同济大学计算机系

OOP 贪吃蛇大作业实验报告

学　　号　2152118

姓　　名　史君宝

专　　业　计算机科学与技术（计科 1 班）

完成时间　2023.01.07

一、设计思路与功能描述

1. 得分点

实现了基本的贪吃蛇游戏的功能，同时完成了入门版、进阶版和高级版中的全部功能。

但是仍有部分功能并未实现，

2. 设计思路

（1）使用之前使用过的 EasyX 库中的基本工具进行 UI 设计

在之前的大作业中，我们使用了 EasyX 库来画图，在这个问题中，我们仍然使用它来作为我们得力趁手的工具。但是之前有很多抗锯齿的算法使用，但是这次赶 DDL 了，并未使用，UI 奇丑无比。

（2）使用面向对象的程序设计方法

在整个项目中，我们使用了类来描述过程中的元素，比如食物，蛇，墙等等。这种设计方法实现的封装型好，体现了面向对象的程序设计方法。

3. 功能描述

（1）类 Food

```
//我们学过的数据结构中链表比较适合储存。
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Food_point
{
    int Food_x;
    int Food_y;

    Food_point *last_point, *next_point;
};

//由食物组成的链表
class Food_list {
private:
    Food_point* List_first;
    Food_point* List_last;
    int Food_num;

public:
    Food_list(Game &G);
    Food_list(istream& in, Game& G);
    ~Food_list();
```

```
    Food_point* Get_first();
    Food_point* Get_last();
    int Get_num();

    void new_food(Game& G);
    void delete_food();
    void check_food(Game& G);

    void Set_num(int new_num)
    {
        Food_num = new_num;
    }
};
```

在最初的设计思想中，我们是想将所有的食物构成一个小的链表，这样我们在之后的保存数据，恢复游戏部分就能够轻松点。

但是最终是没有时间完成这一步了，所以里面有部分函数已经用处不大了，之后会尽力去完善代码。

重点函数：

New_food() 这个函数会随机生成 1-5 个食物。

delete_food() 这个函数会在我们吃掉食物时，将食物总量减一

Check_food() 这个函数会在我们吃掉食物时，检查食物总量，决定是否再次生成新食物

（2）类 Wall

```
//我们学过的数据结构中链表比较适合储存。
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Wall_point
{
    int Wall_x;
    int Wall_y;

    Wall_point* last_point, * next_point;
};

//由食物组成的链表
class Wall_list {
private:
    Wall_point* List_first;
    Wall_point* List_last;
    int Wall_num;

public:
    Wall_list(Game& G);
    Wall_list(istream& in, Game& G);
    ~Wall_list();
```

```
Wall_point* Get_first();
Wall_point* Get_last();
int Get_num();
```

与上面一样，这个类中也有函数并没有用到，因为最后没有保存数据，链表也就没必要创建了。

（3）类 Snake：

```
//我们学过的数据结构中链表比较适合储存。
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Snake_point
{
    int Snake_x;
    int Snake_y;

    Snake_point* last_point, * next_point;
};

//由食物组成的链表
class Snake_list {
private:
    Snake_point* List_first;
    Snake_point* List_last;

    int tail_x;
    int tail_y;

    int head_dir;
    int Snake_num;

public:
    friend class Wall_list;
    friend class Food_list;

    Snake_list(Game& G);
    Snake_list(istream& in, Game& G);
    ~Snake_list();

    void get_tail();


    Snake_point* Get_first();
    Snake_point* Get_last();

    int Get_dir();
    int Get_num();

    void new_snake(int x, int y);
```

```
    void move(int new_dir);

    void new_list(Game& G);

    int get_tail_x(){ ... }

    int get_tail_y(){ ... }

    void update_tail(){ ... }

    bool Snake_wall(bool flag, Game& G);
};
```

重点函数：

new_Snake(int x, int y) 向蛇的链表中添加新节点。

Move(int new_dir) 蛇的移动函数，传入的是新方向。

New_list(Game &G) 在后面的关卡中，会涉及到蛇的复活，所以这个函数是用来在空白位置获得一个新小蛇

Update_tail() 我们在吃的过程中，如果吃到食物，会变长，没吃到就不会。这个函数如果没吃到就更新尾部新位置，如果吃到了，仍然使用这个。

Snake_wall(bool flag) 在后面的任务中我们需要将蛇身变成墙或者食物，这个函数传入 flag=0 变墙，flag = 1 变食物。

（4）类 Game：

```
class Game {
private:
    int Lives;
    int score;
    int Game_type;

public:
    int Game_map[50][50];
    int Blank_num = 0;

public:
    Game(int Game_type);

    int Get_Lives();
    int Get_Score();
    int Get_game_type();
    void set_game_type(int Game_type);
```

```
void update_map(int x, int y, int update_type);
void init_map();

void Game_UI(int Game_type);

void show_data();

bool check_over();

void Save_data();
```

```
void Save_data();

void Dead()
{
    --Lives;
}

void Set_Score(int new_Score)
{
    score = new_Score;
}
```

重要函数：

Update_map() 这个函数是对游戏过程中的地图进行实时更新，帮助我们确定某个位置的物体种类。

Game_UI() 三种游戏模式对应不同的 UI，这个里面实现对应的 UI。

check_over()这个函数会检查小蛇死亡次数到达 5 次没有，进行一个统计。

（5）Menu.h：

```
#pragma once
#include <iostream>
using namespace std;

void init();

int Menu_main();
```

这个函数实现一个菜单功能，分别是初始化和具体的执行。

（6）main.cpp：

```
int is_over = 1;

#define Wall_over 2
#define Snake_over 3
#define ERROR 4

int Get_start() { ... }

int Get_key(double max_time) { ... }

int Game_start(Game& G, Food_list& F, Wall_list& W, Snake_list& S, int Game_type) { ... }

int main() { ... }
```

这个 CPP 中，Get_start()专用于从键盘中获取开始键 空格
Get_key() 专用于获取键盘中获取键 上下左右 回车
Game_start()是游戏的基本逻辑

二、问题与解决方法

1. 遇到的问题

整体上项目难度挺大，我看加分项中也涉及了很多，但是我已经没有时间完成了。主要谈谈我过程中遇到的一些问题
（1）数据转换的问题。
（2）位移符的优先级问题。

2. 解决的方法

（1）数据转换的问题。

在过程中我们会使用到 outtextxy(840, 650 - 20, _T("游戏错误"));
其中的最后一个参数的类型是 LPCTSTR，跟我们常用的 string 等等并不相同，因此需要进行一个数据的转换。

（2）头文件和函数封装习惯。

在过程中我们定义了很多类，我还想过将 Game 用来调用其他类，但是报错了两个 LINK2005 和 LINK2019。中间迟迟得不到解决，应该是写头文件和函数的封装的习惯并不好，这一点我们之后会继续探讨

三、心得体会

这次大作业总体来说比较复杂，我也因为最近一直在忙不同科目的事情，迟迟未做这个大作业，就连小作业也是 OOP 考试结束之后才做的。

这个项目的逻辑并不复杂，但是毕竟是一个项目，将其创建出来并完成保证没有 bug 还是一个系统的事情。自己赶了 DDL，然后现在做出了一个像屎山一样的工程，也是无语了。但是这个 DDL 是不能拖延的，硬着头皮也只能交上去。

希望各位助教手下留情，能让我混点分，确实做的十分不好。

四、源代码

```
#pragma once
#include <iostream>
using namespace std;
#include "Game.h"
#include <graphics.h>

//我们学过的数据结构中链表比较适合储存。
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Food_point
{
    int Food_x;
    int Food_y;

    Food_point *last_point, *next_point;
};

//由食物组成的链表
class Food_list {
private:
    Food_point* List_first;
    Food_point* List_last;
    int Food_num;

public:
    Food_list(Game &G);
    Food_list(istream& in, Game& G);
    ~Food_list();


    Food_point* Get_first();
    Food_point* Get_last();
    int Get_num();
```

```
    void new_food(Game& G);
    void delete_food();
    void check_food(Game& G);

    void Set_num(int new_num)
    {
        Food_num = new_num;
    }
};

#pragma once

#define Easy_type 1
#define Middle_type 2
#define Difficult_type 3
#define Last_Game 4
#define Read_Record 5


#define Blank_type 0
#define Food_type 1
#define Wall_type 2
#define Snake_type 3

class Game {
private:
    int Lives;
    int score;
    int Game_type;

public:
    int Game_map[50][50];
    int Blank_num = 0;

public:
    Game(int Game_type);

    int Get_Lives();
    int Get_Score();
    int Get_game_type();
    void set_game_type(int Game_type);

    void update_map(int x, int y, int update_type);
    void init_map();
```

```
    void Game_UI(int Game_type);

    void show_data();

    bool check_over();

    void Save_data();

    void Dead()
    {
        --Lives;
    }

    void Set_Score(int new_Score)
    {
        score = new_Score;
    }

};

#pragma once

#define Easy_type 1
#define Middle_type 2
#define Difficult_type 3
#define Last_Game 4
#define Read_Record 5


#define Blank_type 0
#define Food_type 1
#define Wall_type 2
#define Snake_type 3

class Game {
private:
    int Lives;
    int score;
    int Game_type;

public:
    int Game_map[50][50];
    int Blank_num = 0;
```

```
public:
    Game(int Game_type);

    int Get_Lives();
    int Get_Score();
    int Get_game_type();
    void set_game_type(int Game_type);

    void update_map(int x, int y, int update_type);
    void init_map();

    void Game_UI(int Game_type);

    void show_data();

    bool check_over();

    void Save_data();

    void Dead()
    {
        --Lives;
    }

    void Set_Score(int new_Score)
    {
        score = new_Score;
    }

};

#pragma once

#include"Game.h"
#include <iostream>
using namespace std;

#define up 1
#define down 2
#define left 3
#define right 4

//我们学过的数据结构中链表比较适合储存。
```

```
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Snake_point
{
    int Snake_x;
    int Snake_y;

    Snake_point* last_point, * next_point;
};

//由食物组成的链表
class Snake_list {
private:
    Snake_point* List_first;
    Snake_point* List_last;

    int tail_x;
    int tail_y;

    int head_dir;
    int Snake_num;

public:
    friend class Wall_list;
    friend class Food_list;

    Snake_list(Game& G);
    Snake_list(istream& in, Game& G);
    ~Snake_list();

    void get_tail();


    Snake_point* Get_first();
    Snake_point* Get_last();

    int Get_dir();
    int Get_num();

    void new_snake(int x, int y);

    void move(int new_dir);

    void new_list(Game& G);
```

```
    int get_tail_x()
    {
        return tail_x;
    }

    int get_tail_y()
    {
        return tail_y;
    }

    void update_tail()
    {
        tail_x = List_last->Snake_x;
        tail_y = List_last->Snake_y;
    }

    bool Snake_wall(bool flag, Game& G);
};
```

```
#pragma once
#include <iostream>
using namespace std;

void init();

int Menu_main();#pragma once
#include <iostream>
#include "Game.h"
using namespace std;


//我们学过的数据结构中链表比较适合储存。
//其中涉及比较多的查找、插入删除的动作，我们采用双向链表。
struct Wall_point
{
    int Wall_x;
    int Wall_y;

    Wall_point* last_point, * next_point;
};

//由食物组成的链表
```

```
class Wall_list {
private:
    Wall_point* List_first;
    Wall_point* List_last;
    int Wall_num;

public:
    Wall_list(Game& G);
    Wall_list(istream& in, Game& G);
    ~Wall_list();


    Wall_point* Get_first();
    Wall_point* Get_last();
    int Get_num();

};


#include "Food.h"
#include <iostream>
using namespace std;

//获取链表头
Food_point* Food_list::Get_first() {
    return List_first;
}

//获取链表尾
Food_point* Food_list::Get_last() {
    return List_last;
}

//获取链表中的食物个数
int Food_list::Get_num() {
    return Food_num;
}

//产生新的食物
void Food_list::new_food(Game &G)
{
    int num = rand() % 5 + 1; //产生的食物数量
    int x = 0, y = 0;
```

```
    for (int i = 0; i < num; ++i)
    {
        while (1)
        {
            x = rand() % 48 + 1;
            y = rand() % 48 + 1;
            if (G.Game_map[x][y] == Blank_type)
            {
                G.Game_map[x][y] = Food_type;
                setlinecolor(RED);
                setfillcolor(RED);
                fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
                break;
                break;
            }
            else
                continue;
        }
        Food_num++;
    }
    return;
}

void Food_list::check_food(Game& G)
{
    if (!Food_num)
    {
        new_food(G);
    }
}


void Food_list::delete_food()
{
    --Food_num;
}

//新游戏的开始
Food_list::Food_list(Game& G)
{
    List_first = NULL;
    List_last = NULL;
    Food_num = 0;
    new_food(G);
```

```
}


//上次游戏的继续
Food_list::Food_list(istream& in, Game& G)
{
    List_first = NULL;
    List_last = NULL;
    new_food(G);
}

//析构函数
Food_list::~Food_list() {
    if (Food_num == 0)
        return;

    Food_point *Help_1, *Help_2;

    for (Help_1 = List_first; Help_1;)
    {
        Help_2 = Help_1;
        Help_1 = Help_1->next_point;
        delete Help_2;
    }
}

#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
using namespace std;
#include <graphics.h>
#include<conio.h>
#include "Game.h"
#include <string>
using namespace std;

Game::Game(int Game_type)
{
    Game_type = Game_type;
    Lives = 5;
    score = 0;
    Blank_num = 50 * 50;
    init_map();
}
```

```
void Game::init_map()
{
    for (int i = 0; i < 50; i++)
    {
        for (int j = 0; j < 50; j++)
        {
            if (i == 0 || i == 49 || j == 0 || j == 49)
                Game_map[i][j] = Wall_type;
            else
                Game_map[i][j] = Blank_type;
        }
    }
    return;
}


void Game::update_map(int x, int y, int update_type)
{
    if (update_type != Blank_type && Game_map[x][y] == Blank_type)
        ++Blank_num;
    else if (update_type == Blank_type && Game_map[x][y] != Blank_type)
        --Blank_num;
    Game_map[x][y] = update_type;

    switch (update_type)
    {
        case Blank_type:
            setlinecolor(WHITE);
            setfillcolor(WHITE);
            fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
            break;
        case Food_type:
            setlinecolor(RED);
            setfillcolor(RED);
            fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
            break;
        case Wall_type:
            setlinecolor(BLACK);
            setfillcolor(BLACK);
            fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
            break;
        case Snake_type:
            setlinecolor(GREEN);
```

```
            setfillcolor(GREEN);
            fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
            break;
        default:
            break;
    }
}

void Game::Game_UI(int Game_type)
{
    cleardevice();
    setbkcolor(WHITE);

    for (int i = 0; i < 50; ++i)
    {
        for (int j = 0; j < 50; ++j)
        {
            switch (Game_map[i][j])
            {
                case Blank_type:
                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(i * 16, j * 16, i * 16 + 15, j * 16 + 15);
                    break;
                case Wall_type:
                    setlinecolor(BLACK);
                    setfillcolor(BLACK);
                    fillrectangle(i * 16, j * 16, i * 16 + 15, j * 16 + 15);
                    break;
                case Snake_type:
                    setlinecolor(GREEN);
                    setfillcolor(GREEN);
                    fillrectangle(i * 16, j * 16, i * 16 + 15, j * 16 + 15);
                    break;
                case Food_type:
                    setlinecolor(RED);
                    setfillcolor(RED);
                    fillrectangle(i * 16, j * 16, i * 16 + 15, j * 16 + 15);
                    break;
                default:
                    break;
            }
        }
    }
```

```
    if (Game_type == Easy_type)
    {
        settextstyle(40, 0, _T("楷体"));
        settextcolor(BLACK);
        outtextxy(840, 50 - 20, _T("游戏模式：入门版")); // 输出文字
        outtextxy(840, 150 - 20, _T("蛇的长度")); // 输出文字

        string num_1 = to_string(3);
        size_t size_1 = num_1.length();
        wchar_t* buffer_1 = new wchar_t[size_1 + 1];
        MultiByteToWideChar(CP_ACP, 0, num_1.c_str(), size_1, buffer_1,
size_1 * sizeof(wchar_t));
        buffer_1[size_1] = 0;
        outtextxy(1050, 150 - 20, buffer_1); // 输出文字
        delete buffer_1;

        outtextxy(840, 250 - 20, _T("当前分数")); // 输出文字

        string num_2 = to_string(0);
        size_t size_2 = num_2.length();
        wchar_t* buffer_2 = new wchar_t[size_2 + 1];
        MultiByteToWideChar(CP_ACP, 0, num_2.c_str(), size_2, buffer_2,
size_2 * sizeof(wchar_t));
        buffer_2[size_2] = 0;
        outtextxy(1050, 250 - 20, buffer_2); // 输出文字
        delete buffer_2;

        outtextxy(840, 350 - 20, _T("历史最高分")); // 输出文字
        outtextxy(840, 450 - 20, _T("空格开始，回车退出"));
    }
    else if (Game_type == Middle_type)
    {
        settextstyle(40, 0, _T("楷体"));
        settextcolor(BLACK);
        outtextxy(840, 50 - 20, _T("游戏模式：进阶版"));
        outtextxy(840, 150 - 20, _T("蛇的长度"));

        string num_1 = to_string(3);
        size_t size_1 = num_1.length();
        wchar_t* buffer_1 = new wchar_t[size_1 + 1];
        MultiByteToWideChar(CP_ACP, 0, num_1.c_str(), size_1, buffer_1,
size_1 * sizeof(wchar_t));
        buffer_1[size_1] = 0;
```

```
        outtextxy(1050, 150 - 20, buffer_1); // 输出文字
        delete buffer_1;

        outtextxy(840, 250 - 20, _T("剩余生命数"));

        string num_2 = to_string(5);
        size_t size_2 = num_2.length();
        wchar_t* buffer_2 = new wchar_t[size_2 + 1];
        MultiByteToWideChar(CP_ACP, 0, num_2.c_str(), size_2, buffer_2,
size_2 * sizeof(wchar_t));
        buffer_2[size_2] = 0;
        outtextxy(1050, 250 - 20, buffer_2); // 输出文字
        delete buffer_2;


        outtextxy(840, 350 - 20, _T("当前分数"));

        string num_3 = to_string(0);
        size_t size_3 = num_3.length();
        wchar_t* buffer_3 = new wchar_t[size_3 + 1];
        MultiByteToWideChar(CP_ACP, 0, num_3.c_str(), size_3, buffer_3,
size_3 * sizeof(wchar_t));
        buffer_3[size_3] = 0;
        outtextxy(1050, 350 - 20, buffer_3); // 输出文字
        delete buffer_3;



        outtextxy(840, 450 - 20, _T("历史最高分"));
        outtextxy(840, 550 - 20, _T("空格开始，回车退出"));
    }
    else if (Game_type == Difficult_type)
    {
        settextstyle(40, 0, _T("楷体"));
        settextcolor(BLACK);
        outtextxy(840, 50 - 20, _T("游戏模式：高级版"));
        outtextxy(840, 150 - 20, _T("蛇的长度"));
        outtextxy(840, 250 - 20, _T("剩余生命数"));
        outtextxy(840, 350 - 20, _T("当前分数"));
        outtextxy(840, 450 - 20, _T("历史最高分"));

        outtextxy(840, 550 - 20, _T("空格开始，回车退出"));
    }
}
```

```
void Game::show_data()
{
    settextstyle(30, 0, _T("楷体"));
    settextcolor(RED);
    outtextxy(800, 0, _T("入门版")); // 输出文字
}

bool Game::check_over()
{
    if (Lives == 0)
        return 1;
    else if (Blank_num < 5)
        return 1;
    else
        return 0;
}


int Game::Get_Lives()
{
    return Lives;
}

int Game::Get_Score()
{
    return score;
}

int Game::Get_game_type()
{
    return Game_type;
}

void Game::set_game_type(int Game_type)
{
    Game_type = Game_type;
}


void Game::Save_data()
{
```

```
}#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

#include <graphics.h>
#include <fstream>

#include <string>
#include <Windows.h>

#include "Food.h"
#include "Game.h"
#include "Menu.h"
#include "Snake.h"
#include "Wall.h"

int is_start = 0;
int is_over = 1;

#define Wall_over 2
#define Snake_over 3
#define ERROR 4

int Get_start()
{
    char black[] = "                    ";     // 为输出填加的尾部空白，用于覆盖输出
    INPUT_RECORD inRec = { 0 };                    // 输入信息结构体，记录输入信息
    DWORD res;
    HANDLE hInput = GetStdHandle(STD_INPUT_HANDLE);       // 获取控制台标准输入句柄
    HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);     // 获取控制台标准输出句柄

    // 开启控制台窗口鼠标输入
    SetConsoleMode(hInput, ENABLE_EXTENDED_FLAGS | ENABLE_WINDOW_INPUT | ENABLE_MOUSE_INPUT);
    while (1) {
        // 获取当前输入信息,采用 WaitForSingleObject 来防止函数阻塞(异步监听)
        if (WaitForSingleObject(hInput, 0) == WAIT_OBJECT_0)
            ReadConsoleInputA(hInput, &inRec, 1, &res);
```

```
        // 将光标移动到左上角，覆盖输出
        COORD zeroPos = { 0,0 };
        SetConsoleCursorPosition(hOutput, zeroPos);

        //没超过 1s，按了下面的键也会退出
        if (inRec.EventType == KEY_EVENT) {
            int key_value = inRec.Event.KeyEvent.uChar.UnicodeChar;
            if (key_value == 32)
                return key_value;
            else
                continue;
        }
    }
    return -1;
}

int Get_key(double max_time)
{
    DWORD startTime = GetTickCount(), endTime;
    DWORD elapsedTime;
    double elapsedTimeInSeconds;

    char black[] = "                ";     // 为输出填加的尾部空白，用于覆盖输出
    INPUT_RECORD inRec = { 0 };                    // 输入信息结构体，记录输入信息
    DWORD res;
    HANDLE hInput = GetStdHandle(STD_INPUT_HANDLE);       // 获取控制台标准输入句柄
    HANDLE hOutput = GetStdHandle(STD_OUTPUT_HANDLE);     // 获取控制台标准输出句柄

    // 开启控制台窗口鼠标输入
    SetConsoleMode(hInput, ENABLE_EXTENDED_FLAGS | ENABLE_WINDOW_INPUT | ENABLE_MOUSE_INPUT);
    while (1) {
        endTime = GetTickCount();
        elapsedTime = endTime - startTime;
        elapsedTimeInSeconds = double(1.0 * double(elapsedTime) / 1000);

        //超过 1s 就直接退出返回-1
        if (elapsedTimeInSeconds >= max_time)
            break;
```

```
        // 获取当前输入信息,采用 WaitForSingleObject 来防止函数阻塞(异步监听)
        if (WaitForSingleObject(hInput, 0) == WAIT_OBJECT_0)
            ReadConsoleInputA(hInput, &inRec, 1, &res);

        // 将光标移动到左上角，覆盖输出
        COORD zeroPos = { 0,0 };
        SetConsoleCursorPosition(hOutput, zeroPos);

        //没超过 1s，按了下面的键也会退出
        if (inRec.EventType == KEY_EVENT) {
            int key_value = inRec.Event.KeyEvent.uChar.UnicodeChar;
            if (key_value == int('w') || key_value == int('s') ||
                key_value == int('a') || key_value == int('d') ||
key_value == 13)
                return key_value;
            else
                continue;
        }
    }
    return -1;
}

int Game_start(Game& G, Food_list& F, Wall_list& W, Snake_list& S, int
Game_type)
{
    int start_key = 0;
    start_key = Get_start();
    while (start_key != 32)
        start_key = Get_start();

    DWORD start_time = GetTickCount(), end_Time;

    //游戏正式开始
    is_start = 1;
    is_over = 0;

    if (Game_type == Easy_type)
    {
        while (is_start == 1)
        {
            int move_key = Get_key(0.5);
            if (move_key == 13)
            {
```

```
            is_start = 0;
            is_over = 1;
            break;
        }
        else {
            if (move_key == int('w'))
                move_key = 1;
            else if (move_key == int('s'))
                move_key = 2;
            else if (move_key == int('a'))
                move_key = 3;
            else if (move_key == int('d'))
                move_key = 4;

            S.move(move_key);
            Snake_point* snake_head = S.Get_first();
            int x = snake_head->Snake_x;
            int y = snake_head->Snake_y;

            int tail_x = S.get_tail_x();
            int tail_y = S.get_tail_y();
            if (G.Game_map[x][y] == Blank_type)
            {
                G.update_map(x, y, Snake_type);
                G.update_map(tail_x, tail_y, Blank_type);
                S.update_tail();
            }
            else if (G.Game_map[x][y] == Food_type)
            {
                G.update_map(x, y, Snake_type);
                S.get_tail();
                F.delete_food();
                F.check_food(G);
                settextstyle(40, 0, _T("楷体"));
                settextcolor(BLACK);

                setlinecolor(WHITE);
                setfillcolor(WHITE);
                fillrectangle(1040, 150 - 30, 1200, 150 + 30); // 输
出文字

                string num_1 = to_string(S.Get_num());
                size_t size_1 = num_1.length();
                wchar_t* buffer_1 = new wchar_t[size_1 + 1];
```

```
                    MultiByteToWideChar(CP_ACP,    0,    num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 150 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 250 - 30, 1200, 250 + 30); // 输
出文字

                    string num_2 = to_string((S.Get_num() - 3) * 5);
                    size_t size_2 = num_2.length();
                    wchar_t* buffer_2 = new wchar_t[size_2 + 1];
                    MultiByteToWideChar(CP_ACP,    0,    num_2.c_str(),
size_2, buffer_2, size_2 * sizeof(wchar_t));
                    buffer_2[size_2] = 0;
                    outtextxy(1050, 250 - 20, buffer_2); // 输出文字
                    delete buffer_2;
                }
                else if (G.Game_map[x][y] == Wall_type)
                {
                    G.Set_Score((S.Get_num() - 3) * 5);
                    G.update_map(tail_x, tail_y, Blank_type);
                    is_start = 0;
                    is_over = Wall_over;
                    S.update_tail();
                }
                else if (G.Game_map[x][y] == Snake_type)
                {
                    G.Set_Score((S.Get_num() - 3) * 5);
                    G.update_map(tail_x, tail_y, Blank_type);
                    is_start = 0;
                    is_over = Snake_over;
                    S.update_tail();
                }
            }
        }
    }
    else if (Game_type == Middle_type) {
        while (!G.check_over())
        {
            int move_key = Get_key(0.5);
            if (move_key == 13)
```

```
            {
                is_start = 0;
                is_over = 1;
                break;
            }
            else {
                if (move_key == int('w'))
                    move_key = 1;
                else if (move_key == int('s'))
                    move_key = 2;
                else if (move_key == int('a'))
                    move_key = 3;
                else if (move_key == int('d'))
                    move_key = 4;

                S.move(move_key);
                Snake_point* snake_head = S.Get_first();
                int x = snake_head->Snake_x;
                int y = snake_head->Snake_y;

                int tail_x = S.get_tail_x();
                int tail_y = S.get_tail_y();
                if (G.Game_map[x][y] == Blank_type)
                {
                    G.update_map(x, y, Snake_type);
                    G.update_map(tail_x, tail_y, Blank_type);
                    S.update_tail();
                }
                else if (G.Game_map[x][y] == Food_type)
                {
                    G.update_map(x, y, Snake_type);
                    S.get_tail();
                    F.delete_food();
                    F.check_food(G);
                    settextstyle(40, 0, _T("楷体"));
                    settextcolor(BLACK);


                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 150 - 30, 1200, 150 + 30); // 输
出文字

                    string num_1 = to_string(S.Get_num());
```

```
                    size_t size_1 = num_1.length();
                    wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 150 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 350 - 30, 1200, 350 + 30); // 输
出文字

                    string num_2 = to_string((S.Get_num() - 3) * 5 +
G.Get_Score());
                    size_t size_2 = num_2.length();
                    wchar_t* buffer_2 = new wchar_t[size_2 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_2.c_str(),
size_2, buffer_2, size_2 * sizeof(wchar_t));
                    buffer_2[size_2] = 0;
                    outtextxy(1050, 350 - 20, buffer_2); // 输出文字
                    delete buffer_2;
                }
                else if (G.Game_map[x][y] == Wall_type)
                {
                    G.Dead();

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 250 - 30, 1200, 250 + 30); // 输
出文字

                    string num_1 = to_string(G.Get_Lives());
                    size_t size_1 = num_1.length();
                    wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 250 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    if (G.Get_Lives() == 0)
                    {
                        G.Set_Score((S.Get_num()   -   3)   *   5   +
```

```
G.Get_Score());
                    S.Snake_wall(0, G);
                    return Wall_over;
                }
                else
                {
                    G.Set_Score((S.Get_num()  -  3)  *  5  +
G.Get_Score());
                    int a = S.Snake_wall(0, G);
                    if (a)
                    {
                        S.new_list(G);
                        setlinecolor(WHITE);
                        setfillcolor(WHITE);
                        fillrectangle(1040, 150 - 30, 1200, 150 +
30); // 输出文字

                        string num_1 = to_string(3);
                        size_t size_1 = num_1.length();
                        wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                        MultiByteToWideChar(CP_ACP,            0,
num_1.c_str(), size_1, buffer_1, size_1 * sizeof(wchar_t));
                        buffer_1[size_1] = 0;
                        outtextxy(1050, 150 - 20, buffer_1); // 输
出文字
                        delete buffer_1;
                    }
                    else
                        return ERROR;
                }
            }
            else if (G.Game_map[x][y] == Snake_type)
            {
                G.Dead();

                setlinecolor(WHITE);
                setfillcolor(WHITE);
                fillrectangle(1040, 250 - 30, 1200, 250 + 30); // 输
出文字

                string num_1 = to_string(G.Get_Lives());
                size_t size_1 = num_1.length();
                wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
```

```
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 250 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    if (G.Get_Lives() == 0)
                    {
                        G.Set_Score((S.Get_num()   -   3)   *   5   +
G.Get_Score());
                        S.Snake_wall(0, G);
                        return Snake_over;
                    }
                    else
                    {
                        G.Set_Score((S.Get_num()   -   3)   *   5   +
G.Get_Score());
                        int a = S.Snake_wall(0, G);
                        if (a)
                        {
                            S.new_list(G);
                            setlinecolor(WHITE);
                            setfillcolor(WHITE);
                            fillrectangle(1040, 150 - 30, 1200, 150 +
30); // 输出文字

                            string num_1 = to_string(3);
                            size_t size_1 = num_1.length();
                            wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                            MultiByteToWideChar(CP_ACP,            0,
num_1.c_str(), size_1, buffer_1, size_1 * sizeof(wchar_t));
                            buffer_1[size_1] = 0;
                            outtextxy(1050, 150 - 20, buffer_1); // 输
出文字
                            delete buffer_1;
                        }
                        else
                            return ERROR;
                    }
                }
            }
        }
    }
    else if (Game_type == Difficult_type) {
        while (!G.check_over())
```

```
{
    int move_key = Get_key(0.5);
    if (move_key == 13)
    {
        is_start = 0;
        is_over = 1;
        break;
    }
    else {
        if (move_key == int('w'))
            move_key = 1;
        else if (move_key == int('s'))
            move_key = 2;
        else if (move_key == int('a'))
            move_key = 3;
        else if (move_key == int('d'))
            move_key = 4;

        S.move(move_key);
        Snake_point* snake_head = S.Get_first();
        int x = snake_head->Snake_x;
        int y = snake_head->Snake_y;

        int tail_x = S.get_tail_x();
        int tail_y = S.get_tail_y();
        if (G.Game_map[x][y] == Blank_type)
        {
            G.update_map(x, y, Snake_type);
            G.update_map(tail_x, tail_y, Blank_type);
            S.update_tail();
        }
        else if (G.Game_map[x][y] == Food_type)
        {
            G.update_map(x, y, Snake_type);
            S.get_tail();
            F.delete_food();
            F.check_food(G);
            settextstyle(40, 0, _T("楷体"));
            settextcolor(BLACK);


            setlinecolor(WHITE);
            setfillcolor(WHITE);
            fillrectangle(1040, 150 - 30, 1200, 150 + 30); // 输
```

出文字

```
                    string num_1 = to_string(S.Get_num());
                    size_t size_1 = num_1.length();
                    wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 150 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 350 - 30, 1200, 350 + 30); // 输
出文字

                    string num_2 = to_string((S.Get_num() - 3) * 5 +
G.Get_Score());
                    size_t size_2 = num_2.length();
                    wchar_t* buffer_2 = new wchar_t[size_2 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_2.c_str(),
size_2, buffer_2, size_2 * sizeof(wchar_t));
                    buffer_2[size_2] = 0;
                    outtextxy(1050, 350 - 20, buffer_2); // 输出文字
                    delete buffer_2;
                }
                else if (G.Game_map[x][y] == Wall_type)
                {
                    G.Dead();

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 250 - 30, 1200, 250 + 30); // 输
出文字

                    string num_1 = to_string(G.Get_Lives());
                    size_t size_1 = num_1.length();
                    wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 250 - 20, buffer_1); // 输出文字
                    delete buffer_1;
```

```
                    if (G.Get_Lives() == 0)
                    {
                        G.Set_Score((S.Get_num()  -  3)  *  5  +
G.Get_Score());
                        S.Snake_wall(1, G);
                        return Wall_over;
                    }
                    else
                    {
                        G.Set_Score((S.Get_num()  -  3)  *  5  +
G.Get_Score());
                        F.Set_num(S.Get_num() + F.Get_num());
                        int a = S.Snake_wall(1, G);
                        if (a)
                        {
                            S.new_list(G);

                            setlinecolor(WHITE);
                            setfillcolor(WHITE);
                            fillrectangle(1040, 150 - 30, 1200, 150 +
30); // 输出文字

                            string num_1 = to_string(3);
                            size_t size_1 = num_1.length();
                            wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                            MultiByteToWideChar(CP_ACP,             0,
num_1.c_str(), size_1, buffer_1, size_1 * sizeof(wchar_t));
                            buffer_1[size_1] = 0;
                            outtextxy(1050, 150 - 20, buffer_1); // 输
出文字
                            delete buffer_1;
                        }
                        else
                            return ERROR;
                    }

                }
                else if (G.Game_map[x][y] == Snake_type)
                {
                    G.Dead();

                    setlinecolor(WHITE);
                    setfillcolor(WHITE);
                    fillrectangle(1040, 250 - 30, 1200, 250 + 30); // 输
```

出文字

```
                    string num_1 = to_string(G.Get_Lives());
                    size_t size_1 = num_1.length();
                    wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                    MultiByteToWideChar(CP_ACP,   0,   num_1.c_str(),
size_1, buffer_1, size_1 * sizeof(wchar_t));
                    buffer_1[size_1] = 0;
                    outtextxy(1050, 250 - 20, buffer_1); // 输出文字
                    delete buffer_1;

                    if (G.Get_Lives() == 0)
                    {
                        G.Set_Score((S.Get_num()   -   3)   *   5   +
G.Get_Score());
                        S.Snake_wall(1, G);
                        return Snake_over;
                    }
                    else
                    {
                        G.Set_Score((S.Get_num()   -   3)   *   5   +
G.Get_Score());
                        int a = S.Snake_wall(1, G);
                        if (a)
                        {
                            S.new_list(G);
                            setlinecolor(WHITE);
                            setfillcolor(WHITE);
                            fillrectangle(1040, 150 - 30, 1200, 150 +
30); // 输出文字

                            string num_1 = to_string(3);
                            size_t size_1 = num_1.length();
                            wchar_t* buffer_1 = new wchar_t[size_1 + 1];
                            MultiByteToWideChar(CP_ACP,            0,
num_1.c_str(), size_1, buffer_1, size_1 * sizeof(wchar_t));
                            buffer_1[size_1] = 0;
                            outtextxy(1050, 150 - 20, buffer_1); // 输
出文字
                            delete buffer_1;
                        }
                        else
                            return ERROR;
                    }
```

```
                }
            }
        }
    }
    return is_over;
}

int main() {
    while (1) {
        srand(time(0));

        int Game_type = Menu_main();

        while (Game_type == 0) {
            Game_type = Menu_main();
        }

        cleardevice();
        setbkcolor(COLORREF(WHITE));

        if (Game_type == Easy_type ||
            Game_type == Middle_type ||
            Game_type == Difficult_type)
        {
            Game new_Game(Game_type);
            Food_list F(new_Game);
            Wall_list W(new_Game);
            Snake_list S(new_Game);

            new_Game.Game_UI(Game_type);
            int a = Game_start(new_Game, F, W, S, Game_type);

            if (a == Wall_over)
                outtextxy(840, 650 - 20, _T("撞墙"));
            else if (a == Snake_over)
                outtextxy(840, 650 - 20, _T("撞到自己"));
            else if (a == ERROR)
                outtextxy(840, 650 - 20, _T("游戏错误"));

            ofstream Record_out("./file/record.txt", ios::app);
            if (!Record_out.is_open()) {
                cout << "历史记录文件打开失败" << endl;
                exit(0);
            }
```

```
            Record_out << Game_type << " " << new_Game.Get_Score() <<
endl;

            Record_out.close();

            new_Game.~Game();

            int key_value = Get_key(0.5);
            while (key_value != 13)
                key_value = Get_key(0.5);
        }
        //else if (Game_type == Last_Game)        {}

        else if (Game_type == Last_Game)
        {
            cleardevice();

            settextstyle(40, 0, _T("楷体"));
            settextcolor(BLACK);
            outtextxy(200, 50 - 20, _T("抱歉，赶 DDL 了"));
            outtextxy(200, 150 - 20, _T("此功能来不及实现了"));
            outtextxy(200, 250 - 20, _T("十分抱歉"));

            int key_value = Get_key(0.5);
            while (key_value != 13)
                key_value = Get_key(0.5);
        }

        else {

            cleardevice();
            setbkcolor(WHITE);

            ifstream Record_in("./file/record.txt");
            if (!Record_in.is_open()) {
                cout << "历史记录文件打开失败" << endl;
                exit(0);
            }

            int a, b = 0;

            settextstyle(40, 0, _T("楷体"));
            settextcolor(BLACK);
```

```
            string num_1;
            size_t size_1;


            for (int i = 0; i < 8; ++i)
            {
                if (Record_in.eof())
                    break;
                Record_in >> a >> b;

                if (a == 1)
                    outtextxy(200, i * 100 + 50 - 20, _T("入门版")); //
输出文字
                if (a == 2)
                    outtextxy(200, i * 100 + 50 - 20, _T("进阶版")); //
输出文字
                if (a == 3)
                    outtextxy(200, i * 100 + 50 - 20, _T("高级版")); //
输出文字

                num_1 = to_string(b);
                size_1 = num_1.length();
                wchar_t* buffer_2 = new wchar_t[size_1 + 1];
                MultiByteToWideChar(CP_ACP, 0, num_1.c_str(), size_1,
buffer_2, size_1 * sizeof(wchar_t));
                buffer_2[size_1] = 0;
                outtextxy(450, i * 100 + 50 - 20, buffer_2); // 输出文
字
                delete buffer_2;

            }

            Record_in.close();

            int key_value = Get_key(0.5);
            while (key_value != 13)
                key_value = Get_key(0.5);
            a = 1;

        }
    }
    return 0;
}
```

```cpp
#define _CRT_SECURE_NO_WARNINGS

#include<iostream>
#include<graphics.h>
#include<conio.h>
#include<fstream>

#include "Game.h"

using namespace std;


int Place_check(int x, int y, bool flag)
{
    double far_1 = sqrt((x - 400) * (x - 400) + (y - 500) * (y - 500));
    double far_2 = sqrt((x - 800) * (x - 800) + (y - 500) * (y - 500));
    double far_3 = sqrt((x - 600) * (x - 600) + (y - 600) * (y - 600));

    if (!flag)
    {
        if (far_1 <= 100)
            return Easy_type;
        else if (far_2 <= 100)
            return Last_Game;
        else if (far_3 <= 100)
            return Read_Record;
    }
    else
    {
        if (far_1 <= 100)
            return Easy_type;
        else if (far_2 <= 100)
            return Middle_type;
        else if (far_3 <= 100)
            return Difficult_type;
    }
    return 0;
}


int init(int flag)
{
    cleardevice();
    setbkcolor(COLORREF(BLACK));
```

```
int last_fill = 0;

//开始页面
IMAGE title;
loadimage(&title, _T("./img/start_background.jpg"), 1200, 800);
putimage(0, 0, &title);

setbkmode(TRANSPARENT);  // 设置背景模式为透明
RECT rect = { 300, 100, 900 , 300 };  // 定义矩形区域的坐标
settextstyle(100, 0, _T("楷体"));
settextcolor(RED);
if (!flag)
    outtextxy(300, 100, _T("贪吃蛇  游戏"));
else
    outtextxy(300, 100, _T("选择游戏模式"));

setlinecolor(RGB(230, 230, 250));
setlinestyle(PS_SOLID, 10);  // 设置线条样式为实线，线宽为 2


ellipse(400 - 100, 500 - 100, 400 + 100, 500 + 100);  // 绘制圆形
settextstyle(40, 0, _T("楷体"));
settextcolor(BLACK);
if (!flag)
    outtextxy(400 - 80, 500 - 20, _T("新的开始"));
else
    outtextxy(400 - 60, 500 - 20, _T("入门版"));

ellipse(800 - 100, 500 - 100, 800 + 100, 500 + 100);  // 绘制圆形
settextstyle(40, 0, _T("楷体"));
settextcolor(BLACK);
if (!flag)
    outtextxy(800 - 80, 500 - 20, _T("继续游戏"));
else
    outtextxy(800 - 60, 500 - 20, _T("进阶版"));


ellipse(600 - 100, 600 - 100, 600 + 100, 600 + 100);  // 绘制圆形
settextstyle(40, 0, _T("楷体"));
settextcolor(BLACK);
if (!flag)
    outtextxy(600 - 80, 600 - 20, _T("历史记录"));
else
```

```
        outtextxy(600 - 60, 600 - 20, _T("高级版"));

    setfillcolor(RGB(230, 230, 250));

    MOUSEMSG m{ 0 };            // 鼠标信息结构体

    while (1)
    {
        if (MouseHit())m = GetMouseMsg();   // 获取鼠标信息
        TCHAR s[20];          // EasyX 设置文字需要的字符串变量类型
        switch (m.uMsg)
        {
            case WM_LBUTTONDOWN:
                if (!flag)
                {
                    int check_flag = Place_check(m.x, m.y, flag);
                    if (check_flag == Easy_type)
                        return init(1);
                    else if (check_flag == Last_Game)
                        return Last_Game;
                    else if (check_flag == Read_Record)
                        return Read_Record;
                }
                else
                {
                    int check_flag = Place_check(m.x, m.y, flag);
                    if (check_flag == Easy_type)
                        return Easy_type;
                    else if (check_flag == Middle_type)
                        return Middle_type;
                    else if (check_flag == Difficult_type)
                        return Difficult_type;
                }
                break;
        }
    }
    closegraph();
}

int Menu_main()
{
    initgraph(1200, 800);
    int menu_type = init(0);
    return menu_type;
```

```
}

#include"Game.h"
#include"Food.h"
#include "Wall.h"
#include"Snake.h"
#include<iostream>
#include<fstream>

using namespace std;

int cx[4] = { 0,0,-1,1 };
int cy[4] = { -1,1,0,0 };

Snake_point* Snake_list::Get_first() {
    return List_first;
}

Snake_point* Snake_list::Get_last() {
    return List_last;
}

int Snake_list::Get_dir() {
    return head_dir;
}

int Snake_list::Get_num() {
    return Snake_num;
}

Snake_list::Snake_list(Game& G) {
    List_first = NULL;
    List_last = NULL;

    int x, y, dir;
    while (1) {
        dir = rand() % 4 + 1;
        x = rand() % 48 + 1;
        y = rand() % 48 + 1;

        if (G.Game_map[x][y] == Blank_type
            && G.Game_map[x + cx[dir - 1]][y + cy[dir - 1]] == Blank_type
            && G.Game_map[x + 2 * cx[dir - 1]][y + 2 * cy[dir - 1]] ==
Blank_type) {
```

```
            G.Game_map[x][y] = Snake_type;
            G.Game_map[x + cx[dir - 1]][y + cy[dir - 1]] = Snake_type;
            G.Game_map[x + 2 * cx[dir - 1]][y + 2 * cy[dir - 1]] =
Snake_type;

            new_snake(x, y);
            new_snake(x + cx[dir - 1], y + cy[dir - 1]);
            new_snake(x + 2 * cx[dir - 1], y + 2 * cy[dir - 1]);

            Snake_num = 3;

            tail_x = x + 2 * cx[dir - 1];
            tail_y = y + 2 * cy[dir - 1];

            if (dir == 1 || dir == 3)
                head_dir = dir + 1;
            else
                head_dir = dir - 1;
            break;
        }
        else
            continue;
    }
}

//向蛇的链表中加入新结点
void Snake_list::new_snake(int x, int y)
{
    Snake_point* new_point = new Snake_point;
    new_point->Snake_x = x;
    new_point->Snake_y = y;
    new_point->last_point = NULL;
    new_point->next_point = NULL;

    if (List_first == NULL)
    {
        List_first = new_point;
        List_last = new_point;
    }
    else {
        List_last->next_point = new_point;
        new_point->last_point = List_last;
        List_last = new_point;
```

```
    }
    Snake_num++;
}

void Snake_list::move(int new_dir)
{
    if (new_dir == -1);
    else if ((head_dir == 1 || head_dir == 3)
        && new_dir == head_dir + 1);
    else if ((head_dir == 2 || head_dir == 4)
        && new_dir == head_dir - 1);
    else
        head_dir = new_dir;

    //蛇头向前一步
    Snake_point* new_point = new Snake_point;
    new_point->Snake_x = List_first->Snake_x + cx[head_dir -1];
    new_point->Snake_y = List_first->Snake_y + cy[head_dir -1];

    new_point->last_point = NULL;
    new_point->next_point = List_first;
    List_first->last_point = new_point;
    List_first = new_point;

    //蛇尾向前一步
    Snake_point* help = List_last;
    List_last = List_last->last_point;
    List_last->next_point = NULL;
    delete help;

    return;
}

void Snake_list::get_tail()
{
    ++Snake_num;
    Snake_point* new_point = new Snake_point;
    new_point->Snake_x = tail_x;
    new_point->Snake_y = tail_y;
    new_point->next_point = NULL;
    new_point->last_point = List_last;
    List_last->next_point = new_point;
    List_last = new_point;
    return;
```

```
}

//上次游戏的继续
Snake_list::Snake_list(istream& in, Game& G)
{
}

//析构函数
Snake_list::~Snake_list() {

    Snake_point* Help_1, * Help_2;

    for (Help_1 = List_first; Help_1;)
    {
        Help_2 = Help_1;
        Help_1 = Help_1->next_point;
        delete Help_2;
    }
}

bool Snake_list::Snake_wall(bool flag, Game &G) {

    Snake_point* Help_1, * Help_2;
    for (Help_1 = List_first; Help_1;)
    {
        Help_2 = Help_1;
        Help_1 = Help_1->next_point;
        if (!flag)
            G.update_map(Help_2->Snake_x, Help_2->Snake_y, Wall_type);
        else
            G.update_map(Help_2->Snake_x, Help_2->Snake_y, Food_type);
        delete Help_2;
    }

    if (!flag)
        G.update_map(tail_x, tail_y, Wall_type);
    else
        G.update_map(tail_x, tail_y, Food_type);

    bool is_free = 0;
    for (int i = 1; i < 49; i++)
    {
        for (int j = 1; j < 49 - 2; j++)
        {
```

```
            if (G.Game_map[i][j] == Blank_type ||
                G.Game_map[i][j + 1] == Blank_type ||
                G.Game_map[i][j + 2] == Blank_type)
            {
                is_free = 1;
                break;
            }
        }
    }

    if (!is_free)
    {
        for (int i = 1; i < 49 - 2; i++)
        {
            for (int j = 1; j < 49; j++)
            {
                if (G.Game_map[i][j] == Blank_type ||
                    G.Game_map[i + 1][j] == Blank_type ||
                    G.Game_map[i + 2][j] == Blank_type)
                {
                    is_free = 1;
                    break;
                }
            }
        }
    }

    if (!is_free)
        return 0;

    return 1;
}


void Snake_list::new_list(Game &G)
{
    List_first = NULL;
    List_last = NULL;

    Snake_num = 3;

    int x, y, dir;
    while (1) {
        dir = rand() % 4 + 1;
```

```
        x = rand() % 48 + 1;
        y = rand() % 48 + 1;

        if (G.Game_map[x][y] == Blank_type
            && G.Game_map[x + cx[dir - 1]][y + cy[dir - 1]] == Blank_type
            && G.Game_map[x + 2 * cx[dir - 1]][y + 2 * cy[dir - 1]] ==
Blank_type) {

            G.Game_map[x][y] = Snake_type;
            G.Game_map[x + cx[dir - 1]][y + cy[dir - 1]] = Snake_type;
            G.Game_map[x + 2 * cx[dir - 1]][y + 2 * cy[dir - 1]] =
Snake_type;

            setlinecolor(GREEN);
            setfillcolor(GREEN);
            fillrectangle(x * 16, y * 16, x * 16 + 15, y * 16 + 15);
            fillrectangle((x + cx[dir - 1]) * 16, (y + cy[dir - 1]) * 16,
(x + cx[dir - 1]) * 16 + 15, (y + cy[dir - 1]) * 16 + 15);
            fillrectangle((x + 2 * cx[dir - 1]) * 16, (y + 2 * cy[dir -
1]) * 16, (x + 2 * cx[dir - 1]) * 16 + 15, (y + 2 * cy[dir - 1]) * 16 +
15);

            new_snake(x, y);
            new_snake(x + cx[dir - 1], y + cy[dir - 1]);
            new_snake(x + 2 * cx[dir - 1], y + 2 * cy[dir - 1]);

            Snake_num = 3;

            tail_x = x + 2 * cx[dir - 1];
            tail_y = y + 2 * cy[dir - 1];

            if (dir == 1 || dir == 3)
                head_dir = dir + 1;
            else
                head_dir = dir - 1;
            break;
        }
        else
            continue;
    }
    return;
}

#include "Wall.h"
```

```cpp
#include "Game.h"
#include <iostream>
using namespace std;

//获取链表头
Wall_point* Wall_list::Get_first() {
    return List_first;
}

//获取链表尾
Wall_point* Wall_list::Get_last() {
    return List_last;
}

//获取链表中的食物个数
int Wall_list::Get_num() {
    return Wall_num;
}


//新游戏的开始
Wall_list::Wall_list(Game& G)
{
    List_first = NULL;
    List_last = NULL;
}


//上次游戏的继续
Wall_list::Wall_list(istream& in, Game& G)
{
    List_first = NULL;
    List_last = NULL;
}

//析构函数
Wall_list::~Wall_list() {
    if (Wall_num == 0)
        return;

    Wall_point* Help_1, * Help_2;

    for (Help_1 = List_first; Help_1;)
    {
```

```
        Help_2 = Help_1;
        Help_1 = Help_1->next_point;
        delete Help_2;
    }
}
```