

Trabajo Práctico Programación I

AL RESCATE DE LOS GNOMOS

PROGRAMACION I.

UNGS, 2024.

GRUPO 8.

INTEGRANTES DEL GRUPO:

| Abogado Matias, Gitto Santino, Herrera Franco, Pacheco Franco |

| Comisión: 05 |

CORREOS:

| matiabogado2015@gmail.com |

| santino.gitto051@gmail.com |

| herrerafranco167@gmail.com |
| francodamianpacheco@gmail.com |

Introducción

Este trabajo practico llamado “Al rescate de los gnomos” se trata de un juego de plataformas o “islas” desarrollado en el segundo semestre del 2024 por el grupo 8 de Programación I de la UNGS, en el lenguaje de programación Java.

Para poner en contexto de que se trata, es un juego en donde hay varias plataformas que son islas que en conjunto forman una piramide, en donde en la cuspide de la misma se encuentra la casa de los gnomos. Estos gnomos van saliendo de su casa y van desplazandose y cayendo de una a la otra y, si llegan hasta las ultimas, pueden llegar a caer al vacio.

Por otro lado, se encuentran los antagonistas: las tortugas. Estos personajes caen desde el cielo, y permanecen moviendose en la isla en la que caen, de izquierda a derecha, eliminando a todo lo que las toque.

Para finalizar con los personajes, hablaremos del protagonista del juego: Pep. Este personaje va a ser el encargado de rescatar a los gnomos. Es decir, la responsabilidad de Pep es salvar a todos los gnomos que pueda. Con un simple toque (eso si, en las islas inferiores), Pep puede salvar a los gnomos antes de que caigan al vacio, o lamentablemente, antes de que sean aniquilados por las tortugas. Por eso, Pep, va a tener la habilidad de disparar con el fin de aniquilar a las tortugas antes de que eliminen a los gnomos. Este personaje va a estar controlado por el jugador, quien tiene 3 intentos o “vidas” antes que el tiempo acabe para salvar a la mayor cantidad de gnomos posibles.

Entendiendo lo anteriormente mencionado, hemos creado este juego con los conocimientos adquiridos durante la cursada, en el cual hemos atravesado obstaculos, dificultades y hasta nos hemos tenido que forzar a incorporar ciertos conceptos que no conociamos para poder lograr la finalizacion de esta tarea. Con mucho gusto, en lo que resta del informe, estaremos proporcionando toda la informacion del codigo realizado: clases, variables de instancia, funcionalidades, y metodos.

Desarrollo

Para comenzar con el desarrollo, en los siguientes párrafos hablaremos acerca de las clases implementadas en este juego, las variables de instancia de cada una, y los metodos que poseen.

En primer lugar, la clase **Casa**:

Variables de instancia

- **int x**: Posición horizontal de la casa en el entorno.
- **int y**: Posición vertical de la casa en el entorno.
- **int ancho**: Ancho de la casa.
- **int alto**: Altura de la casa.
- **Color color**: Color de la casa, utilizado para el color en lo visual.

Métodos

- **Casa (int x, int y, int ancho, int alto, Color color)**: es el constructor de la clase que inicializa la posición (x, y), dimensiones (ancho, alto) y color de la casa.
- **void dibujar (Entorno entorno)**: Método que dibuja la casa en el entorno proporcionado utilizando un rectángulo con los valores de x, y, ancho, alto y el color especificado.
- **int getX ()**: Retorna la posición horizontal (x) de la casa.
- **int getY ()**: Retorna la posición vertical (y) de la casa.
- **int getAncho ()**: Retorna el ancho de la casa.
- **int getAlto ()**: Retorna la altura de la casa.

En segundo lugar, la clase **Isla**:

Variables de instancia

- **int x**: Posición horizontal de la isla en el entorno.
- **int y**: Posición vertical de la isla en el entorno.
- **int ancho**: Ancho de la isla.
- **int alto**: Altura de la isla.
- **Color color**: Color de la isla, utilizado para definir su apariencia visual

Métodos

- **Isla (int x, int y, int ancho, int alto, Color color)**: es el constructor de la clase que inicializa la posición (x, y), dimensiones (ancho, alto) y color de la isla.

- **void dibujar (Entorno entorno):** Método que dibuja la isla en el entorno. Crea un rectángulo en la posición y con las dimensiones y el color especificados.
- **int getX ():** Retorna la posición horizontal (x) de la isla.
- **int getY ():** Retorna la posición vertical (y) de la isla.
- **int getAncho ():** Retorna el ancho de la isla.
- **int getAlto ():** Retorna la altura de la isla.

En tercer lugar, la clase **Disparo**:

Variables de instancia

- **double x:** Posición horizontal del disparo en el entorno.
- **double y:** Posición vertical del disparo en el entorno.
- **double ancho:** Ancho del disparo.
- **double alto:** Altura del disparo.
- **String dir:** Dirección en la que se mueve el disparo ("der" para derecha y otra para izquierda).

Métodos

- **Disparo (double x, double y, double ancho, double alto, String dir):** Constructor de la clase que inicializa la posición (x, y), dimensiones (ancho, alto) y dirección (dir) del disparo.
- **void dibujar (Entorno e):** Dibuja el disparo en el entorno usando un rectángulo amarillo en la posición y con las dimensiones especificadas.
- **void mover ():** Mueve el disparo en la dirección especificada. Si la dirección es "der", el disparo se mueve hacia la derecha en 9 unidades; en caso contrario, se mueve hacia la izquierda.
- **void setDir(String dir):** Establece la dirección del disparo (dir).
- **boolean colisionEntorno (Entorno entorno):** Verifica si el disparo colisionó con el borde del entorno. Devuelve true si el disparo ha salido de los límites del entorno.
- **boolean colisionConTortuga (Tortuga tortuga):** Comprueba si el disparo colisionó con una tortuga. Devuelve true si las posiciones del disparo y la tortuga se superponen.
- **boolean colisionConIsla (Isla isla):** Comprueba si el disparo colisionó con una isla. Devuelve true si las posiciones del disparo y la isla se superponen.

Métodos Getters

- **double getX ()**: Retorna la posición horizontal (x) del disparo.
- **double getY ()**: Retorna la posición vertical (y) del disparo.
- **double getAncho ()**: Retorna el ancho del disparo.
- **double getAlto ()**: Retorna la altura del disparo.
- **String getDir ()**: Retorna la dirección en la que se mueve el disparo (dir).

En cuarto lugar, la clase **Gnomo**:

Variables de instancia

- **float x**: Posición horizontal del gnomo en el entorno.
- **float y**: Posición vertical del gnomo en el entorno.
- **int ancho**: Ancho del gnomo.
- **int alto**: Altura del gnomo.
- **Color color**: Color del gnomo.
- **boolean enIsla**: Indica si el gnomo está actualmente en una isla.
- **float direccion**: Dirección de movimiento del gnomo, donde 1 indica hacia la derecha y -1 hacia la izquierda.
- **int contadorEspera**: Contador que determina el tiempo que el gnomo espera antes de reaparecer después de caer.
- **final int tiempoEspera**: Tiempo de espera en ticks antes de que el gnomo reaparezca. Este valor es fijo (ya que es final).
- **boolean enEspera**: Indica si el gnomo está en estado de espera para reaparecer.

Métodos

- **Gnomo (int x, int y, int ancho, int alto, Color color)**: Constructor que inicializa la posición, dimensiones, y color del gnomo. La dirección de movimiento se asigna aleatoriamente al crearse el gnomo.
- **int getAncho ()** y **int getAlto ()**: Devuelven el ancho y el alto del gnomo, respectivamente.
- **int getTiempoEspera ()**: Retorna el tiempo de espera en ticks (tiempoEspera) que el gnomo necesita para reaparecer.
- **boolean estaEnEspera ()**: Retorna true si el gnomo está en estado de espera, de lo contrario, devuelve false.
- **void mover ()**: Mueve al gnomo en la dirección actual si no está en espera. Si el gnomo intenta salir de los límites horizontales de la pantalla, cambia de dirección.

- **void caer ()**: Hace que el gnomo caiga hacia abajo si no está en una isla y no está en espera. Si cae fuera de los límites de la pantalla, inicia un tiempo de espera antes de reaparecer.
- **private void iniciarEspera ()**: Inicia el estado de espera para el gnomo, estableciendo el contador de espera al valor de tiempoEspera y marcando al gnomo como en espera.
- **private void reiniciarEnCasa ()**: Reposiciona al gnomo en la "casa", en la parte superior de la pantalla, y termina el estado de espera.
- **void actualizar (Isla[] islas)**: Actualiza el estado del gnomo. Si el contador de espera ha terminado, comprueba si el gnomo está sobre alguna isla y ajusta su posición. Si no está sobre una isla, el gnomo cae.
- **void dibujar (Entorno entorno)**: Dibuja al gnomo en el entorno si no está en espera.
- **float getX ()** y **float getY ()**: Devuelven las coordenadas x y del gnomo, respectivamente.

En quinto lugar, la clase **Tortuga**:

Variables de Instancia

- **double x**: Representa la posición horizontal de la tortuga en el entorno del juego.
- **double y**: Representa la posición vertical de la tortuga en el entorno del juego.
- **double ancho**: Representa el ancho de la tortuga; se establece en un valor fijo de 25 para evitar tamaños excesivos.
- **double alto**: Representa la altura de la tortuga; también se establece en un valor fijo de 25.
- **double velocidadX**: Representa la velocidad de movimiento de la tortuga en la dirección horizontal. Se establece inicialmente en 0.5.
- **boolean enIsla**: Indica si la tortuga se encuentra actualmente en una isla. Se inicializa en false.
- **Random random**: Un objeto de la clase Random que se utiliza para generar posiciones y direcciones aleatorias para la tortuga.
- **Color color**: Representa el color de la tortuga, que se establece en Color.PINK.
- **Isla islaActual**: Referencia a la isla en la que la tortuga se encuentra actualmente.

Métodos

- **Tortuga (double limiteCasaGnomosX, double limiteCasaGnomosAncho)**: Constructor de la clase Tortuga. Inicializa

las variables de instancia con valores predeterminados y llama al método `generarPosicionAleatoria` para establecer una posición inicial aleatoria, evitando el área de la casa de los gnomos.

- **private void generarPosicionAleatoria (double limiteCasaGnomosX, double limiteCasaGnomosAncho):** genera una posición aleatoria en el eje X, asegurándose de que la tortuga no aparezca en el área de la casa de los gnomos.
- **private boolean islaEsCasaGnomos (Isla isla, Casa casa):** verifica si la isla dada es la que contiene la casa de los gnomos, comparando las coordenadas de la isla con las de la casa.
- **private void moverEnIsla ():** mueve la tortuga de un borde a otro de la isla. Actualiza la posición en X de la tortuga y verifica si ha alcanzado los límites de la isla para cambiar de dirección.
- **private boolean colisionConIsla (Isla isla):** verifica si la tortuga colisiona con una isla dada, comprobando las coordenadas y dimensiones de ambas.
- **public boolean colisionConGnomo (Gnomo gnomo):** verifica si hay colisión entre la tortuga y un gnomo. Comprueba si las posiciones de ambos se superponen.
- **private boolean islaTieneCasa (Isla isla, Casa casa):** verifica si la isla dada tiene la casa de los gnomos, similar al método `islaEsCasaGnomos`.
- **private boolean islaOcupada (Isla[] islas, Isla isla):** verifica si hay otra tortuga en la misma isla comparando la isla actual con un arreglo de islas.
- **public void actualizar (Isla[] islas, Casa casa):** actualiza el estado de la tortuga. Si la tortuga no está en una isla, llama al método `caer`; si está en una isla, llama a `moverEnIsla`.
- **private void caer (Isla[] islas, Casa casa):** gestiona la lógica de caída de la tortuga. Comprueba si la tortuga ha colisionado con alguna isla. Si colisiona, establece `enIsla` en true y coloca la tortuga sobre la isla; si no, la tortuga cae hacia abajo.
- **public void dibujar (Entorno entorno):** dibuja la tortuga en el entorno del juego utilizando un rectángulo de su color y dimensiones.
- **public double getX ():** Método getter que devuelve la posición X de la tortuga.
- **public double getY ():** Método getter que devuelve la posición Y de la tortuga.

- **public double getAncho ()**: Método getter que devuelve el ancho de la tortuga.
- **public double getAlto ()**: Método getter que devuelve la altura de la tortuga.

En sexto lugar, la clase **Pep**:

Variables de Instancia

- **int x**: Posición horizontal del personaje en el entorno.
- **int y**: Posición vertical del personaje en el entorno.
- **int ancho**: Ancho del personaje.
- **int alto**: Altura del personaje.
- **Color color**: Color que representa al personaje en el entorno gráfico.
- **boolean enSuelo**: Indica si el personaje está en el suelo (true) o en el aire (false).
- **double velocidadY**: Velocidad vertical del personaje, utilizada para simular el salto y la caída.
- **final double gravedad**: Constante que representa la gravedad aplicada al personaje (0.4).
- **Disparo disparo**: Objeto que representa un disparo realizado por el personaje.
- **String ultimaDir**: Dirección en la que el personaje se movió por última vez ("der" para derecha o "izq" para izquierda).

Métodos

- **Pep (int x, int y, int ancho, int alto, Color color)**: Constructor que inicializa el personaje con sus coordenadas (x, y), dimensiones (ancho, alto) y color. Establece enSuelo a true, velocidadY a 0 y ultimaDir a "der".
- **void moverDerecha (int limiteDerecho)**: Mueve al personaje hacia la derecha incrementando su posición x en 4 unidades, siempre que no exceda el límite derecho. Actualiza ultimaDir a "der".
- **void moverIzquierda ()**: Mueve al personaje hacia la izquierda decrementando su posición x en 4 unidades, siempre que no exceda el borde izquierdo. Actualiza ultimaDir a "izq".
- **void saltar ()**: Inicia el salto del personaje si está en el suelo, estableciendo velocidadY a -8 y cambia enSuelo a false.
- **void caer ()**: Aplica la gravedad al personaje si no está en el suelo, actualizando velocidadY y su posición y.
- **void detenerSaltoEnIsla (int yIsla)**: Detiene el salto del personaje al alcanzarlo en una isla, posicionándolo en la altura correcta y estableciendo enSuelo a true.

- **void salirDeIsla():** Cambia el estado del personaje a `enSuelo` a `false`, indicando que ya no está en una isla.
- **void caerInmediatamente ():** Establece `velocidadY` a la gravedad y cambia `enSuelo` a `false`, forzando al personaje a caer.
- **Disparo disparar ():** Crea un nuevo objeto `Disparo` en la posición del personaje, ajustando su altura para que se inicie desde el cuerpo del personaje, y retorna el disparo.
- **boolean estaSaltando ():** Retorna `true` si `velocidadY` es menor que 0, indicando que el personaje está saltando.
- **boolean estaCayendo ():** Retorna `true` si `velocidadY` es mayor que 0, indicando que el personaje está cayendo.
- **void dibujar (Entorno entorno):** Dibuja el personaje en el entorno gráfico usando un rectángulo con su posición, tamaño y color.
- **boolean colisionConTortuga (Tortuga tortuga):** Verifica si hay una colisión entre el personaje y un objeto `Tortuga`, devolviendo `true` si las posiciones y dimensiones se superponen.
- **void reaparecer (int nuevoX, int nuevoY):** Reubica al personaje en nuevas coordenadas `nuevoX` y `nuevoY`.

Métodos Getters

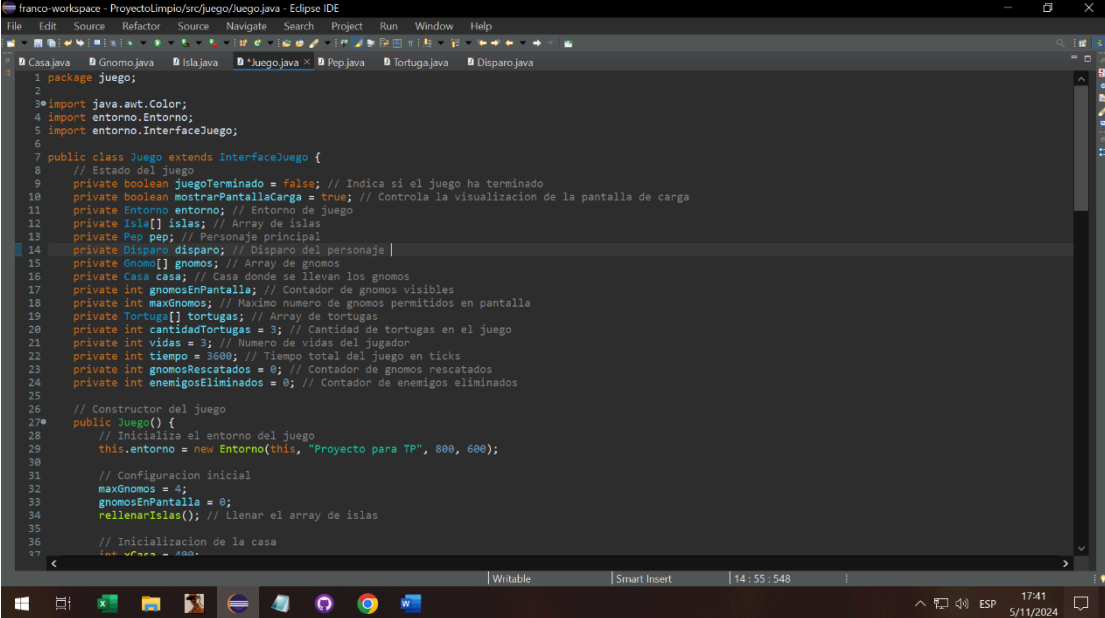
- **int getX ():** Retorna la posición horizontal (x) del personaje.
- **int getY ():** Retorna la posición vertical (y) del personaje.
- **int getAncho ():** Retorna el ancho del personaje.
- **int getAlto ():** Retorna la altura del personaje.
- **Color getColor ():** Retorna el color del personaje.
- **boolean isEnSuelo ():** Retorna `true` si el personaje está en el suelo.
- **double getVelocidadY ():** Retorna la velocidad vertical (`velocidadY`) del personaje.
- **double getGravedad ():** Retorna el valor de la gravedad.
- **Disparo getDisparo ():** Retorna el objeto `Disparo` asociado al personaje.
- **String getUltimaDir ():** Retorna la última dirección en la que se movió el personaje.

Consideramos que esta clase fue una de las más complicadas de implementar debido a que se deben tener en cuenta varios factores a la hora de querer llevarlo al juego.

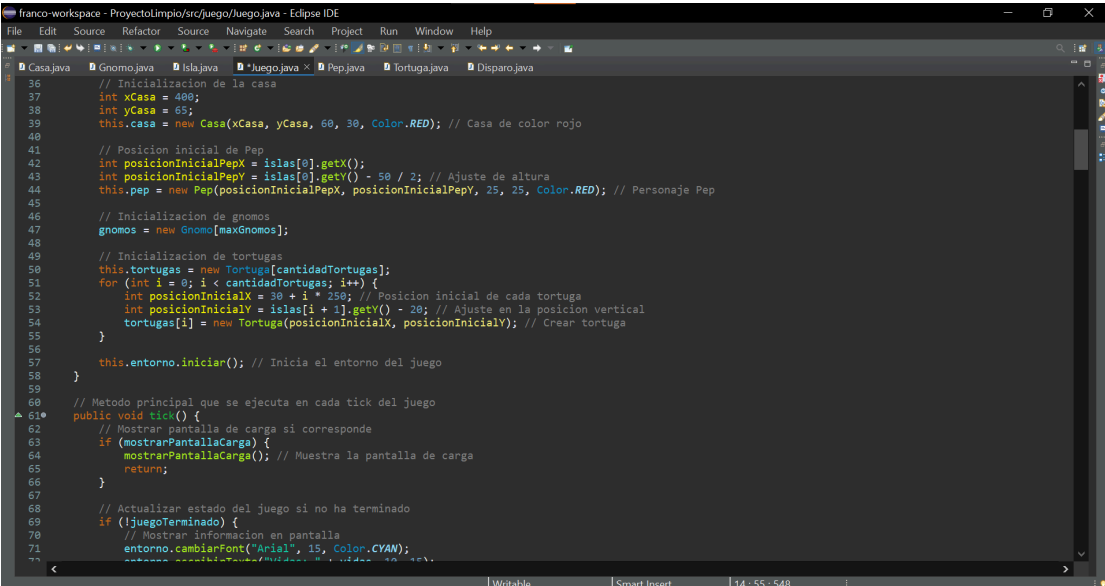
Esta clase en especial nos dio mucho problema con las islas, ya que no sabíamos como detectar las colisiones con las islas, cuando cae o cuando saltaba, lo que hicimos fue dividir la lógica en varias partes, para poder así tener en cuenta todas las posibilidades que podría tener “Pep” cuando

colisiona con las islas. Por eso creamos métodos como `detenerSaltoEnIsla()`, `salirDeIsla()`, `caerInmediatamente()`, que nos permiten tener al alcance todas las situaciones posibles.

Implementación:



```
1 package juego;
2
3 import java.awt.Color;
4 import entorno.Entorno;
5 import entorno.InterfaceJuego;
6
7 public class Juego extends InterfaceJuego {
8     // Estado del juego
9     private boolean juegoTerminado = false; // Indica si el juego ha terminado
10    private boolean mostrarPantallaCarga = true; // Controla la visualización de la pantalla de carga
11    private Entorno entorno; // Entorno de juego
12    private Isla[] islas; // Array de islas
13    private Pep pep; // Personaje principal
14    private Disparo disparo; // Disparo del personaje
15    private Gnomos[] gnomos; // Array de gnomos
16    private Casa casa; // Casa donde se llevan los gnomos
17    private int gnomosEnPantalla; // Contador de gnomos visibles
18    private int maxGnomos; // Maximo numero de gnomos permitidos en pantalla
19    private Tortuga[] tortugas; // Array de tortugas
20    private int cantidadTortugas = 3; // Cantidad de tortugas en el juego
21    private int vidas = 3; // Numero de vidas del jugador
22    private int tiempo = 3600; // Tiempo total del juego en ticks
23    private int gnomosRescatados = 0; // Contador de gnomos rescatados
24    private int enemigosEliminados = 0; // Contador de enemigos eliminados
25
26    // Constructor del juego
27    public Juego() {
28        // Inicializa el entorno del juego
29        this.entorno = new Entorno(this, "Proyecto para TP", 800, 600);
30
31        // Configuración inicial
32        maxGnomos = 5;
33        gnomosEnPantalla = 0;
34        rellenarIslas(); // Llenar el array de islas
35
36        // Inicialización de la casa
37        int xCasa = 400;
38        int yCasa = 65;
39        this.casa = new Casa(xCasa, yCasa, 60, 30, Color.RED); // Casa de color rojo
40
41        // Posición inicial de Pep
42        int posicionInicialPepX = islas[0].getX();
43        int posicionInicialPepY = islas[0].getY() - 50 / 2; // Ajuste de altura
44        this.pep = new Pep(posicionInicialPepX, posicionInicialPepY, 25, 25, Color.RED); // Personaje Pep
45
46        // Inicialización de gnomos
47        gnomos = new Gnomo[maxGnomos];
48
49        // Inicialización de tortugas
50        this.tortugas = new Tortuga[cantidadTortugas];
51        for (int i = 0; i < cantidadTortugas; i++) {
52            int posicionInicialX = 30 + i * 250; // Posición inicial de cada tortuga
53            int posicionInicialY = islas[i + 1].getY() - 20; // Ajuste en la posición vertical
54            tortugas[i] = new Tortuga(posicionInicialX, posicionInicialY); // Crear tortuga
55        }
56
57        this.entorno.iniciar(); // Inicia el entorno del juego
58    }
59
60    // Metodo principal que se ejecuta en cada tick del juego
61    public void tick() {
62        // Mostrar pantalla de carga si corresponde
63        if (mostrarPantallaCarga) {
64            mostrarPantallaCarga(); // Muestra la pantalla de carga
65            return;
66        }
67
68        // Actualizar estado del juego si no ha terminado
69        if (!juegoTerminado) {
70            // Mostrar información en pantalla
71            entorno.cambiarFont("Arial", 15, Color.CYAN);
72            // ... (código omitido) ...
73        }
74    }
75}
```



```
36 // Inicialización de la casa
37 int xCasa = 400;
38 int yCasa = 65;
39 this.casa = new Casa(xCasa, yCasa, 60, 30, Color.RED); // Casa de color rojo
40
41 // Posición inicial de Pep
42 int posicionInicialPepX = islas[0].getX();
43 int posicionInicialPepY = islas[0].getY() - 50 / 2; // Ajuste de altura
44 this.pep = new Pep(posicionInicialPepX, posicionInicialPepY, 25, 25, Color.RED); // Personaje Pep
45
46 // Inicialización de gnomos
47 gnomos = new Gnomo[maxGnomos];
48
49 // Inicialización de tortugas
50 this.tortugas = new Tortuga[cantidadTortugas];
51 for (int i = 0; i < cantidadTortugas; i++) {
52     int posicionInicialX = 30 + i * 250; // Posición inicial de cada tortuga
53     int posicionInicialY = islas[i + 1].getY() - 20; // Ajuste en la posición vertical
54     tortugas[i] = new Tortuga(posicionInicialX, posicionInicialY); // Crear tortuga
55 }
56
57 this.entorno.iniciar(); // Inicia el entorno del juego
58 }
59
60 // Metodo principal que se ejecuta en cada tick del juego
61 public void tick() {
62     // Mostrar pantalla de carga si corresponde
63     if (mostrarPantallaCarga) {
64         mostrarPantallaCarga(); // Muestra la pantalla de carga
65         return;
66     }
67
68     // Actualizar estado del juego si no ha terminado
69     if (!juegoTerminado) {
70         // Mostrar información en pantalla
71         entorno.cambiarFont("Arial", 15, Color.CYAN);
72         // ... (código omitido) ...
73     }
74 }
```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java *Juego.java X Pep.java Tortugas.java Disparos.java
71 entorno.cambiarFont("Arial", 15, Color.CYAN);
72 entorno.escribirTexto("Vidas: " + vidas, 10, 15);
73 entorno.escribirTexto("Tiempo: " + (tiempo / 60), 10, 55); // Convertir ticks a segundos
74 entorno.escribirTexto("Enemigos Eliminados: " + enemigosEliminados, 10, 35);
75 entorno.escribirTexto("Gnomos Rescatados: " + gnomosRescatados, 10, 75);
76
77 tiempo--; // Decrementar el tiempo por cada tick
78
79 // Control de movimiento de Pep
80 if (entorno.estaPresionada(entorno.TECLA_DERECHA)) {
81     pep.moverDerecha(500); // Mover a la derecha
82 }
83 if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
84     pep.moverIzquierda(); // Mover a la izquierda
85 }
86 if (entorno.estaPresionada(entorno.TECLA_ARRIBA)) {
87     pep.saltar(); // Saltar
88 }
89
90 pep.caer(); // Aplicar gravedad a Pep
91
92 boolean pepSobreIsla = false; // Flag para verificar si Pep esta sobre una isla
93
94 // Verificar colisiones de Pep con las islas
95 for (Isla isla : islas) {
96     // Comprobar si Pep esta saltando y colisiona con la isla
97     if (pep.estaSaltando() &&
98         pep.getY() - pep.getAlto() / 2 <= isla.getY() + isla.getAlto() / 2 &&
99         pep.getY() > isla.getY() &&
100         pep.getX() + pep.getAncho() / 2 > isla.getX() - isla.getAncho() / 2 &&
101         pep.getX() - pep.getAncho() / 2 < isla.getX() + isla.getAncho() / 2) {
102         pep.caerInmediatamente(); // Pep cae inmediatamente si colisiona al saltar
103     }
104
105     // Comprobar si Pep esta cayendo y colisiona con la isla

```

```

106     if (pep.estaCayendo() &&
107         pep.getY() + pep.getAlto() / 2 >= isla.getY() - isla.getAlto() / 2 &&
108         pep.getY() < isla.getY() &&
109         pep.getX() + pep.getAncho() / 2 > isla.getX() - isla.getAncho() / 2 &&
110         pep.getX() - pep.getAncho() / 2 < isla.getX() + isla.getAncho() / 2) {
111         pep.detenerSaltoEnIsla(isla.getY() - isla.getAlto() / 2); // Detener el salto en la isla
112         pepSobreIsla = true; // Pep esta sobre la isla
113     }
114 }
115
116 // Si Pep no esta sobre ninguna isla, actualizar su estado
117 if (!pepSobreIsla) {
118     pep.salirDeIsla(); // Pep sale de la isla si no esta sobre ella
119 }
120
121 // Si Pep no esta sobre una isla y esta cayendo, aplicar caida
122 if (!pepSobreIsla && (pep.estaCayendo() || pep.getY() < 500)) {
123     pep.caer(); // Pep continua cayendo
124 }
125
126 // Verificar si Pep cae fuera de la pantalla
127 if (pep.getY() > 600) {
128     vidas--; // Pierde una vida
129     if (vidas == 0) {
130         juegoTerminado = true; // Termina el juego si no quedan vidas
131     } else {
132         // Reaparece en la posición inicial
133         pep.reaparecer(islas[0].getX(), islas[0].getY() - 50 / 2);
134     }
135 }
136
137 if (entorno.sePresiono('c') && disparo == null) {
138     disparo = pep.disparar();
139 }
140
141

```

```

142 if (disparo != null) {
143     disparo.dibujar(entorno);
144     disparo.mover();
145 }
146 // Verificar si el disparo choca con el borde del entorno
147 if (disparo.colisionEntorno(entorno)) {
148     disparo = null;
149 } else {
150     // Verificar colisión con islas
151     for (int i = 0; i < islas.length; i++) {
152         if (disparo == null) break; // Salir si disparo es null
153
154         if (islas[i] != null &&
155             islas[i].getX() + islas[i].getAncho() / 2 > disparo.getX() - disparo.getAncho() / 2 &&
156             islas[i].getX() - islas[i].getAncho() / 2 < disparo.getX() + disparo.getAncho() / 2 &&
157             islas[i].getY() + islas[i].getAlto() / 2 > disparo.getY() - disparo.getAlto() / 2 &&
158             islas[i].getY() - islas[i].getAlto() / 2 < disparo.getY() + disparo.getAlto() / 2) {
159
160             disparo = null; // Eliminar el disparo tras impactar con una isla
161             break; // Salir del bucle tras impactar con una isla
162         }
163     }
164
165     // Verificar colisión con gnomos
166     for (int i = 0; i < gnomos.length; i++) {
167         if (pep.getY() > islas[2].getY()) {
168             if (disparo == null) break; // Salir si disparo es null
169
170             if (gnomos[i] != null &&
171                 gnomos[i].getX() + gnomos[i].getAncho() / 2 > disparo.getX() - disparo.getAncho() / 2 &&
172                 gnomos[i].getX() - gnomos[i].getAncho() / 2 < disparo.getX() + disparo.getAncho() / 2 &&
173                 gnomos[i].getY() + gnomos[i].getAlto() / 2 > disparo.getY() - disparo.getAlto() / 2 &&
174                 gnomos[i].getY() - gnomos[i].getAlto() / 2 < disparo.getY() + disparo.getAlto() / 2) {
175
176                 gnomos[i] = null; // Eliminar al gnomo

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java *Juego.java x Pep.java Tortuga.java Disparo.java
176         gnomos[i] = null; // Eliminar al gnomo
177         gnomosRescatados++; // Aumentar el puntaje
178         disparo = null; // Eliminar el disparo tras impactar
179         break; // Salir del bucle tras eliminar un gnomo
180     }
181 }
182 }
183
184
185 // Verificar colisión con tortugas
186 for (int i = 0; i < tortugas.length; i++) {
187     if (disparo == null) break; // Salir si disparo es null
188
189     if (tortugas[i] != null &&
190         tortugas[i].getX() + tortugas[i].getAncho() / 2 > disparo.getX() - disparo.getAncho() / 2 &&
191         tortugas[i].getX() - tortugas[i].getAncho() / 2 < disparo.getX() + disparo.getAncho() / 2 &&
192         tortugas[i].getY() + tortugas[i].getAlto() / 2 > disparo.getY() - disparo.getAlto() / 2 &&
193         tortugas[i].getY() - tortugas[i].getAlto() / 2 < disparo.getY() + disparo.getAlto() / 2) {
194
195         tortugas[i] = null; // Eliminar a la tortuga
196         enemigosEliminados++; // Aumentar el contador de enemigos eliminados
197         disparo = null; // Eliminar el disparo tras impactar
198         break; // Salir del bucle tras eliminar una tortuga
199     }
200 }
201 }
202 }
203
204
205 for (int i = 0; i < gnomos.length; i++) {
206     if (gnomos[i] != null) {
207         gnomos[i].mover();
208         gnomos[i].actualizar(islas);
209
210         if (gnomos[i].getY() > 600) {
211             gnomos[i] = null;
212         }
213     }
214 }

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java *Juego.java x Pep.java Tortuga.java Disparo.java
210         if (gnomos[i].getY() > 600) {
211             gnomos[i] = null;
212             gnomosEnPantalla--;
213         }
214     }
215 }
216
217 for (int i = 0; i < gnomos.length; i++) {
218     if (gnomos[i] != null) {
219         if (pep.getX() > 300 &&
220             pep.getX() + pep.getAncho() / 2 > gnomos[i].getX() - gnomos[i].getAncho() / 2 &&
221             pep.getX() - pep.getAncho() / 2 < gnomos[i].getX() + gnomos[i].getAncho() / 2 &&
222             pep.getY() + pep.getAlto() / 2 > gnomos[i].getY() - gnomos[i].getAlto() / 2 &&
223             pep.getY() - pep.getAlto() / 2 < gnomos[i].getY() + gnomos[i].getAlto() / 2) {
224
225             gnomos[i] = null;
226             gnomosEnPantalla--;
227             gnomosRescatados++;
228         }
229     }
230 }
231
232 if (gnomosEnPantalla < maxGnomos) {
233     crearGnomoDesdeCasa();
234 }
235
236 for (int i = 0; i < tortugas.length; i++) {
237     Tortuga tortuga = tortugas[i];
238     if (tortuga != null) {
239         tortuga.actualizar(islas, casa);
240         tortuga.dibujar(entorno);
241     }
242
243     // Colisión entre Pep y una tortuga
244     if (pep != null &&
245         pep.getX() + pep.getAncho() / 2 > tortuga.getX() - tortuga.getAncho() / 2 &&
246         pep.getX() - pep.getAncho() / 2 < tortuga.getX() + tortuga.getAncho() / 2 &&
247         pep.getY() + pep.getAlto() / 2 > tortuga.getY() - tortuga.getAlto() / 2 &&
248         pep.getY() - pep.getAlto() / 2 < tortuga.getY() + tortuga.getAlto() / 2) {
249
250         vidas--;
251         if (vidas > 0) {
252             int posicionInicialPepX = islas[0].getX();
253             int posicionInicialPepY = islas[0].getY() - 50 / 2;
254             pep = new Pep(posicionInicialPepX, posicionInicialPepY, 25, 25, Color.RED);
255         } else {
256             juegoTerminado = true;
257         }
258         tortugas[i] = null;
259     }
260
261     // Colisión entre Gnomos y una tortuga
262     for (int j = 0; j < gnomos.length; j++) {
263         if (gnomos[j] != null &&
264             gnomos[j].getX() + gnomos[j].getAncho() / 2 > tortuga.getX() - tortuga.getAncho() / 2 &&
265             gnomos[j].getX() - gnomos[j].getAncho() / 2 < tortuga.getX() + tortuga.getAncho() / 2 &&
266             gnomos[j].getY() + gnomos[j].getAlto() / 2 > tortuga.getY() - tortuga.getAlto() / 2 &&
267             gnomos[j].getY() - gnomos[j].getAlto() / 2 < tortuga.getY() + tortuga.getAlto() / 2) {
268
269             gnomos[j] = null;
270             gnomosEnPantalla--;
271         }
272     }
273 }
274
275 // Resparición de gnomos al ser eliminados por el disparo o caer fuera de pantalla
276 if (gnomosEnPantalla < maxGnomos) {
277     for (int i = 0; i < gnomos.length; i++) {
278         if (gnomos[i] == null) {

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java *Juego.java x Pep.java Tortuga.java Disparo.java
276         if (pep != null &&
277             pep.getX() + pep.getAncho() / 2 > tortuga.getX() - tortuga.getAncho() / 2 &&
278             pep.getX() - pep.getAncho() / 2 < tortuga.getX() + tortuga.getAncho() / 2 &&
279             pep.getY() + pep.getAlto() / 2 > tortuga.getY() - tortuga.getAlto() / 2 &&
280             pep.getY() - pep.getAlto() / 2 < tortuga.getY() + tortuga.getAlto() / 2) {
281
282             vidas--;
283             if (vidas > 0) {
284                 int posicionInicialPepX = islas[0].getX();
285                 int posicionInicialPepY = islas[0].getY() - 50 / 2;
286                 pep = new Pep(posicionInicialPepX, posicionInicialPepY, 25, 25, Color.RED);
287             } else {
288                 juegoTerminado = true;
289             }
290             tortugas[i] = null;
291         }
292     }
293
294     // Colisión entre Gnomos y una tortuga
295     for (int j = 0; j < gnomos.length; j++) {
296         if (gnomos[j] != null &&
297             gnomos[j].getX() + gnomos[j].getAncho() / 2 > tortuga.getX() - tortuga.getAncho() / 2 &&
298             gnomos[j].getX() - gnomos[j].getAncho() / 2 < tortuga.getX() + tortuga.getAncho() / 2 &&
299             gnomos[j].getY() + gnomos[j].getAlto() / 2 > tortuga.getY() - tortuga.getAlto() / 2 &&
300             gnomos[j].getY() - gnomos[j].getAlto() / 2 < tortuga.getY() + tortuga.getAlto() / 2) {
301
302             gnomos[j] = null;
303             gnomosEnPantalla--;
304         }
305     }
306 }
307
308 // Resparición de gnomos al ser eliminados por el disparo o caer fuera de pantalla
309 if (gnomosEnPantalla < maxGnomos) {
310     for (int i = 0; i < gnomos.length; i++) {
311         if (gnomos[i] == null) {

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java Juego.java Pep.java Tortugas.java Disparo.java
278 if (gnomos[i] == null) {
279     // Crear un nuevo gnomos en la posición de la casa o en otra posición inicial
280     gnomos[i] = new Gnomos(casa.getX() + casa.getAncho() / 2, casa.getY(), 25, 25, Color.BLUE);
281     gnomosEnPantalla++; // Incrementa el conteo solo cuando un gnomos reaparece
282     break; // Solo reaparece un gnomos por tick si falta
283 }
284 }
285
286 // Actualizar el conteo cuando un gnomos es eliminado
287 for (int i = 0; i < gnomos.length; i++) {
288     if (gnomos[i] == null) {
289         gnomosEnPantalla--; // Reducir el conteo cuando un gnomos es eliminado
290     }
291 }
292
293 // Reparación de tortugas en el arreglo al ser eliminadas
294 for (int i = 0; i < tortugas.length; i++) {
295     if (tortugas[i] == null) {
296         // Reaparecer una tortuga en una posición inicial, por ejemplo, cerca de la isla
297         int posicionInicialX = 30 + i * 250;
298         int posicionInicialY = islas[i + 1].getY() - 20;
299         tortugas[i] = new Tortuga(posicionInicialX, posicionInicialY);
300         break; // Solo reaparece una tortuga por tick si falta
301     }
302 }
303
304 // Dibujar las islas
305 for (Isla isla : islas) {
306     isla.dibujar(entorno);
307 }
308 // Dibujar la casa
309 casa.dibujar(entorno);
310
311
312
313

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java Juego.java Pep.java Tortugas.java Disparo.java
313 casa.dibujar(entorno);
314
315 // Dibujar a Pep
316 pep.dibujar(entorno);
317
318 // Dibujar los gnomos
319 for (Gnomos gnomos : gnomos) {
320     if (gnomos != null) {
321         gnomos.dibujar(entorno);
322     }
323 }
324
325 if (vidas == 0 || tiempo <= 0) {
326     juegoTerminado = true;
327 }
328
329 // if (juegoTerminado) {
330 //     mostrarPantallaFinal(); // Mostrar pantalla final
331 //     return;
332 // }
333
334
335
336
337 }
338
339 //aca termina el tick
340
341 private void mostrarPantallaCarga() {
342     entorno.cambiarFont("Arial", 24, Color.BLUE);
343
344     // Escribir el texto centrado
345     entorno.escribirTexto("Pantalla de Carga", entorno.ancha() / 2, 100); // Título
346     entorno.escribirTexto("Presiona 'J' para Jugar", entorno.ancha() / 2, 150); // Opción Jugar
347     entorno.escribirTexto("Presiona 'E' para Salir", entorno.ancha() / 2, 200); // Opción Salir
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384

```

```

franco-workspace - ProyectoLimpio/src/juego/Juego.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Casa.java Gnomos.java Islas.java Juego.java Pep.java Tortugas.java Disparo.java
385 // Comprobar entradas
386 if (entorno.sePresiona('j')) {
387     mostrarPantallaCarga = false; // Iniciar el juego
388 } else if (entorno.sePresiona('e')) {
389     System.exit(0); // Salir del juego
390 }
391
392 // Metodo para crear gnomos
393 private void crearGnomosDesdeCasa() {
394     if (gnomosEnPantalla < maxGnomos) {
395         int xGnomos = casa.getX() + casa.getAncho() / 2 - 12;
396
397         Isla islaMasAlta = islas[0];
398         for (Isla isla : islas) {
399             if (isla.getY() < islaMasAlta.getY()) {
400                 islaMasAlta = isla;
401             }
402         }
403
404         int yGnomos = islaMasAlta.getY() - 40;
405
406         for (int i = 0; i < gnomos.length; i++) {
407             if (gnomos[i] == null) {
408                 gnomos[i] = new Gnomos(xGnomos, yGnomos, 25, 25, Color.BLUE);
409                 gnomosEnPantalla++;
410                 break;
411             }
412         }
413     }
414 }
415
416 public static void main(String[] args) {
417     new Juego();
418 }
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

Conclusiones:

Este trabajo fue un desafío muy amplio para todos. En líneas generales, llevar la teoría a la práctica es algo totalmente diferente. Si bien durante la cursada, los ejercicios de práctica solían ser algo “sencillos” creemos en conjunto que lo complicado de este tipo de trabajos son tener en cuenta que cada objeto tiene su propia lógica, que en general, suele afectar cualquier cambio mínimo, con lo cual, hay que tener presente cada cosa que pensamos, que se escribe, pensar cada movimiento y pensar alternativas antes de comenzar a escribir código.

Como principiantes en el mundo de la programación, reconocemos que retos como estos son los que hacen que incorporemos mucho más a fondo la teoría, que podamos pensar de una manera mucho más lógica, paso por paso, entendiendo que lo que hacemos debe seguir instrucciones precisas, con el fin de tener el resultado esperado.

Llegamos a la conclusión de que este trabajo, afronto un reto en lo personal para cada uno, pero en conjunto, como equipo, hemos podido sobrellevarlo. A pesar de nuestra inexperiencia, consideramos que, si bien nos falta bastante para lograr a ser lo que queremos ser, han quedado varios conceptos aclarados, hemos aprendido a saber como funcionan ciertas cosas de este lenguaje que antes no sabíamos.

En conclusión, lo que este grupo se lleva de este trabajo es nada más y nada menos que un gran aprendizaje, incorporamos conceptos nuevos y aprendimos a trabajar en equipo, y una fuerte motivación por aprender cada vez más.