```java
package com.freeradicand.mogul;

import java.awt.AWTEvent;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;
import java.util.LinkedList;
import java.util.Observable;
import java.util.Observer;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

/**
 *
 * @author Dylan Baker
 */
public class GameProgrammeStoryPipelinesView extends JPanel implements Observer {
    private int PREFERRED_WIDTH = 1024;
    private int PREFERRED_HEIGHT = 384;

    private GameStateModel model;
    private Story movingStory;

    // Each bar scrolls by itself
    private int storyOffset;
    private int prOffset;
    private int scheduleOffset;

    // Used to determine offsets when dragging Storys
    private Point mouseLanding = new Point(0,0);
    private Point mouseDelta = new Point(0,0);
    private Point targetLanding = new Point(0,0);

    public GameProgrammeStoryPipelinesView(GameStateModel gameState) {
        model = gameState;
        setBackground(Color.BLACK);
        setPreferredSize(new
        Dimension(PREFERRED_WIDTH,PREFERRED_HEIGHT));
        enableEvents(AWTEvent.MOUSE_EVENT_MASK);
        enableEvents(AWTEvent.MOUSE_MOTION_EVENT_MASK);
        movingStory = null;
    }

    @Override public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setFont(new Font("Monaco",1,12));
        FontMetrics fm = g2.getFontMetrics();
        LinkedList<Story> s = model.getStories();
        for (int i = 0; i < s.size(); i++) {
            if (!s.get(i).isMoved() || mouseDelta.equals(new Point(0,0)))
            {
                // We are operating on the assumption that only up to one
                story is
                // marked isMoved == true at a time. Any more is a
```

```
58                  critical bug
59                  // at any rate and needs to be caught elsewhere
60                  g2.setColor(determineDisplayColorOfStory(s.get(i)));
61                  g2.fillRect(i * 240 - storyOffset,0,240,128);
62                  g2.setColor(Color.BLACK);
63                  g2.fillRect(i * 240 + 8 - storyOffset,8,224,112);
64                  g2.setColor(Color.WHITE);
65                  drawWrappedHeadline(g2,s.get(i).getHeadline(),i * 240 +
                    12 - storyOffset,10 + fm.getHeight());
66                  g2.drawString("" + s.get(i).getLength() + " minutes", i *
                    240 + 12 - storyOffset,10 + (6 * fm.getHeight()));
67              }
68          }
69          s = model.getPR();
70          for (int i = 0; i < s.size(); i++) {
71              if (!s.get(i).isMoved() || mouseDelta.equals(new Point(0,0)))
                {
72                  g2.setColor(determineDisplayColorOfStory(s.get(i)));
73                  g2.fillRect(i * 240 - prOffset,128,240,128);
74                  g2.setColor(Color.BLACK);
75                  g2.fillRect(i * 240 + 8 - prOffset,136,224,112);
76                  g2.setColor(Color.WHITE);
77                  drawWrappedHeadline(g2,s.get(i).getHeadline(),i * 240 +
                    12 - prOffset,138 + fm.getHeight());
78                  g2.drawString("" + s.get(i).getLength() + " minutes", i *
                    240 + 12 - prOffset,138 + (6 * fm.getHeight()));
79              }
80          }
81          s = model.getSchedule();
82          for (int i = 0; i < s.size(); i++) {
83              if (!s.get(i).isMoved() || mouseDelta.equals(new Point(0,0)))
                {
84                  g2.setColor(determineDisplayColorOfStory(s.get(i)));
85                  g2.fillRect(s.get(i).getPositionInSchedule() * 96 -
                    scheduleOffset,288,s.get(i).getLength() * 96,128);
86              }
87          }
88          g2.setColor(Color.GRAY);
89          for (int i = 0; i < 30; i++) {
90              g2.setFont(new Font("Monaco",0,10));
91              g2.drawLine(i * 96,416,i * 96,432);
92              g2.drawString("" + i,i * 96 + 4,428);
93          }
94          if (movingStory != null && !mouseDelta.equals(new Point(0,0))) {
95              g2.setFont(new Font("Monaco",1,12));
96              g2.setColor(determineDisplayColorOfStory(movingStory));
97              Point anchor = new Point((int) (targetLanding.getX() +
                mouseDelta.getX()),(int) (targetLanding.getY() + mouseDelta.
                getY()));
98              g2.fillRect((int) anchor.getX(),(int) anchor.getY(),240,128);
99              g2.setColor(Color.BLACK);
100             g2.fillRect((int) anchor.getX() + 8,(int) anchor.getY() + 8,
                224,112);
101             g2.setColor(Color.WHITE);
102             drawWrappedHeadline(g2,movingStory.getHeadline(),(int)
                anchor.getX() + 12,(int) anchor.getY() + 10 +
                fm.getHeight());
103             g2.drawString("" + movingStory.getLength() + " minutes",
                (int) anchor.getX() + 12,(int) anchor.getY() + 10 + (6 * fm.
                getHeight()));
104         }
105     }
```

```
106
107     private Color determineDisplayColorOfStory(Story s) {
108         if (s.isAdBreak()) {
109             // the Field of a commercial break is immaterial and in fact
110             // meaningless in context
111             return Color.DARK_GRAY;
112         }
113         switch (s.getField()) {
114             case ARTS:
115                 return Color.MAGENTA;
116             case BUSINESS:
117                 return Color.CYAN;
118             case EDUCATION:
119                 return Color.WHITE;
120             case HEALTH:
121                 return Color.RED;
122             case HISTORY:
123                 return Color.YELLOW;
124             case INTERPLANETARY:
125                 return Color.BLUE;
126             case LAW:
127                 return Color.LIGHT_GRAY;
128             case MILITARY:
129                 return Color.ORANGE;
130             case POLICY:
131                 return Color.GRAY;
132             case SPORTS:
133                 return Color.PINK;
134             case TECHNOLOGY:
135                 return Color.GREEN;
136             default:
137                 // error case. no background
138                 return Color.BLACK;
139         }
140     }
141
142     private void drawWrappedHeadline(Graphics2D g2, String s, int x, int
        y) {
143         FontMetrics fm = g2.getFontMetrics();
144     }
145
146     @Override public void processMouseEvent(MouseEvent e) {
147         super.processMouseEvent(e);
148         if (!SwingUtilities.isLeftMouseButton(e)) {
149             return;
150         }
151
152         if (e.getID() == MouseEvent.MOUSE_PRESSED) {
153             Story target = mouseLocationToStory(e.getX(),e.getY());
154             if (target != null) {
155                 if (target.isAdBreak()) {
156                     return; // Commercial breaks are bound by law to be
                            regular. Can't move 'em.
157                 }
158                 mouseLanding = new Point(e.getX(),e.getY());
159                 mouseDelta = new Point(0,0);
160                 movingStory = target;
161                 target.setMoved(true);
162             }
163             repaint();
164         }
165         if (e.getID() == MouseEvent.MOUSE_RELEASED && movingStory !=
```

```
165          null) {
166              // These ridiculously fscking big if statements check where
                 we dropped the story
167              // They're not particularly flexible at the moment because
                 this is a bloody alpha prototype and intended only for system
                 testing, not UX review
168              if (new Rectangle((int) (targetLanding.getX() + mouseDelta.
                 getX()),(int) (targetLanding.getY() + mouseDelta.getY()),240,
                 128).intersection(new Rectangle(0,0,1024,128)).getHeight() >
                 64) {
169                  int targetPosition = ((int)(targetLanding.getX() +
                     mouseDelta.getX()) + storyOffset + 120) / 240;
170                  model.sendStoryToStoriesAtPosition(movingStory,
                     targetPosition);
171              } else if (new Rectangle((int) (targetLanding.getX() +
                 mouseDelta.getX()),(int) (targetLanding.getY() + mouseDelta.
                 getY()),240,128).intersection(new Rectangle(0,128,1024,128)).
                 getHeight() > 64) {
172                  int targetPosition = ((int)(targetLanding.getX() +
                     mouseDelta.getX()) + prOffset + 120) / 240;
173
                     model.sendStoryToPRAtPosition(movingStory,targetPosition)
                     ;
174              } else if (new Rectangle((int) (targetLanding.getX() +
                 mouseDelta.getX()),(int) (targetLanding.getY() + mouseDelta.
                 getY()),240,128).intersection(new Rectangle(0,288,1024,128)).
                 getHeight() > 64) {
175                  int targetPosition = ((int) (targetLanding.getX() +
                     mouseDelta.getX()) + prOffset) / 96;
176                  model.moveStoryToScheduleAtMinuteMark(movingStory,
                     targetPosition);
177              }
178
179              movingStory.setMoved(false);
180              movingStory = null;
181              mouseDelta = new Point(0,0);
182              repaint();
183          }
184      }
185
186      @Override public void processMouseMotionEvent(MouseEvent e) {
187          super.processMouseMotionEvent(e);
188          if (!SwingUtilities.isLeftMouseButton(e)) {
189              return;
190          }
191
192          if (e.getID() == MouseEvent.MOUSE_DRAGGED && movingStory != null)
                 {
193              mouseDelta = new Point((int) (e.getX() -
                 mouseLanding.getX()),(int) (e.getY() - mouseLanding.getY()));
194              repaint();
195          }
196
197      }
198
199      private Story mouseLocationToStory(int x, int y) {
200          if (y >= 0 && y <= 127) {
201              targetLanding = new Point((x / 240) * 240,0);
202              try {
203                  return (model.getStories().get((storyOffset + x) / 240));
204              } catch (IndexOutOfBoundsException e) { // this can't be a
                 good idea.
```

```
205                     return null;
206                 }
207             } else if (y >= 128 && y <= 255) {
208                 targetLanding = new Point((x / 240) * 240,128);
209                 try {
210                     return (model.getPR().get((prOffset + x) / 240));
211                 } catch (IndexOutOfBoundsException e) { // definitely not a
                    good idea
212                     return null;
213                 }
214             } else if (y >= 288 && y <= 415) {
215                 targetLanding = new Point((x / 96) * 96,288);
216                 try {
217                     return (model.getStoryFromScheduleAtMinuteMark((
                        scheduleOffset + x) / 96));
218                 } catch (IndexOutOfBoundsException e) { // absolutely not a
                    good idea
219                     return null;
220                 }
221             }
222             return null;
223         }
224 
225         @Override public void update(Observable o, Object o1) {
226         }
227     }
228 
```

```java
package com.freeradicand.mogul;

import java.util.LinkedList;
import java.util.ListIterator;
import java.util.Observable;

/**
 *
 * @author Dylan Baker
 */
public class GameStateModel extends Observable {

    public enum State { MENU, PLAYING, PAUSED }

    private State gameState;
    private LinkedList<Employee> employees;
    private LinkedList<Advertiser> potentialAdvertisers;
    private LinkedList<Story> availableStories;
    private LinkedList<Story> prPipeline;
    private LinkedList<Story> schedule;
    private int hrOffset = 0;
    private int currentViewedLawyerID = -1;

    public GameStateModel() {
        gameState = State.MENU;
        // The GameStateModel() is always constructed at boot and thus always
        // starts in the MENU state; adjustments to this are made after boot
        // once the user starts a game
    }

    public void setupNewGame() {
        // Method used for initializing a new game.
        employees = new LinkedList<Employee>();
        employees.add(new Journalist());
        employees.add(new Journalist());
        employees.add(new Lawyer());
        currentViewedLawyerID = 2; // bad idea
        potentialAdvertisers = new LinkedList<Advertiser>();
        for (int i = 0; i < 20; i++) {
            potentialAdvertisers.add(new Advertiser());
        }
        availableStories = new LinkedList<Story>();
        availableStories.add(new Story());
        availableStories.add(new Story());
        prPipeline = new LinkedList<Story>();
        prPipeline.add(new Story());
        prPipeline.get(0).setIsPR(true);
        schedule = new LinkedList<Story>();
        schedule.add(new Story(Story.Field.POLICY,1,"*Ad Break*"));
        schedule.get(0).setIsAdBreak(true);
        schedule.get(0).setPositionInSchedule(9);
        schedule.add(new Story(Story.Field.POLICY,1,"*Ad Break*"));
        schedule.get(1).setIsAdBreak(true);
        schedule.get(1).setPositionInSchedule(19);
    }

    public void beginGame() {
        // Method called once we begin a game, new or saved, to kick things off
        gameState = State.PAUSED;
```

```
 60          GameLoopThread thread = new GameLoopThread(this);
 61          thread.start();
 62      }
 63
 64      public void update() {
 65          if (gameState != State.PLAYING) {
 66              setChanged();
 67              notifyObservers();
 68          }
 69      }
 70
 71      public State getGameState() {
 72          return gameState;
 73      }
 74
 75      public LinkedList<Employee> getEmployees() {
 76          return employees;
 77      }
 78
 79      public void nextViewedLawyer() {
 80          ListIterator<Employee> i = employees.listIterator(
             currentViewedLawyerID);
 81          while (i.hasNext()) {
 82              Employee e = i.next();
 83              if (e.getProfession().equals("Lawyer")) {
 84                  currentViewedLawyerID = employees.indexOf(e);
 85              }
 86          }
 87      }
 88
 89      public void prevViewedLawyer() {
 90          ListIterator<Employee> i = employees.listIterator(
             currentViewedLawyerID);
 91          while (i.hasPrevious()) {
 92              Employee e = i.previous();
 93              if (e.getProfession().equals("Lawyer")) {
 94                  currentViewedLawyerID = employees.indexOf(e);
 95              }
 96          }
 97      }
 98
 99      public int getHROffset() {
100          return hrOffset;
101      }
102
103      public int getCurrentViewedLawyerID() {
104          return currentViewedLawyerID;
105      }
106
107      public void hireEmployee(Employee e) {
108          employees.offerLast(e);
109      }
110
111      public void fireEmployee(int e) {
112          Employee l = employees.get(currentViewedLawyerID);
113          employees.remove(e);
114          currentViewedLawyerID = employees.indexOf(l);
115          setChanged();
116          notifyObservers();
117      }
118
119      public LinkedList<Story> getStories() {
```

```
120         return availableStories;
121     }
122
123     public LinkedList<Story> getPR() {
124         return prPipeline;
125     }
126
127     public LinkedList<Story> getSchedule() {
128         return schedule;
129     }
130
131     public void sendStoryToStoriesAtPosition(Story s, int p) {
132         // Redundant method names for the win!
133         // In context, though, it should be apparent that this means
            we're sending
134         // the story to the list of those produced by staff
135         if (s.isAdBreak()) {
136             // Ad breaks start on the schedule and always stay there.
137             return;
138         }
139         if (s.isPR()) {
140             // PR can't be sent to the journalistic feed and vice versa
141             return;
142         }
143         if (p >= 0 && p <= availableStories.size()) {
144             if (availableStories.indexOf(s) >= 0) {
145                 availableStories.add(p,availableStories.remove(
                    availableStories.indexOf(s)));
146             } else { // it must be coming from the schedule
147                 s.setPositionInSchedule(-1);
148
                    availableStories.add(p,schedule.remove(schedule.indexOf(s
                    )));
149             }
150         }
151     }
152
153     public void sendStoryToPRAtPosition(Story s, int p) {
154         // Method does exactly the same thing as the other one except
155         // this operates on the PR Pipeline instead. (Lots of double code
            for this
156         // design choice. No longer sure it was a good idea.)
157         if (s.isAdBreak()) {
158             return;
159         }
160         if (!s.isPR()) {
161             return;
162         }
163         if (p >= 0 && p <= prPipeline.size()) {
164             if (prPipeline.indexOf(s) >= 0) {
165
                    prPipeline.add(p,prPipeline.remove(prPipeline.indexOf(s))
                    );
166             } else {
167                 s.setPositionInSchedule(-1);
168                 prPipeline.add(p,schedule.remove(schedule.indexOf(s)));
169             }
170         }
171     }
172
173     public boolean moveStoryToScheduleAtMinuteMark(Story s, int m) {
174         if (s.getLength() + m > 30) {
```

```
175             // noap
176             return false;
177         }
178         for (int i = 0; i < schedule.size(); i++) {
179             if (schedule.get(i).getPositionInSchedule() < (s.getLength()
                + m) && !schedule.get(i).equals(s)) {
180                 // Check for collisions. The schedule is not forgiving of
                    drag
181                 // errors and kicks the stories out rather than bumping
                    things.
182                 if (schedule.get(i).getPositionInSchedule() + schedule.
                    get(i).getLength() > m) {
183                     return false;
184                     // So, if another story starts before the end *and*
185                     // ends after the beginning of this new story, it
                        cancels the move.
186                 }
187             }
188         }
189         availableStories.remove(s);
190         prPipeline.remove(s);
191         schedule.remove(s); // get it out of wherever it came from
192         s.setPositionInSchedule(m);
193         schedule.add(s);
194         return true;
195     }
196 
197     public Story getStoryFromScheduleAtMinuteMark(int m) {
198         // Because the Schedule isn't stored in order (due to its
            calamitously
199         // poor design which allows for nulls of various sizes), we need
            this
200         // search method to determine where a given marker is
201         if (m >= 30) {
202             System.out.println("I don't know how you pulled this off but
                no, you can't have stories past the 30-minute mark");
203             return null;
204         }
205         for (int i = 0; i < schedule.size(); i++) {
206             if (schedule.get(i).getPositionInSchedule() <= m) {
207                 if (schedule.get(i).getPositionInSchedule() + schedule.
                    get(i).getLength() > m) {
208                     return schedule.get(i);
209                 }
210             }
211         }
212         return null;
213     }
214 
215     private class GameLoopThread extends Thread {
216         private GameStateModel model;
217         GameLoopThread(GameStateModel m) {
218             model = m;
219         }
220         @Override public void run() {
221             model.update();
222         }
223     }
224 }
225 
```