```java
package juego;

import juego.Viga;

public class Personaje {

    private String estado;
    private int posx;
    private int posy;

    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;

    private char ultima; // ultima tecla de sentido (DER o IZQ) presionada (Sirve para saber para donde
                         // debe mirar el personaje).

    private int tiempoSalto; // tick en el cual se ejecutó el último salto (o el 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 no están tocando viga
                                 // alguna.

    private boolean estaCercaEscalera;
    private boolean estaEnEscalera;
    private int enEscalera;

    private int sonando = 1; // Ultimo archivo de sonido que se usó para caminar, hay 3 variantes.
    private int sonandoDesde = 0; // tick en el cual se ejecutó el último sonido de caminar (ayuda a evitar que
                                  // suenen sonidos en cada tick)

    public Personaje(Viga vigasuelo) {

        this.estado = "vivo";
        this.posx = 50;
        this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 35; // 35 pixeles por encima de la viga inicial, genera una
                                                                // linda caida en el spawn

        this.mirandoIzquierda = Herramientas.cargarImagen("rsc/graficos/marito/mira-izquierda.png");
        this.mirandoDerecha = Herramientas.cargarImagen("rsc/graficos/marito/mira-derecha.png");

        this.caminandoIzquierda = Herramientas.cargarImagen("rsc/graficos/marito/camina-izquierda.gif");
        this.caminandoDerecha = Herramientas.cargarImagen("rsc/graficos/marito/camina-derecha.gif");
```

```
58
59          this.saltandoIzquierda =
  Herramientas.cargarImagen("rsc/graficos/marito/salta-izquierda.png");
60          this.saltandoDerecha =
  Herramientas.cargarImagen("rsc/graficos/marito/salta-derecha.png");
61
62          this.subiendo = Herramientas.cargarImagen("rsc/graficos/marito/subiendo.gif");
63          this.subiendo_quieto =
  Herramientas.cargarImagen("rsc/graficos/marito/quieto_subiendo.png");
64
65          this.tiempoSalto = 0;
66          this.estaSaltando = false;
67          this.estaCayendo = false;
68          this.estaEnEscalera = false;
69          this.estaCercaEscalera = false;
70          this.ultima = 39;
71
72      }
73
74      public boolean tocando(Barril[] barriles) {
75
76          for (int i = 0; i < barriles.length; i++) {
77
78              if (barriles[i] != null) {
79
80                  if (this.lateralDerecho() - barriles[i].lateralIzquierdo() > 0
81                          && this.lateralIzquierdo() - barriles[i].lateralIzquierdo() < 0
82                          && this.pies() - barriles[i].pies() >= 0 && this.cabeza() -
  barriles[i].pies() <= 0
83
84                  ) {
85                      System.out.println("[" + i + "] Colision Derecha");
86                      return true;
87
88                  }
89
90                  if (this.lateralIzquierdo() - barriles[i].lateralDerecho() < 0
91                          && this.lateralDerecho() - barriles[i].lateralDerecho() > 0
92                          && this.pies() - barriles[i].pies() >= 0 && this.cabeza() -
  barriles[i].pies() <= 0
93
94                  ) {
95                      System.out.println("[" + i + "] Colision Izquierda");
96                      return true;
97
98                  }
99
100             }
101         }
102         return false;
103
104     }
105
106     /*
107      * Hacer Sonar.
108      *
109      * Esta función ejecuta el sonido de caminar pero evita que suene en cada tick
110      * donde se está caminando. Sino habría una bola de sonido indistinguible.
111      *
112      * Se le debe indicar el momento actual en ticks como parámetro.
113      *
114      * La función decide hacer sonar alguna de las 3 variantes de sonidos de pasos
```

```
115      * que hay. Y sólo hace sonar cuando la distancia entre el sonido anterior y el
116      * actual es de 40 ticks.
117      *
118      */
119
120     public void hacerSonar(int contador) {
121         if (this.sonando == 3 && contador > this.sonandoDesde + 40) {
122             Herramientas.play("rsc/sonidos/caminar" + String.valueOf(this.sonando) +
    ".wav");
123             this.sonando = 1;
124             this.sonandoDesde = contador;
125         }
126
127         else if (this.sonando < 3 && contador > this.sonandoDesde + 40) {
128
129             Herramientas.play("rsc/sonidos/caminar" + String.valueOf(this.sonando) +
    ".wav");
130             this.sonando++;
131             this.sonandoDesde = contador;
132
133         }
134
135     }
136
137     /*
138      * Saltar
139      *
140      * La función saltar se encarga de la parte de un salto que se ejecuta una sola
141      * vez. Es decir que no se encarga de la animación de subida o caida a lo largo
142      * de los ticks de un salto normal.
143      *
144      * Se le debe indicaar el entorno y el contador de ticks actual.
145      *
146      * Cambia el dibujo de caminar por el salto, según hacia que lado este mirando
147      * el personaje. Cambia el estado de estaSaltando a verdadero. Ejecuta el sonido
148      * del salto. Indica el tick en el cual se realizó el salto, guardando el valor
149      * en tiempoSalto.
150      *
151      */
152
153     public void saltar(Entorno entorno, int contador) {
154
155         if (this.ultima == entorno.TECLA_DERECHA) {
156
157             entorno.dibujarImagen(saltandoDerecha, this.posx, this.posy, 0, 0.090);
158
159             this.tiempoSalto = contador;
160             this.estaSaltando = true;
161             Herramientas.play("rsc/sonidos/jump.wav");
162
163         } else {
164
165             entorno.dibujarImagen(saltandoIzquierda, this.posx, this.posy, 0, 0.090);
166
167             this.tiempoSalto = contador;
168             this.estaSaltando = true;
169             Herramientas.play("rsc/sonidos/jump.wav");
170
171         }
172
173     }
174
```

```
175     /*
176      * Saltando
177      *
178      * Esta función se encarga de manipular, a lo largo del tiempo, lo que ocurre
179      * con el personaje cuando no está en el suelo.
180      *
181      * Se la llama por cada tick.
182      *
183      * Requiere el entorno, el contador actual y el arreglo con las vigas.
184      *
185      * Si el momento actual se produce con menos de 30 ticks de diferencia, entonces
186      * hay que elevar 1px al jugador (restar 1 en eje 'y').
187      *
188      * De lo contrario analiza si NO está pisando alguna viga. Si no está pisando
189      * vigas, entonces debe descender un pixel por cada tick, hasta que pise alguna
190      * viga.
191      */
192
193     public void saltando(Entorno entorno, int contador, Viga[] suelos) {
194         if (this.estaEnEscalera == false && this.estado.equals("vivo")) {
195
196             if (estaSaltando && contador - this.tiempoSalto < 30) {
197
198                 if (this.ultima == entorno.TECLA_DERECHA) {
199
200                     this.posy = this.posy - 1;
201                     entorno.dibujarImagen(saltandoDerecha, this.posx, this.posy, 0, 0.090);
202
203                 } else {
204
205                     this.posy = this.posy - 1;
206                     entorno.dibujarImagen(saltandoIzquierda, this.posx, this.posy, 0,
    0.090);
207
208                 }
209
210             } else {
211
212                 this.estaSaltando = false;
213
214                 if (pisando(entorno, suelos) == -1) {
215
216                     if (this.ultima == entorno.TECLA_DERECHA) {
217
218                         this.posy = this.posy + 1;
219                         entorno.dibujarImagen(saltandoDerecha, this.posx, this.posy, 0,
    0.090);
220                     }
221
222                     else {
223
224                         this.posy = this.posy + 1;
225                         entorno.dibujarImagen(saltandoIzquierda, this.posx, this.posy, 0,
    0.090);
226
227                     }
228
229                 }
230
231             }
232         }
233     }
```

```
234
235     /*
236      * Pisando
237      *
238      * Esta funcion devuelve el indice que ocupa la viga en el arreglo de suelos. Si
239      * no se encuentra pisando, entonces devuelve -1.
240      *
241      * Requiere que se entregue el entorno y el arreglo de vigas como parámetros.
242      *
243      * Para saber si no está pisando la viga, el centro 'y' del personaje + 20
244      * pixeles (para llegar al pie del personaje) pies() debe poseer un valor
245      * distinto para la coordenada 'y' donde comienza cada viga (la posy - 12px)
246      * (int)suelos[i].dondeEmpiezaElSuelo().
247      *
248      * En el caso de que el personaje se encuentra pisando la viga. Queda por
249      * conocer si se encuentra dentro de todos los puntos 'x' que conforman el largo
250      * de la viga.
251      *
252      * Por eso la función analiza que el extremo derecho de la viga, sea pisada por
253      * al menos el lateral izquierdo del personaje, y lo mismo de forma invertida.
254      * Si no se cumple esta condición, el personaje está cayendo por estar fuera de
255      * la viga a pesar de estar a la altura de alguna de ellas.
256      *
257      *
258      *
259      */
260
261     public int pisando(Entorno entorno, Viga[] suelos) {
262
263         if (this.estaEnEscalera == false) {
264             for (int i = 0; i < suelos.length; i++) {
265
266                 if (this.pies() == (int) suelos[i].dondeEmpiezaElSuelo()) {
267
268                     if (this.lateralDerecho() < suelos[i].extremoIzquierdo()
269                             || this.lateralIzquierdo() > suelos[i].extremoDerecho()) {
270                         this.estaCayendo = true;
271                         return -1;
272
273                     } else {
274                         this.estaCayendo = false;
275                         return i;
276
277                     }
278                 }
279
280             }
281             this.estaCayendo = true;
282             return -1;
283
284         } else {
285             return this.enEscalera;
286         }
287
288     }
289
290     // Devuelve un entero con el valor que ocupan los pies del personaje en el eje
291     // 'y'
292
293     public int pies() {
294         return this.posy + 20;
295     }
```

```
296
297     public int cabeza() {
298         return this.posy - 20;
299     }
300
301     /*
302      * Dibujar
303      *
304      * Esta función detecta las teclas presionadas y según condiciones ejecuta las
305      * acciones que debe realizar el personaje.
306      *
307      * Se la debe llamar en cada tick
308      *
309      * Recibe como parámetro el entorno y el momento actual medido en ticks.
310      *
311      * Como prioridad, deteca si el usuario solicita saltar, presionando la
312      * espaciadora. Pero solo permite ejecutar dicha acción, si desde la última vez
313      * que saltó pasaron más de 60 tics (lo que requiere como mínimo un salto). Y a
314      * su vez, que el personaje no esté cayendo.
315      *
316      *
317      * Continúa evaluando si se presionan las teclas derecha e izquierda y ejecuta
318      * dichos movimientos, pero sólo si el personaje no está saltando ni tampoco
319      * esta cayendo. ## Este juego no permite desplazarse de izq a der mientras se
320      * está en el aire.
321      *
322      * Sólo permite desplazarse a los costados, si el jugador no sale de pantalla.
323      *
324      * Luego, si ninguna tecla está siendo presionada, deja al jugador mirando hacia
325      * el lado que corresponde según el último movimiento.
326      */
327
328     public void dibujar(Entorno entorno, int contador, Escaleras[] escaleras) {
329
330         if (!this.estado.equals("vivo")) {
331             this.posy = this.posy + 3;
332             entorno.dibujarImagen(saltandoDerecha, this.posx, this.posy, 90, 0.090);
333
334         } else {
335
336             // Unica forma de saltar (saltando siempre que no haya sido muy pronto desde el
337             // salto anterior y no se esté cayendo
338             if (entorno.sePresiono(entorno.TECLA_ESPACIO) && this.tiempoSalto + 60 <
    contador
339                     && this.estaCayendo == false && this.estaEnEscalera == false) {
340
341                 this.saltar(entorno, contador);
342
343             }
344
345             // unica forma de pasar a estar dentro de una escalera (estando cerca de una
346             // escalera pero no dentro de una)
347             if (this.estaCercaEscalera == true && this.estaEnEscalera == false) {
348
349                 if (entorno.sePresiono(entorno.TECLA_ARRIBA)
350                         && this.pies() > escaleras[this.enEscalera].extremoSuperior()) {
351
352                     this.subirEscaleras(entorno, escaleras);
353                 }
354
355                 else if (entorno.sePresiono(entorno.TECLA_ABAJO)
356                         && this.pies() < escaleras[this.enEscalera].extremoInferior()) {
```

```
357
358                     this.bajarEscaleras(entorno, escaleras);
359                 }
360
361             }
362
363             // unica forma de moverse ya dentro de una escalera (estar cerca de una y ya
364             // dentro de una)
365             if (this.estaCercaEscalera == true && this.estaEnEscalera == true) {
366
367                 if (entorno.estaPresionada(entorno.TECLA_ARRIBA)) {
368
369                     this.subirEscaleras(entorno, escaleras);
370                 }
371
372                 else if (entorno.estaPresionada(entorno.TECLA_ABAJO)) {
373
374                     this.bajarEscaleras(entorno, escaleras);
375                 } else {
376
377                     entorno.dibujarImagen(subiendo_quieto, this.posx, this.posy, 0, 0.090);
378
379                 }
380             }
381
382             // unica forma de moverse de izquierda a derecha (no estar cayendo ni saltando
383             // ni dentro de una escalera)
384             if (this.estaCayendo == false && this.estaSaltando == false &&
    this.estaEnEscalera == false) {
385
386                 // caminar a derecha
387                 if (entorno.estaPresionada(entorno.TECLA_DERECHA)) {
388
389                     if (this.posx <= 790) {
390                         this.posx = this.posx + 2;
391                     }
392
393                     entorno.dibujarImagen(caminandoDerecha, this.posx, this.posy, 0,
    0.090);
394                     hacerSonar(contador);
395                     this.ultima = entorno.TECLA_DERECHA;
396                 }
397
398                 // caminar a izquierda
399                 else if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
400
401                     if (this.posx >= 10) {
402                         this.posx = this.posx - 2;
403                     }
404
405                     entorno.dibujarImagen(caminandoIzquierda, this.posx, this.posy, 0,
    0.090);
406                     hacerSonar(contador);
407                     this.ultima = entorno.TECLA_IZQUIERDA;
408                 }
409
410                 // mirar hacia el ultimo lado caminado
411                 else {
412
413                     if (this.ultima == entorno.TECLA_DERECHA) {
414                         entorno.dibujarImagen(mirandoDerecha, this.posx, this.posy, 0,
    0.090);
```

```
415                     } else {
416                         entorno.dibujarImagen(mirandoIzquierda, this.posx, this.posy, 0,
    0.090);
417                     }
418 
419                 }
420 
421             }
422         }
423 
424     }
425 
426     /*
427      * Esta función cambia el valor de estaCercaEscalera a true o false dependiendo
428      * si el personaje está cerca de una escalera como para poder subir o descender
429      * por ella.
430      *
431      * Esta función debe llamarse en cada tick del juego pero sólo si el personaje
432      * no se encuentra dentro de una escalera actualmente.
433      *
434      */
435 
436     public void estoyCercaDeEscalera(Entorno entorno, Escaleras[] escaleras, Viga[] suelos)
    {
437 
438         int hallado = 0;
439         int i = pisando(entorno, suelos);
440 
441         // Sólo analiza la proximidad de una escalera, si la función pisando devuelve el
442         // índice de la viga pisada.
443         // No se analiza proximidad para valores -1 (en el aire) ni si se está cayendo.
444         if (i != -1 && this.estaCayendo == false) {
445 
446             // Comprobación de escaleras para todos los pisos excepto el último
447             if (i != suelos.length - 1) {
448 
449                 // Se analiza una escalera que comienza en el piso actual y sube al próximo
450                 if ((escaleras[i].extremoInferior() - this.pies() <= 5)) {
451 
452                     if (escaleras[i].lateralDerecho() >= this.posx &&
    escaleras[i].lateralIzquierdo() <= this.posx) {
453                         this.estaCercaEscalera = true;
454                         this.enEscalera = i;
455 
456                         hallado += 1;
457                     }
458                 }
459             }
460 
461             // Comprobación de escaleras para todos los pisos excepto la planta baja
462             if (i != 0) {
463                 if (escaleras[i - 1].extremoSuperior() - this.pies() <= 10) {
464 
465                     // Se analiza una escalera que termina en el piso actual y desciende al
    inferior
466                     if (escaleras[i - 1].lateralDerecho() >= this.posx
467                             && escaleras[i - 1].lateralIzquierdo() <= this.posx) {
468                         this.estaCercaEscalera = true;
469                         this.enEscalera = i - 1;
470 
471                         hallado += 1;
472                     }
```

```
473                 }
474             }
475 
476         }
477 
478         if (hallado == 0) {
479             this.estaCercaEscalera = false;
480 
481         }
482 
483     }
484 
485     /*
486      * Esta función ejecuta las animaciones correspondiente a subir escalera y se
487      * encarga de informar si ya terminó de subirla. Es decir que sale de la
488      * escalera y se encuentra en el piso superior.
489      *
490      */
491 
492     public void subirEscaleras(Entorno entorno, Escaleras[] escaleras) {
493 
494         if (this.pies() < escaleras[this.enEscalera].extremoSuperior() &&
    this.estaEnEscalera == true) {
495             // entorno.dibujarImagen(subio, this.posx, this.posy, 0, 0.20);
496             this.estaEnEscalera = false;
497 
498             if (this.enEscalera % 2 == 0) {
499                 this.ultima = entorno.TECLA_IZQUIERDA;
500                 entorno.dibujarImagen(mirandoIzquierda, this.posx, this.posy, 0, 0.090);
501 
502             } else {
503                 this.ultima = entorno.TECLA_DERECHA;
504                 entorno.dibujarImagen(mirandoDerecha, this.posx, this.posy, 0, 0.090);
505 
506             }
507 
508         } else {
509 
510             this.posy = this.posy - 2;
511             this.estaEnEscalera = true;
512             entorno.dibujarImagen(subiendo, this.posx, this.posy, 0, 0.090);
513         }
514     }
515 
516     /*
517      * Esta función ejecuta las animaciones correspondiente a bajar escalera y se
518      * encarga de informar si ya terminó de descender. Es decir que sale de la
519      * escalera y se encuentra en el piso inferior.
520      *
521      */
522 
523     public void bajarEscaleras(Entorno entorno, Escaleras[] escaleras) {
524 
525         if (this.pies() >= escaleras[this.enEscalera].extremoInferior() - 5 &&
    this.estaEnEscalera == true) {
526             this.estaEnEscalera = false;
527 
528             if (this.enEscalera % 2 == 0) {
529                 this.ultima = entorno.TECLA_DERECHA;
530             } else {
531                 this.ultima = entorno.TECLA_IZQUIERDA;
532             }
```

```
533         } else {
534 
535             this.posy = this.posy + 2;
536             this.estaEnEscalera = true;
537             entorno.dibujarImagen(subiendo, this.posx, this.posy, 0, 0.090);
538         }
539 
540     }
541 
542     public int lateralDerecho() {
543         return posx + 15;
544     }
545 
546     public int lateralIzquierdo() {
547         return posx - 15;
548     }
549 
550     public boolean estaEnEscalera() {
551         return this.estaEnEscalera;
552     }
553 
554     public void morir() {
555         this.estado = "muerto";
556     }
557 
558     /*
559      * Retorna verdadero sólo cuando el jugador se encuentra en una posición x igual
560      * o menor a 150 y a la vez en la última viga del arreglo (donde se encuentra
561      * donkey).
562      */
563 
564     public boolean ganar(Entorno entorno, Viga[] suelos) {
565         if (this.pisando(entorno, suelos) == suelos.length - 1 && this.lateralIzquierdo()
    <= 150) {
566             return true;
567         } else {
568             return false;
569         }
570 
571     }
572 
573     public boolean saltandoBarril(Barril barril) {
574 
575         if ((this.posx + 1 == barril.centroX() || this.posx - 1 == barril.centroX() ||
    this.posx == barril.centroX())
576                 && this.pies() - barril.superior() <= 0 && this.pies() - barril.superior()
    > -50
577                 && barril.fueSaltado() == false && this.estaEnEscalera == false) {
578             barril.saltado();
579             return true;
580 
581         }
582 
583         else {
584             return false;
585         }
586 
587     }
588 
589 }
590 
```