

Integrantes del equipo:

Nombre: Reyno Sofía

Legajo: 44.173.232

Email: [sofireynoo30@gmail.com](mailto:sofireynoo30@gmail.com)

Nombre: Ramirez Ivana

Legajo: 42.399.241

Email: ivo.agus.ramirez22@gmail.com

Grupo: 11

**Índice**

1. BIENVENIDOS AL JUEGO

1.1 *Presentación*

*1.2 Modo de juego*

*1.3 Reglas del juego*

1. CLASES

2.1 *Clase Juego*  
 2.2 *Clase Personaje*  
 2.3 *Clase Fondo*

2.4 *Clase Piedras*

2.5 *Clase Poderes*

*2.6 Clase Demonio*

*2.7 Clase Recompensa*

1. PROBLEMAS Y SOLUCIONES
2. CONCLUSIÓN

***Introducción***

Este informe tiene como objetivo presentar el desarrollo del juego "El Camino de Gondolf", un proyecto interactivo que busca desafiar al jugador en un entorno dinámico. A lo largo del documento, se detallarán las mecánicas principales del juego, la estructura de sus clases fundamentales, los desafíos encontrados durante su creación y las soluciones implementadas.

1. **BIENVENIDOS AL JUEGO**
   1. **Presentación**

El juego "El camino de Gondolf" es una experiencia interactiva donde el jugador controla a Gondolf, quien perdió su camino intentando encontrar una pieza de joyería con forma circular y propiedades mágicas, pero tendrá que evitar los enemigos, ya que su enemigo, el mago Suramun, comenzará a enviar un sinfín de sus lacayos demonio para intentar derrotarlo.

* 1. **Modo de juego**

En "El Camino de Gondolf", tu objetivo principal es mover a Gondolf estratégicamente por la pantalla mientras utilizas sus poderes únicos para protegerte de los constantes ataques demoníacos. Eliminar a cada demonio te otorgará puntos (o en este caso, se te acerca a la victoria al reducir la cantidad de demonios restantes), acercándote a la victoria. Pero si Gondolf colisiona con los demonios, su vida disminuirá, y si lo hace demasiadas veces, perderá la partida.

* 1. **Reglas del Juego**

El jugador controlará a Gondolf utilizando las teclas WASD para moverse por la pantalla. Para activar y usar sus poderes, simplemente deberá hacer clic con el botón izquierdo del ratón. La victoria se alcanza al eliminar a todos los demonios, mientras que la derrota sobreviene si Gondolf colisiona con los enemigos y sus vidas llegan a cero..

1. **CLASES**

Aquí se presenta una explicación general de cada clase implementada, incluyendo las variables de instancia y una breve descripción de los métodos principales.

2.1 **Clase juego**

**Variables de instancia:**



En la clase ***Juego***, las variables de instancia incluyen el objeto ***entorno***, que controla el entorno gráfico, y el objeto ***mago***, una instancia de *Personaje* que representa al jugador. También están ***fondo***, una instancia de *Fondo* para añadir detalles al entorno, y ***piedras***, que representa los obstáculos. La variable *demonios* es un arreglo de objetos ***Demonio***, que representan a los enemigos que el mago debe enfrentar, y *random* es una instancia de ***Random*** utilizada para la aleatoriedad en el juego. El booleano ***juegoTerminado*** indica si el juego ha finalizado, mientras que ***juegoGanado*** lleva el registro de si el juego ha sido ganado. ***poderesActivos*** es una lista que ahora contendrá tanto *bolas de fuego* como pociones de *fuerza* para que el mago las use. ***demoniosEliminadosDesdeUltimaRecompensa*** lleva un registro de los enemigos eliminados desde la última recompensa, y ***recompensaActiva*** contiene la recompensa activa actual. Para la gestión de poderes, ***poderActivo*** almacena el poder activo actualmente (por ejemplo, "bola" o "fuerza"), con ***usosPoderRestantes*** indicando los usos restantes, hasta ***MAX\_USOS\_PODER\_BOLA*** (25 para bolas de fuego) y ***MAX\_USOS\_PODER\_FUERZA*** (25 para pociones de fuerza). Finalmente, ***demoniosMuertosContador*** y d***emoniosMatados*** se utilizan para contar y mostrar el número de demonios eliminados por el jugador.

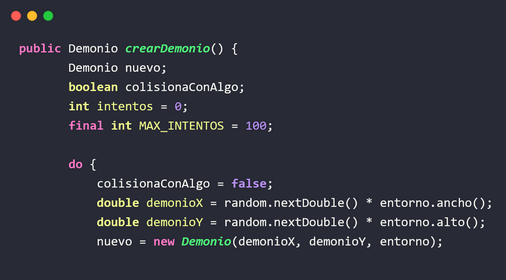
**Métodos principales:**

* **Tick():** Este método actualiza el estado del juego en cada instante. Controla el reinicio y el fin del juego, y actualiza la información en pantalla como vidas y puntuación. También gestiona la interacción del jugador para lanzar poderes, el movimiento del mago y las colisiones con el menú y obstáculos. Se encarga de la lógica de los demonios, incluyendo su persecución, colisiones con el mago y su reemplazo, y la interacción de los poderes con los demonios para sumar puntos. Además, maneja la aparición y recolección de recompensas, y determina las condiciones para ganar o perder la partida. Finalmente, dibuja todos los elementos en la pantalla.
* reiniciarJuego():

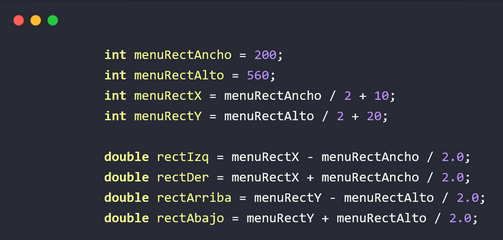


Esta función se encarga de reestablecer el estado completo del juego a su configuración inicial, preparando todo para una nueva partida. Específicamente, inicializa al mago (el personaje del jugador) en el centro de la pantalla y le otorga 100 puntos de vida, además de reiniciar su puntuación a cero. También crea un nuevo fondo y las piedras que actúan como obstáculos en el entorno. Se encarga de inicializar el generador de números aleatorios y de crear un arreglo para cuatro demonios activos simultáneamente, generando cada uno en una posiciín válida. Asimismo, limpia la lista de poderesActivos, establece las banderas juegoTerminado y juegoGanado a false, y reinicia los contadores relacionados con la recompensa y el uso de poderes a cero, asegurando que haya ninguna recompensa o poder activo al inicio.

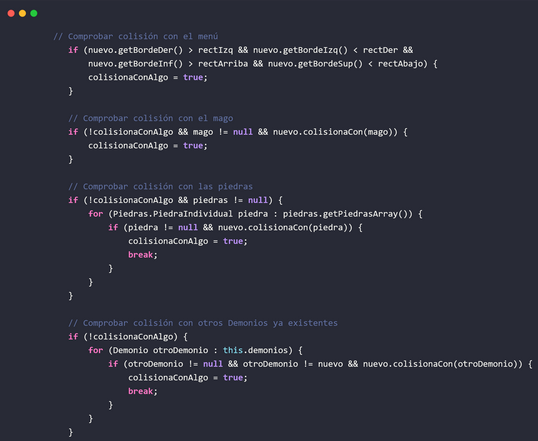
* **crearDemonio():**



Este método inicializa las variables necesarias para generar un *nuevo* demonio, como nuevo, *colisionaConAlgo* e intentos. Un bucle *do- while* intenta encontrar una posición aleatoria *X* e *Y* dentro del entorno para el nuevo demonio. En cada intento, *colisionaConAlgo* se reinicia a *false* antes de evaluar la adaptacion de la posición generada.



Esta seccion del codigo se encarga de definir las dimensiones y la posición exacta del área del menú en la pantalla del juego.



Esta seccion comprueba que no colasione con el menú, el mago, con las piedras y con demonios que ya existen.



**2.2 Clase Personaje**

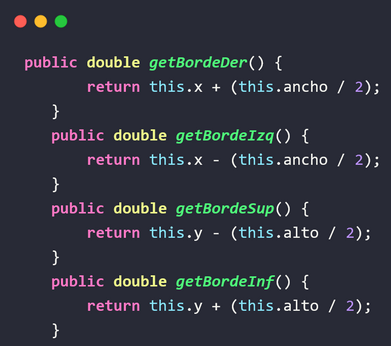
**Variables de instancia:**



En la clase ***Personaje***, las variables de instancia principales definen las características y el estado del protagonista. Las coordenadas ***x*** e ***y*** controlan la posición del personaje en la pantalla, mientras que ***ancho*** y ***alto*** determinan sus dimensiones. Para su representación visual, un conjunto de variables ***Image*** como *imagenDer, imagenIzq, imagenAtras, imagenFrente* almacenan las distintas imágenes del mago para reflejar su orientación, y *inicioImage* guarda la imagen que se está mostrando actualmente. La ***escala*** ajusta el tamaño visual del personaje, y la ***velocidad*** define qué tan rápido se mueve. La variable e es una referencia al objeto ***Entorno***, necesaria para interactuar con la pantalla. Adicionalmente, ***estaApoyado*** indica si el personaje está en una superficie, y *herido* señala si ha recibido daño. La ***vida*** registra la salud del personaje, y ***poderes*** es un arreglo que lista los nombres de las habilidades que el mago puede utilizar. ***ultimaDireccionDerecha*** guarda la última dirección horizontal del movimiento del mago, y ***puntuacion*** lleva el registro de los puntos acumulados por el jugador.

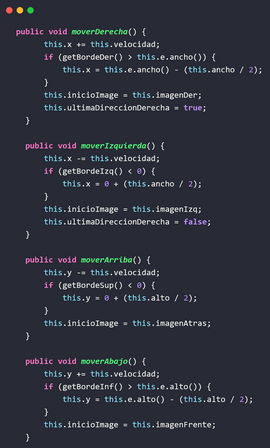
**Métodos principales:**

* ***Personaje ():*** Este metodo inicializa las propiedades esenciales del mago al ser creado. Establece su posición inicial con X e Y, define su escala a 0.16 y su velocidad a 2.0. Carga las imágenes correspondientes a las diferentes orientaciones del mago y establece la imagen frontal por defecto. Tambien asocia el personaj con el entorno del juego. Calcula el ancho y alto del personaje basandose en las dimensiones de la imagen y la escala. Finalmente, lo inicializa sin estar herido, le asigna puntos de vida y le otorga un conjunto inicial de poderes.
* ***getBorde():***

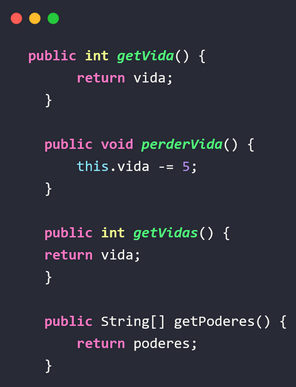


Métodos para obtener los bordes del personaje (útiles para colisiones y límites)

* ***Mover():***



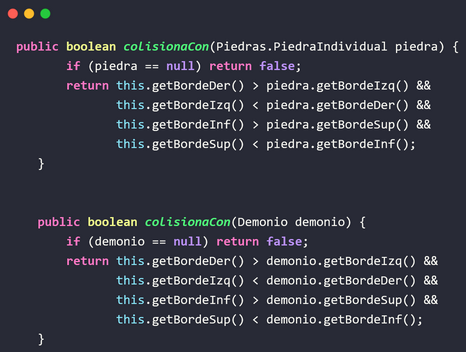
Estas cuatro funciones, *moverDerecha()*, *moverIzquierda()*, *moverArriba()* y *moverAbajo()*, controlan el desplazamiento del personaje en la pantalla.



Estas funciones proporcionan acceso y manipulación a la salud y habilidades del personaje. Los métodos *getVida()* y *getVidas()*permiten obtener el valor actual de la vida del personaje. Por otro lado, *perderVida()*  reduce la vida del personaje en 5 puntos. Finalmente, *getPoderes()* devuelve el arreglo de cadenas que contiene los nombres de las habilidades que el personaje posee.



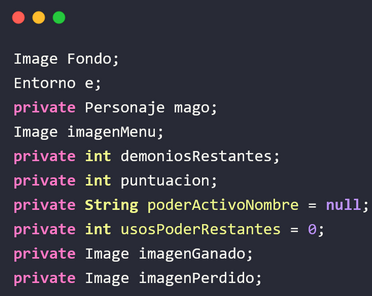
Estos métodos controlan la puntuación, la dirección y la vida del personaje. *getUltimaDireccion()* devuelve un booleano que indica si la última dirección de movimiento del personaje fue hacia la derecha. *getPuntacion()* retorna la puntuación actual del personaje, mientras que *sumarPuntos()* permite añadir puntos a esa puntuación. Finalmente, *setVidas()* permite establecer la cantidad de vida del personaje a un valor especifico.



En los métodos *colisionaCon()* se utiliza para detectar si el objeto (como el mago o un demonio) esta superponiéndose con otro objeto en el juego (como con otro demonio o una piedra).

**2.2 Clase Fondo:**

**Variables de instancia:**



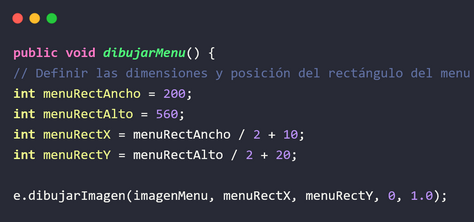
En la clase ***Fondo*** gestiona los elementos visuales y el estado del juego mostrados en pantalla. *Fondo* es una *Image*que representa el telón de fondo del juego. ***Entorno e*** es una referencia al entorno del juego, permitiendo interacciones con la pantalla. ***Personaje mago*** es una referencia al personaje principal, usada para obtener datos relevantes del jugador como su vida o puntuación. ***imageMenu*** es la *Image* que representa el menú del juego. ***demoniosRestantes*** y ***puntuacion*** son variables numéricas que guardan la cantidad de demonios que quedan y el puntaje actual del jugador respectivamente. ***poderActivoNombre*** almacena el nombre del poder que el mago tiene activo en ese momento, y ***imagenGanado*** e ***imagenPerdido*** son *Image* que se muestran al final para indicar si el jugador ganó o perdió, respectivamente.

**Métodos principales:**

* ***Fondo():*** Este método inicializa los elementos visuales principales del juego y su relación con el entorno y el personaje. Carga la imagen de fondo principal, establece una referencia al *Entorno* del juego, y asocia el *Personaje mago* para obtener sus datos. También carga las imágenes específicas para el menú, la pantalla de victoria y la pantalla de derrota.
* ***dibujar():***



* ***dibujarMenu():***

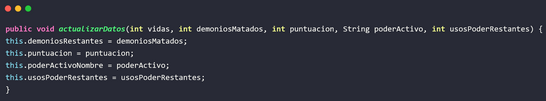


* ***set():***



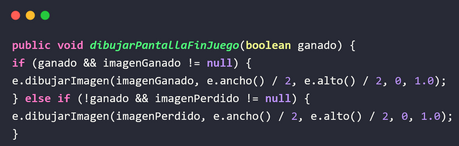
Estos métodos permiten actualizar el estado de las variables relacionadas con la información mostrada en el fondo del juego. *setDemonioRestantes()*  actualiza el conteo de demonios pendientes, *setPuntuacion()* establece el puntuaje actual del juagador, y *setPoderActivo()* configura el nombre del poder activo del mago y la cantidad de usos restantes para dicho poder.

* ***actualizarDatos():***

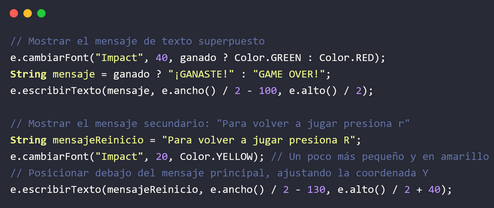


Este método se encarga de refrescar la infomación que se muestra en el fondo del juego. Recibe como parámetros las vidas del mago, la cantidad de demonios eliminados por el jugador, la puntuación actual, el nombre del poder activo y los usos restantes de dicho poder. Internamente, asigna estos valores a las variables de instancia correspondientes para que la interfaz grafica refleje el estado más reciente del juego.

* ***dibujarPantallaFinJuego():***

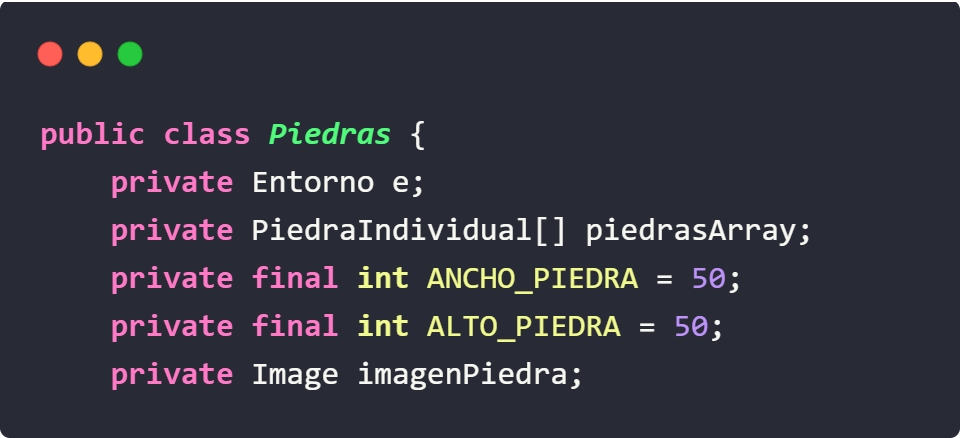


Este método se encarga de mostrar la imagen correspondiente al resultado final de la partida. Si el parámetro *ganado* es *true* y la *imagenGanado* no es nula, se dibuja la imagen de victoria centrada en la pantalla. En caso contrario, si *ganado* es *false* y la *imagenPerdido* no es nula, se dibuja la imagen de derrota tambien centrada.



2.4 Clase Piedras

variables de instancias:



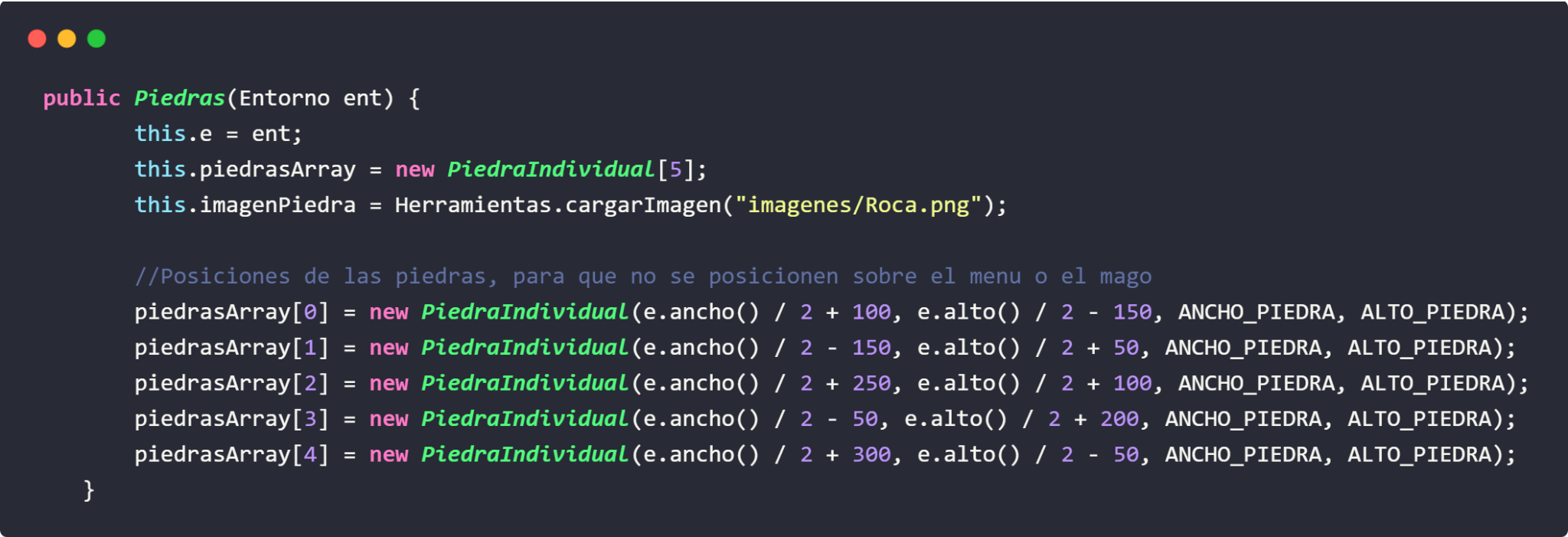
En la clase *Piedras* las variables de instancias principales definen las medidas que deben tener las piedras en el juego, la variable *Entorno* guarda los objetos que deben mostrarse en pantalla, *PiedraIndividual* es un arreglo que guarda las 5 piedras que se van a crear y mostrar en el juego. *ANCHO PIEDRA* define el ancho que va a tener cada piedra y *ALTO\_PIEDRA* define la altura, por último *imagenPiedra* es la imagen que se va a usar para mostrar cada piedra que se carga desde un archivo.

Métodos Principales:



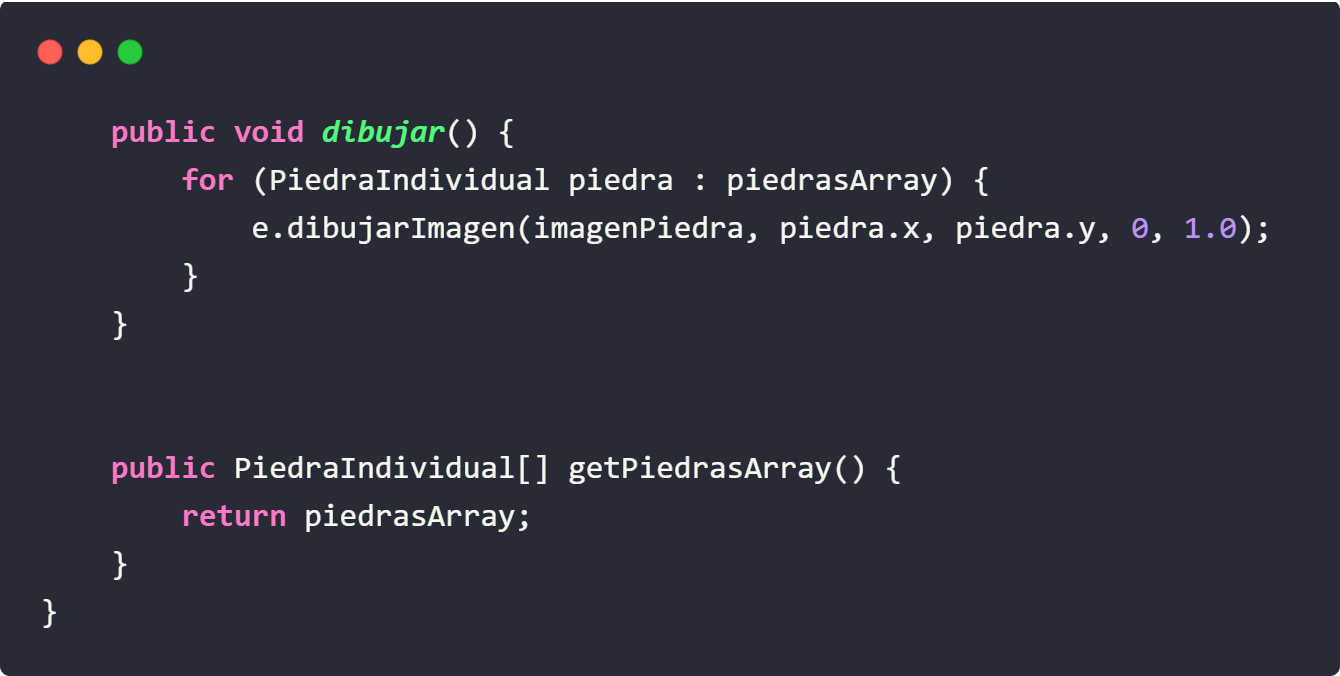
#### La clase interna PiedraIndividual representa una sola piedra y define su posición y tamaño. Además, tiene métodos para saber dónde están sus bordes, lo cual sirve para detectar colisiones. El constructor PiedraIndividual(double x, double y, double ancho, double alto) crea una piedra con su posición y tamaño y guarda los valores recibidos. getBordeDer devuelve la posición del borde derecho de la piedra, getBordeIzq la posición del borde izquierdo, getBordeSup la posición del borde superior y getBordeInf devuelve la posición del borde inferior.

* Constructor Piedras(Entorno ent):



Este método se ejecuta cuando se crea un objeto de tipo Piedras. Lo que hace es guardar el *entorno (ent)* en la variable e inicializar el arreglo de piedras con 5 posiciones luego se carga la imagen de la piedra desde el archivo y creamos 5 piedras en lugares estratégicos del mapa, para que no molesten al menú ni al mago desde el arranque

* Método dibujar():

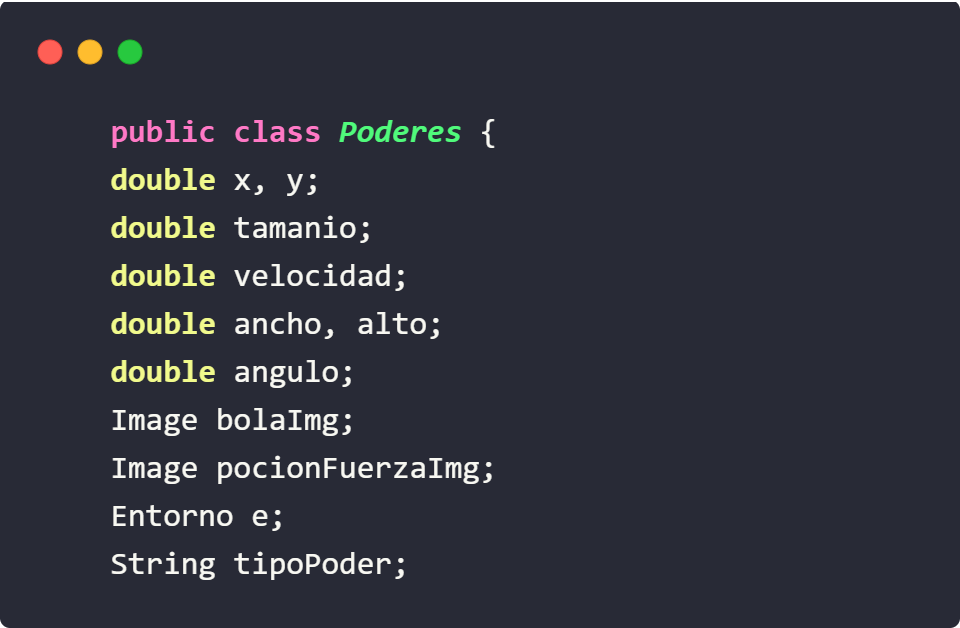


Este método dibuja las piedras en pantalla y recorre el arreglo *piedrasArray* y, una por una, usa *e.dibujarImagen()* para mostrarlas usando la imagen cargada.

2.5 Clase Poderes:

Variables de instancia:

*double x, y* son las coordenadas donde aparece el poder como una bola de fuego o una poción,, *double tamaño* escala el tamaño de la imagen del poder, *double velocidad* es la velocidad con la que el poder se mueve en pantalla, *double ancho, alto*, representan el tamaño del poder ya ajustado con la escala osea el tamanio, *double angulo*, es el ángulo de dirección en el que se va a mover el poder, calculado según hacia dónde apunta el mouse, *Image bolaImg*, guarda la imagen para la bola de fuego, *Image pocionFuerzaImg*, guarda la imagen para el poder tipo poción de fuerza, *Entorno e*, se refiere al entorno del juego, para saber el tamaño de la pantalla, etc y por ultimo *String tipoPoder* define qué tipo de poder es: "bola" o "fuerza".



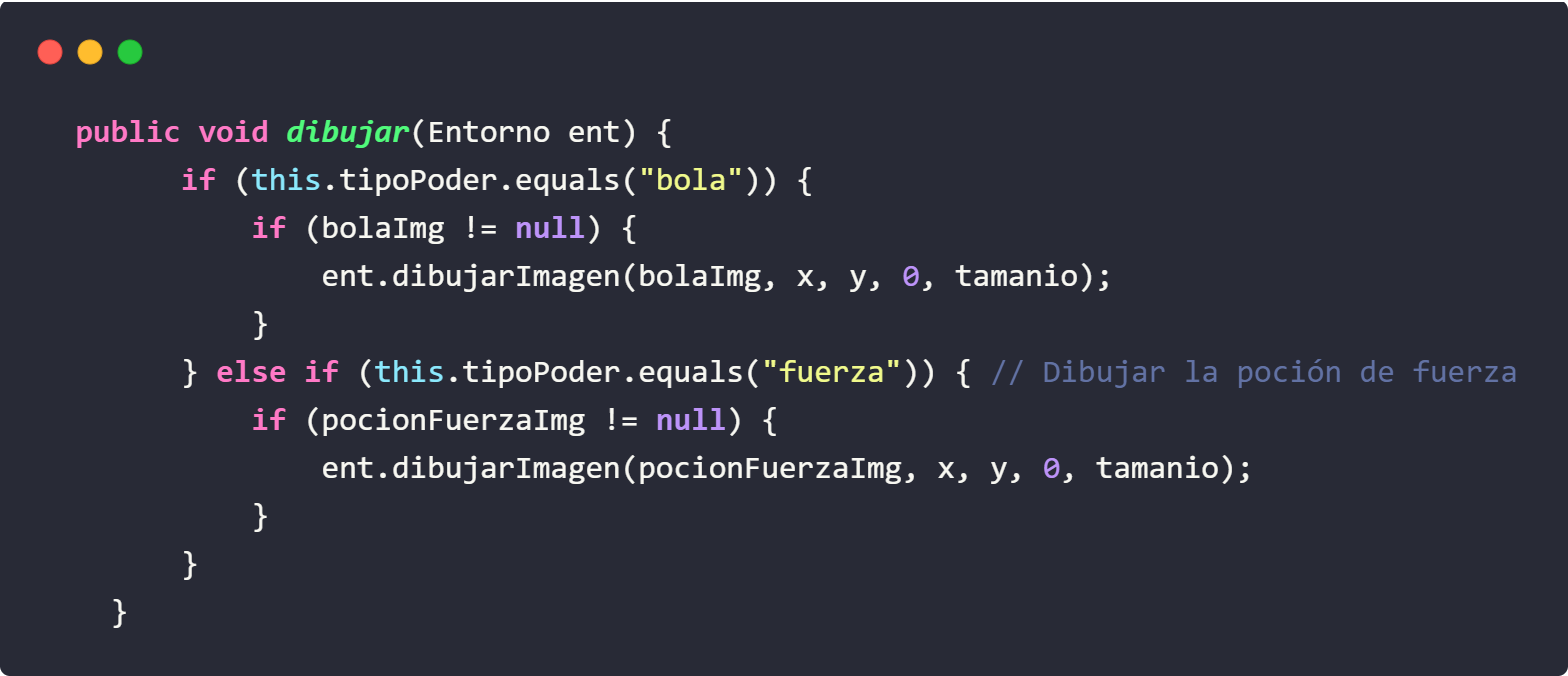
Métodos principales:

* Constructor Poderes():

El constructor de la clase *Poderes* tiene como objetivo inicializar un nuevo objeto de tipo poder, ya sea una bola de fuego o una poción de fuerza, partiendo desde una posición específica y dirigiéndose hacia el punto donde el jugador haga clic con el mouse. *x, y* son coordenadas iniciales del poder que coinciden con la posición del mago, *ent* es un objeto de tipo *Entorno*, utilizado para acceder a dimensiones de pantalla y otras funcionalidades gráficas, *mouseXDestino*, *mouseYDestino* tambien son coordenadas del destino del poder, *tipoPoder* es una cadena que indica el tipo de poder se quiere lanzar, puede ser "bola" (bola de fuego) o "fuerza" (poción de fuerza). Si se elige alguna de las dos opciones, se ajusta el tamizado de la imagen según su ancho y alto y se calcula el ángulo desde el punto inicial hasta donde se presiono con el mouse, para que el poder vaya en esa dirección.



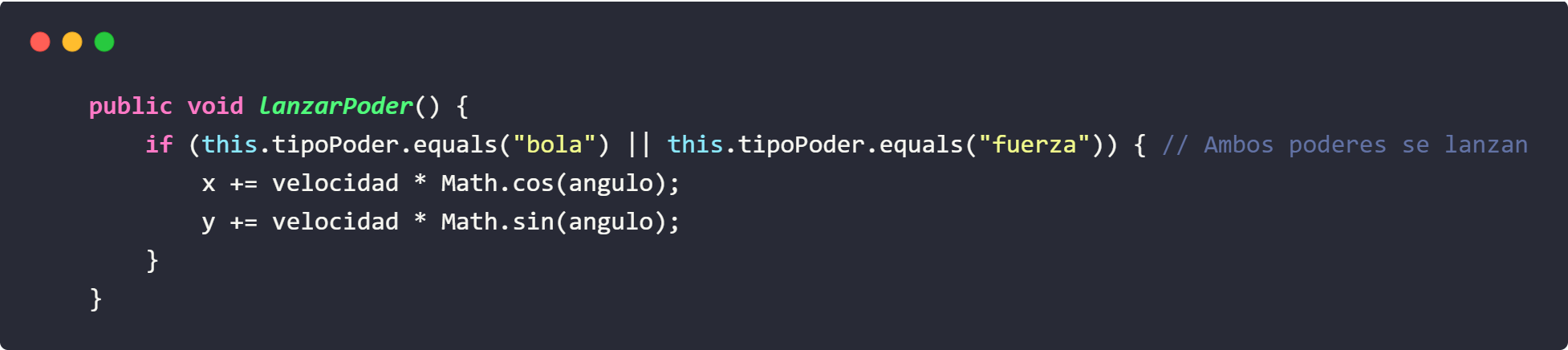
* dibujar():



Este método dibuja el poder en pantalla, dependiendo de si es una bola de fuego o una poción de fuerza a su vez usa la imagen que corresponde al tipo de poder.

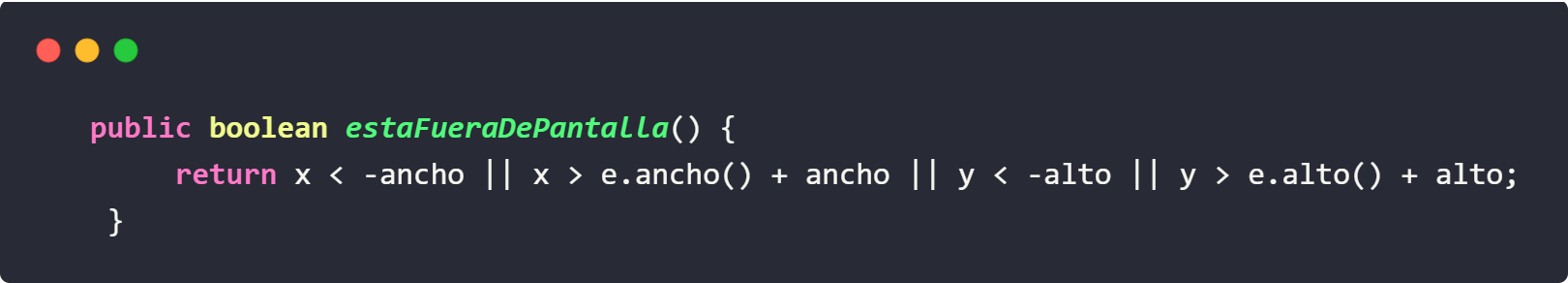
* lanzarPoder():

Este método mueve el poder hacia adelante. Calcula cuánto se mueve en X e Y usando la velocidad y el ángulo. Funciona para ambos tipos de poderes.



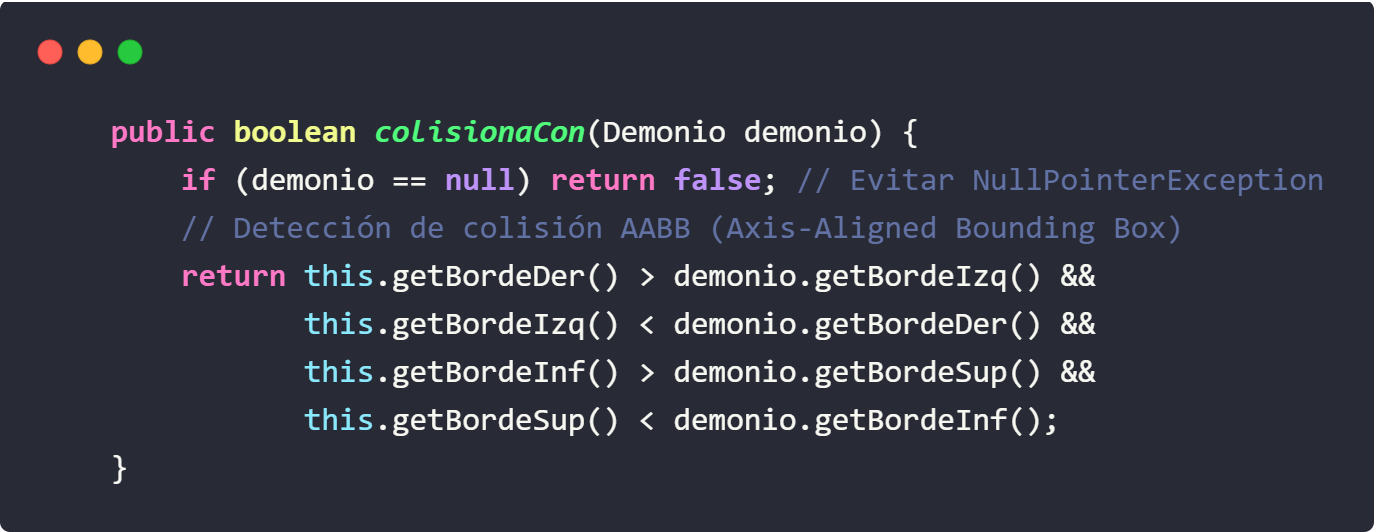
* estaFueraDePantalla():

Revisa si el poder ya salió de los límites de la pantalla. Si está muy lejos del borde, devuelve true para indicar que ya no se debe mostrar.



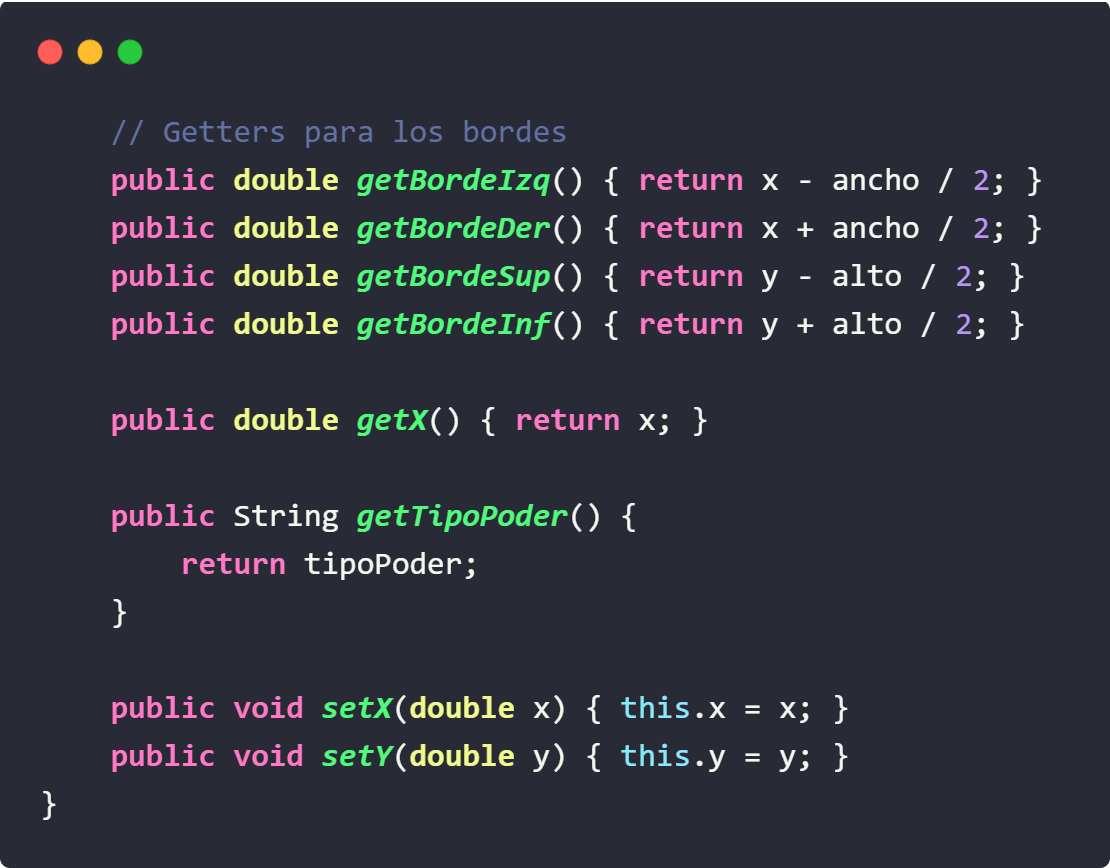
* colisionaCon(Demonio):

Este método detecta si el poder chocó con un demonio. Lo hace comparando los bordes del poder con los del demonio, usando una técnica simple de colisión por cajas. Si se solapan, devuelve true.



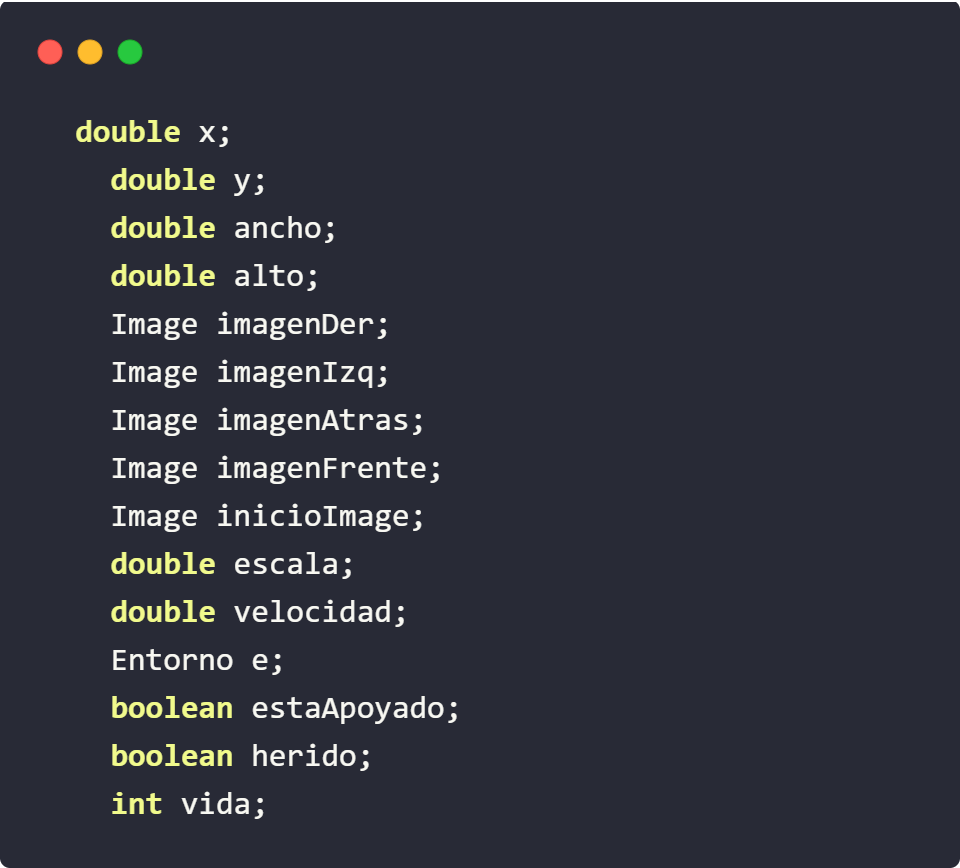
* Getters y Setters:

Estos métodos sirven para obtener o cambiar valores del poder, *getBordeIzq(), getBordeDer(), getBordeSup(), getBordeInf()* estos devuelven los bordes del poder, necesarios para detectar colisiones, *getX()* devuelve la posición X del poder, *getTipoPoder()* devuelve si es una bola o una poción,y por último *setX(double x), setY(double y)* permiten cambiar la posición del poder manualmente (por si se necesita moverlo desde afuera).



2.6 Clase Demonio:

variables de instancia:



Las variables *x, y* es la posición del demonio en la pantalla, el *ancho* y *alto* son el tamaño del demonio (calculado en base a la imagen). La *imagenDer, imagenIzq, imagenAtras, imagenFrente* es la imágenes del demonio según su dirección, el *inicioImage* es la imagen que se va a mostrar en el momento actual. *escala* es el factor que achica o agranda la imagen y la *velocidad* muestra qué tan rápido se mueve. *e,*  hace referencia al entorno gráfico (*Entorno*) donde aparece el demonio, *estaApoyado* hace referencia a si está tocando el suelo o algo más. La variable *herido* nos muestra si el demonio está muerto y por último la variable *vida* son los puntos de vida (empieza con 100).

Métodos Principales:

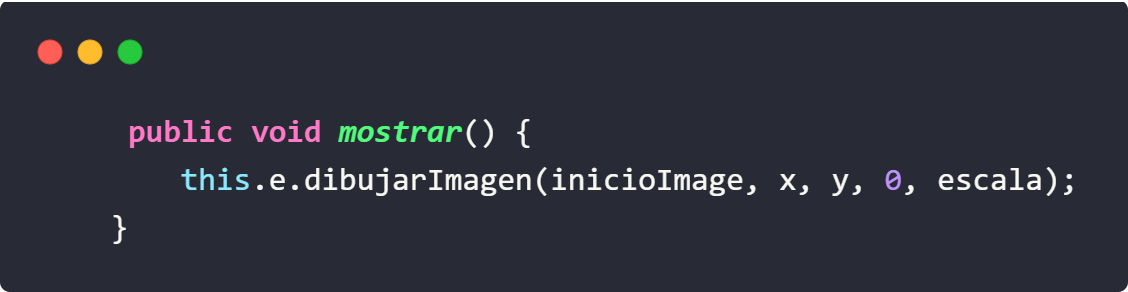
* Demonio



Crea un nuevo demonio en una posición específica luego carga las imágenes y define su tamaño, velocidad, estado inicial y la imagen que va a mostrar primero.

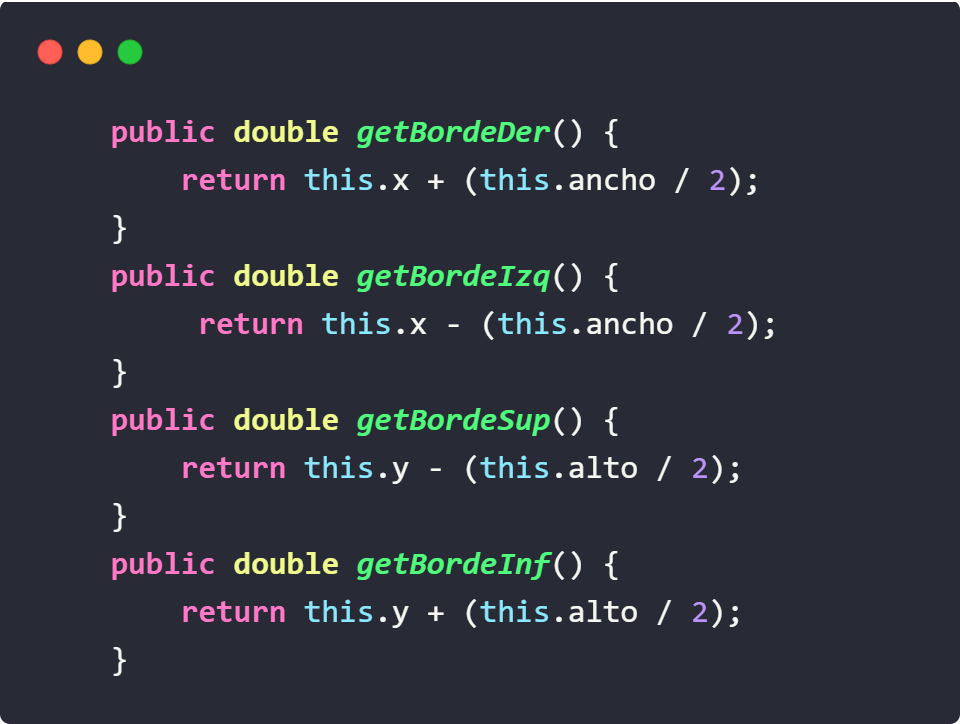
* Mostrar():

Muestra al demonio en la pantalla usando la imagen actual.

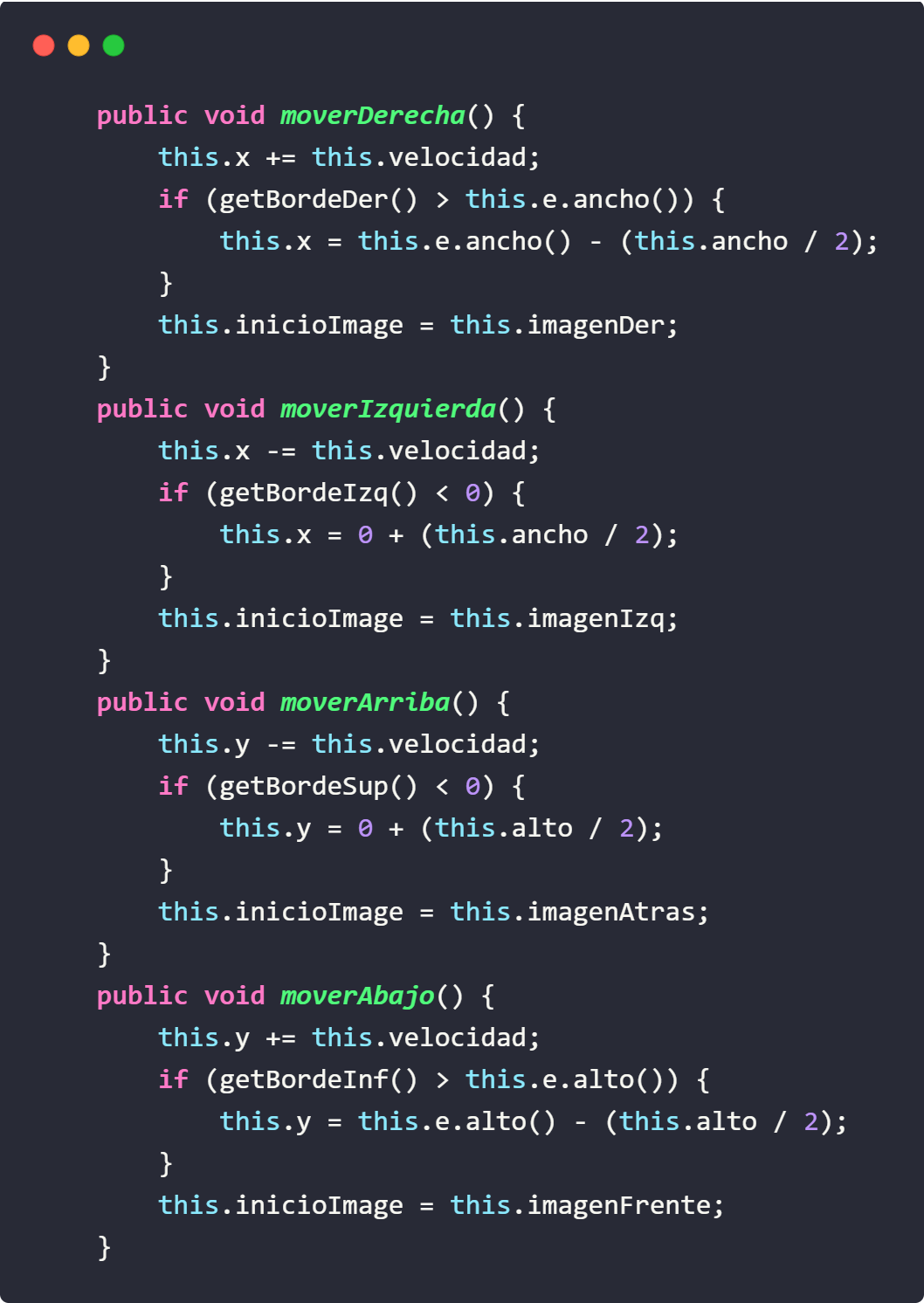


* *Metodos para detectar bordes (colisiones):*

Estos métodos devuelven los bordes del demonio, también nos dicen si tocó a otro objeto.



* *Movimiento:*



Cada uno de estos métodos mueve al demonio en una dirección y actualiza la imagen, También evitan que el demonio se salga de la pantalla (lo frenan en los bordes).

* *Perseguir(Personaje mago):*



La función de este método es hacer que el demonio siga automáticamente al personaje (el mago). A su vez calcula la dirección y se mueve en esa misma direccion. Además, cambia la imagen dependiendo de si va para arriba, abajo, izquierda o derecha.

* *colisionaCon():*

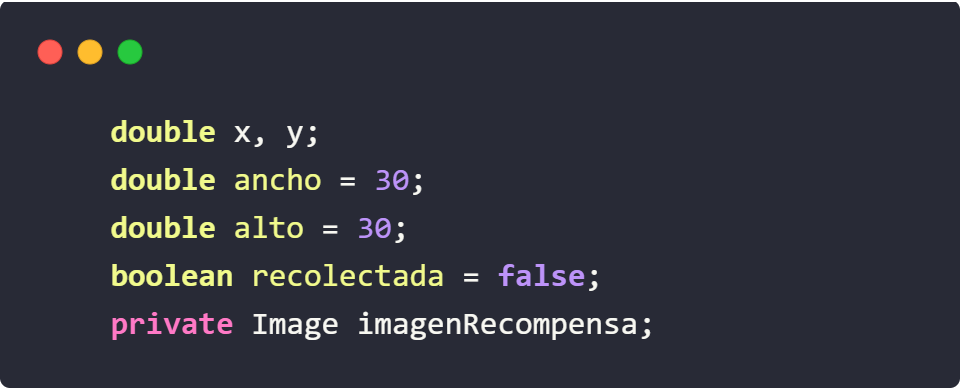


Estos métodos detectan si el demonio está chocando con algo, ya sea el menú, las piedras o los bordes, También usan los bordes para saber si se superponen con otro objeto.

*2.7 Clase Recompensa:*

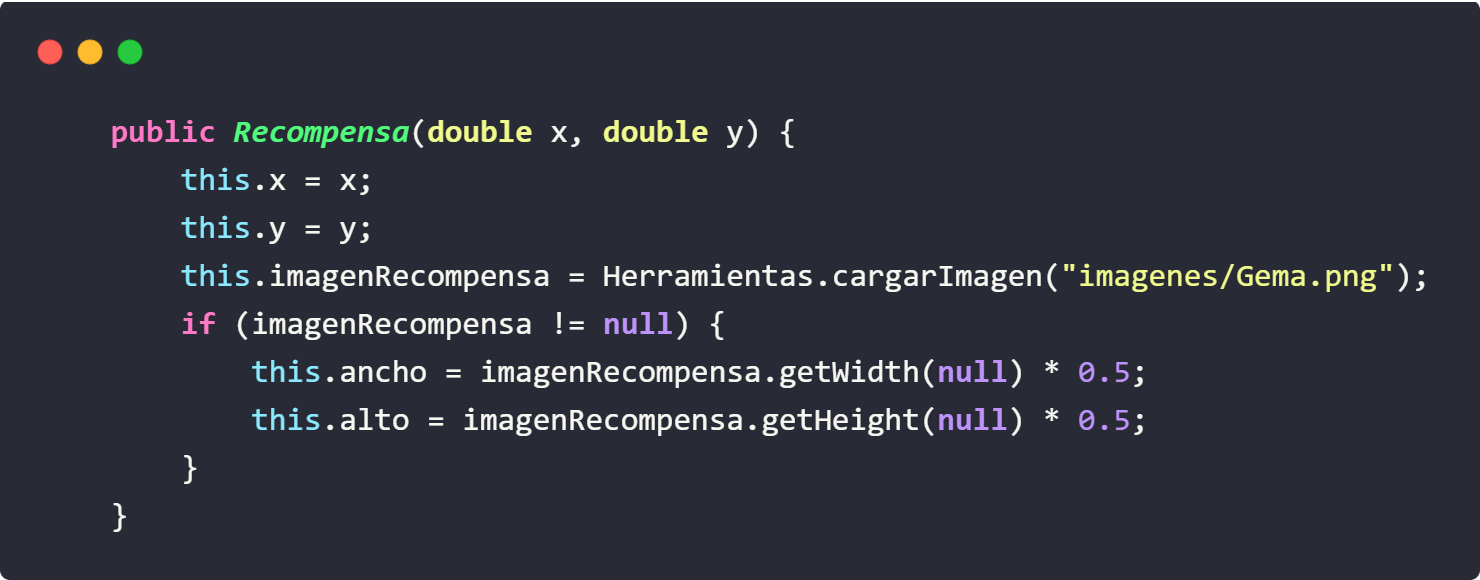
*Variables de instancia:*

Las variables *x, y* son posición donde aparece la recompensa en la pantalla y las variables *ancho, alto* es el tamaño de la recompensa, inicialmente fijo en 30, pero puede ajustarse al tamaño de la imagen.  
La variable *recolectada* indica si la recompensa ya fue obtenida por el jugador (evita que se dibuje nuevamente) y por ultimo la variable *imagenRecompensa*, guarda la imagen que se muestra cuando aparece la recompensa (por ejemplo, una gema).



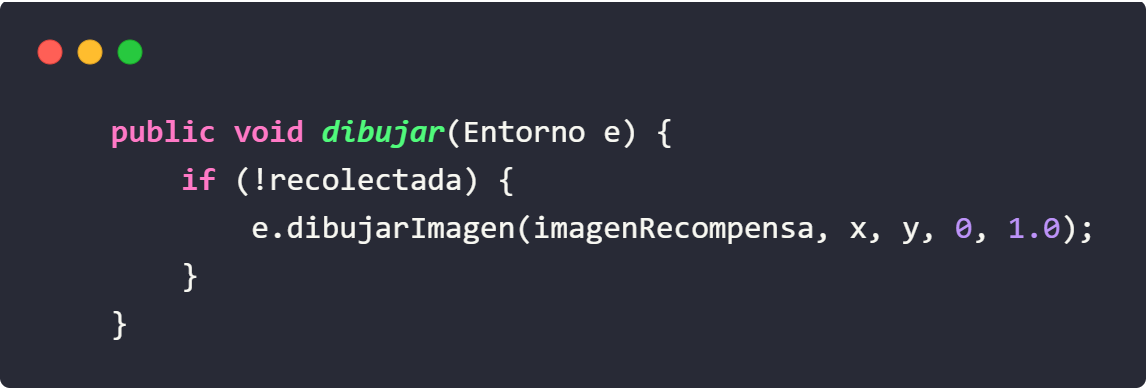
Métodos Principales:

* *Recompensa( );*



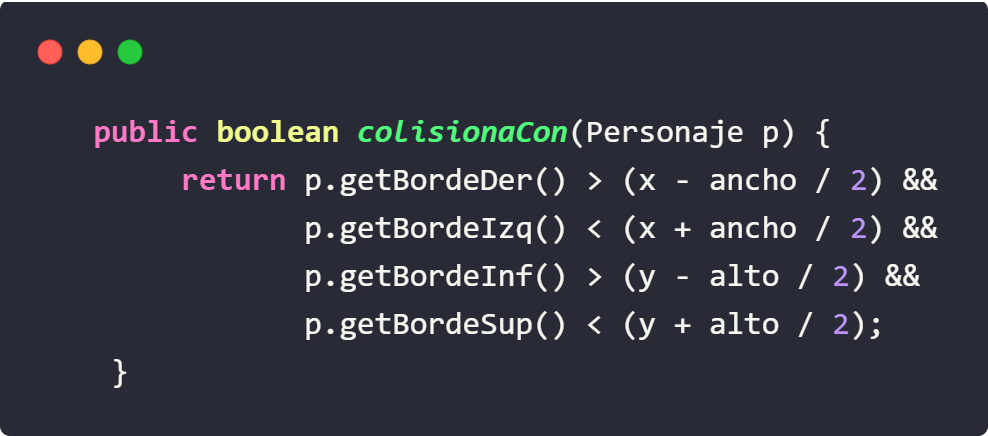
Este método se ejecuta cuando se crea una nueva recompensa. Guarda la posición *x* y *y* donde aparecerá, luego carga una imagen desde el archivo y si la imagen se cargó correctamente, ajusta el tamaño de la recompensa (*ancho* y *alto*) en función de la imagen, escalando al 50%. Así se asegura que la imagen tenga un tamaño visual coherente.

* *Dibujar():*



Dibuja la imagen de la recompensa en pantalla si aún no fue recolectada y usa el entorno *e* para mostrarla en la posición (*x, y*) con una escala de 1.0 (tamaño real de la imagen).

* *ColisionaCon():*



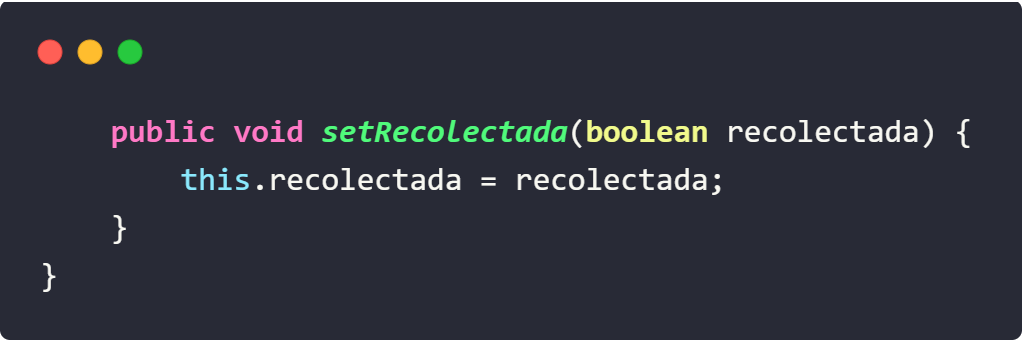
Detecta si el personaje (*p*) tocó la recompensa y si hay intersección entre los bordes del personaje y de la recompensa, devuelve *true*.

* *isRecolectasa():*

Este método devuelve *true* si la recompensa ya fue recogida, sirve para evitar dibujarla o sumar puntos más de una vez.



* *setRecolectada():*



Cambia el estado de la recompensa. Por ejemplo, cuando el personaje la agarra, se llama con *setRecolectada(true)*.

***3.0 Problemas y Soluciones:***

1. Problema con la creacion de los botones de los poderes en el menú.

Solución: Para resolver este problema calculamos las coordenadas donde queriamos que aparecieran los nombres de los poderes, y luego simplemente usamos la funcion de entorno para dibujar texto y dibujar rectángulos invisibles donde luego el mouse haga un click y ahi elegiria el poder

1. Hacer funcionar el mouse haciendo que seleccione el poder y los lance.

Solución: Para resolver este problema, implementamos la interaccion del mouse en el metodo tick de la clase juego, al detectar un click, verifica si las coordenadas corresponden a las del area de los botones y activa los poderes, y luego de eso si se clickea fueras de las coordenadas del menú se lanza el poder.

1. Lograr que todos los demonios generados sigan al mago.

Solución: Para resolver este problema creamos un método llamado perseguir en la clase demonio. En cada tick, el demonio calcula la diferencia en el x e y con el mago para determinar su direccion. Luego se mueve en paso constante hacia el mago, mientras va ajustando su propia imagen tambien.

***4.0 CONCLUSIÓN:***

Este proyecto permitió aplicar conceptos clave de la programación orientada a lo visto en clases. Aprendimos la importancia de planificar la estructura del código y la gestión de colisiones, lo cual fue un desafío en ciertos puntos del desarrollo. En general, el resultado es satisfactorio, y el juego refleja un buen nivel de complejidad y funcionalidad, aplicando los conceptos aprendidos.