# 代码深度研究报告



生成时间：2025-08-19 15:01:29｜本报告由 GitHave AI 生成，仅用于研究目的，不构成任何形式的法律建议或保证，不承担因使用本报告而导致的任何损失或损害。

# 一、前言

* **仓库名称**：chordPrediction
* **仓库描述**：和弦预测
* **仓库分支**：master
* **仓库地址**：<https://github.com/kinglegendzzh/chordPrediction>
* **分析路径**：[/Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py) (单项文件: VirtualKeyboard.py)
* **项目总结**：

ChordPrediction 是一个基于马尔科夫链的智能音乐创作工具，通过分析历史和当前的和弦序列，实时预测和生成高质量的和弦序列。它支持多种和弦序列训练，提供交互界面进行标注、保存和预览，并采用模块化设计便于扩展和定制。

* **当前分析的VirtualKeyboard.py 文件总结**：

`VirtualKeyboard.py` 是一个用于创建智能化音乐创作工具用户界面的Python模块。它支持MIDI输入设备或键盘映射，提供和弦识别、预测模型选择、音乐风格选择、序列输入和保存等功能。模块核心功能包括初始化pygame和pygame.midi，设置MIDI输入设备或启用键盘映射模式，跟踪按下的键，初始化用户界面，处理键盘按键事件，更新和弦信息，以及保存和清除序列。该模块使用Qt的信号与槽机制处理用户界面事件，并通过定时器定时更新界面信息，是整个音乐创作工具的用户界面层。

# 二、项目描述

# ChordPrediction - 智能音乐创作工具

## 项目背景与目标

ChordPrediction 是一个基于马尔科夫链的和弦预测算法的智能音乐创作工具。它旨在帮助用户生成高质量的和弦序列，提高音乐创作的效率和多样性。该工具适用于音乐创作、音乐教育、音乐游戏等多个领域，特别是对于希望快速生成和弦序列的音乐爱好者和专业人士。

## 核心功能与特色亮点

### 主要功能

1. **和弦预测**：通过分析历史和当前的和弦序列，实时预测和生成和弦。
2. **多种和弦序列训练**：支持多种和弦序列训练，以增强不同音乐风格之间的过渡多样性。
3. **交互界面**：提供一个交互界面，允许用户通过简单的演奏生成所需的和弦，并进行标注、保存、预览和模型训练。

### 关键模块

* **sounds**：管理音频相关的资源和功能，包括音频文件的加载、播放和管理。
* **utils**：提供一系列通用的辅助工具类和函数，简化代码的编写和维护。
* **labels**：定义和管理不同音乐风格和模式的标签和模型。
* **models**：存储与处理压弦键映射相关的数据缓存。
* **records**：存储和管理各种音乐相关的数据和模式。
* **service**：处理音乐相关的服务和功能，包括 MIDI 输入管理、和弦序列预测、音阶匹配等。
* **VirtualKeyboard.py**：创建一个智能化音乐创作工具的用户界面，支持 MIDI 输入设备或键盘映射。
* **MatrixView.py**：创建一个图形用户界面，用于显示概率转移矩阵的和弦分布图。

## 项目价值与作用

ChordPrediction 通过智能化的和弦预测算法，为音乐创作提供了强大的支持。它不仅能够帮助用户快速生成高质量的和弦序列，还能够通过多种和弦序列训练，增强不同音乐风格之间的过渡多样性。此外，交互界面的设计使得用户可以方便地进行和弦标注、保存和预览，进一步提高了音乐创作的效率和质量。

## 项目架构设计

ChordPrediction 采用模块化设计，各个模块协同工作，共同实现音乐创作工具的功能。核心模块包括：

* **sounds**：负责音频资源的加载、播放和管理。
* **utils**：提供通用的辅助工具类和函数，提高代码的复用性和可维护性。
* **labels**：定义和管理音乐风格和模式的标签和模型。
* **models**：存储和处理压弦键映射相关的数据缓存。
* **records**：存储和管理各种音乐相关的数据和模式。
* **service**：处理音乐相关的服务和功能，包括 MIDI 输入管理、和弦序列预测、音阶匹配等。
* **VirtualKeyboard.py**：创建用户界面，支持 MIDI 输入设备或键盘映射。
* **MatrixView.py**：创建图形用户界面，用于显示概率转移矩阵的和弦分布图。

## 技术栈与依赖

ChordPrediction 项目使用了以下主要技术栈和依赖：

* **Python**：作为主要编程语言。
* **Pygame**：用于创建用户界面和处理 MIDI 输入设备。
* **PyInstaller**：用于打包 Python 应用程序，生成可执行文件。
* **MIDI 输入设备**：支持 MIDI 输入，用于实时生成和弦序列。

## 使用方式与扩展性

ChordPrediction 提供了一个交互界面，用户可以通过简单的演奏生成所需的和弦，并进行标注、保存、预览和模型训练。此外，项目采用模块化设计，便于扩展和定制。用户可以根据需要添加新的功能模块或修改现有模块，以满足特定的业务需求。

## 安全性与性能

ChordPrediction 项目在设计时注重安全性与性能。通过模块化设计和优化数据访问机制，确保了应用程序的高效运行。同时，项目遵循 GNU Lesser General Public License（LGPL）许可证，保护用户的自由使用和修改软件库的权利。

## 总结

ChordPrediction 是一个基于马尔科夫链的和弦预测算法的智能音乐创作工具。它通过智能化的和弦预测算法，为音乐创作提供了强大的支持。项目采用模块化设计，各个模块协同工作，共同实现音乐创作工具的功能。通过使用 Pygame 和 PyInstaller 等工具，项目能够生成可执行文件，便于在不同环境中部署和运行。

# 三、分析摘要

## 代码结构概览

VirtualKeyboard.py 是 chordPrediction 仓库中的一个重要文件，主要用于创建智能化音乐创作工具的用户界面。该文件依赖于多个外部模块，包括 PyQt5、musicpy 和 service 等。以下是文件的主要组成部分及其关系：

* **类和函数**：VirtualKeyboard 类包含多个函数，如 \_\_init\_\_、initUI、pressingEvent 等，用于初始化界面、处理按键事件、更新和弦信息等。
* **依赖关系**：该文件依赖于多个外部模块，如 PyQt5.QtWidgets、musicpy.musicpy 等，这些模块提供了图形用户界面、音乐处理等功能。

## 核心模块和函数

### 1. \_\_init\_\_

* **作用**：初始化 VirtualKeyboard 类，设置窗口大小和标题，初始化按键网格，显示和弦识别信息等。
* **重要性**：高，作为类的构造函数，初始化了界面和所有必要的组件。
* **复杂度**：31，中等复杂度，主要负责初始化和设置界面组件。

### 2. initUI

* **作用**：初始化用户界面，包括设置窗口大小和标题、创建按键网格、显示和弦识别信息等。
* **重要性**：高，作为界面初始化的核心函数，负责创建和布局所有界面组件。
* **复杂度**：189，高复杂度，包含多个子功能和复杂的布局逻辑。

### 3. pressingEvent

* **作用**：处理键盘按键事件，检测按下的琴键并识别和弦，更新显示和弦信息，并将和弦信息缓存到本地。
* **重要性**：中，处理用户输入的核心函数，负责和弦识别和界面更新。
* **复杂度**：50，中等复杂度，包含多个条件判断和逻辑处理。

### 4. updateChords

* **作用**：更新和弦序列并进行和弦预测。它会检查所有选中的复选框，读取对应的模型文件，构建马尔科夫链，并根据当前的和弦序列进行预测。预测结果会显示在界面上。
* **重要性**：中，负责和弦预测的核心函数，包含复杂的逻辑和模型处理。
* **复杂度**：57，高复杂度，包含多个条件判断和模型预测逻辑。

### 5. shaderLists

* **作用**：在虚拟键盘界面中显示标注库和历史记录，并通过列表和堆叠布局进行切换。
* **重要性**：中，负责界面布局和数据展示的核心函数。
* **复杂度**：70，中等复杂度，包含多个循环和条件判断。

## 代码设计风格分析

### 命名规范

* **变量命名**：变量命名遵循驼峰命名法，如 self.keys\_button、self.key\_pressed 等。
* **函数命名**：函数命名清晰，如 initUI、pressingEvent 等。

### 一致性

* **命名一致性**：命名风格和命名规则在代码中保持一致，如所有变量和函数都使用驼峰命名法。
* **代码风格一致性**：代码风格在文件中保持一致，如缩进、空格等。

### 封装与抽象程度

* **封装**：类和函数内部逻辑封装良好，如 \_\_init\_\_ 和 initUI 函数内部包含多个子功能，但对外部调用者隐藏了具体实现细节。
* **抽象**：函数设计合理，如 pressingEvent 函数通过调用 detectElement 和 getNormalChord 等函数实现了和弦识别的抽象。

## 潜在问题

### 资源释放不当

* **问题**：未看到显式的资源释放代码，如文件关闭、内存释放等。
* **建议**：在函数结束时显式关闭文件和释放资源，确保资源被正确释放。

### 异常未处理

* **问题**：未看到异常处理代码，如 try-except 块。
* **建议**：在关键操作（如文件读写、网络请求等）中添加异常处理，确保程序在异常情况下能够正常退出。

### 重复或冗余的逻辑

* **问题**：在多个函数中存在重复的逻辑，如 changeWhiteSheet 和 changeBlackSheet 函数中对背景颜色的设置。
* **建议**：将重复的逻辑提取到单独的函数中，减少代码冗余。

### 低效的实现

* **问题**：在 pressingEvent 函数中，多次调用 self.QUEUE.last()，可能导致性能问题。
* **建议**：使用缓存机制，避免多次调用，提高性能。

## 重构建议

### 函数职责过于复杂

* **建议**：将 initUI 函数拆分为多个子函数，如 initUI、createUIElements、setupUIConnections 等，每个子函数负责一个具体的功能，提高代码可读性和可维护性。

### 模块边界是否清晰

* **建议**：将 VirtualKeyboard 类中的功能模块化，如将和弦预测逻辑提取到 ChordPredictor 类中，提高模块间的解耦和可复用性。

### 代码是否存在重复可以抽取

* **建议**：将 changeWhiteSheet 和 changeBlackSheet 函数中的公共逻辑提取到一个单独的函数中，减少代码冗余。

### 公用模块是否可以拆分

* **建议**：将一些通用的功能提取到单独的模块中，如 utils 模块，提高代码的复用性和可维护性。

## 测试情况

* **测试覆盖范围**：未看到具体的测试代码，无法评估测试的覆盖范围。
* **健壮性**：未看到具体的测试代码，无法评估测试的健壮性。
* **验证主要功能**：未看到具体的测试代码，无法评估测试是否验证了主要功能。
* **边界情况和异常情况**：未看到具体的测试代码，无法评估测试是否考虑了边界情况和异常情况。

## 补充说明

* **重要性得分**：VirtualKeyboard.py 的重要性得分较高，说明该文件在系统中扮演着核心角色。
* **复杂度**：多个函数的复杂度较高，特别是 initUI 和 updateChords 函数，需要重点关注可维护性。
* **调用链深度**：多个函数的调用链深度较高，特别是 pressingEvent 函数，可能存在性能风险。

通过上述分析，可以发现 VirtualKeyboard.py 文件在系统中扮演着核心角色，但存在一些潜在问题和改进空间。建议在重构时重点关注函数职责分离、模块化设计和异常处理，以提高代码的可维护性和健壮性。

# 四、模块明细

以下是chordPrediction项目中VirtualKeyboard.py子模块的主要模块信息：

## VirtualKeyboard.py

| 模块名称 | 类型 | 完整路径 | 函数数量 |
| --- | --- | --- | --- |
| VirtualKeyboard.py | 文件 | /Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py | 25 |

### VirtualKeyboard.py

**基本信息：**

* **类型：** 文件
* **路径：** VirtualKeyboard.py
* **函数数量：** 25
* **初次分析时间：** 2025-08-19T14:40:17.371473+08:00
* **更新时间：** 2025-08-19T14:41:13.652331+08:00

**模块描述：**

该文件 `VirtualKeyboard.py` 主要用于创建一个智能化音乐创作工具的用户界面，支持MIDI输入设备或键盘映射，提供和弦识别、预测模型选择、音乐风格选择、序列输入和保存等功能。文件中实现的核心功能包括初始化pygame和pygame.midi，设置MIDI输入设备或启用键盘映射模式，跟踪按下的键，初始化用户界面，处理键盘按键事件，更新和弦信息，以及保存和清除序列。该文件在项目中扮演着核心角色，负责提供用户交互界面和处理用户输入，是整个音乐创作工具的用户界面层。文件的设计模式或架构特点包括使用了Qt的信号与槽机制来处理用户界面事件，以及使用了定时器来定时更新界面信息。

# 五、函数明细

## 重点信息统计

### 前五个重点关注文件（按重要性排序）

1. VirtualKeyboard.py (重要性得分: 52.462)

### 函数统计维度分析(排名前五)

| 统计维度 | 数值 | 函数名称 | 文件路径 |
| --- | --- | --- | --- |
| 被引用次数 | 2次 | changeWhiteSheet | VirtualKeyboard.py |
| 被引用次数 | 2次 | pressingEvent | VirtualKeyboard.py |
| 被引用次数 | 2次 | shaderLists | VirtualKeyboard.py |
| 被引用次数 | 2次 | changeBlackSheet | VirtualKeyboard.py |
| 被引用次数 | 2次 | updateButtonColor | VirtualKeyboard.py |
| 扇出数 | 7次 | initUI | VirtualKeyboard.py |
| 扇出数 | 2次 | onLabel | VirtualKeyboard.py |
| 扇出数 | 2次 | virtual\_key\_released | VirtualKeyboard.py |
| 扇出数 | 2次 | **init** | VirtualKeyboard.py |
| 扇出数 | 2次 | virtual\_key\_pressed | VirtualKeyboard.py |
| 复杂度 | 189 | initUI | VirtualKeyboard.py |
| 复杂度 | 70 | shaderLists | VirtualKeyboard.py |
| 复杂度 | 57 | updateChords | VirtualKeyboard.py |
| 复杂度 | 50 | pressingEvent | VirtualKeyboard.py |
| 复杂度 | 40 | reShaderLabels | VirtualKeyboard.py |
| 调用链深度 | 5层 | **init** | VirtualKeyboard.py |
| 调用链深度 | 4层 | on\_virtual\_key\_pressed | VirtualKeyboard.py |
| 调用链深度 | 4层 | on\_virtual\_key\_released | VirtualKeyboard.py |
| 调用链深度 | 4层 | initUI | VirtualKeyboard.py |
| 调用链深度 | 3层 | virtual\_key\_released | VirtualKeyboard.py |

**上述统计中涉及的函数：**

**updateChords** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于更新和弦序列并进行和弦预测。它会检查所有选中的复选框，读取对应的模型文件，构建马尔科夫链，并根据当前的和弦序列进行预测。预测结果会显示在界面上。

**on\_virtual\_key\_pressed** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于处理虚拟键盘按键被按下时的事件，调用虚拟键盘的虚拟按键被按下方法。

**updateButtonColor** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于更新虚拟键盘上按钮的颜色，根据按键状态和键值来决定按钮的背景颜色。

**initUI** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于初始化一个智能化音乐创作工具的用户界面，包括设置窗口大小和标题、创建按键网格、显示和弦识别信息、提供预测模型选择、多选框用于选择音乐风格、输入序列信息以及保存和清除序列的功能。

**virtual\_key\_released** [VirtualKeyboard.py](#virtualkeyboard.py): 处理虚拟钢琴键盘释放事件，更新按键状态并触发相关操作。

**on\_virtual\_key\_released** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于处理虚拟键盘中按键释放的事件。当按键被释放时，它会调用虚拟键盘对象的virtual\_key\_released方法，并传递按键的索引作为参数。

**shaderLists** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于在虚拟键盘界面中显示标注库和历史记录，并通过列表和堆叠布局进行切换。

**pressingEvent** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于处理键盘按键事件，检测按下的琴键并识别和弦，更新显示和弦信息，并将和弦信息缓存到本地。

**virtual\_key\_pressed** [VirtualKeyboard.py](#virtualkeyboard.py): 处理虚拟钢琴键盘按下事件，更新按键状态并触发相关操作。

**onLabel** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于处理用户输入的标签文件名，并根据文件名生成对应的模型文件路径。如果文件已存在，则在文件末尾追加新的数据；如果文件不存在，则创建新文件并写入初始数据。最后，刷新虚拟键盘界面中的标注库和历史记录。

**init** [VirtualKeyboard.py](#virtualkeyboard.py): 该类用于管理MIDI输入设备或键盘映射，支持加载已有的索引和映射文件，或者从头开始创建新的索引和映射。它初始化pygame和pygame.midi，设置MIDI输入设备或启用键盘映射模式，并跟踪按下的键。同时，它还提供了一个用户界面，用于显示和弦识别信息、选择预测模型、选择音乐风格、输入序列信息以及保存和清除序列的功能。

**changeWhiteSheet** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.white\_color。

**changeBlackSheet** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.black\_color。

**reShaderLabels** [VirtualKeyboard.py](#virtualkeyboard.py): 该函数用于重新设置一个滚动区域中的标签布局，动态加载文件夹中的标签文件，并将它们以多选框的形式水平排列，每行最多显示5个多选框。

## 函数信息(按重要性排序)

### 1. initUI ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：99-248
* 重要性得分：3.65802
* 被引用次数：1
* 扇出数：7
* 复杂度：189
* 调用链深度：4
* 功能描述： 该函数用于初始化一个智能化音乐创作工具的用户界面，包括设置窗口大小和标题、创建按键网格、显示和弦识别信息、提供预测模型选择、多选框用于选择音乐风格、输入序列信息以及保存和清除序列的功能。
* 实现流程： 设置窗口大小和标题。 根据是否使用MIDI键盘，显示不同的标题信息。 创建一个网格布局，用于显示按键。 为每个按键设置样式，并连接按键点击事件。 显示和弦识别和预测信息的标签。 添加一个按钮，用于打开概率转移矩阵分析图。 初始化和弦序列显示区域，并连接按键点击事件。 创建单选按钮，用于选择预测模型的准确度。 显示预选音乐风格的多选框，并从指定目录加载文件。 创建输入序列信息的输入框和按钮，并连接保存、标注和清除事件。 显示标注库和历史记录，并通过列表和堆叠布局进行切换。 设置状态展示框，显示当前监听状态。 将所有组件添加到主布局中，并设置窗口的布局。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setFixedSize,setWindowTitle,info,QLabel,setAlignment,QGridLayout,setObjectName,setHorizontalSpacing,setVerticalSpacing,QPushButton,changeWhiteSheet,changeBlackSheet,addWidget,connect,QHBoxLayout,setSpacing,QFont,setFont,onPressed,addLayout,QRadioButton,setChecked,QVBoxLayout,QScrollArea,setWidgetResizable,QWidget,setLayout,setWidget,listdir,filePath,splitext,QCheckBox,setText,QLineEdit,onRecord,onLabel,onClear,shaderLists,setStyleSheet,
* 内部依赖描述：
  + changeWhiteSheet: 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.white\_color。
  + changeBlackSheet: 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.black\_color。
  + onPressed: 该函数用于处理按键事件，当按键被按下时，记录按键信息并检查队列中是否存在该索引位置的元素，如果存在则移除该元素。
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。
  + onRecord: 该函数用于在虚拟键盘界面中记录用户输入的数据，并将其保存到指定的文件中。如果文件已存在，则在文件末尾追加数据；如果文件不存在，则创建新文件并写入初始数据。记录的数据包括用户输入的文本、一个队列数组和另一个文本字段。记录完成后，刷新虚拟键盘界面中的标注库和历史记录。
  + onLabel: 该函数用于处理用户输入的标签文件名，并根据文件名生成对应的模型文件路径。如果文件已存在，则在文件末尾追加新的数据；如果文件不存在，则创建新文件并写入初始数据。最后，刷新虚拟键盘界面中的标注库和历史记录。
  + onClear: 该函数用于清空队列并重置虚拟键盘上的按钮文本。
  + shaderLists: 该函数用于在虚拟键盘界面中显示标注库和历史记录，并通过列表和堆叠布局进行切换。

**代码片段：**

def initUI(self):  
 # 设置窗口大小和标题  
 self.setFixedSize(1200, 850)  
 self.setWindowTitle('智能化音乐创作工具')  
 if self.NoneMIDI:  
 logging.info('使用<英文键盘>映射模式')  
 title = QLabel("使用<英文键盘>映射模式")  
 else:  
 logging.info('成功接入MIDI键盘')  
 title = QLabel('虚拟MIDI键盘')  
 title.setAlignment(Qt.AlignCenter)  
  
 grid = QGridLayout()  
 grid.setObjectName("midi")  
 grid.setHorizontalSpacing(0)  
 grid.setVerticalSpacing(0)  
 grid.setAlignment(Qt.AlignTop)  
 for i in range(len(self.keys)):  
 key = QPushButton('')  
 key.setObjectName(self.values[i].name)  
 if self.keys[i] == 'w':  
 key.setFixedSize(self.key\_width, self.white\_key\_height)  
 self.changeWhiteSheet(key)  
 else:  
 key.setFixedSize(int(self.key\_width / 2), self.black\_key\_height)  
 self.changeBlackSheet(key)  
 # key.clicked.connect(lambda state, i=i: self.onPressed(i))  
 grid.addWidget(key, 0, i)  
 self.keys\_button.append(key)  
  
 grid.setAlignment(Qt.AlignCenter)  
  
 ch = QLabel('识别当前和弦: ')  
 ch.setAlignment(Qt.AlignCenter)  
 ch.setObjectName("chords")  
 self.vbox.addWidget(ch)  
  
 ch = QLabel('预测下一个和弦: ')  
 ch.setAlignment(Qt.AlignCenter)  
 ch.setObjectName("next")  
 self.vbox.addWidget(ch)  
  
 self.buttonV = QPushButton("打开概率转移矩阵分析图", self)  
 self.buttonV.clicked.connect(self.open\_image\_window)  
 self.vbox.addWidget(self.buttonV)  
  
 # 初始化和弦序列  
 chTitle = QLabel("实时记录最新十条和弦(松开琴键以写入,点击和弦以移出序列)：")  
 chTitle.setObjectName("chordsQueueTitle")  
 self.vbox.addWidget(chTitle)  
 self.chordsQueue = QHBoxLayout()  
 self.chordsQueue.setSpacing(0)  
 self.chordsQueue.setObjectName("chordsQueue")  
 # 初始化渲染和弦序列  
 for i in range(0, self.MAX\_QUEUE):  
 ql = QPushButton("")  
 ql.setObjectName("ch" + str(i))  
 font = QFont('Calibri', 16)  
 ql.setFont(font)  
 # self.reShaderButton(ql)  
 ql.clicked.connect(lambda state, i=i: self.onPressed(i))  
 self.chordsQueue.addWidget(ql)  
 self.vbox.addLayout(self.chordsQueue)  
  
 # 创建3个单选按钮  
 self.radio\_btn1 = QRadioButton('一阶')  
 self.radio\_btn2 = QRadioButton('二阶')  
 self.radio\_btn3 = QRadioButton('三阶')  
 # 设置默认选中  
 self.radio\_btn2.setChecked(True)  
 self.hbox\_radio = QHBoxLayout()  
 self.hbox\_radio.setObjectName("radio")  
 self.hbox\_radio.addWidget(QLabel('预测模型准确度'))  
 self.hbox\_radio.addWidget(self.radio\_btn1)  
 self.hbox\_radio.addWidget(self.radio\_btn2)  
 self.hbox\_radio.addWidget(self.radio\_btn3)  
 self.vbox.addLayout(self.hbox\_radio)  
  
 ch = QLabel('预选音乐风格/标签（上下滑动选择更多）: ')  
 ch.setAlignment(Qt.AlignCenter)  
 self.vbox.addWidget(ch)  
 # 遍历目录下的文件，添加到多选框中  
 self.labelBox = QVBoxLayout()  
 self.labelBox.setObjectName("labelO")  
 self.scroll\_area = QScrollArea()  
 self.scroll\_area.setWidgetResizable(True)  
 self.scroll\_area.setObjectName("scroll")  
 self.scroll\_content = QWidget(self.scroll\_area)  
 self.scroll\_content.setObjectName("labelContent")  
 self.scroll\_content.setLayout(self.labelBox)  
 self.scroll\_area.setWidget(self.scroll\_content)  
 labelhbox = QHBoxLayout()  
 count = 0 # 记录已添加的多选框数量  
 for file\_name in os.listdir(filePath('labels/')):  
 name, extension = os.path.splitext(file\_name)  
 check\_box = QCheckBox(name, self)  
 check\_box.setText(name)  
 labelhbox.addWidget(check\_box)  
 count += 1  
 if count >= 5: # 超过x个数量，则另起一行继续水平排列  
 self.labelBox.addLayout(labelhbox)  
 labelhbox = QHBoxLayout()  
 count = 0  
 if count > 0: # 处理剩余不到x个的多选框  
 self.labelBox.addLayout(labelhbox)  
 self.vbox.addWidget(self.scroll\_area)  
  
 hbox2 = QHBoxLayout()  
 hbox3 = QHBoxLayout()  
 hbox4 = QHBoxLayout()  
 hbox5 = QHBoxLayout()  
 self.label\_1 = QLabel("序列命名：", self)  
 self.edit\_1 = QLineEdit(self)  
 self.label\_2 = QLabel("标签：", self)  
 self.edit\_2 = QLineEdit(self)  
 self.label\_3 = QLabel("备注：", self)  
 self.edit\_3 = QLineEdit(self)  
 # 创建“保存”和“标注”按钮  
 self.save\_btn = QPushButton("保存序列", self)  
 ...

### 2. **init** ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：75-97
* 重要性得分：3.06672
* 被引用次数：1
* 扇出数：2
* 复杂度：31
* 调用链深度：5
* 功能描述： 该类用于管理MIDI输入设备或键盘映射，支持加载已有的索引和映射文件，或者从头开始创建新的索引和映射。它初始化pygame和pygame.midi，设置MIDI输入设备或启用键盘映射模式，并跟踪按下的键。同时，它还提供了一个用户界面，用于显示和弦识别信息、选择预测模型、选择音乐风格、输入序列信息以及保存和清除序列的功能。
* 实现流程： 初始化pygame和pygame.midi。 设置MIDI输入设备或启用键盘映射模式。 跟踪按下的键。 初始化用户界面，包括设置窗口大小和标题、创建按键网格、显示和弦识别信息、提供预测模型选择、多选框用于选择音乐风格、输入序列信息以及保存和清除序列的功能。 创建QTimer对象，用于定时更新和弦信息。 创建另一个QTimer对象，用于定时更新按钮颜色和MIDI输入。 清除焦点，确保界面不会自动聚焦到某个控件上。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： **init**,flush,QVBoxLayout,initUI,findChild,QTimer,connect,clearFocus,
* 内部依赖描述：
  + **init**: 该函数用于初始化一个隐马尔可夫模型（HMM），专门用于和弦序列的预测。它接受一个和弦序列列表和隐藏状态数作为输入，并为每个和弦分配一个唯一的索引。初始化过程中，它会计算不同和弦的数量，并建立索引到和弦的映射。此外，它还会随机初始化HMM模型的参数，包括状态转移矩阵A、发射矩阵B和初始状态概率pi。
  + initUI: 该函数用于初始化一个智能化音乐创作工具的用户界面，包括设置窗口大小和标题、创建按键网格、显示和弦识别信息、提供预测模型选择、多选框用于选择音乐风格、输入序列信息以及保存和清除序列的功能。

**代码片段：**

def \_\_init\_\_(self, NoneMIDI=False):  
 super().\_\_init\_\_()  
 sys.stdout.flush()  
 self.NoneMIDI = NoneMIDI  
 # 添加布局  
 self.vbox = QVBoxLayout()  
 self.initUI()  
 # 当前和弦展示项  
 self.chords = self.findChild(QLabel, "chords")  
 self.next = self.findChild(QLabel, "next")  
 self.listWidget = self.findChild(QListWidget, "listO")  
 self.stackedWidget = self.findChild(QStackedLayout, "stackO")  
 self.labelBox = self.findChild(QVBoxLayout, "labelO")  
 self.scroll\_content = self.findChild(QWidget, "labelContent")  
 self.scroll\_area = self.findChild(QScrollArea, "scroll")  
  
 self.timer = QTimer(self)  
 self.timer.timeout.connect(self.updateChords)  
  
 self.updateTimer = QTimer(self)  
 self.updateTimer.timeout.connect(self.updateButtonColor)  
 self.updateTimer.timeout.connect(self.updateMIDI)  
 self.clearFocus()

### 3. shaderLists ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：255-309
* 重要性得分：2.87681
* 被引用次数：2
* 复杂度：70
* 调用链深度：1
* 功能描述： 该函数用于在虚拟键盘界面中显示标注库和历史记录，并通过列表和堆叠布局进行切换。
* 实现流程： 检查传入的listWidget和stackedWidget是否为None，如果是，则创建新的QListWidget和QStackedLayout，并设置它们的属性。 定义标注库和历史记录的目录路径。 遍历标注库目录中的文件，为每个文件创建一个QListWidget项，并读取文件内容到QTextEdit中，设置QTextEdit为只读，然后将QTextEdit添加到QStackedLayout中。 遍历历史记录目录中的文件，为每个文件创建一个QListWidget项，并读取文件内容到QTextEdit中，设置QTextEdit为只读，然后将QTextEdit添加到QStackedLayout中。 将QListWidget的当前行变化信号连接到QStackedLayout的setCurrentIndex方法，实现列表和堆叠布局的切换。 如果创建了新的QListWidget和QStackedLayout，则将它们添加到传入的vbox布局中，使用QHBoxLayout进行水平布局。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： QListWidget,QStackedLayout,setObjectName,setMaximumWidth,listdir,filePath,addItem,QTextEdit,setReadOnly,textCursor,insertText,addWidget,connect,QHBoxLayout,addLayout,
* 内部依赖描述：
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。

**代码片段：**

def shaderLists(self, vbox, listWidget=None, stackedWidget=None):  
 type = 0  
 # 读取模型  
 if listWidget is None and stackedWidget is None:  
 listWidget = QListWidget(self)  
 stackedWidget = QStackedLayout()  
 listWidget.setObjectName("listO")  
 stackedWidget.setObjectName("stackO")  
 listWidget.setMaximumWidth(250)  
 type = 1  
 directory\_labels = "labels/"  
 directory\_records = "records/"  
 for file\_name in os.listdir(filePath(directory\_labels)):  
 item = '标注库- ' + str(file\_name)  
 listWidget.addItem(item)  
 text = ""  
 # 创建一个多行文本域控件  
 text\_edit = QTextEdit(text, self)  
 # 设置文本域为只读  
 text\_edit.setReadOnly(True)  
 cursor = text\_edit.textCursor()  
 with open(filePath(directory\_labels) + str(file\_name), 'r', encoding='utf-8') as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 text += line + '\n'  
 cursor.insertText(line)  
 cursor.insertText('\n')  
 lineNum += 1  
  
 stackedWidget.addWidget(text\_edit)  
 for file\_name in os.listdir(filePath(directory\_records)):  
 item = '历史记录- ' + str(file\_name)  
 listWidget.addItem(item)  
 text = ""  
 # 创建一个多行文本域控件  
 text\_edit = QTextEdit(text, self)  
 # 设置文本域为只读  
 text\_edit.setReadOnly(True)  
 cursor = text\_edit.textCursor()  
 with open(filePath(directory\_records) + str(file\_name), 'r', encoding='utf-8') as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 text += line + "\n"  
 cursor.insertText(line)  
 cursor.insertText('\n')  
 lineNum += 1  
 stackedWidget.addWidget(text\_edit)  
 listWidget.currentRowChanged.connect(stackedWidget.setCurrentIndex)  
 if type == 1:  
 hbox = QHBoxLayout()  
 hbox.addWidget(listWidget)  
 hbox.addLayout(stackedWidget)  
 vbox.addLayout(hbox)

### 4. pressingEvent ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：466-505
* 重要性得分：2.79248
* 被引用次数：2
* 复杂度：50
* 调用链深度：1
* 功能描述： 该函数用于处理键盘按键事件，检测按下的琴键并识别和弦，更新显示和弦信息，并将和弦信息缓存到本地。
* 实现流程： 获取正在按下的所有琴键，并记录它们的索引和名称。 如果按下的琴键数量在3到5之间，检测和弦并更新显示。 如果按下的琴键数量超过预设值，更新预设和弦缓存。 如果按下的琴键数量为0，检查是否有预设和弦，如果有，根据和弦是否重复更新队列，并启动异步线程将和弦信息写入本地文件。 更新预设和弦数量计数器。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： info,detectElement,getNormalChord,QLabel,setText,text,last,FileWriteWorker,globalInstance,start,
* 内部依赖描述：
  + getNormalChord: 该函数用于检测音乐中的正常和弦。
  + last: 该函数用于获取队列的最后一个元素。如果队列不为空，则返回队列的最后一个元素；如果队列为空，则不返回任何内容。

**代码片段：**

def pressingEvent(self):  
 # 获取正在按下的所有琴键  
 i = 0  
 pressingCache = []  
 pressing = []  
 for isPressed in self.key\_pressed:  
 if isPressed:  
 pressingCache.append({'index': i, 'name': self.values[i]})  
 pressing.append(self.values[i])  
 i += 1  
 if 3 <= len(pressing) <= 5:  
 logging.info(f"正在按下的所有琴键：{pressing}")  
 detectElement = musicUtils.detectElement(pressing)  
 chord\_name = detectElement.getNormalChord()  
 item = QLabel('当前和弦： ' + chord\_name)  
 self.chords.setText(item.text())  
 if len(pressing) > self.PRE\_COUNT:  
 self.PRE\_CHORD = (chord\_name, pressingCache)  
 logging.info(self.QUEUE.array)  
 elif len(pressing) == 0:  
 item = QLabel('当前和弦： ')  
 self.chords.setText(item.text())  
 if self.PRE\_CHORD is not None:  
 chord\_name, pressing\_notes = self.PRE\_CHORD  
 if chord\_name != self.QUEUE.last():  
 if self.QUEUE.length() < self.MAX\_QUEUE:  
 logging.info("push")  
 self.QUEUE.push(chord\_name)  
 else:  
 logging.info("pop")  
 self.QUEUE.pop()  
 self.QUEUE.push(chord\_name)  
 # 将 pressing（当前按下的音符）与 self.PRE\_CHORD（当前和弦）的映射关系缓存到本地  
 # 启动异步线程进行文件写入  
 worker = FileWriteWorker(chord\_name, pressing\_notes)  
 QThreadPool.globalInstance().start(worker)  
  
 else:  
 logging.info(f"重复和弦{self.QUEUE.last()}")  
 self.PRE\_COUNT = len(pressing)

### 5. updateChords ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：592-638
* 重要性得分：2.76800
* 复杂度：57
* 调用链深度：1
* 功能描述： 该函数用于更新和弦序列并进行和弦预测。它会检查所有选中的复选框，读取对应的模型文件，构建马尔科夫链，并根据当前的和弦序列进行预测。预测结果会显示在界面上。
* 实现流程： 遍历所有选中的复选框，读取对应的模型文件，构建马尔科夫链。 根据当前的和弦序列进行预测。 更新显示的图像以反映新的和弦序列。 显示预测结果，包括预测的和弦名称和匹配比率。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： findChildren,isChecked,info,filePath,text,set\_chord\_sequences,ChordPredictor,index,predict\_chord,setText,
* 内部依赖描述：
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。
  + set\_chord\_sequences: 该函数用于设置新的和弦序列，并在和弦序列发生变化时更新显示的图像。
  + index: 该函数用于从队列中获取指定索引位置的元素。
  + predict\_chord: 该函数用于预测音乐中的和弦。它根据当前和弦序列和马尔科夫链模型来选择下一个和弦，并返回下一个和弦及其概率。

**代码片段：**

def updateChords(self):  
 chord\_sequences = []  
 for cbox in self.findChildren(QCheckBox):  
 if cbox.isChecked():  
 logging.info(cbox)  
 with open(filePath('labels/') + cbox.text() + '.model', 'r', encoding='utf-8'  
 ) as f:  
 lineNum = 1  
 for line in f:  
 if lineNum != 1:  
 cline = line.split('||')  
 arr\_str = cline[0].split(',,')  
 chord\_sequences.append(arr\_str)  
 chord\_sequences.append([self.ENDING])  
 lineNum += 1  
 if len(chord\_sequences) != 0:  
 logging.info(f"构建马尔科夫链{chord\_sequences}")  
 if self.image\_window:  
 self.image\_window.set\_chord\_sequences(chord\_sequences)  
 else:  
 if self.image\_window:  
 self.image\_window.set\_chord\_sequences([])  
  
 order = 1 if self.radio\_btn1.isChecked() else 2 if self.radio\_btn2.isChecked() else 3  
  
 if self.QUEUE.length() >= order and len(chord\_sequences) > 0:  
 logging.info(f"阶数: {order}")  
 if chord\_sequences != self.current\_chord\_sequences or self.last\_order is None or self.last\_order != order:  
 self.current\_chord\_sequences = chord\_sequences  
 self.predictor = ChordPredictor(chord\_sequences, order)  
 self.last\_order = order  
 logging.info(f"初始化预测器，阶数：{order}")  
 if order == 3:  
 current\_chords = [self.QUEUE.index(self.QUEUE.length() - 3), self.QUEUE.index(self.QUEUE.length() - 2),  
 self.QUEUE.index(self.QUEUE.length() - 1)]  
 else:  
 current\_chords = [self.QUEUE.index(self.QUEUE.length() - 2), self.QUEUE.index(self.QUEUE.length() - 1)]  
 predictions = self.predictor.predict\_chord(current\_chords)  
  
 check\_sequence = []  
 for sequence in chord\_sequences:  
 check\_sequence += sequence  
 logging.info(f'预测结果: {predictions}')  
 self.next.setText(  
 "预测下一个和弦：(<和弦名称>,<匹配比率%>) " + str(predictions))  
 else:  
 self.next.setText("预测下一个和弦: ")

### 6. onLabel ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：367-393
* 重要性得分：2.63177
* 被引用次数：1
* 扇出数：2
* 复杂度：34
* 调用链深度：3
* 功能描述： 该函数用于处理用户输入的标签文件名，并根据文件名生成对应的模型文件路径。如果文件已存在，则在文件末尾追加新的数据；如果文件不存在，则创建新文件并写入初始数据。最后，刷新虚拟键盘界面中的标注库和历史记录。
* 实现流程： 检查输入框中是否有文本内容，如果没有则记录日志并返回。 获取指定目录的完整路径。 将输入框中的文本按逗号分割成多个文件名。 遍历每个文件名，添加’.model’后缀，并记录日志。 检查文件是否存在，如果存在则在文件末尾追加新的数据，如果不存在则创建新文件并写入初始数据。 刷新虚拟键盘界面中的标注库和历史记录。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： text,filePath,info,exists,write,reShaderLists,reShaderLabels,
* 内部依赖描述：
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。
  + reShaderLists: 该函数用于清空虚拟键盘界面中的标注库和历史记录，并重新加载新的标注库和历史记录。
  + reShaderLabels: 该函数用于重新设置一个滚动区域中的标签布局，动态加载文件夹中的标签文件，并将它们以多选框的形式水平排列，每行最多显示5个多选框。

**代码片段：**

def onLabel(self):  
 if self.edit\_2.text() != "":  
 directory = filePath("labels/")  
 filenames = self.edit\_2.text().split(',')  
 for filename in filenames:  
 filename += ".model"  
 logging.info(filename)  
 file\_path = os.path.join(directory, filename)  
 if filename != "":  
 if os.path.exists(file\_path):  
 with open(os.path.join(directory, filename), 'a', encoding='utf-8') as f:  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
  
 else:  
 with open(os.path.join(directory, filename), 'w', encoding='utf-8') as f:  
 f.write(self.edit\_2.text())  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
 # 刷新list  
 self.reShaderLists()  
 # 刷新label  
 self.reShaderLabels()  
 else:  
 logging.info("填写为空")

### 7. updateButtonColor ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：413-439
* 重要性得分：2.52716
* 被引用次数：2
* 扇出数：2
* 复杂度：29
* 调用链深度：2
* 功能描述： 该函数用于更新虚拟键盘上按钮的颜色，根据按键状态和键值来决定按钮的背景颜色。
* 实现流程： 遍历所有按键，检查每个按键是否被按下。 如果按键被按下且键值为’w’，调用changeWhiteSheet函数，传入True以改变按钮背景颜色为灰色。 如果按键被按下且键值不为’w’，调用changeBlackSheet函数，传入True以改变按钮背景颜色为灰色。 如果按键未被按下且键值为’w’，调用changeWhiteSheet函数，不传入参数以恢复按钮背景颜色为白色。 如果按键未被按下且键值不为’w’，调用changeBlackSheet函数，不传入参数以恢复按钮背景颜色为黑色。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： changeWhiteSheet,changeBlackSheet,
* 内部依赖描述：
  + changeWhiteSheet: 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.white\_color。
  + changeBlackSheet: 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.black\_color。

**代码片段：**

def updateButtonColor(self):  
 # # 获取正在按下的所有琴键  
 # j = 0  
 # pressing = []  
 # for isPressed in self.key\_pressed:  
 # if isPressed:  
 # pressing.append(self.values[j])  
 # j += 1  
 #  
 # if self.QUEUE.length()!=0:  
 # detectElement = musicUtils.detectElement(pressing)  
 # ch = detectElement.getChordAttr()  
 # logging.info(ch)  
 # get\_chord(ch.root, ch.chord\_type)  
 i = 0  
 for button in self.keys\_button:  
 if self.key\_pressed[i] == True:  
 if self.keys[i] == 'w':  
 self.changeWhiteSheet(button, True)  
 else:  
 self.changeBlackSheet(button, True)  
 else:  
 if self.keys[i] == 'w':  
 self.changeWhiteSheet(button)  
 else:  
 self.changeBlackSheet(button)  
 i += 1

### 8. reShaderLabels ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：311-335
* 重要性得分：2.52117
* 被引用次数：1
* 复杂度：40
* 调用链深度：1
* 功能描述： 该函数用于重新设置一个滚动区域中的标签布局，动态加载文件夹中的标签文件，并将它们以多选框的形式水平排列，每行最多显示5个多选框。
* 实现流程： 清空当前的QHBoxLayout。 创建一个新的QVBoxLayout用于存放多选框。 创建一个新的QWidget作为滚动区域的子部件，并设置其布局为QVBoxLayout。 遍历指定文件夹中的所有文件，提取文件名并创建QCheckBox。 将QCheckBox添加到当前的QHBoxLayout中，并计数。 当QHBoxLayout中的多选框数量达到5个时，将其添加到QVBoxLayout中，并创建一个新的QHBoxLayout。 处理剩余的多选框，确保它们也被添加到QVBoxLayout中。 将QVBoxLayout设置为QWidget的布局，并将QWidget设置为滚动区域的子部件。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： children,deleteLater,QVBoxLayout,setObjectName,QWidget,setLayout,setWidget,QHBoxLayout,listdir,filePath,splitext,QCheckBox,setText,addWidget,addLayout,
* 内部依赖描述：
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。

**代码片段：**

def reShaderLabels(self):  
 # 清空QHBoxLayout  
 for ch in self.scroll\_content.children():  
 ch.deleteLater()  
  
 self.labelBox = QVBoxLayout()  
 self.labelBox.setObjectName("labelO")  
 self.scroll\_content = QWidget(self.scroll\_area)  
 self.scroll\_content.setObjectName("labelContent")  
 self.scroll\_content.setLayout(self.labelBox)  
 self.scroll\_area.setWidget(self.scroll\_content)  
 labelhbox = QHBoxLayout()  
 count = 0 # 记录已添加的多选框数量  
 for file\_name in os.listdir(filePath('labels/')):  
 name, extension = os.path.splitext(file\_name)  
 check\_box = QCheckBox(name, self)  
 check\_box.setText(name)  
 labelhbox.addWidget(check\_box)  
 count += 1  
 if count >= 5: # 超过x个数量，则另起一行继续水平排列  
 self.labelBox.addLayout(labelhbox)  
 labelhbox = QHBoxLayout()  
 count = 0  
 if count > 0: # 处理剩余不到x个的多选框  
 self.labelBox.addLayout(labelhbox)

### 9. onRecord ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：345-365
* 重要性得分：2.46266
* 被引用次数：1
* 扇出数：1
* 复杂度：27
* 调用链深度：3
* 功能描述： 该函数用于在虚拟键盘界面中记录用户输入的数据，并将其保存到指定的文件中。如果文件已存在，则在文件末尾追加数据；如果文件不存在，则创建新文件并写入初始数据。记录的数据包括用户输入的文本、一个队列数组和另一个文本字段。记录完成后，刷新虚拟键盘界面中的标注库和历史记录。
* 实现流程： 检查编辑框1的文本是否为空，如果为空则记录日志并返回。 生成记录文件的完整路径，路径由当前文件所在目录的上两级目录和’records/’目录拼接而成。 检查记录文件是否存在。 如果文件存在，以追加模式打开文件，写入一个换行符，然后将队列数组和另一个文本字段以特定格式写入文件。 如果文件不存在，以写入模式打开文件，写入初始文本、一个换行符，然后将队列数组和另一个文本字段以特定格式写入文件。 调用reShaderLists函数刷新虚拟键盘界面中的标注库和历史记录。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： text,filePath,exists,write,reShaderLists,info,
* 内部依赖描述：
  + filePath: 该函数用于生成指定目录的完整路径。它通过获取当前文件所在目录的上两级目录，然后将传入的文件目录拼接在一起，最终返回完整的路径。
  + reShaderLists: 该函数用于清空虚拟键盘界面中的标注库和历史记录，并重新加载新的标注库和历史记录。

**代码片段：**

def onRecord(self):  
 if self.edit\_1.text() != "":  
 directory = filePath("records/")  
 filename = self.edit\_1.text() + ".model"  
 file\_path = os.path.join(directory, filename)  
 if os.path.exists(file\_path):  
 with open(os.path.join(directory, filename), 'a', encoding='utf-8') as f:  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
 else:  
 with open(os.path.join(directory, filename), 'w', encoding='utf-8') as f:  
 f.write(self.edit\_2.text())  
 f.write("\n")  
 csv\_str = ',,'.join(self.QUEUE.array)  
 f.write(csv\_str + "||" + self.edit\_3.text())  
  
 # 刷新list  
 self.reShaderLists()  
 else:  
 logging.info("填写为空")

### 10. keyPressEvent ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：573-589
* 重要性得分：2.21286
* 被引用次数：1
* 扇出数：1
* 复杂度：23
* 调用链深度：1
* 功能描述： 该函数处理键盘按键事件，当按下特定键（96、126、183）时，切换监听模式，并更新状态标签的显示和样式。其他按键事件则调用父类的处理方法。
* 实现流程： 设置焦点策略为强焦点。 记录按键事件。 检查按键是否为96、126或183。 如果按键是上述之一，切换监听模式，并更新状态标签的文本和样式。 记录监听模式的当前状态。 如果按键不是上述之一，调用父类的按键处理方法。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setFocusPolicy,info,key,setText,setStyleSheet,keyPressEvent,
* 内部依赖描述：
  + keyPressEvent: 该函数处理键盘按键事件，当按下特定键（96、126、183）时，切换监听模式，并更新状态标签的显示和样式。其他按键事件则调用父类的处理方法。

**代码片段：**

def keyPressEvent(self, event):  
 self.setFocusPolicy(Qt.StrongFocus)  
 logging.info(f"Key pressed: {event.key()}")  
 if str(event.key()) == '96' or str(event.key()) == '126' or str(event.key()) == '183':  
 if self.NoneMIDI:  
 # 切换监听模式  
 self.listening\_enabled = not self.listening\_enabled  
 if self.listening\_enabled:  
 self.status\_label.setText("<~键>控制当前监听状态：已开启")  
 self.status\_label.setStyleSheet("font-size: 12px; color: green;")  
 else:  
 self.status\_label.setText("<~键>控制当前监听状态：已关闭")  
 self.status\_label.setStyleSheet("font-size: 12px; color: red;")  
 logging.info(f"监听模式：{'开启' if self.listening\_enabled else '关闭'}")  
 else:  
 # 调用父类的按键处理  
 super().keyPressEvent(event)

### 11. reShaderLists ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：337-343
* 重要性得分：2.09779
* 被引用次数：2
* 扇出数：1
* 复杂度：11
* 调用链深度：2
* 功能描述： 该函数用于清空虚拟键盘界面中的标注库和历史记录，并重新加载新的标注库和历史记录。
* 实现流程： 清空列表小部件（listWidget）中的所有元素。 删除堆叠小部件（stackedWidget）中的所有子部件。 调用内部函数 shaderLists，重新加载标注库和历史记录，并通过列表和堆叠布局进行切换。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： clear,widget,removeWidget,shaderLists,
* 内部依赖描述：
  + clear: 该函数用于清空队列中的所有元素。
  + shaderLists: 该函数用于在虚拟键盘界面中显示标注库和历史记录，并通过列表和堆叠布局进行切换。

**代码片段：**

def reShaderLists(self):  
 self.listWidget.clear()  
 # 删除所有的QWidget  
 while self.stackedWidget.count() > 0:  
 widget = self.stackedWidget.widget(0)  
 self.stackedWidget.removeWidget(widget)  
 self.shaderLists(self.vbox, self.listWidget, self.stackedWidget)

### 12. virtual\_key\_pressed ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：544-553
* 重要性得分：2.09377
* 被引用次数：1
* 扇出数：2
* 复杂度：13
* 调用链深度：3
* 功能描述： 处理虚拟钢琴键盘按下事件，更新按键状态并触发相关操作。
* 实现流程： 检查按键索引是否在有效范围内（0到87）。 如果有效，更新对应按键的状态为按下。 调用update方法更新界面。 调用pressingEvent方法处理按键按下事件。 调用updateButtonColor方法更新按键颜色以反映当前状态。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： update,pressingEvent,updateButtonColor,
* 内部依赖描述：
  + pressingEvent: 该函数用于处理键盘按键事件，检测按下的琴键并识别和弦，更新显示和弦信息，并将和弦信息缓存到本地。
  + updateButtonColor: 该函数用于更新虚拟键盘上按钮的颜色，根据按键状态和键值来决定按钮的背景颜色。

**代码片段：**

def virtual\_key\_pressed(self, index):  
 """  
 处理虚拟钢琴键盘按下事件。  
 """  
 if index >= 0 and index < 88:  
 # 更新键盘按键状态  
 self.key\_pressed[index] = True  
 self.update()  
 self.pressingEvent()  
 self.updateButtonColor()

### 13. virtual\_key\_released ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：555-564
* 重要性得分：2.09377
* 被引用次数：1
* 扇出数：2
* 复杂度：13
* 调用链深度：3
* 功能描述： 处理虚拟钢琴键盘释放事件，更新按键状态并触发相关操作。
* 实现流程： 检查按键索引是否在有效范围内（0到87）。 如果有效，将对应按键的状态更新为未按下。 调用update方法更新界面显示。 调用pressingEvent方法处理按键释放事件。 调用updateButtonColor方法更新按键颜色以反映当前状态。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： update,pressingEvent,updateButtonColor,
* 内部依赖描述：
  + pressingEvent: 该函数用于处理键盘按键事件，检测按下的琴键并识别和弦，更新显示和弦信息，并将和弦信息缓存到本地。
  + updateButtonColor: 该函数用于更新虚拟键盘上按钮的颜色，根据按键状态和键值来决定按钮的背景颜色。

**代码片段：**

def virtual\_key\_released(self, index):  
 """  
 处理虚拟钢琴键盘释放事件。  
 """  
 if index >= 0 and index < 88:  
 # 更新键盘按键状态  
 self.key\_pressed[index] = False  
 self.update()  
 self.pressingEvent()  
 self.updateButtonColor()

### 14. reShaderButton ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：522-542
* 重要性得分：2.04717
* 复杂度：29
* 调用链深度：1
* 功能描述： 该函数用于调整按钮的字体大小，以确保按钮上的文本能够完全填充按钮的宽度，避免文本过长导致的显示问题。
* 实现流程： 获取按钮的最大宽度。 获取按钮的当前字体。 创建一个QFontMetrics对象，用于计算不同字体/大小下的文本宽度。 计算当前字体下文本的宽度。 记录按钮宽度和文本宽度的日志信息。 如果文本宽度超过按钮宽度，则根据按钮宽度自动调整字体大小，使文本填充按钮。 计算调整后的字体大小，并设置新的字体大小。 将调整后的字体应用到按钮上。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： width,font,QFontMetrics,text,info,pointSize,setPointSize,setFont,
* 内部依赖描述：

**代码片段：**

def reShaderButton(self, button):  
 # 获取按钮的最大宽度  
 max\_width = button.width()  
  
 # 获取按钮的字体  
 font = button.font()  
  
 # 创建一个QFontMetrics对象，用于计算不同字体/大小下的文本大小  
 metrics = QFontMetrics(font)  
  
 # 计算当前字体下文本的宽度  
 text\_width = metrics.width(button.text())  
  
 logging.info(f"按钮宽度{max\_width}和文本宽度{text\_width}")  
  
 # 如果文本太长，则根据按钮宽度自动调整字体大小，使文本填充按钮  
 if text\_width > max\_width:  
 # 计算调整后的字体大小  
 font\_size = max\_width \* font.pointSize() // text\_width  
 font.setPointSize(font\_size)  
 button.setFont(font)

### 15. updateMIDI ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：442-463
* 重要性得分：1.99449
* 复杂度：26
* 调用链深度：1
* 功能描述： 该函数用于更新MIDI界面的和弦序列显示。根据队列的长度，更新相应的按钮文本和字体样式。
* 实现流程： 检查队列长度是否大于10。 如果队列长度大于10，遍历前MAX\_QUEUE个元素，更新对应按钮的文本和字体样式。 如果队列长度小于或等于10，遍历队列中的所有元素，更新对应按钮的文本和字体样式。 对于队列长度小于MAX\_QUEUE的剩余按钮，清空文本并更新字体样式。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： findChild,QFont,setFont,setText,
* 内部依赖描述：

**代码片段：**

def updateMIDI(self):  
 # 渲染实时和弦序列  
 if self.QUEUE.length() > 10:  
 for i in range(0, self.MAX\_QUEUE):  
 label = self.findChild(QPushButton, "ch" + str(i))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText(self.QUEUE.array[i])  
 else:  
 for i in range(0, self.QUEUE.length()):  
 label = self.findChild(QPushButton, "ch" + str(i))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText(self.QUEUE.array[i])  
 for j in range(self.QUEUE.length(), self.MAX\_QUEUE):  
 label = self.findChild(QPushButton, "ch" + str(j))  
 font = QFont('Calibri', 16)  
 label.setFont(font)  
 # self.reShaderButton(label)  
 label.setText("")

### 16. onClear ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：395-398
* 重要性得分：1.70410
* 被引用次数：1
* 复杂度：7
* 调用链深度：1
* 功能描述： 该函数用于清空队列并重置虚拟键盘上的按钮文本。
* 实现流程： 调用self.QUEUE.clear()方法清空队列中的所有元素。 使用for循环遍历从0到self.MAX\_QUEUE-1的索引。 在每次循环中，使用self.findChild(QPushButton, “ch” + str(i))方法找到对应的QPushButton控件。 调用找到的QPushButton控件的setText(““)方法，将按钮文本重置为空字符串。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： clear,findChild,setText,
* 内部依赖描述：
  + clear: 该函数用于清空队列中的所有元素。

**代码片段：**

def onClear(self):  
 self.QUEUE.clear()  
 for i in range(0, self.MAX\_QUEUE):  
 self.findChild(QPushButton, "ch" + str(i)).setText("")

### 17. changeBlackSheet ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：514-520
* 重要性得分：1.66491
* 被引用次数：2
* 复杂度：8
* 调用链深度：1
* 功能描述： 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.black\_color。
* 实现流程： 检查isGray参数的值。 如果isGray为True，则设置按钮的样式为灰色背景。 如果isGray为False，则设置按钮的样式为self.black\_color背景。 样式包括边框、圆角和左边距。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setStyleSheet,
* 内部依赖描述：

### 18. closeEvent ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：566-571
* 重要性得分：1.66355
* 被引用次数：1
* 扇出数：1
* 复杂度：7
* 调用链深度：1
* 功能描述： 处理窗口关闭事件，确保在关闭窗口时执行一些清理操作。
* 实现流程： 调用父类的closeEvent方法，确保父类的关闭逻辑得到执行。 调用self.close()方法，关闭当前窗口。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： closeEvent,
* 内部依赖描述：
  + closeEvent: 处理窗口关闭事件，确保在关闭窗口时执行一些清理操作。

### 19. onPressed ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：400-403
* 重要性得分：1.63733
* 被引用次数：1
* 复杂度：6
* 调用链深度：1
* 功能描述： 该函数用于处理按键事件，当按键被按下时，记录按键信息并检查队列中是否存在该索引位置的元素，如果存在则移除该元素。
* 实现流程： 记录按键信息，格式为 ‘Key [i] was pressed.’，其中 [i] 是按键的索引。 检查队列的长度是否大于按键索引 [i]。 如果队列长度大于按键索引 [i]，则从队列中移除索引为 [i] 的元素。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： info,remove,
* 内部依赖描述：
  + remove: 该函数用于从队列中移除指定索引位置的元素。

### 20. changeWhiteSheet ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：507-512
* 重要性得分：1.60602
* 被引用次数：2
* 复杂度：7
* 调用链深度：1
* 功能描述： 该函数用于改变按钮的样式，根据传入的isGray参数决定按钮的背景颜色。如果isGray为True，则背景颜色为灰色；否则，背景颜色为self.white\_color。
* 实现流程： 接收两个参数：button（按钮对象）和isGray（布尔值，默认为False）。 检查isGray的值，如果为True，则设置按钮的样式为灰色背景，边框为灰色，圆角为3px。 如果isGray为False，则设置按钮的样式为self.white\_color背景，边框为灰色，圆角为3px。 使用setStyleSheet方法应用上述样式到按钮上。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： setStyleSheet,
* 内部依赖描述：

### 21. stop\_timer ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：644-646
* 重要性得分：1.33047
* 复杂度：4
* 调用链深度：1
* 功能描述： 停止定时器和更新定时器，并调用内部函数stop来停止MIDI输入并清理相关资源。
* 实现流程： 停止定时器 停止更新定时器 调用内部函数stop来停止MIDI输入并清理相关资源
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： stop,
* 内部依赖描述：
  + stop: 停止MIDI输入并清理相关资源。

### 22. open\_image\_window ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：250-253
* 重要性得分：1.31953
* 复杂度：6
* 调用链深度：1
* 功能描述： 该函数用于打开一个图像窗口，如果图像窗口不存在，则创建一个新的MatrixView实例并显示它。
* 实现流程： 检查self.image\_window是否为空，如果为空则创建一个新的MatrixView实例并赋值给self.image\_window。 调用self.image\_window的show方法，显示图像窗口。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： MatrixView,show,
* 内部依赖描述：

### 23. on\_virtual\_key\_released ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 行号位置：653-654
* 重要性得分：1.26998
* 扇出数：1
* 复杂度：3
* 调用链深度：4
* 功能描述： 该函数用于处理虚拟键盘中按键释放的事件。当按键被释放时，它会调用虚拟键盘对象的virtual\_key\_released方法，并传递按键的索引作为参数。
* 实现流程： 接收按键释放事件的索引。 调用虚拟键盘对象的virtual\_key\_released方法，并将按键索引作为参数传递给该方法。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： virtual\_key\_released,
* 内部依赖描述：
  + virtual\_key\_released: 处理虚拟钢琴键盘释放事件，更新按键状态并触发相关操作。

### 24. on\_virtual\_key\_pressed ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 行号位置：649-650
* 重要性得分：1.26998
* 扇出数：1
* 复杂度：3
* 调用链深度：4
* 功能描述： 该函数用于处理虚拟键盘按键被按下时的事件，调用虚拟键盘的虚拟按键被按下方法。
* 实现流程： 接收按键索引作为参数。 调用虚拟键盘的虚拟按键被按下方法，并传递按键索引作为参数。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： virtual\_key\_pressed,
* 内部依赖描述：
  + virtual\_key\_pressed: 处理虚拟钢琴键盘按下事件，更新按键状态并触发相关操作。

### 25. start\_timer ([VirtualKeyboard.py](file:///Users/apple/Public/generates-git/chordPrediction/VirtualKeyboard.py))

* 所属模块/包：VirtualKeyboard
* 行号位置：640-642
* 重要性得分：1.15129
* 复杂度：4
* 调用链深度：1
* 功能描述： 该函数用于启动一个定时器，用于更新虚拟键盘的显示和处理音乐预测。
* 实现流程： 调用self.timer.start(1000)启动一个定时器，定时器每1000毫秒（1秒）触发一次。 调用self.updateTimer.start(120)启动另一个定时器，定时器每120毫秒触发一次。 这两个定时器分别用于不同的更新任务，具体实现细节未在代码中展示。
* 引入包： os,io,sys,MatrixView,PyQt5.QtGui,PyQt5.QtWidgets,PyQt5.QtCore,musicpy.musicpy,service.MidiInput,service.numpyMarkov,utils,utils.FileWorker,utils.filePath,logging,
* 调用： start,
* 内部依赖描述：