Universidad Nacional del Sur

Tecnología de Programación (TdP)

Segundo Cuatrimestre 2024

COMISION 16

Santiago Vilas, Juan Segundo Mosqueira, Juan Sebastian Bassi, Juan Manuel Cristobo, Ignacio Iommi

Proyecto cuatrimestral

Modelado e implementación de una versión simplificada del juego.
Caso de estudio: Super Mario Bros.

Super Mario Bros. es un icónico videojuego de plataformas desarrollado por Nintendo. Lanzado en 1985 para la consola NES, el juego sigue las aventuras de Mario, un fontanero que debe rescatar a la Princesa Peach del malvado Bowser. El juego se desarrolla en el Reino Champiñón, donde Mario debe superar diversos niveles llenos de obstáculos, enemigos y trampas. Con su diseño de niveles ingenioso y su jugabilidad adictiva, Super Mario Bros. se ha convertido en uno de los videojuegos más influyentes y reconocidos de todos los tiempos.

En este proyecto, se desarrolló una versión simplificada de Super Mario Bros. que replica las funciones básicas del juego original. El juego registra las cinco mejores puntuaciones, asignadas a cada jugador tras ingresar su nombre al inicio. Consta de tres niveles de dificultad progresiva que incrementan la complejidad a medida que se avanza. Cada nivel presenta enemigos icónicos del juego original, y el juego culmina al completar el último nivel.

La implementación parcial del videojuego fue realizada en base ciertas condiciones dadas en el siguiente enlace: https://moodle.uns.edu.ar/moodle/pluginfile.php/1733056/mod_resource/content/14/Enunciado-general-proyecto.pdf

---

La implementación se realizó en **Java** (versión 22 o posterior) empleando **Eclipse IDE** (versión 2024-09, 4.33.0). Eclipse facilitó la organización, refactorización y comprensión del código gracias a su robusto soporte para Java, así como a herramientas gráficas y editores visuales para el diseño de interfaces complejas. Además, el entorno incluyó diversas librerías integradas que ayudaron a reducir el tiempo de desarrollo y mejorar la legibilidad del proyecto.

Desarrollo y Funcionalidades = Los archivos se pueden encontrar en GitHub. El source de nuestro código base, consta de ocho paquetes:

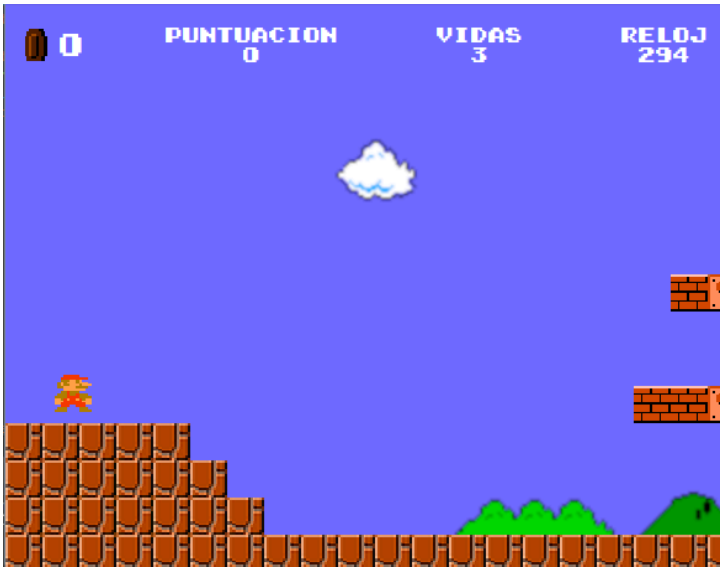
Enemigos: Clases para los enemigos del juego.
Fábricas: Implementa patrones de diseño para la generación de entidades y sprites.
GUI: Gestión de la interfaz de usuario.
Launcher: Configuración y arranque del juego.
Lógica: Clases de lógica del juego, colisiones y estado.
Personaje: Clases de Mario y sus estados.
Plataformas: Clases que representan las plataformas.
PowerUps: Clases para las bonificaciones y potenciadores.

Una vez ejecutado el software, tenemos la opción de elegir entre dos modos de juego, por un lado el juego clásico de Mario Bros, y por el otro una adaptación del juego enfocado a Sonic. La jugabilidad es la misma con la diferencia de que el modo consta de

Sprites distintos al original

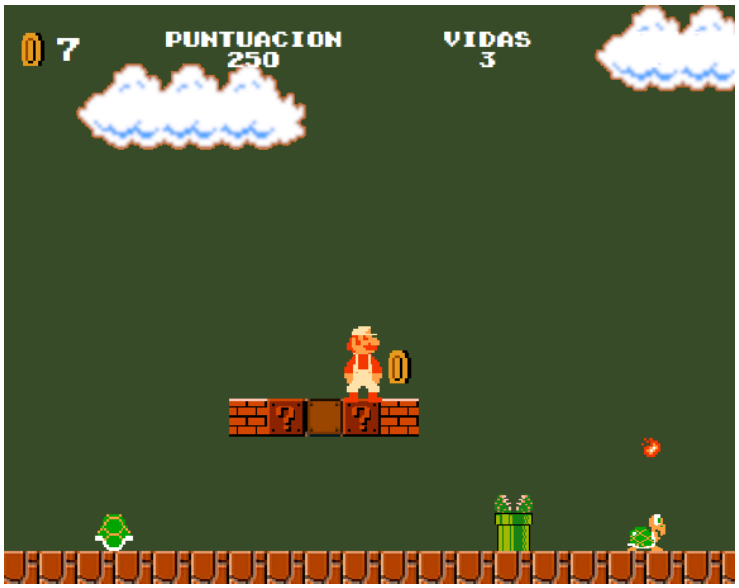
Posterior a la elección del modo de juego el una de las clases implementadas se encargará de transicionar de la pantalla en la cual escoges un modo de juego, a la pantalla inicial, donde puedes escribir un nombre, ingresarlo (comenzará el juego) o ver el ranking.

Mario Bros



Sonic

El juego se iniciara según el modo de juego escogido



Nivel 2 . Estado Flor de Fuego

Nivel 3. Estado Estrella

Gestión de Estados y Colisiones: En el original, la interacción de Mario con los enemigos y objetos está controlada por la consola. Nosotros decidimos optar por una implementación en la cual los estados del personaje y las colisiones son manejados a través de clases Hilos, las cuales son la conexión entre la lógica y la gráfica y estructuras, permitiendo una mayor modularidad y claridad en la implementación de la lógica del juego.

Nivel máximo de dificultad

Patrones Utilizados

- ➔ Visitor
  - ◆ El visitor nos fue muy útil a la hora de manejar a las colisiones ya detectadas. Utilizamos cuatro Visitor distintos, el VisitorEnemigo, VisitorEnemigoAfectado, VisitorEntidad y VisitorBolaDeFuego, gracias a los métodos dentro del mismo los mensajes se enviaban automáticamente y así evitar el instanceOf y preguntar que tipo de enemigo o power up es cada vez que colisionaba con el personaje o con la bola de fuego.

- ➔ Factory Method
  - ◆ Este patrón fue aplicado en el proyecto para manejar la creación de objetos de tipo. Permite agregar nuevos tipos de enemigos, personaje o plataformas con solo implementar una nueva clase de fábrica, sin necesidad de modificar el código existente.Cada fábrica encapsula la lógica de creación de objetos, lo que mantiene la lógica del juego organizada y facilita el mantenimiento.

- ➔ Abstract Factory
  - ◆ Este patrón fue utilizado para el manejo de la ruta de los sprites, ya que dependiendo el modo de juego se asigna la ruta correspondiente al sprite seleccionado.

- ➔ State
  - ◆ El patrón State fue utilizado para representar el comportamiento de entidades como Mario, el Koopa Troopa, la Piranha Plant, y el Bloque de Pregunta. Hacemos que en las clases de las entidades que usan este patrón, se llame al método de la clase del estado de cada uno, para hacer que el comportamiento y la reacción a diferentes llamadas varíe según el estado. Por ejemplo, cada vez que se llame al método recibirDaño() en personaje, este llama al método recibirDaño() de la variable implementada estado de tipo EstadoDePersonaje, que va a estar implementada de diferente forma según el estado del personaje en el que estemos, por ejemplo el Estado Estrella no va reaccionar a recibirDaño(), mientras que el Estado Super Mario se va a encargar de transformar a Mario en Mario Normal (Mario Chiquito).

- ➔ Singleton
  - ◆ Se utilizó en la clase que gestiona la música del juego.Este patrón fue empleado para asegurarse de que solo exista una instancia de esta clase, permitiendo un único punto de acceso al control de la música en todas las pantallas y niveles del juego

---

Movimiento del jugador

El movimiento del jugador se divide en tres secciones la primera se establece la dirección en la cual el jugador está mirando (esto se hace desde el controlador gráfico a través de un keyListener) , la segunda parte se realiza desde el hilo donde éste hace mover al personaje todo el tiempo , y la tercer parte se encuentra en el personaje donde se define todo el comportamiento de “moverse” sea saltar , correr y demás . Cabe destacar que el personaje tiene una Velocidad en base al eje X , permitiéndole “deslizarse” al hacer cambios de dirección bruscos , esto con el fin de obtener una jugabilidad más entretenida y una Velocidad en base al eje Y buscando una gravedad realista de forma progresiva.
A su vez utilizamos una lógica similar para los enemigos , un hilo que los hace mover todo el tiempo y estos tienen su comportamiento definido en su respectiva clase.

CONSIDERACIONES:

Las indicaciones iniciales dadas se cumplieron, sin embargo, existen errores y consideraciones que no tuvieron un desarrollo debido:

- Al iniciar el siguiente nivel, se llama al reseteo para reiniciar a todos los hilos de las entidades: queremos que se llame una vez para no arrastrar memoria basura, pero en casos aleatorios se llama dos veces. Esto genera un bug que duplica la puntuación y las monedas, y hace que podamos presenciar la "recargada" del nivel por segunda vez.

- A la hora de llamar al movimiento lateral, estando en el aire, si el personaje colisiona con alguna plataforma, puede colisionar múltiples veces, haciendo que la gravedad no funcione como se debe. Es decir, si estoy corriendo contra una plataforma/pared, por más de que no tenga piso abajo, si sigo corriendo contra ésta , la velocidad de "caída" o gravedad no es la buscada.

- En el estado de fuego del personaje, encontramos un comportamiento en el que si disparamos reiteradas veces en poco tiempo, dejando que se acumulen múltiples bolas de fuego en la pantalla, el sprite de Mario puede cambiarse erróneamente al sprite gif de la bola de fuego viajando.

- En el Ranking se cargan todos los jugadores pero como bien dice el enunciado y se ordenan los Labels de jugadores de mayor a menor puntaje. El ranking solo se actualiza cuando se cierra el juego, sea en pleno juego, se alcanza a guardar la puntuación vigente o cuando se pierde, pero siempre que se cierra el frame se guarda y actualiza, por lo que si al ganar el juego se quiere ver el ranking en la Pantalla Victoria y no se muestra nuestro puntaje hay que cerrar y volver a abrirlo y ahí en caso de estar en el top 5 aparecerá nuestro nombre. Por otra parte adoptamos la convención de que si no se ingresa ningún nombre se carga el nombre del jugador como "Jugador" por default. En caso de pullearlo y estar vacío, hay que jugar mínimo una vez para que se comience a cargar dentro del archivo score.tdp

- Si se llega al final del último nivel con una sola vida, hay probabilidades de que se muestre la Pantalla Perder. Pasa en situaciones aleatorias por lo que por falta de tiempo no pudimos solucionar este error.

- Las colisiones si bien son funcionales , hay ciertos casos como por ejemplo en estado SuperMario , al saltar y luego aterrizar suele pasar que visualmente

quede con los pies dentro del bloque . La solución propuesta de empujarlo fuera del bloque causaba que este rebote indefinidamente (por la gravedad), por lo que se optó por dejarlo así.

- Lo mismo sucede al saltar alto , cuando aterrizamos notamos una gran reducción de velocidad de movimiento lateral causado por quedar dentro del bloque y moverlo fuera.

- Luego de matar enemigos rara vez podemos notar que no se termina de eliminar su sprite y se lo puede llegar a ver muerto en vacío.