

Trabajo práctico: Juego Súper

Elizabeth Sis

Alumnos:

- Marcelo Palacios

N° de legajo: 31625157/17

Mail: Jpdjcp@gmail.com

- Tomás Artaza

N° de legajo: 41146971/18

Mail: tomasartaza25@gmail.com

- Bogado Diana

N° de legajo: 43326653/18

Mail: dianadiazbogado@gmail.com

Introducción

En el presente trabajo hicimos el desarrollo del juego Súper Elizabeth Sis con sus correspondientes métodos, funciones, clases y librerías exportadas, que más adelante serán explicadas.

El juego cuenta con sonido, objetos con movimiento y un menú, todo en torno a la historia propuesta en la consigna.

Se intentó hacer el trabajo con un código lo más prolijo y legible posible y el juego lo más didáctico y fácil de entender.

Clase Juego

Contiene las variables principales usadas para referenciar a todos los objetos del juego y las llamadas a los métodos principales que controlan los comportamientos e interacciones de todos los objetos.

Clase Menu

Esta clase se encarga de moldear el menú o pantalla principal del juego, donde se muestra el título del juego y se le indica al usuario qué tecla presionar para iniciar el juego. Por lo tanto, es la clase a ejecutar para que el juego inicialice.

Variables de instancia e Invariante de representación

- Entorno app: esta variable hace referencia a la ventana del menú o pantalla de inicio. Solo puede referenciar a un objeto Entorno.

- Fondo fondo: esta variable referencia a un objeto Fondo que contiene una imagen de fondo y otros elementos que se dibujan en pantalla, como el personaje o el título del juego. Solo puede referenciar a un objeto Fondo.
- Clip musicaMenu: esta variable contiene la música de la pantalla de inicio. Solo puede hacer referencia a un archivo de audio.

Métodos de la clase Menu

- Menu: es el constructor del objeto.
- Tick: llama a todos los métodos en cada instante de tiempo mientras no se inicie el juego.
- Main: el método principal de package.

Clase PrincesaPikachu

Esta clase moldea al personaje protagonista del juego.

Variables de instancia e Invariante de representación

- Double x: es la variable que referencia al valor de la posición sobre el “eje x”. Solo puede tener valores superiores $x = 0$ o un poco más, para que el personaje no se salga de la ventana. Y $x = 400$ o un poco menos, para que el personaje no invada la mitad derecha de la ventana.
- Double y: es la variable que referencia al valor sobre el “eje y”. Para que el personaje no “traspase el suelo” y se caiga de la ventana, tiene un máximo de

$y = 525$. El salto no le permitiría llegar a $y = 0$, pero ése sería su “techo” para que no se sal de la ventana por la parte superior.

- Double ancho: el ancho del personaje. El ancho debe ser mayor que cero, de lo contrario, el personaje no “existiría”. Tampoco puede ser demasiado ancho, ocupando toda la ventana del juego. El valor del ancho es constante.
- Double alto: el alto del personaje. Debe ser mayor que cero y no puede ser mayor que el alto de la ventana. El valor del alto es constante.
- Image img, img2, img3, img4: estas variables hacen referencia a las distintas imágenes usadas para mostrar al personaje, en distintas situaciones, ya sea durante el juego, cuando gana, cuando pierde.
- Boolean salto: esta variable guarda el estado del personaje, es decir, si está saltando o no. Solo puede ser True o False según corresponda.
- Double inercia: esta variable hace referencia al impulso que lleva el personaje cuando salta y que va “disminuyendo” a cada instante hasta que comienza a caer y vuelve al suelo. Le asignamos un “pico” de -9 que es lo que se desplaza sobre el “eje y” en el primer instante del salto, y va creciendo hasta que el personaje llega al nivel del suelo y la inercia se setea otra vez en -9 esperando a que Boolean salto sea otra vez True.
- Boolean vulnerable: esta variable permite o no que el personaje reciba más daño. Por ejemplo, al atravesar un obstáculo, el personaje “tocaría” el obstáculo en cada instante del tick, descontando varias vidas por segundo. Esta variable es para que al tocar el obstáculo, “vulnerable” cambie a false y ya no reciba más daño hasta que haya atravesado completamente dicho obstáculo.

Métodos de la clase PrincesaPikachu

- dibujarPersonaje: muestra al personaje en la ventana.
- dibujarGano: muestra la versión del personaje celebrando la victoria.
- dibujarPerdio: muestra la versión del personaje cuando perdió.
- dibujarGato: muestra al gatito rescatado por el personaje.
- siChoca: pregunta si el personaje tocó algún soldado u obstáculo. Si los tocó, devuelve true, sino false.
- tocaObstaculo: método auxiliar que detecta las colisiones entre el personaje y cada uno de los obstáculos.
- tocaSoldado: método auxiliar que detecta las colisiones entre el personaje y los soldados.
- Impulso: controla el impulso “ascendente” y “descendente” del personaje cuando salta.
- avanzar: desplaza el personaje hacia la derecha de la ventana.
- retroceder: desplaza el personaje hacia la izquierda de la ventana.
- disparar: crea un nuevo objeto Fireball cada vez que el jugador presiona ESPACIO y lo guarda en el arreglo correspondiente.
- setSalto: cambia la variable Boolean salto al valor del parámetro.
- esVulnerable: indica si el personaje puede recibir daño o no.
- setVulnerable: cambia el estado de vulnerabilidad a true o false según el parámetro.

Clase Soldado

Esta clase modela a los soldados de la patrulla del maltrato animal, que son los enemigos de nuestro protagonista.

Variables de instancia e Invariantes de representación

- Double x: posición en x. Debe mantenerse en valores dentro de la ventana del juego o próximos a ella, cuando salen por la izquierda o entran por la derecha.
- Double y: posición en y. Se mantienen al nivel del suelo.
- Double ancho: el ancho del soldado. Debe ser un valor positivo, ya no existe un ancho menor o igual que 0. Este valor es fijo.
- Double alto: el alto del soldado. Debe ser un valor positivo. El alto es fijo.
- Boolean avanza: indica si el soldado avanza hacia el personaje o retrocede porque se chocó con un obstáculo.
- Boolean ignoraToque: a veces el soldado se superpone con el obstáculo y cambia de dirección de desplazamiento a cada instante de tiempo. Esta variable se incorporó para deshabilitar el cambio de dirección hasta que el soldado deje de superponerse con el obstáculo. Es true cuando el soldado tocó un obstáculo y false cuando dejó de tocarlo.
- Image img: esta variable solo puede referenciar un archivo de imagen que representa a los soldados.

Métodos de la clase Soldado

- Soldado: constructor del objeto Soldado.
- Inicializar: inicializa todas las posiciones del arreglo con nuevos soldados cuando empieza el juego.

- Dibujar: recibe un arreglo de soldados y muestra en pantalla a cada uno.
- Mover: recibe un arreglo de soldados y los desplaza sobre el “eje x”.
- soldadoChocaObstaculo: recibe un arreglo de soldados y otro de obstáculos y pregunta si hubo algún contacto entre ellos y devuelve true si lo hubo o false en caso contrario.
- tocaObs: recibe un obstáculo y detecta si se “toca” con “this” soldado.
- cambiaDireccion: cambia la variable boolean “avanza” al valor contrario.
- Reaparece: recibe un arreglo de soldados y si encuentra una posición con null, crea un nuevo soldado. También detecta si un soldado salió de la ventana por la izquierda o se alejó mucho por la derecha y lo elimina con null para que en el siguiente instante se cree uno nuevo en esa posición del arreglo. Al momento de crear el soldado, busca el x más grande de todos los soldados y le suma un número random entre 800 y 1200. Esto garantiza que los soldados se creen suficientemente espaciados entre sí, que se creen fuera de la vista del jugador y que la distancia entre los mismo vaya variando.
- xRandom: es un método auxiliar que genera un valor random entre 800 y 1200 (o 1199, si el random va de 0 a 399).
- maximoX: es otro método auxiliar que busca el “x” más grandes de todos los soldados para tomarlo de referencia al momento de crear un nuevo soldado.
- Mover: suma o resta sobre el “eje x” para desplazar al soldado hacia la derecha o izquierda.

Clase Obstaculo

Esta clase modela los obstáculos que le van apareciendo a nuestro protagonista a medida que avanza.

Variables de instancia e Invariantes de representación

- Double x: la posición sobre el “eje x” del obstáculo. Los valores se mantienen cercanos al 0 y al 800, que son los márgenes de la ventana.
- Double y: posición sobre el “eje y”. Estos valores son fijos.
- Double ancho: el ancho del obstáculo. Debe ser un valor positivo. El valor del ancho es fijo.
- Double alto: el alto del obstáculo. Debe ser un valor positivo. El valor del alto es fijo.
- Image img: esta variable hace referencia a la imagen que representa a los obstáculos.

Métodos de la clase Obstaculo

- Obstaculo: constructor del objeto Obstaculo.
- Inicializar: recibe un arreglo e inicializa en todas sus posiciones un nuevo obstáculo.
- Dibujar: recibe un arreglo y dibuja cada obstáculo del mismo.
- Mover: recibe un arreglo de obstáculos y los desplaza sobre el “eje x”.

Clase FireBall

Esta clase modela a las bolas de fuego o energía que dispara el personaje.

Variables de instancia e Invariantes de representación

- Double x: coordenada x. Debe mantenerse dentro de los márgenes de la ventana, 0 y 800.
- Double y: coordenada y. Debe tener la misma “altura” que PrincesaPikachu para aparentar que el personaje ha disparado la Fireball.
- Double diam: diámetro de la Fireball. No puede ser un valor menor o igual que 0. Tampoco puede ser demasiado grande, de manera que tape toda la pantalla. El diámetro es fijo.
- Clip disparo: esta variable hace referencia al archivo de sonido que se escucha cada vez que se dispara una Fireball.
- Clip explosion: referencia al clip de sonido que se escucha cada vez que la FireBall impacta a un soldado.
- Image img: referencia a la imagen que representa a la Fireball.

Métodos de la clase Fireball

- Fireball: constructor del objeto.
- Dibujar: dibuja en pantalla la imagen que representa a la Fireball.
 - tocaSoldado: recibe un arreglo de soldados y pregunta si la Fireball tocó algún soldado. Si hubo un toque, reproduce el sonido de explosión, borra a ese soldado con null y devuelve true para que se sumen los 5 puntos en el método tick de la clase Juego.

- `tocaSold`: es un método auxiliar que recibe un soldado, detecta si hubo un toque entre el soldado pasado por parámetro y la Fireball y devuelve true en ese caso. En caso contrario, devuelve false.
- `tocaObstaculo`: recibe un arreglo de obstáculos y pregunta si la Fireball tocó algún obstáculo. Si hubo un toque, devuelve true y la Fireball se borra con null en el método tick. Si no hubo toque, vuelve con false.
- `tocaObs`: es un método auxiliar que recibe un obstáculo y detecta si se tocó con la Fireball. En ese caso, vuelve con true. Caso contrario, vuelve con false.
- `Mover`: desplaza la Fireball sobre el "eje x".

Clase Escenario

Esta clase controla tanto el paisaje como el suelo que, en su conjunto, conforman el escenario o escenografía del juego. Son varias imágenes que se desplazan y van saliendo de pantalla, mientras la siguiente va cubriendo su lugar. La velocidad del paisaje es más lenta que la del suelo porque está más lejos. Con esto buscamos crear cierta sensación de profundidad.

Variables de instancia e Invariantes de representación

- `Double xPaisaje`: la coordenada x del paisaje. Se mantiene en un rango cercano a los bordes de la ventana, entre 0 y 800.
- `Double xSuelo`: coordenada x del suelo. Se mantiene en un rango cercano a los bordes de la ventana, entre 0 y 800.

- Double ySuelo: coordenada y del suelo. La “altura” es fija cerca del borde inferior de la ventana.
- Image imgSuelo: referencia a una imagen que representa al suelo.
- Image imgPaisaje: referencia al paisaje que está de fondo durante el juego.

Métodos de la clase Escenario

- Escenario: constructor del objeto Escenario.
- inicializarEscenario: recibe un arreglo y lo completa con varios escenarios nuevos.
- dibujarPaisaje: recibe un arreglo de escenarios dibuja las imágenes de fondo.
- dibujarSuelo: recibe un arreglo de suelos y los dibuja.
- moverPaisaje: desplaza las imágenes de fondo sobre el “eje x”.
- moverSuelo: desplaza las imágenes del suelo sobre el “eje x”.

Clase Fondo

Esta clase se ocupa de mostrar los fondos o pantallas de Game Over, Victoria y los marcadores de vidas y puntos.

Variables de instancia e Invariantes de representación

- Image imagenNombreJuego: referencia a la imagen del título del juego.
- Image imagenGanaste: referencia a una imagen con un letrero de “¡Ganaste!”.
- Image imagenMenu: referencia a la imagen de fondo en la pantalla de menú principal.
- Image ImagenFondoGano: la imagen de fondo que se muestra cuando se gana el juego.
- Image imagenGO: referencia a la imagen de Game Over.


Métodos de la clase Fondo


- Fondo: constructor del objeto Fondo.
- dibujarMenu: dibuja el menú principal.
- dibujarGO: dibuja la imagen de fondo de la pantalla de Game Over.
- dibujarNombreJuego: dibuja el título del juego en el menú principal.
- dibujarNombreGanaste: dibuja el letrero de “¡Ganaste!” en la pantalla de victoria.
- dibujarFondoGano: dibuja la imagen de fondo en la pantalla de victoria.
- mostrarVictoria: se encarga de dibujar la pantalla de victoria, con su correspondiente música y la opción de iniciar una nueva partida.
- mostrarGameOver: dibuja la pantalla de Game Over, con su correspondiente música y la opción de iniciar una nueva partida.
- mostrarPuntos: muestra los puntos obtenidos durante el juego.


- mostrarVidas: dibuja el indicador de vidas restantes.


RESUMEN DE PROBLEMAS ENCONTRADOS Y SOLUCIONES

Durante el trabajo tuvimos problemas con:

 el sonido: al principio usamos las ventajas que nos proporcionaba Herramientas dentro del entorno. Pero, nos mostraba error por consola. Para poder resolverlo importamos una de las librerías de java llamada javax.sound.sampled.Clip, esta nos permitió incorporar sonidos en .wav para poder agregar todos los efectos de sonido a nuestro juego: música del menú, juego y otras pantallas, disparos, entre otros.

 imágenes: el inconveniente que surgió fue la superposición de las mismas. A la hora de colocar las imágenes se encontraban mal posicionadas por pantalla, ej: la imagen del Juego tapaba la de los personajes. La solución fue colocar en orden cada dibujador, es decir, primero se dibuja el fondo, luego los personajes y demás imágenes.


 pantallas: luego de crear la clase Menu, quisimos incorporar las clases GameOver y Victoria para poder mostrar una nueva pantalla con sus respectivas características. Una vez hecho ésto, el error apareció a la hora de colocar estas clases en el tick() de la clase Juego. Las pantallas no se cerraban y aparecían infinitamente, por lo que utilizamos el método removeNotify(), del Entorno, para poder cerrarlas. Esto no funcionó y nos daba error por consola. La solución, fue crear una nueva clase llamada Fondo, la cual no recibe parámetros. En ella se dibuja cada pantalla, el removeNotify() las cierra luego de que el usuario genera la acción correspondiente e inicializa un nuevo juego.


 **Cómo hacer que el juego comience:** tenía el problema de hacer que el juego supiera cuándo representar el juego, el menú o las pantallas de Victoria o Game


Over. Lo solucioné agregando un if (empezo) que era true solo cuando el jugador presionara la tecla ESPACIO. Dentro de ese bloque, estaban las llamadas a todos los métodos de dibujar los objetos, moverlos y revisar las interacciones entre ellos. También incorporé los booleanos gano y perdio para que el programa supiera cuándo mostrar sus respectivas pantallas. Incorporé varios booleanos para mejorar la legibilidad, ya que se podía hacer lo mismo con menos y considerando los puntos y las vidas.

🎮 **if (estaPresionada(TECLA_ARRIBA)):** tenía el problema de hacer el código legible, algo del estilo "se se presionó tecla arriba, saltar". El problema es que el personaje solo se movía si en ese instante del tick la tecla arriba había sido presionada. Si no era así, el personaje se quedaba estático "en el aire" y no continuaba desarrollando su salto hasta que no se presionara otra vez la tecla arriba. Lo solucioné incorporando una variable de instancia a la clase PrincesaPikachu que guardara su estado salto = true o salto = false. De esta manera el if(sePresiono(TECLA_ARRIBA)) solo cambiaba el valor de este booleano y el método impulso() continuaba el salto en cada instante del tick mientras salto fuera true.

🎮 **Problema de superposición de soldados:** al principio tenía el problema de que los soldados se superponían. Había probado un método que detectara esta superposición después de crear el último soldado y si había una superposición o estaba por debajo de una distancia mínima, ese soldado creado se volvía a crear con unas coordenadas random nuevas. No funcionaba del todo bien. A sugerencia de la profesora, cambiamos esta idea por la de buscar el x más grande de todos los soldados y a partir de allí, crear un nuevo soldado con una separación mínima para que el personaje pudiera saltar entre ellos. Además se le sumaba un valor aleatorio para que la distancia entre soldados fuera variando.

 **Detección de colisiones:** Al principio elegí que los soldados y el personaje fueran "círculos" y calculaba la distancia entre sus centros con el teorema de Pitágoras más el radio de ambos círculos. Pero después notamos que un rectángulo se ajustaba mejor a la forma que tenían los soldados y el personaje en sus respectivas imágenes y cambiamos a Pitágoras por un método más sencillo similar al usado en el juego PseudoPong.

 **Salto del Personaje:** Con respecto al salto, en el enunciado del TP decía que el personaje no podía desplazarse hacia adelante o atrás cuando saltaba. En principio quería utilizar algo del estilo `angulo * Math.py`, pero terminé usando una idea más sencilla y es una variable `Double impulso = -9` que cuando `salto = true`, va subiendo sobre el "eje y". Esta variable impulso va "creciendo" a cada instante del tick, restando velocidad al salto, hasta que comienza a caer. El salto es recto hacia arriba, por lo que tenía que tener una altura y duración suficientes para dar tiempo a los soldados y obstáculos a pasar por debajo del personaje.

 **soldadoChocaObstaculo:** Este método es para que el soldado rebote y cambie de dirección cuando se topa con un obstáculo. No funciona del todo bien, ya que parece detectar bien las colisiones cuando el soldado se mueve hacia la izquierda, pero no siempre cuando se mueve hacia la derecha.

APÉNDICE CON CÓDIGO RELEVANTE

Detecta las colisiones entre el personaje y los obstáculos.

```
private boolean tocaObstaculo(Obstaculo[] obs) {  
    for (int i = 0; i < obs.length; i++) {  
        if (this.x + this.ancha/2 > obs[i].getX() - obs[i].getAncho()/2 &&  
            this.x - this.ancha/2 < obs[i].getX() + obs[i].getAncho()/2 &&  
            this.y + this.ancha/2 > obs[i].getY() - obs[i].getAlto()/2 &&  
            this.y - this.ancha/2 < obs[i].getY() + obs[i].getAlto()/2) {  
            return true;  
        }  
    }  
}
```

```

    }
    return false;
}

```

Controla el salto del personaje

```

public void impulso() {
    if (!this.salto) {
        this.inercia = -9.0; // el impulso inicial
    }
    if (this.salto) {
        this.y += this.inercia; // le suma a y el impulso
        this.inercia += 0.15; // decrece el impulso
    }
    if (this.y >= 510) {          // cuando llega al nivel del suelo
        this.salto = false;      // el mecanismo del salto se desactiva
        this.y = 510;           // para que no se pase ni una décima
    }
}

```

Este método maneja el cambio de dirección de movimiento de los soldados cuando se topan con un obstáculo.

```

public static void soldadoChocaObstaculo(Soldado[] soldados, Obstaculo[] obs) {
    for (int i = 0; i < soldados.length; i++) {
        for (int j = 0; j < obs.length; j++) {
            if ((soldados[i] != null) &&
                (soldados[i].tocaObs(obs[j])) &&
                (!soldados[i].ignoraToque)) {
                soldados[i].ignoraToque = true;
                soldados[i].cambiaDireccion();
            } else if (soldados[i] != null){

```



```

                                soldados[i].ignoraToque = false;
                                }
                            }
                        }
                    }
}

```

Este método controla la reaparición de los soldados cuando son eliminados por el personaje o se salen de pantalla.

```

public static void reaparece(Soldado[] s) {
    for (int i = 0; i < s.length; i++) {
        if (s[i] == null) {
            s[i] = new Soldado( maximoX(s) + xRandom(), 500);
        }
        if (s[i] != null && (s[i].x <= -22.5) || (s[i].x > 2000)) {
            s[i] = null;
        }
    }
}

```

Estos dos métodos son auxiliares de reaparece(). Generan una nueva coordenada x para los nuevos soldados tomando como referencia el que está más a la derecha y alejándolo un mínimo al menos una pantalla de ancho más un margen aleatorio.

```

private static double xRandom() {
    Random r = new Random();
    return 800 + r.nextInt(400);
}

```

```

private static double maximoX(Soldado[] slds) {
    double max = 0;

```

```

        for (int i = 0; i < slds.length; i++) {
            if (slds[i] != null && slds[i].getX() > max) {
                max = slds[i].getX();
            }
        }
        return max;
    }
}

```

Este método pregunta si la Fireball tocó algún soldado. Si es así, ese soldado se borra con null.

```

public boolean tocaSoldado(Soldado[] sold) {
    for (int i = 0; i < sold.length; i++) {
        if (sold[i] != null) {
            if (tocaSold(sold[i])) {
                explosion.start();
                sold[i] = null;
                return true;
            }
        }
    }
    return false;
}
}

```