PROGRAMACIÓN 1 - 2do Cuatrimestre 2024

Trabajo práctico grupal

“Al rescate de los gnomos”

Integrantes:

Anabella Noemi Santillan 45001454 anabellasantillan023@gmail.com

Rodrigo Aranda 46580335 ra1867332@gmail.com

Andrea Galvan 42315390 andreagalvan0130@gmail.com

Comisión 05

Repositorio github

Introducción

El siguiente trabajo práctico grupal presentado se trata acerca de un videojuego desarrollado en JAVA. En donde nuestro personaje “PEP” tiene que salvar 10 gnomos para ganar la partida. Si PEP pierde 15 gnomos ya sea porque son aniquilados por las tortugas o porque caen al vacío, el juego termina.

Descripción del trabajo práctico

Una explicación general de cada clase implementada, describiendo las variables de instancia y dando una breve descripción de cada método.

También deben incluirse los problemas encontrados, las decisiones que tomaron para poder resolverlos y las soluciones encontradas

- La clase **Bombas** modela el comportamiento de una bomba. Sus variables de instancia son: “x”,”y”, ”moviendoDerecha”,”imagen” y “velocidad”.
    1. x: Un valor double que representa la posición horizontal de la bomba.
    2. y: Un valor double que representa la posición vertical de la bomba.
    3. velocidad: Un valor double que define la velocidad a la cual se mueve la bomba.
    4. moviendoDerecha: Un valor boolean que indica la dirección del movimiento de la bomba.
    5. Imagen: Un objeto de tipo Image que almacena la imagen de la bomba.

    El método “mover” controla tanto su dirección como su velocidad; el método “dibujar” se encarga de mostrar la bomba en la pantalla, y los métodos “getX” y “getY” permiten obtener su posición para que otros elementos del juego puedan interactuar con ella o detectar colisiones.

    En esta clase no se encontró ningún problema porque es casi igual que en la clase Fireball.

- La clase **Casa** representa una casa. Sus variables de instancia son: “x”,”y”,”ancho” y “alto”.

    1. x: Un valor double que representa la posición horizontal de la casa.
    2. y: Un valor double que representa la posición vertical de la casa.
    3. ancho: Un valor int que indica el ancho de la casa.
    4. alto: Un valor int que indica el alto de la casa.

    El método “dibujar” se visualiza en la pantalla del juego utilizando una imagen específica, y los métodos “getX”,”getY”,”getAncho” y “getAlto” permite a otras clases obtener la ubicación y tamaño de la casa.

    En esta clase no se encontró ningún problema.

- La clase **Fireball** modela el comportamiento de una bola de fuego. Sus variables de instancia son: “x”,”y” y “moviendoDerecha”.

    1. x: Un valor double que representa la posición horizontal de la bola de fuego.
    2. y: Un valor double que representa la posición vertical de la bola de fuego.
    3. velocidad: Un valor double que define la velocidad a la cual se mueve la bola de fuego.
    4. moviendoDerecha: Un valor boolean que indica la dirección del movimiento de la bola de fuego.
    5. ancho: Un valor int que indica el ancho de la bola de fuego.
    6. alto: Un valor int que indica el alto de la bola de fuego.

    El método “mover” controla tanto su dirección como su velocidad; el método “chocoConBomba” detecta la colisión con las bombas; el método “dibujar” se visualiza

un rectángulo de color naranja en la pantalla del juego, y los métodos "getX","getY","getAncho" y "getAlto" permite a otras clases obtener la ubicación y tamaño de la bola de fuego.

En esta clase el mini problema fue dibujar a la bola de fuego. No encontrábamos una imagen para esa bola, así que tuvimos que dibujarlo como rectángulo.

- La clase **Gnomos** representa un objeto de tipo gnomo dentro del juego . Sus variables de instancia son:

  X: variable de tipo double que representa la posición x del gnomo

  Y: variable de tipo double que representa la posición y del gnomo

  Ancho : variable de tipo entera que representa el ancho del gnomo

  Alto : variable de tipo entera que representa la altura del gnomo

  Puntos: variable de tipo entera que indica la puntuación inicial del gnomo

  factorDesplazamiento: variable de tipo double que indica el movimiento del gnomo

  Vivo : variable de tipo booleana que indica si el gnomo está vivo o muerto

  Imagen : variable en que se define una imagen , la cual es la que le da el aspecto al gnomo

  Gravedad: variable de tipo entera que representa la fuerza de gravedad que afecta al gnomo

  Cayendo : variable de tipo booleana que indica el estado de caída

  moviendoseALaDerecha: variable de tipo booleana que indica la dirección del movimiento , al principio la dirección es aleatorio

estaEnIslaSuperior: variable de tipo booleana que indica que el gnomo está en una isla superior

Constructores:

Un constructor inicializa el gnomo con coordenadas, dimensiones y carga de imagen.

Un segundo constructor está presente pero no implementado.

Métodos:

mover(): Cambia la posición del gnomo según su dirección de movimiento.

cambiarDireccion(): Invierte la dirección de movimiento.

abajo(): Mueve al gnomo hacia abajo.

llegoFondo(Entorno e): Verifica si el gnomo ha llegado al fondo del entorno.

estaSobreAlgunaIsla(Isla[] islas): Comprueba si el gnomo está sobre alguna isla.

cayoSobreUnaIsla(Isla[] islas): Maneja la lógica de caída sobre una isla.

chocoAlHeroe(Pep h): Verifica colisión con un héroe.

fueChocadoPorUnEnemigo(Tortugas[] tortuga): Comprueba colisión con enemigos.

chocoFireball(Fireball[] fireballs): Verifica colisión con bolas de fuego.

caer(): Aplica gravedad al gnomo.

chocoConBomba(Bombas bomba): Verifica colisión con bombas.

chocoIzquierda(Entorno e) y chocoDerecha(Entorno e): Verifican si el gnomo choca con los bordes del entorno.

dibujar(Entorno entorno): Dibuja la imagen del gnomo en el entorno si está vivo.

En esta clase no se encontro ningun problema

- La clase **Isla** representa una isla en un entorno gráfico con las siguientes variables de instancia y métodos:

  Variables de instancia:

  X: variable de tipo double que representa la posición x de la isla

  Y: variable de tipo double que representa la posición y de la isla

  Ancho : variable de tipo entera que representa el ancho de la isla

  Alto : variable de tipo entera que representa la altura de la isla

  Imagen : variable en que se define una imagen , la cual es la que le da el aspecto de las isla

  tipo: Un valor del tipo enumerado IslaTipo que indica qué tipo de isla es (Casa de Gnomos, Superior o Inferior).

  Métodos:

  Constructor: Inicializa una nueva isla con valores para la posición, tamaño, imagen y tipo.

  dibujar(Entorno e): Dibuja un rectángulo representando la isla y su imagen en el entorno gráfico.

  getX(), getY(), getAncho(), getAlto(): Métodos para obtener las coordenadas y el tamaño de la isla.

  esIslaInferior(): Devuelve true si la isla es del tipo Inferior.

  esIslaDeGnomos(): Devuelve true si la isla es del tipo Casa de Gnomos.

  Enumerado IslaTipo:

  Define tres tipos de islas:

  CasaGnomos: Isla tipo Casa de Gnomos.

Superior: Isla tipo Superior.

Inferior: Isla tipo Inferior.

En esta clase no tuvimos problemas

- La clase **Tortugas** representa una tortuga que se mueve por un entorno 2D . Sus variables de instancia son :

  X: variable de tipo double que representa la posición x de la tortuga

  Y:variable de tipo double que representa la posición y de la tortuga

  Ancho: variable de tipo entera que representa el ancho de la tortuga

  Alto: variable de tipo entera que representa la altura de la tortuga

  Puntos: variable de tipo entera que indica la puntuación inicial de la tortuga

  factorDesplazamiento: variable de tipo double que indica el movimiento de la tortuga

  Vivo: variable de tipo booleana que indica si la tortuga está viva o muerta

  Imágenes: imagenDerecha, imagenIzquierda (variables para representar la tortuga moviéndose a la derecha o izquierda).

  moviendoseALaIzquierda: variable de tipo booleana que indica si la tortuga se mueve hacia la izquierda

  ultimoDisparo: variable que indica el tiempo del ultimo disparo

  Métodos principales:

  Mover y cambiar dirección:

  mover(): Mueve la tortuga en la dirección en la que está mirando.

  cambiarDireccion(): Cambia la dirección de movimiento (izquierda/derecha).

Interacciones con el entorno:

llegoFondo(Entorno e): Verifica si la tortuga ha llegado al fondo del entorno.

estaSobreAlgunaIsla(Isla[] islas): Comprueba si la tortuga está sobre una isla.

cambiarDireccionSiTocaIsla(Isla[] islas): Cambia la dirección si toca una isla.

chocoAlHeroe(Pep h): Detecta si la tortuga choca con un héroe.

chocoConFireball(Fireball[] fireballs): Verifica si la tortuga choca con una bola de fuego.

chocoConGnomo(Gnomos[] gnomos): Detecta colisiones con gnomos.

caer(): Hace que la tortuga caiga (aplica gravedad).

chocoIzquierda(Entorno e) / chocoDerecha(Entorno e): Verifica si la tortuga choca con los bordes del entorno.

moverEnIsla(Isla[] islas): Hace que la tortuga se mueva sobre las islas y cambie de dirección al llegar al borde.

Disparos y tiempo:

puedeDisparar(long tiempoActual, long intervaloDisparo): Verifica si ha pasado suficiente tiempo para disparar de nuevo.

Visualización:

dibujar(Entorno entorno): Dibuja la tortuga en el entorno según su estado y dirección.

- La clase **Juego** representa la interfaz del juego.

  Métodos:

  Juego(): carga las imágenes, define las posiciones y tipos de islas. Crea los objetos principales: la isla, gnomos, pep, tortugas, fuego, bombas, sistema de puntuación.

tick(): Este método se ejecuta en cada ciclo del juego y se encarga de actualizar el estado del juego. Gestiona el movimiento y las acciones de los personajes, la lógica de las islas, las colisiones entre objetos, la generación de nuevos elementos del juego, controla los disparos de bolas de fuego y el movimiento.

movimientoEstado(): registra el estado de las teclas presionadas (A, D, W), es decir, de almacenar la información sobre qué teclas están siendo presionadas en un momento dado. Este método es utilizando en conjunto a **saltar**

- La clase **Score** representa los puntos obtenidos en el juego. Sus **variables de instancia** son:

  puntos: variable del tipo entero que almacena el total de puntos

  gnomosPerdidos: variable del tipo entero que almacena la cantidad de gnomos perdidos (es decir, que fueron eliminados por tortugas o que cayeron al vacío)

  gnomosSalvados: variable del tipo entero que almacena la cantidad de gnomos salvados por Pep

  enemigosEliminados: variable del tipo entero que almacena la cantidad de enemigos (tortugas) que fueron eliminadas por Pep

  tiempoInicio: variable de tipo long que almacena el tiempo de juego en milisegundos.

  Constructor

  El constructor Score() inicializa las variables de instancia en sus valores iniciales, es decir, cero.

Métodos

sumarPuntos(): se suman la cantidad de puntos obtenidos a la variable puntos.

sumarGnomosPerdidos(): se suman la cantidad gnomos perdidos a la variable gnomosPerdidos

sumarGnomosSalvados(): se suman la cantidad gnomos salvados a la variable gnomosSalvados

cantEnemigosEliminados(): se suman la cantidad de enemigos eliminados a la variable enemigosEliminados

getScore(): devuelve la cantidad de puntos obtenidos en la variable puntos.

Interacciones con el entorno

El método **dibujar** muestra en pantalla los puntos totales obtenidos, los gnomos perdidos, los gnomos salvados, los enemigos eliminados y por último la cantidad de **segundos** transcurridos desde el comienzo del juego.

e.cambiarFont(): Cambia la fuente del texto a ser dibujado.

e.escribirTexto(): Escribe el texto que va a ser mostrado en pantalla.

long tiempoActual = System.currentTimeMillis(): obtiene el tiempo actual en milisegundos, y lo guarda en la variable tiempoActual

int tiempoTranscurridoSegundos = (int) ((tiempoActual - tiempoInicio) / 1000): Resta tiempoInicio (almacenado al crear el objeto Score) de tiempoActual, lo que da el tiempo transcurrido en milisegundos desde que el objeto fue creado. Divide el resultado entre 1000 para convertir milisegundos a segundos Convierte el valor resultante a un tipo int, ya que la operación entre tiempoActual y tiempoInicio produce un valor long, y queremos almacenar el tiempo en segundos como int.

- La clase **Pep** representa un objeto de tipo PEP dentro del juego . Sus **variables de instancia** son:

  x: variable de tipo double que representa la posición x de Pep

  y: variable de tipo double que representa la posición x de Pep

  ancho: variable de tipo entera que representa el ancho de Pep.

  alto: variable de tipo entera que representa el alto de Pep

  saltando: variable de tipo boolean que define el estado del salto de Pep

anguloFireball: variable de tipo entero que establece el ángulo de lanzamiento de las bolas de fuego.

ArrayList<Fireball> fireballs: lista de bolas de fuego lanzadas por Pep

factorDesplazamiento: variable de tipo double que establece el movimiento horizontal de Pep

impulso: variable de tipo double que establece el impulso del salto de Pep

limitedeSaltoY: variable de tipo double que establece el límite del salto de Pep en el eje Y.

derecha: variable de tipo boolean que indica si Pep está mirando a la derecha.

gravedad: variable de tipo double que establece la gravedad que afecta a Pep.

imagenDerecha: imagen de Pep mirando a la derecha.

imagenIzquierda: imagen de Pep mirando a la izquierda.

vivo: variable de tipo boolean que establece el estado de vida de Pep

cayendo: variable de tipo boolean que indica el estado de caída de Pep

terminarSalto: variable de tipo boolean que controla si el salto de Pep terminó.

inmortal: variable de tipo boolean que establece si Pep está en estado inmortal.

tiempoInmortalInicio: variable de tipo long que almacena el tiempo de inicio de inmortalidad.

escudo: variable de tipo boolean que establece si Pep tiene un escudo activo.

Métodos principales

activarInmortalidad(): activa el estado de Inmortalidad (no recibe daño) de Pep y registrar el tiempo de inicio.

actualizarInmortalidad(): Actualiza el estado de inmortalidad y se desactiva el mismo luego de 4 segundos.

esInmortal(): retorna si Pep se encuentra en estado Inmortal.

moverIzquierda(): mueve a Pep a la izquierda.

moverDerecha(): mueve a Pep a la derecha.

llegoFondo(): verifica si Pep llegó al fondo del entorno

saltar(): Pep realiza el movimiento de salto.

caer(): hace que Pep caiga.

estaCayendo(): retorna si Pep se encuentra en caída.

dejaDeCaer(): detiene la caída de Pep.

mirarIzquierda(): hace que Pep mire a la izquierda.

mirarDerecha(): hace que Pep mire a la derecha.

estaSaltando(): retorna si Pep está saltando.

moverSalto(): controla el movimiento durante el salto. En el mismo, se aumenta la altura en el salto disminuyendo el valor de Y por la cantidad combinada de gravedad e impulso.

estaSobreAlgunaIsla(): verifica si Pep está sobre alguna isla.

estaVivo(): retorna si Pep está vivo.

lanzarBola(): Pep lanza una bola de fuego. En el mismo, se define el ángulo y se añade la nueva bola de fuego a la lista.

actualizarFireballs(): se actualiza la lista de fireballs y se mueven.

chocoDerecha(): verifica si Pep ha tocado el borde derecho de la pantalla.

chocoIzquierda(): verifica si Pep ha tocado el borde derecho de la izquierdo.

chocoAlgunEnemigo(): verifica si Pep colisionó con algún enemigo (tortugas)

chocoConBomba(): verifica si Pep colisionó con alguna bomba

mirandoDerecha(): retorna si Pep está mirando a la derecha. Se utiliza para el movimiento de la bola de fuego.

morir(): Cambia el estado de Pep a muerto.

Interacciones con el entorno:

dibujar(): dibuja a Pep en el entorno.

dibujarFireballs(): Dibuja cada bola de fuego que es disparada por Pep.

Implementación

Una sección de implementación donde se incluya el código fuente correctamente formateado y comentado, si corresponde.

Requerimientos obligatorios

1. Al comenzar el juego, deben mostrarse entre 4 y 6 filas de islas flotantes y la isla con la casa de los gnomos en el sector central superior de la pantalla. Las dos filas superiores de islas flotantes deben ocupar solo un sector de la pantalla (aproximadamente la mitad) y deben estar intercaladas con sectores vacíos. Las islas en las filas inferiores también deben intercalarse con espacios vacíos ocupando todo el ancho de la pantalla. Pep debe estar ubicado sobre una de las islas en la fila inferior.

Clase isla

```
public void dibujar(Entorno e) {
        // Dibuja la imagen de la isla en la posición especificada
        e.dibujarImagen(imagen, x, y, 0, 1.0);
}
// Método para obtener la coordenada x de la isla
```

```
    public double getX() {
        return this.x; }
    public double getY() { // Método para obtener la coordenada y de la isla
        return this.y; }
    public int getAncho() { // Método para obtener el ancho de la isla
        return this.ancho; }
    public double getAlto() { // Método para obtener el alto de la isla
        return this.alto; }
    public boolean esIslaInferior() {       // Método para verificar si la isla es del tipo Inferior
        return tipo == IslaTipo.Inferior; }
    public enum IslaTipo {          // Enumeración para definir el IslaTipo
        Superior,                   // Tipo de isla en la parte superior del entorno
        Inferior;                   // Tipo de isla en la parte inferior del entorno
    }
```

Clase del juego principal

```
islas = new Isla[15];
int[] xxPos = {entorno.ancho() / 2,entorno.ancho() - 310,entorno.ancho() - 495,entorno.ancho() - 580,entorno.ancho() / 2,entorno.ancho() - 220,entorno.ancho() - 680,entorno.ancho() - 495,entorno.ancho() - 310,entorno.ancho() - 130,entorno.ancho() - 770,entorno.ancho() - 580,entorno.ancho() / 2,entorno.ancho() - 220,entorno.ancho() - 35}  // Define un array con las posiciones x de cada isla en el entorno, calculadas en función del ancho del entorno
        int[] yyPos = {ALTO_ESCENARIO / 6,(ALTO_ESCENARIO / 6) * 2,(ALTO_ESCENARIO / 6) * 2,(ALTO_ESCENARIO / 6) * 3,(ALTO_ESCENARIO / 6) *
```

3,(*ALTO_ESCENARIO* / 6) * 3,(*ALTO_ESCENARIO* / 6) * 4,(*ALTO_ESCENARIO* / 6) * 4,(*ALTO_ESCENARIO* / 6) * 4,(*ALTO_ESCENARIO* / 6) * 5,(*ALTO_ESCENARIO* / 6) * 5,(*ALTO_ESCENARIO* / 6) * 5,(*ALTO_ESCENARIO* / 6) * 5}; // Define un array con las posiciones y de cada isla en el entorno, divididas en secciones del alto total del escenario

IslaTipo[] tipo = {IslaTipo.*CasaGnomos*,IslaTipo.*Superior*,IslaTipo.*Superior*,IslaTipo.*Superior*,IslaTipo.*Superior*,IslaTipo.*Superior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*,IslaTipo.*Inferior*}; // Define un array de tipos IslaTipo para cada isla, asignando a cada posición un tipo (CasaGnomos, Superior o Inferior)

for (int p = 0; p < islas.length; p++) {

islas[p] = new Isla(xxPos[p], yyPos[p],unAncho,unAlto,imagenDeVigas,tipo[p]);

// Crea una nueva isla con las coordenadas x e y de xxPos y yyPos, con un ancho y alto dados,la imagen de vigas y asignando el tipo correspondiente

}

2. Mientras esté sobre las islas flotantes, Pep puede moverse usando las teclas izquierda y derecha, o las teclas ’a’ y ’d’, y puede saltar con la tecla flecha arriba o ’w’. Si no se presiona ninguna tecla, Pep debe permanecer parado sobre la isla. Si Pep no está sobre una isla flotante o no está saltando, debe caer hasta hacer colisión con una isla o caer al precipicio.

Clase Pep

public void dibujar(Entorno e) { // Dibuja a Pep en el entorno.

```
            if (vivo)
                    e.dibujarImagen(derecha ? imagenDerecha : imagenIzquierda, x, y, 0,
                                    0.3);


            else {
                            e.dibujarImagen(derecha ? imagenDerecha : imagenIzquierda, x, y,
0, 0.3);
            }
        }
        public void moverIzquierda(Entorno e) {      // Mueve a Pep hacia la izquierda.
            x -= factorDesplazamiento;
        }
        public void moverDerecha(Entorno e) {       // Mueve a Pep hacia la derecha.
            x += factorDesplazamiento;
        }
        public void saltar() {   // Hace que Pep salte.
            if (vivo && !saltando && !terminarSalto && !estaCayendo()) {
            saltando = true;
            limiteDeSaltoY = getY() - 120;                  // Define el límite del salto.
            }
        }
        public void caer() {    // Hace que Pep caiga.
            cayendo = true;
```

```
            if (vivo) {
                y += gravedad;                          // Aplica la gravedad en la caída.
            }
    }
    public boolean estaCayendo() {          // Retorna si Pep está cayendo.
            return cayendo;
    }
    public void dejarDeCaer() {   // Detiene la caída de Pep.
            cayendo = false;
    }
    public boolean llegoFondo(Entorno e) {        // Verifica si Pep llegó al fondo del entorno.
            return y > e.alto();
    }
    public void mirarIzquierda() {          // Hace que Pep mire a la izquierda.
            this.derecha = false;
    }
    public void mirarDerecha() { // Hace que Pep mire a la derecha.
            this.derecha = true;
    }
    public boolean estaSaltando() {         // Retorna a Pep si está saltando.
            return saltando || terminarSalto;
    }
```

```
public void moverSalto(MovimientoEstado movimientoSalto)        {        // Controla el
movimiento durante el salto.
                if (vivo && saltando) {
                        if (getY() > limiteDeSaltoY)
                        {
                                y -= gravedad + impulso;                                // Aumenta la
altura en el salto.
                                x += movimientoSalto.getNumVal() * factorDesplazamiento * 0.5;
                        }
                        else
                        {
                                saltando = false;
                                terminarSalto = true;
                        }
                }
                if (estaCayendo() && terminarSalto) {
                        x += movimientoSalto.getNumVal() * factorDesplazamiento * 0.5;
                }
                else {
                        terminarSalto = false;
                }
        }
```

```
        public boolean estaSobreAlgunaIsla(Isla[] islas) {    // Verifica si Pep está sobre alguna
isla.
                for (int z = 0; z < islas.length; z++) {
                        if ((x + ancho / 2 >= islas[z].getX() - islas[z].getAncho() / 2)
                                        && (x - +ancho / 2 <= islas[z].getX() + islas[z].getAncho()
                                                        / 2)
                                        && (y + alto / 2 <= islas[z].getY() + islas[z].getAlto()
                                                        / 2)
                                        && (y + alto / 2 >= islas[z].getY() - islas[z].getAlto()
                                                        / 2)) {
                                return true;
                        }
                }
                return false;
        }
        }
   }
public boolean aterrizaSobreIsla(Isla[] islas) {          // Verifica si Pep aterrizó sobre alguna isla
   for (Isla isla : islas) {
      if ((x + ancho / 2 >= isla.getX() - isla.getAncho() / 2) &&
         (x - ancho / 2 <= isla.getX() + isla.getAncho() / 2) &&
         (y + alto / 2 >= isla.getY() - isla.getAlto() / 2) &&
         (y + alto / 2 <= isla.getY() + isla.getAlto() / 2) &&
```

```
            cayendo) {
            y = isla.getY() - alto / 2; // Asegúrate de que Pep no atraviese la isla
            cayendo = false; // Deja de caer
            saltando = false; // Deja de saltar
            return true;
        }
    }
    return false;
}
public boolean mirandoDerecha() {    // Retorna si Pep está mirando hacia la derecha.
            return this.derecha;
}
}
```

Clase del juego principal

```
if (pep.estaVivo()) {
                                    pep.dibujar(entorno);
                                    if (!pep.estaSobreAlgunaIsla(islas)) {
                                                pep.caer();
} else {
                                                pep.dejarDeCaer();
                                                if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA)
|| entorno.estaPresionada('a')) {
```

```
                movimiento = MovimientoEstado.Izquierda;
                pep.moverIzquierda(entorno);
                pep.mirarIzquierda();
            }
            if (entorno.estaPresionada(entorno.TECLA_DERECHA) ||
entorno.estaPresionada('d')) {
                movimiento = MovimientoEstado.Derecha;
                pep.moverDerecha(entorno);
                pep.mirarDerecha();
            }
            if (entorno.estaPresionada(entorno.TECLA_ARRIBA) ||
entorno.estaPresionada(entorno.TECLA_ESPACIO)) {
                pep.saltar();
            }                    }
        if (pep.estaSaltando()) {
            pep.moverSalto(movimiento);
        }}
```

3. Pep tiene el poder del sol en sus manos, lo que le permite arrojar pequeñas bolas de fuego que van a ras del piso para acabar con sus enemigos. Para lanzar un disparo se debe presionar la tecla ’c’ o el botón izquierdo del mouse, la bola de fuego se moverá hacia la dirección donde se movió Pep la última vez.

Clase Fireball

```
        this.x = x;
        this.y = y;
        this.moviendoDerecha = moviendoDerecha;
        this.velocidad = 5; // Ajusta la velocidad de la bola de fuego
    }
    public void mover() {     // Método para mover la bola de fuego
        if (moviendoDerecha) {
            this.x += velocidad; // Mueve a la derecha
        } else {
            this.x -= velocidad; // Mueve a la izquierda
        }}
    public void dibujar(Entorno entorno) {       // Método para dibujar la bola de fuego en pantalla
        entorno.dibujarRectangulo(this.x, this.y, 20, 20, 0, Color.ORANGE);
    }}
```

Clase de juego principal

```
// Lógica para lanzar una nueva fireball si hay un espacio disponible
                else if (entorno.sePresiono('C')) { // Cambia 'C' por la tecla que desees
                    if (tiempoActual - ultimoDisparo >= 5000) {   //Se dispara  y tiene que esperar 5
segundos para volverlo a hacer
                        double xInicial = pep.getX(); // Centro en X
                        double yInicial = pep.getY() + 10; // Centro en Y
                        boolean moviendoDerecha = pep.mirandoDerecha();
                        fireballs[i] = new Fireball(xInicial, yInicial,moviendoDerecha);
```

```
                ultimoDisparo = tiempoActual; // Actualiza el tiempo del último disparo
                break; // Sal del bucle después de lanzar la fireball
            }
        }
```

4. Los gnomos salen de su casa ubicada en la isla más alta y se mueven hacia la izquierda o derecha. Cuando caen a una isla de un nivel más bajo, cambian su dirección de manera aleatoria. Debe haber entre 2 y 4 gnomos en pantalla, reponiéndose cada cierto tiempo ya que serán rescatados por Pep, aniquilados por las tortugas, o caerán al precipicio.

5. Un gnomo es rescatado cuando colisiona con Pep. Pero sólo podría rescatarlo en las islas inferiores (en las primeras dos filas desde abajo hacia arriba). Si un gnomo cae al precipicio o es colisionado por una tortuga, se considera un gnomo perdido. En ambos casos, se debe eliminar la instancia del gnomo.

Clase Gnomos

```
public void mover() {
        if (moviendoseALaDerecha) {   //si moviendoseALaDerecha = true, se desplazará hacia la derecha.
                x -= factorDesplazamiento;  }
        else {
                x += factorDesplazamiento;   //caso contrario, hacia la izquierda.  }
}
public void cambiarDireccion() {
```

```
  moviendoseALaDerecha = !moviendoseALaDerecha;  }  // Invierte la dirección
public boolean estaSobreAlgunaIsla(Isla[] islas) {
        for (int z = 0; z < islas.length; z++) {
            if ((x + ancho / 2 >= islas[z].getX() - islas[z].getAncho() / 2) //Verifica que
el borde izquierdo del gnomo esté a la derecha o sobre el borde izquierdo de la isla
                && (x - +ancho / 2 <= islas[z].getX() + islas[z].getAncho()
//Verifica que el borde derecho del gnomo esté a la izquierda o sobre el borde derecho de la isla
                / 2)
                && (y + alto / 2 <= islas[z].getY() + islas[z].getAlto() //
Verifica que la parte inferior del gnomo esté arriba o sobre el borde inferior de la isla
                / 2)
                && (y + alto / 2 >= islas[z].getY() - islas[z].getAlto()
//Verifica que la parte superior del gnomo esté abajo o sobre el borde superior de la isla
                / 2)) {
            estaEnIslaInferior = islas[z].esIslaInferior();
                return true;  }}
        return false;  }
public boolean chocoAlHeroe(Pep h) {
        if ((x >= h.getX() - h.getAncho() / 2) // Verifica que el borde izquierdo del gnomo
esté a la derecha o sobre el borde izquierdo del héroe
                && (x <= h.getX() + h.getAncho() / 2) // Verifica que el borde
derecho del gnomo esté a la izquierda o sobre el borde derecho del héroe
```

```
                && (y <= h.getY() + h.getAlto() / 2) // Verifica que la parte
inferior del gnomo esté arriba o sobre el borde inferior del héroe
                && (y >= h.getY() - h.getAlto() / 2)) { // Verifica que la parte
superior del gnomo esté abajo o sobre el borde superior del héroe
            return estaEnIslaInferior;  }   //Verifica si pep lo rescató en las islas
inferiores
        return false;  }
public void morir() {
        vivo = false;  }
public boolean fueChocadoPorUnEnemigo(Tortugas[] tortuga) {    //Verifica si hubo colisión con
la tortuga
        for (Tortugas h : tortuga) {
            if (this.estaVivo() && h != null
                    && (x >= h.getX() - h.getAncho() / 2)
                    && (x <= h.getX() + h.getAncho() / 2)
                    && (y <= h.getY() + h.getAlto() / 2)
                    && (y >= h.getY() - h.getAlto() / 2)) {
                return true;  }}
        return false;  }
    public void caer() {
        if (vivo) {
            y += gravedad;
            cayendo = true;  }}
```

```
public boolean ultimaIslaEsInferior() {
        return estaEnIslaInferior;  }
```

Clase de juego principal

```
gnomo = new Gnomos[4]; // Crea un arreglo de gnomos con capacidad para 4
        for (int i = 0;i < gnomo.length;i++) {
            gnomo[i] = new Gnomos(entorno.ancho()/2,(entorno.alto()/6)-40,10,10);
}
for (int i = 0; i < gnomo.length; i++) {
            if (gnomo[i] != null) {  // Verifica si el gnomo existe
                gnomo[i].dibujar(entorno);  // Dibuja el gnomo
                if (!gnomo[i].estaSobreAlgunaIsla(islas)) {
                    gnomo[i].caer();  // Si no está sobre una isla, el gnomo cae.
                    if (gnomo[i].llegoFondo(entorno)) {
                        matarGnomo(i);
}} else {
                    if (gnomo[i] != null && gnomo[i].estaVivo()) {  // Si está
vivo, puede moverse y chequear colisiones
                        gnomo[i].cayoSobreUnaIsla(islas);
                        gnomo[i].mover();
if (gnomo[i] != null && gnomo[i].fueChocadoPorUnEnemigo(tortuga)) {
                            matarGnomo(i);  // Mata al gnomo.
    if (gnomo [i] != null && gnomo[i].chocoAlHeroe(pep)) {
matarGnomo(i);  }    // Mata al gnomo si choca con el héroe, para rescatarlo.
```

if (tiempoActual - tiempoUltimoGnomoSpawneado >= *INTERVALO_SPAWN_GNOMO*) { // Genera un nuevo gnomo en el centro si pasó el intervalo de tiempo de aparición.

gnomo[i] = new Gnomos(entorno.ancho()/2,(entorno.alto()/6)-40,10,10);

tiempoUltimoGnomoSpawneado = tiempoActual; } // Actualiza el último tiempo de aparición.

6. Las tortugas aparecen cayendo desde el borde superior de la pantalla en una posición aleatoria pero no sobre la isla de la casa de los gnomos. Cuando una tortuga llega a una isla flotante empieza a moverse de izquierda a derecha (o al revés) hasta el borde de la isla, luego debe cambiar de dirección para permanecer en la misma. Si una tortuga hace coalición con Pep o con los gnomos los aniquila ya que son muy venenosas.

Clase Tortugas

```
public void mover() {
	if (moviendoseALaIzquierda) {
		x -= factorDesplazamiento;
	}
	else {
		x += factorDesplazamiento;
	}
}
public void cambiarDireccion() {
	moviendoseALaIzquierda = !moviendoseALaIzquierda; // Invierte la dirección
}
public boolean estaSobreAlgunaIsla(Isla[] islas) {
		for (int z = 0; z < islas.length; z++) {
```

```
                if ((x + ancho / 2 >= islas[z].getX() - islas[z].getAncho() / 2)
                        && (x - ancho / 2 <= islas[z].getX() + islas[z].getAncho()
                                / 2)
                        && (y + alto / 2 <= islas[z].getY() + islas[z].getAlto()
                                / 2)
                        && (y + alto / 2 >= islas[z].getY() - islas[z].getAlto()
                                / 2)) {
                    return true;
                }
            }
            return false;
        }
        public void cambiarDireccionSiTocaIsla(Isla[] islas) {
            for (int z = 0; z < islas.length; z++) {
                if ((x + ancho / 2 >= islas[z].getX() - islas[z].getAncho() / 2)
                        && (x - ancho / 2 <= islas[z].getX() + islas[z].getAncho()
                                / 2)
                        && (y + alto / 2 <= islas[z].getY() + islas[z].getAlto()
                                / 2)
                        && (y + alto / 2 >= islas[z].getY() - islas[z].getAlto()
                                / 2)) {
                    cambiarDireccion();
                }
```

```
                }
        }
public boolean chocoConFireball(Fireball[] fireballs) {
            for (Fireball fireball : fireballs) {
               if (fireball != null && this.estaVivo()) {
                  // Coordenadas de los bordes de la bola de fuego
                  double fireballXIzquierda = fireball.getX() - 10;  // 10 es la mitad del ancho de la
fireball
                  double fireballXDerecha = fireball.getX() + 10;
                  double fireballYSuperior = fireball.getY() - 10;   // 10 es la mitad de la altura de la
fireball
                  double fireballYInferior = fireball.getY() + 10;
                  // Coordenadas de los bordes de la tortuga
                  double tortugaXIzquierda = this.getX() - this.getAncho() / 2;
                  double tortugaXDerecha = this.getX() + this.getAncho() / 2;
                  double tortugaYSuperior = this.getY() - this.getAlto() / 2;
                  double tortugaYInferior = this.getY() + this.getAlto() / 2;
                  // Verifica si hay colisión
                  if (fireballXIzquierda <= tortugaXDerecha && fireballXDerecha >=
tortugaXIzquierda &&
                     fireballYInferior >= tortugaYSuperior && fireballYSuperior <=
tortugaYInferior) {
                     return true;  // Colisión detectada
```

```
                }
            }
        }
        return false;  // No hubo colisión
    }
    public boolean chocoConGnomo(Gnomos[] gnomos) {
        for (Gnomos gnomo : gnomos) {
            if (gnomo != null && gnomo.estaVivo()) {
                // Coordenadas de la tortuga
                double tortugaXMin = this.x - this.ancho / 2;
                double tortugaXMax = this.x + this.ancho / 2;
                double tortugaYMin = this.y - this.alto / 2;
                double tortugaYMax = this.y + this.alto / 2;
                // Coordenadas del gnomo
                double gnomoXMin = gnomo.getX() - gnomo.getAncho() / 2;
                double gnomoXMax = gnomo.getX() + gnomo.getAncho() / 2;
                double gnomoYMin = gnomo.getY() - gnomo.getAlto() / 2;
                double gnomoYMax = gnomo.getY() + gnomo.getAlto() / 2;
                // Comprobar si las dos cajas (tortuga y gnomo) se superponen
                boolean colisionX = (tortugaXMin <= gnomoXMax && tortugaXMax >=
gnomoXMin);
                boolean colisionY = (tortugaYMin <= gnomoYMax && tortugaYMax >=
gnomoYMin);
```

```
                if (colisionX && colisionY) {
                    return true;
                }
            }
        }
        return false;
    }
    public void caer() {
            // y += factorDesplazamiento+impulso*3/2;
            if (vivo) {
                    y += gravedad;
                    cayendo = true;
            }
    }
    public boolean chocoIzquierda(Entorno e) {
            return x - ancho / 2 <= 0;
    }
    public boolean chocoDerecha(Entorno e) {
            return x + ancho / 2 >= e.ancho();
    }
    public void moverEnIsla(Isla[] islas) {
        for (int z = 0; z < islas.length; z++) {
```

```
  if ((x + ancho / 2 >= islas[z].getX() - islas[z].getAncho() / 2)              // Verifica si la tortuga
está sobre esta isla
                                        && (x - ancho / 2 <= islas[z].getX() + islas[z].getAncho()
                                                / 2)
                                        && (y + alto / 2 >= islas[z].getY() - islas[z].getAlto()
                                                / 2)
                                        && (y - alto / 2 <= islas[z].getY() + islas[z].getAlto()
                                                / 2)) {
                // Si llega al borde derecho de la isla, cambia de dirección a la izquierda
                if (x - ancho / 2 <= islas[z].getX() - islas[z].getAncho() / 2) {
                        moviendoseALaIzquierda = false;
                }
                // Si llega al borde izquierda de la isla, cambia de dirección a la derecha
                else if (x + ancho / 2 >= islas[z].getX() + islas[z].getAncho() / 2) {
                        moviendoseALaIzquierda = true;
                }
            }
        }
    }
    public boolean izquierda() {
            return this.moviendoseALaIzquierda;
    }
```

Clase de juego principal

```
tortuga = new Tortugas[4];
                int[] xPos = {entorno.ancho() - 690, entorno.ancho() - 590, entorno.ancho() - 215,
entorno.ancho() - 105};
                int yPos = entorno.alto() - 700;
                for (int i = 0; i < tortuga.length; i++) {
                   tortuga[i] = new Tortugas(xPos[i], yPos,10,10);
                }
for (int j = 0; j < tortuga.length; j++) {
                   if (tortuga[j] != null) {
                       tortuga[j].dibujar(entorno);
                     if (!tortuga[j].estaSobreAlgunaIsla(islas)) {
                       tortuga[j].caer();
                     } else {
                       if(tortuga[j].estaVivo()) {
                       tortuga[j].moverEnIsla(islas);
                       tortuga[j].mover();
                       if (tortuga[j].chocoDerecha(entorno) || tortuga[j].chocoIzquierda(entorno))
{
                           tortuga[j].cambiarDireccion();
                       }
                       if (tortuga[j] != null && tortuga[j].chocoConFireball(fireballs)) {
```

```
                    tortuga[j] = null;
                    for (int k = 0;k < fireballs.length;k++) {
                        fireballs[k] = null;
                    }
                }
            }

            // Este `else` debe ir si la tortuga no está viva y debe verificar la colisión
        }
    }
    }
```

7. El juego se gana cuando Pep haya rescatado una cantidad determinada de gnomos a elecci´on del grupo. El juego se pierde cuando se pierde una cierta cantidad de gnomos, o bien cuando Pep cae al precipicio o es aniquilado por las tortugas.

Clase de juego principal

```
fondoWin = Herramientas.cargarImagen("GANADOR.jpeg");
if (!juegoTerminado) {
                    juegoTerminado = true;
                    GameOverImage = Herramientas.cargarImagen("GameOver.gif");
                        }
            else {
```

```
                        // Dibuja la imagen de "Game Over"
                        entorno.dibujarImagen(GameOverImage, entorno.ancho() / 2 ,
entorno.alto() / 2,0, 1.17);
                                        }
if (pep.chocoAlgunEnemigo(tortuga) && !pep.esInmortal()) {
                                pep.morir(entorno);
                                Herramientas.play("Super-Mario-Bros.wav");
                        }
                        if (pep.chocoConBomba(bomba) && !pep.esInmortal()) {
                                pep.morir(entorno);
                                Herramientas.play("Super-Mario-Bros.wav");
                                        }
if (gnomo[i] != null && bomba != null && gnomo[i].chocoConBomba(bomba[r])) {
                                                        puntuacion.sumarPuntos(-10);
                                                        puntuacion.sumarGnomosPerdidos(1);
                                                        matarGnomo(i);
                                                        bomba[r] = null;
                                                        puntosPerdidos++;
                                                        if (puntosPerdidos == 15) {
                                                                pep.morir(entorno);

Herramientas.play("Super-Mario-Bros.wav");
                                                        }
```

```
                                                    break;
                                                }
if (gnomo [i] != null && gnomo[i].chocoAlHeroe(pep)) {
                                                    puntuacion.sumarPuntos(30);  // Aumenta
los puntos al chocar con el héroe
                                                    puntuacion.sumarGnomosSalvados(1);
                                                    pep.activarInmortalidad();  // Activa
inmortalidad
                                                    matarGnomo(i);  // Matar y rescatar
ocasionan lo mismo por lo cual utilizamos el mismo metodo
                                                    puntos++;
                                                    if (puntos == 10) {
                                haGanado = true;  // Marca como ganador
                                puntuacion.getScore();  // Guarda el puntaje final
                                Herramientas.play("BOOEEE.wav");
                                }
                                            }
```

8. Durante todo el juego deberá mostrarse en algún lugar de la pantalla: El tiempo transcurrido del juego, la cantidad de gnomos rescatados y perdidos, y la cantidad de enemigos eliminados.

Clase Score

```
	public void sumarPuntos(int pun) {
		puntos += pun;
	}
	public void sumarGnomosPerdidos (int pun) {
		gnomosPerdidos += pun;
	}
	public void sumarGnomosSalvados (int pun) {
		gnomosSalvados += pun;
	}
	public void cantEnemigosEliminados (int pun) {
		enemigosEliminados += pun;
	}
	public void dibujar(Entorno e) {
		e.cambiarFont("Arial", 32, Color.WHITE);
		e.escribirTexto("Puntos: " + puntos, 560,
				200);
		e.cambiarFont("Arial", 28, Color.GREEN);
		e.escribirTexto("Gnomos salvados: " + gnomosSalvados, 500, 100);
		e.cambiarFont("Arial", 28, Color.RED);
		e.escribirTexto("Gnomos perdidos: " + gnomosPerdidos, 500, 50);
		e.cambiarFont("Arial", 28, Color.GREEN);
		e.escribirTexto("Enemigos eliminados: " + enemigosEliminados,500,150);
		long tiempoActual = System.currentTimeMillis();
```

```
            int tiempoTranscurridoSegundos = (int) ((tiempoActual - tiempoInicio) / 1000);

            // Dibujar el tiempo en pantalla
            e.cambiarFont("Arial", 28, Color.BLACK);
            e.escribirTexto("Tiempo: " + tiempoTranscurridoSegundos + " s", 10, 30);
        }

        public int getScore() {
            return puntos;
        }
}
```

Clase de juego principal

```
if (gnomo[i].llegoFondo(entorno)) {
    puntuacion.sumarGnomosPerdidos(1);
    puntuacion.sumarPuntos(-10);
}
if (gnomo[i] != null && gnomo[i].fueChocadoPorUnEnemigo(tortuga)) {
    puntuacion.sumarPuntos(-10);
   puntuacion.sumarGnomosPerdidos(1);
}
if (gnomo[i] != null && gnomo[i].fueChocadoPorUnEnemigo(tortuga)) {
```

```
    puntuacion.sumarPuntos(-10);
    puntuacion.sumarGnomosPerdidos(1);
}
if (gnomo[i] != null && bomba != null && gnomo[i].chocoConBomba(bomba[r])) {
    puntuacion.sumarPuntos(-10);
    puntuacion.sumarGnomosPerdidos(1);
}
if (gnomo [i] != null && gnomo[i].chocoAlHeroe(pep)) {
    puntuacion.sumarPuntos(30);
    puntuacion.sumarGnomosSalvados(1);
}

if (tortuga[j] != null && tortuga[j].chocoConFireball(fireballs)) {
    puntuacion.cantEnemigosEliminados(1);
    puntuacion.sumarPuntos(20);
}
```

- Disparos de las tortugas: Las tortugas arrojan bombas que se mueven a ras del suelo hasta que hagan colisión con Pep, los gnomos o salgan de la pantalla. Esas bomban solo pueden ser destruidas por los disparos de fuego.

Clase de Bombas

```
public void mover() {
        if (moviendoDerecha) {
            this.x += velocidad; // Mueve a la derecha
```

```
    } else {
        this.x -= velocidad; // Mueve a la izquierda
    }
  }
public double getX() {
    return this.x;
  }
  public double getY() {
    return this.y;
  }
```

Clase de Tortugas

```
        public long getUltimoDisparo() {
            return this.ultimoDisparo;
        }
        public void setUltimoDisparo(long tiempo) {
            this.ultimoDisparo = tiempo;
        }
```

Clase principal

```
if (entorno.sePresiono('C') && pep.estaSobreAlgunaIsla(islas)) {
```

```
                if (tiempoActual - ultimoDisparo >= 5000) {
                double xInicial = pep.getX(); // Centro en X
                double yInicial = pep.getY() + 10; // Centro en Y
                  ultimoDisparo = tiempoActual; // Actualiza el tiempo del último disparo
                break; // Sal del bucle después de lanzar la fireball
}
}
```

- Gnomos con ´Items Especiales: Al hacer colisión con un gnomo, ´este le puede dar (o quitar) a Pep un escudo extra o inmortalidad por unos segundos, siempre y cuando no caiga al precipicio.

Clase de Pep

```
public void activarInmortalidad() {
          this.inmortal = true;
   // Registra el tiempo de inicio.
           this.tiempoInmortalInicio = System.currentTimeMillis();
  }

        // Actualiza el estado de inmortalidad.
        public void actualizarInmortalidad() {
     if (this.inmortal && (System.currentTimeMillis() - this.tiempoInmortalInicio) > 4000) {
            // Se desactiva la inmortalidad después de 4 segundos.
```

```
                this.inmortal = false;
        }
    }


        public boolean esInmortal() {
                // Retorna si Pep es inmortal.
        return this.inmortal;
    }
```

Clase principal

```
pep.actualizarInmortalidad();
if (gnomo [i] != null && gnomo[i].chocoAlHeroe(pep)) {
            pep.activarInmortalidad();  // Activa inmortalidad
}
if (pep.chocoAlgunEnemigo(tortuga) && !pep.esInmortal()) {
             pep.morir(entorno);
          Herramientas.play("Super-Mario-Bros.wav");
          }
if (pep.chocoConBomba(bomba) && !pep.esInmortal()) {
          pep.morir(entorno);
           Herramientas.play("Super-Mario-Bros.wav");
          }
```

Conclusiones

Para concluir este informe podemos decir que este trabajo nos deja distintas formas de cómo se podría afrontar la creación de un juego en java, y distintas formas de cómo trabajar y manipular objetos en dicho lenguaje. Gracias a este tp, lograremos mejorar el uso de las herramientas que nos brinda este lenguaje, y podremos aplicar nuevas formas de encarar un problema que se nos pueda presentar.