Trabajo práctico “*Castlevania, Barbarianna Viking Edition*”

* Integrantes:
  + Franco Hernán Benitez
    - Email: francobentiez980@gmail.com
  + Máximo De Michieli
    - Email: maxidemichieli@gmail.com
  + Gastón Rampazzo
    - Email: gastonrampazzo@gmail.com
* Materia: **Programación I**
* Comisión: **04**
* Año: **2021**

**Índice**

*Introducción ---------------------------------------------------------------------------------------------------------- 3*

*Descripción ----------------------------------------------------------------------------------------------------------- 4*

*Implementación ---------------------------------------------------------------------------------------------------- 11*

*Conclusiones -------------------------------------------------------------------------------------------------------- 35*

**Introducción**

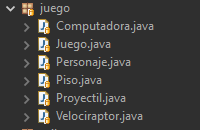
En este trabajo se llevó a cabo el desarrollo de un videojuego denominado “Castlevania, Barbarianna Viking Edition”, que consiste en permitirle a Barbariana recuperar la computadora Commodore 128kb de hackeo del tiempo que le pertenecía a Hackerman y fue robada por el malvado doctor Hero. Para esto, el juego se desarrolla en un mapa que simula ser un castillo con varios pisos. En donde los velociraptors que van a tratar de impedir que Barbariana llegue hacia la computadora, atacaran a esta, hasta tratar de eliminarla del juego. Cada velociraptor con tan solo tocar a Barbariana, logran eliminarla. Además, cuentan con la posibilidad de disparar rayos láser que dañan a la heroína y la dejan fuera de juego.

Por su parte, Barbariana cuenta con el martillo de Thor (el dios del trueno) permitiéndole disparar a los dinos cuando se ve acorralada por estos mismos. Para evitar los disparos de los velociraptors, puede saltar o agacharse.

El desarrollo está hecho en Java, construido por distintas clases que contienen la funcionalidad para cada elemento de la aplicación. Además, cuenta con una librería externa que levanta un entorno en donde ofrece herramientas para dibujar determinadas formas en pantalla, ubicar elementos, etc.

**Descripción**

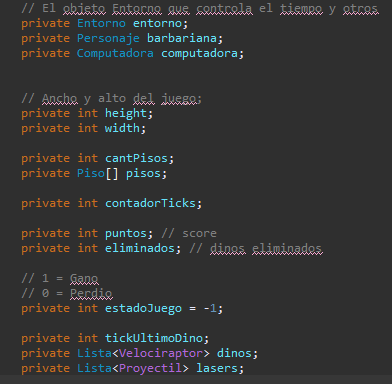
Para el desarrollo del juego, se llevaron a cabo las siguientes clases:



***Clase Juego***

Es la clase principal del proyecto. En esta clase, se crea una instancia del entorno que permite levantar un entorno gráfico para que el juego se ejecute, brinda las herramientas necesarias (métodos) para dibujar los elementos en pantalla e ir actualizando constantemente el estado de los mismos.

La clase cuenta con las siguientes variables de instancia:



* entorno: Instancia del objeto Entorno.
* Barbariana: Instancia del objeto Personaje (personaje principal).
* Computadora: Instancia del objeto Computadora que hace referencia a la computadora 128kb de Hackerman.
* height & width: Dimensiones de la ventana del juego.
* cantPisos: Cantidad de pisos que tiene el juego.
* Pisos: Listado que contiene cada objeto de piso.
* contadorTicks: Contador de cada tick que se ejecuta en el juego.
* puntos: Acumulador de score durante el juego.
* eliminados: Contador de velociraptors eliminados.
* estadoJuego: Tiene 2 estados durante el juego. 1 si gano o 0 si perdió. Permite mostrar un mensaje en específico en pantalla cuando el juego termina.
* tickUltimoDino: El tick en el que se creó el último velociraptor.
* dinos: Contiene una Lista de los velociraptors que hay en pantalla.
* lasers: Contiene una Lista de lasers en pantalla.

Métodos:

* Juego(): Constructor de la clase que instancia los objetos necesarios para el correcto funcionamiento del juego.
* Tick(): Actualiza el estado del personaje, los dinos, sus disparos y láser. Además, chequea colisiones y en el caso de que se deba eliminar un objeto, lo hace.
* \_actualizarMovimientos(): Se encarga de mover al personaje según una tecla presionada y también de disparar cuando sea necesario
* \_inicializarPisos(): Crea los pisos necesarios para el juego y los guarda en la variable de instancia “pisos”
* \_dibujarPisos(): Itera cada instancia de piso que hay almacena en la variable de instancia “pisos” y los dibuja en el mapa
* \_crearDinos(): Crea un nuevo velociraptor cada cierto tiempo y no permite que haya más de 7 velociraptors en el juego.
* \_verificarImpactoDinoAPersonaje(): Verifica si el dino impacta sobre el personaje
* \_verificarImpactoLaserAPersonaje(laser): Verifica si el láser recibido por parámetro está impactando a Barbariana. En caso de que esto suceda, modifica el estado del juego a 0 (perdió).
* verificarImpactoRayoADino(nodoDino): Verifica si el personaje impacta su relámpago sobre el velociraptor recibido por parámetro. En caso de que esto suceda, se eliminan tanto el rayo como el velociraptor que colisionan.
* verificarImpactoRayoALaser(nodoRayo): Verifica si el rayo recibido por parámetro impacta sobre algún láser. En caso de que eso suceda, se eliminan tanto el rayo como el láser que colisionan.
* moverRayo(nodoRayo): Mueve el rayo en la pantalla y lo elimina si sale de la pantalla.
* moverLaser(nodoLaser, lista): Recibe el nodo de una Lista de lásers y una Lista de lásers. Ejecuta el movimiento del láser y si este sale de la pantalla lo elimina de la Lista.
* procesarRayos(): itera el listado de rayos y en cada iteración mueve el rayo y verifica si este impacta con algún láser**.**
* procesarDinos(): Itera la lista de velociraptors y en cada iteración mueve el velociraptor, verifica si éste colisiona con algún rayo y hace disparar al velociraptor.
* moverDino(nodoDino): Recibe el nodo de una Lista de velociraptors y hace avanzar al velociraptor. Además, lo hace descender de piso si es necesario y lo elimina si sale de la pantalla.
* finDelJuego(message,color): Muestra un mensaje con un color determinado en el medio de la pantalla y después de 4 segundos cierra el juego. Es usado cuando el usuario gana o pierde.

***Clase Computadora***

Clase que maneja el comportamiento de la computadora Commodore 128kb de hackeo del tiempo.

Cuenta con los siguientes atributos:



* x, y, ancho y alto: Estos atributos contienen la posición en pantalla que va tener la computadora junto con su tamaño.
* Avatar: Imagen que contiene la foto de la computadora en pantalla.

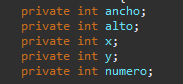
Métodos:

* Computadora(entorno): Crea la computadora para poder dibujarla en pantalla.
* Dibujarse(entorno): Dibuja la computadora en la pantalla.
* estaTocando(xIzq,xDer,yArr,yAba): Recibe una coordenada de un elemento (rectángulo), devuelve true si el elemento está tocando la computadora.

***Clase Piso***

Clase que maneja las dimensiones y las posiciones de cada piso.

Cuenta con los siguientes atributos:



* x, y, ancho y alto: Estos atributos contienen la posición en pantalla que va tener la computadora junto con su tamaño.
* Numero: El número de piso que tiene asignado. (van del 0 al 4)

Métodos:

* Piso(x,y,ancho,numero): Crea el piso con sus coordenadas y dimensiones correspondientes.
* Dibujarse(entorno): Dibuja los pisos en la pantalla.
* posicionSuperior(): Devuelve el lado superior del piso
* posicionInferior(): Devuelve el lado inferior del piso
* posicionExtremoIzquierdo(): Devuelve el lado izquierdo del piso
* posicionExtremoDerecho(): Devuelve el lado derecho del piso

***Clase Proyectil***

Clase que maneja el comportamiento tanto del láser que disparan los dinos como de los rayos que dispara Barbariana.



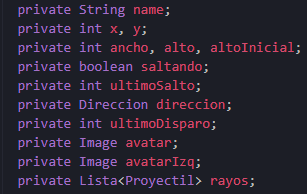
* x, y, ancho y alto: Estos atributos contienen la posición en pantalla que va tener los proyectiles junto con su tamaño.
* Dirección: Representa el sentido en el que se está moviendo el proyectil; puede ser derecha o izquierda.
* Color: Color que va a tener el proyectil en la pantalla.

Métodos:

* Proyectil(x,y,dirección,color): Crea una instancia del proyectil con sus coordenadas, dimensiones y color especificado por parámetro.
* Dibujarse(entorno): Dibuja el proyectil en pantalla
* Moverse (): Mueve el proyectil hacia un sentido determinado. En caso de llegar a los extremos de la pantalla, cambia el sentido.
* esImpactado(xIzq,xDer,yArr,yAba): Recibe 4 coordenadas formando un rectángulo y devuelve true si el proyectil está dentro del rectángulo.
* estaEnPantalla(entorno): Devuelve true si está en pantalla.
* posicionSuperior(): Devuelve el lado superior del proyectil
* posicionInferior(): Devuelve el lado inferior del proyectil
* posicionExtremoIzquierdo(): Devuelve el lado izquierdo del proyectil
* posicionExtremoDerecho(): Devuelve el lado derecho del proyectil

***Clase Personaje:***

Clase que se encarga de manejar todo lo relacionado a Barbariana.



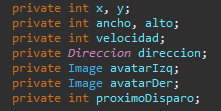
* name: Nombre del personaje
* x & y: coordenadas del personaje
* ancho, alto y altoInicial: dimensiones del personaje
* saltando: Flag que indica si el personaje está saltando
* ultimoSalto: Guarda el tick en el que el personaje emitió su último salto.
* Dirección: sentido en el que se está moviendo el personaje
* ultimoDisparo: Guarda el tick en el que el personaje emitió su último disparo.
* Avatar & avatarIzq: Imagen que representa la imagen personaje en un sentido determinado (izquierda y derecha).
* rayos: La Lista de rayos que disparó el personaje.

Métodos:

* Personaje(name): Crea el personaje en la posición inferior izquierda de la pantalla.
* moverIzquieda(Juego): Mueve a la izquierda el personaje en el caso de que no colisione con el extremo.
* moverDerecha(Juego): Mueve a la derecha el personaje en el caso de que no colisione con el extremo.
* gravedad(Juego): Desciende el personaje si no está pisando sobre un piso.
* saltar(Juego): Eleva el personaje en el caso de que no esté arriba de todo en la pantalla.
* agacharse(): Agacha al personaje.
* pararse(): Retorna al personaje a su altura inicial.
* dibujarse(entorno): Dibuja el personaje en pantalla.
* disparar(contadorTicks): Le permite al personaje disparar, agregando un nuevo rayo a la Lista.
* colisionPiso(pisos): Devuelve true si el personaje toca sobre la parte superior del piso.
* colisionPisoLateral(pisos): Retorna si el personaje está tocando de forma lateral a un piso.
* pisoActual(pisos): Devuelve el piso actual donde se encuentra Barbariana.
* procesarSalto(contadorTicks):
* puedeSubirPiso(pisos): Devuelve true si el personaje se encuentra debajo de un espacio para poder subir de piso.
* subirPiso(pisos,numeroDePiso): lleva al personaje al número de piso recibido como parámetro.
* esImpactado(xIzq,xDer,yArr,yAba): Recibe 4 coordenadas formando un rectángulo y devuelve true si el personaje está dentro del rectángulo.
* posicionCabeza(): Devulve el lado superior del personaje
* posicionPies(): Devuelve el lado inferior del personaje
* posicionExtremoIzquierdo(): Devulve el lado izquierdo del personaje
* posicionExtremoDerecho(): Devuelve el lado derecho del personaje

***Clase Velociraptor***

Clase que maneja las tareas de los dinos.



* x & y: coordenadas del dino
* ancho & alto: dimensiones del dino
* velocidad: Velocidad a la que se desplaza el dino
* Dirección: Sentido en el que se desplaza el dino
* avatarIzq & avatarDer: Imagen del dino en pantalla según su sentido de desplazamiento
* proximoDisparo: contiene el número de tick en donde se va efectuar el próximo disparo del dino aleatoriamente.

Métodos:

* Velociraptor(): Crea un dino
* dibujarse(entorno): Dibuja el dino en el sentido correspondiente
* gravedad(juego): Hace descender al dino cuando llega al espacio de cada piso.
* avanzar(pisos): Mueve al dino en el sentido que corresponda por la pantalla.
* disparar(lasers,contadorTicks): Dispara los lasers hacia Barbariana agregando un nuevo láser a la Lista.
* colisionPiso(pisos): Devuelve true si el dino toca sobre la parte superior de un piso.
* pisoActual(pisos): Devuelve el número de piso en el que está el dino
* estaEnPantalla(entorno): Devuelve true si el dino está en pantalla
* esImpactado(xIzq,xDer,yArr,yAba): Devuelve true si el dino es impactado por las coordenadas que recibe por parámetro (un rectángulo).
* posicionExtremoIzquierdo(): Devuelve el lado izquierdo del dino
* posicionExtremoDerecho(): Devuelve el lado derecho del dino
* posicionCabeza(): Devuelve el lado superior del dino
* posicionPies(): Devuelve el lado inferior del dino

**Implementación**

Código Fuente:

**Clase: Juego**

|  |
| --- |
| **package juego;  import java.awt.Color; import java.awt.Font; import java.awt.Image; import entorno.Herramientas; import entorno.Entorno; import entorno.InterfaceJuego; import utils.Lista; import utils.Nodo;  public class Juego extends InterfaceJuego {  // El objeto Entorno que controla el tiempo y otros  private Entorno entorno;  private Personaje barbariana;  private Computadora computadora;  private Image avatarVida;   // Ancho y alto del juego;  private int height;  private int width;   private int cantPisos;  private Piso[] pisos;   private int contadorTicks;   private int puntos; // score  private int eliminados; // dinos eliminados  private int vidas = 3;   // 1 = Gano  // 0 = Perdio  private int estadoJuego = -1;   private int tickUltimoDino;  private Lista<Velociraptor> dinos;  private Lista<Proyectil> lasers;   Juego() {  // Inicializa alto y ancho  this.height = 600;  this.width = 800;   this.cantPisos = 5;   // Inicializa el objeto entorno  this.entorno = new Entorno(this, "Grupo 9", this.width, this.height);   // Inicializa el personaje  this.barbariana = new Personaje("Barbariana");   this.avatarVida = Herramientas.cargarImagen("vida.png");   // Inicializa dinos  this.dinos = new Lista<Velociraptor>();  this.lasers = new Lista<Proyectil>();  this.tickUltimoDino = -1;   // Crea la computadora  this.computadora = new Computadora(this.entorno);   // crea los pisos con sus respectivas ubicaciones  this.\_inicializarPisos();   // Inicia el juego!  this.entorno.iniciar();   }   // Metodo que se ejecuta todo el tiempo  public void tick() {  if (this.estadoJuego != -1) {  if (this.estadoJuego == 1) {  this.finDelJuego("Ganaste!", Color.GREEN);  } else if (this.estadoJuego == 0) {  this.finDelJuego("Perdiste", Color.RED);  }  } else {  // dibujar el personaje  this.barbariana.dibujarse(this.entorno);   // dibuja los pisos  this.\_dibujarPisos();   // dibuja computadora  this.computadora.dibujarse(this.entorno);   // crea y procesa dinos  this.\_crearDinos();  if (this.dinos.largo() > 0) {  this.procesarDinos();  }   if (this.lasers.largo() > 0) {  this.procesarLasers();  }   // Se verifica el impacto del dino no solo cuando se mueve a un lado, sino  // tambien  // cuando esta quieto el personaje.  this.\_verificarImpactoDinoAPersonaje();   // metodo que maneja las acciones de acuerdo a la tecla que se presione en el  // momento  this.\_actualizarMovimientos();   if (this.barbariana.getRayos().largo() > 0) {  this.procesarRayos();  }   if (this.barbariana.getSaltando()) {  this.barbariana.procesarSalto(this.contadorTicks);  }   if (!this.barbariana.colisionPiso(this.pisos) && !this.barbariana.getSaltando()) {  this.barbariana.gravedad(this);  }   entorno.cambiarFont(Font.SANS\_SERIF, 25, Color.RED);  entorno.escribirTexto("Enemigos Eliminados: " + this.eliminados, 15, 25);  entorno.escribirTexto("Vidas: " + this.vidas, 80, 585);  entorno.escribirTexto("Score: " + this.puntos, this.width - 150, 585);  entorno.dibujarRectangulo(40, 570, 50, 50, 0, Color.WHITE);  entorno.dibujarImagen(this.avatarVida, 40, 570, 0, 0.2);   this.contadorTicks += 1;  }  }   /\*\*  \* De acuerdo a una tecla presioanda, ejecuta si accion correspondiente  \*/  private void \_actualizarMovimientos() {  if (this.entorno.estaPresionada(this.entorno.TECLA\_DERECHA)) {  // chequea que barbariana no llegue a la computadora.  // Si llega, termina el juego y gana  if (this.computadora.estaTocando(this.barbariana.posicionExtremoIzquierdo(),  this.barbariana.posicionExtremoDerecho(), this.barbariana.posicionCabeza(), this.barbariana.posicionPies())) {  this.estadoJuego = 1;  }  this.barbariana.moverDerecha(this);  }   if (this.entorno.estaPresionada(this.entorno.TECLA\_IZQUIERDA)) {  this.barbariana.moverIzquierda(this);  }  // si presiono flecha arriba salta  if (this.entorno.estaPresionada(this.entorno.TECLA\_ARRIBA) && this.barbariana.colisionPiso(this.pisos)) {  this.barbariana.saltar(this);  }   if (this.entorno.estaPresionada(this.entorno.TECLA\_ABAJO)) {  this.barbariana.agacharse();  } else {  this.barbariana.pararse();  }   if (this.entorno.estaPresionada('u') && this.barbariana.puedeSubirPiso(this.pisos)) {  this.barbariana.subirPiso(this.pisos, this.barbariana.pisoActual(this.pisos) + 1);  }   if (this.entorno.estaPresionada(this.entorno.TECLA\_ESPACIO) && this.barbariana.colisionPiso(this.pisos)  && this.barbariana.getUltimoDisparo() + 50 <= this.contadorTicks) {  this.barbariana.disparar(this.contadorTicks);  }  }   private void \_inicializarPisos() {  Piso[] pisosList = new Piso[this.cantPisos];  int xPiso = this.width;  int yPiso = this.height - 50;  int anchoPiso = this.width \* 2;  for (int i = 0; i < this.cantPisos; i++) {  Piso p = new Piso(xPiso, yPiso, anchoPiso, i);  pisosList[i] = p;  yPiso = yPiso - 110;  xPiso = xPiso == this.width ? 0 : this.width;  anchoPiso = (this.width - 150) \* 2;  }  this.pisos = pisosList;  }   private void \_crearDinos() {  if (this.tickUltimoDino == -1 || (this.dinos.largo() <= 6 && this.tickUltimoDino + 300 < this.contadorTicks)) {  this.dinos.agregarAtras(new Velociraptor());  this.tickUltimoDino = this.contadorTicks;  }  }   private void \_dibujarPisos() {  for (Piso piso : this.pisos) {  piso.dibujarse(this.entorno);  }  }   private void \_verificarImpactoDinoAPersonaje() {  this.dinos.forEachElement(dino -> {  if (this.barbariana.esImpactado(dino.posicionExtremoIzquierdo(), dino.posicionExtremoDerecho(),  dino.posicionCabeza(), dino.posicionPies())) {  this.estadoJuego = 0;  }  return null;  });  }   private void \_verificarImpactoLaserAPersonaje(Proyectil laser) {  if (this.barbariana.esImpactado(laser.posicionExtremoIzquierdo(), laser.posicionExtremoDerecho(),  laser.posicionYArriba(), laser.posicionYAbajo()))  this.estadoJuego = 0;  }   private void verificarImpactoRayoADino(Nodo<Velociraptor> dino) {  this.barbariana.getRayos().forEachNodo(nodoRayo -> {  Proyectil rayo = nodoRayo.getElemento();  if (dino.getElemento().esImpactado(rayo.posicionExtremoIzquierdo(), rayo.posicionExtremoDerecho(),  rayo.posicionYArriba(), rayo.posicionYAbajo())) {  this.dinos.quitarPorId(dino.getId());  this.barbariana.getRayos().quitarPorId(nodoRayo.getId());  this.eliminados++;  this.puntos += 15;  }  return null;  });  }   private void verificarImpactoRayoALaser(Nodo<Proyectil> nodoRayo) {  this.lasers.forEachNodo(nodoLaser -> {  Proyectil laser = nodoLaser.getElemento();  if (nodoRayo.getElemento().esImpactado(laser.posicionExtremoIzquierdo(), laser.posicionExtremoDerecho(),  laser.posicionYArriba(), laser.posicionYAbajo())) {  this.lasers.quitarPorId(nodoLaser.getId());  this.barbariana.getRayos().quitarPorId(nodoRayo.getId());  this.puntos += 5;  }  return null;  });  }   private void moverRayo(Nodo<Proyectil> nodoRayo) {  nodoRayo.getElemento().moverse();  nodoRayo.getElemento().dibujarse(this.entorno);  if (!nodoRayo.getElemento().estaEnPantalla(this.entorno)) {  this.barbariana.getRayos().quitarPorId(nodoRayo.getId());  }  }   private void moverLaser(Nodo<Proyectil> nodoLaser, Lista<Proyectil> lista) {  nodoLaser.getElemento().moverse();  nodoLaser.getElemento().dibujarse(this.entorno);  if (!nodoLaser.getElemento().estaEnPantalla(this.entorno)) {  lista.quitarPorId(nodoLaser.getId());  }  }   private void procesarRayos() {  this.barbariana.getRayos().forEachNodo(rayo -> {  this.moverRayo(rayo);  this.verificarImpactoRayoALaser(rayo);  return null;  });  }   private void procesarLasers() {  this.lasers.forEachNodo(nodoLaser -> {  this.moverLaser(nodoLaser, this.lasers);  this.\_verificarImpactoLaserAPersonaje(nodoLaser.getElemento());  return null;  });  }   private void procesarDinos() {  this.dinos.forEachNodo(nodoDino -> {  this.moverDino(nodoDino);  this.verificarImpactoRayoADino(nodoDino);  nodoDino.getElemento().disparar(this.lasers, this.contadorTicks);  return null;  });  }   private void moverDino(Nodo<Velociraptor> nodoDino) {  Velociraptor dino = nodoDino.getElemento();  if (!dino.colisionPiso(this.pisos))  dino.gravedad(this);  dino.avanzar(this.pisos);  dino.dibujarse(this.entorno);  if (!dino.estaEnPantalla(this.entorno)) {  dinos.quitarPorId(nodoDino.getId());  }  }   /\*\*  \* Ejecuta mensaje en pantalla y finaliza la ejecución.  \*/  private void finDelJuego(String message, Color color) {  entorno.cambiarFont(Font.SANS\_SERIF, 30, color);  entorno.escribirTexto(message, this.width / 2 - 50, this.height / 2);   // ejecuta una funcion despues de determinado tiempo en ms (milisegundos)  new java.util.Timer().schedule(new java.util.TimerTask() {  public void run() {  System.exit(0);  }  }, 4000);  }   // Getters and setters   public Entorno getEntorno() {  return entorno;  }   public void setEntorno(Entorno entorno) {  this.entorno = entorno;  }   public Personaje getBarbariana() {  return barbariana;  }   public void setBarbariana(Personaje barbariana) {  this.barbariana = barbariana;  }   public Computadora getComputadora() {  return computadora;  }   public void setComputadora(Computadora computadora) {  this.computadora = computadora;  }   public Image getAvatarVida() {  return avatarVida;  }   public void setAvatarVida(Image avatarVida) {  this.avatarVida = avatarVida;  }   public int getHeight() {  return height;  }   public void setHeight(int height) {  this.height = height;  }   public int getWidth() {  return width;  }   public void setWidth(int width) {  this.width = width;  }   public int getCantPisos() {  return cantPisos;  }   public void setCantPisos(int cantPisos) {  this.cantPisos = cantPisos;  }   public Piso[] getPisos() {  return pisos;  }   public void setPisos(Piso[] pisos) {  this.pisos = pisos;  }   public int getContadorTicks() {  return contadorTicks;  }   public void setContadorTicks(int contadorTicks) {  this.contadorTicks = contadorTicks;  }   public int getPuntos() {  return puntos;  }   public void setPuntos(int puntos) {  this.puntos = puntos;  }   public int getVidas() {  return vidas;  }   public void setVidas(int vidas) {  this.vidas = vidas;  }   public int getTickUltimoDino() {  return tickUltimoDino;  }   public void setTickUltimoDino(int tickUltimoDino) {  this.tickUltimoDino = tickUltimoDino;  }   public Lista<Velociraptor> getDinos() {  return dinos;  }   public void setDinos(Lista<Velociraptor> dinos) {  this.dinos = dinos;  }   public int getEstadoJuego() {  return this.estadoJuego;  }   public void setEstadoJuego(int estado) {  this.estadoJuego = estado;  }   @SuppressWarnings("unused")  public static void main(String[] args) {  Juego juego = new Juego();  } }** |

**Clase: Personaje**

|  |
| --- |
| **package juego;  import java.awt.Color; import java.awt.Image; import entorno.Entorno; import utils.Direccion; import utils.Lista; import entorno.Herramientas;  public class Personaje {  private String name;  private int x, y;  private int ancho, alto, altoInicial;  private boolean saltando;  private int ultimoSalto;  private Direccion direccion;  private int ultimoDisparo;  private Image avatar;  private Image avatarIzq;  private Lista<Proyectil> rayos;   Personaje(String name) {  this.x = 50;  this.y = 505;  this.ancho = 30;  this.alto = 60;  this.altoInicial = 60;  this.name = name;  this.direccion = Direccion.DERECHA;  this.avatar = Herramientas.cargarImagen("barbariana.jpeg");  this.avatarIzq = Herramientas.cargarImagen("barbarianaIzq.jpg");  this.rayos = new Lista<Proyectil>();  }   public void moverIzquierda(Juego j) {  if (this.posicionExtremoIzquierdo() <= 0 || this.colisionPisoLateral(j.getPisos()))  return;  this.x = this.x - 3;  if (this.direccion.equals(Direccion.DERECHA))  this.direccion = Direccion.IZQUIERDA;  }   public void moverDerecha(Juego j) {  if (this.posicionExtremoDerecho() >= j.getWidth() || this.colisionPisoLateral(j.getPisos()))  return;  this.x = this.x + 3;  if (this.direccion.equals(Direccion.IZQUIERDA))  this.direccion = Direccion.DERECHA;  }   public void gravedad(Juego j) {  if (this.posicionPies() >= j.getHeight() || this.saltando)  return;  this.y += 2.5;  }   public void saltar(Juego j) {  if (this.posicionCabeza() <= 0)  return;  this.ultimoSalto = j.getContadorTicks();  this.saltando = true;  }   public void agacharse() {  if (this.alto >= this.altoInicial) {  this.alto = this.alto / 2;  this.y += this.alto / 2;  }  }   public void pararse() {  if (this.alto < this.altoInicial) {  this.y -= this.alto / 2;  this.alto = this.altoInicial;  }  }   public void dibujarse(Entorno e) {  if (this.direccion.equals(Direccion.DERECHA)) {  e.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));  e.dibujarImagen(this.avatar, this.x, this.y, 0, 0.2);  }  if (this.direccion.equals(Direccion.IZQUIERDA)) {  e.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));  e.dibujarImagen(this.avatarIzq, this.x, this.y, 0, 0.2);  }  }   public void disparar(int contadorTicks) {  this.rayos.agregarAtras(new Proyectil(this.x, this.y - 10, this.direccion, Color.YELLOW));  this.ultimoDisparo = contadorTicks;  }   public boolean colisionPiso(Piso[] pisos) {  for (Piso piso : pisos) {  if (piso.posicionSuperior() == this.posicionPies()  && this.posicionExtremoIzquierdo() <= piso.posicionExtremoDerecho()  && this.posicionExtremoDerecho() >= piso.posicionExtremoIzquierdo()) {  this.saltando = false;  return true;  }  }  return false;  }   public boolean colisionPisoLateral(Piso[] pisos) {  for (Piso piso : pisos) {  if (piso.posicionSuperior() < this.posicionPies() && piso.posicionInferior() > this.posicionCabeza()  && this.posicionExtremoIzquierdo() <= piso.posicionExtremoDerecho()  && this.posicionExtremoDerecho() >= piso.posicionExtremoIzquierdo()) {  return true;  }  }  return false;  }   public int pisoActual(Piso[] pisos) {  for (Piso piso : pisos) {  if (piso.posicionSuperior() == this.posicionPies()  && this.posicionExtremoIzquierdo() <= piso.posicionExtremoDerecho()  && this.posicionExtremoDerecho() >= piso.posicionExtremoIzquierdo()) {  this.saltando = false;  return piso.getNumero();  }  }  return -1;  }   public void procesarSalto(int contadorTicks) {  if (this.ultimoSalto + 16 < contadorTicks) {  this.saltando = false;  return;  }  this.y -= 2;  }   public boolean puedeSubirPiso(Piso[] pisos) {  int pisoActual = this.pisoActual(pisos);  try {  if (pisoActual != -1 && (this.posicionExtremoIzquierdo() >= pisos[pisoActual + 1].posicionExtremoDerecho()  || this.posicionExtremoDerecho() <= pisos[pisoActual + 1].posicionExtremoIzquierdo()))  return true;  } catch (Exception e) {  return false;  }  return false;  }   public void subirPiso(Piso[] pisos, int numeroDePiso) {  try {  Piso pisoASubir = pisos[numeroDePiso];  if (Math.abs(pisoASubir.posicionExtremoDerecho() - this.x) < Math  .abs(pisoASubir.posicionExtremoIzquierdo() - this.x)) {  this.y = pisoASubir.posicionSuperior() - this.alto / 2;  this.x = pisoASubir.posicionExtremoDerecho() - this.ancho / 2;  }  if (Math.abs(pisoASubir.posicionExtremoIzquierdo() - this.x) < Math  .abs(pisoASubir.posicionExtremoDerecho() - this.x)) {  this.y = pisoASubir.posicionSuperior() - this.alto / 2;  this.x = pisoASubir.posicionExtremoIzquierdo() + this.ancho / 2;  }  } catch (Exception e) {  return;  }  }   public boolean esImpactado(int xIzq, int xDer, int yArr, int yAba) {  boolean tocandoX = xIzq < this.posicionExtremoDerecho() && this.posicionExtremoIzquierdo() < xDer;  boolean tocandoY = yAba > this.posicionCabeza() && this.posicionPies() > yArr;  return tocandoX && tocandoY;  }   public int posicionExtremoDerecho() {  return this.x + this.ancho / 2;  }   public int posicionExtremoIzquierdo() {  return this.x - this.ancho / 2;  }   public int posicionCabeza() {  return this.y - this.alto / 2;  }   public int posicionPies() {  return this.y + this.alto / 2;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public int getX() {  return x;  }   public void setX(int x) {  this.x = x;  }   public int getY() {  return y;  }   public void setY(int y) {  this.y = y;  }   public int getAncho() {  return ancho;  }   public void setAncho(int ancho) {  this.ancho = ancho;  }   public int getAlto() {  return alto;  }   public void setAlto(int alto) {  this.alto = alto;  }   public boolean getSaltando() {  return this.saltando;  }   public int getUltimoDisparo() {  return this.ultimoDisparo;  }   public Lista<Proyectil> getRayos() {  return this.rayos;  }   public void setRayos(Lista<Proyectil> rayos) {  this.rayos = rayos;  } }** |

**Clase: Velociraptor**

|  |
| --- |
| **package juego;  import java.awt.Color; import java.awt.Image;  import entorno.Entorno; import entorno.Herramientas; import utils.Direccion; import utils.Lista;  public class Velociraptor {  private int x, y;  private int ancho, alto;  private int velocidad;  private Direccion direccion;  private Image avatarIzq;  private Image avatarDer;  private int proximoDisparo;   public Velociraptor() {  this.x = 770;  this.y = 75;  this.ancho = 30;  this.alto = 60;  this.velocidad = 2;  this.direccion = Direccion.IZQUIERDA;  this.avatarDer = Herramientas.cargarImagen("Dino.png");  this.avatarIzq = Herramientas.cargarImagen("DinoIzq.png");  }   public void dibujarse(Entorno entorno) {  entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));  if (this.direccion.equals(Direccion.IZQUIERDA))  entorno.dibujarImagen(this.avatarIzq, this.x, this.y, 0, 1);  if (this.direccion.equals(Direccion.DERECHA))  entorno.dibujarImagen(this.avatarDer, this.x, this.y, 0, 1);  }   public void gravedad(Juego j) {  if (this.posicionPies() >= j.getHeight())  return;  this.y += 2.5;  }   public void avanzar(Piso[] pisos) {  int pisoActual = this.pisoActual(pisos);  if ((this.x >= 20 || pisoActual == 0) && this.direccion.equals(Direccion.IZQUIERDA))  this.x -= velocidad;  if (this.x == 20 && pisoActual != 0)  this.direccion = Direccion.DERECHA;  if (this.x <= 790 && this.direccion.equals(Direccion.DERECHA))  this.x += velocidad;  if (this.x == 790)  this.direccion = Direccion.IZQUIERDA;  }   public void disparar(Lista<Proyectil> lasers, int contadorTicks) {  if (this.proximoDisparo < contadorTicks) {  if (this.proximoDisparo != 0) {  int y = randomEntre(0, 1) == 0 ? this.y - 10 : this.y + 15;  lasers.agregarAtras(new Proyectil(this.x, y, this.direccion, Color.RED));  }  this.proximoDisparo = contadorTicks + this.randomEntre(150, 250);  }  }   public boolean colisionPiso(Piso[] pisos) {  for (Piso piso : pisos) {  if (piso.posicionSuperior() == this.posicionPies()  && this.posicionExtremoIzquierdo() <= piso.posicionExtremoDerecho()  && this.posicionExtremoDerecho() >= piso.posicionExtremoIzquierdo()) {  return true;  }  }  return false;  }   public int pisoActual(Piso[] pisos) {  for (Piso piso : pisos) {  if (piso.posicionSuperior() == this.posicionPies()  && this.posicionExtremoIzquierdo() <= piso.posicionExtremoDerecho()  && this.posicionExtremoDerecho() >= piso.posicionExtremoIzquierdo()) {  return piso.getNumero();  }  }  return -1;  }   public boolean estaEnPantalla(Entorno entorno) {  return this.posicionExtremoDerecho() > 0 && this.posicionExtremoIzquierdo() < entorno.ancho();  }   public boolean esImpactado(int xIzq, int xDer, int yArr, int yAba) {  boolean tocandoX = xIzq < this.posicionExtremoDerecho() && this.posicionExtremoIzquierdo() < xDer;  boolean tocandoY = yAba > this.posicionCabeza() && this.posicionPies() > yArr;  return tocandoX && tocandoY;  }   private int randomEntre(int n1, int n2) {  return (int) Math.floor(Math.random() \* (n2 - n1 + 1) + n1);  }   public int posicionExtremoDerecho() {  return this.x + this.ancho / 2;  }   public int posicionExtremoIzquierdo() {  return this.x - this.ancho / 2;  }   public int posicionCabeza() {  return this.y - this.alto / 2;  }   public int posicionPies() {  return this.y + this.alto / 2;  }   public int getX() {  return x;  }   public int getY() {  return y;  }   public int getAncho() {  return ancho;  }   public int getAlto() {  return alto;  }   public int getVelocidad() {  return velocidad;  } }** |

**Clase: Computadora**

|  |
| --- |
| **package juego;  import java.awt.Color; import java.awt.Image;  import entorno.Entorno; import entorno.Herramientas;  public class Computadora {  private int x, y, ancho, alto;  private Image avatar;   /\*\*  \* Por default se ubica a la computadora arriba a la derecha  \*/  Computadora(Entorno e) {  this.x = e.ancho() - 50;  this.y = 75;  this.ancho = 50;  this.alto = 50;  this.avatar = Herramientas.cargarImagen("PC.png");  }   public void dibujarse(Entorno e) {  e.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, Color.BLACK);  e.dibujarImagen(this.avatar, this.x, this.y, 0, 1);  }   /\*\*  \* Retorna true si el rectángulo recibido está tocando a la computadora  \*   \* @param xIzq  \* @param xDer  \* @param yArr  \* @param yAba  \* @return  \*/  public boolean estaTocando(int xIzq, int xDer, int yArr, int yAba) {  boolean tocandoX = xIzq < this.posicionExtremoDerecho() && this.posicionExtremoIzquierdo() < xDer;  boolean tocandoY = yAba > this.posicionYArriba() && this.posicionYAbajo() > yArr;  return tocandoX && tocandoY;  }   public int getX() {  return x;  }    public void setX(int x) {  this.x = x;  }   public int getY() {  return y;  }   public void setY(int y) {  this.y = y;  }   public int getAncho() {  return ancho;  }   public void setAncho(int ancho) {  this.ancho = ancho;  }   public int getAlto() {  return alto;  }   public void setAlto(int alto) {  this.alto = alto;  }   public int posicionExtremoDerecho() {  return this.x + this.ancho / 2;  }   public int posicionExtremoIzquierdo() {  return this.x - this.ancho / 2;  }   public int posicionYArriba() {  return this.y - this.alto / 2;  }   public int posicionYAbajo() {  return this.y + this.alto / 2;  } }** |

**Clase: Proyectil**

|  |
| --- |
| **package juego;  import java.awt.Color;  import entorno.Entorno; import utils.Direccion;  public class Proyectil {  private int x, y, ancho, alto;  private Direccion direccion;  private Color color;   Proyectil(int x, int y, Direccion direccion, Color color) {  this.x = x;  this.y = y;  this.ancho = 40;  this.alto = 5;  this.direccion = direccion;  this.color = color;  }   public void dibujarse(Entorno e) {  e.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, this.color);  }   public void moverse() {  if (direccion.equals(Direccion.DERECHA)) {  this.x += 5;  } else {  this.x -= 5;  }  }   public boolean esImpactado(int xIzq, int xDer, int yArr, int yAba) {  boolean tocandoX = xIzq < this.posicionExtremoDerecho() && this.posicionExtremoIzquierdo() < xDer;  boolean tocandoY = yAba > this.posicionYArriba() && this.posicionYAbajo() > yArr;  return tocandoX && tocandoY;  }   public boolean estaEnPantalla(Entorno entorno) {  return this.posicionExtremoDerecho() > 0 && this.posicionExtremoIzquierdo() < entorno.ancho();  }   public int posicionExtremoDerecho() {  return this.x + this.ancho / 2;  }   public int posicionExtremoIzquierdo() {  return this.x - this.ancho / 2;  }   public int posicionYArriba() {  return this.y - this.alto / 2;  }   public int posicionYAbajo() {  return this.y + this.alto / 2;  } }** |

**Clase: Lista**

|  |
| --- |
| **package utils;  import java.util.function.Function;  public class Lista<T> {  Nodo<T> primero;  private int proximoId;   public Lista() {  this.primero = null;  this.proximoId = 0;  }   public void mostrar() {  Nodo<T> actual = this.primero;  System.out.print("[");  while (actual != null) {  System.out.print(actual.id + " ");  actual = actual.siguiente;  }  System.out.println("]");  }   public void agregarAtras(T n) {  Nodo<T> nuevo = new Nodo<T>(n, proximoId);  this.proximoId += 1;  if (this.primero == null) {  this.primero = nuevo;  } else {  Nodo<T> actual = this.primero;  while (actual.siguiente != null) {  actual = actual.siguiente;  }  actual.siguiente = nuevo;  }  }   public void agregarAdelante(T n) {  Nodo<T> nuevo = new Nodo<T>(n, proximoId);  this.proximoId += 1;  if (this.primero == null) {  this.primero = nuevo;  } else {  nuevo.siguiente = this.primero;  this.primero = nuevo;  }  }   public int largo() {  Nodo<T> actual = this.primero;  int cant = 0;  while (actual != null) {  cant++;  actual = actual.siguiente;  }  return cant;  }   public void quitarDePosicion(int pos) {  Nodo<T> actual = this.primero;  if (this.primero != null && pos == 0) {  this.primero = this.primero.siguiente;  }  if (pos >= this.largo())  return;  int i = 1;  while (actual != null && actual.siguiente != null) {  if (i == pos) {  actual.siguiente = actual.siguiente.siguiente;  return;  }  actual = actual.siguiente;  i++;  }  }   public void quitarPorId(int id) {  Nodo<T> actual = this.primero;  if (this.primero != null && this.primero.id == id) {  this.primero = this.primero.siguiente;  }  while (actual != null && actual.siguiente != null) {  if (actual.siguiente.id == id) {  actual.siguiente = actual.siguiente.siguiente;  return;  }  actual = actual.siguiente;  }  }   public T obtenerDePosicion(int pos) {  Nodo<T> actual = this.primero;  int i = 0;  while (actual != null && actual.siguiente != null) {  if (i == pos) {  return actual.elemento;  }  actual = actual.siguiente;  i++;  }  return null;  }   public T obtenerPorId(int id) {  Nodo<T> actual = this.primero;  while (actual != null && actual.siguiente != null) {  if (actual.id == id) {  return actual.elemento;  }  actual = actual.siguiente;  }  return null;  }   public void forEachNodo(Function<Nodo<T>, Void> callback) {  Nodo<T> actual = this.primero;  while (actual != null) {  callback.apply(actual);  actual = actual.siguiente;  }  }   public void forEachElement(Function<T, Void> callback) {  Nodo<T> actual = this.primero;  while (actual != null) {  callback.apply(actual.elemento);  actual = actual.siguiente;  }  } }** |

**Clase: Nodo**

|  |
| --- |
| **package utils;  public class Nodo<T> {  T elemento;  Nodo<T> siguiente;  int id;   public Nodo() {  }   public Nodo(T elemento, int id) {  this.elemento = elemento;  this.siguiente = null;  this.id = id;  }   public T getElemento() {  return this.elemento;  }   public int getId() {  return this.id;  } }** |

**Conclusión**

Como conclusión, luego de haber podido finalizar el desarrollo del trabajo, creemos que se hizo un buen trabajo en done la división de tareas y responsabilidades fue lo más equitativa posible. Además, se logró un buen ambiente de trabajo gracias a que se llevó a cabo desde un principio una metodología de trabajo orientada a SCRUM, en donde se planificaron las tareas y cada uno de los integrantes adoptó la que quería e iba modificando el estado de la misma. Los estados de las tareas fueron los siguientes:

* To Do (pendiente)
* WIP (Work in progress)
* Done (Terminado)

Para guardar el código e ir trabajando todos al mismo tiempo, se utilizo un repositorio git alojado en GitHub, y para el relevamiento de tareas se utilizó la plataforma Trello.