МИНИСТЕРСТВА 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, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

КУРСОВОЙ ПРОЕКТ ПО ДИСЦИПЛИНЕ

«Технологии и методы программирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

«Виртуально пианино»

|  |  |
| --- | --- |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | М.А. Замотохина |
|  |  |
| Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | Е.В. Мясников |
|  |  |

САМАРА 2023

МИНИСТЕРСТВА 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, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Студенту ***Замотохиной Марии Андреевне*** группы 6311

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

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты практики | Содержание задания |
| ОПК-3 ‑ способность применять языки, системы и инструментальные средства программирования в профессиональной деятельности | Знать: основные виды программных средств, технологию разработки алгоритмов и программ и методы их отладки, основы объектно-ориентированного подхода к программированию  Уметь: работать с современными системами программирования, самостоятельно осваивать новые программные средства  Владеть: языками процедурного и объектно-ориентированного программирования, навыками разработки и отладки программ. | 1. Изучение существующих методов создания графического приложения  2. Разработка методов реализации виртуального пианино  3. Изучение существующих библиотек для создания игр на языке Python  4. Программная реализация методов реализации виртуального пианино  5. Разработка программы виртуального пианино  6. Отладка и тестирование разработанной программы. |

Дата выдачи задания 16 октября 2023 г.

Срок представления на кафедру отчета о практике 22 декабря 2023 г.

Руководитель курсового проекта

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

*(подпись)*

Задание принял к исполнению

студент группы № \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Замотохина

*(подпись)*

РЕФЕРАТ

**Пояснительная записка к курсовому проекту:** 39 c., 14 рисунков, 0 таблиц, 7 источников, 1 приложение.

ВИРТУАЛЬНОЕ ПИАНИНО, ФОРМАТ MIDI, ИГРА НА PYTHON, ЗАПИСЬ ЗВУКОВ, ВОСПРОИЗВЕДЕНИЕ ЗВУКОВ.

Цель работы – создание оконного приложения игры виртуальное пианино.

Было выполнено сравнение существующих языков программирования для создания игр. Благодаря простоте и наличию большого количества библиотек был выбран язык программирования Python.

Был выполнен обзор существующих библиотек для создания игр, записи и воспроизведения звуков в формате midi на языке Python. Благодаря наличию большого количества обучающего материала были выбраны библиотеки Pygame и mido. Изучены алгоритмы реализации виртуального пианино, записи и воспроизведения на выбранном языке.

Для работы была выбрана среда разработки Visual Studio Code. В качестве операционной системы выступает Windows 10.

Для работы использовался набор звуков пианино в формате .wav.

Содержание

[РЕФЕРАТ 3](#_Toc154131515)

[Содержание 4](#_Toc154131516)

[ВВЕДЕНИЕ 5](#_Toc154131517)

[1 МЕТОДЫ/АЛГОРИТМЫ/БИБЛИОТЕКИ/ ПРОГРАММНЫЕ СРЕДСТВА 7](#_Toc154131518)

[1.1 Анализ адаптации под функционал пианино персонального компьютера. 7](#_Toc154131519)

[1.2 Приложения аналоги. 7](#_Toc154131520)

[1.3 Описание алгоритмов реализации виртуального пианино. 10](#_Toc154131521)

[1.4 Существующие библиотеки создания игр на языке Python. 10](#_Toc154131522)

[1.5 Разработка алгоритмов реализации виртуального пианино. 11](#_Toc154131523)

[1.6 Выводы и результаты 12](#_Toc154131524)

[2 ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 13](#_Toc154131525)

[2.1 Выбор средств разработки и системных программных средств 13](#_Toc154131526)

[2.2 Требования к разрабатываемой программе 13](#_Toc154131527)

[2.3 Структура программы 13](#_Toc154131528)

[2.3.1 Описание основных структур данных 14](#_Toc154131529)

[2.3.2 Описание основных функций и процедур 15](#_Toc154131530)

[2.4 Описание интерфейса пользователя 19](#_Toc154131531)

[2.5 Выводы и результаты 24](#_Toc154131532)

[ЗАКЛЮЧЕНИЕ 25](#_Toc154131533)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc154131534)

[ПРИЛОЖЕНИЕ А 27](#_Toc154131535)

ВВЕДЕНИЕ

Виртуальное пианино имеет высокую актуальность в современном мире по нескольким причинам:

Доступность и удобство: виртуальные пианино обеспечивают доступ к музыкальному обучению и творчеству в любое время и в любом месте.

* Обучение: Виртуальные пианино предоставляют возможность для обучения игре на пианино тем, кто не имеет доступа к настоящему инструменту или профессиональным учителям. Благодаря готовым наборам нот и удобному интерфейсу каждый может самостоятельно научиться играть простые мелодии.
* Творчество: Виртуальные пианино предоставляют средства для творческого процесса, включая композицию и запись музыки. Они могут быть использованы как для профессиональных музыкантов, так и для любителей, чтобы создавать свою музыку.
* Развлечение: Виртуальные пианино предлагают развлекательные возможности для тех, кто ценит музыку, но не обладает навыками игры на реальном инструменте.

Сегодня существует много различных виртуальных пианино, доступные как на телефоне, так и компьютере, обладающих различным функционалом, в том числе запись, воспроизведение, поддержка разных музыкальных инструментов.

Цель работы – создание оконного приложения игры виртуальное пианино.

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

1. Изучение существующих методов создания графического приложения;
2. Разработка методов реализации виртуального пианино;
3. Изучение существующих библиотек для создания игр на языке Python;
4. Программная реализация методов реализации виртуального пианино;
5. Разработка программы виртуального пианино;
6. Реализация защиты программы лицензионным ключом;
7. Отладка и тестирование разработанной программы.

# МЕТОДЫ/АЛГОРИТМЫ/БИБЛИОТЕКИ/ ПРОГРАММНЫЕ СРЕДСТВА

## Анализ адаптации под функционал пианино персонального компьютера.

Реализовать пианино на компьютере можно несколькими различными способами. Во-первых, может различаться способ игры. Воспроизводить звук можно посредством нажатия клавиши или с использованием компьютерной мыши. В данной работе были использованы оба этих метода для создания наиболее полного функционала.

Во-вторых, есть выбор между реализацией полного набора клавиш, т.е. возможностью нажатия клавиши в любой из девяти октав, и некоторого отрезка фортепианной клавиатуры, например четыре основные октавы. В данной работе был реализован второй метод – у пользователя есть возможность воспроизвести звуки из четырех основных октав: большой, малой, первой и второй.

В-третьих, из-за наличия в фортепианной клавиатуре «повышенных звуков» (черные клавиши) можно по-разному связать клавишу и воспроизводимый звук. Первый способ заключается в использовании клавиши Shift для повышения/понижения звука на пол тона. Недостатком данного метода является неудобство воспроизведения аккордов, так как при нажатии нескольких клавиш будут повышать все соответствующие ноты. Второй способ заключается в определения точной зависимости клавиша – нота, т.е. каждой клавише будет соответствовать 1 нота. В данной работе был реализован второй метод из-за ввода ограничения на количество доступных октав.

## Приложения аналоги.

Самое популярное приложение для телефона – «Perfect Piano». В нем доступны режимы свободной игры, записи, игры по нотам в разных режимах, выбор инструмента и многое другое. Также доступны записи игры людей по всему миру и возможность обучения на этих записях.

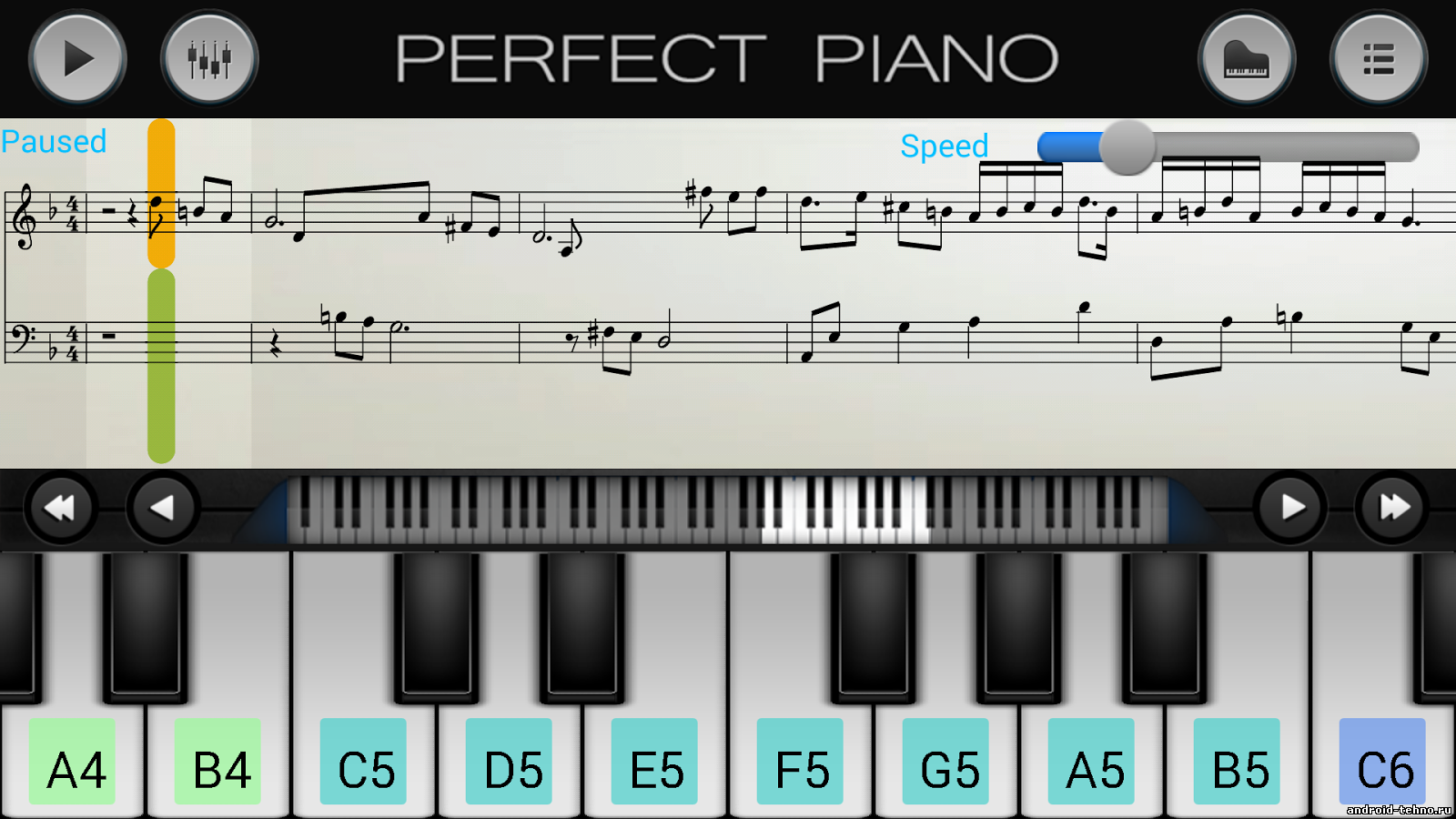


Рисунок – Интерфейс приложения «Perfect Piano»

Приложение «Synthesia» позволяет играть на пианино с помощью компьютерной мыши. В нем доступен выбор разных инструментов, встроенный метроном и игра по нотам.

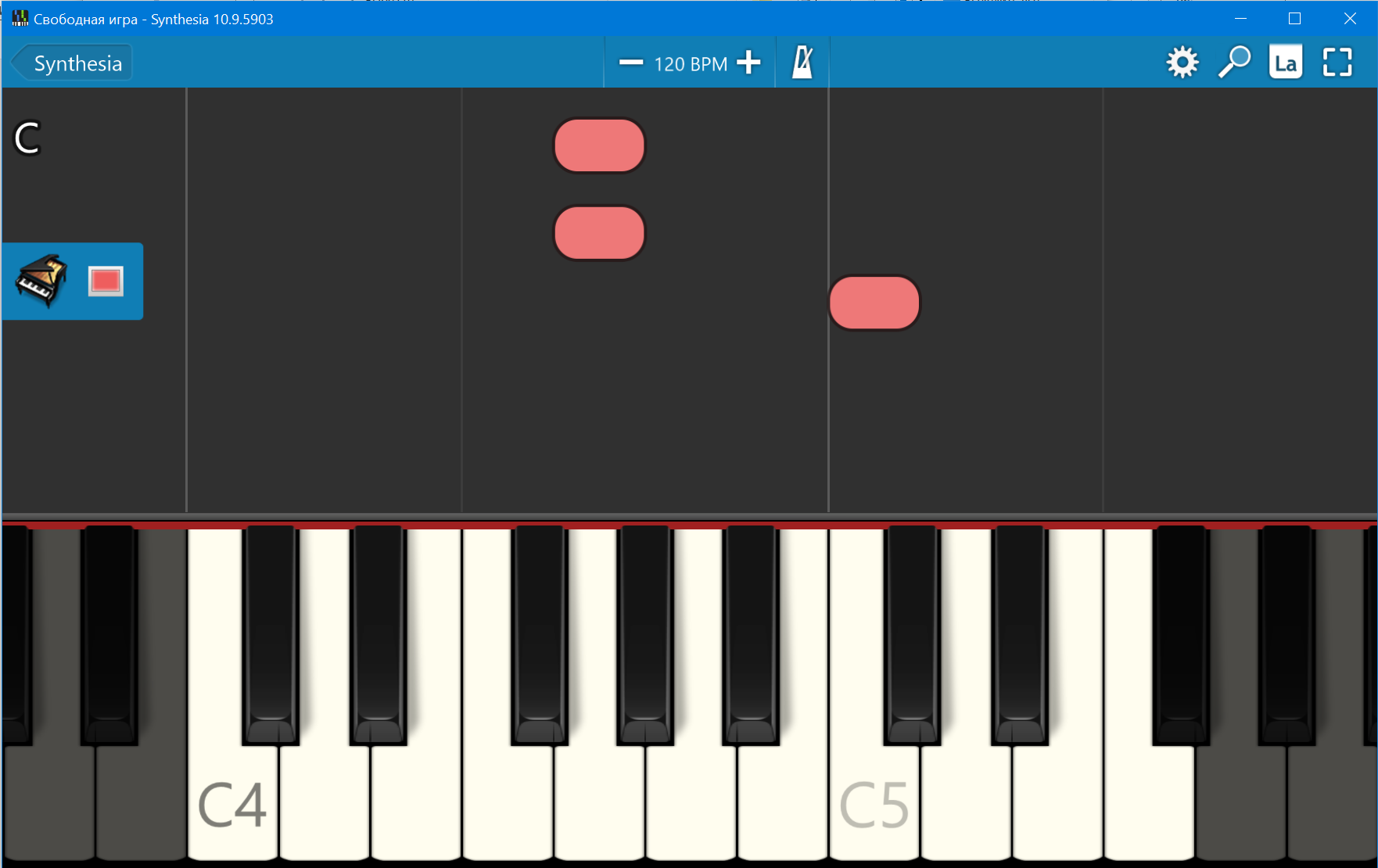


Рисунок - Интерфейс приложения «Synthesia»

Приложение «EveryonePiano» предоставляет широкий выбор настроек клавиатуры, режим записи и воспроизведения, игры с помощью клавиатуры и мыши, а также выбор нескольких языков интерфейса. При установке плагинов доступны игра «Угадай звук».



Рисунок - Интерфейс приложения «EveryonePiano»

## Описание алгоритмов реализации виртуального пианино.

Для создание наиболее красивого и полного графического интерфейса были разработаны следующие алгоритмы:

1. Алгоритм вывода на экран фортепианной клавиатуры;
2. Алгоритм вывод экран компьютерной клавиатуры с соответствием клавиша – нота;
3. Алгоритм создания эффекта нажатой клавиши;
4. Алгоритм записи сыгранных нот;
5. Алгоритм ограничения функционала за счет ввода лицензионного ключа.

## Существующие библиотеки создания игр на языке Python.

Существует несколько библиотек для создания игр на языке Python:

1. Библиотека для создания графического интерфейса PySimpleGUI.

Данная библиотека объединяет в себе несколько библиотек для создания графичекого интерфейса: Tkinter, Qt, Remi, WxPython. Благодаря удобному синтаксису и обширным методам с помощью этой библиотеки можно в несколько строк создать такие игры, как шахматы, змейка, тетрис, пианино.

1. Arcade — движок Python для создания игр с современными графикой и звуком, разработанный профессором Полом Крэйвеном из Симпсон-колледжа (Айова, США). В ней поддерживаются современная OpenGL-графика, аннотации типов Python 3, имеет встроенные физические движки для игр с видом сверху и игр-платформеров.
2. Pygame - определяет множество классов, которые инкапсулируют концепции, не зависящие от аппаратного обеспечения. В результате игры и мультимедийные программы запускаются в любых системах, где можно установить библиотеку. Для данной библиотеки множество понятной и подробной документации, поэтому она подойдет даже новичкам.

Для данной работы была выбрана библиотека Pygame, так она довольно проста и понятна в изучении, а также поддерживает весь необходимый функционал.

## Разработка алгоритмов реализации виртуального пианино.

Основным алгоритмом в данной работе является алгоритм создания эффекта нажатой клавиши, который позволяет пользователю понять, какую клавишу он нажал, и воспроизвести соответствующий звук.

Для этого в массивы black\_sounds и white\_sounds записываются звуки необходимых черных и белых нот. При нажатии клавиши определяется, какая клавиша нажата, черная или белая, и в массивы активных нот active\_blacks и active\_whites записываются индекс ноты в массиве звуков, длительность воспроизведения ноты, индекс ноты в библиотеке midi для записи и флаг наличия или отсутствия записи.

В функциях отрисовки фортепианной и компьютерных клавиатур осуществляется проверка наличия нажатых клавиш. Если клавиша нажата, осуществляется несколько проверок:

1. Проверяется, не закончилось ли воспроизведение ноты (элемент active\_whites [i][1]). Если это так, то звук удаляется из массива. Если установлен флаг записи, то необходимый звук записывается в трек.
2. Если звук пока воспроизводится, то значение active\_whites [i][1] уменьшается на единицу и на экране на месте нажатой клавиши отображается контрастный прямоугольник, который имитирует нажатую клавишу.

## Выводы и результаты

Для создания виртуального пианино на языке Python используются библиотеки Pygame и Mido, которые подходят для реализации необходимого функционала. Пианино будет иметь неполную клавиатуру, а именно, большую, малую, первую и вторую октавы, которые являются основными, и единственное соответствие клавиша – нота для удобства игры сразу нескольких нот.

# ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Выбор средств разработки и системных программных средств

В данной работе используется язык программирования Python благодаря простоте синтаксиса и наличию большого количества библиотек, которые упрощают разработку и позволяют реализовать любую идею. Для создания виртуального пианино используются библиотеки Pygame, Pygame Widgets, Mido, так как они просты в использовании и обладают богатым функционалом.

Для разработки используется среда разработки Visual Studio Code, так она идеально подходит для разработки на языке Python благодаря динамическим подсказкам при написании кода. Выбранная операционная система – Windows 10.

## Требования к разрабатываемой программе

Разрабатываемая программа должна обладать следующим функционалом:

1. Воспроизведение звуков при нажатии клавиши или соответствующей ноты компьютерной мышкой;
2. Запись звуков и ее остановка;
3. Воспроизведение последнего записанного звука;
4. Ввод лицензионного ключа;
5. Выбор нот для обучения;

## Структура программы

Программа состоит из следующих файлов:

1. buttons.py – файл с функцией создания всех кнопок;
2. functions.py – файл со всеми методами, реализующими основной функционал программы;
3. output.mid – файл с записанной композицией;
4. piano\_list2.py – файл, содержащий основные структуры данных;
5. piano2.py – содержащий код оконного приложения;

Также в проекте есть папка assets, в которой содержатся все звуки всех фортепианных нот, иконки всех кнопок, шрифты и ноты для самостоятельной игры.

### Описание основных структур данных

В фале piano\_list2.py описаны следующие структуры данных:

1. piano\_notes – список с названием всех нот большой, малой, первой и второй октавы. Необходим для создания компьютерной клавиатуры, а именно для определения соответствия клавиша – нота;
2. white\_notes – список с названием всех белых клавиш большой, малой, первой и второй октавы. Необходим для создания фортепианной клавиатуры и получения индекса для воспроизведения звуков;
3. black\_flats – список с названием всех черных клавиш большой, малой, первой и второй октавы. Необходим для создания фортепианной клавиатуры и получения индекса для воспроизведения звуков;
4. key\_list – список всех клавиш, нажатие которых воспроизводит звук. Необходимо для того, чтобы при нажатии клавиши, не отвечающих за звук не было ошибок;
5. white\_button\_list – словарь с индексами белых нот и расположением клавиши в компьютерной клавиатуры. Необходимо для создания эффекта нажатой клавиши;
6. black\_button\_list – словарь с индексами черных нот и расположением клавиши в компьютерной клавиатуры. Необходимо для создания эффекта нажатой клавиши;
7. button\_list – список с названиями всех клавиш компьютерной клавиатуры. Необходим для создания компьютерной клавиатуры.
8. notes\_dict – словарь с соответствием нота – клавиша. Необходимо для воспроизведения звуков;
9. midi\_notes – словарь с соответствием нота – индекс звука в библиотеке mido. Необходимо для записи звуков;
10. learn\_samples – словарь с соответствием название песни – путь к файлу с нотами;
11. licence\_keys – список лицензионных ключей.

### Описание основных функций и процедур

В файле **buttons.py** содержится следующая функция:

1. create\_buttons(screen, WIDTH, HEIGHT) – функция создания кнопок, реализующий основной функционал. В ней описаны положения, цвета кнопок и их иконки.

В файле **functions.py** содержатся следующие функции:

1. get\_sounds(white\_notes, black\_flats) – функция получения звуков белых и черных клавиш из папки;
2. gradientRect( window, left\_colour, right\_colour, target\_rect ) – функция рисования градиентных прямоугольников, которые появляются при нажатии клавиши;
3. draw\_piano(active\_whites, active\_blacks, screen, HEIGHT, WIDTH, track, sec, mins) – функция рисования фортепианной клавиатуры. При нажатии клавиши на экране отображается, какая клавиша была нажата путем изменения ее цвета и появлением градиентного прямоугольника над ней. При изменении размера окна клавиши автоматически меняют размер;
4. draw\_keyboard(active\_white, active\_black, screen, HEIGHT, WIDTH, view) – функция рисования клавиатуры компьютера, если на экране не отображаются ноты. При нажатии клавиши на экране отображается, какая клавиша была нажата путем изменения ее цвета. Клавиши, которые не используются для нажатия звуков окрашены в серый незаметный цвет, чтобы сместить фокус внимания на активные клавиши;
5. record\_timer(screen, HEIGHT, WIDTH, if\_record, curr\_sec, sec, mins) – функция рисования таймера при записи композиции;
6. view\_sample(screen, WIDTH, HEIGHT, view, key\_learn) – функция вывода на экран нот для самостоятельной игры, если введен лицензионный ключ;
7. view\_lock(screen, WIDTH, HEIGHT, if\_lock) – функция вывода предупреждения о том, что лицензионный ключ не введен.
8. input\_key(screen, WIDTH, HEIGHT, if\_input, input\_key) – функция для рисования окна ввода лицензионного ключа;
9. check\_key(screen, WIDTH, HEIGHT, if\_check\_key, key) – если лицензионный ключ введен проверяется его правильность и выводится соответствующее информационное окно.

В файле **piano2.py** не содержится функций, но в нем реализован весь основной функционал, поэтому рассмотрим его подробно. В файле содержатся следующие важные переменные:

1. key\_list – список всех клавиш, нажатие которых воспроизводит звук. Необходимо для того, чтобы при нажатии клавиши, не отвечающих за звук не было ошибок;
2. piano\_notes\_key – словарь с соответствием нота – клавиша. Необходимо для воспроизведения звуков;
3. white\_notes\_label – список с названием всех белых клавиш большой, малой, первой и второй октавы. Необходим для создания фортепианной клавиатуры и получения индекса для воспроизведения звуков;
4. black\_flats\_label – список с названием всех черных клавиш большой, малой, первой и второй октавы. Необходим для создания фортепианной клавиатуры и получения индекса для воспроизведения звуков;
5. midi\_notes – словарь с соответствием нота – индекс звука в библиотеке mido. Необходимо для записи звуков;
6. white\_sounds – список со звуками всех белых клавиш;
7. black\_sounds – список со звуками всех черных клавиш;
8. active\_whites – список со всеми нажатыми белыми клавишами фортепианной клавиатуры;
9. active\_blacks – список со всеми нажатыми черными клавишами фортепианной клавиатуры;
10. active\_button\_white – список со всеми нажатыми белыми клавишами компьютерной клавиатуры;
11. active\_button\_black – список со всеми нажатыми черными клавишами компьютерной клавиатуры;
12. run – переменная, отвечающая за отображение окна;
13. if\_record – переменная, отвечающая за наличие/отсутствии записи. По умолчанию ее значение равно False, при нажатии кнопки «Начать запись» устанавливается значение True. Позволяет записывать звуки и отображать таймер.
14. if\_lock – переменная, отвечающая за блокировку функций вывода на экран нот для самостоятельной игры. По умолчанию ее значние равно True, после ввода верного лицензионного ключа, устанавливается значение False. Позволяет блокировать некоторые кнопки и выводить на экран сообщение с предупреждением о необходимости ввода лицензионного ключа;
15. if\_view – переменная, отвечающая за отображение нот для самостоятельного проигрывания. При нажатии кнопки «Показать ноты», ее значение с False меняется на True, благодаря чему скрывается компьютерная клавиатура и показываются ноты;
16. if\_input – переменная, отвечающая за ввод лицензионного ключа. По умолчанию ее значение равно False, после нажатия кнопки меняется на True. Позволяет вводить окно, демонстрирующее введенный лицензионный ключ;
17. key\_learn – переменная, отвечающая за выбор нот для отображения;
18. input\_key – введенный лицензионный ключ. До завершения ввода текста его значение равно пустой строке;
19. input\_text – вводимый с клавиатуры текст. После нажатия клавиши Enter, значение введенного текста копируется в input\_key и сбрасывается до пустой строки;
20. if\_check\_key – переменная, отвечающая за ввод информации о том, что ключ проверяется;
21. if\_key\_right – переменная, отвечающая за то, что введенный ключ верен;
22. sec – количество записанных секунд;
23. mins – количество записанных минут;
24. curr\_sec – текущее время в секундах (вычисляется с помощью библиотеки time);
25. mid – файл музыкального трека;
26. track – музыкальный трек, в который добавляются все записанные звуки.

Основной цикл while отвечает за отображение окна. В самом начале задается цвет фона, рисуются фортепианная и компьютерные клавиатуры, получаются значения записанных секунд и минут, если включена запись и отображаются ноты для самостоятельной игры, если установлен нужный флаг (строки 66-73 файла piano2.py приложения А).

Далее в строках 75 - 157 файла piano2.py приложения А определяется текущее положение мыши на экране и, если курсор наведен на одну из кнопок, показывает всплывающее с описанием функционала кнопки.

Далее в строках 160 – 283 файла piano2.py приложения А обрабатываются все события нажатия мышки или нажатия на клавишу. В строках 163 – 164 проверяется событие выхода окна. В строках 166 - 173 проверяется событие изменение размеров окна для дальнейшей адаптации всех элементов под размер окна. В строках 175 - 201 обрабатываются события нажатия на фортепианную клавишу мышкой и воспроизведение соответствующего звука. В строках 203 – 248 обрабатываются события нажатия всех действующих кнопок. В строках 250 – 288 обрабатываются события ввода лицензионного ключа и нажатия клавиши для игры на пианино.

В строке 286 файла piano2.py приложения А обновляются состояния всех кнопок (наведение, нажатие, отпускание).

В сроках 287 – 396 файла piano2.py приложения А обрабатываются события ввода ключа с клавиатуры и вывод предупреждающего сообщения н экран.

В строке 297 файла piano2.py приложения А обновляются все элементы окна.

В строке 299 файла piano2.py приложения А приложение закрывается.

## Описание интерфейса пользователя

При запуске программы открывается стартовое окно приложения. В нем можно увидеть фортепианную клавиатуру, расположенную внизу, компьютерную клавиатуру, расположенную в центральной части, и панель меню с кнопками, расположенную в верхней части.

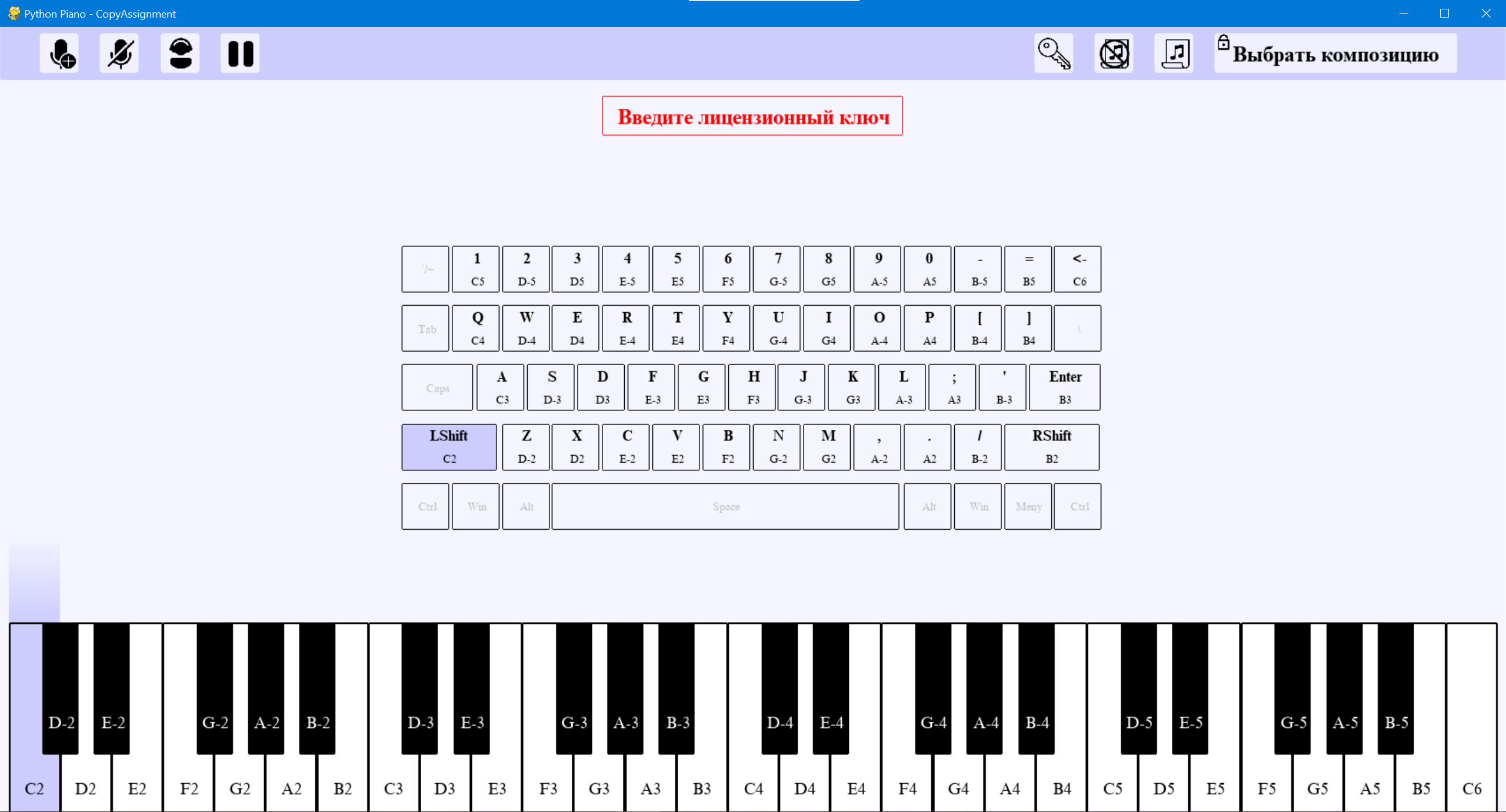


Рисунок – Стартовое окно программы

При нажатии соответствующих клавиш на клавиатуре или мышкой на экране появляются подсвеченные клавиши.

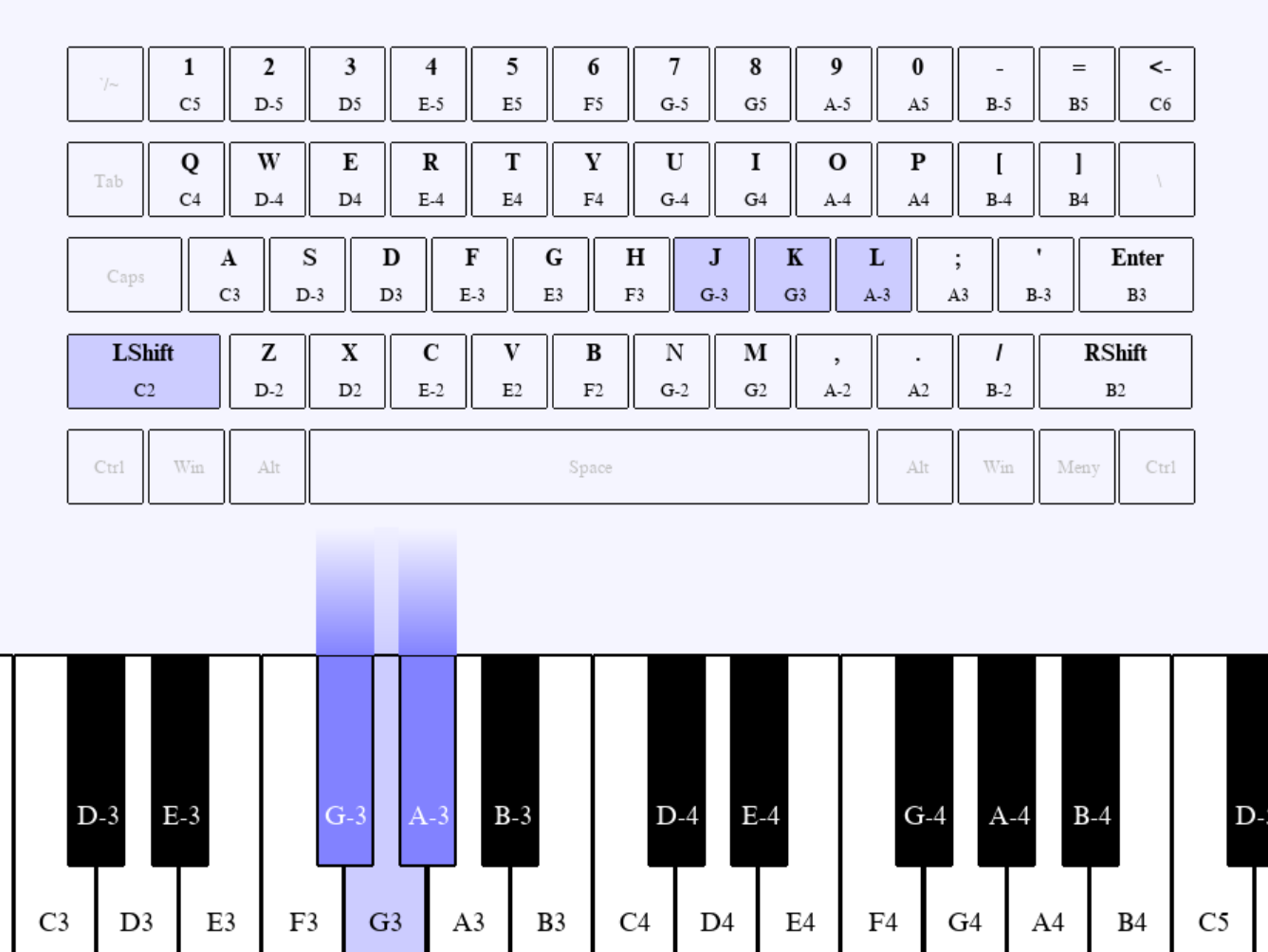


Рисунок – Пример игры на пианино

При нажатии кнопки «Начать запись» на экране появляется таймер, который считает время записанной композиции.

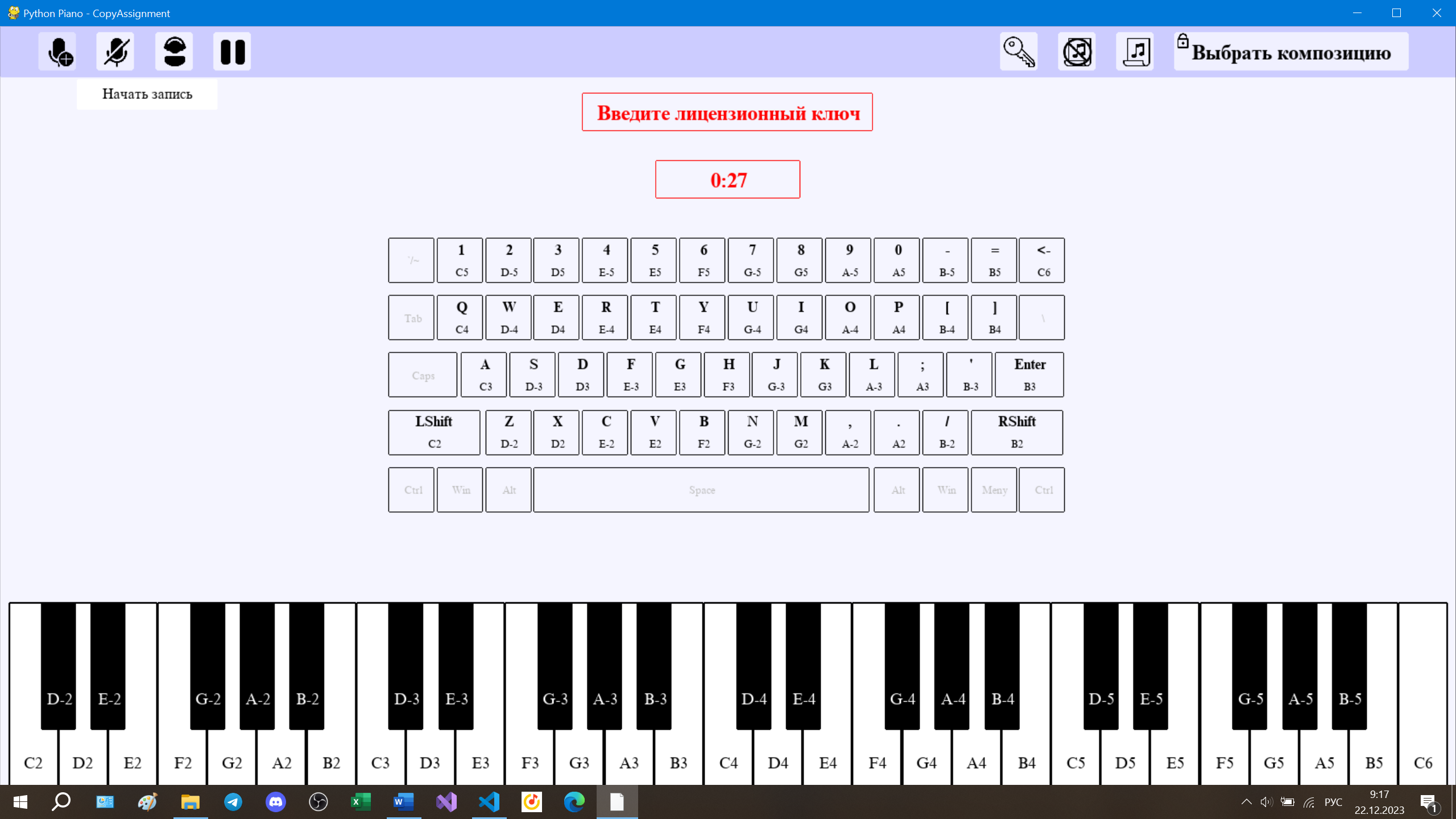


Рисунок – Активный режим записи и кнопка «Начать запись»

При нажатии кнопки «Остановить запись» таймер пропадает, и проигранная композиция записывается в файл.

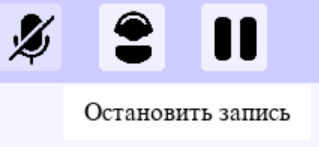


Рисунок – Кнопка «Остановить запись»

При нажатии кнопки «Воспроизвести музыку» воспроизводится последняя записанная композиция. При нажатии кнопки «Остановить воспроизведение», воспроизведение музыкального файла останавливается.

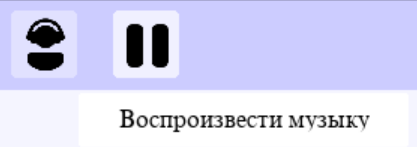


Рисунок – Кнопка «Воспроизвести музыку»

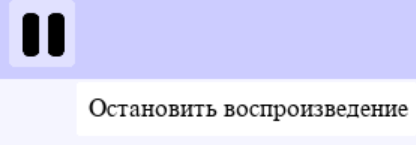


Рисунок – Кнопка «Остановить воспроизведение»

Если лицензионный ключ не введен, то на экране отображается предупреждение о необходимости ввода лицензионного ключа. Для того, чтобы ввести ключ необходимо нажать соответствующею кнопку. При нажатии отобразится область, в которой можно увидеть введенные символы.

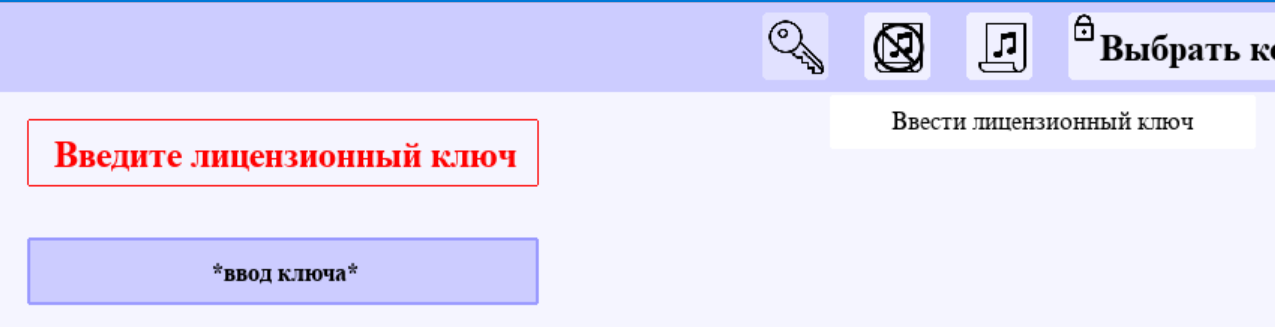


Рисунок – Пример ввода лицензионного ключа

Если введенный ключ неверен, на экране отобразится соответствующее предупреждение.

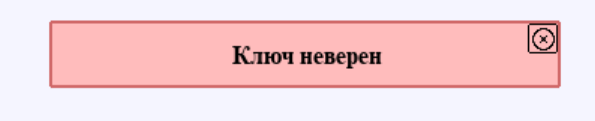


Рисунок 11 – Сообщение о неверно введенном ключе

При вводе верного ключа с экрана пропадает предупреждение.

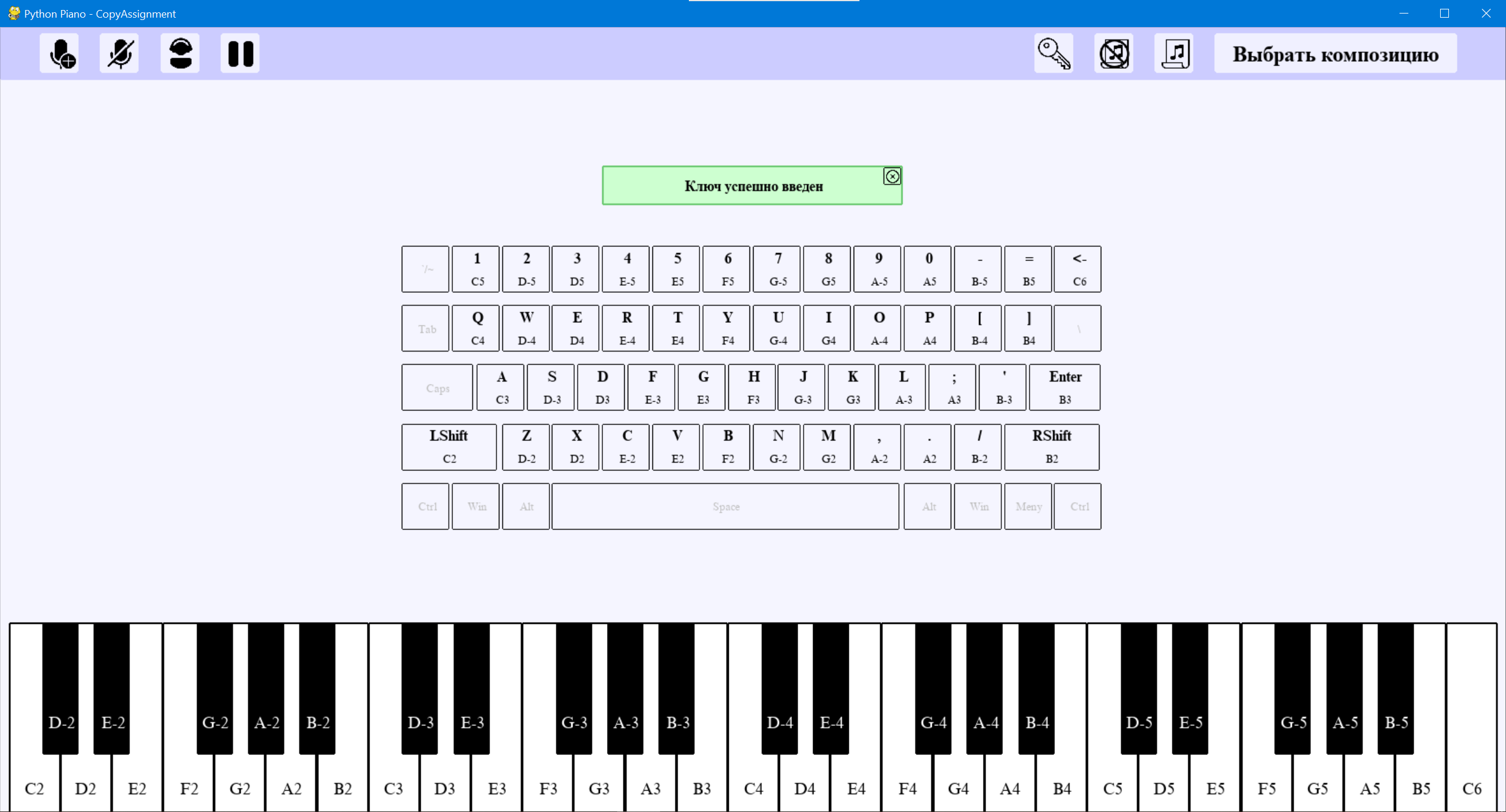


Рисунок 12 – Сообщение о верно введенном ключе

После ввода ключа доступны функции выбора композиции для проигрывания. Для того, чтобы отобразить на экране ноты, нужно выбрать композицию и нажать кнопку «Показать ноты».

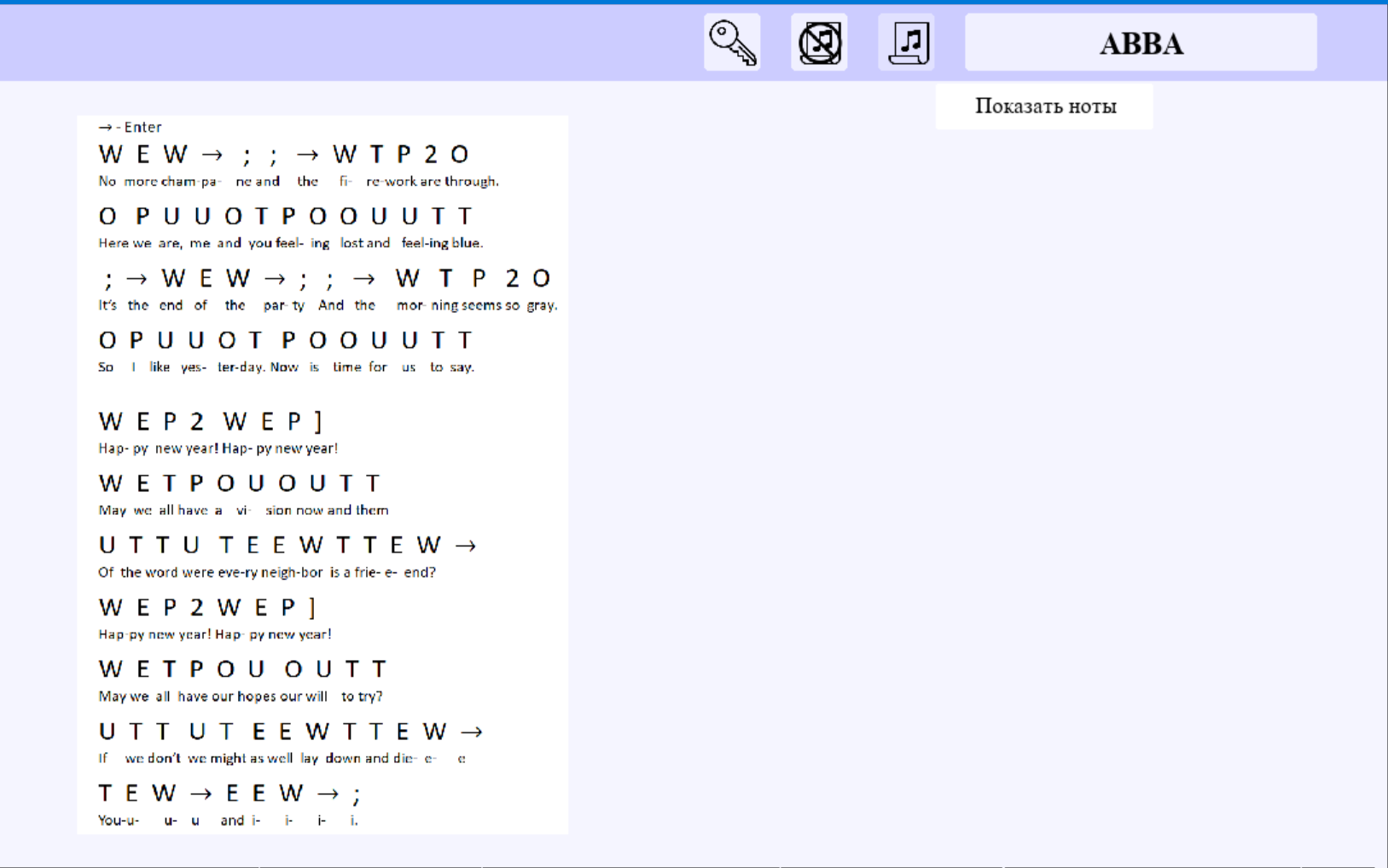


Рисунок – Отображение нот на экране

Чтобы скрыть ноты, необходимо нажать соответствующую кнопку.

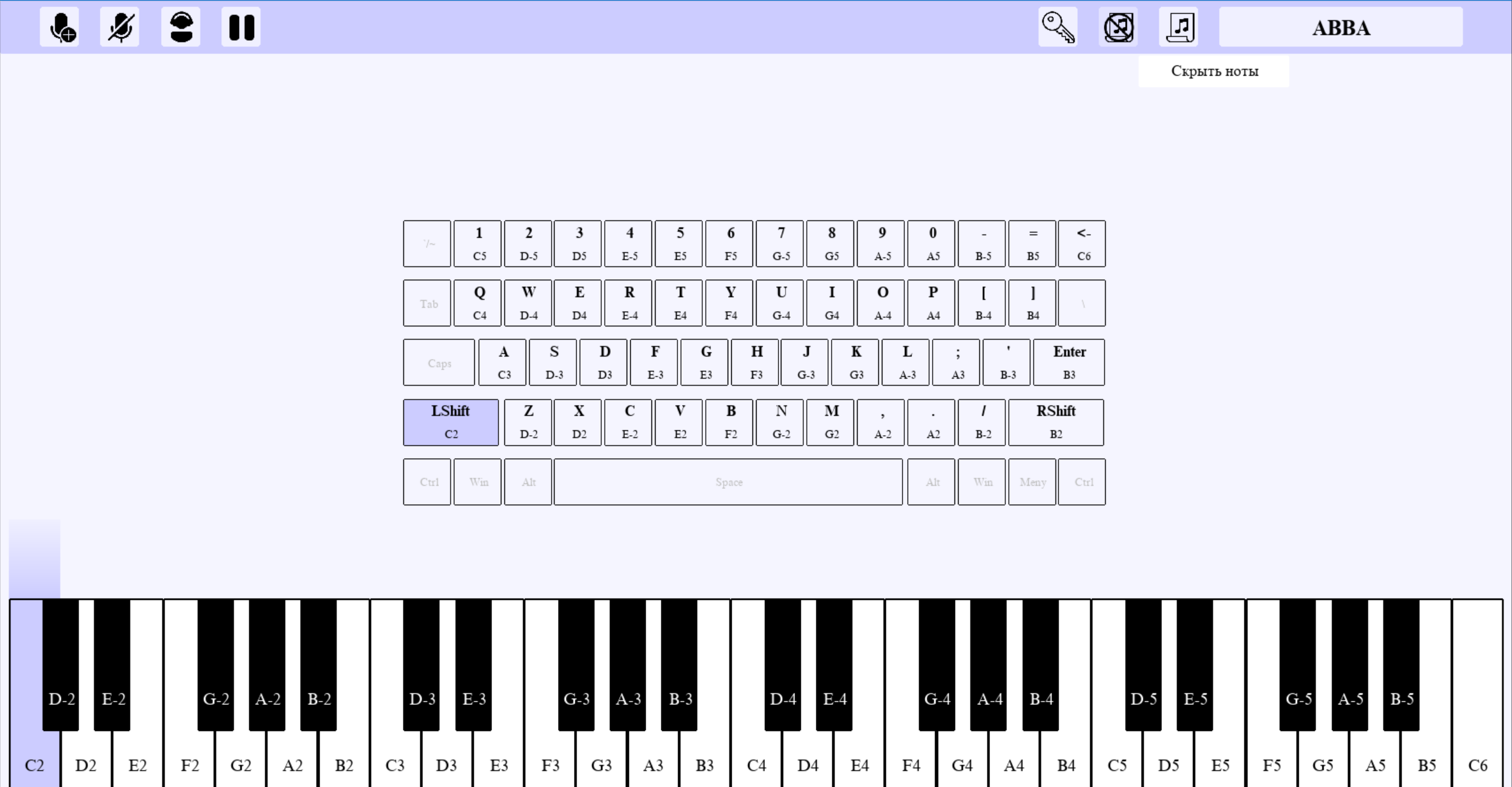


Рисунок – Действие кнопки «Скрыть ноты»

## Выводы и результаты

Было разработано приложение виртуального пианино, поддерживающего функции записи и воспроизведения записанной композиции. Осуществлен принцип защиты приложения с помощью лицензионного ключа.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы была изучены библиотека создания игр на языке Python Pygame и библиотека для работы с аудиофайлами формата .midi. Было разработано графическое приложение виртуального пианино, обладающего функционалом свободной игры, записи и воспроизведения аудиофайлов, а также игры по нотам. Часть функций ограничена лицензионным ключом, что удовлетворяет поставленным требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сopyassignment: [Электронный ресурс] // GUI Piano in Python . URL: https://copyassignment.com/gui-piano-in-python/ (Дата обращения:  1.12.2023)
2. Pygame: [Электронный ресурс] // Pygame documentation. URL: <https://pygame-docs.website.yandexcloud.net/ref/pygame.html> . Дата обращения: 1.12.2023)
3. Хабр: [Электронный ресурс] // Создание игр на Python 3 и Pygame: Часть 1. URL: https://habr.com/ru/articles/347138/. Дата обращения: 1.12.2023)
4. Хабр: [Электронный ресурс] // Библиотека Pygame / Часть 1. Введение. URL: https://habr.com/ru/articles/588605/. Дата обращения: 10.12.2023)
5. Pygame Widgets: [Электронный ресурс] // Pygame Widgets. URL: <https://pygamewidgets.readthedocs.io/en/latest/> Дата обращения: 10.12.2023)
6. Mido: [Электронный ресурс] // Mido - MIDI Objects for Python. URL: [https://mido.readthedocs.io/en/stable/#](https://mido.readthedocs.io/en/stable/) Дата обращения: 10.12.2023)
7. Mido Documentation: [Электронный ресурс] // Mido Documentation. URL: https://buildmedia.readthedocs.org/media/pdf/mido/latest/mido.pdf

ПРИЛОЖЕНИЕ А

Код программы

import pygame

from pygame import mixer

import piano\_list2 as pl2

import time

import mido

pygame.init()

timer\_font = pygame.font.Font('assets/timesnrcyrmt\_bold.ttf', 24)

#key\_font = pygame.font.Font('assets/timesnrcyrmt\_inclined.ttf', 16)

key\_font = pygame.font.Font('assets/timesnrcyrmt.ttf', 18)

button\_font = pygame.font.Font('assets/timesnrcyrmt.ttf', 12)

bold\_button\_font = pygame.font.Font('assets/timesnrcyrmt\_bold.ttf', 16)

lock\_image = pygame.image.load('assets/images/lock.png')

lock\_image = pygame.transform.scale(lock\_image, (20, 20))

close\_image = pygame.image.load('assets/images/close.png')

close\_image = pygame.transform.scale(close\_image, (20, 20))

button\_label = pl2.button\_list

piano\_notes\_label = pl2.piano\_notes

white\_notes\_label = pl2.white\_notes

black\_flats\_label = pl2.black\_flats

white\_button\_list = pl2.white\_button\_list

black\_button\_list = pl2.black\_button\_list

not\_note\_buttom = [0, 1, 2, 3, 4, 5, 6, 7, 20, 33, 46, 47]

clock = pygame.time.Clock()

learn\_samples = pl2.learn\_samples

license\_keys = pl2.licence\_keys

def get\_sounds(white\_notes, black\_flats):

white\_sounds = []

for i in range(len(white\_notes)):

try:

white\_sounds.append(mixer.Sound(f'assets\\notes\\{white\_notes[i]}.wav'))

except IOError:

print('File error')

black\_sounds = []

for i in range(len(black\_flats)):

try:

black\_sounds.append(mixer.Sound(f'assets\\notes\\{black\_flats[i]}.wav'))

except IOError:

print('File error')

return white\_sounds, black\_sounds

def gradientRect( window, left\_colour, right\_colour, target\_rect ):

""" Draw a horizontal-gradient filled rectangle covering <target\_rect> """

colour\_rect = pygame.Surface( ( 2, 2 ) ) # tiny! 2x2 bitmap

pygame.draw.line( colour\_rect, left\_colour, ( 0,0 ), ( 1,0 ) ) # left colour line

pygame.draw.line( colour\_rect, right\_colour, ( 0,1 ), ( 1,1 ) ) # right colour line

colour\_rect = pygame.transform.smoothscale( colour\_rect, ( target\_rect.width, target\_rect.height ) ) # stretch!

window.blit( colour\_rect, target\_rect ) # paint it

def draw\_piano(active\_whites, active\_blacks, screen, HEIGHT, WIDTH, track, sec, mins):

lenght\_key = (WIDTH - 20)/29

height\_key = HEIGHT/4

#pygame.draw.rect(screen, 'black', [10, HEIGHT - (8/5)\*height\_key, lenght\_key \* 29 + 4, (9/5)\*height\_key], 1, 2)

white\_rects = []

for i in range(len(white\_notes\_label)):

rect = pygame.draw.rect(screen, 'white', [10 + i \* lenght\_key, HEIGHT - height\_key, lenght\_key, height\_key], 0, 2)

white\_rects.append(rect)

i = 0

len\_white = len(active\_whites)

while i < len\_white and len\_white > 0:

if active\_whites[i][1] == 0:

if active\_whites[i][3] == 1:

track.append(mido.Message('note\_off', note=active\_whites[i][2], velocity=100, time=round((sec + time.time() % 1 + mins \* 60) \* 65)))

active\_whites.pop(i)

len\_white -= 1

elif active\_whites[i][1] > 0:

j = active\_whites[i][0]

pygame.draw.rect(screen, (204, 204, 255), [10 + j \* lenght\_key, HEIGHT - height\_key, lenght\_key, height\_key], 0, 2)

gradientRect(screen, (240, 240, 255), (204, 204, 255), pygame.Rect(10 + j \* lenght\_key, HEIGHT - (height\_key\*(7/5) - 1), lenght\_key, height\_key\*(2/5)))

active\_whites[i][1] -= 1

if len(active\_whites) > 10000:

active\_whites.clear()

if i < len\_white:

i +=1

for i in range(len(white\_notes\_label)):

pygame.draw.rect(screen, 'black', [10 + i \* lenght\_key, HEIGHT - height\_key, lenght\_key + 1, height\_key], 2, 2)

key\_label = key\_font.render(white\_notes\_label[i], True, 'black')

center = key\_label.get\_rect( center = (10 + i \* lenght\_key + lenght\_key / 2 , HEIGHT - height\_key\*(1/6)))

screen.blit(key\_label, center)

skip\_count = 0

last\_skip = 3

skip\_track = 0

black\_rects = []

for i in range(len(black\_flats\_label)):

rect = pygame.draw.rect(screen, 'black', [10 + lenght\_key\*(1/1.5 + i + skip\_count), HEIGHT - height\_key, lenght\_key/1.4, height\_key\*2/3], 0, 2)

black\_rects.append(rect)

q = 0

len\_black = len(active\_blacks)

while q < len\_black and len\_black > 0:

if active\_blacks[q][0] == i:

if active\_blacks[q][1] == 0:

if active\_blacks[q][3] == 1:

track.append(mido.Message('note\_off', note=active\_blacks[q][2], velocity=100, time=round((sec + time.time() % 1 + mins \* 60) \* 65)))

active\_blacks.pop(q)

len\_black -= 1

elif active\_blacks[q][1] > 0:

pygame.draw.rect(screen, (130, 130, 255), [10 + lenght\_key\*(1/1.5 + i + skip\_count), HEIGHT - height\_key, lenght\_key/1.4, height\_key\*2/3], 0, 2)

pygame.draw.rect(screen, 'black', [10 + lenght\_key\*(1/1.5 + i + skip\_count), HEIGHT - height\_key, lenght\_key/1.4, height\_key\*2/3], 2, 2)

gradientRect(screen, (245, 245, 255), (130, 130, 255), pygame.Rect(10 + lenght\_key\*(1/1.5 + i + skip\_count), HEIGHT - (height\_key\*(7/5) -1), lenght\_key/1.4, height\_key\*(2/5)))

active\_blacks[q][1] -= 1

if len(active\_blacks) > 10000:

active\_blacks.clear()

if q < len\_black:

q += 1

key\_label = key\_font.render(black\_flats\_label[i], True, 'white')

center = key\_label.get\_rect( center = (10 + lenght\_key\*(1/1.5 + i + skip\_count) + lenght\_key/(1.4\*2), HEIGHT - height\_key\*(1/2)))

screen.blit(key\_label, center)

skip\_track += 1

if last\_skip == 2 and skip\_track == 3:

last\_skip = 3

skip\_track = 0

skip\_count += 1

elif last\_skip == 3 and skip\_track == 2:

last\_skip = 2

skip\_track = 0

skip\_count += 1

return white\_rects, black\_rects, active\_whites, active\_blacks

def draw\_keyboard(active\_white, active\_black, screen, HEIGHT, WIDTH, view):

lenght\_button = WIDTH/30

height\_button = HEIGHT/13.4

height\_place = 7.7

if view == False:

i = 0

len\_white = len(active\_white)

while i < len\_white and len\_white > 0:

if active\_white[i][1] == 0:

active\_white.pop(i)

len\_white -= 1

elif active\_white[i][1] > 0:

q = active\_white[i][0]

if q == 0 or q == 6:

pygame.draw.rect(screen, '#CCCCFF', [WIDTH/2 + (white\_button\_list[q][1] - 7)\*lenght\_button, (height\_place - white\_button\_list[q][0])\*height\_button, lenght\_button\*(19/20) \* 2, height\_button\*(4/5)], 0, 2)

elif q == 13:

pygame.draw.rect(screen, '#CCCCFF', [WIDTH/2 + (white\_button\_list[q][1] - 7)\*lenght\_button, (height\_place - white\_button\_list[q][0])\*height\_button, lenght\_button\*(19/20) \* 1.5, height\_button\*(4/5)], 0, 2)

else:

pygame.draw.rect(screen, '#CCCCFF', [WIDTH/2 + (white\_button\_list[q][1] - 7)\*lenght\_button, (height\_place - white\_button\_list[q][0])\*height\_button, lenght\_button\*(19/20) \* 1, height\_button\*(4/5)], 0, 2)

active\_white[i][1] -= 1

elif len(active\_white) > 10000:

active\_white.clear()

if i < len\_white:

i += 1

i = 0

len\_black = len(active\_black)

while i < len\_black and len\_black > 0:

if active\_black[i][1] == 0:

active\_black.pop(i)

len\_black -= 1

elif active\_black[i][1] > 0:

q = active\_black[i][0]

pygame.draw.rect(screen, '#CCCCFF', [WIDTH/2 + (black\_button\_list[q][1] - 7)\*lenght\_button, (height\_place - black\_button\_list[q][0])\*height\_button, lenght\_button\*(19/20) \* 1, height\_button\*(4/5)], 0, 2)

active\_black[i][1] -= 1

elif len(active\_black) > 10000:

active\_black.clear()

if i < len\_black:

i += 1

key\_count = 0

notes\_count = 0

for i in range(5):

j = 0

while j <= 13:

if j == 3 and i == 0:

pygame.draw.rect(screen, 'black', [WIDTH/2 + (j - 7)\*lenght\_button, (height\_place - i)\*height\_button, lenght\_button\*(19/20) \* 7.3, height\_button\*(4/5)], 1, 2)

key\_label = button\_font.render(button\_label[key\_count], True, 'gray')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/40) \* 7.3)\*lenght\_button , (height\_place + 0.4 - i)\*height\_button))

screen.blit(key\_label, center)

j += 7

key\_count += 1

if (j == 0 or j == 12) and i == 1:

pygame.draw.rect(screen, 'black', [WIDTH/2 + (j - 7)\*lenght\_button, (height\_place - i)\*height\_button, lenght\_button\*(19/20) \* 2, height\_button\*(4/5)], 1, 2)

key\_label = bold\_button\_font.render(button\_label[key\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/20))\*lenght\_button , (height\_place + 0.2 - i)\*height\_button))

screen.blit(key\_label, center)

key\_label = button\_font.render(piano\_notes\_label[notes\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/20))\*lenght\_button , (height\_place + 0.6 - i)\*height\_button))

screen.blit(key\_label, center)

key\_count += 1

notes\_count += 1

j += 2

if (j == 0 or j == 12.5) and i ==2:

pygame.draw.rect(screen, 'black', [WIDTH/2 + (j - 7)\*lenght\_button, (height\_place - i)\*height\_button, lenght\_button\*(19/20) \* 1.5, height\_button\*(4/5)], 1, 2)

if key\_count == 32:

key\_label = bold\_button\_font.render(button\_label[key\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/40) \* 1.5)\*lenght\_button , (height\_place + 0.2 - i)\*height\_button))

screen.blit(key\_label, center)

key\_label = button\_font.render(piano\_notes\_label[notes\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/40) \* 1.5)\*lenght\_button , (height\_place + 0.6 - i)\*height\_button))

screen.blit(key\_label, center)

notes\_count += 1

else:

key\_label = button\_font.render(button\_label[key\_count], True, 'gray')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 7 + (19/40) \* 1.5)\*lenght\_button , (height\_place + 0.4 - i)\*height\_button))

screen.blit(key\_label, center)

j += 1.5

key\_count += 1

if j <= 13:

pygame.draw.rect(screen, 'black', [WIDTH/2 + (j - 7)\*lenght\_button, (height\_place - i)\*height\_button, lenght\_button\*(19/20), height\_button\*(4/5)], 1, 2)

if key\_count not in not\_note\_buttom:

key\_label = bold\_button\_font.render(button\_label[key\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 6.5)\*lenght\_button , (height\_place + 0.2 - i)\*height\_button))

screen.blit(key\_label, center)

key\_label = button\_font.render(piano\_notes\_label[notes\_count], True, 'black')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 6.5)\*lenght\_button , (height\_place + 0.6 - i)\*height\_button))

screen.blit(key\_label, center)

notes\_count += 1

else:

key\_label = button\_font.render(button\_label[key\_count], True, 'gray')

center = key\_label.get\_rect(center = (WIDTH/2 + (j - 6.5)\*lenght\_button , (height\_place + 0.4 - i)\*height\_button))

screen.blit(key\_label, center)

j += 1

key\_count += 1

def record\_timer(screen, HEIGHT, WIDTH, if\_record, curr\_sec, sec, mins):

if if\_record == True:

pygame.draw.rect(screen, 'red', [WIDTH\*(9/20) , HEIGHT\*(7/40), WIDTH/10, HEIGHT/20], 1, 2)

record\_time = timer\_font.render('{}:{}'.format(mins, sec), True, 'red')

record\_rect = record\_time.get\_rect(center = (WIDTH\*(1/2), HEIGHT\*(4/20)))

if (curr\_sec - time.time()//1) != 0:

curr\_sec = time.time()//1

sec += 1

screen.blit(record\_time, record\_rect)

if sec > 60:

sec = 0

mins += 1

return curr\_sec, sec, mins

def view\_sample(screen, WIDTH, HEIGHT, view, key\_learn):

if view == True:

if key\_learn == None:

pygame.draw.rect(screen, 'red', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 1, 2)

warn\_text = bold\_button\_font.render('Выберите композицию', True, 'red')

warn\_rect = warn\_text.get\_rect(center = (WIDTH\*(1/2), HEIGHT\*(4/20)))

screen.blit(warn\_text, warn\_rect)

else:

#pygame.draw.rect(screen, 'black', [WIDTH\*(3/20) , HEIGHT/15 + HEIGHT\*(1/40), WIDTH\*(7/10), HEIGHT\*(13/20)], 1, 2)

print(learn\_samples[key\_learn])

sample\_img = pygame.image.load(learn\_samples[key\_learn])

sample\_img = pygame.transform.scale(sample\_img, (WIDTH\*(9/40), HEIGHT\*(25/40)))

# sample\_text = bold\_button\_font.render(learn\_samples[key\_learn], True, 'black')

# sample\_rect = sample\_text.get\_rect(center = (WIDTH\*(3/20) + WIDTH\*(7/20), HEIGHT/15 + HEIGHT\*(1/40) + HEIGHT\*(3/40)))

screen.blit(sample\_img, (WIDTH\*(8/20), HEIGHT/15 + HEIGHT\*(1/40) + 5, ))

def view\_lock(screen, WIDTH, HEIGHT, if\_lock):

if if\_lock == True:

screen.blit(lock\_image, (WIDTH/150 + WIDTH\*(40/50), HEIGHT\*(1/120)))

pygame.draw.rect(screen, 'red', [WIDTH\*(8/20) , HEIGHT\*(7/80), WIDTH\*(2/10), HEIGHT/20], 1, 2)

warn\_text = timer\_font.render('Введите лицензионный ключ', True, 'red')

warn\_rect = warn\_text.get\_rect(center = (WIDTH\*(1/2), HEIGHT\*(7/80) + HEIGHT/40))

screen.blit(warn\_text, warn\_rect)

def input\_key(screen, WIDTH, HEIGHT, if\_input, input\_key):

if if\_input == True:

pygame.draw.rect(screen, '#CCCCFF', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 0, 2)

pygame.draw.rect(screen, '#9A9AFF', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 2, 2)

input\_text = bold\_button\_font.render(input\_key, True, 'black')

input\_rect = input\_text.get\_rect(center = (WIDTH\*(8/20) + WIDTH\*(1/10), HEIGHT\*(7/40) + HEIGHT/40))

screen.blit(input\_text, input\_rect)

def check\_key(screen, WIDTH, HEIGHT, if\_check\_key, key):

if if\_check\_key == True:

if key in license\_keys:

pygame.draw.rect(screen, '#CDFFD0', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 0, 2)

pygame.draw.rect(screen, '#6DCA73', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 2, 2)

input\_text = bold\_button\_font.render('Ключ успешно введен', True, 'black')

input\_rect = input\_text.get\_rect(center = (WIDTH\*(8/20) + WIDTH\*(1/10), HEIGHT\*(7/40) + HEIGHT/40))

screen.blit(input\_text, input\_rect)

screen.blit(close\_image, (WIDTH\*(8/20) + WIDTH\*(2/10) - 23, HEIGHT\*(7/40) + 2))

return True

else:

pygame.draw.rect(screen, '#FFBCBC', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 0, 2)

pygame.draw.rect(screen, '#D06868', [WIDTH\*(8/20) , HEIGHT\*(7/40), WIDTH\*(2/10), HEIGHT/20], 2, 2)

input\_text = bold\_button\_font.render('Ключ неверен', True, 'black')

input\_rect = input\_text.get\_rect(center = (WIDTH\*(8/20) + WIDTH\*(1/10), HEIGHT\*(7/40) + HEIGHT/40))

screen.blit(input\_text, input\_rect)

screen.blit(close\_image, (WIDTH\*(8/20) + WIDTH\*(2/10) - 23, HEIGHT\*(7/40) + 2))

return False

Полный код файла functions.py

import pygame

import pygame\_widgets

from pygame\_widgets.button import Button

from pygame\_widgets.dropdown import Dropdown

def create\_buttons(screen, WIDTH, HEIGHT):

start\_record\_image = pygame.image.load('assets/images/start\_record.png')

start\_record\_image = pygame.transform.scale(start\_record\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_record = Button(

screen, WIDTH/150 + WIDTH\*(1/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = start\_record\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

stop\_record\_image = pygame.image.load('assets/images/stop\_record.png')

stop\_record\_image = pygame.transform.scale(stop\_record\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_stop\_record = Button(

screen, WIDTH/150 + WIDTH\*(3/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = stop\_record\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

start\_play\_image = pygame.image.load('assets/images/play\_music-2.png')

start\_play\_image = pygame.transform.scale(start\_play\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_play\_music = Button(

screen, WIDTH/150 + WIDTH\*(5/50) , HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = start\_play\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

stop\_play\_image = pygame.image.load('assets/images/stop\_music.png')

stop\_play\_image = pygame.transform.scale(stop\_play\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_stop\_music = Button(

screen, WIDTH/150 + WIDTH\*(7/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = stop\_play\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

view\_sample\_image = pygame.image.load('assets/images/note\_list.png')

view\_sample\_image = pygame.transform.scale(view\_sample\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_view\_sample = Button(

screen, WIDTH/150 + WIDTH\*(38/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = view\_sample\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

stop\_view\_sample\_image = pygame.image.load('assets/images/close\_note\_list.png')

stop\_view\_sample\_image = pygame.transform.scale(stop\_view\_sample\_image, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_stop\_view\_sample = Button(

screen, WIDTH/150 + WIDTH\*(36/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = stop\_view\_sample\_image,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

input\_key = pygame.image.load('assets/images/key.png')

input\_key = pygame.transform.scale(input\_key, ( WIDTH\*(1/45), HEIGHT\*(1/25)))

btn\_input\_key = Button(

screen, WIDTH/150 + WIDTH\*(34/50), HEIGHT\*(1/120), WIDTH\*(1/38), HEIGHT\*(1/20),

image = input\_key,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

radius=3

)

drop\_learn\_samples = Dropdown(

screen, WIDTH/150 + WIDTH\*(40/50), HEIGHT\*(1/120), WIDTH\*(5/31), HEIGHT\*(1/20), name='Выбрать композицию',

choices=[

'Выбрать композицию',

'ABBA - Happy new year',

'Конь Любе- Hard',

'Конь Любе- Esay',

],

font = pygame.font.Font('assets/timesnrcyrmt\_bold.ttf', 24),

borderRadius=3,

inactiveColour=(240, 240, 255),

hoverColour=(225, 225, 255),

pressedColour=(160, 160, 255),

values=[None, 'ABBA', 'Konb-hard', 'Konb'], direction='down'

)

return (btn\_record, btn\_stop\_record, btn\_play\_music,

btn\_stop\_music, btn\_view\_sample, btn\_stop\_view\_sample,

btn\_input\_key, drop\_learn\_samples)

Полный код файла buttons.py

import pygame

piano\_notes = ['C2', 'D-2', 'D2', 'E-2', 'E2', 'F2', 'G-2', 'G2', 'A-2', 'A2', 'B-2', 'B2',

'C3', 'D-3', 'D3', 'E-3', 'E3', 'F3', 'G-3', 'G3', 'A-3', 'A3', 'B-3', 'B3',

'C4', 'D-4', 'D4', 'E-4', 'E4', 'F4', 'G-4', 'G4', 'A-4', 'A4', 'B-4', 'B4',

'C5', 'D-5', 'D5', 'E-5', 'E5', 'F5', 'G-5', 'G5', 'A-5', 'A5', 'B-5', 'B5', 'C6']

white\_notes = ['C2', 'D2', 'E2', 'F2', 'G2', 'A2', 'B2',

'C3', 'D3', 'E3', 'F3', 'G3', 'A3', 'B3',

'C4', 'D4', 'E4', 'F4', 'G4', 'A4', 'B4',

'C5', 'D5', 'E5', 'F5', 'G5', 'A5', 'B5',

'C6']

black\_flats = ['D-2', 'E-2', 'G-2', 'A-2', 'B-2',

'D-3', 'E-3', 'G-3', 'A-3', 'B-3',

'D-4', 'E-4', 'G-4', 'A-4', 'B-4',

'D-5', 'E-5', 'G-5', 'A-5', 'B-5']

key\_list = [pygame.K\_PERIOD, pygame.K\_SEMICOLON, pygame.K\_p, pygame.K\_0,

pygame.K\_COMMA, pygame.K\_l, pygame.K\_o, pygame.K\_9,

pygame.K\_RSHIFT, pygame.K\_LSHIFT, pygame.K\_RIGHTBRACKET, pygame.K\_EQUALS,

pygame.K\_SLASH, pygame.K\_QUOTE, pygame.K\_LEFTBRACKET, pygame.K\_MINUS,

pygame.K\_RETURN, pygame.K\_a, pygame.K\_q, pygame.K\_1, pygame.K\_BACKSPACE,

pygame.K\_x, pygame.K\_d, pygame.K\_e, pygame.K\_3,

pygame.K\_z, pygame.K\_s, pygame.K\_w, pygame.K\_2,

pygame.K\_v, pygame.K\_g, pygame.K\_t, pygame.K\_5,

pygame.K\_c, pygame.K\_f, pygame.K\_r, pygame.K\_4,

pygame.K\_b, pygame.K\_h, pygame.K\_y, pygame.K\_6,

pygame.K\_m, pygame.K\_k, pygame.K\_i, pygame.K\_8,

pygame.K\_n, pygame.K\_j, pygame.K\_u, pygame.K\_7

]

white\_button\_list = {0: [1, 0], 1: [1, 3], 2: [1, 5], 3: [1, 6], 4: [1, 8], 5: [1, 10], 6: [1, 12],

7: [2, 1.5], 8: [2, 3.5], 9: [2, 5.5], 10: [2, 6.5], 11: [2, 8.5], 12: [2, 10.5], 13: [2, 12.5],

14: [3, 1], 15: [3, 3], 16: [3, 5], 17: [3, 6], 18: [3, 8], 19: [3, 10], 20: [3, 12],

21: [4, 1], 22: [4, 3], 23: [4, 5], 24: [4, 6], 25: [4, 8], 26: [4, 10], 27: [4, 12], 28: [4, 13]}

black\_button\_list = {0: [1, 2], 1: [1, 4], 2: [1, 7], 3: [1, 9], 4: [1, 11],

5: [2, 2.5], 6: [2, 4.5], 7: [2, 7.5], 8: [2, 9.5], 9: [2, 11.5],

10: [3, 2], 11: [3, 4], 12: [3, 7], 13: [3, 9], 14: [3, 11],

15: [4, 2], 16: [4, 4], 17: [4, 7], 18: [4, 9], 19: [4, 11]}

not\_note\_button\_list = {0: 'Ctrl', 1: 'Win', 2: 'Alt', 3: 'Space', 4: 'Alt', 5: 'Win', 6: 'Meny', 7: 'Ctrl',

20: 'Caps', 33: 'Tab', 46: '\\', 47: '`/~'}

button\_list = ['Ctrl', 'Win', 'Alt', 'Space', 'Alt', 'Win', 'Meny', 'Ctrl',

'LShift', 'Z', 'X', 'C', 'V', 'B', 'N', 'M', ',', '.', '/', 'RShift',

'Caps', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ';','\'', 'Enter',

'Tab', 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '[', ']', '\\',

'`/~', '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '<-']

notes\_dict = {

f'{pygame.K\_PERIOD}': 'A2', f'{pygame.K\_SEMICOLON}': 'A3', f'{pygame.K\_p}': 'A4', f'{pygame.K\_0}': 'A5',

f'{pygame.K\_COMMA}': 'A-2', f'{pygame.K\_l}': 'A-3', f'{pygame.K\_o}': 'A-4', f'{pygame.K\_9}': 'A-5',

f'{pygame.K\_RSHIFT}': 'B2', f'{pygame.K\_RETURN}': 'B3', f'{pygame.K\_RIGHTBRACKET}': 'B4', f'{pygame.K\_EQUALS}': 'B5',

f'{pygame.K\_SLASH}': 'B-2', f'{pygame.K\_QUOTE}': 'B-3', f'{pygame.K\_LEFTBRACKET}': 'B-4', f'{pygame.K\_MINUS}': 'B-5',

f'{pygame.K\_LSHIFT}': 'C2', f'{pygame.K\_a}': 'C3', f'{pygame.K\_q}': 'C4', f'{pygame.K\_1}': 'C5', f'{pygame.K\_BACKSPACE}': 'C6',

f'{pygame.K\_x}': 'D2', f'{pygame.K\_d}': 'D3', f'{pygame.K\_e}': 'D4', f'{pygame.K\_3}': 'D5',

f'{pygame.K\_z}': 'D-2', f'{pygame.K\_s}': 'D-3', f'{pygame.K\_w}': 'D-4', f'{pygame.K\_2}': 'D-5',

f'{pygame.K\_v}': 'E2', f'{pygame.K\_g}': 'E3', f'{pygame.K\_t}': 'E4', f'{pygame.K\_5}': 'E5',

f'{pygame.K\_c}': 'E-2', f'{pygame.K\_f}': 'E-3', f'{pygame.K\_r}': 'E-4', f'{pygame.K\_4}': 'E-5',

f'{pygame.K\_b}': 'F2', f'{pygame.K\_h}': 'F3', f'{pygame.K\_y}': 'F4', f'{pygame.K\_6}': 'F5',

f'{pygame.K\_m}': 'G2', f'{pygame.K\_k}': 'G3', f'{pygame.K\_i}': 'G4', f'{pygame.K\_8}': 'G5',

f'{pygame.K\_n}': 'G-2', f'{pygame.K\_j}': 'G-3', f'{pygame.K\_u}': 'G-4', f'{pygame.K\_7}': 'G-5'}

midi\_notes = {'C2': 36, 'D-2': 37, 'D2': 38, 'E-2': 39, 'E2': 40, 'F2' :41, 'G-2': 42, 'G2': 43, 'A-2': 44, 'A2': 45, 'B-2': 46, 'B2': 47,

'C3': 48, 'D-3': 49, 'D3': 50, 'E-3': 51, 'E3': 52, 'F3': 53, 'G-3': 54, 'G3': 55, 'A-3': 56, 'A3': 57, 'B-3': 58, 'B3': 59,

'C4': 60, 'D-4': 61, 'D4': 62, 'E-4': 63, 'E4': 64, 'F4': 65, 'G-4': 66, 'G4': 67, 'A-4': 68, 'A4': 69, 'B-4': 70, 'B4': 71,

'C5': 72, 'D-5': 73, 'D5': 74, 'E-5':75, 'E5': 76, 'F5': 77, 'G-5':78, 'G5':79, 'A-5': 80, 'A5': 81, 'B-5': 82, 'B5':83, 'C6': 84}

learn\_samples = {

'ABBA' : 'assets\\samples\\ABBA.png',

'Konb' : 'assets\\samples\\Konb.png',

'Konb-hard' : 'assets\\samples\\Konb-hard.png'

}

licence\_keys = ['1234', 'qwerty']

Полный код файла piano\_list.py

import pygame

import pygame\_widgets

from pygame\_widgets.button import Button

from pygame\_widgets.dropdown import Dropdown

import mido

from mido import MidiFile, MidiTrack

import time

import piano\_list2 as pl2

import functions as f

import buttons

pygame.init()

pygame.mixer.set\_num\_channels(50)

pygame.mixer.init()

timer = pygame.time.Clock()

infoObject = pygame.display.Info()

WIDTH = infoObject.current\_w

HEIGHT = infoObject.current\_h - 60

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

pygame.display.set\_caption("Python Piano - CopyAssignment")

current\_size = screen.get\_size()

fps = 60

key\_list = pl2.key\_list

piano\_notes\_key = pl2.notes\_dict

white\_notes\_label = pl2.white\_notes

black\_flats\_label = pl2.black\_flats

midi\_notes = pl2.midi\_notes

white\_sounds, black\_sounds = f.get\_sounds(white\_notes\_label, black\_flats\_label)

active\_whites = []

active\_blacks = []

active\_button\_white = []

active\_button\_black = []

run = True

if\_record = False

if\_lock = True

if\_view = False

if\_input = False

key\_learn = None

input\_key = ''

input\_text = ''

if\_check\_key = False

if\_key\_right = False

btn\_close = pygame.draw.rect(screen, 'white', [0, 0, 0, 0], 1, 2)

sec =0

mins = 0

curr\_sec = 0

btn\_font = pygame.font.Font('assets/timesnrcyrmt.ttf', 16)

input\_font = pygame.font.Font('assets/timesnrcyrmt.ttf', 14)

btn\_record, btn\_stop\_record, btn\_play\_music, btn\_stop\_music, btn\_view\_sample, btn\_stop\_view\_sample, btn\_input\_key, drop\_learn\_samples = buttons.create\_buttons(screen, WIDTH, HEIGHT)

mid = MidiFile()

track = MidiTrack()

while run:

timer.tick(fps)

screen.fill('#F5F5FF')

white\_keys, black\_keys, active\_whites, active\_blacks = f.draw\_piano(active\_whites, active\_blacks, screen, HEIGHT, WIDTH, track, sec, mins)

f.draw\_keyboard(active\_button\_white, active\_button\_black, screen, HEIGHT, WIDTH, if\_view)

pygame.draw.rect(screen, '#CCCCFF', [0, 0, WIDTH, HEIGHT/15], 0, 2)

curr\_sec, sec, mins = f.record\_timer(screen, HEIGHT, WIDTH, if\_record, curr\_sec, sec, mins)

f.view\_sample(screen, WIDTH, HEIGHT, if\_view, key\_learn)

mouse = pygame.mouse.get\_pos()

if (btn\_record.getX() <= mouse[0] and (btn\_record.getWidth() + btn\_record.getX()) >= mouse[0] and

btn\_record.getY() <= mouse[1] and (btn\_record.getHeight() + btn\_record.getY()) >= mouse[1]):

pygame.draw.rect(screen, 'white', [btn\_record.getWidth() + btn\_record.getX(),

btn\_record.getHeight() + btn\_record.getY() + 10,

WIDTH\*(3/31), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Начать запись', True, 'black')

center = key\_label.get\_rect(center = (btn\_record.getWidth() + btn\_record.getX() + WIDTH\*(3/62),

btn\_record.getHeight() + btn\_record.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_stop\_record.getX() <= mouse[0] and (btn\_stop\_record.getWidth() + btn\_stop\_record.getX()) >= mouse[0] and

btn\_stop\_record.getY() <= mouse[1] and (btn\_stop\_record.getHeight() + btn\_stop\_record.getY()) >= mouse[1]):

pygame.draw.rect(screen, 'white', [btn\_stop\_record.getWidth() + btn\_stop\_record.getX(),

btn\_stop\_record.getHeight() + btn\_stop\_record.getY() + 10,

WIDTH\*(3/31), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Остановить запись', True, 'black')

center = key\_label.get\_rect(center = (btn\_stop\_record.getWidth() + btn\_stop\_record.getX() + WIDTH\*(3/62),

btn\_stop\_record.getHeight() + btn\_stop\_record.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_play\_music.getX() <= mouse[0] and (btn\_play\_music.getWidth() + btn\_play\_music.getX()) >= mouse[0] and

btn\_play\_music.getY() <= mouse[1] and (btn\_play\_music.getHeight() + btn\_play\_music.getY()) >= mouse[1]):

pygame.draw.rect(screen, 'white', [btn\_play\_music.getWidth() + btn\_play\_music.getX(),

btn\_play\_music.getHeight() + btn\_play\_music.getY() + 10,

WIDTH\*(4/31), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Воспроизвести музыку', True, 'black')

center = key\_label.get\_rect(center = (btn\_play\_music.getWidth() + btn\_play\_music.getX() + WIDTH\*(4/62),

btn\_play\_music.getHeight() + btn\_play\_music.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_stop\_music.getX() <= mouse[0] and (btn\_stop\_music.getWidth() + btn\_stop\_music.getX()) >= mouse[0] and

btn\_stop\_music.getY() <= mouse[1] and (btn\_stop\_music.getHeight() + btn\_stop\_music.getY()) >= mouse[1]):

pygame.draw.rect(screen, 'white', [btn\_stop\_music.getWidth() + btn\_stop\_music.getX(),

btn\_stop\_music.getHeight() + btn\_stop\_music.getY() + 10,

WIDTH\*(4/30), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Остановить воспроизведение', True, 'black')

center = key\_label.get\_rect(center = (btn\_stop\_music.getWidth() + btn\_stop\_music.getX() + WIDTH\*(4/60),

btn\_stop\_music.getHeight() + btn\_stop\_music.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_view\_sample.getX() <= mouse[0] and (btn\_view\_sample.getWidth() + btn\_view\_sample.getX()) >= mouse[0] and

btn\_view\_sample.getY() <= mouse[1] and (btn\_view\_sample.getHeight() + btn\_view\_sample.getY()) >= mouse[1] and

btn\_view\_sample.\_disabled == False):

pygame.draw.rect(screen, 'white', [btn\_view\_sample.getWidth() + btn\_view\_sample.getX(),

btn\_view\_sample.getHeight() + btn\_view\_sample.getY() + 10,

WIDTH\*(3/30), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Показать ноты', True, 'black')

center = key\_label.get\_rect(center = (btn\_view\_sample.getWidth() + btn\_view\_sample.getX() + WIDTH\*(3/60),

btn\_view\_sample.getHeight() + btn\_view\_sample.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_stop\_view\_sample.getX() <= mouse[0] and (btn\_stop\_view\_sample.getWidth() + btn\_stop\_view\_sample.getX()) >= mouse[0] and

btn\_stop\_view\_sample.getY() <= mouse[1] and (btn\_stop\_view\_sample.getHeight() + btn\_stop\_view\_sample.getY()) >= mouse[1] and

btn\_stop\_view\_sample.\_disabled == False):

pygame.draw.rect(screen, 'white', [btn\_stop\_view\_sample.getWidth() + btn\_stop\_view\_sample.getX(),

btn\_stop\_view\_sample.getHeight() + btn\_stop\_view\_sample.getY() + 10,

WIDTH\*(3/30), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Скрыть ноты', True, 'black')

center = key\_label.get\_rect(center = (btn\_stop\_view\_sample.getWidth() + btn\_stop\_view\_sample.getX() + WIDTH\*(3/60),

btn\_stop\_view\_sample.getHeight() + btn\_stop\_view\_sample.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

if (btn\_input\_key.getX() <= mouse[0] and (btn\_input\_key.getWidth() + btn\_input\_key.getX()) >= mouse[0] and

btn\_input\_key.getY() <= mouse[1] and (btn\_input\_key.getHeight() + btn\_input\_key.getY()) >= mouse[1] and

btn\_input\_key.\_disabled == False):

pygame.draw.rect(screen, 'white', [btn\_input\_key.getWidth() + btn\_input\_key.getX(),

btn\_input\_key.getHeight() + btn\_input\_key.getY() + 10,

WIDTH\*(5/30), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Ввести лицензионный ключ', True, 'black')

center = key\_label.get\_rect(center = (btn\_input\_key.getWidth() + btn\_input\_key.getX() + WIDTH\*(5/60),

btn\_input\_key.getHeight() + btn\_input\_key.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

elif (btn\_input\_key.getX() <= mouse[0] and (btn\_input\_key.getWidth() + btn\_input\_key.getX()) >= mouse[0] and

btn\_input\_key.getY() <= mouse[1] and (btn\_input\_key.getHeight() + btn\_input\_key.getY()) >= mouse[1] and btn\_input\_key.\_disabled == True):

pygame.draw.rect(screen, 'white', [btn\_input\_key.getWidth() + btn\_input\_key.getX(),

btn\_input\_key.getHeight() + btn\_input\_key.getY() + 10,

WIDTH\*(5/30), HEIGHT/25], 0, 2)

key\_label = btn\_font.render('Ключ уже введен', True, 'black')

center = key\_label.get\_rect(center = (btn\_input\_key.getWidth() + btn\_input\_key.getX() + WIDTH\*(5/60),

btn\_input\_key.getHeight() + btn\_input\_key.getY() + HEIGHT/50 + 8))

screen.blit(key\_label, center)

events = pygame.event.get()

for event in events:

if event.type == pygame.QUIT:

run = False

if event.type == pygame.VIDEORESIZE:

screen = pygame.display.set\_mode((event.w, event.h), pygame.RESIZABLE)

WIDTH = event.w

HEIGHT = event.h

lenght\_key = (WIDTH - 20)/29

height\_key = HEIGHT/4

lenght\_button = WIDTH/20

height\_button = HEIGHT/13.4

if event.type == pygame.MOUSEBUTTONDOWN:

black\_key = False

for i in range(len(black\_keys)):

if black\_keys[i].collidepoint(event.pos):

black\_sounds[i].play(0, 1500)

black\_key = True

active\_button\_black.append([i, 30])

if if\_record == True:

active\_blacks.append([i, 30, midi\_notes[black\_flats\_label[i]], 1])

track.append(mido.Message('note\_on', note=midi\_notes[black\_flats\_label[i]], velocity=100, time=round((sec + time.time() % 1 + mins \* 60 ) \* 65)))

else:

active\_blacks.append([i, 30, midi\_notes[black\_flats\_label[i]], 0])

for i in range(len(white\_keys)):

if white\_keys[i].collidepoint(event.pos) and not black\_key:

white\_sounds[i].play(0, 1500)

active\_button\_white.append([i, 30])

if if\_record == True:

active\_whites.append([i, 30, midi\_notes[white\_notes\_label[i]], 1])

track.append(mido.Message('note\_on', note=midi\_notes[white\_notes\_label[i]], velocity=100, time=round((sec + time.time() % 1 + mins \* 60) \* 65)))

else:

active\_whites.append([i, 30, midi\_notes[white\_notes\_label[i]], 0])

if btn\_close.collidepoint(event.pos):

btn\_close = pygame.draw.rect(screen, 'white', [0, 0, 0, 0], 1, 2)

if\_check\_key = False

if btn\_record.clicked == True:

print(btn\_record.clicked)

if\_record = True

curr\_sec = time.time()//1

sec = 0

mins = 0

if btn\_stop\_record.clicked == True:

if\_record = False

mid.tracks.append(track)

mid.save('output.mid')

track.clear()

if btn\_play\_music.clicked == True:

if if\_record == True:

if\_record = False

mid.tracks.append(track)

mid.save('output.mid')

pygame.mixer.music.load("output.mid")

pygame.mixer.music.play()

if btn\_stop\_music.clicked == True:

pygame.mixer.music.stop()

if btn\_input\_key.clicked == True:

if if\_key\_right == False:

if\_input = True

if if\_key\_right == True:

btn\_input\_key.disable()

if if\_lock == False:

btn\_view\_sample.enable()

btn\_stop\_view\_sample.enable()

drop\_learn\_samples.enable()

if btn\_view\_sample.clicked == True:

if\_view = True

if btn\_stop\_view\_sample.clicked == True:

if\_view = False

key\_learn = drop\_learn\_samples.getSelected()

else:

btn\_view\_sample.disable()

btn\_stop\_view\_sample.disable()

drop\_learn\_samples.disable()

if event.type == pygame.KEYDOWN:

if if\_input == False:

if event.key in key\_list:

if piano\_notes\_key[str(event.key)] in black\_flats\_label:

index = black\_flats\_label.index(piano\_notes\_key[str(event.key)])

black\_sounds[index].play(0, 1000)

active\_button\_black.append([index, 30])

if if\_record == True:

active\_blacks.append([index, 30, midi\_notes[piano\_notes\_key[str(event.key)]], 1])

track.append(mido.Message('note\_on', note=midi\_notes[piano\_notes\_key[str(event.key)]], velocity=100, time=round((sec + time.time() % 1 + mins \* 60 ) \* 65)))

else:

active\_blacks.append([index, 30, midi\_notes[piano\_notes\_key[str(event.key)]], 0])

if piano\_notes\_key[str(event.key)] in white\_notes\_label:

index = white\_notes\_label.index(piano\_notes\_key[str(event.key)])

white\_sounds[index].play(0, 1000)

active\_button\_white.append([index, 30])

if if\_record == True:

active\_whites.append([index, 30, midi\_notes[piano\_notes\_key[str(event.key)]], 1])

track.append(mido.Message('note\_on', note=midi\_notes[piano\_notes\_key[str(event.key)]], velocity=100, time=round((sec + time.time() % 1 + mins \* 60) \* 65)))

else:

active\_whites.append([index, 30, midi\_notes[piano\_notes\_key[str(event.key)]], 0])

if if\_input == True:

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

input\_text = input\_text[:-1]

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

input\_key = input\_text

input\_text = ''

if\_input = False

if\_check\_key = True

else:

input\_text += event.unicode

pygame\_widgets.update(events)

f.view\_lock(screen, WIDTH, HEIGHT, if\_lock)

f.input\_key(screen, WIDTH, HEIGHT, if\_input, input\_text)

if if\_check\_key == True:

if\_key\_right = f.check\_key(screen, WIDTH, HEIGHT, if\_check\_key, input\_key)

if if\_check\_key == True and if\_key\_right == True:

if\_lock = False

if if\_check\_key == True:

btn\_close = pygame.draw.rect(screen, 'black', [WIDTH\*(8/20) + WIDTH\*(2/10) - 23, HEIGHT\*(7/40) + 2, 20, 20], 1, 2)

pygame.display.flip()

#this will quite the window of the pygame

pygame.quit()

Полный код файла piano.py