音乐游戏与谱面编辑器

1、课题介绍

音乐游戏（Music Game）是一种让玩家根据音乐节拍，对游戏中的音符做出相应操作，通过操作的时间精准度与连击数来计算分数的游戏，而音符出现的时机与位置则是根据谱面来确定的。谱面（Score）则是用来记录歌曲中音符信息的文档。音符（Note）是玩家的操作对象，音符拥有其对应的时间（小节与节拍）、位置、持续时间等信息，玩家在相应的时间、在相应的位置、做出单击或按下相应的持续时间等操作，则可以正确击打音符并获得分数。根据玩家击打音符的时间与音符实际对应时间的差值，游戏将给出相应的判定结果（如Perfect、Good、Bad、Miss 等）。满足一定判定结果的音符将被记入连击数（Combo），一般来说连击数越大，每个音符的最大分数也就越高。

本项目希望能够制作出一款满足以上定义的音乐游戏，并制作配套的谱面编辑器，让用户能自由地在游戏中加入自己所喜爱的歌曲，并按照自己的想法设计谱面。同时本项目也尝试引入了用户系统，做了简单的用户管理操作与密码处理。在游戏逻辑合理的基础上，本项目也精心设计了用户界面，让游戏画面更丰富美观，而谱面编辑器更加简洁易用。

2、任务需求分析报告

2.1 任务概述

该项目需建立游戏与谱面编辑器两个工程。由于游戏工程不需要做文档打开等处理，我们将这个工程设置为基于对话框的程序。而谱面编辑器需要对谱面文档进行读写等操作，因此我们将这个工程设置为单文档应用程序。

游戏工程需要进行游戏界面的设计与流程、操作逻辑，让游戏能够读取磁盘上的歌曲数据并进行处理，明确游戏规则并进行实现，在此基础上做界面的改进。

谱面编辑器需要利用文档/视图结构，将谱面数据的操作反映到各个相关界面中。谱面中的音符操作与音频控制各自分栏，视图也占一栏，共三栏。音符信息视图中需要列出所有音符，能够设置音符的小节、节拍、持续时间等信息，并能查看作为参考的音符对应时间。音频控制界面能够控制音频的播放与暂停，调整音频位置，根据小节定位音频位置。视图中能够预览当前谱面，并通过音频时间控制当前所预览的时间。此外还需记录歌曲名、节拍速度（BPM）、作者、音符总数、第一小节所对应的音频时间（起始时间）、歌曲背景图片路径、封面图片路径、预览音乐路径、全场音乐路径等信息。

2.2 目标

程序不违背设计初衷与游戏规则，保证程序逻辑正确，考虑多种使用情景，尽量减少意外情况的发生。在此基础上进行界面的改进，让程序符合相应的设计规范。若仍有余力，可对整体代码做一遍梳理，对代码结构进行优化。

2.3 用户

本项目适合广大音乐游戏玩家使用，也适合对音乐感兴趣的人群、想尝试音乐游戏的人群使用。

3、程序功能设计说明

3.1 音乐游戏 Melodia

游戏拥有一个菜单栏，可进行界面跳转、显示或隐藏 FPS（帧率）、调整窗口大小、查看程序信息等操作。游戏共拥有开始、帮助、设置、歌曲选择、游戏、结算六个界面，可通过相应的鼠标或键盘操作实现界面跳转。

3.1.1 开始界面

开始界面是游戏的初始界面，拥有以下按钮：开始游戏、游戏帮助、设置、退出、登录、注册。点击开始游戏按钮可以进入歌曲选择界面。点击游戏帮助按钮可进入帮助界面。点击设置按钮可进入设置界面。点击退出按钮可在确认后退出程序。点击登录按钮将弹出登录对话框，登陆成功后将更新用户登录情况与用户存档数据。点击注册按钮将弹出注册对话框，注册成功后将更新磁盘上的用户信息文本，并将所注册的用户登录游戏。

3.1.2 帮助界面

帮助界面显示了游戏的一些基本操作，并可通过返回按钮返回至开始界面。

3.1.3 设置界面

设置界面供玩家调整音符下落速度，打开或关闭音符击打音效，并可通过返回按钮返回至开始界面。

3.1.4 用户系统

程序以文本文档（txt）的方式存储了用户的总数量、用户名、密码等信息。存储密码时程序将对密码进行加密操作后存储，读取用户信息是程序将对读取到的密码进行解密操作，以保证用户在能够正常登录的同时，密码不会被打开用户信息文本就能直接知道。

每建立一个用户时，程序将新建一个以用户名为名的文本文档，在其中每一行记录对应编号的歌曲的最高分信息（刚注册时初始化为 0）。在已有用户登录状态下游戏的歌曲选择界面将显示对应歌曲的最高分，在结算界面若获得更高分则显示“NEW BEST”字样，并在程序结束后更新存档。

3.1.5 歌曲选择界面

在歌曲选择界面，程序将读取 songs 文件夹下对应文件夹中的歌曲信息，并绘制相应的背景、封面，显示相应的歌曲信息，并能够通过按钮切换歌曲、返回开始界面、游玩该曲目。

3.1.6 游戏界面

游戏界面显示了歌曲信息、连击数与分数等信息，并在中间绘制八个音符轨道，在下方有一条判定线。程序将读取歌曲对应的谱面数据，依据这些数据进行音符的绘制与判定处理。每次按下键盘（“A”“S”“D”“F”“J”“K”“L”“;”八个键）只对相应轨道最近的音符响应一次。若在判定时间范围内，则设置该音符的判定信息并更新分数统计信息。键盘按下时可通过轨道亮起进行操作反馈。

在游戏界面中，可随时按下“ESC”键返回至歌曲选择界面，按下“Enter”键重新游玩该歌曲，也可通过菜单栏的操作进行界面跳转。

3.1.7 结算界面

结算界面显示了歌曲信息、最大连击数、分数、评价、各个判定种类的音符数量等信息。可通过与游戏界面相同的操作进行界面跳转。

3.1.8 动画系统

游戏拥有流畅的切换动画，在各个界面间切换时不生硬，进行动画绘制时占用系统资源少，并由一个控制器做动画状态的统一控制。

3.2 谱面编辑器 MelodiaEditor

项目拥有菜单栏与工具栏。菜单栏中可执行新建、打开、保存、另存为、导出的文件处理功能与查看工程信息功能。工具栏可实现新建、打开、保存、关于等快捷操作。

3.2.1 工程信息对话框

工程信息包括工程名称（曲名）、BPM（拍速）、歌曲作者、音符总数、起始时间、背景图片、封面图片、预览音乐、全长音乐。其中音符总数为只读信息，由存储音符信息的数组的大小确定。背景图片、封面图片、预览图片、全长音乐可通过打开文件对话框选取文件路径，并能检测图片大小是否符合要求（背景图片为 360px×225px，封面图片为 300px×300px）。

3.2.2 音符信息视图

该视图包括音符列表与音符信息两部分。音符列表列举了当前谱面中的所有音符的部分信息，音符信息中可设置音符所在小节与节拍、持续节拍、所在轨道，并能预览当前所选择的音符的时间与持续时间，通过添加操作将当前的音符信息添加至谱面中，通过删除操作将当前所选择的音符删除。

3.2.3 音频控制

音频时间能以滑条或数值的方式设定，可以从当前时间播放与暂停，并能通过小节与节拍定位音频时间。

3.2.4 谱面预览

谱面预览界面与游戏中的游玩界面类似，但更加简洁，并显示了各个轨道对应的号码。谱面预览界面能够显示当前时间下的在一定时间范围内的音符位置，以方便用户了解谱面设置的合理性并进行调整。

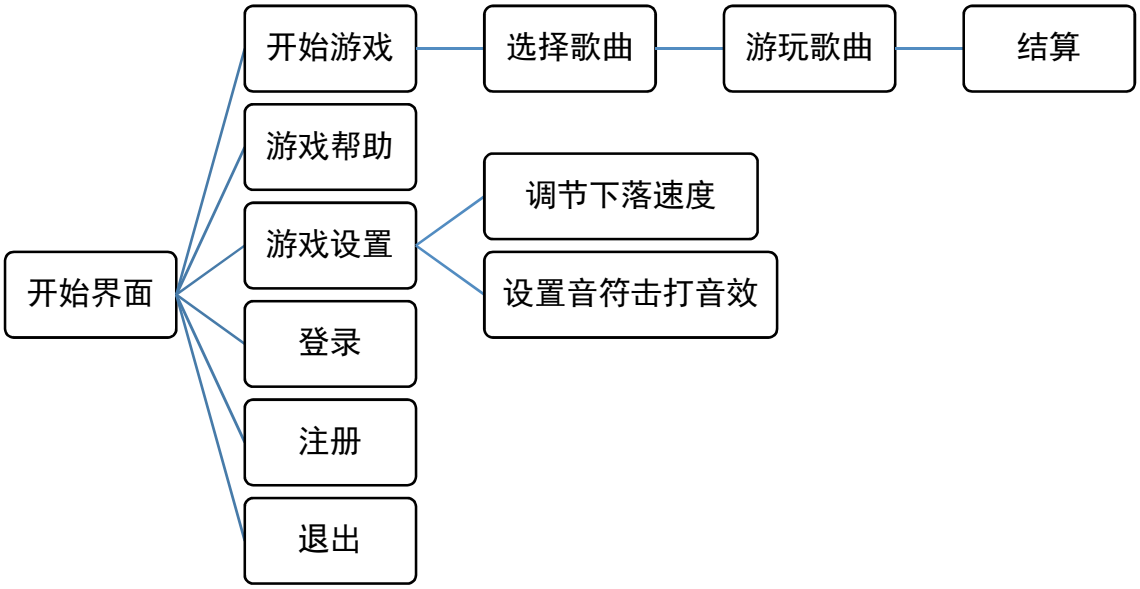
3.2.5 导出歌曲

完成后的谱面能够通过“导出”功能添加至游戏中。导出时需要将相关文件复制到对应文件夹下，将谱面信息转化成游戏所能读取的数据并写入相应的文本文档中。最后将游戏的歌曲数据进行修改（修改歌曲数、添加曲名与作曲者）。

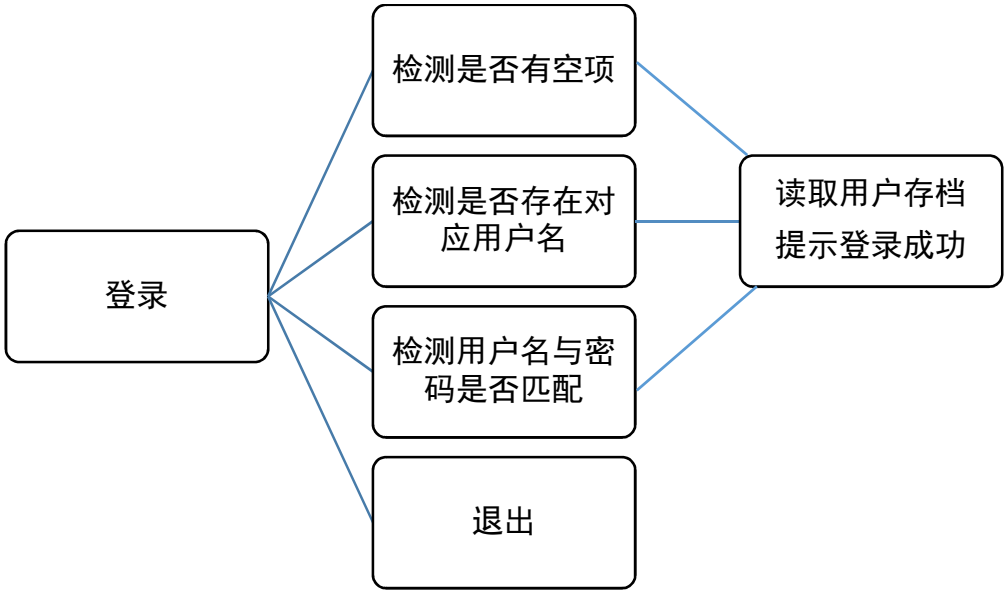
4、程序模块框图

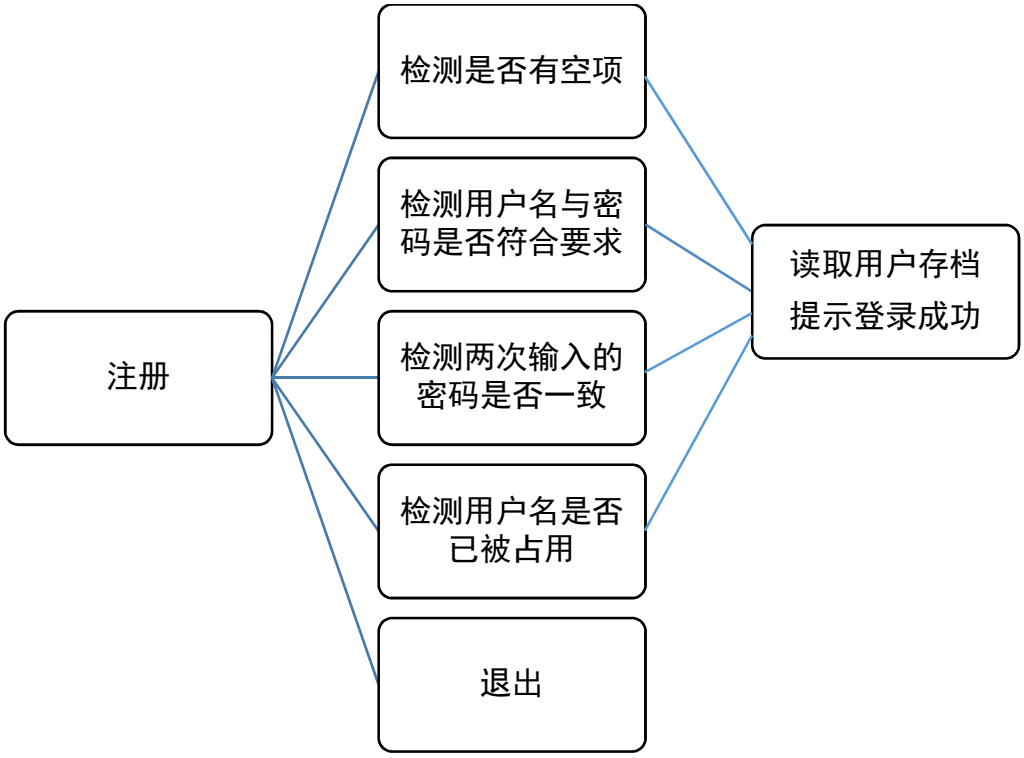
4.1 音乐游戏 Melodia 程序框图

4.1.1 主程序框图

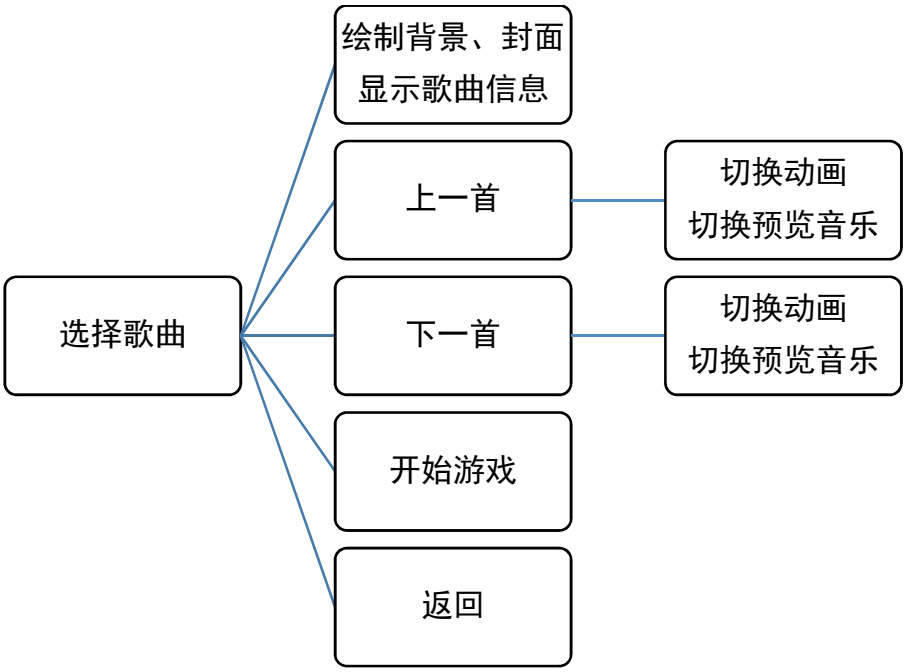


4.1.2 用户系统框图

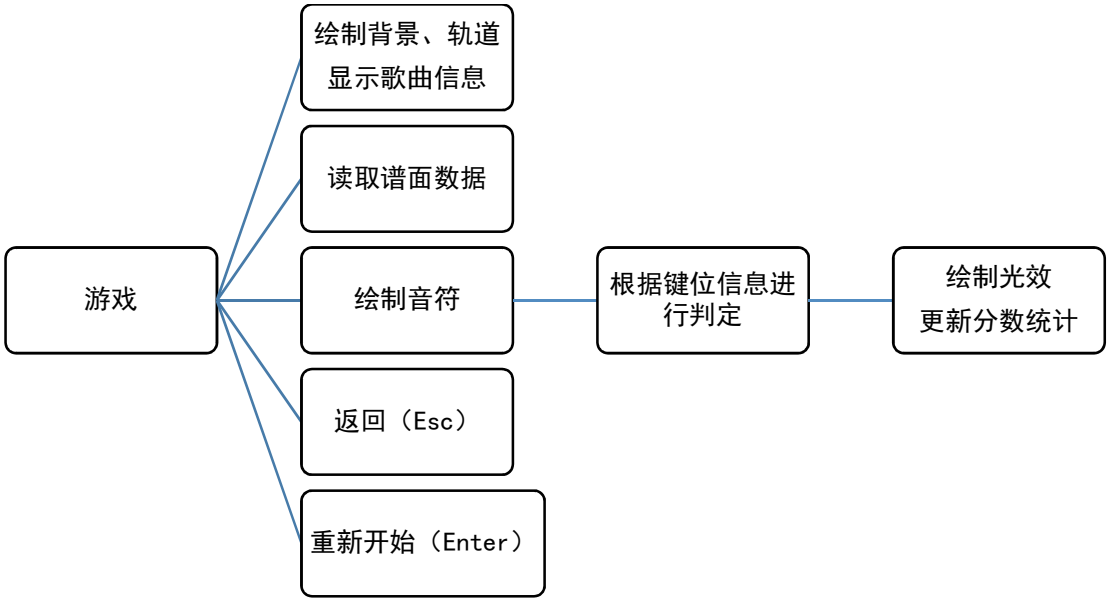




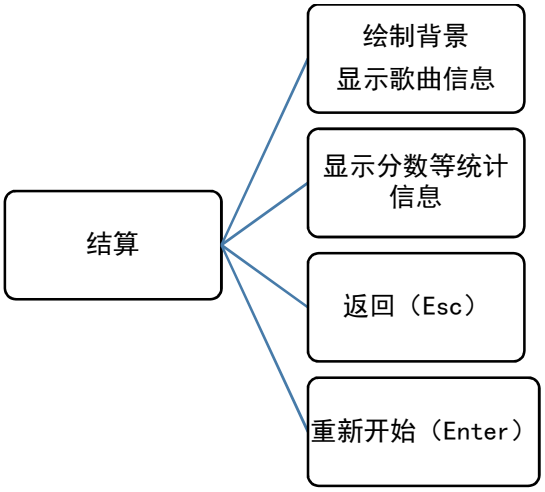
4.1.3 歌曲选择界面框图



4.1.4 游戏界面框图

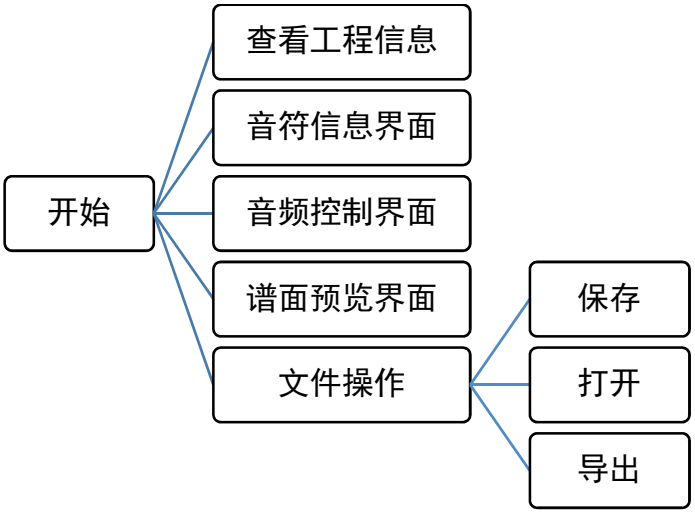


4.1.4 结算界面框图

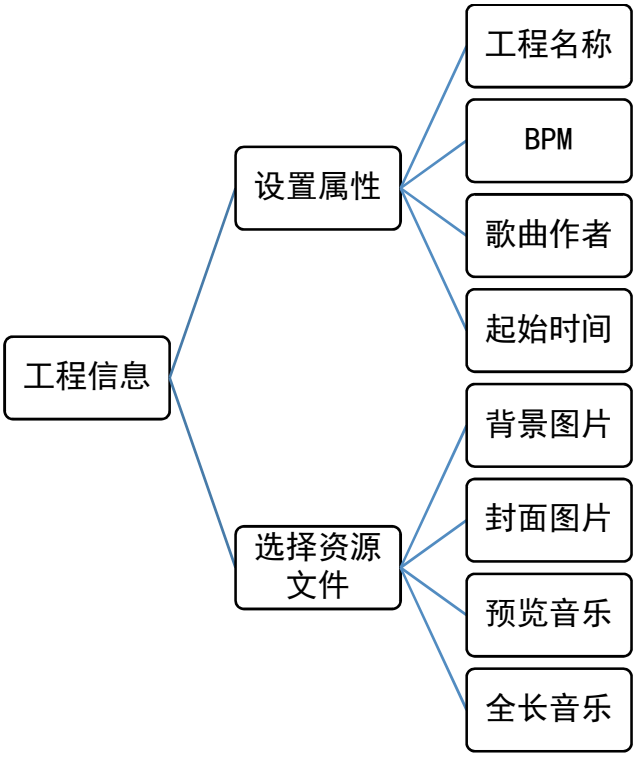


4.2 谱面编辑器 MelodiaEditor 程序框图

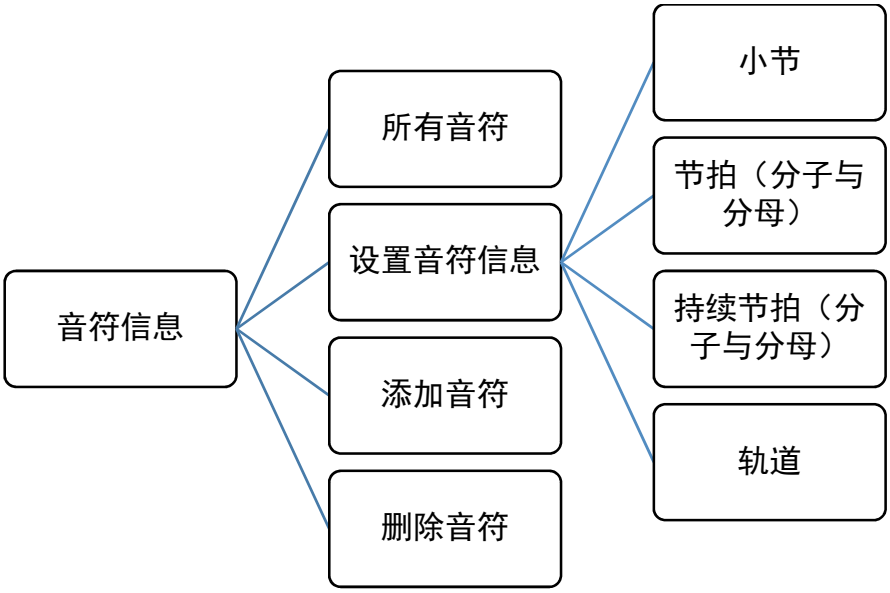
4.2.1 主程序框图



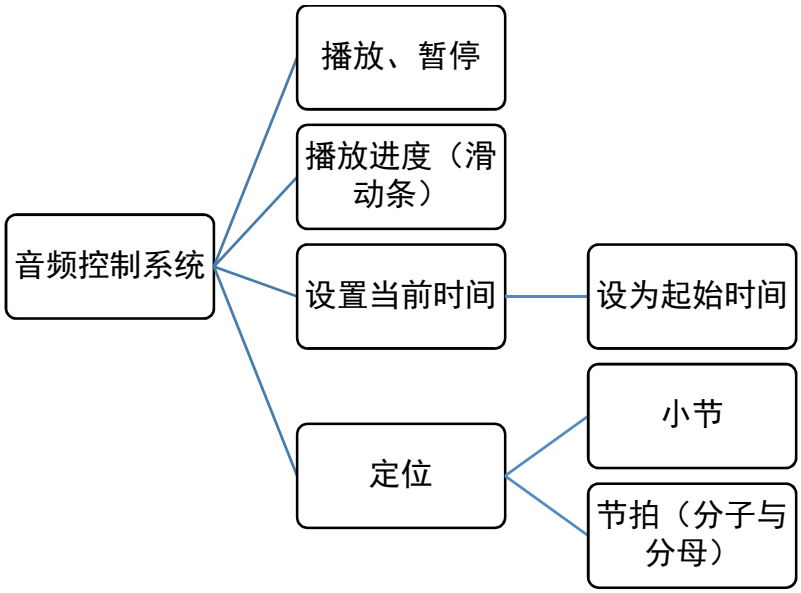
4.2.2 工程信息框图



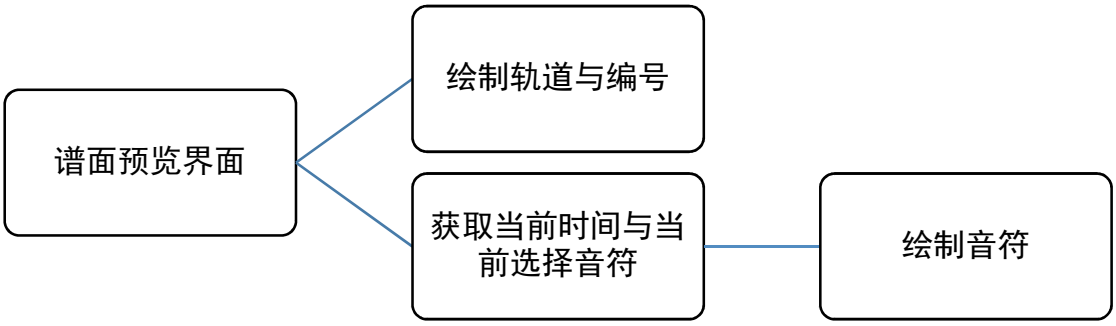
4.2.3 音符信息框图



4.2.4 音频控制系统框图



4.2.5 谱面预览视图框图



5、对象设计与算法设计描述

5.1 音乐游戏 Melodia 算法思想简介

5.1.1 双缓冲技术

在对话框的 OnPaint()函数中，所有的图形并不是同时绘制到 DC 中的，这就会导致在画面的刷新过程中出现图形闪烁的现象。为解决这个问题，我们使用了双缓冲技术。

双缓冲技术的原理是：创建一个兼容的 CDC 对象与 Cbitmap 对象，并使它们关联起来，之后的所有绘图操作都在这个兼容的 CDC 对象中进行，绘制完毕后使用 OnPaint()中的 CPaintDC 对象的 BitBlt 方法，将兼容的 CDC 对象的图像直接复制过来。这样，在屏幕中看到的画面都是完成全部绘制过程的画面，而不会有图形的闪烁。

```
GetClientRect(&area);
CPaintDC dc(this);
CDC memDC;
CBitmap memBmp;
memDC.CreateCompatibleDC(NULL);
memBmp.CreateCompatibleBitmap(&dc, area.Width(), area.Height());
memDC.SetBkMode(TRANSPARENT);
CBitmap *oldBmp = memDC.SelectObject(&memBmp);
///
//绘图函数
///
dc.BitBlt(0, 0, area.Width(), area.Height(), &memDC, 0, 0, SRCCOPY);
memDC.SelectObject(oldBmp);
memDC.DeleteDC();
memBmp.DeleteObject();
CDialog::OnPaint();
```

仅仅是这样的话，在每次画面刷新时候都会把原先的画面擦除，还原为背景色，再把绘制好的画面显示出来。为消除这一现象，我们在每次画面刷新（在计时器中）的时候都调用 Invalidate(false)函数，让画面刷新时不把原有的画面擦除。

5.1.2 界面切换与过渡动画

为了更好地控制游戏所在的界面状态，我们在对话框类中定义了一个整型变量 state，用来描述游戏当前界面。在每次界面更新时，OnPaint()函数都将根据 state 的值，选择相应的绘制函数进行绘制。

```
//界面更新函数
void UpdateStart(CDC *pDC);                    //state = 1，开始界面
void UpdateHelp(CDC *pDC);                     //state = 2，帮助界面
void UpdateSettings(CDC *pDC);                 //state = 3，设置界面
```

```
void UpdateSelect(CDC *pDC);                    //state = 4，歌曲选择界面
void UpdatePlay(CDC *pDC);                      //state = 5，游戏界面
void UpdateResult(CDC *pDC);                    //state = 6，结算界面
```

为了实现更好的画面效果，我们在每个界面的切换时都加入了过渡动画。

为了控制动画，我们定义了一个结构体来描述动画的状态：

```
struct Animator
{
    int state;          //当前动画状态，0 表示无动画，其他分别对应一个动画
    long duration;      //动画所经过的时长
};
```

对话框类中有一个 Animator 类型的对象 animator。在 animator.state 不为零的状态下,OnPaint()函数根据 animator.state 的值将调用相应的动画绘制函数。根据动画的演算方式与要求，我们在每一个动画绘制的函数的参数列表中都添加了一个动画所需要的数据（如：进行按钮的缩放时需要确定是哪一个按钮，界面淡去时是从哪个界面淡去）：

```
//动画绘制函数
void FadeIn(int currentState, CDC *pDC);        //animator.state = 1
void FadeOut(int currentState, CDC *pDC);       //animator.state = 2
void ButtonExpand(int nID, CDC *pDC);           //animator.state = 3
void ButtonWithdraw(int nID, CDC *pDC);         //animator.state = 4
void SongChange(int change, CDC *pDC);          //animator.state = 5
void SceneChange(int change, CDC *pDC);         //animator.state = 6
```

每个动画结束时都要进行一些收尾工作，如：所有动画结束后都应将动画的时长（animator.duration）置为 0，播放相应界面的音频，根据动画结束后的界面的需求读取数据或释放内存空间，淡出动画结束后需要将动画的状态改为淡入，此外的动画结束后需要将动画的状态置为 0，等等。

5.1.3 用户系统数据处理

在初始化对话框时，程序将读取用户信息文件中的数据，并对这些数据做相应的处理：

```
CStdioFile usersFile("data\\users.txt", CFile::typeText | CFile::modeRead);
usersFile.ReadString(str);
usersNum = atoi(str);
userName = "";
if (usersNum > 0)
{
    usersInfo = new CString[usersNum][2];
    for (int i = 0; i < usersNum; i++)
    {
        usersFile.ReadString(str);
        usersInfo[i][0] = str.Left(str.Find('\t'));
        usersInfo[i][1] = str.Mid(str.Find('\t') + 1);
        Decode(usersInfo[i][1]);
```

```
        }
    }
```

注意到在读取密码时，程序调用了 Decode()函数，这个函数是用一种简单方法对用户数据进行加密的，稍后将详细说明。

登录时，程序将对所输入的用户名与密码与用户信息文件所存储的用户信息作比较，以对已注册的用户进行配对。

注册时，确认输入信息合法后，程序将把用户密码做加密处理（即调用 Encode()函数），防止外人能打开用户信息文件直接读取密码。

关于密码的加密处理，我们采用的方法是简单地对每一个字符进行或运算（^），做运算的数据除了字符外，另一个数据为从 1 至 7 共 7 个整数的循环。

```
void Encode(CString &str)
{
    int key[] = {1, 2, 3, 4, 5, 6, 7};
    int length = str.GetLength();
    char *ch = str.GetBuffer(0);
    for (int i = 0; i < length; i++)
    {
        ch[i] ^= key[i % 7];
    }
    str.ReleaseBuffer();
}
```

至于解密，只需将这一步骤重复一遍即可。

在注册或登录时，程序将动态创建一个整型数组（大小与歌曲数一致）用于存放用户存档，即每首歌曲的最高分。在歌曲选择界面，若有用户登录，则会在界面中显示对应歌曲的最高分。在结算界面，若玩家取得了更高的分数，在退出结算界面时将把这个分数更新至用户存档数组中。由于结算界面中每次绘制都会对玩家原高分数据与当前得分作比较以显示 NEW BEST 字样，所以我们不在结算界面中把数据直接更新至数组。

在退出游戏时，游戏将把用户存档放在与用户名同名的文本文件中，以供下次登录时读取。

```
if (userName != "")
{
    CString str;
    str.Format("data\\%s.txt", userName);
    CStdioFile userDataFile(str, CFile::modeCreate | CFile::modeWrite);
    for (int i = 0; i < songsNum; i++)
    {
        str.Format("%d\n", userData[i]);
        userDataFile.WriteString(str);
    }
}
```

5.1.4 键盘操作信息处理

为描述玩家操作，我们定义了一个结构体：

```
struct KeyInfo
{
    bool down;              //按键是否被按下
    long downTime;          //按键被按下的时间
    long upTime;            //按键被抬起的时间
    void Reset()
    {
        down = false;
        downTime = 0;
        upTime = 0;
    }
};
```

在对话框类中，我们定义了该结构体的一个数组 keys[8]，分别对应 8 个按键的信息。

在游戏界面中，玩家在对轨道所对应的按键进行操作时，相应的消息处理函数将对按键信息进行更新：

```
void CMelodiaDlg::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
    if (state != 5 || animator.state != 0)
        return;
    int keyNum = -1;
    switch (nChar)
    {
    case 'A':
        keyNum = 0;
        break;
    case 'S':
        keyNum = 1;
        break;
    case 'D':
        keyNum = 2;
        break;
    case 'F':
        keyNum = 3;
        break;
    case 'J':
        keyNum = 4;
        break;
    case 'K':
        keyNum = 5;
        break;
    case 'L':
        keyNum = 6;
        break;
    case 186:
```

```
            keyNum = 7;
            break;
        }
        if (keyNum != -1 && !keys[keyNum].down)
        {
            keys[keyNum].down = true;
            keys[keyNum].downTime = GetTickCount() - startTime;
        }
    }
```

按键抬起的消息处理类似。

在游戏界面中，音符将根据键位的按下时间与抬起时间进行判定，无论判定是否生效，都会在该音符的处理过程中将键位的按下时间或抬起时间置为 0，以防止后续的音符也对该按键进行判定。

5.1.5 音符绘制

为了保证音符在视觉效果上匀速下落，且在音符的对应时间恰好落在判定线上，我们对音符的位置计算使用以下的表达式：

$$y = l - \frac{(\Delta t + h)v}{10}$$

其中 y 为音符 y 坐标，l 为判定线 y 坐标，Δt 为音符对应时间与当前时间之差，h 为音符持续时间，v 为下落速度。

一般情况下音符的高度则为：

$$H = 20 + \frac{hv}{10}$$

若音符为长按音符且已被按住，则音符高度为：

$$H = 15 + \frac{\Delta t + hv}{10}$$

这样便可明确音符在 y 方向所要绘制的位置与高度。x 方向的处理较为简单，此处省略。

5.1.6 判定规则

音符的判定共有四种类型，分别为 Perfect（-60ms～+60ms，金色光效），Good（-200ms～+150ms，绿色光效），Bad（-400ms～+300ms，蓝色光效），Miss（<-400ms，即错过音符，无光效）。其中 Perfect 与 Good 计入连击，Bad 与 Miss 不计入连击。

长按音符在按下与抬起时均会进行判定，但分数至记为一个音符的分数，音符结束时的光效即代表最终判定。具体判定方法如下：

按下 \ 抬起	Perfect	Good	Bad	Miss
Perfect	Perfect	Perfect	Good	Miss

按下 \ 抬起	Perfect	Good	Bad	Miss
Good	Perfect	Good	Good	Miss
Bad	Good	Good	Bad	Miss
Miss	Miss	Miss	Miss	Miss

5.1.7 计分规则

每首歌曲总分为1000000分，其中90%为判定分，10%为连击分。

每个音符均分判定分，音符判定为Perfect时获得100%的分数，音符判定为Good时获得70%的分数，音符判定为Bad时获得30%的分数，音符判定为Miss为Miss时不获得分数。

设共有n个音符，则将连击分分为 $\frac{(1+n)n}{2}$ 份（即从1累加到n），每次击打音符时，若判定被计入连击，则加上与连击数等份数的连击分。这样计算的目的是实现连击越高，得分越高的效果，同时相同的连击可以得到相同的分数。

根据总分的大小，共有 6 种评价：M（1000000），S（950000～999999），A（900000～949999），B（800000～899999），C（700000～799999），F（<700000）。

5.2 谱面编辑器MelodiaEditor算法思想简介

5.2.1 文档数据与系列化方法

在文档类中，我们定义了以下共有变量与方法：

```
CString m_name;              //工程名称
CString m_composer;          //作曲者
int m_bpm;                   //歌曲拍速
int m_notesNum;              //音符总数
int m_startPos;              //起始时间
int m_curPos;                //当前时间
int m_selection;             //当前选择音符
CString m_bg;                //背景图片路径
CString m_cover;             //封面图片路径
CString m_preview;           //预览音乐路径
CString m_full;              //全长音乐路径
CNoteInfo * GetNoteInfo(int nIndex);              //获取对应编号的音符信息
void AddNote(int nStartIndex, CNoteInfo *note); //在指定位置添加一个音符
bool DeleteNote(int nIndex);                      //删除指定的音符
```

以及一个受保护的数组：

```
CTypedPtrArray<CObArray, CNoteInfo*> m_noteArray;
```

CNoteInfo为一类，其定义为：

```
class CNoteInfo : public CObject
```

```
{
public:
    int bar;                //音符所在小节
    int numerator;          //音符所在节拍的分子
    int denominator;        //音符所在节拍的分母
    int holdNumerator;      //音符持续时间的分子
    int holdDenominator;    //音符持续时间的分母
    int track;              //音符所在轨道
    CNoteInfo();
    virtual ~CNoteInfo();
};
```

记录节拍与持续时间的分数而不是具体的时间与持续时间，是为了在修改 BPM 与起始时间后，音符的时间与持续时间也能做出相应的更改而保证音符所在节拍与持续节拍不变。

在系列化时，若存储文件，则先将工程信息中的所有数据（工程名称、作曲者、拍速、音符总数、起始时间、相关资源路径）保存，再通过一个 for 循环将所有的音符信息存储。在读取文件时，程序所做的操作类似。

```
void CMelodiaEditorDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << m_name << m_composer << m_bpm << m_notesNum << m_startPos <<
            m_bg << m_cover << m_preview << m_full;
        for (int i = 0; i < m_notesNum; i++)
        {
            CNoteInfo *note = m_noteArray.GetAt(i);
            ar << note->bar << note->numerator << note->denominator <<
                note->holdNumerator << note->holdDenominator << note->track;
        }
    }
    else
    {
        // TODO: add loading code here
        ar >> m_name >> m_composer >> m_bpm >> m_notesNum >> m_startPos >>
            m_bg >> m_cover >> m_preview >> m_full;
        for (int i = 0; i < m_notesNum; i++)
        {
            CNoteInfo *note = new CNoteInfo;
            ar >> note->bar >> note->numerator >> note->denominator >>
                note->holdNumerator >> note->holdDenominator >> note->track;
            m_noteArray.Add(note);
        }
    }
}
```

5.2.2 各个视图的数据处理

各个视图的数据皆通过文档类的数据进行交换，以确保各个视图中的数据一致。一个视图的数据更新时会通过文档类指针调用 UpdateAllViews()函数，确保各个视图中的数据都得到及时更新。

5.2.3 导出歌曲

将歌曲导出时，需要检验工程信息是否完善、选择游戏目录、创建歌曲资源文件夹、复制歌曲资源、修改游戏歌曲数据中的歌曲数并添加新歌曲的曲名与作者、将音符信息转换成游戏所能读取的数据并创建谱面数据文本文档。具体操作如下：

```
void CMainFrame::OnFileExport()
{
    // TODO: Add your command handler code here
    CMelodiaEditorDoc* pDoc = (CMelodiaEditorDoc *)GetActiveDocument();
    ASSERT_VALID(pDoc);
    if (pDoc->m_name == "" || pDoc->m_bg == "" || pDoc->m_cover == "" ||
        pDoc->m_preview == "" || pDoc->m_full == "")
    {
        AfxMessageBox("请完善工程信息！");
        return;
    }

    char path[256];
    CString str;
    BROWSEINFO bi;

    ZeroMemory(path, sizeof(path));
    bi.hwndOwner = m_hWnd;
    bi.pidlRoot = NULL;
    bi.pszDisplayName = path;
    bi.lpszTitle = "请选择导出目录：(Melodia游戏目录)";
    bi.ulFlags = 0;
    bi.lpfn = NULL;
    bi.lParam = 0;
    bi.iImage = 0;

    LPITEMIDLIST lp = SHBrowseForFolder(&bi);
    if(lp && SHGetPathFromIDList(lp, path))
    {
        pDoc->m_name.Replace(' ', '_');
        str.Format("%s\\songs\\%s", path, pDoc->m_name);
        CreateDirectory(str, NULL);
        str.Format("%s\\songs\\%s\\bg.bmp", path, pDoc->m_name);
        CopyFile(pDoc->m_bg, str, false);
        str.Format("%s\\songs\\%s\\cover.bmp", path, pDoc->m_name);
        CopyFile(pDoc->m_cover, str, false);
        str.Format("%s\\songs\\%s\\preview.mp3", path, pDoc->m_name);
        CopyFile(pDoc->m_preview, str, false);
        str.Format("%s\\songs\\%s\\full.mp3", path, pDoc->m_name);
```

```
        CopyFile(pDoc->m_full, str, false);

        str.Format("%s\\data\\songs.txt", path);
        CStdioFile songs(str, CFile::modeRead | CFile::typeText);
        songs.ReadString(str);
        int songsNum = atoi(str);
        CString *songsName = new CString[songsNum];
        for (int i = 0; i < songsNum; i++)
        {
            songs.ReadString(str);
            songsName[i] = str;
        }
        songs.Close();
        str.Format("%s\\data\\songs.txt", path);
        songs.Open(str, CFile::modeCreate | CFile::modeWrite);
        str.Format("%d\n", songsNum + 1);
        songs.WriteString(str);
        for (i = 0; i < songsNum; i++)
        {
            songs.WriteString(songsName[i]);
            songs.WriteString("\n");
        }
        str.Format("%s\t%s\n", pDoc->m_name, pDoc->m_composer);
        songs.WriteString(str);
        songs.Close();

        str.Format("%s\\songs\\%s\\score.txt", path, pDoc->m_name);
        CStdioFile score(str, CFile::modeCreate | CFile::modeWrite);
        str.Format("%d\n", pDoc->m_notesNum);
        score.WriteString(str);
        for (i = 0; i < pDoc->m_notesNum; i++)
        {
            CNoteInfo *note = pDoc->GetNoteInfo(i);
            int noteTime = (int)(240000.0 / pDoc->m_bpm * (note->bar - 1 +
                (double)note->numerator / note->denominator) +
                pDoc->m_startPos);
            int noteHold = 0;
            if (note->holdDenominator != 0)
                noteHold = (int)(240000.0 / pDoc->m_bpm *
                    note->holdNumerator / note->holdDenominator);
            str.Format("%d\t%d\t%d\n", noteTime, note->track, noteHold);
            score.WriteString(str);
        }
        delete []songsName;
        AfxMessageBox("导出完成！");
    }
}
```

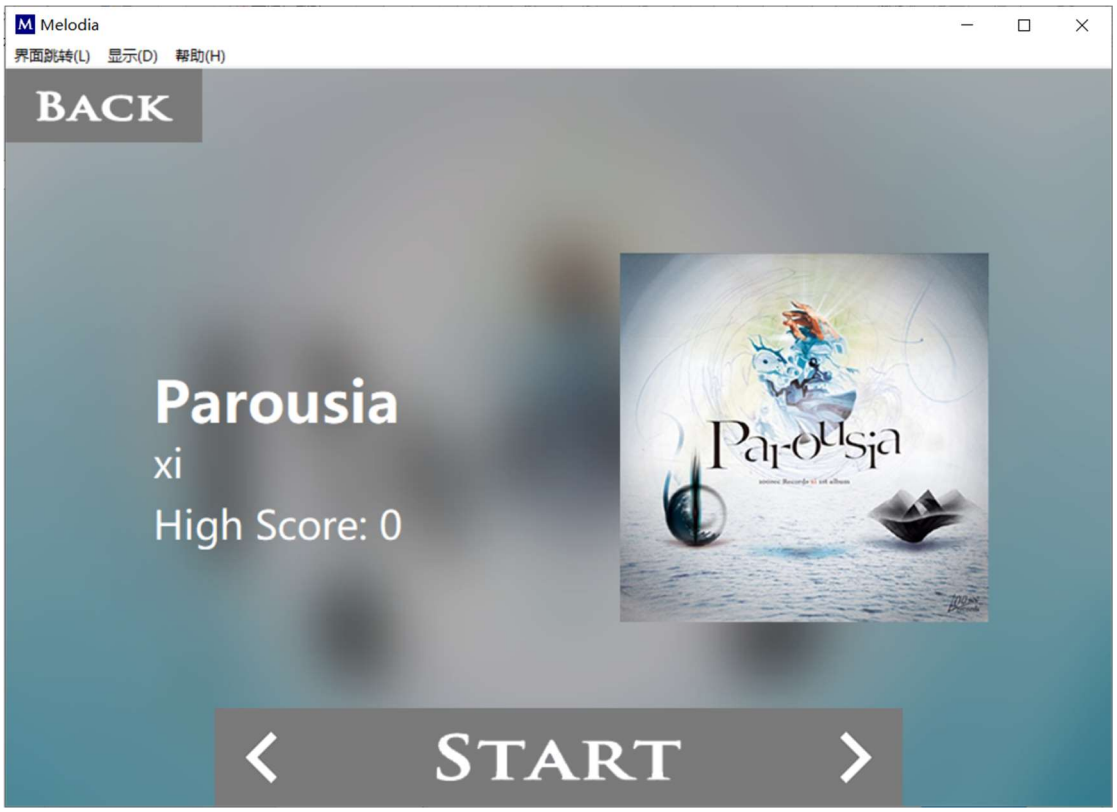
6、可提供的测试数据

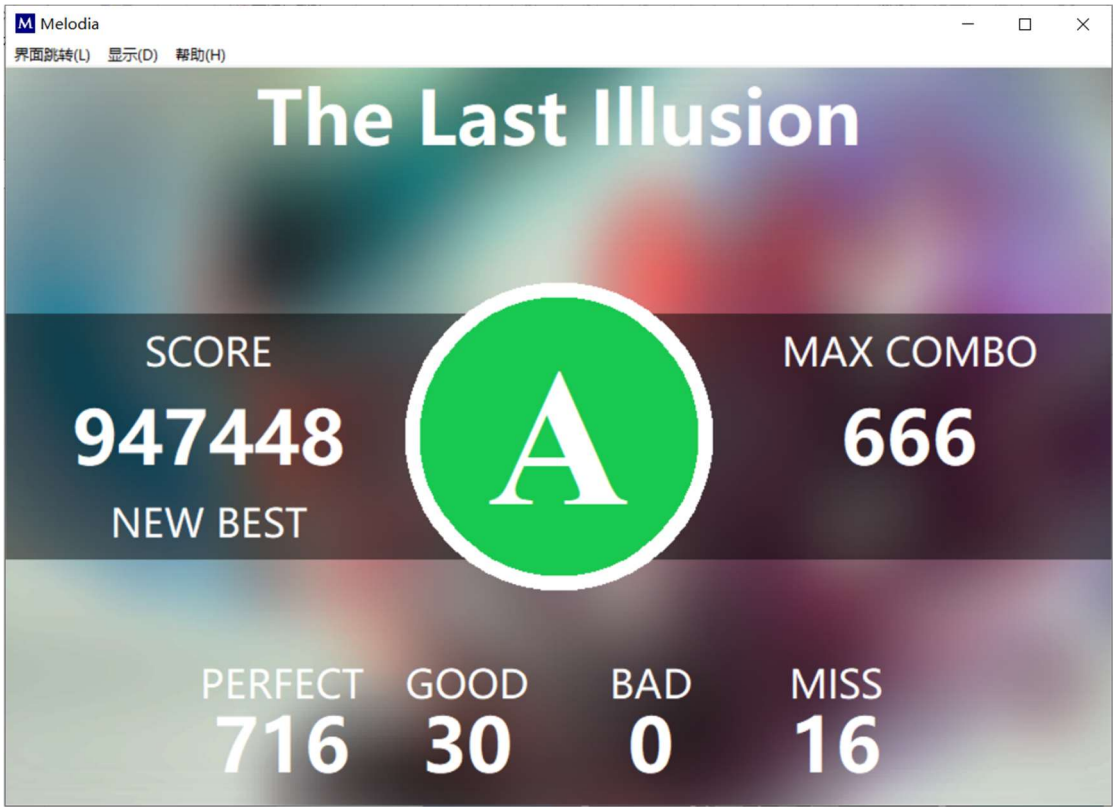

Melodia
界面跳转(L) 显示(D) 帮助(H)
开始游戏
游戏帮助
欢迎回来，SEU

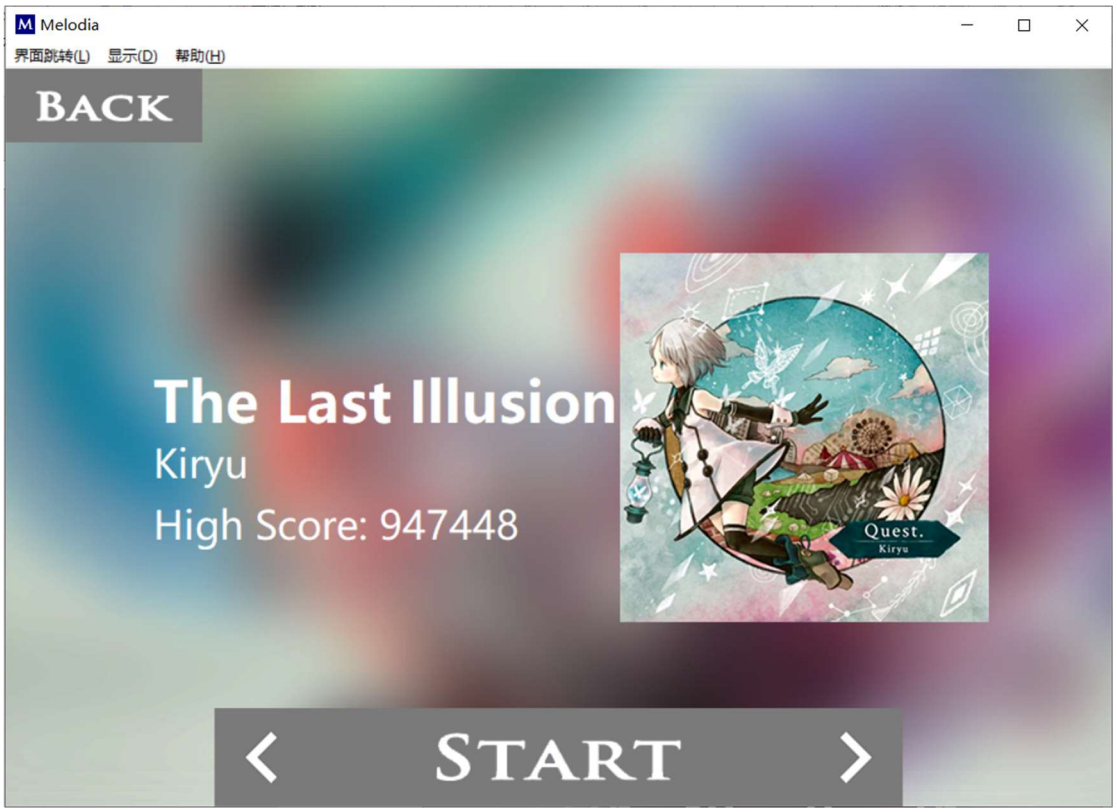

Melodia
界面跳转(L) 显示(D) 帮助(H)
游戏帮助
欢迎回来，SEU


Melodia
界面跳转(L) 显示(D) 帮助(H)
游戏帮助
A S D F J K L ;
分别对应8个轨道
游戏中与结算界面
Esc 返回至歌曲选择界面
Enter 重新开始
单击
长按
(带抬起判定)

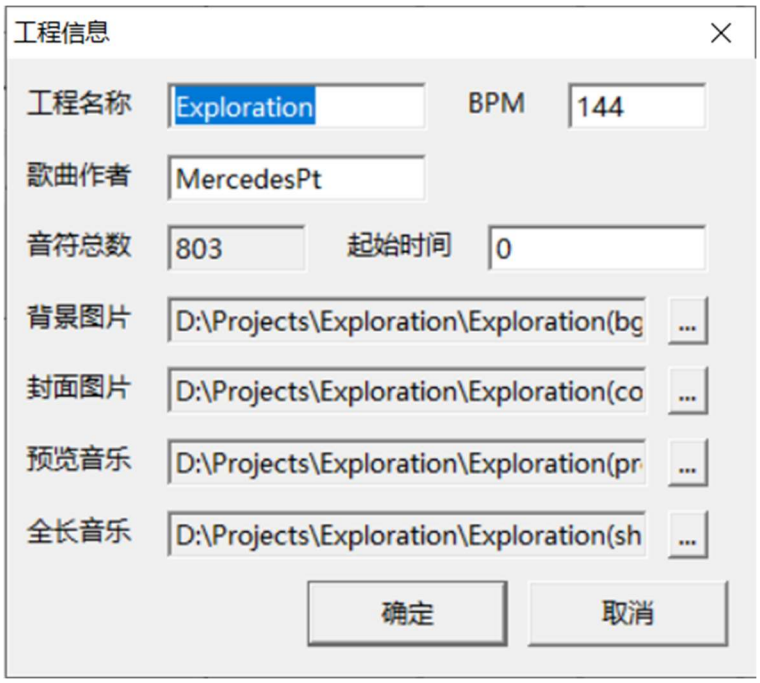

Melodia
界面跳转(L) 显示(D) 帮助(H)
下落速度 - 6 +
击打音 ON

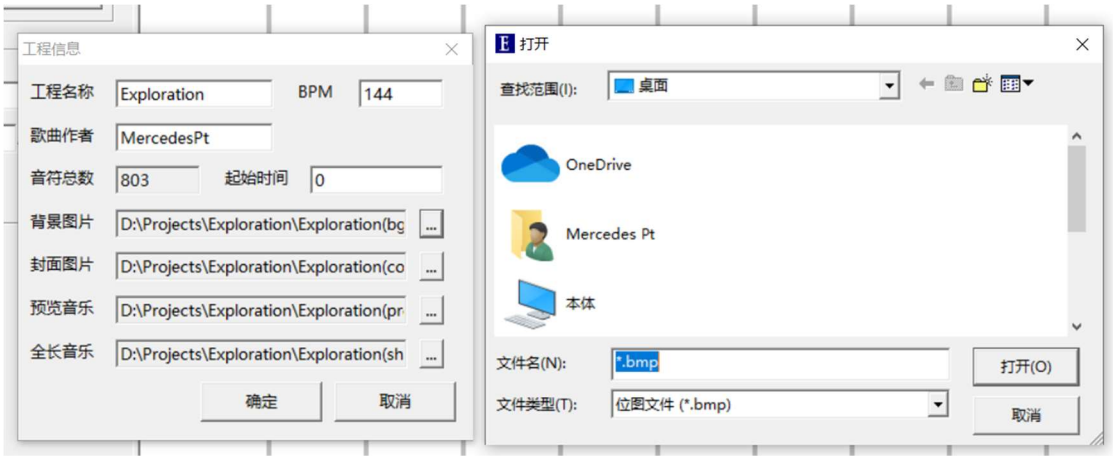

Melodia
界面跳转(L) 显示(D) 帮助(H)
BACK
Parousia
xi
High Score: 0
START

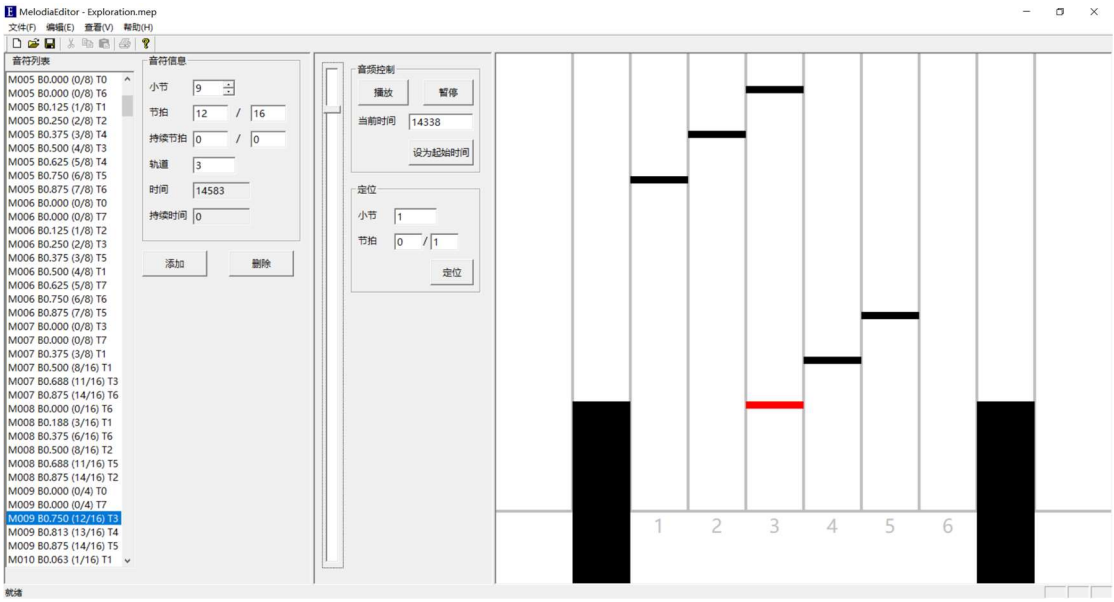

Melodia
界面跳转(L) 显示(D) 帮助(H)
The Last Illusion
SCORE
947448
NEW BEST
A
MAX COMBO
666
PERFECT
716
GOOD
30
BAD
0
MISS
16


Melodia
界面跳转(L) 显示(D) 帮助(H)
BACK
The Last Illusion
Kiryu
High Score: 947448
Quest.
Kiryu
START


users.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(
1
SEU RGV657>
SEU.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
0
947448
0


工程信息
工程名称
Exploration
BPM
144
歌曲作者
MercedesPt
音符总数
803
起始时间
0
背景图片
D:\Projects\Exploration\Exploration(bg
封面图片
D:\Projects\Exploration\Exploration(co
预览音乐
D:\Projects\Exploration\Exploration(pr
全长音乐
D:\Projects\Exploration\Exploration(sh
确定
取消


工程信息
工程名称
Exploration
BPM
144
歌曲作者
MercedesPt
音符总数
803
起始时间
0
背景图片
封面图片
预览音乐
全长音乐
确定
取消
打开
查找范围(I):
桌面
OneDrive
Mercedes Pt
本体
文件名(N):
*.bmp
打开(O)
文件类型(T):
位图文件 (*.bmp)
取消


MelodiaEditor - Exploration.mep
文件(F) 编辑(E) 查看(V) 帮助(H)
音符列表
音符信息
小节
节拍
持续节拍
轨道
时间
持续时间
添加
删除
音频控制
播放
暂停
当前时间
设为起始时间
定位
小节
节拍
定位
1
2
3
4
5
6
就绪

附录 1 使用说明

主游戏 Melodia 的操作参见游戏帮助界面。游戏的窍门是要仔细聆听音乐，根据韵律与节奏击打音符，同时要熟悉各个键位与轨道的对应关系。

谱面编辑器 MelodiaEditor 的操作：

打开程序或者新建文件时，程序将弹出工程信息对话框，在可写的编辑框中输入相应信息，点击资源路径后的“…”按钮可弹出打开文件对话框以选择对应的文件。工程信息对话框也可在“查看->工程信息”中打开。

主程序界面中，在音符信息视图中编辑好音符信息后可按“添加”按钮将该音符信息添加至谱面中。在左侧的音符列表中将列出所有的音符，单击一项后将在音符信息视图中显示该音符的所有信息。选中一个音符后可按“删除”按钮将该音符从音符列表中移除。

若在工程信息中“全长音频”一项为有效资源路径，可在音频控制视图进行相应操作。拖动滑动条可改变当前时间，也可直接在当前时间编辑框中输入数值，或者在定位栏中输入小节与节拍信息，程序将根据拍速与起始时间定位对应的音频时间。

正在编辑的谱面可通过保存操作以 mep 格式（MelodiaEditor Project）保存在磁盘中以供下次打开时读取。完成的谱面可点击“文件->导出”选项，选择游戏所在文件夹，将该歌曲添加至游戏中。添加至游戏的歌曲可同其他歌曲一样保存用户存档信息。

附录 2 参考文献

序号	文献类型	
1	学术著作	1　马十安，魏文平．Visual C++程序设计与应用教程[J]．第二版 北京：清华大学出版社，2011， p3-150