**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： 胡瑞珍**

**报告人： 学号： 班级：**

**实验时间： 2024年10月1日 -- 2024年10月28日**

**实验报告提交时间： 2024年10月27日**

**教务部制**

|  |
| --- |
| 实验目的与要求：   1. 强化OpenGL的基本绘制方法、键盘等交互事件的响应逻辑，实现更加复杂的绘制操作，完成一个简化版俄罗斯方块游戏。 2. 方块/棋盘格的渲染和方块向下移动。创建OpenGL绘制窗口，然后绘制网格线来完成对棋盘格的渲染。随机选择方块并赋上颜色，从窗口最上方中间开始往下自动移动，每次移动一个格子。初始的方块类型和方向也必须随机选择，另外可以通过键盘控制方块向下移动的速度，在方块移动到窗口底部的时候，新的方块出现并重复上述移动过程。 3. 方块叠加。不断下落的方块需要能够相互叠加在一起，即不同的方块之间不能相互碰撞和叠加。另外，所有方块移动不能超出窗口的边界。 4. 键盘控制方块的移动。通过方向键（上/下/左/右）来控制方块的移动。按“上”键使方块以旋转中心顺（逆）时针旋转，每次旋转90°，按“左”和“右”键分别将方块向左/右方向移动一格，按“下”键加速方块移动。 5. 游戏控制。当游戏窗口中的任意一行被方块占满，该行即被消除，所有上面的方块向下移动一格子。当整个窗口被占满而不能再出现新的方块时，游戏结束。通过按下“q”键结束游戏，和按下“r”键重新开始游戏。 6. 其他扩展。在以上基本内容的基础上，可以增加更多丰富游戏性的功能，如通过空格键使方块快速下落等。 |
| 实验过程及内容：   1. **绘制’J’、’Z’等形状的方块**     参考上图，绘制出以下不同形状且不同方向的方块。     1. **随机生成方块并赋上不同的颜色**         newtile 函数的主要功能是生成新的方块并在游戏中进行有效性检查，以确保在创建新方块时有足够的空间。如果没有足够空间，游戏将结束。  函数的第一步是将新方块的初始位置设置为棋盘的最顶部中央位置，即 (5, 18)。这一位置确保新方块总是从棋盘的中心开始，提供了统一的起点。  接下来，函数利用随机数生成器随机选择当前方块的类型、颜色和形状。首先，定义了随机数生成的范围，并通过 std::random\_device 获取硬件生成的种子。随后，使用 std::ranlux48 创建一个随机数引擎，并通过 std::uniform\_int\_distribution 设置均匀分布的随机数范围。随机数生成器生成一个方块类型（nowBlock）、颜色（nowBlockColor）和形状（nowBlockshape）。  然后，函数通过循环将生成的方块形状赋值给当前方块数组 tile，这使得新方块能够在后续的渲染中显示出来。为了确保随机生成方块不会超出网格范围，会先进行判断，若超过，则让初始位置下移一个单位。为了确保新方块不会与已经存在的方块重叠，函数调用 checkvalid 来检查每个顶点的位置。如果发现新方块的位置无效，游戏状态将被设置为结束，且输出提示信息告知玩家游戏已结束。  在确保新方块位置有效后，调用 updatetile 更新当前方块的状态。此后，函数为新方块分配颜色，并将颜色数据存储在一个数组中，以便在 OpenGL 渲染时使用。通过 glBindBuffer 和 glBufferSubData，函数将颜色数据传入 GPU，使新方块能够以随机选择的颜色渲染显示。最后，函数解绑缓冲区和顶点数组，确保状态恢复到默认。   1. **方块的自动向下移动**         以上代码中，方块的自动下落机制通过 autoMove 函数来实现。该函数首先获取当前的时间，并与记录的上次移动时间进行比较，以决定是否进行下落操作。具体来说，代码会检查自上次移动以来是否已经过了0.5秒（即500毫秒），这是控制方块下落速度的核心逻辑。如果时间条件满足且游戏状态不是“结束”或“暂停”，程序将执行方块的下落操作。  movetile 函数负责实际的方块移动。它接收一个方向参数，表示方块要移动的方向。通过循环计算新方块的位置，代码会将当前方块的位置和指定的移动方向相加，生成新位置数组 newtilepos。接着，程序会调用 checkvalid 函数检查新位置是否有效，确保方块不会移动到无效区域或与其他方块重叠。如果所有新位置都是有效的，方块将被成功移动至新位置，并通过 updatetile 函数更新方块的状态。  如果在尝试向下移动方块时 movetile 返回 false，这意味着方块已经到达棋盘的底部或与其他方块碰撞。在这种情况下，程序会调用 settile 来放置当前方块，并通过 newtile 函数生成一个新的方块以继续游戏。通过这种机制，实现了方块的自动下落，同时确保游戏逻辑正确地处理方块的移动与碰撞。  最后，在主函数中，每次循环都调用autoMove函数，以实现方块的自动下移。   1. **方块之间、方块与边界之间的碰撞检测**       以上代码实现了一个用于检测方块在棋盘上位置有效性的函数 checkvalid。  棋盘的状态由一个布尔数组 board 表示，其中 board[x][y] 为 true 表示对应的格子已被方块占用，由此跟踪哪些位置被填充，以便进行碰撞检测。  checkvalid 函数的核心作用是判断给定位置 cellpos 是否有效。它通过检查该位置的坐标是否在棋盘的边界内（即 0 <= x < board\_width 和 0 <= y < board\_height），确保不发生越界访问。此外，函数还检查该位置是否被其他方块填充，即对应的 board 值是否为 false。只有在这两个条件都满足时，函数才返回 true，表示该位置可以安全地放置新的方块。  如此，如果新方块的任意部分尝试移动到一个已被占用的格子或者超出棋盘边界，游戏将禁止该移动，避免非法状态的产生，从而实现了方块之间、方块与边界之间的碰撞检测。   1. **棋盘格中每一行填充满之后自动消除**           以上代码实现了在棋盘中检查并消除填满的行的功能。checkfullrow 函数逐行遍历棋盘，并判断每一行是否已被填满。填满的行是指该行中所有格子均被方块占用，即 board 数组中对应的值均为 true。如果某一行被标记为填满，函数会增加得分并清空该行的状态，同时将该行的颜色更新为黑色，以表示该行已被消除。  在消除某一行之后，所有位于该行上方的方块会向下移动一格。函数通过内层循环遍历上方的每一行，并将每个被占用的格子下移。这个过程还包括更新对应的颜色，以确保视觉上的一致性。通过调用 changecellcolour 函数，程序能够实时更新VBO（顶点缓冲区对象）中的颜色数据。  此外，函数还实现了连续消除行的奖励机制，记录连续消除的行数，并在消除时给予额外的分数奖励。这种设计增强了游戏的策略性和趣味性，鼓励玩家尝试消除多行。  最后，在主函数中，在每次循环结束前并且在游戏未结束和未暂停时，去检查是否有填满的行，是的话则进行行清空。   1. **游戏重置**     restart 函数用于重新启动游戏。当玩家选择重启时，函数首先将 gameover 状态设为 false，表示游戏仍在进行中。然后，它将分数重置为0，以便重新开始统计分数。接着，函数记录当前时间为最后一次方块移动和计算的时间，确保计时器在重启后能够正常运作。timerSeconds 被重置为0，以重新开始计时。函数最后通过输出提示信息告知玩家游戏已重启，并调用 init() 函数重新初始化游戏的设置和资源，准备好一个全新的游戏轮次。   1. **游戏控制**       key\_callback 函数用于处理键盘按键事件。当游戏未结束时，函数根据不同的按键执行相应的操作，例如方块的移动、旋转和生成新方块。具体而言，按下上箭头键会旋转方块，而按下下箭头键和空格键则会使方块向下移动，即加速方块下移；如果方块无法再向下移动，则会将其固定在当前位置并生成新的方块。左右箭头键则用于左右移动方块。  如果游戏结束，函数则提供退出和重启的选项，允许玩家通过按 ESC 或 Q 键退出游戏，按 R 键重启游戏。  此外，按下 S 键可以暂停和继续游戏，并在控制台输出相应的状态信息。   1. **计分器**           以上代码实现了一个用于游戏得分显示的功能，包括数字的绘制和得分的更新。  首先，drawDigit 函数负责绘制单个数字。函数接收一个数字、位置、顶点数组和线段数量等参数。它首先初始化一个布尔数组 segments，用于表示数字的七个段（上、下、左、右、中间和水平段）。通过 switch-case 语句，根据传入的数字确定哪些段被激活。  接下来，定义了每个段的相对坐标，函数会根据缩放因子调整这些坐标的大小。在绘制数字时，循环遍历 segments 数组，检查哪些段被激活，并在顶点数组中添加相应的线段顶点。这种处理方式使得数字的显示能够根据不同的数字自动调整，并且便于后续渲染。  updateScore 函数负责更新得分的显示。首先，它将得分限制在最后三位，并将这些位数提取到 digits 数组中。然后，设置得分的显示位置，在这里选择右上角的位置来进行绘制。通过调用 drawDigit，函数逐位绘制得分的每个数字，并调整位置以确保它们排列整齐。  最后，该函数将得分的顶点和颜色数据上传到 GPU 中，以便在渲染过程中使用。通过这种方式，玩家在游戏中可以实时看到得分的变化。  主函数中每次循环也调用updateScore 函数，以此来实现分数的实时更新。   1. **计时器**                 以上代码实现了一个游戏计时器的功能，包括时间的更新和显示。  updateTime 函数负责计算和更新当前时间，并将其格式化为分钟和秒数进行显示。首先，通过 glfwGetTime() 获取当前时间，判断自上次更新时间是否已经超过一秒。如果条件满足，计时器的秒数 timerSeconds 就会增加。接着，将时间转换为分钟和秒，提取出这两个数值的每一位，存储在 timer 数组中，以便后续绘制。  计时器的显示位置设置在左上角，通过调用 drawDigit 函数来绘制每一位数字。在绘制过程中，通过调整位置使得数字有适当的间隔，同时在分钟和秒之间增加额外的空间以区分它们。所有数字的颜色设置为白色，以便在游戏中清晰可见。  接下来，通过 glBindBuffer 和 glBufferSubData 将更新后的顶点数据和颜色数据上传到 GPU，以便渲染时使用。此时，timePoints 数组存储了时间数字的顶点，而 timeColor 数组则存储了对应的颜色信息。  此外，代码中还为计时器绘制了两个圆圈，表示分钟和秒之间的冒号。这是通过 drawCircle 函数实现的，确保了计时器的视觉效果更加生动。  最后，使用 setupVertexArray 来设置 VAO（顶点数组对象），将时间和得分的 VBO（顶点缓冲对象）与着色器关联，以便在渲染过程中能够正确显示得分和时间。  主函数中每次循环也调用updateTime 函数，以此来实现时间的实时更新。   1. **绘制暂停标志**         以上代码实现了游戏中的暂停标志，通过绘制一个圆形和一个三角形来表示暂停状态。首先定义了用于存储圆形和三角形顶点及其颜色的数组，然后使用 drawCircle 和 drawTriangle 函数在指定的位置绘制这些形状。绘制的圆形位于游戏界面的中心位置，颜色为灰色，而三角形则放置在圆形内，颜色为白色。接着，使用 setupVertexArray 函数将这些图形的顶点和颜色数据上传到GPU，以便在渲染时使用。   1. **绘制游戏失败标志**               以上代码实现了一个“LOSE”标志的绘制，包括绘制字母和背景矩形。  首先，drawChar 函数负责绘制单个字符，通过设置不同的段来表示字符的形状。字符 “L”、“O”、“S”和 “E” 是通过开关标志设置每个字符的显示段。然后，drawLoseFlag 函数利用 drawChar 函数逐个绘制字符串 "LOSE"，并在绘制过程中调整位置。  接着，通过 drawRectangle 函数绘制一个背景矩形，并在矩形上方绘制“LOSE”标志。最后，使用 setupVertexArray 将绘制的矩形和文字的顶点数据上传到GPU，准备进行渲染。   1. **渲染**       以上代码实现了游戏界面的渲染逻辑，通过逐一绘制游戏中的各个元素来构建完整的图形界面。首先，glClear(GL\_COLOR\_BUFFER\_BIT) 清空颜色缓冲区，确保新的渲染帧不受之前内容影响。接着，通过 glUniform1i 传递窗口的尺寸参数，确保着色器能够根据屏幕大小调整图像比例。  然后，代码依次绘制了棋盘格、当前方块和棋盘线条。计时器冒号、计分器、计时器、暂停标志和失败标志也被分别绑定和绘制。暂停标志和失败标志的显示受到游戏状态的控制，确保只有在暂停或游戏结束时显示相关元素。此外，glLineWidth 用于调整绘制线条的宽度，适应计分器和失败标志的线条显示需求。 |

深圳大学学生实验报告用纸

|  |
| --- |
| 实验结论：  游戏运行过程中的部分截图。          可以看到，游戏能正确运行，并且实现功能较多，趣味性较强。  实验心得：  通过本次实验，我不仅加深了对OpenGL基本绘制方法和交互事件处理的理解，还掌握了如何运用这些技术构建一个动态的游戏环境。在实现俄罗斯方块的过程中，我遇到了诸如方块旋转时边界检测、方块堆叠逻辑以及行消除算法等挑战，通过查阅资料和反复调试，最终克服了这些问题。这个项目让我认识到理论知识与实践应用之间的差距，同时也激发了我对计算机图形学的兴趣。此外，通过添加额外的功能，如计分器、游戏重置等等，我学会了如何根据用户需求灵活地扩展程序功能。这次经历对我来说是非常宝贵的学习机会。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  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。