```java
package juego;

import java.awt.Color;

import entorno.Entorno;
import entorno.Herramientas;
import entorno.InterfaceJuego;

public class Juego extends InterfaceJuego {

    // El objeto Entorno que controla el tiempo y otros
    private Entorno entorno;

    // Puede ser "juganddo", "ganado" o "perdido"
    private String estadoDelJuego = "jugando";

        // Creación del arreglo de vigas
    static Viga suelos[] = new Viga[] {

            new Viga(1), new Viga(2), new Viga(3), new Viga(4), new Viga(5),
            new Viga(6)

    };

    // Creación del arreglo de escaleras
    static Escaleras escaleras[] = new Escaleras[] {
            new Escaleras(0, suelos), new Escaleras(1, suelos),
            new Escaleras(2, suelos), new Escaleras(3, suelos),
            new Escaleras(4, suelos), new Escaleras(5, suelos),
            new Escaleras(6, suelos), new Escaleras(7, suelos),
            new Escaleras(8, suelos), new Escaleras(9, suelos) };

    // Antagonista y Personaje principal
    private Donkey donkeyKong = new Donkey();
    private Personaje jugador = new Personaje();

    // Puntuador
    private Puntaje puntuador = new Puntaje();

    // El reloj medido en ticks
    int contador = 0;

    // Creación del arreglo de barriles
    private Barril barriles[] = new Barril[]{
            new Barril(suelos[suelos.length - 3]),
            null, null, null, null, null,
            null, null, null, null, null,
            null, null, null, null, null,
            null, null, null, null

    };

    // ...

    Juego() {
        // Inicializa el objeto entorno
        this.entorno = new Entorno(this, "Donkey - Grupo Pereira - Sanchez -
        Tula - V2", 800, 600);

        // Inicializar lo que haga falta para el juego
        // ...

        Herramientas.loop("rsc/sonidos/musica.wav");

        // Inicia el juego!
        this.entorno.iniciar();

```

```
66      }
67
68      /**
69       * Durante el juego, el método tick() será ejecutado en cada instante y
         por lo
70       * tanto es el método más importante de esta clase. Aquí se debe
         actualizar el
71       * estado interno del juego para simular el paso del tiempo (ver el
         enunciado
72       * del TP para mayor detalle).
73       */
74      public void tick() {
75
76
77          // Al inicio de cada ciclo aumentar una unidad el reloj
78          contador++;
79
80
81
82          // Ejecuta la función dibujar por cada miembro del arreglo de vigas.
83          for (int i = 0; i < suelos.length; i++) {
84              suelos[i].dibujar(entorno);
85          }
86
87          // Ejecuta la función dibujar por cada miembro del arreglo de
            escaleras.
88          for (int i = 0; i < escaleras.length; i++) {
89              escaleras[i].dibujar(entorno);
90          }
91
92          // Ejecuta la función dibujar para donkey
93          donkeyKong.gorilear(entorno, contador);
94
95          // Ejecuta la función dibujar para el contador
96          puntuador.dibujar(entorno);
97
98
99          // Ejecuta la función dibujar por cada elemento no NULL del arreglo
            de barriles
100         // , también analiza si un barril debe destruirse.
101         for (int i = 0; i < barriles.length; i++) {
102             if (barriles[i] != null) {
103
104                 barriles[i].dibujar(entorno, contador, suelos, escaleras);
105                 if (barriles[i].deboDestruirme(entorno, suelos)) {
106                     barriles[i] = null;
107                 }
108
109             }
110         }
111
112
113
114         /*
115          * Analisis que ocurren mientras el juego se desarrolla
116          * Es decir que el jugador no ganó ni perdió aún.
117          */
118         if (this.estadoDelJuego.equals("jugando")) {
119
120
121             // Analizar si el personaje se encuentra cerca de escalera
122             jugador.estoyCercaDeEscalera(escaleras, suelos);
123
124
125
126             /*
127              * CAER
```

```
128                 * NO escalera, NO saltando, NO pisando
129                 */
130                if (!jugador.obtenerEstaEnEscalera() && !jugador.
                   obtenerEstaSaltando() && jugador.pisando(suelos) == -1) {
131
132
133                    jugador.cambiarImagen("saltando");
134                    jugador.cambiarY(1);
135
136                }
137
138
139                /*
140                 * SALTAR (Parte del proceso de saltar de una única ejecución)
141                 * Si presionada tecla espacio, Salto anterior dista más de 60
                    ticks, No pisando, No está en Escalera
142                 */
143                if (entorno.sePresiono(entorno.TECLA_ESPACIO) && jugador.
                   obtenerMomentoDeSalto() + 60 < contador
144                        && jugador.pisando(suelos) != -1 && !jugador.
                           obtenerEstaEnEscalera()) {
145
146                    /*
147                     * Esta es la parte de un salto que se ejecuta una sola vez.
                        Es decir que no se
148                     * encarga de la animación de subida o caida a lo largo de
                        los ticks de un salto
149                     * normal.
150                     *
151                     * Cambia el estado de estaSaltando a verdadero. Ejecuta el
                        sonido
152                     * del salto. Indica el tick en el cual se realizó el salto,
                        guardando el valor
153                     * en tiempoSalto.
154                     *
155                     */
156                    jugador.cambiarImagen("saltando");
157                    jugador.cambiarMomentoDeSalto(contador);
158                    jugador.cambiarEstaSaltando(true);
159                    Herramientas.play("rsc/sonidos/jump.wav");
160
161                }
162
163
164                /*
165                 * INGRESAR A ESCALERA
166                 * La única forma de pasar a estar dentro de una escalera
                    (estando cerca de una
167                 * escalera pero no dentro de una)
168                 */
169                if (jugador.obtenerEstaCercaEscalera() && !jugador.
                   obtenerEstaEnEscalera()) {
170
171                    // Entrar subiendo la escalera
172                    if (entorno.sePresiono(entorno.TECLA_ARRIBA) && jugador
173                            .obtenerPosPies() > escaleras[jugador.
                               obtenerSubidoAEscaleraNro()].extremoSuperior()) {
174
175                        jugador.cambiarEstaEnEscalera(true);
176                        jugador.cambiarY(-2);
177                        jugador.cambiarImagen("subiendo");
178
179
180
181                    }
182
183                    // Entrar bajando la escalera
```

```
184                 else if (entorno.sePresiono(entorno.TECLA_ABAJO) && jugador
185                         .obtenerPosPies() < escaleras[jugador.
                            obtenerSubidoAEscaleraNro()].extremoInferior()) {
186
187                     jugador.cambiarEstaEnEscalera(true);
188                     jugador.cambiarY(2);
189                     jugador.cambiarImagen("subiendo");
190
191
192
193                 }
194             }
195
196
197             /*
198              * Moverse dentro de una escalera. Estando dentro de una.
199              */
200             if (jugador.obtenerEstaEnEscalera()) {
201
202
203                 /*
204                  * SUBIR ESCALERA
205                  * Esta función ejecuta las animaciones correspondiente a
                    subir escalera y se
206                  * encarga de informar si ya terminó de subirla. Es decir que
                    sale de la
207                  * escalera y se encuentra en el piso superior.
208                  *
209                  */
210                 if (entorno.estaPresionada(entorno.TECLA_ARRIBA)) {
211
212                     // Subió tanto la esclaera que salió al piso superior
213                     if (jugador.obtenerPosPies() < escaleras[jugador.
                        obtenerSubidoAEscaleraNro()].extremoSuperior()) {
214
215                         // Ya no está en escalera
216                         jugador.cambiarEstaEnEscalera(false);
217
218                         // Para las vigas con indice par, el personaje debe
                            ir hacia izquierda. Para las impares, hacia la derecha.
219                         if (jugador.obtenerSubidoAEscaleraNro() % 2 == 0) {
220                             jugador.cambiarMiraDerecha(false);
221                         } else {
222                             jugador.cambiarMiraDerecha(true);
223                         }
224
225                     }
226
227                     // Si aún no salió de escalera, solamente se desplaza
                        hacia arriba.
228                     else {
229
230                         jugador.cambiarY(-2);
231                         jugador.cambiarEstaEnEscalera(true);
232
233                     }
234
235                 }
236
237                 /*
238                  * BAJAR ESCALERA
239                  * Esta función ejecuta las animaciones correspondiente a
                    subir escalera y se
240                  * encarga de informar si ya terminó de subirla. Es decir que
                    sale de la
241                  * escalera y se encuentra en el piso superior.
242                  *
```

```
243                  */
244                 else if (entorno.estaPresionada(entorno.TECLA_ABAJO)) {
245 
246                     // Bajó tanto la esclaera que salió al piso inferior
247                     if (jugador.obtenerPosPies() >= escaleras[jugador.
                    obtenerSubidoAEscaleraNro()].extremoInferior()
248                             - 5) {
249 
250                         // Ya no está en escalera
251                         jugador.cambiarEstaEnEscalera(false);
252 
253                         // Para las vigas con indice par, el personaje debe
                        ir hacia izquierda. Para las impares, hacia la derecha.
254                         if (jugador.obtenerSubidoAEscaleraNro() % 2 == 0) {
255                             jugador.cambiarMiraDerecha(false);
256                         } else {
257                             jugador.cambiarMiraDerecha(true);
258                         }
259 
260                         // Si aún no salió de escalera, solamente se desplaza
                        hacia abajo.
261                     } else {
262 
263                         jugador.cambiarY(2);
264                         jugador.cambiarEstaEnEscalera(true);
265 
266                     }
267 
268 
269 
270                     /*
271                      * Si está en escalera pero el usuario no presionó ni la
                     tecla Arriba ni la tecla abajo, se queda inmóvil.
272                      */
273                 } else {
274 
275                     jugador.cambiarImagen("quieto");
276 
277                 }
278 
279             }
280 
281 
282 
283             /*
284              * SALTO
285              * Resto de la animación del salto.
286              *
287              * Si no está en escalera,
288              * Si se está saltando y el momento actual dista a menos de 30
                ticks del inicio del salto:
289              * Se está en la parte ascendente del salto.
290              *
291              * Sino, ya no se está saltando (Se informa que ya no se está
                saltando).
292              * La caida se produce por el primer llamado CAER
293              *
294              */
295             if (!jugador.obtenerEstaEnEscalera()) {
296 
297                 if (jugador.obtenerEstaSaltando() && contador - jugador.
                obtenerMomentoDeSalto() < 30) {
298 
299                     jugador.cambiarImagen("saltando");
300                     jugador.cambiarY(-1);
301 
302                 } else {
```

```
303                     jugador.cambiarEstaSaltando(false);
304
305                 }
306             }
307
308
309             /*
310              * DESPLAZARSE
311              * Solo cuando no se cae, no se está saltando y no está en escalera
312              */
313             if (!jugador.obtenerEstaCayendo() && !jugador.obtenerEstaSaltando
                () && !jugador.obtenerEstaEnEscalera()) {
314
315                 // Moverse a Derecha
316                 if (entorno.estaPresionada(entorno.TECLA_DERECHA) && jugador.
                    lateralDerecho() <= 800) {
317                     jugador.cambiarX(2);
318                     jugador.cambiarImagen("caminando");
319                     jugador.hacerSonar(contador);
320                     jugador.cambiarMiraDerecha(true);
321
322                 }
323
324                 // Moverse a izquierda
325                 else if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA) &&
                    jugador.lateralIzquierdo() >= 0) {
326
327                     jugador.cambiarX(-2);
328                     jugador.cambiarImagen("caminando");
329                     jugador.hacerSonar(contador);
330                     jugador.cambiarMiraDerecha(false);
331
332                 }
333
334                 // No moverse (en el suelo)
335                 else {
336
337                     jugador.cambiarImagen("mirando");
338
339                 }
340
341             }
342
343             /*
344              * Luego de analizar todas las posibles situaciones cambiantes
                por cada tick en el personaje
345              * Se ejecuta la función dibujar, que toma el estado de ciertas
                variables para producir
346              * la imagen correcta del personaje.
347              */
348             jugador.dibujar(entorno, 0);
349
350
351             /*
352              * GANAR
353              * Acercarse a determinada posición del juego (sin haber perdido)
354              * Genera la victoria automática.
355              */
356             if (jugador.pisando(suelos) == suelos.length - 1 && jugador.
                lateralIzquierdo() <= 150) {
357                 this.estadoDelJuego = "ganado";
358                 puntuador.ganar();
359             }
360
361
362             /*
363              * PERDER
```

```
364                 * Si la función que reporta si el personaje tocó algún barril da
                    verdadero
365                 * Se genera la derrota automática.
366                 */
367                if (jugador.tocando(barriles)) {
368                    this.estadoDelJuego = "perdido";
369                }
370
371                /*
372                 * PUNTUACIÓN
373                 * Por cada barril se analiza si el jugador lo saltó.
374                 * Para la puntuación.
375                 */
376                for (int i = 0; i < barriles.length; i++) {
377                    if (barriles[i] != null) {
378                        if (jugador.saltandoBarril(barriles[i])) {
379                            puntuador.saltarbarril();
380                        }
381
382                    }
383                }
384
385
386                /*
387                 * Donkey arrojando barriles
388                 *
389                 * Si donkey decide arrojar un barril en el tick actual, se crea
                    uno nuevo en la primera
390                 * posició no NULL del arreglo de barriles.
391                 */
392                if (donkeyKong.decidir(contador)) {
393                    int creados = 0;
394
395                    for (int i = 0; i < barriles.length && creados == 0; i++) {
396                        if (barriles[i] == null) {
397                            barriles[i] = new Barril(suelos[suelos.length +
                                donkeyKong.arribaOabajo()]);
398                            creados = 1;
399                        }
400                    }
401                }
402
403            }
404
405
406            /*
407             * Analisis que ocurren mientras el juego está ganado
408            */
409            else if (this.estadoDelJuego.equals("ganado")) {
410
411                donkeyKong.noMasViolencia();
412
413                jugador.cambiarImagen("mirando");
414                jugador.dibujar(entorno, 0);
415
416                entorno.dibujarRectangulo(400, 300, 200, 75, 0, Color.GREEN);
417                entorno.cambiarFont("terminal", 20, Color.WHITE);
418                entorno.escribirTexto("G A N A S T E", 335, 310);
419
420                /*
421                 * Analisis que ocurren mientras el juego está perdido
422                */
423            } else {
424                entorno.dibujarRectangulo(400, 300, 200, 75, 0, Color.GREEN);
425                entorno.cambiarFont("terminal", 20, Color.WHITE);
426                entorno.escribirTexto("G A M E   O V E R", 315, 310);
427
```

```
428                 jugador.cambiarY(3);
429                 jugador.cambiarImagen("saltando");
430                 jugador.dibujar(entorno, 90);
431 
432                 donkeyKong.noMasViolencia();
433             }
434 
435 
436 
437         }
438 
439         @SuppressWarnings("unused")
440         public static void main(String[] args) {
441             Juego juego = new Juego();
442         }
443     }
444 
```