Trabajo Práctico “Corre mono Corre”

**Materia:** Programación I

**Grupo 6:**

Nombre: Ormeño Martin Felipe

Legajo: 44.505.023

Contacto: fel.martin202@gmail.com

Nombre: Paez Rocco Nicolas

Legajo: 44.171.874

Contacto: Rocconicopaez@gmail.com

Nombre: Aquino Gaspar Abel

Legajo: 44.213.587

Contacto: abelaquino02j@gmail.com

**Objetivo:** Diseñar, coordinar e implementar lo visto en clases en el desarrollo del juego propuesto por los docentes.

**Introducción:**

El proyecto consistía en crear una aplicación que tiene como protagonista a un mono en una selva, manteniendo un plano de dos dimensiones, el cual, junto al mono, siempre se mantenía estático, pero que, al mover los distintos objetos, creaba la ilusión de desplazamiento por parte del mono. Los distintos objetos que observamos en el escenario representan una clase.

**Descripción:**

Para la creación de los distintos objetos que podemos observar en el escenario, se crearon clases, las cuales tienen sus respectivas variables de instancia. A continuación, se nombrarán las clases y una breve explicación (no se explicarán dos veces la misma variable o método implementados):

Arbol :

Con esta clase creamos los árboles, se toman 9 variables de instancia (de las cuales 7 comparte con las clases Tigre, Serpiente, Saco, Piedra, Fruta y Basura).

* Las variables “x” e “y” son de tipo double y representan la coordenada del objeto en la pantalla.
* Las variables enteras “al” y “an” permite definir el tamaño del hitbox del objeto.
* La variable “escala” como su nombre bien lo expresa, es usada para controlar el tamaño del objeto.
* Para la variable “observer” se importó una clase externa y es utilizada para detectar los cambios de las medidas.
* Al igual que con la anterior variable de instancia, para esta también se importa una clase y nos sirve para insertarle una imagen al objeto.
* Variable “rama”, es de tipo Rama y representa si un árbol tiene el objeto rama o no.

Esta clase tiene un constructor y dos métodos.

* El constructor recibe como parámetros una coordenada “x” y una variable de tipo “Suelo” (más adelante veremos de qué se trata), crea los árboles con una probabilidad de que tengan una rama.
* public void dibujarse (Entorno entorno), no hay mucho para explicar, el método dibuja la imagen dentro del entorno, utiliza variables de instancia para determinar el tamaño a dibujar.

public void dibujarse(Entorno entorno) {

entorno.dibujarImagen(arbol, x, y, 0, escala);

}

* public void moverse(), este método decrementa en 4 la posición del x, esto da la ilusión de que el mono corre y va dejando atrás los objetos del escenario (el árbol en este caso).

public void moverse() {

this.x -= 4;

}

Basura / Piedra / Saco / Serpiente / Tigre / Fruta:

Con estas clases se crean los sacos de bolsa de basura, las piedras que lanza el mono y los depredadores que el mono se puede encontrar en el juego.

Todas las variables de instancia que contienen las clases fueron explicadas en la sección “Arbol” (esto porque como anticipamos, varias clases comparten variables de instancia).

Los métodos que tienen dichas clases son los mismos que los de “Arbol”, por lo tanto, no se volverán a detallar. Cabe destacar que, en el caso del tigre, el método “moverse” decrementa más rápido la variable “x” debido a que el tigre al igual que el mono, corre, por lo que debemos tomarlo en cuenta para la ilusión de las posiciones.

Corazon:

Para añadirle un poco de versatilidad al juego, se propuso que este tenga un contador de vidas, por lo cual al perder tres veces se termine el juego. Para llevarlo a la pantalla se implementó la clase “Corazon”.

Además de las variables de instancias que comparte con las anteriores clases (x, y, escala, al, an, y observer), se implementó una de tipo booleana y dos imágenes para diferenciar las vidas que le quedan al jugador.

Su constructor lleva como parámetro un entero para definir el espacio que se quiera ver en la pantalla entre los corazones.

* Tiene solo un método y es el public void dibujarse(Entorno entorno), el cual depende de la variable “lastimado”, si vale true, el corazón se dibujará rojo. Por otro lado, si es false, el corazón se dibujará negro (indicando que no se cuenta con esa vida).

public void dibujarse(Entorno entorno) {

if(!lastimado) {

entorno.dibujarImagen(corazonImg, x, y, 0, escala);

}else {

entorno.dibujarImagen(corazonLasImg, x, y, 0, escala);

}

}

Suelo:

Sus variables de instancias son las vistas hasta ahora, “y, x, an, al, superficie, escala, pisoimg y observer” con las cuales se posiciona el objeto en pantalla, se define un tamaño y se le coloca una imagen.

* Solamente tiene el método public void dibujarse(Entorno entorno) con el cual se dibuja en pantalla el suelo que podemos ver debajo del mono.

Rama:

Uno de los requisitos que debían tener los arboles es que algunos tenían que tener una rama para que el mono pueda saltar sobre ella. Con la clase Rama se logró implementar esto, la clase tiene catorce variables de instancia, algunas anteriormente explicadas, otras nuevas, tales como:

* Para que la rama tenga un tamaño se optó por una variable donde indica el inicio de coordenada de la rama “double principio”.

Por lo tanto, debería tener otra variable que indique donde termina, tal es el caso de “double fin”

* Para que las ramas no tengan siempre una serpiente encima, se implementó la variable “random”, para que haya una probabilidad de aparición.
* Como uno de los requisitos lo pedía, las ramas debían tener un depredador, para eso usamos la variable de tipo Serpiente “serpiente”.
* Se optó por añadir frutas que el mono puede agarrar en las ramas, para eso se usa una lista de tipo frutas que contiene las bananas, esto lo hacemos con la variable “frutas”

Sus métodos son los mismos que vimos en otras clases, “dibujarse(Entorno entorno)” y “public void moverse()”.

Mono:

Como no podía ser de otra manera, nuestro protagonista es un mono, el cual se crea mediante la clase “Mono” cuyas variables de instancia ya vimos a excepción de una:

* Para que el mono no tenga una infinita cantidad de piedras, se decidió que la variable “piedras” sea una lista de tipo Piedra, que contenga la cantidad de piedras que se quiera.

Dentro de la clase se pueden observar cuatro métodos, omitiremos el método dibujarse:

* Para que el mono salte, se usa el método “saltar()” que decrementa la coordenada y del mono.

public void saltar() {

y -= 20;

}

* Como todo lo que sube debe caer, tiene el método “caer()” que va incrementando la coordenada.

public void caer(Entorno entorno) {

this.y += 6;

if(entorno.estaPresionada(entorno.TECLA\_ABAJO)) {

this.y += 10;

}

}

* Por último, encontramos el método “lanzar()” el cual añade un objeto piedra a la variable de instancia “piedras”.

public void lanzar() {

piedras.add(new Piedra(this.x, this.y));

}

Juego:

En la clase juego podemos observar varias variables de instancia, que son:

* “Mono mono;” como su nombre lo dice, es el objeto mono que el jugador controlará.
* “Suelo piso;” es la variable donde se guardará el suelo que podemos ver que pisa el mono.
* “Basura basura;” contiene la basura que vemos en el suelo, la cual cuando el mono la toca, le descuenta puntos al jugador.
* “int tiempo\_sal;” se usa para calcular el tiempo de salto del mono.
* “int tiempo\_disparo;” sirve como contador del lanzamiento de las piedras.
* “int separacion;” se usa para crear un espacio entre dos objetos.
* “int cantPiedras;” indica la cantidad de piedras que tiene el mono.
* “int puntaje;” es un contador de los puntos que lleva el jugador.
* “int puntos\_rama;” establece la cantidad de puntos que va a sumar si el mono se sube a una rama.
* “int record;” se guarda el máximo puntaje obtenido por un jugador.
* “int vidas;” tendrá como valor la cantidad de vidas que tiene el jugador.
* “int obtenerVida;” cuando el jugador obtiene un determinado puntaje, el jugador recibe una vida extra.
* Las listas que forman parte de las variables de instancia se usan para saber cuántos elementos se generan o para ponerlos en “null”, y así, eliminarlos de la pantalla.

Dentro de la clase Juego, podemos encontrar el método “intersecan()” que devuelve un booleano para indicar si dos objetos se intersecan o no.

public boolean intersecan(double x1, double x2, double y1, double y2, int al1, int al2, int an1, int an2) {

boolean intersecY = false;

double distanciaY = Math.max(y1, y2) - Math.min(y1, y2);

double sumaAlt = (al1/2) + (al2/2);

if(distanciaY - sumaAlt < 0) {

intersecY = true;

}

boolean intersecX = false;

double distanciaX = Math.max(x1, x2) - Math.min(x1, x2);

double sumaAnch = (an1/2) + (an2/2);

if(distanciaX - sumaAnch < 0) {

intersecX = true;

}

if(intersecX && intersecY) {

return true;

}else {

return false;

}

}

* **Metodo Tick() :**

Para esta seccion nos pararemos desde la línea 94 del archivo Juego.java hasta la línea 108.

En el primer condicional, si obtener vida y vida es menor 3, recorremos la lista corazones con un for each hasta que el primer corazón el cual su variable booleana lastimado es igual a true, lo va a cambiar a false, va a sumar a la variable vida y resetea la variable “obtenerVida”

En el segundo condicional reccorremos la lista corazones y dibuja el corazón con el método dibujarse (véase la clase arbol).

Para crear los arboles no situamos desde la línea 111 hasta la 113, con un condicional que dicta si el largo de la lista “arboles” es menor a 1000 crea un árbol con una posición inicial y una separación.

Desde la 116 hasta la 132 se establecen las mecánicas del mono, ya sean salto, caer y lanzar piedras.

En la línea 135 se le aplica al método dibujarse() al suielo

Entre las líneas 139 hasta la 162, se reccorre cada objeto árbol y los objetos que este contenga, ya sea si tiene ramas, frutas o serpientes, y si los tiene, les aplica los métodos dibujarse() y moverse().

Entre las líneas 165 y 174, el bloque de código permite que el mono pueda subirse a la rama y que sume puntos por ello.

Entre la 176 y 178, se implementa la mecánica que el mono no traspase la rama.

Entre la 180 hasta la 193, recorremos la lista frutas y por cada fruta si se interseca con el mono se pone en null la fruta y nos suma puntos. Tambien, si la fruta se sale de la pantalla se pone en null.

Desde la 195 hasta la 216, el bloque de código permite que las piedras desaparezcan a la serpiente volviéndolas null, y si la serpiente toca el mono lo ponga en null.

Linea 218 hasta la 220, si el árbol se sale de la pantalla se pone en null.

Desde la 225 hasta la 246, se crea aleatoriamente la basura que vemos en la superficie, a su vez, si la basura interseca con el mono. se le resta puntos al contador de puntos del jugador. Si la basura se sale de la pantalla, se vuelve null.

Desde la línea 249 hasta la 290, creamos objetos tigre aleatoriamente y los añadimos a la iista tigres. Recorremos la lista tigre dibujando cada tigre y si un tigre interseca con el mono, el mono valdra null. En cambio, si una piedra es la que interseca con el tigre o el tigre se sale de la pantalla se vuelve null

Desde la 300 hasta la 315 dibujamos el saco, cuando el mono interseca la bolsa, la bolsa se vuelve null y suma 10 piedras. Si quedan pocas piedras se genera un saco.

Desde la línea 318 hasta la 325, por cada piedra que contenga la lista piedras se le aplica el método dibujarse() y moverse().

Desde la línea 328 hasta la 330 dibuja el mono utilizando el método dibujarse().

Desde la línea 332 hasta la 334 si el jugador presiona la tecla espacio, mientras

tenga vidas, el mono reaparecerá.

Desde la 338 hasta la 344 si una piedra se sale de la vista del jugador se pondrá en null.

Desde la 345 hasta la 354 si las vidas del jugador llegan a 0, todos los objetos se vuelven null y aparecerá un cartel indicando que el juego terminó y mostrará el puntaje que sumo el jugador.

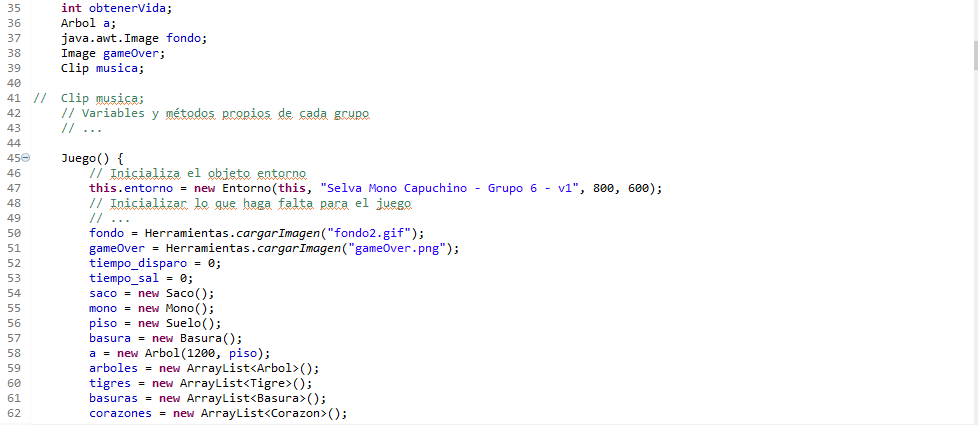
* Método intersecan()

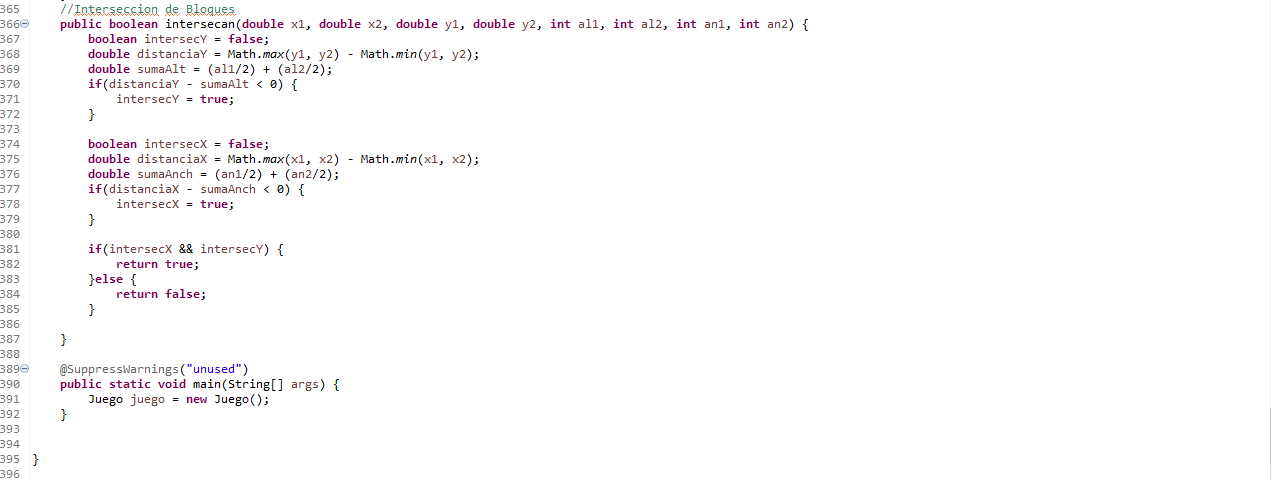
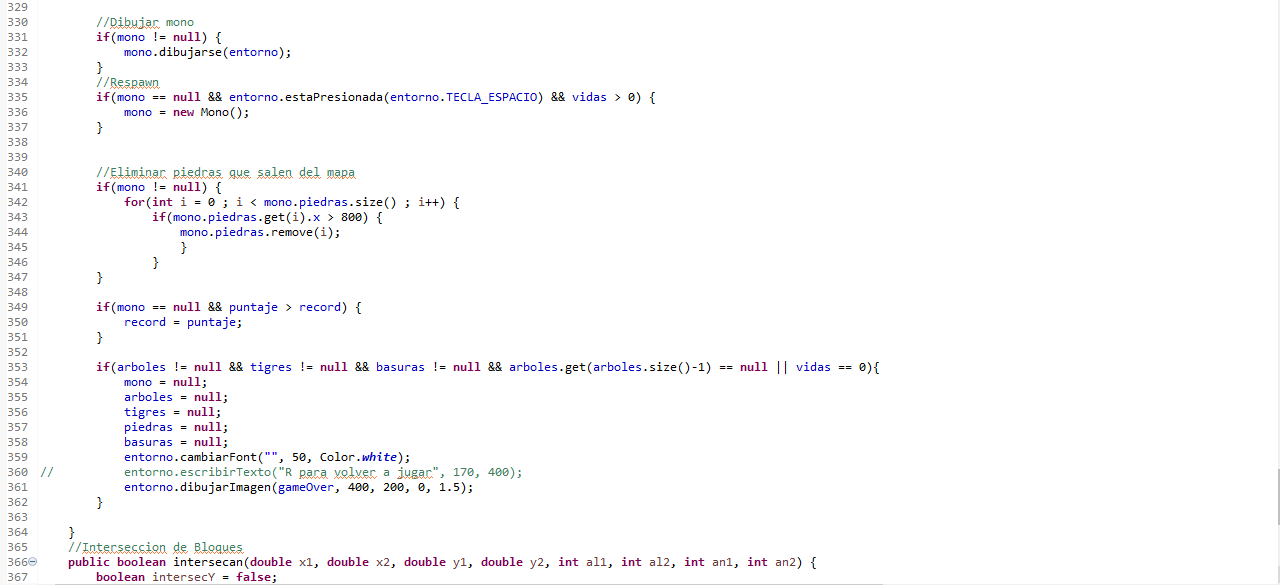
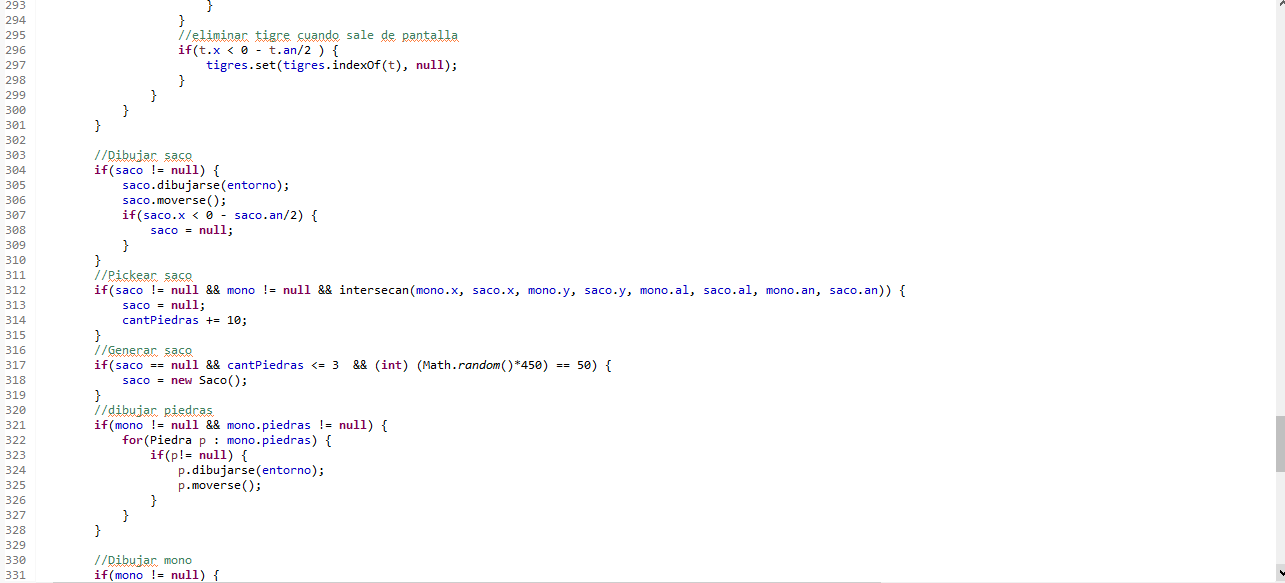
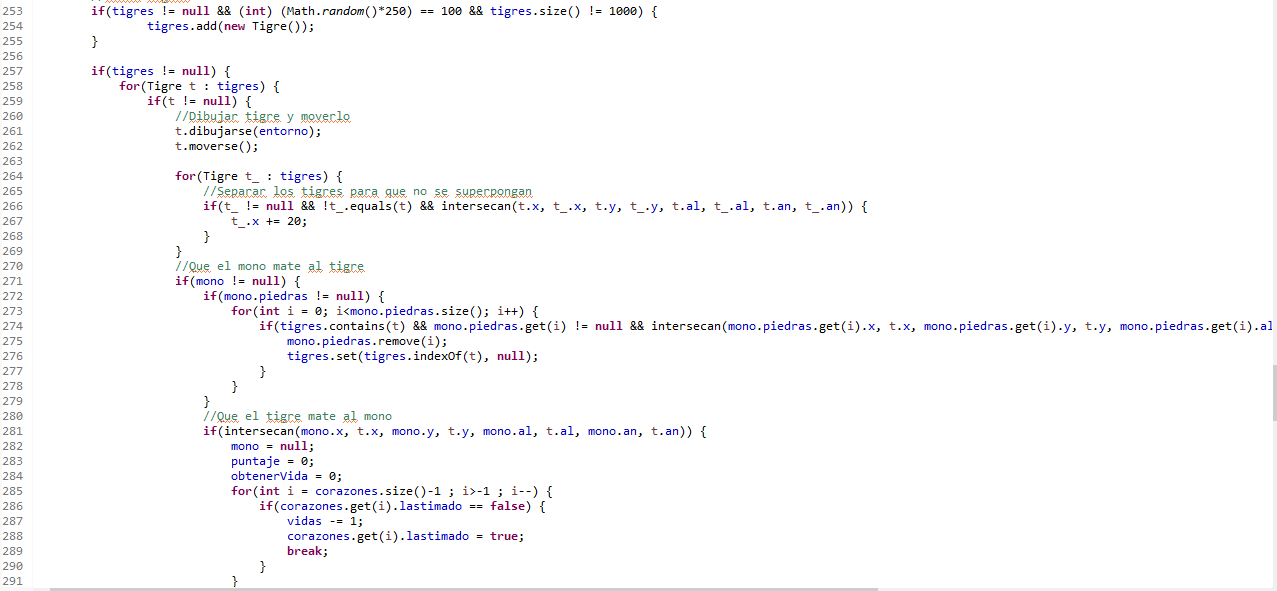
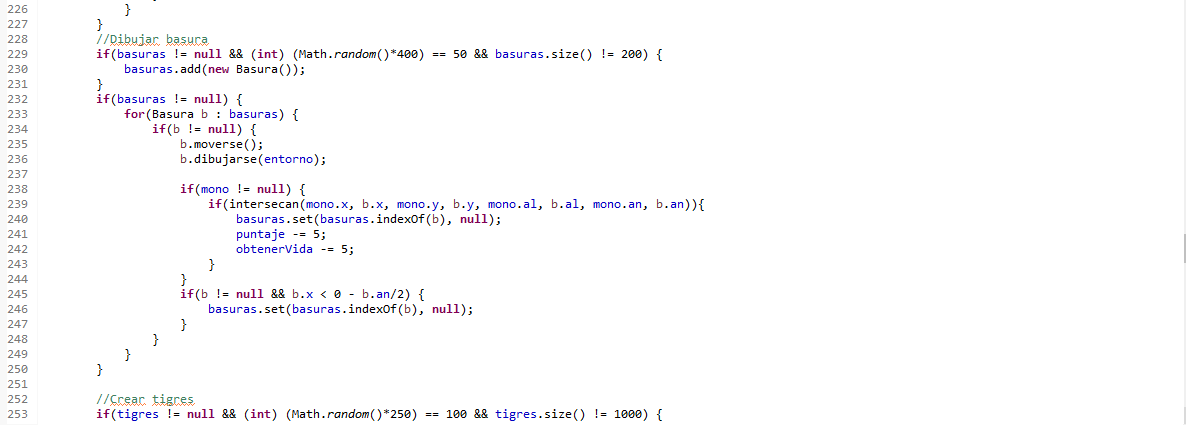
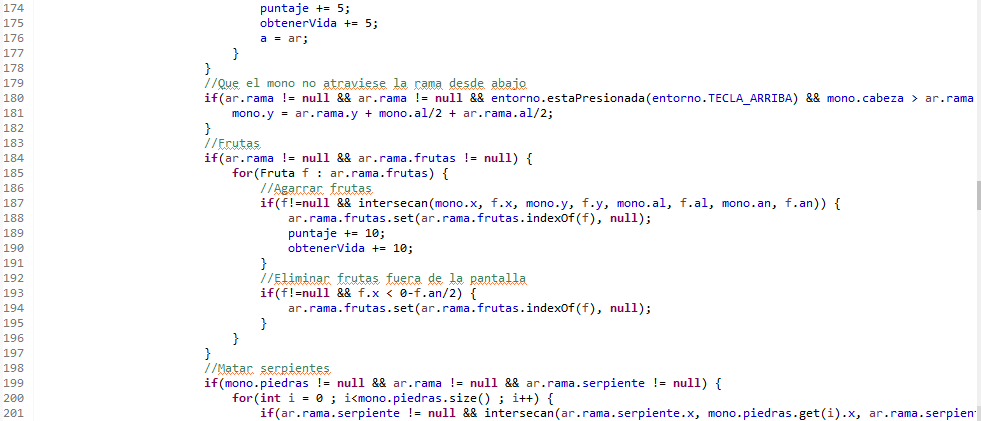
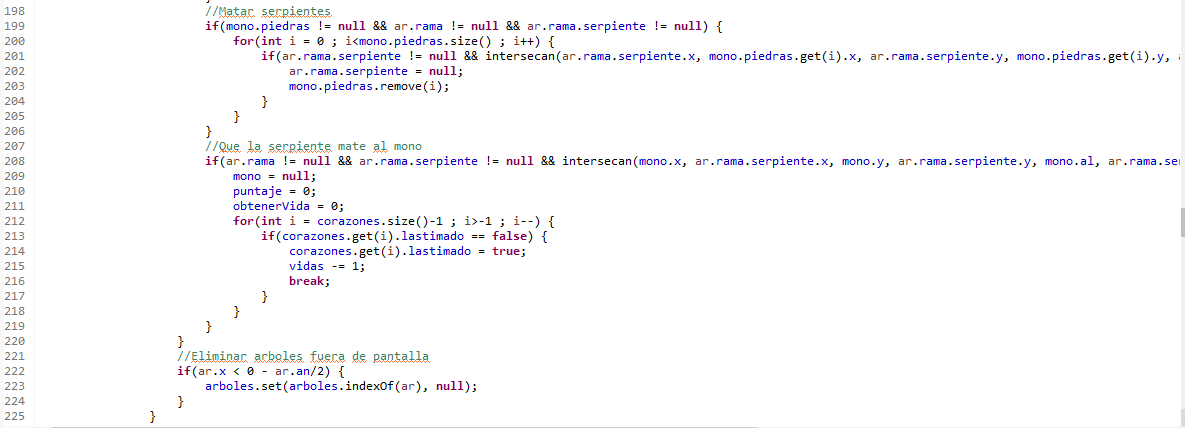
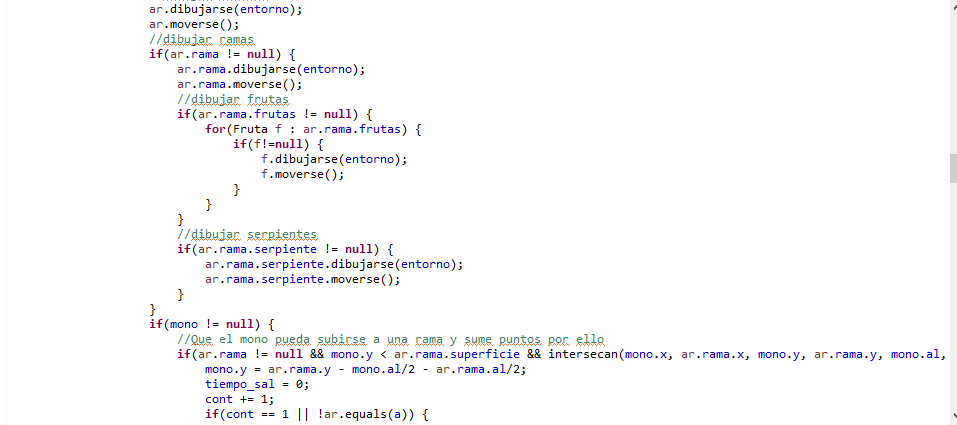
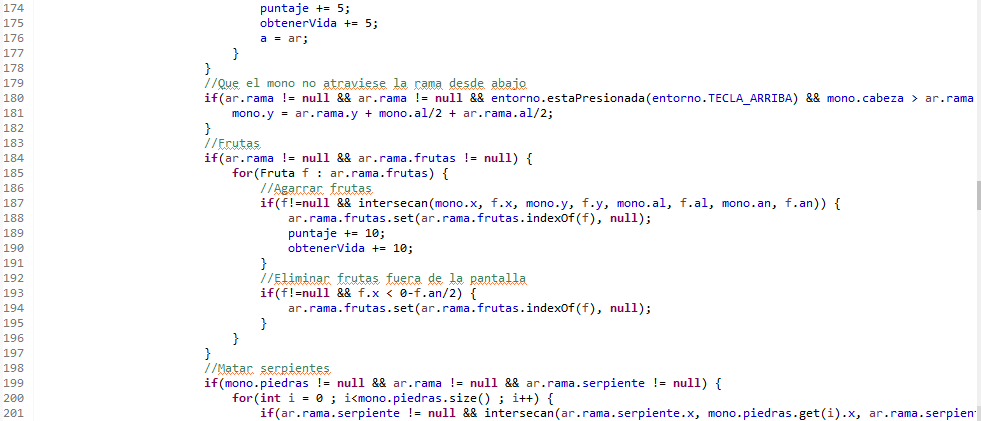
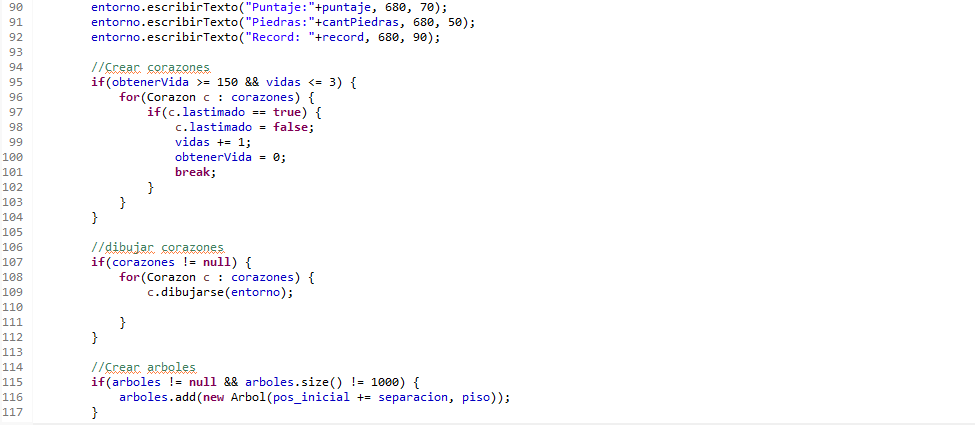
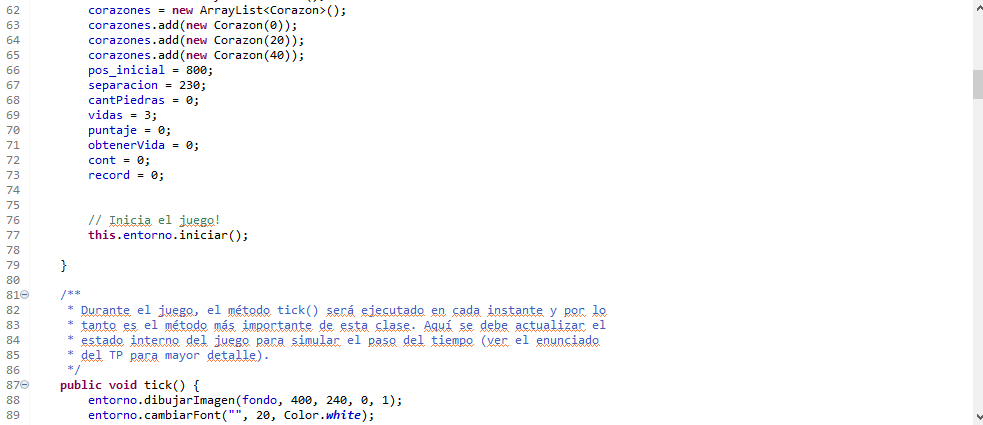
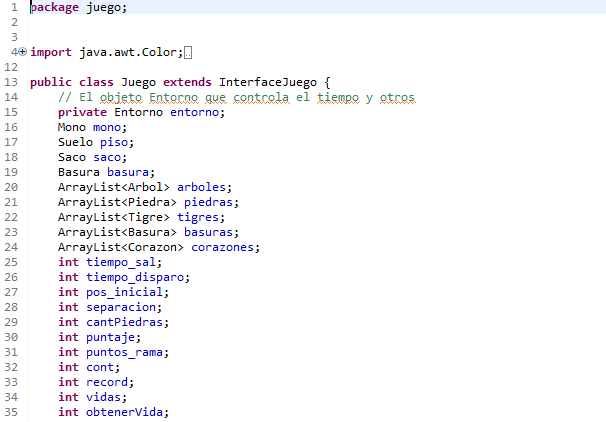
Este método recibe las posiciones “x”, “y”, el alto y el ancho de los bloques y con ellas calcula, la diferencia entre los centros y suma la mitad de los altos de los bloques . Si la distancia entre los centros menos la suma de lois altos es menor a 0, entonces la “variable intersecY” se pone en true, lo que indica que se intersecan en la coordenada “y”.

Para la coordenada “x” se utiliza un código igual al anterior, solo que toma en cuenta la mitad de la suma de los anchos.

Si se intersecan en la coordenada “x” e “y” el método devolverá true, de lo contrario valdrá false.

**Código fuente:**





**Conclusiones:**

Una de las dudas que más se generó en uno de los participantes fue el abordaje de tantos problemas para resolver, a medida que se fue resolviendo el trabajo y con el paso de los días, comprendió que era mejor separar el proyecto grande en pequeñas funcionalidades, que a medida que se iban implementando, creaban el juego interconectándose.

Es importante mantener las buenas prácticas y nombres claros para las variables o métodos, porque permite que alguien que no fue parte del trabajo, leerlo (siempre que tenga los conocimientos correspondientes claro está), así como también comentar las secciones para diferenciar lo que hace cada fragmento de código.

Si tuviésemos que trabajar nuevamente en un proyecto así, sin dudas, usaríamos la herramienta GitLab, esto porque al no usarlo, se complicó algo las implementaciones a archivos con versiones “anteriores” o funcionalidades que un integrante tenía y otro no.

Podemos decir que, se cumplió con el objetivo de desarrollar el juego propuesto por los docentes.