



同濟大學

课程作业实验报告



题目 《贪吃蛇大作战》大作业实验报告

课 程 面向对象的程序设计

姓 名 蔡宇轩

学 号 2252429

学 院 电子与信息工程学院

专 业 计算机科学与技术22级1班

教 师 陈宇飞

二〇二四 年 一 月 七 日

1 设计思路

1.1 功能描述

本程序完成了一个简易的贪吃蛇小游戏的制作与实现，主要包括如下几个功能模块：

(1) 主页面：

实现单人模式下小蛇皮肤的自主选择，游戏模式的自主选择，以及各个功能页面的跳转。



图 1.1 主菜单界面示意图

(2) 单人模式：

单机模式，分为[1]入门版、[2]进阶版、[3]高级版三个游戏难度。在本页面可以显示本轮游戏的相关信息，同时可以实现游戏暂停和 AI 挂机模式。



图 1.2 单人模式界面示意图

(3) 人机对战模式：

人机对战，红色小蛇为玩家，蓝色小蛇为

AI 蛇，同样具备[1]入门版、[2]进阶版、[3]高级版三个游戏难度。游戏区上方显示实时的能量槽与比分，保留了游戏暂停功能。

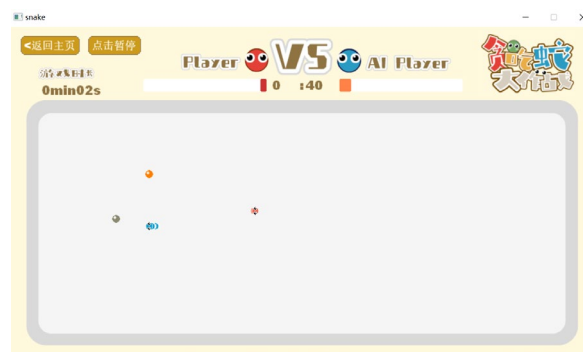


图 1.3 人机对战模式界面示意图

(4) 双人模式：

实现 2 位玩家同时游戏的功能，也分为 3 种不同的游戏难度，界面 UI 与人机对战模式相同。



图 1.4 双人模式界面示意图

(5) 排行榜界面：

查看不同游戏模式下的历史记录前 3 名，包含本次游戏加载以前的排名。



图 1.5 排行榜界面示意图

1.2 总体设计思路

由于本题要求使用类封装的思想，因此我们在设计时将游戏中所有具有相同属性的对象都封装为一个类。

在游戏主要功能的实现上，我封装了包括：蛇(SNAKE)模块、食物(FOOD)模块、障碍 墙壁 (WALL) 模块、综合管理模块 (MANAGER)共 4 个类。

在 GUI 界面的设计上，我封装了包括：页面(PAGE)模块和按钮(BUTTON)模块共 2 个类。

每个类的具体说明及其功能描述，详见下一节。

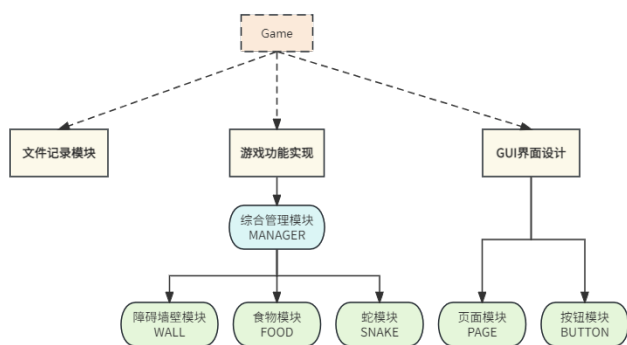


图 1.6 本程序中各个类的关系示意图

2 类的说明及功能描述

2.1 蛇 SNAKE

2.1.1 成员声明

```

class SNAKE {
private:
    void snake_invalid(int index);
public:
    Level level;
    //当前小蛇所属的游戏难度，具体的功能设定
    MODE mode;
    //决定页面上是一条蛇还是两条蛇

```

```

    int color[2];
    //小蛇的皮肤颜色
    int length[2];
    //蛇的长度
    enum DIR direction[2]; //蛇的方向
    int radio; //蛇身的半径
    int speed[2]; //蛇的前进速度
    vector<vector<pos>>> snake_pos;
    //每一节蛇身的坐标
    int score[2];
    //这条蛇当前得到的总分
    int life[2];
    //生命值
    int crashtimes[2];
    //撞墙次数
    bool cgtoAI;
    //是否切换 AI 模式

public:
    SNAKE(); //构造函数
    void snake_eyeRender(int x,int y);
    //蛇的眼睛绘制
    void snake_bodyRender(int index);
    //蛇的身体绘制
    void set_skincolor(int index,int color);
    //重置皮肤颜色
    void snake_move(int index);
    //蛇的移动
    void snake_event(int inedx,BYTE bkcode);
    //蛇的事件处理
    void snake_increase(int index,int
increase);
    //蛇的长度增加
    void snake_decrease(int index,int
decrease);
    //蛇的长度缩短
    void way_to_exit();
    //不同游戏模式下的退出方式
    void snake_reset(int index);
    //重置蛇的各项属性
    bool snake_checklife(int index);
    //检查蛇的存活状态
    void snake_easyAI(int index,pos target);
    //单条蛇的 AI 模式算法
};

```

2.1.2 重要成员函数的实现

本节选取蛇 SNAKE 在游戏过程中的几个重要功能的实现作具体阐述。

(1) 蛇的存储方式:

由于蛇的长度可能无限制增加，因此考虑使用容量可以动态增减的数据结构存储每一节蛇身的坐标及相应的颜色:

```
typedef struct pos{
    int x;//坐标的X位置
    int y;//坐标的Y位置
    COLORREF color;//该身体坐标对应的颜色
}pos;
```

蛇的每一节身体都对应一个这样的坐标，再考虑到容量无限制的特性，我采用了一个 pos 的 vector 用来存储蛇身上每一节的位置坐标，定义为 `vector<pos>snake_pos`，相关的对蛇身体增减的操作，对应 vector 的操作如下：

```
//初始化容量
snake_pos.resize();
//向容器中添加新的坐标，用于蛇的长度增加
snake_pos.push_back();
//将容器中的坐标去除，用于蛇的长度减少
snake_pos.pop_back();
//清空容器，用于游戏结束时重置蛇的状态
snake_pos.clear();
```

(2) 蛇的移动：

移动模块的基本思想即每一次蛇的第 i 块的坐标为上一次移动时第 $i - 1$ 块的坐标。因此当没有按下方向键时，只需要利用循环语句完成各元素的传递赋值即可。

```
//后一个的位置是前一个在上次的位置
for (int i = this->length[index] - 1;
i > 0; i--) {
    snake_pos[index][i].x
    = snake_pos[index][i - 1].x;
    snake_pos[index][i].y
    = snake_pos[index][i - 1].y;
}
```

若发生方向键的移动，则还需根据方向改变石佛合法来改变蛇的方向。

(3) 蛇的事件处理：

蛇的事件处理即通过键盘事件来判断并改变蛇头的方向。单机模式只需要处理上下左右四个方向键的键值；双人模式需再额外

处理 WSAD 四个键的键值。

(4) AI 蛇的算法：

成员函数 `snake_easyAI` 可以控制一条小蛇在 AI 模式下的移动。具体实现为通过比较目标食物坐标与蛇头坐标，来改变蛇头的方向。需要注意的是，如果出现了蛇头需要反向掉头的情况，则需要顺时针（或逆时针）改变蛇头的转向，而不能直接 180° 改变方向；简而言之，每次蛇头只能发生 90° 的方向转变，否则会导致错误。

2.1.3 完整操作流程

蛇的每一次移动，都需先后经过：事件判断、移动、渲染等主要步骤。完整的一次流程如下图所示：

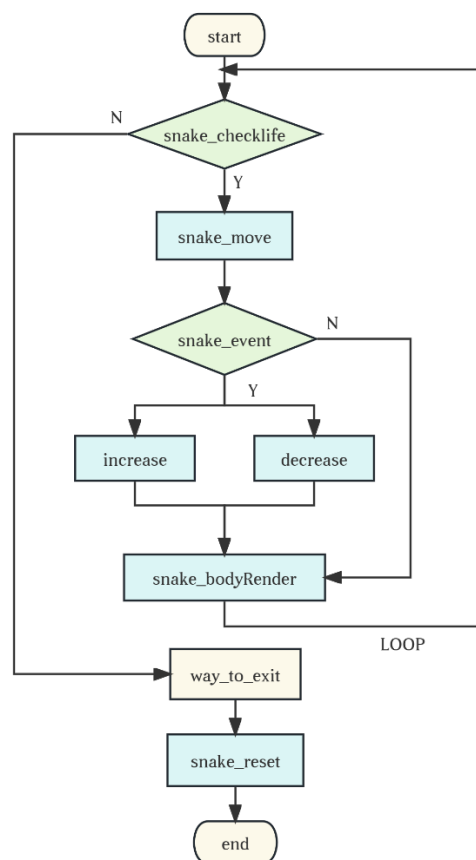


图 1.7 一次完整的 SNAKE 各项操作流程图

2.2 食物 FOOD

2.2.1 成员声明

```
class FOOD {
private:
    bool good_bad[50];
    //设置是好果子还是坏果子
    int score[50];
    //每个果子对应得到的分数
    bool alive[50];
    //果子是否被吃掉
public:
    COLORREF color[50];
    //好果子-彩色, 坏果子-灰色
    int sum;
    //每一次的果子总数
    int radio;
    //绘制半径
    pos food_pos[50];
    //食物的坐标
    int left;
    //剩余的还没被吃掉的好果子
public:
    FOOD() {};//默认构造函数
    void food_set(SNAKE&
snake_group,WALL&wall);
    //设置所有果子的属性
    //构造函数调用, 重置函数就是 set 本身)
    void food_render();//绘制渲染
    bool food_getAlive(int index);
    //判断食物存活状态
    void food_setAlive(int index,bool
setAlive);//设置食物存活状态
    int food_getScore(int index);
    //获取食物对应分数
    bool food_getQuality(int index);
    //获取食物好坏性质
    bool food_checkpos(int food_index);
    //检查当前生成的食物是否与已有食物重合
    void
food_Cgbysnake(vector<pos>&snake_pos);
    //在撞墙后的蛇的尾部生成随机食物
};
```

2.2.2 重要成员函数的实现

(1) 食物的生成:

食物生成需要遍历新生成的位置是否与已经生成的食物、当前蛇的各节坐标、所有障碍墙壁的坐标重合, 若重合则重新生成直至不重合为止。

(2) 高级模式下蛇身变食物:

高级版的游戏难度下, 在蛇的撞墙之后需要将蛇的尸身转换为食物, 这就需要新生成的食物与蛇在撞墙前的每一节身体坐标一一对应, 其余的生成食物的方式与(1)中食物的生成相同。

2.3 障碍墙壁 WALL

2.3.1 成员声明

```
//障碍物类
class WALL {
private:
    int wall_num;
    //墙壁总数
    int wall_radio;
    //绘制墙壁半径
    vector<bool>wall_soft_hard;
    //软墙壁和硬墙壁的标志
    vector<COLORREF>wall_color;
    //软墙壁和硬墙壁的颜色
public:
    vector<pos>wall_pos;
    //除了边框以外, 所有障碍物的位置
public:
    WALL();
    //构造函数
    void
wall_build(vector<pos>dead_snake_pos);
    //把死掉的蛇蛇变成墙壁
    void wall_render();
    //绘制所有墙壁
    bool wall_getQuality(int wall_index);
    //获取当前墙壁的属性
    bool wall_checkpos(int food_x,int food_y);
    //检查新生成的食物是否出现在了墙壁上
    void wall_reset();
    //重置墙壁属性
};
```

2.3.2 重要成员函数的实现

(1) 墙壁的生成:

墙壁的生成时机为当小蛇在进阶模式下撞墙后, 蛇的身体会变成障碍物墙壁。设计

将墙壁分为软墙壁和硬墙壁两种属性，在撞墙时随机生成。由于每次撞墙时蛇身体的长度未知，因此考虑使用 `vector` 作为形参。

在生成墙壁时，只需要遍历当前小蛇身体每一节所在的位置坐标 `pos` 并将其 `push` 进表示 `WALL` 的坐标容器 `vector` 即可。

对于墙壁的软硬属性，每次可以通过生成随机值来判断，例如本次程序中实现的就是 `rand()` 得到的数字为奇数时是 `soft` 软墙壁，偶数为 `hard` 硬墙壁。同时，由于不同位置墙壁的软硬属性不同，我们需要一个与 `wall_pos` 一一对应的容器 `wall_soft_hard` 来记录每一块墙壁的软硬属性。

2.4 综合管理模块 MANAGER

由于游戏中的所有蛇都被 `SNAKE` 类管理，所有食物都被 `FOOD` 管理，所有障碍墙壁都被 `WALL` 类管理，因此我们需要设计一个它们上层的 `class` 来作为管理这 3 个类的综合交互模块。`MANAGER` 类起到这样的作用。

2.4.1 成员声明

```
//综合管理模块
class MANAGER {
private:
    int time;//每次游戏的计时起点
public:
    SNAKE snake;//管理页面中的所有蛇
    FOOD food;//管理页面中的所有食物
    WALL wall;//管理页面中的所有障碍物
    wchar_t user1[15];//用于输出显示：第一条蛇的用户名
    wchar_t user2[15];//用于输出显示：第二条蛇的用户名
    int winner;//存放每一局的游戏赢家
    bool checkRank;
private:
    void manager_draw_info();//游戏进行中的相关信息
    void manager_readranking();//读取文件中存储的排行榜信息
public:
    MANAGER();//构造函数
    void manager_update();//更新状态
```

```
void manager_ai();//单机游戏下的简易 AI 模式
void manager_render();//渲染
void manager_event(BYTE& vkcode);//事件管理
void manager_eatfood();//吃食物的相关操作
void manager_reset_time();//游戏时间重置
void manager_crashwall();//撞墙的相关操作
bool manager_isRunning();//检查当局游戏是否还要继续
void manager_multi_setwinner(int index);//双人模式下设定胜者
void manager_renderRanking();//打印排行榜
};
```

2.4.2 重要成员函数的实现

`Manager` 应该至少包括对当前游戏状态的判断、对画面实时渲染绘制、关键事件处理的功能，其主要实现的功能框图如下：

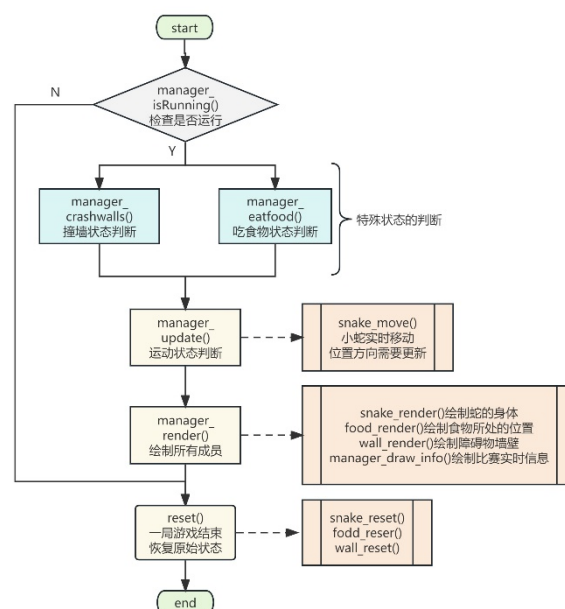


图 2.1 Manager 管理游戏进程的流程图

(1) 游戏特殊状态的判断:

撞墙和吃到食物是游戏中的两个特殊状态，其具体实现为坐标位置的遍历。每一时刻判断蛇头坐标是否与画面中的所有食物（或所有障碍物墙壁）的坐标重合，若重合则说明发生了上述特殊事件，需做相应的处理。

(2) 普通状态的判断:

普通游戏状态的判断包括小蛇的移动，而小蛇的移动控制源于 `start` 之前的 `snake_event()` 判断(上方流程图中并未展示)。该函数的主要功能即通过 `easyX` 内置的消息工具不断获取鼠标和键盘信息，来对 `SNAKE` 类和 `BUTTON` 类的成员函数调用作相应的控制。然后，这些控制信号改变 `SNAKE` 和 `BUTTON` 的状态，并在后续的循环体中显示出来。

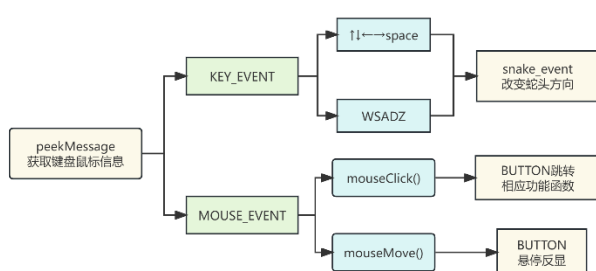


图 2.2 Manager_event 管理的事件

(3) 画面渲染:

画面渲染包括：①所有 `BUTTON` 的绘制②所有小蛇的绘制③所有食物的绘制④所有障碍物墙壁的绘制⑤游戏实时信息栏的绘制共 5 个板块的内容，分别由对应的类成员函数实现，示意图如下：

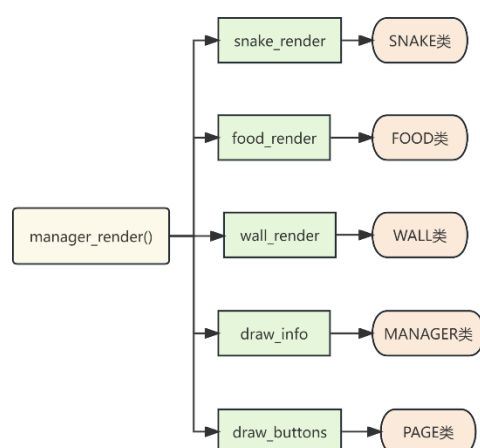


图 2.3 Manager_render 管理的画面渲染模块

(4) 游戏参数重置:

在遇到小蛇撞墙、食物吃光或者游戏结束的事件以后，要相应地依据情况对 `manager` 中的三个对象 `snake, food, wall` 的全部参数作重置，主要包括：蛇的坐标和长度恢复到初始位置，`food` 重新随机生成以及所有障碍物清空的操作。

2.5 文件记录模块

2.5.1 模块功能

本程序需要记录每次游戏的历史记录，以及排行榜模块下每种游戏模式的前 3 名。这需要文件记录与读取的相关操作，其具体实现封装在 `Record.cpp` 和 `Record.h` 中。

2.5.2 重要函数的实现

(1) 全部历史记录:

在每次游戏结束后将本次游戏的用户名、游戏模式、得分写入 `record.txt` 作为历史记录。

(2) 排行榜:

排行榜的实现需要每次游戏结束后都对当前模式下的所有比分进行排序，由于我们设计只展示每个模块下的前三名，并且不同的游戏难度之间不作区分，因此我们只需要分别开设 3 个容量为 3 的数组，对应单人模式、双人模式、人机对战模式下的各项分数、用户名和游戏难度即可。

每次游戏结束后 `update_rank()` 函数都比较本轮得分是否具备进入当前模式下前三名

的资格，若具备则更新数组；若不具备则不再作任何操作。

排行榜更新的机制为，人机模式和双人模式的 2 名玩家都上榜（双人模式下 Player1 默认优先于 Player2，人机模式下 Player 默认优先于 AI-Player）；当出现分数相等的情况时，新上榜的分数优先于原来的分数。

3 UI 设计及相关类的封装

由于本程序涉及多个页面间的跳转，且每个页面上都有不同的按钮并对应执行相应的跳转和功能。虽然每个页面的布局与按钮的功能不同，但是其运行机制应该是完全相同的，因此我考虑分别定义 BUTTON 和 PAGE 两个类来泛化管理所有页面和相关的按钮功能。由此将所有页面的运行都用同一段代码管理，更具简洁性也更好管理。

与 UI 界面相关的内容全部封装在 MyGUI.cpp 和 MyGUI.h 中。

3.1 按钮类 BUTTON

3.1.1 按钮的生成

考虑将添加按钮这一操作进行封装，对外留下：①按钮在画布中的 (x,y) 位置、②按钮上的文字提示信息、③按下按钮后所执行功能的函数指针共 3 个接口，如此在每个页面上添加按钮信息就只需要调用此函数即可，可以使代码具备良好的封装性。

我们将每个页面的所有按钮存储在 BUTTON 类的一个 vector 里，从而管理一个页面中的所有 BUTTON。

3.1.2 按钮功能的实现

BUTTON 类需要实现的主要功能有：

- ① 鼠标悬停在上方时按钮反显
- ② 鼠标点击按钮内部执行相关功能



图 3.1 鼠标悬停时按钮的反显效果

为了在视觉上实现按钮类的“点击”反显效果，我们需要再 BUTTON 里定义如下数据成员：

```
private:
    pair<int, int>pos; //按钮位置坐标
    wstring text; //按钮文本
    pair<int, int>size; //按钮大小
    float scale; //缩放比例，用于展示鼠标悬停效果
    bool isMouseOver; //鼠标是否处于按钮上方
    void(*onClick)(); //鼠标点击时执行的函数指针
```

(1) 检查鼠标悬停：

isMouseOver 是用来判断鼠标是否位于按钮上方的 bool 型变量。其具体判别方式即判断当前 ExMessage 传入的鼠标坐标是否在按钮坐标范围以内。一旦位置成立，则改变当前这个 BUTTON 的缩放属性 scale：

```
if (isMouseOver) //鼠标悬停在上方时尺寸缩小
    scale = 0.9f;
else //不悬停时保持正常尺寸
    scale = 1.0f;
```

图 3.2 鼠标悬停时改变按钮尺寸

(2) 检查鼠标点击：

鼠标点击发生在判断鼠标悬停之后，一

且鼠标左键单击发生在 **BUTTON** 内部，就应跳转执行本 **BUTTON** 的跳转指针成员 **onClick** 所指向的函数。执行完毕后，各项属性恢复到悬停之前：

```
isMouseOver = true;
onClick(); // 执行对应功能指针所指向的函数
// BUTTON 的各项属性恢复到悬停前
isMouseOver = false;
scale = 1.0f;
```

图 3.3 鼠标点击后执行的相关操作

3.2 页面类 PAGE

页面类的数据成员定义如下：

```
private:
    int page_width; // 页面宽度
    int page_height; // 页面高度
    IMAGE background; // 背景图
    Level level; // 游戏难度
    PageType pagetype; // 当前页面的类别
    vector<BUTTON> buttons; // 当前页面上的所有按钮
    int button_nums;
    ExMessage msg; // 本页面的消息
public:
    wstring bkpaper_name; // 页面背景图片的文件名
```

其中，游戏难度 **level** 和页面类别 **pagetype** 是枚举类型，相关的定义如下：

```
// 各种页面的枚举类型
enum PageType {
    menu_page, // 主菜单页面
    easy_game_page, // 单机-简单模式
    multi_game_page, // 双蛇-人机模式 or 双人模式
    rank_page, // 排行榜
};
// 游戏难度的枚举类型
enum Level {
    first, // 入门版
    second, // 进阶版
    third, // 高级版
};
```

此外，由于一个页面中含有数量不定的按钮，因此我们考虑使用一个 **vector** 来存储

所有 **BUTTON** 的相关信息。每次判断鼠标悬停和鼠标点击事件时，我们需要遍历 **vector** 中的所有 **BUTTON** 来判断。

3.2.1 向当前页面添加按钮

根据前一小节的阐述我们可以知道，添加新的按钮只需给出该按钮的基本属性即可。我们在 **PAGE** 类里定义 **add_Button()** 函数，用于给当前页面添加新的按钮信息，并纳入 **vector buttons** 管理：

```
BUTTON player_mode(830, 440, 85, 35, _T("双人模式"), myGame_multi);
this->buttons.push_back(player_mode);
button_nums++;
```

图 3.4 向当前页面添加 1 个按钮的操作

3.2.2 当前页面的运行

所有页面的运行都依靠 **PAGE** 类的类成员函数 **run()** 实现，其运行机制为：

- ② 不断执行消息循环
- ③ 依据消息处理事件
- ④ ③ 页面绘制
- ⑤ ④ manager 管理游戏进程

其中的消息循环使用 **easyX** 内置的 **ExMessage** 消息体，并用 **peekMessage()** 不断获取键盘与鼠标信息。

4 实验过程遇到的问题及解决

4.1 二维 vector 的越界

由于本程序中从始至终只有一个 **SNAKE** 类管理所有小蛇，在 2 条蛇的情况下我们就需要使用二维的 **vector** 来存储页面中所有小蛇的坐标 **pos**。由于起初不会正确使用多维 **vector** 的初始化方式，报错 **vector 越界**：



图 4.1.1 二维 vector 未初始化导致的越界

上网搜索资料后，学到二维 vector 在容器大小未知的情况下可以使用 `resize()` 成员函数做初始化，由于我可以确定本程序中最多有 2 条小蛇，所以我在初始化时将外层的 vector 的大小 `resize` 为 2，问题得到解决。

此外，相同的问题也在“进阶版”生成障碍物墙壁时遇到。当小蛇撞到墙壁需要按当前蛇的坐标在对应位置生成软硬障碍墙壁时，调 `WALL` 类成员函数 `wall_build` 出现同样的 vector 越界问题：

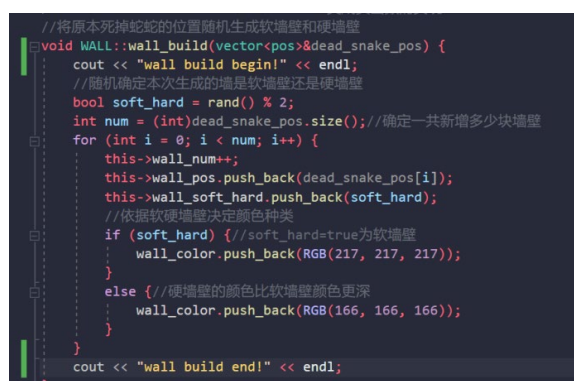


图 4.1.2 wall_build 函数出现 vector 越界问题

尝试单步执行，只能完成第 1 次的 vector 填入：

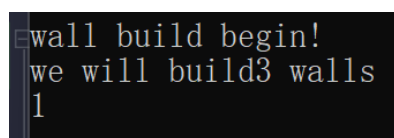


图 4.1.3 单步执行输出提示结果

使用控制台调试，发现从 `pos` 第 2 个坐标开始显示为 `NULL` 的结构：

值	类型
<NULL 处的结构>	pos &
{x=458 y=300 color=4349166}	const pos &

图 4.1.4 调试显示 vector 中填入 NULL 的结构

错误原因是当一次游戏结束之后，对小蛇的 `snake_pos` 没有作 vector 的重置，我们在 `reset` 函数中将蛇的大小重置为 `resize(2)` 问题得到解决。

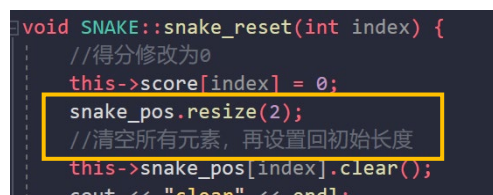


图 4.1.5 reset 函数中重置二维 vector 的大小

4.2 AI 模式的越界

按照我的游戏设定，每当界面中所有好果子（彩色果子）都被吃掉之后，就需要重新生成所有果子。但在调试过程中出现了在 AI 模式下，只剩下灰色果子（即坏果子）时，果实不更新生成，有 Windows 弹窗报错。

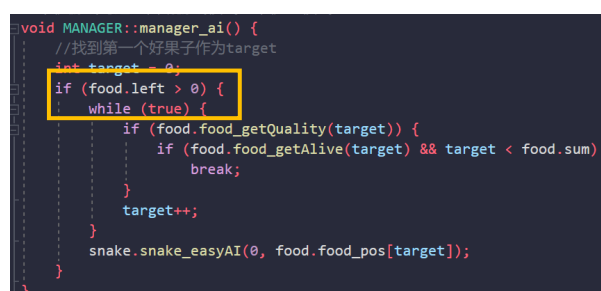


图 4.2.1 AI 模式下对好果子的搜索

原因：缺少一个条件判断“`food.left>0`”，如果当前页面没有好果子，就不需要再判断方向了，等待果子重新生成就可以，不然的话：

```

if (this->pagetype == easy_game_page) {
    if (manager.manager_isRunning()) {
        manager.manager_crashwall();
        manager.manager_eatfood();
        manager.manager_update();
        manager.manager_render();
    }
}

```

图 4.2.2 manager 对游戏进程的顺序控制

因为 eatfood 在 update 里，同时 AI 模式控制方向的函数也写在了 update 里，所以当 eatfood 已经把所有好果子吃完，但是本次吃完还没有 reset（要到下一次才会 reset，因为我把 reset 写在 eatfood 函数的最前面了）此时实际上已经没有好果子了（left 已经是 0），但是我还要去 AI 模式下面搜索好果子，搜不到就一直进入死循环（上一个图），所以就非法爆炸了。

4.3 AI 爆闪蛇头

在挂机模式（简单模式下的 AI 蛇）或人机对战模式下，发现 AI 蛇有时会在目标食物左右 1 个像素的位置反复摇动，画面极度抖动，我愿称之为“爆闪蛇头”：

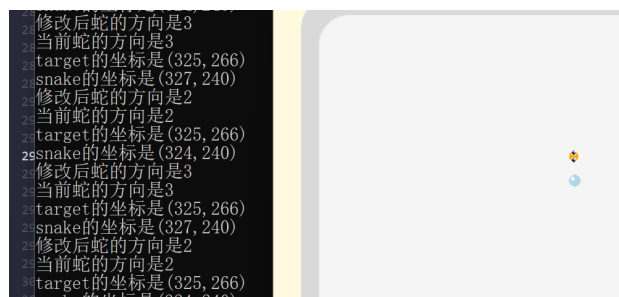


图 4.3.1 AI 爆闪蛇头的错误

由于初始设置的小蛇速度是 3（加速状态），因此小蛇每次会移动 3 个像素，一旦蛇头位于“少迈一步到不了，多迈一步超范围”的尴尬境地，就会反复出现掉头折返的“摇头爆闪”景象。

解决方法一为改变蛇的步速为一次移动

1 个像素，一为增大吃到果实的坐标检索范围。

4.4 好果实吃完后不重新生成

在小蛇吃完全部好果实（彩色果子），但游戏区中还剩余坏果实（灰色果子）时，按照我的规则设定应该全部重新生成果子，但在调试过程中，出现了游戏进入死循环的情况：界面卡住，小蛇无法继续移动，所有食物不生成。

```

reset begin
reset done
reset begin
reset done
reset begin
reset done
reset begin
reset done
reset done
reset begin

```

图 4.4.1 食物重置进入死循环

经单步输出调试，这里检查食物不重合进死循环了：

```

bool valid = true;
while (true) {
    food_pos[i].x = rand() % (GAME_WIDTH - 40) + GAME_BEGIN_X + 20;
    food_pos[i].y = rand() % (GAME_HEIGHT - 40) + GAME_BEGIN_Y + 20;
    if (this->food_checkpos(i) == false)
        continue; // 若生成的位置与任意一个已有的食物重合，也需要重新生成
    cout << "检查食物不重合 done" << endl;
    // 确保新生成的果子位置不在任何一条蛇的身上
}

```

图 4.4.2 食物重置进入死循环

这说明下面这个条件分支无法将 while 循环 break 出去：

```

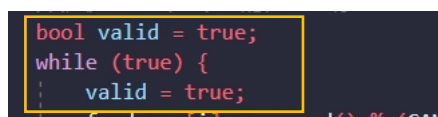
cout << "small for done" << endl;
// 假如通过条件分支成功，就跳出当前循环，否则就返回循环最开始再次生成
if (snake_index == snake_group.mode+1 && index == snake_group.length[snake_group.mode])
    break;

```

图 4.4.3 break 条件不成立无法跳出 while 循环

其原因在于，一旦在随机生成食物时出现了食物与原有食物、已有障碍物墙壁和小蛇身体的任一坐标重合时，就会置有效生成标志 bool valid 为 false，循环需要重新进行。

这就需要每次生成的第一次循环进入前将标志量 `value` 置为 `true`，我由于忘记这一步骤导致进入死循环：



```
bool valid = true;
while (true) {
    valid = true;
    ...
}
```

图 4.4.4 重置标志量后修改成功

5 游戏攻略与心得体会

5.1 算法设计心得

本次作业中我将所有具备单独属性的对象都封装成类：包括 GUI 界面的按钮类 `BUTTON`，页面类 `PAGE`；游戏对象小蛇类 `SNAKE`，食物类 `FOOD`，障碍物墙壁 `WALL` 类；游戏综合控制类 `MANAGER` 类。真真切切体会到“封装性”这一性质，尤其是在后续为各个页面实现 `run` 和 `buttons` 添加时，我只需要一句代码就可以建立接口，让代码更具泛化意义。

然而在设计 AI 蛇的算法时，我发现自己的考虑不周。我为了让游戏体验更加丝滑，没有将游戏区划分为一定的网格，而是采用按像素移动的方式。虽然这样设计对游戏者的体验感来说“更爽”，但是也为 AI 蛇的算法设计带来了问题：

由于小蛇按单个像素移动的特性，实现 `snake` 到 `target food` 的最短路径 `BFS` 算法变得十分困难，计算体量非常大。因此我只能舍弃这种让 AI 蛇变得更聪明的办法，每次只能选择与当前 `snake_head` 路径距离最短的食物作为 `target food`，并且直接走“一次转向的折线”路径。

虽然如此，但我只得安慰自己，这样给 AI 蛇降智商的做法可以大大提升人机模式的玩

家体验感，以不至于节节败退、屡战屡败。在算法设计上还是考虑欠妥，以后更要多加注意！

5.2 游戏攻略说明

为了方便使用者更好地体验本游戏，一下对游戏过程可能遇到的问题作简要的攻略说明：

5.2.1 皮肤颜色选择

由于完成程序的时间有限，本程序只对单人模式下的 `小蛇` 开放了皮肤颜色选择权限；因此主菜单界面上第一排的选择键只对单人模式有效。双人模式和 AI 人机对战模式下，默认 `Player1(Human Player)` 为红色皮肤，`Player2(AI-player)` 为蓝色皮肤。

5.2.2 游戏控制

本程序除了可以通过方向键和 `WSAD(wsad)` 对 `小蛇` 的方向进行控制以外，还可以分别使用空格键和字母键 `Z(z)` 对 `小蛇` 的速度进行控制。每按下一次可以切换一次 `小蛇` 的行进速度。

5.2.3 历史记录与排行榜

本游戏的历史记录和排行榜记录分别存储在 `record.txt` 和 `rank.txt` 两个文件中。需要注意的是，`record` 会存储所有游戏的历史记录，即使程序结束记录仍然保留。

`rank` 存储所有游戏历史中各个模式的前 3 名，即使程序结束依旧可以加载历史排行榜。但需要注意的是，首次编译程序时需要将 `rank.txt` 从根目录下删除，否则可能会带

来 windows 错误。

5.3 OOP 回忆录

又一学期，这次是 OOP 结束了。

时常觉得不可思议，其实自己接触编程也只有短短不到 9 个月的时间。从第一次看到 **hello world**，到汉诺塔和消灭星星的丝滑移动，再到这学期一次又一次的大作业。我看过文件压缩的乱码，看过 **lena** 和我玩躲猫猫，看过钟表逆向旋转，也看过贪吃蛇的漏洞百出。我总是不敢相信曾经几行代码都写不出来的自己能独立完成一个上千行的大作业，然而事实就是如此，自己可以做到！

从上学期第一次打开魔鬼般的高程网站开始，我的大学生活便是天翻地覆了。在图书馆一坐就是一天，因为自己总是 **de** 不出 **bug**，写汉诺塔看到色块乱跑，想哭；写消灭星星临到 **ddl** 却还是跑不起来，想哭；写图像压缩，看到周围的朋友们都完成了只有自己还停留在乱码的阶段，更是直接在导论课摸鱼写 **Lena** 的间隙泪崩（当时着实把授课的张老师吓到了），还是想哭。

看到周围总是有资深的“经验型选手”和异禀的“天赋型选手”和我站在同一赛道，说自己不难过、不内耗、不焦虑是很难的。我的 **clock** 刚转起来，人家已经做出动态的颜色渐变特效；我的小蛇在反复摇头爆闪的时候，人家已经开启了无尽的剧情模式……但是，那又如何呢？对于我自己而言，我相信我已经在我现有水平以内做到最好，编程给我带来的不只有痛苦，也有快乐。我想它之所以有如此之大的魅力，就是因为 **debug** 成功那一刻的心情是任何事情都无法比拟的，那不是一种单纯的激动和高兴，也不是一种解脱，就好像跟着红军长征了几万里，最后连草鞋都走掉了、粮食都吃没了，好不容易走到延

安却告诉你明天还要打仗。然而此时此刻你只在乎你到没到延安，明天死不死的事情，此刻我不在乎。

坦白而言，我们不能说自己通过高程和 OOP 这两门课程就无比爱上了写代码。这九个月的学习，带给我更多的不是“发现自己的真命天子和挚爱兴趣居然是编程”，而是那种面对一次又一次束手无策的难题能够一点一点去解决的成长，我很感谢我自己。

道阻且长，往事暗沉不可追，愿来日研习之路光明灿烂。

6 源代码

6.1 对象类

6.1.1 Item.h

```
/*计算机 2252429 蔡宇轩*/
#pragma once
#include "myGUI.h"

const COLORREF skin_color[3] =
{ RGB(238,92,66),RGB(255,165,0),RGB(0,154,205)
};

typedef struct pos {
    int x;//位置 X 坐标
    int y;//位置 Y 坐标
    COLORREF color;//当前位置的颜色
}pos;
//枚举出 4 种可能的方向
enum DIR {
    up, down, left, right
};
enum MODE {
    easy,//单机模式：页面上只有一条蛇
    multi//双机模式：页面上有 2 条蛇，包括人机对战和双人模式
};

class SNAKE {
private:
    void snake_invalid(int index);
public:
    Level level;//当前小蛇所属的游戏难度，决定具体的功能设定
    MODE mode;//决定页面上是一条蛇还是两条蛇
    int color[2];//小蛇的皮肤颜色
```



```

    int length[2]; //蛇的长度
    enum DIR direction[2]; //蛇的方向
    int radio; //蛇身的半径
    int speed[2]; //蛇的前进速度
    vector<vector<pos>>> snake_pos; //每一节蛇身的坐标
    int score[2]; //这条蛇当前得到的总分
    int life[2]; //生命值
    int crashtimes[2]; //撞墙次数
    bool cgtoAI; //简单模式下是否切换 AI 模式

public:
    SNAKE(); //构造函数
    void snake_eyeRender(int x, int y); //蛇的眼睛绘制
    void snake_bodyRender(int index); //蛇的身体绘制
    void set_skincolor(int index, int color); //重置皮肤颜色
    void snake_move(int index); //蛇的移动
    void snake_event(int inedx, BYTE brcode); //蛇的事件处理
    void snake_increase(int index, int increase); //蛇的长度增加
    void snake_decrease(int index, int decrease); //蛇的长度缩短
    void way_to_exit(); //不同游戏模式下的退出方式
    void snake_reset(int index); //重置蛇的各项属性
    bool snake_checklife(int index); //检查蛇的存活状态
    void snake_easyAI(int index, pos target); //单条蛇的 AI 模式算法
};

class WALL;

class FOOD {
private:
    bool good_bad[50]; //设置是好果子还是坏果子
    int score[50];
    bool alive[50]; //果子是否被吃掉
public:
    COLORREF color[50]; //食物的颜色: 好果子-彩色, 坏果子-灰色
    int sum; //每一次的果子总数
    int radio; //绘制半径
    pos food_pos[50]; //食物的坐标
    int left; //剩余的还没被吃掉的果子
public:
    FOOD() {}; //默认构造函数
    void food_set(SNAKE& snake_group, WALL& wall); //设置所有果子的属性 (构造函数调用, 重置函数就是 set 本身)
    void food_render(); //绘制渲染
    bool food_getAlive(int index); //判断食物存活状态
    void food_setAlive(int index, bool setAlive); //设置食物存活状态

```

```

    int food_getScore(int index); //获取食物对应分数
    bool food_getQuality(int index); //获取食物好坏性质
    bool food_checkpos(int food_index); //检查当前生成的食物是否与已有食物重合
    void food_Cgbysnake(vector<pos>& snake_pos); //在撞墙后的蛇的尾部生成随机食物
};

//障碍物类
class WALL {
private:
    int wall_num; //墙壁总数
    int wall_radio;
    vector<bool> wall_soft_hard; //软墙壁和硬墙壁
    vector<COLORREF> wall_color; //软墙壁和硬墙壁
public:
    vector<pos> wall_pos; //除了边框以外, 所有障碍物的位置
public:
    WALL(); //构造函数
    void wall_build(vector<pos> dead_snake_pos); //把死掉的蛇蛇变成墙壁
    void wall_render(); //绘制所有墙壁
    bool wall_getQuality(int wall_index); //获取当前墙壁的属性
    bool wall_checkpos(int food_x, int food_y); //检查新生成的食物是否出现在了墙壁上
    void wall_reset(); //重置墙壁属性
};

```

6.1.2 Item.cpp

```

#include "manager.h"
#include "Record.h"

extern MANAGER manager;
const COLORREF good_food_color[5] = { RGB(255, 127, 0), RGB(154, 205, 50), RGB(205, 96, 144), RGB(173, 216, 230), RGB(255, 215, 0) };
#define GAME_BEGIN_X 46
#define GAME_BEGIN_Y 93
#define GAME_WIDTH 415
#define GAME_HEIGHT 398
#define GAME_MULTI_BEGIN_X 49
#define GAME_MULTI_BEGIN_Y 145
#define GAME_MULTI_WIDTH 860
#define GAME_MULTI_HEIGHT 353

//菜单函数声明
void myMenu();
//写入文件记录函数声明
void write_record(PageType pagetype, MODE mode, Level level, char user1[], int score1,

```

```

int score2 = 0, char user2[] =
(char*)"AI_snake");

//简易模式下退出游戏页面
void easy_exit() {
    HWND h = GetHwnd();
    if (MessageBox(h, _T("游戏结束!\n 按确定键返回菜单"), _T("游戏提示"), MB_OK)) {
        //这里需要更新一下文件中的历史记录
        char user1[15] = { 0 };
        sprintf_s(user1, "%ws",
manager.user1);
        write_record(PageType::easy_game_page,
manager.snake.mode, manager.snake.level,
user1, manager.snake.score[0]);
        //更新排行榜
        easy_num++;
        update_rank(manager.snake.score[0],
user1, manager.snake.level, 1);
        write_rank();
        //简单状态下直接重置当前蛇的各项属性为初始
        状态, 不保存当前地图
        manager.food.food_set(manager.snake,
manager.wall);
        manager.snake.snake_reset(0);
        manager.wall.wall_reset();
        myMenu();
    }
}

//两个蛇的模式退出
void multi_exit() {
    HWND h = GetHwnd();
    if (manager.snake.cgtoAI == false) { //双人
        模式
        if (manager.winner == 1) //player1 win
            MessageBox(h, _T("本局赢家:
Player1"), _T("游戏提示"), MB_OK);
        else if (manager.winner == 2) //player2
        win
            MessageBox(h, _T("本局赢家:
Player2"), _T("游戏提示"), MB_OK);
        else //win equal
            MessageBox(h, _T("本局平局! "),
_T("游戏提示"), MB_OK);
        multi_num += 2;
        //multi_max =
max(manager.snake.score[0],
manager.snake.score[1]);
    }
    else { //人机对战模式
        if (manager.winner == 1) //player1 win
            MessageBox(h, _T("本局赢家:
Player"), _T("游戏提示"), MB_OK);
        else if (manager.winner == 2) //AI
        player2 win
            MessageBox(h, _T("本局赢家: AI-
Player"), _T("游戏提示"), MB_OK);
        else //win equal
            MessageBox(h, _T("本局平局! "),
_T("游戏提示"), MB_OK);
        ai_num++;
    }
    ai_num++;
}

```

```

}

//这里需要更新一下文件中的历史记录
char user1[15] = { 0 };
char user2[15] = { 0 };
//sprintf_s(user1, "%ws", manager.user1);
//sprintf_s(user2, "%ws", manager.user2);

WideCharToMultiByte(CP_ACP, 0,
manager.user1, -1, user1, 15, NULL, NULL);
WideCharToMultiByte(CP_ACP, 0,
manager.user2, -1, user2, 15, NULL, NULL);

write_record(PageType::easy_game_page,
manager.snake.mode, manager.snake.level,
user1, manager.snake.score[0],
manager.snake.score[1], user2);

//更新排行榜
//双人模式和人机模式都同时更新两个玩家的游戏记
录, 包括 AI 蛇的记录
update_rank(manager.snake.score[0], user1,
manager.snake.level, 3 -
manager.snake.cgtoAI);
update_rank(manager.snake.score[1], user2,
manager.snake.level, 3 -
manager.snake.cgtoAI);
write_rank(); //写入排行榜文件

//简单状态下直接重置当前蛇的各项属性为初始状
态, 不保存当前地图
manager.food.food_set(manager.snake,
manager.wall);
manager.wall.wall_reset();
manager.snake.snake_reset(0);
manager.snake.snake_reset(1); //双人对战第二
条蛇也需要重置
//返回主菜单
myMenu();
}

//单机模式和双机模式下的同退出情况
void SNAKE::way_to_exit() {
    if (this->mode == MODE::easy) {
        easy_exit();
    }
    else if (this->mode == MODE::multi) {
        multi_exit();
    }
}

//3 个版本下蛇撞到边界时的不同处理
void SNAKE::snake_invalid(int index) {
    bool ai = this->cgtoAI;
    bool notset = (this->mode == MODE::multi)
&& ai;
    int score_before = 0;
    switch (this->level) {
        case Level::first:
            manager.manager_multi_setwinner(index
+ 1);
            way_to_exit();
            break;
        case Level::second:

```

```

        manager.wall.wall_build(this->snake_pos[index]);
        score_before =
manager.snake.score[index];
        manager.snake.snake_reset(index);
        manager.snake.score[index] =
score_before;
        if (notset)
            this->cgtoAI = ai;
        break;
        case Level::third:
            manager.food.food_Cgbysnake(manager.snake.snake_pos[index]);
            score_before =
manager.snake.score[index];
            manager.snake.snake_reset(index);
            manager.snake.score[index] =
score_before;
            if (notset)
                this->cgtoAI = ai;
            break;
    }
}

//初始化蛇
SNAKE::SNAKE() {
    this->cgtoAI = false;
    //先初始化存放各个蛇头的坐标二维的 vector
    snake_pos.resize(2);
    this->radio = 5;
    //2 条蛇的默认颜色
    this->color[0] = 0; color[1] = 2;
    this->set_skincolor(0, color[0]);
    this->set_skincolor(1, color[1]);
    for (int index = 0; index < 2; index++) {
        this->direction[index] = DIR::right;
    }
    //初始方向 right
    this->length[index] = 3; //初始长度为 3
    this->score[index] = 0; //初始得分
    this->speed[index] = 1;
    this->life[index] = 10; //一节身体 (不算蛇头) 5 个生命值
    //初始蛇头位置放入 vector
    pos begin = { 300 + index * 100, 300 + index * 100, skin_color[this->color[index]] };
    snake_pos[index].push_back(begin);

    //初始化各个身体节的颜色
    for (int i = 1; i <
this->length[index]; i++) {
        pos section = { 0 };
        if (i % 2)
            section.color = WHITE;
        else
            section.color =
skin_color[this->color[index]];

        section.x = snake_pos[index][i - 1].x - radio;
        section.y = snake_pos[index][i - 1].y;

```

```

        snake_pos[index].push_back(section)
    }
}

    crashtimes[0] = 0, crashtimes[1] = 0;
}

//渲染蛇的眼睛
void SNAKE::snake_eyeRender(int x, int y) {
    setfillcolor(WHITE);
    solidcircle(x, y, 3);
    setfillcolor(BLACK);
    solidcircle(x, y, 2);
    setfillcolor(WHITE);
    solidcircle(x + 1, y - 1, 1);
}

//渲染蛇的身体
void SNAKE::snake_bodyRender(int index) {
    for (int i = this->length[index] - 1; i >= 0; i--) {
        setfillcolor(this->snake_pos[index][i].color);
        solidcircle(snake_pos[index][i].x, snake_pos[index][i].y, this->radio);
    }
    //画眼睛, 针对当前蛇头朝向作分类讨论
    int eyewidth = 4; //两眼之间的宽度
    switch (this->direction[index]) {
        case DIR::up:
        case DIR::down:
            snake_eyeRender(snake_pos[index][0].x - eyewidth, snake_pos[index][0].y);
            snake_eyeRender(snake_pos[index][0].x + eyewidth, snake_pos[index][0].y);
            break;
        case DIR::left:
        case DIR::right:
            snake_eyeRender(snake_pos[index][0].x, snake_pos[index][0].y - eyewidth);
            snake_eyeRender(snake_pos[index][0].x, snake_pos[index][0].y + eyewidth);
            break;
    }
}

//更改蛇的皮肤颜色
void SNAKE::set_skincolor(int index, int color) {
    this->color[index] = color;
    for (int i = 0; i < this->length[index]; i++) {
        if (i % 2 == 0)
            this->snake_pos[index][i].color = skin_color[color];
    }
}

//蛇的移动模块
void SNAKE::snake_move(int index) {
    int up_limit = 0, down_limit = 0, left_limit = 0, right_limit = 0;
    switch (this->mode) {
        case MODE::easy:
            up_limit = GAME_BEGIN_Y;
            down_limit = GAME_BEGIN_Y + GAME_HEIGHT;

```

```

        left_limit = GAME_BEGIN_X;
        right_limit = GAME_BEGIN_X +
GAME_WIDTH;
        break;
    case MODE::multi:
        up_limit = GAME_MULTI_BEGIN_Y;
        down_limit = GAME_MULTI_BEGIN_Y +
GAME_MULTI_HEIGHT;
        left_limit = GAME_MULTI_BEGIN_X;
        right_limit = GAME_MULTI_BEGIN_X +
GAME_MULTI_WIDTH;
        break;
    }

    //后一个的位置是前一个在上次的位置
    for (int i = this->length[index] - 1; i >
0; i--) {
        snake_pos[index][i].x =
snake_pos[index][i - 1].x;
        snake_pos[index][i].y =
snake_pos[index][i - 1].y;
    }
    //根据方向做移动
    switch (this->direction[index]) {
    case DIR::up:
        snake_pos[index][0].y -=
this->speed[index];
        if (snake_pos[index][0].y <= up_limit)
        { /*这里依据游戏模式作特判*/ /*直接死亡还是
变成食物和墙*/
            snake_invalid(index);
            crashtimes[index]++;
        }
        break;
    case DIR::down:
        snake_pos[index][0].y +=
this->speed[index];
        if (snake_pos[index][0].y >=
down_limit)
        { /*这里依据游戏模式作特判*/ /*直接死亡还是
变成食物和墙*/
            snake_invalid(index);
            crashtimes[index]++;
        }
        break;
    case DIR::left:
        snake_pos[index][0].x -=
this->speed[index];
        if (snake_pos[index][0].x -
this->radio <= left_limit)
        { /*这里依据游戏模式作特判*/ /*直接死亡还是
变成食物和墙*/
            snake_invalid(index);
            crashtimes[index]++;
        }
        break;
    case DIR::right:
        snake_pos[index][0].x +=
this->speed[index];
        if (snake_pos[index][0].x +
this->radio >= right_limit)
        { /*这里依据游戏模式作特判*/ /*直接死亡还是
变成食物和墙*/
            snake_invalid(index);

```

```

        crashtimes[index]++;
    }
    break;
}
}
//蛇的事件
void SNAKE::snake_event(int index, BYTE
vkcode) {
    if (index == 0) {
        switch (vkcode) {
        case VK_LEFT:
            if (this->direction[index] !=
DIR::right) {
                this->direction[index] =
DIR::left;
            }
            break;
        case VK_RIGHT:
            if (this->direction[index] !=
DIR::left) {
                this->direction[index] =
DIR::right;
            }
            break;
        case VK_UP:
            if (this->direction[index] !=
DIR::down) {
                this->direction[index] =
DIR::up;
            }
            break;
        case VK_DOWN:
            if (this->direction[index] !=
DIR::up) {
                this->direction[index] =
DIR::down;
            }
            break;
        case VK_SPACE:
            if (this->speed[index] == 1) //加速
处理
                this->speed[index] = 3;
            else if (this->speed[index] == 3)
                this->speed[index] = 1; //已经处
于加速状态, 再次按空格还原回初始速度
            break;
        }
    }
    else if (index == 1) {
        switch (vkcode) {
        case 'a':
        case 'A':
            if (this->direction[index] !=
DIR::right) {
                this->direction[index] =
DIR::left;
            }
            break;
        case 'd':
        case 'D':
            if (this->direction[index] !=
DIR::left) {
                this->direction[index] =
DIR::right;
            }
            break;
        case 'w':
        case 'W':
            if (this->direction[index] !=
DIR::down)

```

```

        this->direction[index] =
DIR::up;
        break;
        case 's':
        case 'S':
            if (this->direction[index] !=
DIR::up)
                this->direction[index] =
DIR::down;
            break;
        case 'z':
        case 'Z':
            if (this->speed[index] == 1)//加速
处理
                this->speed[index] = 3;
            else if (this->speed[index] == 3)
                this->speed[index] = 1;//已经处
于加速状态, 再次按空格还原回初始速度
            break;
        }
    }
}
//蛇的增长
void SNAKE::snake_increase(int index, int
increase) {
    ///新增证明得分, 游戏提示音
    //mciSendString(_T("close incorrect"),
NULL, 0, NULL);
    //mciSendString(_T("open ./increase.mp3
alias incorrect"), NULL, 0, NULL);
    //mciSendString(_T("play incorrect"),
NULL, 0, NULL);
    //对新增节数的颜色作初始化
    for (int i = this->length[index]; i <
length[index] + increase; i++) {
        pos incre_section = { 0 };
        if (i % 2 == 0)
            incre_section.color =
skin_color[this->color[index]];
        else
            incre_section.color = WHITE;
        this->snake_pos[index].push_back(incre
_section);
    }
    //改变蛇身长度
    this->length[index] += increase;// 吃好果子
得分
    //改变当前蛇的得分
    this->score[index] += increase * 10;
    //改变当前蛇的生命数
    this->life[index] += increase * 5;
}
//蛇的减少
void SNAKE::snake_decrease(int index, int
decrease) {
    //判断减少的长度是否合法
    if (decrease >= this->length[index]) {
        this->life[index] = 0;
        this->score[index] -= decrease * 10;
        return;
    }
    //若长度减少后蛇还能存活, 则从 vector 中最后
一个元素开始按长度减少
    for (int i = 0; i < decrease; i++)
        this->snake_pos[index].pop_back();

```

```

        this->length[index] -= decrease;
        this->life[index] -= decrease * 5;//吃坏果
子不得分 减生命
    }
    //重置蛇的各项属性
    void SNAKE::snake_reset(int index) {
        //重置两条蛇颜色
        this->color[0] = 0; color[1] = 2;
        this->set_skincolor(0, color[0]);
        this->set_skincolor(1, color[1]);
        //得分修改为 0
        this->score[index] = 0;
        snake_pos.resize(2);
        //清空所有元素, 再设置回初始长度
        this->snake_pos[index].clear();
        this->length[index] = 3;//蛇身长度设回初始长
度=3
        //重置蛇头位置及其后两节的位置
        //初始蛇头位置放入 vector
        pos begin = { 300 + index * 100, 300 +
index * 100, skin_color[this->color[index]] };
        snake_pos[index].push_back(begin);
        //初始化各个身体节的颜色
        for (int i = 1; i < this->length[index];
i++) {
            pos section = { 0 };
            if (i % 2)
                section.color = WHITE;
            else
                section.color =
skin_color[this->color[index]];

            section.x = snake_pos[index][i - 1].x
- radio;
            section.y = snake_pos[index][i - 1].y;
            snake_pos[index].push_back(section);
        }
        //蛇的速度修改回初始速度
        this->speed[index] = 1;
        //是否进入 AI 模式修改回 false
        this->cgtoAI = false;
        //蛇的生命值修改回初始状态
        this->life[index] = 10;
    }
    //检查蛇的死活状态
    bool SNAKE::snake_checklife(int index) {
        //蛇的生命值降为 0 及以下, third 模式下撞墙超 5
次, 头尾相接: 游戏结束的判定条件
        if (this->life[index] <= 0 || (level ==
Level::third && crashtimes[index] > 5) /*||
(snake_pos[index][0].x ==
snake_pos[index][length[index] - 1].x &&
snake_pos[index][0].y ==
snake_pos[index][length[index] - 1].y)*/)
        {
            //设置本局游戏的胜利者
            manager.manager_multi_setwinner(index
+ 1);

            ///分数清空
            //manager.snake.score[index] = 0;
            if (level == Level::third &&
crashtimes[index] > 5) {
                Hwnd h = GetHwnd();

```



```

        MessageBox(h, _T("Sorry!!在当前模式
        下撞墙超过 5 次就会结束游戏哦~"), _T("游戏提示"),
        MB_OK);
    }
    return false; //蛇到了该死的时候就返回
false
}
else if (mode == MODE::multi &&
abs(score[0] - score[1]) > 200) {
    //设置游戏胜者
    int winner = (score[0] > score[1]) ?
1 : 0;
    manager.manager_multi_setwinner(winner
+ 1);
    return false;
}
else
    return true; //小蛇还活着
}
//简单模式下的 AI 蛇
void SNAKE::snake_easyAI(int index, pos
target) {
    //获取当前的蛇头坐标
    int head_x = this->snake_pos[index][0].x;
    int head_y = this->snake_pos[index][0].y;

    if (abs(head_x - target.x) > 3 &&
abs(head_y - target.y) > 3)
        speed[index] = 3;
    else if (head_x == target.x && abs(head_y
- target.y) > 3)
        speed[index] = 3;
    else if (head_y == target.y && abs(head_x
- target.x) > 3)
        speed[index] = 3;
    else
        speed[index] = 1;

    //将蛇头坐标与目标坐标比较
    if (head_x < target.x) {
        if (direction[index] != DIR::left)
            direction[index] = DIR::right;
        else {
            if (target.y > head_y)
                direction[index] = DIR::down;
            else if (target.y < head_y)
                direction[index] = DIR::up;
        }
    }
    else if (head_x > target.x) {
        if (direction[index] != DIR::right)
            direction[index] = DIR::left;
        else {
            if (target.y > head_y)
                direction[index] = DIR::down;
            else if (target.y < head_y)
                direction[index] = DIR::up;
        }
    }
    else if (head_y < target.y) {
        if (direction[index] != DIR::up)
            direction[index] = DIR::down;
        else {
            if (target.x > head_x)
                direction[index] = DIR::right;

```

```

        else if (target.x < head_x)
            direction[index] = DIR::left;
    }
}
else if (head_y > target.y) {
    if (direction[index] != DIR::down)
        direction[index] = DIR::up;
    else {
        if (target.x > head_x)
            direction[index] = DIR::right;
        else if (target.x < head_x)
            direction[index] = DIR::left;
    }
}
}

/*****FOOD 类成员函数的实现*****/

//判读食物当前状态
bool FOOD::food_getAlive(int index) {
    return this->alive[index]; //还没被吃掉返回
true
}
//绘制食物
void FOOD::food_render() {
    for (int i = 0; i < this->sum; i++) {
        if (this->alive[i] == true) {
            setfillcolor(this->color[i]);
            solidcircle(food_pos[i].x,
food_pos[i].y, this->radio);
            setfillcolor(WHITE);
            solidcircle(food_pos[i].x - 2,
food_pos[i].y - 2, 2);
        }
    }
}
//初始化或者重置当前页面所有果子的状态
void FOOD::food_set(SNAKE& snake_group, WALL&
wall) {

    int begin_x = (manager.snake.mode ==
MODE::easy) ? GAME_BEGIN_X :
GAME_MULTI_BEGIN_X;
    int begin_y = (manager.snake.mode ==
MODE::easy) ? GAME_BEGIN_Y :
GAME_MULTI_BEGIN_Y;
    int width = (manager.snake.mode ==
MODE::easy) ? GAME_WIDTH : GAME_MULTI_WIDTH;
    int height = (manager.snake.mode ==
MODE::easy) ? GAME_HEIGHT : GAME_MULTI_HEIGHT;

    //先将所有果子的 alive 属性都设为 false
    for (int i = 0; i < 50; i++)
        this->alive[i] = false;
    //首先随机生成初始时的果子总数
    this->sum = rand() % 5 + 1;
    this->left = 0; //好果子的初始数量为 0, 根据后
面生成的果子属性计数
    //依次设置果子的属性
    for (int i = 0; i < this->sum; i++) {
        //设置所有果子的属性都为没被吃掉
        this->alive[i] = true;
        //随机生成果子的分数
        this->score[i] = rand() % 3 + 5;
    }
}

```

```

        //随机生成这些果子的好坏属性
        this->good_bad[i] = rand() % 2; //1-true-good,0-false-bad
        //若是生成好果子,则计数加1
        if (good_bad[i])
            this->left++;
        //并依据好坏属性设置果子的颜色
        if (good_bad[i] == true) //good
            this->color[i] =
good_food_color[rand() % 5];
        else //bad
            this->color[i] = RGB(139, 137,
112);

        //随机生成当前果子的位置坐标
        bool valid = true;
        while (true) {
            valid = true;
            food_pos[i].x = rand() % (width -
40) + begin_x + 20;
            food_pos[i].y = rand() % (height -
40) + begin_y + 20;
            //若生成的位置与任意一个已有的食物重
            合,也需要重新生成
            while (this->food_checkpos(i) ==
false)
            {
                food_pos[i].x = rand() % (width
- 40) + begin_x + 20;
                food_pos[i].y = rand() %
(height - 40) + begin_y + 20;
            }

            //确保新生成的果子不在任何一个墙上
            while
(wall.wall_checkpos(food_pos[i].x,
food_pos[i].y) == false) {
                food_pos[i].x = rand() % (width
- 40) + begin_x + 20;
                food_pos[i].y = rand() %
(height - 40) + begin_y + 20;
            }

            //确保新生成的果子位置不在任何一条蛇的
            身上
            int snake_index = 0;
            int index = 0;
            for (snake_index = 0; snake_index
<= (int)sake_group.mode; snake_index++) {
                for (index = 0; index <
snake_group.length[snake_index]; index++) {
                    if (food_pos[i].x ==
snake_group.snake_pos[snake_index][index].x ||
                        food_pos[i].y ==
snake_group.snake_pos[snake_index][index].y) {
                        valid = false;
                        break;
                    }
                }
                if (valid == false)
                    break;
            }
            if (valid == false)
                continue;
            else

```

```

                break;
            } //生成了正确的食物坐标
        }
    }
    //设置某一个食物的存活状态
    void FOOD::food_setAlive(int index, bool
setAlive) {
        this->alive[index] = setAlive;
    }
    //获取某一个食物的分数
    int FOOD::food_getScore(int index) {
        return this->score[index];
    }
    //判断某一个食物的好坏质量
    bool FOOD::food_getQuality(int index) {
        return this->good_bad[index]; //好果子返回
true
    }
    //判断当前生成的这个位置会不会与已有的食物重叠。若
    不重叠说明位置有效返回 true
    bool FOOD::food_checkpos(int food_index) {
        for (int i = 0; i < food_index; i++) {
            if (this->food_pos[food_index].x ==
food_pos[i].x && this->food_pos[food_index].y
== food_pos[i].y)
                return false;
        }
        return true;
    }
    //在撞墙的蛇的尾部随机生成食物
    void FOOD::food_Cgbysnake(vector<pos>&
snake_pos) {
        if (this->sum >= 50)
            return;
        //蛇在撞墙的位置随机变成 1~3 个食物
        int length = snake_pos.size();
        int newpos[3] = { -1, -1, -1 };
        //若长度小于 3 个则默认变成墙
        if (length > 3) {
            //在蛇的身上找 3 个随机位置
            for (int i = 0; i < 3; i++) {
                newpos[i] = rand() % (length - 1) +
1;
                for (int j = i - 1; j >= 0; j--) {
                    while (newpos[j] == newpos[i])
                        newpos[i] = rand() %
length;
                } //找到互不相同的 3 个位置
                this->alive[sum + i] = true;
            } //还没被吃掉
            this->score[sum + i] = rand() % 3 +
5; //随机分数
            this->good_bad[sum + i] = rand() %
2; //好坏果子
            //根据好坏属性设置颜色
            if (good_bad[sum + i] == true)
                { //good
                    this->color[sum + i] =
good_food_color[rand() % 5];
                    this->left++;
                }
            else //bad
                this->color[sum + i] = RGB(139,
137, 112);
        }
    }

```

```

        food_pos[sum + i] =
snake_pos[newpos[i]];
    }
    this->sum += 3;
}
}

/*****WALL 类成员函数的实现*****/
WALL::WALL() {
    this->wall_num = 0;
    this->wall_radio = 5;
}
//将原本吃掉蛇蛇的位置随机生成软墙壁和硬墙壁
void WALL::wall_build(vector<pos>dead_snake_pos) {
    //随机确定本次生成的墙是软墙壁还是硬墙壁
    bool soft_hard = rand() % 2;
    int num = (int)dead_snake_pos.size();//确定
    一共新增多少块墙壁
    for (int i = 0; i < num; i++) {
        this->wall_num++;
        this->wall_pos.push_back(dead_snake_po
s[i]);
        this->wall_soft_hard.push_back(soft_ha
rd);
        //依据软硬墙壁决定颜色种类
        if (soft_hard) { //soft_hard=true 为软墙
            wall_color.push_back(RGB(217, 217,
217));
        }
        else { //硬墙壁的颜色比软墙壁颜色更深
            wall_color.push_back(RGB(166, 166,
166));
        }
    }
}
//绘制所有墙壁
void WALL::wall_render() {
    for (int i = 0; i < wall_num; i++) {
        setfillcolor(this->wall_color[i]);
        int x = this->wall_pos[i].x -
wall_radio;
        int y = this->wall_pos[i].y -
wall_radio;
        solidroundrect(x, y, x + 2 *
wall_radio, y + 2 * wall_radio, 3, 3);
    }
}
//判断当前墙壁是软墙壁还是硬墙壁
bool WALL::wall_getQuality(int wall_index) {
    return this->wall_soft_hard[wall_index];
}
//检查新生成的食物是否出现在了墙壁上
bool WALL::wall_checkpos(int food_x, int
food_y) {
    for (int i = 0; i <
(int)this->wall_pos.size(); i++) {
        if (food_x == this->wall_pos[i].x &&
food_y == wall_pos[i].y)
            return false;
    }
    return true;
}

```

```

}

//重置墙壁属性
void WALL::wall_reset() {
    this->wall_num = 0;
    this->wall_soft_hard.clear();
    this->wall_color.clear();
    this->wall_pos.clear();
}

```

6.2 综合控制类

6.2.1 manager.h

```

#pragma once
#include "item.h"

//综合管理模块
class MANAGER {
private:
    int time; //每次游戏的计时起点
public:
    SNAKE snake; //管理页面中的所有蛇
    FOOD food; //管理页面中的所有食物
    WALL wall; //管理页面中的所有障碍物
    wchar_t user1[15]; //用于输出显示：第一条蛇的
    用户名
    wchar_t user2[15]; //用于输出显示：第二条蛇的
    用户名
    int winner; //存放每一局的游戏赢家
    bool checkRank;
private:
    void manager_draw_info(); //游戏进行中的相关
    信息
    void manager_readranking(); //读取文件中存储
    的排行榜信息
public:
    MANAGER(); //构造函数
    void manager_update(); //更新状态
    void manager_ai(); //单机游戏下的简易 AI 模式
    void manager_render(); //渲染
    void manager_event(BYTE& vkcode); //事件管理
    void manager_eatfood(); //吃食物的相关操作
    void manager_reset_time(); //游戏时间重置
    void manager_crashwall(); //撞墙的相关操作
    bool manager_isRunning(); //检查当局游戏是否
    还要继续
    void manager_multi_setwinner(int index); //
    双人模式下设定胜者
    void manager_renderRanking(); //打印排行榜
};

```

6.2.2 manager.cpp

```

#include "manager.h"
#include "Record.h"

```

```

//manager 类构造函数
MANAGER::MANAGER() {
    //snake 的构造函数已经隐式调用完毕
    //wall 的构造函数已经隐式调用完毕
    //food 调用的是无参空体, 所以需要显示设置
    food.radio = 6;
    food.food_set(snake, wall);
    //初始将胜者定为 0, 最终 player1 赢则 winner=1
    player2 赢则 winner=2
    winner = 0;
    //初始认为不查看排行榜
    checkRank = false;
    //加载原有的排行榜
    this->manager_readranking();
}
//单机游戏时进入自动游戏模式: 单机 ai 模式或人机对战模式
void MANAGER::manager_ai() {
    //找到最近的好果子作为 target
    int target = 0;
    int index = 0;
    //蛇头的坐标
    int head_x =
snake.snake_pos[snake.mode][0].x;
    int head_y =
snake.snake_pos[snake.mode][0].y;
    int min_d = INT_MAX;
    if (food.left > 0) {
        while (true) {
            if (index >= food.sum)
                break;
            if (food.food_getQuality(index))
            {
                if (food.food_getAlive(index))
                {
                    int food_x =
food.food_pos[index].x;
                    int food_y =
food.food_pos[index].y;
                    int distance = abs(head_x -
food_x) + abs(head_y - food_y);
                    if (distance < min_d)
                        target = index;
                }
            }
            index++;
        }
        if (target < food.sum)
            snake.snake_easyAI(snake.mode,
food.food_pos[target]);
    }
}
//渲染过程
void MANAGER::manager_render() {
    if (this->snake.mode == MODE::multi)
        Sleep(10);
    this->wall.wall_render();
    this->food.food_render();
    this->manager_draw_info();
    for (int i = 0; i <= (int)snake.mode; i++)
    {
        this->snake.snake_bodyRender(i);
    }
}

```

```

}
//处理事件
void MANAGER::manager_event(BYTE& vkcode) {
    if (snake.mode == MODE::multi &&
snake.cgtoAI == true)//人机对战模式
        this->snake.snake_event(0, vkcode);
    else { //单机自主模式或双人模式
        for (int i = 0; i <= snake.mode; i++)
            this->snake.snake_event(i, vkcode);
    }
}
//处理吃食物
void MANAGER::manager_eatfood() {
    for (int i = 0; i <= snake.mode; i++) {
        //蛇的坐标和食物的坐标相等就是发生了吃食物
        事件
        pos snakepos =
this->snake.snake_pos[i][0]; //某条蛇的蛇头坐标
        //当前页面中的每一个食物都要遍历判断一次
        for (int foodnum = 0; foodnum <
food.sum; foodnum++) {
            pos foodpos =
this->food.food_pos[foodnum]; //某一个食物的坐标
            int r = food.radio * 2; //获取食物半
            径
            if (food.left == 0) //所有好果子都已
            经被吃掉
            {
                food.food_set(snake, wall); //重
                置食物状态
                break;
            }
            if (snakepos.x >= foodpos.x - r &&
snakepos.x <= foodpos.x + r && snakepos.y >=
foodpos.y - r && snakepos.y <= foodpos.y + r
&& food.food_getAlive(foodnum))
            {
                //相撞状态, 吃到食物
                food.food_setAlive(foodnum,
false); //食物的 alive 属性改变
                food.color[foodnum] = RGB(245,
245, 245); //食物的颜色设为背景色(好像也不用设
                置。。。)
                //依据当前这个食物的属性作蛇的加分
                减分\长度增减等处理
                int change =
food.food_getScore(foodnum) - 4;
                if
(food.food_getQuality(foodnum)) { //好果子
                    snake.snake_increase(i,
change);
                    food.left--; //剩余食物数量减
                    1
                }
                else { //坏果子
                    snake.snake_decrease(i,
change);
                }
            }
        }
    }
}

```

```

}
//事件更新
void MANAGER::manager_update() {
    if (snake.cgtoAI) {
        manager_ai();
    }
    for (int i = 0; i <= snake.mode; i++)
        this->snake.snake_move(i);
}
//重置时间
void MANAGER::manager_reset_time() {
    this->time = clock(); //游戏开始使获取当前时间
}
//游戏进行中的相关信息展示
void MANAGER::manager_draw_info() {
    //从游戏开始到当前时刻已经经过的时间
    int passtime = (clock() - this->time) /
    CLOCKS_PER_SEC;
    int minute = passtime / 60; //已经经过的分钟数
    int second = passtime % 60; //已经经过的秒数
    TCHAR myTime[15] = { 0 };
    _stprintf(myTime, _T("%dmin%02ds"),
    minute, second);

    if (snake.mode == MODE::easy) {
        //显示单机模式下的时间显示
        settextrcolor(143, 115, 66));
        settextrstyle(40, 0, _T("Arial
BLACK"));
        setbkmode(TRANSPARENT);
        outtextxy(760, 170, myTime);

        //显示当前得分
        TCHAR myScore[5] = { 0 };
        _stprintf(myScore, _T("%d"),
        this->snake.score[0]);
        outtextxy(600, 240, myScore);

        //打印当前模式下的历史最高分
        TCHAR maxscore[5] = { 0 };
        easy_max = max(easy_max,
        snake.score[0]); //若当前得分一直在刷新历史记录,
        历史最高分同步刷新
        _stprintf(maxscore, _T("%d"),
        easy_max);
        outtextxy(760, 240, maxscore);

        //能量槽底色
        setfillcolor(WHITE);
        solidroundrect(600, 110, 800, 130, 3,
        3);

        //显示剩余生命数和能量槽
        TCHAR myLife[5] = { 0 };
        _stprintf(myLife, _T("%d"),
        this->snake.life[0]);
        //依据生命数的多少设置显示颜色
        if (this->snake.life[0] <= 10) {
            setfillcolor(154, 205, 50));
            settextrcolor(205, 51, 51)); //生命
            过少时用红色显示

```

```

        }
        else if (this->snake.life[0] > 10 &&
        this->snake.life[0] <= 50) {
            setfillcolor(255, 130, 71));
            settextrcolor(255, 130, 71));
        } //较为安全时用黄色显示
        else {
            setfillcolor(154, 205, 50));
            settextrcolor(154, 205, 50));
        } //完全安全时用绿色显示

        outtextxy(600, 173, myLife); //打印生命
        数

        if (snake.score[0] <= 660)
            solidroundrect(600, 110, 600 +
            snake.length[0] * 3, 130, 3, 3); //打印能量槽
        else
            solidroundrect(600, 110, 800, 130,
            3, 3);

        //打印小蛇的长度
        settextrcolor(143, 115, 66));
        TCHAR myLen[5] = { 0 };
        _stprintf(myLen, _T("%d"),
        this->snake.length[0]);
        outtextxy(810, 100, myLen);
    }
    else if (snake.mode == MODE::multi) {
        //显示双蛇模式下的时间显示
        settextrcolor(143, 115, 66));
        settextrstyle(30, 0, _T("Arial
BLACK"));
        setbkmode(TRANSPARENT);
        outtextxy(47, 88, myTime);

        //能量槽底色
        setfillcolor(WHITE);
        solidroundrect(540, 85, 740, 105, 3,
        3); //蓝色小蛇的能量槽
        solidroundrect(220, 85, 420, 105, 3,
        3); //红色小蛇的能量槽
        //打印能量槽
        for (int i = 0; i < 2; i++) {
            //依据生命数的多少设置显示颜色
            if (this->snake.life[i] <= 10)
                setfillcolor(205, 51, 51));
            else if (this->snake.life[i] > 10
            && this->snake.life[i] <= 50) //较为安全时用黄
            色显示
                setfillcolor(255, 130,
                71));
            else //完全安全时用绿色显示
                setfillcolor(154, 205,
                50));

            if (i == 1) {
                if (snake.score[1] <= 660)
                    solidroundrect(540, 85, 540
                    + snake.length[1] * 3, 105, 3, 3);
                else
                    solidroundrect(540, 85,
                    740, 105, 3, 3);
            } //打印蓝色小蛇能量槽
            else {
                if (snake.score[0] <= 660)

```



```

        solidroundrect(420 -
snake.length[0] * 3, 85, 420, 105, 3, 3);
        else
            solidroundrect(220, 85,
420, 105, 3, 3);
        } //打印红色小蛇能量槽
    }
    //打印分数比
    settextrstyle(30, 0, _T("Arial
BLACK"));
    outtextxy(475, 80, _T(":"));
    TCHAR score1[5] = { 0 };
    TCHAR score2[5] = { 0 };
    _stprintf(score1, _T("%d"),
this->snake.score[0]);
    _stprintf(score2, _T("%d"),
this->snake.score[1]);
    outtextxy(430, 80, score1);
    outtextxy(485, 80, score2);
}
}
//处理撞墙
void MANAGER::manager_crashwall() {
    int prespeed[2];
    prespeed[0] = snake.speed[0];
    prespeed[1] = snake.speed[1];
    bool alive = true;
    for (int i = 0; i <= snake.mode; i++) {
        alive = true;
        //蛇头坐标
        pos snakepos =
this->snake.snake_pos[i][0];

        //当前页面中的所有墙壁都要遍历判断一次
        for (size_t wallnum = 0; wallnum <
this->wall.wall_pos.size(); wallnum++) {
            pos wallpos =
this->wall.wall_pos[wallnum]; //某一块墙壁的坐标
            int r = snake.radio; //墙壁的厚度等于
蛇的半径
            if (snakepos.x >= wallpos.x - r &&
snakepos.x <= wallpos.x + r
                && snakepos.y >= wallpos.y - r
&& snakepos.y <= wallpos.y + r) {
                //发生了撞墙事件

                if
(this->wall.wall_getQuality(wallnum)) {
                    int prelength =
this->snake.length[i];
                    //撞到了软墙壁
                    HWND h = GetHWND();
                    MessageBox(h, _T("oops!!!蛇
蛇撞到了【软墙壁】!\n 你将回到起点继续本轮游戏
~\n>>按下回车[确定]继续游戏"), _T("GAME-蛇蛇撞墙
提示"), MB_OK);
                    this->snake.snake_reset(i);
                    if (prelength > 5) {
                        snake.snake_increase(i,
prelength - 5);
                    }
                    else if (prelength == 4) {
                        snake.snake_decrease(i,
prelength - 3);
                    }
                }
            }
        }
    }
}

```

23 / 32

```

        else if (prelength == 3) {
            snake.snake_decrease(i,
2);
        }
        break;
    }
    else {
        //撞到了硬墙壁小蛇直接死
        HWND h = GetHWND();
        MessageBox(h, _T("oops!!!蛇
蛇撞到了【硬墙壁】!\n>>很遗憾本轮游戏结束啦~"),
_T("GAME-蛇蛇撞墙提示"), MB_OK);
        this->snake.life[i] = 0; //
生命值降为0
        alive = false;
        this->manager_multi_setwinner(i + 1); //设置胜者
        break;
    }
}
else
    snake.speed[i] = prespeed[i];
}
if (alive = false)
    break;
}
return;
}
//判断当前这局游戏是否还要继续
bool MANAGER::manager_isRunning() {
    switch (this->snake.mode) {
        case MODE::easy:
            return snake.snake_checklife(0);
            break;
        case MODE::multi:
            return snake.snake_checklife(0) &&
snake.snake_checklife(1);
            break;
        default:
            return false;
    }
}
//双人模式下设定胜者
void MANAGER::manager_multi_setwinner(int
index) {
    if (index == 1)
        this->winner = 2;
    else if (index == 2)
        this->winner = 1;
    else
        this->winner = 0;
}
//展示排行榜
void MANAGER::manager_readranking() {
    //以二进制读入方式打开文件
    ifstream ranking;
    ranking.open("rank.txt", ios::binary);
    if (ranking.is_open()) { //确保文件正确打开
        //简单模式
        ranking >> easy_num;
        for (int i = 0; i < easy_num; i++) {
            ranking >> easy_rank[i]; //读入用户
名

            int inlevel = 0;

```

```

        ranking >> inlevel;
        level[0][i] = (Level)inlevel;//读入
游戏难度

        ranking >> score[0][i];//读入分数
    }
    //人机模式
    ranking >> ai_num;
    for (int i = 0; i < ai_num; i++) {
        ranking >> ai_rank[i];//读入用户名

        int inlevel = 0;
        ranking >> inlevel;
        level[1][i] = (Level)inlevel;//读入
游戏难度

        ranking >> score[1][i];//读入分数
    }
    //双人模式
    ranking >> multi_num;
    for (int i = 0; i < multi_num; i++) {
        ranking >> multi_rank[i];//读入用户
名

        int inlevel = 0;
        ranking >> inlevel;
        level[2][i] = (Level)inlevel;//读入
游戏难度

        ranking >> score[2][i];//读入分数
    }
    ranking.close();//关闭文件
}
}
void judge_edition(Level inlevel, TCHAR
myEdition[]) {
    switch (inlevel) {
        case Level::first:
            wcscpy(myEdition, _T("[Level 1]"));
            break;
        case Level::second:
            wcscpy(myEdition, _T("[Level 2]"));
            break;
        case Level::third:
            wcscpy(myEdition, _T("[Level 3]"));
            break;
    }
}
//打印排行榜
void MANAGER::manager_renderRanking() {

    //读取文件
    this->manager_readranking();

    //输出到屏幕端的用户名
    TCHAR show_user[15] = { 0 };
    //输出到屏幕端的版本
    TCHAR show_edition[15] = { 0 };
    //输出到屏幕端的分数
    TCHAR show_score[5] = { 0 };

    //输出简单模式的排行榜

```

```

        for (int i = 0; i < easy_num; i++) {
            //字节向宽字节转换
            //char 类型的用户名转换为宽字节
            MultiByteToWideChar(CP_ACP, 0,
            easy_rank[i], -1, show_user, 15);
            //难度版本
            judge_edition(level[0][i],
            show_edition);

            //分数
            _stprintf(show_score, _T("%d"),
            score[0][i]);

            //输出
            settextrcolor(143, 115, 66));
            settextrstyle(20, 0, _T("Arial
BLACK"));
            setbkmode(TRANSPARENT);
            outtextxy(100, 235 + i * 108,
            show_user);
            outtextxy(100, 270 + i * 108,
            show_edition);
            outtextxy(180, 270 + i * 108,
            show_score);
        }
        //输出人机模式的排行榜
        for (int i = 0; i < ai_num; i++) {
            //字节向宽字节转换
            //char 类型的用户名转换为宽字节
            MultiByteToWideChar(CP_ACP, 0,
            ai_rank[i], -1, show_user, 15);
            //难度版本
            judge_edition(level[1][i],
            show_edition);
            //分数
            _stprintf(show_score, _T("%d"),
            score[1][i]);

            //输出
            settextrcolor(143, 115, 66));
            settextrstyle(20, 0, _T("Arial
BLACK"));
            setbkmode(TRANSPARENT);
            outtextxy(410, 235 + i * 108,
            show_user);
            outtextxy(410, 270 + i * 108,
            show_edition);
            outtextxy(490, 270 + i * 108,
            show_score);
        }
        //输出双人模式的排行榜
        for (int i = 0; i < multi_num; i++) {
            //字节向宽字节转换
            //char 类型的用户名转换为宽字节
            MultiByteToWideChar(CP_ACP, 0,
            multi_rank[i], -1, show_user, 15);
            //难度版本
            judge_edition(level[2][i],
            show_edition);
            //分数
            _stprintf(show_score, _T("%d"),
            score[2][i]);

            //输出

```

```

        settextrcolor(RED);
        settextrstyle(20, 0, _T("Arial
BLACK"));
        setbkmode(TRANSPARENT);
        outtextxy(720, 235 + i * 108,
show_user);
        outtextxy(720, 270 + i * 108,
show_education);
        outtextxy(800, 270 + i * 108,
show_score);
    }
}

```

6.3 GUI 类

6.3.1 myGUI.h

```

#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include<graphics.h>
#include<iostream>
#include<algorithm>
#include<time.h>
#include<functional>
#include<vector>
#include<Windows.h>
#include<string.h>
using namespace std;
#define BORDER_WIDTH 960
#define BORDER_HEIGHT 540

//各种页面的枚举类型
enum PageType {
    menu_page, //主菜单页面
    easy_game_page, //单机-简单模式
    multi_game_page, //双蛇-人机模式 or 双人模式
    rank_page, //排行榜
};

//游戏难度的枚举类型
enum Level {
    first, //入门版
    second, //进阶版
    third //高级版
};

//封装按钮
class BUTTON {
private:
    pair<int, int> pos; //按钮位置坐标
    wstring text; //按钮文本
    pair<int, int> size; //按钮大小
    float scale; //缩放比例, 用于展示鼠标悬停效果
    bool isMouseOver; //鼠标是否处于按钮上方
    void(*onClick)(); //发生鼠标点击的时候执行的函数
public:
    //构造函数
    BUTTON(int x, int y, int width, int
height, const wstring& info, void(*myFun)());

```

```

//绘制按钮
void myGUI_draw_BUTTON();
//检测鼠标是否在按钮上方
void checkMouseOver(int mouseX, int
mouseY);
//检查鼠标点击是否发生在按钮内并执行后序函数
bool checkClick(int mouseX, int mouseY);
};

//封装 1 个页面
class PAGE {
private:
    int page_width; //页面宽度
    int page_height; //页面高度
    IMAGE background; //背景图
    Level level; //游戏难度
    PageType pagetype; //当前页面的类别
    vector<BUTTON> buttons; //当前页面上的所有按钮
    int button_nums;
    ExMessage msg; //本页面的消息
public:
    wstring bkpaper_name;
public:
    //构造函数
    PAGE(int width, int height, PageType
pagetype, wstring paper) : page_width(width),
page_height(height), pagetype(pagetype),
bkpaper_name(paper) {
        level = Level::first;
        msg = { 0 };
        button_nums = 0;
    }
    //初始化当前页面
    void init_page();
    //在当前页面添加按钮信息
    void add_Button();
    //绘制按钮
    void draw();
    //处理鼠标移动
    void mouseMove(int mouseX, int mouseY);
    //处理鼠标点击
    void mouseClicked(int mouseX, int mouseY);
    //运行页面
    void run();
};

```

6.3.2 myGUI.cpp

```

#include "myGUI.h"
#include "page.h"

const int border_line[3] = { 139, 115, 85 };
const int normal_button[3] = { 205, 155, 29 };
const int light_button[3] = { 255, 236, 139 };
#define redsnake 0
#define yellowsnake 1
#define bluesnake 2

void red() {
    game_easy.bkpaper_name = name[0];
}

```

```

    HWND h = GetHwnd();
    if (MessageBox(h, _T("你选择的小蛇皮肤为:
RED"), _T("皮肤颜色选择"), MB_OK)) {
        manager.snake.set_skincolor(0,
redsnake);
        manager.snake.set_skincolor(1,
redsnake);
    }
}
void yellow() {
    game_easy.bkpaper_name = name[1];

    HWND h = GetHwnd();
    if (MessageBox(h, _T("你选择的小蛇皮肤为:
YELLOW"), _T("皮肤颜色选择"), MB_OK)) {
        manager.snake.set_skincolor(0,
yellowsnake);
        manager.snake.set_skincolor(1,
yellowsnake);
    }
}
void blue() {
    game_easy.bkpaper_name = name[2];

    HWND h = GetHwnd();
    if (MessageBox(h, _T("你选择的小蛇皮肤为:
BLUE"), _T("皮肤颜色选择"), MB_OK)) {
        manager.snake.set_skincolor(0,
bluesnake);
        manager.snake.set_skincolor(1,
bluesnake);
    }
}
//暂停处理
void myPause() {
    HWND h = GetHwnd();
    MessageBox(h, _T("你已将游戏暂停, 按下[确定]
键游戏继续"), _T("游戏暂停"), MB_OK);
}
//简易模式的AI蛇
void myeasyAI() {
    if (manager.snake.cgtoAI == true)
        manager.snake.cgtoAI = false; //离开AI
模式
    else {
        manager.snake.cgtoAI = true;
        manager.snake.speed[0] = 1;
    } //设置进入AI模式
}
//BUTTON类: 构造函数
BUTTON::BUTTON(int x, int y, int width, int
height, const wstring& info, void(*myFun)()) {
    pos.first = x;
    pos.second = y;
    size.first = width;
    size.second = height;
    text = info;
    scale = 1.0f;
    isMouseOver = false;
    onClick = myFun;
}
//BUTTON类: 绘制按钮(分为悬停与不悬停)
void BUTTON::myGUI_draw_BUTTON() {
    //设置按钮的基本信息

```

```

    int width = int(size.first * scale); //缩放
后按钮宽度
    int height = int(size.second * scale); //缩
放后按钮高度
    int x = pos.first + (size.first - width) /
2; //缩放后按钮的X坐标
    int y = pos.second + (size.second -
height) / 2; //缩放后按钮的Y坐标

    //设置边框线条颜色
    setlinecolor(RGB(border_line[0],
border_line[1], border_line[2]));
    //根据鼠标是否悬停设置按钮填充色
    if (isMouseOver)
        setfillcolor(RGB(light_button[0],
light_button[1], light_button[2]));
    else
        setfillcolor(RGB(normal_button[0],
normal_button[1], normal_button[2]));

    setbkmode(TRANSPARENT); //设置文本背景为透明
    fillroundrect(x, y, x + width, y + height,
15, 15);
    settextstyle(25, 0, _T("Arial BLACK"), 0,
0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    settextcolor(WHITE);

    //将文字绘制在矩形中心
    int tx = pos.first + (size.first -
textwidth(text.c_str())) / 2;
    int ty = pos.second + (size.second -
textheight(text.c_str())) / 2;
    //打印出文本
    outtextxy(tx, ty, text.c_str());
}
//BUTTON类: 检查鼠标悬停于按钮上方
void BUTTON::checkMouseOver(int mouseX, int
mouseY) {
    isMouseOver = ((mouseX >= pos.first &&
mouseX <= pos.first + size.first)
&& (mouseY >= pos.second && mouseY <=
pos.second + size.second));

    if (isMouseOver) //鼠标悬停在上方时尺寸缩小
        scale = 0.9f;
    else //不悬停时保持正常尺寸
        scale = 1.0f;
}
//BUTTON类: 检查鼠标在按钮内部发生点击时间
bool BUTTON::checkClick(int mouseX, int
mouseY) {
    if ((mouseX >= pos.first && mouseX <=
pos.first + size.first)
&& (mouseY >= pos.second && mouseY <=
pos.second + size.second)) {
        isMouseOver = true;
        onClick(); //执行对应功能指针所指向的函数
        //BUTTON的各项属性恢复到悬停前
        isMouseOver = false;
        scale = 1.0f;
        return true;
    }
}

```

```

    return false;
}

//PAGE 类: 初始化页面绘制
void PAGE::init_page() {
    initgraph(this->page_width,
this->page_height, SHOWCONSOLE);
}

//PAGE 类: 添加按钮信息并绘制
void PAGE::add_Button() {
    switch (this->pagetype) {
    case menu_page:
    {
        BUTTON red_snake(665, 365, 50, 30,
_T("YES"), red);
        this->buttons.push_back(red_snake);
        button_nums++;

        BUTTON yellow_snake(775, 365, 50, 30,
_T("YES"), yellow);
        this->buttons.push_back(yellow_snake);
        button_nums++;

        BUTTON blue_snake(885, 365, 50, 30,
_T("YES"), blue);
        this->buttons.push_back(blue_snake);
        button_nums++;

        BUTTON easy_mode(630, 440, 85, 35,
_T("单人模式"), myGame_easy);
        this->buttons.push_back(easy_mode);
        button_nums++;

        BUTTON hard_mode(730, 440, 85, 35,
_T("人机模式"), myGame_ai);
        this->buttons.push_back(hard_mode);
        button_nums++;

        BUTTON player_mode(830, 440, 85, 35,
_T("双人模式"), myGame_multi);
        this->buttons.push_back(player_mode);
        button_nums++;

        BUTTON ranking(740, 490, 70, 35, _T("排
行榜"), myRank);//待修改
        this->buttons.push_back(ranking);
        button_nums++;

        BUTTON helper(830, 490, 85, 35,
_T("HELP"), myHelp);
        this->buttons.push_back(helper);
        button_nums++;

        BUTTON quit(850, 20, 85, 35, _T("点此退
出"), quitMenu);
        this->buttons.push_back(quit);
        button_nums++;
        break;
    }
    case easy_game_page:
    {
        BUTTON rule(580, 348, 85, 30, _T("点击
查看"), myRule);
        this->buttons.push_back(rule);
    }
    }
}

```

```

        button_nums++;

        BUTTON pause(580, 422, 85, 30, _T("点击
暂停"), myPause);
        this->buttons.push_back(pause);
        button_nums++;

        BUTTON ai(580, 493, 85, 30, _T("点击切
换"), myeasyAI);//进入 AI 模式
        this->buttons.push_back(ai);
        button_nums++;

        BUTTON back(20, 15, 95, 30, _T("<返回主
页"), myMenu);
        this->buttons.push_back(back);
        button_nums++;
        break;
    }
    case multi_game_page: {
        BUTTON pause(130, 15, 85, 30, _T("点击
暂停"), myPause);
        this->buttons.push_back(pause);
        button_nums++;

        BUTTON back(20, 15, 95, 30, _T("<返回主
页"), myMenu);
        this->buttons.push_back(back);
        button_nums++;
        break;
    }
    case rank_page: {
        BUTTON back(20, 15, 95, 30, _T("<返回主
页"), myMenu);
        this->buttons.push_back(back);
        button_nums++;
        break;
    }
}

void PAGE::draw() {
    //加载背景图
    loadimage(&background,
bkpaper_name.c_str(), this->page_width,
this->page_height);
    putimage(0, 0, &background);
    //绘制按钮
    for (int index = 0; index < button_nums;
index++)
        buttons[index].myGUI_draw_BUTTON();
}

//处理鼠标单击事件
void PAGE::mouseClick(int mouseX, int mouseY)
{
    for (int index = 0; index < button_nums;
index++) {
        if (buttons[index].checkClick(mouseX,
mouseY))
            break;
    }
}

//处理鼠标移动事件
void PAGE::mouseMove(int mouseX, int mouseY) {
    for (int index = 0; index < button_nums;
index++) {

```



```

        buttons[index].checkMouseOver(mouseX,
mouseY);
    }
}

void PAGE::run() {
    BeginBatchDraw();//开始批量绘制
    while (true) {
        if (peekmessage(&msg, EM_KEY |
EM_MOUSE)) {
            switch (msg.message) {
                case WM_KEYDOWN://键盘按下事件
                    if (manager.snake.cgtoAI ==
false && manager.snake.mode == MODE::easy ||
manager.snake.mode == MODE::multi)
                        manager.manager_event(msg.v
kcode);
                    break;
                case WM_LBUTTONDOWN://鼠标左键按下事
件
                    mouseClicked(msg.x, msg.y);//处理
鼠标单击
                    break;
                case WM_MOUSEMOVE://鼠标移动事件
                    mouseMove(msg.x, msg.y);//处理
鼠标移动事件
                    break;
            }
        }
        draw();
        if (this->pagetype ==
PageType::easy_game_page || this->pagetype ==
PageType::multi_game_page) {
            if (manager.manager_isRunning()) {
                manager.manager_crashwall();
                manager.manager_eatfood();
                manager.manager_update();
                manager.manager_render();
            }
            else {
                manager.snake.way_to_exit();
            }
        }
        else if (this->pagetype ==
PageType::rank_page) {
            manager.manager_renderRanking();
        }
        FlushBatchDraw();//将缓冲区内容显示在屏
幕上
    }
    EndBatchDraw();//结束批量绘制
}

```

6.4 页面类

6.4.1 page.h

```

#pragma once
#include"manager.h"

//弹窗提示和各页面运行函数的声明
void myMenu();
void myGame_easy();
void myGame_multi();
void myGame_ai();
void myHelp();
void myRule();
void myRank();
void quitMenu();

extern PAGE menu;
extern PAGE game_easy;
extern PAGE game_multi;
extern PAGE game_ai;
extern PAGE ranking;

extern MANAGER manager;
extern wstring name[4];

```

6.4.2 page.cpp

```

#include"myGUI.h"
#include"manager.h"

extern MANAGER manager;

wstring name[4] =
{ _T("bk_easy_red.png"),_T("bk_easy_yellow.png"),
_T("bk_easy_blue.png"),_T("bk_multi.png") }
;
PAGE menu(960, 540, menu_page,
_T("bk_paper.png"));
PAGE game_easy(960, 540, easy_game_page,
name[0]);
PAGE game_multi(960, 540, multi_game_page,
name[3]);
PAGE game_ai(960, 540, multi_game_page,
_T("bk_ai.png"));
PAGE ranking(960, 540, rank_page,
_T("bk_rank.png"));

//菜单界面
void myMenu() {

    bool ai = manager.snake.cgtoAI;
    bool notset = (manager.snake.mode ==
MODE::multi) && ai;

    //回到菜单重置双人模式的Ai 属性
    manager.snake.cgtoAI = false;
    //对当前地图作清空处理
}

```

```

    manager.food.food_set(manager.snake,
manager.wall);
manager.wall.wall_reset();
for (int i = 0; i < manager.snake.mode;
i++)
    manager.snake.snake_reset(i);

//弹窗提示
HWND h = GetHwnd();
if (MessageBox(h, _T("你即将返回主页!
\n>>[确定]-返回主页 [取消]-保留在当前页面"),
_T("GAME-游戏提示"), MB_OKCANCEL) == IDOK) {
    //查看排行榜的标记量设为 false
    manager.checkRank = false;
    menu.draw();
    menu.run();
}
else {
    if (notset)
        manager.snake.cgtoAI = ai;
}
}
//一条蛇的界面
void myGame_easy() {
    manager.snake.mode = easy;
    manager.snake.crashtimes[0] =
manager.snake.crashtimes[1] = 0;
    wchar_t level[2] = { 0 };
    InputBox(level, 2, _T("欢迎来到蛇蛇的世界!
\n[GUIDE]请输入游戏难度: \n [1]-入门版\n [2]-
进阶版\n [3]-高级版"), _T("简单模式-难度选择"));
    //输入单机模式下的用户名
    wchar_t user[15] = { 0 };
    InputBox(user, 15, _T("在蛇蛇的世界留下你的
名字吧! \n>>请输入本轮游戏你要使用的用户名:\n>>蛇
蛇只认识【英文】, 请不要超过 15 个字符~"), _T("简单
模式-用户名输入"));
    wcscpy(manager.user1, user);
    //设置游戏难度
    if (level[0] == '1')
        manager.snake.level = Level::first;
    else if (level[0] == '2')
        manager.snake.level = Level::second;
    else if (level[0] == '3')
        manager.snake.level = Level::third;
    //绘制界面
    game_multi.draw();
    //重置游戏时间
    manager.manager_reset_time();

    game_easy.run();
}
//双人对战界面
void myGame_multi() {
    //基本信息设置
    manager.snake.mode = multi;
    manager.snake.crashtimes[0] =
manager.snake.crashtimes[1] = 0;

    //输入双人模式下的用户名
    wchar_t user1[16] = { 0 }, user2[16] =
{ 0 };

```

```

    InputBox(user1, 16, _T("在蛇蛇的世界留下你的
名字吧! \n>>请输入本轮游戏【Player1】的用户
名:\n>>蛇蛇只认识【英文】, 请不要超过 15 个字符~"),
_T("双人模式-用户名输入"));
    wcscpy(manager.user1, user1);
    InputBox(user2, 16, _T("在蛇蛇的世界留下你的
名字吧! \n>>请输入本轮游戏【Player2】的用户
名:\n>>蛇蛇只认识【英文】, 请不要超过 15 个字符~"),
_T("双人模式-用户名输入"));
    wcscpy(manager.user2, user2);

    //设置游戏难度
    wchar_t level[2] = { 0 };
    InputBox(level, 2, _T("BATTLE 升级! 你们要挑
战哪种难度? \n[GUIDE]请输入游戏难度: \n [1]-入门
版\n [2]-进阶版\n [3]-高级版"), _T("双人模式-
难度选择"));
    if (level[0] == '1')
        manager.snake.level = Level::first;
    else if (level[0] == '2')
        manager.snake.level = Level::second;
    else if (level[0] == '3')
        manager.snake.level = Level::third;

    //绘制界面
    game_multi.draw();

    //重置游戏时间
    manager.manager_reset_time();
    game_multi.run();
}
//人机对战界面
void myGame_ai() {
    //基本信息设置
    manager.snake.mode = multi;
    manager.snake.crashtimes[0] =
manager.snake.crashtimes[1] = 0;
    //设置 AI 控制信息
    manager.snake.cgtoAI = true;
    //输入人机对战模式下的用户名
    wchar_t user1[15] = { 0 };
    InputBox(user1, 15, _T("在蛇蛇的世界留下你的
名字吧! \n>>请输入本轮游戏【Player】的用户名:\n>>
蛇蛇只认识【英文】, 请不要超过 15 个字符~"), _T("人
机模式-用户名输入"));
    wcscpy(manager.user1, user1);
    wcscpy(manager.user2, _T("AI-Player"));

    //设置游戏难度
    wchar_t level[2] = { 0 };
    InputBox(level, 2, _T("BATTLE 升级! 想要挑战
哪种难度? \n[GUIDE]请输入游戏难度: \n [1]-入门版
\n [2]-进阶版\n [3]-高级版"), _T("人机模式-难
度选择"));
    if (level[0] == '1')
        manager.snake.level = Level::first;
    else if (level[0] == '2')
        manager.snake.level = Level::second;
    else if (level[0] == '3')
        manager.snake.level = Level::third;

    //绘制界面
    game_ai.draw();

```

```

//重置游戏时间
manager.manager_reset_time();
game_ai.run();
}
//帮助界面
void myHelp() {
    //帮助界面的弹窗
    HWND h = GetHwnd();
    MessageBox(h, _T("【单机模式】:\n>>刚入蛇
    门? 从简单模式开始吧! [1]入门版 [2]进阶版 [3]高级
    版\n【人机模式】:\n>>想要挑战计算机的算力? 来人机
    模式和 AI_snake 一起 battle 吧~\n【双人模式】:\n>>
    想和室友组团开吃? 来试试双人模式! \n\n>>你可以在
    【排行榜】中查看你的过往历史成绩哦! "), _T("HELP
    详情"), MB_OK);
}
//退出菜单界面
void quitMenu() {
    closegraph();
    exit(0);
}
//规则界面
void myRule() {
    //规则界面的弹窗
    HWND h = GetHwnd();
    if (manager.snake.level == Level::first)
        MessageBox(h, _T("【入门版】游戏规则:
        \n>>使用【方向键】控制小蛇移动\n>>按下【空格】小蛇
        加速, 再次按下【空格】恢复原速\n>>【彩色】果实加
        分, 【灰色】果实失分\n>>每次只有将所有【彩色】果实
        吃光, 才会产生新果实哦! \n>>小心控制蛇蛇, 一旦【撞
        墙】游戏将结束! \n>>当你的【能量槽】或【分数】为 0
        时, 游戏也将结束! "), _T("GAME-游戏规则"),
        MB_OK);
    else if (manager.snake.level ==
        Level::second)
        MessageBox(h, _T("【进阶版】游戏规则:
        \n>>使用【方向键】控制小蛇移动\n>>按下【空格】小蛇
        加速, 再次按下【空格】恢复原速\n>>【彩色】果实加
        分, 【灰色】果实失分\n>>每次只有将所有【彩色】果实
        吃光, 才会产生新果实哦! \n>>小心控制蛇蛇, 一旦【撞
        墙】蛇身将变成【墙壁】, 并回到初始位置重新开始!
        \n>>【浅灰色】为【软墙壁】, 会导致你失去一定分数!
        \n>>【深灰色】为【硬墙壁】, 会直接结束游戏! \n>>当
        你的【能量槽】或【分数】为 0 时, 游戏也将结束! "),
        _T("GAME-游戏规则"), MB_OK);
    else if (manager.snake.level ==
        Level::third)
        MessageBox(h, _T("【高级版】游戏规则:
        \n>>使用【方向键】控制小蛇移动\n>>按下【空格】小蛇
        加速, 再次按下【空格】恢复原速\n>>【彩色】果实加
        分, 【灰色】果实失分\n>>每次只有将所有【彩色】果实
        吃光, 才会产生新果实哦! \n>>小心控制蛇蛇, 一旦【撞
        墙】蛇身将变成【果实】, 并回到初始位置重新开始!
        \n>>当撞墙次数超过【5次】, 游戏将结束! \n>>当你的
        【能量槽】或【分数】为 0 时, 游戏也将结束! "),
        _T("GAME-游戏规则"), MB_OK);
}
//排行榜界面
void myRank() {
    manager.checkRank = true;

```

```

ranking.draw();
ranking.run();
}

```

6.5 文件记录

6.5.1 Record.h

```

#pragma once
#include<fstream>
enum MODE;
enum Level;
extern char easy_rank[3][15];
extern char ai_rank[3][15];
extern char multi_rank[3][15];
extern int easy_num, ai_num, multi_num;
extern int score[3][3];
extern Level level[3][3];

void update_rank(int newscore, char
newname[15], Level newlevel,int choice);
void write_rank();

extern int easy_max;
extern int ai_max;
extern int multi_max;

```

6.5.2 Record.cpp

```

#define _CRT_SECURE_NO_WARNINGS
//保存记录
#include<fstream>
#include"item.h"

//记录所有游戏记录的文件流
fstream record("record.txt", ios::binary |
ios::app);
//4 个模式下的历史最高分
int easy_max = 0;
int ai_max = 0;
int multi_max = 0;
//写入历史记录到文件中
void write_record(PageType pagetype, MODE
mode, Level level, char user1[], int score1,
int score2 = 0, char user2[] =
(char*)"AI_snake") {
    if (mode == MODE::easy) { //单机模式
        //输出用户名
        record << "[用户名]:" << user1 << " [版
        本]:";
        if (pagetype ==
        PageType::easy_game_page) { //简单模式
            switch (level) {
                case Level::first:
                    record << "单人模式-入门版 ";
                    break;
                case Level::second:
                    record << "单人模式-进阶版 ";

```

```

        break;
    case Level::third:
        record << "单人模式-高级版 ";
        break;
    }
    //更新历史记录
    easy_max = max(easy_max, score1);
}
record << "[得分]:" << score1 << endl;
}
else if (mode == MODE::multi) { //双人模式或
AI 模式
    if (user2 != "AI_snake") { //双人模式
        record << "[用户名 1]:" << user1 <<
" [版本]:双人模式 [得分]:" << score1 << endl;
        record << "[用户名 2]:" << user2 <<
" [版本]:双人模式 [得分]:" << score2 << endl;
        //更新历史记录
        score1 = max(score1, score2);
        multi_max = max(score1, multi_max);
    }
    else { //AI 人机对战模式
        record << "[用户名 1]:" << user1 <<
" [版本]:AI 模式-人机对战 [得分]:" << score1 <<
endl;
        record << "[用户名 2]:AI_snake [版
本]:AI 模式-人机对战 [得分]:" << score2 << endl;
        //更新历史记录
        ai_max = max(ai_max, score1);
    }
}
}

//记录排行榜的文件流
char easy_rank[3][15] = { 0 }; //简单模式下的前 3
名
char ai_rank[3][15] = { 0 }; //ai 模式下的前 3 名
char multi_rank[3][15] = { 0 }; //双人模式下的前
三名
int easy_num = 0, ai_num = 0, multi_num = 0; //
记录当前在榜上的各个模式下的玩家个数, 便于读取文件
时的操作控制
int score[3][3] = { 0 }; //3 种模式下对应前三名的
分数
Level level[3][3];
//将各种模式下的前三名写入 rank.txt
/*每种模式下, 排行榜文件都按照
* num
* username1 level1 score1
* username2 level2 score2
* username3 level3 score3 的格式存储
* AI 人机对战模式不用写 level
*/
//排序函数
void sort_rank(char name[][15], int score[],
Level level[]) {
    for (int i = 0; i < 3; i++) {
        for (int j = i + 1; j < 3; j++) {
            if (score[j] > score[i]) {

```

```

                swap(score[i], score[j]); //交换
分数
                swap(level[i], level[j]); //交换
游戏模式

                char tmp[15] = { 0 };
                strcpy(tmp, name[i]);
                strcpy(name[i], name[j]);
                strcpy(name[j], tmp); //交换用户
名
            }
        }
    }
}
//文件写入函数
void write_rank() {
    ofstream ranking("rank.txt", ios::binary);

    if (ranking.is_open()) { //判断文件是否成功打
开

        //先对得到的数组进行排序
        sort_rank(easy_rank, score[0],
level[0]); //单机
        sort_rank(ai_rank, score[1],
level[1]); //人机
        sort_rank(multi_rank, score[2],
level[2]); //双人

        //简单模式
        ranking << ((easy_num > 3) ? 3 :
easy_num) << endl;
        for (int i = 0; i < easy_num; i++) {
            //打印用户名
            ranking << easy_rank[i] << ' ';
            ranking << (int)level[0][i] << ' '
<< score[0][i] << endl;
        }
        //人机模式
        ranking << ((ai_num > 3) ? 3 : ai_num)
<< endl;
        for (int i = 0; i < ai_num; i++) {
            //打印用户名
            ranking << ai_rank[i] << ' ';
            ranking << (int)level[1][i] << ' '
<< score[1][i] << endl;
        }
        //双人模式
        ranking << ((multi_num > 3) ? 3 :
multi_num) << endl;
        for (int i = 0; i < multi_num; i++) {
            //打印用户名
            ranking << multi_rank[i] << ' ';
            ranking << (int)level[2][i] << ' '
<< score[2][i] << endl;
        }
        ranking.close(); //关闭文件
    }
}
//依据新成绩的值做插入
void insert_rank(char name[][15], int score[],
Level level[], char newname[], Level newlevel,
int newscore) {

```

```

        for (int i = 0; i < 3; i++) {
            if (score[i] <= newscore) { //找到正确的
插入位置
                //从 i 开始的所有元素后移 1 个
                for (int j = 2; j > i; j--) {
                    score[j] = score[j - 1];
                    level[j] = level[j - 1];
                    strcpy(name[j], name[j - 1]);
                }
                //在第 i 个位置插入新分数
                score[i] = newscore;
                level[i] = newlevel;
                strcpy(name[i], newname);
                //插入结束后跳出即可
                break;
            }
        }
    }

//比对新产生的结果, 判断是否要更新
void update_rank(int newscore, char
newname[15], Level newlevel, int choice) {
    //更新
    //数组是降序排列的

    if (choice == 1) { //简单模式的更新
        if (newscore >= score[0][2])
            insert_rank(easy_rank, score[0],
level[0], newname, newlevel, newscore);
    }
    else if (choice == 2) { //AI 模式的更新
        if (newscore >= score[1][2])
            insert_rank(ai_rank, score[1],
level[1], newname, newlevel, newscore);
    }
    else if (choice == 3) { //双人模式的更新
        if (newscore >= score[2][2])
            insert_rank(multi_rank, score[2],
level[2], newname, newlevel, newscore);
    }
}
}

```

```

game_ai.add_Button();
ranking.add_Button();
//绘制菜单界面
menu.draw();
//主页面的弹窗
HWND h = GetHwnd();
MessageBox(h, _T("欢迎来到蛇蛇的世界! \n 若你
选择【单机模式】:\n>>可以选择你喜欢的小蛇【皮
肤】! \n 若你选择【人机模式】或【双人模式】:\n>>程
序媛姐姐还没有为你开设自选皮肤的权限~\n[HELP]:更多
帮助请在【HELP】键查看~"), _T("菜单页-NOTICE"),
MB_OK);
//进入菜单界面开始运行
menu.run();
}

```

6.6 主函数

6.6.1 Snake_main.cpp

```

#include "myGUI.h"
#include "page.h"

MANAGER manager;

int main() {

    //随机生成函数的种子
    srand((unsigned int)time(NULL));
    //各页面 GUI 的初始化
    menu.init_page();
    menu.add_Button();
    game_easy.add_Button();
    game_multi.add_Button();
}

```