**南京航空航天大学**

程序设计课程设计报告

**图 书 馆 管 理 系 统**

|  |  |
| --- | --- |
| 学号： | 082420122 |
| 姓名： | 周奕轩 |

程序演示视频播放地址：https://www.bilibili.com/video/XXXXXX

2025年5月18日

[**1. 需求分析** 3](#_Toc197791309)

[**2. 架构设计** 3](#_Toc197791310)

[**3. 模块设计** 3](#_Toc197791311)

[**4. 类与函数定义** 6](#_Toc197791312)

[**5. 算法说明** 16](#_Toc197791313)

[**6. 存在的问题与不足及对策** 17](#_Toc197791314)

[**7. 使用说明** 18](#_Toc197791315)

**1. 需求分析**

**系统功能说明**

* **播放功能**：支持播放、暂停、上一曲、下一曲等基本操作。
* **播放列表管理**：支持添加、删除、清空播放列表。
* **文件信息显示**：显示当前播放音乐的详细信息。
* **多语言支持**：支持中文和其他语言切换。
* **跨平台支持**：支持 Windows 和其他主流操作系统。

**2. 架构设计**

**系统总框图**

**3. 模块设计**

**1. 主界面模块（MainWidget）**

* 负责主窗口的显示和交互。
* 包括播放控制按钮、播放列表、歌词显示等功能。

**2. 播放控制模块**

* 控制音乐的播放、暂停、上一曲、下一曲等操作。
* 包括播放模式切换（如随机播放、单曲循环）。
* 相关类：
  + PlayPauseButton
  + LoopModeSwitcher

**3. 播放列表模块**

* 管理音乐文件的添加、删除、清空等操作。
* 相关类：
  + MusicItem

**4. 歌词显示模块**

* 显示当前播放音乐的歌词，并支持滚动和高亮。
* 相关类：
  + DisplayWidget
  + LyricsDelegate

**5. 音量控制模块**

* 控制音量大小，支持静音功能。
* 相关类：
  + ChangeVolumeButton
  + ChangeVolumeWidget

**6. 播放速度控制模块**

* 调节音乐播放速度。
* 相关类：
  + ChangePlaybackSpeedButton
  + ChangePlaybackSpeedWidget

**7. 设置模块**

* 提供用户界面，用于调整字体大小、主题等设置。
* 相关类：
  + Settings

**8. 配置管理模块**

* 负责保存和加载用户设置及播放列表。
* 相关类：
  + ConfigManager

**9. 全局配置模块**

* 定义全局变量和常量，如配置文件路径、应用设置等。
* 相关类：
  + GlobalConfigs

**11. 基础组件模块**

* 提供基础的 UI 组件和工具类。
* 相关类：
  + BasicWidget

**12. 算法模块**

* 实现随机播放和异步加载元数据等功能。
* 相关代码：
  + 随机播放算法
  + 异步加载算法

**13. 多语言支持模块**

* 提供中文和英文的界面翻译。
* 相关文件：
  + MusicPlayer\_zh\_CN.ts

**14. 样式与主题模块**

* 提供深色和浅色主题切换功能。
* 相关代码：
  + loadStyleSheet方法

**15. 跨平台支持模块**

* 使用 QWindowKit 实现跨平台的窗口管理和样式支持。

**4. 类与函数定义**

**MainWidget**

class MainWidget : public QMainWindow

{

Q\_OBJECT

public:

MainWidget(QWidget \*parent = nullptr);

~MainWidget();

Theme currentTheme{};

QWK::WidgetWindowAgent \*windowAgent;

Q\_SIGNALS:

void themeChanged(Theme t);

protected:

void resizeEvent(QResizeEvent\* event) override;

void dragEnterEvent(QDragEnterEvent \*event) override;

void dropEvent(QDropEvent \*event) override;

void keyPressEvent(QKeyEvent\* event) override;

protected Q\_SLOTS:

void updateMusicList(const QStringList& list);

void changeMusic(QListWidgetItem\* item);

void on\_volumeChanged(int value);

void on\_positionChanged(qint64 value);

private slots:

void on\_pushButton\_ShowPlayList\_clicked();

void on\_playPauseButton\_clicked();

void on\_horizontalSlider\_Progress\_valueChanged(int value);

void on\_horizontalSlider\_Progress\_sliderReleased();

void showContextMenu(const QPoint &pos);

void removeSelectedItem(QListWidgetItem \*item);

void on\_loopModeSwitched(LoopModeSwitcher::Mode mode);

void on\_mediaStatusChanged(QMediaPlayer::MediaStatus status);

void on\_enableListWidget();

void on\_pushButton\_Previous\_clicked();

void on\_pushButton\_Next\_clicked();

void on\_pushButton\_Maximize\_clicked();

void on\_pushButton\_showFileDetails\_clicked();

private:

void installWindowAgent();

void loadStyleSheet(Theme theme);

bool event(QEvent\* event) override;

void paintEvent(QPaintEvent\* event) override;

void updateTimeLabel(qint64 current, qint64 total);

void updateMusicNameLabel(const QString& musicInfo, int availableWidth);

Ui::MainWidget \*ui;

bool m\_paintTransparentBackground{ true };

std::shared\_ptr<QMediaPlayer> m\_mediaPlayer;

QMediaMetaData m\_currentMetaData;

std::unique\_ptr<QTimer> m\_playTimer;

QString m\_currentMusicInfo{ QStringLiteral("...") };

QListWidgetItem\* m\_playingMusicItem{ nullptr };

QStringList m\_musicList;

LoopModeSwitcher::Mode m\_loopMode = LoopModeSwitcher::Mode::ListLoop;

std::unique\_ptr<QTimer> m\_playbackTimer;

std::unique\_ptr<QTimer> m\_autoFocusTimer;

std::vector<QListWidgetItem\*> m\_playedMusics;

};

**MusicItem**

class MusicItem : public QListWidgetItem

{

*//Q\_OBJECT*

public:

QUrl url;

MusicItem(QListWidget \*parent, const QUrl& url);

~MusicItem();

QFuture<QMediaMetaData> load(std::shared\_ptr<QMediaPlayer>& player);

QMediaMetaData mediaMetaData() const;

private:

void updateText();

QMediaMetaData m\_mediaMetaData;

};

**Settings**

class Settings : public BasicWidget

{

Q\_OBJECT

public:

explicit Settings(QWidget \*parent = nullptr);

~Settings();

private slots:

void on\_spinBox\_fontSize\_valueChanged(int arg1);

private:

Ui::Settings\* ui;

protected:

void closeEvent(QCloseEvent\* event) override;

};

**PlayPauseButton**

class PlayPauseButton : public QPushButton

{

Q\_OBJECT

public:

PlayPauseButton(QWidget\* parent);

~PlayPauseButton();

bool isPlaying() const;

PlayPauseButton& setIsPlaying(bool isPlaying);

Q\_SIGNALS:

void PlayPauseClicked();

void IsPlayingChanged(bool);

protected:

void mousePressEvent(QMouseEvent\* event) override {

if (event->button() == Qt::MouseButton::LeftButton) {

setIsPlaying(!isPlaying());

emit PlayPauseClicked();

}

QPushButton::mousePressEvent(event);

}

private:

void \_updateText();

bool \_isPlaying{ false };

};

**LoopModeSwitcher**

class LoopModeSwitcher : public QPushButton {

Q\_OBJECT

public:

enum Mode {

ListLoop,

SingleLoop,

RandomPlay

};

Q\_ENUM(Mode)

explicit LoopModeSwitcher(QWidget\* parent = nullptr);

Mode currentMode() const;

signals:

void playModeSwitched(Mode mode);

private slots:

void switchMode();

private:

Mode mode{ ListLoop };

void updateIcon();

};

**LyricsDelegate**

class LyricsDelegate : public QStyledItemDelegate

{

Q\_OBJECT

public:

explicit LyricsDelegate(QObject \*parent = nullptr);

void setHighlightedIndex(int index);

void setMaxWidth(int width); *// New method to set maximum width for lyrics*

void paint(QPainter \*painter, const QStyleOptionViewItem &option, const QModelIndex &index) const override;

QSize sizeHint(const QStyleOptionViewItem &option, const QModelIndex &index) const override;

private:

int highlightedIndex;

int maxWidth; *// Stores the maximum width for lyrics*

};

**GlobalConfigs**

class GlobalConfigs {

public:

static inline const QString CONFIG\_FILE\_PATH() {

return QCoreApplication::applicationDirPath() + QStringLiteral("/config.cfg");

}

static inline const QString LOCAL\_PLAY\_LIST() {

return QCoreApplication::applicationDirPath() + QStringLiteral("/local.playlist");

}

static inline QMap<QString, QVariant> APPLICATION\_SETTINGS;

};

**DisplayWidget**

class DisplayWidget : public QWidget

{

Q\_OBJECT

public:

DisplayWidget(QWidget \*parent = nullptr);

~DisplayWidget();

void updateMetaData(const QString &musicFilePath, const QPixmap &albumCover);

void loadLyrics(const QString &lyricsFilePath);

QListView\* lyricsView; *// Replace QTextBrowser with QListView*

signals:

void requestJumpToTimestamp(qint64 timestamp); *// Signal to request a jump to a specific timestamp*

protected:

void resizeEvent(QResizeEvent \*event) override; *// Override resizeEvent*

public slots:

void updateHighlightedLyric(qint64 currentTime); *// New method for updating highlighted lyric*

private slots:

void handleLyricDoubleClicked(const QModelIndex &index); *// Slot to handle double-click events*

private:

void adjustLyricsMaxWidth(); *// Adjust maximum width for lyrics*

QVector<QPair<qint64, QString>> lyricsData; *// Stores parsed LRC data*

int currentLyricIndex; *// Tracks the currently highlighted lyric*

LyricsDelegate \*lyricsDelegate; *// Custom delegate for styling lyrics*

};

**ChangeVolumeWidget && ChangeVolumeButton**

class ChangeVolumeWidget : public BasicWidget {

Q\_OBJECT

public:

ChangeVolumeWidget(QWidget\* parent);

~ChangeVolumeWidget();

QTimer \_hideTimer;

QSlider\* slider;

*//Q\_PROPERTY(QColor backgroundColor READ backgroundColor WRITE setBackgroundColor NOTIFY backgroundColorChanged FINAL)*

*//QColor backgroundColor() const;*

*//void setBackgroundColor(const QColor &newBackgroundColor);*

public slots:

void on\_themeChanged(Theme t);

*//signals:*

*// void backgroundColorChanged();*

protected:

void focusOutEvent(QFocusEvent\* event) override;

void paintEvent(QPaintEvent\* event) override;

private:

QVBoxLayout\* verticallayout;

Theme currentTheme{ Dark };

*//QColor m\_backgroundColor;*

};

class ChangeVolumeButton : public QPushButton

{

Q\_OBJECT

public:

ChangeVolumeButton(QWidget \*parent);

~ChangeVolumeButton();

Q\_PROPERTY(int volume READ volume WRITE setVolume NOTIFY volumeChanged FINAL)

int volume() const;

void setVolume(int newVolume);

QChar getVolumeIcon() const;

signals:

void volumeChanged(int i);

protected slots:

void sliderMoved(int value);

void updateTooltip(int value);

private:

void \_setVolumeIcon();

int m\_volume{ 100 };

std::unique\_ptr<ChangeVolumeWidget> volumeWidget;

bool \_ignoreNextShow{ false };

protected Q\_SLOTS:

void mousePressEvent(QMouseEvent\* event) override;

};

**ChangePlaybackSpeedWidget && ChangePlaybackSpeedButton**

class ChangePlaybackSpeedWidget : public BasicWidget {

Q\_OBJECT

public:

ChangePlaybackSpeedWidget(QWidget\* parent);

~ChangePlaybackSpeedWidget();

QListWidget\* speedList;

QTimer \_hideTimer;

void adjustHeightBasedOnItems();

signals:

void backgroundColorChanged();

protected:

void focusOutEvent(QFocusEvent\* event) override;

void paintEvent(QPaintEvent\* event) override;

private:

QVBoxLayout\* layout;

};

class ChangePlaybackSpeedButton : public QPushButton {

Q\_OBJECT

public:

ChangePlaybackSpeedButton(QWidget\* parent);

~ChangePlaybackSpeedButton();

Q\_PROPERTY(double currentSpeed READ currentSpeed WRITE setCurrentSpeed NOTIFY currentSpeedChanged FINAL)

double currentSpeed() const;

void setCurrentSpeed(double newCurrentSpeed);

signals:

void currentSpeedChanged(double speed);

protected:

void mousePressEvent(QMouseEvent\* event) override;

private:

std::unique\_ptr<ChangePlaybackSpeedWidget> speedWidget;

bool \_ignoreNextShow{ false };

void setupSpeedOptions();

double m\_currentSpeed{ 1.0 };

};

**BasicWidget**

class BasicWidget : public QWidget

{

Q\_OBJECT

public:

BasicWidget(QWidget \*parent);

~BasicWidget();

void paintEvent(QPaintEvent\* event) override;

Q\_PROPERTY(QColor backgroundColor READ backgroundColor WRITE setBackgroundColor NOTIFY backgroundColorChanged FINAL)

Q\_PROPERTY(bool enableDWM READ enableDWM WRITE setEnableDWM NOTIFY enableDWMChanged FINAL)

QColor backgroundColor() const;

void setBackgroundColor(const QColor &newBackgroundColor);

bool enableDWM() const;

void setEnableDWM(bool newEnableDWM);

signals:

void backgroundColorChanged();

void enableDWMChanged(bool status);

protected:

QColor m\_backgroundColor;

bool m\_enableDWM{ false };

private:

};

**5. 算法说明**

**算法 1：随机播放未播放的音乐**

case LoopModeSwitcher::RandomPlay:

{

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> distrib(0, ui->listWidget\_PlayList->count() - 1);

int peddingMusicRow;

while ((peddingMusicRow = distrib(gen)) == currentMusicRow || (std::find(m\_playedMusics.begin(), m\_playedMusics.end(), ui->listWidget\_PlayList->item(peddingMusicRow)) != m\_playedMusics.end())) {};

ui->listWidget\_PlayList->setCurrentRow(peddingMusicRow);

changeMusic(ui->listWidget\_PlayList->item(peddingMusicRow));

}

**说明**：通过随机数生成器选择未播放的音乐，避免重复播放。

**算法 2：异步加载音乐元数据**

return QtConcurrent::run([this, player]() -> QMediaMetaData {

static std::atomic<bool> locker = false;

locker.wait(true);

locker.store(true);

...

locker.store(false);

locker.notify\_one();

return m\_mediaMetaData;

});

**说明**：使用 QtConcurrent::run 实现异步加载，避免阻塞主线程。

**6. 存在的问题与不足及对策**

**问题与不足**

1. **随机算法效率低**：当前随机播放算法在播放列表较大时，可能会因多次查找未播放音乐而导致性能下降。
2. **锁机制存在性能瓶颈**：使用原子变量实现的锁机制可能会在高并发场景下导致线程等待时间过长，影响程序响应速度。
3. **歌词显示功能不完善**：歌词滚动效果不够平滑.
4. **缺乏单元测试覆盖**：部分核心功能（如随机播放、异步加载元数据）缺乏充分的单元测试，可能导致潜在的稳定性问题。
5. **不支持逐字歌词**：目前仅支持逐行歌词显示，未实现逐字歌词的解析和显示。
6. **不支持收藏功能**：当前播放列表不支持收藏功能，用户无法标记喜欢的音乐。
7. **不支持切换播放列表**：当前实现仅支持单一播放列表，无法切换不同的播放列表。
8. **仅支持中文和英文且未提供主动切换开关**：目前仅实现了中文和英文两种语言的翻译，且切换语言的开关未提供给用户，仅能通过删除 qm文件切换中英文。

**对策**

1. **优化随机算法**：使用哈希表记录已播放音乐，减少查找时间。
2. **改进锁机制**：使用更高效的锁机制或无锁编程。
3. **改进歌词显示**：
   * 使用更高效的绘制方法，减少重绘次数。
   * 增加歌词滚动速度调节功能。
4. **增加单元测试**：为核心功能编写单元测试，确保代码的稳定性和可靠性。
5. **支持逐字歌词**：增加对逐字歌词的解析和显示功能。
6. **添加收藏功能**：在播放列表中添加收藏功能，允许用户标记喜欢的音乐。
7. **支持切换播放列表**：实现播放列表的切换功能，允许用户在不同的播放列表之间切换。
8. **增加语言切换功能**：提供用户界面，允许用户主动选择语言，避免手动删除 qm文件。

**7. 使用说明**

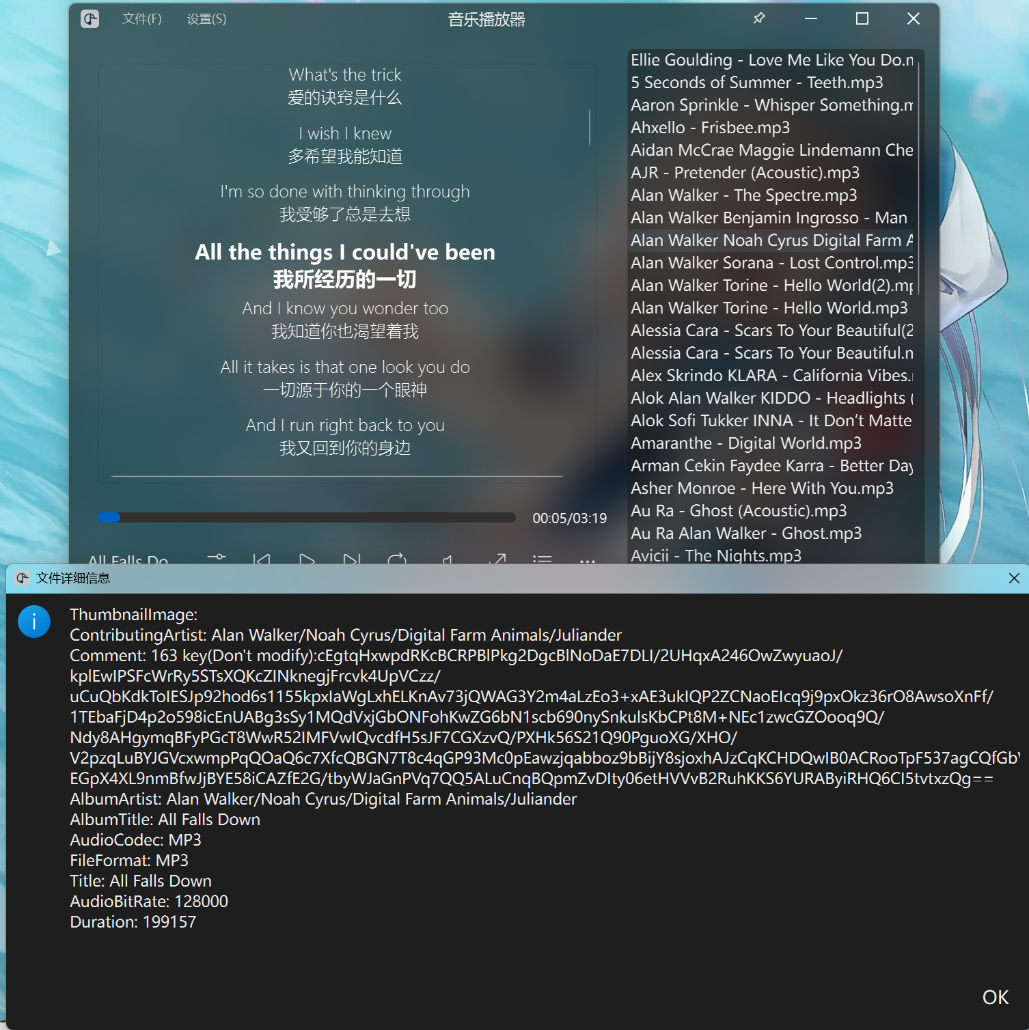
1. 添加音频到播放列表：
   * 将文件直接拖进播放器窗口即可在添加到播放列表中。
   * 将文件拖放到可执行文件上（即传入文件所在路径的启动参数）即可在添加到播放列表中并自动播放传入的第一个音频文件。
   * 可在程序菜单中批量添加音频文件，支持多选。
   * 支持多个文件拖入，自动筛选音频文件放入播放列表，可过滤非音频文件。
   * 支持通过右键菜单删除播放列表内的音频文件，支持通过键盘上的Delete键删除。
   * 支持通过右键菜单清空播放列表。
2. 播放音频：
   * 双击播放列表中的音频文件名即可开始播放。
   * 点击前一首或后一首按钮可切换到上一首或下一首音频。
     + 若当前播放模式为随机播放两个按钮行为一致，均为随机选择非当前音频进行播放。
     + 若当前模式为单曲循环或列表循环则播放当前音频的下一首或上一首。
   * 播放速度调节：
     + 提供多种常见倍速选项供用户选择。
   * 播放进度和状态调节：
     + 播放进度条可拖动，拖动后会自动更新当前播放时间。
     + 支持按空格键暂停或继续播放。
     + 支持按左右箭头按键调整播放进度。
     + 支持双击歌词跳转至歌词所在时间。
   * 支持音频文件元数据的解析和显示。
3. 歌词显示：
   * 支持解析.lrc格式歌词文件，歌词文件名需与音频文件名一致。
   * 歌词支持自动滚动，当前歌词加粗显示，字号加大。
4. 部分设置记忆：
   * 支持调整和记忆歌词字体大小。
   * 支持记忆上次音量。

**截图**

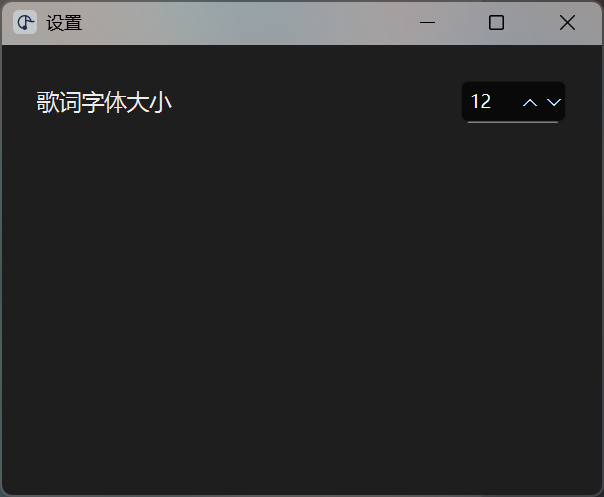
**主界面**

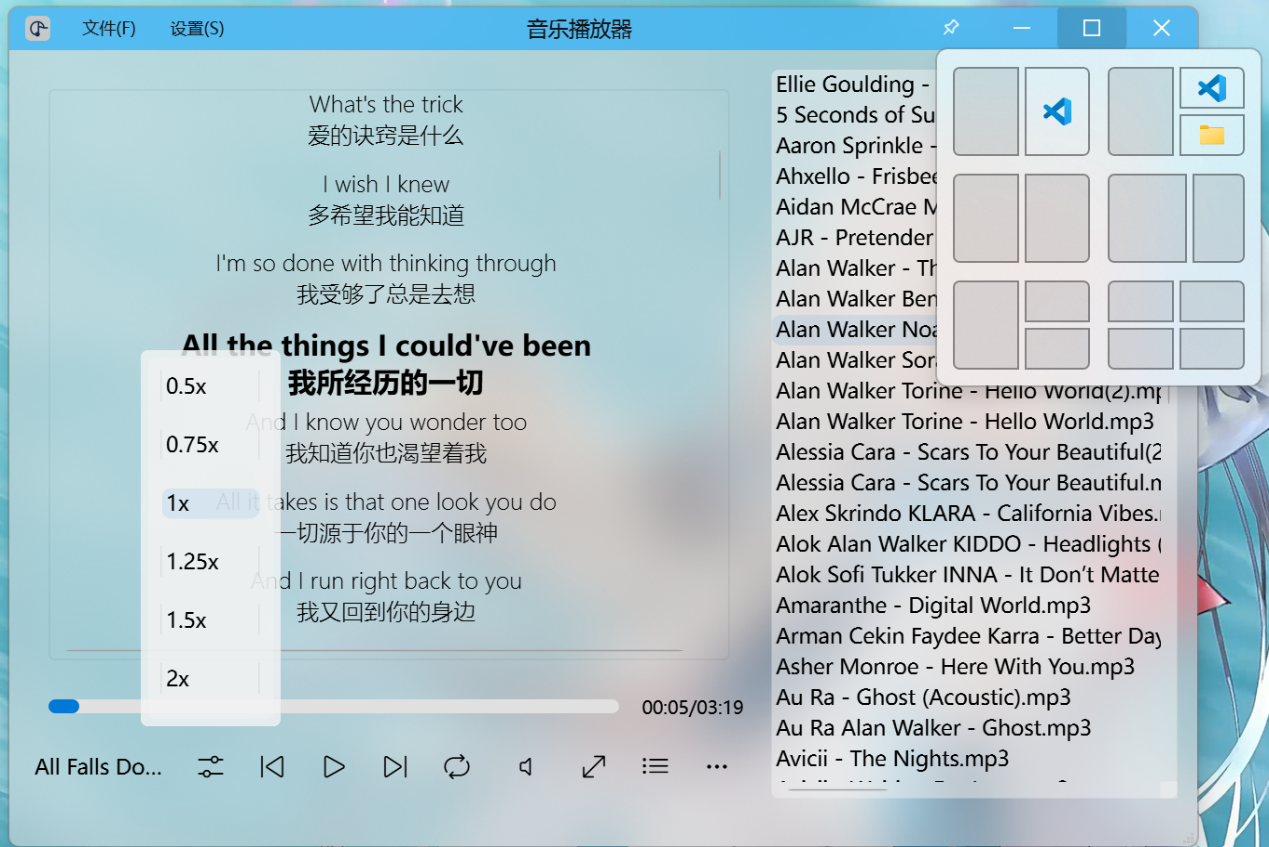


**播放中 && 音频文件详细信息**



**设置界面**



**播放速度调节 && 支持Windows 11原生分屏功能**

**隐藏播放列表 && 置于顶层**图形用户界面, 应用程序

AI 生成的内容可能不正确。

1. **程序源代码**