TTS 7.0 COOKBOOK

（JAVA TETRIS UNIT 01）

版本编号 7.0

2013-12

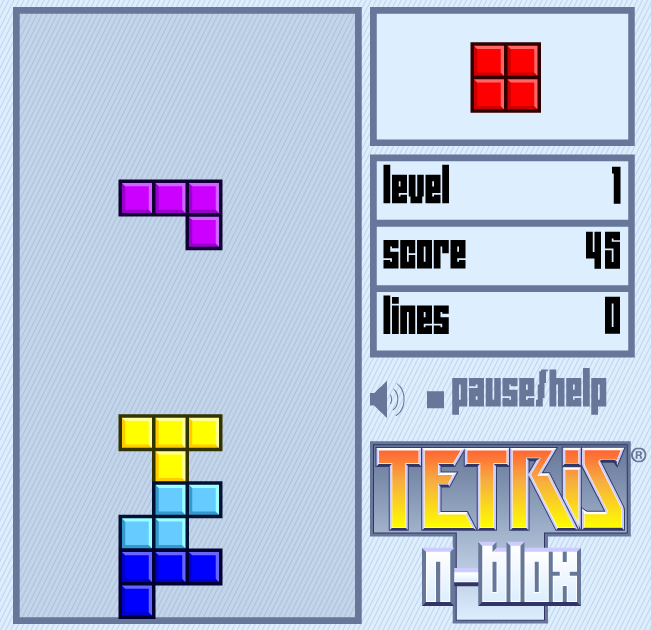
达内IT培训集团

JAVA TETRIS Unit01

1. Tetris项目

* 问题

Tetris俄罗斯方块（Tetris, 俄文：Тетрис）是一款风靡全球的电视[游戏机](http://baike.baidu.com/view/15830.htm)和掌上游戏机游戏，它由俄罗斯人阿列克谢·帕基特诺夫发明，故得此名。俄罗斯方块的基本规则是移动、旋转和摆放游戏自动输出的各种方块，使之排列成完整的一行或多行并且消除得分。由于上手简单、老少皆宜，从而家喻户晓，风靡世界。其界面效果如图－1所示：

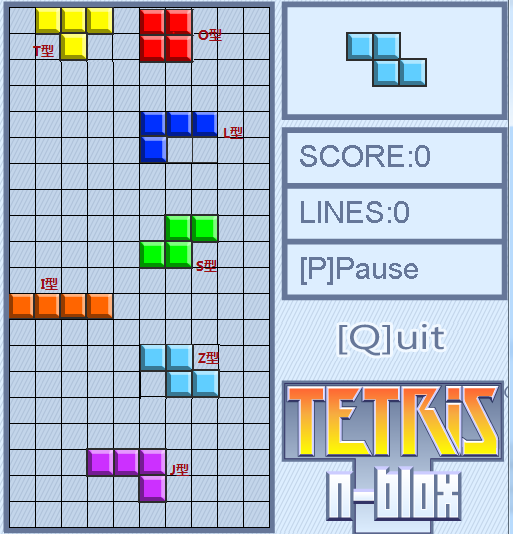


图－1

Tetris游戏的基本规则为：

1 一个用于摆放正方形格子的平面虚拟场地（Wall），其标准大小：行宽为10，列高为20，以格子为单位计算，宽为10个格子，高为20个格子；

2 一组由4个小型正方形组成的规则图形（Tetromino），共有7种，分别以S、Z、L、J、I、O、T这7个字母的形状来命名，如图－2所示：



图－2

Tetris游戏的逻辑为：

1. 玩家操作有：旋转方块；以格子为单位左右移动方块；让方块加速落下。

2. 方块移到区域最下方或是着地到其他方块上无法移动时，就会固定在该处，而新的方块出现在区域上方开始落下。

3. 当区域中某一行横向格子全部由方块填满，则该列会消失并成为玩家的得分。同时删除的行数越多，得分指数上升。

4. 当固定的方块堆到区域最上方而无法消除层数时，则游戏结束。

5. 一般来说，游戏还会提示下一个要落下的方块，熟练的玩家会考虑到下一个方块，评估要如何进行。由于游戏能不断进行下去，对商用游戏不太理想，所以一般还会随着游戏的进行而加速提高难度。

6. 预先设置的随机发生器不断地输出单个方块到场地顶部。

* 方案

软件的开发过程如下：

1) 需求(软件功能的文字描述)

2) 需求分析(找对象)

3) 概要设计

3.1) 数学模型

3.2) 类的设计

4) 详细功能的设计

4.1) 数据初始化

4.2) 界面绘制

4.3) 左右移动功能设计

4.4) 下落功能设计

Tetris项目开发过程如下：

1. 业务需求分析

本案例中的问题部分就是我们的需求。

2. 业务分析

根据图-2进行分析，找出有哪些业务对象。可以看出窗口上有如下内容：

窗口

|-- tetris(俄罗斯方块)

|-- score 累计分数

|-- lines 销毁的行数

|-- wall(墙 20行X10列)

| |-- 20行X10列的Cell

|-- tetromino 正在下落的（4格方块，有7种形态）

| |-- 4个Cell

|-- nextOne 下一个准备下落的方块

|-- 4个Cell

3. 类设计

要想创建对象，就要有类。找出对象中的数据，根据对象定义类。本项目中，Cell类（格子）、Tetromino类（四格方块）、Tetris类（俄罗斯方块）这几个类的数据情况如下：

Cell类

|-- int row

|-- int col

|-- image 贴图

Tetromino类

|--Cell[] cells

|-- Cell \* 4

Tetris 类

|-- Cell[][] wall = 20\*10

|-- Tetromino tetromino 正在下落的方块

|-- Tetromino nextOne 下一个方块

|-- int lines

|-- int score

本项目中类的属性和方法，以及类之间的如图-3所示：



图-3

在后续的步骤中，详细介绍了这些类中的方法的实现，请参考步骤。

4.编码

编码就是我们所说的coding的过程，下面的步骤，就是本项目的全部编码过程。

* 步骤

实现此案例需要按照如下步骤进行。

步骤一：新建工程和包

新建名为TETRIS的工程，然后，在工程下的src目录下新建包com.tarena.tetris，工程结构如图-4所示：

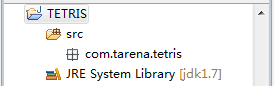


图- 4

在项目中，包的名字一般为公司域名倒过来，再加上项目名称，即为包名。如图-4中中的包名为com.tarena.tetris，其中，com.tarena是达内公司的域名倒过来，tetris为本项目的名称。

步骤二：新建Cell类

在com.tarena.tetris包下新建名为Cell的类，代码如下所示：

package com.tarena.tetris;

import java.awt.image.BufferedImage;

/\*\*

\* 格子

\*/

public class Cell {

private int row;//格子的行

private int col;//格子的列

private BufferedImage image;//格子的贴图

public Cell(int row, int col, BufferedImage image) {

super();

this.row = row;

this.col = col;

this.image = image;

}

public int getRow() {

return row;

}

public void setRow(int row) {

this.row = row;

}

public int getCol() {

return col;

}

public void setCol(int col) {

this.col = col;

}

public BufferedImage getImage() {

return image;

}

public void setImage(BufferedImage image) {

this.image = image;

}

public void drop(){

row++;

}

public void moveRight(){

col++;

}

public void moveLeft(){

col--;

}

@Override//重写

public String toString() {

return row+","+col;

}

}

步骤三：新建 Tetromino,表示四格方块

在com.tarena.tetris包下新建名为Tetromino的类，用于表示四个格子的方块，代码如下所示：

package com.tarena.tetris;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

}

上述代码中的cells属性，来存放四个格子，即一个方块。

步骤四：新建Tetris类，表示俄罗斯方块

在com.tarena.tetris包下新建名为Tetris的类，用于构建俄罗斯方块项目的整体流程，代码如下所示：

package com.tarena.tetris;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

public static void main(String[] args) {

}

}

Tetris类介绍：

Tetris

|-- Cell[][] wall = 20行\*10列 表示墙

|-- Tetromino tetromino 表示正在下落的方块

|-- Tetromino nextOne 表示下一个方块

|-- int lines 表示销毁的行数

|-- int score 表示游戏得分

步骤五：构建Tetromino类的结构

构建Tetromino类，在该类中添加方块下落方法softDrop，左移方法moveLeft，右移方法moveRight以及toString方法。另外，Tetromino类的同一文件中，新建类T、I、S、Z、O、L、J，他们都为Tetromino类的子类，用于表示各种形状的方块。代码如下所示：

package com.tarena.tetris;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

return null;

}

public void softDrop() {

}

public void moveLeft() {

}

public void moveRight() {

}

public String toString() {

return null;

}

}

class T extends Tetromino {

}

class I extends Tetromino {

}

class S extends Tetromino {

}

class Z extends Tetromino {

}

class O extends Tetromino {

}

class L extends Tetromino {

}

class J extends Tetromino {

}

步骤六：重构Tetromino类，实现上一步定义的方法

重构Tetromino类，实现方块下落方法softDrop，左移方法moveLeft，右移方法moveRight、随机获取7种形状中的一种的randomOne方法以及toString方法，代码如下所示：

package com.tarena.tetris;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

Random random = new Random();

int type = random.nextInt(7);

switch (type) {

case 0:

return new T();

case 1:

return new I();

case 2:

return new S();

case 3:

return new J();

case 4:

return new L();

case 5:

return new Z();

case 6:

return new O();

}

return null;

}

/\*\*

\* 方块下落一个格子

\*/

public void softDrop() {

for (int i = 0; i < cells.length; i++) {

cells[i].drop();

}

}

/\*\*

\* 方块左移一个格子

\*/

public void moveLeft() {

for (int i = 0; i < cells.length; i++) {

cells[i].moveLeft();

}

}

/\*\*

\* 方块右移一个格子

\*/

public void moveRight() {

for (int i = 0; i < cells.length; i++) {

cells[i].moveRight();

}

}

/\*\*

\* 覆盖toString方法，显示方块中每个格子的行列信息

\*/

public String toString() {

return Arrays.*toString*(cells);

}

}

/\*\*

\* T形方块

\*/

class T extends Tetromino {

}

/\*\*

\* S形方块

\*/

class I extends Tetromino {

}

/\*\*

\* T形方块

\*/

class S extends Tetromino {

}

/\*\*

\* Z形方块

\*/

class Z extends Tetromino {

}

/\*\*

\* O形方块

\*/

class O extends Tetromino {

}

/\*\*

\* L形方块

\*/

class L extends Tetromino {

}

/\*\*

\* J形方块

\*/

class J extends Tetromino {

}

步骤七：重构Tetromino类，为各个子类添加构造方法

重构Tetromino类，为各个子类添加构造方法，代码如下所示：

package com.tarena.tetris;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

... ...

}

/\*\*

\* 方块下落一个格子

\*/

public void softDrop() {

... ...

}

/\*\*

\* 方块左移一个格子

\*/

public void moveLeft() {

... ...

}

/\*\*

\* 方块右移一个格子

\*/

public void moveRight() {

... ...

}

/\*\*

\* 覆盖toString方法，显示方块中每个格子的行列信息

\*/

public String toString() {

return Arrays.*toString*(cells);

}

}

/\*\*

\* T形方块

\*/

class T extends Tetromino {

public T() {

cells[0] = new Cell(0, 4, null);

cells[1] = new Cell(0, 3, null);

cells[2] = new Cell(0, 5, null);

cells[3] = new Cell(1, 4, null);

}

}

/\*\*

\* I形方块

\*/

class I extends Tetromino {

public I() {

cells[0] = new Cell(0, 4, null);

cells[1] = new Cell(0, 3, null);

cells[2] = new Cell(0, 5, null);

cells[3] = new Cell(0, 6, null);

}

}

/\*\*

\* S形方块

\*/

class S extends Tetromino {

public S() {

cells[0] = new Cell(1, 4, null);

cells[1] = new Cell(1, 3, null);

cells[2] = new Cell(0, 4, null);

cells[3] = new Cell(0, 5, null);

states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),

new State(0, 0, -1, 0, 0, 1, 1, 1) };

}

}

/\*\*

\* Z形方块

\*/

class Z extends Tetromino {

public Z() {

cells[0] = new Cell(1, 4, null);

cells[1] = new Cell(0, 3, null);

cells[2] = new Cell(0, 4, null);

cells[3] = new Cell(1, 5, null);

}

}

/\*\*

\* O形方块

\*/

class O extends Tetromino {

public O() {

cells[0] = new Cell(0, 4, null);

cells[1] = new Cell(0, 5, null);

cells[2] = new Cell(1, 4, null);

cells[3] = new Cell(1, 5, null);

}

}

/\*\*

\* L形方块

\*/

class L extends Tetromino {

public L() {

cells[0] = new Cell(0, 4, null);

cells[1] = new Cell(0, 3, null);

cells[2] = new Cell(0, 5, null);

cells[3] = new Cell(1, 3, null);

}

}

/\*\*

\* J形方块

\*/

class J extends Tetromino {

public J() {

cells[0] = new Cell(0, 4, null);

cells[1] = new Cell(0, 3, null);

cells[2] = new Cell(0, 5, null);

cells[3] = new Cell(1, 5, null);

}

}

以上代码对各种形状进行初始化，初始化时，cell数组的第一个元素为后期旋转时的旋转轴位置。上述代码中，S、Z两种形状，为了旋转时美观的考虑，cell[0]的坐标为（1，4），其余形状的cell[0]坐标都为（0，4），在此请注意区别。

步骤八：重构Tetris类，构建界面

首先， 将Tetris类继承自JPanel类； 然后，在Tetris类的main方法中，使用 JFrame创建框体，并将Tetris类所在的面板放置到窗体上，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

public static void main(String[] args) {

JFrame frame = new JFrame();

// 在加载Tetris类的时候, 会执行静态代码块

// 静态代码块,装载了图片素材, 为图片对象

Tetris tetris = new Tetris();

//将面板的颜色设置为蓝色，用于测试

tetris.setBackground(new Color(0x0000ff));

frame.add(tetris);

frame.setSize(530, 580);

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);//"尽快"显示框口

}

}

上述代码中，设置了面板的背景色，以便运行后看到面板放到窗体上的效果。

运行上述代码，显示界面如图-5所示：

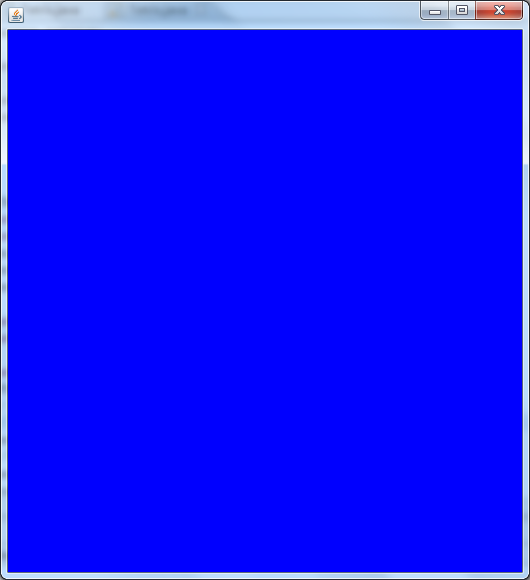


图- 5

步骤九：重构Tetris类和Tetromino类，加载图片

首先，在Tetris类中的static块中，使用ImageIO类的read方法加载图片，在此，请注意将图片文件放置在com.tarena.tetris包下。Tetris类代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

try {

// Tetris.class的同一个包中找 "tetris.png"

*background* = ImageIO.*read*(Tetris.class.getResource("tetris.png"));

*T* = ImageIO.*read*(Tetris.class.getResource("T.png"));

*I* = ImageIO.*read*(Tetris.class.getResource("I.png"));

*S* = ImageIO.*read*(Tetris.class.getResource("S.png"));

*Z* = ImageIO.*read*(Tetris.class.getResource("Z.png"));

*J* = ImageIO.*read*(Tetris.class.getResource("J.png"));

*L* = ImageIO.*read*(Tetris.class.getResource("L.png"));

*O* = ImageIO.*read*(Tetris.class.getResource("O.png"));

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

... ...

}

}

然后，更改各种形状中方块的图片，Tetromino类代码如下所示：

package com.tarena.tetris;

package com.tarena.tetris;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

... ...

}

/\*\*

\* 方块下落一个格子

\*/

public void softDrop() {

... ...

}

/\*\*

\* 方块左移一个格子

\*/

public void moveLeft() {

... ...

}

/\*\*

\* 方块右移一个格子

\*/

public void moveRight() {

... ...

}

/\*\*

\* 覆盖toString方法，显示方块中每个格子的行列信息

\*/

public String toString() {

return Arrays.toString(cells);

}

}

/\*\*

\* T形方块

\*/

class T extends Tetromino {

public T() {

cells[0] = new Cell(0, 4, Tetris.T);

cells[1] = new Cell(0, 3, Tetris.T);

cells[2] = new Cell(0, 5, Tetris.T);

cells[3] = new Cell(1, 4, Tetris.T);

}

}

/\*\*

\* I形方块

\*/

class I extends Tetromino {

public I() {

cells[0] = new Cell(0, 4, Tetris.*I*);

cells[1] = new Cell(0, 3, Tetris.*I*);

cells[2] = new Cell(0, 5, Tetris.*I*);

cells[3] = new Cell(0, 6, Tetris.*I*);

}

}

/\*\*

\* S形方块

\*/

class S extends Tetromino {

public S() {

cells[0] = new Cell(1, 4, Tetris.S);

cells[1] = new Cell(1, 3, Tetris.S);

cells[2] = new Cell(0, 4, Tetris.S);

cells[3] = new Cell(0, 5, Tetris.S);

}

}

/\*\*

\* Z形方块

\*/

class Z extends Tetromino {

public Z() {

cells[0] = new Cell(1, 4, Tetris.*Z*);

cells[1] = new Cell(0, 3, Tetris.*Z*);

cells[2] = new Cell(0, 4, Tetris.*Z*);

cells[3] = new Cell(1, 5, Tetris.*Z*);

}

}

/\*\*

\* O形方块

\*/

class O extends Tetromino {

public O() {

cells[0] = new Cell(0, 4, Tetris.*O*);

cells[1] = new Cell(0, 5, Tetris.*O*);

cells[2] = new Cell(1, 4, Tetris.*O*);

cells[3] = new Cell(1, 5, Tetris.*O*);

}

}

/\*\*

\* L形方块

\*/

class L extends Tetromino {

public L() {

cells[0] = new Cell(0, 4, Tetris.*L*);

cells[1] = new Cell(0, 3, Tetris.*L*);

cells[2] = new Cell(0, 5, Tetris.*L*);

cells[3] = new Cell(1, 3, Tetris.*L*);

}

}

/\*\*

\* J形方块

\*/

class J extends Tetromino {

public J() {

cells[0] = new Cell(0, 4, Tetris.*J*);

cells[1] = new Cell(0, 3, Tetris.*J*);

cells[2] = new Cell(0, 5, Tetris.*J*);

cells[3] = new Cell(1, 5, Tetris.*J*);

}

}

步骤十：测试加载图片是否成功

首先，在Tetris类中，添加画墙的方法paintWall；

然后，在该类中，重写JPanel类的paint方法，在该方法中，加载背景图片、坐标系平移15个像素 、调用paintWall方法；

第三，在Tetris类中添加action方法，在该方法中，初始化数组wall，将墙的第二行第二列设置为T行方块中的一个格子（2，2），也就是说墙的第二行第二列有东西了，不再是空白的；

最后，重构Tetris类的main方法，添加对action方法的调用，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

// 画背景, 画墙, 画正在下落的方块 画下一个方块...

g.drawImage(*background*, 0, 0, null);

g.translate(15, 15);// 坐标系平移

paintWall(g);// 画墙

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[ROWS][COLS];

wall[2][2] = new Cell(2,2, T);

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

for (int row = 0; row < wall.length; row++) {

Cell[] line = wall[row];

// line 代表墙上的每一行

for (int col = 0; col < line.length; col++) {

Cell cell = line[col];

// cell 代表墙上的每个格子

int x = col \* *CELL\_SIZE*;

int y = row \* *CELL\_SIZE*;

if (cell == null) {

g.drawRect(x,y,*CELL\_SIZE*,*CELL\_SIZE*);

} else {

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

//g.drawString(row+","+col, x,y+CELL\_SIZE);

}

}

}

public static void main(String[] args) {

JFrame frame = new JFrame();

// 在加载Tetris类的时候, 会执行静态代码块

// 静态代码块,装载了图片素材, 为图片对象

Tetris tetris = new Tetris();

//将面板的颜色设置为蓝色，用于测试

tetris.setBackground(new Color(0x0000ff));

frame.add(tetris);

frame.setSize(530, 580);

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

frame.setVisible(true);// 在显示窗口时候,会"尽快"的调用paint()方法绘制界面

tetris.action();

}

}

上述代码中，在paintWall方法中，通过循环判断墙上的每一个区域是否有格子，如果Cell类的对象cell为null，则说明没有格子，如果不为null，则说明有格子。如果有格子，获取格子图片贴图到墙的相应位置。另外，在绘制图片时，出现x-1,y-1这样的代码，是为了不使两个格子黑色边框重叠，否则，美观效果不好。

运行上述代码，界面显示效果如图-6所示：

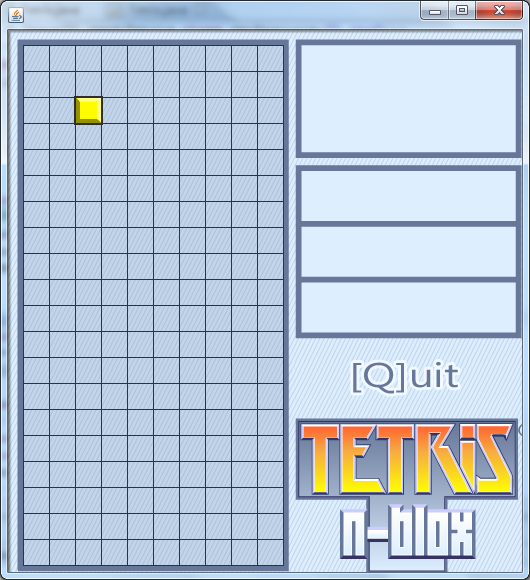


图- 6

在图-6中，墙的第二行第二列显示了一个黄色格子。并且背景色图片也加载成功了。

步骤十一：绘制正在下落的方块

重构Tetris类，在类中添加方法paintTetromino， 该方法实现绘制正在下落的方块。首先，将row，col坐标转换drawImage方法需要的x，y坐标，然后进行贴图，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

... ...

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

if (tetromino == null) {

return;

}

// 将每个格子的row,col 换算为x,y 然后贴图

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

// i = 0 1 2 3

Cell cell = cells[i];

// cell 每个格子

int x = cell.getCol() \* *CELL\_SIZE*;

int y = cell.getRow() \* *CELL\_SIZE*;

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

public static void main(String[] args) {

... ...

}

}

步骤十二：将正在下落的方块显示在界面上

重构Tetris类中的action方法，初始化tetromino属性，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

wall[2][2] = new Cell(2,2, T);

tetromino= Tetromino.*randomOne*();

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，界面效果如图-7所示：

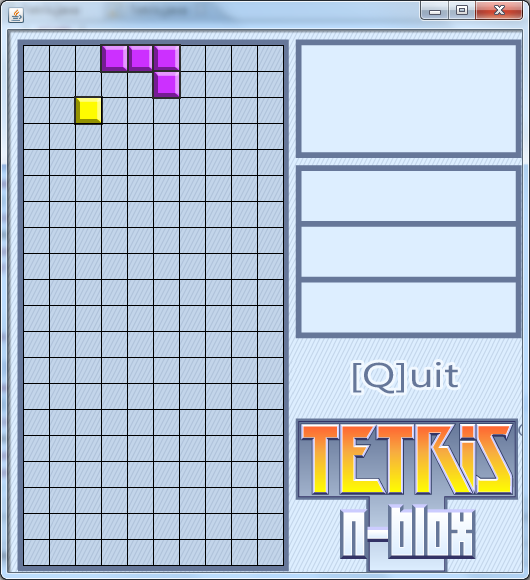


图- 7

图-7中显示了J形方块，也可能是其它6种形状，因为，我们是通过随机的方式获取的方块。

步骤十三：绘制下一个要下落的方块

绘制下一个要下落的方块的过程，与绘制正在下落的方块的过程相同，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

// 画背景, 画墙, 画正在下落的方块 画下一个方块...

g.drawImage(*background*, 0, 0, null);

g.translate(15, 15);// 坐标系平移

paintWall(g);// 画墙

paintTetromino(g);//绘制正在下落的方块

paintNextOne(g);

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

wall[2][2] = new Cell(2,2, T);

tetromino= Tetromino.*randomOne*();

nextOne=Tetromino.*randomOne*();

}

/\*\*

\* 绘制下一个要下落的方块

\* @param g

\*/

private void paintNextOne(Graphics g) {

if (nextOne == null) {

return;

}

// 将每个格子的row,col 换算为x,y 然后贴图

Cell[] cells = nextOne.cells;

for (int i = 0; i < cells.length; i++) {

// i = 0 1 2 3

Cell cell = cells[i];

// cell 每个格子

int x = (cell.getCol() + 10) \* *CELL\_SIZE*;

int y = (cell.getRow() + 1) \* *CELL\_SIZE*;

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，界面显示效果如图-8所示：

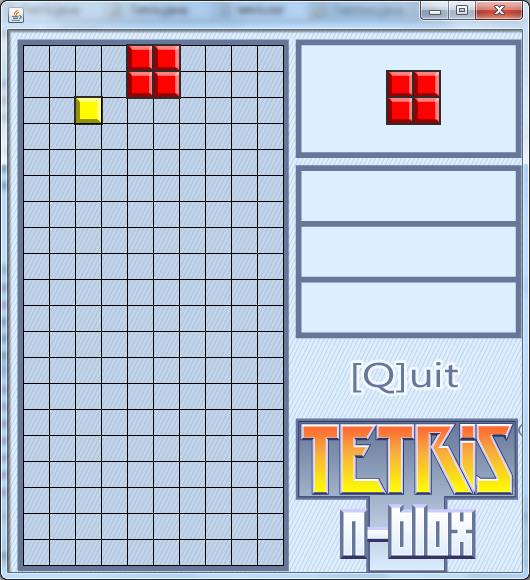


图- 8

在图-8中，右侧区域显示了O行方块，即为下一次要出现在墙的方块。

步骤十四：处理键盘按下事件

重构Tetris类的action方法，添加处理键盘事件的方法。在按下键盘的向下键时执行下落方法、左方向键时执行左移方法、右方向键时执行右移方法，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

//wall[2][2] = new Cell(2,2, T);

tetromino= Tetromino.*randomOne*();

nextOne=Tetromino.*randomOne*();

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();// [c]

switch (key) {

case KeyEvent.*VK\_DOWN*:

tetromino.softDrop();

break;

case KeyEvent.*VK\_RIGHT*:

tetromino.moveRight();

break;

case KeyEvent.*VK\_LEFT*:

tetromino.moveLeft();

break;

}

repaint();// 再画一次!

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

}

/\*\*

\* 绘制下一个要下落的方块

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，显示界面后，按下键盘上的向下键、左方向键和右方向键，会发现方块实现了下落、左移和右移。

步骤十五：检查当前正在下落的方块左右移动时是否出界 、是否重合

在Tetris类中，新建moveRightAction、moveLeftAction、outOfBounds以及coincide方法，其中，outOfBounds实现是否左右出界，coincide实现是否重合。

outOfBounds方法实现的逻辑为：如果方块中，有一个格子的列小于0或者列大于等于10，则说明该方块左右出界了。

coincide方法的实现逻辑为：循环获取正在下落的方块中，每一个格子的row、col坐标，如果对应墙上的wall[row][col]该位置是否为null，如果不为null，则说明墙上的该位置有格子存在，也就是我们说的出现了重合现象。代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

//wall[2][2] = new Cell(2,2, T);

tetromino= Tetromino.*randomOne*();

nextOne=Tetromino.*randomOne*();

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();// [c]

switch (key) {

case KeyEvent.*VK\_DOWN*:

tetromino.softDrop();

break;

case KeyEvent.*VK\_RIGHT*:

moveRightAction();

break;

case KeyEvent.*VK\_LEFT*:

moveLeftAction();

break;

}

repaint();// 再画一次!

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

}

/\*\*

\* 绘制下一个要下落的方块

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int col = cell.getCol();

if (col < 0 || col >= *COLS*) {

return true;

}

}

return false;

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

int col = cell.getCol();

// 如果墙的row,col位置上有格子,就重叠了!

if (row >= 0 && row < *ROWS* && col >= 0 && col <= *COLS*

&& wall[row][col] != null) {

return true;// 重叠

}

}

return false;

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

// 尝试先向右移动, 如果发现超出了边界, 就

// 向左移动, 修正回来.

tetromino.moveRight();// coincide重叠

if (outOfBounds() || coincide()) {

tetromino.moveLeft();

}

}

public void moveLeftAction() {

tetromino.moveLeft();

if (outOfBounds() || coincide()) {

tetromino.moveRight();

}

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，会发现左右移动不会出现越界的现象，并且移动方块不会和界面上的黄色方块重叠。

步骤十六： 实现下落流程

处理下落的流程为：

1) 判断方块能否继续下落，如果能下落, 则下落一步。

2) 如果方块不能下落，则落到墙上。

3) 落到墙面上以后, 销毁充满的行, 并且计算得分。

4) 检查游戏是否结束。

解决方案：

1） 创建canDrop方法判断，判断方块能否继续下落。方块不能继续下落的条件有两个，一是当前下落的方块中的某一个格子的行坐标，与最后一行相等；二是当前下落的方块中的某一个格子的row，col坐标，与其对应的wall[row+1][col]不为null，则说明这个格子对应的下一行，在墙已经存在格子了。其余情况方块都是可以下落的。

2） 创建landIntoWall方法，实现方块着落在墙上。实现该方法首先构建循环，然后，在循环中，获取正在下落方块的每一个格子，格子的行列坐标用(row,col)表示，最后将格子放置到墙对应的位置wall[row][col]。

3） 构建destoryLines方法，在该方法中，循环墙上的每一行，如果墙上的某一行对应的都有格子，说明该行是满的（构建fullCells方法实现进行判断）。如果这一行是满的，则销毁这一行。请看图-9。

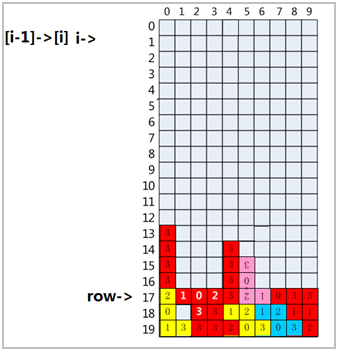


图 – 9

在此，构建destoryRow方法，实现销毁一行，在该方法中，从row循环到1（row为满的一行），在循环中，使用System类的arraycopy方法将（row-1）复制到row行，如图-8所示。最后，使用Arrays的fill方法将墙的第0行的所有格子填充为null。在destoryLines方法的最后，计算得分。

4）检查游戏是否结束的方法在后续介绍。代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

... ...

}

/\*\*

\* 绘制下一个要下落的方块

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

if (canDrop()) {

tetromino.softDrop();

} else {

landIntoWall();

destoryLines();

tetromino = nextOne;

nextOne = Tetromino.*randomOne*();

}

}

private static int[] *scoreTable* = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

int lines = 0;

for (int row = 0; row < wall.length; row++) {

if (fullCells(row)) {

deleteRow(row);

lines++;

}

}

this.score += *scoreTable*[lines];

this.lines += lines;

}

private void deleteRow(int row) {

for (int i = row; i >= 1; i--) {

System.*arraycopy*(wall[i - 1], 0, wall[i], 0, *COLS*);

}

Arrays.*fill*(wall[0], null);

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

Cell[] line = wall[row];

for (Cell cell : line) {

if (cell == null) {// 如果有null返回false 否则返回 true

return false;

}

}

return true;

}

private void landIntoWall() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

int col = cell.getCol();

wall[row][col] = cell;

}

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

if (row == *ROWS* - 1) {

return false;

}

}

for (Cell cell : cells) {// Java 5 以后可以使用

int row = cell.getRow() + 1;

int col = cell.getCol();

if (row >= 0 && row < *ROWS* && col >= 0 && col <= *COLS*

&& wall[row][col] != null) {

return false;

}

}

return true;

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，发现可以实现行的销毁，另外，可以将Tetromino类的randomOne方法的实现都改成O形方块，这样界面上生成的方块就都为O行方块，可以方便的测试销毁行的操作。

步骤十七：硬下落流程

硬下落的意思为方块下落到不能下落为止；当不能下落时，让方块着落，然后销毁满的行，并将该流程绑定到空格键上，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

wall[2][2] = new Cell(2, 2, *T*);

tetromino = Tetromino.*randomOne*();

nextOne = Tetromino.*randomOne*();

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();// [c]

switch (key) {

case KeyEvent.*VK\_DOWN*:

softDropAction();

break;

case KeyEvent.*VK\_RIGHT*:

moveRightAction();

break;

case KeyEvent.*VK\_LEFT*:

moveLeftAction();

break;

case KeyEvent.*VK\_SPACE*:

hardDropAction();

break;

}

repaint();// 再画一次!

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

... ...

}

private static int[] *scoreTable* = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

while (canDrop()) {

tetromino.softDrop();

}

landIntoWall();

destoryLines();

tetromino = nextOne;

nextOne = Tetromino.*randomOne*();

}

public static void main(String[] args) {

... ...

}

}

步骤十八：绘制分数

在Tetris类中，构建paintScore方法，在该方法中使用Java API提供的方法将分数绘制到界面上，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

// 画背景, 画墙, 画正在下落的方块 画下一个方块...

g.drawImage(*background*, 0, 0, null);

g.translate(15, 15);// 坐标系平移

paintWall(g);// 画墙

paintTetromino(g);// 绘制正在下落的方块

paintNextOne(g);//绘制下一个要下落的方块

paintScore(g);//绘制分数

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

... ...

}

public static final int *FONT\_COLOR* = 0x667799;

public static final int *FONT\_SIZE* = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

int x = 290;

int y = 160;

g.setColor(new Color(*FONT\_COLOR*));

Font font = g.getFont();// 取得g当前字体

font = new Font(font.getName(), font.getStyle(), *FONT\_SIZE*);

g.setFont(font);// 更改了g的字体

String str = "SCORE:" + score;

g.drawString(str, x, y);

y += 56;

str = "LINES:" + lines;

g.drawString(str, x, y);

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

... ...

}

private static int[] *scoreTable* = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

... ...

}

public static void main(String[] args) {

... ...

}

}

运行上述代码，界面显示效果如图-10所示：

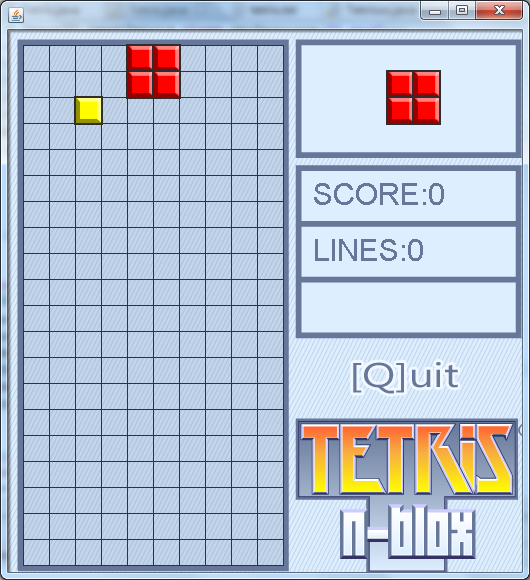


图- 10

观察图-10会发现，在图的右侧出现了“SCORE：0”和“LINES：0”字样的文字。

步骤十九：旋转流程控制

各种形状的方块都可以实现旋转，有的方块的旋转状态有两种，有的方块的旋转状态为四种，下面首先以T形为例能介绍旋转算法。T形方块旋转后有四种状态，如图-11所示：



图 – 11

在图-11中，显示了T形方块进行旋转后的四种状态。并且在旋转过程中，展示了如何从一种状态到另一种状态，例如：从T0到T1 ，坐标换算的公式为T0.轴+S1，那么这里的T0.轴和S1分别指什么呢？请看图-12，该图中的cells即为Tetromino类的cells属性。



图 – 12

观察图-12可以看出，T0.轴指的是cells[0]，其在图中的行列坐标为（2，2）。另外，计算S1的公式为：S1 = T1 – T0.轴，其中，T1为旋转后的T形方块。以 cells[0]为轴，从T0到T1坐标变换为：

cells [ 0 ].row = cells [ 0 ].row + (0)

cells [ 0 ].col = cells [ 0 ].col + (0)

cells [ 1 ].row = cells [ 0 ].row + (-1)

cells [ 1 ].col = cells [ 0 ].col + (0)

cells [ 2 ].row = cells [ 0 ].row + (1)

cells [ 2 ].col = cells [ 0 ].col + (0)

cells [ 3 ].row = cells [ 0 ].row + (0)

cells [ 3 ].col = cells [ 0 ].col + (-1)

以上代码中，等号前面的代码即为T1状态的四个格子的行列坐标；等号后边的代码即为“T0.轴+S1”。

根据以上分析，可以得出从T0旋转到T1，变换坐标用的八个值为：

0, 0, -1, 0, 1, 0, 0, -1

以上八个值即为S1。

以此类推，从T1旋转到T2、T2旋转到T3、T3旋转到T0的原理和T0旋转到T1相同，如图-13所示。



图 – 13

L形旋转过程坐标变换如图-14所示：



图 – 14

J形旋转过程坐标变换如图-15所示。



图 – 15

I形旋转过程坐标变换如图-16所示。



图 – 16

O形旋转过程坐标变换如图-17所示。



图 – 17

S形旋转过程坐标变换如图-18所示。



图 – 18

Z形旋转过程坐标变换如图-19所示。



图 – 19

实现方块旋转的代码如下所示：

package com.tarena.tetris;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 旋转状态 \*/

protected State[] states;

/\*\* 旋转状态的序号 states[index%states.length] \*/

protected int index = 10000;

/\*\* 内部类 \*/

protected class State {

int row0, col0, row1, col1, row2, col2, row3, col3;

public State(int row0, int col0, int row1, int col1, int row2,

int col2, int row3, int col3) {

this.row0 = row0;

this.col0 = col0;

this.row1 = row1;

this.col1 = col1;

this.row2 = row2;

this.col2 = col2;

this.row3 = row3;

this.col3 = col3;

}

}

/\*\* 向右转 \*/

public void rotateRight() {

// 取得变换的下个数据状态 states[n]

// 取得当前轴的 row, col

// 旋转以后的数据=(row,col) + states[n]

index++;

State s = states[index % states.length];

Cell o = cells[0];

int row = o.getRow();

int col = o.getCol();

cells[1].setRow(row + s.row1);

cells[1].setCol(col + s.col1);

cells[2].setRow(row + s.row2);

cells[2].setCol(col + s.col2);

cells[3].setRow(row + s.row3);

cells[3].setCol(col + s.col3);

}

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

... ...

}

/\*\*

\* 方块下落一个格子

\*/

public void softDrop() {

... ...

}

/\*\*

\* 方块左移一个格子

\*/

public void moveLeft() {

... ...

}

/\*\*

\* 方块右移一个格子

\*/

public void moveRight() {

... ...

}

/\*\*

\* 覆盖toString方法，显示方块中每个格子的行列信息

\*/

public String toString() {

return Arrays.toString(cells);

}

}

/\*\*

\* T形方块

\*/

class T extends Tetromino {

public T() {

cells[0] = new Cell(0, 4, Tetris.*T*);

cells[1] = new Cell(0, 3, Tetris.*T*);

cells[2] = new Cell(0, 5, Tetris.*T*);

cells[3] = new Cell(1, 4, Tetris.*T*);

states = new State[4];

states[0] = new State(0, 0, 0, -1, 0, 1, 1, 0);

states[1] = new State(0, 0, -1, 0, 1, 0, 0, -1);

states[2] = new State(0, 0, 0, 1, 0, -1, -1, 0);

states[3] = new State(0, 0, 1, 0, -1, 0, 0, 1);

}

}

/\*\*

\* I形方块

\*/

class I extends Tetromino {

public I() {

cells[0] = new Cell(0, 4, Tetris.*I*);

cells[1] = new Cell(0, 3, Tetris.*I*);

cells[2] = new Cell(0, 5, Tetris.*I*);

cells[3] = new Cell(0, 6, Tetris.*I*);

states = new State[] { new State(0, 0, 0, -1, 0, 1, 0, 2),

new State(0, 0, -1, 0, 1, 0, 2, 0) };

}

}

/\*\*

\* S形方块

\*/

class S extends Tetromino {

public S() {

cells[0] = new Cell(1, 4, null);

cells[1] = new Cell(1, 3, null);

cells[2] = new Cell(0, 4, null);

cells[3] = new Cell(0, 5, null);

states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),

new State(0, 0, -1, 0, 0, 1, 1, 1) };

}

}

/\*\*

\* Z形方块

\*/

class Z extends Tetromino {

public Z() {

cells[0] = new Cell(1, 4, Tetris.*Z*);

cells[1] = new Cell(0, 3, Tetris.*Z*);

cells[2] = new Cell(0, 4, Tetris.*Z*);

cells[3] = new Cell(1, 5, Tetris.*Z*);

states = new State[] { new State(0, 0, -1, -1, -1, 0, 0, 1),

new State(0, 0, -1, 1, 0, 1, 1, 0) };

}

}

/\*\*

\* O形方块

\*/

class O extends Tetromino {

public O() {

cells[0] = new Cell(0, 4, Tetris.*O*);

cells[1] = new Cell(0, 5, Tetris.*O*);

cells[2] = new Cell(1, 4, Tetris.*O*);

cells[3] = new Cell(1, 5, Tetris.*O*);

states = new State[] { new State(0, 0, 0, 1, 1, 0, 1, 1),

new State(0, 0, 0, 1, 1, 0, 1, 1) };

}

}

/\*\*

\* L形方块

\*/

class L extends Tetromino {

public L() {

cells[0] = new Cell(0, 4, Tetris.*L*);

cells[1] = new Cell(0, 3, Tetris.*L*);

cells[2] = new Cell(0, 5, Tetris.*L*);

cells[3] = new Cell(1, 3, Tetris.*L*);

states = new State[] { new State(0, 0, 0, 1, 0, -1, -1, 1),

new State(0, 0, 1, 0, -1, 0, 1, 1),

new State(0, 0, 0, -1, 0, 1, 1, -1),

new State(0, 0, -1, 0, 1, 0, -1, -1) };

}

}

/\*\*

\* J形方块

\*/

class J extends Tetromino {

public J() {

cells[0] = new Cell(0, 4, Tetris.*J*);

cells[1] = new Cell(0, 3, Tetris.*J*);

cells[2] = new Cell(0, 5, Tetris.*J*);

cells[3] = new Cell(1, 5, Tetris.*J*);

states = new State[] { new State(0, 0, 0, -1, 0, 1, 1, 1),

new State(0, 0, -1, 0, 1, 0, 1, -1),

new State(0, 0, 0, 1, 0, -1, -1, -1),

new State(0, 0, 1, 0, -1, 0, -1, 1) };

}

}

重构Tetris类的代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage *background*;

public static BufferedImage *T*;

public static BufferedImage *S*;

public static BufferedImage *I*;

public static BufferedImage *L*;

public static BufferedImage *J*;

public static BufferedImage *O*;

public static BufferedImage *Z*;

public static final int *ROWS* = 20;// 背景墙的行数

public static final int *COLS* = 10;// 背景墙的列数

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[*ROWS*][*COLS*];

wall[2][2] = new Cell(2, 2, *T*);

tetromino = Tetromino.*randomOne*();

nextOne = Tetromino.*randomOne*();

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();// [c]

switch (key) {

case KeyEvent.*VK\_DOWN*:

softDropAction();

break;

case KeyEvent.*VK\_RIGHT*:

moveRightAction();

break;

case KeyEvent.*VK\_LEFT*:

moveLeftAction();

break;

case KeyEvent.*VK\_SPACE*:

hardDropAction();

break;

case KeyEvent.*VK\_UP*:

rotateRightAction();

break;

}

repaint();// 再画一次!

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

}

public static final int *FONT\_COLOR* = 0x667799;

public static final int *FONT\_SIZE* = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

... ...

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int *CELL\_SIZE* = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

... ...

}

private static int[] *scoreTable* = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

... ...

}

/\*\* 在Tetris类中添加 旋转流程控制方法 \*/

public void rotateRightAction() {

tetromino.rotateRight();

if (outOfBounds() || coincide()) {

tetromino.rotateLeft();

}

}

public static void main(String[] args) {

... ...

}

}

步骤二十：实现自由下落

在action 方法中添加定时计划任务，实现自由下落功能，代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import java.util.Timer;

import java.util.TimerTask;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage background;

public static BufferedImage T;

public static BufferedImage S;

public static BufferedImage I;

public static BufferedImage L;

public static BufferedImage J;

public static BufferedImage O;

public static BufferedImage Z;

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

/\*\* 在Tetris类中增加定时器 \*/

private Timer timer;

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[ROWS][COLS];

wall[2][2] = new Cell(2, 2, T);

tetromino = Tetromino.randomOne();

nextOne = Tetromino.randomOne();

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();// [c]

switch (key) {

case KeyEvent.VK\_DOWN:

softDropAction();

break;

case KeyEvent.VK\_RIGHT:

moveRightAction();

break;

case KeyEvent.VK\_LEFT:

moveLeftAction();

break;

case KeyEvent.VK\_SPACE:

hardDropAction();

break;

case KeyEvent.VK\_UP:

rotateRightAction();

break;

}

repaint();// 再画一次!

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

// 在Action 方法中添加,定时计划任务

timer = new Timer();

timer.schedule(new TimerTask() {

public void run() {

softDropAction();

repaint();

}

}, 10, 10);

}

public static final int FONT\_COLOR = 0x667799;

public static final int FONT\_SIZE = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

... ...

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int CELL\_SIZE = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

... ...

}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

... ...

}

/\*\* 在Tetris类中添加 旋转流程控制方法 \*/

public void rotateRightAction() {

... ...

}

public static void main(String[] args) {

... ...

}

}

步骤二十二：开始流程控制、暂停流程控制、继续流程控制以及结束流程控制

本案例，游戏分为三种状态：Running、Pause、GameOver，这三种状态之间的相互切换如图-20所示。



图 – 20

从图-20可以看出，游戏开始首先进入到Running状态， 在Running状态时按下键【P】则可以进入Pause状态； 在Pause状态按下键【C】则转换到Running状态； 在Running状态如果下一个方块不能出场了则进入GameOver状态； 在GameOver状态按下键【S】则转换到Running状态；在Running状态、Pause状态、GameOver状态时，按下键【Q】则，退出系统。图-21展示了本案例中各个键的用途。

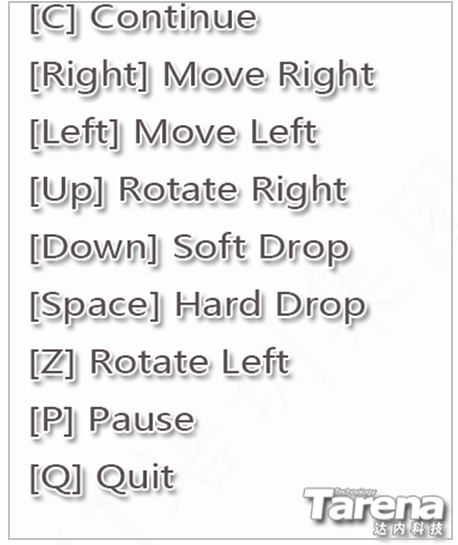


图 – 21

各个状态之间的转换的实现过程如下：

1. 在Tetris中添加状态管理的属性state，同时增加三个常量RUNNING、PAUSE、 GAME\_OVER，值分别为0、1、2。

2. 在定时器中，如果为Running状态，则执行下落流程。在此过程中，发现游戏结束了，则将状态更新为GameOver。

3. 键盘事件中，优先处理按键【Q】，退出系统。如果按下的为【P】键，则将状态转换为Pause状态；如果为Pause状态，则只能响应按键【C】，继续游戏，并将游戏状态更新为Running；当游戏状态为GameOver时，可以响应按键【S】，重新开始游戏，并将游戏状态更新为Running。

4. 在下落流程中，如果游戏结束了，则将游戏状态更新GameOver。

代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import java.util.Timer;

import java.util.TimerTask;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

/\*\* 游戏的当前状态: RUNNING PAUSE GAME\_OVER \*/

private int state;

public static final int RUNNING = 0;

public static final int PAUSE = 1;

public static final int GAME\_OVER = 2;

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage background;

private static BufferedImage gameOver;

private static BufferedImage pause;

public static BufferedImage T;

public static BufferedImage S;

public static BufferedImage I;

public static BufferedImage L;

public static BufferedImage J;

public static BufferedImage O;

public static BufferedImage Z;

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

/\*\* 在Tetris类中增加定时器 \*/

private Timer timer;

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

try {

// Tetris.class的同一个包中找 "tetris.png"

background = ImageIO.read(Tetris.class.getResource("tetris.png"));

gameOver = ImageIO.read(Tetris.class.getResource("game-over.png"));

pause = ImageIO.read(Tetris.class.getResource("pause.png"));

T = ImageIO.read(Tetris.class.getResource("T.png"));

I = ImageIO.read(Tetris.class.getResource("I.png"));

S = ImageIO.read(Tetris.class.getResource("S.png"));

Z = ImageIO.read(Tetris.class.getResource("Z.png"));

J = ImageIO.read(Tetris.class.getResource("J.png"));

L = ImageIO.read(Tetris.class.getResource("L.png"));

O = ImageIO.read(Tetris.class.getResource("O.png"));

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

// 画背景, 画墙, 画正在下落的方块 画下一个方块...

g.drawImage(background, 0, 0, null);

g.translate(15, 15);// 坐标系平移

paintWall(g);// 画墙

paintTetromino(g);// 绘制正在下落的方块

paintNextOne(g);//绘制下一个要下落的方块

paintScore(g);//绘制分数

paintState(g);// 绘制游戏的状态

}

private void paintState(Graphics g) {

switch (state) {

case PAUSE:

g.drawImage(pause, -15, -15, null);

break;

case GAME\_OVER:

g.drawImage(gameOver, -15, -15, null);

break;

}

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[ROWS][COLS];

//wall[2][2] = new Cell(2, 2, T);

tetromino = Tetromino.randomOne();

nextOne = Tetromino.randomOne();

state = RUNNING;

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

switch (state) {

case GAME\_OVER:

processGameoverKey(key);

break;

case PAUSE:

processPauseKey(key);

break;

case RUNNING:

processRunningKey(key);

}

repaint();

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

// 在Action 方法中添加,定时计划任务

timer = new Timer();

timer.schedule(new TimerTask() {

public void run() {

if (state == RUNNING) {

softDropAction();

}

repaint();

}

}, 10, 10);

}

private void processPauseKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_C:

state = RUNNING;

break;

}

}

protected void processRunningKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_RIGHT:

Tetris.this.moveRightAction();

break;

case KeyEvent.VK\_LEFT:

Tetris.this.moveLeftAction();

break;

case KeyEvent.VK\_DOWN:

softDropAction();

break;

case KeyEvent.VK\_SPACE:

hardDropAction();

break;

case KeyEvent.VK\_UP:

rotateRightAction();

break;

case KeyEvent.VK\_P:

state = PAUSE;

break;

}

}

protected void processGameoverKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_S:

/\*\* 游戏重新开始 \*/

this.lines = 0;

this.score = 0;

this.wall = new Cell[ROWS][COLS];

this.tetromino = Tetromino.randomOne();

this.nextOne = Tetromino.randomOne();

this.state = RUNNING;

this.index = 0;

break;

}

}

public static final int FONT\_COLOR = 0x667799;

public static final int FONT\_SIZE = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

... ...

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int CELL\_SIZE = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

if (canDrop()) {

tetromino.softDrop();

} else {

landIntoWall();

destoryLines();

if (isGameOver()) {

state = GAME\_OVER;

} else {

tetromino = nextOne;

nextOne = Tetromino.randomOne();

}

}

}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

while (canDrop()) {

tetromino.softDrop();

}

landIntoWall();

destoryLines();

if (isGameOver()) {

state = GAME\_OVER;

} else {

tetromino = nextOne;

nextOne = Tetromino.randomOne();

}

}

/\*\* 在Tetris类中添加 旋转流程控制方法 \*/

public void rotateRightAction() {

... ...

}

/\*\* 检查游戏是否结束 \*/

private boolean isGameOver() {

// 如果下一个方块没有出场位置了, 则游戏结束

// 就是: 下一个出场方块每个格子行列对应的

// 墙上位置如果有格子, 就游戏结束

Cell[] cells = nextOne.cells;

for (Cell cell : cells) {

int row = cell.getRow();

int col = cell.getCol();

if (wall[row][col] != null) {

return true;

}

}

return false;

}

public static void main(String[] args) {

... ...

}

}

步骤二十三：处理下落的级别

设置游戏中方块下落的初始速度为40，当销毁掉的行数为100行时，则速度speed加快1。游戏级别初始值为1，随着下落速度的加快，级别增加。当定时器运行speed次时，方块下落一步。每次暂停在开始以后，定时器执行的次数从新开始计数。实现代码如下所示：

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import java.util.Timer;

import java.util.TimerTask;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

/\*\* 游戏的当前状态: RUNNING PAUSE GAME\_OVER \*/

private int state;

public static final int RUNNING = 0;

public static final int PAUSE = 1;

public static final int GAME\_OVER = 2;

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage background;

private static BufferedImage gameOver;

private static BufferedImage pause;

public static BufferedImage T;

public static BufferedImage S;

public static BufferedImage I;

public static BufferedImage L;

public static BufferedImage J;

public static BufferedImage O;

public static BufferedImage Z;

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

/\*\* 在Tetris类中增加定时器 \*/

private Timer timer;

/\*\* 速度 \*/

private int speed;

/\*\* 难度级别 \*/

private int level;

/\*\* 下落计数器 当 index%speed==0 时候下落一次 \*/

private int index;

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

... ...

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

... ...

}

private void paintState(Graphics g) {

... ...

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[ROWS][COLS];

//wall[2][2] = new Cell(2, 2, T);

tetromino = Tetromino.randomOne();

nextOne = Tetromino.randomOne();

state = RUNNING;

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

switch (state) {

case GAME\_OVER:

processGameoverKey(key);

break;

case PAUSE:

processPauseKey(key);

break;

case RUNNING:

processRunningKey(key);

}

repaint();

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

// 在Action 方法中添加,定时计划任务

timer = new Timer();

timer.schedule(new TimerTask() {

public void run() {

speed = 40 - (lines / 100);

speed = speed < 1 ? 1 : speed;

level = 41 - speed;

if (state == RUNNING && index % speed == 0) {

softDropAction();

}

index++;

repaint();

}

}, 10, 10);

}

private void processPauseKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_C:

index = 0;

state = RUNNING;

break;

}

}

protected void processRunningKey(int key) {

... ...

}

protected void processGameoverKey(int key) {

... ...

}

public static final int FONT\_COLOR = 0x667799;

public static final int FONT\_SIZE = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

int x = 290;

int y = 160;

g.setColor(new Color(FONT\_COLOR));

Font font = g.getFont();// 取得g当前字体

font = new Font(font.getName(), font.getStyle(), FONT\_SIZE);

g.setFont(font);// 更改了g的字体

String str = "SCORE:" + score;

g.drawString(str, x, y);

y += 56;

str = "LINES:" + lines;

g.drawString(str, x, y);

y += 56;

g.drawString("LEVEL:" + level, x, y);

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

... ...

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

... ...

}

public static final int CELL\_SIZE = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

... ...

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

... ...

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

... ...

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

... ...

}

public void moveLeftAction() {

... ...

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

... ...

}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

... ...

}

private void deleteRow(int row) {

... ...

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

... ...

}

private void landIntoWall() {

... ...

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

... ...

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

... ...

}

/\*\* 在Tetris类中添加 旋转流程控制方法 \*/

public void rotateRightAction() {

... ...

}

/\*\* 检查游戏是否结束 \*/

private boolean isGameOver() {

... ...

}

public static void main(String[] args) {

... ...

}

}

* 完整代码

本案例中，Cell类的完整代码如下所示：

package com.tarena.tetris;

import java.awt.image.BufferedImage;

/\*\*

\* 格子

\*/

public class Cell {

private int row;//格子的行

private int col;//格子的列

private BufferedImage image;//格子的贴图

public Cell(int row, int col, BufferedImage image) {

super();

this.row = row;

this.col = col;

this.image = image;

}

public int getRow() {

return row;

}

public void setRow(int row) {

this.row = row;

}

public int getCol() {

return col;

}

public void setCol(int col) {

this.col = col;

}

public BufferedImage getImage() {

return image;

}

public void setImage(BufferedImage image) {

this.image = image;

}

public void drop(){

row++;

}

public void moveRight(){

col++;

}

public void moveLeft(){

col--;

}

@Override//重写

public String toString() {

return row+","+col;

}

}

Tetromino类的完整代码如下所示:

package com.tarena.tetris;

import java.util.Arrays;

import java.util.Random;

/\*\*

\* 4格方块

\*/

public abstract class Tetromino {

protected Cell[] cells = new Cell[4];

/\*\* 旋转状态 \*/

protected State[] states;

/\*\* 旋转状态的序号 states[index%states.length] \*/

protected int index = 10000;

/\*\* 内部类 \*/

protected class State {

int row0, col0, row1, col1, row2, col2, row3, col3;

public State(int row0, int col0, int row1, int col1, int row2,

int col2, int row3, int col3) {

this.row0 = row0;

this.col0 = col0;

this.row1 = row1;

this.col1 = col1;

this.row2 = row2;

this.col2 = col2;

this.row3 = row3;

this.col3 = col3;

}

}

/\*\* 向右转 \*/

public void rotateRight() {

// 取得变换的下个数据状态 states[n]

// 取得当前轴的 row, col

// 旋转以后的数据=(row,col) + states[n]

index++;

State s = states[index % states.length];

Cell o = cells[0];

int row = o.getRow();

int col = o.getCol();

cells[1].setRow(row + s.row1);

cells[1].setCol(col + s.col1);

cells[2].setRow(row + s.row2);

cells[2].setCol(col + s.col2);

cells[3].setRow(row + s.row3);

cells[3].setCol(col + s.col3);

}

public void rotateLeft() {

index--;

State s = states[index % states.length];

Cell o = cells[0];

int row = o.getRow();

int col = o.getCol();

cells[1].setRow(row + s.row1);

cells[1].setCol(col + s.col1);

cells[2].setRow(row + s.row2);

cells[2].setCol(col + s.col2);

cells[3].setRow(row + s.row3);

cells[3].setCol(col + s.col3);

}

/\*\* 工厂方法, 随机产生4格方块 \*/

public static Tetromino randomOne() {

Random random = new Random();

int type = random.nextInt(7);

switch (type) {

case 0:

return new O();

case 1:

return new S();

case 2:

return new Z();

case 3:

return new J();

case 4:

return new L();

case 5:

return new I();

case 6:

return new T();

}

return null;

}

/\*\*

\* 方块下落一个格子

\*/

public void softDrop() {

for (int i = 0; i < cells.length; i++) {

cells[i].drop();

}

}

/\*\*

\* 方块左移一个格子

\*/

public void moveLeft() {

for (int i = 0; i < cells.length; i++) {

cells[i].moveLeft();

}

}

/\*\*

\* 方块右移一个格子

\*/

public void moveRight() {

for (int i = 0; i < cells.length; i++) {

cells[i].moveRight();

}

}

/\*\*

\* 覆盖toString方法，显示方块中每个格子的行列信息

\*/

public String toString() {

return Arrays.toString(cells);

}

}

/\*\*

\* T形方块

\*/

class T extends Tetromino {

public T() {

cells[0] = new Cell(0, 4, Tetris.*T*);

cells[1] = new Cell(0, 3, Tetris.*T*);

cells[2] = new Cell(0, 5, Tetris.*T*);

cells[3] = new Cell(1, 4, Tetris.*T*);

states = new State[4];

states[0] = new State(0, 0, 0, -1, 0, 1, 1, 0);

states[1] = new State(0, 0, -1, 0, 1, 0, 0, -1);

states[2] = new State(0, 0, 0, 1, 0, -1, -1, 0);

states[3] = new State(0, 0, 1, 0, -1, 0, 0, 1);

}

}

/\*\*

\* I形方块

\*/

class I extends Tetromino {

public I() {

cells[0] = new Cell(0, 4, Tetris.*I*);

cells[1] = new Cell(0, 3, Tetris.*I*);

cells[2] = new Cell(0, 5, Tetris.*I*);

cells[3] = new Cell(0, 6, Tetris.*I*);

states = new State[] { new State(0, 0, 0, -1, 0, 1, 0, 2),

new State(0, 0, -1, 0, 1, 0, 2, 0) };

}

}

/\*\*

\* S形方块

\*/

class S extends Tetromino {

public S() {

cells[0] = new Cell(1, 4, Tetris.*S*);

cells[1] = new Cell(1, 3, Tetris.*S*);

cells[2] = new Cell(0, 4, Tetris.*S*);

cells[3] = new Cell(0, 5, Tetris.*S*);

states = new State[] { new State(0, 0, 0, -1, -1, 0, -1, 1),

new State(0, 0, -1, 0, 0, 1, 1, 1) };

}

}

/\*\*

\* Z形方块

\*/

class Z extends Tetromino {

public Z() {

cells[0] = new Cell(1, 4, Tetris.*Z*);

cells[1] = new Cell(0, 3, Tetris.*Z*);

cells[2] = new Cell(0, 4, Tetris.*Z*);

cells[3] = new Cell(1, 5, Tetris.*Z*);

states = new State[] { new State(0, 0, -1, -1, -1, 0, 0, 1),

new State(0, 0, -1, 1, 0, 1, 1, 0) };

}

}

/\*\*

\* O形方块

\*/

class O extends Tetromino {

public O() {

cells[0] = new Cell(0, 4, Tetris.*O*);

cells[1] = new Cell(0, 5, Tetris.*O*);

cells[2] = new Cell(1, 4, Tetris.*O*);

cells[3] = new Cell(1, 5, Tetris.*O*);

states = new State[] { new State(0, 0, 0, 1, 1, 0, 1, 1),

new State(0, 0, 0, 1, 1, 0, 1, 1) };

}

}

/\*\*

\* L形方块

\*/

class L extends Tetromino {

public L() {

cells[0] = new Cell(0, 4, Tetris.*L*);

cells[1] = new Cell(0, 3, Tetris.*L*);

cells[2] = new Cell(0, 5, Tetris.*L*);

cells[3] = new Cell(1, 3, Tetris.*L*);

states = new State[] { new State(0, 0, 0, 1, 0, -1, -1, 1),

new State(0, 0, 1, 0, -1, 0, 1, 1),

new State(0, 0, 0, -1, 0, 1, 1, -1),

new State(0, 0, -1, 0, 1, 0, -1, -1) };

}

}

/\*\*

\* J形方块

\*/

class J extends Tetromino {

public J() {

cells[0] = new Cell(0, 4, Tetris.*J*);

cells[1] = new Cell(0, 3, Tetris.*J*);

cells[2] = new Cell(0, 5, Tetris.*J*);

cells[3] = new Cell(1, 5, Tetris.*J*);

states = new State[] { new State(0, 0, 0, -1, 0, 1, 1, 1),

new State(0, 0, -1, 0, 1, 0, 1, -1),

new State(0, 0, 0, 1, 0, -1, -1, -1),

new State(0, 0, 1, 0, -1, 0, -1, 1) };

}

}

Tetris类的完整代码如下所示:

package com.tarena.tetris;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import java.awt.image.BufferedImage;

import java.util.Arrays;

import java.util.Timer;

import java.util.TimerTask;

import javax.imageio.ImageIO;

import javax.swing.JFrame;

import javax.swing.JPanel;

/\*\*

\* 俄罗斯方块

\*/

public class Tetris extends JPanel {

/\*\* 游戏的当前状态: RUNNING PAUSE GAME\_OVER \*/

private int state;

public static final int RUNNING = 0;

public static final int PAUSE = 1;

public static final int GAME\_OVER = 2;

private int score;// 分数

private int lines;// 销毁的行数

private Cell[][] wall;// 背景墙

private Tetromino tetromino;// 正在下落的四格方块

private Tetromino nextOne;// 下一个四格方块

/\*\* 背景图片 \*/

private static BufferedImage background;

private static BufferedImage gameOver;

private static BufferedImage pause;

public static BufferedImage T;

public static BufferedImage S;

public static BufferedImage I;

public static BufferedImage L;

public static BufferedImage J;

public static BufferedImage O;

public static BufferedImage Z;

public static final int ROWS = 20;// 背景墙的行数

public static final int COLS = 10;// 背景墙的列数

/\*\* 在Tetris类中增加定时器 \*/

private Timer timer;

/\*\* 速度 \*/

private int speed;

/\*\* 难度级别 \*/

private int level;

/\*\* 下落计数器 当 index%speed==0 时候下落一次 \*/

private int index;

// 将图片素材, 复制到 com.tarena.tetris包中.

/\*\* 使用静态代码块加载静态的图片 \*/

static {

try {

// Tetris.class的同一个包中找 "tetris.png"

background = ImageIO.read(Tetris.class.getResource("tetris.png"));

gameOver = ImageIO.read(Tetris.class.getResource("game-over.png"));

pause = ImageIO.read(Tetris.class.getResource("pause.png"));

T = ImageIO.read(Tetris.class.getResource("T.png"));

I = ImageIO.read(Tetris.class.getResource("I.png"));

S = ImageIO.read(Tetris.class.getResource("S.png"));

Z = ImageIO.read(Tetris.class.getResource("Z.png"));

J = ImageIO.read(Tetris.class.getResource("J.png"));

L = ImageIO.read(Tetris.class.getResource("L.png"));

O = ImageIO.read(Tetris.class.getResource("O.png"));

} catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* JPanel paint() paint画 重写 paint() 修改原有的绘制方法

\*\*/

@Override

public void paint(Graphics g) {

// 画背景, 画墙, 画正在下落的方块 画下一个方块...

g.drawImage(background, 0, 0, null);

g.translate(15, 15);// 坐标系平移

paintWall(g);// 画墙

paintTetromino(g);// 绘制正在下落的方块

paintNextOne(g);//绘制下一个要下落的方块

paintScore(g);//绘制分数

paintState(g);// 绘制游戏的状态

}

private void paintState(Graphics g) {

switch (state) {

case PAUSE:

g.drawImage(pause, -15, -15, null);

break;

case GAME\_OVER:

g.drawImage(gameOver, -15, -15, null);

break;

}

}

/\*\* 在Tetris 添加启动方法 action() \*/

public void action() {

wall = new Cell[ROWS][COLS];

//wall[2][2] = new Cell(2, 2, T);

tetromino = Tetromino.randomOne();

nextOne = Tetromino.randomOne();

state = RUNNING;

// 处理键盘按下事件, 在按下按键时候执行下落方法

KeyAdapter l = new KeyAdapter() {

@Override

// key 按键 Pressed按下了

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

switch (state) {

case GAME\_OVER:

processGameoverKey(key);

break;

case PAUSE:

processPauseKey(key);

break;

case RUNNING:

processRunningKey(key);

}

repaint();

}

};

// 下落流程: 监听键盘事件->如果下箭头按下->

// 执行下落算法tetromino.softDrop()->

// 修改每个格子对象的数据->调用repaint()->

// 尽快调用paint()->paint方法会根据当前的数据

// 重新绘制界面 -> 看到移动以后的方块了

// 绑定事件到当前面板

this.requestFocus();

this.addKeyListener(l);

// 在Action 方法中添加,定时计划任务

timer = new Timer();

timer.schedule(new TimerTask() {

public void run() {

speed = 40 - (lines / 100);

speed = speed < 1 ? 1 : speed;

level = 41 - speed;

if (state == RUNNING && index % speed == 0) {

softDropAction();

}

index++;

repaint();

}

}, 10, 10);

}

private void processPauseKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_C:

index = 0;

state = RUNNING;

break;

}

}

protected void processRunningKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_RIGHT:

Tetris.this.moveRightAction();

break;

case KeyEvent.VK\_LEFT:

Tetris.this.moveLeftAction();

break;

case KeyEvent.VK\_DOWN:

softDropAction();

break;

case KeyEvent.VK\_SPACE:

hardDropAction();

break;

case KeyEvent.VK\_UP:

rotateRightAction();

break;

case KeyEvent.VK\_P:

state = PAUSE;

break;

}

}

protected void processGameoverKey(int key) {

switch (key) {

case KeyEvent.VK\_Q:

System.exit(0);

break;

case KeyEvent.VK\_S:

/\*\* 游戏重新开始 \*/

this.lines = 0;

this.score = 0;

this.wall = new Cell[ROWS][COLS];

this.tetromino = Tetromino.randomOne();

this.nextOne = Tetromino.randomOne();

this.state = RUNNING;

this.index = 0;

break;

}

}

public static final int FONT\_COLOR = 0x667799;

public static final int FONT\_SIZE = 30;

/\*\*

\* 绘制分数

\* @param g

\*/

private void paintScore(Graphics g) {

int x = 290;

int y = 160;

g.setColor(new Color(FONT\_COLOR));

Font font = g.getFont();// 取得g当前字体

font = new Font(font.getName(), font.getStyle(), FONT\_SIZE);

g.setFont(font);// 更改了g的字体

String str = "SCORE:" + score;

g.drawString(str, x, y);

y += 56;

str = "LINES:" + lines;

g.drawString(str, x, y);

y += 56;

g.drawString("LEVEL:" + level, x, y);

}

/\*\*

\* 绘制下一个要下落的方块

\*

\* @param g

\*/

private void paintNextOne(Graphics g) {

if (nextOne == null) {

return;

}

// 将每个格子的row,col 换算为x,y 然后贴图

Cell[] cells = nextOne.cells;

for (int i = 0; i < cells.length; i++) {

// i = 0 1 2 3

Cell cell = cells[i];

// cell 每个格子

int x = (cell.getCol() + 10) \* CELL\_SIZE;

int y = (cell.getRow() + 1) \* CELL\_SIZE;

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

}

/\*\*

\* 绘制正在下落的方块

\*

\* @param g

\*/

public void paintTetromino(Graphics g) {

if (tetromino == null) {

return;

}

// 将每个格子的row,col 换算为x,y 然后贴图

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

// i = 0 1 2 3

Cell cell = cells[i];

// cell 每个格子

int x = cell.getCol() \* CELL\_SIZE;

int y = cell.getRow() \* CELL\_SIZE;

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

}

public static final int CELL\_SIZE = 26;

/\*\* 画墙 \*/

private void paintWall(Graphics g) {

for (int row = 0; row < wall.length; row++) {

Cell[] line = wall[row];

// line 代表墙上的每一行

for (int col = 0; col < line.length; col++) {

Cell cell = line[col];

// cell 代表墙上的每个格子

int x = col \* CELL\_SIZE;

int y = row \* CELL\_SIZE;

if (cell == null) {

g.drawRect(x, y, CELL\_SIZE, CELL\_SIZE);

} else {

g.drawImage(cell.getImage(), x - 1, y - 1, null);

}

// g.drawString(row+","+col, x,y+CELL\_SIZE);

}

}

}

/\*\* 检查当前正在下落的方块是否出界了 \*/

private boolean outOfBounds() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int col = cell.getCol();

if (col < 0 || col >= COLS) {

return true;

}

}

return false;

}

/\*\* 检查正在下落的方块是否与墙上的砖块重叠 \*/

private boolean coincide() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

int col = cell.getCol();

// 如果墙的row,col位置上有格子,就重叠了!

if (row >= 0 && row < ROWS && col >= 0 && col <= COLS

&& wall[row][col] != null) {

return true;// 重叠

}

}

return false;

}

/\*\* 在Tetris 类上添加方法, 向右移动的流程控制 \*/

public void moveRightAction() {

// 尝试先向右移动, 如果发现超出了边界, 就

// 向左移动, 修正回来.

tetromino.moveRight();// coincide重叠

if (outOfBounds() || coincide()) {

tetromino.moveLeft();

}

}

public void moveLeftAction() {

tetromino.moveLeft();

if (outOfBounds() || coincide()) {

tetromino.moveRight();

}

}

/\*\* 下落流程控制 \*/

public void softDropAction() {

if (canDrop()) {

tetromino.softDrop();

} else {

landIntoWall();

destoryLines();

if (isGameOver()) {

state = GAME\_OVER;

} else {

tetromino = nextOne;

nextOne = Tetromino.randomOne();

}

}

}

private static int[] scoreTable = { 0, 1, 10, 50, 100 };

// 0 1 2 3 4

private void destoryLines() {

int lines = 0;

for (int row = 0; row < wall.length; row++) {

if (fullCells(row)) {

deleteRow(row);

lines++;

}

}

this.score += scoreTable[lines];

this.lines += lines;

}

private void deleteRow(int row) {

for (int i = row; i >= 1; i--) {

System.arraycopy(wall[i - 1], 0, wall[i], 0, COLS);

}

Arrays.fill(wall[0], null);

}

/\*\*

\* 检查当前行的每个格子,是否是满的，如果满了则返回true，否则返回false

\*/

private boolean fullCells(int row) {

Cell[] line = wall[row];

for (Cell cell : line) {

if (cell == null) {// 如果有null返回false 否则返回 true

return false;

}

}

return true;

}

private void landIntoWall() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

int col = cell.getCol();

wall[row][col] = cell;

}

}

/\*\* 检查当前的方块是否能够下落, 返回true能够下落 \*/

private boolean canDrop() {

Cell[] cells = tetromino.cells;

for (int i = 0; i < cells.length; i++) {

Cell cell = cells[i];

int row = cell.getRow();

if (row == ROWS - 1) {

return false;

}

}

for (Cell cell : cells) {// Java 5 以后可以使用

int row = cell.getRow() + 1;

int col = cell.getCol();

if (row >= 0 && row < ROWS && col >= 0 && col <= COLS

&& wall[row][col] != null) {

return false;

}

}

return true;

}

/\*\*

\* 硬下落流程, 下落到不能下落为止 绑定到 空格(VK\_SPACE)事件上

\* \*/

public void hardDropAction() {

while (canDrop()) {

tetromino.softDrop();

}

landIntoWall();

destoryLines();

if (isGameOver()) {

state = GAME\_OVER;

} else {

tetromino = nextOne;

nextOne = Tetromino.randomOne();

}

}

/\*\* 在Tetris类中添加 旋转流程控制方法 \*/

public void rotateRightAction() {

tetromino.rotateRight();

if (outOfBounds() || coincide()) {

tetromino.rotateLeft();

}

}

/\*\* 检查游戏是否结束 \*/

private boolean isGameOver() {

// 如果下一个方块没有出场位置了, 则游戏结束

// 就是: 下一个出场方块每个格子行列对应的

// 墙上位置如果有格子, 就游戏结束

Cell[] cells = nextOne.cells;

for (Cell cell : cells) {

int row = cell.getRow();

int col = cell.getCol();

if (wall[row][col] != null) {

return true;

}

}

return false;

}

public static void main(String[] args) {

JFrame frame = new JFrame();

// 在加载Tetris类的时候, 会执行静态代码块

// 静态代码块,装载了图片素材, 为图片对象

Tetris tetris = new Tetris();

// 将面板的颜色设置为蓝色，用于测试

tetris.setBackground(new Color(0x0000ff));

frame.add(tetris);

frame.setSize(530, 580);

frame.setLocationRelativeTo(null);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);// 在显示窗口时候,会"尽快"的调用paint()方法绘制界面

tetris.action();

}

}