大学计算

实 验 报 告

实验名称：俄罗斯方块小游戏

学　　员：_____　学　　号：_____

年　　级：_____　所属学院：_____

专　　业：_____　实验日期：_____

指导教员：_____　职　　称：_____

《实验报告》填写说明

实验报告内容编排及打印应符合以下要求：

（1）采用 A4（21cm×29.7cm）白色复印纸，单面黑字打印。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

（2）报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）”……，第三级用“1.”、“2.”……，第四级用“（1）”、“（2）”……，分别按序连续编排。

（3）正文插图、表格中的文字字号均为 5 号。

一、实验目的和内容

实验目的：能够熟练掌握 C++语言的基本知识和技能。能够利用所学的基本知识和技能，解决简单的 C++程序设计问题。

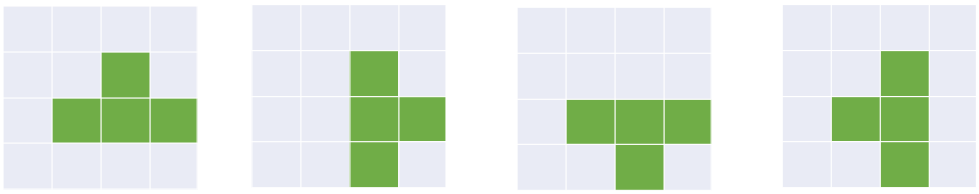
实验内容：实现一个俄罗斯方块主要功能模块。

二、实验设计及实现

完成了俄罗斯方块小游戏的基本功能：方块的向下移动、向左移动、向右移动及其可移动性判断，游戏的开始与结束，游戏分值的增加与难度动态调控，存档和读档功能，排行榜功能。在实验基本要求的基础上，加入了彩蛋功能，要求玩家通过分析游戏的实现，进行某些特定操作，才能够跳出分数循环，从而拿到达到最高等级结束游戏。

（一）游戏运行逻辑

开始游戏时，随机选取一种方块类型自顶而下掉落，通过改变[type][state]来确定具体的方块类型。例如，对于 T 型方块，可以通过维护一个二维数组 array[4][4]来描述其具体形态(图 2-1)。因此，要想通过枚举的方式表示所有方块，只需要维护一个四维数组 bricks[type][state][4][4]。数组 workRegion[20][10]描述了整个游戏界面所有方块的情况，即若 workRegion[i][j]处的数值为零，说明此处没有方块，反之则说明此处已经有方块。



```
{0,0,0,0},      {0,0,0,0},      {0,0,0,0},      {0,0,0,0},
{0,0,2,0},      {0,0,2,0},      {0,0,0,0},      {0,0,2,0},
{0,2,2,2},      {0,0,2,2},      {0,2,2,2},      {0,2,2,0},
{0,0,0,0}       {0,0,2,0}       {0,0,2,0}       {0,0,2,0}
```

图 2-1 T 型方块在数组中的表现形式

方块的下落由函数 MoveDown()实现，在每次下落后都会检测是否有某一行已经全部占满。若是，则通过函数 DeleteLines()将对应行全部消去，并将其上方的行依次顺势下移。每次执行完后都会刷新屏幕。

玩家的初始分数是 1 分，每消去一行，分数将乘以 2。游戏约定，当玩家的分数达到 16 分时，玩家获得游戏胜利。然而，全局变量 magicnum 的初始值为 0，这

意味着玩家的分数一旦达到 16 分，就会发生溢出现象(即游戏中所述的 overflow)，分数会重新变为 1 分。因此，玩家必须找到游戏的彩蛋功能，程序才会将 magicnum 设为 1，才不会出现 overflow 现象。

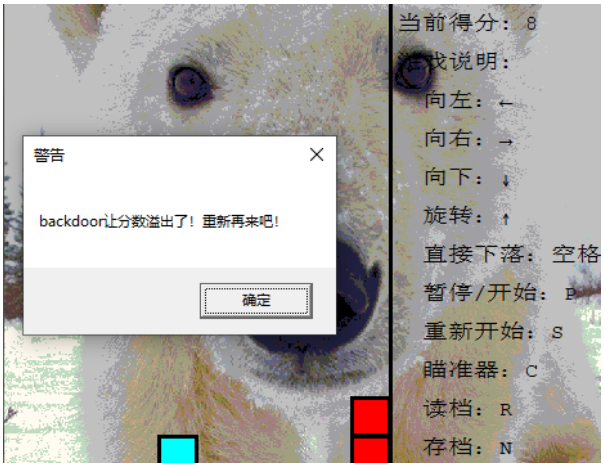


图 2-2 “分数溢出”示意图

存档操作由函数 saveNow()实现。该函数保存数组 workRegion 中的全部内容，并将其写入 fly.dat 文件中。此外，还将玩家目前的分数和 6 个随机数写入该文件中。这里的 6 个随机数与游戏彩蛋相关。

读档操作由函数 readSaved()实现。该函数从 fly.dat 中读取保存的 workRegion 的数据，并用读取的数据覆盖当前 workRegion 数组的全部值，实现屏幕内容的更新。此外，还会读入玩家先前分数和 6 个随机数。readSaved()函数会调用另一个函数 backdoor()，backdoor()函数用于检验 6 个随机数是否符合某种匹配关系。如果符合，则会将 magicnum 设置为 1，不会出现 overflow 现象；反之则将 magicnum 设置为 0。

（二）玩家操作

在游戏运行过程中，玩家通过按键盘上的↓键、←键、→键和↑键分别完成方块的下移、左移、右移和旋转。按空格键使得该方块在当前条件下移到它能到达的最低处。通过按 P 键实现游戏的暂停和开始；在游戏结束后，按 S 键可以重新开始。此外，按 C 键可以实现瞄准功能；按 R 键和 N 键可以分别实现读档和存档。

（三）游戏彩蛋

玩家必须先找出游戏彩蛋，才能绕过 overflow 现象。为了方便玩家通关，代码本身再优化时进行了一些处理，使玩家接下来得到的逆向文件具有更好的可读性。

从玩家本身的角度分析，游戏彩蛋的寻找具体方式如下。

注意到游戏过程中按下 N 键会将游戏存档，即在当前运行目录下创建 fly.dat 文

件。用文本编辑器打开 fly.dat 文件，一共有 207 个数。联想到界面只有 10×20 为 200 个单元格，且在暂停状态下多次按下 N 键，发现只有第 202 到第 207 个数发生了变化。推测这六个数字应当与当前游戏进度和状态无关。结合消行之后第 201 个数增大，且每消一行第 201 个数增加 1，猜想第 201 个数保存的是得分信息。而初始得分为 1 分，每消一行分数变为之前的 2 倍，说明实际分数等于 2 的第 201 个数次方。

现在重点关注第 202 到第 207 个数。在每次出现 overflow 现象时，程序都弹窗提示“backdoor”，猜想 backdoor 应当与阻断 overflow 现象有关。

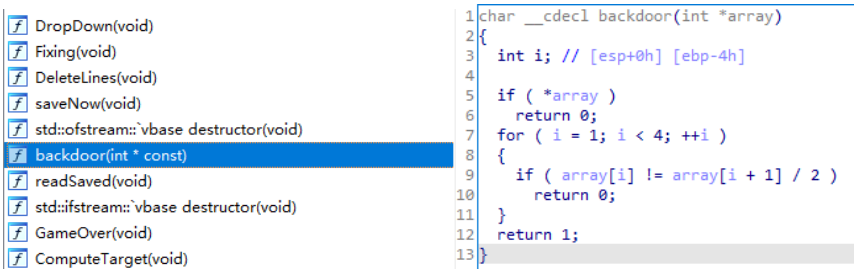


图 2-3 在 IDA 中找到逆向之后的 backdoor 函数

通过观察 backdoor 函数的 C-like 代码，玩家很容易明白这是一个判断传入的数组是否符合某种规则的函数。第 5 行，如果第一个数不是 0，则返回 0；从第三个数到第五个数，若都满足第 i 个数是第 i-1 个函数的 2 倍，才返回 1。而后通过搜索 magicnum 变量的设置位置，可以找到是在 readSaved()函数中。第六位数没有特别的要求。

```
void __cdecl readSaved()
{
  std::ifstream infile; // [esp+8h] [ebp-440h] BYREF
  int j; // [esp+C0h] [ebp-388h]
  int i; // [esp+C4h] [ebp-384h]
  int r; // [esp+C8h] [ebp-380h]
  int l; // [esp+CCh] [ebp-37Ch]
  int k; // [esp+D0h] [ebp-378h]
  char s; // [esp+D7h] [ebp-371h] BYREF
  int savedArray[21][10]; // [esp+D8h] [ebp-370h] BYREF
  int backdoorarray[6]; // [esp+420h] [ebp-28h] BYREF
  int v9; // [esp+444h] [ebp-4h]

  std::ifstream::ifstream(&infile, "fly.dat", 1, 64);
  v9 = 0;
  for ( i = 0; i < 21; ++i )
  {
    for ( j = 0; j < 10; ++j )
    {
      std::operator>><char,std::char_traits<char>>(&infile, &s);
      savedArray[i][j] = s - 48;
    }
  }
  for ( k = 0; k < 20; ++k )
  {
    for ( l = 0; l < 9; ++l )
      workRegion[k][l] = savedArray[k][l];
  }
  tScore = (int)pow<int,int,0>(2, savedArray[20][0]);
  for ( r = 0; r < 6; ++r )
    backdoorarray[r] = savedArray[20][r + 1];
  *(&winwin + 1) = backdoor(backdoorarray);
  std::ifstream::close(&infile);
  v9 = -1;
  std::ifstream::`vbase destructor(&infile);
}
```

图 2-4 readSaved()函数逆向后得到的 C-like 代码

观察代码，玩家完全可以知道游戏设置 overflow 的原理，也就是说，如果存档文本文件的最后六位顺利通过 backdoor()函数的测试，就将 magicnum 置 1，从而不会出现 overflow 现象。

至此，玩家通过修改 fly.dat 文件，就可以绕过 overflow 的检测，从而获得游戏的胜利。

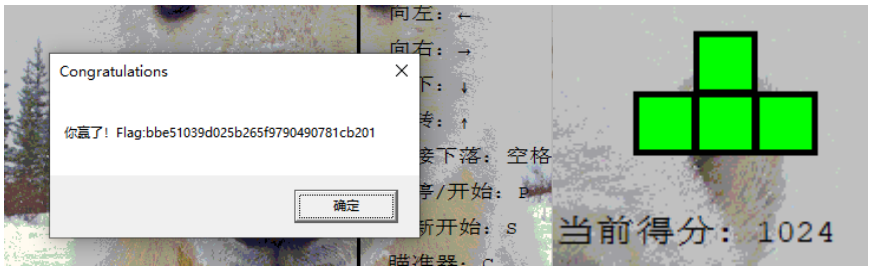


图 2-5 成功后提示，成功后分数会变成 1024

（四）难度提升

随 tScore 值的增加，方块掉落速度和等待下一方块时间随 tScore 呈线性变化。具体对应关系为

$$v = (\text{int})v_0(1-\frac{\text{tScore}}{20}),$$

其中 v 指当前速度，v_0 指初始速度，tScore 表示当前得分。

（五）其他细节功能

1. 菜单栏

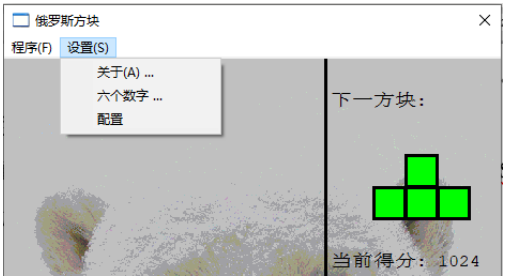


图 2-6 菜单栏

点击菜单栏可以看到关于、六个数字和配置，其中关于一栏给出了故事背景，六个数字一栏给出了绕过 backdoor 的提示，配置一栏可以进行有关配置。

2. wudi 模式

“wudi 模式”是帮助玩家快速消除整块区域的方法，单次使用、立即生效，点击后若游戏处于有效状态(不是游戏结束状态)界面将被清空，即强行消除所有的方

块。

3. 背景音乐的开关

图 2-7 背景音乐的开关

通过单击背景音乐开关按钮，可以开启或关闭背景音乐。

三、实验结果 总结与展望

学习使用了 win32 开发的 windows.h 库；练习了 C++基本操作及语言知识，例如数组、函数、指针；练习了使用 objdump、gdb 和 IDA 对二进制文件进行逆向分析；分析了编译时采用的不同优化等级对程序的影响。

实现了一个俄罗斯方块的主要功能。

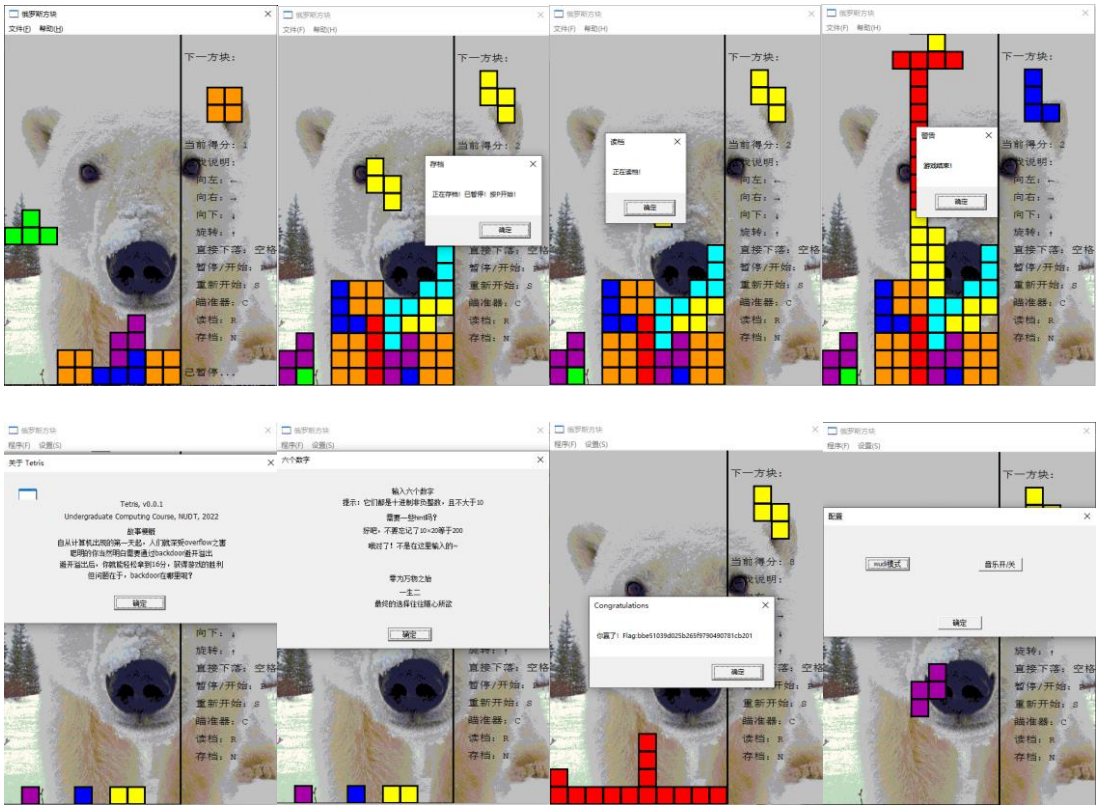


图 3-1 程序运行示意图

源代码开源在 https://www.gitlink.org.cn/hexu/tetris。

四、问题与建议

该工作还有一部分提升空间，例如在之后的工作中，可以使窗口大小自适应屏幕。此外，配置信息都被确定在程序内部，可以考虑创建配置文件，使玩家能够根据自身需求进行个性化配置。在程序的优化上，一方面可以改变数据结构与算法，使数据和函数更好地组织，另一方面，还有基于计算机原理的优化方式，例如（1）注意到对 workRegion 和 bricks 这两个数组的操作可以通过改进函数，使其具有较好的时间局部性，从而提高 cache 命中率；（2）通过循环展开，减小在遍历 workRegion 数组过程中带来的不必要的计算，优化程序性能；（3）在通过对数组中的元素判断其是否为零来确定某区域是否存在方块时，采用 switch 方式，减小分支预测错误带来的额外开销；（4）采用更高的编译优化方案，例如-O2 或-O3（本实验中的所有代码均未开启任何编译优化）。

五、附录：部分代码说明

（一）判断能否向下移动

```
1.  // 判断当前方块是否可以向下移动
2.  bool CanMoveDown() {
3.    int maxy = -20;
4.    int anyx = 20;
5.    int flag = 0;
6.    for (int i = 0; i < 4; i++) {
7.      for (int j = 0; j < 4; j++) {
8.        flag = bricks[type][state][i][j];
9.        if (flag > 0 && workRegion[point.y + i + 1][point.x + j] > 0)
10.         return 0;
11.       if (flag > 0 && maxy < i) {
12.         anyx = j;
13.         maxy = i;
14.     }
15.     }
16.   }
17.   if (bricks[type][state][maxy][anyx] > 0 && point.y + maxy == 19)
18.     return 0;
19.   return 1;
20. }
```

实现思路：首先找出最靠左下方的值不为 0 的元素，并标记其下标；通过其下标计算该元素相对于基准点 point 的位置。返回 0 的条件是：最靠左下方的元素已经移动到最后一行（line 17）或位于该方块最底一行的方块下方紧挨着还有方块（line 9），则不能够继续下移。

事实上，代码可以写得更简短一些。

```
1.  for (int i = 0; i < 4; i++) {
```

```
    for (int j = 0; j < 4; j++) {
      int flag = bricks[type][state][i][j];
      if ((flag > 0 && point.y + i + 1 >= 0) && (point.y + i == 19 || workRegion[point.y + i + 1][point.x + j] > 0))
        return 0;
    }
  }
  return 1;
```

左移和右移的实现逻辑与此类似。

（二）旋转当前方块

```
void Rotate() {
 state = (state + 1) % 4;
 Point newp = point;
 for (int i = 0; i < 4; i++) {
  for (int j = 0; j < 4; j++) {
   if (bricks[type][state][i][j] > 0) {
    if (point.x + j < 0)
     point.x = -j;
    if (point.x + j > 9)
     point.x = 9 - j;
   }
  }
 }
 for (int i = 0; i < 4; i++) {
  for (int j = 0; j < 4; j++) {
   if (bricks[type][state][i][j] > 0 && workRegion[point.y + i][point.x + j] > 0) {
    state = (state + 3) % 4;
    return;
   }
  }
 }
}
```

值得注意的部分是对于 point.x 坐标的修正。原理还是相对简单的，即如果旋转之后图形超过边界，则将图形对其到边界。

（三）存档

```
// 存档
void saveNow() {
  ofstream outfile;
  outfile.open("fly.dat", ios::trunc);
  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 10; j++) {
      outfile << workRegion[i][j] << " ";
    }
  }
  int powtScore = 0;
  switch (tScore){
  case 1:
    powtScore = 0;
    break;
  case 2:
    powtScore = 1;
```

```
    break;
  case 4:
    powtScore = 2;
    break;
  case 8:
    powtScore = 3;
    break;
  case 16:
    powtScore = 4;
    break;
  default:
    break;
  }
  outfile << powtScore << " ";
  for (int r = 0; r < 6; r++) {
    outfile << rand() % 10 << " ";
  }
  outfile.close();
}
```

存档时，将 workRegion 的全部元素依次输出保存到 fly.dat 文件中，以空格作为间隔。由于分数是 2 的幂次，所以只保存了 2 的幂的值；这是由于读入文件使将其存放到 char 类型的数组中而做出的兼容性考虑。同时，还在最后保存了 6 位随机数字，与彩蛋相关。

（四）读档

```
void readSaved() {
  ifstream infile("fly.dat");
  char s;
  int savedArray[21][10];
  for (int i = 0; i < 21; i++) {
    for (int j = 0; j < 10; j++) {
      infile >> s;
      savedArray[i][j] = (int)s - 48;
    }
  }
  for (int i = 0; i < 20; i++) {
    for (int j = 0; j < 9; j++) {
      workRegion[i][j] = savedArray[i][j];
    }
  }
  tScore = pow(2, savedArray[20][0]);
  int backdoorarray[6];
  for (int r = 0; r < 6; r++) {
    backdoorarray[r] = savedArray[20][r + 1];
  }
  magicnum = backdoor(backdoorarray);
  infile.close();
}
```

读档时，将文档中数字依次读入，并存放到数组 savedArray 中，然后通过幂次算出 tScore 的值（事实上，此处将一个浮点数截断为了整数，编译时会有警告可能有精度缺失，但并不影响数值准确性），最后调用 backdoor()函数检验随机数是否符

合规范。