其中形状代号为 0～6：0—I，1—S，2—Z，3—J，4—O，5—L，6—T。

旋转状态为 0～3：0—初始方块；1—旋转 1 次后的方块；2—旋转两次后的方块；3—旋转 3 次后的方块。

为了存储当前方块和下一方块的形状代号和旋转状态以及当前方块的初始位置，在 TetrisPanel 类中需要定义如下变量。

(1) 定义存储当前方块的形状代号和状态的变量。

```
int blockType,turnState;
```

(2) 定义存储下一方块的形状代号和状态的变量。

```
int nextBlockType = -1,nextTurnState = -1;
```

(3) 定义存储当前方块位置的变量。

```
int x,y;
```

(4) 定义 newBlock()方法。

```
public void newBlock() {
  Random r = new Random();
  //如果没有下一方块
  if (nextBlockType == -1 && nextTurnState == -1) {
      blockType = r.nextInt(7);
      turnState = r.nextInt(4);
      nextBlockType = r.nextInt(7);
      nextTurnState = r.nextInt(4);
  } else {
      blockType = nextBlockType;
      turnState = nextTurnState;
      nextBlockType = r.nextInt(7);
      nextTurnState = r.nextInt(4);
  }
  //当前方块的出场位置
  x = 4;
  y = 0;
}
```

(5) 在 TetrisPanel 的构造方法中，调用 newBlock()方法生成当前方块和下一方块的形状与旋转状态。

8. 画出当前方块

(1) 定义画出当前方块的方法。

```
public void paintCurrentBlock(Graphics g) {
            for (int i = 0; i < 16; i++) {
                if (shapes[blockType][turnState][i] == 1) {
                    g.setColor(Color. MAGENTA);
                    g.fillRect((i % 4 + x ) * 25 + 21, (i / 4 + y) * 25 + 16,
                              23, 23);
                    g.setColor(Color.red);
                    g.drawRect((i % 4 + x ) * 25 + 20, (i / 4 + y) * 25 + 15,
                              25, 25);
                }
            }
}
```

(2) 在 paint()方法中调用该方法,画出当前方块。

```
paintCurrentBlock(g);
```

9. 画出下一方块

(1) 定义画出下一个方块的方法。

```
public void paintNextBlock(Graphics g) {
           for (int i = 0; i < 16; i++) {
               if (shapes[nextBlockType][nextTurnState][i] == 1) {
                   g.setColor(Color.BLUE);
                   g.fillRect(i % 4 * 25 + 12 * 25 + 10, i / 4 * 25 + 20,
                              23, 23);
                   g.setColor(Color.red);
                   g.drawRect(i % 4 * 25 + 12 * 25 + 9, i / 4 * 25 + 19,
                              25, 25);
                }
           }
}
```

(2) 在 paint()方法中调用该方法,画出下一方块。

```
paintNextBlock(g);
```

10. 画出已经固定在墙上的方块

(1) 定义画出墙上方块的方法。

```
public void paintWall(Graphics g) {
    for (int i = 0; i < wall.length; i++) {
             for (int j = 0; j < wall[i].length; j++) {
                 if (wall[i][j] == 1) {
                     g.setColor(Color.BLUE);
                     g.fillRect(i * 25 + 21, j * 25 + 16, 23, 23);
                     g.setColor(Color.red);
                     g.drawRect(i * 25 + 20, j * 25 + 15,25, 25);
                 }
             }
    }
}
```

(2) 在 paint()方法中调用该方法,画出已经固定在墙上的方块。

```
paintWall(g);
```

运行该游戏,此时会得到如图 4-8 所示的画出当前方块和下一方块的窗口。

该窗口中虽然画出了当前方块和下一方块,但当前方块是静止的。

那么如何让当前方块动起来呢?

11. 当前方块下移

这里利用 javax.swing.Timer 类实现当前方块下移的效果。

(1) 在 TetrisPane 类中定义定时器变量。

```
Timer timer;
```

(2) 定义定时器监听类 TimerListener(作为 TetrisPanel 类的内部类),实现 ActionListener 接口。

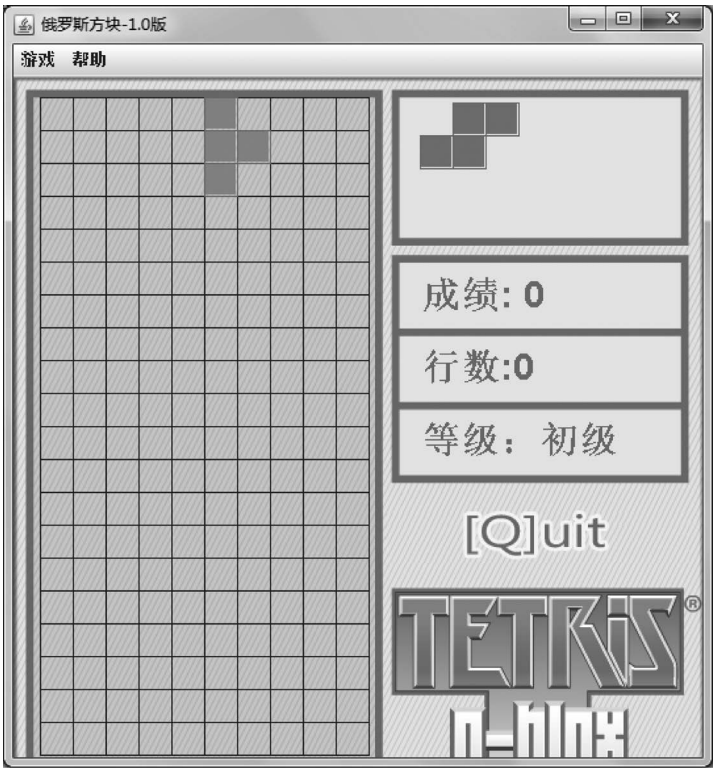

图 4-8　画出当前方块和下一方块的窗口

```
class TimerListener implements ActionListener {
          public void actionPerformed(ActionEvent e) {
            y = y + 1;                          //下落一格
            repaint();                          //屏幕重绘
            }
}
```

(3) 在 TetrisPanel 类的构造方法中创建定时器，每隔 0.5s 触发一次。

```
timer = new Timer(500, new TimerListener());
```

(4) 启动定时器。

```
timer.start();
```

此时运行该程序，发现当前方块能定时向下移动了，但下落到屏幕的最低端时消失了，方块并没有留到游戏墙上。

那么如何控制方块下落的过程呢？

12. 当前方块下落流程设计

分析：当前方块下落前，应先判断还能不能下落，如果当前方块已经移动到区域最下方或是落到其他方块上无法移动时，就应该将当前方块固定在该处(即放到屏幕墙上)；接着判断有没有出现满行，如果有满行，则消除满行并统计得分，然后产生新的方块出现在游戏区域的上方并开始下落。

(1) 判断当前方块能否继续下落。

```
public boolean canDrop() {
            for (int i = 0; i < 4; i++) {
                for (int j = 0; j < 4; j++) {
                    if ((shapes[blockType][turnState][i * 4 + j] == 1
                            && y + i < 19 && wall[x + j][y + i + 1] == 1)
                            ||(shapes[blockType][turnState][i * 4 + j] == 1
                                    && y + i == 19))
                                    //当前方块和墙上的方块重合或到底部
                        return false;
                }
            }
            return true;
}
```

(2) 当前方块不能下落时,将方块固定到墙上。

```
public void landToWall() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (shapes[blockType][turnState][i * 4 + j] == 1)
                    wall[x + j][y + i] = 1;
            }
        }
}
```

(3) 消去满行并得分。

```
public void deleteLine() {
  for (int i = 0; i < 20; i++) {
      int n = 0;
      for (int j = 0; j < 10; j++) {
          if (wall[j][i] == 1)
              n++;
      }
      if (n == 10) {
          score++;                              //统计得分
          //上方的方块下移
          for (int a = i; a > 0; a--) {
              for (int b = 0; b < 10; b++) {
                  wall[b][a] = wall[b][a - 1];
              }
          }
      }
  }
}
```

(4) 修改定时器监听类中的 actionPerformed()方法,控制方块下落的流程,即让方块在能下落的情况下下落一格,如果不可以下落,则固定当前方块,并消去满行,同时产生新的当前方块。

```
public void actionPerformed(ActionEvent e) {
  if (canDrop()) {                              //如果当前方块可以下落
      y = y + 1;
  } else {                                      //固定当前方块,消去满行,产生新的方块
       landToWall();
```

```
            deleteLine();
            newBlock();
        }
        repaint();                                    //屏幕重绘
    }
```


至此，运行该游戏，得到如图 4-9 所示的效果，说明该游戏实现了当前方块下落过程的控制。

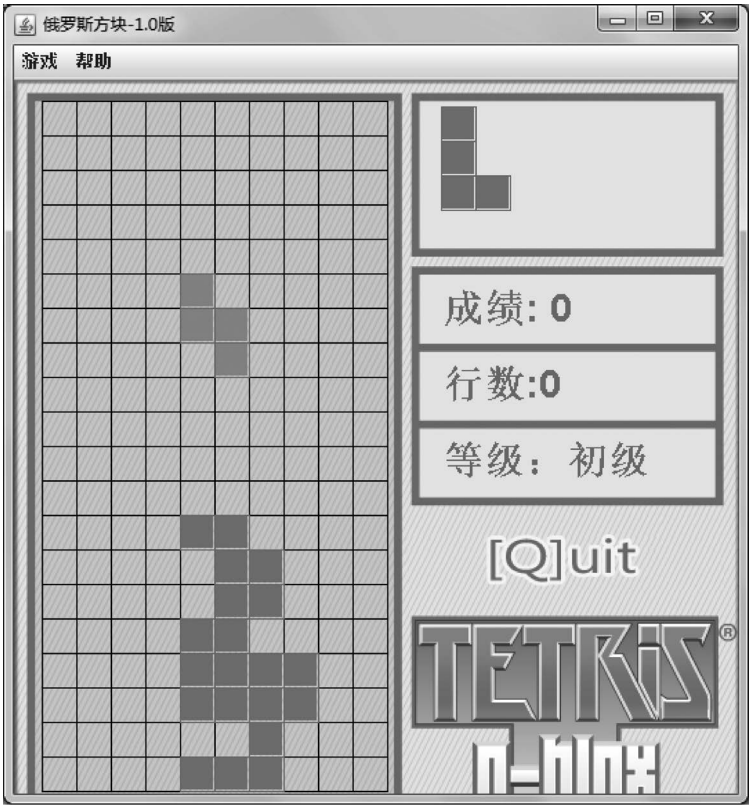

图 4-9　能控制当前方块下落的窗口

现在虽然能控制当前方块的下落过程，但是这种下落只是计时器触发的，用户无法干预。实际的游戏过程应该是用户能通过键盘方向键←、→控制方块左右移动、↑控制方块旋转变形、↓加速下落，以达到消除满行和计分的功能。

13. 当前方块左右移动和旋转控制设计

分析：如果要想用键盘方向键←、→、↑、↓来控制方块的移动，首先游戏面板应实现键盘接口 KeyListener，然后重写该接口中的 keyPressed()方法，实现键盘的如下操作。

方向键↑：方块逆时针旋转 90°。

方向键↓：方块下移一格。

方向键←：方块向左移一格。

方向键→：方块向右移一格。

因为 KeyListener 接口中有 3 种抽象方法，所以另外两种抽象方法虽然没有作用，但也要实现出来。

代码如下：

```
public class TetrisPanel extends JPanel implements KeyListener{
    //其他代码省略
    //重写 keyPressed()方法
    public void keyPressed(KeyEvent e) {
        switch (e.getKeyCode()) {
            case KeyEvent.VK_LEFT:
                x--;
                repaint();
                break;
            case KeyEvent.VK_RIGHT:
                x++;
                repaint();
                break;
            case KeyEvent.VK_UP:
                if (turnState == 3)
                    turnState = 0;
                else
                    turnState++;
                repaint();
                break;
            case KeyEvent.VK_DOWN:
                if (canDrop())
                    y++;
                else {
                    landToWall();
                    deleteLine();
                    newBlock();
                }
                repaint();
                break;
            default:
                break;
            }
    }
    public void keyTyped(KeyEvent e) {

    }
    public void keyReleased(KeyEvent e) {

    }
}
```

这时，TetrisPanel 既是游戏面板，又是键盘监听类，只需要在主窗口的构造方法中添加键盘监听即可。代码如下：

```
this.addKeyListener(tpanel);
```

此时，运行该游戏程序，键盘的上、下、左、右方向键能操控当前方块了。可是还有如下问题存在：当向左、向右移动方块时，如果方块移出游戏墙，则会产生异常；当旋转方块时，也可能由于旋转出界而产生异常。下一步应着手解决这些异常问题。

14. 左移操作控制

当用户想使用左方向键向左移动方块时，应先判断当前方块能否左移，如果能左移，则让方块左移一格，否则不让方块左移。

(1) 判断当前方块能否左移。

```
public boolean canMoveLeft() {
            if (x > 0) {
                for (int i = 0; i < 4; i++) {
                    for (int j = 0; j < 4; j++) {
                        if (shapes[blockType][turnState][i * 4 + j] == 1
                                && wall[x + j - 1][y + i] == 1)
                            return false;
                    }
                }
                return true;
            } else if (x == 0
                  && (shapes[blockType][turnState][0] == 0
                  && shapes[blockType][turnState][4] == 0
                  && shapes[blockType][turnState][8] == 0 && shapes[blockType][turnState][12] == 0))
                return true;
            else
                return false;
}
```

(2) 修改 keyPressed(KeyEvent e)方法。

```
case KeyEvent.VK_LEFT:
    if (canMoveLeft())
        x--;
    repaint();
    break;
```

15. 右移操作控制

当用户想使用右方向键向右移动方块时，应先判断当前方块能否右移，如果能右移，则让方块右移一格，否则不让方块右移。

(1) 判断当前方块能否右移。

```
public boolean canMoveRight() {
            if (x < 8) {
                for (int i = 0; i < 4; i++) {
                    for (int j = 0; j < 4; j++) {
                        if (shapes[blockType][turnState][i * 4 + j] == 1
                                && x + j + 1 < 10 && wall[x + j + 1][y + i] == 1)
                            return false;
                    }
                }
                if (x < 6) {
                    return true;
                } else if (x == 6) {
                    if (shapes[blockType][turnState][3] == 0
                            && shapes[blockType][turnState][7] == 0
                            && shapes[blockType][turnState][11] == 0
                            && shapes[blockType][turnState][15] == 0)
```

```
                    return true;
                else
                    return false;
            } else if (x == 7) {
                if (shapes[blockType][turnState][2] == 0
                        && shapes[blockType][turnState][6] == 0
                        && shapes[blockType][turnState][10] == 0
                        && shapes[blockType][turnState][14] == 0)
                    return true;
                else
                    return false;
            }
        }
        return false;
    }
```

(2) 修改 keyPressed(KeyEvent e)方法。

```
case KeyEvent.VK_RIGHT:
    if (canMoveRight())
        x++;
    repaint();
    break;
```

16. 旋转操作控制

当用户想使用上方向键逆时针旋转方块时,应先判断当前方块能否旋转,如果能旋转,则让方块逆时针旋转 90°;否则不让方块旋转。

(1) 判断当前方块能否旋转。

```
public boolean canTurn() {
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
            if (shapes[blockType][turnState][i * 4 + j] == 1)
                if (x + j < 0 || x + j > 9 || y + i > 19|| wall[x + j][y + i] == 1)
                    return false;
      }
    }
    return true;
}
```

(2) 修改 keyPressed(KeyEvent e)方法。

```
caseKeyEvent.VK_UP:
    int temp = turnState;
    if (turnState == 3)
        turnState = 0;
    else
        turnState++;
    if (canTurn() == false || canDrop() == false)
        turnState = temp;
    repaint();
    break;
```

17. 结束游戏

完成到这一步，该游戏中方块的左右移动、向下移动、旋转动作都基本正常了。但是当游戏墙中存放的方块已到达顶部，新的方块不能进入游戏墙时，该游戏应该正常结束，不应再继续生成新方块，这时应该适时结束游戏。

(1) 判断游戏是否结束。

```
public boolean gameOver() {
    if(y == 0&&!canDrop())
        return true;
    else return false;
}
```

(2) 修改定时器监听类 TimerListener 中的 actionPerformed()方法，增加如下代码：

```
if(gameOver()){
    JOptionPane.showMessageDialog(null,"游戏结束");
    timer.stop();
}
```

至此，游戏中方块的移动、旋转、消行、计分功能都能正常使用了。通过图 4-2、图 4-3 可知，该游戏窗口中还有菜单的使用，下面实现对菜单项的事件处理。

18. 菜单项的事件处理

(1) 定义监听类。

将 Tetris 类实现 ActionListener 接口作为监听类。

代码如下：

```
public class Tetris extends JFrame implements ActionListener{
}
```

(2) 重写 actionPerformed()方法。

```
public void actionPerformed(ActionEvent e) {
            String str = e.getActionCommand();
            if(str.equals("新游戏"))
                tpanel.newGame();
            else if(str.equals("暂停"))
                tpanel.pauseGame();
            else if(str.equals("继续"))
                tpanel.continueGame();
            else if(str.equals("退出"))
                tpanel.quitGame();
            else if(str.equals("初级"))
                tpanel.primaryLevel();
            else if(str.equals("中级"))
                tpanel.intermediateLevel();
            else if(str.equals("高级"))
                tpanel.advancedLevel();
            else if(str.equals("关于"))
                JOptionPane.showMessageDialog(this,"方向键↑:方块逆时针旋转 90 度\n 方向
键↓:方块下移一格\n 方向键←:方块向左移一格\n 方向键→:方块向右移一格\n");
}
```

注意，因为在 TetrisPanel 类中还没有定义如下方法：newGame()、pauseGame()、

continueGame()、quitGame()、primaryLevel()、intermediateLevel()、advancedLevel()，所以上述代码暂时会出现编译错误。

(3) 菜单项注册监听器。

修改 Tetris 类的构造方法，增加如下代码：

```
newgame.addActionListener(this);
pause.addActionListener(this);
goon.addActionListener(this);
quit.addActionListener(this);
primary.addActionListener(this);
intermediate.addActionListener(this);
advanced.addActionListener(this);
about.addActionListener(this);
```

(4) 在 TetrisPanel 类中定义 actionPerformed()方法中调用的各个方法。

修改 TetrisPanel 类，定义如下方法。

① 开始新游戏的方法。

```
public void newGame() {
  //清空游戏墙
  for (int i = 0; i < wall.length; i++)
    for (int j = 0; j < wall[i].length; j++)
        wall[i][j] = 0;
  lines = 0;
  score = 0;
  level = "初级";
  newBlock();                    //生成新方块和下一方块的形状代号和状态号
  timer.setDelay(500);           //设置 Timer 的事件间延迟为 500ms
  timer.start();                 //启动定时器
}
```

② 暂停游戏的方法。

```
public void pauseGame() {
    timer.stop();                //停止定时器
}
```

③ 继续游戏的方法。

```
public void continueGame() {
    timer.restart();             //启动定时器
}
```

④ 退出游戏的方法。

```
public void quitGame() {
    System.exit(0);              //退出系统
}
```

⑤ 设置游戏等级为初级的方法。

```
public void primaryLevel() {
    level = "初级";
    timer.setDelay(500);         //设置 Timer 的事件间延迟为 500ms
}
```

⑥ 设置游戏等级为中级的方法。

```
public void intermediateLevel() {
    level = "中级";
    timer.setDelay(300);          //设置 Timer 的事件间延迟为 300ms
}
```

⑦ 设置游戏等级为高级的方法。

```
public void advancedLevel() {
    level = "高级";
    timer.setDelay(100);          //设置 Timer 的事件间延迟为 100ms
}
```

至此，该《俄罗斯方块》游戏的各项功能都已实现，并且游戏运行正常。

4.7 总　　结

希望读者通过本项目的开发练习，熟练掌握利用 Java Swing 图形用户界面编程、键盘事件的处理过程、菜单事件的处理过程以及多线程编程开发游戏的方法。如果感兴趣，可以提出自己的想法，进一步修改、完善该游戏。