****

**Web游戏-俄罗斯方块**

**姓 名：李翠琪**

**学 号：1751022**

**所在院系：软件学院**

**学科专业：软件工程**

**指导教师：金伟祖**

**二〇二〇年七月**

目录

[**一、** **整体架构** 2](#_Toc45908120)

[1.1 项目介绍 2](#_Toc45908121)

[1.2 项目架构 2](#_Toc45908122)

[**二、** **项目实现** 3](#_Toc45908123)

[2.1 index.html实现 3](#_Toc45908124)

[2.2 index.css实现 3](#_Toc45908125)

[2.3 const.js实现 4](#_Toc45908126)

[2.4 tetris.js实现 5](#_Toc45908127)

[2.5 view.js实现 6](#_Toc45908128)

[2.6 controller.js实现 8](#_Toc45908129)

[**三、** **项目展示** 15](#_Toc45908130)

1. **整体架构**

## 项目介绍

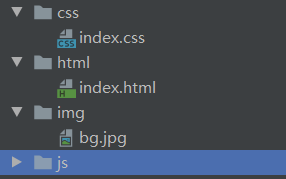
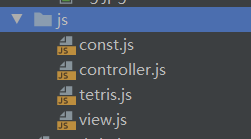
该项目是基于html+css+js开发的俄罗斯方块前端游戏。项目已部署到远程服务器,可访问:

<http://47.103.203.188:8080/tetris/html/index.html>

## 1.2 项目架构

本项目总体分为四个文件夹:html,css,js,img。img包含了背景图片。html中只有index.html，负责游戏的展示。css文件中只有index.css，负责对index.html的美化。游戏具体逻辑包含在js文件中。

js文件夹中，const.js定义了整个游戏的常量，例如游戏长宽，方块颜色，键盘的键与数值等。tetris.js代表着一种特点的方块类型,方块位置，也包含了方块变换等。controller.js执行了游戏的具体逻辑，如控制方块下落，方块变体，是否得分，游戏是否结束等等。view.js负责游戏区与下一方块区表格的动态生成以及游戏方块的绘制等等。

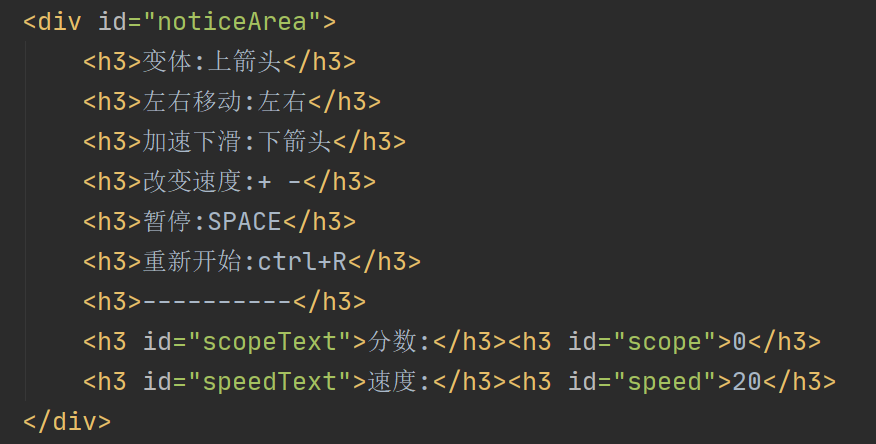
 

1. **项目实现**

## index.html实现

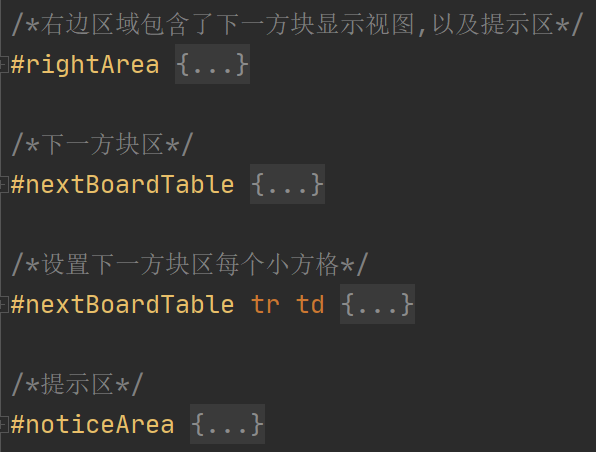
index.html中首先引入了相关的css文件以及js文件。主体通过div将页面分为三部分：左边为游戏区域,右边为辅助区域(右上角为下一方块预告区,右下角为操作说明区)。

在游戏区以及下一方块预告区，都只放了一个table标签，并且赋予一个id，之后通过index.css控制他的样式，通过view.js动态往表格添加行列，以可以根据传入值动态改变游戏区的长宽。预告区则是一些文本，其中的分数和速度则有id，可以在js中根据游戏进度动态改变。



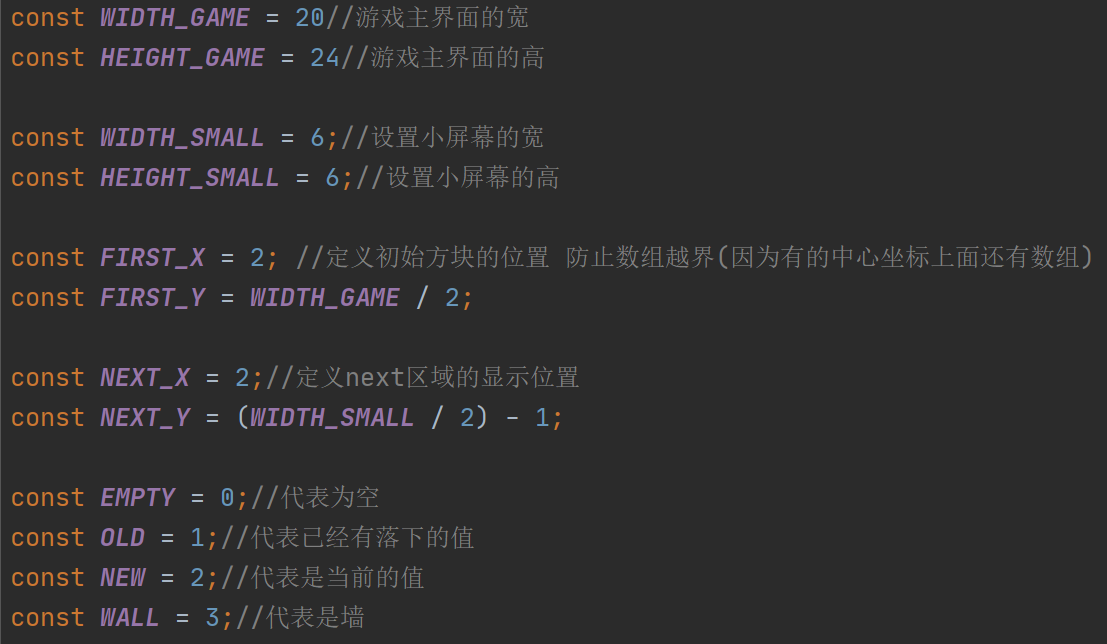
## index.css实现

主要设置了整体的背景，以及控制了游戏区，下一方块预告区，操作提示区的相对位置。以及游戏区，下一方块区中每个小方格的样式。



## const.js实现

该文件定义了整体程序需要用到的一些常量，一是为了便于修改，二是通过字符串代替数字，便于代码阅读。首先定义了游戏主界面的宽高以及下一方块提示区的宽高。其次定义了初始方块产生的位置。然后定义了不同种类的方块对应的字符串，例如WALL代表墙。还定义了使用到的键盘的键与值的对应关系。最后定义了使用到的颜色。

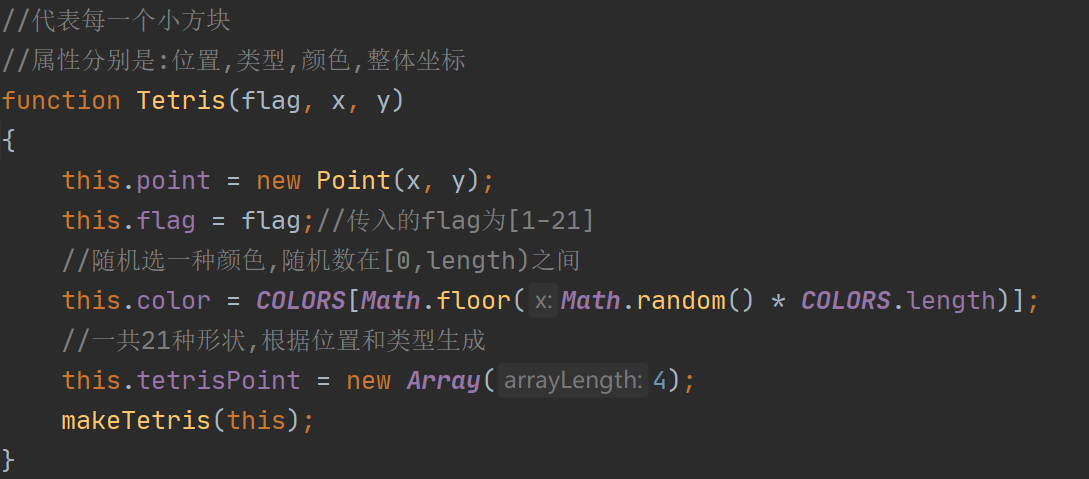
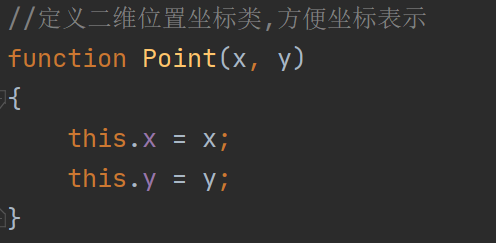




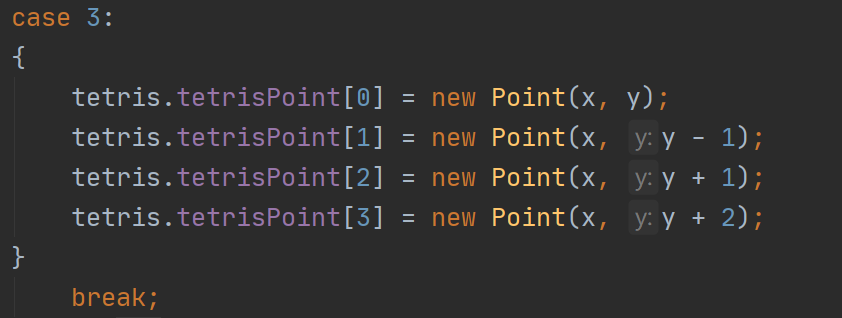
## tetris.js实现

定义二维位置坐标类,方便坐标表示。然后定义了方块类，包含了重要点坐标，方块类型，方块颜色，整体形状坐标。

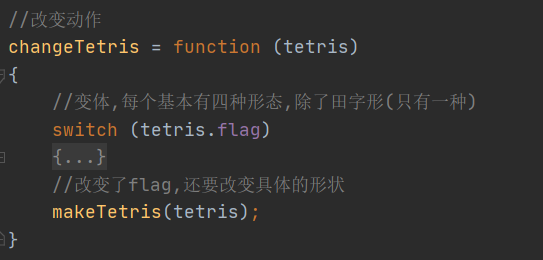
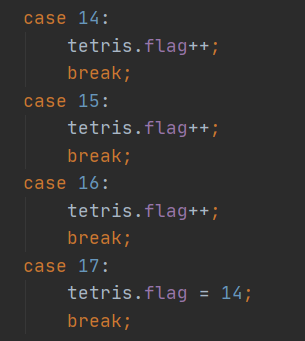
众所周知，一个整体的方块由四个原子方块组成。所以重要点坐标标记了该整体方块的一个点的位置，然后其他三个点则可以通过该重要点的坐标以及该方块类型得到（通过makeTetris（）函数生成）。每个整体方块的颜色是随机生成的。



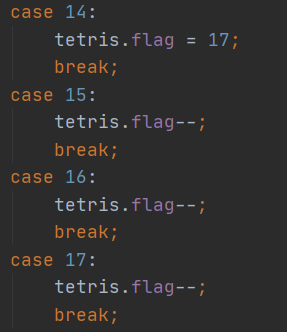
makeTetris（）函数则就是根据上述的重要点坐标（x,y）以及方块类型flag来生成。俄罗斯方块一共21种类型，对应每种类型生成他四个点的坐标并存储在数组中。



changeTetris（）函数是为了支持方块变形而存在的。首先改变flag到他的下一个转体的flag。因为除了田字形只有一种类型外，其他四个为一组，所以可以简单的操作即可得到转变后的flag。得到之后，调用之前的makeTetris（）即可得到转变后的整体方块的四个坐标。

rechangeTetris（）函数是changeTetris（）函数的逆运算，为的是模拟转体后发现空间不够则要转回原本的形状。实现起来差不多，只是形状改变的方向不一样。

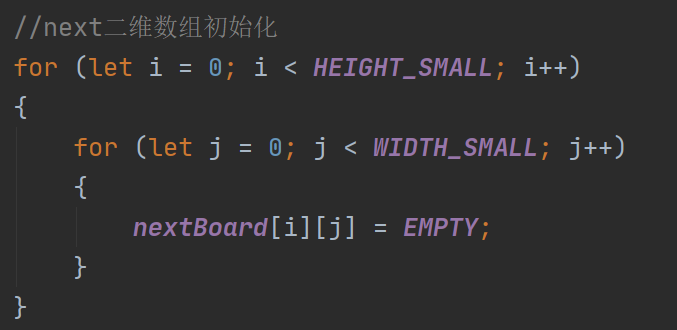


## view.js实现

主要是负责图像的绘制，即往表中根据const.js中定义的宽高，动态的生成表格的行和列，代表了每个小方块。还绘制了小方块的颜色。

首先，定义了游戏区，下一方块预告区（以下简称next区）的二维数组，之后基本通过操作二维数组来实现功能。

初始化函数init（）。负责将游戏区和next区的二维数组初始化。游戏区一部分初始化为EMPTY代表此时没有方块，边缘初始化为WALL，代表是墙。next区则全部初始化为EMPTY。



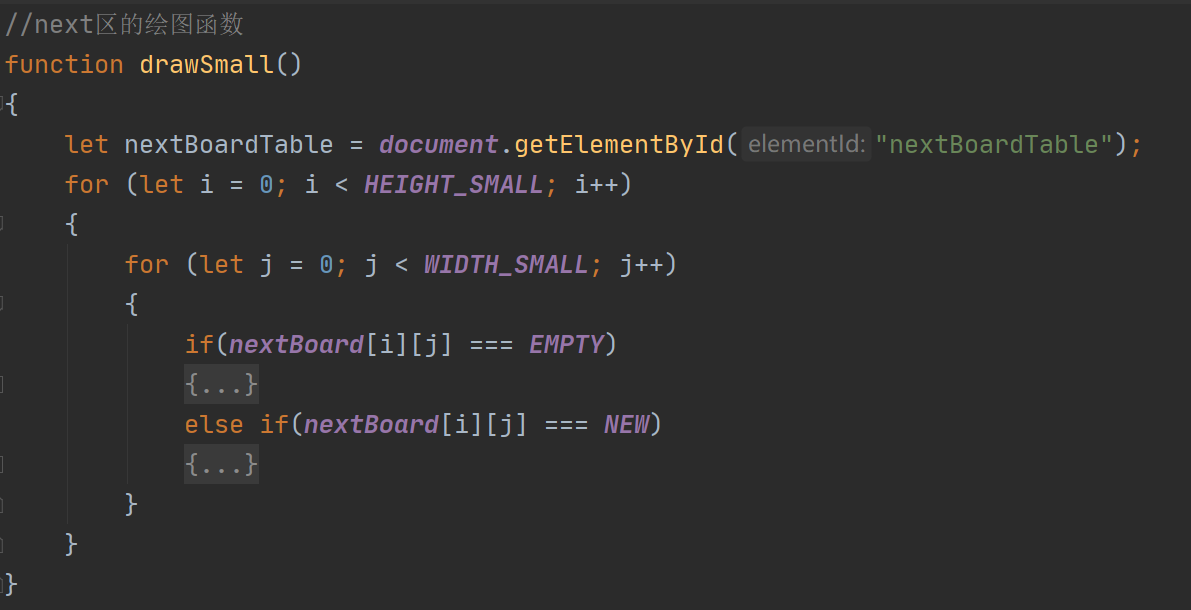
append（）函数。负责在屏幕上动态添加表格，即根据const定义的宽高往游戏区，next区动态添加对应的表格。



draw（）函数。负责绘制游戏区的方块颜色，根据不同种类的方块（空白，墙，已经落下的，正在落下的），绘制不同的颜色。

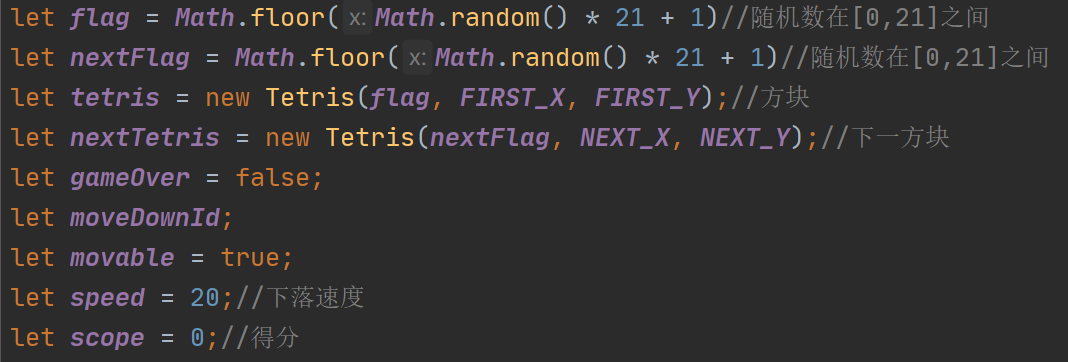


drawSmall（）函数。负责next区的绘图。和draw（）原理一样。

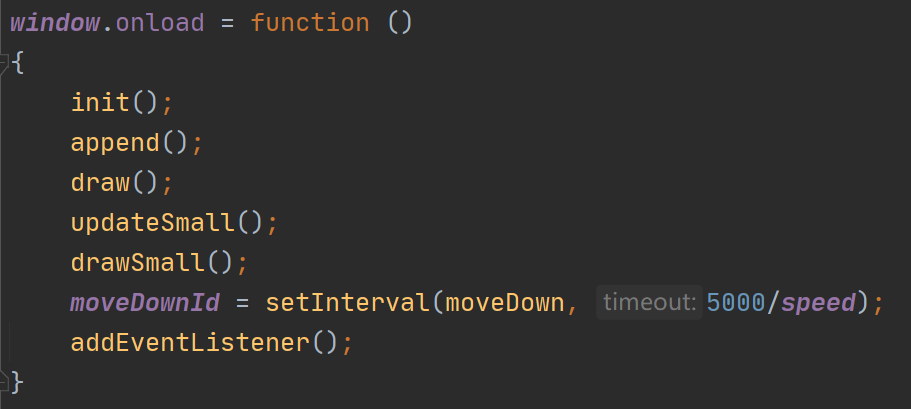


## controller.js实现

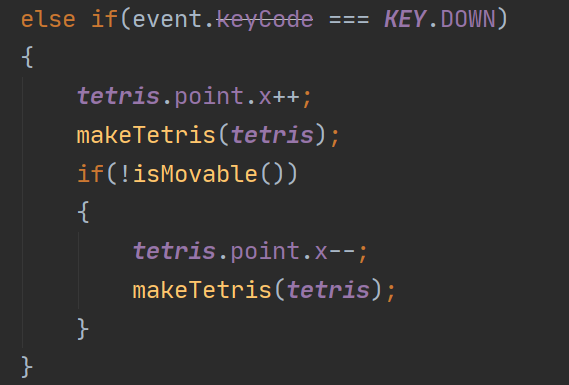
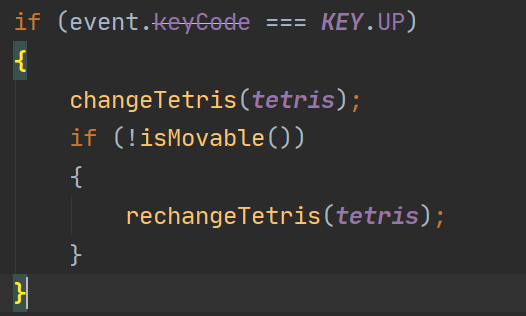
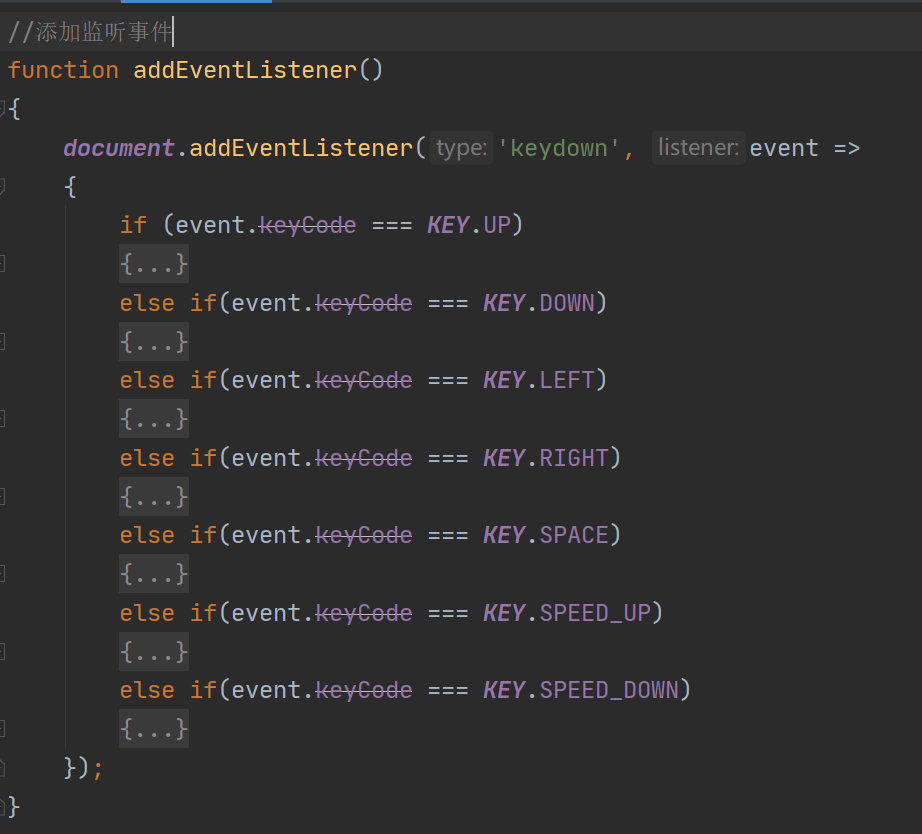
该js文件负责游戏的具体逻辑实现。首先定义了一些全局变量。例如当前方块类型flag，下一方块类型nextFlag。当前方块tetris，下一方块nextTetris。以及两个变量movable，gameOver表示是否能移动，是否游戏结束。以及两个变量speed，scope代表速度和得分。最后moveDownId是控制方块不断下移函数moveDown的定时器的返回值，为了能停止定时器，调整速度等等。



window.onload（）函数。是该js文件的入口，当html页面一加载就进入该函数。他调用了之后用到的函数，大概逻辑是：先初始化游戏区，next区的二维数组，然后将初始的表格给动态添加好。随后就绘制游戏区和next区的方块。随后就调用定时器，每隔一段时间调用moveDown函数，让当前方块下移。最后还添加了事件监听。



addEventListener（）函数。添加键盘事件监听。根据不同的键盘按键，采取对应的措施。如果是UP，则方块先变形，如果不能变形，则回退。如果是DOWN，则让方块向下移动一位，且调用makeTetris()函数，让核心小方块之外的其他三个小方块坐标也发生改变，同样的，如果是不能向下移动，则取消刚刚的操作。LEFT，RIGHT函数以此类推。SPACE，则简单的将moveable=！moveable就实现了暂停。而+-号则改变速度，首先先把speed变量做相应的改变然后先通过之前提到的定时器ID：moveDownId来停止movedown函数的定时器，再新建一个定时器（因为不这样的话，定时器的速度会按照之前的速度，即不会改变）。同时，获取html上的对应speed元素，并且修改值。





moveDown()函数：之前所提到的，就是通过定时器调用moveDown()函数实现方块的自动下落，是一个比较关键的函数。

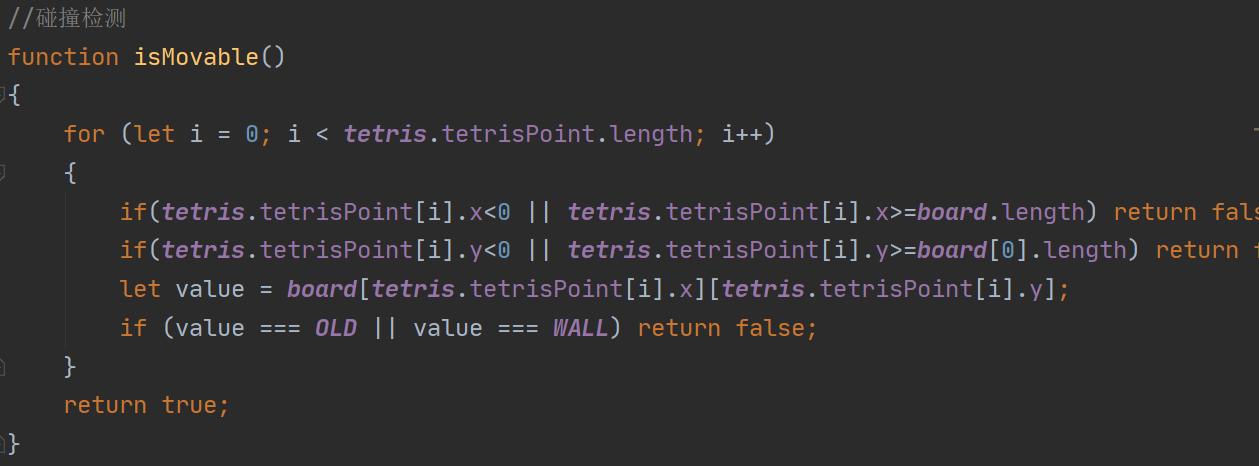
该函数首先会判断，如果没有暂停，则将tetris的核心小方块x坐标下移，并且将其他三个小方块也下移。随后判断是否可移动，如果是不可移动的，说明该方块已经落到了下面，则调用newStart（）函数生成新的方块继续。如果是可以移动的，则调用updateBoard()函数更新方块位置，并且通过draw（）函数绘制当前方块情况。



updateBoard()函数：负责每次方块下落时将游戏区的二维数组更新，具体逻辑是先将数组中值等于NEW，即刚刚落下的给赋值为EMPTY。再将当前tetris的四个小方块位置坐标标记为NEW。



isMovable()函数：负责碰撞检测。具体做法是，对方块中的每一个小方块，首先判断是否数组越界，然后判断坐标所在位置是否是OLD或者WALL，是则返回false，代表不能移动。（因为碰撞检测是在updateBoard()之前，所以还不会覆盖）。

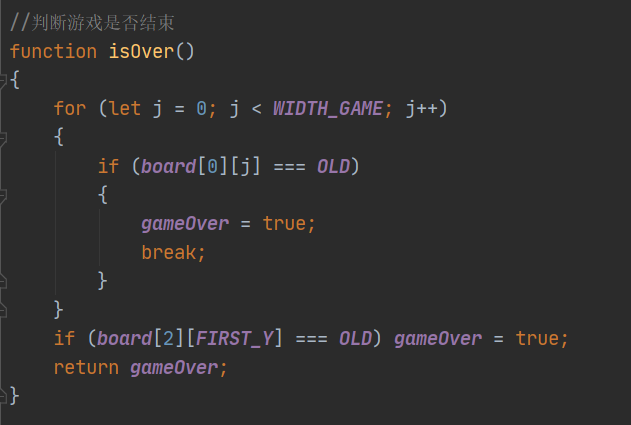


newStart()函数：当一个方块下落到不能继续下落时调用该方法。首先将方块整体上移一格，以抵消之前的下移操作。随后更新方块，将此方块变成已下落状态。并且此时还调用clearLine()进行满行清空操作。最后还判断游戏是否结束，结束则调用newGame（）函数进行相关清理工作，未结束则把之前的nextTetris变成当前的tetris，并且重新生成下一方块并展示。



newGame()函数：初始化数组，重新绘图，并且终止定时器。

isOver()函数：用于判断游戏是否结束，具体是查看游戏区的顶部是否已经有了已经落下的方块，或者是应该落下新方块的位置是否已经有了方块。



clearLine()函数则负责消除满行。具体做法是，从游戏区的底部往上进行扫描操作，并且计算每一行的方块数，如果某一行满了，则让该行清空。并且让这一行上面的所有方块都不断下移，把空出来的位置填满，并且又从底部重新开始扫描（因为可能消去之后下面的区域又全满了）。最后更新分数，并且展示在屏幕上。

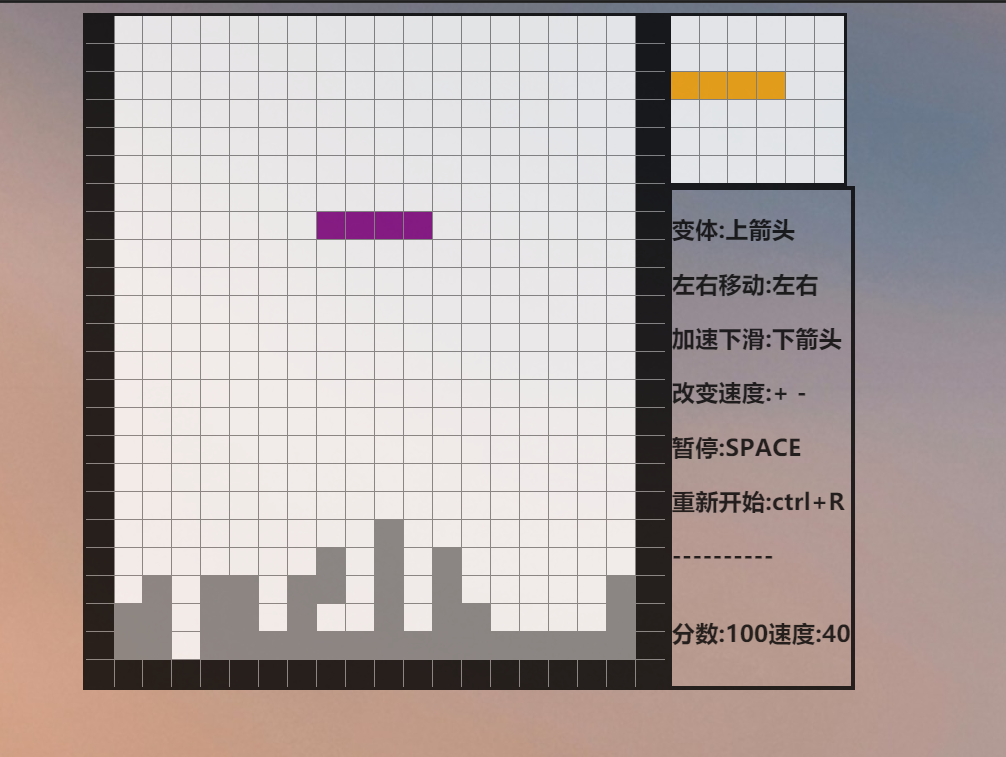


1. **项目展示**

初始页面：

****

消除一行并且改变了速度的页面：

****