华容道

游戏简介

连连看游华容道游戏取自著名的三国故事,曹操在赤壁大战中被刘备和孙权的"苦肉计"、"火烧连营"打败,被迫退逃到华容道,又遇上诸葛亮的伏兵,关羽为了报答曹操对他的恩情,明逼实让,终于帮助曹操逃出了华容道. 游戏就是依照 "曹瞒兵败走华容,正与关公狭路相逢。只为当初恩义重,放开金锁走蛟龙"这一故事情节,但是这个游戏的起源,却不是一般人认为的是"中国最古老的游戏之一".实际上它的历史可能很短,华容道的现在样式是 1932 年 JohnHarold Fleming 在英国申请的专利,并且还附上横刀立马的解法。

游戏原理

普通模式

首先，游戏区域在Java 程序中看作是一个二维数组对象，游戏区中可以看作是一个容器对象，二维数组中一维的值可以看作是游戏区域中x 坐标的值，二维数组中二维的值可以看作是游戏区域中y 坐标的值，容器根据这个二维数组去构造游戏区域。

其次，当点击游戏区域的某个点时，我们可以通过相关的计算找到该点在二维数组中对应的某个值。 当拖动游戏中的人物方块是，我们也可以经过相关的坐标转换，计算出方块的目的位置以及检测拖动位置的合法性，然后重绘游戏区域。

自动寻路

将华容道游戏中每一个盘面（也即每一个由游戏当前人物方块位置转换的二维数）组视作一个节点，采用广度优先搜索（BFS）的方式，这样第一个找到的解一定是最佳解。为了避免搜索进入循环以致无法继续搜索下去，在每次展开儿子节点时都检查此儿子节点之前有无出现过，仅保存之前没有出现过的儿子节点。另外，在搜索到目标节点后，还要能够找出从初始节点通往目标节点的通路，这是一个回溯的过程。所以，每次由当前节点生成儿子节点时，还要给儿子节点标记上父亲节点的位置。这样就可以从最终找到的目标节点回溯到初始节点，从而找到初始节点与目标节点之间的通路。求解出通路的路径后，按所求解的路径来设置游戏区域中人物方块的位置，达到自动寻路的效果。

变换游戏模式

华容道的布阵方法非常之多，但实质上也就是初始化的时候人物的初始位置以及图片不同，因此采用策略模式，实现华容道游戏的多种场景，

设计过程

创建游戏界面与游戏区域

创建游戏界面

游戏界面如图1所示，使用的是Jpanel

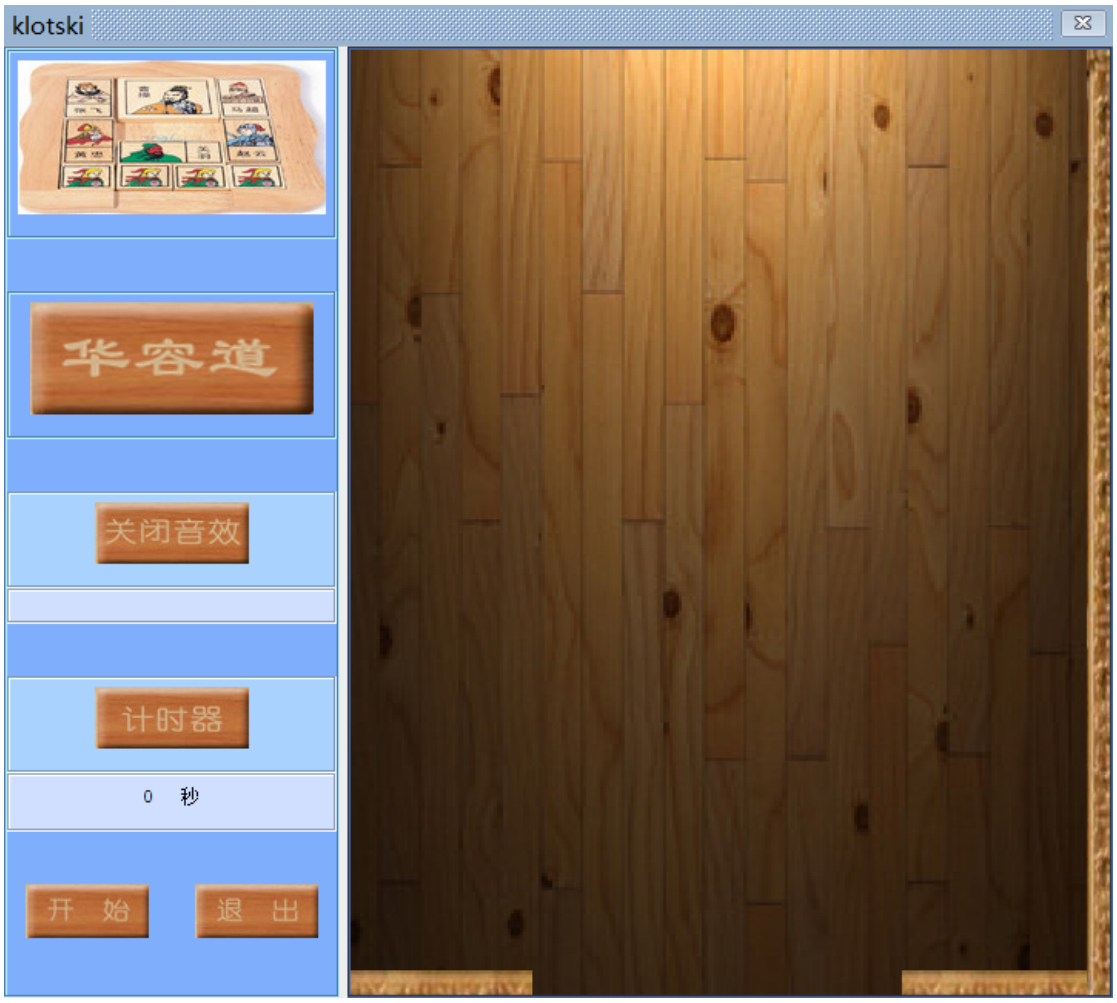

图1

游戏区域实现原理

将游戏区看作是一个坐标。游戏区的最左上角的那一点可以看作是坐标的（0，0）点，游戏区的最上面的边就是 x 轴，游戏区最左面的边就是 y 轴。这样，我们就可以将游戏区域中的图片看作是一个二维数组，数组的一维值是 x 坐标的值，数组的二维值是 y 坐标的值，那么，当游戏中在游戏区域中选择了某一点的时候，我们就可以定位到该数组中的唯一的值，再去判断这个值并执行操作。

创建图片方块对象

首先创建一个二维数组用于表示一个游戏区域的图片，我们所需要的图片方块可以看作是一个具体的对象，因此，我们新建一个`Cell`类，表示为一个图片方块，一个`Cell`对象即一个图片方块，那么它当然会包括一个图片对象，图片占有一定的位置，那么我们需要记录它的开始的点的x 坐标和y 坐标，还有结束点的x 坐标和y 坐标，什么是`Cell`对象的开始坐标和结束坐标如图2 所示。

图2

并为其提供这些属性和方法，如图3所示：

Cell
-image: Image -beginX: int -beginY: int -width: int -height: int -indexPoint: Point = new Point(0, 0)
<<create>>+Cell(image: Image, beginX: int, beginY: int, width: int, height: int) <<create>>+Cell(image: Image, beginX: int, beginY: int) <<create>>+Cell(beginX: int, beginY: int, width: int, height: int) <<create>>+Cell(cell: Cell) +getImage(): Image +setImage(image: Image) +getBeginX(): int +setBeginX(beginX: int) +getBeginY(): int +setBeginY(beginY: int) +setLocation(x: int, y: int) +getLocation(): Point +getWidth(): int +setWidth(width: int) +getHeight(): int +setHeight(height: int) +getIndexPoint(): Point +setIndexPoint(indexPoint: Point) +getXIndex(): int +getYIndex(): int +setXIndex(xIndex: int) +setYIndex(yIndex: int) +setCellXLocation(x: int) +setCellYLocation(y: int) +equals(obj: Object): boolean +intersects(cell: Cell): boolean +intersects1(cell: Cell): boolean

图3

我们为每个属性提供get和set方法，对于每个Cell的坐标

```
// 方块在矩阵中的位置  即Cell的编码
private Point indexPoint=new Point(0, 0);
private int beginX;  // 开始横坐标
private int beginY; // 开始纵坐标
```

创建游戏处理类

由于游戏区域的方块还有其他的信息，例如人物的姓名：关羽、张飞、赵子龙、马超、黄盖、曹操和士兵。因此我们将Cell做进一步封装，作为Person对象的成员变量，如图4所示：

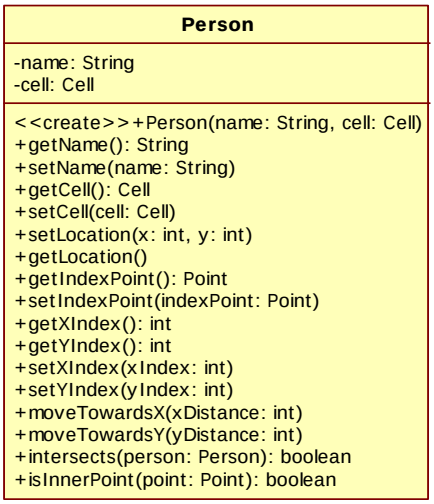

图 4

在游戏中，各种对象最好能直接监听和处理自己的事件(例如点击事件和拖拽事件)，因此我们将Person做进一步封装，作为PersonLabel对象的成员变量，其中PersonLabel继承自JLabel如图5所示：

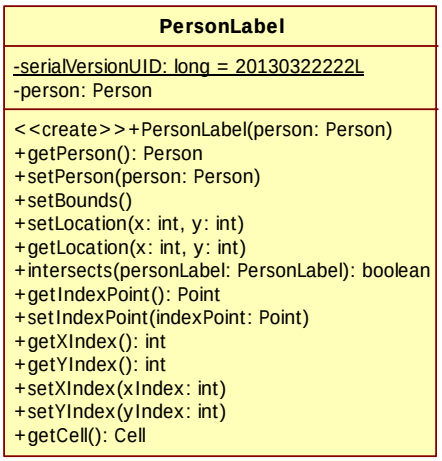

图 5

新建一个 GameService 的接口，用于定义游戏逻辑的接口方法，再为这个接口新增一个实现类GameServiceImpl，将GameService 接口设置到GamePanel 这个视图组件中，这们就可以达到逻辑与视图分离了。

代码清单：Klotski\src\com\games\klotski\service\GameService.java

```
public interface GameService {
    //游戏开始
    void start();
    //判断游戏输赢
    boolean win();
    //根据坐标(x,y)寻找点击的是哪个人物
    Person findPerson(int x,int y);
    //根据配置 config 构造不同的游戏场景
    GameModel createGameModel(GameConfiguration config);
    //检测游戏对象是否越界
    boolean outOfBound(Person person);
}
```

我们需要实现某些游戏逻辑，可以在这个接口里面定义方法，并由 GameServiceImpl 去实现了。这样可以简化视图的代码，也遵循了单一职责的原则。现在我们为GamePanel 类加入一个GameService 的属性，并为它的构造器中加入设置GameService的代码，用于设置GameService，使得在GameService中可以拿到GameService对象，代码简要清单如下。

代码清单：Klotski\src\com\games\klotski\service\impl\GameServiceImpl.java

```
public class GameServiceImpl implements GameService {
    private GameConfiguration config;// 配置
    private GameModel gameModel;// 游戏模式,随机产生
    private long grade = 0;// 加入分数属性,初始值为0
    private boolean win=false;
    public GameServiceImpl(GameConfiguration config) {
        this.config = config;
    }
    public void start() {
        this.gameModel = createGameModel(config);

    }

    public GameModel createGameModel(GameConfiguration config) {
        Random random = new Random();
        switch (random.nextInt(config.getMODELNUM())) {
        case 0:
            return new SimpleGameModelA();
        case 1:
            return new SimpleGameModelB();
        default:
            return null;
        }
    }

    public GameModel getGameModel() {
        return gameModel;
    }
```

需要能够实时的刷新游戏中人物的位置，由于游戏中显示的真正的实体是PersonLabel, 因此可以重写其setLocation、setBounds方法，然后让代表游戏区域的集合`Map<String, PersonLabel>` personLabels (为了方便的以人物名称取得其中每一个人物对象，同时也以Map集合保存他们) 调用，显示人物的的创建与变化，我们可以放到GameService的实现类中去，在GameService 中提供一些 接口方法，GamePanel通过这些方法去获取这个集合，而如何去创建，设置这个集合里面的值，GamePanel不再需要去理会这些过程。

代码清单：Klotski\src\com\games\klotski\ui\PersonLabel.java

```
//依照人物现在的参数设置人物位置
public void setBounds() {
    this.setBounds(this.person.getCell().getBeginX(), this.person.getCell()
            .getBeginY(), this.person.getCell().getWidth(), this.person
            .getCell().getHeight());
    this.setVisible(true);
}

/**
 * 依照指定的参数设置人物位置
 */
public void setLocation(int x, int y) {
    super.setLocation(x, y);
    this.getPerson().setLocation(x, y);
}
```

图片的读取

去创建用于读取图片和处理图片的 ImageUtil 工具类，在为这个工具类添加方法前，我们必须要明确这个类的作用，读取或者处理图片的一些公共方法可以放到这里，做成静态方法给外部使用。现在我们新建一个读取图片的方法，读取某个文件夹下面的符合后缀的图片，我们把方块的图片独立放到一个文件夹，再使用程序读取它们，部分实现代码如下。

```
public class ImageUtils {
    final private static String HEAFIMAGETYPE = ".jpg";
    // 读取默认类型为HEAFIMAGETYPE的图片
    public static BufferedImage getImage(String imageName) {
        try {
            // 使用ImageIO读取图片
            return ImageIO.read(new File(new ConfigDaoImpl().getImagesPath()
                    + File.separator + imageName + HEAFIMAGETYPE));
        } catch (IOException e) {
            // 读取图片发生异常，抛出GameException
            throw new GameException("read image error:no such resource");
        }
    }
    //读取指定类型的图片
    public static BufferedImage getImage(String imageName,String type) {
        try {
            // 使用ImageIO读取图片
            return ImageIO.read(new File(new ConfigDaoImpl().getImagesPath()
                    + File.separator + imageName + type));
        } catch (IOException e) {
            // 读取图片发生异常，抛出GameException
            e.printStackTrace();
            throw new GameException("read image error:no such resource");
        }
    }
```

创建游戏区图片

在画游戏区域之前，我们必须要为游戏区域personLabels进行赋值，将游戏区域看作一个二维数组（这个二维数组是逻辑上的），那么这个数组的每个值的变化，将会映射到到personLabels中相应对象的坐标现，因此这个逻辑二维数组对我们的游戏尤为重要。在BeginListener中mousePressed方法里，用于执行游戏开始时的一些动作，例如初始化游戏区域，重新计时，重新计分等，当玩家点击了开始时，我们就要调用这个start 方法开始游戏，我们可以为开始按钮创建 鼠标监听器。当然，我们这里先讲创建游戏区域。

创建不同的游戏场景

华容道有很多布阵方法，例如横刀立马、层层设防、峰回路转、水泄不通等，为了能在游戏中可以呈现不同的游戏场景，这些场景实质上只是初始位置和人物的图片不同，因此采用策略模式，抽象出GameModel类，其中有各个人物的初始化方法，具体的游戏模式类去继承GameModel并重写其相应方法，代码如下：

代码清单：Klotski\src\com\games\klotski\service\GameModel.java

```
public abstract class GameModel {
    private GameConfiguration gameConfig = new GameConfiguration();
    private CellCreatorImpl cellCreator = new CellCreatorImpl();
    // 存放产生的人物
    private static Map<String, Person> persons
                = new HashMap<String, Person>();
    protected Cell exit = new Cell(120, 360, 240, 240);// 出口
    protected Person guanyu;
    protected Person zhangfei;
    protected Person zhaozilong;
    protected Person machao;
    protected Person huangzhong;
    protected Person caocao;
    protected Person[] samurais;
    protected Person[] blanks;

    // 创建各个人物，包括位置，大小等
    public abstract Person guanyu();

    public abstract Person zhangfei();

    public abstract Person zhaozilong();

    public abstract Person machao();

    public abstract Person huangzhong();

    public abstract Person caocao();
```

具体的游戏模式拥有一个固定布阵方法，继承 GameModel，以其中一个为例

代码清单：Klotski\src\com\games\klotski\service\impl\SimpleGameModelA.java

```
public Person guanyu() {
    if (this.guanyu == null) {
        String name = "guanyu";
        Cell cell = getCellCreator().createCell( getGameConfig().getBaseImageLength(),
                2* getGameConfig().getBaseImageLength(), name);
        this.guanyu = new Person(name, cell);
        PersonUtils.countIndex(this.guanyu, getGameConfig().getBaseImageLength());
        GameModel.addPerson(this.guanyu);
    }

    return this.guanyu;

}

public Person zhangfei() {
    if (this.zhangfei == null) {
        String name = "zhangfei";
        Cell cell = getCellCreator().createCell(0,
                0, name);
        this.zhangfei = new Person(name, cell);
        PersonUtils.countIndex(this.zhangfei, getGameConfig().getBaseImageLength());
        GameModel.addPerson(this.zhangfei);
    }
    return this.zhangfei;
}
```

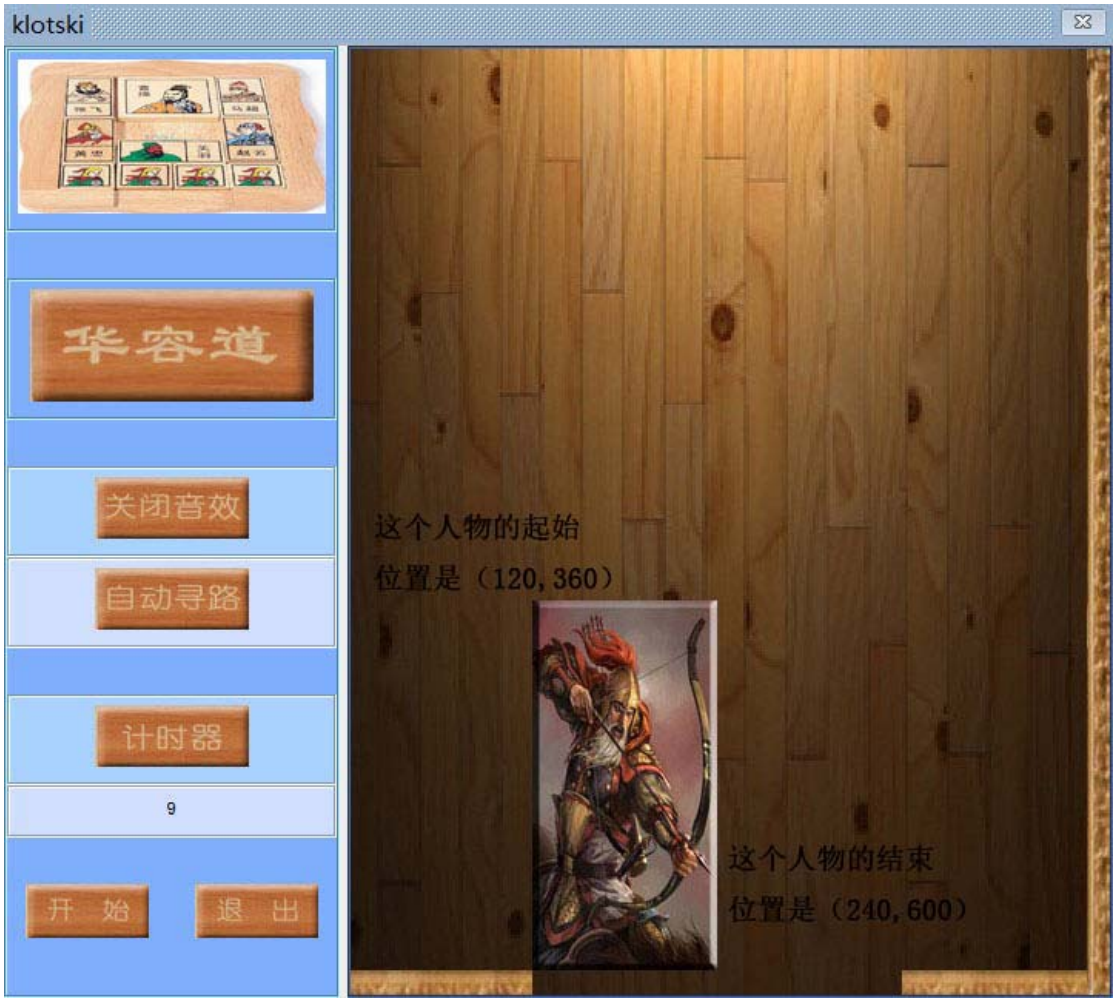

图 6

设置一个监听游戏开始或者结束的监听器BeginListener，在其mousePressed中方法设置如下初始化各种对象初始状态的代码：

代码清单：Klotski\src\com\games\klotski\listener\BeginListener.java

```
public void mousePressed(MouseEvent e) {
    mainFrame.romoveBackGroundPanel();
    // 重复的按开始按钮：首先必须清除上面的组件
    gamePanel.removeAll();
    mainFrame.add(gamePanel, BorderLayout.CENTER);
    if (this.timer != null) {
        this.timer.cancel();
    }

    this.timer = new Timer();
    gamePanel.setOverImage(null);
    timeLabel.setText(String.valueOf(config.getLimitedTime()));
    gameService.setWin(false);// 设置状态为未赢
    gameService.start();

    autoFindPath.setVisible(true);//游戏开始后自动寻路 设置为可见 可操作
    autoFindPath.setEnabled(true);
    autoFindPath.changeImage("autoFindOnIn", "autoFindOnOut");//重新开始
    personLabels = new HashMap<String,PersonLabel>();
    for (Entry<String, Person> person : GameModel.getPersons().entrySet()) {
        // 给"caocao"加监听：判断游戏是否结束
        if (person.getValue().getName().equals("caocao")) {
            gameListener = new GameListener(gameService,
                    gamePanel, this);
            gameListener.setExit(gameService.getGameModel().getExit());
            PersonLabel personLabel = new PersonLabel(person.getValue());
            personLabel.addMouseListener(gameListener);
            personLabels.put(person.getValue().getName(),personLabel);
        } else {
            personLabels.put(person.getValue().getName(),new PersonLabel(person.getValue()));
            }
        }
        gamePanel.setPersonLabels(personLabels);
        // 初始化
        gamePanel.initPersonLabel();
        //计时器
        task = new GameTimer( 0,this.timeLabel);
        timer.schedule(task, 0, 1000);
        gamePanel.repaint();
    }
```

游戏区域的绘制

根据 personLabels 绘制游戏区域，遇到图片为 null 的对象，则不绘制。由于 PersonLabel 继承自 JLabel，只要设置 PersonLabel 对象的坐标，便自动绘制。

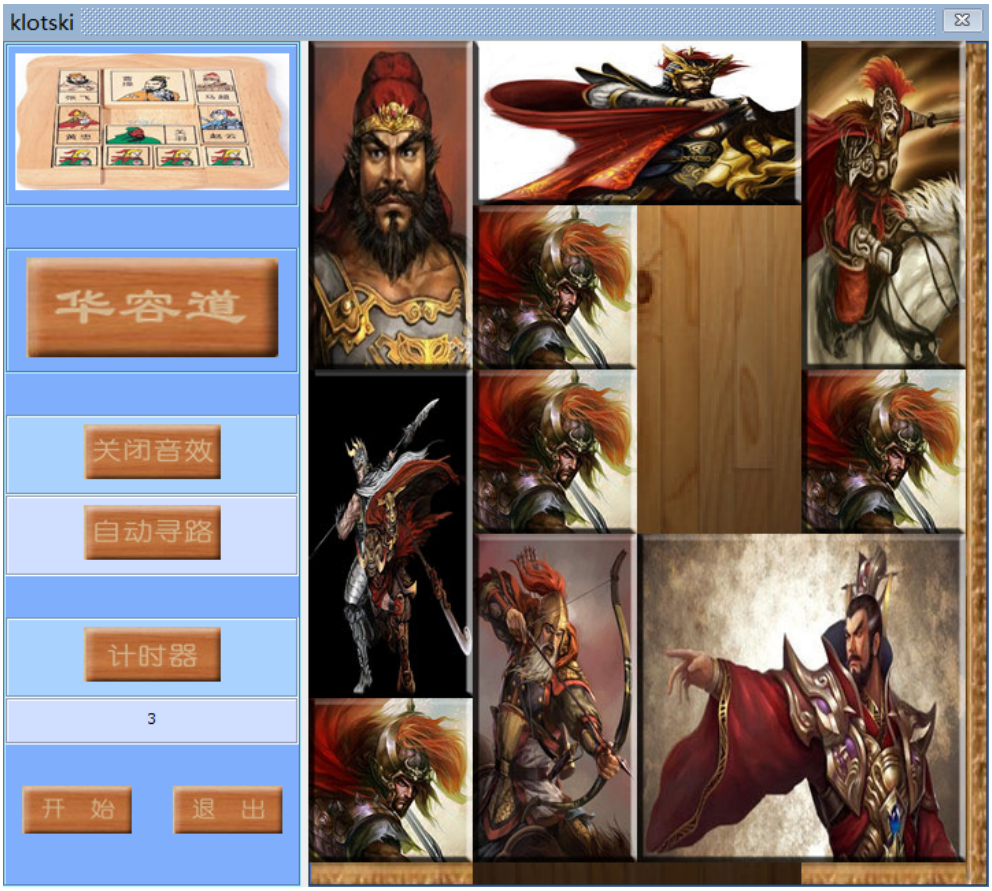

图 7

实现游戏业务逻辑

人物的拖拽

在 Swing 中拖动一个组件或者图片，抛开定位布局不说，是件很容易的事情。拖动图片和拖动组件是相同的道理，如果打算拖动一个组件，只要给组件增加两个监听器就可以了。一个是 addMouseListener(MouseListener e)，另一个是：addMouseMotionListener(MouseMotionListener e) ;由于 MouseListener 和 MouseMotionListener 都继承同一个父类 EventListener，所以我们给组件增加监听器时，只要增加同一个监听器就可以了。即

```
DragPicListener listener = new DragPicListener();
component.addMouseListener(listener);
component.addMouseMotionListener(listener);
```

对于监听器DragPicListener，我们只要实现鼠标按下时的触发事件

```
public void mousePressed(MouseEvent e);
```

和鼠标拖动时的触发事件：

```
public void mouseDragged(MouseEvent e);
```

就可以完成组件的拖动了。当鼠标按下时我们记录下鼠标的位置；当鼠标移动时，获得鼠标当前的位置，并把组件移动到当前位置。同时，如果玩家拖动组件越界、与其它组件重叠，这种拖动应该是无效的，为此，将程序拖动的的距离分解为x和y两个方向，在那个方向的拖动无效，则将其分量的增量置为零，这样可以得到一个比较好的游戏效果，程序部分清单如下：

代码清单：Klotski\src\com\games\klotski\listener\DragListener.java

```
public class DragListener extends MouseInputAdapter {
    /** 坐标点 */
    private Point point = new Point(0, 0);
    private PersonLabel dragLabel;// 被监听的对象
    private GameServiceImpl gameService;

    public DragListener(PersonLabel dragLabel, GameServiceImpl gameService) {
        this.dragLabel = dragLabel;
        this.gameService = gameService;
    }

    /**
     * 当鼠标拖动时触发该事件。 记录下鼠标按下(开始拖动)的位置。
     */
    public void mouseDragged(MouseEvent e) {
        if (!gameService.win()) {// 还没有赢
            // 转换坐标系统
            Point newPoint = SwingUtilities.convertPoint(dragLabel,
                    e.getPoint(), dragLabel.getParent());

            int dx = newPoint.x - point.x;
            int dy = newPoint.y - point.y;

            // 如果移动的地方有方块,则不移动,分为x和y两个方向讨论
            dragLabel.setLocation(dragLabel.getX() + dx, dragLabel.getY());
            if (intersects(dragLabel.getPerson())) {
                dragLabel.setLocation(dragLabel.getX() - dx, dragLabel.getY());
            }
```

```
            dragLabel.setLocation(dragLabel.getX(), dragLabel.getY() + dy);
            if (intersects(dragLabel.getPerson())) {
                dragLabel.setLocation(dragLabel.getX(), dragLabel.getY() - dy);
            }
            // 设置标签的新位置
            // dragLabel.setLocation(dragLabel.getX() + dx, dragLabel.getY() +
            // dy);
            // 更改坐标点
            PersonUtils.countIndex(dragLabel.getPerson(), gameService
                    .getGameModel().getGameConfig().getBaseImageLength());
            point = newPoint;
            PersonUtils.to2DArray(GameModel.getPersons(),
                    gameService.getGameModel().getGameConfig()
                            .getBaseImageLength(), 5, 4);
        }
        public void mousePressed(MouseEvent e) {
            if (!gameService.win()) {
                // 得到当前坐标点
                point = SwingUtilities.convertPoint(dragLabel, e.getPoint(),
                        dragLabel.getParent());
                MusicService.play(dragLabel.getPerson().getName());
            }

        }
```

加入计时功能与游戏的胜利判断、音效、计时

游戏开始开始计时，就算游戏胜利所用的时间，因此 BeginListener 类与游戏区的监听类 GameListener 都必须可以控制时间。

先编写一个 TimerTask 类，用于定时执行任务。

代码清单：Klotski\src\com\games\klotski\timer\GameTimer.java

```
public class GameTimer extends TimerTask {
    // 当前用掉的时候
    private long time;
    // 游戏设定的时间
    private static long gameTime;

    public void run() {
        // 游戏时间
        this.timeLabel.setText(String.valueOf(GameTimer.gameTime + this.time));
        this.timeLabel.repaint();
        this.time += 1;
    }

    public GameTimer(long gameTime, JLabel timeLabel) {
        this.time = 0;
        GameTimer.gameTime = gameTime;
        this.timeLabel = timeLabel;
    }
```

该类继承java.util.TimerTask，实现run 方法，表示需要执行的动作。在TimerTask 的构造器中，需要将gameTime，timeLabel 作为参数传入，首先timeLabel 是时间显示的的一个JLabel，因此必须加入，gameTime 我们保存在GameConfiguration 对象中，gamePanel 中保存了游戏时间计算的标准。另外再为TimerTask 类添加一个用掉的时间属性，用于记录游戏已经用掉的时间，并 在run方法中加一，表示run 每执行一次，time的值加一。

对于音效设置，播放背景音效用MusicUtils，但是有以下要点：

1) 由于需要很好的用户体验，音效应该可以选择关闭或者打开，但是这样操作不应该在一个相对底层的MusicUtils中进行，因为这是一个公用的工具类，如果直接在这个类中进行控制，必定影响其它对象要使用这个类的功能，因此应设置中间代理，音效的播放控制在代理层实现。
2) 游戏的音效非常多（随着项目的不断完善和提升，音效元素将会更多），因此不便在初始的时候就全部载入内存；同时由于要频繁的使用音效，因此，不能使用一次就读一次音效文件；应该注意到，很多音效是可以共享的；因此，解决的策略是改进享元模式：用static Map<String, AudioClip> *sounds* 存储加载到内存的音效，要播放音效时，先在sounds中查找，如果有就播放，如果没有找到，再从内存中读取，并家加入到sounds集合中，具体代码如下：

代码清单：Klotski\src\com\games\klotski\util\MusicUtils.java

```
public class MusicUtils {
    static Map<String, AudioClip> sounds = new HashMap<String, AudioClip>();

    @SuppressWarnings("deprecation")
    public static void play(String name) {
        AudioClip sound = sounds.get(name);
        if (sound == null) {
            try {
                sound = Applet.newAudioClip((new File("resource"
                        + File.separator + "sounds" + File.separator + name
                        + ".wav")).toURL());
                sounds.put(name, sound);
            } catch (MalformedURLException e) {
                e.printStackTrace();
                throw new GameException("GameException:no such resource:"
                        + name);
            }
        }
        sound.play();
    }
}
```

代理层代码如下

代码清单：Klotski\src\com\games\klotski\service\MusicService.java

```
public class MusicService {
    /**
     * 根据GameConfiguration中isPlayMusic()返回值播放音乐，
     * isPlayMusic()返回true就播放,否则不播放
     *
     * @param name
     *            要播放的的音乐名称
     *
     */
    public static void play(String name) {
        if (true == GameConfiguration.isPlayMusic()) {
            MusicUtils.play(name);
        }
    }
}
```

游戏的自动寻路

自动寻路的原理：将游戏中每一个盘面视作一个节点，采用广度优先搜索（BFS）的方式，这　样第一个找到的解一定是最佳解。为了避免搜索进入循环以致无法继续搜索下去，在每次展开儿子节点时都检查此儿子节点之前有无出现过，仅保存之前没有出现过的儿子节点。另外，在搜索到目标节点后，还要能够找出从初始节点通往目标节点的通路，这是一个回溯的过程。所以，每次由当前节点生成儿子节点时，还要给儿子节点标记上父亲节点的位置。这样就可以从最终找到的目标节点回溯到初始节点，从而找　到初始节点与目标节点之间的通路。

从原理中我们可以看出，要自动寻路，是用到了深层的递归，需要分配大量的空间，因此，不可能直接用 PersonLabel、Person、Cell 等其中任何一个对象；另外，对时间也有较高的效率要求，因此，我们采用 JNI 用 C++实现自动寻路算法，而且将棋盘抽象成一个数组，以节约空间。如图 8 所示，将游戏区域以方块划分，然后给每个人物一个数值，从而抽象出一个数组，将其作为参数传入自动寻路的的函数中，经过处理得到路径后，返回路径，其格式是 int[][2] pthInt，pthInt[0]表示是哪个数字（映射到哪个方块）移动，pathInt[1]表示移动的方向：数字 1 表示向上移动一格，数字 2 向下移动一格，数字 3 向左移动一格，数字 4 向右移动一格。

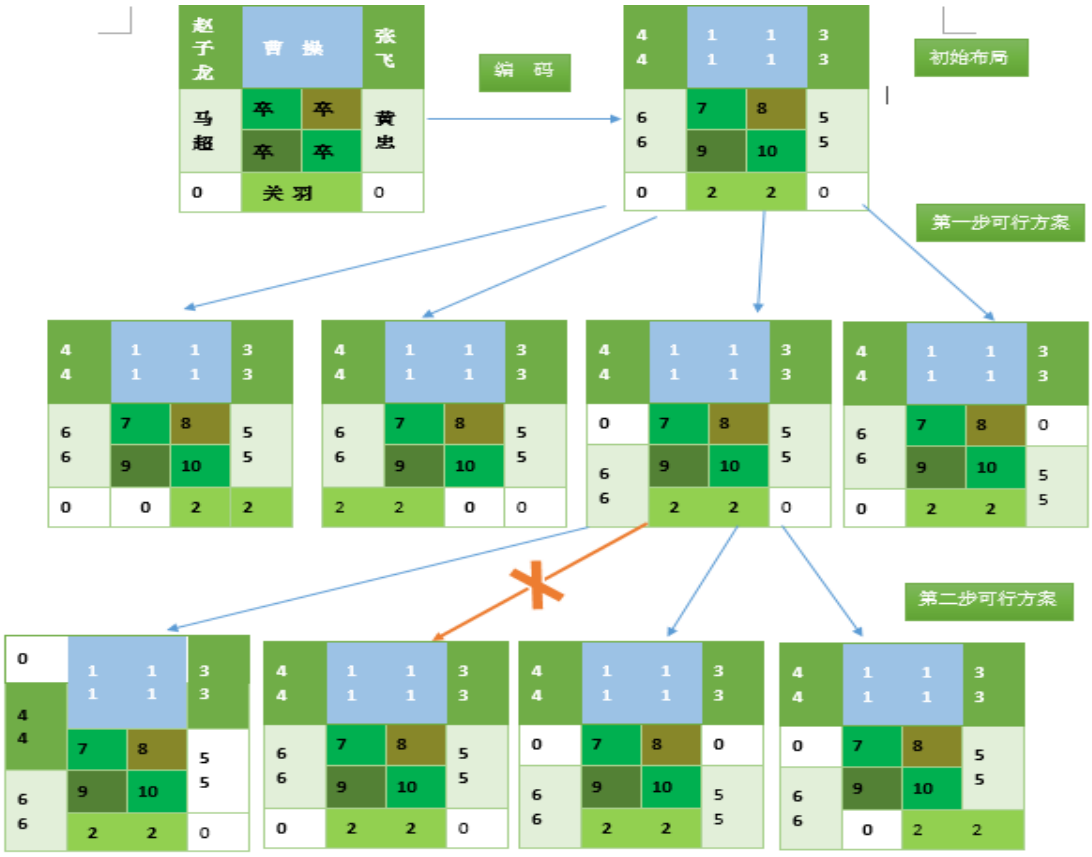

图 8

代码清单：Klotski\src\com\games\klotski\dao\DataProcessUtils.java

```
public class DataProcessUtils {
    private static final int CL = 2;
    /**
     *
     * @param presentPersonsPosition
     *            前人物的坐标
     * @return 一条通路
     */
    public static native Object[] getPath(int[][] presentPersonsPosition);

    public static int[][] autoFindPath(int[][] presentPersonsPosition) {
         Object []pathObj=getPath(presentPersonsPosition);
        // 用枚举应该更好
                // 1上,2下,3左,4右
        int length = pathObj.length / 2;
        int[][] pathInt = new int[length][CL];
        // 将c中返回的一维数组转化为二维整型数组
        for (int columnIndex = 0; columnIndex < CL; columnIndex++) {
            for (int rowIndex = 0; rowIndex < length; rowIndex++) {
                pathInt[rowIndex][columnIndex] = Integer.parseInt(
                        (pathObj[rowIndex * CL+ columnIndex].toString()));
            }
        }
        for(int i=0;i<pathInt.length/2;i++){//因为传过来的要为倒序
            int[] temp=pathInt[i];
            pathInt[i]=pathInt[length-i-1];
            pathInt[length-i-1]=temp;
        }
```

```
        return pathInt;

    }

    // 加载动态链接库
    static {
        System.loadLibrary("msvcp110d");
        System.loadLibrary("msvcr110d");
        System.loadLibrary("DataProcessUtils");
    }
}
```

将游戏区域转化为二维数组代码

代码清单：Klotski\src\com\games\klotski\util\PersonUtils.java

```
public class PersonUtils {
    public static int[][] to2DArray(Map<String, Person> persons,
            int baseLength, int rows, int columns) {
        int[][] positionArray = new int[rows][columns];
        for (Entry<String, Person> person : persons.entrySet()) {
            switch (person.getKey()) {

            case "zhangfei":
                setPerIndex(positionArray, person.getValue(), baseLength, 3);
                break;
            case "guanyu":
                setPerIndex(positionArray, person.getValue(), baseLength, 2);
                break;
            case "zhaozilong":
                setPerIndex(positionArray, person.getValue(), baseLength, 4);
                break;
            case "machao":
                setPerIndex(positionArray, person.getValue(), baseLength, 6);
                break;
            case "huangzhong":
                setPerIndex(positionArray, person.getValue(), baseLength, 5);
                break;
            case "caocao":
                setPerIndex(positionArray, person.getValue(), baseLength, 1);
                break;
            case "samurai0":
                setPerIndex(positionArray, person.getValue(), baseLength, 7);
                break;
            case "samurai1":
                setPerIndex(positionArray, person.getValue(), baseLength, 8);
                break;
            case "samurai2":
                setPerIndex(positionArray, person.getValue(), baseLength, 9);
                break;
            case "samurai3":
                setPerIndex(positionArray, person.getValue(), baseLength, 10);
                break;
            default:
                break;
            }
        }
        return positionArray;

    }
```

小结

1. 代码还可以进行重构优化，例如将每个图形界面组件做成单独中的类，再通过工厂模式或者Java 的反射去获取这些类，用于去创建游戏的界面。
2. 在节点开展程度较深时（如100层以上），改用深度优先搜索以节约搜索时间和存储空间。
3. 改进判断重复的方法：华容道棋盘左右对称，如果开局时的棋子布局也左右对称，可将左右对称的节点看作是重复节点不予展开，从而大为减少搜索在时间和空间上的消耗。
4. 优化存储：可以将每个盘面进一步缩减到更小存储格式中去；优化解空间的存储。
5. 由于顾及游戏区域的绘制效率和便捷，封装太过于复杂；游戏区域转化为二维数组时的效率低下。
6. 界面应该有待美化，游戏体验有待提高。