```java
package juego;

import java.awt.Image;
import java.util.Random;

import juego.Viga;
import entorno.Entorno;
import entorno.Herramientas;

public class Barril {

    private double posx;
    private double posy;
    private int diametro;
    private double escala;

    private Image spin_izquierda;
    private Image spin_derecha;
    private Image cayendo;

    // Conserva la dirección en la que estaba moviendo el barril "derecha" o
    // "izquierda"
    private String ultima;

    /*
     * Es necesaria una variable que indique que ee barril fue saltado para que no
     * sea contado doble en las siguientes situaciones: En el proceso de salto, la
     * función que detecta el correcto salto puede dar varios positivos durante
     una
     * cantidad de ticks cercanos entre si. Atrapando el primer tick donde se
     * detecta que el barril fue saltado, se evita que en los ticks siguientes
     donde
     * también es positivo, se cuenten los puntos inecesariamente. Evitar que el
     * jugador salte el barril en una viga y luego intente saltarlo en una viga
     * inferior para contador doble puntaje.
     */

    private boolean saltado;

    // Conocer si está cayendo por escalera permite detener el movimiento hacia
    // izquierda o derecha. Ayuda a diferenciar una caida desde una viga con
    // respecto a la de escalera.
    private boolean cayendoPorEscalera;

    // Como un barril tarda varios ticks en atravesar el ancho de una escalera.
    // Necesitamos indicar en que tick se tomó la decisión de caer o no por la
    // misma. Para que en el tick siguiente
    // no sobreescriba la decisión. Toma una decisión por escalera,y bloquea
    decidir
    // de nuevo por un cierto tiempo.
    private int ultimaEleccion;

    // Para animar la caida por escalera se utiliza esta variable para
    intercambiar
    // entre 10° y -10°.
    private int anguloRotacion;

    // Asiste en la elección del anguloRotacion
    private boolean sentidoRotacionDerecha;

    public Barril(Viga vigasuelo) {

        this.diametro = 17;
```

```
60          this.escala = (double) this.diametro / 108;
61
62          /*
63           * Se crea el barril en la posición desde donkey decidió arrojarlo
64           */
65          if (vigasuelo.getPos() == 6) {
66              this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 20;
67              this.ultima = "derecha";
68              this.posx = 120;
69          } else if (vigasuelo.getPos() == 4) {
70              this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 190;
71              this.ultima = "izquierda";
72              this.posx = 120;
73          }
74
75          this.spin_izquierda = Herramientas.cargarImagen(
            "rsc/graficos/barriles/spin-izquierda.gif");
76          this.spin_derecha = Herramientas.cargarImagen(
            "rsc/graficos/barriles/spin-derecha.gif");
77          this.cayendo = Herramientas.cargarImagen(
            "rsc/graficos/barriles/cayendo.png");
78
79          this.saltado = false;
80          this.ultimaEleccion = 0;
81          this.anguloRotacion = 0;
82          this.sentidoRotacionDerecha = true;
83
84      }
85
86      /*
87       * Esta función al ser llamada retorna si el barril debe destruirse por estar
88       * fuera de la pantalla del juego
89       *
90       */
91      public boolean deboDestruirme(Entorno entorno, Viga[] suelos) {
92          if (this.posx < 15 && this.pisando(suelos) == 0) {
93              return true;
94          } else {
95              return false;
96          }
97      }
98
99      /*
100      * Esta función se encarga de dibujar en pantalla al barril y calcular su
101      * moviento.
102      *
103      */
104
105     public void dibujar(Entorno entorno, int contador, Viga[] suelos, Escaleras[]
         escaleras) {
106
107         // Si está rodando sobre el suelo
108         if (pisando(suelos) != -1) {
109
110             // Si la decisión de caer por la escalera es afirmativa
111             if (caerPorEscalera(escaleras, suelos, contador)) {
112
113                 // Este movimiento inicial detendrá en el próximo los
                    movimientos a izquierda o
114                 // derecha
115                 // Porque la función pisando devolverá -1
116                 this.posy = this.posy + 1;
117                 entorno.dibujarImagen(cayendo, this.posx, this.posy, this.
```

```
                    anguloRotacion, this.escala);
118
119             } else {
120
121                 // En vigas con indice par desplazar a izquierda
122                 if (this.posx >= 10 && pisando(suelos) % 2 == 0) {
123                     this.posx = this.posx - 1.7;
124                     entorno.dibujarImagen(spin_izquierda, this.posx, this.posy, 0
                        , this.escala);
125                     this.ultima = "izquierda";
126                 }
127
128                 // En vigas con indice impar desplazar a derecha
129                 else if (this.posx <= 800 && pisando(suelos) % 2 == 1) {
130                     this.posx = this.posx + 1.7;
131                     entorno.dibujarImagen(spin_derecha, this.posx, this.posy, 0,
                        this.escala);
132                     this.ultima = "derecha";
133                 }
134
135             }
136         }
137
138         // Si NO está rodando sobre el suelo
139         if (pisando(suelos) == -1) {
140
141             // cambia la posición con respecto al eje "y" hacia abajo
142             this.posy += 1;
143
144             // Si la caida está producida por caer por escalera
145             if (this.cayendoPorEscalera) {
146
147                 // Cada vez que el contador es divisible por 10, entonces hay
                    que cambiar el
148                 // sentido de rotación
149                 // Esto es puramente gráfico, da la sensación de un barril
                    cayendo por
150                 // escaleras.
151                 if (contador % 10 == 0) {
152                     if (this.sentidoRotacionDerecha) {
153                         this.sentidoRotacionDerecha = false;
154                     } else {
155                         this.sentidoRotacionDerecha = true;
156                     }
157                 }
158
159                 // Dependiendo el sentido de Rotación, el engulo es positivo o
                    negativo
160                 if (this.sentidoRotacionDerecha) {
161                     this.anguloRotacion = 10;
162                 } else {
163                     this.anguloRotacion = -10;
164                 }
165
166                 // dibujar la caida por escalera
167                 entorno.dibujarImagen(cayendo, this.posx, this.posy, this.
                    anguloRotacion, this.escala);
168
169                 // Si la caida es producida por el final de una viga
170             } else {
171
172                 // Si venia desplazandose a derecha pero está cayendo y hay
                    espacio en el x,
```

```
173                 // se sigue desplazando a derecha
174                 if (this.posx <= 800 && this.ultima.equals("derecha")) {
175                     this.posx = this.posx + 1.7;
176                     entorno.dibujarImagen(spin_derecha, this.posx, this.posy, 0,
                        this.escala);
177                 }
178
179                 // De lo contrario hay que indicarle que en el próximo tick se
                    desplace a
180                 // izquierda.
181                 else {
182                     this.ultima = "izquierda";
183                 }
184
185                 // Si venia desplazandose a izquierda pero está cayendo y hay
                    espacio en el x,
186                 // se sigue desplazando a izquierda
187                 if (this.posx >= 10 && this.ultima.equals("izquierda")) {
188                     this.posx = this.posx - 1.7;
189                     entorno.dibujarImagen(spin_izquierda, this.posx, this.posy, 0
                        , this.escala);
190
191                 } else {
192                     // De lo contrario hay que indicarle que en el próximo tick
                        se desplace a
193                     // derecha.
194                     this.ultima = "derecha";
195                 }
196
197             }
198         }
199
200     }
201
202     // Igual que pisando de Personaje
203     public int pisando(Viga[] suelos) {
204
205         for (int i = 0; i < suelos.length; i++) {
206
207             if (this.pies() == (int) suelos[i].dondeEmpiezaElSuelo()) {
208
209                 if (this.lateralDerecho() < suelos[i].extremoIzquierdo()
210                         || this.lateralIzquierdo() > suelos[i].extremoDerecho()) {
211
212                     return -1;
213
214                 } else {
215
216                     return i;
217
218                 }
219             }
220
221         }
222
223         return -1;
224
225     }
226
227     /*
228      * Esta función toma la decisión de decidir si el barril caerá por la
        siguiente
229      * escalera.
```

```
230         */
231
232         public boolean caerPorEscalera(Escaleras[] escaleras, Viga[] suelos, int
            contador) {
233
234             // Sólo se toma la decisión de nuevo, si ha pasado el suficiente tiempo
                (o sea
235             // que se está decisión sobre una escalera
236             // diferente a la anterior decidida.
237             if (this.ultimaEleccion + 30 < contador) {
238
239                 Random rnd = new Random();
240                 int numero = rnd.nextInt(1200);
241
242                 // i es el piso actual por el cual rueda el barril
243                 int i = pisando(suelos);
244
245                 // Comprobación de escaleras para todos los pisos excepto la planta
                    baja
246                 if (i != 0) {
247
248                     /*
249                      * Analisis de una escalera cerca de tipo obligatoria, se toma
                        la decisión si se
250                      * cumple una segunda condición: Estar cerca de una escalera, de
                        forma de que
251                      * exista posibilidad en el proximo tick de caer por ella.
252                      */
253
254                     // A la altura de los pies del barril
255                     if (escaleras[i - 1].extremoSuperior() - this.pies() <= 10) {
256
257                         // Entre las coordinadas x del barril debe haber una escalera
258                         if (escaleras[i - 1].lateralDerecho() - 9 >= this.posx
259                                 && escaleras[i - 1].lateralIzquierdo() + 9 <= this.
                                    posx) {
260
261                             // Si el random es divisible por 6 cae, sino no. 1/6 de
                                posibilidades de caer.
262                             if (numero % 6 == 0) {
263                                 this.cayendoPorEscalera = true;
264                                 this.ultimaEleccion = contador;
265                                 return true;
266                             } else {
267                                 this.cayendoPorEscalera = false;
268                                 this.ultimaEleccion = contador;
269                                 return false;
270                             }
271                         }
272                     }
273
274                     /*
275                      * Mismo analisis de una escalera cerca pero de tipo adicional,
                        se toma la
276                      * decisión si se cumple una segunda condición: Estar cerca de
                        una escalera, de
277                      * forma de que exista posibilidad en el proximo tick de caer
                        por ella.
278                      */
279
280                     // A la altura de los pies del barril (Útil para eliminar
                        escaleras no
281                     // completas)
```

```
282                     if (escaleras[i + 4].extremoSuperior() - this.pies() <= 10) {
283
284                         // Entre las coordinadas x del barril debe haber una escalera
285                         if (escaleras[i + 4].lateralDerecho() >= this.posx
286                                 && escaleras[i + 4].lateralIzquierdo() <= this.posx) {
287
288                             // Si el random es divisible por 6 cae, sino no. 1/6 de
                                posibilidades de caer.
289                             if (numero % 6 == 0) {
290                                 this.cayendoPorEscalera = true;
291                                 this.ultimaEleccion = contador;
292                                 return true;
293                             } else {
294                                 this.cayendoPorEscalera = false;
295                                 this.ultimaEleccion = contador;
296                                 return false;
297                             }
298                         }
299                     }
300
301                 }
302             }
303             // Para todos los demás casos no hay caida posible por escalera
304             cayendoPorEscalera = false;
305             return false;
306
307         }
308
309         public int pies() {
310             return (int) this.posy + diametro / 2 - 2;
311         }
312
313         public int superior() {
314             return (int) this.posy - diametro / 2 + 2;
315         }
316
317         public int lateralDerecho() {
318             return (int) this.posx + diametro / 2;
319         }
320
321         public int lateralIzquierdo() {
322             return (int) this.posx - diametro / 2;
323         }
324
325         public int centroX() {
326             return (int) this.posx;
327         }
328
329         public void saltado() {
330             this.saltado = true;
331         }
332
333         public boolean fueSaltado() {
334             return this.saltado;
335         }
336
337     }
```