英语依存树编辑器的设计与实现 作业报告

1200012863 刘敏行（组长） 1200012768 陈庆英

需求分析、设计、实现、软件使用说明、作业评价与总结、成员分工与贡献

一. 需求分析

呈现在用户面前的依存树应该含有如下几部分：

（1） 表示单词的点；
（2） 表示两个单词之间关系的有向线；
（3） 有向线上的对关系的说明。

应该实现的用户对依存树可进行的编辑有：

（1） 在两个单词之间添加一条有向线；
（2） 删除两个单词之间的一条有向线；
（3） 修改一条有向线的关系说明；
（4） 修改有向线的位置；
（5） 修改表示单词的点和表示关系的线的颜色，这可以满足当点和线（特别是线）比较多时用不同颜色使得界面更友好些，但并不是必要的。

用户针对应用本身的需求：

（1）应用美观简洁
（2）操作简单易用，不会混淆
（3）对于一些非常规操作应该有提醒，同时不会出错

二. 设计

针对依存树的显示内容要求，做如下设计：

（1） 根据要显示的点的个数，计算出每个点相应的在画板上的显示位置，以每个位置为圆心、适当的距离为半径画一个圆，并填充该圆即可实现单词的表示。可以设计一个表示这样的点的类；
（2） 用一条圆弧表示两个单词之间的依存关系，圆弧的两个端点是这表示两个单词的相应的点，应该在圆弧的中点添加一个箭头。可以设计一个表示弧线的类；
（3） 在弧线上的箭头附近显示对依存关系进行说明的文字，这个说明可以作为弧线类的一个成员变量。

针对实现的用户对依存树可进行的编辑，做如下设计：

（1） 要在两个点之间添加一条有向线，必须选够两个点，可以设计一个类表示当前已做选择的状态。每当有一个点被选中，就会产生一个事件，当被选的点的个数为 2 时，弹出一个输入对话框，由用户输入一段文字表示这两个点的依存关系；
（2） 要实现对弧线的编辑（即包括：删除两个单词之间的一条有向线，修改一条有向线的关系说明，修改有向线的位置，修改表示关系的线的颜色），可以这样设计：用户点击弧线上的箭头，表示选中该弧线，然后做相应处理。

在选择点、箭头时，都涉及对某个点是否被选中的判断，可以这样实现，当

用户在画板上点击时，获取点击位置，对每个点判断点击位置与该点或箭头的距离是否小于某个值，若是，则判定该点或箭头被选中，而且只要确定有一个点或箭头被选中，便不再判断其它的点和箭头是否被选中。

针对用户对操作和界面方面的需求，设计如下：
（1）界面分菜单栏、菜单工具栏、依存树编辑画板、文本框四部分，从上到下按比例依次放置
（2）对于每一个需要引发事件响应的组件，编写对应的相应函数
（3）对于一些非常规的操作，需要正确处理并弹出提示框。

三. 实现

我们总共有 6 个 java 文件，其中 DrawArc.java（主类）和 Sentence.java 由刘敏行编写完成，Dot.java, ArcArrow.java, DrawMonitor.java, TwoDot.java 由陈庆英编写完成，下面是详细介绍：

【from 陈庆英】

类 Dot 是一个表点（单词）的类，关键方法有：

（1）判断点是否被点击了：

```
public boolean dotSelected(Point selectedPoint)
{
    double dist = Math.sqrt(
                  (Math.pow(selectedPoint.getX() - location.getX(), 2) +
                  Math.pow(selectedPoint.getY() - location.getY(), 2)) / 4);
    return dist <= size;
}
```

location 是一个表示点位置的成员，类型为 Point；
size 是表示点大小的成员；

（2） 在画板上画点：

```
public void drawDot(Graphics2D g2, String label)
{
    Color refColor = g2.getColor();
    Color tempColor = new Color(refColor.getRed(), refColor.getGreen(), refColor.getBlue());
    g2.setColor(color);
    g2.fillOval((int)location.getX() - size, (int)location.getY() - size,   size * 2, size * 2);
    g2.setColor(tempColor);
    int len = label.length();
    if (!TOO_MANY_DOT)
        g2.drawString(label, (int)location.getX() - len * OFFSET,
            (int)location.getY() + DISTANCE);
    else
        g2.drawString(label, (int)location.getX() - len * OFFSET,
            (int)location.getY() + DISTANCE + (dotIndex % 2) * DOWN_DIST);
}
```

label 是对依存关系做说明的字符串；
color 是表示点的颜色的成员；
TWO_MANY_DOT 是 Dot 类的静态成员，表示在画板上的点是否大于一定数量，//若是，则将单词分两行显示；

类 ArcArrow 是一个表示弧线的类，关键方法有：

（1） 判断箭头被选中：

```java
public boolean arrowSelected(Point selectedPoint)
{
    double dist = Math.sqrt(
                  (Math.pow(selectedPoint.getX() - pos3.getX(), 2) +
                  Math.pow(selectedPoint.getY() - pos3.getY(), 2)) / 4);
    return dist <= ARROW_SIZE;
}
```

pos3 就是箭头的位置；
selectePoint 是鼠标点击的位置。

（2） 画弧线：

```java
public void drawArcArrow(Graphics2D g2, String relation)
{
    Color refColor = g2.getColor();
    Color tempColor = new Color(refColor.getRed(), refColor.getGreen(),refColor.getBlue());
    Font tempFont = new Font(g2.getFont().getName(),
        g2.getFont().getStyle(), g2.getFont().getSize());
    if (arcChangeSelectedState)
        g2.setFont(new Font("Arc", Font.BOLD, 30));
    g2.setColor(color);
    g2.draw(arc);
    GeneralPath triangle = new GeneralPath();
    if (direction == RIGHT)
    {
        triangle.moveTo(pos3.getX() - ARROW_SIZE, pos3.getY() + ARROW_SIZE);
        triangle.lineTo(pos3.getX() - ARROW_SIZE, pos3.getY() - ARROW_SIZE);
        triangle.lineTo(pos3.getX() + ARROW_SIZE * 2, pos3.getY());
    }
    else
    {
        triangle.moveTo(pos3.getX() + ARROW_SIZE, pos3.getY() + ARROW_SIZE);
        triangle.lineTo(pos3.getX() + ARROW_SIZE, pos3.getY() - ARROW_SIZE);
        triangle.lineTo(pos3.getX() - ARROW_SIZE * 2, pos3.getY());
    }
    triangle.closePath();
    g2.fill(triangle);
    g2.drawString(relation, (int)pos3.getX() - relation.length() * OFFSET,
        (int)pos3.getY() - DISTANCE);
    g2.setColor(tempColor);
    g2.setFont(tempFont);
}
```

relation 是是对依存关系做说明的字符串；
箭头画在弧线的中点（即代码中的 pos3）；
在画弧线过程中，要备份一下 Graphics2D 的参数，画完一条弧线之后，将其复原。

（3） 计算弧线：

这个方法是在箭头被拖动时调用，用来计算弧线的相关参数。

```
private void calcArcByPos3(Point selectedPoint)
{
    if (selectedPoint.getY() < LEAST_HEIGHT)
        pos3.setLocation(pos3.getX(), LEAST_HEIGHT);
    else if (selectedPoint.getY() < pos1.getY())
        pos3.setLocation(pos3.getX(), selectedPoint.getY());

    midPoint.setLocation((pos1.getX() + pos2.getX()) / 2,
        (pos1.getY() + pos2.getY()) / 2);

    double distPow2 = (Math.pow(pos1.getX() - pos2.getX(), 2) +
                Math.pow(pos1.getY() - pos2.getY(), 2)) / 4;
    double lenPow2 = Math.pow(pos3.getX() - midPoint.getX(), 2) +
                Math.pow(pos3.getY() - midPoint.getY(), 2);
    double len = Math.sqrt(lenPow2);

    radius = (distPow2 + lenPow2) / (2 * len);

    // fit for p1.x == p2.x
    double centerX = pos3.getX();
    double centerY = pos3.getY() + radius;
    center.setLocation(centerX, centerY);
    arc.setArcByCenter(centerX, centerY, radius, 0, 0, 0);
```

```
    if (direction == RIGHT)
        arc.setAngles(pos2, pos1);
    else
        arc.setAngles(pos1, pos2);

    //calc extent
    double sameSide = (centerY - midPoint.getY()) * (centerY - pos3.getY());
    if (sameSide > 0)// pos3 and midPoint are at the same side of center
        extent = Math.asin(Math.sqrt(distPow2) / radius);
    else if (sameSide < 0)
        extent = Math.PI - Math.asin(Math.sqrt(distPow2) / radius);
}
```

selectPoint 是鼠标拖动时的位置；
direction 表示弧线的指向；

类 TwoDot 是一个表示选中状态的类，关键方法有：
（1）当有一个点被选中时，调用下面的方法，更新状态：

```
public void addOneDot(Dot dot)
{
    if (state == NO_DOT || state == TWO_DOT)
    {
        firstDot = dot;
        firstDot.setColor(Dot.SELECTED_COLOR);
        firstDot.setSelectedState(true);
        state = ONE_DOT;
    }
    else if (state == ONE_DOT)
    {
        if (dot == firstDot)
        {
            firstDot.setToMyNormalColor();
            firstDot.setSelectedState(false);
            state = NO_DOT;
            return;
        }
        secondDot = dot;
        secondDot.setColor(Dot.SELECTED_COLOR);
        secondDot.setSelectedState(true);
        state = TWO_DOT;
    }
}
```

(2) 当有两个点被选中且做了相应处理之后就要恢复没有点被选的状态；还有当选中一个点之后，又去选一个弧线时，也要将状态恢复到恢复没有点被选的状态。

```
public void resetState()
{
    if (state > NO_DOT)
    {
        firstDot.setToMyNormalColor();
        firstDot.setSelectedState(false);
    }
    if (state > ONE_DOT)
    {
        secondDot.setToMyNormalColor();
        secondDot.setSelectedState(false);
    }
    state = NO_DOT;
}
```

类 DrawMonitor 是一个继承自 JPanel 的类，用作画板。该类注册了 MouseListener 和 MouseMotionListener 两个监听器，当用户在画板上点击时，DrawMonitor 将做出合理的事件处理。该类的关键方法有：

（1） 重写的 paint（）方法：

```
public void paint(Graphics g)
{
    super.paint(g);
    Graphics2D g2 = (Graphics2D)g;
    for (int index = 0; index < dotVector.size(); ++index)
        dotVector.elementAt(index).drawDot(g2, sentence.getLabel(index));
    for (int index = 0; index < sentence.getNodeNum(); ++index)
        for (int index1 = 0; index1 < sentence.getNodeNum(); ++index1)
        {
            if (!(sentence.getString(index, index1).equals("")))
            {
                arcVector[index][index1].setColor(sentence.getColor(index, index1));
                arcVector[index][index1].drawArcArrow(g2, sentence.getString(index, index1));
            }
        }
}
```

dotVector 保存的是点的引用，类型是 Vector<Dot>；
arcVector 保存的是弧线的引用，类型为 ArcArrow[][];
sentence 就是与当前要画的依存数的句子。

（2） 当用户在画板上点击时，下面的方法被调用：

public void mouseClicked(MouseEvent e)；

在该方法中，判断表示单词的某个 Dot 是否被点击了，只要确定有某一个 Dot 被点击，便不再判断是否有其它的 Dot 被点击：

```
for (int index = 0; index < dotVector.size(); ++index)
{   //warn: make sure that no more than one dot can be selected
    Dot tempDot = dotVector.elementAt(index);
    if (tempDot.dotSelected(selectedPoint))
    {
        if (lastSelectedArc != null)
            lastSelectedArc.setArcChangeSelectedState(false);
        twoDot.addOneDot(tempDot);
```

```
        if (twoDot.getState() == TwoDot.TWO_DOT)
        {

            String relation = JOptionPane.showInputDialog(this,
                "use a string to describe the relation:", "Input a string",
                JOptionPane.PLAIN_MESSAGE);
            while (relation == null || relation.equals(""))
            {
                int option = JOptionPane.showConfirmDialog(this,
                    "Don't want to add an arc?", "Are you sure?",
                    JOptionPane.YES_NO_OPTION);
                if (option == JOptionPane.NO_OPTION)
                    break;
                relation = JOptionPane.showInputDialog(this,
                "use a string to describe the relation:", "Input a string",
                JOptionPane.PLAIN_MESSAGE);
            }

            if (relation == null)
                relation = "";
            sentence.setline(twoDot.getFirstDot().getDotIndex(),
                twoDot.getSecondDot().getDotIndex(),
                relation);
```

```
            if (sentence.CheckTree() == false) {
                JOptionPane.showMessageDialog(this,
                        "Warning: it is not a tree now!");
            }
            if (relation.equals("") == false)
                showOperation("add an arc");

            da.TreeModified=true;

            sentence.clearBack();
            sentence.saveMatrix();

            da.undoItem.setEnabled(true);
            da.unDoButton.setEnabled(true);
            da.redoItem.setEnabled(false);
            da.reDoButton.setEnabled(false);
            twoDot.resetState();
        }
        repaint();
        return;
    }
}
```

如果没有 Dot 被点击，就去判断是否有弧线上的箭头被选中，处理方法与上面的处理类似，不再将代码贴出。

（3） 除了重写 public void mouseClicked(MouseEvent e);方法，还要重写如下方法：

public void mousePressed(MouseEvent e)；

public void mouseReleased(MouseEvent e)；

public void mouseDragged(MouseEvent e)；

public void mouseEntered(MouseEvent e) ；

public void mouseExited(MouseEvent e)；

public void mouseMoved(MouseEvent e)；

重写的内容与 public void mouseClicked(MouseEvent e);的类似。

【from 刘敏行】

我主要实现了主类 DrawArc 和从类 Sentence。

在主类中我主要需要搭建好应用程序的基本框架，设置基本的界面布局以及各个组件对应的监听事件的行为。从类则负责把一个分析的句子涉及的各个信息全部包装起来，方便其他类直接调用。我们首先来看从类 Sentence。

一个 Sentence 类的实例就是一个句子，以此为基础，Sentence 类中需要记录构成句子的词法单元，我对这一性质又进行了包装，成为 TreeNode 类；还要记录句子对应的树的连线信息，信息包括标签和颜色，对这一性质我封装为 TreeLine 类进行表示。除此之外，还要记录点的个数，线的条数，句子内容，以及一个为了实现“撤销/重做”功能的 ArrayList 等，如下图所示：

```
class TreeLine implements Serializable {
    public String rel;
    public Color col;
    public TreeLine(String s, Color c) {
        rel = new String(s);
        col = new Color(c.getRGB());
    }
}

class TreeNode implements Serializable {
    public String label;
    public TreeNode(String s) {
        label = new String(s);
    }

}
public class Sentence implements Serializable {
    public String content;
    public int node_num, line_num;
    public int cur_version;
    public TreeNode node[];
    public TreeLine tempMatrix[][];
    public ArrayList<TreeLine[][]> record;
    public int father[], flag[];
```

Sentence 类的其他函数大致可以分成三类，第一类是设置性函数，包括 clearMatrix,saveMatrix,clearback,setnode,addline,setline,setlinecolor,removeline,undo,redo 用于添加结点、添加/修改连线，保存信息，标识修改；第二类是获取性函数，包括 curMatrix, getNodeNum, getLineNum, getString, getColor, getLabel，返回对象的各种信息，方便其他类调用；第三类是判断性函数，包括 CheckTree（find 和 unionset 为其子函数），canUndo, canRedo,第一个用来判断当前呈现在画板上的图是否仍然是一颗树，用到了并查集的方法。后两个则是判断是否有撤销/重做的余地，用于设置相应按键的激活状态。还有另外两个函数 writeToFile 和 readFromFile 在之后详细解释。

以上函数的编写总体比较清晰和基本，阅读起来应该比较轻松，故更多有关实现的细节和源码的分析在此就略去了。

下面介绍主类。这个类中包含了所有要显示在主框架中的程序，包括菜单项，面板，按键等，如下所示：

```
private JPanel showarea = new JPanel();
private JTextArea display = new JTextArea(10,100);

private JMenuBar mBar = new JMenuBar();
private JMenu fileMenu, texteditMenu, treeeditMenu,showMenu,helpMenu;
private JMenuItem quitItem, opentextItem,opentreeItem,savetextItem, savetreeItem, closeItem;
private JMenuItem cutItem, copyItem, pasteItem, selectItem;
public JMenuItem undoItem,redoItem,clearItem;
private JMenuItem showStanford,showEmpty,showPrevious;
private JMenuItem helpItem;

private ImageIcon icon;
public JButton unDoButton, reDoButton, openButton, saveResButton, clearTreeButton;
```

其中 showarea 是包含操作快捷键的面板，display 是文本框，mBar 是菜单栏，mainPanel 是显示依存树的面板。

之后会对整个应用程序的显示框进行一些显示参数的设置，包括设置应用框大小，设置文本框大小及字体，设置组件放置位置等，由于比较基本且繁琐，这里不再列出，详见 DrawAcr 类的构造函数及几个 initMenu()函数。

不过，构造函数中的一个代码块需要特别说明，如下所示：

```
try {
    int space;
    String tempStr;
       File MapInfo = new File(TreeStoragePosition);//文件路径
       BufferedReader br = new BufferedReader(new FileReader(MapInfo));
       while ((tempStr = br.readLine()) != null) {
           sentence_cnt++;
           space=tempStr.lastIndexOf(' ');
           steMap.put(tempStr.substring(0, space), tempStr.substring(space+1, tempStr.length()))
       }
   } catch (FileNotFoundException e) {
       e.printStackTrace();
   } catch (IOException e) {
       e.printStackTrace();
   }
```

这里是在进行应用程序本地的树库中的导入，通过一个 Map（steMap），将句子与其对应依存树信息联系起来，从而可以方便用户继续编辑已有的依存树。关于这一点，后面还会有更详细的介绍。

主类中的其他函数均是为监听事件的动作服务，在主面板上可以发生的动作包括如下几类：
1.点击菜单下拉栏
2.点击菜单工具栏
3.点击依存树编辑画板
4.点击文本框
5.在文本框中输入
其中，除了第 3 项事件的监听事件函数由陈庆英在另一个类完成外，其余 3 个事件的监听事件函数均由我在主类中实现。下面将重点讨论监听事件函数的设计与实现。

1.点击菜单下拉栏
我们的依存树编辑器有 5 个下拉菜单，分别是文件操作（File），文本框操作（EditText），依存树操作（EditTree），依存树初始显示选项（ShowOption）和帮助（Help）

（1）文件操作中有 6 个菜单项，分别是打开文本文件（OpenText），打开一棵已经编辑过的树（OpenTree），保存文本文件（SaveText），保存树的信息（SaveTree）以及关闭（Close）和退出（Quit）

打开文本文件时的操作与之前上机时做的一个小作业类似，如下所示：

```
FileDialog d = new FileDialog(this, "open file", FileDialog.LOAD);// 打开文件对话框
d.setVisible(true);
if (d.getDirectory() == null)
    return;
File f = new File(d.getDirectory() + d.getFile()); // 建立新文件
fileName = d.getDirectory() + d.getFile();// 得到文件名
char ch[] = new char[(int) f.length()];// /用此文件的长度建立一个字符数组
try// 异常处理
{
    // 读出数据，并存入字符数组ch中
    BufferedReader bw = new BufferedReader(new FileReader(f));
    bw.read(ch);
    bw.close();
} catch (FileNotFoundException fe) {
    System.out.println("file not found");
    System.exit(0);
} catch (IOException ie) {
    System.out.println("IO error");
    System.exit(0);
}
String s = new String(ch);
display.setText(s);// 设置文本区为所打开文件的内容
```

关闭文本文件与之类似：

```
FileDialog d = new FileDialog(this, "save file",
        FileDialog.SAVE);// 保存文件对话框
d.setVisible(true);
String s = display.getText();// 得到所输入的文本内容
try// 异常处理
{
    File f = new File(d.getDirectory() + d.getFile());// 新建文件
    fileName = d.getDirectory() + d.getFile();// 得到文件名
    if(d.getDirectory()==null)
        return ;
    BufferedWriter bw = new BufferedWriter(new FileWriter(f));// 输入到文件中
    bw.write(s, 0, s.length());
    bw.close();
} catch (FileNotFoundException fe_) {
    System.out.println("file not found");
    System.exit(0);
} catch (IOException ie_) {
    System.out.println(" IO error");
    System.exit(0);
}
```

保存树的信息用到了对象的序列化操作，它将表征一棵树的对象 ste（Sentence 类对象）的信息存入到一个文件中，之后要读取只需从这个文件读取，如下所示：

```
TreeModified=false;
String resName = savePosition + "res" + sentence_cnt    //生成保存文件名
        + ".txt";
writeRecords(resName);      //将对象序列化
steMap.put(parseSentence, resName); //加入到小树库中
JOptionPane.showMessageDialog(this, "Tree infomation saved in " + resName);
try{
    FileWriter writer = new FileWriter(TreeStoragePosition, true);  //为总的树库添加信息
    writer.write(parseSentence+" "+resName+"\n");
    writer.close();
}catch (IOException e) {
    e.printStackTrace();
}
sentence_cnt++;
```

对于保存树，我们还提供了保存树生成的图片的功能，实际上是利用 java 的截取屏幕的函数将依存树面板截取下来，如下所示：

```
try {
    // 拷贝屏幕到一个BufferedImage对象screenshot
    BufferedImage screenshot = (new Robot())
            .createScreenCapture(new Rectangle(mainPanel.getX(), mainPanel.
            getY()+50,
                    (int) (d.getWidth()-150), (int)( d.getHeight()/2-
                    mainPanel.getY()-10)));

    String name = picPosition+"res"+sentence_cnt +".jpg";
    File f = new File(name);
    // 将screenshot对象写入图像文件
    ImageIO.write(screenshot, "jpg", f);
    JOptionPane.showMessageDialog(this, "Image saved in " + name);
} catch (Exception ex) {
    System.out.println(ex);
}
```

从树库打开一棵树的操作与保存相对应，用到了去序列化操作，如下所示：

```
if(TreeName.equals("/"))              //从文件中读出树的信息
    readRecords(steMap.get(parseSentence));
else
    readRecords(TreeName);

ste.saveMatrix();
//重画主面板
this.getContentPane().remove(mainPanel);
showarea.remove(mainPanel.getLabelButton());
showarea.remove(mainPanel.getDeleteButton());
showarea.remove(mainPanel.getColorButton());
mainPanel=new DrawMonitor(ste,(int)(d.getWidth()-75),(int)(d.getHeight()-400),this);
mainPanel.repaint();
mainPanel.setBackground(Color.WHITE);
this.getContentPane().add("Center",mainPanel);
showarea.add(mainPanel.getDeleteButton());
showarea.add(mainPanel.getColorButton());
showarea.add(mainPanel.getLabelButton());
mainPanel.updateUI();
hasTree=true;
```

这里便用到了之前提到的 Sentence 类中的 readRecords 和 writeRecords 函数。这两个函数作为辅助函数，通过对 ObjectOutputStream 和 ObjectInputStream 的应用，将对象信息写入文件或从文件中读出对象信息。

读者可能注意到上述代码中“重画主面板”的代码段，这段代码首先从主框架中删去依存树编辑画板及监听其事件的按键，通过调用画图功能的主类 DrawMonitor 的构造函数，将初始化好的待分析的句子对象 ste 传入，通过 repaint 函数画出新图，再将画板及对应的按键重新加入主框架中，从而实现了画板的初始化。updatUI 函数使得画板的变化可以很快显示。这段代码在每次新生成依存树的时候都会调用，在之后的介绍中不会再重复这一部分。

最后，File 菜单的关闭和退出操作与之前上机时的文本编辑器类似，只不过这里加入了有关保存的提示，以免用户因误操作而丢失编辑结果，不再赘述。

（2）文本框操作包括复制、粘贴、剪切、全选，这里基本沿用了上机中文本编辑器的内容，这里仅简单给出代码，不讨论细节。

```
} else if (m == cutItem) {//剪切
    scratchPad = display.getSelectedText();
    display.replaceRange("", display.getSelectionStart(),
            display.getSelectionEnd());
} else if (m == copyItem) {//复制
    scratchPad = display.getSelectedText();
} else if (m == pasteItem) {//粘贴
    display.insert(scratchPad, display.getCaretPosition());
} else if (m == selectItem) {//全选
    display.selectAll();
```

（3）树编辑操作包括撤销、重做、清空。撤销操作只是通过 Sentence 类的 unDo

函数指示当前应该显示的对象信息，同时会设置撤销、重做键的激活信息，如下所示：

```
mainPanel.showOperation("undo");
ste.undo();
if (ste.canUndo() == false)
{
    undoItem.setEnabled(false);
    unDoButton.setEnabled(false);
}
redoItem.setEnabled(true);
reDoButton.setEnabled(true);
this.getContentPane().remove(mainPanel);
mainPanel.repaint();
mainPanel.setBackground(Color.WHITE);
this.getContentPane().add("Center",mainPanel);
mainPanel.updateUI();
```

重做操作函数与此完全一样，只是把 reDo 和 unDo 对换。清空操作函数 clearOp 也基本类似，只需用 clearTree 函数将对象的线关系全部清空。

（4）显示选项栏包括显示斯坦福库解析树（showStanford），显示没有线的空树（showEmpty），显示之前编辑结果（showPrevious）三个菜单项。前两个都调用了斯坦福的语料库及语法分析函数，通过某种转换得到所有依存关系，从而构建一个 Sentence 对象，以备后面画图使用。

```
TokenizerFactory<CoreLabel> tokenizerFactory = PTBTokenizer.factory(
        new CoreLabelTokenFactory(), "");
List<CoreLabel> rawWords = tokenizerFactory.getTokenizer(
        new StringReader(sent)).tokenize();
Tree parse = lp.apply(rawWords);
TreebankLanguagePack tlp = new PennTreebankLanguagePack();
GrammaticalStructureFactory gsf = tlp.grammaticalStructureFactory();
GrammaticalStructure gs = gsf.newGrammaticalStructure(parse);
List<TypedDependency> tdl = gs.typedDependenciesCCprocessed();
Iterator<TypedDependency> it = tdl.iterator();
ste = new Sentence(sent, rawWords.size(), tdl.size());
```

showEmpty 不需要设置线关系，而 showStanford 需要通过截取斯坦福返回的解析信息设置线关系。这部分主要频繁使用了 String 的 lastIndexof，Indexof 和 substring 函数，难度不大，这里就不贴代码了。

showPrevious 选项当分析的句子在树库中时有效，它会从树库中取出对应树的信息，此处与之前的“打开树”的操作十分类似，只不过此处是从句子所在的确定的位置打开信息，而不是之前的由用户选择打开文件。得到句子对应的位置是靠 Map<String,String>实现，第一个键值是句子，第二个键值是文件位置。

（5）帮助菜单只是简单地提示用户去查阅我们的帮助文档。

2.点击菜单工具栏

菜单工具栏中共有“撤销”,“重做”,“打开树”,“保存树”,“清空树”,“删除线”,“改变颜色”,“修改线标签”8 个按键，其中前 5 个按键与对应菜单项的功能完全一样，实现也完全一样，这里设置这些案件主要是使得应用的整体外观更加美观、活泼，同时给用户提供了便捷性。后三个按键则由陈庆英在别的类中实现，这里为了体现整体性，我将那三个按键也整合在工具栏中罢了。

3.点击文本框

用户在文本框中通过鼠标点击，选择出要分析的一句话，我会在 mousePressed 函数中通过判断光标的位置，确定用户选择的句子，以高亮显示，设该句为选中

```
DefaultHighlighter h = (DefaultHighlighter) display
        .getHighlighter();  //高亮类
DefaultHighlighter.DefaultHighlightPainter p = new DefaultHighlighter.
DefaultHighlightPainter(
        new Color(226, 239, 255));  //设置高亮颜色
try {
    int offset = display.getCaretPosition();    //得到插入符位置
    if (offset >= display.getText().length())
        return;
    int start = getStartPos(offset) //得到选中的句子
    int end = getEndPos(offset);
    parseSentence = new String(display.getText().substring(
            start + 1, end + 1 ));
    if (steMap.containsKey(parseSentence) == true)
        showPrevious.setEnabled(true);
    h.removeAllHighlights();
    h.addHighlight(start + 1, end + 1 , p); //高亮显示
} catch (BadLocationException e1) {
    e1.printStackTrace();
```

状态，如下所示：
其中的子函数 getStartPos 和 getEndPos 分别通过识别句子结尾符（./!/?/;）得到用户选择的句子，主要使用的是 String 类的 indexOf 和 lastIndexOf 函数，原理较易，不再赘述。

4.在文本框中输入
我用 KeyListener 监听用户是否在文本框自己进行了输入，从而标记之后用户退出时是否要提醒用户保存文本，这里只是简单地将一个 boolean 变量设为 true

四.　　软件使用说明

欢迎使用我们设计的英语依存树编辑器！
下面为您简要介绍我们的编辑器：
首先是布局方面，最上面是菜单栏，上部是显示依存树并且进行编辑的部分，下部是选择、编辑文本的区域。

接下来向您介绍具体操作方面。

【打开文件】您可以通过 File 菜单下的 Open 项打开一个已有的 txt 文件对其中的文本进行分析，也可以直接在下部的文本框中直接编辑，有关文本的具体编辑均在 EditText 菜单中。

【选择句子】在文本框中选中想要分析的句子，在 ShowOption 中选择树的呈现方式，showStanford 代表用 Stanford 的标准语料库解析，显示已经带有依存关系的树；showEmpty 代表显示一棵只有结点没有依存关系的树；showPrevious 表示如果当前的句子之前已经被依存分析过并将结果保存在应用对应的树库中，则可以选择呈现之前保存的依存树。这个项有效当且仅当句子对应的依存树在树库中可以找到。

【编辑依存树】我们支持以下对树的编辑操作————

在两个点间添加一条线：选中两个点，并输入依存关系，注意，可能在编辑后整个树的结构会被破坏，对此我们会发出警告。

删去一条线：点击一条线上的箭头以选中该线，点击菜单工具栏的“叉”按键，可以将其删除。

调整线/点的颜色：选中一条线或者一个点，点击菜单工具栏的“调色板”按键，选择任意颜色。

调整线的高度：点击一条线上的箭头，进行拖动以改变其高度。

清空所有线：点击菜单工具栏的“垃圾桶”按键或者菜单栏 EditTree 中 Clear 项。

以上的每一种操作我们都会在画板最右侧的消息栏中给出显示。同时，对于以上操作我们都提供了“撤销/重做”操作，可以点击菜单工具栏的“左箭头”、“右箭头”按键或菜单栏 EditTree 中 Undo,Redo 项来实现

【树库管理】用户在对依存树进行适当的编辑之后，可以点击菜单工具栏的“保存”按键或菜单栏 File 中的 SaveTree 项将依存树保存在树库中，以备后用，我们同时提供保存依存树图片的功能。同样地，您可以直接从树库中打开一棵树，直接进行编辑。

【关闭文件和退出】点击 File 菜单栏的 Close 项可以关闭当前文件，Quit 项则退出整个应用程序。请您一定注意在关闭和退出前是否已经保存好您的编辑结果。

【错误处理】如果用户进行一些非法操作，例如在还没有选中任何句子的时候点生成斯坦福依存树或空依存树，或者希望删去一个结点，希望把连线拖到句子下面等，我们都做了相应的检测，保证用户不允许做这样的操作，同时弹出提示框提醒用户。因此，您可以放心地使用我们的应用。

如有任何问题请联系开发者：刘敏行 thomas1201@126.com 陈庆英 1432077025@qq.com

再次感谢您的使用！

五. 作业评价与总结

此次作业很锻炼图形界面形式和人机交互手段的设计能力。给用户呈现一个操作简单、布局很好的界面一直是我们在开发过程中关注的一个重点，在布局管理器的选择、组件的放置、相关参数的设置等方面，我们下了很大的功夫。成果是令人欣慰的，也算不枉费我们的努力。

另外，在开发过程中遇到的许多问题，在作业基本完工之后看来是别有

意味的。有些问题不正面解决是很不理想的，有些问题是可避免的，有些问题则是可减轻的。通过在某些地方做合理的限制，带来的是问题的简单化，未尝不是一个好方法。

很高兴有这么一次宝贵的合作大作业的机会，我们在这一过程中提高的很快，感谢老师和助教的指导，也感谢对方在大作业编写过程中的努力与配合！

六. 成员分工与贡献

刘敏行：负责搭建应用的图像界面框架，编写针对各个组件的事件响应函数，负责为陈庆英提供不可见的依存树信息，方便其进行画板上的绘图。实现的类有 DrawArc（主类）和 Sentence。

陈庆英：负责画图以及当用户在画板上对依存树编辑时做出响应。实现的类有 DrawMonitor、Dot、TwoDot、ArcArrow.

七. 软件运行截图