**深 圳 大 学 实 验 报 告**

**课程名称： 计算机图形学**

**实验项目名称： 期中作业 俄罗斯方块**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 黄惠、胡瑞珍**

**报告人： 刘清影 学号： 2017152021 班级： 06**

**实验时间： 2018年 09月26日 -- 2018年10月 31日**

**实验报告提交时间： 2018 年 10 月 31 日**

**教务部制**

|  |
| --- |
| 目录  [实验目的 3](#_Toc528749911)  [实验步骤 3](#_Toc528749912)  [一、 棋盘和方块的渲染 3](#_Toc528749913)  [二、 新生成一个方块 5](#_Toc528749914)  [三、 移动方块 7](#_Toc528749915)  [四、 旋转方块 9](#_Toc528749916)  [五、 方块放置 11](#_Toc528749917)  [六、 格子的行消去 11](#_Toc528749918)  [七、 鼠标键盘交互 12](#_Toc528749919)  [八、 实验过程中需要补充的细节 16](#_Toc528749920)  [实验结果 21](#_Toc528749921)  [实验心得 22](#_Toc528749922) 实验目的  1. 设计一个简化版的俄罗斯方块游戏。 2. 更加熟练二维图形的描绘。 3. 添加合适的鼠标键盘交互，添加菜单。 4. 学会熟练运用glut的函数，比如使用空闲回调函数glutIdleFunc、定时器glutTimerFunc等等。  实验步骤  一、 棋盘和方块的渲染 1. 生成棋盘的纵向的11条线（因为有10格子）。  对于第i条线，x坐标为(33.0 + (33.0 \* i)；对于y，一端为33，另一端为33\*21=693。两端的y值相差33\*20，这样就能够在纵轴方向平分20个长度为33的格子了。(注：33.0为一个格子的宽度)  // 纵向线  **for** **(**int i **=** 0**;** i **<** 11**;** i**++)**  **{**  //相对于世界坐标系，这个点是位于y=33.0的位置  gridpoints**[**2**\***i**]** **=** vec4**((**33.0 **+** **(**33.0 **\*** i**)),** 33.0**,** 0**,** 1**);**  //693=33\*21，相对于世界坐标系，这个点是位于y=33.0\*21的位置，一共相差20个格子  gridpoints**[**2**\***i **+** 1**]** **=** vec4**((**33.0 **+** **(**33.0 **\*** i**)),** 693.0**,** 0**,** 1**);**  **}**  2. 同1理，生成棋盘的水平方向的21条线的代码如下：  // 水平线  **for** **(**int i **=** 0**;** i **<** 21**;** i**++)**  **{**  //相对于世界坐标系，这个点是位于x=33.0的位置  gridpoints**[**22 **+** 2**\***i**]** **=** vec4**(**33.0**,** **(**33.0 **+** **(**33.0 **\*** i**)),** 0**,** 1**);**  //相对于世界坐标系，这个点是位于x=11\*33=363的位置  gridpoints**[**22 **+** 2**\***i **+** 1**]** **=** vec4**(**363.0**,** **(**33.0 **+** **(**33.0 **\*** i**)),** 0**,** 1**);**  **}**  3. 给线赋值颜色：  // 将所有线赋成白色  **for** **(**int i **=** 0**;** i **<** 64**;** i**++)**  gridcolours**[**i**]** **=** white**;**  4. 生成棋盘的格子。  每个正方形格子看成两个三角形组成的，如下图：  C:\Users\Lewin\AppData\Local\Temp\WeChat Files\300090556332333735.jpg  如上图，我们可以把正方形p1p2p3p4看成三角形p1p2p3和三角形p2p3p4。所以生成正方形格子的代码如下：  // 对每个格子，初始化6个顶点，表示两个三角形，绘制一个正方形格子  **for** **(**int i **=** 0**;** i **<** 20**;** i**++)**  **for** **(**int j **=** 0**;** j **<** 10**;** j**++)**  **{**  vec4 p1 **=** vec4**(**33.0 **+** **(**j **\*** 33.0**),** 33.0 **+** **(**i **\*** 33.0**),** .5**,** 1**);**  vec4 p2 **=** vec4**(**33.0 **+** **(**j **\*** 33.0**),** 66.0 **+** **(**i **\*** 33.0**),** .5**,** 1**);**  vec4 p3 **=** vec4**(**66.0 **+** **(**j **\*** 33.0**),** 33.0 **+** **(**i **\*** 33.0**),** .5**,** 1**);**  vec4 p4 **=** vec4**(**66.0 **+** **(**j **\*** 33.0**),** 66.0 **+** **(**i **\*** 33.0**),** .5**,** 1**);**  boardpoints**[**6**\*(**10**\***i **+** j**)** **]** **=** p1**;**  boardpoints**[**6**\*(**10**\***i **+** j**)** **+** 1**]** **=** p2**;**  boardpoints**[**6**\*(**10**\***i **+** j**)** **+** 2**]** **=** p3**;**  boardpoints**[**6**\*(**10**\***i **+** j**)** **+** 3**]** **=** p2**;**  boardpoints**[**6**\*(**10**\***i **+** j**)** **+** 4**]** **=** p3**;**  boardpoints**[**6**\*(**10**\***i **+** j**)** **+** 5**]** **=** p4**;**  **}**  5. 设置每一个格子的颜色。  一开始默认格子全部为黑色。  // 初始化棋盘格，并将没有被填充的格子设置成黑色  vec4 boardpoints**[**1200**];**  **for** **(**int i **=** 0**;** i **<** 1200**;** i**++)**  boardcolours**[**i**]** **=** black**;**  6. 然后将顶点和颜色信息传给GPU描绘即可。 二、 新生成一个方块 1. 首先判断是否要生成一个方块。  如果当前游戏已经结束或者当前游戏处于暂停状态，则不生成方块。  //如果游戏结束或者当前模式为PAUSE，则不再生成新的方块  **if** **(**gameover**||**curMode**==**PAUSE**)** **{**  **return;**  **}**  2. 定义新的方块的状态。  方块**中心位置tilepos**的默认位置在(5,19)。但是这样可能会遇到新生成的方块溢出边界的问题。于是，我就先判定当前位置(即当中心在(5,19)的时候)是否会溢出，如果会溢出，则整个方块整体下移。  除了中心位置，还需要随机生成**方块的类型**、**方块的方向**和**方块的颜色**。代码如下：  // 将新方块放于棋盘格的最上行中间位置  tilepos **=** vec2**(**5 **,** 19**);**  //如果当前生成的方块超出边界，则将方块下移一格就不会超出边界了，downOneCell的作用就是用来记录是否需要下移一格  bool downOneCell **=** **false;**  //随机生成一个方向  rotation **=** rand**()%**4**;**  //随机生成一种方块的类型  type **=** rand**()** **%** 7**;**  //生成一个随机颜色  int randColor **=** rand**()** **%** colorTypeNumber**;**  //用来存储新的方块的颜色，一共有4个格子，每个格子需要6个定点，所以数组的大小为4\*6=24  vec4 newcolours**[**24**];**  //检测当前方块是否超出边界，如果是则需要下移一格  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)**  **{**  //设置当前的tile  tile**[**i**]** **=** allRotationsshapes**[**type**][**rotation**][**i**];**  //当前生成的方块坐标  int x **=** tilepos**.**x **+** tile**[**i**].**x**;**  int y **=** tilepos**.**y **+** tile**[**i**].**y**;**  //如果超出边界  **if** **(**y **>** 19**)** **{**  downOneCell **=** **true;**  **break;**  **}**  **}**  //如果需要往下移动一个格子，使得生成的方块在边界内  **if** **(**downOneCell**)** **{**  //当前方块下移一格  tilepos**.**y**--;**  **}**  3. 将新生成的方块的坐标信息和颜色信息保存在board和boardcolours数组中。  因为在保存的过程中需要遍历当前方块的四个格子，所以我们也可以同时检查是否会导致游戏结束。当新生成的格子中的board已经是被填充的话就说明游戏需要结束了。  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)**  **{**  //获取当前方块的四个格子的位置  tile**[**i**]** **=** allRotationsshapes**[**type**][**rotation**][**i**];**  int x **=** tilepos**.**x **+** tile**[**i**].**x**;**  int y **=** tilepos**.**y **+** tile**[**i**].**y**;**  //判断是否被填充，如果刚生成的位置都被填充的话，则游戏结束  **if** **(**board**[**x**][**y**])** **{**  gameover **=** **true;**  //输出游戏结束的信息  printInformation**();**  //直接返回，不再继续后继操作  **return;**  **}**  //保存每一个新生成的格子的颜色信息  changeBorderColors**(**tilepos **+** tile**[**i**],** tileColor**[**randColor**]);**  //将每一个新生成的格子的位置的board设置为true  setTrail**(**tilepos **+** tile**[**i**]);**  **}**  4. 最后还需要更新VBO，将顶点和颜色的信息传递给GPU。 三、 移动方块 1. 检查是否能够移动方块。  这里值得注意的是，不能够直接去判断，因为如果没有清除原来的方块的坐标的信息，那么**有可能出现移动后的方块和原来的方块有重叠的部分**，从而导致下面的check有误。所以这里的解决办法是，先消去未移动的方块的坐标信息，然后再去检查是否能够移动。  比如下面这个例子，红色的是没有移动的方块，黑色的是移动后的方块。如果没有清除原来的board信息，那么移动后的正方形的上面的两个格子的board是true，所以这个时候就会导致无法移动。（**因为移动需要检查四个格子的board**）  C:\Users\Lewin\AppData\Local\Temp\WeChat Files\681778924026676341.jpg  清除了原来的方块的board的信息之后，就可以进行检查是否能移动了。  // 计算移动之后的方块的位置坐标  vec2 newtilepos**[**4**];**  //先尝试移动,去除没有移动前的board的信息（设置为false）  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  newtilepos**[**i**]** **=** tile**[**i**]** **+** tilepos **+** direction**;**  clearTrail**(**tile**[**i**]** **+** tilepos**);**  **}**  **if** **(**checkvalid**(**newtilepos**[**0**])**  **&&** checkvalid**(**newtilepos**[**1**])**  **&&** checkvalid**(**newtilepos**[**2**])**  **&&** checkvalid**(**newtilepos**[**3**]))**  **{**  ……  **}**  **这里值得注意的是，如果检查到不能移动，则需要恢复当前方块的board信息。**  //如果不能够移动，则恢复方块的填充信息  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  setTrail**(**tile**[**i**]** **+** tilepos**);**  **}**  2. 如何移动方块？  前面已经说过了，移动需要先清除原来的格子。因为前面已经删除了board信息，所以这里只需要删除原来格子的颜色信息(boardcolours)即可。但是值得注意的是，在清除之前，我们还需要保存它的颜色信息。  vec4 lastColor**[**6**];**  //保存移动前的颜色  **for** **(**int i **=** 0**;** i **<** 6**;** i**++)** **{**  int x **=** **(**tile**[**i**]** **+** tilepos**).**x**;**  int y **=** **(**tile**[**i**]** **+** tilepos**).**y**;**  lastColor**[**i**]** **=** boardcolours**[**6 **\*** **(**10 **\*** y **+** x**)];**    **}**  //删除原来的格子  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  //将当前位置设置成黑色  changeBorderColors**(**tile**[**i**]+**tilepos**,** black**);**  **}**  删除完了原来的信息之后，我们就可以生成新的方块了。  生成方块只需要设定方块的board和boardcolours即可。  //重新生成新的格子  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  setTrail**(**newtilepos**[**i**]);**  //给新的位置的方块赋予移动前的颜色，即移动不改变方块的颜色  changeBorderColors**(**newtilepos**[**i**],** lastColor**[**0**]);**  **}**  最后我们需要更新tilepos和VBO  //重新更新tilepos的位置，不涉及到旋转，所以不必更新tile  tilepos**.**x **=** tilepos**.**x **+** direction**.**x**;**  tilepos**.**y **=** tilepos**.**y **+** direction**.**y**;**  //更新VBO  updatetile**();** 四、 旋转方块 1. 先计算选择后的下一个方向。  // 计算得到下一个旋转方向  int nextrotation **=** **(**rotation **+** 1**)** **%** 4**;**  下一个方向就是当前方向加1。对4求余的作用是：因为方向的下标只有0，1，2，3，所以当旋转前的方向为3时，下一个方向应该为0。  2. 跟移动方块类似，我们需要先尝试移动，删除原来的board信息，然后再去判断是否能够旋转。  //先尝试rotate，这样再去检测有有效性，类似于回溯法  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  clearTrail**(**tile**[**i**]** **+** tilepos**);**  **}**  3. 如果能够旋转，先清除原来方块的颜色的信息，在删除之前需要先保存原来的颜色信息在lastColor[4]中。  vec4 lastColor**[**4**];**  //删除当前格子  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  //保存格子的颜色信息  lastColor**[**i**]** **=** getColor**(**tilepos **+** tile**[**i**]);**  //这个方块的格子全部设置成黑色  changeBorderColors**(**tilepos **+** tile**[**i**],** black**);**  //清除填充信息  clearTrail**(**tile**[**i**]** **+** tilepos**);**  **}**  //重新生成一个新的格子  rotation **=** nextrotation**;**  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)** **{**  //更新tile  tile**[**i**]** **=** allRotationsshapes**[**type**][**rotation**][**i**];**  //设置旋转后的颜色  changeBorderColors**(**tilepos **+** tile**[**i**],** lastColor**[**i**]);**  //重新设置填充信息  setTrail**(**tile**[**i**]** **+** tilepos**);**  **}**  4. 生成新的顶点和颜色后，我们最后需要更新VBO  updatetile**();** //更新VBO 五、 方块放置 1. 储存方块4个格子的的board和boardcolours信息。  // 每个格子  **for** **(**int i **=** 0**;** i **<** 4**;** i**++)**  **{**  // 获取格子在棋盘格上的坐标  int x **=** **(**tile**[**i**]** **+** tilepos**).**x**;**  int y **=** **(**tile**[**i**]** **+** tilepos**).**y**;**  // 将格子对应在棋盘格上的位置设置为填充  board**[**x**][**y**]** **=** **true;**  // 并将相应位置的颜色修改  //更新VBO  changecellcolour**(**vec2**(**x**,**y**),** boardcolours**[**6**\*(**10**\***y**+**x**)]);**  **}**  2. 当放置方块的时候，需要检查是否需要消去行。  //从y=0开始扫描（从下往上），判断是否需要消去行。  **for** **(**int row **=** 0**;** row **<** 19**;** row**++)** **{**  **if** **(**checkfullrow**(**row**))** **{**  clearRow**(**row**);**  //如果消去了一行，则上面的行掉了下来，所以此时row不必加加  row**--;**  **}**  **}** 六、 格子的行消去 1. 当消去一行的时候，需要更新游戏信息（比如分数和下降速度）  //每消去一行，分数都加上speed的平方，speed变成原来的1.2倍  score **+=** speed**\***speed**;**  speed **\*=** 1.2**;**  2. 消去第row行，需要将row行上面的所有方块下移一格。即把上一格的board和boardcolors的信息赋值给当前格，最后需要更新VBO。  **for** **(**int i **=** row**;** i **<** 19**;** i**++)** **{**  **for** **(**int j **=** 0**;** j **<** 10**;** j**++)** **{**  //将上一行往下移  setTrail**(**vec2**(**j**,** i**),** vec2**(**j**,** i **+** 1**));**  //注意下面两个函数是不一样的  //将当前行的颜色变成上一行的颜色  changeBorderColors**(**vec2**(**j**,** i**),** getColor**(**vec2**(**j**,** i **+** 1**)));**  //更新当前颜色的VBO  changecellcolour**(**vec2**(**j**,** i**),** getColor**(**vec2**(**j**,** i**)));**  **}**  **}**  3. 最后还需要更新终端的游戏提示信息。  //更新终端的游戏信息  printInformation**();** 七、 鼠标键盘交互 1. 当游戏处于游戏中(CONTINUE)的状态时的**特殊按键响应**。  a. 向按下向上键的时候，旋转方块  b. 当按下向下键的时候，如果可以向下移动，则向下移动方块，否则放置该方块，然后再重新生成一个新的方块。  c. 当按下左键的时候，向左移动方块  d. 当按下右键的时候，向右移动方块  if(!gameover&&curMode==CONTINUE)  {  switch(key)  {  // 向上按键旋转方块  case GLUT\_KEY\_UP:  rotate();  break;  // 向下按键移动方块  case GLUT\_KEY\_DOWN:  {  //判断是否可以向下移动  if (!movetile(vec2(0, -1)))  {  //如果能向下移动，则放置当前的方块，重新生成方块  settile();  newtile();  }  break;  }  // 向左按键移动方块  case GLUT\_KEY\_LEFT:  movetile(vec2(-1, 0));  break;  // 向右按键移动方块  case GLUT\_KEY\_RIGHT:  movetile(vec2(1, 0));  break;  }  }    2. 当游戏处于游戏中(CONTINUE)的状态时的**普通按键响应**。  a. ESC键和’q’键退出游戏  b. ‘r’键重新开始游戏  c. ‘p’键暂停游戏  d. ‘c’键继续游戏  e. ‘s’键减小移动速度  f. ‘w’键加速  void keyboard**(**unsigned char key**,** int x**,** int y**)**  **{**    **switch(**key**)**  **{**  // ESC键 和 'q' 键退出游戏  **case** 033**:**  exit**(**EXIT\_SUCCESS**);**  **break;**  **case** 'q'**:**  exit **(**EXIT\_SUCCESS**);**  **break;**  // 'r' 键重启游戏  **case** 'r'**:**  restart**();**  **break;**  //'p'键暂停游戏  **case** 'p'**:**  curMode **=** PAUSE**;**  **break;**  //'c'键继续游戏  **case** 'c'**:**  curMode **=** CONTINUE**;**  **break;**  //'s'键减速方块下落的速度  **case** 's'**:**  **if(**speed**/**2**>**0**)**  speed **/=** 2**;**  //重新输出速度的信息  printInformation**();**  **break;**  //'w'键加速方块下落的速度  **case** 'w'**:**  speed **\*=** 2**;**  //重新输出速度的信息  printInformation**();**  **break;**  //重绘  glutPostRedisplay**();**  **}**    **}**  3. **鼠标响应回调**  a. 左键继续游戏  b. 右键暂停游戏。  //鼠标响应： 继续游戏和暂停游戏  void mouse**(**int button**,** int state**,** int x**,** int y**)** **{**  //如果点击左键，则继续游戏  **if** **(**button **==** GLUT\_LEFT\_BUTTON**)** **{**  curMode **=** CONTINUE**;**  **}**  **if** **(**button**==** GLUT\_RIGHT\_BUTTON**)** **{**  curMode **=** PAUSE**;**  **}**  **}**  4. **添加菜单**  当按下鼠标中键的时候可以选择我们的菜单。  a. 创建菜单  创建菜单入口： Continue, Pause, Restart, Quit，然后跟鼠标中键关联起来。  void setupMenu**()**  **{**  //创建菜单回调函数  glutCreateMenu**(**menuEvents**);**  //为菜单添加选择模式的选项  glutAddMenuEntry**(**"Continue"**,** CONTINUE**);**  glutAddMenuEntry**(**"Pause"**,** PAUSE**);**  glutAddMenuEntry**(**"Restart"**,** RESTART**);**  glutAddMenuEntry**(**"Quit"**,** QUIT**);**  //将菜单和中键关联  glutAttachMenu**(**GLUT\_MIDDLE\_BUTTON**);**  **}**  b. 创建菜单事件按钮  当鼠标传给了不同的值，鼠标的响应事件也不同。  void menuEvents**(**int menuChoice**)**  **{**  **switch** **(**menuChoice**)**  **{**  **case** CONTINUE**:**  //设置当前模式为CONTINUE,即游戏中  curMode **=** CONTINUE**;**  **break;**  **case** PAUSE**:**  //设置当前模式为PAUSE，即暂停游戏  curMode **=** PAUSE**;**  **break;**  **case** RESTART**:**  //重新开始游戏  restart**();**  **break;**  **case** QUIT**:**  //强制按下'q'键，以实现强制退出  keyboard**(**'q'**,** 0**,** 0**);**  **break;**  **}**  **}** 八、 实验过程中需要补充的细节 1. 补充所有可能出现的方块和它的方向。用三维数组来表示，第一维表示7种类型的形状，第二种表示它的方向，最后一维表示方块的四个格子相对于中心的位置。代码详情请看源代码的allRotationsshapes**[**7**][**4**][**4**]**数组。  2. 定义游戏中的状态和游戏的模式：  //定义游戏当前的模式---暂停或游戏中  const int CONTINUE **=** 1**;** //游戏中  const int PAUSE **=** 0**;** //暂停  const int RESTART **=** 2**;** //重新游戏  const int QUIT **=** 3**;** //退出游戏  int curMode **=** CONTINUE**;** //当前模式，默认为游戏中  //定义游戏的状态---分数和速度  const int INIT\_SPEED **=** 10**;** //游戏的初始速度  int score **=** 0**;** //游戏的分数  int speed **=** INIT\_SPEED**;** //游戏的速度  3. 定义方块的种类和颜色  //设置方格的种类  int type**;**  //设置格子的颜色种类个数  const int colorTypeNumber **=** 7**;**  //设置格子的颜色  vec4 tileColor**[**colorTypeNumber**]** **=** **{**  vec4**(**1.0**,** 0.5**,** 0.0**,** 1.0**),** //橙色  vec3**(**1.0**,** 1.0**,** 0.0**),** // Yellow  vec3**(**0.0**,** 1.0**,** 0.0**),** // Green  vec3**(**0.0**,** 1.0**,** 1.0**),** // Cyan  vec3**(**1.0**,** 0.0**,** 1.0**),** // Magenta  vec3**(**1.0**,** 0.0**,** 0.0**),** // Red  vec3**(**0.0**,** 0.0**,** 1.0**)** // Blue  **};**  4. 终端提示信息  void printInformation**()** **{**  //清屏  system**(**"cls"**);**  //输出当前的分数  cout **<<** "\*\*\*\*\*\*\*\*\*\*\*current score\*\*\*\*\*\*\*\*\*\*\*" **<<** endl**;**  std**::**cout **<<** "score: " **<<** score **<<** "\n"**;**  cout **<<** endl **<<** endl**;**  //输出当前的下降速度  cout **<<** "\*\*\*\*\*\*\*\*\*\*\*current speed\*\*\*\*\*\*\*\*\*\*\*" **<<** endl**;**  std**::**cout **<<** "speed: " **<<** speed **<<** "\n"**;**  cout **<<** endl **<<** endl**;**  //输出键盘的交互操作  cout **<<** "\*\*\*\*\*\*\*\*\*\*\*press keyboard\*\*\*\*\*\*\*\*\*\*\*" **<<** endl**;**  cout **<<** "r---restart" **<<** endl**;**  cout **<<** "q---quit" **<<** endl**;**  cout **<<** "c---continue" **<<** endl**;**  cout **<<** "p---pause" **<<** endl**;**  cout **<<** "w---speed up" **<<** endl**;**  cout **<<** "s---slow down" **<<** endl**;**  cout **<<** endl **<<** endl**;**  //输出鼠标的交互曹祖  cout **<<** "\*\*\*\*\*\*\*\*\*\*\*click mouse\*\*\*\*\*\*\*\*\*\*\*" **<<** endl**;**  cout **<<** "left button---continue" **<<** endl**;**  cout **<<** "right button---pause" **<<** endl**;**  cout **<<** "middle button---menu" **<<** endl**;**  cout **<<** endl **<<** endl**;**  //如果游戏结束，则输出"gameover"提示  **if** **(**gameover**)** **{**  std**::**cout **<<** "\*\*\*\*\*\*\*\*\*\*\*gameover\*\*\*\*\*\*\*\*\*\*\*"**<<**endl**;**  cout **<<** "Your final score is " **<<** score **<<** endl**;**  **}**  **}**  5. 检查cellpos的格子是否能够填充。  如果已经被填充或者不在边界内，则不能够被填充。  如果在边界内且没有被填充，那么就能够被填充。  // 检查在cellpos位置的格子是否被填充或者是否在棋盘格的边界范围内。  bool checkvalid**(**vec2 cellpos**)**  **{**  //如果单元格在棋盘内  **if** **((**cellpos**.**x **>=** 0**)** **&&** **(**cellpos**.**x **<** 10**)** **&&** **(**cellpos**.**y **>=** 0**)** **&&** **(**cellpos**.**y **<** 20**))**  **{**  int x **=** cellpos**.**x**;**  int y **=** cellpos**.**y**;**  //如果没有被填充，则返回true  **return** board**[**x**][**y**]** **==** **false;**  **}**  **else**  **return** **false;**  **}**  6. 判断第row行是否是满的。  扫描第row行，如果有一个格子没有被填充，则返回false。  如果没有扫描到没有被填充的格子，即都是被填充的，则返回true  // 检查棋盘格在row行有没有被填充满  bool checkfullrow**(**int row**)**  **{**  //遍历每一个x值  **for** **(**int j **=** 0**;** j **<** 10**;** j**++)** **{**  **if** **(!**board**[**j**][**row**])** **{**  **return** **false;**  **}**  **}**  **return** **true;**  **}**  7. 重新开始游戏，重新设置当前的游戏状态  void restart**()**  **{**  init**();**  **}**  8. 定时向下移动当前方块：  在moveDown函数中，在函数的最后继续定时调用moveDown函数，这样就能达到循环的作用了。  定时器为：void glutTimerFunc(unsigned int msecs, void (\*Func)(int value), int value);  //将方块向下移动一格  void moveDown**(**int value**)** **{**  //按下向下键，相当于强行向下移动一格  special**(**GLUT\_KEY\_DOWN**,**0**,**0**);**  //定时器，定时向下移动，1是向moveDown传递的参数，无实际意义，但是glutTimerFunc的回调函数必须需要一格int的参数  glutTimerFunc**(**10000 **/** speed**,** moveDown**,** 1**);**  **}**  9. 在主函数中需要调用响应的函数。  //显示回调函数  glutDisplayFunc**(**display**);**  glutReshapeFunc**(**reshape**);**  //特殊按键回调函数  glutSpecialFunc**(**special**);**  //普通按键回调函数  glutKeyboardFunc**(**keyboard**);**  //鼠标回调函数  glutMouseFunc**(**mouse**);**  //创建菜单并绑定响应事件  setupMenu**();**  //设置空闲回调函数  glutIdleFunc**(**idle**);**  //定时向下移动方块  moveDown**(**1**);** 实验结果   这里的分数的计算方式是：每次消去一行，分数加上当前速度的平方。   实验心得 本次实验完成了一个功能完整的俄罗斯方块游戏，功能分别包括：  生成棋盘、生成新的方块、移动方块、旋转方块、放置方块、消去方块、统计分数、改变速度等等。  值得注意的是，在移动方块和旋转方块的时候，需要先设置没有移动前的方块board为false，这样在check的时候才不会干扰能否移动或旋转。如果移动或者旋转，则需要保存没有移动时的方块的boardcolours，然后设置为黑色，在新的位置上重新标记board和填充颜色。  在消去方块的时候，需要从下往上判断，这样做的好处是，一旦发现需要消去，不需要重新扫描。如果发现消去，上面的格子会掉下一格，那么当前row指向的就是下一个格子了，所以此时row就不需要加1了。  在实验中遇到的还有一个问题就是，我之前在回到函数中不断调用printInformation函数来输出提示信息，发现这样会导致游戏的卡顿，因为我们没有必要再这里输出提示信息，只需要在可能改变游戏状态的时候才调用printInformation。  实验总体完成地很好。这包括：   * 实验要求的内容全部完成，还添加了一些自己的想法。比如我自己添加了用键盘、鼠标和菜单控制游戏的继续和停止，通过w和s键可以改变游戏的速度，极大地改善了游戏的体验。 * 在游戏体验方面也很不错，很流畅。 * 控制台输出提示很到位，让玩家一看就懂。 * 代码注释足够。基本上每一句都有自己的注释。 * 添加了实验目录，增加了实验报告的可读性，能够清楚得看出游戏思路。 |
|  |

深圳大学学生实验报告用纸

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字： 2018年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。