俄罗斯方块实验报告

Experiment Presentation of

Easy Tetris

PB22111679 孙婧雯

PB22111711 陈昕琪

2022年12月25日

功能简介

Easy Tetris 程序所要达到的功能是，呈现具有三种难度模式的、简易的Tetris小游戏，游戏界面有基本操作提示和当前、历史得分记录，力求最终达到流畅易上手、游戏体验良好的效果。

注：游戏的BGM为《日常的风景》（《凉宫春日的忧郁》），得分音效选自手游《开心消消乐》。

目录

  \* [概要设计](#概要设计)

  \* [详细设计](#详细设计)

  \* [功能展示](#func)

  \* [存在的问题](#呜呜)

  \* [团队分工](#呜呜)

  \* [总结与建议](#呜呜)

  \* [参考资料](#refer)

1. 概要设计 —————————————————[返回目录](#目录)

1.1 使用的头文件与库函数

<stdio.h>

    fclose() 用于关闭文件流，具体的文件是历史最高分记录(.txt) 。

    fopen() 用于打开文件流，具体的文件是历史最高分记录(.txt) 。

    fread() 用于从文件流中读取历史最高分的数据。

    fwrite() 用于写历史最高分到文件流。

    printf() 格式化输出函数。

    scanf() 格式化输入函数。

    if()-else 与 switch()-case 均为选择分支语句使用的函数。

    for() 与 while() 均为循环语句使用的函数。

<Windows.h>

    system("cls") 用于清屏。

    system("title") 用于设置.exe会话窗口的标题

    system("mode") 用于配置.exe会话窗口，具体是长x宽的尺寸。

    system("pause>nul") 用于暂停程序运行，按任意键继续。

    Sleep() 用于延迟程序运行。

    SetConsoleTextAttribute() 用于设置控制台窗口字体颜色和背景色；使用十进制颜色对照表(如图1.1-1)。

    SetConsoleCursorInfo() 用于获取控制台窗口光标大小和可见性的信息。

    SetConsoleCursorPosition() 用于获取控制台窗口光标位置。

    GetStdHandle() 用于获得Windows标准句柄，具体的对象在本实验中是光标。

<stdlib.h>

    exit() 用于终止程序运行。

    rand() 用于生成随机数。

    srand() 用于初始化随机种子 (发生器)。

    此头文件中也含有system()函数。

<time.h>

    time() 用于获取系统时间，常见于随机种子生成器。

<conio.h>

    getch() 用于无回显地从控制台输入一个字符。

kbhit() 用于检查键盘是否有输入，有返回-1，无返回0.

<mmsystem.h>

PlaySound() 用于导入BGM

mciSendString() 用于导入得分音效



1.1-1十进制颜色对照表

1.2 定义的函数与调用关系

代码中所有自定义的函数如下：

void HideCursor();         //用于隐藏光标

void Gotoxy(int x, int y);//用于使光标跳转到指定的(x,y)位置

void InitPlayArea();      //用于初始化游戏区，绘制边框，给出游戏说明

void InitBlock();         //用于初始化方块形态

int ShowMenu();           //用于显示初始菜单并选择模式

void ClearArea();         //用于游戏开始前清除初始菜单

void color(int num);      //用于设置控制台窗口字体颜色，使用十进制颜色对照表

void DrawBlock(int shape, int form, int x, int y);//用于画出方块

void EraseBlock(int shape, int form, int x, int y);//用于擦除方块

int IsLegal(int shape, int form, int x, int y);  //用于合法性判断(能否移动/旋转)

int IsCount();             //用于计算得分

void Congratulations();   //得分后的小动画

void IsGameOver();        //用于判断是否gameover

void StartGame();         // \*此为游戏的主函数

void SPause();            //用于暂停游戏

void ReadRecord();        //用于从文件读取最高分

void WriteRecord();       //用于更新最高分到文件

int main()                //main函数

程序运行时，首先调用main()函数，main()函数内部依次调用HideCursor()隐藏光标、ShowMenu()选择游戏模式、ReadRecord()加载历史分数、InitPalyArea()加载游戏区、InitBlock()加载方块形态、StartGame()开始游戏。

ShowMenu()函数中，调用Gotoxy()函数在指定位置写出文字，玩家选择游戏模式之后会调用ClearArea()函数清屏。

InitPlayArea()函数中，调用Gotoxy()函数在指定位置画出游戏区边框、写出文字提示游戏玩法。

StartGame()函数中，主体框架是两层while()循环，外层循环控制每一个方块下落全程并提示玩家下一个方块的形态，内层循环控制每一个方块下落一格并接收键盘上的移动、旋转、暂停等操作。

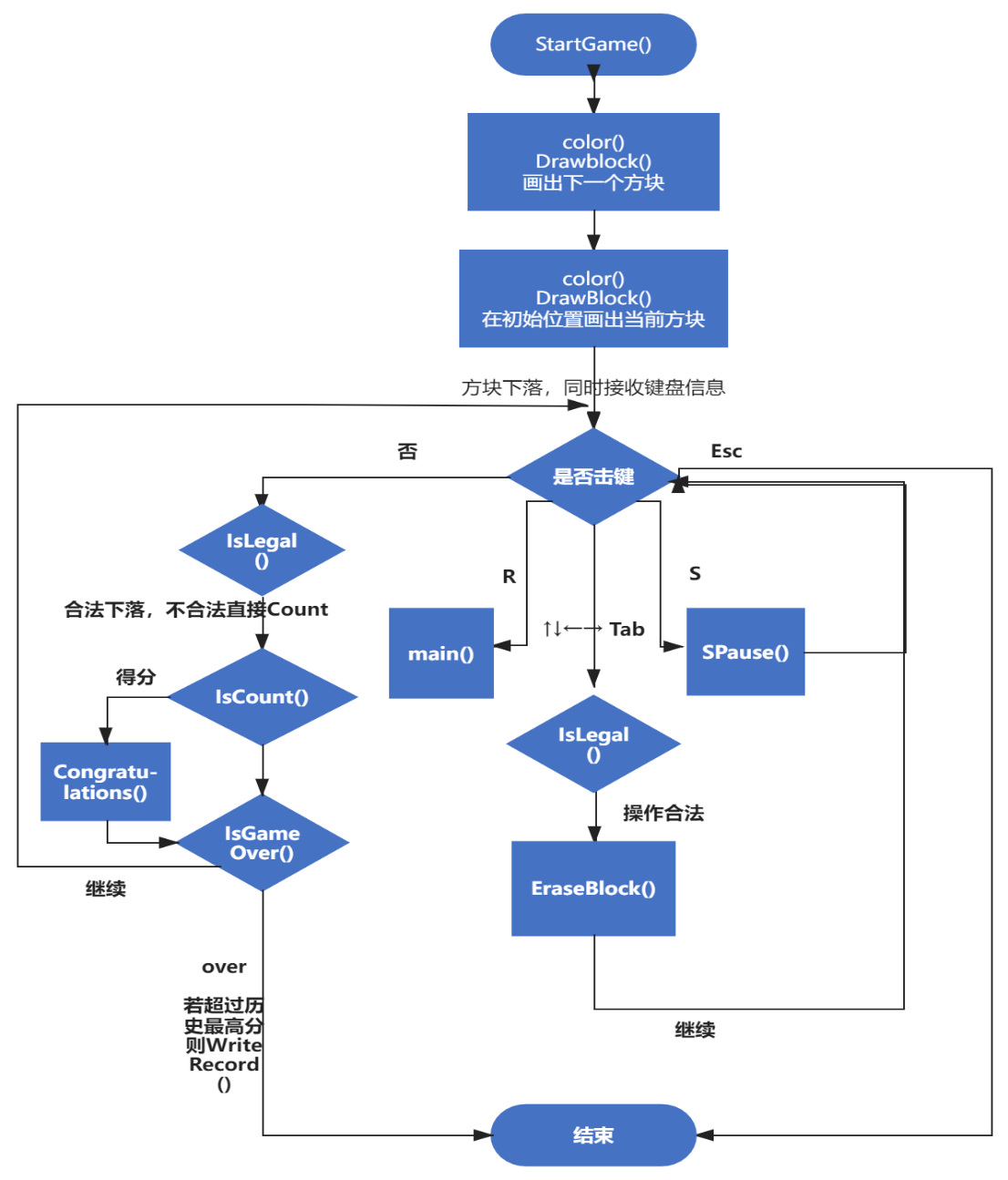
顺序执行时，程序首先进入第1层循环，调用color()和DrawBlock()画出下一个方块，再进入第2层循环，调用DrawBlock()画出当前方块，方块根据玩家选择的模式以不同速度下落。每下落一格接收一次键盘上是否有操作，若有且为移动/旋转操作，调用IsLegal()判断是否合法操作，合法的则调用EraseBlock()擦除当前位置显示的方块，预备进入下一次内层循环时在新的位置画出方块；若有且为暂停操作，调用SPause()暂停；若有且为退出操作，终止游戏；若有且为重开操作，直接调用main()重新加载游戏。接收结束后若游戏继续，会重新进入内层循环。



1.2-1, 1.2-2 StartGame()中接收键盘信息使用的键码值表

若没有接收到键盘操作，调用IsLegal()、IsCount()判断是否继续向下移动、是否得分。IsCount()中，若得分会调用Congratulations()出现加分小动画。在IsCount()的结尾，调用IsGameOver()判断是否结束游戏。每次结束游戏时，若打破最高分记录，则调用WriteRecord()记录新高分。

因此StartGame()的执行流程图如图1.2-3：



1.2-1 StartGame()流程图

经测试，一旦开始执行StartGame()函数，玩家可以随时进行键盘操作，且gameover和暂停均有反应时间，可以获得较良好的游戏体验。具体代码将在[详细设计](#start)中给出。

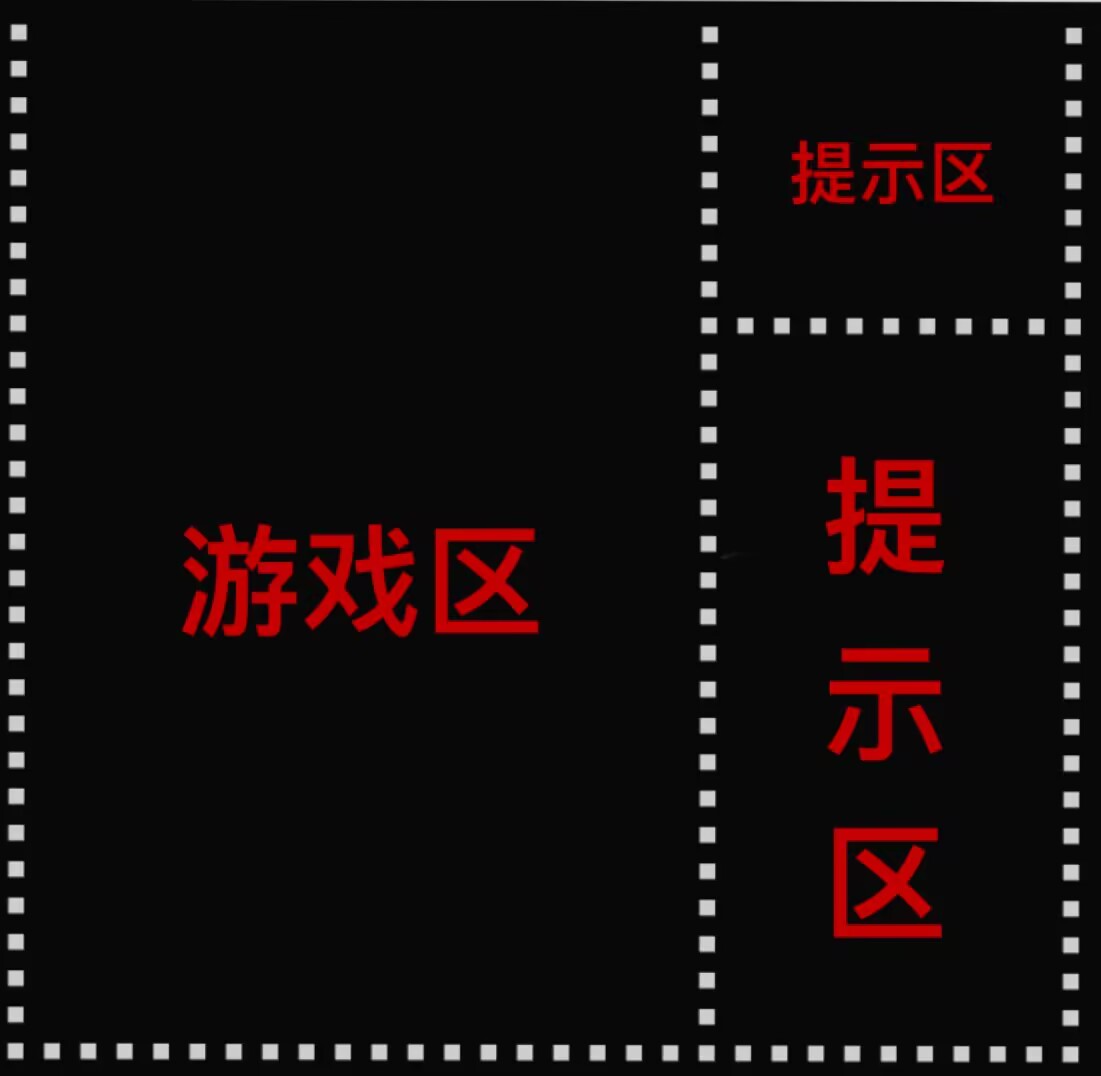
2.详细设计 ————————————————————————————[返回目录](#目录)

2.1设置图形区域大小

1.首先定义一下游戏区域大小，定义行数和列数。界面分为游戏区和提示区，方块堆积的地方为游戏区，提示按键以及下一个方块的地区为提示区。

#define ROW 28 + 1 //游戏区行数

#define COL 18 + 2 //游戏区列数



2.1-1 游戏界面

2.根据需要用到的按键的键码值对其进行宏定义

#define DOWN 80 //方向键：下

#define LEFT 75 //方向键：左

#define RIGHT 77 //方向键：右

#define TAB 9 //空格键

#define ESC 27 //Esc键

3.设定一个结构体，该结构体记录界面的每个位置是否有方块，若有方块还需记录该位置方块的颜色。

struct PlayArea {

int data[ROW][COL + 10]; //用于标记指定位置是否有方块（1为有，0为无）

int color[ROW][COL + 10]; //用于记录指定位置的方块颜色编码

}area;

4.再设置一个结构体，该结构体当中存储着一个4行4列的二维数组，用于存储单个方块的基本信息。对于7种基本形状的方块，每种方块通过顺时针旋转有4种形态，共28种。因此，用该结构体定义一个7行4列的二维数组存储这28个方块的信息。

struct Block{

int space[4][4];

}block[7][4]; //用于存储7种基本形状方块的各自的4种形态的信息

2.2主函数

1. 为cmd窗口命名，隐藏光标。

2. 播放游戏的BGM，初始化界面，包括主菜单，方块，游戏界面等等。

3. 开始游戏。

int main(){

system("title Easy\_Tetris by PB22111679 and PB22111711"); //设置cmd窗口的名字

HideCursor(); //隐藏光标

PlaySound("BGM.wav",NULL,SND\_FILENAME | SND\_ASYNC | SND\_LOOP);

Max = 0, Record = 0;

while(Speed == -1) Speed = ShowMenu();

ReadRecord(); //从文件读取最高分到Max记录

InitPlayArea();

InitBlock();

srand((UL)time(NULL));

StartGame();

return 0;

}

2.3显示主菜单

显示主菜单并且选择难易程度，根据用户键入的字母，设定不同的下落速度，由此实现难易程度的区分。若键入错误的字母，可重新选择。

int ShowMenu(){

Gotoxy(4, 4);

printf("欢迎进入Tetris小游戏！");

Gotoxy(4, 6);

printf("请选择游戏模式：");

Gotoxy(4, 8);

printf("按E 简单模式");

Gotoxy(4, 10);

printf("按N 一般模式");

Gotoxy(4, 12);

printf("按H 困难模式");

Gotoxy(4, 15);

printf("请选择...");

char ch = getch();

switch(ch){

case 'e':

case 'E': {

ClearArea();

return 20000;

}

case 'n':

case 'N': {

ClearArea();

return 13500;

}

case 'H':

case 'h': {

ClearArea();

return 9000;

}

default:{

Gotoxy(4, 16);

printf("选择错误，请再次选择...");

Sleep(500);

Gotoxy(4, 16);

printf(" ");

return -1;

}

}

}

2.4清屏

为了实现游戏界面的转换，需要进行清屏操作。借助system函数，可实现清除目前屏幕上的图案或者文字。

void ClearArea(){

system("cls");

Sleep(300);

}

2.5隐藏光标

进行游戏时不需要用到光标，且光标干扰视线，因此需要对光标进行隐藏。

void HideCursor(){

CONSOLE\_CURSOR\_INFO curInfo; //定义光标信息的结构体变量

curInfo.dwSize = 1; //赋值后使隐藏光标有效

curInfo.bVisible = FALSE; //将光标设置为不可见

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE); //获取控制台句柄

SetConsoleCursorInfo(handle, &curInfo); //设置光标信息

}

注：<windows.h>中光标可见性结构体

typedef struct \_CONSOLE\_CURSOR\_INFO{ //光标信息结构体

DWORD dwSize; //光标尺寸大小，范围是1~100

BOOL bVisible; //表示光标是否可见，true表示可见

} CONSOLE\_CURSOR\_INFO, \*PCONSOLE\_CURSOR\_INFO;

获得光标和设置光标信息的函数如下：

BOOL GetConsoleCursorInfo( //获得光标信息

HANDLE hConsoleOutput, //句柄

PCONSOLE\_CURSOR\_INFO lpConsoleCursorInfo //光标信息指针

);

BOOL SetConsoleCursorInfo( //设置光标信息

HANDLE hConsoleOutput, //句柄

const CONSOLE\_CURSOR\_INFO \*lpConsoleCursorInfo //光标信息

);

2.6光标跳转

游戏进行时，使光标可以跳转到任意位置。（课本上定义为SetPos函数）

void Gotoxy(int x, int y){ //(其实是课本上的SetPos)

COORD pos;//定义光标位置的结构体变量

pos.X = x;//定义横坐标

pos.Y = y;//定义纵坐标

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);//控制句柄//

SetConsoleCursorPosition(handle, pos);

}

注：<windows.h>中光标位置结构体

typedef struct\_COORD {

SHORT X;//横坐标

SHORT Y;//纵坐标

}COORD，\*PCOORD；

设置光标位置的函数：

BOOL SetConsoleCursorPosition(

\_In\_ HANDLE hConsoleOutput,

\_In\_ COORD dwCursorPosition

);

2.7初始化界面

初始化界面实现边框的打印，借助光标跳转函数实现游戏界面的划分，并打印提示信息.

void InitPlayArea(){

color(7); //颜色设置为白色

int i, j;

for(i = 0; i < ROW; i++){

for(j = 0; j < COL + 10; j++){

if(j == 0 || j == COL - 1 || j == COL + 9) {

area.data[i][j] = 1; //绘制竖直边框

area.color[i][j] = 7;

Gotoxy(2 \* j, i);

printf("■");

}

else if(i == ROW - 1){ //绘制底边

area.data[i][j] = 1;

area.color[i][j] = 7;

printf(" ■");

}

else

area.data[i][j] = 0;

}

}

for(i = COL; i < COL + 10; i++) { //绘制旁栏

area.data[8][i] = 1;

area.color[8][i] = 7;

Gotoxy(2 \* i, 8);

printf("■");

}

Gotoxy(2 \* COL + 4, 1); //打印旁栏提示

printf("下一个方块：");

Gotoxy(2 \* COL + 4, ROW - 19);

printf("左移：←");

Gotoxy(2 \* COL + 4, ROW - 17);

printf("右移：→");

Gotoxy(2 \* COL + 4, ROW - 15);

printf("加速：↓");

Gotoxy(2 \* COL + 4, ROW - 13);

printf("旋转：Tab");

Gotoxy(2 \* COL + 4, ROW - 11);

printf("暂停: S");

Gotoxy(2 \* COL + 4, ROW - 9);

printf("退出: Esc");

Gotoxy(2 \* COL + 4, ROW - 7);

printf("重新开始:R");

Gotoxy(2 \* COL + 4, ROW - 5);

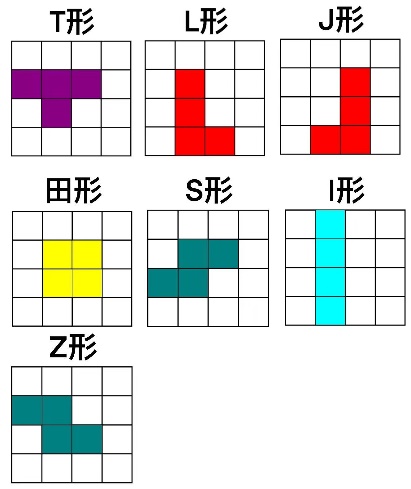
printf("最高纪录:%d", Max);

Gotoxy(2 \* COL + 4, ROW - 3);

printf("当前分数：%d", Record);

}

2.8初始化方块信息

初始化七个方块如图：

2.8-1 七种方块的初始形态

旋转之后的28种形态储存在block[7][4]中，旋转图形的原理是原来坐标为（i，j）的方块顺时针旋转后坐标为（j，3-i）,借助for函数循环，可实现后面每一种形态，都由前一种形态顺时针旋转得到。由此得到28个方块图形，并储存在数组中。

void InitBlock(){

int i;

//“T”形

for(i = 0; i <= 2; i++)

block[0][0].space[1][i] = 1;

block[0][0].space[2][1] = 1;

//“L”形

for(i = 1; i <= 3; i++)

block[1][0].space[i][1] = 1;

block[1][0].space[3][2] = 1;

//“J”形

for(i = 1; i <= 3; i++)

block[2][0].space[i][2] = 1;

block[2][0].space[3][1] = 1;

for(i = 0; i <= 1; i++){

//“Z”形

block[3][0].space[1][i] = 1;

block[3][0].space[2][i + 1] = 1;

//“S”形

block[4][0].space[1][i + 1] = 1;

block[4][0].space[2][i] = 1;

//“田”形

block[5][0].space[1][i + 1] = 1;

block[5][0].space[2][i + 1] = 1;

}

//“I”形

for(i = 0; i <= 3; i++)

block[6][0].space[i][1] = 1;

int temp[4][4], j, shape, form;

for(shape = 0; shape < 7; shape++) { //7种形状

for(form = 0; form < 3; form++) {

//获取第form种形态

for(i = 0; i < 4; i++){

for(j = 0; j < 4; j++){

temp[i][j] = block[shape][form].space[i][j];

}

}

//将第form种形态顺时针旋转，得到第form+1种形态

for(i = 0; i < 4; i++) for(j = 0; j < 4; j++){

block[shape][form + 1].space[i][j] = temp[3 - j][i];

}

}

}

}

}

2.9颜色设置

颜色设置函数，实现不同形态方块的不同颜色设置。（使用十进制颜色对照表）

|  |  |
| --- | --- |
| T形 | 紫 |
| L形，J形 | 红 |
| Z形S形 | 绿 |
| 田形 | 黄 |
| I形 | 蓝 |

void color(int c){

switch (c){

case 0:

c = 5; //“T”形

break;

case 1:

case 2:

c = 12; //“L”形和“J”形

break;

case 3:

case 4:

c = 3; //“Z”形和“S”形

break;

case 5:

c = 14; //“O”形

break;

case 6:

c = 11; //“I”形

break;

default:

c = 7; //其他默认设置为白色

break;

}

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), c);

}

2.10画出方块

借助存储在block数组中的图形信息，遍历4×4方格，在应该存在方块的位置打印出一个“■”，由此可以打印出不同类型的方块。

void DrawBlock(int shape, int form, int x, int y){

int i, j;

for(i = 0; i < 4; i++){

for(j = 0; j < 4; j++){

if(block[shape][form].space[i][j] == 1) { //如果该位置有方块

Gotoxy(2 \* (x + j), y + i); //光标跳转到指定位置

printf("■"); //输出方块

}

}

}

}

2.11擦除方块

游戏进行时，对方块进行旋转或者平移变换时，都需要擦除原来的方块，然后再打印新的方块。实现方法是，先用光标定位方块的位置，再用两个空格覆盖原来方块的位置。

void EraseBlock(int shape, int form, int x, int y){

int i, j;

for(i = 0; i < 4; i++){

for(j = 0; j < 4; j++){

if(block[shape][form].space[i][j] == 1) {

Gotoxy(2 \* (x + j), y + i); //光标跳转到指定位置

printf(" "); //打印空格覆盖（两个空格）

}

}

}

}

2.12合理性判断

方块在下落或者变换的过程中，需要随时判断正常下落（下移一格）或者变换后是否合法。合法即可正常运行，非法（下落到底或者变化后的位置本来就有方块）则不进行变换。

int IsLegal(int shape, int form, int x, int y){

int i, j;

for(i = 0; i < 4; i++){

for(j = 0; j < 4; j++){

//如果方块落下or旋转的位置本来就已经有方块了，则不合法

if((block[shape][form].space[i][j] == 1) && (area.data[y + i][x + j] == 1))

return 0; //不合法

}

}

return 1; //合法

}

2.13判断是否得分

从下向上判断是否满格，若有一行满方块即可得分，更改分数并进行加分小动画，再将这一行清除（用空格覆盖），然后将上方的方块全部下移一行。下移结束后返回1，再次使用该函数检测是否有满格现象，直到返回0，继续进行游戏。

int IsCount(){

int i, j;

for(i = ROW - 2; i > 4; i--){

int sum = 0; //记录第i行的方块个数

for(j = 1; j < COL - 1; j++){

sum += area.data[i][j];

}

if(sum == 0) //该行没有方块，无需再判断其上的层次（剪枝）

break;

if(sum == COL - 2) { //该行全是方块，可得分

Record += 10; //满一行加10分

Congratulations(); //加分音乐和动画

color(7);

Gotoxy(2 \* COL + 4, ROW - 3); //光标跳转到当前分数的位置

printf("当前分数：%d", Record); //更新当前分数

for(j = 1; j < COL - 1; j++){ //清除得分行

area.data[i][j] = 0; //得分后被清除的位置标记为无方块

Gotoxy(2 \* j, i); //光标跳转到该位置

printf(" "); //打印空格覆盖（两个空格）

}

//把被清除行上面的行整体向下挪一格

for(j = i; j > 1; j--){

sum = 0; //记录上一行的方块个数

int k;

for(k = 1; k < COL - 1; k++){

sum += area.data[j - 1][k];

area.data[j][k] = area.data[j - 1][k]; //将上一行方块的标识移到下一行

area.color[j][k] = area.color[j - 1][k]; //将上一行方块的颜色编号移到下一行

if(area.data[j][k] == 1) {

Gotoxy(2 \* k, j);

color(area.color[j][k]); //颜色设为原方块颜色

printf("■");

}

else {

Gotoxy(2 \* k, j);

printf(" "); //打印空格覆盖（两个空格）

}

}

if (sum == 0) return 1; //返回1，表示还需调用该函数进行判断（移动下来的可能还有满行）

}

}

}

IsGameOver();

return 0; //判断结束，无需再调用该函数进行判断

}

2.14判断是否结束

1. 直接检测最上方一行是否有方块，如果有则游戏结束。

2. 游戏结束后判断是否打破最高纪录，并相应输出语句。

3. 询问玩家是否再来一局，根据键入y/n判断，若输入错误，可以再次选择。

void IsGameOver(){

int i;

for(i = 1; i < COL - 1; i++){

if(area.data[1][i] == 1){ //以第1行为顶层，不是第0行

system("cls");

color(7);

Gotoxy(2 \* (COL / 3), ROW / 2 - 3);

if(Record > Max){

printf("恭喜你打破最高记录，最高记录更新为%d", Record);

WriteRecord();

}

else if(Record == Max){

printf("与最高记录持平，加油再创佳绩", Record);

}

else{

printf("请继续加油，当前与最高记录相差%d", Max - Record);

}

Gotoxy(2 \* (COL / 3), ROW / 2);

printf("GAME OVER");

while(1)

{

char ch;

Gotoxy(2 \* (COL / 3), ROW / 2 + 3);

printf("再来一局?(y/n):");

ch = getch();

if(ch == 'y' || ch == 'Y'){

system("cls");

main(); //重新开始

}

else if(ch == 'n' || ch == 'N'){

Gotoxy(2 \* (COL / 3), ROW / 2 + 5);

exit(0); //结束程序

}

else{

Gotoxy(2 \* (COL / 3), ROW / 2 + 4);

printf("选择错误，请再次选择");

}

}

}

}

}

2.15暂停界面

1. 当键入暂停键时，界面停止变换，中间用空格覆盖，并打印出暂停语句。

2. 当键入随意键，表示继续游戏，则重新绘制被擦除的地方，界面继续移动。

void SPause(){

int d[COL + 2], c[COL + 2], i, y = (ROW) / 2;

for(i = 0; i < COL; i++){

d[i] = area.data[y][i], c[i] = area.color[y][i];

Gotoxy(2 \* i, y);

printf(" ");

}

Gotoxy(2, y);

printf("暂停中，请按任意键继续...");

system("pause>nul"); //暂停（按任意键继续）

Gotoxy(2, y);

printf(" ");

for(i = 0; i < COL; i++){ //重新绘制被消掉的地方

if(d[i] == 1){

color(c[i]);

Gotoxy(2 \* i, y);

printf("■");

}

}

}

2.16游戏主体

1. 随机获取方块的形状和形态，更新下一个方块并打印在右上角的提示区。

2. 方块下落过程中，每下落一格有一定的时间间隔，在这段时间内，如果用户有操作，则先判断进行什么操作，相应变换后再下落。

3. 若没有操作，则正常下落。

4. 每次操作前都需要判断是否合法，若游戏落到底部，则判断是否有加分或结束的情况。

5. 若游戏未结束，则重复以上操作。

可以参考本报告第一部分的[流程图](#流程图)。

void StartGame(){

int shape = rand() % 7, form = rand() % 4; //随机获取方块的形状和形态

while(1){

int t = 0; //此处需要先更新下一个块，再下落

int nextShape = rand() % 7, nextForm = rand() % 4; //随机获取下一个方块的形状和形态

int x = (COL - 8) / 2 , y = 0; //方块初始下落位置的横纵坐标

color(nextShape); //颜色设置为下一个方块的颜色

DrawBlock(nextShape, nextForm, COL + 3, 3); //将下一个方块显示在右上角

while(1){

color(shape); //颜色设置为当前正在下落的方块

DrawBlock(shape, form, x, y); //将该方块显示在初始位置

if(t == 0){

t = Speed; //规定下落一格的速度

}

while(--t){ //此处必须是--t

if(kbhit() != 0) //下落一格的时间之内进行了其他操作

break;

}

if(t == 0){ //玩家未进行操作，或者方块已经到底

if(IsLegal(shape, form, x, y + 1) == 0){ //到底部：更新游戏区地图的状态

int i, j;

for(i = 0; i < 4; i++){

for(j = 0; j < 4; j++){

if(block[shape][form].space[i][j] == 1){

area.data[y + i][x + j] = 1;

area.color[y + i][x + j] = shape;

}

}

}

while(IsCount());

break;

}

else { //未到底部

EraseBlock(shape, form, x, y); y++; //准备移动

}

}

else { //进行了其他操作

char ch = getch();

switch(ch){

case DOWN: //方向键：下

if(IsLegal(shape, form, x, y + 1) == 1){

EraseBlock(shape, form, x, y); y += 1;

}

break;

case LEFT: //方向键：左

if(IsLegal(shape, form, x - 1, y) == 1){

EraseBlock(shape, form, x, y); x--;

}

break;

case RIGHT: //方向键：右

if(IsLegal(shape, form, x + 1, y) == 1){

EraseBlock(shape, form, x, y); x++;

}

break;

case TAB: //Tab键

if(IsLegal(shape, (form + 1) % 4, x, y + 1) == 1){

EraseBlock(shape, form, x, y);

form = (form + 1) % 4;

y++;

}

break;

case ESC: //Esc键

system("cls");

color(7);

Gotoxy(COL, ROW / 2);

printf(" 游戏结束 ");

Gotoxy(COL, ROW / 2 + 2);

exit(0);

case 's':

case 'S':

SPause();

break;

case 'r':

case 'R':

system("cls");

main(); //重新游戏

}

}

}

shape = nextShape, form = nextForm;

EraseBlock(nextShape, nextForm, COL + 3, 3); //将右上角的方块信息用空格覆盖

}

}

2.17从文件读取最高分

首次游戏时自动创建最高分纪录文件“俄罗斯方块最高得分记录.txt”，并初始化最高分为0。此后进行游戏时，读取文件当中的历史最高记录并储存在max变量中。

void ReadRecord(){

FILE\* pf = fopen("俄罗斯方块最高得分记录.txt", "r");

if (pf == NULL){ //未创建文件

pf = fopen("俄罗斯方块最高得分记录.txt", "w");

fwrite(&Record, sizeof(int), 1, pf); //将max写入文件（此时max为0），即将最高历史得分初始化为0

}

fread(&Max, sizeof(int), 1, pf); //读取文件中的最高历史得分

fclose(pf);

pf = NULL;

}

2.18更新最高分到文件

用fopen函数打开最高分文件“俄罗斯方块最高得分记录.txt”，并将本局的得分覆盖到文件中。

void WriteRecord(){

FILE\* pf = fopen("俄罗斯方块最高得分记录.txt", "w");

if (pf == NULL) {

printf("保存最高得分记录失败\n");

exit(0);

}

fwrite(&Record, sizeof(int), 1, pf); //更新最高历史得分

fclose(pf);

pf = NULL;

}

2.19加分小动画

1. 播放得分效果音。

2. 光标定位到分数的地方，并打印“+=10”，设定一个时间值，然后用空格覆盖原来的语句，再在偏高的位置打印“+=10”，再用空格覆盖掉，以此来达到动画移动的效果。

void Congratulations(){

mciSendString("play Con.wav", NULL, 0, NULL);

color(12);

Gotoxy(2 \* COL + 16, ROW - 3);

printf("+=10");

Sleep(80);

Gotoxy(2 \* COL + 16, ROW - 3);

printf(" ");

Gotoxy(2 \* COL + 16, ROW - 4);

printf("+=10");

Sleep(80);

Gotoxy(2 \* COL + 16, ROW - 4);

printf(" ");

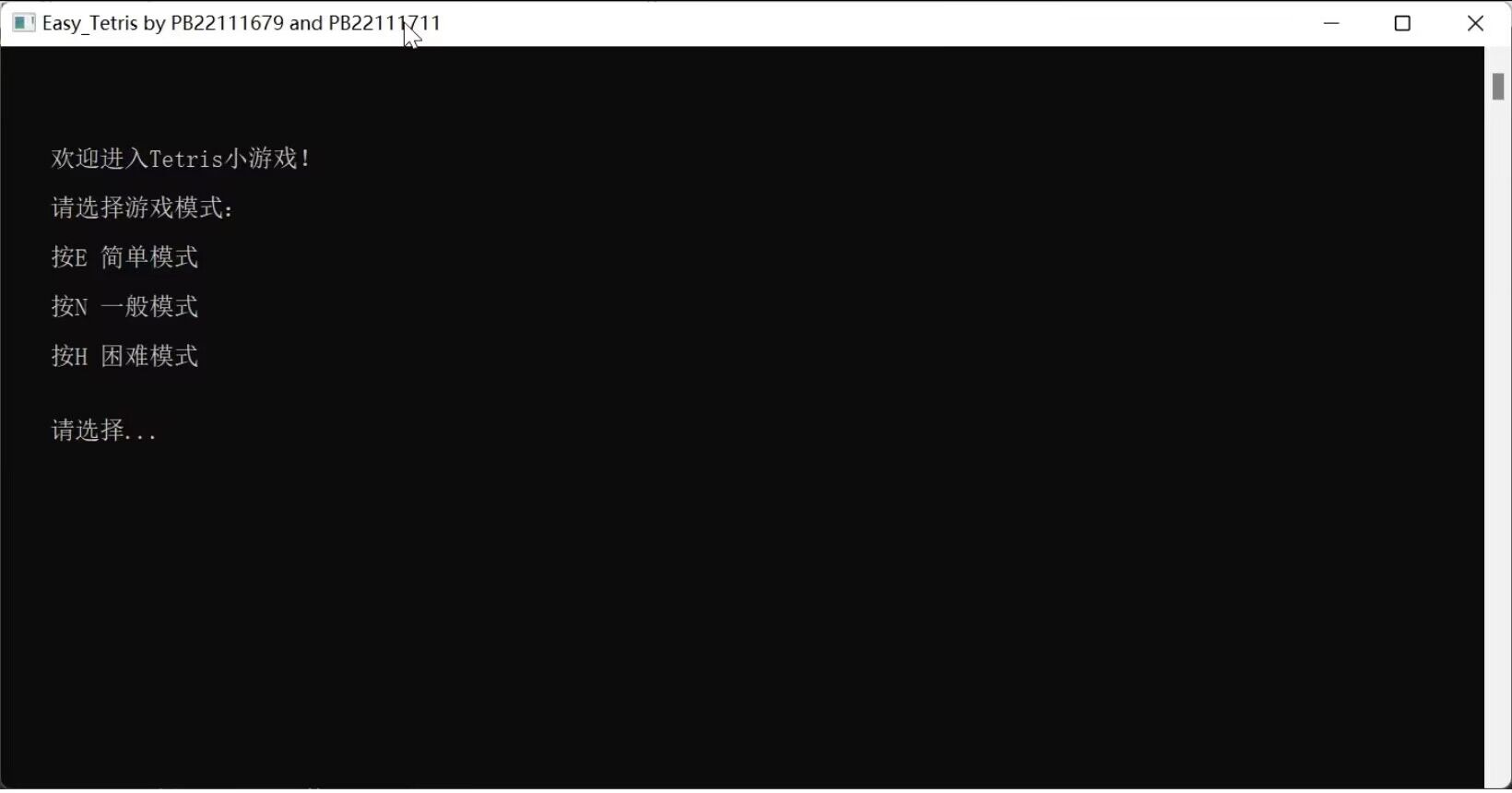
}

3.功能展示 ————————————————————————————[返回目录](#目录)

此部分使用GIF动画展示。

3.1主菜单选择

游戏开始界面，可以选择游戏模式，开始播放背景音，不同模式方块下落速度有所不同。



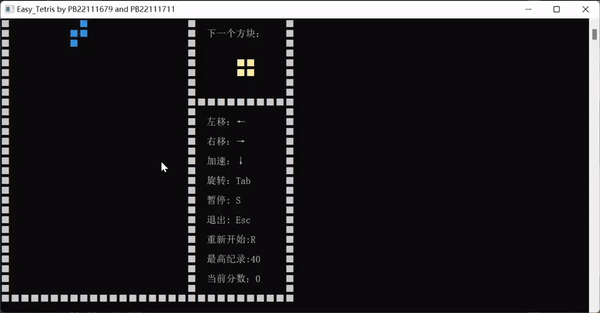
简单模式



一般模式



困难模式

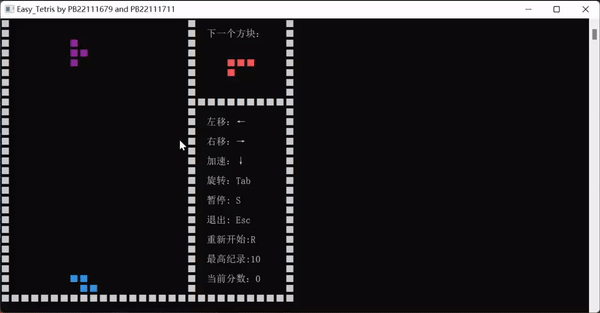


选择游戏模式时，按键错误时可重新选择

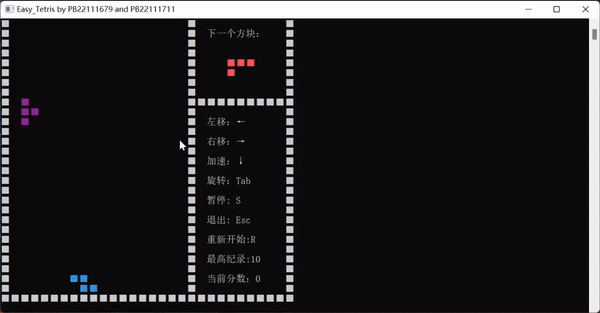


3.2 按键变换

左移 ；按“←”实现左移。



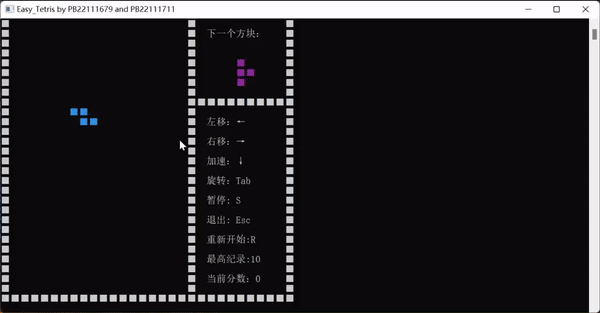
右移 ；按“→”实现右移。



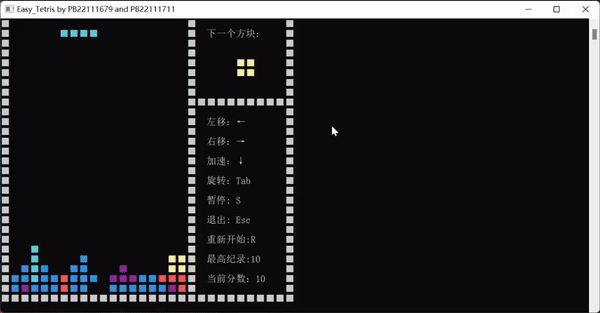
加速 ；按“↓”实现加速。



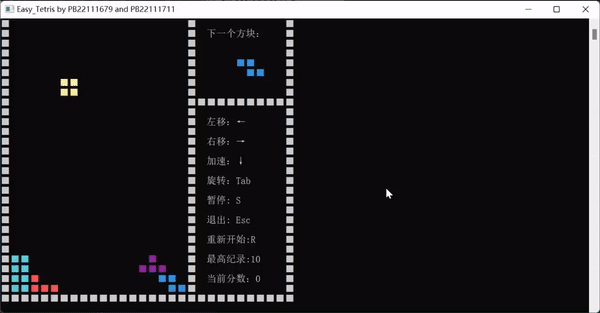
旋转 ；按“Tab”实现旋转。



暂停 ；按“S”实现暂停。

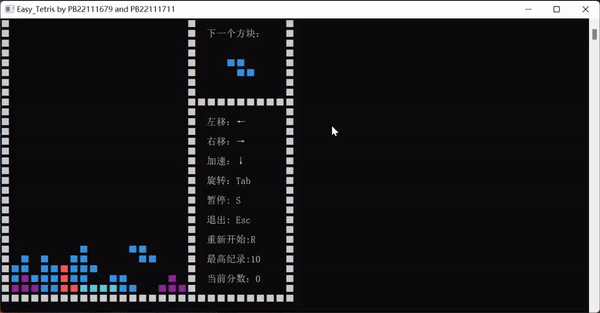


重新开始：按“R”实现重新开始。



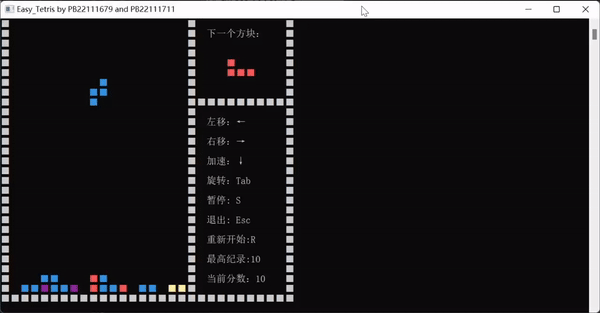
3.3满格消格

当最下方一行满格，实现：1.自动消格；2.判断得分并加分；3.加分背景音，加分小动画



3.4合理性判断

判断方块移动的合理性，当方块触及边缘无法移动，或者到底时，即使按键也不进行相应变换。

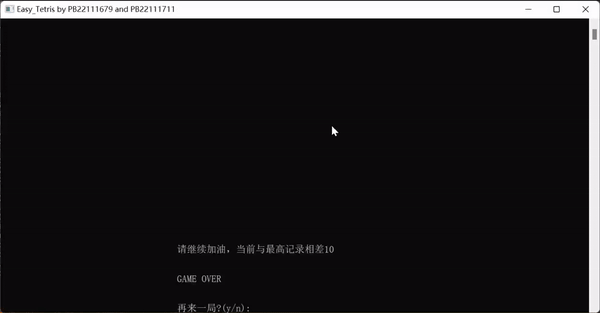


3.5游戏结束

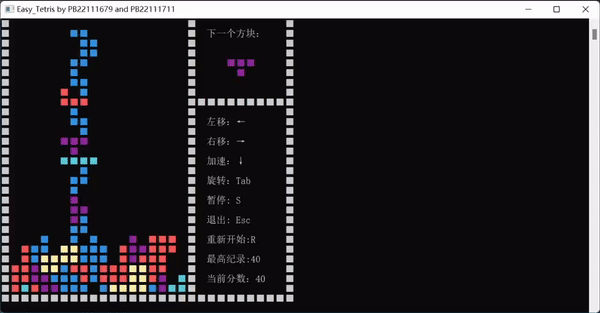
当方块堆叠到顶，判断游戏结束，记录分数。



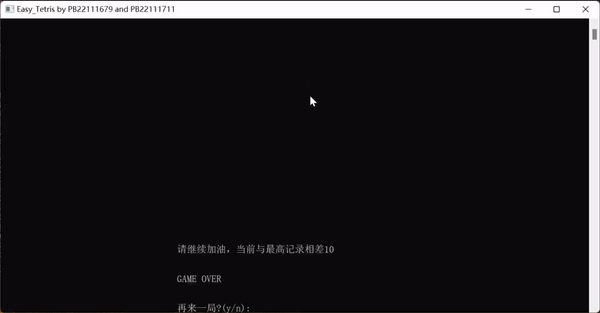
询问是否再来一局，玩家自行选择。



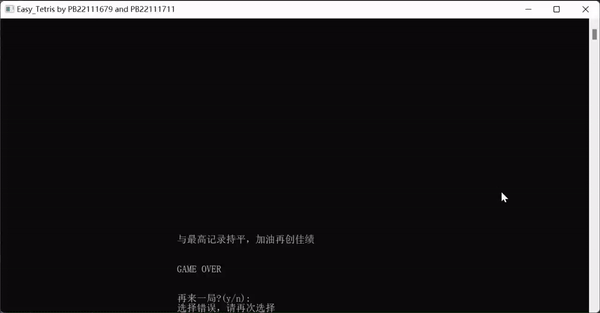
（1）选择错误可再次选择



（2）选择再来一局

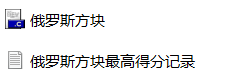


（3）直接结束游戏



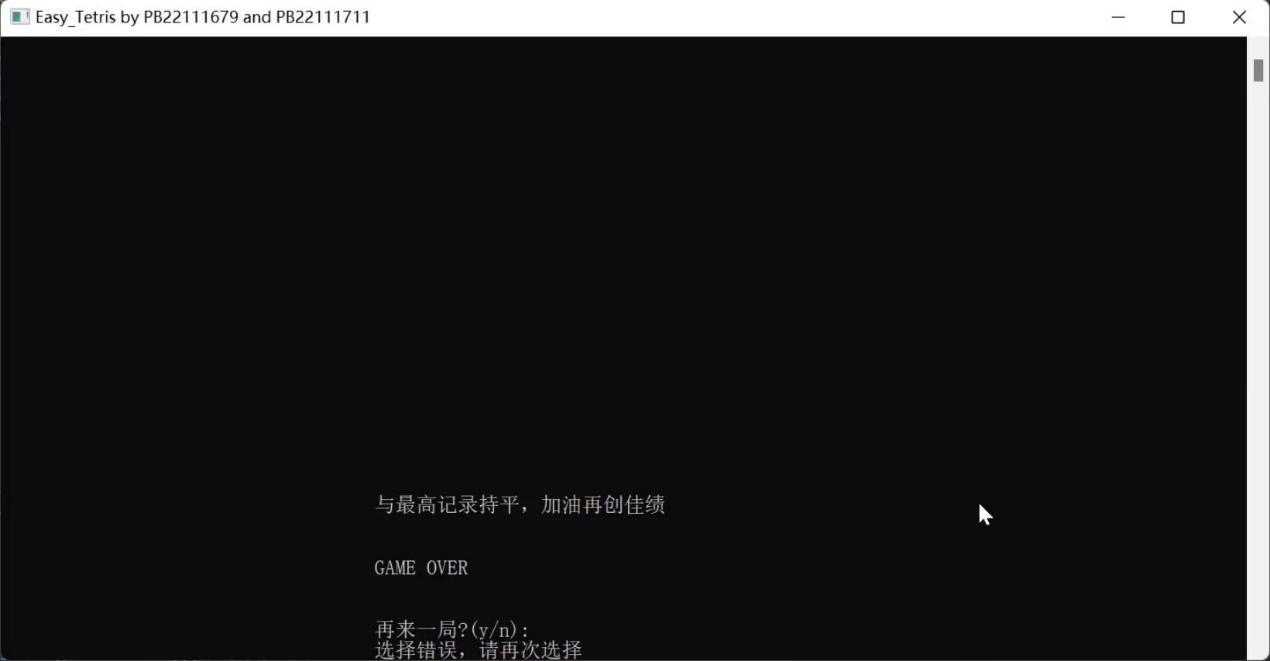
3.6判断分数

首次游戏，自动创建“俄罗斯方块最高得分记录.txt”文档，用于储存游戏分数，并实时更新最高分数。



3.6-1 游戏文件夹内部示意图

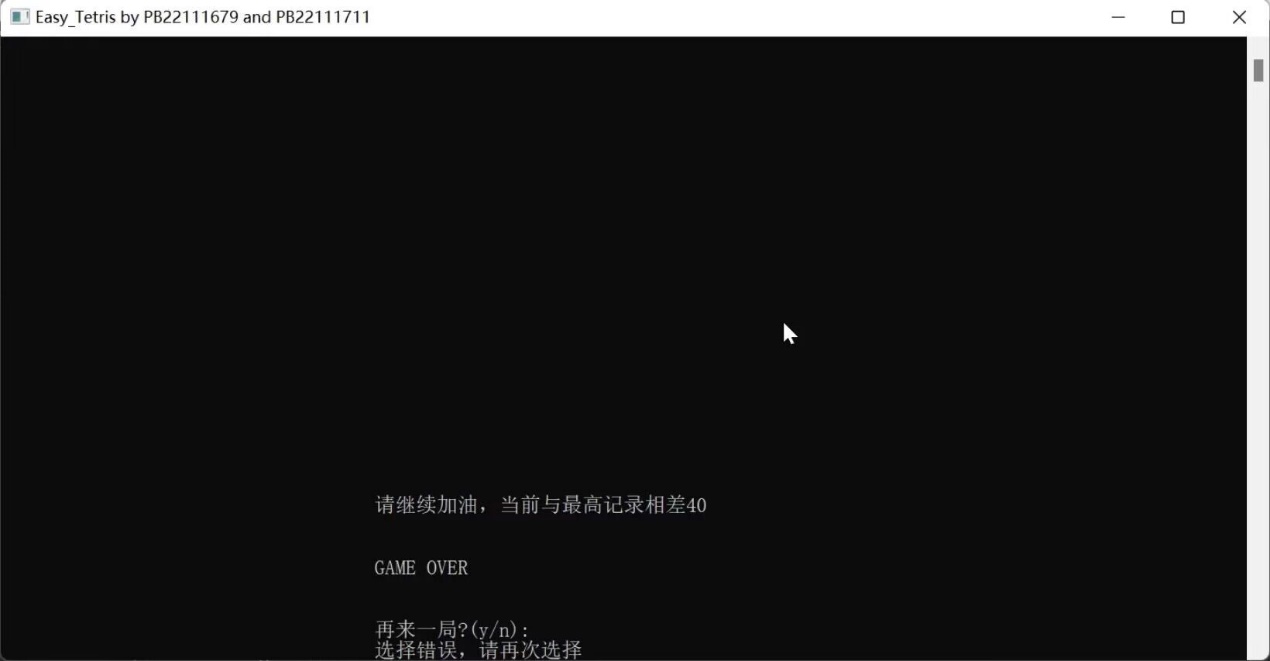
（1）与最高分持平



（2）打破最高分，自动更新最高纪录



（3）判断与最高纪录相差多少分，并进行鼓励。



4.存在的问题 ———————————————————————————[返回目录](#目录)

我们在邀请同学进行游戏测评的过程中发现，如果持续按左键或右键，会导致方块下落的速度变快。这是因为在StartGame()函数中，每一次检测键盘信息的时间由t决定，而持续按键导致t没有充分自减为0。这与t = Speed的设置形成了矛盾。

我们借鉴了网络帖子的经验[2]，但问题仍然存在。经分析，原因是DevC++中程序语句是顺序执行的，若要控制任何按键情况下时间均匀流逝，可能需要双线程设置，即t与检测键盘分开进行。我们暂时没有找到解决方法或者可移植的经验。

5.团队分工 ———————————————————————————[返回目录](#目录)

陈昕琪：编写和优化代码；撰写实验报告2、3部分，制作GIF；查找资料。

孙婧雯：编写代码和优化；撰写实验报告其他部分并排版；查找资料。

6.总结和建议 ——————————————————————————[返回目录](#目录)

本次实验制作游戏和撰写报告共耗时15天，团队内两人的代码能力、检索能力以及其他实验报告技能都得到了锻炼和提升，从搭建框架、分块编程到插入音乐、优化细节，收获颇丰。就目前而言，我们编写C语言小程序或小游戏的能力还有很大提升空间。同时，学期内所学到的用函数操作文件、结构体知识也得到了实践，不过链表与指针的应用仍然欠缺，可以在实验之外加以练习。

建议助教和老师给予更多的讲解，例如课本上的SetConsoleCursorPosition()并没有任何提示就出现在参考程序中，容易给初学者造成误解，网络上的资料也良莠不齐。

7.参考资料 ————————————————————————————[返回目录](#目录)

[1] 王雷,王百宗,李玉虎,刘勇.计算机程序设计学习实践-实验指导书.合肥:中国科学技术大学出版社.2022:202-209.

参考了书中贪吃蛇示例程序SetConsoleCursorPosition(),COORD,HANDLE等的用法.

[2] https://blog.csdn.net/qq\_54169998/article/details/122800521

参考了文章中每接收一次玩家键入就延时等待的做法，试图解决 存在的问题 中所述的持续按键导致下落变快的问题.