# 详细设计

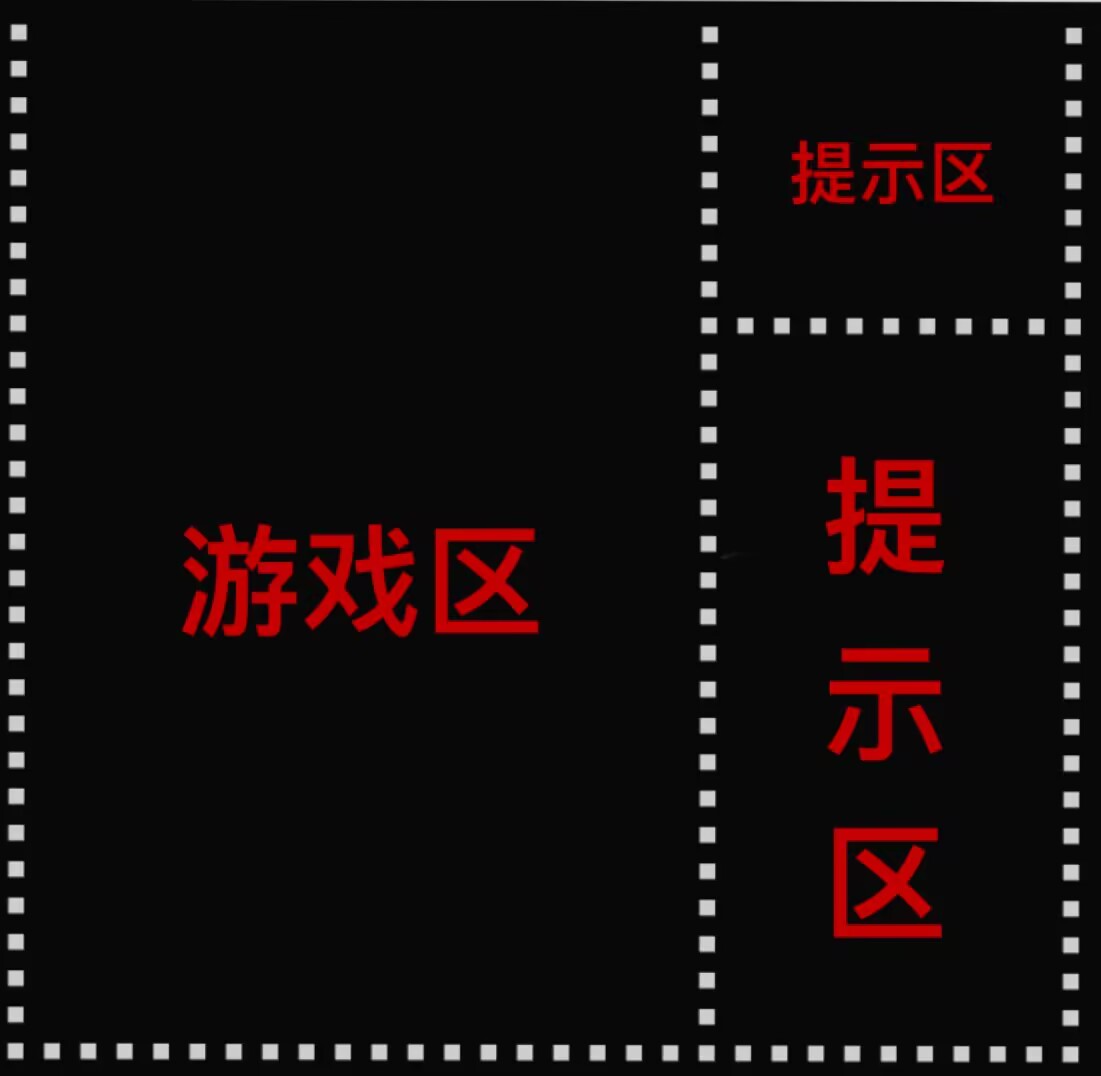
## 设置图形区域大小：

1.首先定义一下游戏区域大小，定义行数和列数。

#define ROW 28 + 1 //游戏区行数

#define COL 18 + 2 //游戏区列数

界面分为游戏区和提示区，方块堆积的地方叫做游戏区，提示按键以及下一个方块的地区叫做提示区。



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，这个二维数组就用于存储单个方块的基本信息。对于7种基本形状的方块，每种方块有通过顺时针旋转有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

## 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;

}

## 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;

}

}

}

## 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);

}

## 隐藏光标

进行游戏时不需要用到光标，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：光标结构体

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，注意这是个指针类型

);

BOOL SetConsoleCursorInfo( //设置光标信息

HANDLE hConsoleOutput, //句柄

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

);

## 光标跳转

游戏进行时，使光标可以跳转到任意位置需要光标跳转函数。（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：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

);

## 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（实现操作所需按键）

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)//绘制竖直边框

{

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

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("■");

}

//打印旁栏提示

Gotoxy(2 \* COL + 4, 1);

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);

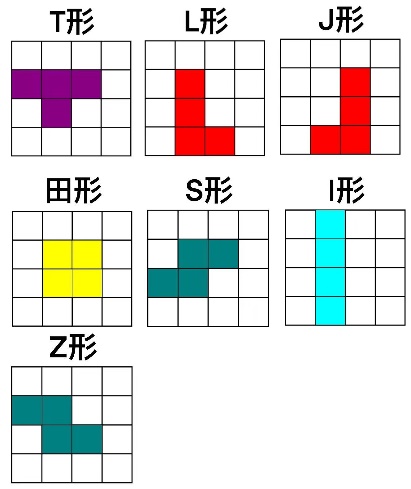
Gotoxy(2 \* COL + 4, ROW - 3);

printf("当前分数：%d", Record);

}

## 初始化方块信息

初始化七个方块如图：



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];

}

}

}

}

}

## 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。

|  |  |
| --- | --- |
| 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);

}

## 画出方块

借助存储在4×8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

}

}

}

}

## 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，这里用到的擦除方块的方法是，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) //如果该位置有方块

{

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）

}

}

}

}

## 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落在了本来就有方块的位置）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

}

## 判断是否得分

1. 从下向上判断是否满格，若有一行方块满了，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），然后将上面的方块全部下移一行。
2. 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]; //统计第i行的方块个数

}

if(sum == 0) //该行没有方块，无需再判断其上的层次（无需再继续判断是否得分）

break; //跳出循环

if(sum == COL - 2) //该行全是方块，可得分

{

Record += 10; //满一行加10分

Congratulations();

color(7); //颜色设置为白色

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

printf("当前分数：%d", Record); //更新当前分数

for(j = 1; j < COL - 1; j++) //清除得分行的方块信息

{

area.data[i][j] = 0; //该位置得分后被清除，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]; //统计上一行的方块个数

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) //上一行移下来的是方块，打印方块

{

Gotoxy(2 \* k, j); //光标跳转到该位置

color(area.color[j][k]); //颜色设置为还方块的颜色

printf("■"); //打印方块

}

else //上一行移下来的是空格，打印空格

{

Gotoxy(2 \* k, j); //光标跳转到该位置

printf(" "); //打印空格（两个空格）

}

}

if (sum == 0) //上一行移下来的全是空格，无需再将上层的方块向下移动（移动结束）

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

}

## 判断是否结束

1. 直接检测最上方一行有没有方块，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. 询问玩家是否想再来一局，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();

}

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

{

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

exit(0);

}

else

{

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

printf("选择错误，请再次选择");

}

}

}

}

}

## 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("■");

}

}

}

## 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。

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下落位置

if(t == 0)

{

t = Speed; //规定下落一格的速度

}

while(--t) //这里必须是--t，不然之后t一直是负的

{

if(kbhit() != 0) //下落一格的时间之内进行了其他操作（余下的时间足以忽略）

break;

}

if(t == 0) //玩家未进行操作，或者方块已经到底了

{

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) //判断方块向下移动一位后是否合法

{

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

}

break;

case LEFT: //方向键：左

if(IsLegal(shape, form, x - 1, y) == 1) //判断方块向左移动一位后是否合法

{

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

}

break;

case RIGHT: //方向键：右

if(IsLegal(shape, form, x + 1, y) == 1) //判断方块向右移动一位后是否合法

{

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

}

break;

case TAB: //空格键

if(IsLegal(shape, (form + 1) % 4, x, y + 1) == 1) //判断方块旋转后是否合法

{

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

}

//while(--t);

}

}

shape = nextShape, form = nextForm;

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

}

}

## 从文件读取最高分

首次玩时，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，关闭文件。

void ReadRecord()

{

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

if (pf == NULL) //还没有创建文件

{

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到max当中

fclose(pf);

pf = NULL;

}

## 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grade覆盖式储存到文件中。

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;

}

## 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);

//mciSendString("close 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(" ");

}