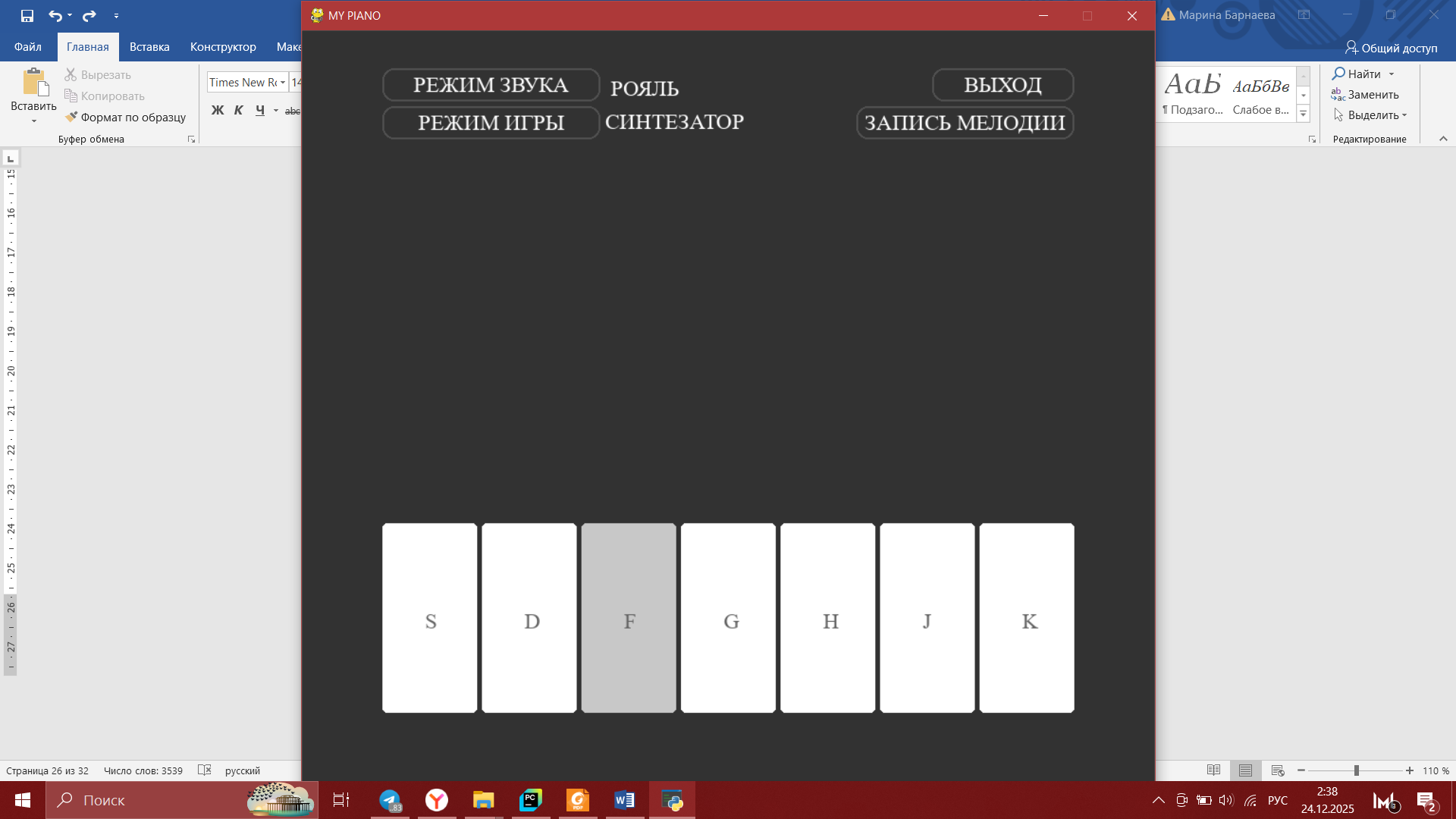


Рисунок 2 – Сцена пианино

Сцена игрового режима (FallingPianoScene). Наследует интерфейс основной сцены родительского класса (PianoScene). Дополнительно в верхней части появляется панель управления игрой: кнопки с иконками пуск/пауза, «СБРОС» и кнопка выбора мелодии с её названием. В центральной части расположена игровая область с падающими нотами, ниже — зелёная зона попадания. В верхнем левом углу отображается статистика: прогресс, счёт и количество промахов. Внизу экрана находится текстовая инструкция.

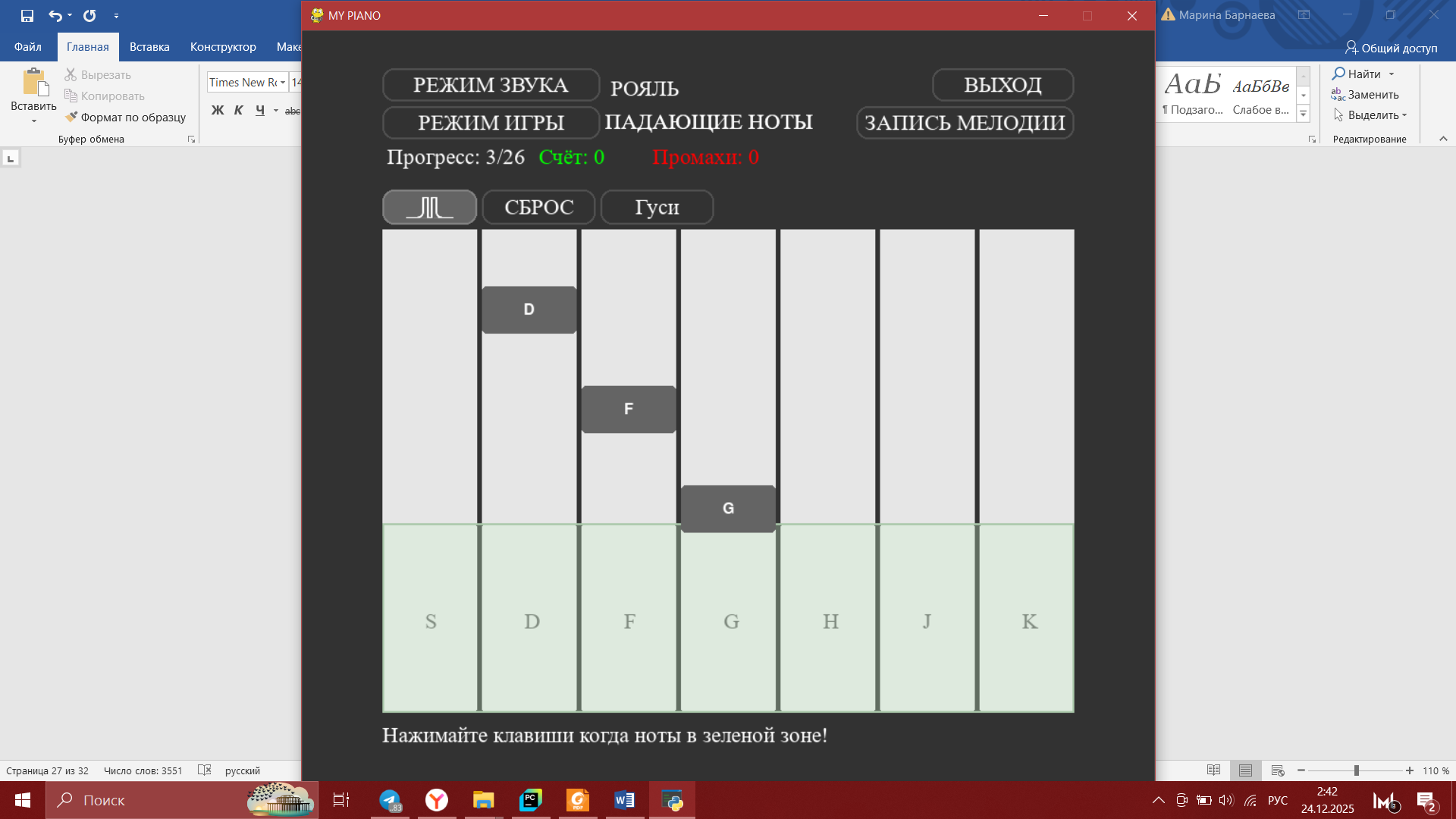


Рисунок 3 – Сцена пианино

Сцена записи мелодий (MelodyRecorderScene). Наследует интерфейс основной сцены. Слева располагается панель со списком сохранённых мелодий с возможностью прокрутки. Справа — вертикальная панель управления с кнопками: «НАЧАТЬ ЗАПИСЬ/ОСТАНОВИТЬ», «СОХРАНИТЬ», «СОХРАНИТЬ .WAV», «ВОСПРОИЗВЕСТИ», «УДАЛИТЬ». Под списком мелодий отображается информация о количестве записанных нот, под панелью управления кнопками - детали выбранной мелодии.

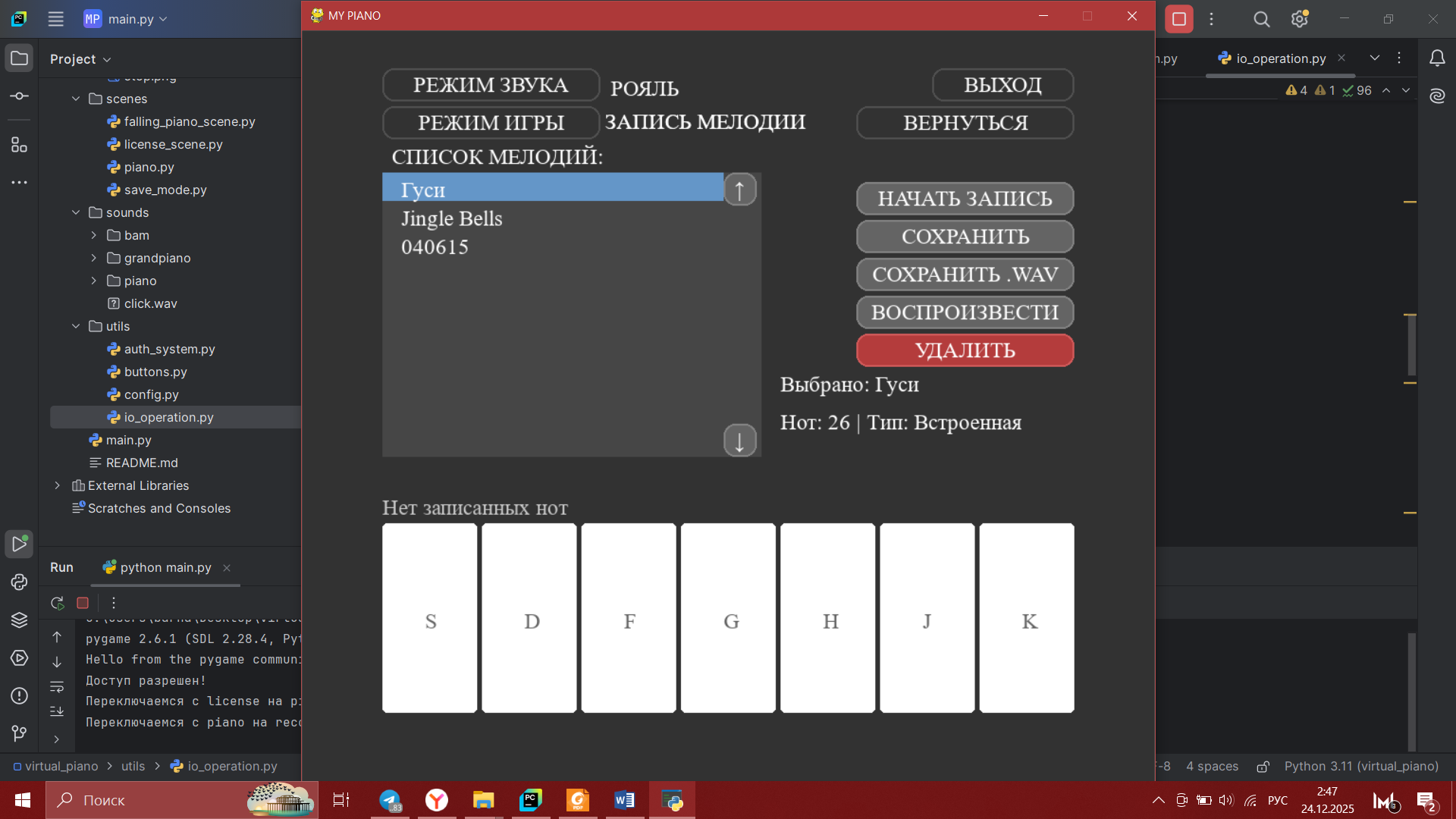


Рисунок 4 – Сцена записи мелодий

Навигация между сценами осуществляется нажатием соответствующих кнопок в интерфейсе. Все кнопки обладают визуальным отображением при наведении курсором и звуковым сопровождением при нажатии.

## Контрольный пример и результаты тестирования

Пример 1. Тестирование сцены авторизации, попытка ввести неверный код.

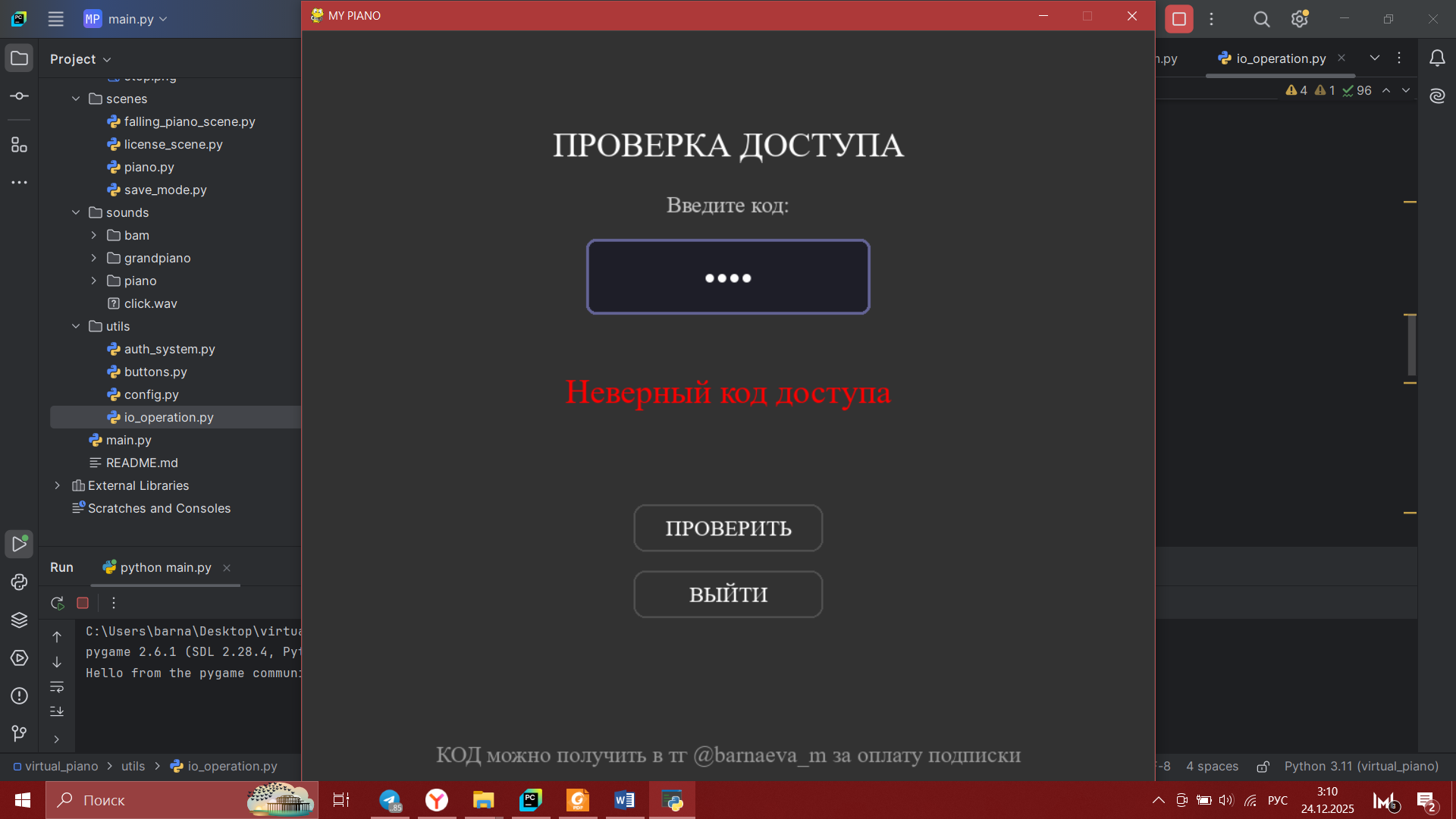


Рисунок 5 – Тестирование проверки доступа

Пример 2. Тестирование удаления/записи мелодии.

Точка А (сцена до проведения тестирования): существует мелодия «040615».

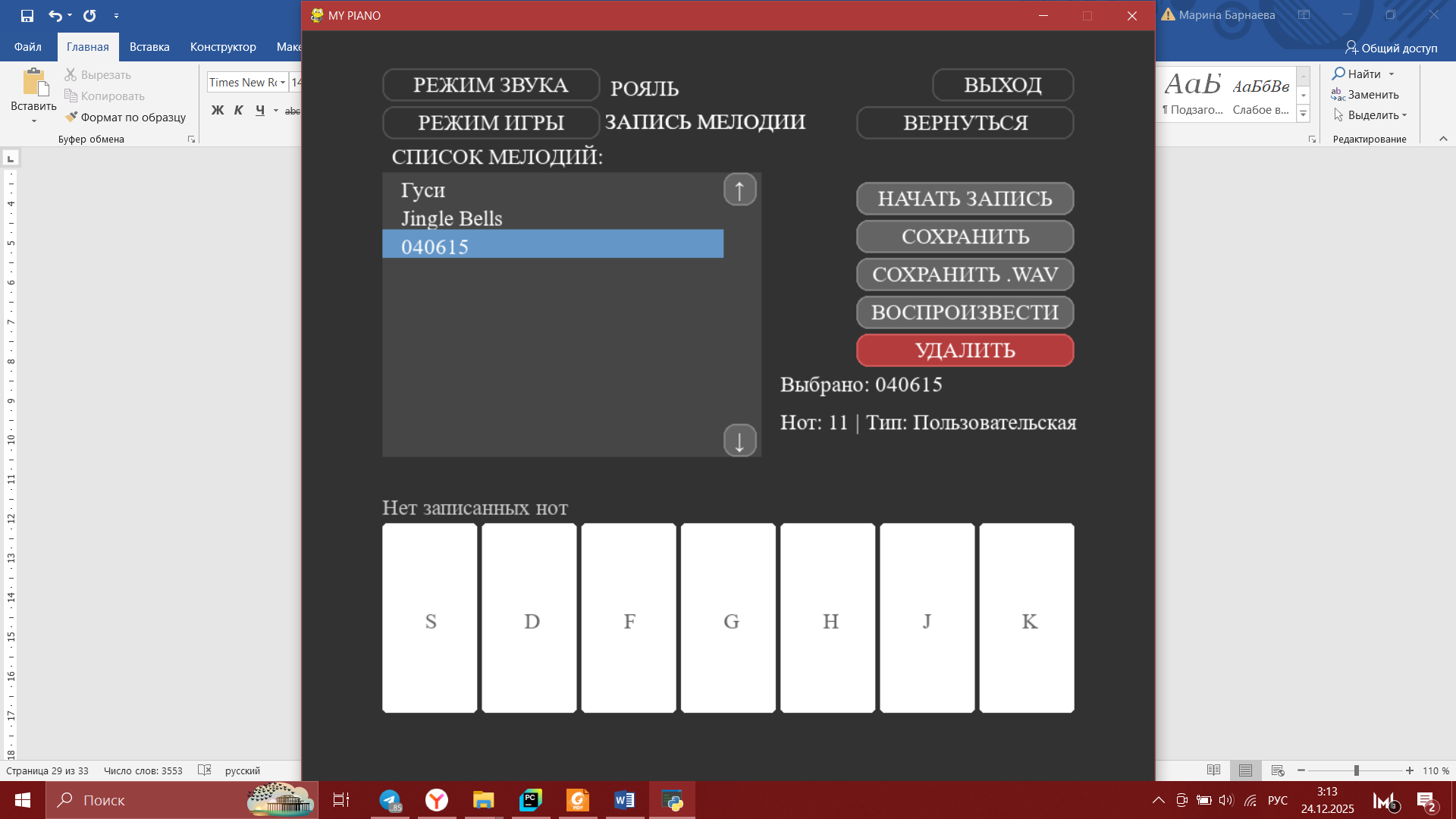


Рисунок 6 – Точка А

Точка Б (сцена после совершения удаления и записи): отображается новая мелодия «031403», мелодия «040615» отсутствует.

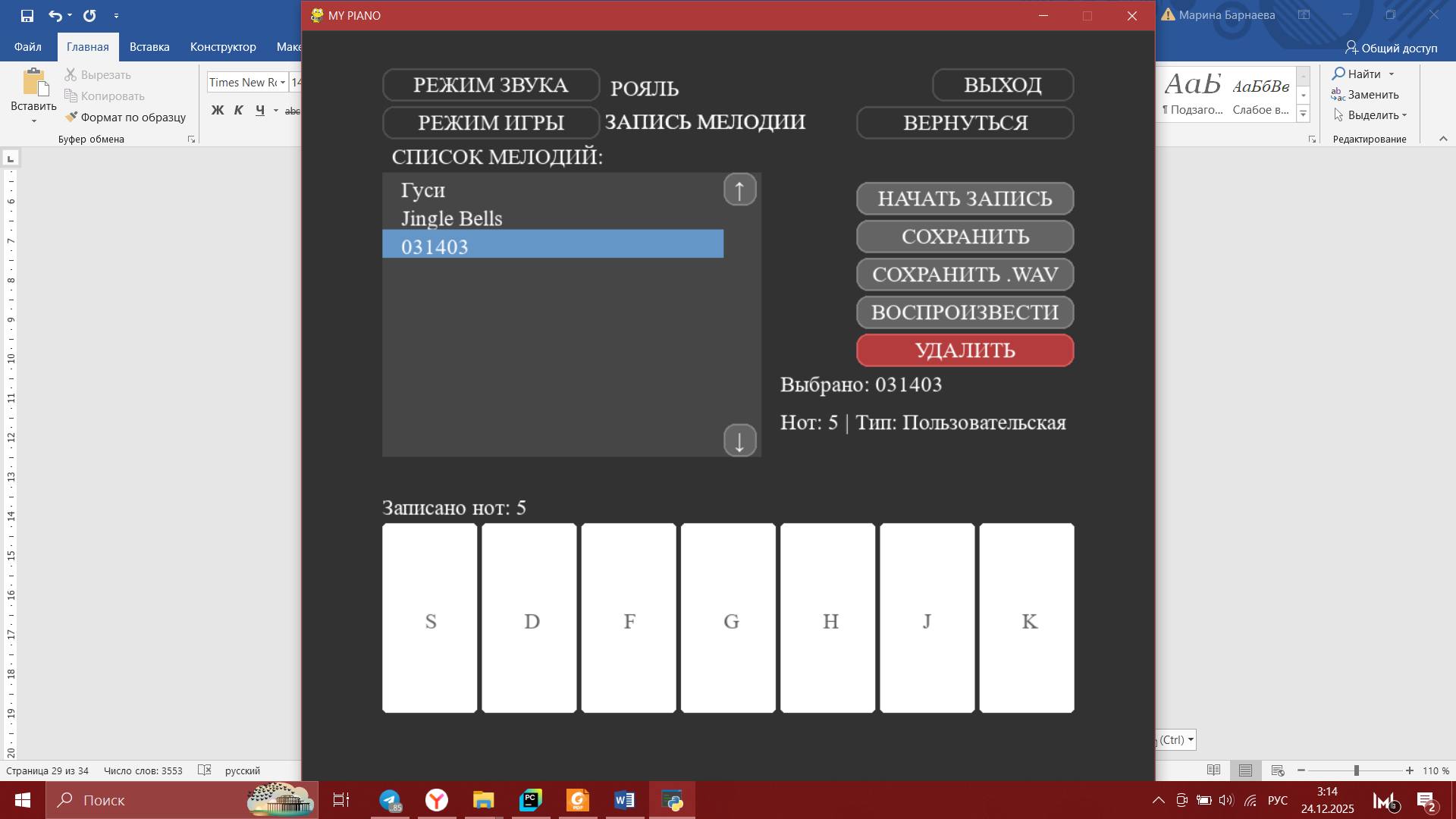


Рисунок 7 – Точка Б

Пример 3. Тестирование подсчета промохов/попаданий сцены «Падающие ноты».

Для тестирования намерено пропущены две ноты при прохождении выбранной мелодии.

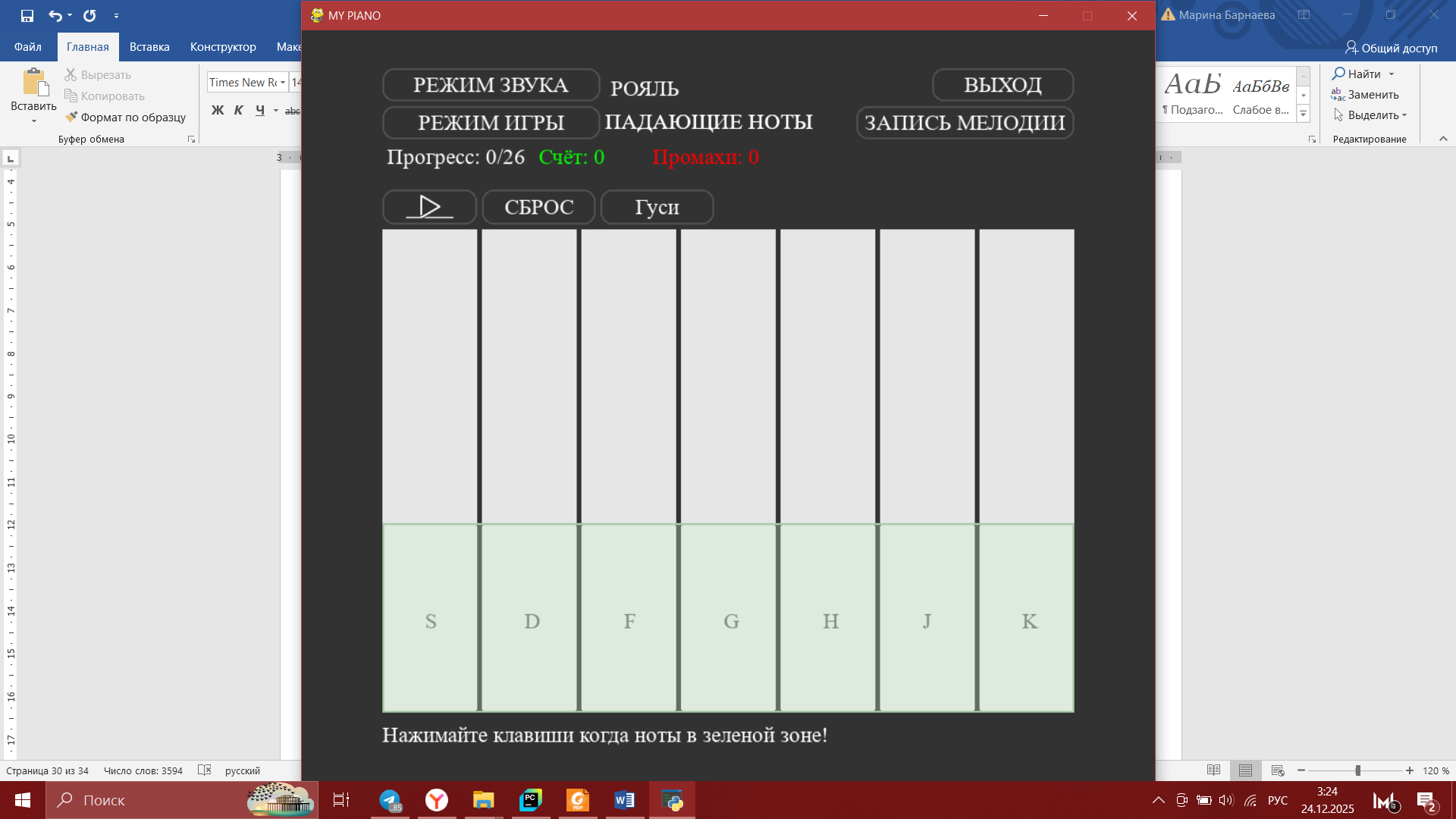


Рисунок 8 – Запуск режима

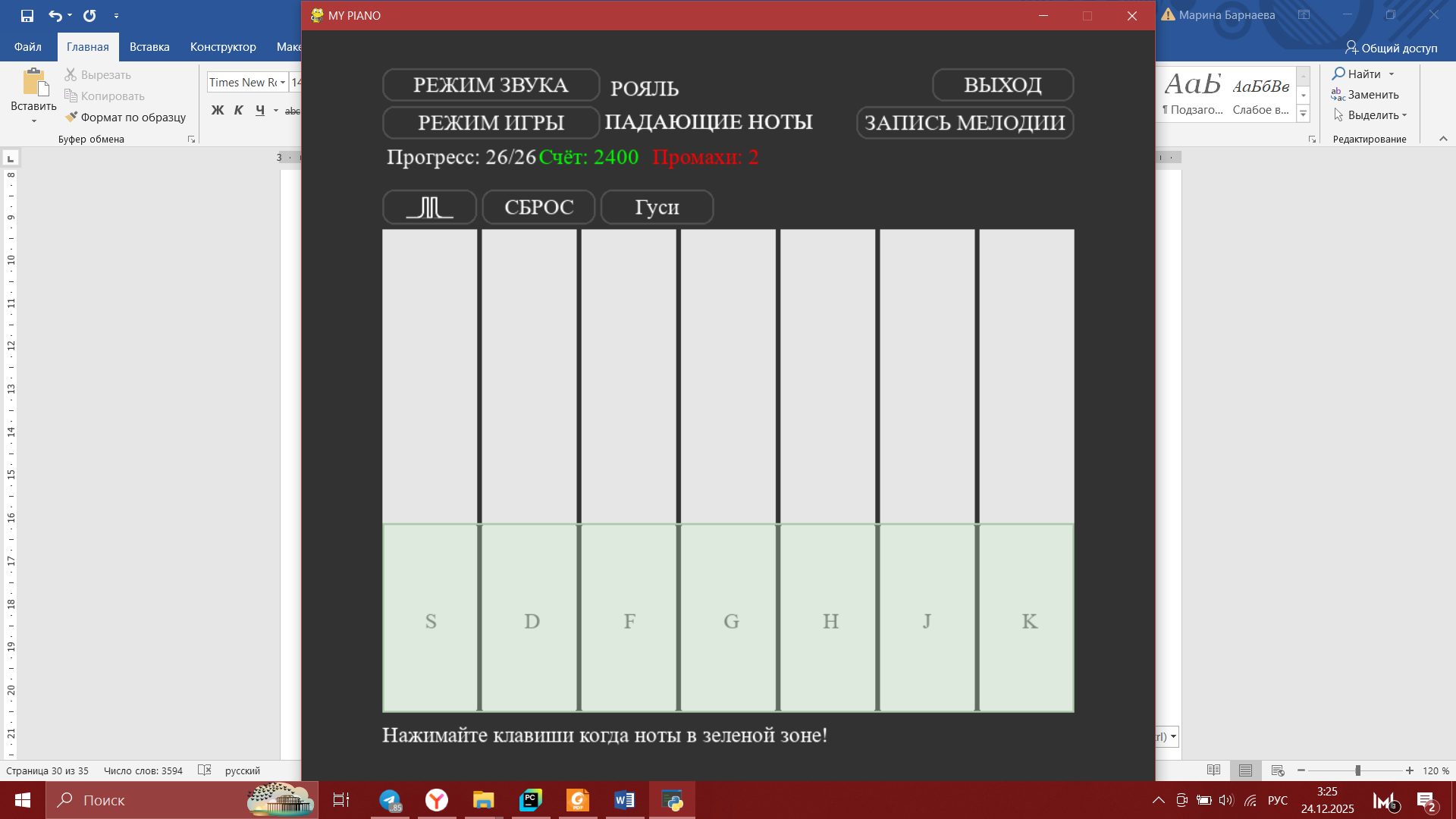


Рисунок 9 – Сцена после прохождения

## Выводы и результаты

В результате курсового проекта было разработано приложение «Виртуальное пианино», полностью соответствующее сформулированным требованиям. Реализована модульная архитектура на основе сцен, обеспечивающая четкое разделение функциональности. Разработан интуитивно понятный графический интерфейс с тремя основными режимами работы. Программа успешно прошла тестирование по контрольным примерам, подтвердив корректность работы алгоритмов записи, удаления, игровой механики и системы авторизации. Все функциональные требования выполнены.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения курсового проекта была разработана программа «Виртуальное пианино», представляющая собой интерактивное приложение с графическим интерфейсом. Программа объединяет функции музыкального инструмента, обучающей игры и студии звукозаписи.

Разработанное программное обеспечение соответствует сформулированным требованиям. Приложение обладает интуитивно понятным пользовательским интерфейсом. Ключевыми особенностями программы являются модульность архитектуры, возможность сохранения и конвертации пользовательских композиций, а также сочетание возможностей для творчества и обучения в рамках единого интерфейса.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pygame документация [Электронный ресурс] // Официальный сайт Pygame. URL: <https://www.pygame.org/docs/>.

2. Python 3.9 документация [Электронный ресурс] // Официальный сайт Python. URL: <https://docs.python.org/3.9/>.

3. Создание WAV-файлов с помощью модуля wave [Электронный ресурс] // Документация Python. URL: <https://docs.python.org/3/library/wave.html#wave.Wave_write.writeframes>.

4. Модуль hashlib: безопасные хэши и дайджесты сообщений [Электронный ресурс] // Документация Python. URL: <https://docs.python.org/3/library/hashlib.html>.

5. Набор звуков фортепиано [Электронный ресурс] // База звуков Freesound. URL: <https://zvukogram.com/category/zvuki-not/>.

# ПРИЛОЖЕНИЕ А

Основные модули разработанной программы

## А.1 Главный управляющий модуль game.py

import pygame

from utils.config import \*

from scenes.license\_scene import LicenseScene

from scenes.piano import PianoScene

from scenes.falling\_piano\_scene import FallingPianoScene

from scenes.save\_mode import MelodyRecorderScene

class Game:

    def \_\_init\_\_(self):

        pygame.init()

        pygame.mixer.init(\*\*SOUND\_SETTINGS)

        pygame.mixer.set\_num\_channels(NUM\_CHANNELS)

        self.screen = pygame.display.set\_mode((WIDTH, HEIGHT))

        pygame.display.set\_caption("MY PIANO")

        self.clock = pygame.time.Clock()

        self.running = True

        # Состояние доступа

        self.access\_granted = False

        self.is\_premium = False

        # Сначала создаем сцену лицензии

        self.license\_scene = LicenseScene(self.screen, self.clock)

        self.current\_scene = self.license\_scene

        self.current\_mode = "license"

    def switch\_scene(self, new\_mode):

        """Переключает сцены"""

        if new\_mode == self.current\_mode:

            return

        print(f"Переключаемся с {self.current\_mode} на {new\_mode}")

        if new\_mode == "license":

            self.current\_scene = LicenseScene(self.screen, self.clock)

        elif new\_mode == "piano":

            self.current\_scene = PianoScene(self.screen, self.clock)

        elif new\_mode == "falling":

            self.current\_scene = FallingPianoScene(self.screen, self.clock)

        elif new\_mode == "recorder":

            self.current\_scene = MelodyRecorderScene(self.screen, self.clock)

        self.current\_mode = new\_mode

    def apply\_restrictions(self):

        if not self.is\_premium:

            print("Бесплатная версия - ограничения применены")

    def check\_premium\_features(self, feature\_name):

        if not self.is\_premium:

            print(f"Функция '{feature\_name}' доступна только в премиум версии!")

            return False

        return True

    def run(self):

        while self.running:

            if not self.access\_granted:

                self.running = self.license\_scene.run()

                if hasattr(self.license\_scene, 'verified') and self.license\_scene.verified:

                    self.access\_granted = True

                    self.is\_premium = True

                    print("Доступ разрешен!")

                    self.switch\_scene("piano")

                    self.apply\_restrictions()

                else:

                    continue

            self.running = self.current\_scene.run()

            if hasattr(self.current\_scene, 'need\_switch\_to\_game') and self.current\_scene.need\_switch\_to\_game:

                if self.current\_mode == "piano":

                    if self.check\_premium\_features("Режим игры с падающими нотами"):

                        self.switch\_scene("falling")

                elif self.current\_mode == "falling":

                    self.switch\_scene("piano")

            elif hasattr(self.current\_scene, 'need\_switch\_to\_record') and self.current\_scene.need\_switch\_to\_record:

                if self.current\_mode == "piano":

                    if self.check\_premium\_features("Режим записи мелодий"):

                        self.switch\_scene("recorder")

            elif hasattr(self.current\_scene, 'need\_switch\_to\_normal') and self.current\_scene.need\_switch\_to\_normal:

                if self.current\_mode == "recorder":

                    self.switch\_scene("piano")

            elif hasattr(self.current\_scene, 'need\_exit\_to\_license'):

                self.access\_granted = False

                self.is\_premium = False

                self.switch\_scene("license")

        pygame.quit()

## А.2 Модуль сцены основного пианино scenes/piano.py

import pygame

from utils.config import \*

from utils.buttons import ImageButton

class PianoScene:

    def \_\_init\_\_(self, screen, clock):

        self.screen = screen

        self.clock = clock

        self.sound\_modes = self.load\_sounds()

        self.current\_mode = "1"

        self.current\_sounds = self.sound\_modes[self.current\_mode]

        self.key\_glow\_time = [0] \* KEY\_COUNT

        self.mode\_button = ImageButton(

            MODE\_BUTTON\_X, MODE\_BUTTON\_Y, BUTTON\_WIDTH, BUTTON\_HEIGHT, "РЕЖИМ ЗВУКА",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH

        )

        self.exit\_button = ImageButton(

            EXIT\_BUTTON\_X, EXIT\_BUTTON\_Y, EXIT\_BUTTON\_WIDTH, EXIT\_BUTTON\_HEIGHT, "ВЫХОД",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH

        )

        self.mode\_button\_game = ImageButton(

            MODE\_BUTTON\_X, MODE\_GAME\_BUTTON\_Y, BUTTON\_WIDTH, BUTTON\_HEIGHT, "РЕЖИМ ИГРЫ",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH

        )

        self.record\_button = ImageButton(

            RECORD\_BUTTON\_X, RECORD\_BUTTON\_Y, BUTTON\_WIDTH, BUTTON\_HEIGHT, "ЗАПИСЬ МЕЛОДИИ",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH

        )

        self.need\_switch\_to\_game = False

        self.need\_switch\_to\_record = False

        self.current\_mode\_game = "1"

    def load\_sounds(self):

        sound\_modes = {}

        for mode\_name, sound\_paths in SOUND\_PATHS.items():

            sound\_modes[mode\_name] = {}

            for note, path in sound\_paths.items():

                try:

                    sound\_modes[mode\_name][note] = pygame.mixer.Sound(path)

                except pygame.error as e:

                    print(f"Ошибка загрузки звука {path}: {e}")

                    sound\_modes[mode\_name][note] = None

        return sound\_modes

    def switch\_mode(self):

        self.current\_mode = "2" if self.current\_mode == "1" else "3" if self.current\_mode == "2" else "1"

        self.current\_sounds = self.sound\_modes[self.current\_mode]

    def handle\_events(self):

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                return False

            if self.mode\_button.handle\_event(event):

                self.switch\_mode()

            if self.exit\_button.handle\_event(event):

                pygame.time.delay(130)

                return False

            if self.mode\_button\_game.handle\_event(event):

                self.current\_mode\_game = "2" if self.current\_mode\_game == "1" else "1"

                self.need\_switch\_to\_game = True

            if self.record\_button.handle\_event(event):

                self.current\_mode\_game = "3"

                self.need\_switch\_to\_record = True

            elif event.type == pygame.KEYDOWN:

                match event.key:

                    case pygame.K\_s:

                        self.current\_sounds['S'].play()

                        self.key\_glow\_time[0] = pygame.time.get\_ticks()

                    case pygame.K\_d:

                        self.current\_sounds['D'].play()

                        self.key\_glow\_time[1] = pygame.time.get\_ticks()

                    case pygame.K\_f:

                        self.current\_sounds['F'].play()

                        self.key\_glow\_time[2] = pygame.time.get\_ticks()

                    case pygame.K\_g:

                        self.current\_sounds['G'].play()

                        self.key\_glow\_time[3] = pygame.time.get\_ticks()

                    case pygame.K\_h:

                        self.current\_sounds['H'].play()

                        self.key\_glow\_time[4] = pygame.time.get\_ticks()

                    case pygame.K\_j:

                        self.current\_sounds['J'].play()

                        self.key\_glow\_time[5] = pygame.time.get\_ticks()

                    case pygame.K\_k:

                        self.current\_sounds['K'].play()

                        self.key\_glow\_time[6] = pygame.time.get\_ticks()

        return True

    def update(self):

        current\_time = pygame.time.get\_ticks()

        for i in range(KEY\_COUNT):

            if self.key\_glow\_time[i] > 0 and current\_time - self.key\_glow\_time[i] > GLOW\_DURATION:

                self.key\_glow\_time[i] = 0

        self.mode\_button.check\_hover(pygame.mouse.get\_pos())

        self.exit\_button.check\_hover(pygame.mouse.get\_pos())

        self.mode\_button\_game.check\_hover(pygame.mouse.get\_pos())

        self.record\_button.check\_hover(pygame.mouse.get\_pos())

    def draw(self):

        self.screen.fill(BACKGROUND)

        for i in range(KEY\_COUNT):

            x = PIANO\_START\_X + i \* (KEY\_WIDTH + KEY\_SPACING)

            if (self.key\_glow\_time[i] > 0 and

                    (pygame.time.get\_ticks() - self.key\_glow\_time[i]) < GLOW\_DURATION):

                color = KEYS\_PIANO

            else:

                color = TEXT

            pygame.draw.rect(self.screen, color,

                             (x, PIANO\_START\_Y, KEY\_WIDTH, KEY\_HEIGHT), 0, 5)

        self.mode\_button.draw(self.screen)

        self.exit\_button.draw(self.screen)

        self.mode\_button\_game.draw(self.screen)

        self.record\_button.draw(self.screen)

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        mode\_name = CURRENT\_MODE.get(self.current\_mode, f"Режим {self.current\_mode}")

        mode\_text = font.render(f" {mode\_name} ", True, TEXT)

        self.screen.blit(mode\_text, (MODE\_TEXT\_X, MODE\_TEXT\_Y))

        for i, label in enumerate(KEY\_LABELS):

            key\_x = PIANO\_START\_X + i \* (KEY\_WIDTH + KEY\_SPACING) + KEY\_TEXT\_OFFSET\_X

            key\_y = PIANO\_START\_Y + KEY\_TEXT\_OFFSET\_Y

            key\_text = font.render(label, True, BUTTON)

            self.screen.blit(key\_text, (key\_x, key\_y))

        self.draw\_mode\_game()

    def draw\_mode\_game(self):

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        mode\_text = font.render(CURRENT\_MODE\_GAME[0], True, TEXT)

        self.screen.blit(mode\_text, (MODE\_TEXT\_X, MODE\_TEXT\_Y + 35))

    def run(self):

        self.clock.tick(FPS)

        if not self.handle\_events():

            return False

        self.update()

        self.draw()

        pygame.display.flip()

        return True

## А.3 Модуль сцены игрового режима scenes/falling\_piano\_scene.py

from utils.config import \*

from scenes.piano import PianoScene

from utils.buttons import ImageButton

from utils.io\_operation import \*

class FallingPianoScene(PianoScene):

    def \_\_init\_\_(self, screen, clock):

        super().\_\_init\_\_(screen, clock)

        self.play\_icon = load\_icon("pic/play.png")

        self.stop\_icon = load\_icon("pic/stop.png")

        self.falling\_notes = []

        self.fall\_speed = FALLING\_SPEED

        self.is\_playing = False

        self.all\_melodies = read\_json('melodies/mel.json')

        self.current\_melody\_index = 0

        self.current\_melody\_data = self.all\_melodies[self.current\_melody\_index]

        self.current\_note\_index = 0

        self.melody\_start\_time = 0

        self.next\_note\_time = 0

        self.hit\_zone\_top = PIANO\_START\_Y

        self.hit\_zone\_bottom = PIANO\_START\_Y + KEY\_HEIGHT

        self.start\_button = ImageButton(

            x=PIANO\_START\_X,

            y=GAME\_BUTTON\_Y,

            width=START\_BUTTON\_WIDTH,

            height=GAME\_BUTTON\_HEIGHT,

            text=None, color=BACKGROUND, hover\_color=BUTTON,

            sound\_path=BUTTON\_SOUND\_PATH,

            icon=self.play\_icon,

            icon\_size=(50, 50)

        )

        self.reset\_button = ImageButton(

            x=PIANO\_START\_X + START\_BUTTON\_WIDTH + GAME\_BUTTON\_SPACING,

            y=GAME\_BUTTON\_Y,

            width=GAME\_BUTTON\_WIDTH,

            height=GAME\_BUTTON\_HEIGHT,

            text="СБРОС", color=BACKGROUND, hover\_color=BUTTON,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.mel\_button = ImageButton(

            x=PIANO\_START\_X + START\_BUTTON\_WIDTH+GAME\_BUTTON\_WIDTH + GAME\_BUTTON\_SPACING \* 2,

            y=GAME\_BUTTON\_Y,

            width=GAME\_BUTTON\_WIDTH,

            height=GAME\_BUTTON\_HEIGHT,

            text=self.current\_melody\_data['name'],

            color=BACKGROUND,

            hover\_color=BUTTON,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.score = 0

        self.misses = 0

        self.pause\_start\_time = 0

    def switch\_melody(self):

        """Переключает на следующую мелодию по кругу"""

        self.current\_melody\_index = (self.current\_melody\_index + 1) % len(self.all\_melodies)

        self.current\_melody\_data = self.all\_melodies[self.current\_melody\_index]

        self.mel\_button.text = self.current\_melody\_data['name']

        if self.is\_playing:

            self.reset\_game()

    def reset\_game(self):

        """Сбрасывает игру с учетом текущей мелодии"""

        self.falling\_notes.clear()

        self.current\_note\_index = 0

        self.melody\_start\_time = 0

        self.next\_note\_time = 0

        self.pause\_start\_time = 0

        self.score = 0

        self.misses = 0

        self.is\_playing = False

        if self.play\_icon:

            self.start\_button.set\_icon(self.play\_icon, (50, 50))

    def add\_falling\_note(self, note\_name):

        """Добавляет падающую ноту"""

        if note\_name not in KEY\_LABELS:

            return

        note\_index = KEY\_LABELS.index(note\_name)

        note = {

            'note\_index': note\_index,

            'note\_name': note\_name,

            'x': PIANO\_START\_X + note\_index \* (KEY\_WIDTH + KEY\_SPACING),

            'y': FALLING\_NOTES\_START\_Y,

            'width': KEY\_WIDTH,

            'height': FALLING\_NOTE\_HEIGHT,

            'color': FALLING\_NOTE\_COLOR,

            'active': True,

            'hit': False

        }

        self.falling\_notes.append(note)

    def update\_falling\_notes(self):

        """Обновляет падающие ноты с учетом относительного времени"""

        if not self.is\_playing:

            return

        current\_time = pygame.time.get\_ticks()

        if self.current\_note\_index == 0 and self.melody\_start\_time == 0:

            self.melody\_start\_time = current\_time

            self.next\_note\_time = current\_time

        while (self.current\_note\_index < len(self.current\_melody\_data['notes']) and

               current\_time >= self.next\_note\_time):

            note\_data = self.current\_melody\_data['notes'][self.current\_note\_index]

            self.add\_falling\_note(note\_data['note'])

            if self.current\_note\_index + 1 < len(self.current\_melody\_data['notes']):

                next\_delay = self.current\_melody\_data['notes'][self.current\_note\_index + 1]['delay']

                self.next\_note\_time += next\_delay

            self.current\_note\_index += 1

        for note in self.falling\_notes[:]:

            note['y'] += self.fall\_speed

            note\_center\_y = note['y'] + note['height'] // 2

            if self.hit\_zone\_top <= note\_center\_y <= self.hit\_zone\_bottom:

                if self.key\_glow\_time[note['note\_index']] > 0:

                    note['hit'] = True

                    note['color'] = FALLING\_NOTE\_HIT\_COLOR

                    self.score += SCORE\_PER\_HIT

                    self.falling\_notes.remove(note)

            if note['y'] > HEIGHT:

                self.falling\_notes.remove(note)

                self.misses += 1

    def handle\_events(self):

        events = pygame.event.get()

        for event in events:

            if self.start\_button.handle\_event(event):

                if not self.is\_playing:

                    self.is\_playing = True

                    if self.stop\_icon:

                        self.start\_button.set\_icon(self.stop\_icon, (50, 50))

                    if self.current\_note\_index == 0:

                        self.melody\_start\_time = pygame.time.get\_ticks()

                        self.next\_note\_time = self.melody\_start\_time

                    else:

                        if self.pause\_start\_time > 0:

                            pause\_duration = pygame.time.get\_ticks() - self.pause\_start\_time

                            self.next\_note\_time += pause\_duration

                            self.pause\_start\_time = 0

                else:

                    self.is\_playing = False

                    self.pause\_start\_time = pygame.time.get\_ticks()

                    if self.play\_icon:

                        self.start\_button.set\_icon(self.play\_icon, (50, 50))

                continue

            elif self.reset\_button.handle\_event(event):

                self.reset\_game()

                continue

            elif self.mel\_button.handle\_event(event):

                self.switch\_melody()

                continue

        for event in events:

            pygame.event.post(event)

        return super().handle\_events()

    def update(self):

        super().update()

        self.update\_falling\_notes()

        mouse\_pos = pygame.mouse.get\_pos()

        self.start\_button.check\_hover(mouse\_pos)

        self.reset\_button.check\_hover(mouse\_pos)

        self.mel\_button.check\_hover(mouse\_pos)

    def draw\_additional(self):

        pygame.draw.rect(self.screen, NOTES\_AREA\_COLOR,

                         (PIANO\_START\_X, NOTES\_AREA\_START\_Y,

                          KEY\_COUNT \* (KEY\_WIDTH + KEY\_SPACING) - 5,

                          NOTES\_AREA\_HEIGHT))

        for i in range(KEY\_COUNT):

            x = (PIANO\_START\_X - KEY\_SPACING) + i \* (KEY\_WIDTH + KEY\_SPACING)

            pygame.draw.rect(self.screen, BACKGROUND,

                             (x, NOTES\_AREA\_START\_Y, KEY\_SPACING, NOTES\_AREA\_HEIGHT), 0)

        zone\_rect = pygame.Rect(

            PIANO\_START\_X,

            self.hit\_zone\_top,

            KEY\_COUNT \* (KEY\_WIDTH + KEY\_SPACING) - 5,

            self.hit\_zone\_bottom - self.hit\_zone\_top

        )

        s = pygame.Surface(zone\_rect.size, pygame.SRCALPHA)

        s.fill((\*HIT\_ZONE\_COLOR, HIT\_ZONE\_ALPHA))

        self.screen.blit(s, zone\_rect.topleft)

        pygame.draw.rect(self.screen, HIT\_ZONE\_BORDER\_COLOR, zone\_rect, 2)

        for note in self.falling\_notes:

            pygame.draw.rect(self.screen, note['color'],

                             (note['x'], note['y'], note['width'], note['height']), 0, 5)

            font = pygame.font.Font(None, FALLING\_NOTE\_FONT\_SIZE)

            text = font.render(note['note\_name'], True, FALLING\_NOTE\_TEXT\_COLOR)

            text\_rect = text.get\_rect(

                center=(note['x'] + note['width'] // 2,

                        note['y'] + note['height'] // 2)

            )

            self.screen.blit(text, text\_rect)

        self.start\_button.draw(self.screen)

        self.reset\_button.draw(self.screen)

        self.mel\_button.draw(self.screen)

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        score\_text = font.render(f"Счёт: {self.score}", True, SCORE\_COLOR)

        misses\_text = font.render(f"Промахи: {self.misses}", True, MISSES\_COLOR)

        total\_notes = len(self.current\_melody\_data['notes'])

        progress\_text = font.render(

            f"Прогресс: {self.current\_note\_index}/{total\_notes}",

            True, TEXT

        )

        self.screen.blit(score\_text, (TEXT\_OFFSET\_X + 160, TEXT\_OFFSET\_Y))

        self.screen.blit(misses\_text, (TEXT\_OFFSET\_X + 280 , TEXT\_OFFSET\_Y))

        self.screen.blit(progress\_text, (TEXT\_OFFSET\_X, TEXT\_OFFSET\_Y))

        instruction = font.render(INSTRUCTION\_TEXT, True, TEXT)

        self.screen.blit(instruction, (PIANO\_START\_X, INSTRUCTION\_POS\_Y))

        self.draw\_mode\_game()

    def draw\_mode\_game(self):

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        mode\_text = font.render(CURRENT\_MODE\_GAME[1], True, TEXT)

        self.screen.blit(mode\_text, (MODE\_TEXT\_X, MODE\_TEXT\_Y + 35))

    def draw(self):

        super().draw()

        self.draw\_additional()

## А.4 Модуль сцены записи мелодий scenes/save\_mode.py

from datetime import datetime

from utils.config import \*

from scenes.piano import PianoScene

from utils.buttons import ImageButton

from utils.io\_operation import \*

import wave

import numpy as np

class MelodyRecorderScene(PianoScene):

    """Сцена для записи мелодий пользователя"""

    def \_\_init\_\_(self, screen, clock):

        super().\_\_init\_\_(screen, clock)

        self.all\_melodies = read\_json('melodies/mel.json')

        self.current\_melody\_index = 0

        self.list\_scroll\_offset = 0

        if self.all\_melodies:

            self.current\_melody\_data = self.all\_melodies[self.current\_melody\_index]

        else:

            self.current\_melody\_data = {'name': 'Нет мелодий', 'flag': 0, 'notes': []}

        self.recording = False

        self.record\_start\_time = 0

        self.recorded\_notes = []

        self.record\_button.text = "ВЕРНУТЬСЯ"

        os.makedirs("melodies", exist\_ok=True)

        self.up\_button = ImageButton(

            x=MELODY\_LIST\_X + MELODY\_LIST\_WIDTH - 40,

            y=MELODY\_LIST\_Y,

            width=35,

            height=35,

            text="↑",

            color=BUTTON,

            hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.down\_button = ImageButton(

            x=MELODY\_LIST\_X + MELODY\_LIST\_WIDTH - 40,

            y=MELODY\_LIST\_Y + MELODY\_LIST\_HEIGHT - 35,

            width=35,

            height=35,

            text="↓",

            color=BUTTON,

            hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        button\_x = RECORD\_BUTTON\_X

        button\_y\_start = RECORD\_BUTTON\_Y + (BUTTON\_HEIGHT + KEY\_SPACING) \* 2

        self.record\_control\_button = ImageButton(

            x=button\_x, y=button\_y\_start,

            width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT,

            text="НАЧАТЬ ЗАПИСЬ",

            color=BUTTON, hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.save\_button = ImageButton(

            x=button\_x, y=button\_y\_start + (BUTTON\_HEIGHT + KEY\_SPACING) \* 1,

            width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT,

            text="СОХРАНИТЬ",

            color=BUTTON, hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.convert\_wav\_button = ImageButton(

            x=button\_x, y=button\_y\_start + (BUTTON\_HEIGHT + KEY\_SPACING) \* 2,

            width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT,

            text="СОХРАНИТЬ .WAV",

            color=BUTTON, hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.play\_button = ImageButton(

            x=button\_x, y=button\_y\_start + (BUTTON\_HEIGHT + KEY\_SPACING) \* 3,

            width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT,

            text="ВОСПРОИЗВЕСТИ",

            color=BUTTON, hover\_color=KEYS\_PIANO,

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.del\_button = ImageButton(

            x=button\_x, y=button\_y\_start + (BUTTON\_HEIGHT + KEY\_SPACING) \* 4,

            width=BUTTON\_WIDTH, height=BUTTON\_HEIGHT,

            text="УДАЛИТЬ",

            color=(180, 60, 60), hover\_color=(220, 80, 80),

            sound\_path=BUTTON\_SOUND\_PATH

        )

        self.control\_buttons = [

            self.record\_control\_button,

            self.save\_button,

            self.convert\_wav\_button,

            self.play\_button,

            self.del\_button,

        ]

        self.need\_switch\_to\_normal = False

        self.last\_note\_time = 0

    def convert\_json\_to\_wav(self, melody\_data=None, filename=None):

        if melody\_data is None:

            melody\_data = self.current\_melody\_data

        if not melody\_data.get('notes'):

            return False

        try:

            os.makedirs("melodies", exist\_ok=True)

            if filename is None:

                timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

                filename = f"melodies/{melody\_data['name']}\_{timestamp}.wav"

            sample\_rate = 44100

            channels = 1

            current\_time = 0

            max\_duration = 0

            for note in melody\_data['notes']:

                current\_time += note['delay']

                max\_duration = max(max\_duration, current\_time + 2000)

            total\_duration = max\_duration + 1000

            total\_samples = int(sample\_rate \* total\_duration / 1000)

            audio\_buffer = np.zeros(total\_samples, dtype=np.float32)

            current\_time = 0

            for note\_info in melody\_data['notes']:

                current\_time += note\_info['delay']

                note\_name = note\_info['note']

                if note\_name in self.current\_sounds and self.current\_sounds[note\_name]:

                    sound = self.current\_sounds[note\_name]

                    sound\_array = pygame.sndarray.array(sound)

                    if len(sound\_array.shape) == 2:

                        sound\_array = sound\_array.mean(axis=1)

                    sound\_float = sound\_array.astype(np.float32) / 32768.0

                    start\_sample = int(current\_time \* sample\_rate / 1000)

                    end\_sample = min(start\_sample + len(sound\_float), total\_samples)

                    if start\_sample < total\_samples:

                        length = end\_sample - start\_sample

                        audio\_buffer[start\_sample:end\_sample] += sound\_float[:length]

            max\_val = np.max(np.abs(audio\_buffer))

            if max\_val > 0:

                audio\_buffer = audio\_buffer / max\_val

            audio\_int16 = (audio\_buffer \* 32767).astype(np.int16)

            with wave.open(filename, 'w') as wav\_file:

                wav\_file.setnchannels(channels)

                wav\_file.setsampwidth(2)

                wav\_file.setframerate(sample\_rate)

                wav\_file.writeframes(audio\_int16.tobytes())

            return True

        except Exception as e:

            print(f"✗ Ошибка: {e}")

            import traceback

            traceback.print\_exc()

            return False

    def scroll\_list(self, direction):

        if direction == "up" and self.list\_scroll\_offset > 0:

            self.list\_scroll\_offset -= 1

        elif direction == "down":

            max\_offset = max(0, len(self.all\_melodies) - MAX\_VISIBLE\_MELODIES)

            if self.list\_scroll\_offset < max\_offset:

                self.list\_scroll\_offset += 1

    def select\_melody(self, index):

        """Выбрать мелодию по индексу"""

        if 0 <= index < len(self.all\_melodies):

            self.current\_melody\_index = index

            self.current\_melody\_data = self.all\_melodies[index]

            return True

        return False

    def select\_melody\_by\_click(self, mouse\_pos):

        """Выбрать мелодию кликом мыши"""

        list\_rect = pygame.Rect(MELODY\_LIST\_X, MELODY\_LIST\_Y,

                                MELODY\_LIST\_WIDTH, MELODY\_LIST\_HEIGHT)

        if list\_rect.collidepoint(mouse\_pos):

            rel\_y = mouse\_pos[1] - MELODY\_LIST\_Y

            clicked\_index = self.list\_scroll\_offset + (rel\_y // MELODY\_ITEM\_HEIGHT)

            if 0 <= clicked\_index < len(self.all\_melodies):

                self.select\_melody(clicked\_index)

                return True

        return False

    def start\_recording(self):

        """Начать запись новой мелодии"""

        self.recording = True

        self.record\_start\_time = pygame.time.get\_ticks()

        self.recorded\_notes = []

        self.last\_note\_time = 0

        self.record\_control\_button.text = "ОСТАНОВИТЬ"

    def stop\_recording(self):

        """Остановить запись"""

        if self.recording:

            self.recording = False

            self.record\_control\_button.text = "НАЧАТЬ ЗАПИСЬ"

            return True

        return False

    def toggle\_recording(self):

        """Переключить режим записи"""

        if not self.recording:

            self.start\_recording()

        else:

            self.stop\_recording()

    def record\_note(self, note\_name):

        """Записать нажатую ноту с текущим временем"""

        if not self.recording:

            return

        current\_time = pygame.time.get\_ticks()

        time\_since\_start = current\_time - self.record\_start\_time

        note\_data = {

            'note': note\_name,

            'time': time\_since\_start,

            'delay': time\_since\_start - self.last\_note\_time if self.recorded\_notes else 0

        }

        self.recorded\_notes.append(note\_data)

        self.last\_note\_time = time\_since\_start

    def save\_melody(self):

        """Сохранить записанную мелодию в JSON файл"""

        if not self.recorded\_notes:

            return False

        notes\_with\_delay = []

        prev\_time = 0

        for note in self.recorded\_notes:

            delay = note['time'] - prev\_time

            notes\_with\_delay.append({

                'note': note['note'],

                'delay': delay

            })

            prev\_time = note['time']

        timestamp = datetime.now().strftime("%H%M%S")

        melody\_data = {

            'name': f"{timestamp}",

            'flag': 0,

            'notes': notes\_with\_delay

        }

        add\_mel\_json('melodies/mel.json', melody\_data)

        self.all\_melodies = read\_json('melodies/mel.json')

        if self.all\_melodies:

            self.select\_melody(len(self.all\_melodies) - 1)

        return True

    def convert\_current\_to\_wav(self):

        """Конвертировать текущую мелодию в WAV"""

        if self.all\_melodies and self.current\_melody\_index < len(self.all\_melodies):

            melody = self.all\_melodies[self.current\_melody\_index]

            if melody.get('notes'):

                self.convert\_json\_to\_wav(melody)

            else:

                print("У выбранной мелодии нет нот!")

        else:

            print("Нет выбранной мелодии!")

    def play\_selected\_melody(self):

        if not self.all\_melodies or self.current\_melody\_index >= len(self.all\_melodies):

            return

        melody = self.all\_melodies[self.current\_melody\_index]

        notes = melody.get('notes', [])

        if not notes:

            return

        for note\_data in notes:

            delay = note\_data.get('delay', 0)

            if delay > 0:

                pygame.time.delay(delay)

            note\_name = note\_data.get('note', '')

            if note\_name in self.current\_sounds and self.current\_sounds[note\_name]:

                self.current\_sounds[note\_name].play()

    def delete\_selected\_melody(self):

        """Удалить выбранную мелодию"""

        if not self.all\_melodies or self.current\_melody\_index >= len(self.all\_melodies):

            return False

        melody\_name = self.all\_melodies[self.current\_melody\_index].get('name', '')

        if delete\_melody('melodies/mel.json', melody\_name):

            self.all\_melodies = read\_json('melodies/mel.json')

            if self.all\_melodies:

                self.current\_melody\_index = min(self.current\_melody\_index, len(self.all\_melodies) - 1)

                self.current\_melody\_data = self.all\_melodies[self.current\_melody\_index]

            else:

                self.current\_melody\_index = 0

                self.current\_melody\_data = {'name': 'Нет мелодий', 'flag': 0, 'notes': []}

            return True

        return False

    def handle\_events(self):

        """Обработка событий с добавлением записи"""

        events = pygame.event.get()

        for event in events:

            if event.type == pygame.QUIT:

                return False

            if self.record\_button.handle\_event(event):

                self.need\_switch\_to\_normal = True

                continue

            if self.mode\_button\_game.handle\_event(event):

                continue

            if self.up\_button.handle\_event(event):

                self.scroll\_list("up")

                continue

            if self.down\_button.handle\_event(event):

                self.scroll\_list("down")

                continue

            if self.record\_control\_button.handle\_event(event):

                self.toggle\_recording()

                continue

            if self.save\_button.handle\_event(event):

                self.save\_melody()

                continue

            if self.convert\_wav\_button.handle\_event(event):

                self.convert\_current\_to\_wav()

                continue

            if self.play\_button.handle\_event(event):

                self.play\_selected\_melody()

                continue

            if self.del\_button.handle\_event(event):

                self.delete\_selected\_melody()

                continue

            if self.mode\_button.handle\_event(event):

                self.switch\_mode()

                continue

            if self.exit\_button.handle\_event(event):

                pygame.time.delay(130)

                return False

            if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

                self.select\_melody\_by\_click(pygame.mouse.get\_pos())

            elif event.type == pygame.KEYDOWN:

                note\_name = self.get\_note\_from\_key(event.key)

                if note\_name:

                    self.play\_note(note\_name)

                    if self.recording:

                        self.record\_note(note\_name)

        return True

    def get\_note\_from\_key(self, key):

        """Определяет имя ноты по нажатой клавише"""

        note\_mapping = {

            pygame.K\_s: 'S',

            pygame.K\_d: 'D',

            pygame.K\_f: 'F',

            pygame.K\_g: 'G',

            pygame.K\_h: 'H',

            pygame.K\_j: 'J',

            pygame.K\_k: 'K',

        }

        return note\_mapping.get(key)

    def play\_note(self, note\_name):

        """Воспроизвести ноту"""

        if note\_name in self.current\_sounds and self.current\_sounds[note\_name]:

            self.current\_sounds[note\_name].play()

        if note\_name in KEY\_LABELS:

            index = KEY\_LABELS.index(note\_name)

            self.key\_glow\_time[index] = pygame.time.get\_ticks()

    def update(self):

        """Обновление состояния"""

        super().update()

        mouse\_pos = pygame.mouse.get\_pos()

        self.up\_button.check\_hover(mouse\_pos)

        self.down\_button.check\_hover(mouse\_pos)

        for button in self.control\_buttons:

            button.check\_hover(mouse\_pos)

    def draw\_melody\_list(self):

        """Отрисовка списка мелодий"""

        pygame.draw.rect(self.screen, MELODY\_LIST\_BG\_COLOR,

                         (MELODY\_LIST\_X, MELODY\_LIST\_Y,

                          MELODY\_LIST\_WIDTH, MELODY\_LIST\_HEIGHT))

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        title = font.render("СПИСОК МЕЛОДИЙ:", True, TEXT)

        self.screen.blit(title, (MELODY\_LIST\_X + 10, MELODY\_LIST\_Y - 30))

        start\_idx = self.list\_scroll\_offset

        end\_idx = min(start\_idx + MAX\_VISIBLE\_MELODIES, len(self.all\_melodies))

        for i in range(start\_idx, end\_idx):

            melody = self.all\_melodies[i]

            y\_pos = MELODY\_LIST\_Y + (i - start\_idx) \* MELODY\_ITEM\_HEIGHT

            if i == self.current\_melody\_index:

                pygame.draw.rect(self.screen, MELODY\_SELECTED\_COLOR,

                                 (MELODY\_LIST\_X, y\_pos, MELODY\_LIST\_WIDTH - 40, MELODY\_ITEM\_HEIGHT))

                text\_color = TEXT

            else:

                text\_color = MELODY\_NORMAL\_COLOR

            name = melody.get('name')

            text = font.render(name, True, text\_color)

            self.screen.blit(text, (MELODY\_LIST\_X + 20, y\_pos + 5))

        self.up\_button.draw(self.screen)

        self.down\_button.draw(self.screen)

    def draw(self):

        """Отрисовка сцены"""

        super().draw()

        self.draw\_melody\_list()

        for button in self.control\_buttons:

            button.draw(self.screen)

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        if self.recording:

            record\_text = font.render("ЗАПИСЬ...", True, (255, 0, 0))

            self.screen.blit(record\_text, (WIDTH - 150, 20))

        y\_offset = MELODY\_LIST\_Y + MELODY\_LIST\_HEIGHT + 40

        if self.recorded\_notes:

            notes\_text = font.render(

                f"Записано нот: {len(self.recorded\_notes)}",

                True, TEXT

            )

            self.screen.blit(notes\_text, (MELODY\_LIST\_X, y\_offset))

        else:

            no\_notes\_text = font.render("Нет записанных нот", True, (200, 200, 200))

            self.screen.blit(no\_notes\_text, (MELODY\_LIST\_X, y\_offset))

        if self.all\_melodies and self.current\_melody\_index < len(self.all\_melodies):

            melody = self.all\_melodies[self.current\_melody\_index]

            note\_count = len(melody.get('notes', []))

            melody\_type = "Встроенная" if melody.get('flag', 0) == 1 else "Пользовательская"

            selected\_text = font.render(

                f"Выбрано: {melody.get('name', '')}",

                True, TEXT

            )

            info\_text1 = font.render(

                f"Нот: {note\_count} | Тип: {melody\_type}",

                True, TEXT

            )

            self.screen.blit(selected\_text, (MELODY\_LIST\_X + MELODY\_LIST\_WIDTH + 20,

                                             RECORD\_BUTTON\_Y + (BUTTON\_HEIGHT + KEY\_SPACING) \* 7))

            self.screen.blit(info\_text1, (MELODY\_LIST\_X + MELODY\_LIST\_WIDTH + 20,

                                          RECORD\_BUTTON\_Y + (BUTTON\_HEIGHT + KEY\_SPACING) \* 8))

        self.draw\_mode\_game()

    def draw\_mode\_game(self):

        font = pygame.font.SysFont(FONT, SIZE\_TEXT)

        mode\_text = font.render(CURRENT\_MODE\_GAME[2], True, TEXT)

        self.screen.blit(mode\_text, (MODE\_TEXT\_X, MODE\_TEXT\_Y + 35))

    def run(self):

        """Главный цикл сцены"""

        self.clock.tick(FPS)

        if not self.handle\_events():

            return False

        self.update()

        self.draw()

        pygame.display.flip()

        return True

## А.5 Модуль сцены авторизации scenes/license\_scene.py

import pygame

from utils.config import \*

from utils.buttons import ImageButton

from utils.auth\_system import AuthSystem

class LicenseScene:

    def \_\_init\_\_(self, screen, clock):

        self.screen = screen

        self.clock = clock

        self.auth\_system = AuthSystem()

        self.input\_code = ""

        self.submit\_button = ImageButton(

            WIDTH // 2 - 100, HEIGHT // 2 + 100, 200, 50, "ПРОВЕРИТЬ",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH,)

        self.exit\_button = ImageButton(

            WIDTH // 2 - 100, HEIGHT // 2 + 170, 200, 50, "ВЫЙТИ",

            BACKGROUND, BUTTON, BUTTON\_SOUND\_PATH

        )

        self.font = pygame.font.SysFont(FONT, 36)

        self.small\_font = pygame.font.SysFont(FONT, 24)

        self.verified = False

        self.message = ""

        self.show\_demo\_info = True

    def handle\_events(self):

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                return False

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_RETURN:

                    self.check\_code()

                elif event.key == pygame.K\_BACKSPACE:

                    self.input\_code = self.input\_code[:-1]

                elif event.key in [pygame.K\_0, pygame.K\_1, pygame.K\_2, pygame.K\_3,

                                   pygame.K\_4, pygame.K\_5, pygame.K\_6, pygame.K\_7,

                                   pygame.K\_8, pygame.K\_9]:

                    key\_map = {

                        pygame.K\_0: '0', pygame.K\_1: '1', pygame.K\_2: '2',

                        pygame.K\_3: '3', pygame.K\_4: '4', pygame.K\_5: '5',

                        pygame.K\_6: '6', pygame.K\_7: '7', pygame.K\_8: '8',

                        pygame.K\_9: '9'

                    }

                    self.input\_code += key\_map[event.key]

                elif event.unicode.isdigit():  # Только цифры

                    if len(self.input\_code) < 4:

                        self.input\_code += event.unicode

            if self.submit\_button.handle\_event(event):

                self.check\_code()

            if self.exit\_button.handle\_event(event):

                return False

        return True

    def check\_code(self):

        """Проверка кода доступа"""

        if self.auth\_system.check\_code(self.input\_code):

            self.verified = True

            self.show\_demo\_info = False

            self.message = "Доступ разрешен!"

        else:

            self.message = "Неверный код доступа"

    def update(self):

        mouse\_pos = pygame.mouse.get\_pos()

        self.submit\_button.check\_hover(mouse\_pos)

        self.exit\_button.check\_hover(mouse\_pos)

    def draw(self):

        self.screen.fill(BACKGROUND)

        title = self.font.render("ПРОВЕРКА ДОСТУПА", True, TEXT)

        self.screen.blit(title, (WIDTH // 2 - title.get\_width() // 2, 100))

        instr = self.small\_font.render("Введите код:", True, (200, 200, 200))

        self.screen.blit(instr, (WIDTH // 2 - instr.get\_width() // 2, 170))

        input\_bg = pygame.Rect(WIDTH // 2 - 150, 220, 300, 80)

        pygame.draw.rect(self.screen, (30, 30, 40), input\_bg, border\_radius=10)

        pygame.draw.rect(self.screen, (100, 100, 150), input\_bg, 3, border\_radius=10)

        display\_code = "•" \* len(self.input\_code)

        code\_text = self.font.render(display\_code, True, (255, 255, 255))

        self.screen.blit(code\_text, (input\_bg.centerx - code\_text.get\_width() // 2,

                                     input\_bg.centery - code\_text.get\_height() // 2))

        if self.message:

            msg\_text = self.font.render(self.message, True, (255, 0, 0))

            self.screen.blit(msg\_text, (WIDTH // 2 - msg\_text.get\_width() // 2, 360))

        self.submit\_button.draw(self.screen)

        self.exit\_button.draw(self.screen)

        if self.show\_demo\_info:

            demo\_text = self.small\_font.render("КОД можно получить в тг @barnaeva\_m за оплату подписки", True, (150, 150, 150))

            self.screen.blit(demo\_text, (WIDTH // 2 - demo\_text.get\_width() // 2, HEIGHT - 50))

    def run(self):

        self.clock.tick(FPS)

        if not self.handle\_events():

            return False

        self.update()

        self.draw()

        pygame.display.flip()

        return True

## А.6 Модуль проверки лицензии utils/auth\_system.py

import hashlib

from utils.config import CODE

class AuthSystem:

    def \_\_init\_\_(self):

        self.code\_hash = CODE

    def check\_code(self, input\_code):

        input\_hash = hashlib.sha256(input\_code.encode()).hexdigest()

        return input\_hash == self.code\_hash

## А.7 Модуль конфигурации utils/config.py

WIDTH = 900

HEIGHT = 800

FPS = 30

BACKGROUND = (50, 50, 50)

BUTTON = (100, 100, 100)

KEYS\_PIANO = (200, 200, 200)

TEXT = (255, 255, 255)

KEY\_COUNT = 7

KEY\_LABELS = ['S', 'D', 'F', 'G', 'H', 'J', 'K']

KEY\_WIDTH = 100

KEY\_HEIGHT = 200

KEY\_SPACING = 5

PIANO\_START\_X = (WIDTH - (KEY\_COUNT \* KEY\_WIDTH + (KEY\_COUNT - 1) \* KEY\_SPACING)) // 2

PIANO\_START\_Y = HEIGHT \* 0.65

KEY\_TEXT\_OFFSET\_X = KEY\_WIDTH // 2 - 5

KEY\_TEXT\_OFFSET\_Y = KEY\_HEIGHT // 2 - 10

GLOW\_DURATION = 300

BUTTON\_WIDTH = 230

BUTTON\_HEIGHT = 35

EXIT\_BUTTON\_WIDTH = 150

EXIT\_BUTTON\_HEIGHT = 35

MODE\_BUTTON\_X = PIANO\_START\_X

MODE\_BUTTON\_Y = HEIGHT \* 0.05

MODE\_GAME\_BUTTON\_X = PIANO\_START\_X

MODE\_GAME\_BUTTON\_Y = HEIGHT \* 0.1

EXIT\_BUTTON\_X = WIDTH - EXIT\_BUTTON\_WIDTH - PIANO\_START\_X

EXIT\_BUTTON\_Y = HEIGHT \* 0.05

RECORD\_BUTTON\_X = WIDTH - BUTTON\_WIDTH - PIANO\_START\_X

RECORD\_BUTTON\_Y = HEIGHT \* 0.1

MODE\_TEXT\_X = MODE\_BUTTON\_X + BUTTON\_WIDTH + KEY\_SPACING

MODE\_TEXT\_Y = HEIGHT \* 0.06

GAME\_BUTTON\_WIDTH = 120

GAME\_BUTTON\_HEIGHT = 37

GAME\_BUTTON\_SPACING = 5

START\_BUTTON\_WIDTH = 100

GAME\_BUTTON\_X = MODE\_BUTTON\_X

GAME\_BUTTON\_Y = 210 - KEY\_SPACING - GAME\_BUTTON\_HEIGHT

FALLING\_NOTES\_START\_Y = 200

FALLING\_NOTE\_HEIGHT = 50

FALLING\_SPEED = 5

NOTES\_AREA\_START\_Y = 210

NOTES\_AREA\_HEIGHT = 310

NOTES\_AREA\_COLOR = (230, 230, 230)

HIT\_ZONE\_ALPHA = 100

HIT\_ZONE\_COLOR = (170, 200, 170)

HIT\_ZONE\_BORDER\_COLOR = (170, 200, 170)

FALLING\_NOTE\_COLOR = BUTTON

FALLING\_NOTE\_HIT\_COLOR = (0, 255, 0)

FALLING\_NOTE\_TEXT\_COLOR = TEXT

FALLING\_NOTE\_FONT\_SIZE = 24

SCORE\_PER\_HIT = 100

SCORE\_COLOR = (0, 255, 0)

MISSES\_COLOR = (255, 0, 0)

TEXT\_OFFSET\_X = PIANO\_START\_X + KEY\_SPACING

TEXT\_OFFSET\_Y = HEIGHT \* 0.15

SCORE\_POS\_X\_OFFSET = 250

MISSES\_POS\_X\_OFFSET = 400

PROGRESS\_POS\_Y = 100

INSTRUCTION\_TEXT = "Нажимайте клавиши когда ноты в зеленой зоне!"

INSTRUCTION\_POS\_Y = HEIGHT - 70

MODE\_GAME\_TEXT = " ПАДАЮЩИЕ НОТЫ"

MEL\_BUTTON\_TEXT = "Гуси"

MELODY\_LIST\_X = PIANO\_START\_X

MELODY\_LIST\_Y = 150

MELODY\_LIST\_WIDTH = 400

MELODY\_LIST\_HEIGHT = 300

MELODY\_ITEM\_HEIGHT = 30

MAX\_VISIBLE\_MELODIES = 10

MELODY\_LIST\_BG\_COLOR = (70, 70, 70)

MELODY\_SELECTED\_COLOR = (100, 150, 200)

MELODY\_NORMAL\_COLOR = TEXT

SOUND\_SETTINGS = {

    'frequency': 22050,

    'size': -16,

    'channels': 2,

    'buffer': 512

}

NUM\_CHANNELS = 16

BUTTON\_SOUND\_PATH = "../virtual\_piano/sounds/click.wav"

SOUND\_PATHS = {

    "1": {

        'S': "sounds/grandpiano/do.wav",

        'D': "sounds/grandpiano/re.wav",

        'F': "sounds/grandpiano/mi.wav",

        'G': "sounds/grandpiano/fa.wav",

        'H': "sounds/grandpiano/salt.wav",

        'J': "sounds/grandpiano/la.wav",

        'K': "sounds/grandpiano/si.wav"

    },

    "2": {

        'S': "sounds/piano/do.wav",

        'D': "sounds/piano/re.wav",

        'F': "sounds/piano/mi.wav",

        'G': "sounds/piano/fa.wav",

        'H': "sounds/piano/salt.wav",

        'J': "sounds/piano/la.wav",

        'K': "sounds/piano/si.wav"

    },

    "3": {

        'S': "sounds/bam/do.wav",

        'D': "sounds/bam/re.wav",

        'F': "sounds/bam/mi.wav",

        'G': "sounds/bam/fa.wav",

        'H': "sounds/bam/salt.wav",

        'J': "sounds/bam/la.wav",

        'K': "sounds/bam/si.wav"

    }

}

CURRENT\_MODE = {"1": 'РОЯЛЬ', "2": 'ПИАНИНО', "3": 'УДАРНЫЕ'}

CURRENT\_MODE\_GAME = ["СИНТЕЗАТОР", "ПАДАЮЩИЕ НОТЫ", "ЗАПИСЬ МЕЛОДИИ"]

FONT = 'timesnewroman'

SIZE\_TEXT = 23

DEFAULT\_WAV\_FILENAME = "my\_melody.wav"

WAV\_EXPORT\_PATH = "../virtual\_piano/melodies/"

LICENSE\_INPUT\_X = WIDTH // 2 - 200

LICENSE\_INPUT\_Y = 220

LICENSE\_INPUT\_WIDTH = 400

LICENSE\_INPUT\_HEIGHT = 60

CODE="c6eaf20a01893966ad8466d523a9acbe655af849b802fef67abfca1db04af1df"

## А.8 Модуль работы с файлами utils/io\_opertion.py

import json

import pygame

import os

def read\_json(filename: str) -> dict:

    """

    Read a JSON file and return a dictionary.

    :param filename: The name of the JSON file.

    :return: A dictionary with the data.

    """

    try:

        with open(filename, "r", encoding="utf-8") as file:

            return json.load(file)

    except FileNotFoundError:

        print(f"File {filename} not found.")

        return {}

    except json.JSONDecodeError as e:

        print(f"JSON format error in file {filename}: {e}")

        return {}

    except Exception as exc:

        print(f"Error reading JSON: {exc}")

        return {}

def write\_json(filename: str, data: dict) -> None:

    """

    Write data to a JSON file.

    :param filename: The name of the file to write.

    :param data: The data to write.

    """

    try:

        os.makedirs("melodies", exist\_ok=True)

        with open(filename, "w", encoding="utf-8") as file:

            json.dump(data, file, indent=4, ensure\_ascii=False)

    except PermissionError:

        print(f"No permission to write to file {filename}.")

    except Exception as exc:

        print(f"Error writing JSON: {exc}")

def add\_mel\_json(filename: str, melody\_data: dict) -> bool:

    """Добавить сохраненную мелодию в указанный файл как элемент массива"""

    try:

        folder = os.path.dirname(filename)

        if folder:

            os.makedirs(folder, exist\_ok=True)

        if not os.path.exists(filename):

            melodies = [melody\_data]

            print(f"Создан новый файл с первой мелодией: {filename}")

        else:

            with open(filename, 'r', encoding='utf-8') as f:

                content = json.load(f)

            if isinstance(content, list):

                melodies = content

                melodies.append(melody\_data)

            elif isinstance(content, dict):

                melodies = [content, melody\_data]

                print(f"Преобразовано в массив: {filename}")

            else:

                melodies = [melody\_data]

                print(f"Неизвестный формат, создан новый массив: {filename}")

        with open(filename, 'w', encoding='utf-8') as f:

            json.dump(melodies, f, ensure\_ascii=False, indent=2)

        print(f"Добавлено в {filename}, всего: {len(melodies)} мелодий")

        return True

    except Exception as e:

        print(f"Ошибка добавления в {filename}: {e}")

        return False

def update\_melody\_name(filename, old\_name, new\_name):

    """Обновляет имя мелодии в JSON файле"""

    try:

        with open(filename, 'r', encoding='utf-8') as file:

            melodies = json.load(file)

        for melody in melodies:

            if melody.get('name') == old\_name:

                melody['name'] = new\_name

                break

        with open(filename, 'w', encoding='utf-8') as file:

            json.dump(melodies, file, ensure\_ascii=False, indent=2)

        return True

    except Exception as e:

        print(f"Ошибка обновления имени мелодии: {e}")

        return False

def load\_icon(filename: str):

    """Загружает иконку из файла"""

    try:

        icon = pygame.image.load(filename).convert\_alpha()

        return icon

    except Exception as e:

        print(f"Не удалось загрузить иконку {filename}: {e}")

        return None

def delete\_melody(filename: str,melody\_name: dict):

    """

    Удаляет мелодию по имени из файла mel.json

    Args:

        melody\_name: Имя мелодии для удаления (например, "Гуси")

    Returns:

        True если удалено, False если нет

    """

    try:

        with open(filename, 'r', encoding='utf-8') as f:

            melodies = json.load(f)

        initial\_count = len(melodies)

        kept\_melodies = [

            m for m in melodies

            if not (m.get("name") == melody\_name and m.get("flag") == 0)

        ]

        deleted\_count = initial\_count - len(kept\_melodies)

        if deleted\_count == 0:

            print(f"Мелодия '{melody\_name}' не найдена или системная")

            return False

        with open(filename, 'w', encoding='utf-8') as f:

            json.dump(kept\_melodies, f, ensure\_ascii=False, indent=2)

        return True

    except Exception as e:

        print(f"Error delete melody: {e}")

        return False

## А.9 Модуль кнопок utils/button.py

import pygame

from utils.config import SIZE\_TEXT, FONT

class ImageButton:

    def \_\_init\_\_(self, x, y, width, height, text=None, color=None, hover\_color=None,

                 sound\_path=None, icon=None, icon\_size=None):

        """

        Создание кнопки.

        Параметры:

        - text: текст кнопки (может быть None если только иконка)

        - color: цвет фона (может быть None если только иконка)

        - icon: объект pygame.Surface с иконкой

        - icon\_size: размер иконки (ширина, высота)

        """

        self.x = x

        self.y = y

        self.width = width

        self.height = height

        self.text = text

        self.color = color

        self.hover\_color = hover\_color if hover\_color else color

        self.icon = icon

        self.icon\_size = icon\_size or (height - 10, height - 10)

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

        self.sound = None

        if sound\_path:

            self.sound = pygame.mixer.Sound(sound\_path)

        self.is\_hovered = False

        if self.icon and self.icon\_size:

            self.icon = pygame.transform.smoothscale(self.icon, self.icon\_size)

    def set\_icon(self, icon, size=None):

        """Устанавливает или меняет иконку кнопки"""

        self.icon = icon

        if size:

            self.icon\_size = size

        if self.icon and self.icon\_size:

            self.icon = pygame.transform.smoothscale(self.icon, self.icon\_size)

    def set\_text(self, text):

        """Меняет текст кнопки"""

        self.text = text

    def draw(self, screen):

        if self.color:

            current\_color = self.hover\_color if self.is\_hovered else self.color

            pygame.draw.rect(screen, current\_color, self.rect, border\_radius=12)

            border\_color = (min(current\_color[0] + 30, 255),

                            min(current\_color[1] + 30, 255),

                            min(current\_color[2] + 30, 255))

            pygame.draw.rect(screen, border\_color, self.rect, 2, border\_radius=12)

        if self.icon:

            icon\_rect = self.icon.get\_rect(center=self.rect.center)

            if self.text:

                icon\_rect.x = self.rect.x + 10

                icon\_rect.centery = self.rect.centery

            screen.blit(self.icon, icon\_rect)

        if self.text:

            font = pygame.font.SysFont(FONT, SIZE\_TEXT)

            text\_surface = font.render(self.text, True, (255, 255, 255))

            if self.icon:

                text\_x = self.rect.x + (self.icon\_size[0] if self.icon else 0) + 20

                text\_rect = text\_surface.get\_rect(midleft=(text\_x, self.rect.centery))

            else:

                text\_rect = text\_surface.get\_rect(center=self.rect.center)

            screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.is\_hovered = self.rect.collidepoint(mouse\_pos)

    def handle\_event(self, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1 and self.is\_hovered:

            if self.sound:

                self.sound.play()

            pygame.event.post(pygame.event.Event(pygame.USEREVENT, button=self))

            return True

        return False

## А.10 Точка входа в программу main.py

import pygame

from core.game import Game

def main():

    try:

        game = Game()

        game.run()

    except Exception as e:

        print(f"Ошибка запуска: {e}")

        import traceback

        traceback.print\_exc()

        input("Нажмите Enter для выхода...")

if \_\_name\_\_ == "\_\_main\_\_":

    main()