UNIVERSIDAD NACIONAL DE GENERAL SARMIENTO

• Extra: puntaje
• Sonido: salto
↳ colisión

PROGRAMACIÓN I

Trabajo Práctico Final

Integrantes

• Revisar métodos que solo reciben los parámetros que necesitan

APELLIDO Y NOMBRE	LEGAJO	EMAIL
Pereira, Fabián	37.247.692/2015	fabianeze93@gmail.com
Sánchez, Matías Alejandro	38.391.082/2015	mattisanchez94@gmail.com
Tula, Ignacio Mariano	35.226.620/2014	itula@ungs.edu.ar

• Personaje: • Variable instancia estado, en realidad es estado serio del juego. Por qué esto en el personaje?

• Char último. Las teclas se presionan en el juego, el personaje no tiene que saber cuál fue la última que se presionó.

• No está bien la definición del constructor de la clase.

• Saltando y pisando no necesitan recibir el entorno, este solo es necesario a la hora de dibujar.

- Dibujar: Si el personaje no esta encima de Roca ni al final de la pantalla. Si el juego esta terminado el personaje no deberia moverse.
- El manejo de las teclas si esta que estan en el juego.
- SubirEscalera y BajarEscalera solo deberia encargarse de que el personaje se mueva hacia arriba o hacia abajo, no dibuja, de eso debe encargarse el metodo dibujar.

Donkey Las variables de instancia solo se declaran no deben ser creadas. Y deben ser privadas.

INTRODUCCIÓN:

Se desarrolló un videojuego inspirado en el antiguo juego "Donkey Kong". Donde el jugador controla un personaje que debe desplazarse por el nivel hasta llegar a la posición del antagonista del juego, sin que ninguna de las dificultades y obstáculos móviles toquen al personaje.

Para ello se intentó que el apartado gráfico sea lo más fiel posible al juego original. Se han utilizado las herramientas propuestas por los métodos de la clase entorno. Principalmente las que permiten cargar una imagen (que puede ser estática o un gif animado).

Para la representación gráfica de objetos más sencillos se han utilizado las herramientas gráficas de dibujos (principalmente rectángulos y triángulos) que combinados entre ellos en posiciones y colores estratégicos permitieron simular vigas y escaleras.

De manera general, los métodos más importantes calculan las trayectorias y las posiciones de los objetos no controlados por el usuario.

Y en el caso particular del personaje, analizan diferentes escenarios posibles a la hora de permitirle al usuario o no ejecutar la acción requerida (no es posible saltar si se está cayendo por una cornisa, o posterior a un salto).

A su vez se detecta en cada instante de tiempo las posiciones extremas de cada objeto para conocer si hay colisiones que hagan perder al usuario.

DIFICULTADES:

Entre las dificultades que se hallaron fueron las acciones que realizaban acciones variadas a lo largo de un intervalo de tiempo.

Un ejemplo de ellas fué el salto, que en determinado momento se desplaza hacia arriba, hasta cierto punto donde cae. Pero la caída no debería estar limitada por una cantidad de tiempo sino que debería producirse hasta que en algún momento encuentre suelo.

Otra dificultad relacionada con el tiempo fué la producción del sonido que representan los pasos del personaje. El sonido no puede ejecutarse en cada tick, sino se produce un exceso de sonidos indistinguibles.

En todos los casos, fue necesaria la aplicación de una variable numérica que guardara el tick en el cual se produjo la última acción y prohibir a los métodos su ejecución hasta que no se dieran ciertas condiciones, entre ellas que hubiera una distancia temporal razonable.

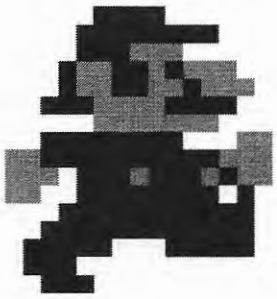
Otra complejidad añadida fueron las colisiones del personaje con los barriles, donde los puntos de contacto pueden ser múltiples y provenir de una combinación de diferentes lados de los objetos estudiados.

Aquí la dificultad era matemática y eminentemente referida al resultados que arrojaban las diferentes inecuaciones de las distancias entre puntos "x" e "y" de cada extremo de cada objeto, pero a nivel informático realizable con condicionales encadenados o con varias condiciones.

Similar al problema anterior, lo fué la detección de cercanías del personaje con escaleras o los saltos de barriles, pero la complejidad nuevamente viene dada por la combinación de al menos dos puntos "x" por cada objeto y dos puntos "y" por cada objeto para conocer la verdadera cercanía y no obtener un falso positivo.

Un falso positivo de un salto de barril sería que uno de estos se encontrara en la misma posición "x" que el personaje pero en un piso inferior. Evidentemente, aquí no hay salto alguno.

Clase: Personaje



Alto Original:	500px
Ancho Original:	500px
Escala utilizada:	0.090
Alto Utilizado:	40px
Ancho Utilizado:	40px
Distancia centro-suelo:	20px
Distancia centro-lateral:	15px

La clase Personaje permite generar un objeto que se encargará de todas las tareas relacionadas al protagonista del juego. Desde mostrar su apartado gráfico, desplazarse por el escenario según lo indique el usuario y las posibilidades en las que se encuentre en determinado momento y a lo largo del tiempo.

VARIABLES

`private String estado;`

Indica si el jugador está vivo o muerto. Permite o impide a las funciones que controlan los movimientos del personaje actuar en consecuencia.

`private int posx;`
`private int posy;`

La posición con respecto al eje "x" y al eje "y".

```
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;
```

Las variables que contienen las animaciones e imágenes en formato GIF o PNG para ilustrar las diferentes acciones del protagonista.

`private char ultima;`

Última tecla de sentido (DER o IZQ) presionada (Sirve para saber para dónde 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

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

private int sonando = 1;
```

Último 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)

MÉTODOS *

[1]

- **Constructor** (Requiere que se le pasé un parámetro de tipo Viga) X

  Para que necesito la Viga ???

  El constructor asigna:
  - El estado a "vivo".
  - La posición x en 50.
  - La posición y 35 píxeles más arriba que el suelo de la Viga del argumento.
  - Las rutas URL donde se encuentran los archivos GIFs y PNGs
  - A la última tecla presionada como la tecla derecha para que mire hacia el lado conveniente. X
  - Y los estados "está cayendo", "está saltando", "está en una escalera", y "está cerca de una escalera" como falsos.

- **hacerSonar** (Debe recibir el momento actual expresado como un número entero, donde cero es el inicio de ejecución) ✓

  Esta función ejecuta el sonido de caminar pero evita que suene en cada tick donde se está caminando. Sino habría una bola de sonido indistinguible.

  La función decide sonar alguna de las 3 variantes de sonidos de pasos que hay. Y sólo hace sonar cuando la distancia entre el sonido anterior y el actual es de 40 ticks.

- **saltar** (Debe recibir el entorno como parámetro y el momento actual expresado como un número entero, donde cero es el inicio de ejecución)

  La función saltar se encarga de la parte del salto que se ejecuta una sola vez. Es decir que **no** se encarga de la animación de subida o caída a lo largo de los ticks de un salto normal. ✓

  Se le debe indicar el entorno y el contador de ticks actual.

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

Cambia el dibujo de caminar por el salto, según hacia qué lado este mirando el personaje. Cambia el estado de **`estaSaltando`** a verdadero. Ejecuta el sonido del salto. Indica el tick en el cual se realizó el salto, guardando el valor en **`tiempoSalto`**.

- **saltando** (Debe recibir el entorno como parámetro; el momento actual expresado como un número entero, donde cero es el inicio de ejecución; y el arreglo con las vigas utilizadas en la instancia Juego)

Esta función se encarga de manipular, a lo largo del tiempo, lo que ocurre con el personaje cuando no está en el suelo (si debe ascender porque saltó, sí debe caer o ninguna de las anteriores si ya se encuentra en el suelo). Para que funcione correctamente se la llamar por cada tick.

Si el personaje está vivo y no se encuentra desplazándose en una escalera, se evalúa lo siguiente:

- Si el personaje está saltando, y dicho salto se produjo con menos de 30 ticks de diferencia con el momento actual, entonces hay que elevar 1px al jugador (restar 1 en eje 'y').
- En cualquier otro caso, se indica que el personaje ya no está saltando y se consulta al método **`pisando`**.
  Si el método **`pisando`** informa que no se está pisando ninguna viga (valor -1) el jugador debe caer (aumentar 1 en el eje "y") hasta que sus pies toquen suelo. Si por contrario, **`pisando`** informa que se está exáctamente sobre una viga (devuelve el índice de la viga pisada).

- **pisando**(Debe recibir el entorno como parámetro y el arreglo con las vigas utilizadas en la instancia Juego)

Esta función devuelve el índice que ocupa la viga en el arreglo de suelos que el jugador está pisando[2]. Si no se encuentra pisando, entonces devuelve -1.

Para saber si **no** está pisando la viga, el centro 'y' del personaje + 20 pixeles (para llegar al pie del personaje) **`pies()`** debe poseer un valor distinto para la coordenada 'y' donde comienza cada viga (la posy - 12px) **`(int)suelos[i].dondeEmpiezaElSuelo()`**.

En el caso de que el personaje se encuentra pisando la viga. Queda por conocer si se encuentra dentro de todos los puntos 'x' que conforman el largo de la viga. Porque de lo contrario, no se encontraría pisándola.

Por eso la función analiza que el extremo derecho de la viga, sea pisada por al menos el lateral izquierdo del personaje, y lo mismo de forma opuesta.

Si no se cumple esta condición, el personaje está cayendo por estar fuera de la viga a pesar de estar a la altura de alguna de ellas.

[2] Definamos pisando como ocupar el pixel igual o inmediatamente superior del último píxel superior ocupado por una viga. Dicho pixel ocupado debe ser el primero inferior del personaje.

- **pies**(No requiere parámetros adicionales)

  Devuelve la posición del extremo inferior del personaje.

- **cabeza** (No requiere parámetros adicionales)

  Devuelve la posición del extremo superior del personaje.

- **dibujar**(Debe recibir el entorno como parámetro; el momento actual expresado como un número entero, donde cero es el inicio de ejecución; y el arreglo con las escaleras utilizadas en la instancia Juego)

  Este método se encarga de ejecutar todas las animaciones y de desplazar (u ordenar a otra función desplazar) al personaje según se cumplan los criterios que lo permitan.

  En primer lugar la función analiza si el estado del personaje es "vivo". De lo contrario lo hace caer hasta quitarlo de pantalla. Esto es cuando el jugador pierde.
  Chequear si no termino el juego. El juego debería
  Para el estado "vivo" el método analiza diferentes condiciones para comprender cómo se debe actuar:

  - Si se presiona la tecla espacio (saltar), ordenará ejecutar la función **`saltar`**. Pero sólo si a su vez se cumple lo siguiente: La distancia temporal del momento actual con respecto al salto anterior (**`tiempoSalto`**) debe ser superior a 60 ticks; no se debe estar cayendo; no se debe estar desplazando dentro de una escalera.

  - Si se presiona la tecla arriba o abajo, ordenará desplazarse por una escalera en la dirección solicitada. Pero sólo si a su vez se cumple lo siguiente: Se debe estar cerca de una escalera.
    - Este apartado a su vez controla el ingresar a una escalera. Puesto que si la escalera comunica el piso actual con el inferior, la única forma de ingresar a la escalera, es hacia abajo. Si la escalera comunica el piso actual con el superior, es hacia arriba.
    - El desplazamiento por una escalera donde el personaje ya se encuentre.
    - La imágen estática del personaje si decide quedarse quieto en la escalera.

  - Si se presiona las teclas izquierda o derecha, mostrará la animación correspondiente y desplazará al jugador. Pero sólo si a su vez se cumple lo siguiente: No se debe estar cayendo, no se debe estar saltando, no se debe estar dentro de una escalera.

Porque el entorno? Si solo debe ver si estoy cerca de una escalera.

- **estoyCercaDeEscalera** (Debe recibir el entorno como parámetro; el arreglo con las vigas utilizadas en la instancia Juego; y el arreglo con las escaleras utilizadas en la instancia Juego)

  Esta función cambia el valor de **`estaCercaEscalera`** a true o false dependiendo si el personaje está cerca de una escalera como para poder subir o descender por ella.

  Esta función debe llamarse en cada tick del juego pero sólo si el personaje no se encuentra dentro de una escalera actualmente.

  Sólo analiza la proximidad de una escalera, si la función pisando devuelve el índice de la viga pisada. No se analiza proximidad para valores -1 (en el aire) ni si se está cayendo.

  Las escaleras se analizan en dos casos separados. Las que comienzan en el piso actual del personaje y ascienden al superior, y las que terminan en el piso actual porque descienden al inferior.

  Para estar cerca de una escalera los puntos "x" extremos de la escalera deben contener al punto central "x" del personaje. Y la ubicación del punto "y" ocupada por los pies del personaje debe estar a una distancia cercana al extremo correspondiente de la escalera.

  Este metodo solo comprueban X e Y

- **subirEscaleras**(Debe recibir el entorno como parámetro; y el arreglo con las escaleras utilizadas en la instancia Juego)

  Esta función ejecuta las animaciones correspondiente a subir escalera y se encarga de informar si ya terminó de subirla. Es decir que sale de la escalera y se encuentra en el piso superior.

  Según si el nuevo piso al que se ascendió, la escalera se encontrara a derecha o izquierda, voltea al personaje en la dirección correcta.

  Este metodo solo comprueban X e Y

- **bajarEscaleras** (Debe recibir el entorno como parámetro; y el arreglo con las escaleras utilizadas en la instancia Juego)

  Esta función ejecuta las animaciones correspondiente a bajar escalera y se encarga de informar si ya terminó de descender. Es decir que sale de la escalera y se encuentra en el piso inferior.

  Según si el piso al que descendió, la escalera se encontrara a derecha o izquierda, voltea al personaje en la dirección correcta.

- **lateralDerecho** (No requiere parámetros adicionales)

  Devuelve la posición del extremo derecho del personaje.

- **lateralIzquierdo** (No requiere parámetros adicionales)

  Devuelve la posición del extremo izquierdodel personaje.

- **estaEnEscalera**(No requiere parámetros adicionales)

  Devuelve verdadero o falso según si está en escalera o no.

- **morir**(No requiere parámetros adicionales)

  Cambia el estado a "muerto".

- **ganar** (Debe recibir el entorno como parámetro; y el arreglo con las vigas utilizadas en la instancia Juego)

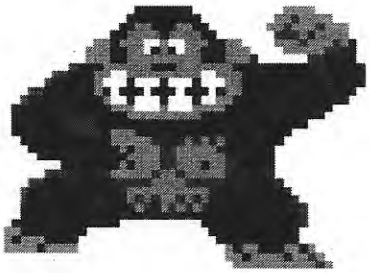
  Retorna verdadero sólo cuando el jugador se encuentra en una posición x igual o menor a 150 y a la vez en la última viga del arreglo (donde se encuentra donkey).

- **saltandoBarril** (Debe recibir el arreglo con los barriles utilizadas en la instancia Juego)

  Retorna verdadero cuando se realiza un salto exitoso sobre un barril. Se debe ejecutar en cada tick y se analiza cada barril.

  El salto es exitoso cuando:
  - La posición "x" del barril es igual a la posición "x" del personaje (con un ayuda de +/- 1 píxel a cada lado)
  - Los pies del personaje están por encima de la parte superior del barril
  - Pero no a tanta diferencia (tan alto no salta el personaje, sin esta condición los barriles en pisos inferiores serían considerados como saltados)
  - Que el barril no haya sido previamente saltado.
  - No estar dentro de una escalera.

Clase: Donkey



Alto Original:	**350px**
Ancho Original:	**500px**
Escala utilizada:	**0.19**
Alto Utilizado:	**66px**
Ancho Utilizado:	**95px**
Distancia centro-suelo:	**33px**

La clase Donkey permite generar un objeto que se encargará, de arrojar barriles según una elección aleatoria del tiempo. También, desde el apartado gráfico, genera una animación del antagonista del juego, que varía entre estar enojado de forma constante, y la de simular que arroja barriles en la creación de uno de estos.

VARIABLES

- `int ultimoLanzamiento;`
  Indica el momento (número de tick desde que comenzó la ejecución) en que se realizó el último lanzamiento de un barril.

- `Random rnd = new Random();`
  Objeto Random para ayudar a generar números pseudo aleatorios que permitan la decisión de en qué momento se lanzará un barril.

- `int lanzarRandom;`
  Indica el momento (número de tick desde que comenzó la ejecución) en que se realizará el próximo lanzamiento de un barril.

- `String violencia;`
  Sirve para indicar el estado de donkey con respecto a si debe arrojar barriles o no.

MÉTODOS *

[1]

- **Constructor** (No requiere parámetros adicionales)

  El constructor solamente asigna como último lanzamiento el momento cero. Y el estado de violencia como "violento".

- **Gorilear** (Debe recibir el entorno como parámetro y el momento actual expresado como un número entero, donde cero es el inicio de ejecución)

  Este método genera una constante animación. Es solamente un decorado.

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

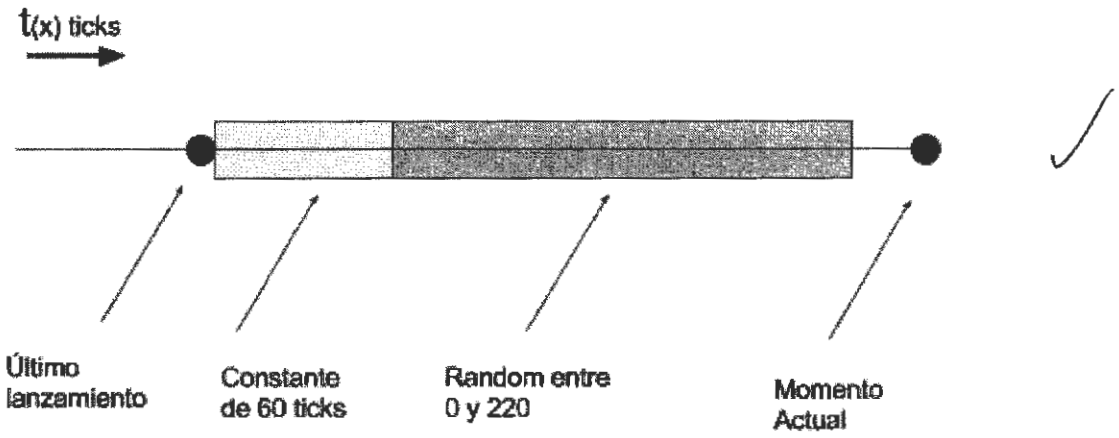
Si el último lanzamiento se realizó hace menos de 30 ticks debe mostrar la animación "tirar". De lo contrario debe mostrar la simple animación llamada "gorilear".

- **decidir**(Debe recibir el momento actual expresado como un número entero, donde cero es el inicio de ejecución)

  Es el método que se encarga de ejecutar un algoritmo que decide de forma aleatoria en cual tick del futuro (momento del juego) se lanzará el siguiente barril. También es el método que indica que debe ser lanzado el barril en ese preciso instante, si el momento actual es igual al tick que fue planeado su lanzamiento. Lo anterior ocurre si la `violencia` está seteada en "violento".

  Retorna `true` si el momento actual es igual a `lanzarRandom`. Retorna `false` para los demás casos.
  Una vez lanzado el barril, `lanzarRandom` pasa a tener valor 0 (no hay lanzamiento a posterior planificado).

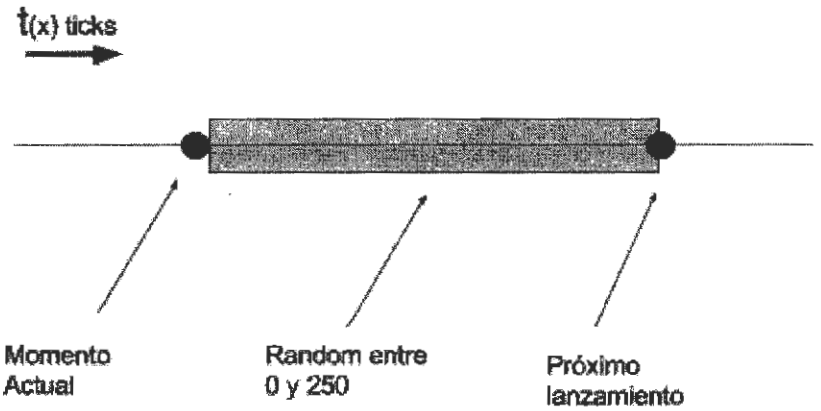
  Existe un algoritmo basado en dos randoms diferentes para decidir el próximo lanzamiento.



  Se decidirá una planificación de un lanzamiento futuro, si ocurre primero que el momento actual es mayor o igual al último lanzamiento más 60 ticks y un random de entre 0 y 220 ticks.

  Esto garantiza que no se dé un paso inicial hacia lanzar barriles cada menos de 60 ticks (aprox 2 segundos o menos).

  Cuando se cumple lo anterior. Se establece un random entre 0 y 250 hacia el futuro para planificar el próximo lanzamiento.



- **noMasViolencia**(No requiere parámetros adicionales)

  Este método cambia la variable **`violencia`** a "noviolento". Lo que impide al método "decidir" de arrojar barriles o planificar futuros lanzamientos. Esta función es llamada desde la clase principal cuando el juego termina, ya sea al perder o al ganar.

- **arribaOabajo**(No requiere parámetros adicionales)

  Este método genera un random entre 0 y 60. Si el número elegido aleatoriamente es múltiplo de 3, entonces se decide arrojar el barril directamente hacia la viga inferior, de lo contrario se arroja por la misma viga donde está donkey.

  Este método se diseñó para agregarle dificultad al juego.

  Se utiliza un random y que el resultado sea múltiplo de 3 para que en general se respete que haya un aproximado de 33% de posibilidades de arrojar el barril por debajo y un 66% por la viga normal.

  Se retorna -3 o -1 con la intención de que el resultado de esta función le reste dichas unidades al valor length del arreglo de Vigas[].

ANOTACIONES

Para la correcta aparición gráfica de Donkey se recomienda colocar ciertos valores de **`entorno.dibujarImagen`**

```
viga = Viga (pos = 6)
y = distancia ( viga.y - viga.alto / 2)
```

La distancia entre el "y" de la viga (con pos = 6) menos la mitad de su alto

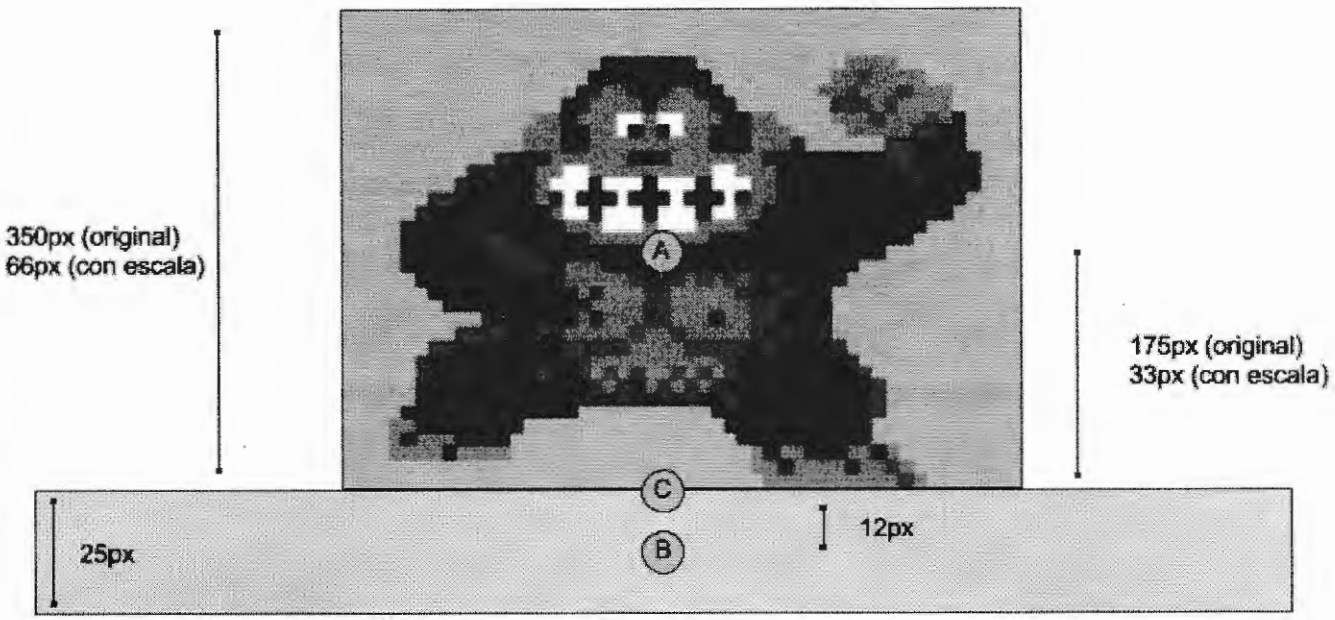
Para una viga en posición 6 con los siguientes valores:

```
x = 325;
y = 75;
largo = 700;
alto = 25;
```

Se recomienda ubicar a Donkey en:

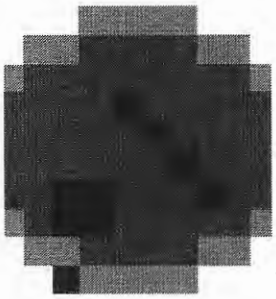
```
x = 50
y = 30
escala = 0.19
```

Esto produce que el último píxel inferior del Donkey pise (o límite inmediata y superiormente con) el último píxel superior de la viga).



A = Punto centro de la imagen (x,y) Donkey.
B = Punto centro del rectángulo que dibuja la Viga.
C = Límite entre la viga y la imagen.

Clase: Barril



Alto Original:	**108px**
Ancho Original:	**130px**
Escala utilizada:	**0.157**
Alto Utilizado:	**17px**
Ancho Utilizado:	**17px**
Distancia centro-suelo:	**10px**
Distancia centro-lateral:	**10px**

La clase Barril permite generar un objeto que se encargará de dibujar barriles que se desplazarán en el sentido correcto por las vigas del nivel, caerán cuando no haya suelo. Su función es la de ser un obstáculo móvil en el desarrollo del juego que el personaje no debe tocar.

VARIABLES

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

```
// Contiene las animaciones de rotar hacia derecha o izquierda para dar sensacion de movimiento.
private Image spin_izquierda;
private Image spin_derecha;
```

```
// Indica hacia donde se movía por última vez el barril "izquierda" o "derecha"
private String ultima;
```

```
// Indica si ya fue saltado alguna vez por el jugador
private boolean saltado;
```

para que ???

MÉTODOS *

- **Constructor** (Requiere que se le pasé una variable de tipo Viga)

  Asigna el diámetro fijado y calcula la escala. Iguales para todos los barriles

  Calcula las posiciones "x" e "y" según en qué viga es lanzado el barril.

  Se asignan las rutas URL de los gifs que proveen la animación de rodamiento.

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

- **deboDestruirme** (Debe recibir el entorno como parámetro; y el arreglo con las vigas utilizadas en la instancia Juego)

  Esta función debe ser llamada en cada tick por cada barril que exista.

  Analiza si la posición del barril es en la planta baja y en el extremo izquierdo. Entonces retorna verdadero para indicar que este barril debe ser destruido para dejar paso a que donkey pueda crear otro.

- **dibujar** (Debe recibir el entorno como parámetro; el momento actual expresado como un número entero, donde cero es el inicio de ejecución; y el arreglo con las vigas utilizadas en la instancia Juego)

  Esta función debe ser llamada en cada tick por cada barril que exista. Por cada tick se dibuja el barril.

  Esta función se encarga de mostrar el apartado gráfico del barril y calcular su trayectoria y movimiento.

  Calcula su desplazamiento en el eje "x" y analiza si dicho desplazamiento debe cambiar de sentido. También se encarga de las caidas en el eje "y" cuando no existe suelo sobre el cual rodar.

- **pisando** (Debe recibir el entorno como parámetro y el arreglo con las vigas utilizadas en la instancia Juego)

  Esta función devuelve el índice que ocupa la viga en el arreglo de suelos que el barril está pisando[2]. Si no se encuentra pisando, entonces devuelve -1.

- **pies**(No requiere parámetros adicionales)

  Devuelve la posición del extremo inferior del barril.

- **superior**(No requiere parámetros adicionales)

  Devuelve la posición del extremo superior del barril.

- **lateralDerecho** (No requiere parámetros adicionales)

  Devuelve la posición del extremo derecho del personaje.

[2] Definamos pisando como ocupar el pixel igual o inmediatamente superior del último píxel superior ocupado por una viga. Dicho pixel ocupado debe ser el primero inferior del personaje.

- **lateralIzquierdo** (No requiere parámetros adicionales)

  Devuelve la posición del extremo izquierdo del barril.

- **centroX**(No requiere parámetros adicionales)

  Devuelve la posición del centro "x" del barril.

- **saltado**(No requiere parámetros adicionales)

  Cambia el valor de `saltado` a verdadero.

- **fueSaltado**(No requiere parámetros adicionales)

  Devuelve el valor de verdad sobre el atributo `saltado` que indica si fue saltado el barril.

Clase: Viga

La clase Viga permite generar un objeto que se encargará de dibujar piso y techo del nivel del juego. Las vigas en este software son implementadas como rectángulos sólidos que tienen dentro de ellos parejas triángulos (uno normal y otro invertido) con el mismo color de fondo para asemejar una estructura metálica.

VARIABLES

```
private int pos;
```

Un indicador propio de posición (no el índice en el arreglo de Vigas). La posición 1 es la planta baja, la 2 el primer piso y así sucesivamente.

```
private double x;
private double y;
private double largo;
private double alto;
```

MÉTODOS *

[1]

- **Constructor** (Requiere que se le pasé un parámetro de tipo numérico de posición)

  El constructor asigna valores a "x", "y" , "largo" y "alto" según la posición solicitada para construir gráficamente los pisos del nivel.

  Este equipo de trabajo decidió que las vigas tuvieran un grosor (o alto) de 25px, un largo de 700px (excepto para la planta baja que ocupa todo el ancho).

- **dibujar** (Debe recibir el entorno como parámetro)

  Esta función debe ser llamada en cada tick por cada viga que exista. Por cada tick dibuja la viga.

  La construcción gráfica de la viga tiene un condimento especial. Para dar la sensación de que es una estructura metálica, se dibuja un rectángulo de color rojo de fondo, y sobre él de forma

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

estratégica, triángulos del mismo color que el fondo, en juegos de a dos. Cada uno invertido 90° con respecto al anterior.

- **getPosx** (No requiere parámetros adicionales)

  Devuelve la posición del centro x de la viga.

- **getPosy** (No requiere parámetros adicionales)

  Devuelve la posición del centro y de la viga.

- **getAncho** (No requiere parámetros adicionales)

  Devuelve el largo o ancho de la viga. Longitud en el eje "x".

- **getPos** (No requiere parámetros adicionales)

  Devuelve la pos que se indicó al momento de su creación "x".

- **dondeEmpiezaElSuelo** (No requiere parámetros adicionales)

  Devuelve la posición "y" del extremo inferior de la viga.

- **dondeTerminaElTecho** (No requiere parámetros adicionales)

  Devuelve la posición "y" del extremo superior de la viga.

- **extremoIzquierdo**(No requiere parámetros adicionales)

  Devuelve la posición "x" del extremo izquierdo de la viga.

- **extremoDerecho**(No requiere parámetros adicionales)

  Devuelve la posición "x" del extremo derecho de la viga.

Clase: Escalera

La clase Escalera permite generar un objeto que en el apartado gráfico simula una escalera y al existir una cercanía adecuada con el personaje, le permitirá a este último desplazarse por el eje "y" del juego.

VARIABLES

```
private int pos;
```

Un indicador propio de posición (no el índice en el arreglo de Escaleras). La posición 1 es la escalera de planta baja hacia primer piso, la 2 del primer piso con el segundo.

```
private double x;
private double y;
private double largo;
private double alto;
```

MÉTODOS *

[1]

- **Constructor** (Requiere que se le pasé un parámetro de tipo numérico de posición y el arreglo con las vigas utilizadas en la instancia Juego)

  El constructor asigna valores a "x", "y" , "largo" y "alto" según la posición solicitada para construir gráficamente las escaleras que conectar un piso con otro.

  Se determinado un ancho fijo de 30px para cada escalera. Un alto lo suficiente para que cubra la distancia entre el suelo inferior y el suelo de la viga superior.

  El punto "y" se sitúa en la mediatriz de dicha distancia.

  El punto "x" se sitúa en un random que varía 50px en el extremo de la viga superior y con un margen de seguridad de otros 30 píxeles. Esto proporciona una mínima diferencia en la posición horizontal de las escaleras con respecto a cada ejecución del juego.

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

- **dibujar** (Debe recibir el entorno como parámetro)

  Este método dibuja la escalera. Debe ser llamado en cada tick de la ejecución del juego.

  Dibuja el rectángulo con los atributos que posee la instancia creada, y a su vez realiza un cálculo para dibujar pequeños rectángulos inscritos dentro del principal, del mismo color que el fondo de la ventana, con la finalidad de simular una escalera.

- **extremoInferior**(No requiere parámetros adicionales)

  Devuelve la posición "y" del extremo inferior de la escalera.

- **extremoSuperior**(No requiere parámetros adicionales)

  Devuelve la posición "y" del extremo superior de la escalera.

- **lateralIzquierdo**(No requiere parámetros adicionales)

  Devuelve la posición "x" del extremo izquierdo de la escalera.

- **lateralDerecho**(No requiere parámetros adicionales)

  Devuelve la posición "x" del extremo derecho de la escalera.

Clase: Mensajes

En el juego.

La clase Mensajes permite generar un objeto que mostrará información en pantalla de porqué finalizó el juego.

VARIABLES

```
String mensajePerdedor = "G A M E   O V E R";
String mensajeGanador = "G A N A S T E";
```

MÉTODOS *[1]

- **dibujar** (Debe recibir el entorno como parámetro y una cadena de caracteres)

  Muestra el mensaje ganador si el parámetro String es igual a "ganar" o muestra el mensaje perdedor si el String recibido es "perder".

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.

Clase: Puntaje

La clase Puntaje permite generar un objeto que contará puntos por cada barril saltado y al llegar a cumplir el objetivo. También se encarga de mostrar dicha información en pantalla.

VARIABLES

```
private int puntos;
```

MÉTODOS *[1]

- **Constructor** (No requiere parámetros adicionales)

  Asigna el puntaje en cero.

- **saltarbarril** (No requiere parámetros adicionales)

  Suma quince puntos al puntaje actual

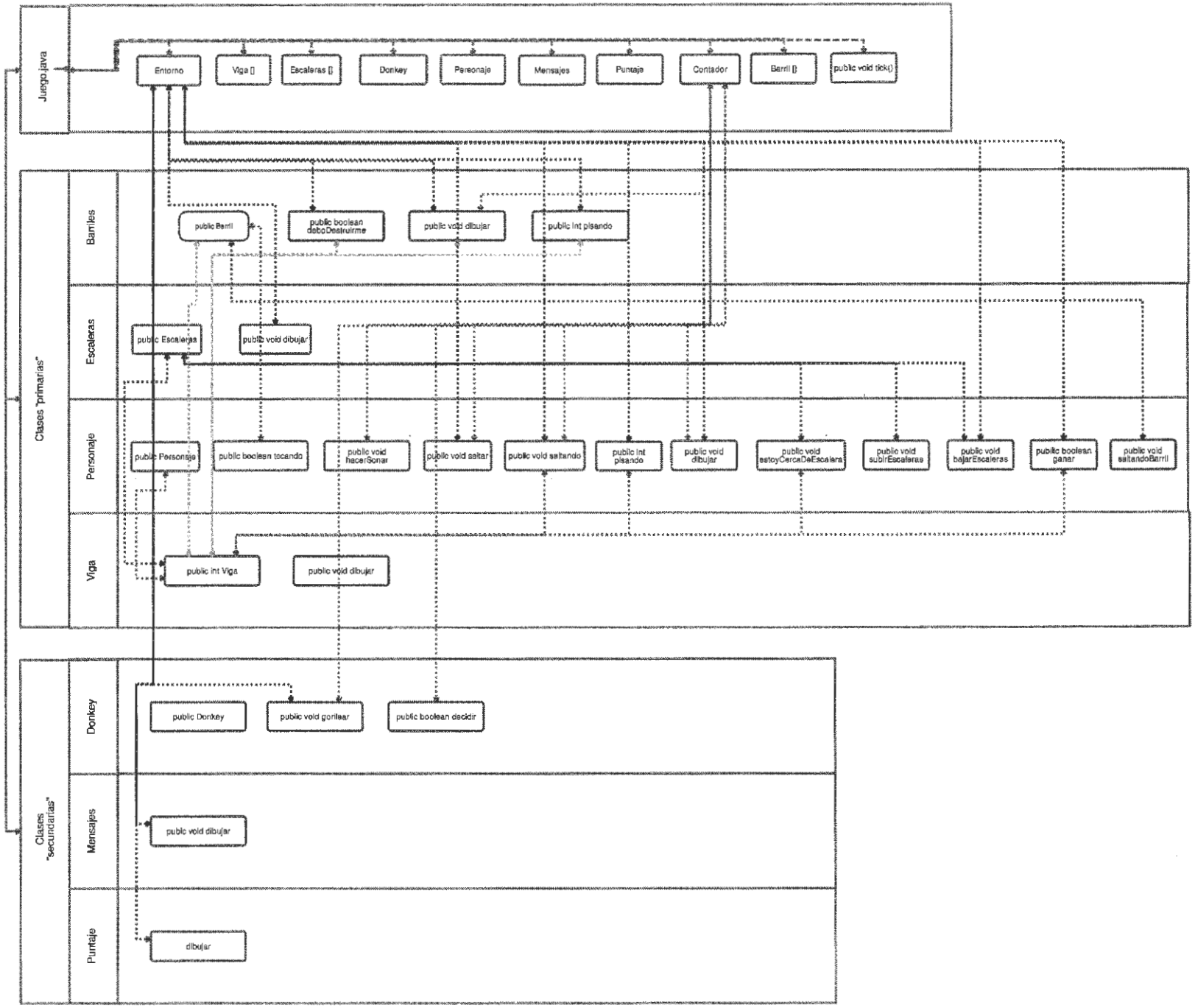
- **ganar** (No requiere parámetros adicionales)

  Suma cien puntos al puntaje actual

- **dibujar** (Debe recibir el entorno como parámetro)

  Esta función muestra en pantalla la cantidad de puntos en el extremo superior derecho de la pantalla.

[1] (*) A excepción que se indique lo contrario, los métodos reciben una variable de tipo **Entorno** y que debe ser la instancia creada por la clase **Juego**.


Juego.java
Entorno
Viga []
Escaleras []
Donkey
Personaje
Mensajes
Puntaje
Contador
Barril []
public void tick()
Clases "primarias"
Barriles
public Barril
public boolean deboDestruirme
public void dibujar
public int pisando
Escaleras
public Escaleras
public void dibujar
Personaje
public Personaje
public boolean tocando
public void hacerSonar
public void saltar
public void saltando
public int pisando
public void dibujar
public void estoyCercaDeEscaleras
public void subirEscaleras
public void bajarEscaleras
public boolean ganar
public void saltandoBarril
Viga
public int Viga
public void dibujar
Clases "secundarias"
Donkey
public Donkey
public void gorilear
public boolean decidir
Mensajes
public void dibujar
Puntaje
dibujar

Juego.java

```
1 package juego;
2
3 import entorno.Entorno;
6
7 public class Juego extends InterfaceJuego {
8     // El objeto Entorno que controla el tiempo y otros
9     private Entorno entorno;
10    boolean juegoPerdido = false;
11    boolean juegoGanado = false;
12
13    // Variables y métodos propios de cada grupo
14
15    // Creación del arreglo de vigas
16    static Viga suelos[] = new Viga[] {
17
18            new Viga(1), new Viga(2), new Viga(3), new Viga(4), new Viga(5), new Viga(6
19
20    };
21
22    static Escaleras escaleras[] = new Escaleras[] { new Escaleras(0, suelos), new
  Escaleras(1, suelos),
23            new Escaleras(2, suelos), new Escaleras(3, suelos), new Escaleras(4, suelos
24
25    Donkey donkeyKong = new Donkey();
26    Personaje jugador = new Personaje(suelos[0]);
27    Mensajes terminal = new Mensajes();
28    Puntaje puntuador = new Puntaje();
29
30    int contador = 0;
31
32    Barril barriles[] = new Barril[
33
34        { new Barril(suelos[suelos.length - 3]), null, null, null, null, null, null, null,
  null, null, null, null, null,
35            null, null, null, null, null, null, null
36
37    };
38
39    // ...
40
41    Juego() {
42        // Inicializa el objeto entorno
43        this.entorno = new Entorno(this, "Donkey - Grupo Pereira - Sanchez - Tula - V1",
  800, 600);
44
45        // Inicializar lo que haga falta para el juego
46        // ...
47
48        // Inicia el juego!
49        this.entorno.iniciar();
50
51    }
52
53    /**
54     * Durante el juego, el método tick() será ejecutado en cada instante y por lo
55     * tanto es el método más importante de esta clase. Aquí se debe actualizar el
56     * estado interno del juego para simular el paso del tiempo (ver el enunciado
57     * del TP para mayor detalle).
58     */
59    public void tick() {
60        // Procesamiento de un instante de tiempo
61        // ...
```

```
62
63          // Ejecuta la función dibujar por cada miembro del arreglo de vigas.
64          for (int i = 0; i < suelos.length; i++) {
65              suelos[i].dibujar(entorno);
66          }
67
68          // Ejecuta la función dibujar por cada miembro del arreglo de escaleras.
69          for (int i = 0; i < escaleras.length; i++) {
70              escaleras[i].dibujar(entorno);
71          }
72
73          // Ejecuta la función dibujar para donkey
74          donkeyKong.gorilear(entorno, contador);
75
76          // Ejecuta la función dibujar para el jugador
77          jugador.dibujar(entorno, contador, escaleras);
78
79          puntuador.dibujar(entorno);
80
81          // Ejecuta la función que analiza si el jugador está EN una escalera.
82          if (jugador.estaEnEscalera() == false) {
83              jugador.estoyCercaDeEscalera(entorno, escaleras, suelos);
84          }
85
86          // Ejecuta la función que analiza si el jugador está EN el medio de un salto o
87          // caida.
88          jugador.saltando(entorno, contador, suelos);
89
90          // Contador de tiempo, medido en ticks
91          contador = contador + 1;
92
93          // Si donkey decidió arrojar, se crea un nuevo barril en la primera posición
94          // NULL del arreglo de barriles.
95          if (donkeyKong.decidir(contador)) {
96              int creados = 0;
97
98              for (int i = 0; i < barriles.length && creados == 0; i++) {
99                  if (barriles[i] == null) {
100                     barriles[i] = new Barril(suelos[suelos.length +
    donkeyKong.arribaOabajo()]);
101                     creados = 1;
102                 }
103             }
104         }
105
106         // Ejecuta la función dibujar por cada elemento no NULL del arreglo de barriles
107         // , también analiza si un barril debe destruirse.
108
109         for (int i = 0; i < barriles.length; i++) {
110             if (barriles[i] != null) {
111
112                 barriles[i].dibujar(entorno, contador, suelos);
113                 if (barriles[i].deboDestruirme(entorno, suelos)) {
114                     barriles[i] = null;
115                 }
116
117             }
118         }
119
120         // Es la función que indica si el jugador está tocando algún barril y por lo
121         // tanto game over si es verdadero.
122
```

```
123         if (jugador.tocando(barriles) && juegoPerdido == false) {
124             juegoPerdido = true;
125         }
126 
127         for (int i = 0; i < barriles.length; i++) {
128             if (barriles[i] != null) {
129 
130                 if (jugador.saltandoBarril(barriles[i])) {
131                     puntuador.saltarbarril();
132                 }
133 
134             }
135         }
136 
137         if (juegoPerdido) {
138             terminal.dibujar("perder", entorno);
139             jugador.morir();
140             donkeyKong.noMasViolencia();
141         }
142 
143         if (jugador.ganar(entorno, suelos) && juegoGanado == false) {
144             juegoGanado = true;
145             puntuador.ganar();
146         }
147 
148         if (juegoGanado) {
149             terminal.dibujar("ganar", entorno);
150             donkeyKong.noMasViolencia();
151         }
152 
153     }
154 
155                     ("unused")
156     public static void main(String[] args) {
157         Juego juego = new Juego();
158     }
159 }
160 
```

Personaje.java

```
1 package juego;
2
3 import juego.Viga;
8
9 public class Personaje {
10
11     private String estado;
12     private int posx;
13     private int posy;
14
15     private Image mirandoIzquierda;
16     private Image mirandoDerecha;
17
18     private Image caminandoIzquierda;
19     private Image caminandoDerecha;
20
21     private Image saltandoIzquierda;
22     private Image saltandoDerecha;
23
24     private Image subiendo;
25
26     private Image subiendo_quieto;
27
28     private char ultima; // ultima tecla de sentido (DER o IZQ) presionada (Sirve para
   saber para donde
29                          // debe mirar el personaje).
30
31     private int tiempoSalto; // tick en el cual se ejecutó el último salto (o el actual)
32
33     private boolean estaSaltando; // Indica si está saltando (ascendiendo) o no.
34
35     private boolean estaCayendo; // Indica si está cayendo (es decir que sus pies no está
   no están tocando viga
36                                  // alguna.
37
38     private boolean estaCercaEscalera;
39     private boolean estaEnEscalera;
40     private int enEscalera;
41
42     private int sonando = 1; // Ultimo archivo de sonido que se usó para caminar, hay 3
   variantes.
43     private int sonandoDesde = 0; // tick en el cual se ejecutó el último sonido de camin
   (ayuda a evitar que
44                                  // suenen sonidos en cada tick)
45
46     public Personaje(Viga vigasuelo) {
47
48         this.estado = "vivo";
49         this.posx = 50;
50         this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 35; // 35 pixeles por encima
   la viga inicial, genera una
51                                                                 // linda caida en el spaw
52
53         this.mirandoIzquierda =
   Herramientas.cargarImagen("rsc/graficos/marito/mira-izquierda.png");
54         this.mirandoDerecha =
   Herramientas.cargarImagen("rsc/graficos/marito/mira-derecha.png");
55
56         this.caminandoIzquierda =
   Herramientas.cargarImagen("rsc/graficos/marito/camina-izquierda.gif");
57         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
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 saltand
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[] suelo
    {
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 próxi
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
    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 
```

Donkey.java

```
1 package juego;
2
3 import java.awt.Color;
12
13 public class Donkey {
14
15     int ultimoLanzamiento;
16     Random rnd = new Random();
17     int lanzarRandom;
18     String violencia;
19
20     public Donkey() {
21
22         ultimoLanzamiento = 0;
23         violencia = "violento";
24
25     }
26
27     public void gorilear(Entorno entorno, int contador) {
28
29         if (contador - ultimoLanzamiento < 30) {
30
31             Image gorila = Herramientas.cargarImagen("rsc/graficos/donkey/tirar.gif");
32             Image stock = Herramientas.cargarImagen("rsc/graficos/barriles/stock.png");
33             entorno.dibujarImagen(gorila, 100, 30, 0, 0.19);
34             entorno.dibujarImagen(stock, 30, 32, 0, 0.13);
35
36         } else {
37             Image gorila = Herramientas.cargarImagen("rsc/graficos/donkey/gorilear.gif");
38             Image stock = Herramientas.cargarImagen("rsc/graficos/barriles/stock.png");
39             entorno.dibujarImagen(gorila, 100, 30, 0, 0.19);
40             entorno.dibujarImagen(stock, 30, 32, 0, 0.13);
41         }
42     }
43
44     public boolean decidir(int contador) {
45
46         if (this.violencia.equals("violento")) {
47             if (this.lanzarRandom == contador) {
48                 this.ultimoLanzamiento = contador;
49                 this.lanzarRandom = 0;
50
51                 return true;
52             }
53
54             if (contador >= this.ultimoLanzamiento + this.rnd.nextInt(220) + 60 &&
  this.lanzarRandom == 0) {
55
56                 lanzarRandom = this.rnd.nextInt(250) + contador;
57
58                 return false;
59             }
60
61             return false;
62
63         }
64
65         return false;
66     }
67
68     public void noMasViolencia() {
69         this.violencia = "noviolento";
```

```
70      }
71
72      public int arribaOabajo() {
73
74          int eleccion = this.rnd.nextInt(60);
75          if (eleccion % 3 == 0) {
76              return -3;
77          } else {
78              return -1;
79          }
80      }
81
82  }
83
```

Barril.java

```java
package juego;

import java.awt.Color;

public class Barril {

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

    private Image spin_izquierda;
    private Image spin_derecha;
    private String ultima;
    private boolean saltado;

    public Barril(Viga vigasuelo) {

        // this.estado = "vivo";
        this.diametro = 17;
        this.escala = (double) this.diametro / 108;

        // this.posy = 48;

        if (vigasuelo.getPos() == 6) {
            this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 20;
            this.ultima = "derecha";
            this.posx = 120;
        } else if (vigasuelo.getPos() == 4) {
            this.posy = (int) vigasuelo.dondeEmpiezaElSuelo() - 190;
            this.ultima = "izquierda";
            this.posx = 120;
        }

        this.spin_izquierda =
  Herramientas.cargarImagen("rsc/graficos/barriles/spin-izquierda.gif");
        this.spin_derecha =
  Herramientas.cargarImagen("rsc/graficos/barriles/spin-derecha.gif");

        this.saltado = false;

    }

    public boolean deboDestruirme(Entorno entorno, Viga[] suelos) {
        if (this.posx < 15 && this.pisando(entorno, suelos) == 0) {
            return true;
        } else {
            return false;
        }
    }

    public void dibujar(Entorno entorno, int contador, Viga[] suelos) {

        // Image barril =
        // Herramientas.cargarImagen("rsc/graficos/barriles/cayendo.png");
        // entorno.dibujarCirculo(posx, posy, diametro, Color.blue);

        // Si está rodando sobre el suelo
        if (pisando(entorno, suelos) != -1) {

            // En vigas con indice par desplazar a izquierda
            if (this.posx >= 10 && pisando(entorno, suelos) % 2 == 0) {
```

```
66                  this.posx = this.posx - 1.7;
67                  entorno.dibujarImagen(spin_izquierda, this.posx, this.posy, 0,
  this.escala);
68                  this.ultima = "izquierda";
69              }
70
71              // En vigas con indice impar desplazar a derecha
72              else if (this.posx <= 800 && pisando(entorno, suelos) % 2 == 1) {
73                  this.posx = this.posx + 1.7;
74                  entorno.dibujarImagen(spin_derecha, this.posx, this.posy, 0, this.escala
75                  this.ultima = "derecha";
76              }
77
78          }
79
80          // Si NO está rodando sobre el suelo
81          if (pisando(entorno, suelos) == -1) {
82
83              // cambia la posición con respecto al eje "y" hacia abajo
84              this.posy += 1;
85
86              // Si venia desplazandose a derecha pero está cayendo y hay espacio en el x,
87              // se sigue desplazando a derecha
88              if (this.posx <= 800 && this.ultima.equals("derecha")) {
89                  this.posx = this.posx + 1.7;
90                  entorno.dibujarImagen(spin_derecha, this.posx, this.posy, 0, this.escala
91              }
92
93              // De lo contrario hay que indicarle que en el próximo tick se desplace a
94              // izquierda.
95              else {
96                  this.ultima = "izquierda";
97              }
98
99              // Si venia desplazandose a izquierda pero está cayendo y hay espacio en el x
100             // se sigue desplazando a izquierda
101             if (this.posx >= 10 && this.ultima.equals("izquierda")) {
102                 this.posx = this.posx - 1.7;
103                 entorno.dibujarImagen(spin_izquierda, this.posx, this.posy, 0,
  this.escala);
104             } else {
105                 // De lo contrario hay que indicarle que en el próximo tick se desplace a
106                 // derecha.
107                 this.ultima = "derecha";
108             }
109
110         }
111
112     }
113
114     // Igual que pisando de Personaje
115     public int pisando(Entorno entorno, Viga[] suelos) {
116
117         for (int i = 0; i < suelos.length; i++) {
118
119             if (this.pies() == (int) suelos[i].dondeEmpiezaElSuelo()) {
120
121                 if (this.lateralDerecho() < suelos[i].extremoIzquierdo()
122                         || this.lateralIzquierdo() > suelos[i].extremoDerecho()) {
123
124                     return -1;
125
```

```
126                 } else {
127
128                     return i;
129
130                 }
131             }
132
133         }
134
135         return -1;
136
137     }
138
139     public int pies() {
140         return (int) this.posy + diametro / 2 - 2;
141     }
142
143     public int superior() {
144         return (int) this.posy - diametro / 2 + 2;
145     }
146
147     public int lateralDerecho() {
148         return (int) this.posx + diametro / 2;
149     }
150
151     public int lateralIzquierdo() {
152         return (int) this.posx - diametro / 2;
153     }
154
155     public int centroX() {
156         return (int) this.posx;
157     }
158
159     public void saltado() {
160         this.saltado = true;
161     }
162
163     public boolean fueSaltado() {
164         return this.saltado;
165     }
166
167 }
```

Viga.java

```
1 package juego;
2
3 import java.awt.Color;
8
9 public class Viga {
10
11     private int pos;
12     private double x;
13     private double y;
14     private double largo;
15     private double alto;
16
17     /*
18      * Este constructor, ya tiene definida de forma estricta y estática las
19      * posiciones de las vigas
20      */
21
22     public Viga(int pos) {
23
24         switch (pos) {
25         case 1:
26             this.x = 400;
27             this.y = 575;
28             this.largo = 820;
29             this.alto = 25;
30             break;
31         case 2:
32             this.x = 325;
33             this.y = 475;
34             this.largo = 700;
35             this.alto = 25;
36             break;
37         case 3:
38             this.x = 475;
39             this.y = 375;
40             this.largo = 700;
41             this.alto = 25;
42             break;
43         case 4:
44             this.x = 325;
45             this.y = 275;
46             this.largo = 700;
47             this.alto = 25;
48             break;
49         case 5:
50             this.x = 475;
51             this.y = 175;
52             this.largo = 700;
53             this.alto = 25;
54             break;
55         case 6:
56             this.x = 325;
57             this.y = 75;
58             this.largo = 700;
59             this.alto = 25;
60             break;
61         }
62
63         this.pos = pos;
64
65     }
66
```

```
67      /*
68       * Dibujar
69       *
70       * Esta función debe ser llamada en cada tick por cada viga que exista. Por cada
71       * tick dibuja la viga.
72       *
73       * La construcción gráfica de la viga tiene un condimento especial. Para dar la
74       * sensación de que es una estructura metálica, se dibuja un rectángulo de color
75       * rojo de fondo, y sobre el de forma estratégica, triángulos del mismo color
76       * que el fondo, en juegos de a dos. Cada uno invertido 90º con respecto al
77       * anterior.
78       *
79       */
80      public void dibujar(Entorno entorno) {
81
82          // Rectángulo básico de la viga, respetando los valores indicados por el
83          // constructor
84          entorno.dibujarRectangulo(this.x, this.y, this.largo, this.alto, 0.0, Color.GRAY);
85
86          // El extremo izquierdo de la viga corrido 10px
87          double paso = this.x - (this.largo / 2) + 10;
88
89          // Se decide que la suma de la base de un triangulo, la punta del triángulo
90          // adyacente y un espacio
91          // extra sea la 25ava parte del ancho de la viga - 4 pixeles
92          double triangulos = (this.largo / 25) - 4;
93
94          // Indica la cantidad de parejas de triangulos dibujados. Una pareja es un
95          // triangulo con la punta hacia arriba
96          // y el otro con la punta hacia abajo.
97          int dibujados = 0;
98
99          // Este bucle dibuja la pareja de triángulos a lo largo de la viga.
100         while (dibujados <= triangulos) {
101
102             entorno.dibujarTriangulo(paso, this.y, 21, 21, Herramientas.radianes(90),
    java.awt.Color.BLACK);
103             paso += 14;
104             entorno.dibujarTriangulo(paso, this.y, 21, 21, Herramientas.radianes(270),
    java.awt.Color.BLACK);
105             paso += 14;
106             dibujados += 1;
107
108         }
109
110     }
111
112     // Devuelve la posX
113     public int getPosx() {
114         return (int) this.x;
115     }
116
117     // Devuelve la posY
118     public int getPosy() {
119         return (int) this.y;
120     }
121
122     // Devuelve el Ancho
123     public int getAncho() {
124         return (int) this.largo;
125     }
126
```

```
127     public int getPos() {
128         return pos;
129     }
130 
131     /*
132      * DondeEmpiezaElSuelo
133      *
134      * Esta función devuelve el valor en el 'y' en el cual comienza la viga.
135      * Sabiendo que el 'y' se encuentra en el centro.
136      *
137      */
138     public double dondeEmpiezaElSuelo() {
139 
140         return this.y - (this.alto / 2) - 1;
141     }
142 
143     public double dondeTerminaElTecho() {
144 
145         return this.y + (this.alto / 2) + 1;
146     }
147 
148     /*
149      * Esta función indica donde comienza la viga en el eje X.
150      */
151     public int extremoIzquierdo() {
152         return this.getPosx() - this.getAncho() / 2;
153     }
154 
155     /*
156      * Esta función indica donde termina la viga en el eje X.
157      */
158     public int extremoDerecho() {
159         return this.getPosx() + this.getAncho() / 2;
160     }
161 
162 }
163 
```

Escaleras.java

```
1 package juego;
2
3 import java.awt.Color;
8
9 public class Escaleras {
10
11     int pos;
12     double x;
13     double y;
14     double ancho;
15     double alto;
16
17     public Escaleras(int pos, Viga[] suelos) {
18
19         Random rnd = new Random();
20         int offsetEscalera = rnd.nextInt(50);
21
22         if (pos % 2 == 0) {
23             this.x = suelos[pos + 1].extremoDerecho() - 30 - offsetEscalera;
24         } else {
25             this.x = suelos[pos + 1].extremoIzquierdo() + 30 + offsetEscalera;
26         }
27
28         this.y = ((suelos[pos].dondeEmpiezaElSuelo() - suelos[pos +
  1].dondeEmpiezaElSuelo()) / 2)
29                 + suelos[pos + 1].dondeEmpiezaElSuelo();
30         this.ancho = 30;
31         this.alto = suelos[pos].dondeEmpiezaElSuelo() - suelos[pos +
  1].dondeEmpiezaElSuelo();
32
33     }
34
35     public void dibujar(Entorno entorno) {
36
37         // Rectángulo básico de la viga, respetando los valores indicados por el
38         // constructor
39         entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0.0, Color.RED);
40
41         double paso = this.y + (this.alto / 2) - 3;
42
43         // Se decide que la suma de la base de un triangulo, la punta del triángulo
44         // adyacente y un espacio
45         // extra sea la 25ava parte del ancho de la viga - 4 pixeles
46         double rectangulos = (this.alto / 10);
47
48         // Indica la cantidad de parejas de triangulos dibujados. Una pareja es un
49         // triangulo con la punta hacia arriba
50         // y el otro con la punta hacia abajo.
51         int dibujados = 0;
52
53         // Este bucle dibuja la pareja de triángulos a lo largo de la viga.
54
55         while (dibujados <= rectangulos) {
56
57             entorno.dibujarRectangulo(this.x, paso, 28, 9, 0.0, java.awt.Color.BLACK);
58             paso -= 10;
59
60             dibujados += 1;
61
62         }
63
64     }
```

```
65
66      public int lateralDerecho() {
67          return (int) this.x + 15;
68      }
69
70      public int lateralIzquierdo() {
71          return (int) this.x - 15;
72      }
73
74      public int extremoSuperior() {
75          return (int) (this.y - (this.alto / 2));
76      }
77
78      public int extremoInferior() {
79          return (int) (this.y + (this.alto / 2));
80      }
81
82  }
83
```

Mensajes.java

```
1 package juego;
2
3 import java.awt.Color;
7
8 public class Mensajes {
9
10     String mensajePerdedor = "G A M E   O V E R";
11     String mensajeGanador = "G A N A S T E";
12
13     public void dibujar(String tipo, Entorno entorno) {
14
15         if (tipo.equals("ganar")) {
16
17             entorno.dibujarRectangulo(400, 300, 200, 75, 0, Color.GREEN);
18             entorno.cambiarFont("terminal", 20, Color.BLACK);
19             entorno.escribirTexto(mensajeGanador, 335, 310);
20         }
21
22         else if (tipo.equals("perder")) {
23
24             entorno.dibujarRectangulo(400, 300, 200, 75, 0, Color.GREEN);
25             entorno.cambiarFont("terminal", 20, Color.WHITE);
26             entorno.escribirTexto(mensajePerdedor, 315, 310);
27         }
28
29     }
30
31 }
32
```

Puntaje.java

```java
package juego;

import entorno.Entorno;

public class Puntaje {
    private int puntos;

    public Puntaje() {
        this.puntos = 0;
    }

    public void saltarbarril() {

        this.puntos += 15;
    }

    public void ganar() {

        this.puntos += 100;
    }

    public void dibujar(Entorno entorno) {

        entorno.cambiarFont("terminal", 18, Color.GREEN);

        entorno.escribirTexto("Puntos: " + String.valueOf(this.puntos), 685, 15);
    }

}
```