View Javadoc

```
1   package controller;
2
3   import java.util.Random;
4
5   import javax.swing.JOptionPane;
6
7   import contract.ControllerOrder;
8   import contract.ElementType;
9   import contract.IController;
10  import contract.IElement;
11  import contract.IModel;
12  import contract.IView;
13
14  /**
15   * The Class Controller.
16   *
17   * @author Group 5
18   */
19  public final class Controller implements IController {
20
21      /** The view. */
22      private IView   view;
23
24      /** The model. */
25      private IModel  model;
26
27      /** The order */
28      private ControllerOrder stackOrder;
29
30      /** The gravity */
31      private int gravity = 0;
32
33
34      /**
35       * Instantiates a new controller.
36       *
37       * @param view
38       *          the view
39       * @param model
40       *          the model
41       */
42      public Controller(final IView view, final IModel model) {
43              this.setView(view);
44              this.setModel(model);
45      }
46
47      /**
48       * The method control.
49       *
50       * Print a message on start.
51       */
52      /*
53       * (non-Javadoc)
54       *
55       * @see contract.IController#control()
56       */
57      public void control() {
58              this.view.printMessage("Use the directional arrows to move around.");
59      }
60
61      /**
62       * Sets the view.
63       *
64       * @param pview
65       *            the new view
66       */
67      private void setView(final IView pview) {
68              this.view = pview;
69      }
70
71      /**
72       * Sets the model.
73       *
74       * @param model
75       *          the new model
76       */
77      private void setModel(final IModel model) {
78              this.model = model;
79      }
80
81      /**
82       * Gets the model.
83       *
84       * @return model
85       */
86      private IModel getModel() {
87              return this.model;
88      }
89
90      /**
91       * The play method.
92       *
93       * Controls the movement of all entities.
94       *
95       */
96      public void play() {
97
98              while(this.getModel().getLevelMap().getPlayer().isExist()){
99
100                     try {
101                             Thread.sleep(40);
```

```
102                     } catch (InterruptedException e) {
103                             e.printStackTrace();
104                     }
105
106             IElement player = this.getModel().getLevelMap().getPlayer();
107
108             boolean col = this.getElementNext(player, this.getStackOrder());
109
110                     switch (this.getStackOrder()) {
111                             case Up:
112                                     if(col) {
113                                             player.moveUp();
114                                     }
115                                     break;
116                             case Down:
117                                     if(col) {
118                                             player.moveDown();
119                                     }
120                                     break;
121                             case Left:
122                                     if(col) {
123                                             player.moveLeft();
124                                     }
125                                     break;
126                             case Right:
127                                     if(col) {
128                                             player.moveRight();
129                                     }
130                                     break;
131                             default :
132                                     player.doNothing();
133                                     break;
134
135                     }
136                     this.clearStackOrder();
137
138
139                     for(int x=0; x<40; x++) {
140                             for(int y=0; y<22; y++) {
141                                     IElement e1 = this.getModel().getLevelMap().getElement(x, y);
142                                     boolean cole1 = false;
143                                     if(e1 != null && e1.getElementType() == ElementType.ENEMY) {
144                                             Random r = new Random();
145                                             int valeur = 1 + r.nextInt(5 - 1);
146                                             switch(valeur) {
147                                             case 1:
148                                                     cole1 = this.getElementNext(e1, ControllerOrder.Up);
149                                                     if(cole1) {
150                                                             e1.moveUp();
151                                                     }
152                                                     break;
153                                             case 2:
154                                                     cole1 = this.getElementNext(e1, ControllerOrder.Down);
155                                                     if(cole1) {
156                                                             e1.moveDown();
157                                                     }
158                                                     break;
159                                             case 3:
160                                                     cole1 = this.getElementNext(e1, ControllerOrder.Right);
161                                                     if(cole1) {
162                                                             e1.moveRight();
163                                                     }
164                                                     break;
165                                             case 4:
166                                                     cole1 = this.getElementNext(e1, ControllerOrder.Left);
167                                                     if(cole1) {
168                                                             e1.moveLeft();
169                                                     }
170                                                     break;
171                                             default:
172                                                     e1.doNothing();
173                                                     break;
174                                             }
175                                     }
176
177                                     else if(e1 != null && (e1.getElementType() == ElementType.ROCK || e1.getElementType() == ElementType.DIAMOND)
178                                             if(this.getStackOrder() != ControllerOrder.Default) break;
179                                             try {
180                                                     Thread.sleep(9);
181                                             } catch (InterruptedException e) {
182
183                                                     e.printStackTrace();
184                                             }
185
186                                             cole1 = this.getElementNext(e1, ControllerOrder.Down);
187
188                                             if(cole1) {
189
190                                                     e1.moveDown();
191
192                                             }
193
194                                     }
195
196                             }
197
198                     }
199             }
200     }
201
202     /**
203      * Gets the order.
204      *
205      * @return stackOrder
```

```
206      */
207     public ControllerOrder getStackOrder() {
208 
209             return stackOrder;
210     }
211 
212     /**
213      * Sets the stackOrder.
214      *
215      * @param stackOrder
216      *                              The stackOrder.
217      */
218     public void setStackOrder(ControllerOrder stackOrder) {
219 
220             this.stackOrder = stackOrder;
221     }
222 
223     /**
224      * Reset the stackOrder.
225      *
226      */
227     private void clearStackOrder() {
228 
229         this.stackOrder = ControllerOrder.Default;
230     }
231 
232     /**
233      * Gets the controller.
234      *
235      * @return controller
236      */
237     public IController getOrderPeformer() {
238         return this;
239     }
240 
241     /*
242      * (non-Javadoc)
243      *
244      * @see contract.IController#orderPerform()
245      */
246     @Override
247     public final void orderPerform(final ControllerOrder userOrder) {
248 
249         this.setStackOrder(userOrder);
250     }
251 
252     /**
253      * Gets the next element.
254      *
255      * @param element
256      *                      The element.
257      * @param controllerOrder
258      *                      The controllerOrder.
259      * @return boolean
260      */
261     public boolean getElementNext(IElement element, ControllerOrder controllerOrder) {
262 
263             boolean res = true;
264 
265             IElement elementA = element;
266             IElement elementN = null;
267 
268 
269                     switch (controllerOrder) {
270                     case Up:
271                             elementN = this.model.getLevelMap().getElement(elementA.getX(), elementA.getY()-1);
272                             break;
273                     case Down:
274                             elementN = this.model.getLevelMap().getElement(elementA.getX(), elementA.getY()+1);
275                             break;
276                     case Left:
277                             elementN = this.model.getLevelMap().getElement(elementA.getX()-1, elementA.getY());
278                             break;
279                     case Right:
280                             elementN = this.model.getLevelMap().getElement(elementA.getX()+1, elementA.getY());
281                             break;
282                     case Default :
283                             elementN = null;
284                             break;
285                     }
286 
287             res = collision(elementA, elementN, controllerOrder);
288             return res;
289 
290 
291     }
292 
293     /**
294      * checks for collisions.
295      *
296      * @param elementA
297      *                              The actual element.
298      * @param elementN
299      *                              The next element.
300      * @param controllerOrder
301      *                              The controllerOrder
302      * @return boolean
303      */
304     public boolean collision(IElement elementA, IElement elementN, ControllerOrder controllerOrder) {
305 
306             ElementType TelementN = null;
307             ElementType TelementA = elementA.getElementType();
308 
309             if(elementN != null) {
```

```
310                 TelementN = elementN.getElementType();
311             }
312             else {
313                 TelementN = ElementType.DEFAULT;
314             }
315 
316 
317             boolean res = false;
318 
319             switch(TelementA) {
320             case ROCK :
321             case DIAMOND :
322                 switch(TelementN) {
323                 case ENEMY:
324                     res = true;
325                     this.model.getLevelMap().popDiamond(elementN.getX(), elementN.getY());
326                     break;
327                 case PLAYER:
328                     if(gravity >= 1) {
329                         res = true;
330                         this.model.getLevelMap().getPlayer().setExist(false);;
331                         JOptionPane.showMessageDialog(null, "You die !");
332                         System.exit(0);
333                     }
334                     break;
335                 case DIAMOND:
336                     this.gravity = 0;
337                 case ROCK:
338                     this.gravity = 0;
339                 case UNBREAKABLEBLOCK:
340                     this.gravity = 0;
341                     break;
342                 case DEFAULT:
343                     this.gravity++;
344                     res = true;
345                     break;
346                 default :
347                     this.gravity = 0;
348                     res = false;
349                     break;
350                 }
351                 break;
352             case ENEMY:
353                 switch(TelementN) {
354                 case PLAYER :
355                     res = true;
356                     this.model.getLevelMap().getPlayer().setExist(false);;
357                     JOptionPane.showMessageDialog(null, "You die !");
358                     System.exit(0);
359                 break;
360                 case DEFAULT :
361                     res = true;
362                 break;
363                 default :
364                     res = false;
365                     break;
366                 }
367                 break;
368             case PLAYER:
369                 switch(TelementN) {
370                 case BLOCK:
371                     res = true;
372                     break;
373                 case DIAMOND:
374                     res = true;
375                     this.model.getLevelMap().getPlayer().setScore(this.model.getLevelMap().getPlayer().getScore()+1);;
376                     break;
377                 case ENEMY:
378                     res = true;
379                     this.model.getLevelMap().getPlayer().setExist(false);;
380                     JOptionPane.showMessageDialog(null, "You die !");
381                     System.exit(0);
382                     break;
383                 case EXIT:
384                     res = true;
385                     if(this.model.getLevelMap().getPlayer().getScore() >= 5) {
386                         JOptionPane.showMessageDialog(null, "You win !");
387                         System.exit(0);
388                     }
389                     else {
390                         res = false;
391                     }
392                     break;
393                 case ROCK:
394                     if(getElementNext(elementN, controllerOrder))
395                     {
396                         if(controllerOrder == ControllerOrder.Left) {
397                             elementN.moveLeft();
398                             res = true;
399                         }
400                         else if(controllerOrder == ControllerOrder.Right) {
401                             elementN.moveRight();
402                             res = true;
403                         }
404                     }
405                     else {
406                         res = false;
407                     }
408 
409                     break;
410                 case UNBREAKABLEBLOCK:
411                     res = false;
412                     break;
413                 case DEFAULT :
```

```
414                             res = true;
415                             break;
416                     default:
417                             res = false;
418                             break;
419                     }
420                     break;
421             default:
422                     res = true;
423                     break;
424
425             }
426             return res;
427
428     }
429 }
```