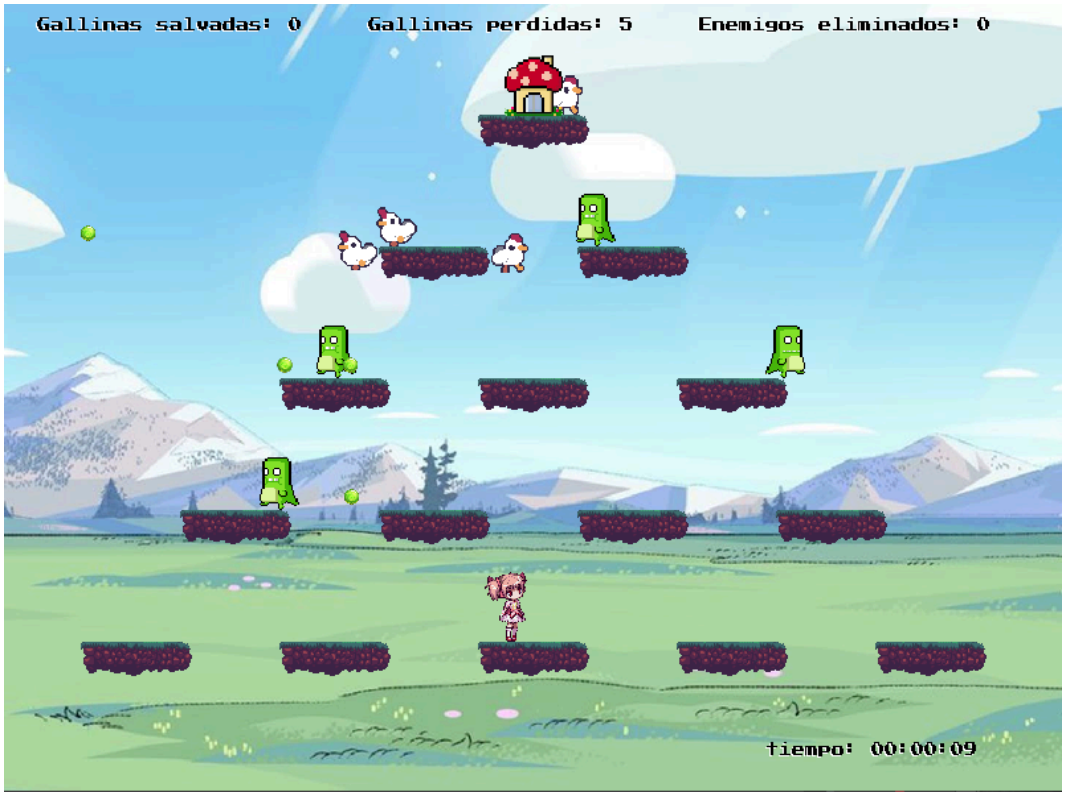
Universidad Nacional de General Sarmiento



Trabajo Práctico: “Al rescate de los Gnomos”

Universidad Nacional de General Sarmiento

Programación I - Segundo Cuatrimestre 2024 - Comisión 01

Docentes: Leonardo Waingarten - Lautaro Malleret

Polcaro Bianca
Rocha Mayra Solana
Sandoval Celeste Macarena

Introducción

En el marco de la asignatura Programación I, se nos ha planteado como trabajo práctico el desarrollo de un videojuego titulado *Al rescate de los Gnomos*, el mismo tiene lugar en un mundo mágico, donde sus habitantes, los gnomos, vivían pacíficamente en islas flotantes en el cielo. Sin embargo, su vida tranquila se ve amenazada cuando un día son atacados por tortugas de caparazón de acero, las cuales caen desde el cielo, ocupando las islas y eliminando a los gnomos con su veneno. Los gnomos, en su intento de huir, caen de una isla a otra, arriesgándose a caer al vacío o ser víctimas de una tortuga. Ante esta amenaza, los gnomos contratan a Pep, un caballero medieval, con el objetivo de salvarlos antes de que caigan o sean vencidos por las tortugas. El juego será un simulador donde Pep rescate a los gnomos enfrentándose a las tortugas y evitando que estas acaben con los habitantes de las islas flotantes.

Este proyecto ha tenido como objetivo poner en práctica conceptos de programación orientada a objetos, estructura de datos, y lógica aplicada a la creación de interacciones y movimientos en un entorno gráfico utilizando el lenguaje de programación Java. El juego requiere implementar personajes, manejar sus interacciones en pantalla, y configurar colisiones para crear una experiencia de juego dinámica.

A lo largo de su desarrollo, hemos enfrentado desafíos técnicos significativos, como la detección de colisiones entre objetos, el diseño del movimiento de los personajes y la implementación de un sistema de control para cada elemento en el juego. En este informe se detallará cómo se ha llevado a cabo el proceso de programación, presentando las clases y los métodos definidos en ellas, las dificultades encontradas y las soluciones aplicadas. Además, se presentarán imágenes del proceso de desarrollo y las decisiones creativas que sustentaron la elección de los personajes y sus características.

La finalidad de este proyecto no es solo mostrar la creación de un juego funcional, sino también reflejar el aprendizaje y la adaptación de técnicas de programación para la solución de problemas.

1. Estructura de Clases y Funcionalidades

Para el desarrollo de este juego se ha utilizado el lenguaje Java, con programación orientada a objetos como paradigma, por lo que uno de los primeros pasos a seguir fue la creación de diferentes clases para cada objeto donde se declaran sus variables de instancia, constructor y métodos. A continuación, se presentarán las clases implementadas para este proyecto y sus características.

Pep, el personaje principal que será controlado por el usuario:

La clase Pep representa al protagonista del juego, un caballero que debe rescatar a los gnomos y enfrentarse a las tortugas. Esta clase define sus atributos como posición (x,y), dimensiones (ancho,alto), y estado actual, es decir, si está cayendo, saltando, apoyado en una isla, entre otros. Además maneja las imágenes correspondientes a cada animación (imagenDerecha, imagenIzquierda, imagenCayendo) y contiene varios métodos que controlan el movimiento y la visualización de pep en el entorno.

los métodos principales de Pep son:

- mostrarAPep: Dibuja a Pep en pantalla, mostrando diferentes animaciones según su estado.
- movHorizontal: Controla el movimiento horizontal de Pep, permitiendo que se desplace dentro de los límites del entorno.
- movVertical y saltar: Gestionan el movimiento vertical, incluyendo la simulación de la caída y el salto.
- iniciarSalto: Inicia el salto cuando Pep está apoyado, estableciendo un límite de altura.

La clase también incluye varios *getters* y *setters* que facilitan la detección de colisiones mediante los bordes del personaje y permiten acceder y modificar su posición.

Representación visual de Pep. Animación de salto, espera y corriendo.

Gnomo, las víctimas a las que Pep viene a ayudar

La clase Gnomo representa a los personajes que Pep debe rescatar en el juego. Cada gnomo tiene atributos de posición (x,y), dimensiones (ancho,alto), e imágenes para representar la animación en movimiento hacia la derecha e izquierda (imagenDerecha e imagenIzquierda). También cuenta con propiedades de estado como estaApoyado, y dirección para determinar hacia qué lado irá su movimiento horizontal.

Los métodos principales de la clase Gnomo incluyen:

- mostrar: Dibuja la imagen del gnomo en pantalla, dependiendo de si se mueve a la derecha o a la izquierda.
- movVertical: controla el movimiento vertical hacia abajo cuando no está apoyado en una isla, simulando su caída.
- movHorizontal y cambiarDireccion: Definen el movimiento horizontal y permiten cambiar la dirección del gnomo de manera aleatoria.
- *getters* y *setters*: para obtener los bordes del gnomo, facilitando la detección de colisiones.

La combinación de estos métodos y atributos permite que los gnomos se muevan de forma realista agregando dificultad para rescatarlos.

Tortuga, los villanos de la historia

La clase Tortuga representa a los enemigos del juego, los cuales atacan a los gnomos. Cada instancia de tortuga tiene atributos de posición (x,y), dimensiones (ancho, alto) y un desplazamiento que determina su velocidad. Cuenta con propiedades de estado como estaApoyado. Y además, se utilizan imágenes para mostrar su animación en ambas direcciones (imagenDer e imagenIzq).

Los métodos principales de tortuga incluyen:

- mostrarTortugas: Dibuja la imagen de la tortuga en pantalla segun la direccion en la que se mueva (derecha o izquierda)
- moverHorizontalmente y rebote: Controlan el desplazamiento horizontal de la tortuga y permiten invertir su dirección cuando choca con un límite.
- movVertical: Permite que la tortuga se caiga si no está apoyada.
- *getters* y *setters*: para los bordes de la tortuga o sus atributos de posición.

Estos métodos permiten un comportamiento que enriquece la mecánica del juego.

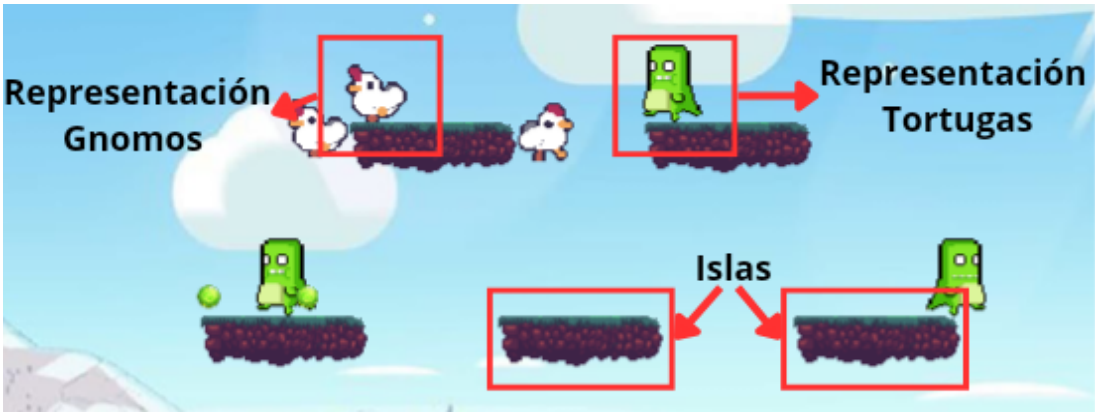
Isla, estructuras flotantes donde habitan los gnomos

La clase Isla representa las plataformas flotantes en el juego, sobre las que se podrán apoyar los personajes. Cada isla tiene su posición (x,y), dimensiones (ancho, alto) y una escala para ajustar su tamaño en pantalla. La isla se representa visualmente mediante una imagen.

Los métodos de isla incluyen:

- mostrar: Dibuja la isla en la posición actual utilizando la imagen asignada.
- *getters* y *setters*: Para obtener los bordes de la isla y sus posiciones.

Estos métodos permiten que la clase Isla funcione como base de apoyo para personajes, contribuyendo a la interacción del entorno.



Representación visual de Gnomos, Tortugas e Islas.

Disparo de Pep, el poder del sol

La clase Disparo de Pep representa el proyectil que el personaje Pep lanza en el juego, es mortal para las tortugas. El disparo tiene atributos de posición (x,y), dimensiones (ancho, alto), y una imagen que lo representa visualmente como una bola de fuego. La velocidad del disparo se define mediante el atributo velocidadDisparo, lo cual permite que el proyectil se desplace en pantalla. Además, la dirección del disparo se controla con el atributo mirandoDerecha.

Los métodos clave de Disparo de Pep incluyen:

- mostrar: Dibuja el disparo en su posición actual.
- movimientoDisparo; Ajusta la posición del disparo en función de su dirección, ya sea izquierda o derecha.
- *getters* y *setters*: Para calcular y definir los bordes del disparo y sus coordenadas x,y.

Estos métodos aseguran que el disparo se visualice correctamente y se mueva en el juego, cumpliendo su rol de ataque por parte de Pep.

Disparo de Tortuga, su habilidad de ataque

La clase Disparo Tortuga representa el disparo lanzado por las tortugas en el juego, el cual es mortal tanto para Pep como para los gnomos. Funciona de manera muy similar al disparo de Pep, contiene atributos de posición (x,y), dimensiones (alto,ancho), velocidad, y una imagen (disparoTortuga) que representa visualmente el proyectil. También cuenta con un atributo que define su dirección (mirandoDerecha),

Los principales métodos de DIsparo Tortuga son:

- mostrar: Dibuja el disparo en pantalla en su posición actual.
- dispararDerecha y dispararIzquierda: Controlan el movimiento del disparo en la dirección correspondiente según la velocidad.
- *getters* y *setters*: Calculan y establecen los bordes y coordenadas del disparo.

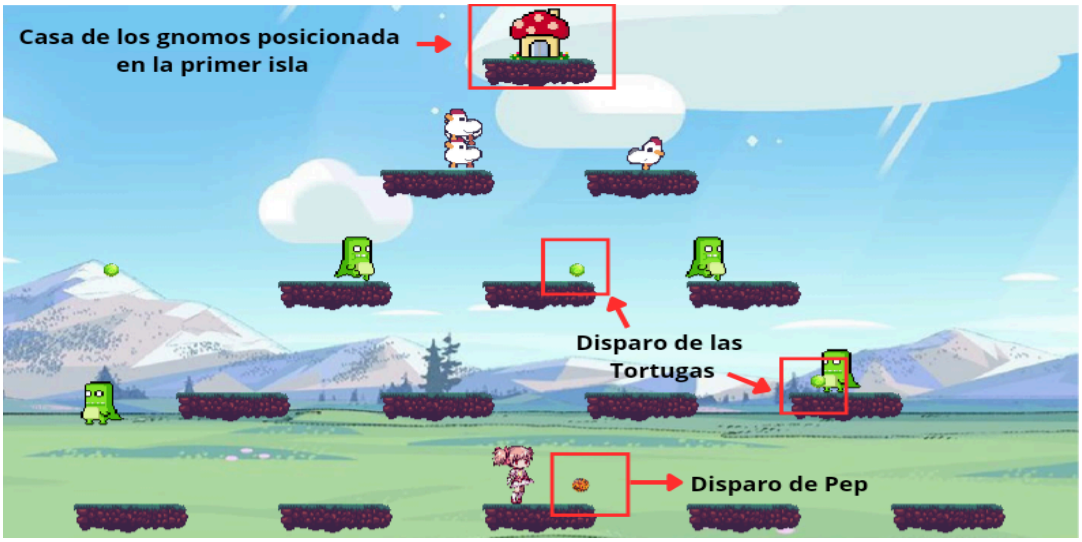
Estos métodos permiten que el disparo de la tortuga se mueva de manera autónoma y se visualice correctamente en el entorno del juego.

Casa Gnomos, lugar de origen de los gnomos

La clase casa Gnomos representa la casa de los gnomos en el juego, de la cual saldrán los gnomos y es un lugar inaccesible para las tortugas. Contiene atributos para la posición (x,y), dimensiones (alto, ancho) y bordes. Además, cuenta con una imagen que representa al objeto visualmente.

Los principales métodos de casa Gnomos son:

- mostrar: Dibuja la casa en una posición especificada en el entorno.
- *getters* y *setters*: Devuelven las coordenadas y dimensiones de la casa.



Representación visual de la casa de los gnomos, disparos de tortugas y disparos de Pep.

Controlador Colisiones

La clase Controlador Colisiones gestiona la detección de colisiones entre diferentes objetos del juego, como personajes (Pep, Gnomo,Tortuga) y elementos del entorno (Isla, Disparos). Proporciona métodos específicos para verificar las colisiones y los límites de la pantalla.

Controlador Pantallas

La clase Controlador Pantallas se encarga de gestionar y mostrar las imágenes para los diferentes estados del juego tales como la imagen de fondo del juego, victoria o derrota. Tiene atributos de posición (x,y). También cuenta con imágenes para cada estado de juego.

Sus métodos son:

- MostarPantallaInicial: Se muestra la imagen de fondo donde se transcurrirá el juego, a su vez por texto informa cuantos gnomos fueron salvados, cuantos gnomos se perdieron y la cantidad de enemigos eliminados.

- MostrarPartidaGanada: Dibuja la imagen de fondo de victoria en la posición x,y junto con un mensaje indicando la victoria, el tiempo transcurrido y la cantidad de gnomos salvados y perdidos y la cantidad de enemigos eliminados.
- MostarPartidaPerdida: Aparecerá un mensaje informándonos que perdimos el juego, del mismo modo que el método anterior se informarán los datos de la partida.

Pantalla de juego, pantalla de partida ganada, pantalla de partida perdida.

Reloj

La clase Reloj se encarga de manejar y mostrar el tiempo transcurrido en el juego. Contiene atributos para horas, minutos y segundos.

Métodos de Reloj:

- Mostrar: Actualiza y muestra el tiempo transcurrido en formato HH:MM:SS en pantalla.
- CongelarTiempo: Detiene el reloj al momento de perder o ganar el juego.
- MostrarTiempoCongelado: Imprime por pantalla el tiempo transcurrido hasta el momento de ganar o perder la partida.

Estos métodos permiten tener un seguimiento del tiempo transcurrido en el juego.

2. Logica y programacion en la clase Juego

La clase Juego implementa la lógica del juego en un entorno gráfico utilizando la biblioteca entorno. Aquí es donde se hace uso de las clases y sus métodos mencionados anteriormente y también se implementan nuevos métodos que relacionan a los objetos en el entorno.

En sus atributos se encuentra el entorno y se llama a los objetos del juego, en algunos casos, se hace arreglos de objetos para que se muestran varios en pantalla:

- Pep, el personaje manejado por el usuario.
- Disparo de Pep, el poder de Pep.
- Tortuga[] Arreglo que contiene a las tortugas que atacan.
- Gnomo[] Arreglo que contiene a los gnomos que deben ser rescatados.
- Isla [] Arreglo que contiene las islas en las que se moverán los personajes.
- Casa Gnomos, de donde aparecerán los gnomos.
- Reloj, para manejar el tiempo de juego.
- Controlador Colisiones, encargado de gestionar las colisiones entre objetos con otros objetos o los bordes de la pantalla.
- Controlador de Pantalla, utilizado para mostrar imágenes y textos.

También se definen atributos propios de juego que serán utilizados en los modos de Juego, tales como contadores de enemigos eliminados, gnomos salvados y gnomos perdidos y cantidades máximas de gnomos (4) y tortugas (4) que pueden aparecer en pantalla a la vez.

En el constructor de la Clase juego se inicializan el entorno y los objetos mencionados anteriormente. En el caso de las Islas, se recorre el Array de uno en uno para ubicar cada isla de manera correcta en el entorno, deben ser 5 filas ubicadas en forma de pirámide y los pisos inferiores deben cubrir los espacios vacíos que hay en pisos superiores. Una vez inicializado todo lo necesario, se inicia el juego.

Método tick

El método tick es el ciclo principal del juego, se ejecuta repetidamente para actualizar el estado de los elementos en pantalla. Durante el tick se recorren los métodos de Juego que verifican las acciones de los objetos y cómo se relacionan entre ellos.

Los métodos que se llaman durante cada tick son: mostrarPantallaInicial(), chequearColisiones(), chequearTeclas() , verificarMovimientos(), mostrarObjetos(), spawnObjetos(), verificarSiGanaOPierde(). Simplificar el contenido de tick solo llamando a otros métodos de Juego hace que el código sea más fácil seguir y se mantenga organizado.

El método **mostrar pantalla inicial** es el encargado de mostrar la cantidad de gnomos salvados, los gnomos perdidos y los enemigos eliminados, y en la parte inferior el tiempo transcurrido.

El método **chequear colisiones** determina las colisiones entre objetos comprobando si ocurren las colisiones definidas en la clase Controlador Colisiones.

Se verifican colisiones de:

- Pep con islas: Define si Pep está parado sobre una isla o no para establecer el atributo de Pep “está apoyado” en verdadero o falso.
- Pep con Gnomos: Si Pep colisiona con un gnomo, el mismo será salvado lo que aumentará el contador de gnomos salvados en 1. Si Pep colisiona con un gnomo, el gnomo se vuelve *null*, por lo tanto se deja de mostrar en pantalla. Pep solo puede salvar gnomos a partir del tercer piso, por lo que la colisión será válida sólo si se encuentran en el lugar adecuado.
- Pep con tortugas: Si Pep colisiona con una tortuga, muere. Por lo tanto Pep se vuelve *null* y el juego no continua.
- Pep con disparos de tortugas: Al igual que el punto anterior, si Pep colisiona con un disparo de tortuga, Pep muere, se vuelve *null* y el juego no continua.
- Pep con bordes: si Pep sale fuera de la pantalla también se vuelve *null* y termina el juego.

- Gnomos con islas: Define si el gnomo está parado sobre una isla para establecer su atributo “está apoyado” en true o false
- Gnomos con tortugas: Al igual que Pep, si un gnomo colisiona con una tortuga morirá, por lo que debe ser *null* y aumentar el contador de gnomos perdidos en 1.
- Gnomos con disparos de tortugas: Igual que el punto anterior, si un gnomo es alcanzado por un disparo de tortuga morirá y será *null*.
- Gnomos con bordes: Si un Gnomo toca los bordes de la pantalla, caerá al vacío y morirá, por lo tanto ese gnomo tiene que volverse null y aumentar el contador de gnomos perdidos en 1.
- Tortugas con Islas: Al igual que sucede con Pep y los gnomos, verifica si una tortuga está sobre una isla, y define si está apoyada o no.
- Tortuga con bordes de la isla: Una vez que establecimos que la tortuga está apoyada en una isla y se mueve, al tocar el borde de esta misma va a cambiar de dirección.
- Tortugas con disparos de Pep: si una tortuga es alcanzada por un disparo de Pep, la tortuga morirá, por lo que debe volverse null.
- Tortugas con bordes: si una tortuga sale de los bordes del entorno cae al vacío, por lo que se vuelve null.
- Disparo de tortuga con disparo de Pep: Si estos colisionan, ambos se vuelven null.
- Disparo de tortuga/disparo de Pep con bordes: Si alguno de estos llega a los límites de la pantalla, se vuelve null.

El método **chequear teclas** verifica que teclas presionar el jugador para controlar a Pep mediante el llamado a métodos de entorno. Mientras las teclas izquierda o derecha estén apretadas, Pep se moverá hacia esas direcciones respectivamente.

Si se presiona la tecla de arriba o la tecla “w” y Pep está apoyado sobre una isla (es decir, su atributo está apoyado es true), se llama al método de Pep iniciarSalto()

Si se presiona la tecla “c” y no hay ningún disparo de Pep en pantalla en ese momento, se creará un nuevo disparo de Pep que se moverá de forma autónoma según sus métodos definidos.

El método **verificar movimientos** gestiona los movimientos de distintos personajes:

- Pep: Se añade el movimiento vertical hacia abajo si Pep no está apoyado haciendo uso de su método movVertical(), o hacia arriba haciendo uso de su método saltar(). También se gestiona su movimiento horizontal si las teclas izquierda o derecha están apretadas.
- Disparo de Pep: Se ejecutará el método movimientoDisparo() en la última dirección en la que miró Pep.
- Gnomos: Se añaden sus movimientos vertical hacia abajo si no están apoyados haciendo uso de su método movVertical(), horizontal hacia derecha o izquierda si están apoyados y luego de hacer colisión con una isla cambian de dirección de manera aleatoria con el método cambiarDireccion().
- Tortugas: se añade movimiento vertical hacia abajo cuando no está apoyada haciendo llamado a su método movVertical(). Al estar apoyada la tortuga se va a mover de forma horizontal con el método moverHorizontalmente().
- Disparo de Tortuga: Dependiendo de la dirección en la que esté mirando cada tortuga se utilizará dispararDerecha() o dispararIzquierda() para avanzar el disparo.

El método **mostrar objetos** se encarga de llamar a la función mostrar de cada uno de los objetos, en el caso de tortugas, gnomos e islas, recorre el array para mostrar a todos los que existan.

El método **spawn objetos** hace que siempre haya la cantidad necesaria de objetos en pantalla, generando nuevos objetos si alguno se hace null.

En el caso de los gnomos, recorre el arreglo y si alguno de ellos es null genera una posición dentro de los límites de la casita de gnomos y crea un nuevo gnomo en esas coordenadas.

En el caso de las tortugas también recorre el arreglo y si alguna de las tortugas es null, genera una posición válida (las tortugas no pueden aparecer sobre la casita de los gnomos) con una distancia mínima entre ellas para que no aparezcan pegadas o en el mismo lugar. Luego crea una nueva tortuga en la posición generada.

En ambos casos, para generar una posición aleatoria se hace uso del método random de Java.

El método **verificar si gana o pierde** hace uso de la clase controlador pantallas para mostrar diferentes cosas en pantalla dependiendo si el usuario ganó o perdió. Si el contador de gnomos salvados supera los 15 gnomos, se considera una victoria por lo que se mostrará en pantalla que el usuario ganó el juego. Si Pep se vuelve null por algún motivo, significa una derrota por lo que se mostrará en pantalla que el usuario perdió el juego.

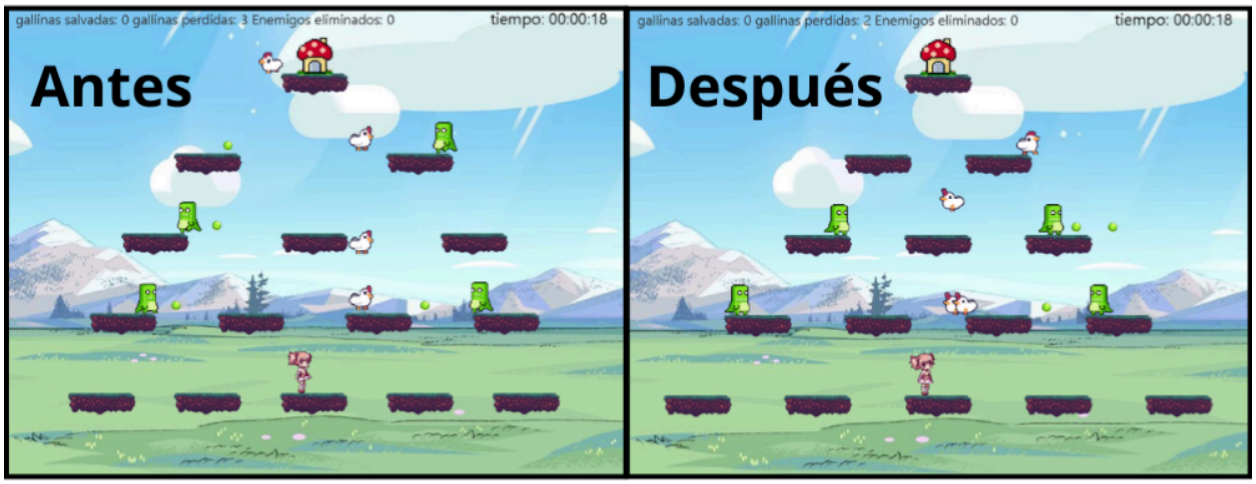
Todos estos métodos que conforman la estructura de Juego permiten que Pep navegue por el entorno y rescate a los gnomos en las islas flotantes, evadiendo o eliminando a las tortugas enemigas.

3. Desafíos técnicos y soluciones implementadas

Una de las dificultades que se nos presentó al crear el proyecto, surgió al darle movimiento a las tortugas. Uno de los requerimientos pedía que al llegar al borde de la isla la tortuga cambiará de dirección, el problema que apareció fue que el personaje al llegar a un borde cambiaba su dirección constantemente y no avanzaba. La solución se le dio para que eso no suceda fue que antes que cayera la tortuga a la isla restarle su borde a ambas, luego al estar apoyada aumentar el borde de la isla y así lograr un rebote exitoso.

Funcionamiento de bordes reducidos.

Otro problema que tuvimos a la hora de desarrollar el proyecto fue lograr una posición correcta de las islas. En un principio se pensó en utilizar un ciclo for que recorra las islas para ubicarlas en forma piramidal aumentando en 1 la cantidad de islas que había en cada piso y acomodándose de acuerdo al ancho del entorno y a la distancia deseada entre ellas. Pero esto no las situaba de forma correcta llenando los espacios vacíos, en el caso del segundo y tercer piso, las islas quedaban tan separadas que los gnomos salían del primer piso directamente hacia el cuarto, por lo que la solución encontrada fue manipular las islas de manera individual adecuandolas a lo solicitado para este trabajo.



Ubicación inicial de las islas antes y después del cambio de código.

Como última dificultad encontrada, durante el desarrollo del proyecto fue la necesidad de reposicionar y reorganizar el código en múltiples ocasiones para cumplir con los lineamientos de la materia. Tuvimos que dividir, reestructurar y mover funciones entre clases para asegurar que cada método cumpliera con un propósito específico, manteniendo un flujo de trabajo claro. Este proceso, ayudó a mejorar la organización y legibilidad del código, contribuyendo a un diseño más prolijo y alineado con los objetivos de aprendizaje de la materia.

Conclusión

El desarrollo del videojuego *Al rescate de los Gnomos* fue una experiencia que nos ha permitido aplicar conocimientos de programación orientada a objetos y explorar la lógica necesaria para crear un entorno de juego dinámico en Java. Durante el proceso, enfrentamos varios desafíos técnicos, como la detección y manejo de colisiones, la sincronización de movimientos entre personajes, el diseño de una estructura de clases que permitiera la interacción entre diversos elementos del juego y la necesidad de ajustar y reorganizar el código constantemente que resaltó la importancia de un diseño estructurado y flexible. Cada obstáculo fue una oportunidad para aprender y mejorar nuestras habilidades en programación.

Otro aspecto importante a destacar de este proyecto fue el trabajo en equipo. La delegación de tareas, la organización y el manejo de prioridades fueron muy importantes para poder avanzar. Al dividir las responsabilidades, cada integrante

del grupo pudo enfocarse en áreas específicas como la lógica de juego, clases y sus métodos, el control de colisiones o el apartado visual, lo que permitió una mayor profundidad en cada sección del proyecto.

En resumen, este proyecto no solo permitió la creación de un juego funcional, sino que también reforzó nuestro entendimiento de conceptos clave en programación. La experiencia adquirida nos servirá en futuros proyectos de software, donde la capacidad de planificar, adaptarse y resolver problemas de forma efectiva es muy importante.