INFORME GRUPO 7

Trabajo Práctico: Attack on Titan, Final Season

Integrantes:

- Karin Daniela Pellegrini Farnholc - karinfarnholc@gmail.com
- Facundo Martin Gil - facugil61@gmail.com
- Luca Diaz - luca-diaz@live.com

Fecha de entrega: 24/05/2022

Primer Semestre 2022.

Programación I, COM-04 - Universidad de General Sarmiento.

Introducción al juego “Attack on Titan, Final Season”

El trabajo práctico consiste en la implementación de la parte lógica para lograr el correcto funcionamiento del videojuego “Attack on Titan, Final Season”.
Los alumnos tienen la tarea de implementar diversas clases, y sus respectivos métodos, para lograr el funcionamiento de los distintos apartados obligatorios del juego, los cuales son:

- La creación de un personaje principal (Mikasa) el cual el usuario debe ser capaz de manejar a su voluntad,
- enemigos (Kyojines) que aparecen cada cierto tiempo y que deben perseguir a Mikasa con el objetivo de eliminarla sin salirse de los márgenes de la pantalla ni atravesar obstáculos,
- un proyectil con el cual Mikasa puede ejecutar a los Kyojines con el objetivo de ganar la partida eliminandolos a todos,
- obstáculos que impidan el paso a Mikasa y a los enemigos,
- sueros que aparecen cada cierto tiempo y permiten a Mikasa transformarse y eliminar un Kyojin cuando entre en contacto con uno.

Por otra parte, los alumnos deben implementar un código con un buen diseño y legible para el fácil entendimiento.

Descripción

A continuación se describirán las distintas clases que conforman el videojuego.

Clase Juego:

Esta clase se encarga del correcto funcionamiento del videojuego.
La misma posee trece variables de instancia.

1. **entorno**: Esta variable de tipo Entorno crea la pantalla del juego con un ancho y alto determinados.
2. **mika**: La variable mika es una de tipo Mikasa. La misma posee al personaje principal y sus respectivos datos y métodos.
3. **enemigos**: Esta variable de tipo Kyojin posee un arreglo de los recién mencionados con sus métodos definidos en su clase.
4. **proyectil**: Esta variable de instancia de tipo Proyectil posee los datos del mismo.
5. **sueros**: Esta variable de tipo Suero posee un arreglo de los mismos.
6. **obstáculos**: Variable de tipo Obstáculo que posee un arreglo de los mismos.
7. **fondo**: Esta variable de tipo Image posee la imagen de fondo que es dibujada en cada instante del videojuego.
8. **contadorDeTicks**: Esta variable de tipo entero cuenta la cantidad de ticks en cada momento del juego.
9. **puntos**: Variable de tipo entero que posee el acumulado del puntaje del jugador, el cual se actualiza por cada enemigo eliminado.
10. **kyojinesEliminados**: Variable de tipo entero que contiene la cantidad de Kyojines eliminados por Mikasa.
11. **vidas**: Esta variable de tipo entero contiene la cantidad de vidas restantes de Mikasa.

12. **tiempoDeInvulnerabilidad**: Esta variable de tipo entero indica el tiempo restante de inmunidad que posee Mikasa.
13. **pantallaInicioActiva**: Variable booleana que indica si debe mostrarse la pantalla de inicio.

Para comenzar el juego, se carga la imágen del fondo creada por los alumnos. En la variable de instancia de tipo Image “fondo” se utiliza la función que ya venía realizada previamente en el paquete de archivos del entorno, en la clase “Herramientas”: cargarImagen, en donde se pasa en forma de “String” la carpeta y el archivo de la imagen. Luego se le asignan valores a varias variables de instancia, y se crean objetos que se deben mostrar obligatoriamente en el instante que se corre el juego; cuatro obstáculos con posiciones predefinidas (facilita el movimiento de los enemigos), cuatro enemigos con posiciones aleatorias (mientras que no se superpongan con un obstáculo o enemigo), un suero con posición aleatoria (mientras que no se superpongan con un obstáculo o enemigo). Y , finalmente, se inicia el juego.

En éste apartado, se explicará el funcionamiento del juego dentro del **tick**, que es la medición de tiempo del juego. Es decir, lo que se realiza en cada instante del juego:
En cada tick, se ejecuta el método nombrado. El mismo posee ciertas verificaciones para comprobar que el juego cumple con lo requerido para funcionar en óptimas condiciones.
En primera instancia, se muestra una pantalla de inicio al instante de comenzar el videojuego. La variable de instancia booleana “pantalla Inicio Activa” es declarada como “true” en el constructor de la clase, por ende, se mostrará inicialmente dicha pantalla. Mientras esta variable continúe siendo true, el tick no se ejecutará en su completitud, y solo se mostrará la imagen de fondo, la posición inicial de Mikasa, de los Kyojines, y de los obstáculos en pantalla. Si el usuario oprime la tecla Enter, la variable cambiará su valor a “false” y se ejecutará el resto del tick, y por lo tanto, el juego comenzará sus funcionalidades.
Al comienzo de cada tick, se corroborará si el juego finalizó mediante el método **finalizoJuego()**, la cual anexa otros dos métodos:

- **seGano(**): retorna un booleano “true” si todos los enemigos del juego tienen un valor nulo, es decir, fueron eliminados en su totalidad.
- **sePerdio()**: retorna un booleano “true” y elimina al objeto Mikasa si las vidas llegan a 0.

Si una de estas dos resulta devolver “true”, lo mismo hará el método finalizoJuego(). Si esto sucede, se analizará cuál fue el caso y se mostrará una pantalla correspondiente al resultado de la partida, con el puntaje, enemigos eliminados, y una instrucción para cerrar la pantalla (tecla FIN/END).
Normalmente se ejecutará el resto del tick, que inicialmente dibuja el fondo y otras variables que determinan cómo avanza el videojuego (vidas, puntaje, enemigos eliminados).
El método realizará en cada corrida ciertas validaciones. Las mismas, se llevan a cabo con asistencia de métodos, tanto de la clase Juego, como de los diferentes objetos del juego, lo cual ayuda a que el código sea más legible y evita que el mismo sea repetido innecesariamente. Esta parte es vital ya que comprobando y realizando cambios en los lugares necesarios, se llega al resultado de la partida, ya sea derrota o victoria. Entre estas validaciones, se encuentran principalmente:

- Si Mikasa puede moverse a un lado específico.

- Si Mikasa tocó un suero.
- Si Mikasa se encuentra en un estado de poción activa.
- Si Mikasa posee tiempo de invulnerabilidad.
- Si Mikasa colisionó con un Kyojin.
- Si se presionó la tecla específica para lanzar un misil.
- Si fue creado un suero.
- Si fueron creados dos Kyojines.
- Si un Kyojin se salió de los márgenes de la pantalla.
- Si dos Kyojines colisionaron entre ellos.
- Si un Kyojin puede moverse libremente.
- Si un Kyojin colisionó con un obstáculo.

Éstas son las principales validaciones realizadas en cada tick. Si se cumplen, o no, se actúa en consecuencia:

- Si Mikasa puede moverse a un lado especificado por el jugador (tomado el valor mediante un método del entorno), comprobación realizada mediante el método **colisionMikasaObstaculo<Lado>** y agregando que Mikasa no debe salirse de los márgenes de la pantalla, se la mueve mediante el método **mover<Lado>()** de la clase Mikasa.
- Si Mikasa tocó un suero, las variables de instancia “alto”, “ancho”, y “pociónActiva” del objeto Mikasa son modificados mediante métodos para hacerlo (**cambiarAlto()**, **cambiarAncho()**, **cambiarEstado()**).
- Si Mikasa se encuentra en un estado de poción activa, posee una invulnerabilidad al daño de los Kyojines mientras el estado siga. Si el Kyojin la toca, este muere, y el estado de Mikasa cambia a “false” (**cambiarEstado()**).
- Si Mikasa posee un tiempo de invulnerabilidad (variable de instancia), el cual puede ser otorgado debido a una colisión reciente entre la misma y un Kyojin, es inmune al daño durante el periodo que la invulnerabilidad persista (100 ticks, tiempo considerado razonable por el grupo para el desarrollo de la partida). Además, durante estos instantes, Mikasa, al ser invulnerable, tiene permitido atravesar a los enemigos para alejarse mientras tenga oportunidad. Durante este tiempo, se dibujará en el margen superior derecho de la pantalla un escudo que indica el tiempo de invulnerabilidad restante.
- Si Mikasa colisionó con un Kyojin, comprobado por los métodos **colisionMikasaKyojin<Lado>**, se debe consultar también si está activo tanto el tiempo de invulnerabilidad como si está en un estado de poción activo. Si este último lo está, Mikasa pierde el estado y el Kyojin con el que contactó, muere. Si el tiempo de invulnerabilidad está activo, no sucede nada. Si no está activo ninguno de los dos factores mencionados, Mikasa pierde una vida (comienza con tres), y se le asignan cien ticks de invulnerabilidad.
- Si se presionó la tecla específica para lanzar un misil (barra espaciadora), se invoca al método de Mikasa **disparar()**, el cual crea un proyectil en la clase de Mikasa y lo retorna con el sentido al que haya mirado Mikasa al momento de presionar la barra.
- Si fue creado un suero en el presente tick, a la variable booleana seCreoEnTick se le asigna el valor “true”. Es decir, solo se creará como máximo un suero en un tick.
- Si fueron creados dos Kyojines, el condicional para crear Kyojines cada cierto periodo de tiempo (200 ticks, tiempo considerado óptimo para el grupo para la

dificultad) no puede cumplirse. Es decir, solo se crearán como máximo dos Kyojines en un tick.

- Si un Kyojin se salió de los márgenes de la pantalla, comprobado calculando el lado del Kyojin y corroborando que no sobreponga el ancho o alto de la pantalla respectivamente, se llamará al método de la clase Kyojin **mover<Lado>Choque()**, el cuál permitirá contrarrestar el movimiento indebido del titán. Esta implementación se debe únicamente a que, en las colisiones entre Kyojines, se los separa en direcciones opuestas si llegan a tocarse, por lo que si uno de los involucrados se encuentra cerca de un márgen, puede llegar a salirse.
- Si dos Kyojines colisionaron entre ellos, comprobación realizada por el método de Juego **colisionKyojinKyojin<Lado>()**, se llama al método de la clase Kyojin **mover<Lado>Choque()** para separar en direcciones opuestas a los Kyojines involucrados.
- Si un Kyojin puede moverse libremente, es decir, no está limitado por un obstáculo cercano, un margen, u otro Kyojin, se le permite decidir por sí mismo mediante el método **mover<Lado>(x/y de Mikasa)** (en la clase Kyojin), si debe moverse en “x” o “y” en relación a la posición en la que se encuentre Mikasa.
- Si un Kyojin colisionó con un obstáculo, comprobación realizada mediante los métodos de la clase juego **colisionKyojinObstaculo<Lado>**, se lo mueve en sentido contrario a las agujas del reloj rodeando el obstáculo. Por ejemplo, si colisiona con un obstáculo por el lado izquierdo, el Kyojin rodeará al obstáculo yendo por arriba.

En otro aspecto, el **tick** se encarga de dibujar a los distintos objetos presentes en el juego. Para ello, también se corrobora antes que el objeto que se está queriendo dibujar no sea nulo. Por encima, varios objetos deben cumplir ciertas condiciones para ser dibujados de una determinada manera.
Por ejemplo, Mikasa posee cinco métodos para ser dibujada; dibujarMovimiento, dibujarQuieta, dibujarDisparando, dibujarDisparandoKyojin, y dibujarTransformada.
Para que se cumpla cada una, deben darse ciertas condiciones.

1. Si se está presionando continuamente una tecla, se dibujará a **Mikasa** con un archivo “.gif”, el cual superpone dos imágenes y crea una sensación de **movimiento**.
2. Por otra parte, si no se presionó continuamente una tecla de movimiento, se dibujará a **Mikasa quieta** y mirando para el sentido que tenga guardado (último lado para el cual se dirigió).
3. Si el proyectil de la clase Juego no es nulo, es decir, fue disparado y aún sigue en trayectoria, se dibujará a **Mikasa** con una foto simulando un **disparo**.
4. Si añadido a esto, Mikasa está en estado de poción activo, se la dibujara **disparando** con una imágen de **Kyojina**.
5. Si está transformada sin disparar, se la dibujará con el método **dibujarTransformada**.

Además, se creó una variable booleana “fotoActiva” que comienza declarada en “false”, y que si se comienza a dibujar una imágen se la cambia a “true”. Esta variable sirve de referencia para no dibujar varias imágenes de Mikasa que se solapen si se presionan dos teclas al mismo tiempo.
Con los Kyojines ocurre algo más simplificado.

1. Si la coordenada “x” de Mikasa es menor que la coordenada “x” del Kyojin, se lo dibujará mediante un archivo .gif mirando para el **lado izquierdo**.
2. Si la coordenada “y” de Mikasa es mayor que la coordenada “x” del Kyojin, se lo dibujará mediante un archivo .gif mirando para el **lado derecho**.

Los objetos de tipo Obstáculo difieren al ser dibujados dependiendo el tipo que sea (variable de instancia de su clase). Puede tomar dos valores; “árbol” o “casa”. Según el tipo, se dibujara una foto correspondiente.
El proyectil se dibuja mediante un archivo .gif según la dirección que posea.
Los sueros tienen una sola imágen y no varía por ningún factor.

En otro aspecto del juego, existe la variable de instancia “contadorDeTicks” que mantiene el acumulado de ticks desde que comenzó la partida. Mediante esta variable, se logra que aparezcan tanto enemigos, como sueros, cada cierto periodo de tiempo determinado por el grupo, el cual considero que los implementados le dan cierta dificultad al juego. De manera complementaria, tanto como los enemigos como los sueros deben crearse en unas coordenadas limpias de colisiones. Es decir, no pueden estar colisionando ni con Mikasa, ni con un obstáculo, ni con otro Kyojin existente.

- ➔ Los sueros aparecen (si el arreglo de sueros no está lleno ni se creó uno en el tick) cada 500 ticks.
- ➔ Los Kyojines aparecen (si el arreglo de enemigos no está lleno ni se crearon dos en el tick) cada 200 ticks.

El tiempo de invulnerabilidad anteriormente mencionado, fue implementado de manera necesaria ya que, al correrse tan rápidos los ticks por segundo, si un enemigo tocaba a Mikasa, le quitaba las tres vidas en 3 ticks, haciendo inútiles a las vidas implementadas. De esta manera, se necesitó implementar la mencionada variable de instancia que a la que se le asigna un valor predeterminado cada vez que tocan a Mikasa y es restado en 1 por cada tick (Mikasa posee 100 ticks de invulnerabilidad).

<u>Métodos de la clase:</u>

Juego: En este apartado constructor de la clase Juego, se definen ciertas variables de instancia y se crean los objetos y funcionalidades necesarias para poder comenzar el juego, al final del constructor, se inicia el entorno con las variables ya definidas.

darX/Y<Lado>: Estos cuatro métodos (darXIzq, darXDer, darYSup, darYInf) retornan un entero aleatorio separando la pantalla en cuatro partes (superior izquierda, superior derecha, inferior izquierda, inferior derecha)

randomPos: creamos una variable tipo random llamada x que genera un valor aleatorio entre 0 y 1. Luego en la variable de tipo entero valorRandom se almacena el valor de x y luego se retorna.

seGano: La función seGano determina, si todos los enemigos son “null”, que el jugador ganó.

sePerdio: en esta función si las vidas son 0, el valor de mikasa se iguala a null para eliminarla y si la condición se cumple devuelve true. Mientras esto no suceda devuelve false.

terminoJuego: Esta función se encarga de determinar, mediante las últimas dos funciones mencionadas, si el juego debe darse por finalizado.

colisionMikasaKyojin<Lado>: Estos cuatro métodos (colisionMikasaKyojinDer, colisionMikasaKyojinIzq, colisionMikasaKyojinArriba, colisionMikasaKyojinAbajo), determinan si existe una colisión en el presente tick entre Mikasa y algún Kyojin pasado por índice.
Para escribir estos métodos, tomamos lados necesarios para hacer la corroboración con el objetivo de simplificar el código. Devuelve “true” si; el lado corroborando esta dentro del lado opuesto del Kyojin pero no llega a tocar al lado paralelo al corroborado, y además que el valor “y” o “x” de Mikasa (dependiendo el método), están en el mismo rango del correspondiente valor del Titan.
Por ejemplo: Para que la colisión de Mikasa con un Kyojin por el lado derecho de verdadero, el lado derecho de Mikasa tiene que ser mayor que el lado izquierdo del titán, menor que el lado derecho del mismo, y además el “y” inferior y superior de Mikasa debe estar entre los mismos nombrados correspondientes al Kyojin (esto se debe a que el alto del Kyojin es mayor al de Mikasa).

colisionMikasaObstaculo<Lado>: Estos cuatro métodos de tipo booleano recorren el arreglo de obstáculos con un for y se crean las variables xLado<Lado> en donde se guardan la coordenada x (si solicitamos el lado derecho o izquierdo de Mikasa) o y (si pedimos las coordenadas de la parte de arriba/abajo del personaje). Esto se realiza con la función getXLado<Lado> que se encuentran en el archivo de Mikasa.
Luego hace esto mismo con los obstáculos.
Finalmente en un condicional evaluamos si la coordenada del lado de Mikasa que queremos averiguar más 4 es:

- Si el lado que queremos averiguar es abajo o derecha se utiliza un mayor o igual.
- Si es arriba o izquierda utilizamos menor o igual.

Al lado contrario del obstáculo (arriba->abajo, izquierda->derecha y viceversa) y así con todos los lados. Si esto sucede devuelve true o si no falso.

colisionKyojinObstaculo<Lado>: Estos cuatro métodos de tipo booleano recorren el arreglo de obstáculos con un for y se crean las variables xLado<Lado> en donde se guardan la coordenada x (si solicitamos el lado derecho o izquierdo del Kyojin) o y (si pedimos las coordenadas de la parte de arriba/abajo del Kyojin. Esto se realiza con la función getXLado<Lado> que se encuentran en el archivo de Kyojin.
Luego hace esto mismo con los obstáculos.
Finalmente en un condicional evaluamos si la coordenada del lado que queremos averiguar más 4 es:

- Si el lado que queremos averiguar del Kyojin es abajo o derecha se utiliza un mayor o igual.

- Si es arriba o izquierda utilizamos menor o igual.

Al lado contrario (arriba->abajo, izquierda->derecha y viceversa) y así con todos los lados. Si esto sucede devuelve true o si no falso.

colisionKyojinKyojin<Lado>: estas funciones de tipo booleano se utilizan para evaluar si hay Kyojines colisionando entre sí. Se le pasa como parámetro el índice del Kyojin y el índice del segundo Kyojin con el que lo queremos comparar. Luego se realiza un condicional que compara si estos índices son iguales o si la posición del segundo Kyojin en el arreglo de enemigos es nula (ya que el Kyojin puede haber sido eliminado por Mikasa). Cuando esto sucede se retorna falso.
En caso contrario, se crean variables de tipo entero que almacenan las coordenadas x (si solicitamos el lado derecho o izquierdo del Kyojin) o y (si pedimos las coordenadas de la parte de arriba/abajo del Kyojin) y lo mismo se hace con las coordenadas del segundo Kyojin. Luego si la condición del return se cumple devuelve true.

colisionKyojinKyojinGeneral: Este método reúne los cuatro métodos de colisionKyojinKyojin<Lado>. Es utilizado en el tick para corroborar que al momento de crear un nuevo objeto (Kyojin o Suero) no se superpongan.

impactoProyectilEnemigo: Este método devuelve un booleano que depende de si el proyectil impactó con un enemigo. Para determinarlo, se corrobora que la punta del proyectil impacta con la cara visible del enemigo, es decir, el lado contrario al sentido del proyectil.

impactoProyectilObstaculo: Al igual que el método de impactoProyectilEnemigo, éste devuelve un booleano que depende de si el proyectil impactó con un obstáculo. Se toma en consideración el mismo mecanismo, es decir, si la punta del proyectil impacta con la cara visible del obstáculo, se considera true.

deteccionProyectilErrado: Este método recibe un índice de un obstáculo para corroborar colisiones mediante los sentidos del proyectil en el método impactoProyectilObstaculo.

tocoSuero: Este método devuelve un booleano que determina si Mikasa toca el suero para convertirse. Para esto se realiza un for que recorre todos los sueros (siempre que estos no sean null) y se utilizan condicionales basados en los lados de Mikasa y los lados del suero para que se corrobore la colisión entre estos.
Por ejemplo, para que se determine que está colisionando Mikasa con el suero por su parte derecha, se comprueba que el lado derecho de Mikasa sea igual o mayor al lado izquierdo del suero pero menor que su lado derecho. Además, para que se cumpla la condición, el valor de “y” de Mikasa tiene que cumplir que toque al suero por arriba a la derecha (con el borde superior de Mikasa fuera del suero) o, que Mikasa toque al suero por abajo a la derecha (con el borde inferior de mikasa fuera de contacto con el suero).

colisionSueroObstaculo: En esta función booleana se solicita un índice para luego guardar las coordenadas de los sueros y los objetos en variables. Luego se compara que haya una colisión evaluando si hay un choque entre caras opuestas (es decir, por ejemplo si la colisión del suero se da del lado izquierdo es porque está haciendo contacto con el lado derecho del obstáculo). Si esto ocurre se devuelve true.

colisionSueroKyojin: Este método retorna un booleano “true” si se comprueba que colisionó un suero determinado pasado por parámetro con un Kyojin. En el método se recorre al arreglo de enemigos para hacer la comprobación. El método es únicamente utilizado para evitar que se cree un Suero superpuesto con un Kyojin. Si no se comprueba, retorna “false”.

<u>En esta clase nos encontramos con las siguientes dificultades:</u>
En primer lugar, las colisiones entre Mikasa y los obstáculos nos generó dificultades para escribirlo y tenerlas funcionales. Para solucionar este problema, pensamos las colisiones basándonos en un pequeño esquema (*Fig 1.*) el cual nos dió las bases para entender cómo funcionaban los valores de “X” e “Y” vinculados al contacto entre los objetos nombrados.
Mikasa puede colisionar un obstáculo desde determinado lado, pero a cada lado se lo parte en dos opciones. Por ejemplo, si Mikasa colisiona con el obstáculo yendo hacia la derecha, puede hacerlo por el lado derecho superior, o el lado derecho inferior. Esto nos sirvió para lograr que no haya puntos ciegos las colisiones y todas las opciones de colisión estén cubiertas.

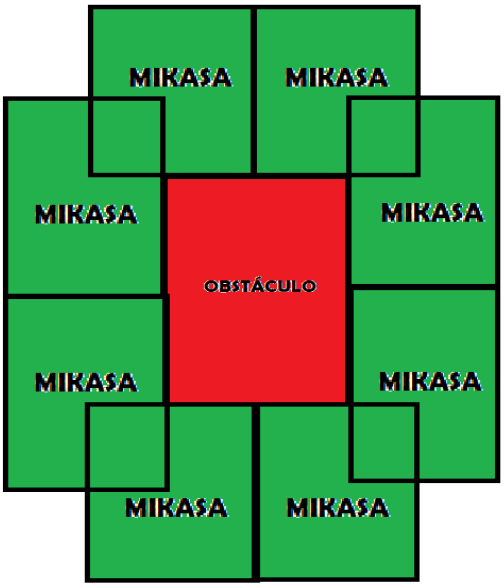
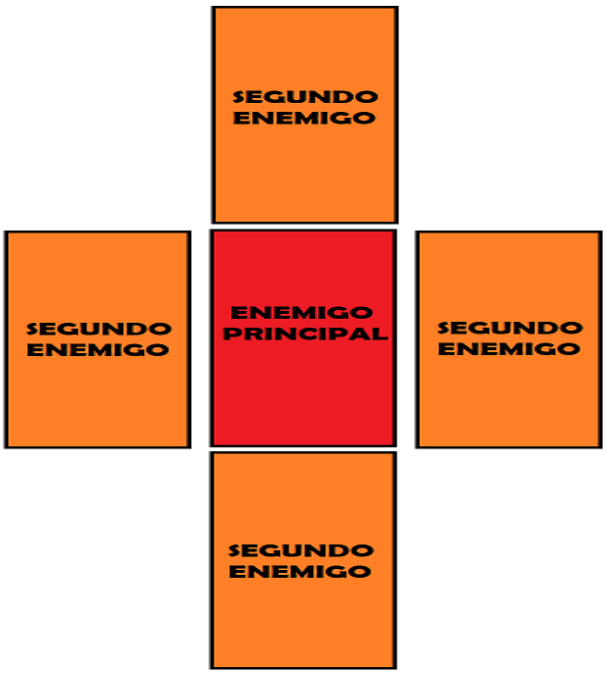


Fig. 1

En segundo lugar, la colisión de Kyojin-Kyojin también nos causó problemas incluso siguiendo las bases de este último esquema, por lo tanto tuvimos que adaptar la idea (*Fig 2*). Para implementar esta colisión, sí utilizamos un código parecido a la comprobación del choque entre Mikasa y un obstáculo, pero en el tick realizamos más chequeos. Por ejemplo, hay variables que contienen ciertas condiciones para que el Kyojin pueda moverse para determinado lado, en ellas se encuentran que el Kyojin no se salga de pantalla a voluntad, y que el mismo no esté colisionando con ningún obstáculo, ya que si se da esto, se llama a otro método para mover al enemigo, moverChoque<Lado>. Éste método se usa en los momentos que el Kyojin esté colisionando con algún obstáculo.
Al mismo tiempo, se realiza una comprobación de colisión entre Kyojines, si hay una colisión general y el Kyojin analizado no es el mismo, se actúa en consecuencia en base al lado por el que se colisione, si se colisiona por los ejes verticales (arriba, abajo), entonces se los separa para el lado opuesto por el que sea el choque. La práctica es la misma para el eje horizontal.



(*Fig. 2*)

Finalmente otra colisión que causó problemas fue la de Mikasa-Kyojin. Debido a que la altura de Mikasa con los enemigos difiere, la colisión debió ser pensada de distinta manera al método colisionKyojinKyojin. Para solucionarlo nos basamos en el siguiente esquema(*Fig. 3.*). Para que se confirme una colisión, un lado de Mikasa debe chocar con el lado opuesto del enemigo y los lados perpendiculares al lado que choca deben estar dentro de los correspondientes del Kyojin.
Por ejemplo, si se colisiona por el lado de arriba de Mikasa, el lado izquierdo debe ser menor que el lado derecho del Kyojin, y el lado izquierdo también debe ser mayor que el lado derecho del Kyojin.

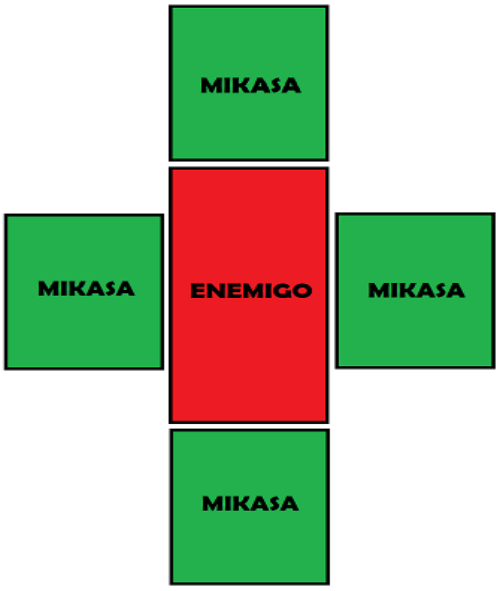


Fig 3.

Por otra parte, cuando terminaba la partida y se apretaba la tecla fin, la pantalla del juego se cerraba pero la ejecución del mismo continuaba, esto debido a que usábamos

this.entorno.dispose() que solo cierra la ventana pero no finaliza el videojuego. Para solucionar este problema utilizamos el método “exit” de la clase System, el cual termina la Máquina Virtual de Java (JVM) que sale del programa actual que estamos ejecutando. Para lograr esto le pasamos un 0 como parámetro.

Finalmente, al querer utilizar los métodos incluídos en la clase herramienta para musicalizar el videojuego aparecía el siguiente error: javax.sound.sampled.UnsupportedAudioFileException: URL of unsupported format. Esto sucedía debido a que el formato .wav que queríamos utilizar no era el correcto, este debe ser de 8khz, 16 bit y mono. Para esto utilizamos el siguiente conversor de audio: https://convertio.co/es/ donde los usuarios pueden introducir los sonidos, seleccionar el formato que necesitan y descargarlo.

Clase Mikasa:

Ésta clase representa a la personaje principal del videojuego, Mikasa, y posee métodos para dibujarla en el juego, disparar proyectiles, y devolver varias variables.

Tiene 7 variables de instancia:

- Enteros “x” e “y” para guardar sus coordenadas en pantalla.
- Enteros “alto” y “ancho” para guardar el espacio que ocupa en pantalla.
- Una cadena llamada “sentido” que contiene la dirección a la que Mikasa avanza y dispara.
- La variable booleana pocionActiva que vale true si Mikasa está poderosa, es decir, si tomó el suero.
- La variable Image “imagen” a la que se le asigna una imagen para el personaje.

Métodos de la clase:

Mikasa: Este método constructor define los parámetros necesarios para crear un personaje de tipo Mikasa en la clase Juego.

dibujarMovimiento: Recibe como parámetro el entorno del juego y dibuja a Mikasa en movimiento (solo derecha o izquierda) ya que se utilizan distintos gifs de movimiento dependiendo qué sentido tiene Mikasa. Si va hacia el lado izquierdo el personaje en el gif debe estar mirando hacia ese lado, por ejemplo.

dibujarQuieta: Realiza lo mismo que **dibujarMovimiento** pero esta función se utiliza cuando ella se queda quieta. Si no utilizáramos esta función cuando ella está quieta las piernas se continuarían moviendo.

dibujarDisparando: Realiza lo mismo que **dibujarMovimiento** y **dibujarQuieta** pero se insertan las imágenes en las que Mikasa está disparando el cohete.

dibujarDisparandoKyojin: Realiza lo mismo que **dibujarDisparando** pero en este caso se utilizan las imágenes de Mikasa disparando transformada en Kyojina.

dibujarTransformada: Dibuja en pantalla a Mikasa cuando está transformada en Kyojina y está en movimiento.

mover<Arriba/Abajo>: Funciones que se encargan de modificar la variable sentido asignándole hacia qué dirección se está dirigiendo Mikasa y de sumarle (si va hacia abajo) o restarle (si va hacia arriba) la variable “y” de la posición del personaje para que pueda moverse en pantalla. Va de 3 en 3 para que la velocidad a la que se mueve no sea tan lenta ni tan rápida.

mover<Derecha/Izquierda>: Hace lo mismo que mover<Arriba/abajo> pero le suma/resta a la coordenada x.

getX y getY: Devuelve la coordenada x/y en la que se encuentra el personaje.

getAncho y getAlto: Devuelve el valor del ancho/alto de Mikasa.

getSentido: Retorna el sentido hacia el que está “mirando” Mikasa.

getEstado: Retorna si Mikasa consumió algún suero, es decir, si la variable pocionActiva es verdadera o falsa.

cambiar<Alto/Ancho>: Permite asignar un nuevo valor a las variables ancho y alto para modificar el tamaño de Mikasa.

cambiarEstado: Permite asignar un nuevo valor a la variable pocionActiva para cambiar el estado de Mikasa.

disparar: Dependiendo el sentido hacia el que está el personaje va a tener cierta coordenada “x” y cierta coordenada “y”, ya que sino podría salir el proyectil desde fuera del personaje. Para que esto no ocurra se modifica la variable “x” o la variable “y” del método Proyectil y se guarda en la variable p1 para luego ser retornada.

getXLado<Derecho/izquierdo>: Retorna el valor del lado correspondiente (Derecho, Izquierdo) de Mikasa en pantalla. Para esto se suma/resta el valor de la coordenada “x” más/menos (según si es derecha o izquierda) el ancho del personaje dividido 2.

getXLado<Arriba/Abajo>: Busca hacer lo mismo que getXLado<Derecho/izquierdo> pero con el extremo inferior y superior del personaje. Para esto se suma/resta el valor de la coordenada “y” más/menos el alto del personaje dividido 2.

Clase Kyojin:

Ésta clase posee métodos de movimiento, dibujo y retorno de varias variables de los Kyojines, enemigos del videojuego.

Tiene 4 variables de instancia:

- Coordenadas x e y para obtener su ubicación.
- Ancho y alto para saber cuánto espacio ocupan en la pantalla.

Métodos de la clase:

Kyojin: Éste método constructor define los parámetros necesarios para crear un personaje de tipo Kyojin en la clase Juego.

dibujar<Derecha/Izquierda>: Dibuja en pantalla un gif del Kyojin según la dirección que tome.

mover<Lado>Choque: Estos métodos mueven al titán (aumentando las coordenadas que correspondan según el lado) si se encuentra colisionando con un obstáculo u otro titán. En éste método, se define que todos los lados se mueven +2 (+1 en relación a mover<Lado>). Esto se debe a que si se pone el movimiento de choque (colisión con un obstáculo u otro Kyojin) en +1, entra en conflicto con mover<Lado>, y los enemigos se quedan trabados en las colisiones.

mover<Lado>: Recibe una coordenada de Mikasa y actúa dando movimiento al titán por la pantalla si se cumplen las condiciones de cada lado. Por ejemplo, para que un Kyojin se mueva para abajo, se debe cumplir que el “y” del Kyojin sea menor que el “y” de Mikasa pasado por parámetro (Mikasa está por debajo del Kyojin).

getX y getY: Devuelve la coordenada “x”/”y” en la que se encuentra el Kyojin.

getAncho y getAlto: Devuelve el valor del ancho/alto del Kyojin para saber cuánto espacio ocupa en pantalla.

getXLado<Derecho/izquierdo>: Retorna el valor del lado correspondiente (Derecho, Izquierdo) del Kyojin en pantalla. Para esto se suma/resta el valor de la coordenada “x” más/menos (según si es derecha o izquierda) el ancho del enemigo dividido 2.

getXLado<Arriba/Abajo>: Busca hacer lo mismo que getXLado<Derecho/izquierdo> pero con el extremo inferior y superior del enemigo. Para esto se suma/resta el valor de la coordenada “y” más/menos el alto del Kyojin dividido 2.

En esta clase, tuvimos el siguiente problema:

En ciertas condiciones de la clase Juego, se contrarrestaban los métodos **mover<Lado>** y **mover<Lado>Choque**, por lo que los enemigos, al llegar a un obstáculo, continuaban quedándose trabados. El problema fue solucionado haciendo que **mover<Lado>Choque** tenga un valor más alto (± 2) que **mover<Lado>** (± 1).

Clase Obstáculo:

Ésta clase brinda métodos para dibujar y retornar varias variables de los objetos Obstáculo. En el juego, su función es dificultar el paso a Mikasa y a los Kyojines en el juego.

Contiene 6 variables de instancia:

- Coordenadas “x” e “y” para obtener su ubicación.
- Variables “ancho” y “alto” para saber cuánto espacio ocupan en la pantalla.
- Una cadena “tipo” a la que le asignamos qué obstáculo es. Hay dos opciones posibles: una casa y un árbol.
- La variable Image imagen a la que se le asigna una imagen para el obstáculo según su tipo.

Obstáculo: Este método constructor define los parámetros necesarios para dibujar un obstáculo en la clase Juego.

dibujar: Se pasa como parámetro el entorno y según el tipo de obstáculo dibuja en pantalla un árbol o una casa en la posición indicada.

getX y getY: Devuelve la coordenada x/y en la que se encuentra el obstáculo.

getAncho y getAlto: Devuelve el valor del ancho/alto del objeto para saber cuánto espacio ocupa en pantalla.

getTipo: Retorna qué tipo de obstáculo es (“árbol”/”casa”).

getXLado<Derecho/izquierdo>: Retorna el valor del lado correspondiente (Derecho, Izquierdo) del obstáculo en pantalla. Para esto se suma/resta el valor de la coordenada “x” más/menos (según si es derecha o izquierda) el ancho del obstáculo dividido 2.

getXLado<Arriba/Abajo>: Busca hacer lo mismo que getXLado<Derecho/izquierdo> pero con el extremo inferior y superior del obstáculo. Para esto se suma/resta el valor de la coordenada “y” más/menos el alto del obstáculo dividido 2.

Clase Proyectil:

Ésta clase es la encargada de proveer métodos de utilidad y el constructor a la clase Mikasa, la encargada de crear un proyectil, utilizado para eliminar Kyojines.

Tiene 5 variables de instancia:
Coordenadas x e y para obtener su ubicación.
Ancho y alto para saber cuánto espacio ocupan en la pantalla.
Sentido para saber a donde va a ir dirigido el proyectil.

Los métodos de esta clase son los siguientes:

Proyectil: Este método constructor define los parámetros necesarios para crear un proyectil en la clase Juego.

dibujar: Este método dibuja el proyectil cargando la imagen que corresponda al sentido del mismo. Para esto contamos con 4 gif diferentes para cuando el sentido sea arriba, abajo, derecha, izquierda.

mover: Este método mueve el proyectil dependiendo el sentido que este adquiera. Para hacerlo le sumamos o restamos 4 al valor de x o y dependiendo de a donde esté dirigido el proyectil.

getSentido: Este método devuelve el sentido del proyectil.

getY: Este método devuelve el valor de Y.

getX: Este método devuelve el valor de X.

getBordeProyectil: Este método devuelve el borde del proyectil según la dirección que posea.

Clase Suero:

Ésta clase es la encargada de proveer el objeto Suero que es utilizado en el videojuego por Mikasa para transformarse en una versión más poderosa de sí misma. Además, devuelve varias variables mediante métodos.

Tiene 4 variables de instancia:
Coordenadas “x” e “y” para obtener las coordenadas del suero.
“ancho” y “alto” para saber cuánto espacio ocupan en la pantalla.

Los métodos de esta clase son los siguientes:

Suero: Este método constructor define los parámetros necesarios para crear un suero en la clase Juego.

dibujar: Este método dibuja el suero cargando una imagen de tipo png.

getX: Este método devuelve la coordenada x en la que se encuentra el suero.

getY: Este método devuelve la coordenada y en la que se encuentra el suero.

getAncho: Este método devuelve el ancho del suero.

getAlto: Este método devuelve el alto del suero.

getXLado<Derecho/izquierdo>: Retorna el valor del lado correspondiente (Derecho, Izquierdo) del suero en pantalla. Para esto se suma/resta el valor de la coordenada “x” más/menos (según si es derecha o izquierda) el ancho del suerodividido 2.

getXLado<Arriba/Abajo>: Busca hacer lo mismo que getXLado<Derecho/izquierdo> pero con el extremo inferior y superior del suero. Para esto se suma/resta el valor de la coordenada “y” más/menos el alto del suero dividido 2.

Implementación:

```
package juego;
import java.awt.Color;
import java.awt.Image;
import java.awt.Font;
import java.util.Random;
import entorno.InterfaceJuego;
import entorno.Entorno;
import entorno.Herramientas;

public class Juego extends InterfaceJuego
{
	// El objeto Entorno que controla el tiempo y otros
	private Entorno entorno;

	// Variables y metodos propios de cada grupo
private Mikasa mika;
private Kyojin[] enemigos;
private Proyectil proyectil;
private Suero[] sueros;
private Obstaculo[] obstaculos;
private Image fondo;
private int contadorDeTicks;
private int puntos;
private int kyojinesEliminados;
private int vidas;
private int tiempoDeInvulnerabilidad;
private boolean pantallaInicioActiva;
Juego()
{
	// Inicializa el objeto entorno
	this.entorno = new Entorno(this, "Attack on Titan, Final Season -
Grupo 7- v1", 800, 600);

	// Inicializar lo que haga falta para el juego

	// Carga valores a ciertas variables de instancia.
	this.fondo = Herramientas.cargarImagen("Imagenes/fondo.png");
	this.contadorDeTicks = 0;
	this.puntos = 0;
	this.kyojinesEliminados = 0;
	this.vidas = 3;
```

```
	this.tiempoDeInvulnerabilidad = 0;
	this.pantallaInicioActiva = true;

	//Mikasa
	this.mika = new Mikasa(400,300,40,70,"derecha", false,"mikasaMov1");

	//Obstaculos
	this.obstaculos = new Obstaculo[4];
	// Crea 4 obstaculos para comenzar el juego distribuidos en 4 partes de la pantalla (arribaIzq, arribaDer, abajoIzq, abajoDer).
	this.obstaculos[0] = new Obstaculo(200, 175, 70, 70, "arbol"); //Crea un obstaculo de tipo arbol
	this.obstaculos[1] = new Obstaculo(200, 425, 70, 70, "casa"); //Crea un obstaculo de tipo casa
	this.obstaculos[2] = new Obstaculo(600, 425, 70, 70, "arbol"); //Crea un obstaculo de tipo arbol
	this.obstaculos[3] = new Obstaculo(600, 175, 70, 70, "casa"); //Crea un obstaculo de tipo casa

	//Enemigos

	this.enemigos = new Kyojin[4];

	for (int i = 0; i < 4; i++) { //Crea 4 Kyojines para comenzar el juego.
		int derOIzq = this.randomPos(); //Numero random entre 0 y 1 para definir derecha o izquierda.
		int supOInf = this.randomPos(); //Numero random entre 0 y 1 para definir superior o inferior.
		int yKyojin;
		int xKyojin;

		if(derOIzq == 0) {
			xKyojin = this.darXDer();
		}
		else {
			xKyojin = this.darXIzq();
		}
		if(supOInf == 0) {
			yKyojin = this.darYSup();
		}
		else {
			yKyojin = this.darYInf();
		}
		this.enemigos[i] = new Kyojin(xKyojin, yKyojin, 60, 140); //Recibe un X y un Y random para darle una posicion al Kyojin, los valores ancho y alto son fijos.
```

```
            boolean colisionMikasaTitanGeneral =
this.colisionMikasaKyojinAbajo(i) || this.colisionMikasaKyojinArriba(i) ||
this.colisionMikasaKyojinDer(i) || this.colisionMikasaKyojinIzq(i);
            boolean colisionKyojinObstaculoGeneral =
this.colisionKyojinObstaculoAbajo(i) ||
this.colisionKyojinObstaculoArriba(i) ||
this.colisionKyojinObstaculoDer(i) || this.colisionKyojinObstaculoIzq(i);
            if(colisionKyojinObstaculoGeneral ||
colisionMikasaTitanGeneral || this.colisionKyojinKyojinGeneral(i)) { //Si
hay colisiones al momento de inicializar al titan, se lo borra y se resta
1 a la variable "i" (para crear otro).
                  this.enemigos[i] = null;
                  i--;
            }

      }

      //Sueros
      this.sueros = new Suero[3];
      boolean seCreo = false;
      while(seCreo != true) {
            seCreo = true;
            int derOIzq = this.randomPos();
            int supOInf = this.randomPos();
            int xSuero;
            int ySuero;
            if(derOIzq == 0) {
                  xSuero = this.darXDer();
            }
            else {
                  xSuero = this.darXIzq();
            }
            if(supOInf == 0) {
                  ySuero = this.darYSup();
            }
            else {
                  ySuero = this.darYInf();
            }
            this.sueros[0] = new Suero(xSuero,ySuero,20,35);
                  if(this.colisionSueroKyojin(0) ||
this.colisionSueroObstaculo(0) || this.tocoSuero(0)) {
                  seCreo = false; //Recibe un X y un Y random para darle
una posicion al suero. (Crea solo uno para comenzar el juego)
                  }
      }
```

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

}

public void tick()
{
	// Procesamiento de un instante de tiempo
	if(this.pantallaInicioActiva == true) { //Mientras la pantallaDeInicioSeaActiva, se ejecuta este bloque y no ejecuta las funcionalidades principales.
		Image oprimaJugar = Herramientas.cargarImagen("Imagenes/jugar.png");
		Image pantCarga = Herramientas.cargarImagen("Imagenes/pantCarga.png");
		this.entorno.dibujarImagen(this.fondo,400,300,0,1);
		this.mika.dibujarQuieta(entorno);

		for (int i = 0; i < enemigos.length; i++) {
			if(this.enemigos[i] != null) {
				this.enemigos[i].dibujarDerecha(entorno);
			}
		}
		for (int i = 0; i < this.obstaculos.length; i++) {
			this.obstaculos[i].dibujar(entorno);
		}
		this.entorno.dibujarImagen(pantCarga, 410, 60, 0, 1); //Dibuja una pequeña pantalla de carga
		this.entorno.dibujarImagen(oprimaJugar, 410, 540, 0, 1); //Dibuja un cuadro con instrucciones para comenzar el juego.
		if(this.entorno.sePresiono(this.entorno.TECLA_ENTER)) { //Si se oprime enter, comienza el juego.
			this.pantallaInicioActiva = false;
			Herramientas.play("Sonido/musica.wav");
		}
	}
	else {


	//Corrobora si termino el juego, sino, ejecuta el resto del tick.
	if(this.terminoJuego()) {
		this.entorno.dibujarImagen(this.fondo, 400, 300, 0, 1); //Si termino, dibuja el fondo, posibles enemigos restantes, obstaculos y un cuadro correspondiente al resultado obtenido.
		for (int i = 0; i < enemigos.length; i++) {
			if(this.enemigos[i] != null) {
```

```
                        this.enemigos[i].dibujarDerecha(entorno);
                  }
            }
            for (int i = 0; i < this.obstaculos.length; i++) {
                  this.obstaculos[i].dibujar(entorno);
            }
            if(this.seGano()) {
            this.entorno.dibujarRectangulo(410, 300, 600, 300, 0,
Color.WHITE);
            this.entorno.dibujarRectangulo(410, 300, 590, 290, 0,
Color.BLACK);

            this.entorno.cambiarFont(Font.SERIF, 60, Color.GREEN);

            this.entorno.escribirTexto("GANASTE", 280, 250);

            this.entorno.cambiarFont(Font.SERIF, 25, Color.GREEN);

            this.entorno.escribirTexto("Puntaje final: " + this.puntos,
190, 310);
            this.entorno.escribirTexto("Titanes eliminados: " +
this.kyojinesEliminados, 190, 350);
            this.entorno.escribirTexto("Oprima la tecla 'End/Fin' para
cerrar el juego.", 190, 390);
            }
            else {
            this.entorno.dibujarRectangulo(410, 300, 600, 300, 0,
Color.WHITE);
            this.entorno.dibujarRectangulo(410, 300, 590, 290, 0,
Color.BLACK);

            this.entorno.cambiarFont(Font.SERIF, 60, Color.RED);

            this.entorno.escribirTexto("PERDISTE", 280, 250);

            this.entorno.cambiarFont(Font.SERIF, 25, Color.RED);

            this.entorno.escribirTexto("Puntaje final: " + this.puntos,
190, 310);
            this.entorno.escribirTexto("Titanes eliminados: " +
this.kyojinesEliminados, 190, 350);
            this.entorno.escribirTexto("Oprima la tecla 'End/Fin' para
cerrar el juego.", 190, 390);
            }
            if(this.entorno.sePresiono(this.entorno.TECLA_FIN)) { //Si se
presiona la tecla end/fin, se cierra la pantalla.
                  this.entorno.dispose();
```

```
            }
      }
      else {

      //Dibujar fondo
      this.entorno.dibujarImagen(this.fondo, 400, 300, 0, 1);

      //Dibuja las vidas restantes de mikasa.
      Image corazones;
      if(this.vidas == 3) {
            corazones =
Herramientas.cargarImagen("Imagenes/tresCorazones.png");
            this.entorno.dibujarImagen(corazones, 50, 25, 0, 1);
      }
      else if(this.vidas == 2) {
            corazones =
Herramientas.cargarImagen("Imagenes/dosCorazones.png");
            this.entorno.dibujarImagen(corazones, 50, 25, 0, 1);
      }
      else {
            corazones =
Herramientas.cargarImagen("Imagenes/unCorazon.png");
            this.entorno.dibujarImagen(corazones, 50, 25, 0, 1);
      }

      //Contador de ticks y actualizador de ticks de invulnerabilidad (Si
mikasa perdio una vida recientemente.)
      this.contadorDeTicks++;

      if(this.tiempoDeInvulnerabilidad > 0){ //Disminuye en cada tick el
tiempo restante de invulnerabilidad
            this.tiempoDeInvulnerabilidad--;
            Image invulnerabilidad =
Herramientas.cargarImagen("Imagenes/invulnerabilidad2.png");
            if(this.tiempoDeInvulnerabilidad < 50) { //Si falta la mitad
del tiempo de invulnerabilidad, carga la segunda imagen (contiene un 1,
indicando los "segundos" restantes de la misma.)
                  invulnerabilidad =
Herramientas.cargarImagen("Imagenes/invulnerabilidad1.png");
            }
            this.entorno.dibujarImagen(invulnerabilidad, 750, 40, 0, 0.3);
//Dibuja el tiempo de invulnerabilidad restante
      }

      // Dibujar puntaje y kyojines eliminados
      this.entorno.cambiarFont(Font.SERIF, 14, Color.WHITE);
      this.entorno.escribirTexto("PUNTAJE: " + this.puntos, 10, 60);
```

```
      this.entorno.escribirTexto("KYOJINES ELIMINADOS: " +
this.kyojinesEliminados, 620, 590);

      //Mikasa

      boolean condicionMoverArriba = this.entorno.estaPresionada('w' ) &&
this.colisionMikasaObstaculoArriba() != true && this.mika.getY() -
this.mika.getAlto()/2 > 0;
      boolean condicionMoverAbajo = this.entorno.estaPresionada('s') &&
this.colisionMikasaObstaculoAbajo() != true && this.mika.getY() +
this.mika.getAlto()/2 < 600;
      boolean condicionMoverDerecha = this.entorno.estaPresionada('d') &&
this.colisionMikasaObstaculoDer() != true && this.mika.getX() +
this.mika.getAncho()/2 < 800;
      boolean condicionMoverIzquierda = this.entorno.estaPresionada('a')
&& this.colisionMikasaObstaculoIzq() != true && this.mika.getX() -
this.mika.getAncho()/2 >0;

      boolean fotoActiva = false; // Este booleano comprueba que no se
este dibujando ninguna otra foto de movimiento, esto se hace para no
solapar dos imagenes de movimiento.

      if(condicionMoverIzquierda) {//Movimiento del personaje segun la
tecla apretada.
            this.mika.moverIzquierda();
            if(this.proyectil == null && fotoActiva == false) { // Se
comprueba que no haya proyectil activo ya que cuando lo esta tiene su
propia imagen.
                  if(this.mika.getEstado() == false)
                  {
                        this.mika.dibujarMovimiento(entorno);
                  }
                  else  // Si esta transformada, se usa el metodo
dibujarTransformada
                  {
                        this.mika.dibujarTransformada(entorno);
                  }
                  fotoActiva = true;
            }
      }
      if(condicionMoverDerecha) {
            this.mika.moverDerecha();
            if(this.proyectil == null && fotoActiva == false) {
                  if(this.mika.getEstado() == false)
                  {
                        this.mika.dibujarMovimiento(entorno);
                  }
```

```
                else
                {
                        this.mika.dibujarTransformada(entorno);
                }
                fotoActiva = true;
        }
    }

    if(condicionMoverArriba) {
        this.mika.moverArriba();
        if(this.proyectil == null && fotoActiva == false) {
                if(this.mika.getEstado() == false)
                {
                        this.mika.dibujarMovimiento(entorno);
                }
                else
                {
                        this.mika.dibujarTransformada(entorno);
                }
                fotoActiva = true;
        }
    }

    if(condicionMoverAbajo) {
        this.mika.moverAbajo();
        if(this.proyectil == null && fotoActiva == false) {
                if(this.mika.getEstado() == false)
                {
                        this.mika.dibujarMovimiento(entorno);
                }
                else
                {
                        this.mika.dibujarTransformada(entorno);
                }
                fotoActiva = true;
        }
    }

    // Si no se esta presionando ninguna tecla, se dibuja a Mikasa con
una foto quieta. (segun su orientacion tambien)
    if(!condicionMoverArriba && !condicionMoverAbajo &&
!condicionMoverDerecha && !condicionMoverIzquierda && this.proyectil ==
null) {
        if(this.mika.getEstado() == false) {
                this.mika.dibujarQuieta(entorno);
        }
```

```
            else { // Si esta transformada, se usa el metodo
dibujarTransformada
                  this.mika.dibujarTransformada(entorno);
            }
      }

      if (this.mika.getEstado() == true) { //Comprobacion del estado de
mikasa (si tomo el suero)
            this.mika.cambiarAlto(138); //Si lo tomo se cambia la altura,
y luego el ancho, al de un titan
            this.mika.cambiarAncho(40);
      }

      for (int i = 0; i < enemigos.length; i++) {
            if(this.enemigos[i] != null) {
                  boolean colisionMikasaTitanGeneral =
this.colisionMikasaKyojinAbajo(i) || this.colisionMikasaKyojinArriba(i) ||
this.colisionMikasaKyojinDer(i) || this.colisionMikasaKyojinIzq(i);
                  if(colisionMikasaTitanGeneral && this.mika.getEstado())
{ /*Si un titan choca a Mikasa, y esta se encuentra en el estado de
pocion,

                                          se la devuelve a su altura y
ancho base, y se cambia el estado de pocion a false.*/
                        this.mika.cambiarAlto(70);
                        this.mika.cambiarAncho(40);
                        this.mika.cambiarEstado(false);
                        this.enemigos[i] = null;
//Ademas, el titan muere y se actualiza el puntaje y contador de enemigos
eliminados.
                        this.puntos = this.puntos + 10;
                        this.kyojinesEliminados++;
                  }
                  else if(colisionMikasaTitanGeneral &&
this.mika.getEstado() == false && this.tiempoDeInvulnerabilidad == 0) {
//Comprobacion colision sin pocion activa
                        this.vidas--; //Le quita una vida.
                        this.tiempoDeInvulnerabilidad = 100; // Otorga un
margen de invulnerabilidad para permitirle a mikasa escapar del enemigo
(100 ticks).
                  }
            }
      }



      //Proyectil
```

```
	if(this.entorno.sePresiono(entorno.TECLA_ESPACIO) && this.proyectil
== null) { //Si se presiono espacio y no hay proyectil, se le asigna a
this.proyectil el creado por la clase de Mikasa.
		this.proyectil = this.mika.disparar();
	}

	if(this.proyectil != null) { //Mientras no sea null (haya
proyectil), este es dibujado y se mueve segun su metodo propio.
		this.proyectil.dibujar(entorno);
		this.proyectil.mover();
	}

	for (int i = 0; i < obstaculos.length; i++) { {
			if(this.deteccionProyectilErrado(i)) { //Si el proyectil
fallo y choco con un objeto u obstaculo, se elimina.
				this.proyectil = null;
			}
		}
	}

	if(this.proyectil != null) { //Dibuja a Mikasa disparando (mientras
el proyectil este activo) segun su estado.
		if(this.mika.getEstado() == false)
		{
			this.mika.dibujarDisparando(entorno);
		}
		else {
			this.mika.dibujarDisparandoKyojin(entorno);
		}
	}

	//Sueros

	for (int i = 0; i < this.sueros.length; i++) {
		if(this.sueros[i] != null) { //Mientras que el suero no sea
nulo, lo dibuja y chequea si Mikasa lo toco.
			this.sueros[i].dibujar(entorno);
			if(this.tocoSuero(i)) {
				this.mika.cambiarEstado(true);
				this.sueros[i] = null;
			}
		}
	}

	boolean seCreoEnTick = false; //Booleano para saber si se creo un
suero en este tick.
```

```
	for (int i = 0; i < this.sueros.length; i++) {
		if(this.sueros[i] == null && seCreoEnTick == false &&
this.contadorDeTicks % 500 == 0) {
			seCreoEnTick = true;
			int derOIzq = this.randomPos();
			int supOInf = this.randomPos();
			int xSuero;
			int ySuero;
			if(derOIzq == 0) {
				xSuero = this.darXDer();
			}
			else {
				xSuero = this.darXIzq();
			}
			if(supOInf == 0) {
				ySuero = this.darYSup();
			}
			else {
				ySuero = this.darYInf();
			}
			this.sueros[i] = new Suero(xSuero,ySuero,20,35);
			if(this.colisionSueroKyojin(i) ||
this.colisionSueroObstaculo(i) || this.tocoSuero(i)) { //Si se detecta
colision (superpuesto con algun objeto) al crearlo, se borra y se crea
otro.
				this.sueros[i] = null;
				seCreoEnTick = false;
				i--;
			}
		}
	}


	//Enemigos
	for (int i = 0; i < enemigos.length; i++) {
		if(this.enemigos[i] != null) { //Si el enemigo no es nulo, lo
dibuja.

			if(this.mika.getX() > this.enemigos[i].getX()) // Si
Mikasa esta mas hacia la derecha (x mayor que el x del titan), dibuja al
kyojin mirando hacia la derecha.
			{
				this.enemigos[i].dibujarDerecha(entorno);
			}
```

```
                  else
               // Si el X de Mikasa es menor que el X del titan, lo dibuja
mirando hacia la izquierda.
                  {
                        this.enemigos[i].dibujarIzquierda(entorno);
                  }

                  if (enemigos[i] != null && this.proyectil != null &&
this.impactoProyectilEnemigo(i)) { // Si el Kyojin es impactado por un
proyectil, es borrado igual que el proyectil.
                        this.enemigos[i] = null;
                        this.proyectil = null;
                        this.puntos = this.puntos + 10;
                        this.kyojinesEliminados++;
                  }
            }
      }

      int contKyojinesCreadosTick = 0; //Booleano que cuenta kyojines
creados en el tick
      for (int i = 0; i < this.enemigos.length; i++) { //Crea Kyojines
cada cierto tiempo y si no se crearon mas de 2 en el tick
            if(this.enemigos[i] == null && this.contadorDeTicks % 200 == 0
&& contKyojinesCreadosTick < 2) {
                  contKyojinesCreadosTick++;
                  int derOIzq = this.randomPos();
                  int supOInf = this.randomPos();
                  int yKyojin;
                  int xKyojin;

                  if(derOIzq == 0) {
                        xKyojin = this.darXDer();
                  }
                  else {
                        xKyojin = this.darXIzq();
                  }
                  if(supOInf == 0) {
                        yKyojin = this.darYSup();
                  }
                  else {
                        yKyojin = this.darYInf();
                  }

                  this.enemigos[i] = new Kyojin(xKyojin, yKyojin, 60,
140); //Recibe un X y un Y random para darle una posicion al Kyojin, los
valores ancho y alto son fijos.
```

```
                  boolean colisionMikasaTitanGeneral =
this.colisionMikasaKyojinAbajo(i) || this.colisionMikasaKyojinArriba(i) ||
this.colisionMikasaKyojinDer(i) || this.colisionMikasaKyojinIzq(i);
                  boolean colisionKyojinObstaculoGeneral =
this.colisionKyojinObstaculoAbajo(i) ||
this.colisionKyojinObstaculoArriba(i) ||
this.colisionKyojinObstaculoDer(i) || this.colisionKyojinObstaculoIzq(i);

                  if(colisionKyojinObstaculoGeneral ||
colisionMikasaTitanGeneral || this.colisionKyojinKyojinGeneral(i)) { //Si
se detecta colision al crearlo, se borra para crear otro.
                        this.enemigos[i] = null;
                        i--;
                        contKyojinesCreadosTick--;
                  }
            }
      }

      //Movimiento del Kyojin
      //Evitar que el Kyojin se salga de los margenes
      for (int i = 0; i < enemigos.length; i++) {
            if(this.enemigos[i] != null) {
                  boolean chocaBordeDer = this.enemigos[i].getX() +
this.enemigos[i].getAncho()/2 > 800;
                  boolean chocaBordeIzq = this.enemigos[i].getX() -
this.enemigos[i].getAncho()/2 < 0;
                  boolean chocaBordeArriba = this.enemigos[i].getY() -
this.enemigos[i].getAlto()/2 < 0;
                  boolean chocaBordeAbajo = this.enemigos[i].getY() +
this.enemigos[i].getAlto()/2 > 600;

                  if(chocaBordeDer) {
                        this.enemigos[i].moverIzqChoque();
                  }

                  if(chocaBordeIzq) {
                        this.enemigos[i].moverDerChoque();
                  }

                  if(chocaBordeArriba) {
                        this.enemigos[i].moverAbajoChoque();
                  }

                  if(chocaBordeAbajo) {
                        this.enemigos[i].moverArribaChoque();
                  }
            }
```

```
	}


	//Cuando choca con otro titan.
	for (int i = 0; i < this.enemigos.length; i++) { //Comparo un Kyojin
		if(this.enemigos[i] != null) {
			for (int j = 0; j < this.enemigos.length; j++) { //Con
todos los demas. (Menos el mismo, se corrobora en el metodo).
				boolean colisionKyojinGeneral =
(this.colisionKyojinKyojinDer(i,j) || this.colisionKyojinKyojinIzq(i,j) ||
this.colisionKyojinKyojinArriba(i,j) ||
this.colisionKyojinKyojinArriba(i,j));
				if(colisionKyojinGeneral) {  //Si hay colision
entre Kyojines, se corrobora cual ocurrio, y se mueve a los kyojines en
consecuencia.
					if(colisionKyojinKyojinDer(i,j) &&
!colisionKyojinObstaculoIzq(i) && !colisionKyojinObstaculoDer(j)) {
						this.enemigos[i].moverIzqChoque();
						this.enemigos[j].moverDerChoque();
					}
					else {
						this.enemigos[i].moverArribaChoque();
						this.enemigos[j].moverAbajoChoque();
					}
					if(colisionKyojinKyojinIzq(i,j) &&
!colisionKyojinObstaculoDer(i) && !colisionKyojinObstaculoIzq(j)) {
						this.enemigos[i].moverDerChoque();
						this.enemigos[j].moverIzqChoque();
					}
					else {
						this.enemigos[i].moverAbajoChoque();
						this.enemigos[j].moverArribaChoque();
					}
					if(colisionKyojinKyojinArriba(i,j) &&
!colisionKyojinObstaculoAbajo(i) && !colisionKyojinObstaculoAbajo(j)) {
						this.enemigos[i].moverAbajoChoque();
						this.enemigos[j].moverArribaChoque();
					}
					else {
						this.enemigos[i].moverDerChoque();
						this.enemigos[i].moverIzqChoque();
					}
					if(colisionKyojinKyojinAbajo(i,j) &&
!colisionKyojinObstaculoArriba(i) && !colisionKyojinObstaculoAbajo(j)) {
						this.enemigos[i].moverArribaChoque();
						this.enemigos[j].moverAbajoChoque();
					}
```

```
                              else {
                                    this.enemigos[i].moverAbajoChoque();
                                    this.enemigos[j].moverArribaChoque();
                              }
                        }
                  }
            }
      }

      //Movimiento sin colisiones y con colisiones con obstaculos.

      for (int i = 0; i < enemigos.length; i++) { //Mover kyojines. //El
que estoy viendo
            if (this.enemigos[i]!=null) { // Si no es nulo, lo deja
moverse.
                  boolean colisionGeneralObstaculo =
this.colisionKyojinObstaculoAbajo(i) ||
this.colisionKyojinObstaculoArriba(i) ||
this.colisionKyojinObstaculoDer(i) || this.colisionKyojinObstaculoIzq(i);
                  boolean puedeMoverDer =
(!this.colisionKyojinObstaculoDer(i) && this.enemigos[i].getX() +
this.enemigos[i].getAncho()/2 < 800 && !(colisionGeneralObstaculo &&
this.enemigos[i].getY() > this.mika.getY()) && this.enemigos[i].getX() <
this.mika.getX());
                  boolean puedeMoverIzq =
(!this.colisionKyojinObstaculoIzq(i) && this.enemigos[i].getX() -
this.enemigos[i].getAncho()/2 >0 && !(colisionGeneralObstaculo &&
this.enemigos[i].getY() < this.mika.getY()) && this.enemigos[i].getX() >
this.mika.getX());
                  boolean puedeMoverArriba =
(!this.colisionKyojinObstaculoArriba(i) && this.enemigos[i].getY() -
this.enemigos[i].getAlto()/2 > 0 && !(colisionGeneralObstaculo &&
this.enemigos[i].getX() > this.mika.getX()) && this.enemigos[i].getY() >
this.mika.getY());
                  boolean puedeMoverAbajo =
(!this.colisionKyojinObstaculoAbajo(i) && this.enemigos[i].getY() +
this.enemigos[i].getAlto()/2 < 600 && !(colisionGeneralObstaculo &&
this.enemigos[i].getX() < this.mika.getX()) && this.enemigos[i].getY() <
this.mika.getY());


                  boolean colisionaObstaculoAbajo =
this.colisionKyojinObstaculoAbajo(i) && this.enemigos[i].getY() <
this.mika.getY();
                  boolean colisionaObstaculoArriba =
this.colisionKyojinObstaculoArriba(i) && this.enemigos[i].getY() >
this.mika.getY();
```

```
                boolean colisionaObstaculoDer =
this.colisionKyojinObstaculoDer(i) && this.enemigos[i].getX() <
this.mika.getX();
                boolean colisionaObstaculoIzq =
this.colisionKyojinObstaculoIzq(i) && this.enemigos[i].getX() >
this.mika.getX();


                //Sin colisiones

                if(puedeMoverDer) {
                      this.enemigos[i].moverDer(this.mika.getX());
                }
                if(puedeMoverIzq) {
                      this.enemigos[i].moverIzq(this.mika.getX());
                }
                if(puedeMoverArriba) {
                      this.enemigos[i].moverArriba(this.mika.getY());
                }
                if(puedeMoverAbajo) {
                      this.enemigos[i].moverAbajo(this.mika.getY());
                }


                //Colision con obstaculos

                if(colisionaObstaculoAbajo) {
                      this.enemigos[i].moverIzqChoque();
                }
                if(colisionaObstaculoArriba) {
                      this.enemigos[i].moverDerChoque();
                }
                if(colisionaObstaculoDer) {
                      this.enemigos[i].moverAbajoChoque();
                }
                if(colisionaObstaculoIzq) {
                      this.enemigos[i].moverArribaChoque();
                }
          }
     }



     //Obstaculos
     for (int i = 0; i < obstaculos.length; i++) {//Mientras que el
obstaculo no sea nulo, lo dibuja.
                this.obstaculos[i].dibujar(entorno);
```

```
		}

			}
		}
}


public int darXIzq() {
		Random x = new Random();
		int posX = x.nextInt(370-80) + 80; //Numero aleatorio en X desde 80
y 370
		return posX;
}
public int darXDer() {
		Random x = new Random();
		int posX = x.nextInt(720-430) + 430; //Numero aleatorio en X desde
430 y 720
		return posX;
}
public int darYSup() {
		Random y = new Random();
		int posY = y.nextInt(260-100) + 100; //Numero aleatorio entre 100 y
260
		return posY;
}
public int darYInf() {
		Random y = new Random();
		int posY = y.nextInt(470-340) + 340 ; //Numero aleatorio entre 340 y
470
		return posY;
}

public int randomPos() { //Da un valor aleatorio (0 o 1)
		Random x = new Random();
		int valorRandom = x.nextInt(2);
		return valorRandom;
}

public boolean terminoJuego() { //Determina si finalizo el juego.
		if(seGano() || sePerdio())
			return true;
		return false;
}

public boolean seGano() { //Determina si se gano.
		for (int i = 0; i < this.enemigos.length; i++) {
			if(this.enemigos[i] != null) {
```

```
                  return false;
            }
      }
      return true;
}
public boolean sePerdio() { //Determina si se perdio.
      if(this.vidas == 0) {
            this.mika = null;
            return true;
      }
      return false;
}

// Colisiones mikasa y kyojines.
public boolean colisionMikasaKyojinDer(int indice) {
      int xLadoDer = this.mika.getXLadoDer(); //Lados necesarios para
detectar colision de Mikasa.
      int yLadoArriba = this.mika.getYLadoArriba();

      int xLadoDerKyojin = this.enemigos[indice].getXLadoDer();
//Definicion de los lados de los Kyojines
      int xLadoIzqKyojin = this.enemigos[indice].getXLadoIzq();
      int yLadoArribaKyojin = this.enemigos[indice].getYLadoArriba();
      int yLadoAbajoKyojin = this.enemigos[indice].getYLadoAbajo();

      return(xLadoDer > xLadoIzqKyojin && xLadoDer < xLadoDerKyojin &&
yLadoArriba < yLadoAbajoKyojin && yLadoArriba > yLadoArribaKyojin);
}

public boolean colisionMikasaKyojinIzq(int indice) {
      int xLadoIzq = this.mika.getXLadoIzq(); //Lados necesarios para
detectar colision de Mikasa.
      int yLadoArriba = this.mika.getYLadoArriba();

      int xLadoDerKyojin = this.enemigos[indice].getXLadoDer();
//Definicion de los lados de los Kyojines
      int xLadoIzqKyojin = this.enemigos[indice].getXLadoIzq();
      int yLadoArribaKyojin = this.enemigos[indice].getYLadoArriba();
      int yLadoAbajoKyojin = this.enemigos[indice].getYLadoAbajo();

      return(xLadoIzq < xLadoDerKyojin && xLadoIzq > xLadoIzqKyojin &&
yLadoArriba < yLadoAbajoKyojin && yLadoArriba > yLadoArribaKyojin);
      }

public boolean colisionMikasaKyojinArriba(int indice) {
      int xLadoDer = this.mika.getXLadoDer(); //Lados necesarios para
detectar colision de Mikasa.
```

```
      int yLadoAbajo = this.mika.getYLadoAbajo();

      int xLadoDerKyojin = this.enemigos[indice].getXLadoDer();
//Definicion de los lados de los Kyojines
      int xLadoIzqKyojin = this.enemigos[indice].getXLadoIzq();
      int yLadoArribaKyojin = this.enemigos[indice].getYLadoArriba();
      int yLadoAbajoKyojin = this.enemigos[indice].getYLadoAbajo();

      return(xLadoDer > xLadoIzqKyojin && xLadoDer < xLadoDerKyojin &&
yLadoAbajo > yLadoArribaKyojin && yLadoAbajo < yLadoAbajoKyojin);
}

public boolean colisionMikasaKyojinAbajo(int indice) {
      int xLadoIzq = this.mika.getXLadoIzq(); //Lados necesarios para
detectar colision de Mikasa.
      int yLadoAbajo = this.mika.getYLadoAbajo();

      int xLadoDerKyojin = this.enemigos[indice].getXLadoDer();
//Definicion de los lados de los Kyojines
      int xLadoIzqKyojin = this.enemigos[indice].getXLadoIzq();
      int yLadoArribaKyojin = this.enemigos[indice].getYLadoArriba();
      int yLadoAbajoKyojin = this.enemigos[indice].getYLadoAbajo();

      return(xLadoIzq < xLadoDerKyojin && xLadoIzq > xLadoIzqKyojin &&
yLadoAbajo > yLadoArribaKyojin && yLadoAbajo < yLadoAbajoKyojin);
      }

//Colisiones de mikasa y obstaculos
public boolean colisionMikasaObstaculoDer() {
      for (int i = 0; i < obstaculos.length; i++) {
      int xLadoDer = this.mika.getXLadoDer();
//Lados necesarios para detectar colision de Mikasa.
            int yLadoArriba = this.mika.getYLadoArriba();
            int yLadoAbajo = this.mika.getYLadoAbajo();

            int xLadoDerObstaculo = this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
            int xLadoIzqObstaculo = this.obstaculos[i].getXLadoIzq();
            int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
            int yLadoAbajoObstaculo = this.obstaculos[i].getYLadoAbajo();

            if(xLadoDer+4 >= xLadoIzqObstaculo && xLadoDer <=
xLadoDerObstaculo && ((yLadoArriba >= yLadoAbajoObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo) || (yLadoArriba <= yLadoAbajoObstaculo && yLadoAbajo
>= yLadoArribaObstaculo))) {
                  return true;
```

```
            }
      }
      return false;
}

public boolean colisionMikasaObstaculoIzq() {
      for (int i = 0; i < obstaculos.length; i++) {
            int xLadoIzq = this.mika.getXLadoIzq();
//Lados necesarios para detectar colision de Mikasa.
            int yLadoArriba = this.mika.getYLadoArriba();
            int yLadoAbajo = this.mika.getYLadoAbajo();

            int xLadoDerObstaculo = this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
            int xLadoIzqObstaculo = this.obstaculos[i].getXLadoIzq();
            int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
            int yLadoAbajoObstaculo = this.obstaculos[i].getYLadoAbajo();

            if(xLadoIzq-4 <= xLadoDerObstaculo && xLadoIzq >=
xLadoIzqObstaculo && ((yLadoArriba >= yLadoAbajoObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo) || (yLadoArriba <= yLadoAbajoObstaculo && yLadoAbajo
>= yLadoArribaObstaculo))) {
                  return true;
            }
      }
      return false;
}
public boolean colisionMikasaObstaculoArriba() {
      for (int i = 0; i < obstaculos.length; i++) {
            int xLadoDer = this.mika.getXLadoDer();
//Lados necesarios para detectar colision de Mikasa.
            int xLadoIzq = this.mika.getXLadoIzq();
            int yLadoArriba = this.mika.getYLadoArriba();
            int xLadoDerObstaculo = this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
            int xLadoIzqObstaculo = this.obstaculos[i].getXLadoIzq();
            int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
            int yLadoAbajoObstaculo = this.obstaculos[i].getYLadoAbajo();

            if(yLadoArriba-4 <= yLadoAbajoObstaculo && yLadoArriba >=
yLadoArribaObstaculo && ((xLadoDer >= xLadoIzqObstaculo && xLadoIzq <=
xLadoDerObstaculo) || (xLadoIzq <= xLadoDerObstaculo && xLadoDer >=
xLadoIzqObstaculo))) {
                  return true;
            }
```

```
	}
	return false;
}

public boolean colisionMikasaObstaculoAbajo() {
	for (int i = 0; i < obstaculos.length; i++) {
			int xLadoDer = this.mika.getXLadoDer();
//Lados necesarios para detectar colision de Mikasa.
			int xLadoIzq = this.mika.getXLadoIzq();
			int yLadoAbajo = this.mika.getYLadoAbajo();

			int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
			int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
			int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
			int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

			if(yLadoAbajo+4 >= yLadoArribaObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo && ((xLadoDer >= xLadoIzqObstaculo && xLadoIzq <=
xLadoDerObstaculo) || (xLadoIzq <= xLadoDerObstaculo && xLadoDer >=
xLadoIzqObstaculo))) {
				return true;
			}
		}
	return false;
}

//Colisiones Kyojines con obstaculos
public boolean colisionKyojinObstaculoDer(int indice) {
	for (int i = 0; i < obstaculos.length; i++) {
		if (this.enemigos[indice] == null) {
			return false;
		}
		else {
			int xLadoDer = this.enemigos[indice].getXLadoDer();
//Definicion de los lados necesarios del Kyojin
			int yLadoArriba =
this.enemigos[indice].getYLadoArriba();
			int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();

			int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
```

```
                  int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
                  int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
                  int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

                  if(xLadoDer+4 > xLadoIzqObstaculo && xLadoDer <
xLadoDerObstaculo && ((yLadoArriba > yLadoAbajoObstaculo && yLadoAbajo <
yLadoAbajoObstaculo) || (yLadoArriba < yLadoAbajoObstaculo && yLadoAbajo >
yLadoArribaObstaculo))) {
                        return true;
                  }
            }
      }
      return false;
}

public boolean colisionKyojinObstaculoIzq(int indice) {
      for (int i = 0; i < obstaculos.length; i++) {
            if (this.enemigos[indice] == null) {
                  return false;
                  }
            else {
                  int xLadoIzq = this.enemigos[indice].getXLadoIzq();
//Definicion de los lados necesarios del Kyojin
                  int yLadoArriba =
this.enemigos[indice].getYLadoArriba();
                  int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();

                  int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
                  int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
                  int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
                  int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

                  if(xLadoIzq-4 < xLadoDerObstaculo && xLadoIzq >
xLadoIzqObstaculo && ((yLadoArriba > yLadoAbajoObstaculo && yLadoAbajo <
yLadoAbajoObstaculo) || (yLadoArriba < yLadoAbajoObstaculo && yLadoAbajo >
yLadoArribaObstaculo))) {
                        return true;
                        }
                  }
```

```
            }
      return false;
}

public boolean colisionKyojinObstaculoArriba(int indice) {
      for (int i = 0; i < obstaculos.length; i++) {
            if (this.enemigos[indice] == null) {
                  return false;
            }
            else {
                  int xLadoDer = this.enemigos[indice].getXLadoDer();
//Definicion de los lados necesarios del Kyojin
                  int xLadoIzq = this.enemigos[indice].getXLadoIzq();
                  int yLadoArriba =
this.enemigos[indice].getYLadoArriba();

                  int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
                  int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
                  int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
                  int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

                  if(yLadoArriba-4 < yLadoAbajoObstaculo && yLadoArriba >
yLadoArribaObstaculo && ((xLadoDer > xLadoIzqObstaculo && xLadoIzq <
xLadoDerObstaculo) || (xLadoIzq < xLadoDerObstaculo && xLadoDer >
xLadoIzqObstaculo))) {
                        return true;
                  }
            }
      }
      return false;
}
public boolean colisionKyojinObstaculoAbajo(int indice) {
      for (int i = 0; i < obstaculos.length; i++) {
            if (this.enemigos[indice] == null) {
                        return false;
                  }
            else {
                  int xLadoDer = this.enemigos[indice].getXLadoDer();
//Definicion de los lados necesarios del Kyojin
                  int xLadoIzq = this.enemigos[indice].getXLadoIzq();
                  int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();
```

```
                  int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
//Definicion de los lados de los obstaculos
                  int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
                  int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
                  int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

                  if(yLadoAbajo+4 > yLadoArribaObstaculo && yLadoAbajo <
yLadoAbajoObstaculo && ((xLadoDer > xLadoIzqObstaculo && xLadoIzq <
xLadoDerObstaculo) || (xLadoIzq < xLadoDerObstaculo && xLadoDer >
xLadoIzqObstaculo))) {
                  return true;
                  }
            }
      }
      return false;
}
//Colisiones entre kyojines
      public boolean colisionKyojinKyojinDer(int indice, int
indiceSegundo) {
      if (indiceSegundo == indice || this.enemigos[indiceSegundo] == null)
{
            return false;
            }
      else {
            int xLadoDer = this.enemigos[indice].getXLadoDer();
//Definicion de los lados necesarios del primer Kyojin
            int yLadoArriba = this.enemigos[indice].getYLadoArriba();
            int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();

            int xLadoDerSegundo =
this.enemigos[indiceSegundo].getXLadoDer(); //Definicion de los lados
necesarios del segundo Kyojin
            int xLadoIzqSegundo =
this.enemigos[indiceSegundo].getXLadoIzq();
            int yLadoArribaSegundo =
this.enemigos[indiceSegundo].getYLadoArriba();
            int yLadoAbajoSegundo =
this.enemigos[indiceSegundo].getYLadoAbajo();

            return(xLadoDer+4 > xLadoIzqSegundo && xLadoDer <
xLadoDerSegundo && ((yLadoArriba > yLadoAbajoSegundo && yLadoAbajo <
yLadoAbajoSegundo) || (yLadoArriba < yLadoAbajoSegundo && yLadoAbajo >
yLadoArribaSegundo)));
```

```
            }
      }

      public boolean colisionKyojinKyojinIzq(int indice, int
indiceSegundo) {
      if (indiceSegundo == indice || this.enemigos[indiceSegundo] == null)
{
            return false;
      }
      else {
            int xLadoIzq = this.enemigos[indice].getXLadoIzq();
//Definicion de los lados necesarios del primer Kyojin
            int yLadoArriba = this.enemigos[indice].getYLadoArriba();
            int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();

            int xLadoDerSegundo =
this.enemigos[indiceSegundo].getXLadoDer();
//Definicion de los lados necesarios del segundo Kyojin
            int xLadoIzqSegundo =
this.enemigos[indiceSegundo].getXLadoIzq();
            int yLadoArribaSegundo =
this.enemigos[indiceSegundo].getYLadoArriba();
            int yLadoAbajoSegundo =
this.enemigos[indiceSegundo].getYLadoAbajo();

            return(xLadoIzq-4 < xLadoDerSegundo && xLadoIzq >
xLadoIzqSegundo && ((yLadoArriba > yLadoAbajoSegundo && yLadoAbajo <
yLadoAbajoSegundo) || (yLadoArriba < yLadoAbajoSegundo && yLadoAbajo >
yLadoArribaSegundo)));
      }
}

public boolean colisionKyojinKyojinArriba(int indice, int indiceSegundo) {
      if (indiceSegundo == indice || this.enemigos[indiceSegundo] == null)
{
            return false;
            }
      else {
            int xLadoDer = this.enemigos[indice].getXLadoDer();
 //Definicion de los lados necesarios del primer Kyojin
            int xLadoIzq = this.enemigos[indice].getXLadoIzq();
            int yLadoArriba = this.enemigos[indice].getYLadoArriba();

            int xLadoDerSegundo =
this.enemigos[indiceSegundo].getXLadoDer();
//Definicion de los lados necesarios del segundo Kyojin
```

```
            int xLadoIzqSegundo =
this.enemigos[indiceSegundo].getXLadoIzq();
            int yLadoArribaSegundo =
this.enemigos[indiceSegundo].getYLadoArriba();
            int yLadoAbajoSegundo =
this.enemigos[indiceSegundo].getYLadoAbajo();

            return(yLadoArriba-4 < yLadoAbajoSegundo && yLadoArriba >
yLadoArribaSegundo && ((xLadoDer > xLadoIzqSegundo && xLadoIzq <
xLadoDerSegundo) || (xLadoIzq < xLadoDerSegundo && xLadoDer >
xLadoIzqSegundo)));
            }
      }

public boolean colisionKyojinKyojinAbajo(int indice, int indiceSegundo) {
      if (indiceSegundo == indice || this.enemigos[indiceSegundo] == null)
{
            return false;
      }
      else {
            int xLadoDer = this.enemigos[indice].getXLadoDer();
//Definicion de los lados necesarios del primer Kyojin
            int xLadoIzq = this.enemigos[indice].getXLadoIzq();
            int yLadoAbajo = this.enemigos[indice].getYLadoAbajo();

            int xLadoDerSegundo =
this.enemigos[indiceSegundo].getXLadoDer(); //Definicion de los lados
necesarios del segundo Kyojin
            int xLadoIzqSegundo =
this.enemigos[indiceSegundo].getXLadoIzq();
            int yLadoArribaSegundo =
this.enemigos[indiceSegundo].getYLadoArriba();
            int yLadoAbajoSegundo =
this.enemigos[indiceSegundo].getYLadoAbajo();

            return(yLadoAbajo+4 > yLadoArribaSegundo && yLadoAbajo <
yLadoAbajoSegundo && ((xLadoDer > xLadoIzqSegundo && xLadoIzq <
xLadoDerSegundo) || (xLadoIzq < xLadoDerSegundo && xLadoDer >
yLadoAbajoSegundo)));
            }
      }

      public boolean colisionKyojinKyojinGeneral(int indice) {
//Agrupacion de las funciones especificas de colision de kyojines con
kyojines solo con un indice.
      for (int i = 0; i < this.enemigos.length; i++) {
```

```
            if(colisionKyojinKyojinDer(indice,i) ||
colisionKyojinKyojinIzq(indice,i) || colisionKyojinKyojinArriba(indice,i)
|| colisionKyojinKyojinAbajo(indice,i)) {
                  return true;
                  }
            }
            return false;
      }

      public boolean impactoProyectilEnemigo(int indice) { // La punta del
proyectil (depende la direccion/sentido) solo impacta con la cara visible
del enemigo, es decir, el lado contrario al sentido del proyectil

      int bordeProyectil = proyectil.getBordeProyectil();
      int xProyectil = proyectil.getX();
      int yProyectil = proyectil.getY();

      int xLadoDerKyojin = this.enemigos[indice].getXLadoDer();
      int xLadoIzqKyojin = this.enemigos[indice].getXLadoIzq();
      int yLadoArribaKyojin = this.enemigos[indice].getYLadoArriba();
      int yLadoAbajoKyojin = this.enemigos[indice].getYLadoAbajo();

            if(proyectil.getSentido() == "arriba") { //Corrobora impacto
proyectil con enemigo segun el sentido del mismo.
            return(bordeProyectil < yLadoAbajoKyojin && bordeProyectil >
yLadoArribaKyojin && xProyectil > xLadoIzqKyojin && xProyectil <
xLadoDerKyojin);
            }

      if(proyectil.getSentido() == "abajo") {
            return(bordeProyectil > yLadoArribaKyojin && bordeProyectil <
yLadoAbajoKyojin && xProyectil > xLadoIzqKyojin && xProyectil <
xLadoDerKyojin);
            }

      if(proyectil.getSentido() == "derecha") {
            return(bordeProyectil > xLadoIzqKyojin && bordeProyectil <
xLadoDerKyojin && yProyectil > yLadoArribaKyojin && yProyectil <
yLadoAbajoKyojin);
            }

      if(proyectil.getSentido() == "izquierda") {
            return(bordeProyectil < xLadoDerKyojin && bordeProyectil >
xLadoIzqKyojin && yProyectil > yLadoArribaKyojin && yProyectil <
yLadoAbajoKyojin);
            }
            return false;
```

```
	}

	public boolean impactoProyectilObstaculo(int indice) { // La punta
del proyectil (depende la direccion/sentido) solo impacta con la cara
visible del obstaculo, es decir, el lado contrario al sentido del
proyectil

		int bordeProyectil = proyectil.getBordeProyectil();
		int xProyectil = proyectil.getX();
		int yProyectil = proyectil.getY();

		int xLadoDerObstaculo = this.obstaculos[indice].getXLadoDer();
		int xLadoIzqObstaculo = this.obstaculos[indice].getXLadoIzq();
		int yLadoArribaObstaculo =
this.obstaculos[indice].getYLadoArriba();
		int yLadoAbajoObstaculo =
this.obstaculos[indice].getYLadoAbajo();

	if(proyectil.getSentido() == "arriba") {
			return(bordeProyectil < yLadoAbajoObstaculo &&
bordeProyectil > yLadoArribaObstaculo && xProyectil > xLadoIzqObstaculo &&
xProyectil < xLadoDerObstaculo);
			}

	if(proyectil.getSentido() == "abajo") {
			return(bordeProyectil > yLadoArribaObstaculo &&
bordeProyectil < yLadoAbajoObstaculo && xProyectil > xLadoIzqObstaculo &&
xProyectil < xLadoDerObstaculo);
			}

	if(proyectil.getSentido() == "derecha") {
			return(bordeProyectil > xLadoIzqObstaculo &&
bordeProyectil < xLadoDerObstaculo && yProyectil > yLadoArribaObstaculo &&
yProyectil < yLadoAbajoObstaculo);
			}

		if(proyectil.getSentido() == "izquierda"){
			return(bordeProyectil < xLadoDerObstaculo &&
bordeProyectil > xLadoIzqObstaculo && yProyectil > yLadoArribaObstaculo &&
yProyectil < yLadoAbajoObstaculo);
			}

		return false;
	}

public boolean tocoSuero(int indice) {
	for (int i = 0; i < sueros.length; i++) {
```

```
            int xLadoDer = this.mika.getXLadoDer();
            int xLadoIzq = this.mika.getXLadoIzq();
            int yLadoArriba = this.mika.getYLadoArriba();
            int yLadoAbajo = this.mika.getYLadoAbajo();

            int xLadoDerSuero = this.sueros[indice].getXLadoDer();
            int xLadoIzqSuero = this.sueros[indice].getXLadoIzq();
            int yLadoArribaSuero = this.sueros[indice].getYLadoArriba();
            int yLadoAbajoSuero = this.sueros[indice].getYLadoAbajo();

            if(xLadoIzq <= xLadoDerSuero && xLadoIzq >= xLadoIzqSuero &&
((yLadoArriba >= yLadoAbajoSuero && yLadoAbajo <= yLadoAbajoSuero) ||
(yLadoArriba <= yLadoAbajoSuero && yLadoAbajo >= yLadoArribaSuero))) {
                  return true;
            }
                  if(xLadoDer >= xLadoIzqSuero && xLadoDer <=
xLadoDerSuero && ((yLadoArriba >= yLadoAbajoSuero && yLadoAbajo <=
yLadoAbajoSuero) || (yLadoArriba <= yLadoAbajoSuero && yLadoAbajo >=
yLadoArribaSuero))) {
                  return true;
            }
                  if(yLadoArriba <= yLadoAbajoSuero && yLadoArriba >=
yLadoArribaSuero && ((xLadoDer >= xLadoIzqSuero && xLadoIzq <=
xLadoDerSuero) || (xLadoIzq <= xLadoDerSuero && xLadoDer >=
xLadoIzqSuero))) {
                  return true;
            }
                  if(yLadoAbajo >= yLadoArribaSuero && yLadoAbajo <=
yLadoAbajoSuero && ((xLadoDer >= xLadoIzqSuero && xLadoIzq <=
xLadoDerSuero) || (xLadoIzq <= xLadoDerSuero && xLadoDer >=
xLadoIzqSuero))) {
                  return true;
            }
      }
      return false;
      }

public boolean colisionSueroObstaculo(int indice) {
      for (int i = 0; i < this.obstaculos.length; i++) {
                  int xLadoDer = this.sueros[indice].getXLadoDer();
                  int xLadoIzq = this.sueros[indice].getXLadoIzq();
                  int yLadoArriba = this.sueros[indice].getYLadoArriba();
                  int yLadoAbajo = this.sueros[indice].getYLadoAbajo();

                  int xLadoDerObstaculo =
this.obstaculos[i].getXLadoDer();
```

```
                  int xLadoIzqObstaculo =
this.obstaculos[i].getXLadoIzq();
                  int yLadoArribaObstaculo =
this.obstaculos[i].getYLadoArriba();
                  int yLadoAbajoObstaculo =
this.obstaculos[i].getYLadoAbajo();

                  if(xLadoIzq <= xLadoDerObstaculo && xLadoIzq >=
xLadoIzqObstaculo && ((yLadoArriba >= yLadoAbajoObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo) || (yLadoArriba <= yLadoAbajoObstaculo && yLadoAbajo
>= yLadoArribaObstaculo))) {
                        return true;
                  }
                  if(xLadoDer >= xLadoIzqObstaculo && xLadoDer <=
xLadoDerObstaculo && ((yLadoArriba >= yLadoAbajoObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo) || (yLadoArriba <= yLadoAbajoObstaculo && yLadoAbajo
>= yLadoArribaObstaculo))) {
                        return true;
                  }
            if(yLadoArriba <= yLadoAbajoObstaculo && yLadoArriba >=
yLadoArribaObstaculo && ((xLadoDer >= xLadoIzqObstaculo && xLadoIzq <=
xLadoDerObstaculo) || (xLadoIzq <= xLadoDerObstaculo && xLadoDer >=
xLadoIzqObstaculo))) {
                        return true;
                  }
            if(yLadoAbajo >= yLadoArribaObstaculo && yLadoAbajo <=
yLadoAbajoObstaculo && ((xLadoDer >= xLadoIzqObstaculo && xLadoIzq <=
xLadoDerObstaculo) || (xLadoIzq <= xLadoDerObstaculo && xLadoDer >=
xLadoIzqObstaculo))) {
                        return true;
                  }
            }
      return false;
}


      public boolean colisionSueroKyojin(int indice) {
            for (int i = 0; i < this.enemigos.length; i++) {
                  if(this.enemigos[i]==null) {
                        return false;
            }
      else {

                  int xLadoDer = this.sueros[indice].getXLadoDer();
                  int xLadoIzq = this.sueros[indice].getXLadoIzq();
                  int yLadoArriba = this.sueros[indice].getYLadoArriba();
                  int yLadoAbajo = this.sueros[indice].getYLadoAbajo();
```

```
                  int xLadoDerKyojin = this.enemigos[i].getXLadoDer();
                  int xLadoIzqKyojin = this.enemigos[i].getXLadoIzq();
                  int yLadoArribaKyojin =
this.enemigos[i].getYLadoArriba();
                  int yLadoAbajoKyojin = this.enemigos[i].getYLadoAbajo();

                  if(xLadoIzq <= xLadoDerKyojin && xLadoIzq >=
xLadoIzqKyojin && ((yLadoArriba >= yLadoAbajoKyojin && yLadoAbajo <=
yLadoAbajoKyojin) || (yLadoArriba <= yLadoAbajoKyojin && yLadoAbajo >=
yLadoArribaKyojin))) {
                        return true;
                  }
                  if(xLadoDer >= xLadoIzqKyojin && xLadoDer <=
xLadoDerKyojin && ((yLadoArriba >= yLadoAbajoKyojin && yLadoAbajo <=
yLadoAbajoKyojin) || (yLadoArriba <= yLadoAbajoKyojin && yLadoAbajo >=
yLadoArribaKyojin))) {
                        return true;
                  }
                  if(yLadoArriba <= yLadoAbajoKyojin && yLadoArriba >=
yLadoArribaKyojin && ((xLadoDer >= xLadoIzqKyojin && xLadoIzq <=
xLadoDerKyojin) || (xLadoIzq <= xLadoDerKyojin && xLadoDer >=
xLadoIzqKyojin))) {
                        return true;
                  }
                  if(yLadoAbajo >= yLadoArribaKyojin && yLadoAbajo <=
yLadoAbajoKyojin && ((xLadoDer >= xLadoIzqKyojin && xLadoIzq <=
xLadoDerKyojin) || (xLadoIzq <= xLadoDerKyojin && xLadoDer >=
xLadoIzqKyojin))) {
                        return true;
                  }
            }
      }
      return false;
}

      public boolean deteccionProyectilErrado(int indice) {
//Recibe un indice de un obstaculo para corroborar colisiones en el metodo
impactoProyectilObstaculo
            if(this.proyectil != null) {
                  if(this.proyectil.getSentido() == "derecha") {
// Si el proyectil toca un borde de la pantalla o se corrobora en el
metodo impactoProyectilObstaculo una colision segun la direccion del
proyectil, se devuelve true.
                  return(this.proyectil.getBordeProyectil() >=800 ||
(this.obstaculos[indice] != null &&
this.impactoProyectilObstaculo(indice)));
```

```
                }
                if(this.proyectil.getSentido() == "izquierda") {
// Si el proyectil toca un borde de la pantalla o se corrobora en el
metodo impactoProyectilObstaculo una colision segun la direccion del
proyectil, se devuelve true.
                return(this.proyectil.getBordeProyectil() <= 0 ||
(this.obstaculos[indice] != null &&
this.impactoProyectilObstaculo(indice)));
                }
                if(this.proyectil.getSentido() == "arriba") {
// Si el proyectil toca un borde de la pantalla o se corrobora en el
metodo impactoProyectilObstaculo una colision segun la direccion del
proyectil, se devuelve true.
                return(this.proyectil.getBordeProyectil() <= 0 ||
(this.obstaculos[indice] != null &&
this.impactoProyectilObstaculo(indice)));
                }
                if(this.proyectil.getSentido() == "abajo") {
// Si el proyectil toca un borde de la pantalla o se corrobora en el
metodo impactoProyectilObstaculo una colision según la dirección del
proyectil, se devuelve true.
                return(this.proyectil.getBordeProyectil() >= 600 ||
(this.obstaculos[indice] != null &&
this.impactoProyectilObstaculo(indice)));
            }
      }
      return false;
}

      @SuppressWarnings("unused")
      public static void main(String[] args)
      {
            Juego juego = new Juego();
      }
}

package juego;

import java.awt.Image;
import entorno.Entorno;
import entorno.Herramientas;


public class Kyojin {
      private int x;
      private int y;
      private int ancho;
```

```
	private int alto;


public Kyojin(int x, int y, int ancho, int alto) {
	this.x = x;
	this.y = y;
	this.ancho = ancho;
	this.alto = alto;

}

public void dibujarIzquierda(Entorno entorno) {
	Image imagen;
	imagen = Herramientas.cargarImagen("Imagenes/kyojinMovIzq.gif");
	entorno.dibujarImagen(imagen, this.x, this.y, 0, 1);
}
public void dibujarDerecha(Entorno entorno) {
	Image imagen;
	imagen = Herramientas.cargarImagen("Imagenes/kyojinMovDer.gif");
	entorno.dibujarImagen(imagen, this.x, this.y, 0, 1);
}

public void moverIzqChoque() {
//Metodos para mover al titan si se encuentra colisionando con un
obstaculo u otro titan.
		this.x = this.x - 2;
}

public void moverDerChoque() {
	this.x = this.x + 2;
}

public void moverArribaChoque() {
	this.y = this.y - 2;
}

public void moverAbajoChoque() {
	this.y = this.y + 2;
}

public void moverIzq(int x) {
//Metodos para mover al titan libremente.
	if(this.x > x) {
		this.x = this.x -1;
	}
}
```

```
public void moverDer(int x) {
	if(this.x < x) {
		this.x = this.x +1;
	}
}

public void moverArriba(int y) {
	if(this.y > y) {
		this.y = this.y-1;
	}
}

public void moverAbajo(int y) {
	if(this.y < y) {
		this.y = this.y +1;
	}
}
public int getX() {
//Devuelve la posicion X.
	return this.x;
	}

public int getY() {
//Devuelve la posicion Y.
	return this.y;
}
public int getAncho() { //Devuelve el valor del ancho.
	return this.ancho;
}
public int getAlto() { //Devuelve el valor del alto.
	return this.alto;
}

public int getXLadoDer() { //Devuelve los lados del Kyojin
	return this.x + this.ancho/2;
}
public int getXLadoIzq() {
	return this.x - this.ancho/2;
}
public int getYLadoArriba() {
	return this.y - this.alto/2;
}
public int getYLadoAbajo() {
	return this.y + this.alto/2;
}
}
package juego;
```

```
import java.awt.Image;
import entorno.Herramientas;
import entorno.Entorno;

public class Mikasa {
	private int x;
	private int y;
	private int alto;
	private int ancho;
	private String sentido;
	private boolean pocionActiva;
	private Image imagen;

public Mikasa(int x, int y, int ancho, int alto, String sentido, boolean
pocionActiva, String imagen) {
	this.x = x;
	this.y = y;
	this.ancho = ancho;
	this.alto = alto;
	this.sentido = sentido;
	this.pocionActiva = pocionActiva;
	this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaQuietaDer.png");
	}

public void dibujarMovimiento(Entorno entorno) { //Dibuja a mikasa en
movimiento (Solo derecha o izquierda)

	if(sentido == "derecha") {
		this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaDerMov.gif");
	}
	if(sentido == "izquierda") {
		this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaIzqMov.gif");
	}

	entorno.dibujarImagen(imagen, this.x, this.y, 0, 0.7);

	}
public void dibujarQuieta(Entorno entorno) { //Dibuja a Mikasa quieta

	if(sentido == "derecha") {
		this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaQuietaDer.png");
	}
	if(sentido == "izquierda") {
```

```
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaQuietaIzq.png");
      }
      if(sentido == "abajo") {
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaAbajo.png");
      }
      if(sentido == "arriba") {
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaArriba.png");
      }

      entorno.dibujarImagen(imagen, this.x, this.y, 0, 0.7);

}

public void dibujarDisparando(Entorno entorno) { //Dibuja a Mikasa
disparando
      if(this.sentido == "derecha") {
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaDisparoDer.png");
      }
      else{
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaDisparoIzq.png");
      }

      entorno.dibujarImagen(imagen, this.x, this.y, 0, 0.7);

}
public void dibujarDisparandoKyojin(Entorno entorno) { //Dibuja a Mikasa
disparando transformada
      if(this.sentido == "derecha") {
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaDisparoKyojinDer.png");
      }
      else{
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaDisparoKyojinIzq.png");
      }

      entorno.dibujarImagen(imagen, this.x, this.y, 0, 1);

}
public void dibujarTransformada(Entorno entorno) { //Dibuja a Mikasa en
movimiento (transformada)
      if(this.sentido == "izquierda") {
```

```
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaKyojinIzqMov.gif");
      }

      else {
            this.imagen =
Herramientas.cargarImagen("Imagenes/mikasaKyojinDerMov.gif");
      }

      entorno.dibujarImagen(imagen, this.x, this.y, 0, 1);
}

public void moverArriba()
//Mueve al personaje y cambia la direccion de Mikasa.
{
      this.y = this.y - 3;
      this.sentido = "arriba";
}

public void moverAbajo()
//Mueve al personaje y cambia la direccion de Mikasa.
{
      this.y = this.y + 3;
      this.sentido = "abajo";
}

public void moverDerecha()
//Mueve al personaje y cambia la direccion de Mikasa.
{
      this.x = this.x + 3;
      this.sentido = "derecha";
}

public void moverIzquierda() //Mueve al personaje y cambia la direccion de
Mikasa.
{
      this.x = this.x - 3;
      this.sentido = "izquierda";
}

public int getX() { //Devuelve la posicion X.
      return this.x;
}

public int getY() {
//Devuelve la posicion Y.
      return this.y;
```

```
}
public int getAncho() {
//Devuelve el valor del ancho.
	return this.ancho;
}
public int getAlto() {
//Devuelve el valor del alto.
	return this.alto;
}
public String getSentido(){
//Devuelve el sentido de Mikasa.
	return this.sentido;
}
public boolean getEstado() { //Devuelve el estado en relacion al suero de
Mikasa.
	return this.pocionActiva;
}

public void cambiarAlto(int alto) { //Permite cambiar el alto de Mikasa.
	this.alto = alto;
}

public void cambiarAncho(int ancho) { //Permite cambiar el ancho de
Mikasa.
	this.ancho = ancho;
}
public void cambiarEstado(boolean nuevoEstado) { //Permite cambiar el
estado de Mikasa.
	this.pocionActiva = nuevoEstado;
}

public Proyectil disparar(){
// Depende la direccion del personaje va a tener cierto x y cierto y, ya
que sino podría salir el proyectil desde fuera del personaje.
	if(this.sentido == "derecha") {
		Proyectil p1 = new Proyectil(this.x + this.ancho, this.y, 80,
10, this.sentido);
		return p1;
		}
	else if(this.sentido == "arriba") {
		Proyectil p1 = new Proyectil(this.x, this.y - this.alto,
10,80, this.sentido);
		return p1;
		}
	else if(this.sentido == "izquierda") {
		Proyectil p1 = new Proyectil(this.x - this.ancho, this.y, 80,
10, this.sentido);
```

```
            return p1;
      }
      else {
            Proyectil p1 = new Proyectil(this.x, this.y + this.alto,
10,80, this.sentido);
            return p1;
      }
}

public int getXLadoDer() { //Devuelve los lados de Mikasa
      return this.x + this.ancho/2;
}
public int getXLadoIzq() {
      return this.x - this.ancho/2;
}
public int getYLadoArriba() {
      return this.y - this.alto/2;
}
public int getYLadoAbajo() {
      return this.y + this.alto/2;
}
}
package juego;

import java.awt.Image;
import entorno.Entorno;
import entorno.Herramientas;

public class Obstaculo {
      private int x;
      private int y;
      private int ancho;
      private int alto;
      private String tipo;
      private Image imagen;

public Obstaculo(int x, int y, int ancho, int alto, String tipo) {
      this.x = x;
      this.y = y;
      this.ancho = ancho;
      this.alto = alto;
      this.tipo = tipo;
}

public void dibujar(Entorno entorno) {
//Depende el tipo, carga la imagen correspondiente.
      if(this.tipo == "casa") {
```

```
	this.imagen = Herramientas.cargarImagen("Imagenes/casa.png");
	entorno.dibujarImagen(this.imagen, this.x, this.y, 0, 0.3);
	}
	if(this.tipo == "arbol") {
		this.imagen = Herramientas.cargarImagen("Imagenes/arbol.png");
		entorno.dibujarImagen(this.imagen, this.x, this.y, 0, 1);
	}
}

public int getX() { //Devuelve varias variables de instancia
	return this.x;
}
public int getY() {
	return this.y;
}
public int getAncho() {
	return this.ancho;
}
public int getAlto() {
	return this.alto;
}
public String getTipo() {
	return this.tipo;
}

public int getXLadoDer() { //Devuelve los lados del obstaculo
	return this.x + this.ancho/2;
}
public int getXLadoIzq() {
	return this.x - this.ancho/2;
}
public int getYLadoArriba() {
	return this.y - this.alto/2;
}
public int getYLadoAbajo() {
	return this.y + this.alto/2;
}
}
package juego;

import entorno.Entorno;
import entorno.Herramientas;

import java.awt.Image;
public class Proyectil {
	private int x;
	private int y;
```

```
	private int ancho;
	private int alto;
	private String sentido;

public Proyectil(int x, int y, int ancho, int alto, String sentido) {
	this.x = x;
	this.y = y;
	this.ancho = ancho;
	this.alto = alto;
	this.sentido = sentido;
}

public void dibujar(Entorno entorno) {
//Depende la direccion del proyectil, carga la respectiva imagen.
		Image imagen =
Herramientas.cargarImagen("Imagenes/misilDer.gif");
		if(this.sentido == "arriba") {
		imagen =
Herramientas.cargarImagen("Imagenes/misilArriba.gif");
	}

	if(this.sentido == "abajo") {
		imagen = Herramientas.cargarImagen("Imagenes/misilAbajo.gif");
	}

	if(this.sentido == "derecha") {
		imagen = Herramientas.cargarImagen("Imagenes/misilDer.gif");
	}

	if(this.sentido == "izquierda") {
		imagen = Herramientas.cargarImagen("Imagenes/misilIzq.gif");
	}

	entorno.dibujarImagen(imagen, this.x, this.y, 0, 0.3);

}
public void mover() {
// Mueve al proyectil dependiendo el sentido que este tenga.
	if(this.sentido == "arriba" )
		this.y = this.y - 4;
	else if(this.sentido == "abajo")
		this.y = this.y + 4;
	else if(this.sentido == "derecha")
		this.x = this.x + 4;
	else
		this.x = this.x - 4;
}
```

```
public String getSentido() { //Devuelve el sentido del proyectil.
	return this.sentido;
}
public int getY() {
	return this.y;
}
public int getX() {
	return this.x;
}
public int getBordeProyectil() {
// Se consigue el borde del proyectil segun la direccion que posea.

	if(this.sentido == "izquierda")
		return this.x - this.ancho/2;

	else if(this.sentido == "derecha")
		return this.x + this.ancho/2;

	else if(this.sentido == "arriba")
		return this.y - this.alto/2;

	else
		return this.y + this.alto/2;

}

}
package juego;
import entorno.Entorno;
import entorno.Herramientas;
import java.awt.Image;

public class Suero {
	private int x;
	private int y;
	private int ancho;
	private int alto;

public Suero(int x, int y, int ancho, int alto) {
	this.x = x;
	this.y = y;
	this.ancho = ancho;
	this.alto = alto;
}

public void dibujar(Entorno entorno) {
	Image imagen = Herramientas.cargarImagen("Imagenes/pocion.png");
```

```
	entorno.dibujarImagen(imagen, this.x, this.y, 0, 0.5);

}
public int getX() { //Devuelve la posicion X.
	return this.x;
}
public int getY() {	//Devuelve la posicion Y.
	return this.y;
}
public int getAncho() { //Devuelve el valor del ancho.
	return this.ancho;
}
public int getAlto() {  //Devuelve el valor del alto.
	return this.alto;
}

public int getXLadoDer() { //Devuelve los lados del suero
	return this.x + this.ancho/2;
}
public int getXLadoIzq() {
	return this.x - this.ancho/2;
}
public int getYLadoArriba() {
	return this.y - this.alto/2;
}
public int getYLadoAbajo() {
	return this.y + this.alto/2;
}
}
```

Conclusiones:

En conclusión, el trabajo práctico resultó muy desafiante, y los resultados, gratificantes. Se considera que la implementación de un código de la magnitud final beneficia en gran medida a los alumnos tanto en el manejo del lenguaje y sus apartados, como en el pensamiento lógico al deber implementar un código legible y con buen diseño.
El constante contacto con arreglos, objetos, sus variables y sus métodos, entre otros apartados, genera una afinidad que facilita el entendimiento de dichos temas.