Universidad Nacional de General Sarmiento (UNGS)

Primer semestre 2024

Programación I

Princesa Saltarina

Integrantes:

-Moragues, Paula Abril

DNI: 46.343.328

EMAIL: paulamoraguesabril30@gmail.com

-Monsegur, Felipe

DNI: 45.736.829

EMAIL: felipemonsegur@gmail.com

-Escalante, Alessandro

DNI: 44.650.053

EMAIL: alessandroescalante99@gmail.com

Introducción:

En este trabajo práctico, nuestro objetivo es desarrollar un videojuego. Este es protagonizado por la princesa Elizabeth, y la historia se centra en su enfrentamiento con el Rey Camir, quien secuestró a su mascota y la llevó a la cima de un volcán. La misión de la princesa consiste en rescatar a su gato. Para lograr su objetivo, la princesa Elizabeth debe romper bloques y ascender por ellos. Además, podrá eliminar a los enemigos que se interpongan en su camino.

El objetivo de este trabajo práctico no solo trata de crear un videojuego entretenido, sino también aprender y aplicar nuestros conocimientos de programación. A lo largo del desarrollo del proyecto, nos enfrentamos a diversos desafíos que serán desarrollados a lo largo del informe.

Descripción:

En este apartado explicaremos cada clase implementada, describiendo las variables de instancia y métodos utilizados. Por otro lado, incluiremos los problemas encontrados y las decisiones que tomamos para resolverlos.

Clase Juego:

Variables de instancia:

Private Entorno entorno; variable donde se define el objeto entorno, es decir, la ventana de juego.

Private Fondo fondo; variable donde se encuentra la imagen del fondo del juego.

Private Ladrillo [] [] pisos; arreglo de objetos de la clase Ladrillo donde se definen los pisos utilizados en el juego.

Private Princesa princesa; variable donde se define el objeto princesa, que representa el personaje principal del juego.

Private Bala bala; variable donde se define un objeto de la clase Bala, que representa un proyectil disparado por la princesa.

Private Trex [] [] trexs; arreglo de objetos de la clase Trex que representa una colección de enemigos (Trex) que aparecen en cada piso del juego.

Private int trexsEnPantalla; variable de tipo int que lleva la cuenta de la cantidad de trex presentes en pantalla.

Private Image imag; variable de tipo Image utilizada para almacenar imágenes temporales que se van a dibujar en la pantalla, como el corazón que representa las vidas de la princesa.

Private int trexsEliminados; variable de tipo int que lleva la cuenta de la cantidad de Trexs eliminados por la princesa.

Private int puntos; variable de tipo int que lleva la cuenta de los puntos acumulados por el jugador durante el juego.

Private int vidas; variable de tipo int que representa la cantidad de vidas restantes de la princesa. Comienza en 3 y disminuye cuando la princesa es golpeada por un enemigo o proyectil.

Private boolean inmunidad; variable de tipo booleana que indica si la princesa es inmune al daño. Es true al inicio del juego para evitar que la princesa pierda una vida inmediatamente.

Private boolean estadoInicio; variable de tipo booleana que indica si el juego está en la pantalla de inicio. Si es true, se muestra la pantalla de inicio con las instrucciones del juego, si es false, el juego está en progreso.

Clase Princesa: clase donde se crea el objeto princesa, quien es el personaje principal del juego.

Variables de instancia:

Private double x; en esta variable se define un dato de tipo double que representa la posición horizontal (coordenada x) de la princesa en el entorno del juego.

Private double y; en esta variable se define un dato de tipo double que representa la posición vertical (coordenada y) de la princesa en el entorno del juego.

Private double ancho; indica el ancho de la princesa. Es importante para detectar colisiones con otros objetos.

Private double alto; indica la altura de la princesa. Al igual que el ancho, es importante para detectar colisiones con otros objetos.

Private int dirección; esta variable almacena la dirección en la que se está moviendo la princesa. Se utiliza tanto para cambiar su imagen, como para orientar sus disparos en el juego.

Private Image imag; esta variable contiene la imagen de la princesa.

Private double velocidadY; esta variable representa la velocidad en el eje vertical de la princesa. Es importante para implementar adecuadamente la física del salto y la caída.

Private double gravedad; esta variable se usa para simular el efecto de la gravedad al momento de saltar.

Private boolean enelaire; variable de tipo booleano que indica si la princesa está en el aire o no. Es importante para determinar cuando la princesa puede saltar nuevamente.

Métodos:

- void dibujar (Entorno e): dibuja la imagen de la princesa.
- void cambiarImagen (): cambia la imagen de la princesa cuando es golpeada, considerando su dirección.
- void moverDerecha (Entorno e): mueve hacia la derecha a la princesa, modifica su dirección y carga la imagen correspondiente.

- void moverIzquierda (): mueve hacia la izquierda a la princesa, modifica su dirección y carga la imagen correspondiente.
- Bala disparar (): crea y devuelve un objeto de tipo Bala en la posición y dirección actual en la que se encuentra la princesa.
- void saltar (): este método hace que la princesa salte estableciéndole a "velocidadY" un valor negativo, con el fin de simular un salto hacia arriba, en el caso de que esta no se encuentre en el aire.
- void mover (Ladrillo [] [] pisos): mueve a la princesa de acuerdo a las colisiones con los ladrillos. Verifica si la princesa está en el suelo o en el aire y ajusta su velocidad y posición en consecuencia. Además permite que la princesa rompa ladrillos destructibles cuando esta colisiona con ellos desde abajo.
- boolean colision (Ladrillo ladrillo): devuelve un booleano que verifica si la princesa colisiona con un ladrillo dado. Esto se hace comprobando si los bordes de la princesa intersectan con los bordes del ladrillo.

Clase Trex: clase donde se crea el objeto Trex, enemigo de la princesa.

Variables de instancia:

Private double x; en esta variable se define un dato de tipo double que representa la posición horizontal (coordenada x) del Trex en el entorno del juego.

Private double y; en esta variable se define un dato de tipo double que representa la posición vertical (coordenada y) del Trex en el entorno del juego.

Private double ancho; representa el ancho del Trex.

Private double alto; representa la altura del Trex.

Private int direccion; esta variable almacena la dirección en la que se está moviendo el Trex. Se utiliza tanto para cambiar su imagen, como para orientar sus disparos en el juego.

Private Image imag; almacena la imagen del Trex.

Private double velocidadY; almacena la velocidad vertical del Trex.

Private double gravedad; Almacena la fuerza de gravedad que actúa sobre el Trex.

Private Random random; variable de tipo Random utilizada para generar números aleatorios.

Private Hueso hueso; Almacena una instancia de la clase Hueso que el Trex puede disparar.

Métodos:

- void dibujar (Entorno e): dibuja la imagen del Trex.
- void mover (Ladrillo [] [] pisos, Entorno e, Trex [] [] otrosTrexs): mueve al Trex de acuerdo a las colisiones con los ladrillos, y ajusta su posición en consecuencia. Cambia la direccion de este si llega al borde izquierdo o derecho del entorno. Además, evita que este se superponga con otros Trex.

- void moverDerecha (): mueve el Trex hacia la derecha.
- void moverIzquierda (): mueve el Trex hacia la izquierda.
- void dispararHueso: permite al Trex disparar un hueso con cierta probabilidad, teniendo en cuenta la direccion del Trex.
- boolean colision (Ladrillo ladrillo): método que verifica si hay colisión entre el Trex y un objeto de la clase Ladrillo.
- boolean colisionConBala (Bala balas): método que verifica si hay colisión entre el Trex y una instancia de la clase Bala.
- boolean colisionConTrex (Trex otroTrex): método que verifica si hay colisión entre el Trex y otro Trex.
- boolean colisionConPrincesa (Princesa princesa): método que verifica si hay colisión entre el Trex y una instancia de la clase Princesa.

Clase Bala: clase donde se crea el objeto Bala, que se trata del arma de la princesa.

Variables de instancia:

Private double x; Almacena la coordenada x de la posición de la bala en el entorno.

Private double y; Almacena la coordenada y de la posición de la bala en el entorno.

Private double diametro; Almacena el diámetro de la bala.

Private double velocidad; Almacena la velocidad de la bala.

Private Image imag; Almacena la imagen asociada a la bala.

Private int direccion; Almacena la dirección en la que se desplaza la bala.

Métodos:

- void dibujar (Entorno e): dibuja la imagen de la bala.
- void mover (): método que mueve la bala en la dirección correspondiente multiplicando la velocidad por la dirección.
- boolean chocaConEntorno (Entorno e): método que verifica si la bala choca con los límites del entorno.

Clase Hueso: clase donde se crea el objeto Hueso, que se trata del arma de los Trexs.

Variables de instancia:

Private double x; Almacena la coordenada x de la posición del hueso en el entorno.

Private double y; Almacena la coordenada y de la posición del hueso en el entorno.

Private double diametro; Almacena el diámetro del hueso.

Private double velocidad; Almacena la velocidad de desplazamiento del hueso.

Private Image imag; Almacena la imagen asociada al hueso.

Private int direccion; Almacena la dirección en la que se desplaza el hueso

Métodos:

- void dibujar (Entorno e): método que dibuja la imagen del hueso en la posición (x, y) del entorno e.
- void mover (): método que mueve el hueso en la dirección correspondiente multiplicando la velocidad por la dirección.
- boolean chocaConEntorno (Entorno e): método que verifica si el hueso choca con los límites del entorno.
- boolean colisionConBala (Bala bala): método que verifica si hay colisión entre el hueso y una instancia de la clase Bala.
- boolean colisionConPrincesa (Princesa princesa): método que verifica si hay colisión entre el hueso y una instancia de la clase Princesa.

Clase Ladrillo: clase donde se crea el objeto Ladrillo, por donde se mueven los personajes lo largo del juego.

Variables de instancia:

Private double x; almacena la coordenada x de la posición del ladrillo en el entorno.

Private double y; almacena la coordenada y de la posición del ladrillo en el entorno.

Private double ancho; almacena el ancho del ladrillo.

Private double alto; almacena la altura del ladrillo.

Private Image imag; almacena la imagen asociada al ladrillo (en caso de que sea destructible).

Private Image imagM; almacena la imagen asociada al ladrillo de metal (en caso de que no sea destructible).

Private boolean destructible: Indica si el ladrillo es destructible o no.

Métodos:

- boolean isDestructible (): método que devuelve true si el ladrillo es destructible, false si no lo es.
- void setDestructible (boolean destructible): método que permite establecer si el ladrillo es destructible o no.
- void dibujar (Entorno e): método que dibuja el ladrillo en el entorno. Si el ladrillo es destructible, se dibuja con su imagen asociada (imag), si no es destructible, se dibuja con la imagen de metal (imagM).

Clase fondo: clase donde se crea el objeto Fondo, que representa el fondo del juego.

Variables de instancia:

Private image imag; carga la imagen del fondo

Métodos:

- void dibujar: dibujar la imagen del fondo

Problemas encontrados, decisiones tomadas y soluciones encontradas:

Nos enfrentamos a varios desafíos durante el desarrollo del proyecto, siendo el primero la colisión de los personajes con los pisos de ladrillos. En un principio, observamos que la princesa solo colisionaba con la parte inferior del ladrillo, lo que impedía su caída a través de los pisos. Este problema también afectaba a los trexs, lo que generó complicaciones significativas en el funcionamiento del juego. Para abordar esta dificultad, implementamos un booleano que indicaba cuando la princesa estaba en el suelo, y agregamos una condición adicional para verificar la colisión con la parte superior del bloque. Esta solución permitió resolver el problema de manera efectiva.

El segundo gran desafío surgió al intentar establecer como nulos los bloques eliminados. Esta acción causó un mal funcionamiento en el juego cuando tratamos de implementarla inicialmente. Optamos temporalmente por no dibujar los bloques eliminados, aunque esta solución no era óptima y requería una corrección. Al analizar más a fondo, identificamos que al establecer los bloques como nulos solo estábamos considerando la colisión de la princesa con ellos, omitiendo la colisión de los trexs. Para resolver este problema, ajustamos el manejo de las colisiones para incluir a ambos personajes, lo que finalmente permitió solucionar esta dificultad de manera satisfactoria.

Implementación:

```
public Juego() {
    // Inicializa el objeto entorno
    this.entorno = new Entorno(this, "Princesa Saltarina - Grupo 9 - Monsegur - Moragues - Escalante - V0.50", 800,
            600);
    // Inicializar lo que haga falta para el juego
    // ...

    this.fondo = new Fondo(entorno.ancho() / 2, entorno.alto() / 2);
    this.pisos = new Ladrillo[5][]; // Crear 5 pisos en total
    this.estadoInicio = true; // Comienza en la pantalla de inicio
    reiniciar();

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

Se inicializa el fondo, y se define un arreglo de arreglos de tipo Ladrillo para representar los 5 pisos del juego. Además, se establece el estado inicial del juego y este se inicia.

```
// reinicia el juego y vuelve a dibujar los pisos de manera diferente, los trexs
// y la princesa. Se reinicia el contador también
public void reiniciar() {
    inmunidad = true;
    Random rand = new Random();

    // Crear los ladrillos para cada piso
    for (int i = 0; i < pisos.length; i++) {
        pisos[i] = new Ladrillo[entorno.ancho() / 50]; // Cantidad de ladrillos por piso
        int metales = 0;
        for (int j = 0; j < pisos[i].length; j++) {
            int tipoBloque;
            if (i == 0) {
                // Para el primer piso, todos los ladrillos son normales
                tipoBloque = 1;
            } else {
                if (metales < 6 && j <= pisos[i].length - 3) {
                    if (rand.nextInt(4) == 3) { // Solo cuando randomX es 3 se crean 3 ladrillos de metal seguidos
                        tipoBloque = 3;
                        for (int k = 0; k < 3; k++) {
                            if (j + k < pisos[i].length) {
                                pisos[i][j + k] = new Ladrillo((j + k) * 50 + 25,
                                        entorno.alto() + 120 - ((i + 1) * 140), 50, 50, entorno, tipoBloque);
                            }
                        }
                        metales += 3;
                        j += 2; // Avanzar el índice para no sobrescribir los ladrillos de metal
                        continue;
                    } else {
                        tipoBloque = 1;// Ladrillo
                    }
                } else {
                    tipoBloque = 1;
                }
            }
            pisos[i][j] = new Ladrillo(j * 50 + 25, entorno.alto() + 120 - ((i + 1) * 140), 50, 50, entorno,
                    tipoBloque);
        }
    }
```

‘inmunidad’ se establece como true, lo que genera que la princesa sea invulnerable temporalmente.

Luego, se recorre cada piso, y para cada uno se crea un arreglo de Ladrillo con una longitud igual al ancho de pantalla dividido por 50. Para el primer piso, todos los ladrillos son normales, mientras que para los otros pisos, se pueden crear ladrillos de metal. Se verifica que no hayan más de 6 ladrillo de metal por piso.

```
  this.puntos = 0;
  this.trexsEliminados = 0;
  this.trexsEnPantalla = 0;
  this.vidas = 3;
  this.princesa = new Princesa(entorno.ancho() / 2, entorno.alto() - 70);
  this.balas = null;

  Random random = new Random();
  this.trexs = new Trex[4][2]; // Crear 2 trexs por piso
  for (int i = 0; i < trexs.length; i++) {
      for (int j = 0; j < trexs[i].length; j++) {
          trexs[i][j] = new Trex(random.nextInt(entorno.ancho()), entorno.alto() - (70 + i * 140));
          // Genera un numero aleatorio para aparecer el trex
          this.trexsEnPantalla += 1;
      }
  }
```

Se inicializan los puntos, el contador de trexs eliminado, trexs en pantalla y las vidas, así como también la princesa y las balas. Por último, se crean 2 trexs por cada piso.

```
public void tick() {
    // Procesamiento de un instante de tiempo
    // ...

    // antes de empezar el juego
    if (estadoInicio) {
        // Mostrar pantalla de inicio con instrucciones
        entorno.cambiarFont("Calibri", 40, java.awt.Color.magenta);
        entorno.escribirTexto("Bienvenido a Princesa Saltarina", entorno.ancho() / 2 - 250,
                entorno.alto() / 2 - 150);
        entorno.cambiarFont("Calibri", 30, java.awt.Color.white);
        entorno.escribirTexto("[←] Mover a la izquierda", entorno.ancho() / 2 - 200, entorno.alto() / 2 - 70);
        entorno.escribirTexto("[→] Mover a la derecha", entorno.ancho() / 2 - 200, entorno.alto() / 2 - 40);
        entorno.escribirTexto("Instrucciones:", entorno.ancho() / 2 - 200, entorno.alto() / 2 - 100);
        entorno.escribirTexto("[C] Dispara", entorno.ancho() / 2 - 200, entorno.alto() / 2 - 10);
        entorno.escribirTexto("[X] Salta", entorno.ancho() / 2 - 200, entorno.alto() / 2 + 20);
        entorno.escribirTexto("Reglas del Juego:", entorno.ancho() / 2 - 200, entorno.alto() / 2 + 60);
        entorno.escribirTexto("Tenes 3 vidas.", entorno.ancho() / 2 - 200, entorno.alto() / 2 + 90);
        entorno.escribirTexto("Si un hueso te toca, pierdes una vida.", entorno.ancho() / 2 - 200,
                entorno.alto() / 2 + 120);
        entorno.escribirTexto("Si un Trex te toca, pierdes todas las vidas.", entorno.ancho() / 2 - 200,
                entorno.alto() / 2 + 150);
        entorno.escribirTexto("Presiona [Enter] para comenzar", entorno.ancho() / 2 - 180,
                entorno.alto() / 2 + 200);

        if (entorno.sePresiono(entorno.TECLA_ENTER)) {
            estadoInicio = false; // Iniciar el juego cuando se presiona Enter
            reiniciar();
        }
        return; // Salir del método tick para no ejecutar el resto del juego hasta que se
                // presione Enter
    }


// empieza el juego
fondo.dibujar(entorno);

// dibuja cada Ladrillo
for (Ladrillo[] piso : pisos) {
    for (Ladrillo ladrillo : piso) {
        if (ladrillo != null) {
            ladrillo.dibujar(entorno);
        }
    }
}

// mostrar puntos
entorno.cambiarFont("Calibri", 23, java.awt.Color.black);
entorno.escribirTexto("TREXS ELIMINADOS:" + this.trexsEliminados, 0, entorno.alto() - 24);
entorno.escribirTexto("PUNTOS:" + this.puntos, 0, entorno.alto() - 1);
for (int i = 0; i < vidas; i++) {
    this.imag = Herramientas.cargarImagen("heart.png");
    entorno.dibujarImagen(imag, entorno.ancho() - 30 - (i * 40), entorno.alto() - 23, 0, 4);
}
```

```
// Aparecer muchos trexs
for (int i = 0; i < trexs.length; i++) {
    for (int j = 0; j < trexs[i].length; j++) {
        if (trexs[i][j] != null) {
            trexs[i][j].dibujar(entorno);
            trexs[i][j].mover(pisos, entorno, trexs);
            if (this.princesa != null && !inmunidad && trexs[i][j].colisionConPrincesa(princesa)) {
                // trex le saca una vida a la princesa si colisionan
                vidas = 0;
            }

            // Cada rex dispara un hueso si puede
            trexs[i][j].dispararHueso();
            Hueso hueso = trexs[i][j].getHueso();
            if (hueso != null) {
                hueso.dibujar(entorno);
                hueso.mover();
                if (hueso.chocaConEntorno(entorno)) {
                    trexs[i][j].setHueso(null); // Eliminar hueso si sale de la pantalla
                }
                // Verificar colisión entre balas y huesos
                if (this.balas != null && hueso.colisionConBala(this.balas)) {
                    trexs[i][j].setHueso(null); // Eliminar hueso si colisiona con la bala
                }
                // Verificar colisión entre princesa y huesos
                if (this.princesa != null && !inmunidad && hueso.colisionConPrincesa(this.princesa)) {
                    trexs[i][j].setHueso(null); // Eliminar hueso si colisiona con princesa
                    this.vidas -= 1;
                    this.princesa.cambiarImagen();
                }

            }

        }
    }
}
```

Esta parte del código se encarga de dibujar los trexs y permitirles moverse dentro del entorno del juego. Si el trex colisiona con la princesa, se reduce su vida a cero.

Luego, cada uno de los trexs tiene la posibilidad de disparar huesos, que se dibujan y mueven por pantalla. Si hay balas presentes en el juego y el hueso colisiona con alguna de ellas, se elimina el hueso. Por último, si el hueso colisiona con la princesa, se elimina el hueso y se reduce la cantidad de vidas de la princesa en uno.

```
// aparece 1 trexs mas si solo hay 2 trexs en pantalla
if (this.trexsEnPantalla == 2) {
    Random random = new Random();
    int nuevoI = random.nextInt(4); // Obtener un índice aleatorio para la fila
    int nuevoJ = random.nextInt(1); // Obtener un índice aleatorio para la columna
    // Verificar que la posición no este ocupada por un trex existente
    while (trexs[nuevoI][nuevoJ] != null) {
        nuevoI = random.nextInt(4);
        nuevoJ = random.nextInt(trexs[nuevoI].length);
    }
    // Agregar un nuevo Trex en la posición aleatoria
    trexs[nuevoI][nuevoJ] = new Trex(random.nextInt(entorno.ancho()), entorno.alto() - (70 + nuevoI * 140));
    this.trexsEnPantalla++;
}
```

Verifica que haya 2 trexs por pantalla durante todo el juego.

```
if (this.vidas == 0) {
    this.princesa = null;
}

if (this.princesa != null) {
    princesa.dibujar(entorno);

    // Manejar entradas del jugador
    if (entorno.estaPresionada('d') || entorno.estaPresionada(entorno.TECLA_DERECHA)) {
        princesa.moverDerecha(entorno);
        inmunidad = false;
    }
    if (entorno.estaPresionada('a') || entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
        princesa.moverIzquierda();
        inmunidad = false;
    }
    if (entorno.estaPresionada('w') || entorno.estaPresionada('x')) {
        princesa.saltar();
        inmunidad = false;
    }

    // Actualizar posición y estado de la princesa
    princesa.mover(pisos); // Colisionar con todos los pisos
```

Dibujar e interactuar con la princesa.

```
   // Dibujar y mover balas si están activas
   if (entorno.sePresiono(entorno.TECLA_ESPACIO) || entorno.sePresiono('c') && this.balas == null) {
       this.balas = princesa.disparar();
   }

   // dibuja las balas
   if (this.balas != null) {
       this.balas.dibujar(entorno);
       this.balas.mover();

       // detecta colision de la bala con los trex
       for (int i = 0; i < trexs.length; i++) {
           for (int j = 0; j < trexs[i].length; j++) {
               if (trexs[i][j] != null && trexs[i][j].colisionConBala(this.balas)) {
                   trexs[i][j] = null; // convierte en null al trex y a las balas
                   this.balas = null;
                   this.trexsEnPantalla -= 1;
                   this.trexsEliminados += 1; // Incrementar contador de trexs eliminados
                   this.puntos += 2;
               }
           }

       }
       // bala desaparece si choca con los bordes
       if (this.balas != null && (this.balas.getX() > entorno.ancho() || this.balas.getX() < 0)) {
           this.balas = null;
       }
   }
}
```

```
    if (this.princesa == null) {
        // Marcar que perdiste el juego
        this.imag = Herramientas.cargarImagen("fondonegro.png");
        entorno.dibujarImagen(imag, entorno.ancho() / 2, entorno.alto() / 2, 0, 1);
        entorno.cambiarFont("Calibri", 50, java.awt.Color.red);
        entorno.escribirTexto("PERDISTE", entorno.ancho() / 2 - 110, entorno.alto() / 2 - 50);
        entorno.cambiarFont("Calibri", 40, java.awt.Color.white);
        entorno.escribirTexto("TREXS ELIMINADOS:" + this.trexsEliminados, entorno.ancho() / 2 - 250,
                entorno.alto() / 2 + 100);
        entorno.escribirTexto("PUNTOS:" + this.puntos, entorno.ancho() / 2 - 250, entorno.alto() / 2 + 140);
        entorno.escribirTexto("JUEGO TERMINADO", entorno.ancho() / 2 - 180, entorno.alto() / 2);
        entorno.escribirTexto("Presiona [r] para volver a jugar", entorno.ancho() / 2 - 250,
                entorno.alto() / 2 + 50);
        // Detectar si se presiona la tecla 'r' para reiniciar el juego
    }
    if (this.princesa != null && princesa.getY() < 0) {
        // Marcar que ganaste el juego
        princesa.setY(-10000);
        this.imag = Herramientas.cargarImagen("fondonegro.png");
        entorno.dibujarImagen(imag, entorno.ancho() / 2, entorno.alto() / 2, 0, 1);
        entorno.cambiarFont("Calibri", 50, java.awt.Color.green);
        entorno.escribirTexto("GANASTE", entorno.ancho() / 2 - 110, entorno.alto() / 2 - 50);
        entorno.cambiarFont("Calibri", 40, java.awt.Color.white);
        entorno.escribirTexto("TREXS ELIMINADOS:" + this.trexsEliminados, entorno.ancho() / 2 - 250,
                entorno.alto() / 2 + 100);
        entorno.escribirTexto("PUNTOS:" + this.puntos, entorno.ancho() / 2 - 250, entorno.alto() / 2 + 140);
        entorno.escribirTexto("JUEGO TERMINADO", entorno.ancho() / 2 - 180, entorno.alto() / 2);
        entorno.escribirTexto("Presiona [r] para volver a jugar", entorno.ancho() / 2 - 250,
                entorno.alto() / 2 + 50);
        // Detectar si se presiona la tecla 'r' para reiniciar el juego
    }

    // boton de reiniciar para cualquier momento del juego
    if (entorno.sePresiono('r')) {
        reiniciar();
    }
}
```

Conclusiones:

Para concluir con este informe, podemos afirmar que este proyecto nos proporcionó diversas perspectivas acerca de cómo abordar el desarrollo de un juego en java. Aprendimos a manejar de manera más eficiente los objetos y otras herramientas que nos brinda este lenguaje.

Desde el principio hasta el final del proyecto, nos enfrentamos a diversos desafíos que nos permitieron crecer y aprender colectivamente. De esta manera, podríamos decir que el proyecto no solo mejoró nuestros conocimientos de programación, sino que también fortaleció nuestras habilidades de trabajo en equipo.