

****

**2018-2019程序设计基础**

**第21组程序报告**

**选题：**  **俄罗斯方块**

**组长姓名： 顾哲涵 3180102065**

**组员姓名： 宗威旭 3180102776**

**指导教师： 应晶**

**提交日期： 2019年1月11日**

**浙江大学2018–2019 学年冬学期**

**《程序设计基础》课程程序报告**

**课程号**： 211Z0040 \_，**开课学院**： 计算机科学与技术学院

**考查形式**： 程序检查 ，

**考查日期**： 2019 年 1 月 11 日

**负责人姓名： 顾哲涵**  **学号： 3180102065**

**所属院系：** 竺可桢学院**\_ 联系方式：** 15306530508 **\_**

目录

[1 程序功能说明 3](#_Toc534944468)

[1.1 设计说明 3](#_Toc534944469)

[1.2 玩法介绍 4](#_Toc534944470)

[2 程序设计思路 5](#_Toc534944471)

[2.1 数据结构定义 5](#_Toc534944473)

[2.1.1 TetrisTable 数组 5](#_Toc534944474)

[2.1.2 InitialPool 数组 5](#_Toc534944475)

[2.1.3 TetrisState，TetrisOperation结构 6](#_Toc534944476)

[2.2 功能模块设计 7](#_Toc534944477)

[2.2.1 主程序流程 7](#_Toc534944478)

[2.2.2 运行游戏 7](#_Toc534944479)

[2.2.3 方块操纵 7](#_Toc534944480)

[2.2.4 游戏板块控制 9](#_Toc534944481)

[2.2.5 碰撞检测 11](#_Toc534944482)

[2.2.6 满行检测 12](#_Toc534944483)

[2.2.7 游戏功能设计 12](#_Toc534944484)

[2.2.8 游戏界面设计 13](#_Toc534944485)

[3 难点分析与解决方案 16](#_Toc534944486)

[4 总结与分析 17](#_Toc534944487)

[5 组内人员分工 18](#_Toc534944488)

[6 附录（源代码） 18](#_Toc534944489)

# 程序功能说明

## 设计说明

俄罗斯方块（Tetris）于1984年由阿列克谢·帕基特诺夫发明，因为规则简单而趣味性强风靡一时。俄罗斯方块原名是俄语Тетрис，这个名字来源于希腊语tetra，意思是“四”，而游戏的作者最喜欢网球（tennis）。于是，他把两个词tetra和tennis合而为一，命名为Tetris，这也就是俄罗斯方块名字的由来。

在我们的程序中，屏幕中央会随机产生以“I”,“J”,“L”,“O”,“S”,“Z”,“T”等基本形状的二十八种由四个方块组成的板块，玩家可以通过向左移动，向右移动，旋转，加速下落，硬降等方式调整板块的位置和方向，使它们在屏幕底部拼出完整的一条或几条。这些完整的横条会随即消失，给新落下来的板块腾出空间，与此同时，玩家得到分数奖励。没有被消除掉的方块不断堆积起来，一旦堆到屏幕顶端，玩家便告输，游戏结束。此外，程序设计了暂停、重新开始、退出等功能，满足玩家的游戏体验。玩家分数提高的同时，方块的下落速度也会对应增加，以增加游戏难度，满足玩家对于游戏难度的需要。

设计程序界面的过程中，我们秉持界面简洁而明丽的原则。我们的界面文字包含对游戏操作的必要说明，玩家消除各方块数量及总方块数量以及玩家当前获得分数，使界面避免繁琐的细节，设计了显示下两个方块的出现提示，简洁明了。每种基本类型的方块保持不同的颜色，以增加画面的可观赏性。

学习程序设计基础课程过程中，组内成员根据所学内容及相关代码的学习完成程序设计。希望玩家在方块的建与除中收获乐趣。

## 玩法介绍

玩家可以通过按W/8/↑键实现板块顺时针旋转，按z键实现板块逆时针旋转，按A/4/←键实现板块左移，按D/6/→键实现板块右移，按S/2/↓键实现向下移动，按空格键硬降，按回车键暂停游戏。如果出现满行，则该行会消失，同时玩家得到加分。如果消除一行，得分加10；同时消除两行，则得分加30；同时消除三行，则得分加90；如果同时消除4行，则得分加150。如果方块堆积到屏幕顶端，则游戏结束。

游戏目的，便是通过高超的游戏技巧，得到更高的分数。

# 程序设计思路



## 数据结构定义

### TetrisTable 数组

**使用状态压缩，将板块的每一种姿态（类型和旋转取向）用一个16位整数来表示，并存入7（种类）\*4（取向）的二维常量数组中。**

我们通过定义以静态常量的短整型（16位）为元素的二维数组，记录了七种基本类型，以及其通过旋转点旋转后得到的每一种对应的其他三种旋转类型。以十六进制数字代表板块，以四位为板块所在4\*4矩阵的一行，也就是十六进制的一位。最高的四位表示矩阵的第一行，以此类推，最低四位表示最后一行。其中，每一位的1代表此处有方块，0代表没有方块。例如：“0x00F0U”代表矩阵的第三行充满方块，也就是横着的“I”形板块。

### InitialPool 数组

**使用hash表，将一行的状态（每格有无方块）用一个16位整数来表示。**

我们通过定义以静态常量的短整型型（16位）为元素的二维数组，定义了初始状态的游戏池。里面的每个元素代表游戏池的一行，共28行。其中，顶部四行用于方块准备落下，以实现方块逐渐出现的效果。底部放置两行，两端各置两行，将这些方块对应的数位为1，来创造基本的游戏池并检测后续的输入会不会使板块碰撞。玩家可进行方块堆叠的区域范围共22行12列。

同时，我们定义COLUMN\_BEGIN=2，COLUMN\_END=14，ROW\_BEGIN=4，ROW\_END=26。记录游戏池边界。

### TetrisState，TetrisOperation结构

我们定义了两个结构体，分别存储游戏相关数据、存储控制相关数据，前者记录方块、游戏池状态，游戏运行信息，后者记录用户输入导致的数据改变以及颜色信息。

在TetrisState中，我们存储了当前游戏池pool数组，以及当前方块所在4\*4区域的左上角的x,y坐标（并且该坐标为旋转点），用type[3]，orientation[3]数组分别记录当前及以后两个方块的旋转状态。定义正数score记录分值，定义erasedCount[4]分别记录一次消除若干行的次数，定义erasedTotal记录消行总数，tetrisTotal记录出现方块总数，dead判断游戏是否结束。

在TetrisOperation中，定义bool类型pause来实现暂停功能。clockwise定义旋转方向。direction代表左、右的移动方向，其中0表示向左移动，1表示向右移动。此外，定义了数组color实现用户移动时对于颜色信息的储存以及更新。

## 功能模块设计

### 主程序流程

进入主程序后，定义TetrisState，TetrisOperation结构初始化数据，并调用初始化函数对游戏初始化后，显示游戏信息和初始游戏池，而后进入游戏运行。检测游戏结束后，主函数通过调用ifPlayAgain函数检测用户是否希望重新开始游戏。如果重新开始，则通过SetConsoleTextAttribute语句重置控制台，同时清屏，调用StartGame函数重新开始；否则关闭工作台，退出程序。

### 运行游戏

在游戏运行函数runGame中，我们定义clock\_t类型变量clockLast, clockNow记录两次时间，并根据当前得分来控制下落速度，来达到难度递增的效果。首先通过clockLast获取初始时间开始计时，并通过printTetrisPool函数显示游戏池。当游戏没有结束时，检测是否产生按键；如果产生按键，则调用PressProcess函数处理按键。如果并未按下暂停键，且两次计时的间隔超过delay秒（delay由score决定），则将方块自由下落，等价为按了一次“↓”键。注意，程序将判断碰撞与满行的步骤在下移函数中进行，保障程序正确运行。在游戏没有结束时，程序将反复进行RunGame函数。

### 方块操纵

1. 生成一个板块：程序通过调用FormTetris函数得到一个新的板块。首先，将备用的两个板块依次前置一个，并生成一个新的下下个备用方块。定义短整型变量tetris记录取出的当前方块的类型和旋转状态。并根据板块类型的高度确定它的初始y坐标。方块只在游戏池中显示一行，其余部分位于上方缓冲区中，并把横坐标置于游戏池中央。先检查是否碰撞，如果碰撞，则判定游戏结束。否则，通过调用insertTetris函数将方块插入游戏池。同时，对方块总数、相应方块数更新计数，并通过调用printNextTetris和 printScore函数显示下一个方块和得分信息。此外，函数在重新开始，判断满行函数后才会给出新的方块。
2. 插入方块：仍然定义tetris保存取出板块的状态。通过对于tetris进行位运算，从上到下依次取出矩阵每一行的状态，再将其通过位或运算复制到游戏池对应位置，即插入游戏池。
3. 移除方块：用tetris保存取出的板块的状态。通过对tetris的位运算从上到下依次取出每一行的状态，再将通过取反运算将对应位置变为0，复制入游戏池对应位置，即移出游戏池。
4. 设置颜色：tetris保存板块状态。用i从0到15循环枚举矩阵的每一格，如果没有方块则不用染色，跳过。对于池底的格子也不用染色。

### 游戏板块控制

对于游戏过程中，板块包括左移、右移、下移、旋转、硬降这些操控的实现。

首先，我们定义了PressProcess函数，对于玩家按键进行相应。其中，“W/8/↑（小键盘）”表示顺时针旋转方块，并把clockwise标记赋值为true，进入rotateTetris旋转函数；“z”表示逆时针旋转方块，并把clockwise赋值为false，进入rotateTetris函数。“A/4/←（小键盘）”表示左移方块，并把direction标记为0，进入FlatlyMoveTetris函数；“D/6/→（小键盘）”表示右移方块，并把direction标记为1，进入FlatlyMoveTetris函数。“S/2/↓（小键盘）”表示下移方块，进入DownMoveTetris 函数；空格键表示硬降，进入dropDownTetris函数（硬降）。

1. **左（右）移**：考虑到大多数的时候，玩家都在上方进行移动定位，贴壁移动较少。因此直接假设移动是允许的，移动后判断是否发生碰撞。如果发生了，那么再把本次移动取消，恢复原有状态（这是较少数的情况）。

**具体实现：**检测到玩家输入“A/4/←”（“D/6/→”）后，进入移动函数。记录原来的x，并通过removeTetris函数移除当前板块。根据direction的值判断向左（右）移动，对于x的值进行自减（增）。而后检测是否碰撞，如果发生碰撞，则通过insertTetris函数在原列位置插入原板块；否则通过insertTetris函数在当前位置插入当前板块，以及通过setPoolColo函数设置颜色，调用printCurrentTetris函数显示当前板块。

1. **下移：**原理同水平移动，区别在于不用判断方向（不存在向上移动）并且需要考虑是否出现消行。

**具体实现：**检测到玩家输入“S/s/2（小键盘）”后，进入DownMoveTetris 函数。通过变量x记录原列坐标（当前板块左上角坐标），并通过removeTetris函数移除当前板块。对坐标y进行自增。而后检测是否碰撞，如果发生碰撞，则通过insertTetris函数在原列位置插入原板块。另外，我们将满行检测设置在下移操作处；通过checkErasing函数判断是否满行。如果满行，则进入printTetrisPool函数显示消除后游戏池。如果没有发生碰撞，则通过insertTetris函数在当前位置插入当前板块，以及通过setPoolColo函数设置颜色，调用printCurrentTetris函数显示当前板块。

1. **硬降：**

检测到玩家输入“空格”后，进入函数。首先移除当前板块。从当前y开始逐行向下扫描至池底；如果检测到碰撞，则跳出循环。注意，由于此时方块的位置发生重叠，故应将y进行自增，再通过insertTetris函数在当前位置插入当前板块，以及通过SetColor函数设置颜色。同样，我们也需要在插入板块后通过printTetrisPool函数进行满行检测，并通过printTetrisPool函数显示游戏池。

1. **旋转方块：**同样，大多数的旋转不会碰壁。因此假定旋转合法，进行操作后再进行判断：如果发生碰撞则旋转不合法，恢复原状。

**具体实现：**检测到玩家输入“W/w/8（小键盘）”后，进入rotateTetris函数。通过定义变量ori记录原板块旋转状态，并通过removeTetris函数移除当前板块。Clockwise的值为true（false），进行对ori加一取三的余数（ori加三取三的余数）。而后检测是否碰撞，如果发生碰撞，则通过insertTetris函数在原列位置插入原板块；否则通过insertTetris函数在当前位置插入当前板块，以及通过setPoolColo函数设置颜色，调用printCurrentTetris函数显示当前板块。

### 碰撞检测

用16位整数dest状态压缩保存板块所在4\*4区域对应游戏池的状态。同样用tetris取出板块区域的状态，两者按位与，如果得到的不是0，说明有重叠，即发生了碰撞。

**具体实现：**通过checkCollision函数对当前板块是否发生碰撞进行检测。首先定义短整型tetris得到当前板块的位置状态与旋转状态，以及用于得到方块所在区域的变量dest。首先，取游戏池第y行第x列（即模块的左上角），通过左移得到第四行，通过位交运算得到该行中游戏池有方块的位置，并赋值给的dest对应1~4位。而后依次进行左移运算，依次得到第三行，第二行，第一行中游戏池有方块的位置，并赋值给dest对应的5~8，9~12，13~16位，然后与tetris取位交运算，如果交不为0，说明有碰撞，返回1；否则无碰撞，返回0。

### 满行检测

**从下往上扫描板块区域对应游戏池的四行逐行检查：如果满行则用memmove将该行以上的pool值往下平移一行，并更新得分等记录。**

**具体实现：**调用checkErasing函数检测满行。先定义数组score定义消除行数对应得分，定义变量count与k分别记录消除行数与移动行数。对所取坐标点（左上角）所在行开始计算的以下四行自下而上进行遍历，检查是否满行，即pool[y]的值是否为0xFFFFU，如果是，则通过memmove语句将上面一行对下面一行的方块状态以及颜色进行覆盖，并将积分器count自增；否则将坐标指向下一行，同时对计数器k进行自增，当检查完4行时跳出循环，并根据count的值计算出玩家的最新的分以及消行总数。满行检查后通过FormTetris函数得到下一个板块，并通过SetColor函数对板块设置颜色。

### 游戏功能设计

**使用pause标记暂停状态，在下落函数和按键函数时都判断pause是否为真，如果暂停则不作响应。**

**具体实现：**

1. 暂停：检测到玩家输入暂停键“\n”（回车）时，程序中runGame函数调用PressProcess函数判断key的值是否等于回车对应的值。检测到用户输入回车后，游戏将交换pause的值，实现暂停与解除的切换。否则返回runGame函数继续进行循环。
2. 重新开始游戏：程序通过ifPlayAgain函数以及StartGame函数实现重新开始。首先，主程序在游戏结束后中判断ifPlayAgain的条件是否为真。如果为真，则返回true；之后清屏，并进入StartGame函数。而后通过memset语句将state与operation结构初始化。并通过memcpy对相应数据保存。随机生成下面三个方块后，通过FormTetris函数得到下一个方块，通过SetColor函数设置颜色。如果为假，则返回false退出程序。

### 游戏界面设计

1. 定位函数

程序通过GotoXY函数实现光标移动。

1. 界面初始化

程序通过initGame函数初始化游戏。

CONSOLE\_CURSOR\_INFO cursorInfo = { 1, 0 }；Console\_output=GetStdHandle(STD\_OUTPUT\_HANDLE)；SetConsoleCursorInfo(Console\_output, &cursorInfo)；

这一语句块将光标隐藏，并调用SetConsoleTitleA函数更改程序名称。

1. 显示游戏边界

程序通过调用printPoolBorder函数显示游戏池边界。通过语句SetConsoleTextAttribute(Console\_output, 0xF0)对方块进行颜色设置。遍历ROW\_BEGIN到ROW\_END间的每一行，将每一行对应的左、右两列边界的最左边输出两个有色空格，来设置边界。对于底边，输出28列有色空格，完成底边填充。

1. 显示游戏池

遍历xy，判断有方块则输出■否则输出空白

**具体实现：**首先，先定义一个定位函数XYinPool(x, y)= GotoXY(x + 9, y - 3)来建立窗口坐标和池中坐标的联系。

而后以（2，ROW\_BEGIN）为初始位置对y与x分别进行遍历，如果游戏池中某一位置对应的数位右移到第一位后与1有交，则表示该位置存在方块，则通过SetConsoleTextAttribute(Console\_output,operation->color[y][x]);

printf("■");语句显示一个对应颜色的实心方块。否则通过语句SetConsoleTextAttribute(Console\_output, 0);printf("%2s", "")显示空白表示此处没有方块。

显示游戏池用于运行、下落、直接下落部分。

1. 显示当前板块：

考虑到该函数用于板块变化位置后，需要删除变化前的板块，因此删除前所在位置的池子状态也会改变，因此除了除了扫描44区域的每一格，还要根据移动的方向拓展扫描区域。

1. 显示下一个和下下一个板块

**具体实现：**程序通过调用printNextTetris函数实现显示下面两个方块。首先，通过SetConsoleTextAttribute(Console\_output, 0xF);语句设置颜色，通过GotoXY和printf的语句组合显示边框。另外，定义局部变量tetris得到下一个方块的类型和旋转状态，通过SetConsoleTextAttribute(Console\_output, state->type[1] | 8)语句给方块填充颜色，而后用i遍历数位得到有方块的位置，在相应的位置输出有色方块。并用同样的方法，在下个方块旁边显示下下个（无色）方块。

1. 提示信息显示

程序通过调用printPrompting函数输出提示信息，通过GotoXY和printf的语句组合对游戏的控制按键进行说明。

1. 得分信息显示

函数通过SetConsoleTextAttribute(Console\_output, 0xE)语句来改变字体颜色为白色。通过GotoXY和printf的语句组合对玩家的分数、消除总数、不同行的消除数、已显示方块总数、已显示的不同类型的方块数进行说明。其中，定义字符串tetrisName来结合state结构中的tetrisCount数组来显示每种形状板块的出现数量。

1. 再玩一次界面

程序在ifPlayAgain函数中设计了相应页面。游戏结束后，ifPlayAgain函数通过SetConsoleTextAttribute(Console\_output, 0xF0)语句设置颜色，并通过GotoXY和printf的语句组合对玩家进行说明。

# 难点分析与解决方案

程序设计中我们遇到了一些难点，经过许多思考，我们找出了较好的解决办法。

第一，我们利用了状态压缩的思想来表示板块。用一个数字代表一行的状态，用一个16位整数表示一种板块的姿态（类型与旋转取向）。这种表示方式大大节省了程序运行的空间。游戏池的28行状态只需要int pool[28]即可表示。并且我们根据实际需要来申请变量，对于只需要16位的整数就申请short而不是int，以此来节省内存。对于会溢出的部分，多次调试之后使用了unsigned short。

但同样，这样的做法也给我们带来了麻烦：我们用游戏池和当前方块取位运算来判断该位置是否有方块的运算方法来判断是否碰撞的算法是一大难点。取出游戏池状态存入dest以及取出tetris状态的位运算公式需要经过细致的推算。这个过程花费了很长的时间和精力。

第二，自动下落与按键操作的交错，处理起来可能比较麻烦。最后我们将自动下落等价为按了一次下键，比较方便地解决了这个问题。

第三，我们定义了两个结构体来储存游戏过程中的一些参数和操作的参数，这样使得众多的数组不会显得散乱，并且在初始化时能统一处理，较为简便。

第四，为了使不同形状的板块呈现不同的颜色，我们将方块的种类序号（0-6的整数）位或 上一个“1” ，成为一个8-14的整数，将这个数作为颜色代码，方便地实现了目的。

总体上，我们采用自顶而下的构思方式，先确定了程序的总体框架，再逐步完善各个函数的具体实现方法，对于难点逐个解决。

# 总结与分析

编写几百行代码的俄罗斯方块，对于仅在程序设计基础学习的我们有些难度。我们也在十二月份才确定选题，并在一月份开始尝试编写。经过广泛的学习，并为了锻炼自身的能力，我们特意对程序进行了一些优化，虽然花费了我们很多的时间，但是我们也收获了很多，对位运算的理解大大加深，写代码和debug的能力也得到了锻炼。同时，我们也注意维护了代码风格，增强程序的可读性。在部分使用位运算优化的地方标上了注释。为了阐清我们的算法，我们用了许多时间写注释，考虑到时间有限，为了加快阅读没有使用英文注释，使得程序更容易被理解。

# 组内人员分工

顾哲涵：分配任务；总体框架搭建；算法设计；调试debug；撰写注释；结构体设计等；

宗威旭：报告文案；界面输出设计；计算板块姿态和初始游戏池的状压整数；debug等

# 附录（源代码）

**#pragma once**

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#include <conio.h>**

**#include <windows.h>**

**#include <graphics.h>**

**#define Column\_Begin 2**

**#define Column\_End 14**

**#define Row\_Begin 4**

**#define Row\_End 26**

**#include"IGNB.h"**

**// 第i行第j列代表第i种方块的第j种姿态**

**// 以4\*4的正方形为考虑对象 用十六进制数保存这4行4列的状态**

**static const short TetrisTable[7][4] =**

**{**

**{ 0x00F0, 0x2222, 0x00F0, 0x2222 }, // I型**

**{ 0x0660, 0x0660, 0x0660, 0x0660 }, // O型**

**{ 0x0223, 0x0074, 0x0622, 0x0170 }, // L型**

**{ 0x0226, 0x0470, 0x0322, 0x0071 }, // J型**

**{ 0x0072, 0x0262, 0x0270, 0x0232 }, // T型**

**{ 0x0063, 0x0264, 0x0063, 0x0264 }, // Z型**

**{ 0x006C, 0x0462, 0x006C, 0x0462 }, // S型**

**};**

**// 初始状态的游戏池**

**// 每个元素表示游戏池的一行，下标大的是游戏池底部**

**// 顶部4行作为缓冲区，不显示出来**

**// 底部2行作为池底**

**// 左右各有2列作为池壁**

**static const short InitialPool[28] =**

**{**

**0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003,**

**0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003,**

**0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xC003,**

**0xC003, 0xC003, 0xC003, 0xC003, 0xC003, 0xFFFF, 0xFFFF**

**};**

**typedef struct TetrisState// 存储游戏过程中的一些状态参数**

**{**

**unsigned short pool[28]; // 池子的状态**

**char x; // 当前4\*4正方形的左上角的X坐标**

**char y; // 当前4\*4正方形的左上角的Y坐标**

**char type[3]; // 当前0、下一个1和下下一个2方块类型**

**char orientation[3]; // 当前0、下一个1和下下一个2方块旋转状态**

**unsigned score; // 得分**

**unsigned erasedCount[4]; // 消行数**

**unsigned erasedTotal; // 消行总数**

**unsigned tetrisTotal; // 方块总数**

**bool dead; // 是否死亡**

**} TetrisState;**

**typedef struct TetrisOperation // 这个结构体存储控制相关数据**

**{**

**bool pause; // 暂停**

**bool clockwise; // 旋转方向：顺时针为true**

**char direction; // 移动方向：0向左移动 1向右移动**

**char color[28][16]; // 存储每一格的颜色信息**

**} TetrisOperation;**

**HANDLE Console\_output; // 控输出句柄**

**// 函数声明**

**void InitGame(TetrisState \*state, TetrisOperation \*operation); // 初始化游戏**

**void PrintBorder(); // 打印游戏池边框**

**void PrintScore(const TetrisState \*state); // 显示得分信息**

**void PrintInstruction(); // 显示游戏说明**

**void StartGame(TetrisState \*state, TetrisOperation \*operation); // 开始游戏**

**void RunGame(TetrisState \*state, TetrisOperation \*operation); // 运行游戏**

**void FormTetris(TetrisState \*state); // 生成一个方块**

**void InsertTetris(TetrisState \*state); // 插入方块**

**void RemoveTetris(TetrisState \*state); // 移除方块**

**void FlatMoveTetris(TetrisState \*state, TetrisOperation \*operation); // 水平移动方块**

**void DownMoveTetris(TetrisState \*state, TetrisOperation \*operation); // 向下移动方块**

**void RotateTetris(TetrisState \*state, TetrisOperation \*operation); // 旋转方块**

**bool CheckCollision(const TetrisState \*state); // 碰撞检测**

**bool CheckErasing(TetrisState \*state, TetrisOperation \*operation); // 消行检验**

**void DropDown(TetrisState \*state, TetrisOperation \*operation); // 硬降**

**void PressProcess(TetrisState \*state, TetrisOperation \*operation, int key); // 处理键盘输入**

**void SetColor(const TetrisState \*state, TetrisOperation \*operation); // 刷颜色**

**void GotoXY(short x, short y); // 定位输出位置**

**void PrintPool(const TetrisState \*state, const TetrisOperation \*operation); // 打印游戏池**

**void PrintCurrentTetris(const TetrisState \*state, const TetrisOperation \*operation); // 打印当前方块**

**void DisplayNextTetris(const TetrisState \*state); // 显示下一个和下下一个方块的提示**

**bool PlayAgain(); // 再玩一次**

**// 主函数**

**int main()**

**{**

**TetrisState tetrisState;**

**TetrisOperation tetrisOperation;**

**InitGame(&tetrisState, &tetrisOperation); // 初始化游戏**

**do // 维护游戏运行**

**{**

**PrintInstruction(); // 显示提示信息**

**PrintBorder(); // 显示游戏池边界**

**RunGame(&tetrisState, &tetrisOperation); // 运行游戏**

**if (PlayAgain()) // 再玩一次**

**{**

**SetConsoleTextAttribute(Console\_output, 0x3);**

**system("cls"); // 清屏**

**StartGame(&tetrisState, &tetrisOperation); // 重新开始游戏**

**}**

**else break;**

**} while (1);**

**GotoXY(0, 0);**

**CloseHandle(Console\_output);**

**return 0;**

**}**

**// 初始化游戏**

**void InitGame(TetrisState \*state, TetrisOperation \*operation)**

**{**

**CONSOLE\_CURSOR\_INFO cursorInfo = { 1, 0 }; // 光标信息**

**Console\_output = GetStdHandle(STD\_OUTPUT\_HANDLE); // 获取控制台输出句柄**

**SetConsoleCursorInfo(Console\_output, &cursorInfo); // 设置光标隐藏**

**SetConsoleTitleA("应晶老师太帅了！！！！！！！！");**

**StartGame(state, operation);**

**return;**

**}**

**// 重新开始游戏**

**void StartGame(TetrisState \*state, TetrisOperation \*operation)**

**{**

**memset(state, 0, sizeof(TetrisState)); // 初始状态 各个参数全部置0**

**memset(operation, 0, sizeof(TetrisOperation)); // 初始没有经过操作，操作状态置0**

**memcpy(state->pool, InitialPool, sizeof(unsigned short[28])); // 初始化池子**

**srand((unsigned)time(NULL)); // 设置随机种子**

**state->type[1] = rand() % 7; // 生成下一个方块**

**state->orientation[1] = rand() & 3;**

**state->type[2] = rand() % 7; // 生成下下一个方块**

**state->orientation[2] = rand() & 3;**

**FormTetris(state); // 给下一个方块**

**SetColor(state, operation); // 设置颜色**

**return;**

**}**

**// 生成一个方块**

**void FormTetris(TetrisState \*state)**

**{**

**unsigned short tetris;**

**state->type[0] = state->type[1]; // 下一个方块拿给当前**

**state->orientation[0] = state->orientation[1];**

**state->type[1] = state->type[2];// 下下一个方块拿给下一个**

**state->orientation[1] = state->orientation[2];**

**state->type[2] = rand() % 7;// 随机生成下下一个方块**

**state->orientation[2] = rand() & 3;**

**tetris = TetrisTable[state->type[0]][state->orientation[0]]; // 取出当前方块**

**// 根据不同方块种类的高度设置当前方块y坐标**

**// 保证最开始的时候只露出方块的最下面一行的样子**

**// 效果是使方块慢慢出现，而不是突然出现**

**// 这样可以使得方块堆积即将到顶的时候玩家不会原地暴毙，而是有机会抢救一下**

**if (tetris & 0xF000)**

**{**

**state->y = 0;**

**}**

**else**

**{**

**state->y = (tetris & 0xFF00) ? 1 : 2;**

**}**

**state->x = 6; // 出现位置在屏幕正中**

**if (CheckCollision(state)) // 检测到碰撞**

**{**

**state->dead = true; // 新出来的块就被碰撞了 自然是暴毙game over**

**}**

**else // 未检测到碰撞**

**{**

**InsertTetris(state); // 将当前方块加入池子**

**}**

**++state->tetrisTotal; // 累加方块总数**

**DisplayNextTetris(state); // 显示下一个方块**

**PrintScore(state); // 显示得分信息**

**return;**

**}**

**// ----------------------------------------------------------------------------**

**// 碰撞检测**

**bool CheckCollision(const TetrisState \*state)**

**{**

**unsigned short tetris = TetrisTable[state->type[0]][state->orientation[0]];// 取出当前方块**

**unsigned short dest = 0; //dest用来存储 方块所在的4\*4区域内的池子状态**

**//接下来分四行获取池子状态，并存储在dest中**

**//dest中每四位表示一行的状态 用按位或存到dest中**

**dest |= (((state->pool[state->y + 0] >> state->x) << 0x0) & 0x000F);**

**dest |= (((state->pool[state->y + 1] >> state->x) << 0x4) & 0x00F0);**

**dest |= (((state->pool[state->y + 2] >> state->x) << 0x8) & 0x0F00);**

**dest |= (((state->pool[state->y + 3] >> state->x) << 0xC) & 0xF000);**

**// 若当前方块与目标区域存在重叠（碰撞），则位与的结果中会有1**

**return ((dest & tetris) != 0);**

**}**

**// 插入方块**

**void InsertTetris(TetrisState \*state)**

**{**

**// 取出当前方块**

**unsigned short tetris = TetrisTable[state->type[0]][state->orientation[0]];**

**// 当前方块每4位（一行）取出，位或给游戏池相应位置**

**state->pool[state->y + 0] |= (((tetris >> 0x0) & 0x000F) << state->x);**

**state->pool[state->y + 1] |= (((tetris >> 0x4) & 0x000F) << state->x);**

**state->pool[state->y + 2] |= (((tetris >> 0x8) & 0x000F) << state->x);**

**state->pool[state->y + 3] |= (((tetris >> 0xC) & 0x000F) << state->x);**

**return;**

**}**

**// 移除方块**

**void RemoveTetris(TetrisState \*state)**

**{**

**// 当前方块**

**unsigned short tetris = TetrisTable[state->type[0]][state->orientation[0]];**

**// 当前方块每4位取出，按位取反后位与到游戏池相应位置，即完成移除方块**

**state->pool[state->y + 0] &= ~(((tetris >> 0x0) & 0x000F) << state->x);**

**state->pool[state->y + 1] &= ~(((tetris >> 0x4) & 0x000F) << state->x);**

**state->pool[state->y + 2] &= ~(((tetris >> 0x8) & 0x000F) << state->x);**

**state->pool[state->y + 3] &= ~(((tetris >> 0xC) & 0x000F) << state->x);**

**return;**

**}**

**// 设置颜色**

**void SetColor(const TetrisState \*state, TetrisOperation \*operation)**

**{**

**// 先判断池子的对应位置有没有方块，如果没有自然不用涂颜色 跳过**

**// 当移动方块位置或生成一个新方块时需要涂色**

**char i, x, y;**

**// 取出当前方块存入tetris**

**unsigned short tetris = TetrisTable[state->type[0]][state->orientation[0]];**

**// 设置4\*4的区域内每一格的颜色**

**// 用i循环枚举每一格**

**// i除以4的商为列 余数为行**

**// 正好可以遍历每一格**

**for (i = 0; i < 16; ++i)**

**{ // 取出当前目标列**

**y = (i >> 2) + state->y; // i%4**

**if (y > Row\_End) // 池底以下自然不用涂色**

**{**

**break;**

**}**

**x = (i & 3) + state->x; // 当前目标行**

**if ((tetris >> i) & 1) // 检测的到小方格属于当前方块区域**

**{**

**operation->color[y][x] = (state->type[0] | 8); // 设置颜色**

**}**

**}**

**return;**

**}**

**// 旋转方块**

**void RotateTetris(TetrisState \*state, TetrisOperation \*operation)**

**{**

**char ori = state->orientation[0]; // 记录方块原本姿态**

**RemoveTetris(state); // 删除当前方块**

**// 顺/逆时针旋转**

**state->orientation[0] = (operation->clockwise) ? ((ori + 1) & 3) : ((ori + 3) & 3);**

**if (CheckCollision(state)) // 检测到碰撞 旋转不合法**

**{**

**state->orientation[0] = ori; // 恢复为原姿态**

**InsertTetris(state); // 插入当前方块**

**}**

**else**

**{**

**InsertTetris(state); // 插入当前方块**

**SetColor(state, operation); // 刷颜色**

**PrintCurrentTetris(state, operation); // 显示当前方块**

**}**

**return;**

**}**

**// 水平移动方块**

**void FlatMoveTetris(TetrisState \*state, TetrisOperation \*operation)**

**{**

**int x = state->x; // 记录原列位置**

**// 考虑到大多数的时候，玩家都在上方进行移动定位，贴壁移动较少**

**// 因此直接假设移动是允许的，移动后判断是否发生碰撞**

**// 如果发生了，那么再把本次移动取消，恢复原有状态（这是较少数的情况）**

**RemoveTetris(state); // 删除当前方块**

**operation->direction == 0 ? (--state->x) : (++state->x); // 判断方向**

**if (CheckCollision(state)) // 检测到碰撞 则移动不合法**

**{**

**state->x = x; // 恢复为原列位置**

**InsertTetris(state); // 插入当前方块。由于移动取消，位置没有改变，不需要重新刷颜色**

**}**

**else**

**{**

**InsertTetris(state); // 插入当前方块**

**SetColor(state, operation); // 刷颜色**

**PrintCurrentTetris(state, operation); // 显示当前方块**

**}**

**}**

**// 向下移动方块**

**void DownMoveTetris(TetrisState \*state, TetrisOperation \*operation)**

**{**

**char y = state->y; // 记录原行位置**

**// 同左右移动，先假设移动是合法的**

**// 注意向下移动的处理和左右的不同之处：**

**// 1.不能向上 因此不用判断方向**

**// 2.向下可能导致消行 需要判断**

**RemoveTetris(state); // 删除当前方块**

**++state->y; // 向下移动**

**if (CheckCollision(state)) // 检测到碰撞 移动不合法**

**{**

**state->y = y; // 恢复为原行位置**

**InsertTetris(state); // 插入当前方块。由于移动不合法，位置没改变，不需要刷颜色**

**if (CheckErasing(state, operation)) // 检测到消行**

**{**

**PrintPool(state, operation); // 消行后重新显示游戏池**

**}**

**}**

**else // 没有发生消行，那么直接插入方块即可**

**{**

**InsertTetris(state); // 插入当前方块**

**SetColor(state, operation); // 刷颜色**

**PrintCurrentTetris(state, operation); // 显示当前方块**

**}**

**}**

**// 硬降的处理**

**void DropDown(TetrisState \*state, TetrisOperation \*operation)**

**{**

**RemoveTetris(state); // 删除当前方块**

**for (; state->y < Row\_End; ++state->y) // 从方块当前所在行开始往下扫描到池底**

**{**

**if (CheckCollision(state)) // 检测到碰撞 那么就此打住**

**{**

**break;**

**}**

**}**

**--state->y; // 注意刚才break的位置是发生了碰撞的，因此硬降后到达的位置还要往上一行**

**InsertTetris(state); // 插入当前方块**

**SetColor(state, operation); // 刷颜色**

**CheckErasing(state, operation); // 检测消行**

**PrintPool(state, operation); // 重新打印游戏池**

**}**

**// 消行检测**

**bool CheckErasing(TetrisState \*state, TetrisOperation \*operation)**

**{**

**static const unsigned scores[5] = { 0, 10, 30, 90, 150 }; // 单次消行同时消的行数越多，得分越高**

**char count = 0;**

**char k = 0, y = state->y + 3; // 消行时因为要下移，因此从4\*4区域的最下面一行开始往上扫描**

**// k是计数器**

**do**

**{**

**if (y < Row\_End && state->pool[y] == 0xFFFFU) // 排除池底情况和一行没有填满的情况**

**{**

**++count;**

**// 消除一行方块**

**memmove(state->pool + 1, state->pool, sizeof(unsigned short) \* y);**

**// 第y行被消除 因此y行以上的池子状态都往下平移一行**

**memmove(operation->color[1], operation->color[0], sizeof(char[16]) \* y);**

**// 颜色数组当然也要往下挪**

**}**

**else**

**{**

**--y;**

**++k;**

**}**

**} while (y >= state->y && k < 4); // k控制判断只做4次（因为区域只有4行）**

**state->erasedTotal += count; // 更新消行总数**

**state->score += scores[count]; // 更新得分**

**if (count > 0)**

**{**

**++state->erasedCount[count - 1]; // 把 x连消 的次数 分开记录**

**}**

**FormTetris(state); // 生成下一个方块**

**SetColor(state, operation); // 新方块上颜色**

**return (count > 0);**

**}**

**// 键盘按键处理**

**void PressProcess(TetrisState \*state, TetrisOperation \*operation, int key)**

**{**

**if (key == 13) // 回车键代表暂停**

**{**

**operation->pause = !operation->pause;**

**}**

**if (operation->pause) // 如果处于暂停状态，不作响应**

**{**

**return;**

**}**

**switch (key)**

**{**

**case 'w': case 'W': case '8': case 72: // 上 72是方向键的“上”的代码**

**operation->clockwise = true; // 顺时针旋转**

**RotateTetris(state, operation); // 旋转方块**

**break;**

**case 'a': case 'A': case '4': case 75: // 左**

**operation->direction = 0; // 向左移动**

**FlatMoveTetris(state, operation); // 水平移动方块**

**break;**

**case 'd': case 'D': case '6': case 77: // 右**

**operation->direction = 1; // 向右移动**

**FlatMoveTetris(state, operation); // 水平移动方块**

**break;**

**case 's': case 'S': case '2': case 80: // 下**

**DownMoveTetris(state, operation); // 向下移动方块**

**break;**

**case ' ': // 空格键 硬降**

**DropDown(state, operation);**

**break;**

**case 'z': // 反向旋转**

**operation->clockwise = false; // 逆时针旋转**

**RotateTetris(state, operation); // 旋转方块**

**break;**

**default:**

**break;**

**}**

**}**

**void GotoXY(short x, short y)**

**{**

**static COORD cd;**

**cd.X = (short)(x << 1);**

**cd.Y = y;**

**SetConsoleCursorPosition(Console\_output, cd);**

**return;**

**}**

**// 打印游戏池边界**

**void PrintBorder()**

**{**

**char y; // 本质是当成一个8位的整数来用**

**SetConsoleTextAttribute(Console\_output, 0xF0);**

**for (y = Row\_Begin; y < Row\_End; ++y) // 不显示顶部缓冲区和池底**

**{**

**GotoXY(10, y - 3);**

**printf("%2s", " ");**

**GotoXY(23, y - 3);**

**printf("%2s", "");**

**}**

**GotoXY(10, y - 3); // 池底**

**printf("%28s", "");**

**return;**

**}**

**// 定位到池子中的x，y处**

**// 池子中的xy和窗口的xy是有区别的 要去掉边界部分**

**#define XYinPool(x, y) GotoXY(x + 9, y - 3)**

**// 打印游戏池**

**void PrintPool(const TetrisState \*state, const TetrisOperation \*operation)**

**{**

**char x, y; // 8位整数**

**for (y = Row\_Begin; y < Row\_End; ++y) // 不显示顶部缓冲区和池底**

**{**

**XYinPool(2, y); // 定位到游戏池中的坐标**

**for (x = Column\_Begin; x < Column\_End; ++x) // 不显示左右边界**

**{**

**if ((state->pool[y] >> x) & 1) // 游戏池该方格有方块**

**{**

**// 用对应的颜色，显示一个实心方块**

**SetConsoleTextAttribute(Console\_output, operation->color[y][x]);**

**printf("■");**

**}**

**else // 没有方块那就显示空白**

**{**

**SetConsoleTextAttribute(Console\_output, 0);**

**printf("%2s", "");**

**}**

**}**

**}**

**}**

**// 打印当前方块所在的4\*4区域**

**void PrintCurrentTetris(const TetrisState \*state, const TetrisOperation \*operation)**

**{**

**char x, y; // 8位整数**

**// 显示当前方块是在移动后调用的，移动前的方块被删除了，区域也发生了变化，因此更新区域需要扩展**

**// 由于不可能向上移动，因此不需要向下扩展**

**y = (state->y > Row\_Begin) ? (state->y - 1) : Row\_Begin; // 向上扩展一格**

**for (; y < Row\_End && y < state->y + 4; ++y)**

**{**

**x = (state->x > Column\_Begin) ? (state->x - 1) : Column\_Begin; // 向左扩展一格**

**for (; x < Column\_End && x < state->x + 5; ++x) // 向右扩展一格**

**{**

**XYinPool(x, y); // 定点到游戏池中的方格**

**if ((state->pool[y] >> x) & 1) // 游戏池该方格有方块**

**{**

**// 用相应颜色，显示一个实心方块**

**SetConsoleTextAttribute(Console\_output, operation->color[y][x]);**

**printf("■");**

**}**

**else // 没有方块，显示空白**

**{**

**SetConsoleTextAttribute(Console\_output, 0);**

**printf("%2s", "");**

**}**

**}**

**}**

**return;**

**}**

**// 打印下一个和下下一个方块**

**void DisplayNextTetris(const TetrisState \*state)**

**{**

**char i;**

**unsigned short tetris;**

**// 打印提示框**

**SetConsoleTextAttribute(Console\_output, 0xF);**

**GotoXY(26, 1);**

**printf("┏━━━━┳━━━━┓");**

**GotoXY(26, 2);**

**printf("┃%8s┃%8s┃", "", "");**

**GotoXY(26, 3);**

**printf("┃%8s┃%8s┃", "", "");**

**GotoXY(26, 4);**

**printf("┃%8s┃%8s┃", "", "");**

**GotoXY(26, 5);**

**printf("┃%8s┃%8s┃", "", "");**

**GotoXY(26, 6);**

**printf("┗━━━━┻━━━━┛");**

**// 打印下一个方块，显示颜色**

**tetris = TetrisTable[state->type[1]][state->orientation[1]];**

**SetConsoleTextAttribute(Console\_output, state->type[1] | 8);**

**for (i = 0; i < 16; ++i)**

**{**

**GotoXY((i & 3) + 27, (i >> 2) + 2);**

**((tetris >> i) & 1) ? printf("■") : printf("%2s", "");**

**}**

**// 打印下下一个方块，不显示彩色**

**tetris = TetrisTable[state->type[2]][state->orientation[2]];**

**SetConsoleTextAttribute(Console\_output, 8);**

**for (i = 0; i < 16; ++i)**

**{**

**GotoXY((i & 3) + 32, (i >> 2) + 2);**

**((tetris >> i) & 1) ? printf("■") : printf("%2s", "");**

**}**

**return;**

**}**

**// 显示得分信息**

**void PrintScore(const TetrisState \*state)**

**{**

**char i;**

**SetConsoleTextAttribute(Console\_output, 0xE);**

**GotoXY(2, 2);**

**printf("得分：%u", state->score);**

**GotoXY(1, 6);**

**printf("消行总数：%u", state->erasedTotal);**

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

**{**

**GotoXY(2, 8 + i);**

**printf("%d连消：%u", i + 1, state->erasedCount[i]);**

**}**

**GotoXY(1, 15);**

**printf("方块总数：%u", state->tetrisTotal);**

**return;**

**}**

**// 显示提示信息**

**void PrintInstruction()**

**{**

**SetConsoleTextAttribute(Console\_output, 0xB);**

**GotoXY(26, 10);**

**printf(" 游戏说明：");**

**GotoXY(27, 12);**

**printf("向左移动：← A 4");**

**GotoXY(27, 13);**

**printf("向右移动：→ D 6");**

**GotoXY(27, 14);**

**printf("向下移动：↓ S 2");**

**GotoXY(27, 15);**

**printf("顺时针转：↑ W 8");**

**GotoXY(27, 16);**

**printf("逆时针转：z");**

**GotoXY(27, 17);**

**printf("直接落地：空格");**

**GotoXY(27, 18);**

**printf("暂停游戏：回车");**

**GotoXY(25, 23);**

**printf(" By: IGNB");**

**return;**

**}**

**// 维护游戏运行**

**void RunGame(TetrisState \*state, TetrisOperation \*operation)**

**{**

**clock\_t clockLast, clockNow;**

**clockLast = clock(); // 计时**

**PrintPool(state, operation); // 运行时需要持续显示游戏池**

**while (!state->dead) // 没有死亡**

**{**

**while (\_kbhit()) // 一旦有按键盘的操作**

**{**

**PressProcess(state, operation, \_getch()); // 处理按键**

**}**

**if (!operation->pause) // 暂停状态不需要计时下落**

**{**

**clockNow = clock(); // 计时第二次**

**// 50分和100分的时候提升难度加快下落速度**

**float delay;**

**if (state->score >= 50 && state->score < 100) delay = 0.2;**

**else if (state->score < 50) delay = 0.4;**

**else delay = 0.1;**

**if (clockNow - clockLast > delay \* CLOCKS\_PER\_SEC) //把clock()的单位转换成秒**

**{**

**clockLast = clockNow;**

**PressProcess(state, operation, 80); // 把自由下落等价于按了一次下键**

**}**

**}**

**}**

**return;**

**}**

**// 再玩一次**

**bool PlayAgain()**

**{**

**int ch;**

**SetConsoleTextAttribute(Console\_output, 0xF0);**

**// 打印信息**

**GotoXY(15, 10);**

**printf("游戏结束");**

**GotoXY(13, 11);**

**printf("按Y重玩，按N退出");**

**// 等待玩家选择是否继续玩**

**do**

**{**

**ch = \_getch();**

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

**{**

**return true;**

**}**

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

**{**

**return false;**

**}**

**} while (1);**

**}**