第1章 俄罗斯方块游戏

俄罗斯方块游戏是一款风靡全球的游戏产品，它曾经造就了一个无与伦比的商业价值，影响了一代游戏产业链。本章使用C语言开发一个俄罗斯方块游戏，并详细介绍其具体的实现流程。

1.1 背景介绍

俄罗斯方块游戏最初是由苏联的游戏制作人阿列克谢·帕基特诺夫(Alexey Pajitnov)推出的，它表面看似简单却变化无穷，令人上瘾，并且引发出人的无限遐想。

扫码看视频

1.1.1 游戏行业发展现状

近年来，随着互联网、移动互联网技术的快速发展，互联网基础设施越来越完善，互联网用户规模迅速增长。得益于整个互联网产业的爆炸式增长，我国网络游戏产业呈现出飞速发展的态势，网络游戏整体用户规模持续扩大。随着我国游戏用户规模的不断扩大，我国游戏行业发展迅速，市场规模快速增加。在 2022 年度中国游戏产业年会上，《2022 年中国游戏产业报告》显示，2022 年中国游戏市场实际销售收入 2658.84 亿元，同比下降 10.33%。游戏用户规模为 6.64 亿，同比下降了 0.33%。2022 年自主研发游戏在国内市场的实际销售收入为 2223.77 亿元，同比下降了 13.07%。

中国客户端游戏市场已步入成熟期，进入存量竞争阶段，面对来自移动游戏的竞争压力，行业内部竞争激烈，发展速度逐渐放缓。与客户端游戏一样，网页游戏也面临着移动游戏的竞争，并且日渐没落。自 2015 年起，中国网页游戏市场实际收入持续下降，2019 年收入仅为 98.7 亿元，同比 2018 年减少 27.8 亿元，下降 22.0%，呈逐年下降之势。中国网页游戏市场近几年受到移动端市场的冲击，从事网页游戏的企业越来越少，用户逐步向移动游戏转移，用户人数由 2015 年的 3 亿人下降至 2022 年的 1.7 亿人。

1.1.2 虚拟现实快速发展

根据 Newzoo(权威数据机构)的 2022 年 VR(Virtual Reality 的缩写，表示虚拟现实)游戏市场报告，到 2022 年底，VR 游戏市场的当年收入达到 18 亿美元。未来几年，这一市场的复合增长率将达到 45%。Newzoo 还表示，随着游戏设备性能的提升，VR 用户数量正在以前所未有的速度增长。其中，独立 VR 耳机的增加是用户增长的主要原因之一，例如无须额外硬件即可设置和使用的 Meta Quest 等设备。

电子游戏行业已经对 VR 技术投入了大量时间和资源，创造能够投入市场的产品的尝试可以追溯到 20 世纪 80 年代，但在接口设备方面的进度一直停滞不前。2013 年，Oculus Rift 头戴显示器的推出带来了接口方面的突破性进展，重新激发了游戏开发商和技术提供商对 VR 技术的兴趣。

2020 年《半衰期：爱莉克斯》的到来，全面解决了如玩家的动作、数字环境中的物理存在感、与游戏内物体和环境的交互、可玩性以及通过 VR 媒体对故事的叙述等更多 VR 游戏的问题，提升了商业产品的可行性，让越来越多的 PC 游戏开发商拥抱 VR 技术。

如今，VR 技术仍在不断发展，并吸引了大厂的大力投资，其中最为瞩目的，当属 2022 Meta Connect 大会上，Meta 宣布收购了三家拥有深厚经验和成熟技术的 VR 工作室。在陆续收购多家游戏工作室后，Meta 为其元宇宙事业进一步招兵买马。

1.1.3 云游戏持续增长

2022 年，云游戏技术和服务取得了重大进展，包括罗技(Logitech)和腾讯合作开发的手持游戏机罗技 G CLOUD、雷蛇旗下的支持 5G 连接的全新云驱动手持游戏设备 Razer Edge 5G 预告等。目前云游戏还处于发展阶段，好消息是，从市场规模来看，被视为“游戏的未来”的云游戏行业，正处于增长趋势中。

Newzoo 的 2022 年全球云游戏报告显示，在 2022 年，云游戏服务吸引了超过 3000 万名付费用户，这些用户的总消费额预计将达到 24 亿美元。到 2025 年，全球云游戏市场的年收入有望增长至 82 亿美元。Newzoo 表示，随着新的云游戏服务不断推出，云游戏市场正变得越来越成熟。在付费设计和使用场景等方面的创新，使得云游戏吸引了更多的用户并且更好地满足其需求。与此同时，微软 Xbox 和索尼 PlayStation 等传统游戏巨头也在积极拓展云游戏业务。

美国软件公司 Perforce 在采访了 300 多名行业专业人士后，得出结论：流媒体和云游戏将在不久的将来成为玩电子游戏的主要方式。随着 5G 覆盖范围的增加，甚至 6G 技术的发展，云游戏将进一步巩固其在游戏领域的地位。

1.1.4 移动游戏重回增长轨道

Newzoo 表示，随着生活回归正常，移动游戏尤其是超休闲类型将迎来下滑，2022 移动业务收入下降 6.4%至 922 亿美元，此前 data.ai 发布的移动应用预测报告中，2023 年全球收入也将再下降 3% 至 1070 亿美元。长期来看，全球游戏市场将恢复增长轨迹，Newzoo 预计到 2025 年，游戏市场将创造 2112 亿美元的收益，年复合增长率为 3.4%，游戏市场未来几年很有前景，尤其是主机游戏。Sensor Tower 预计，全球手游市场营收也将在 2023 年重回增长轨迹，到 2026 年将上升至 1170 亿美元，未来几年内的年均复合增长率约为 5.6%。混合休闲等新趋势的兴起和中度手游的复苏，也将为手游市场带来新的活力。

此外，近年来大厂纷纷布局手游市场，今年备受关注的微软暴雪收购案中，Xbox 表示希望通过这一交易扩大其在移动游戏中的影响力，今年早些时候，微软还提到了打造 Xbox 手机游戏商店的计划。移动游戏已经越来越成为全球游戏市场的主导力量。

1.2 项目分析

对于软件开发来说，项目分析是第一步工作，本节将对俄罗斯方块游戏进行详细分析，为后期的开发工作做好充分的准备工作。

扫码看视频

1.2.1 项目分析介绍

很多开发者写程序，特别是一些初学者，总是看到功能后就立即投入到代码编写工作中，需要什么功能就编写函数去一一实现。但是在后期调试时，常常会出现这样或那样的错误，需要返回重新修改。幸运的是，初学者接触到的都是小项目，修改的工作量也不是很大，但如果在大型项目中，几千行代码的返回修改就会是一件很恐怖的事情。所以，在项目开发之前，就需要做好项目的前期规划。

一个软件项目的开发主要分为五个阶段，即需求分析阶段、设计阶段、编码阶段、测试阶段和维护阶段。需求分析阶段得到的结果，是软件项目开发中其他四个阶段的必备条件。从以往的经验来看，需求分析中的一个小的偏差，就可能会导致整个项目无法达到预期的效果，或者说最终开发出的产品不是用户所需要的。

1.2.2 规划开发流程

要想更好地完成俄罗斯方块游戏项目的功能分析工作，需要将这款游戏从头到尾地试玩几次，彻底了解俄罗斯方块游戏的具体玩法和过程。为此笔者特意从网络中下载了一款俄罗斯方块游戏，并全面地进行了试玩。俄罗斯方块游戏的典型界面如图 1-1 所示。

图 1-1　俄罗斯方块游戏的界面

根据俄罗斯方块的游戏规则和要求，可以总结出俄罗斯方块游戏的基本功能模块。当然，因为俄罗斯方块游戏是一款在市面中流行多年的游戏，所有游戏的基本玩法和功能大家都很熟悉。如果要开发一款全新的游戏，这就需要在项目规划开始进行玩法规划设计，相关内容将在本书后面的章节中进行讲解。

根据软件项目的开发流程，可以做出一个简单的项目规划书，整个规划书分为如下两个部分：

- 系统需求分析；
- 结构规划。

俄罗斯方块游戏项目的开发流程如图 1-2 所示。

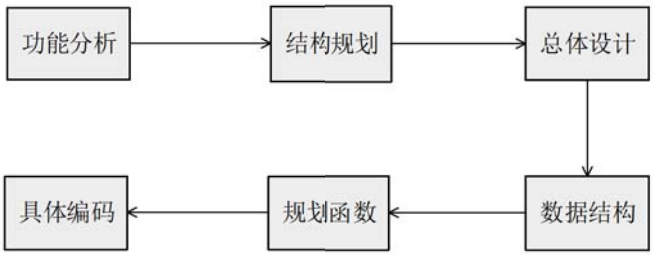

图 1-2 开发流程图

- 功能分析：分析整个系统所需要的功能；
- 结构规划：规划系统中所需要的功能模块；
- 总体设计：分析系统处理流程，探索系统核心模块的运作；
- 数据结构：设计系统中需要的数据结构；
- 规划函数：预先规划系统中需要的功能函数；
- 具体编码：编写系统的具体实现代码。

1.2.3 系统需求分析

系统需求分析中的关键数据是从试玩游戏过程中得到的，一款俄罗斯方块游戏必须具备如下所述的基本功能。

(1) 游戏方块预览功能

在此游戏中共有 19 种方块，当游戏运行后并在下落区域出现一个游戏方块时，为了便于玩家做好提前判断工作，在方块预览区内要显示随机生成的游戏方块。

(2) 游戏方块控制功能

游戏玩家可以对出现的方块进行处理，分别实现左移、右移、快速下移、自由下落、

旋转和行满自动消除功能效果。

(3) 更新游戏显示

当游戏在进行方块移动处理时，要清除先前的游戏方块，用新坐标重绘游戏方块。

(4) 游戏速度设置和分数更新

通过游戏分数能够对行数进行划分，例如可以设置完成完整的一行为 10 分。当达到一定数量后，需要给游戏者进行等级上的升级。当玩家级别升高后，方块的下落速度将加快，从而对应地提高游戏的难度。

(5) 游戏帮助功能

游戏玩家进入游戏系统后，通过帮助了解游戏的操作方法，介绍游戏的玩法规则。

1.2.4 结构规划

现在开始步入结构规划阶段。为了加深印象，可以绘制一个模块结构图，如图 1-3 所示。

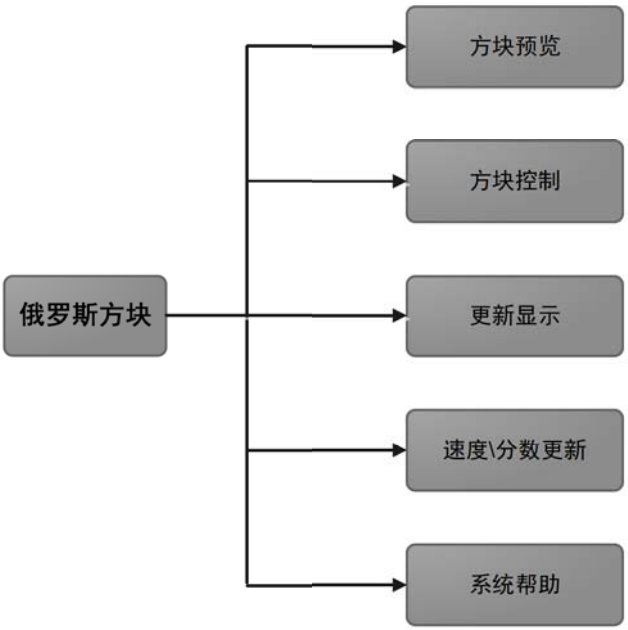

图 1-3 游戏模块结构

1.2.5 选择开发工具

在当今高校进行 C 语言教学时，大多使用 Turbo C 作为开发工具，并且在计算机二级

考试中，也大多使用 Turbo C 工具。除了 Turbo C 语言，还有很多工具可以实现 C 语言程序开发，例如 DEV-C++、Visual C++6.0 和 Visual Studio.NET。其中 DEV-C++是一个轻量级的开发工具，适合初学者使用，Visual C++6.0 已经被 Visual Studio.NET 完全替代。另外，因为 C 语言和 C++语言的相似性，所以大多数开发工具既能开发 C++程序，也能开发 C 程序。

本章俄罗斯方块游戏项目只是一个简单的程序，可以首选 Turbo C 和 DEV-C++开发工具。如果客户要求使用 graphics.h 实现，就选择 Turbo C 作为开发工具，因为 Turbo C 使用 graphics.h 的方法更加简单。

1.3 总体设计

经过总体结构的规划分析，接下来就可以根据各结构功能模块进行对应的总体设计处理。总体设计阶段主要包括如下两个方面的工作：

扫码看视频

- 运行流程分析；
- 核心处理模块分析。

1.3.1 运行流程分析

根据总体的模块功能和规划的结构图可以绘制出整个游戏的运行流程，具体如图 1-4 所示。

为了解决问题，在图 1-4 所示的运行流程中，判断键值时可使用 VK_LEFT(左移)、VK_RIGHT(右移)、VK_DOWN(下移)、VK_UP(旋转)和 VK_ESC(退出)键进行判断。上述几个按键移动处理的具体说明如下：

- VK_LEFT：调用函数 MoveAble()，判断是否能左移，如果可以左移则调用 EraseBox 函数，清除当前的游戏方块。并在下一步调用函数 show_box()，在左移位置显示当前游戏的方块。
- VK_RIGHT：右移处理，和上面的 VK_LEFT 处理类似。
- VK_DOWN：下移处理，如果不能再下移，必须将 flag_newbox 标志设置为 1。
- VK_UP：旋转处理，首先判断旋转动作是否执行，如果不符合条件，则不予执行。
- VK_ESC：按下 Esc 键后将退出游戏。

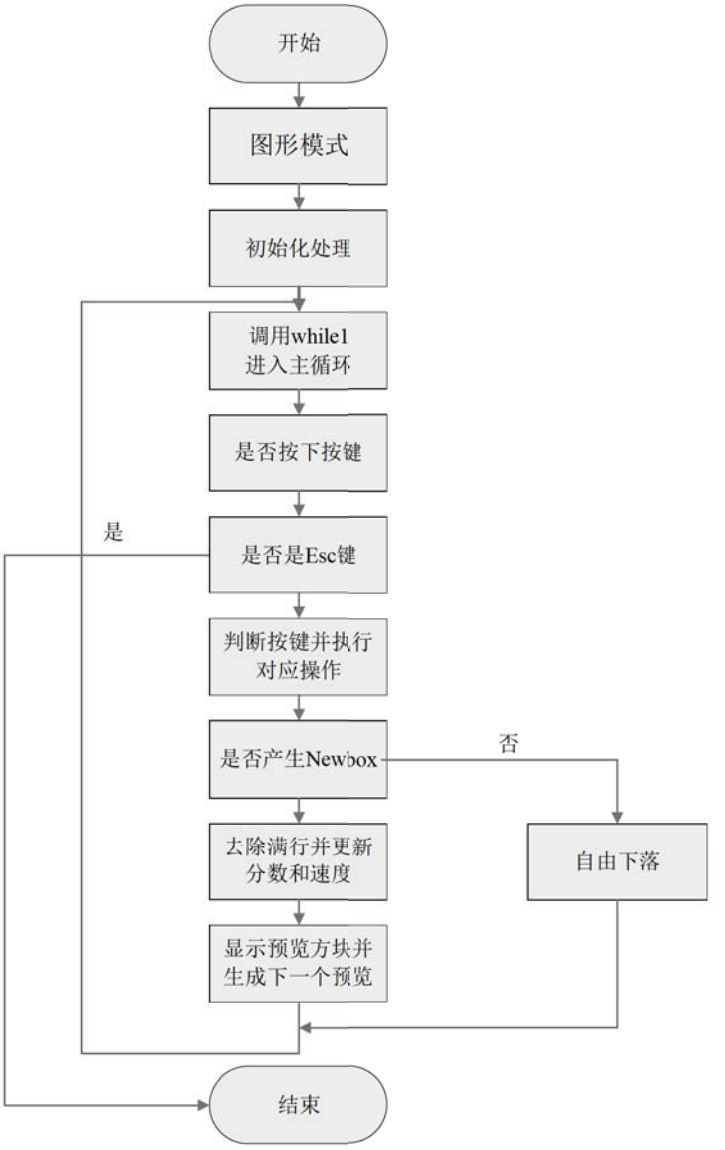

图 1-4　游戏运行流程

1.3.2　核心处理模块分析

对于本俄罗斯方块游戏来说，整个项目的核心内容：方块预览、方块控制、更新显示，速度和分数更新。整个项目就是通过编码来实现上述功能。

1. 方块预览

新游戏的方块将在 4×4 的正方形小方块中预览，使用随机函数 rand()可以产生 1～19 间的游戏方块编号，并作为预览的方块编号，其中品尼高正方形小方块的大小由 BSIZE×BSIZE 来计算。

2. 方块控制处理

方块的移动控制是整个游戏的重点和难点，具体说明如下。

(1) 左移处理，处理过程如下

- 判断是否能够左移，判断条件有两个：左移一位后方块不能超游戏底板的左边线，否则将越界；游戏方块有值的位置，游戏底板不能被占用。
- 清除左移前的游戏方块。
- 在左移一位的位置处，重新显示此游戏的方块。

(2) 右移处理，处理过程如下

- 判断是否能够右移，判断条件有两个：右移一位后方块不能超越游戏底板的右边线，否则将越界；游戏方块有值的位置，游戏底板不能被占用。
- 清除右移前的游戏方块。
- 在右移一位的位置处，重新显示此游戏的方块。

(3) 下移处理，处理过程如下

- 判断是否能够下移，判断条件有两个：下移一位后方块不能超越游戏底板的底边线，否则将越界；游戏方块有值的位置，游戏底板不能被占用。满足上述两个条件后，可以被下移处理。否则将 flag_newbox 设置为 1，在主循环中会判断此标志。
- 清除下移前的游戏方块。
- 在下移一位的位置处，重新显示此游戏的方块。

(4) 旋转处理，处理过程如下

- 判断是否能够旋转，判断条件有两个：旋转后方块不能超越游戏底板的底边线、左边线和右边线，否则将越界；游戏方块有值的位置，游戏底板不能被占用。
- 清除旋转前的游戏方块。
- 在游戏方块显示区域(4×4)的位置，使用当前游戏方块的数据结构中的 next 值作为旋转后形成的新游戏方块的编号，并重新显示这个编号的游戏方块。

3. 更新显示

当游戏中的方块进行移动处理时，需要清除先前的游戏方块，并用新坐标重新绘制游戏方块。当消除满行后，要重绘游戏底板的当前状态。清除游戏方块的具体过程为：首先绘制一个轮廓，然后使用背景色填充小方块，最后使用前景色画一个游戏底板中的小方块。

循环处理这个过程，变化当前的坐标来绘制并填充 19 个这样的方块。从而在游戏中清除了此方块。

4. 速度和分数更新处理

当行满消除后，积分变量 score 会增加一个固定的值，然后将等级变量 level 和速度变量 speed 相关联，以实现等级越高速度越快的效果。

1.4 设计数据结构

结构体是 C 语言中另一种常用的构造数据类型，它相当于其他高级语言中的记录。“结构”是一种构造类型，它是由若干“成员”组成的，每一个成员可以是基本数据类型或者构造类型。在本项目中，使用数据结构保存项目中用到的数据，包括方块、方块形状、底板等。

扫码看视频

1.4.1 使用数据结构可以提高运行和存储效率

在 C 语言应用中，数据结构可以作为计算机存储、组织数据的方式。数据结构是相互之间存在一种或多种特定关系的数据元素的集合。在通常情况下，合理的数据结构可以带来更高的运行或者存储效率。例如下面就定义了一个结构 stu，里面包含了 4 个成员。

```
struct stu{
       int num;
       char name[20];
       char sex;
       float score;
};
```

在上述结构定义中，结构名为 stu，该结构由如下 4 个成员组成。

- 第一个成员为 num，整型变量；
- 第二个成员为 name，字符数组；
- 第三个成员为 sex，字符变量；
- 第四个成员为 score，实型变量。

在此应该注意，程序结尾处括号后的分号是不可少的。定义结构之后，即可进行变量说明。在上述代码中，结构 stu 的变量都由上述 4 个成员组成。由此可见，结构是一种复杂的数据类型，是数目固定、类型不同的若干有序变量的集合。

在定义一个结构体时，应该注意如下 3 点。

- 不要忽略最后的分号，例如在上面代码中指定了一个新的结构体类型 struct stu(struct 是声明结构体类型时所必须使用的关键字，不能省略)，它向编译系统声明，这是一个“结构体类型”，包括 num、name、sex、score 等不同类型的数据项。
- struct xxx 是一个类型名，它和系统提供的标准类型(如 int、char、float、double 等)具有相同的作用，都可以用来定义变量的类型，只不过结构体类型需要由用户自己指定。
- 可以把“成员表列”(member list)称为“域表”(field list)。每一个成员也称结构体中的一个域，成员名命名规则与变量名相同。

本俄罗斯方块游戏项目很简单，涉及的结构也不是不多。纵观整个项目，涉及数据类型的游戏对象有三个，具体说明如下。

- 游戏底板：需要判断下落的方块是否碰撞到底板，另外，为了更加美观，需要用不同的颜色表现底板。
- 游戏中的俄罗斯方块：为每个方块分别设置编号和颜色。
- 不同形状的方块：用多个正方形方块组成不同形状的模型。

本项目将详细讲解实现上述三个数据结构的具体过程。

1.4.2 设计游戏底板结构体

本项目游戏的底板结构体是 BOARD，具体代码如下：

```
struct BOARD                    /*游戏底板结构,表示每个小方块所具有的属性*/
{
   int var;                     /*当前状态 只有 0 和 1,1 表示此小方块已被占用*/
   int color;                   /*颜色,游戏底板的每个小方块可以设置不同的颜色使画面美观*/
}Table_board[Vertical_boxs][Horizontal_boxs];
```

其中，BOARD 结构体表示游戏底板中每个小方块的属性，var 表示当前的状态，为 0 时表示未被占用，为 1 时表示已经被占用。

1.4.3 游戏方块结构体

本项目游戏的方块结构体是 SHAPE，具体代码如下：

```
struct SHAPE{
/*一个字节等于 8 位,每 4 位来表示一个方块的一行，例如:box[0]="0x88",box[1]="0xc0"表示的是:
1000
1000
1100
0000*/
   char box[2];
```

```
    int color;              /*每个方块的颜色*/
    int next;               /*下个方块*/
};
```

SHAPE 结构体表示某个小方块的属性，char box[2]表示用 2 个字节来表示块的形状，每 4 位来表示一个方块的一行。color 表示每个方块的颜色，颜色值可以根据需要而设置。

1.4.4 SHAPE 结构数组

本项目游戏的 SHAPE 结构数组的具体代码如下：

```
struct SHAPE shapes[MAX_BOX]=
{
/*
 *   □     □□□   □□      □
 *   □     □       □  □□□
 *   □□            □
 */
    {0x88,  0xc0,   CYAN,   1},
    {0xe8,  0x0,    CYAN,   2},
    {0xc4,  0x40,   CYAN,   3},
    {0x2e,  0x0,    CYAN,   0},
/*
 *   □       □□ □□□
 *   □ □     □       □
 * □□ □□□ □
 */
    {0x44,  0xc0,   MAGENTA,  5},
    {0x8e,  0x0,    MAGENTA,  6},
    {0xc8,  0x80,   MAGENTA,  7},
    {0xe2,  0x0,    MAGENTA,  4},
/*
 *   □
 *   □□          □□
 *     □       □□
 */
    {0x8c,  0x40,   YELLOW, 9},
    {0x6c,  0x0,    YELLOW, 8},
/*
 *   □       □□
 * □□          □□
 * □
 */
    {0x4c,  0x80,   BROWN,  11},
    {0xc6,  0x0,    BROWN,  10},
```

```
/*
 *   □       □              □
 * □□□     □□   □□□   □□
 *          □      □      □
 */
   {0x4e,  0x0,    WHITE,  13},
   {0x8c,  0x80,   WHITE,  14},
   {0xe4,  0x0,    WHITE,  15},
   {0x4c,  0x40,   WHITE,  12},
/* □
 * □
 * □       □□□□
 * □
 */
   {0x88,  0x88,   RED,    17},
   {0xf0,  0x0,    RED,    16},
/*
 * □□
 * □□
 */
   {0xcc,  0x0,    BLUE,   18}
};
```

在上述代码中，定义了 MAX_BOX 个 SHAPE 类型的结构数组，并进行了初始化处理。因为共有 19 种不同的方块类型，所以 MAX_BOX 的值为 19。

1.5 规划系统函数

在进行具体编码工作前，需要先规划好项目中需要的函数，并做好具体的定义。在本俄罗斯方块游戏项目中，需要用到如下所示的函数。

扫码看视频

1. 函数 NewTimer()

函数 NewTimer()用于实现新的时钟，具体代码如下：

```
void interrupt newtimer(void)
```

2. 函数 SetTimer()

函数 SetTimer()用于设置新时钟的处理过程，具体代码如下：

```
void SetTimer(void interrupt(*IntProc)(void))
```

3. 函数 KillTimer()

函数 KillTimer()用于恢复原有的时钟处理过程，具体代码如下：

```
void KillTimer()
```

4. 函数 initialize()

函数 initialize()用于初始化界面，具体代码如下：

```
void initialize(int x,int y,int m,int n)
```

5. 函数 DelFullRow()

函数 DelFullRow()用于删除满行，y 设置删除的行数，具体代码如下：

```
int DelFullRow(int y)
```

6. 函数 setFullRow()

函数 setFullRow()用于查询满行，并调用 DelFullRow 函数进行处理，具体代码如下：

```
void setFullRow(int t_boardy)
```

7. 函数 MkNextBox()

函数 MkNextBox()用于生成下一个游戏方块，并返回方块号，具体代码如下：

```
int MkNextBox(int box_numb)
```

8. 函数 EraseBox()

函数 EraseBox()用于清除以(x,y)位置开始的编号为 box_numb 的游戏方块，具体代码如下：

```
void EraseBox(int x,int y,int box_numb)
```

9. 函数 show_box()

函数 show_box()用于显示以(x,y)位置开始、编号为 box_numb、颜色值为 color 的游戏方块，具体代码如下：

```
void show_box(int x,int y,int box_numb,int color)
```

10. 函数 MoveAble()

函数 MoveAble()首先判断方块是否可以移动，其中(x,y)是当前的位置，box_numb 是方

块号，direction 是方向标志，具体代码如下：

```
int MoveAble(int x,int y,int box_numb,int direction)
```

到此为止，已经完成了本项目前期的所有工作。项目中的所有功能都是通过函数来实现的，函数构成了整个项目的主体。

1.6 具体实现

前面的工作都只能称之为前期准备工作，接下来将以前期分析和规划资料为基础编写各段代码，步入正式的编码阶段，实现具体过程。

扫码看视频

1.6.1 预处理

预处理是在程序源代码被编译之前，由预处理器(Preprocessor)对程序源代码进行的处理。这个过程并不对程序的源代码进行解析，但它把源代码分割或处理成为特定的符号用来支持宏调用。

预处理也是一个准备工作，在开始之前开发人员画了一个简单的实现流程图，如图 1-5 所示。

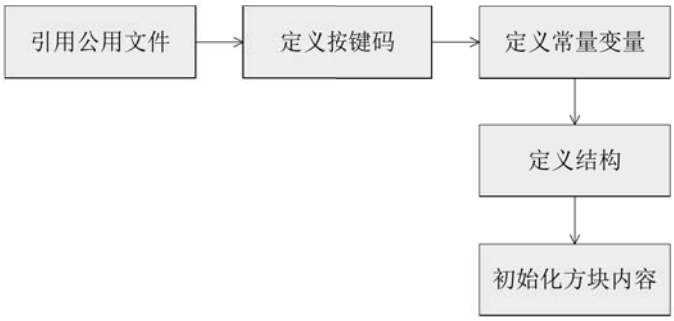

图 1-5 简单实现的流程图

(1) 先引用图形函数库等公用文件，具体代码如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include <graphics.h>                    /*图形函数库*/
```

(2) 定义按键码，即操控游戏的按键：左移、右移、下移、旋转等，具体代码如下所示：

```
/*定义按键码*/
#define VK_LEFT  0x4b00
#define VK_RIGHT 0x4d00
#define VK_DOWN  0x5000
#define VK_UP    0x4800
#define VK_ESC   0x011b
#define TIMER 0x1c                          /*设置中断号*/
```

(3) 定义系统中需要的常量，例如方块种类、方块大小、方块颜色等，具体实现代码如下所示：

```
/*定义常量*/
#define MAX_BOX 19                  /*总共有 19 种不同形态的方块*/
#define BSIZE 20                    /*方块的边长是 20 个像素*/
#define Sys_x 160                   /*显示方块界面的左上角 x 坐标*/
#define Sys_y 25                    /*显示方块界面的左上角 y 坐标*/
#define Horizontal_boxs 10          /*水平的方向以方块为单位的长度*/
#define Vertical_boxs 15            /*垂直的方向以方块为单位的长度*/
#define Begin_boxs_x Horizontal_boxs/2          /*产生第一个方块时出现的起始位置*/

#define FgColor 3                   /*前景颜色*/
#define BgColor 0                   /*背景颜色,0-blac*/

#define LeftWin_x Sys_x+Horizontal_boxs*BSIZE+46  /*右边状态栏的 x 坐标*/

#define false 0
#define true 1
/*移动的方向*/
#define MoveLeft 1
#define MoveRight 2
#define MoveDown 3
#define MoveRoll 4
/*以后坐标的每个方块可以看作是像素点是 BSIZE*BSIZE 的正方形*/
```

(4) 定义系统中需要的全局变量，例如方块的下落速度、玩家的分数、当前的方块编号等，具体实现代码如下所示：

```
/*定义全局变量*/
int current_box_numb;               /*保存当前方块编号*/
/*x,y 是保存方块当前坐标的*/
int Curbox_x=Sys_x+Begin_boxs_x*BSIZE,Curbox_y=Sys_y;
int flag_newbox=false;              /*是否要产生新方块的标记 0*/
int speed=0;                        /*下落速度*/
int score=0;                        /*总分*/
int speed_step=30;                  /*每等级所需要分数*/
/* 指向原来时钟中断处理过程入口的中断处理函数指针 */
void interrupt (*oldtimer)(void);
```

(5) 定义底板结构和方块结构。每一个新出现的方块结构是不同的，当方块下落到游戏底板后，结构也是不同的，所以必须编写 2 个结构来存储即时结构。具体实现代码如下所示：

```
struct BOARD                    /*游戏底板结构,表示每个点所具有的属性*/
{
   int var;                     /*当前状态只有 0 和 1,1 表示此点已被占用*/
   int color;                   /*颜色,游戏底板的每个点可以拥有不同的颜色,增强美观*/
}Table_board[Vertical_boxs][Horizontal_boxs];

/*方块结构*/
struct SHAPE{
   char box[2];                 /*一个字节等于 8 位,每 4 位来表示一个方块的一行
    如:box[0]="0x88",box[1]="0xc0"表示的是:
      1000
      000
      1100
      0000*/
   int color;                   /*每个方块的颜色*/
   int next;                    /*下个方块的编号*/
};
```

(6) 开始初始化方块内容，即定义允许最多箱子(MAX_BOX 个)预定义(SHAPE)类型的结构数组，并初始化。初始化就是把变量赋为默认值，把控件设为默认状态，把没准备的准备好。

1.6.2 主函数

在 C 语言中，任何程序执行都是从主函数 main()开始，到主函数的结束为止，退出程序。主函数可以调用其他函数，其他函数可以互相调用，但不能调用主函数。一般而言，编写一个能运行在操作系统上的程序，都需要一个主函数。

注意：主函数要尽量简洁，做到一目了然，因为主函数肩负着入口和出口的重任，所以尽量不要把太多的细节逻辑直接放入主函数，这不利于维护和扩展。

本项目主函数 main()的具体实现代码如下所示。

```
void main(){
   int GameOver=0;
   int key,nextbox;
   int Currentaction=0;/*标记当前动作状态*/
   int gd=VGA,gm=VGAHI,errorcode;
   initgraph(&gd,&gm,"");
   errorcode = graphresult();
   if (errorcode != grOk)
```

```
    {
        printf("\nNotice:Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to quit!");
        getch();
        exit(1);
    }
    setbkcolor(BgColor);
    setcolor(FgColor);
    randomize();
    SetTimer(newtimer);
    initialize(Sys_x,Sys_y,Horizontal_boxs,Vertical_boxs);    /*初始化*/
    nextbox=MkNextBox(-1);
    show_box(Curbox_x,Curbox_y,current_box_numb,shapes[current_box_numb].color);
    show_box(LeftWin_x,Curbox_y+200,nextbox,shapes[nextbox].color);
    show_intro(Sys_x,Curbox_y+320);
    getch();
    while(1)
    {
        /* Currentaction=0; flag_newbox=false; 检测是否有按键*/
        if (bioskey(1)){key=bioskey(0);     }
        else          {         key=0;     }
        switch(key)
        {
            case VK_LEFT:
                if(MoveAble(Curbox_x,Curbox_y,current_box_numb,MoveLeft))
                {EraseBox(Curbox_x,Curbox_y,current_box_numb);Curbox_x-=BSIZE;
                          Currentaction=MoveLeft;}
                break;
            case VK_RIGHT:
                if(MoveAble(Curbox_x,Curbox_y,current_box_numb,MoveRight))
                {EraseBox(Curbox_x,Curbox_y,current_box_numb);Curbox_x+=BSIZE;
                          Currentaction=MoveRight;}
                break;
            case VK_DOWN:
                if(MoveAble(Curbox_x,Curbox_y,current_box_numb,MoveDown))
                {EraseBox(Curbox_x,Curbox_y,current_box_numb);Curbox_y+=BSIZE;
                          Currentaction=MoveDown;}
                else flag_newbox=true;
                break;
            case VK_UP:/*旋转方块*/
                if(MoveAble(Curbox_x,Curbox_y,shapes[current_box_numb].next,MoveRoll))
                {EraseBox(Curbox_x,Curbox_y,current_box_numb);current_box_numb=
                          shapes[current_box_numb].next;
                   Currentaction=MoveRoll;
                }
                break;
            case VK_ESC:
```

```
            GameOver=1;
            break;
        default:
            break;
    }
    if(Currentaction)
    {   /*表示当前有动作,移动或转动*/
        show_box(Curbox_x,Curbox_y,current_box_numb,shapes[current_box_numb].color);
        Currentaction=0;
    }
        /*按了往下键，但不能下移,就产生新方块*/
    if(flag_newbox)
    {
        /*这时相当于方块到底部了,把其中出现点满一行的清除，置0*/
        ErasePreBox(LeftWin_x,Sys_y+200,nextbox);
        nextbox=MkNextBox(nextbox);
        show_box(LeftWin_x,Curbox_y+200,nextbox,shapes[nextbox].color);
        if(!MoveAble(Curbox_x,Curbox_y,current_box_numb,MoveDown))
        {
            show_box(Curbox_x,Curbox_y,current_box_numb,
            shapes[current_box_numb].color); GameOver=1;
        }
        else
        {
            flag_newbox=false;
        }
        Currentaction=0;
    }
    else    /*自由下落*/
    {
        if (Currentaction==MoveDown || TimerCounter> (20-speed*2))
        {
            if(MoveAble(Curbox_x,Curbox_y,current_box_numb,MoveDown))
            {
                EraseBox(Curbox_x,Curbox_y,current_box_numb);Curbox_y+=BSIZE;
                  show_box(Curbox_x,Curbox_y,current_box_numb,
                  shapes[current_box_numb].color);
            }
            TimerCounter=0;
        }
    }
    if(GameOver )/*|| flag_newbox==-1*/
    {
        printf("game over,thank you! your score is %d",score);
        getch();
        break;
    }
```

```
    }
    getch();
    KillTimer();
    closegraph();
}
```

1.6.3 界面初始化

在每次开始游戏时，首先需要对游戏的界面进行初始化处理，然后在主函数中对其进行调用。初始化界面的处理流程如下所示：

(1) 循环调用函数 line()，用于绘制当前的游戏板；

(2) 调用函数 ShowScore()，显示得分，初始得分是 0；

(3) 调用函数 ShowSpeed()，显示等级速度，初始速度是 1。

在这里有两个参数需要特别注意：

- x，y：代表左上角坐标；
- m，n：对应于 Vertical_boxs，Horizontal_boxs，分别表示纵横方向上方块的个数(以方块为单位)。

界面初始化的具体实现代码如下所示：

```
/*************初始化界面*************/
void initialize(int x,int y,int m,int n){
    int i,j,oldx;
    oldx=x;
    for(j=0;j<n;j++)
    {
        for(i=0;i<m;i++)
        {
            Table_board[j][i].var=0;
            Table_board[j][i].color=BgCclor;
            line(x,y,x+BSIZE,y);
            line(x,y,x,y+BSIZE);
            line(x,y+BSIZE,x+BSIZE,y+BSIZE);
            line(x+BSIZE,y,x+BSIZE,y+BSIZE);
            x+=BSIZE;
        }
        y+=BSIZE;
        x=oldx;
    }
    Curbox_x=x;
    Curbox_y=y;                              /*x,y 是保存方块当前坐标的*/
    flag_newbox=false;                       /*是否要产生新方块的标记 0*/
    speed=0;
```

```
    score=0;
    ShowScore(score);
    ShowSpeed(speed);
}
```

1.6.4 时钟中断处理

在本项目中，为了提高玩家的兴趣，迎合玩家在虚拟世界中喜欢挑战刺激的心态，特意设置如果用户的级别越高则方块的下落速度就越快这一模式，这样就增加了游戏的难度。与之相对应的是，下落的速度越快，时间中断的间隔就越小。

在本项目中，时钟中断处理的流程如下：

(1) 定义时钟中断处理函数 newtimer()；

(2) 使用函数 SetTimer()来设置时钟中断处理的过程；

(3) 定义中断恢复函数 KillTimer()。

时钟中断处理后具体实现代码如下所示：

```
void interrupt newtimer(void)
{
    (*oldtimer)();
    TimerCounter++;
}
/* 设置新的时钟中断处理过程 */
void SetTimer(void interrupt(*IntProc)(void))
{
    oldtimer=getvect(TIMER);          /*获取中断号为 TIMER 的中断处理函数的入口地址*/
    disable();                        /* 设置新的时钟中断处理过程时，禁止所有中断 */
    setvect(TIMER,IntProc);
    /*将中断号为 TIMER 的中断处理函数的入口地址改为 IntProc()函数的入口地址
    即中断发生时，将调用 IntProc()函数。*/
    enable();                         /* 开启中断 */
}
/* 恢复原有的时钟中断处理过程 */
void KillTimer(){
    disable();
    setvect(TIMER,oldtimer);
    enable();
}
```

1.6.5 更新速度和成绩，显示帮助信息

随着游戏的进行，玩家的得分、方块的速度也在不断变化，项目的成绩、速度和帮助

是此游戏的重要组成部分，具体实现流程如下：

(1) 调用函数 ShowScore()，显示当前用户的成绩；

(2) 调用函数 ShowSpeed()，显示当前游戏的下落速度；

(3) 调用函数 Show_help()，显示和此游戏有关的帮助信息。

更新速度、成绩和显示帮助的具体实现代码如下所示：

```
/*显示分数*/
void ShowScore(int score){
    int x,y;
    char score_str[5];                                    /*保存游戏得分*/
    setfillstyle(SOLID_FILL,BgColor);
    x=LeftWin_x;
    y=100;
    bar(x-BSIZE,y,x+BSIZE*3,y+BSIZE*3);
    sprintf(score_str,"%3d",score);
    outtextxy(x,y,"SCORE");
    outtextxy(x,y+10,score_str);
}
/*显示速度*/
void ShowSpeed(int speed){
    int x,y;
    char speed_str[5];                                    /*保存速度值*/
    setfillstyle(SOLID_FILL,BgColor);
    x=LeftWin_x;
    y=150;
    bar(x-BSIZE,y,x+BSIZE*3,y+BSIZE*3);
    /*确定一个以(x1,y1)为左上角,(x2,y2)为右下角的矩形窗口，再按规定图模和颜色填充。*/
    sprintf(speed_str,"%3d",speed+1);
    outtextxy(x,y,"Level");
    outtextxy(x,y+10,speed_str);
    /*输出字符串指针 speed_str 所指的文本在规定的(x, y)位置*/
    outtextxy(x,y+50,"Nextbox");
}
void show_help(int xs,int ys)
{
char stemp[50];
setcolor(15);
rectangle(xs,ys,xs+239,ys+100);
sprintf(stemp," -Roll -Downwards");
stemp[0]=24;
stemp[8]=25;
setcolor(14);
outtextxy(xs+40,ys+30,stemp);
sprintf(stemp," -Turn Left    -Turn Right");
stemp[0]=27;
```

```
stemp[13]=26;
outtextxy(xs+40,ys+45,stemp);
outtextxy(xs+40,ys+60,"Esc-Exit");
setcolor(FgColor);
}
```

1.6.6 满行处理

在俄罗斯方块游戏中，如果方块满一行则代表成功，系统就会加分。因此，当用户对方块的左移、右移和旋转等操作不能处理时，需要对游戏进行是否满行的判断。如果有满行，则必须消除。满行处理的过程分为查找和消除两个步骤，具体实现流程如下所示。

(1) 调用函数 setFullRow()，查找是否有满行。

对当前方块的位置从上到下逐行判断，如果该行方块值为 1 的个数大于一行的块数时，则此时为满行。此时将调用函数 DelFullRow()进行满行处理，并返回当前游戏非空行的最高点，否则将继续对上一行进行判断，直到游戏的最上行。

如果有满行，则根据函数 DelFullRow()处理后的游戏主板 Table_board 数组中的值，进行游戏主板重绘，显示消除满行后的游戏界面，并同时对游戏成绩和速度进行更新。

函数 setFullRow()的具体实现代码如下所示。

```
/*找到一行满的情况*/
void setFullRow(int t_boardy)
{
   int n,full_numb=0,top=0;                          /*top 保存的是当前方块的最高点*/
   register m;
/*
t_boardy □      5
         □      6
   □□□□□□□   7
n  □□□□□□□   8
 */
   for(n=t_boardy+3;n>=t_boardy;n--)
   {
      if(n<0 || n>=Vertical_boxs ){continue;}        /*超过底线了*/
      for(m=0;m<Horizontal_boxs;m++)                 /*水平的方向*/
      {
         /*如果有一个是空就跳过该行*/
         if(!Table_board[n+full_numb][m].var)break;
      }
      if(m==Horizontal_boxs)                         /*找到满行了*/
      {
         if(n==t_boardy+3)
            top=DelFullRow(n+full_numb);             /*消除游戏板里的该行,并下移数据*/
```

```
            else
               DelFullRow(n+full_numb);
            full_numb++;                                    /*统计找到的行数*/
         }
      }
      if(full_numb)
      {
         int oldx,x=Sys_x,y=BSIZE*top+Sys_y; oldx=x;
         score=score+full_numb*10;                          /*加分数*/
         /*这里相当于重显调色板*/
         for(n=top;n<t_boardy+4;n++)
         {
            if(n>=Vertical_boxs)continue;                   /*超过底线了*/
            for(m=0;m<Horizontal_boxs;m++)                  /*水平的方向*/
            {
               if(Table_board[n][m].var)
                  setfillstyle(SOLID_FILL,Table_board[n][m].color);
               else
                  setfillstyle(SOLID_FILL,BgColor);
               bar(x,y,x+BSIZE,y+BSIZE);
               line(x,y,x+BSIZE,y);
               line(x,y,x,y+BSIZE);
               line(x,y+BSIZE,x+BSIZE,y+BSIZE);
               line(x+BSIZE,y,x+BSIZE,y+BSIZE);
               x+=BSIZE;
            }
            y+=BSIZE;
            x=oldx;
         }
         ShowScore(score);
         if(speed!=score/speed_step)
           {speed=score/speed_step; ShowSpeed(speed);}
         else
           {ShowSpeed(speed);}
      }
}
```

(2) 调用函数 DelFullRow()，删除满行。

当消除满行后，将上行的方块移至下行。函数 DelFullRow()的具体实现代码如下所示。

```
/*   删除一行满的情况,这里的 y 为具体哪一行为满*/
int DelFullRow(int y)
{
   /*该行游戏板往下移一行*/
   int n,top=0;      /*top 保存的是当前最高点,出现一行全空就表示为最高点了,移动到最高点结束*/
   register m,totoal;
   for(n=y;n>=0;n--)/*从当前行往上看*/
```

```
    {
        totoal=0;
        for(m=0;m<Horizontal_boxs;m++)
        {
            if(!Table_board[n][m].var)tctoal++;          /*没占有方格+1*/
           /*上行不等于下行就把上行传给下行 xor 关系*/
if(Table_board[n][m].var!=Table_board[n-1][m].var)
            {
                Table_board[n][m].var=Table_board[n-1][m].var;
                Table_board[n][m].color=Table_board[n-1][m].color;
            }
        }
        if(totoal==Horizontal_boxs)                      /*发现上面有连续的空行提前结束*/
        {
            top=n;
            break;
        }
    }
    return(top);                                         /*返回最高点*/
}
```

1.6.7 显示/消除方块

显示和消除是两个过程，首先出现一个新的方块供用户控制，我们可以控制方块的左移、右移、下移、旋转。当方块被放置以后，就需要让这个方块消失。表面看来整个过程十分复杂，其实实现起来很简单，具体实现流程如下：

(1) 调用函数 show_box()，设置从(x，y)位置开始，使用指定颜色 color 显示编号为 box_number 的方块；

(2) 调用函数 EraseBox()，消除从(x，y)处开始的编号为 box_number 的方块；

(3) 调用函数 MkNextBox()，将编号为 box_number 的方块作为当前的游戏编号，并随机生成下一个游戏方块编号。

显示和消除方块的具体实现代码如下所示：

```
void show_box(int x,int y,int box_numb,int color)
{
    int i,ii,ls_x=x;
    if(box_numb<0 || box_numb>=MAX_BOX)          /*指定的方块不存在*/
        box_numb=MAX_BOX/2;
    setfillstyle(SOLID_FILL,color);
/*********************************
 *   移位来判断哪一位是 1
 *   方块是每一行用半个字节来表示
 *   128d=1000 0000b
```

```
*********************************/
   for(ii=0;ii<2;ii++)
   {
      int mask=128;
      for(i=0;i<8;i++)
      {
         if(i%4==0 && i!=0)                    /*转到方块的下一行了*/
         {
            y+=BSIZE;
            x=ls_x;
         }
                if((shapes[box_numb].box[ii])&mask)
         {
            bar(x,y,x+BSIZE,y+BSIZE);
            line(x,y,x+BSIZE,y);
            line(x,y,x,y+BSIZE);
            line(x,y+BSIZE,x+BSIZE,y+BSIZE);
            line(x+BSIZE,y,x+BSIZE,y+BSIZE);
         }
         x+=BSIZE;
         mask/=2;
      }
      y+=BSIZE;
      x=ls_x;
   }
}
/*
*消除(x,y)位置开始的编号为box_numb的box.
*/
void EraseBox(int x,int y,int box_numb)
{
   int mask=128,t_boardx,t_boardy,n,m;
   setfillstyle(SOLID_FILL,BgColor);
   for(n=0;n<4;n++)
   {
      for(m=0;m<4;m++)                         /*设置四个单元*/
      {
         /*最左边有方块且当前游戏板也有方块*/
         if( ((shapes[box_numb].box[n/2]) & mask) )
         {
            bar(x+m*BSIZE,y+n*BSIZE,x+m*BSIZE+BSIZE,y+n*BSIZE+BSIZE);
            line(x+m*BSIZE,y+n*BSIZE,x+m*BSIZE+BSIZE,y+n*BSIZE);
            line(x+m*BSIZE,y+n*BSIZE,x+m*BSIZE,y+n*BSIZE+BSIZE);
            line(x+m*BSIZE,y+n*BSIZE+BSIZE,x+m*BSIZE+BSIZE,y+n*BSIZE+BSIZE);
            line(x+m*BSIZE+BSIZE,y+n*BSIZE,x+m*BSIZE+BSIZE,y+n*BSIZE+BSIZE);
         }
         mask=mask/(2);
```

```
            if(mask==0)mask=128;
        }
    }
}
/*将新形状的方块放置在游戏板上，并返回此方块号*/
int MkNextBox(int box_numb){
    int mask=128,t_boardx,t_boardy,n,m;
    t_boardx=(Curbox_x-Sys_x)/BSIZE;
    t_boardy=(Curbox_y-Sys_y)/BSIZE;
    for(n=0;n<4;n++)
    {
        for(m=0;m<4;m++)
        {
            if( ((shapes[current_box_numb].box[n/2]) & mask) )
            {
                Table_board[t_boardy+n][t_boardx+m].var=1;      /*设置游戏板*/
  Table_board[t_boardy+n][t_boardx+m].color=shapes[current_box_numb].color;
            }
            mask=mask/(2);
            if(mask==0)mask=128;
        }
    }
    setFullRow(t_boardy);
    Curbox_x=Sys_x+Begin_boxs_x*BSIZE,Curbox_y=Sys_y;          /*再次初始化坐标*/
    if(box_numb==-1) box_numb=rand()%MAX_BOX;
    current_box_numb=box_numb;
    flag_newbox=false;
    return(rand()%MAX_BOX);
}
```

1.6.8 对方块的操作处理

对俄罗斯方块的操作包括：左移、右移、下移、旋转和加速。在处理前要首先进行判断，如果满足条件则返回 True，即循序操作。此处的判断工作由函数 MoveAble 实现，(x，y)表示当前的方块位置，box_number 是方块的编号，direction 是左移、下移、右移和旋转的标志。方块操作的具体实现代码如下所示：

```
int MoveAble(int x,int y,int box_numb,int direction){
    /*t_boardx 是当前方块最左边在游戏板的位置*/
    int n,m,t_boardx,t_boardy;
    int mask;
    if(direction==MoveLeft)                                    /*如果向左移*/
    {
        mask=128;
        x-=BSIZE;
```

```
        t_boardx=(x-Sys_x)/BSIZE;
        t_boardy=(y-Sys_y)/BSIZE;
        for(n=0;n<4;n++)
        {
            for(m=0;m<4;m++)                                    /*最左边四个单元*/
            {
                /*如果最左边有方块并且当前游戏板也有方块*/
                if((shapes[box_numb].box[n/2]) & mask)
                {
                    if((x+BSIZE*m)<Sys_x)return(false);          /*如果碰到最左边了*/
                    /*左移一个方块后，此 4*4 的区域与游戏板有冲突*/
                    else if(Table_board[t_boardy+n][t_boardx+m].var)
                    {
                        return(false);
                    }
                }
                mask=mask/(2);
                if(mask==0)mask=128;
            }
        }
        return(true);
    }
    else if(direction==MoveRight)                               /*如果向右移*/
    {
        x+=BSIZE;
        t_boardx=(x-Sys_x)/BSIZE;
        t_boardy=(y-Sys_y)/BSIZE;
        mask=128;
        for(n=0;n<4;n++)
        {
            for(m=0;m<4;m++)                                    /*最右边四个单元*/
            {
                /*如果最右边有方块并且当前游戏板也有方块*/
                if((shapes[box_numb].box[n/2]) & mask)
                {
                    /*如果碰到最右边了*/
                    if((x+BSIZE*m)>=(Sys_x+BSIZE*Horizontal_boxs) )return(false);
                    else if( Table_board[t_boardy+n][t_boardx+m].var)
                    {
                        return(false);
                    }
                }
                mask=mask/(2);
                if(mask==0)mask=128;
            }
        }
        return(true);
```

```
}
else if(direction==MoveDown)                          /*如果向下移*/
{
    y+=BSIZE;
    t_boardx=(x-Sys_x)/BSIZE;
    t_boardy=(y-Sys_y)/BSIZE;
    mask=128;
    for(n=0;n<4;n++)
    {
        for(m=0;m<4;m++)                              /*最下边四个单元*/
        {
            /*最下边有方块并且当前游戏板也有方块*/
            if((shapes[box_numb].box[n/2]) & mask)
            {
                if((y+BSIZE*n)>=(Sys_y+BSIZE*Vertical_boxs) ||
                    Table_board[t_boardy+n][t_boardx+m].var)
                {
                    flag_newbox=true;
                    break;
                }
            }
            mask=mask/(2);
            /*mask 依次为:10000000,01000000,00100000,00010000
                        00001000,00000100,00000010/00000001
             */
            if(mask==0)mask=128;
        }
    }
    if(flag_newbox)
    {
        return(false);
    }
    else
        return(true);
}
else if(direction==MoveRoll)                          /*转动*/
{
    t_boardx=(x-Sys_x)/BSIZE;
    t_boardy=(y-Sys_y)/BSIZE;
    mask=128;
    for(n=0;n<4;n++)
    {
        for(m=0;m<4;m++)                              /*最下边四个单元*/
        {
            /*如果最下边有方块并且当前游戏板也有方块*/
            if((shapes[box_numb].box[n/2]) & mask)
            {
                /*如果碰到最下边了*/
                if((y+BSIZE*n)>=(Sys_y+BSIZE*Vertical_boxs) )return(false);
```

```
                /*如果碰到最左边了*/
                if((x+BSIZE*n)>=(Sys_x+BSIZE*Horizontal_boxs) )return(false);
                /*如果碰到最右边了*/
                if((x+BSIZE*m)>=(Sys_x+BSIZE*Horizontal_boxs) )return(false);
                else if( Table_board[t_boardy+n][t_boardx+m].var)
                {
                    return(false);
                }
            }
            mask=mask/(2);
            if(mask==0)mask=128;
        }
    }
    return(true);
  }
  else
  {
    return(false);
  }
}
```

1.7 测试运行

系统测试是整个项目的最后一步工作，本项目的程序文件命名为“youxi.c”，在 Turbo C 中打开该文件，如图 1-6 所示。

扫码看视频

按下 F9 键进行编译，按下快捷键 Alt+F5 开始运行，运行后的初始界面如图 1-7 所示。按下任意键后即可开始试玩游戏，游戏的试玩界面如图 1-8 所示。

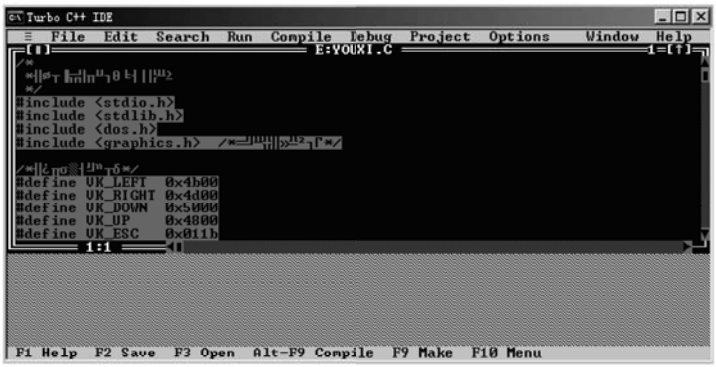

图 1-6　Turbo C 中的程序

图 1-7 初始界面

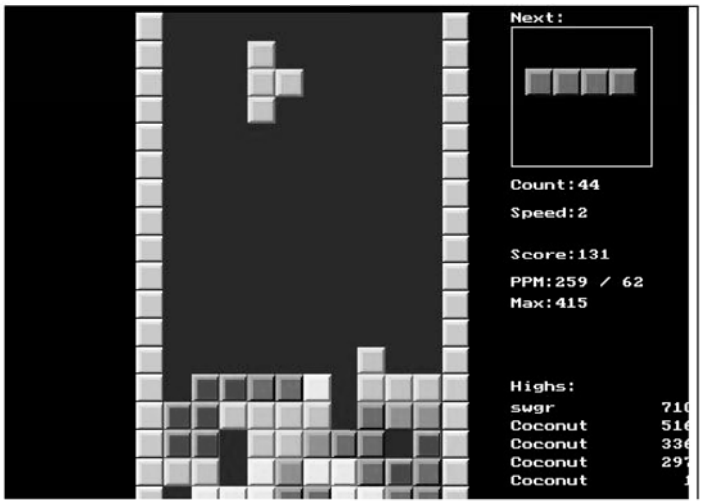

图 1-8 试玩界面