```java
package juego;

import juego.Viga;
import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class Personaje {

    private int posx;
    private int posy;

    // Se reservan las referencias necesarias para cada animacion utilizada
    private Image mirandoIzquierda;
    private Image mirandoDerecha;
    private Image caminandoIzquierda;
    private Image caminandoDerecha;
    private Image saltandoIzquierda;
    private Image saltandoDerecha;
    private Image subiendo;
    private Image subiendo_quieto;

    // Esta referencia realizará un aliasing a la imagen que debe ser mostrada
    // Se muestra una por vez
    private Image imagenMario;

    private int tiempoSalto; // tick en el cual se ejecutó el último salto (o el salto actual)
    private boolean estaSaltando; // Indica si está saltando (ascendiendo) o no.
    private boolean estaCayendo; // Indica si está cayendo (es decir que sus pies no están tocando viga alguna.

    private boolean estaCercaEscalera; // Indica si el personaje se encuentra lo suficientemente cerca de una escalera
                                       // (para poder usarla)
    private boolean estaEnEscalera; // Indica si el personaje se encuentra dentro (usando) una escalera
    private int subidoAEscaleraNro; // Indica que el indice que corresponde a la posición de la escalera que se está
                                    // usando dentro del arreglo de escaleras

    private int sonando; // Ultimo archivo de sonido que se usó para caminar, hay 3 variantes.
    private int sonandoDesde; // tick en el cual se ejecutó el último sonido de caminar (ayuda a evitar que
                                    // suenen sonidos en cada tick)

    private boolean miraDerecha; // Esta variable indica si el personaje está mirando a derecha o no (Vital para
                                    // que se cargue la imagen correcta del personaje según los movimientos que
                                    // indique el usuario)

    public Personaje() {

        // Posición por defecto de spawn
        this.posx = 50;
        this.posy = 530;

        // direcciones URL de las imagenes y animaciones
        this.mirandoIzquierda = Herramientas.cargarImagen(
        "rsc/graficos/marito/mira-izquierda.png");
```

```
54          this.mirandoDerecha = Herramientas.cargarImagen(
            "rsc/graficos/marito/mira-derecha.png");
55          this.caminandoIzquierda = Herramientas.cargarImagen(
            "rsc/graficos/marito/camina-izquierda.gif");
56          this.caminandoDerecha = Herramientas.cargarImagen(
            "rsc/graficos/marito/camina-derecha.gif");
57          this.saltandoIzquierda = Herramientas.cargarImagen(
            "rsc/graficos/marito/salta-izquierda.png");
58          this.saltandoDerecha = Herramientas.cargarImagen(
            "rsc/graficos/marito/salta-derecha.png");
59          this.subiendo = Herramientas.cargarImagen(
            "rsc/graficos/marito/subiendo.gif");
60          this.subiendo_quieto = Herramientas.cargarImagen(
            "rsc/graficos/marito/quieto_subiendo.png");
61
62          // Por defecto no hubo salto y no momentos anterior al cero. Tampoco el
63          // personaje esta saltando, cayendo ni en una escalera ni cerca de alguna.
64          this.tiempoSalto = 0;
65          this.estaSaltando = false;
66          this.estaCayendo = false;
67          this.estaEnEscalera = false;
68          this.estaCercaEscalera = false;
69
70          this.sonando = 1;
71          this.sonandoDesde = 0;
72
73          // Por defecto la imagen a mostrarse es mirando a derecha
74          this.imagenMario = this.mirandoDerecha;
75
76          // Por defecto debe mirar a derecha
77          this.miraDerecha = true;
78      }
79
80
81      /*
82       * Es la función que indica si el jugador está tocando el barril pasado como
         parámetro
83       */
84      public boolean tocando(Barril[] barriles) {
85
86          for (int i = 0; i < barriles.length; i++) {
87
88              if (barriles[i] != null) {
89
90                  if (this.lateralDerecho() - barriles[i].lateralIzquierdo() > 0
91                          && this.lateralIzquierdo() - barriles[i].lateralIzquierdo
                            () < 0
92                          && this.obtenerPosPies() - barriles[i].superior() >= 3
93                          && this.obtenerPosCabeza() - barriles[i].pies() <= -10) {
94
95                      System.out.println("[" + i + "] Colision Derecha");
96                      return true;
97
98                  }
99
100                 if (this.lateralIzquierdo() - barriles[i].lateralDerecho() < 0
101                         && this.lateralDerecho() - barriles[i].lateralDerecho() >
                            0
102                         && this.obtenerPosPies() - barriles[i].superior() >= 3
103                         && this.obtenerPosCabeza() - barriles[i].pies() <= -10) {
104                     System.out.println("[" + i + "] Colision Izquierda");
105                     return true;
106
```

```
107                 }
108
109             }
110         }
111         return false;
112
113     }
114
115     /*
116      * Hacer Sonar.
117      *
118      * Esta función ejecuta el sonido de caminar pero evita que suene en cada tick
119      * donde se está caminando. Sino habría una bola de sonido indistinguible.
120      *
121      * Se le debe indicar el momento actual en ticks como parámetro.
122      *
123      * La función decide hacer sonar alguna de las 3 variantes de sonidos de pasos
124      * que hay. Y sólo hace sonar cuando la distancia entre el sonido anterior y
         el
125      * actual es de 40 ticks.
126      *
127      */
128
129     public void hacerSonar(int contador) {
130         if (this.sonando == 3 && contador > this.sonandoDesde + 40) {
131             Herramientas.play("rsc/sonidos/caminar" + String.valueOf(this.sonando
                ) + ".wav");
132             this.sonando = 1;
133             this.sonandoDesde = contador;
134         }
135
136         else if (this.sonando < 3 && contador > this.sonandoDesde + 40) {
137
138             Herramientas.play("rsc/sonidos/caminar" + String.valueOf(this.sonando
                ) + ".wav");
139             this.sonando++;
140             this.sonandoDesde = contador;
141
142         }
143
144     }
145
146     /*
147      * Pisando
148      *
149      * Esta funcion devuelve el indice que ocupa la viga en el arreglo de
         suelos. Si
150      * no se encuentra pisando, entonces devuelve -1.
151      *
152      * Requiere que se entregue el entorno y el arreglo de vigas como parámetros.
153      *
154      * Para saber si no está pisando la viga, el centro 'y' del personaje + 20
155      * pixeles (para llegar al pie del personaje) obtenerPosPies() debe poseer un
156      * valor distinto para la coordenada 'y' donde comienza cada viga (la posy -
157      * 12px) (int)suelos[i].dondeEmpiezaElSuelo().
158      *
159      * En el caso de que el personaje se encuentra pisando la viga. Queda por
160      * conocer si se encuentra dentro de todos los puntos 'x' que conforman el
         largo
161      * de la viga.
162      *
163      * Por eso la función analiza que el extremo derecho de la viga, sea pisada
         por
```

```
164         * al menos el lateral izquierdo del personaje, y lo mismo de forma invertida.
165         * Si no se cumple esta condición, el personaje está cayendo por estar fuera
            de
166         * la viga a pesar de estar a la altura de alguna de ellas.
167         *
168         *
169         *
170         */
171
172        public int pisando(Viga[] suelos) {
173
174            if (this.obtenerEstaEnEscalera() == false) {
175                for (int i = 0; i < suelos.length; i++) {
176
177                    if (this.obtenerPosPies() == (int) suelos[i].dondeEmpiezaElSuelo
                        ()) {
178
179                        if (this.lateralDerecho() < suelos[i].extremoIzquierdo()
180                                || this.lateralIzquierdo() > suelos[i].extremoDerecho
                                ()) {
181                            this.estaCayendo = true;
182                            return -1;
183
184                        } else {
185                            this.estaCayendo = false;
186                            return i;
187
188                        }
189                    }
190
191                }
192                this.estaCayendo = true;
193                return -1;
194
195            } else {
196
197                if (this.subidoAEscaleraNro > 4) {
198                    return this.subidoAEscaleraNro - 5;
199                } else {
200                    return this.subidoAEscaleraNro;
201                }
202            }
203
204        }
205
206        /*
207         * Esta función cambia el valor de estaCercaEscalera a true o false
            dependiendo
208         * si el personaje está cerca de una escalera como para poder subir o
            descender
209         * por ella.
210         *
211         * Esta función debe llamarse en cada tick del juego pero sólo si el personaje
212         * no se encuentra dentro de una escalera actualmente.
213         *
214         */
215
216        public void estoyCercaDeEscalera(Escaleras[] escaleras, Viga[] suelos) {
217
218            int hallado = 0;
219            int i = pisando(suelos);
220
221            // Sólo analiza la proximidad de una escalera, si la función pisando
```

```
            devuelve el
222         // índice de la viga pisada.
223         // No se analiza proximidad para valores -1 (en el aire) (se está
            cayendo).
224         if (i != -1 && this.estaCayendo == false) {
225
226             // Comprobación de escaleras para todos los pisos excepto el último
227             if (i != suelos.length - 1) {
228
229                 // Se analiza una escalera que comienza en el piso actual y sube
                    al próximo
230                 if ((escaleras[i].extremoInferior() - this.obtenerPosPies() <= 5
                    )) {
231
232                     if (escaleras[i].lateralDerecho() >= this.posx && escaleras[i
                        ].lateralIzquierdo() <= this.posx) {
233                         this.estaCercaEscalera = true;
234                         this.subidoAEscaleraNro = i;
235
236                         hallado += 1;
237                     }
238                 }
239
240                 // Se analiza una escalera adicional que comienza en el piso
                    actual y sube al
241                 // próximo o quizas no sube del todo
242                 if ((escaleras[i + 5].extremoInferior() - this.obtenerPosPies()
                    <= 5)) {
243
244                     if (escaleras[i + 5].lateralDerecho() >= this.posx
245                             && escaleras[i + 5].lateralIzquierdo() <= this.posx) {
246                         this.estaCercaEscalera = true;
247                         this.subidoAEscaleraNro = i + 5;
248
249                         hallado += 1;
250                     }
251                 }
252
253             }
254
255             // Comprobación de escaleras para todos los pisos excepto la planta
                baja
256             if (i != 0) {
257                 if (escaleras[i - 1].extremoSuperior() - this.obtenerPosPies() <=
                     10) {
258
259                     // Se analiza una escalera que termina en el piso actual y
                        desciende al inferior
260                     if (escaleras[i - 1].lateralDerecho() >= this.posx
261                             && escaleras[i - 1].lateralIzquierdo() <= this.posx) {
262                         this.estaCercaEscalera = true;
263                         this.subidoAEscaleraNro = i - 1;
264
265                         hallado += 1;
266                     }
267                 }
268
269                 if (escaleras[i + 4].extremoSuperior() - this.obtenerPosPies() <=
                     10) {
270
271                     // Se analiza una escalera adicional que termina en el piso
                        actual y desciende
272                     // al inferior
```

```
273                     if (escaleras[i + 4].lateralDerecho() >= this.posx
274                             && escaleras[i + 4].lateralIzquierdo() <= this.posx) {
275                         this.estaCercaEscalera = true;
276                         this.subidoAEscaleraNro = i + 4;
277 
278                         hallado += 1;
279                     }
280                 }
281 
282             }
283 
284         }
285 
286         if (hallado == 0) {
287             this.estaCercaEscalera = false;
288 
289         }
290 
291     }
292 
293     /*
294      * Realiza los calculos geométricos para saber que un barril fue correctamente
295      * saltado.
296      */
297 
298     public boolean saltandoBarril(Barril barril) {
299 
300         if ((this.posx + 1 == barril.centroX() || this.posx - 1 == barril.centroX
        () || this.posx == barril.centroX())
301                 && this.obtenerPosPies() - barril.superior() <= 0 && this.
                obtenerPosPies() - barril.superior() > -50
302                 && barril.fueSaltado() == false && this.obtenerEstaEnEscalera()
                == false) {
303 
304             Herramientas.play("rsc/sonidos/salta_barril.wav");
305             barril.saltado();
306             return true;
307 
308         } else {
309             return false;
310         }
311 
312     }
313 
314     /*
315      * Dibuja al personaje. Se deben calcular las situaciones y cambiar las
316      * variables previamente con otros métodos.
317      */
318 
319     public void dibujar(Entorno entorno, int rotacion) {
320         entorno.dibujarImagen(imagenMario, this.posx, this.posy, rotacion, 0.090);
321 
322     }
323 
324     /*
325      * Devuelven la posicion extrema lateral correspondiente
326      */
327     public int lateralDerecho() {
328         return posx + 15;
329     }
330 
331     public int lateralIzquierdo() {
332         return posx - 15;
```

```
333         }
334 
335         /*
336          * Setters de las posiciones X e Y
337          */
338         public void cambiarY(int pixeles) {
339             this.posy = this.posy + pixeles;
340         }
341 
342         public void cambiarX(int pixeles) {
343             this.posx = this.posx + pixeles;
344         }
345 
346         /*
347          * Devuelven la posicion extremas verticales correspondiente
348          */
349 
350         public int obtenerPosPies() {
351             return this.posy + 20;
352         }
353 
354         public int obtenerPosCabeza() {
355             return this.posy - 20;
356         }
357 
358         /*
359          * Getters
360          */
361 
362         public boolean obtenerEstaEnEscalera() {
363             return this.estaEnEscalera;
364         }
365 
366         public int obtenerMomentoDeSalto() {
367             return this.tiempoSalto;
368         }
369 
370         public boolean obtenerEstaCayendo() {
371             return this.estaCayendo;
372         }
373 
374         public boolean obtenerEstaCercaEscalera() {
375             return this.estaCercaEscalera;
376         }
377 
378         public int obtenerSubidoAEscaleraNro() {
379             return this.subidoAEscaleraNro;
380         }
381 
382         public boolean obtenerMiraDerecha() {
383             return this.miraDerecha;
384         }
385 
386         public boolean obtenerEstaSaltando() {
387             return this.estaSaltando;
388         }
389 
390         /*
391          * Setters
392          */
393 
394         public void cambiarMomentoDeSalto(int i) {
395             this.tiempoSalto = i;
```

```
396
397         }
398
399         public void cambiarEstaEnEscalera(boolean escalera) {
400             this.estaEnEscalera = escalera;
401         }
402
403         public void cambiarMiraDerecha(boolean mira) {
404             this.miraDerecha = mira;
405         }
406
407         public void cambiarEstaSaltando(boolean salta) {
408             this.estaSaltando = salta;
409         }
410
411         /*
412          * Ayudan a cambiar por la imagen correcta, según el String indicado, y
             según a
413          * que lado esté mirando el personaje
414          */
415         public void cambiarImagen(String s) {
416
417             if (s.equals("mirando") && !this.imagenMario.equals(mirandoIzquierda) &&
             !this.miraDerecha) {
418                 imagenMario = mirandoIzquierda;
419             }
420
421             else if (s.equals("mirando") && !this.imagenMario.equals(mirandoDerecha)
             && this.miraDerecha) {
422                 imagenMario = mirandoDerecha;
423             }
424
425             else if (s.equals("caminando") && !this.imagenMario.equals(
             caminandoIzquierda) && !this.miraDerecha) {
426                 imagenMario = caminandoIzquierda;
427             }
428
429             else if (s.equals("caminando") && !this.imagenMario.equals(
             caminandoDerecha) && this.miraDerecha) {
430                 imagenMario = caminandoDerecha;
431             }
432
433             else if (s.equals("saltando") && !this.imagenMario.equals(
             saltandoIzquierda) && !this.miraDerecha) {
434                 imagenMario = saltandoIzquierda;
435             }
436
437             else if (s.equals("saltando") && !this.imagenMario.equals(saltandoDerecha
             ) && this.miraDerecha) {
438                 imagenMario = saltandoDerecha;
439             }
440
441             else if (s.equals("subiendo") && !this.imagenMario.equals(subiendo)) {
442                 imagenMario = subiendo;
443             } else if (s.equals("quieto") && !this.imagenMario.equals(subiendo_quieto
             )) {
444                 imagenMario = subiendo_quieto;
445             }
446
447         }
448
449     }
450
```