



Universidad Autónoma de Baja California  
Facultad de Ciencias Químicas e Ingeniería

Introducción a videojuegos

**GDD: Graphic Design Document**  
**“LOST ARTIFACTS”**



Grupo: 351

Nombres:

- Maribel Itzel Méndez Ascendió 1279725
- Milka Yamil Trinidad Gutiérrez 1283472

Maestro: Omar Zamarron

Fecha: 31 de mayo del 2023



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## ÍNDICE

1. Historia.....	2-3 p
2. Sprite Colors.....	3 p
3. Personajes	
a. Jugador:Diagrama de Transiciones.....	4-5 p
b. Guardianes y artefactos.....	5-8 p
c. Enemigo.....	8 p
d. Inventario: Gemas y Plantas.....	9 p
e. Dragon.....	10 p
4. 1er Nivel.....	11-12 p
5. TIPO DE CÁMARA: Position-Locking.....	13-14 p
6. Diseño de clases.....	15 p
7. Mecánicas de clases	
a. Clase SDLApp.....	16-17 p
b. Clase MotorFisico2d.....	17-18 p
c. Clase Cámara.....	18-19 p
d. Clase Atlas.....	19-20 p
e. Clase FSMJugador .....	36-42 p
f. Clase FSMEnemigo .....	42-47 p
g. Clase FSMGuardian .....	47-53 p
8. Capturas de Juego Final.....	53-55 p



## 1. Historia

Eres de los mejores mercenarios de Fiory, ciudad libre que queda justo en medio de los reinos Elvin, Seemar, HailFrost y el bosque oscuro del continente. Creciste y vives en esta ciudad cazando monstruos del bosque oscuro que logran infiltrarse en los reinos por dinero. Un día los reyes y reinas de cada reino te ofrecieron un trabajo que no puedes rechazar.

Información dada por los reyes: Un dragón, especie que se creía extinta desde hace años, ha robado los artefactos cuyos poderes protegen a cada reino de los monstruos del bosque oscuro, junto a sus guardianes (espíritu de animales poderosos que siempre están juntos a los artefactos). Hasta ahora, no se han podido recuperar los artefactos, pero saben que el dragón los ha colocado en el bosque oscuro, por lo que habrá monstruos al que te enfrentaras, así como un guardián que protege a cada artefacto.

- **Tu misión:** recuperar los 3 artefactos de cada reino y enfrentarte al dragón. Si tienes el artefacto, el guardián aparecerá, y viceversa.
- **Recompensa:** Vida de perezoso. Te darán una mina de minerales del reino en Elvin, Una mansión cerca de los mares en Seemar, y el arma de tu elección hecha por el mejor artesano de HailFrost.
- **Fracaso:** La muerte.

Los artefactos representan, protegen y dan vida a las características que definen a cada reino desde hace cientos de años.

**El reino de Elvin:** Reino al oeste del continente con árboles grandes y verdes, donde los ciudadanos viven uno con la naturaleza y las ciudades se encuentran escondidas entre los árboles. Así mismo, tiene una afinidad con las plantas por las que tienen de los mejores campos agrícolas del continente.

Su símbolo y artefacto que los protege es el **escudo de madera**, ya que como la madera que protege a todo el reino, este escudo los mantiene a salvo y oculto de los monstruos. En tiempos de guerra, le da al usuario el poder de protegerse contra cualquier ataque.

**Consecuencias:** Luego de que el escudo fue robado, los árboles que rodeaban el reino se debilitaron y comenzaron a marchitarse, dejando que más monstruos lleguen al reino. Así mismo, sus campos de siembra pararon de dar fruto.

**El reino de Seemar:** Reino este del continente, rodeado de lagos y ríos con corrientes fuertes y animales marinos de todo tipo. Principal recurso económico es la pesca.

Su símbolo y artefacto es el **Water Whip/Latigo de agua**, rápido y peligroso como los lagos y ríos que protegen a dicho reino. Quien lo use, obtiene la habilidad de que sus movimientos sean más rápidos.

**Consecuencias:** Los ríos comenzaron a secarse y se están quedando sin sus bordes naturales y su principal recurso económico.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

**El reino de HailFrost:** Reino al norte del continente que siempre se encuentra nevado y lleno de icebergs. Pero a pesar de estas condiciones, es considerado el reino más seguro de los tres reinos con los mejores soldados y mejor elaboración de armas y avances tecnológicos.

Su símbolo y artefacto es la ***espada de hielo***, quién protege a su reino cubriendolo de paredes de hielos inquebrantables y filosos. Quien usa esta espada obtiene la habilidad de cortar cualquier cosa.

**Consecuencias:** El hielo se ha comenzado a derretir y los lugares se han empezado a inundar.

**Bosque oscuro:** Al sur del continente, la región lleno de monstruos de la que se conoce poco. Casi ningún arma sirve contra los monstruos de este bosque, hasta ahora, la única forma de debilitar a un monstruo es con la hierba “Shigmul”.

**Consecuencia:** Los monstruos comienzan a infiltrar el resto de los reinos debido a que todos los poderes de los artefactos se encuentran en su hogar.





Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## 2. “Sprite Colores”



Figure: 6 bits (64 colores)

Escogimos el “pixel art” debido a que es un arte conceptual muy llamativo de ver y al ser un arte visualmente sencillo creemos que esto le puede dar un toque más fantasioso y clásico a nuestro juego cuyo mundo e historia es de fantasía. Así mismo tomamos en cuenta que la paleta de colores que utilizaremos es la paleta de colores de 6 bits debido a que tiene un mayor rango de colores disponibles lo cual proporciona más vida a los personajes y artefactos quedando mejor con la temática del juego.

## 3. Personajes

### “Jugador”

Característica y Puntuación

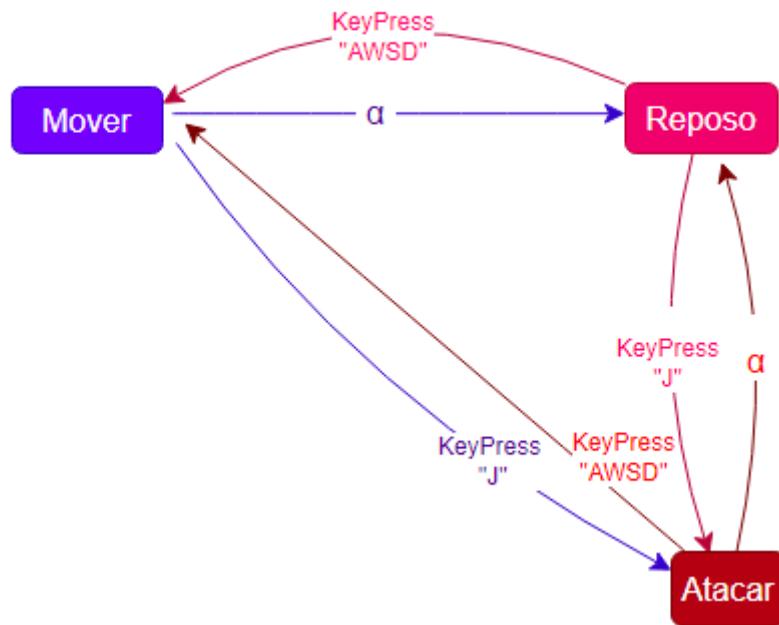


Paleta de colores: Gris, Café y Azul

- **Modo Fácil:** Inicia el primer nivel con 100 puntos. Se bajan 1 punto cada que entra en contacto con un monstruo. Baja 1 punto si lo ataca un guardián. Baja 5 puntos el dragón.
- **Modo medio:** Inicia en 80, se bajan 2 puntos cuando entre en contacto con un monstruo. Baja 2 puntos si lo ataca un guardián. Baja 6 puntos el dragón.
- **Modo difícil:** Inicia en 80, baja 3 puntos cuando entre en contacto con un monstruo. Baja 3 puntos si lo ataca un guardián. Baja 7 puntos el dragón.
- Gana 25 puntos cada que encuentre un artefacto.
- Si llega a cero en un nivel, pierde el juego y debe iniciar de nuevo.
- Los puntos se van acumulando cada nivel.



Diagrama de transición



**Tabla de transición**

<b>M</b>	$\{Q, \Sigma, \alpha, a, F\}$
<b><math>\Sigma</math></b>	$\{\alpha, J, W, A, D, S\}$
<b><math>\alpha</math></b>	Tabla abajo
<b>F</b>	$\{\text{Reposo}\}$

	$\alpha$	AWSD	J
Reposo	Reposo	Mover	Atacar
Mover	Reposo	Mover	Atacar
Atacar	Reposo	Mover	Atacar



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## “Búho”

Característica y Puntuación



Paleta de colores: Blanco, beige, rosa, negro.

- Se mantiene en estado dormido cuando inicia el nivel.
- Comienza a atacar al jugador cuando el jugador entra en contacto con él por primera vez.
- Se ataca al jugador en tiempos aleatorios entre 5-15 segundos.
- El artefacto que protege, “Escudo de madera”, aparece cuando el búho está despierto.
- Su ataque le resta 1, 2 y 3 puntos al jugador en el nivel correspondiente.
- El jugador no le puede hacer daño.

## “Escudo de madera”

Característica y Puntuación



Paleta de colores: Diferentes tonos de gris, café, rojo y blanco.

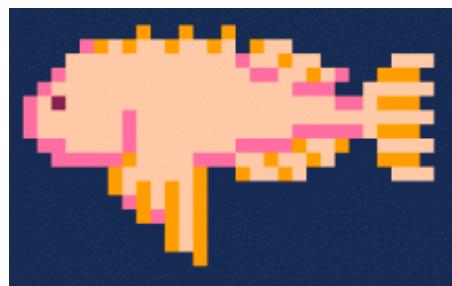
- Símbolo y artefacto mágico de madera que protege al reino Elvin, manteniendo sus fronteras a salvo y oculto de los monstruos.
- Le da al usuario el poder de protegerse contra cualquier ataque, incrementando la defensa del jugador de un 1.5 a 2.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## “Pez”

Características y Puntuación

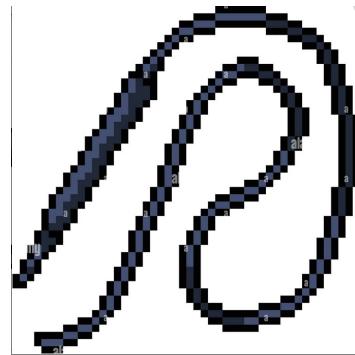


Paleta de colores: Rosa, salmón y naranja.

- Se mantiene en estado dormido cuando inicia el nivel.
- Se ataca al jugador en tiempos aleatorios entre 5-15 segundos.
- Comienza a atacar al jugador cuando el jugador entra en contacto con él por primera vez.
- El artefacto que protege, “Látigo de Agua”, aparece cuando el búho está despierto.
- Su ataque le resta 1, 2 y 3 puntos al jugador en el nivel correspondiente.
- El jugador no le puede hacer daño.

## “Látigo de agua”

Característica y Puntuación



Paleta de colores: Diferentes tonos de gris y azul.

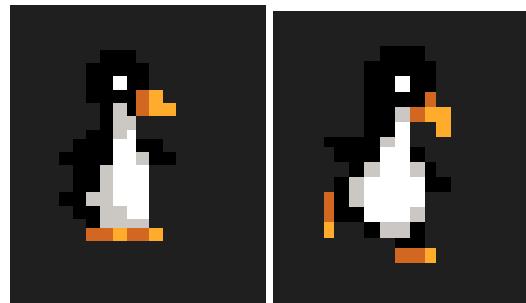
- Símbolo y artefacto mágico que protege al reino Seemar.
- Rápido y peligroso como los lagos y ríos que protegen a dicho reino.
- Le da al usuario el poder de que sus movimientos sean más rápidos incrementando la velocidad del jugador de un 1 a 2.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## “Pingüino”

Características y Puntuación



Paleta de colores: Negro, Blanco y naranja.

- Se mantiene en estado dormido cuando inicia el nivel.
- Se ataca al jugador en tiempos aleatorios entre 5-15 segundos.
- Comienza a atacar al jugador cuando el jugador entra en contacto con él por primera vez.
- El artefacto que protege, “La espada de Hielo”, aparece cuando el búho está despierto.
- Su ataque le resta 1, 2 y 3 puntos al jugador en el nivel correspondiente.
- El jugador no le puede hacer daño.

## “Espada de Hielo”

Característica y Puntuación



Paleta de colores: Diferentes tonos de gris y azul.

- Símbolo y artefacto mágico que protege al reino Seemar.
- Rápido y peligroso como los lagos y ríos que protegen a dicho reino.
- Le da al usuario el poder de que sus movimientos sean más rápidos incrementando la velocidad del jugador de un 1 a 2.



## “Monstruo”

### Características y Puntuación



Paleta de colores: Varios tonos de café, gris y blanco.

- Se sigue al jugador y le hace daño a este al entrar en contacto con ellos. Inicia con **15 puntos de vida en el primer nivel**. Monstruos de segundo nivel tiene 24 puntos. Y de los últimos niveles tiene 30 puntos.
- Un ataque de él le resta 1 punto al jugador.
- Luego de atacar al jugador, reposa por 3 segundos y comienza a seguirlo de nuevo.
- Única debilidad son la hierba Shigmul:
  - Modo fácil= El shigmul les resta 3 puntos de vida.
  - Modo medio=El shigmul les resta 4 puntos.
  - Modo difícil= El shigmul les resta 5 puntos.

## “Shigmul”

### Características y Puntuación



Paleta de colores: Verde, verde claro, negro y café.

- Hierba utilizada para combatir contra monstruos.
- Única debilidad que se ha encontrado de los monstruos.
- Crece en todo el bosque oscuro y en todas condiciones, incluyendo los diferentes niveles del juego.
- La cantidad de puntos de vida que le quita al monstruo equivale a la fuerza de ataque del jugador.



## “Gemas”

Características y Puntuación

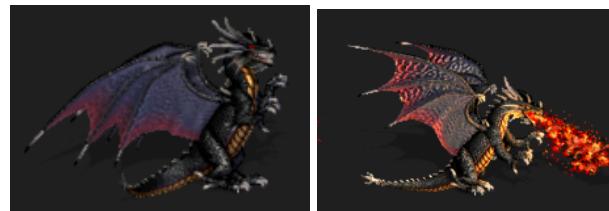


Paleta de colores: Rojo oscuro, rojo, blanco y naranja.

- Gemas mágicas que solo se encuentran en el bosque oscuro y que los reinos dentro del juego consideran un lujo.
- Sirven para enfrentarse con el dragón y para comunicarse con él.
- **Le da 2 puntos de vida al jugador** cada gema que obtiene.
- El jugador ocupa un mínimo de 60% de las gemas obtenidas entre los 3 niveles del juego para poder enfrentarse al dragón, si no, el jugador empieza el juego de nuevo.
- Una gema le resta 2 puntos al dragón.
- Si el jugador obtiene mínimo el 85% de gemas entre los 3 niveles del juego, el sistema le dará la opción de hablar con el dragón al escuchar su historia y el jugador elegirá si pelear con el dragón o seguir la propuesta del este.

## “Dragón”

Características y Puntuación



Paleta de colores: Negro, naranja, morado, rojo, gris.

- Ladrón de los artefactos que esconde cada uno de ellos en áreas conectadas del bosque oscuro.
- Tiene 50 puntos de vida al momento de iniciar el enfrentamiento con el jugador.
- Su ataque le resta 4 puntos al jugador.
- Las gemas mágicas logran lastimarlo (le restan 2 puntos) pero si un personaje tiene una cantidad grande de gemas entonces le dan la habilidad de hablar con el dragón.
- Si el jugador escogió hablar con el dragón se mostrará la siguiente información:
  - El dragón salió de su huevo hace menos de un año y luego de observar el mundo donde vivía se dio cuenta de que ya no había más de su especie y que no podía comunicarse con otros. El dragón reconoció que los artefactos contienen magia de dragones por lo que se los robó para tener algo familiar con él.
  - El dragón le propone regresar los artefactos si el jugador se vuelve su amigo con quien conversar. Si el jugador rechaza la propuesta, tendrá que pelear con el dragón, si la acepta, automáticamente ganará el juego.



#### 4. “1.er Nivel”

##### Laberinto

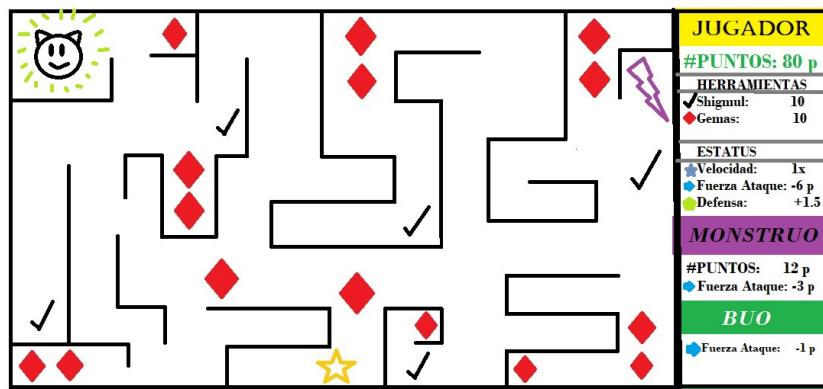
Este nivel consiste en un laberinto a la entrada de la guarida del dragón dentro del bosque oscuro hecho por el dragón para almacenar el artefacto Escudo de madera.

El protagonista debe de encontrar la ruta que lo lleve al primer protector del artefacto mágico, el búho, quien está en un estado de reposo mientras no entra en contacto con el jugador, pero el jugador tiene que buscarlo antes de que un monstruo lo mate. Los ataques del jugador impactan a todo el cuarto donde esta, por lo que puede atacar desde cualquier lugar al monstruo, pero solo cada 10 segundos

Este nivel otorgará al protagonista gemas y Shigmul, y el Shigmul puede ser recogido y usado por el jugador para atacar al monstruo. El monstruo puede dañar al protagonista al momento de encontrarlo, por lo cual el usuario tendrá que decidir si matar al monstruo o escapar de él.

La primera vez que el jugador entre en contacto con el guardián del artefacto, en este caso el búho, éste lo comenzará a atacar en intervalos de tiempo aleatorio que van de 5-15 segundos ya que intenta proteger al artefacto. Cuando el usuario haya encontrado al primer protector, este hará visible al artefacto, es decir al escudo. Y en cuanto haga contacto con el artefacto, el jugador acabará el nivel.

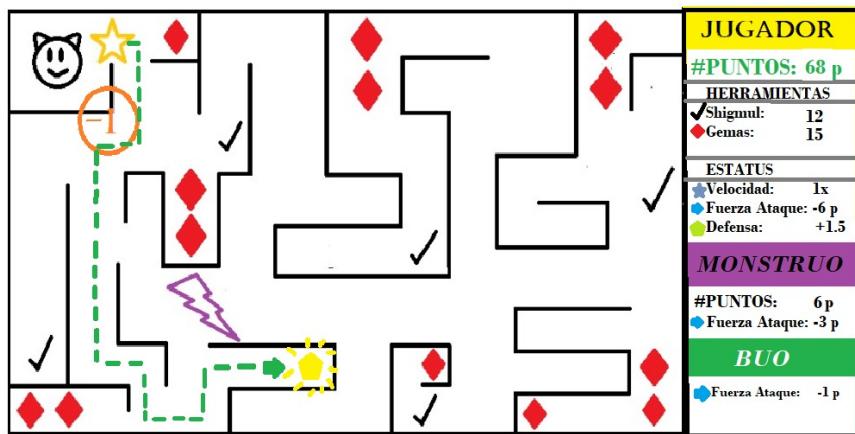
**Escenario 1:** En la esquina superior derecha se encontrará el enemigo, en la esquina contraria se encontrará el guardián, y el protagonista se encontrará en la parte media del lado inferior.



**Camino correcto:** Cuando el protagonista se acerque el protector, el búho, este revelara el lugar del escudo, por el cual el protagonista debe de agarrar el escudo evitando tanto los ataques del búho como del monstruo. Cuando el protagonista recoja el escudo, el mismo búho desaparecerá y se le mostrará la puerta para que avance al siguiente nivel.



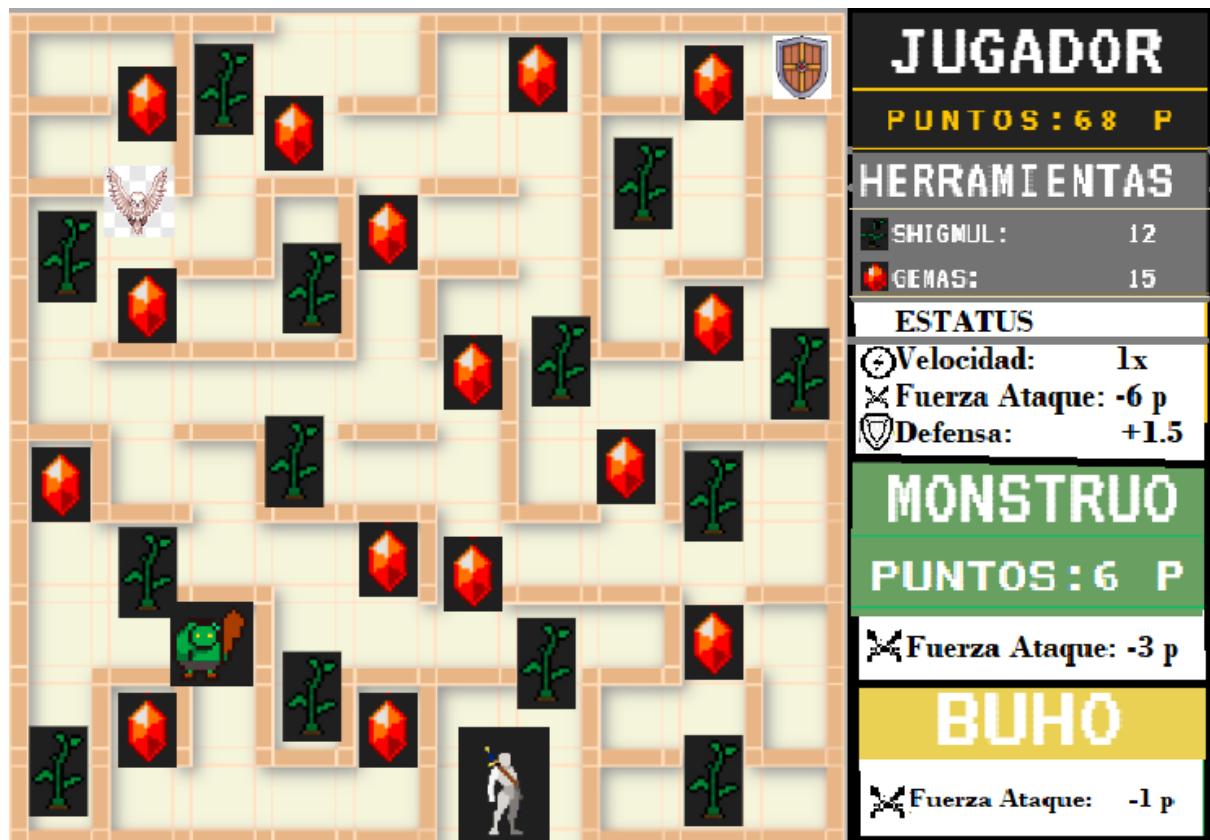
Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez



El símbolo de sol es el lugar en donde se encontrará el artefacto.

En la parte donde dice -1 es donde el búho comenzará a atacar al protagonista.

### PROTOTIPO-2da versión:



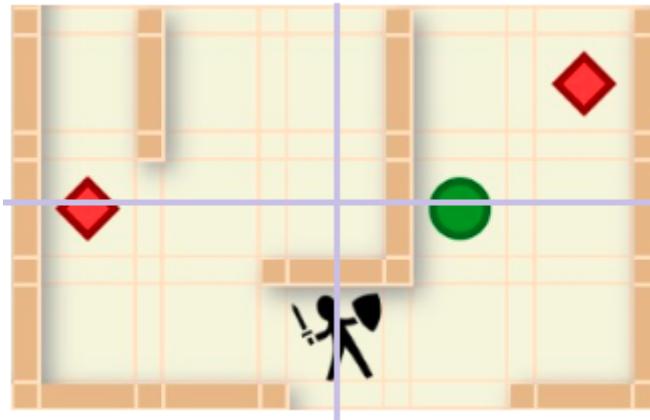


## 5. TIPO DE CÁMARA: Position-Locking

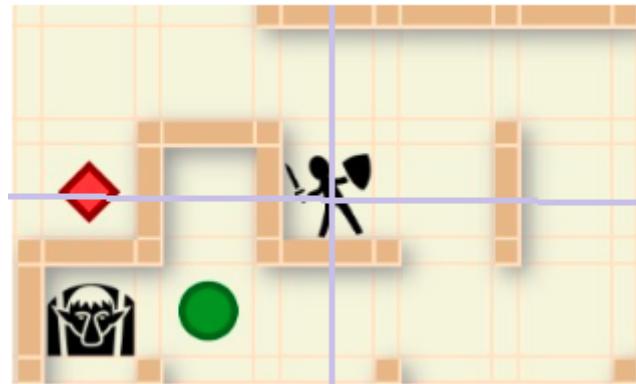
La cámara escogida para el videojuego fue la de Position-Locking, y se refiere a que la cámara se queda fija en la posición de un objeto, cuando dicho objeto se mueve, también lo hace la cámara, dando el efecto que lo va siguiendo. En el caso de nuestro juego, cada nivel del mundo se desarrolla en una especie de cuarto conforme a una temática relacionado a uno de los artefactos que debe de recuperar el jugador. Por lo tanto, el jugador podrá recorrer todo el cuarto hasta llegar a los bordes de este por lo que la cámara igual solo mostrara las partes del mundo mas un espacio reducido de color blanco indicando que ya no existe mas información relevante fuera del cuarto.

Esta cámara utilizada en videojuegos proporciona al jugador una vista más intuitiva y dinámica del mundo del juego. Por ejemplo, en el caso del primer nivel del juego que consiste en un laberinto, el jugador al principio tendrá una vista limitada del laberinto que podría aumentar mientras se mueva a través del mundo. Esto se logra gracias a que la cámara lo estará siguiendo y estará revelando poco a poco más del nivel y de las ubicaciones de los monstruos, el guardián(búho) así como las gemas y shigmuls.

### Ejemplos de posiciones de Cámara



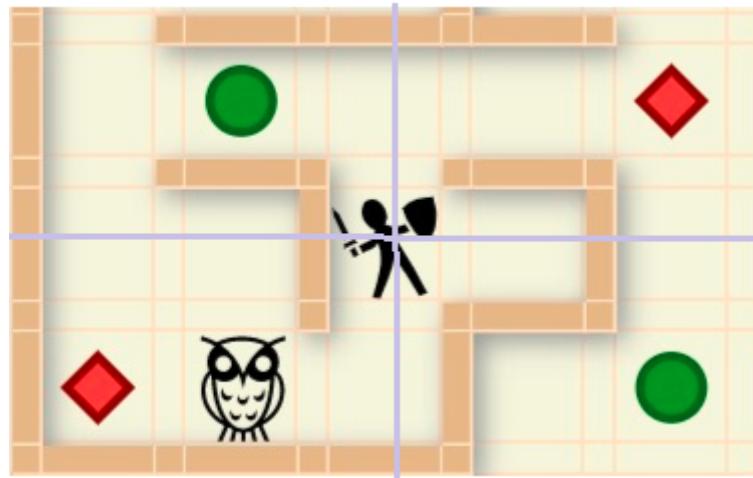
**Imagen 1.** El jugador tendrá visualización reducida del laberinto al inicio del juego.



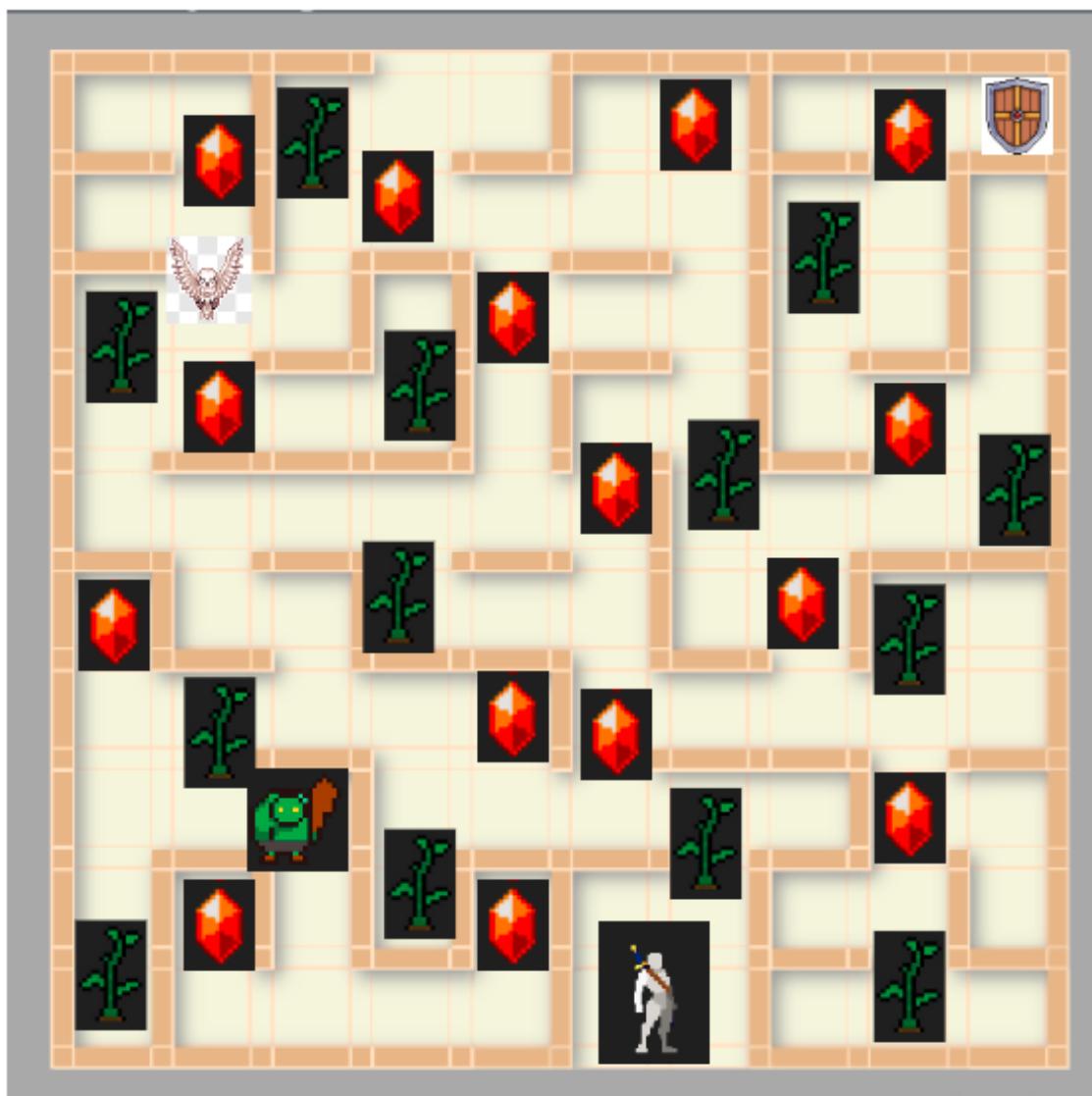
**Imagen 2.** Una vez que el jugador vaya avanzando, la cámara lo seguirá y estará revelando poco a poco el mapa del juego, al igual que las localizaciones de atributos y del mismo monstruo.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez



**Imagen 3.** El jugador tendrá que ir a ciegas con lo poco que se revela del mismo hasta toparse con el primer guardián quien es el búho.

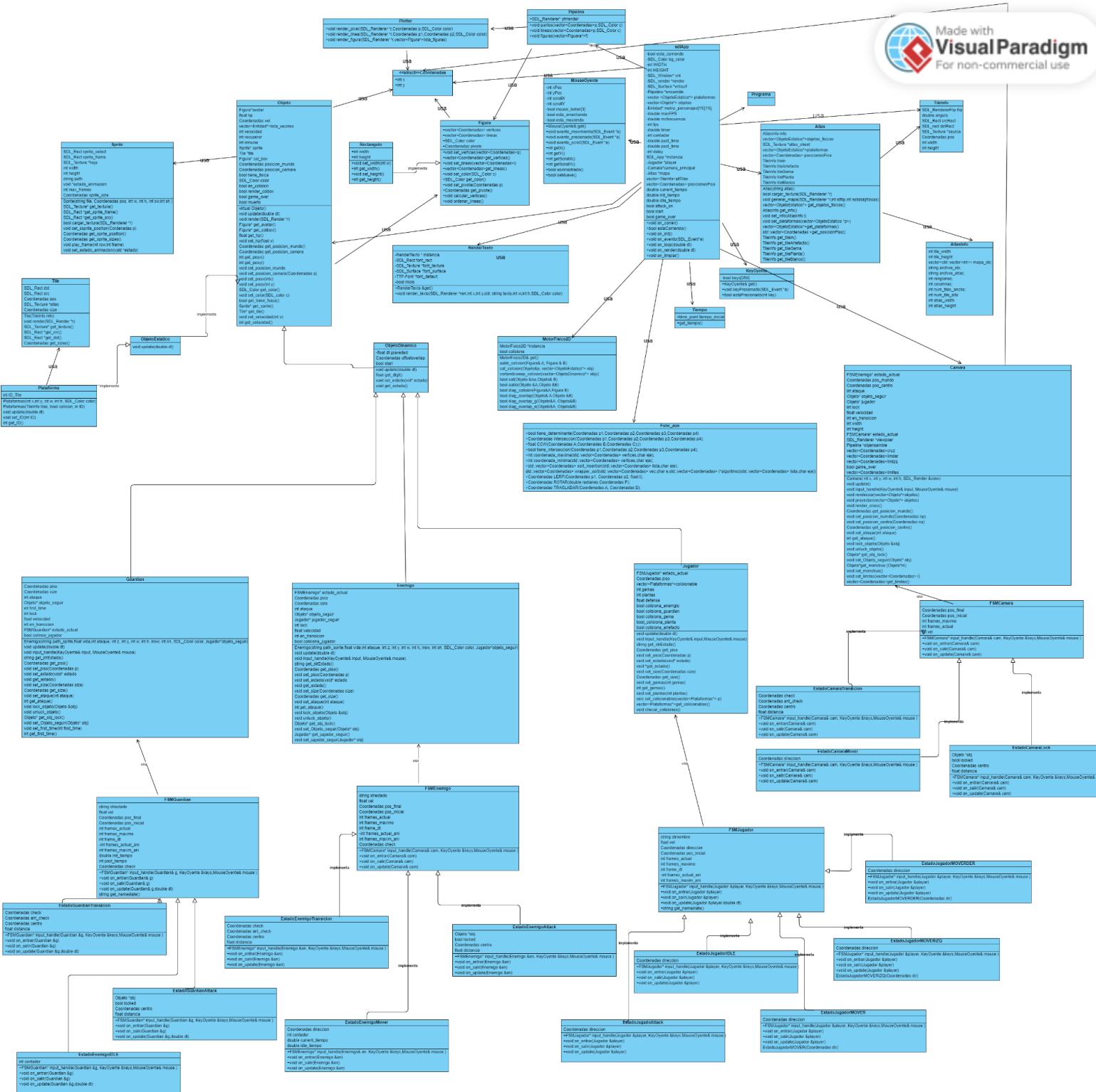


**Imagen 4.** Prototipo de laberinto completo.



## 6. “Diseño de clases”

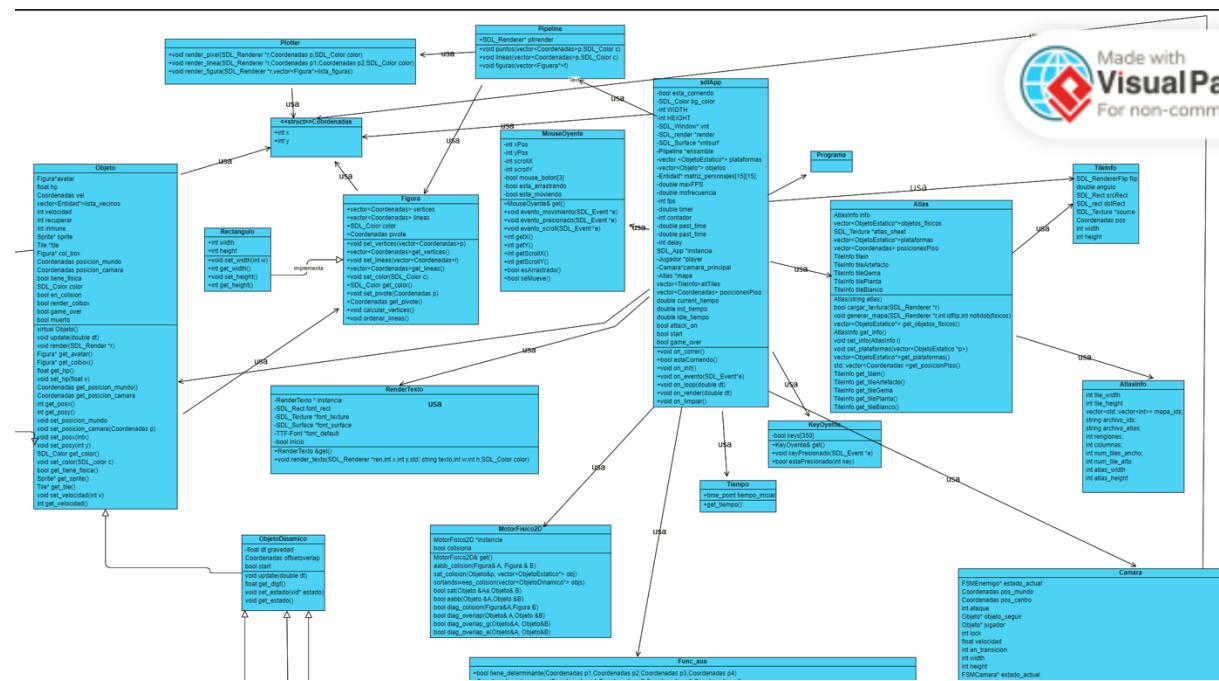
**NOTA:** Mejor visualizado en el archivo *JUEGO-LostArtifact-DC.svg* colocado en el repositorio de github del juego. Este es el link:  
<https://github.com/MaribelMOA/Game2D-Equipo3>





## 7. “Mecánicas de las clases”

En la clase principal, **SDLApp**, es donde se llamará a todos los objetos que se utilizaran durante el juego. Aquí se crearán e dibujara con las clases *pipeline* y *plotter* el mapa del juego con la clase *Atlas*, los objetos dinámicos como el *jugador*, *el enemigo* y *el guardián* con sus atributos respectivos, así como los objetos estáticos como el conjunto de *gemas*, *plantas* y *el artefacto* a través del mapa. Todos los objetos se inicializan de manera aleatoria en el mapa y en el caso del artefacto, este no será visible al inicio del juego.



### Método Init

#### Generar posiciones aleatorias de objetos dinámicos

Para la posiciones de los objetos dinámicos como jugador, enemigo y guardián, se recorre las un arreglo de las coordenadas de aquellos tiles que representan el piso (generado en el objeto *Atlas*), y se escoge un índice aleatorio e único en donde inicia cada objeto.

#### Inicializar las gemas, plantas y el artefacto.

Se recorre la lista de plataformas que representan los tiles del mundo obtenido del Objeto *Atlas*. Se checa si el ID de la plataforma es del piso y si es así, hay un 50% de probabilidad de que sea reemplazado por el tile de algún objeto estático como gema, planta o artefacto. Luego hay otra condición para cambiar el tile de la plataforma con probabilidad del 5%.

#### Inicializar Cámara principal:

En la cámara principal representada por la clase *Cámara* sigue al jugador y va mostrando y moviendo parte del mundo dependiendo del movimiento del jugador.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

## Pseudocódigo

*Si el resultado de "SDL\_Init(SDL\_INIT\_EVERYTHING)" es menor que 0, entonces retornar "false".*

- *Crear una ventana con los siguientes parámetros:*
- *Título de la ventana: "Juego"*
- *Posición X de la ventana: "SDL\_WINDOWPOS\_UNDEFINED"*
- *Posición Y de la ventana: "SDL\_WINDOWPOS\_UNDEFINED"*
- *Ancho de la ventana: "get().WIDTH" (ancho establecido anteriormente)*
- *Alto de la ventana: "get().HEIGHT" (tamaño anteriormente establecido)*
- *Uso de OpenGL: "SDL\_WINDOW\_OPENGL"*

*Iniciar la biblioteca TTF.*

*Iniciar la biblioteca IMG con soporte para archivos PNG y JPG.*

*Revisar si la ventana no se creó correctamente. Si es así, imprimir el mensaje "No se creó la ventana por: " seguido del mensaje de error obtenido con "SDL\_GetError()", y retornar "false".*

*Configurar la forma de procesamiento en GPU utilizando OpenGL mediante "SDL\_SetHint(SDL\_HINT\_RENDER\_BATCHING, "opengl")".*

*Crear el renderizador (canvas) con los siguientes parámetros:*

- *La ventana "get().vnt".*
- *El driver "-1" (que se seleccionará automáticamente).*
- *Los flags del driver "SDL\_RENDERER\_ACCELERATED | SDL\_RENDERER\_PRESENTVSYNC".*

*Revisar si el renderizador no se creó correctamente.*

*De ser así, imprimir el mensaje "No se creó el renderizer por: " seguido del mensaje de error obtenido con "SDL\_GetError()", y retornar "false".*

*Si se creó correctamente el renderizador, agregarlo al Pipeline (proceso no especificado en el código proporcionado).*

*Se crea una instancia de "Pipeline" y asignarla a "get().ensamble", pasando como parámetro el renderizador "\*get().render".*

*Obtener el tiempo actual utilizando "Tiempo::get\_tiempo()" y asignarlo a "current\_tiempo".*

*Se le asigna el valor "10" a "idle\_tiempo".*



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

*Crear una instancia de "Atlas" y asignarla a "mapa", pasando como parámetros la ruta del archivo "assets/sprites/mundo/ids/FinalAtlas\_ids.txt".*

*Generar el mapa llamando al método "generar\_mapa" de "mapa", pasando como parámetros el renderizador "get().render", el valor "2" y el valor "0".*

*Obtener las posiciones de los pisos mediante "mapa->get\_posicionesPiso()" y asignarlas a "posicionesPiso".*

*Asignar el valor "false" a "attack\_on", "start" y "game\_over".*

*Crear tres variables "random1", "random2" y "random3" de tipo entero.*

*Generar números aleatorios entre 0 y el tamaño de "posicionesPiso" (exclusivo) utilizando el operador "%" y asignarlos a las variables "random1", "random2" y "random3" respectivamente.*

*Entrar en un bucle "while" que se ejecuta mientras "random1" sea igual a "random2", o "random1" sea igual a "random3", o "random2" sea igual a "random3". Dentro del bucle, generar nuevos números aleatorios y actualizar las variables "random1", "random2" y "random3" hasta que se cumplan las condiciones.*

*Asignar las coordenadas correspondientes a las posiciones aleatorias a las variables "posJugador", "posEnemigo" y "posGuardian", obteniéndolas de "posicionesPiso" utilizando los índices "random1", "random2" y "random3" respectivamente.*

*Crear una instancia de "Jugador" llamada "player", pasando como parámetros la ruta de la imagen "assets/sprites/hero/myHero.png", los valores "100", "3", las coordenadas "posJugador.x" y "posJugador.y", "135", "93", "100", "70" y el color "{255,0,255,255}".*

*Cargar las texturas del jugador al "Pipeline" llamando al método "cargar\_texturas" de "get().ensamble" y pasando como parámetro "player->get\_sprite()".*

*Establecer el número de plantas del jugador en "10" llamando al método "set\_plantas" de "player".*

*Crear una instancia de "Enemigo" llamada "enemigo", pasando como parámetros la ruta de la imagen "assets/sprites/hero/monster.png", los valores "50", "1", las coordenadas "posEnemigo.x" y "posEnemigo.y", "100", "87", "100", "100" y el color "{255,0,255,255}".*

*Crear una instancia de "Guardian" llamada "guardian", pasando como parámetros la ruta de la imagen "assets/sprites/hero/buho3.png", los valores "15", "1", las coordenadas "posGuardian.x" y "posGuardian.y", "60", "50", "100", "100" y el color "{255,0,255,255}".*



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

*Cargar las texturas del enemigo y del guardián al "Pipeline" llamando al método "cargar\_texturas" de "get().ensamble" y pasando como parámetros "enemigo->get\_sprite()" y "guardian->get\_sprite()" respectivamente.*

*Obtener los objetos físicos del mapa llamando al método "get\_objetos\_fisicos" de "mapa" y asignarlos a la variable "plataformas".*

*Agregar los tiles de gema, planta, artefacto, blanco e in a "allTiles" llamando a los métodos "get\_tileGema", "get\_tilePlanta", "get\_tileArtefacto", "get\_tileBlanco" y "get\_tilein" de "mapa" respectivamente.*

*Recorrer cada elemento "p" en "plataformas" utilizando un bucle "for" y realizar lo siguiente:*

- Obtener el ID del elemento llamando al método "get\_ID" de "p" y asignarlo a la variable "id".
- Si "id" es igual a "ARTEFACTO" y el valor de "guardian->get\_first\_time" es igual a 0, realizar lo siguiente:
  - Establecer el tile del elemento como un nuevo objeto "Tile" creado a partir del elemento "allTiles[PISO-1]".
  - Establecer el ID del elemento como "PISO".

*Inicializar las variables "numGemas" y "numPlantas" con el valor 0, y "numBlancos" con el valor 0.*

*Crear una variable booleana "existeArtefacto" y asignarle el valor "false".*

*Recorrer cada elemento "p" en "plataformas" utilizando un bucle "for" y realizar lo siguiente:*

- Obtener el ID del elemento llamando al método "get\_ID" de "p" y asignarlo a la variable "id".
- Si "id" es igual a "PISO" y el resultado de "rand()%100" es menor que 50, realizar lo siguiente:
  - -Si el resultado de "rand()%100" es menor que 5 y "numGemas" es menor o igual a 30, realizar lo siguiente:
    - Establecer el tile del elemento como un nuevo objeto "Tile" creado a partir del elemento "allTiles[GEMAS-1]".
    - Establecer el ID del elemento como "GEMAS".
    - Incrementar "numGemas" en 1.
  - Si el resultado de "rand()%100" es menor que 5 y "numPlantas" es menor o igual a 30, realizar lo siguiente:
    - Establecer el tile del elemento como un nuevo objeto "Tile" creado a partir del elemento "allTiles[PLANTAS-1]".
    - Establecer el ID del elemento como "PLANTAS".



- Incrementar "numPlantas" en 1.
- Si el resultado de "rand()%100" es menor que 5 y "existeArtefacto" es igual a "false", realizar lo siguiente:
  - Establecer el tile del elemento como un nuevo objeto "Tile" creado a partir del elemento "allTiles[ARTEFACTO-1]".
  - Establecer el ID del elemento como "ARTEFACTO".
  - Asignar "true" a la variable "existeArtefacto".
- Si el resultado de "rand()%100" es menor que 5 y "numBlancos" es menor a 10, realizar lo siguiente:
  - Establecer el tile del elemento como un nuevo objeto "Tile" creado a partir del elemento "allTiles[ATTACK\_ZONE-1]".
  - Establecer el ID del elemento como "ATTACK\_ZONE".
  - Incrementar "numBlancos" en 1.

Crear una nueva instancia de la clase "Camara" llamada "camara\_principal" con los parámetros de posición (0, 0), ancho y alto obtenidos de "get().WIDTH" y "get().HEIGHT", respectivamente, y el objeto renderizador obtenido de "\*get().render".

Agregar todos los elementos de "plataformas" a la lista "objetos".

Agregar los siguientes elementos a la lista "objetos": "player", "enemigo" y "guardian".

Imprimir en pantalla el mensaje "Se crearon los test exitosamente".

Establecer el color de fondo del renderizador utilizando "SDL\_SetRenderDrawColor", pasando como argumentos el objeto renderizador obtenido de "get().render" y los valores de color "get().bg\_color.r", "get().bg\_color.g" y "get().bg\_color.b". El último argumento "SDL\_ALPHA\_TRANSPARENT" indica que se utilizará el valor de transparencia alpha especificado.

Devolver "true" para indicar que el proceso se realizó exitosamente.

## Método física update

Aquí se estará actualizando constantemente todos los objetos utilizados en el juego y se estará checando las colisiones entre los objetos para cambiar el booleano de en colisión del jugador con ayuda de la instancia de *MotorFisico2D*. Pero como dependiendo de con que este colisionando el jugador se realizarán diferentes acciones, habrá una condición que esté checando por cual objeto fue causado la colisión y cambiando las variables booleanas de colisión respectivas de cada objeto. Si el jugador colisiona con el monstruo y está en estado atacar, se le restará vida al monstruo. El jugador solo podrá atacar al monstruo cada 10 segundos y se le indicara tanto con texto mostrado por RenderTexto como con la aparición de



tiles de color blanco. Por otro lado, si el jugador colisionó por primera vez con el búho/guardián (lo cual se checa en la clase FSMEstados del guardián), se hará visible el objeto estático representando el artefacto. Aquí igual checa si el jugador colisionó con el artefacto ya visible, y cuando sucede esto, termina el juego y se le indica cambiando a verdadero la variable booleana game\_over de todos los objetos dinámicos.

## Pseudocódigo

Si la tecla "L" está presionada, se realizan las siguientes acciones:

- Se establece "start" como verdadero.
- Se establece "camara\_principal->start" como verdadero.
- Se establece "enemigo->start" como verdadero.
- Se llama al método "lock\_objeto" de "camara\_principal", pasando "\*player" como argumento.
- Se llama al método "lock\_objeto" de "enemigo", pasando "\*player" como argumento.
- Se llama al método "lock\_objeto" de "guardian", pasando "\*player" como argumento.

Se llama al método "input\_handle" de "player", pasando las instancias de "KeyOyente::get()" y "MouseOyente::get()" como argumentos.

Se calcula la diferencia de tiempo "elapsed\_tiempo" entre el tiempo actual y "current\_tiempo". Si "elapsed\_tiempo" es mayor que "idle\_tiempo", se realizan las siguientes acciones:

- Si "attack\_on" es falso, se establece como verdadero.
- De lo contrario, se establece como falso.
- Se llama al método "set\_attack\_on" de "player", pasando "attack\_on" como argumento.
- Se actualiza el valor de "current\_tiempo" al tiempo actual.

Si la tecla "J" está presionada y "attack\_on" es verdadero, se realizan las siguientes acciones:

- Si la cantidad de plantas de "player" es mayor que 0, se reduce el HP de "enemigo" por el valor de ataque de "player" y se disminuye la cantidad de plantas de "player" en 1.

Para cada objeto "p" en la lista "plataformas", se realizan las siguientes acciones:

- Se obtiene el ID del objeto "p".
- Si el ID es igual a "ATTACK\_ZONE":
  - Si "attack\_on" es verdadero y "start" es verdadero, se establece el tile del objeto "p" como el tile de la zona de ataque.
  - De lo contrario, se establece el tile del objeto "p" como el tile de piso.
- Si el ID es igual a "MURO":



- Se realiza una prueba de superposición entre "player" y "p" utilizando el motor de física "MotorFisico2D".
- Se establece "player->en\_colision" como falso.
- Si el ID es igual a "GEMAS" y "start" es verdadero:
  - Se realiza una prueba de superposición entre "player" y "p" utilizando el motor de física "MotorFisico2D".
  - Si "player->en\_colision" es verdadero:
    - Se establece el tile del objeto "p" como el tile de piso.
    - Se cambia el ID del objeto "p" a "PISO".
    - Se establece "player->en\_colision" como falso.
    - Se aumenta el HP de "player" en 2.
    - Se establece "player->colision\_gemas" como verdadero.
- Si el ID es igual a "PLANTAS" y "start" es verdadero:
  - Se realiza una prueba de superposición entre "player" y "p" utilizando el motor de física "MotorFisico2D".
  - Si "player->en\_colision" es verdadero:
    - Se establece el tile del objeto "p" como el tile de piso.
    - Se cambia el ID del objeto "p" a "PISO".
    - Se establece "player->en\_colision" como falso.
    - Se establece "player->colision\_plantas" como verdadero.
    - Se incrementa la cantidad de plantas de "player" en 1.
- Si el ID es igual a "ARTEFACTO" y "guardian->get\_first\_time()" es igual a 0:
  - Se establece el tile del objeto "p" como el tile de piso.
- De lo contrario, si el ID es igual a "ARTEFACTO" y "guardian->get\_first\_time()" es igual a 1:
  - Se establece el tile del objeto "p" como el tile del artefacto.
  - Se realiza una prueba de superposición entre "player" y "p" utilizando el motor de física "MotorFisico2D".
  - Si "player->en\_colision" es verdadero:
    - Se establece el tile del objeto "p" como el tile de piso.
    - Se establece "player->colision\_artefacto" como verdadero.
    - Se aumenta el HP de "player" en 25.
    - Se establece "player->game\_over", "enemigo->game\_over", "guardian->game\_over", "camara\_principal->game\_over" y "game\_over" como verdaderos.

Para cada objeto "p" en la lista "objetos", se llama al método "update(dt)" del objeto "p".

Se llama al método "update(dt)" de "player".

Se llama al método "update(dt)" de "enemigo".

Se obtiene el estado del enemigo mediante el método "get\_strEstado()" y se almacena en la variable "estado".



Se realiza una colisión entre "player" y "enemigo" usando el método "diag\_overlap\_e()" de "MotorFisico2D".

Si hay colisión entre "player" y "enemigo", se reduce la salud ("hp") del jugador en función del ataque del enemigo.

Se llama al método "update(dt)" de "guardian".

Se realiza una colisión entre "player" y "guardian" usando el método "diag\_overlap\_g()" de "MotorFisico2D".

Si hay colisión entre "player" y "guardian", se establece el valor de "first\_time" del guardian a 1 y se reduce la salud ("hp") del jugador en función del ataque del guardian.

Si la salud del jugador llega a 0 o menos, se establece el estado "game\_over" en true y se establece el estado de juego ("game\_over") de "player", "enemigo", "guardian" y "camara\_principal" en true.

Si la salud del enemigo llega a 0 o menos, se establece el estado "muerto" del enemigo en true.

Se llama al método "input\_handle()" de "camara\_principal" pasando las instancias "KeyOyente::get()" y "MouseOyente::get()" para manejar la entrada de teclado y ratón.

Se llama al método "update()" de "camara\_principal".

Se llama al método "proyectar()" de "camara\_principal" pasando la lista "objetos" para actualizar la proyección de los objetos en la cámara.

## Método frame update

Aquí mostramos el mensaje de cómo iniciar el juego, los resultados actuales de los objetos dinámicos jugador, enemigo y guardian así como que ha finalizado el juego, ya sea ganado o perdido, con ayuda de la clase RenderTexto.

## Pseudocódigo

Se limpia el frame del renderizador utilizando el método "SDL\_RenderClear()" y pasando el renderizador obtenido mediante "get().render".

Se obtiene la posición del mouse.

Se calcula el número de cuadros por segundo (FPS) dividiendo "dt" (delta de tiempo) entre "get().msfrecuencia" (frecuencia de actualización en milisegundos).



Se renderiza el texto del FPS utilizando el método "render\_texto()" de "RenderTexto::get()". Se pasan los parámetros: el renderizador obtenido mediante "get().render", la posición X (get().WIDTH-200), la posición Y (30), el texto a renderizar (el número de FPS convertido a string), el tamaño de la fuente (100), el espaciado de la fuente (30) y el color del texto (SDL\_Color{0,135,62}).

Renderiza los objetos en la cámara principal.

Se crean variables de tipo cadena para almacenar información sobre el jugador, enemigo y guardian:

- vidaJugador: Almacena la cadena "Vida: " concatenada con la vida del jugador convertida a cadena.
- gemasJugador: Almacena la cadena "Gemas: " concatenada con la cantidad de gemas del jugador convertida a cadena.
- plantasJugador: Almacena la cadena "Plantas: " concatenada con la cantidad de plantas del jugador convertida a cadena.
- ataqueJugador: Almacena la cadena "Fuerza de Ataque: " concatenada con el valor de ataque del jugador convertido a cadena.
- vidaEnemigo: Almacena la cadena "Vida: " concatenada con la vida del enemigo convertida a cadena.
- ataqueEnemigo: Almacena la cadena "Fuerza de Ataque: " concatenada con el valor de ataque del enemigo convertido a cadena.
- ataqueGuardian: Almacena la cadena "Fuerza de Ataque: " concatenada con el valor de ataque del guardian convertido a cadena.

Si start es verdadero, entonces:

1. Renderizar el texto "JUGADOR" en la posición (50, 50) con color rojo.
2. Renderizar el texto de la variable vidaJugador en la posición (50, 90) con color rojo.
3. Renderizar el texto de la variable gemasJugador en la posición (50, 130) con color rojo.
4. Renderizar el texto de la variable plantasJugador en la posición (50, 170) con color rojo.
5. Renderizar el texto de la variable ataqueJugador en la posición (50, 200) con color rojo.
6. Renderizar el texto "ENEMIGO" en la posición (400, 50) con color verde.
7. Renderizar el texto de la variable vidaEnemigo en la posición (400, 100) con color verde.
8. Renderizar el texto de la variable ataqueEnemigo en la posición (400, 150) con color verde.
9. Renderizar el texto "GUARDIAN" en la posición (700, 50) con color azul.



10. Renderizar el texto de la variable ataqueGuardian en la posición (700, 100) con color azul.

- Si attack\_on es verdadero y player->get\_colision\_artefacto() es falso y game\_over es falso y enemigo->get\_muerto() es falso, entonces:
  - Renderizar el texto "ATTACK NOW" en la posición (400, 300) con color amarillo.
  -
- Si player->get\_hp() es menor o igual a 0 y game\_over es verdadero, entonces:
  - Renderizar el texto "GAME OVER" en la posición (400, 350) con color morado.
  - Establecer enemigo->set\_muerto() como verdadero.
  - Establecer game\_over como verdadero.
  - Si game\_over es verdadero, entonces:
    - Si KeyOyente::get().estaPresionado(SDL\_SCANCODE\_L) es verdadero, entonces:
      - Reiniciar el juego.
- Si player->get\_colision\_artefacto() es verdadero, entonces:
  - Renderizar el texto "YOU WIN" en la posición (400, 430) con color morado.
  - Establecer enemigo->set\_muerto() como verdadero.
  - Establecer game\_over como verdadero.
  - Si game\_over es verdadero, entonces:
    - Si KeyOyente::get().estaPresionado(SDL\_SCANCODE\_L) es verdadero, entonces:
      - Reiniciar el juego.

Si start es falso entonces:

- Renderizar el texto "PRESS L TO START" en la posición (400, 300) con ancho 200, altura 90 y color amarillo.

Actualizar la pantalla mediante `SDL_RenderPresent(get().render)`.

Restablecer el color del marco utilizando `SDL_SetRenderDrawColor()` con los componentes RGB obtenidos de `get().bg_color` y el valor de transparencia `SDL_ALPHA_TRANSPARENT`.

## Clase MotorFisica2d:

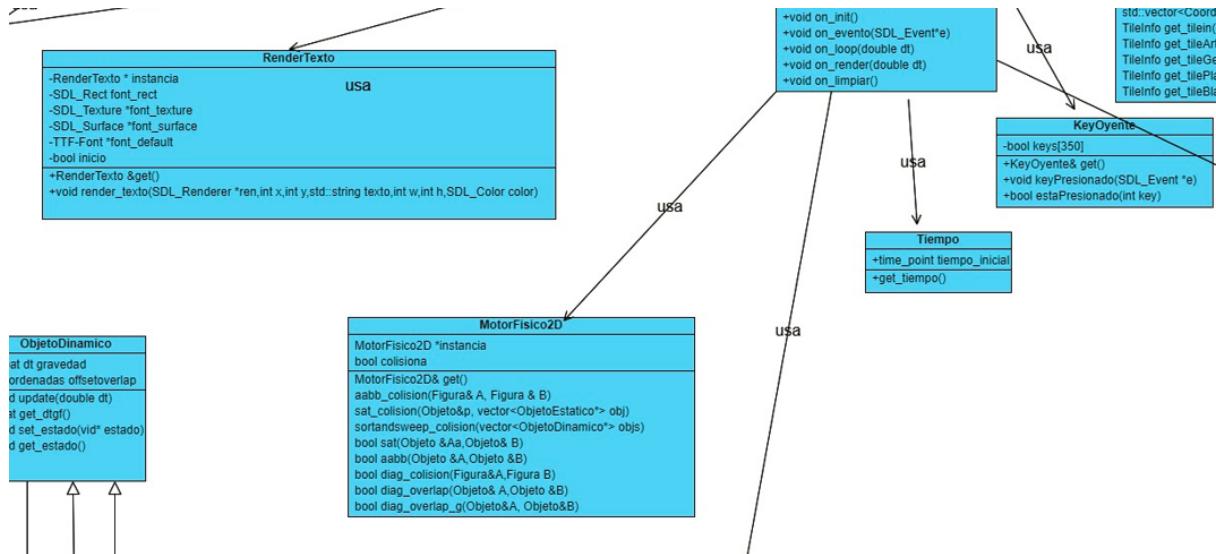
### *Método diag\_overlap\_g*

Aquí se checan si la figura col\_box de cada objeto pasado por parámetro a este método, sobrelapan entre sí. Esto se hace sacando la distancia del centro a la diagonal y la intersección de líneas. Si se encuentra que hay intersección y sobrelapan, se guarda lo que se le tiene que



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

compensar para luego mover el objeto y se cambia a verdadero el atributo booleano de en colisión.



## Pseudocódigo

Definir TA como referencia a A

Definir TB como referencia a B

Si i es igual a 1 entonces:

- Asignar TB como referencia a A
- Asignar TA como referencia a B

Obtener los vértices vA del colbox de TA

Obtener los vértices vB del colbox de TB

Calcular el punto medio Ac como el promedio de los puntos  $((vA[0].x + vA[3].x) / 2, (vA[0].y + vA[1].y) / 2)$

Calcular el punto medio Bc como el promedio de los puntos  $((vB[0].x + vB[3].x) / 2, (vB[0].y + vB[1].y) / 2)$

Para cada elemento n en vA:

- Definir lineaA\_inicio como Ac
- Definir lineaA\_fin como vA[n]
- Definir offset como {0, 0}

Para cada elemento m en vB:

- Definir lineaB\_inicio como Bc
- Definir lineaB\_fin como vB[(m + 1) % tamaño de vB]



- Calcular h como  $(\text{lineaB\_fin.x} - \text{lineaB\_inicio.x}) * (\text{lineaA\_inicio.y} - \text{lineaA\_fin.y}) - (\text{lineaA\_inicio.x} - \text{lineaA\_fin.x}) * (\text{lineaB\_fin.y} - \text{lineaB\_inicio.y})$
- Calcular t1 como  $((\text{lineaB\_inicio.y} - \text{lineaB\_fin.y}) * (\text{lineaA\_inicio.x} - \text{lineaB\_inicio.x}) + (\text{lineaB\_fin.x} - \text{lineaB\_inicio.x}) * (\text{lineaA\_inicio.y} - \text{lineaB\_inicio.y})) / h$
- Calcular t2 como  $((\text{lineaA\_inicio.y} - \text{lineaA\_fin.y}) * (\text{lineaA\_inicio.x} - \text{lineaB\_inicio.x}) + (\text{lineaA\_fin.x} - \text{lineaA\_inicio.x}) * (\text{lineaA\_inicio.y} - \text{lineaB\_inicio.y})) / h$
- Si t1 es mayor o igual a 0.0 y t1 es menor que 1.0 y t2 es mayor o igual a 0.0 y t2 es menor que 1.0 entonces:
  - Incrementar offset.x por  $(1.0 - t1) * (\text{lineaA\_fin.x} - \text{lineaA\_inicio.x})$
  - Incrementar offset.y por  $(1.0 - t1) * (\text{lineaA\_fin.y} - \text{lineaA\_inicio.y})$
  - Establecer A.en\_colision como verdadero
- Obtener la posición pos de A en el mundo
- Incrementar pos.x por offset.x multiplicado por (si i es igual a 0 entonces -1, de lo contrario 1)
- Incrementar pos.y por offset.y multiplicado por (si i es igual a 0 entonces -1, de lo contrario 1)
- Establecer la posición de A en el mundo como pos

### *Método diag\_overlap\_e*

Similar a diag\_overlap\_g ya que checa si la figura col\_box de cada objeto pasado por parámetro a este método, sobrelapan entre sí. Saca la distancia del centro a la diagonal y la intersección de líneas y si se encuentra que hay intersección y sobrelapan, solo se cambia a verdadero el atributo booleano de en colisión.



Imagen:

Representación visual de como es que se verifica si hay colisión entre los objetos

## Pseudocódigo

Definir TA como referencia a A

Definir TB como referencia a B

Obtener los vértices vA del colbox de TA

Calcular Ac como el punto medio entre  $((vA[0].x + vA[3].x) / 2, (vA[0].y + vA[1].y) / 2)$

Obtener los vértices vB del colbox de TB

Calcular Bc como el punto medio entre  $((vB[0].x + vB[3].x) / 2, (vB[0].y + vB[1].y) / 2)$

Recorrer cada diagonal n en vA:

- Se define lineaA\_inicio como Ac
- Se define lineaA\_fin como vA[n]
- Se define offset como {0, 0}

Recorrer cada diagonal m en vB:

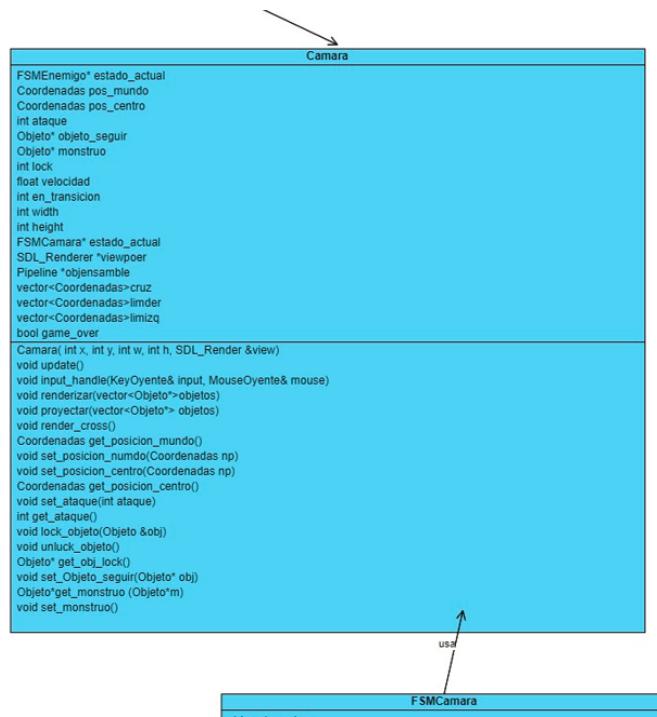
- Se define lineaB\_inicio como Bc
- Se define lineaB\_fin como vB[(m+1) % tamaño de vB]
- Se calcula h como  $(lineaB_fin.x - lineaB_inicio.x) * (lineaA_inicio.y - lineaA_fin.y) - (lineaA_inicio.x - lineaA_fin.x) * (lineaB_fin.y - lineaB_inicio.y)$
- Se calcula t1 como  $((lineaB_inicio.y - lineaB_fin.y) * (lineaA_inicio.x - lineaB_inicio.x) + (lineaB_fin.x - lineaB_inicio.x) * (lineaA_inicio.y - lineaB_inicio.y)) / h$

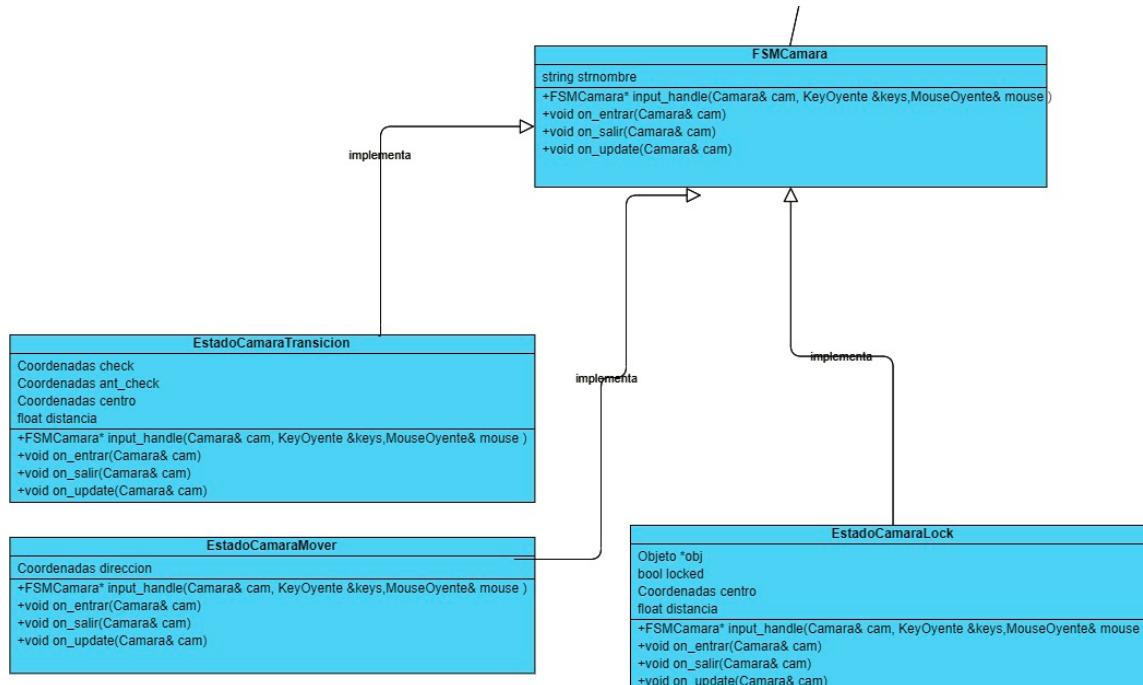


- Se calcula t2 como  $((lineaA\_inicio.y - lineaA\_fin.y) * (lineaA\_inicio.x - lineaB\_inicio.x) + (lineaA\_fin.x - lineaA\_inicio.x) * (lineaA\_inicio.y - lineaB\_inicio.y)) / h$
- Si t1 es mayor o igual a 0.0 y t1 es menor que 1.0 y t2 es mayor o igual a 0.0 y t2 es menor que 1.0 entonces:
  - A.en\_colision es verdadero

### Clase cámara:

Se creará la cámara, estará en estado de mover y se establecerá una velocidad fija para ello. Así mismo, tendrá métodos para renderizar y proyectar los objetos del mundo en la pantalla del juego lo cual va cambiando mientras se mueve la cámara. Por otro lado, tendrá métodos que dirán si está siguiendo al jugador o no. La cámara tiene una variable que representa el estado y acción que realiza. Se estará constantemente comparando las coordenadas del jugador en comparación de los límites que representan vértices de una especie de cuadro, y cuando el jugador esté dentro de cierto límite establecido, la cámara lo dejará de seguir. Pero cuando el jugador vaya más allá del límite, la cámara lo seguirá de nuevo, esto con la intención de no marear tanto al jugador.





## Pseudocódigo

### *EstadoEnemigoMover::EstadoEnemigoMover*

- `frames_actual_ani` se inicializa en 0.
- `frames_maxim_ani` se establece en 5.
- `direccion` se establece como el valor de la variable "dir".
- `strestado` se establece como "mover".
- `contador` se inicializa en 0.
- `current_tiempo` se inicializa en 0.0.
- `idle_tiempo` se establece en 5, lo cual representa un tiempo de espera de 10 segundos.

### *FSMEnemigo\* EstadoEnemigoMover::input\_handle*

Si `en.lock` es igual a 1 y `en.start` es verdadero y `en.get_muerto()` es falso, entonces:

- `en.set_estado(new EstadoEnemigoTransicion())`

De lo contrario, se retorna un nuevo `EstadoEnemigoMover` con el parámetro {0,0}.

### *void EstadoEnemigoMover::on\_entrar*

- Se asigna el valor de la velocidad del enemigo a la variable `vel`.
- Se establece `frames_actual_ani` en 0.
- Se establece `frames_maxim_ani` en 5.
- Se establece `contador` en 0.
- Se obtiene el tiempo actual utilizando la función `Tiempo.get_tiempo()` y se asigna a la variable `current_tiempo`.

### *void EstadoEnemigoMover::on\_salir*



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

No se hace nada

### ***void EstadoEnemigoMover::on\_update***

Si en.lock es igual a 1 y en.start es verdadero y en.get\_muerto() es falso, entonces:

- Se calcula elapsed\_tiempo como Tiempo::get\_tiempo() - current\_tiempo.
- Si elapsed\_tiempo es mayor que idle\_tiempo y en.get\_muerto() es falso, entonces:
- Se establece el estado del enemigo como un nuevo EstadoEnemigoTransicion().

Se reproduce el frame 0 del sprite de en.

Si frame\_dt es mayor que 5, entonces:

- frame\_dt se establece en 0.
- Se incrementa frames\_actual\_ani en 1.

Se incrementa frame\_dt en 1.

### ***EstadoEnemigoTransicion::EstadoEnemigoTransicion***

- frames\_actual se inicializa en 0.
- frames\_maximo se establece en 100.
- frames\_actual\_ani se inicializa en 0.
- frames\_maxim\_ani se establece en 5.
- strestdado se establece como "TRANSICION"

### ***FSMEnemigo\* EstadoEnemigoTransicion::input\_handle***

Si en.lock es falso entonces:

- Se retorna un nuevo EstadoEnemigoMover con las coordenadas {0, 0} como parámetro.

### ***void EstadoEnemigoTransicion::on\_entrar***

- Se asigna el valor 1 a en.en\_transicion.
- Se asigna el valor de en.velocidad a la variable vel.
- Se inicializa frames\_actual\_ani en 0.
- Se establece frames\_maxim\_ani en 5.
- Se calcula el valor de lim como la diferencia entre la coordenada x del tercer vértice y la coordenada x del primer vértice del avatar obtenido desde en.get\_obj\_lock().
- Si la distancia es menor que lim, se asigna el valor 10 a la variable frames\_maximo

### ***void EstadoEnemigoTransicion::on\_salir***

Se asigna el valor 0 a la variable en.en\_transicion.



***void EstadoEnemigoTransicion::on\_update***

Si frames\_actual es mayor que frames\_maximo, entonces

- retornar

Se obtiene la posición inicial del objeto "en"

Se obtiene la posición final del objeto al que "en" está "bloqueado"

Calcula el factor de interpolación basado en el progreso actual

Se interpola suavemente entre las posiciones inicial y final utilizando el factor "t"

Se calcula el desplazamiento en el eje X entre el check actual y el anterior

Se calcula el desplazamiento en el eje Y entre el check actual y el anterior

Se obtiene la posición del objeto al que "en" está "bloqueado"

Se calcula la distancia entre el centro y la posición del objeto "bloqueado"

Imprimir "Check: " seguido de check.x y check.y

Imprimir "Pos final: " seguido de pos\_final.x y pos\_final.y

Se establece la posición del objeto "en" en el mundo como check

Se establece el estado del objeto "en" como un nuevo EstadoEnemigoAttack basado en el objeto al que está "bloqueado"

Incrementar frames\_actual en 1

Se guarda la posición anterior en ant\_check

Reproduce el fotograma 1 en el sprite del objeto "en" basado en frames\_actual\_ani y frames\_maxim\_ani

Si frame\_dt es mayor que 5, entonces:

- se define frame\_dt = 0
- Se incrementa frames\_actual\_ani en 1

Incrementar frame\_dt en 1

***EstadoEnemigoAttack::EstadoEnemigoAttack***



- Se asigna el valor "ATTACK" a la variable strestado
- Se asigna la referencia del objeto objlock a la variable obj
- Se inicializa frames\_actual\_ani a 0
- Se asigna el valor 5 a la variable frames\_maxim\_ani
- Se inicializa contador a 0
- Se inicializa current\_tiempo a 0.0
- Se asigna el valor 3 a la variable idle\_tiempo (10 segundos de tiempo de espera)

***FSMEnemigo\* EstadoEnemigoAttack::input\_handle***

Si en.lock es falso o en.get\_muerto() es verdadero, entonces:

- Retorna un nuevo EstadoEnemigoMover con el parámetro {0,0}

***void EstadoEnemigoAttack::on\_entrar***

- Se obtiene la posición del objeto "en" en el mundo y la asigna a centro
- Se inicializa el contador de frames actual de la animación en 0
- Se establece el número máximo de frames para la animación
- Se inicializa un contador en 0
- Se obtiene el tiempo actual del sistema y lo asigna a current\_tiempo

***void EstadoEnemigoAttack::on\_salir***

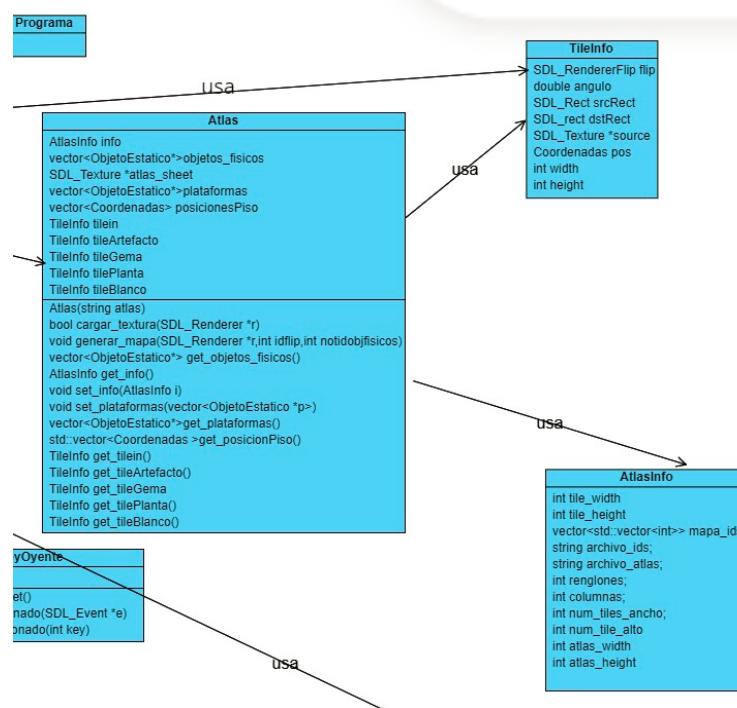
No se hace nada

***void EstadoEnemigoAttack::on\_update***

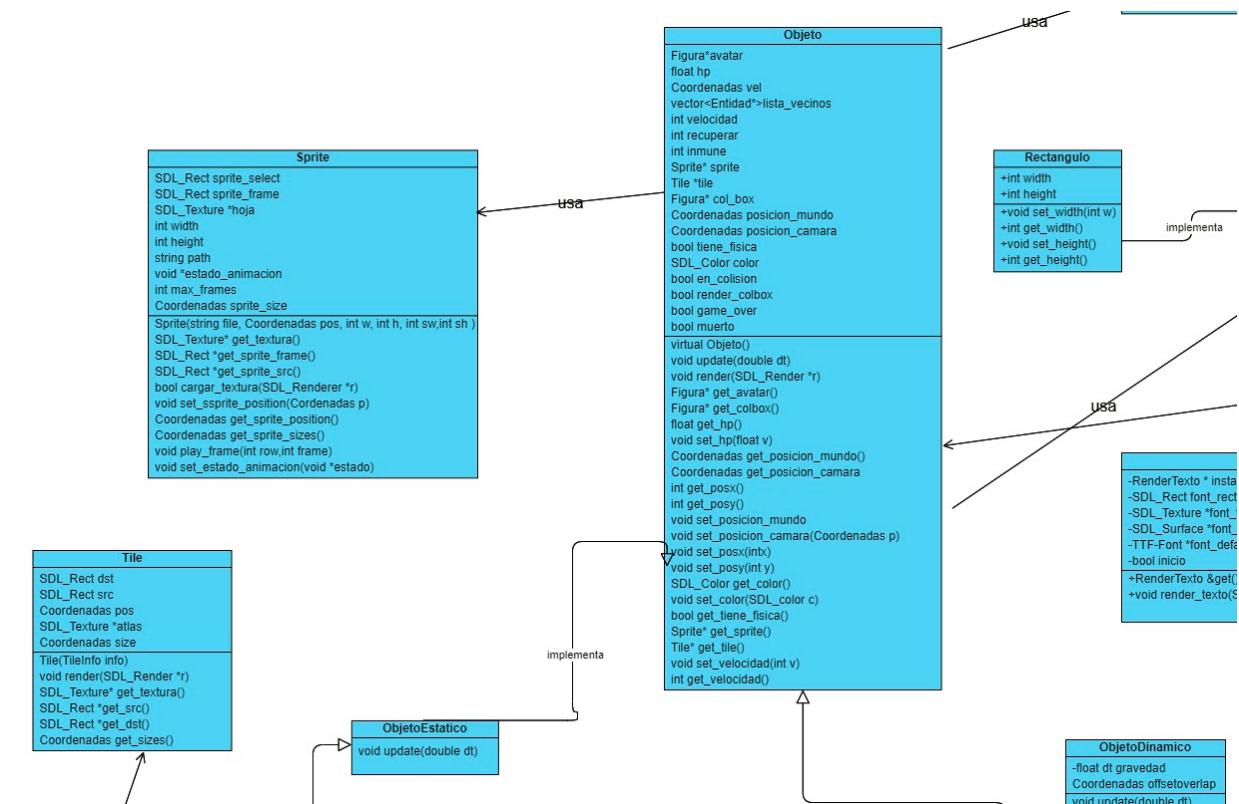
- Imprime "Enemigo en ataque".
- Calcula el tiempo transcurrido desde un momento anterior llamado current\_tiempo y guárdalo en la variable elapsed\_time.
- Si ha pasado más tiempo del especificado en idle\_time y el enemigo no está muerto, cambia su estado al EstadoEnemigoTransicion.
- Reproduce el segundo fotograma en el sprite del enemigo, utilizando el cálculo frames\_actual\_ani % frames\_maxim\_ani.
- Si frame\_dt es mayor que 5, reinícialo a 0 y aumenta el contador frames\_actual\_ani en 1.
- Incrementa el contador frame\_dt en 1.

**Clase Atlas:**

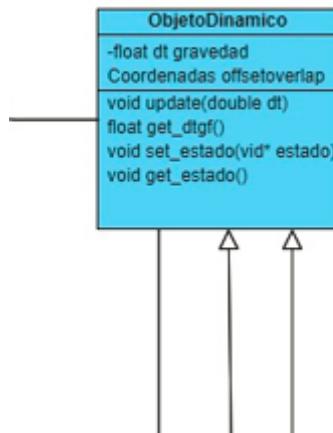
La clase atlas lee un archivo txt con la información del tamaño de cada tile para guardar toda la información recibida y crear un vector de objetos estáticos el cual se le pasa a la cámara para que esta pueda mover todo el mundo incluyendo el mapa creado. A su vez se crea un arreglo de coordenadas de aquellos objetos tipo plataforma que representan el piso del mundo. y otras variables que guardan la información de los cinco tipos de tiles en el juego (objeto TileInfo) con ayuda de constantes para poder utilizarlo en sdlapp y actualizar los tiles a que se desee (lo cual se explica en la clase Objeto estático).



## Objetos:



## Objetos dinámicos:



## Pseudocódigo

### *Atlas::Atlas*

Se asigna el valor de atlas a la propiedad archivo\_ids de la variable info.

### *Atlas::~Atlas*

Limpiar el contenido de la lista mapa\_ids de info.

### *bool Atlas::cargar\_textura*

Cargar la textura del atlas utilizando el archivo indicado en info.archivo\_atlas.

Si la carga es exitosa y la textura se ha cargado correctamente, entonces:

- Devolver verdadero, indicando que la operación fue exitosa.

Si la carga falla y la textura no se puede cargar, entonces:

- Imprimir el mensaje de error obtenido durante la carga.
- Devolver falso, indicando que la operación ha fallado.

### *void Atlas::generar\_mapa*

- Abrimos un archivo llamado "info.archivo\_ids" para leer información.
- Se verifica si el archivo se abrió correctamente. Si no se pudo abrir, mostramos un mensaje de error y salimos del programa.
- Se lee la información del archivo, que incluye el nombre de un archivo de atlas, el número de filas y columnas en el mundo, el tamaño de los tiles, el número de tiles en el atlas y el tamaño del atlas.

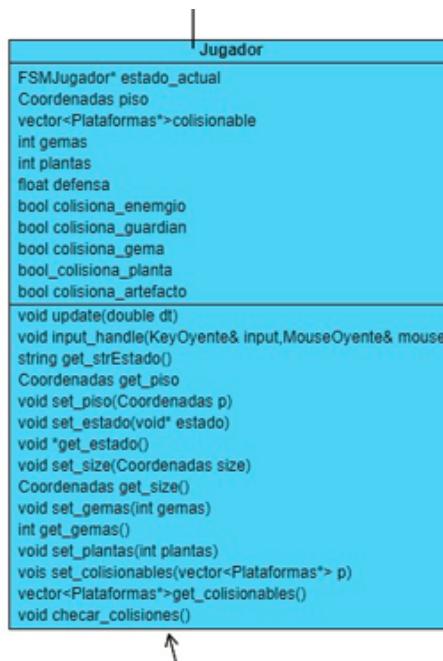


- Se crea una matriz para almacenar los IDs de los tiles, inicializándolos todos en 0.
- Se llena la matriz de IDs leyendo los valores del archivo y los guardamos en la matriz.
- Para cada valor en la matriz de IDs, se crea un objeto TileInfo que contiene información sobre el tile, como su posición, tamaño, etc.
- Dependiendo del valor del ID del tile, se asigna un tipo específico al objeto TileInfo.
- Se almacena el objeto TileInfo en un vector y lo asociamos con una plataforma física.
- Se muestra el número de objetos creados y cerramos el archivo.

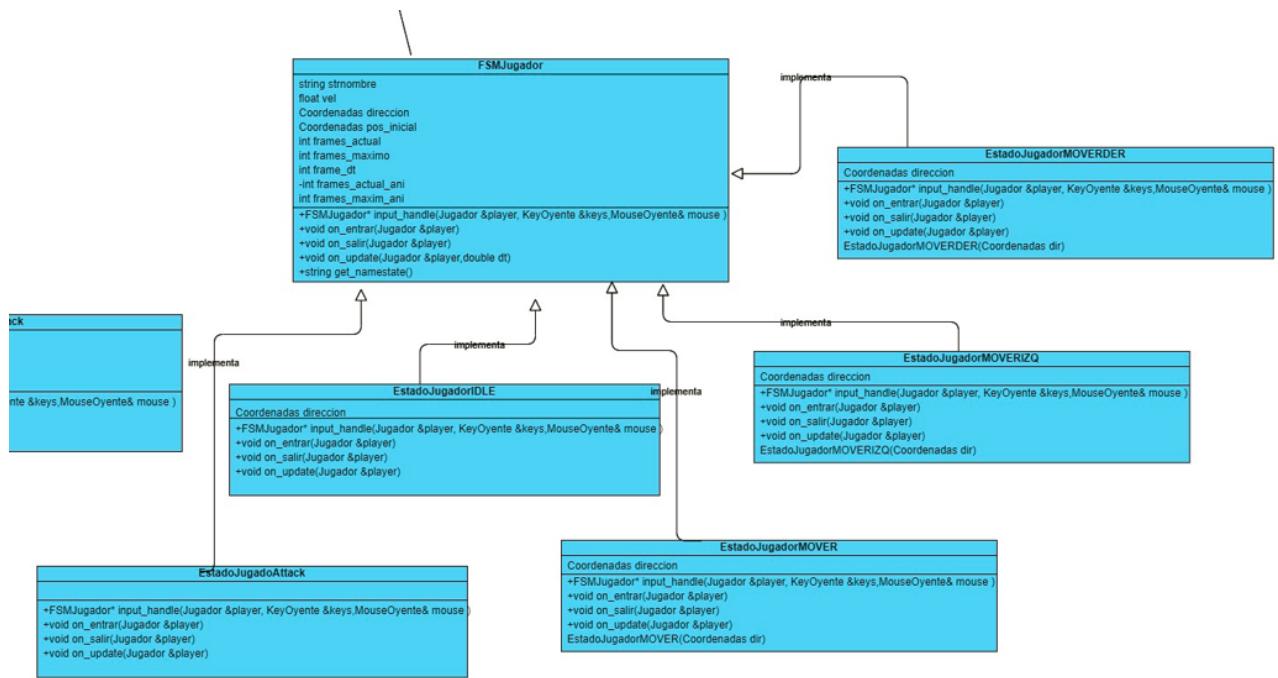
## Clase jugador

Para crear un jugador se le establecerá la ruta para llegar a su sprite, su vida, sus dimensiones tanto en la imagen como en el mapa así como un arreglo de plataformas sacado de la clase atlas cuando recién se inicia el juego para conocer cuándo es que el jugador se encuentra en un muro, lo que indicaría que hay una colisión. Así mismo, tendrá más variables booleanas para conocer si está en colisión con un muro, un monstruo, una gema, una planta y el artefacto del nivel que se estarán modificando con ayuda de la clase de motor Físico 2D.

Para esto se crearon los métodos de update e input handle, que son los que están checando constantemente si está en colisión el jugador y cambiando de estado si es necesario dependiendo de las entradas por teclado y otras variables, respectivamente. Luego está el método de *chechar colisiones* donde se checa si el jugador colisionó con una gema para sumar una gema a su inventario de gema, sucediendo lo mismo con una planta.



En la **clase FSMJugador**, tendrá los estados de idle, mover izquierda, mover derecha y atacar. En donde el jugador se moverá y se mostrará el sprite indicado dependiendo de la tecla que presiona. Y atacará si está en colisión con el monstruo y presione la tecla J, lo que restará puntos al monstruo y tendrá una planta menos en su inventario.



## Pseudocódigo

### *EstadoJugadorIDLE::EstadoJugadorIDLE*

- Se asigna el valor "IDLE" a la variable strnombre
- Se inicializa la variable frames\_actual\_ani con el valor 0
- Se asigna el valor 6 a la variable frames\_maxim\_ani

### *FSMJugador\* EstadoJugadorIDLE::input\_handle*

- Si la tecla "D" está presionada, devuelve un nuevo estado del jugador para moverse hacia la derecha.
- Si la tecla "A" está presionada, devuelve un nuevo estado del jugador para moverse hacia la izquierda.
- Si la tecla "W" está presionada, devuelve un nuevo estado del jugador para moverse hacia arriba.
- Si la tecla "S" está presionada, devuelve un nuevo estado del jugador para moverse hacia abajo.
- Si la tecla "J" está presionada y la variable attack\_on del jugador es verdadera, devuelve un nuevo estado del jugador para atacar.
- Si ninguna de las teclas anteriores está presionada, se retorna NULL, lo que indica que no hay un nuevo estado específico del jugador en este momento.

### *void EstadoJugadorIDLE::entrar*

- Se establece frames\_actual\_ani como 0
- Se establece frames\_maxim\_ani como 6



***void EstadoJugadorIDLE::salir***

- No se hace nada

***void EstadoJugadorIDLE::update***

- La línea `player.get_sprite()->play_frame(0, frames_actual_ani % frames_maxim_ani)` reproduce un fotograma específico en el sprite del jugador. El número del fotograma reproducido se determina utilizando el valor de `frames_actual_ani` y el número máximo de fotogramas permitidos (`frames_maxim_ani`).
- Después, se verifica si `frame_dt` (un contador de tiempo) es mayor que 3. Si es así, significa que ha pasado suficiente tiempo y se debe cambiar al siguiente fotograma. En ese caso, se reinicia `frame_dt` a 0 y se incrementa `frames_actual_ani` en 1.
- Finalmente, se incrementa `frame_dt` en 1 para seguir el conteo del tiempo transcurrido.

***EstadoJugadorAttack::EstadoJugadorAttack***

- Se asigna el valor "ATTACK" a la variable `strnombre`
- Se inicializa la variable `frames_actual_ani` a 0
- Se asigna el valor 6 a la variable `frames_maxim_ani`

***FSMJugador\* EstadoJugadorAttack::input\_handle***

- Si la tecla "D" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia la derecha.
- Si la tecla "A" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia la izquierda.
- Si la tecla "W" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia arriba.
- Si la tecla "S" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia abajo.
- Si la tecla "J" está presionada y el jugador está en modo de ataque, se crea y retorna un nuevo estado de ataque para el jugador.
- Si ninguna de las teclas anteriores está presionada o el jugador no está en modo de ataque, se retorna NULL, lo que indica que no se debe cambiar el estado actual del jugador.

***void EstadoJugadorAttack::entrar***

- Establecer `frames_actual_ani` como 0
- Establecer `frames_maxim_ani` como 6

***void EstadoJugadorAttack::salir***

- No se hace nada



***void EstadoJugadorAttack::update***

- El código reproduce un fotograma específico (fotograma 3) en el sprite del jugador.
- Si el contador de tiempo frame\_dt es mayor que 6, se reinicia a 0.
- Además, se incrementa el contador frames\_actual\_ani en 1 para avanzar al siguiente fotograma en la secuencia de animación.
- Por último, se incrementa en 1 el valor del contador frame\_dt para llevar un seguimiento del tiempo transcurrido.

***EstadoJugadorMOVER::EstadoJugadorMOVER***

- A la variable strnombre se le asigna el valor "MOVER".
- A la variable direccion se le asigna el valor de la variable dir.
- A la variable velocidad se le asigna el valor numérico 15.
- A la variable frames\_actual\_ani se le asigna el valor 0, indicando que es el primer frame de la animación.
- A la variable frames\_maxim\_ani se le asigna el valor 6, indicando el número máximo de frames en la animación.

***FSMJugador\* EstadoJugadorMOVER::input\_handle***

- Si el jugador está presionando las teclas "S" y "D" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERIZQ" con un parámetro {1, -1}.
- Si el jugador está presionando las teclas "S" y "A" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERDER" con un parámetro {-1, -1}.
- Si el jugador está presionando las teclas "W" y "D" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERIZQ" con un parámetro {1, 1}.
- Si el jugador está presionando las teclas "W" y "A" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERDER" con un parámetro {-1, 1}.
- Si el jugador está presionando la tecla "J" y el jugador está en modo de ataque, se retornará un nuevo estado de jugador llamado "Attack".
- Si no se ha presionado ninguna tecla o botón de entrada, entonces:
  - Retorna un nuevo objeto del tipo EstadoJugadorIDLE()

***void EstadoJugadorMOVER::entrar***

- Establecer frames\_actual\_ani como 0
- Establecer frames\_maxim\_ani como 6

***void EstadoJugadorMOVER::salir***

- No se hace nada

***void EstadoJugadorMOVER::update***

- Obtener la posición actual del jugador en el mundo.
- Si el jugador está en colisión con otro objeto:



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

- Calcular una nueva posición basada en el desplazamiento de la colisión.
- Actualizar la posición del jugador según la nueva posición calculada.
- De lo contrario, si el jugador no está en colisión:
  - Actualizar la posición del jugador según su velocidad y dirección.
- Establecer la posición actualizada del jugador en el mundo.
- Reproducir el siguiente fotograma en la animación del jugador.
- Actualizar contadores de fotogramas.
  - Sí ha transcurrido suficiente tiempo (más de 5 unidades), avanzar al siguiente fotograma en la animación.
  - Incrementar el contador de tiempo transcurrido en 1

#### ***EstadoJugadorMOVEIZQ::EstadoJugadorMOVEIZQ***

- Asignar "MOVE IZQ" a la variable strnombre
- Asignar dir a la variable direccion
- Asignar 15 a la variable velocidad
- Asignar 0 a la variable frames\_actual\_ani
- Asignar 6 a la variable frames\_maxim\_ani

#### ***FSMJugador\* EstadoJugadorMOVEIZQ::input\_handle***

- Si la tecla "D" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia la derecha.
- Si la tecla "A" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia la izquierda.
- Si la tecla "W" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia arriba.
- Si la tecla "S" está presionada, se crea y retorna un nuevo estado para mover al jugador hacia abajo.
- Si la tecla "J" está presionada y el jugador está en modo de ataque, se crea y retorna un nuevo estado de ataque para el jugador.
- Si ninguna de las teclas anteriores está presionada o el jugador no está en modo de ataque, se retorna NULL, lo que indica que no se debe cambiar el estado actual del jugador.

#### ***void EstadoJugadorMOVEIZQ::entrar***

- Establecer frames\_actual\_ani como 0
- Establecer frames\_maxim\_ani como 6

#### ***void EstadoJugadorMOVEIZQ::salir***

- No se hace nada

#### ***void EstadoJugadorMOVEIZQ::update***

- Obtener la posición actual del jugador en el mundo.



- Si el jugador está en colisión con otro objeto:
  - Calcular una nueva posición basada en el desplazamiento de la colisión.
  - Actualizar la posición del jugador según la nueva posición calculada.
- De lo contrario, si el jugador no está en colisión:
  - Actualizar la posición del jugador según su velocidad y dirección.
- Establecer la posición actualizada del jugador en el mundo.
- Reproducir el siguiente fotograma en la animación del jugador.
- Actualizar contadores de fotogramas.
  - Sí ha transcurrido suficiente tiempo (más de 5 unidades), avanzar al siguiente fotograma en la animación.
  - Incrementar el contador de tiempo transcurrido en 1

#### ***EstadoJugadorMOVERDER::EstadoJugadorMOVERDER***

- Asignar "MOVER DER" a la variable strnombre
- Asignar dir a la variable direccion
- Asignar 15 a la variable velocidad
- Asignar 0 a la variable frames\_actual\_ani
- Asignar 6 a la variable frames\_maxim\_ani

#### ***FSMJugador\* EstadoJugadorMOVERDER::input\_handle***

- Si el jugador está presionando las teclas "S" y "D" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERIZQ" con un parámetro {1, -1}.
- Si el jugador está presionando las teclas "S" y "A" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERDER" con un parámetro {-1, -1}.
- Si el jugador está presionando las teclas "W" y "D" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERIZQ" con un parámetro {1, 1}.
- Si el jugador está presionando las teclas "W" y "A" al mismo tiempo, se retornará un nuevo estado de jugador llamado "MOVERDER" con un parámetro {-1, 1}.
- Si el jugador está presionando la tecla "J" y el jugador está en modo de ataque, se retornará un nuevo estado de jugador llamado "Attack".
- Si no se ha presionado ninguna tecla o botón de entrada, entonces:
  - Retorna un nuevo objeto del tipo EstadoJugadorIDLE()

#### ***void EstadoJugadorMOVERDER::entrar***

- Establecer frames\_actual\_ani como 0
- Establecer frames\_maxim\_ani como 6

#### ***void EstadoJugadorMOVERDER::salir***

- No se hace nada

#### ***void EstadoJugadorMOVERDER::update***

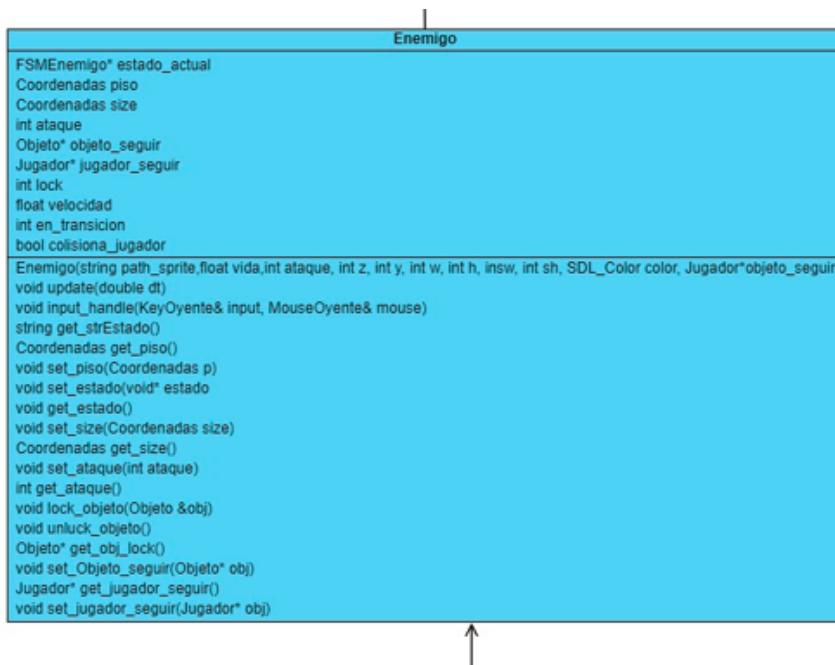
- Obtener la posición actual del jugador en el mundo.



- Si el jugador está en colisión con otro objeto:
  - Calcular una nueva posición basada en el desplazamiento de la colisión.
  - Actualizar la posición del jugador según la nueva posición calculada.
- De lo contrario, si el jugador no está en colisión:
  - Actualizar la posición del jugador según su velocidad y dirección.
- Establecer la posición actualizada del jugador en el mundo.
- Reproducir el siguiente fotograma en la animación del jugador.
- Actualizar contadores de fotogramas.
  - Sí ha transcurrido suficiente tiempo (más de 5 unidades), avanzar al siguiente fotograma en la animación.
  - Incrementar el contador de tiempo transcurrido en 1

## Clase enemigo

Esta clase es similar a la clase jugador ya que se crea al igual que uno pero no le afecta si colisiona con una gema, planta ni el guardián. Se encarga de seguir al jugador y restarle puntos a este cuando colisionan. El objeto enemigo termina siguiendo al jugador mientras el enemigo continúa teniendo puntos vida mayor a cero.



Las clases estado **FSMEnemigo** cuenta con 3 subclases o estados principales, el enemigo inicia principalmente en el estado mover donde se queda en su posición inicial mientras el jugador inicia el juego. EN cuanto inicia el juego el enemigo para al estado transición cuando este llega a la posición del jugador, pasa al estado atacar. Se queda en este estado por 3 segundos para darle oportunidad de escapar al jugador. También se aprovecha y se utilizan los estados para cambiar los frames a leer del sprite sheet dependiendo de su estado como para



```
Guardian
Coordenadas piso
Coordenadas size
int ataque
Objeto* objeto_seguir
int first_time
int lock
float velocidad
int en_transicion
FSMGuardian* estado_actual
bool colision_jugador
Enemigo(string path_sprite,float vida,int ataque, int z, int y, int w, int h, insw, int sh, SDL_Color color, Jugador*objeto_seguir)
void update(double dt)
void input_handle(KeyOyente& input, MouseOyente& mouse)
string get_strEstado()
Coordenadas get_piso()
void set_piso(Coordenadas p)
void set_estado(void* estado)
void get_estado()
void set_size(Coordenadas size)
Coordenadas get_size()
void set_ataque(int ataque)
int get_ataque()
void lock_objeto(Objeto &obj)
void unlock_objeto()
Objeto* get_obj_lock()
void set_Objeto_seguir(Objeto* obj)
void set_first_time(int first_time)
int get_first_time()
```

## Pseudocódigo

### *EstadoEnemigoMover::EstadoEnemigoMover*

- La variable frames\_actual\_ani representa el número actual de fotogramas en una animación y se inicia en 0.
- La variable frames\_maxim\_ani representa el número máximo de fotogramas permitidos en la animación y se establece en 5.
- La variable direccion almacena una dirección específica y su valor se asigna a dir.
- La variable strestado indica el estado de movimiento y se establece en "mover".
- La variable contador es un contador y se inicia en 0.
- La variable current\_tiempo guarda un valor de tiempo y se inicia en 0.0.
- La variable idle\_tiempo representa el tiempo de espera en segundos y se establece en 5, que equivaldría a 10 segundos.

### *FSMEnemigo\* EstadoEnemigoMover::input\_handle*

Si (en.lock == 1 y en.start es verdadero y en.get\_muerto() es falso) entonces

- en.set\_estado(new EstadoEnemigoTransicion())

De lo contrario

- retornar new EstadoEnemigoMover({0,0})

### *void EstadoEnemigoMover::on\_entrar*

- La variable vel guarda el valor de la velocidad del objeto en.
- La variable frames\_actual\_ani se establece en 0, lo cual indica que no hay ningún fotograma actual de animación.



- La variable frames\_maxim\_ani se establece en 5, indicando que el número máximo de fotogramas de animación permitidos es 5.
- La variable contador se inicializa en 0.
- La variable current\_tiempo guarda el valor del tiempo actual obtenido a través de una función llamada Tiempo::get\_tiempo().

### ***void EstadoEnemigoMover::on\_salir***

- No se hace nada.

### ***void EstadoEnemigoMover::on\_update***

Primero se verifican tres condiciones:

- El candado del objeto "en" está activado (valor igual a 1).
- El objeto "en" ha sido iniciado.
- El objeto "en" no está muerto.

Si se cumplen estas condiciones, se realiza lo siguiente:

- Se calcula el tiempo transcurrido ("elapsed\_time") restando el tiempo actual ("Tiempo::get\_tiempo()") al tiempo almacenado anteriormente ("current\_tiempo").
- Si el tiempo transcurrido es mayor que el tiempo de espera establecido ("idle\_time") y el objeto "en" no está muerto, entonces se cambia su estado a transición ("EstadoEnemigoTransicion()").

Luego se ejecutan las siguientes acciones, independientemente de si se cumplieron las condiciones anteriores:

- Se reproduce un fotograma específico del sprite del objeto "en", utilizando el método "play\_frame()" con los parámetros 0 y el resultado de la operación "frames\_actual\_ani % frames\_maxim\_ani".
- Se verifica si el valor de "frame\_dt" es mayor que 5. Si es así, se reinicia a 0 y se incrementa en uno el valor de "frames\_actual\_ani".
- Finalmente, se incrementa en 1 el valor de "frame\_dt".

### ***EstadoEnemigoTransicion::EstadoEnemigoTransicion***

- check es un arreglo o conjunto de dos valores, ambos inicializados en cero.
- frames\_actual es una variable que indica el número actual de cuadros en una secuencia y se inicia en cero.
- frames\_maximo es una variable que define el número máximo permitido de cuadros en una secuencia y se establece en 100.
- frames\_actual\_ani es una variable que representa el número actual de cuadros en una animación y se inicia en cero.



- frames\_maxim\_ani es una variable que define el número máximo permitido de cuadros en la animación y se establece en 5.
- strestado es una variable de texto que almacena el estado actual y se establece como "TRANSICION".

### ***FSMEnemigo\* EstadoEnemigoTransicion::input\_handle***

- Si el candado del objeto en no está activado, se devuelve un nuevo objeto del tipo EstadoEnemigoMover con los valores {0,0}. Esto significa que el enemigo se mantendrá en un estado de movimiento con una posición de (0,0).
- En caso contrario, si el candado está activado, se devuelve un valor nulo. Esto indica que no se realizará ninguna acción.

### ***void EstadoEnemigoTransicion::on\_entrar***

- Se asigna el valor de 1 a la variable en.en\_transicion.
- La variable vel se establece con el valor de en.velocidad.
- Se inicia la variable frames\_actual\_ani con el valor de 0.
- La variable frames\_maxim\_ani se inicializa en 5.
- Se calcula el valor de lim restando la coordenada x del vértice 3 del objeto en.get\_obj\_lock().get\_avatar() menos la coordenada x del vértice 0 del mismo objeto.
- Si el valor de distancia es menor que lim, entonces se asigna el valor de 10 a la variable frames\_maximo.

### ***void EstadoEnemigoTransicion::on\_salir***

- Se inicia a la variable en.en\_transicion el valor de 0.

### ***void EstadoEnemigoTransicion::on\_update***

- Si el número de cuadros actuales ("frames\_actual") es mayor que el número máximo de cuadros permitidos ("frames\_maximo"), entonces no se hace nada y se finaliza.
- Se obtiene la posición inicial del objeto "en" en el mundo y se guarda en la variable "pos\_inicial" con los valores (0,0).
- Se obtiene la posición final del objeto "en" a través de otro objeto al que está "bloqueado" ("en.get\_obj\_lock()") y se guarda en la variable "pos\_final".
- Se calcula un valor "t" dividiendo "frames\_actual" entre "frames\_maximo".
- Se realiza una interpolación lineal entre "pos\_inicial" y "pos\_final" utilizando el valor "t" y se guarda en la variable "check".
- Se calculan las diferencias entre las coordenadas de "check" y las coordenadas previas guardadas en "ant\_check".
- Se obtiene la posición actual del objeto al que está "bloqueado" ("en.get\_obj\_lock()") y se guarda en "objpos".



- Se calcula la distancia entre la posición del objeto "centro" y la posición de "objpos" utilizando la fórmula de la distancia euclidiana.
- Imprime en pantalla las coordenadas "Check: x,y" y "Pos final: x,y".
- Establece la posición del objeto "en" en el mundo como las coordenadas "check".
- Si las coordenadas "check" son iguales a las coordenadas "pos\_final" y el objeto "en" no está muerto, cambia su estado a "EstadoEnemigoAttack" con un objeto de bloqueo específico.
- Incrementa el contador de frames actuales.
- Guarda las coordenadas anteriores en la variable "ant\_check".
- Reproduce el frame de la animación del objeto "en" correspondiente al índice "frames\_actual\_ani % frames\_maxim\_ani".
- Si el contador "frame\_dt" es mayor a 5, lo reinicia a 0 y aumenta el contador de frames de animación "frames\_actual\_ani".
- Incrementa el contador "frame\_dt".

### ***EstadoEnemigoAttack::EstadoEnemigoAttack***

- La variable strestado se establece como "ATTACK", lo que indica que el estado actual es de ataque.
- La variable obj se asigna con la dirección de memoria de objlock, lo que permite acceder a ese objeto.
- La variable frames\_actual\_ani se inicializa en 0, lo que indica que no hay fotogramas de animación actuales.
- La variable frames\_maxim\_ani se establece en 5, indicando el número máximo de fotogramas permitidos en la animación.
- La variable contador se inicializa en 0.
- La variable current\_tiempo se inicia en 0.0, lo que indica que no ha pasado ningún tiempo.
- La variable idle\_tiempo se establece en 3, lo que representa un tiempo de espera de 3 segundos.

### ***FSMEnemigo\* EstadoEnemigoAttack::input\_handle***

- Si el candado del objeto en no está activado, entonces se devuelve un nuevo objeto con el estado de movimiento (EstadoEnemigoMover({0,0})).
- En caso contrario, se devuelve un valor nulo.

### ***void EstadoEnemigoAttack::on\_entrar***

- La variable frames\_actual\_ani se establece en 0.
- La variable frames\_maxim\_ani se establece en 5.
- La variable contador se establece en 0.
- La variable current\_tiempo recibe el valor de la función Tiempo::get\_tiempo().



### ***void EstadoEnemigoAttack::on\_salir***

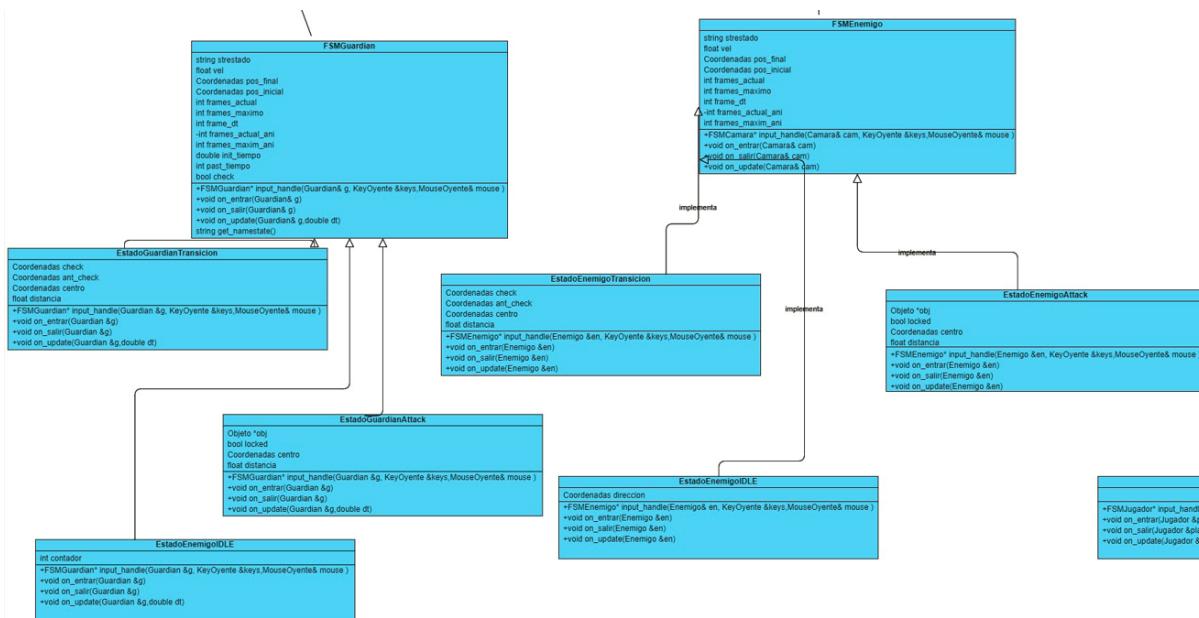
- No se hace nada.

### ***void EstadoEnemigoAttack::on\_update***

- Se imprime la frase "Enemigo en ataque" utilizando la función imprimir (o su equivalente en el lenguaje de programación que estés utilizando).
- Se calcula la diferencia entre el tiempo actual (Tiempo::get\_tiempo()) y current\_tiempo, y se guarda en la variable elapsed\_time.
- Si elapsed\_time es mayor que idle\_time y el enemigo no está muerto (en.get\_muerto() es falso), entonces se ejecuta la acción de cambiar el estado del enemigo a EstadoEnemigoTransicion() utilizando la función en.set\_estado().
- Se reproduce el fotograma número 2 del sprite del enemigo utilizando la función play\_frame() de su sprite. El parámetro frames\_actual\_ani % frames\_maxim\_ani se utiliza para obtener el número de fotograma dentro del rango permitido.
- Si el valor de frame\_dt es mayor que 5, se reinicia a 0 y se incrementa en 1 el valor de frames\_actual\_ani.
- Se incrementa en 1 el valor de frame\_dt.

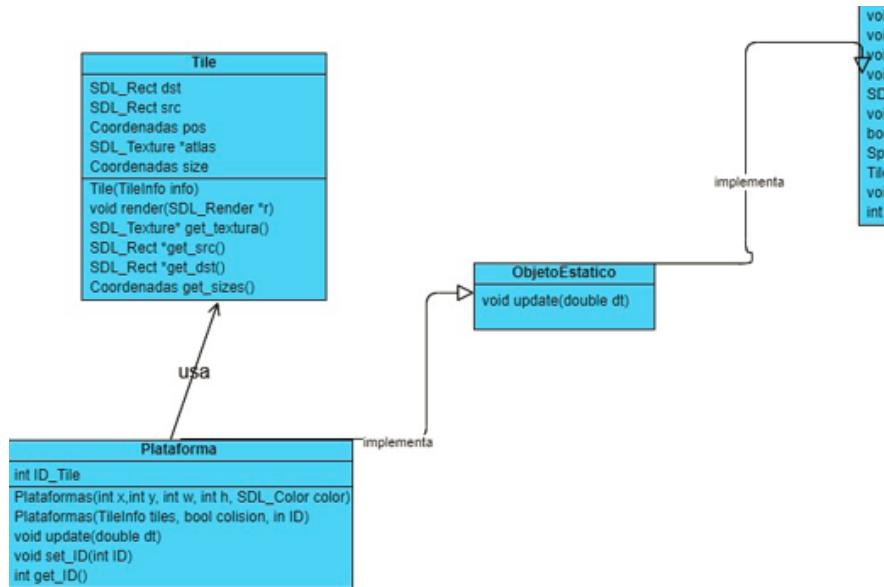
### **Clase guardián:**

La clase guardian comienza en estadoIDLE y solo comienza a estar en estadoTransicion y estadoAtacar (es decir comienza a moverse a la posición del jugador) cuando colisionó por primera vez con el jugador. Para ello hay una variable que actúa como bool para saber si es la primera vez que colisionan, ya que si no han colisionado 1 vez, se mantendrá el guardián en estadoIDLE. Después de eso se hará visible el artefacto creado en SDLApp, el guardián actuará como enemigo y lo atacará en intervalos de tiempos aleatorios de 5-15 segundos. Esto se repetirá hasta que el jugador encuentre el artefacto.



*FSMEnemigo y FSMGuardian*

## Clase Objeto Estático:



## Pseudocódigo

### *EstadoGuardianIDLE::EstadoGuardianIDLE*

- La variable `frames_actual_ani` se inicializa en 0, lo cual indica el número actual de fotogramas en una animación.
- La variable `frames_maxim_ani` se establece en 3, que representa el número máximo de fotogramas permitidos en la animación.



- El contador contador se inicia en 0.
- La variable current\_tiempo se inicia en 0.0 y se utiliza para realizar un seguimiento del tiempo actual.
- La variable idle\_tiempo se establece en un número aleatorio entre 5 y 15. Este valor indica el tiempo de espera antes de realizar alguna acción. Por ejemplo, podría ser el tiempo que el personaje permanece inactivo antes de moverse.
- La variable strestado se establece en "IDLE", lo cual indica que el estado actual del objeto es de inactividad o espera.

#### ***FSMGuardian\* EstadoGuardianIDLE::input\_handle***

- Se debe retornar el valor “NULL”.

#### ***void EstadoGuardianIDLE::on\_entrar***

- La variable frames\_actual\_ani se inicializa con el valor 0, lo que indica que no hay ningún fotograma actualmente en la animación.
- La variable frames\_maxim\_ani se establece en 3, lo que indica el número máximo de fotogramas permitidos en la animación.
- El contador (contador) se inicia en 0, lo que sugiere que no se ha realizado ninguna cuenta o recuento hasta el momento.
- La variable current\_tiempo se asigna utilizando una función o método llamado Tiempo::get\_tiempo(), que obtiene el tiempo actual. Esto nos proporciona el valor actual del tiempo.

#### ***void EstadoGuardianIDLE::on\_salir***

- No se hace nada.

#### ***void EstadoGuardianIDLE::on\_update***

- Se incrementa el contador en 1.
- Se calcula la diferencia de tiempo entre el tiempo actual (Tiempo::get\_tiempo()) y el tiempo guardado en current\_tiempo, y se almacena en la variable elapsed\_tiempo.
- Si elapsed\_tiempo es mayor que idle\_tiempo y el valor de g.get\_first\_time() es igual a 1, se ejecuta la acción de establecer el estado de g como EstadoGuardianTransicion().
- Se reproduce el fotograma 0 del sprite de g utilizando los valores 0 y (frames\_actual\_ani modulo frames\_maxim\_ani).
- Si el valor de frame\_dt es mayor que 5, se reinicia a 0 y se incrementa en 1 el valor de frames\_actual\_ani.
- Se incrementa en 1 el valor de frame\_dt.
- Se incrementa el contador en 1 nuevamente.



### ***EstadoGuardianTransicion::EstadoGuardianTransicion***

- La variable check tiene un valor de {0, 0}.
- La variable ant\_check tiene un valor de {0, 0}.
- La variable centro tiene un valor de {0, 0}.
- La variable distancia tiene un valor de 10.0.
- La variable frames\_actual tiene un valor de 0.
- La variable frames\_maximo tiene un valor de 45.
- La variable frames\_actual\_ani tiene un valor de 0.
- La variable frames\_maxim\_ani tiene un valor de 8.
- La variable strestado tiene un valor de "TRANSICION".

### ***FSMGuardian\* EstadoGuardianTransicion::input\_handle***

- Se debe retornar el valor "NULL".

### ***void EstadoGuardianTransicion::on\_entrar***

- Se obtiene un objeto llamado "objeto\_seguir" mediante la función "g.get\_obj\_lock()".
- Si el objeto "objeto\_seguir" existe (no es nulo), se realiza lo siguiente:
  - Se obtiene la posición en el mundo del objeto "objeto\_seguir" y se almacena en la variable "centro".
  - Se calcula la diferencia en coordenadas (en el eje X y en el eje Y) entre la posición del "centro" y la posición en el mundo de "g" (presumiblemente otro objeto). Estas diferencias se almacenan en las variables "diferenciaX" y "diferenciaY", respectivamente.
  - Se calcula la distancia entre el "centro" y la posición en el mundo de "g" utilizando la fórmula de la distancia euclídea (raíz cuadrada de la suma de los cuadrados de las diferencias en coordenadas). El resultado se guarda en la variable "distancia".
- Se establece el valor de "frames\_actual\_ani" como 0.
- Se establece el valor de "frames\_maxim\_ani" como 8.

### ***void EstadoGuardianTransicion::on\_salir***

- Se inicia a la variable g.en\_transicion el valor de 0.

### ***void EstadoGuardianTransicion::on\_update***



- La variable objetivo\_x almacena la coordenada x del objetivo obtenida a partir del método get\_posicion\_mundo() del objeto g.get\_obj\_lock().
- La variable objetivo\_y almacena la coordenada y del objetivo obtenida a partir del método get\_posicion\_mundo() del objeto g.get\_obj\_lock().
- La variable direccion\_x se calcula como la diferencia entre objetivo\_x y la coordenada x de la posición actual obtenida a partir del método get\_posicion\_mundo() del objeto g.
- La variable direccion\_y se calcula como la diferencia entre objetivo\_y y la coordenada y de la posición actual obtenida a partir del método get\_posicion\_mundo() del objeto g.
- Se calcula la distancia al objetivo utilizando la fórmula de la distancia euclíadiana entre dos puntos en un plano.
- Se calcula el máximo desplazamiento permitido (max\_desplazamiento) multiplicando la velocidad del objeto g por dt.
- Se calculan los desplazamientos en los ejes X e Y (desplazamiento\_x y desplazamiento\_y) multiplicando la dirección correspondiente por el mínimo entre la distancia y el máximo desplazamiento permitido.
- Se actualiza la posición del objeto g sumando los desplazamientos calculados a su posición actual en el mundo.
- Se verifica si se ha alcanzado la posición objetivo, si la distancia es menor o igual a 10. En caso afirmativo, se cambia el estado del objeto g al estado de ataque (EstadoGuardianAttack). En caso contrario, se cambia al estado de transición (EstadoGuardianTransicion).
- Se reproduce un fotograma específico en el sprite del objeto g utilizando los valores 2 y frames\_actual\_ani % frames\_maxim\_ani.
- Se actualiza el contador frame\_dt. Si su valor es mayor a 5, se reinicia a 0 y se incrementa frames\_actual\_ani en 1.

### ***EstadoGuardianAttack::EstadoGuardianAttack***

- La variable centro se inicializa con un par de coordenadas (0, 0), lo que representa el centro de algo.
- La variable strestado se establece en "ATTACK", indicando un estado de ataque.
- La variable obj guarda la dirección de memoria del objeto objlock, lo que permite acceder y manipular dicho objeto.
- La variable frames\_actual\_ani se inicializa en 0, representando el número actual de fotogramas en una animación.
- La variable frames\_maxim\_ani se establece en 8, indicando el número máximo de fotogramas permitidos en la animación.

### ***FSMGuardian\* EstadoGuardianAttack::input\_handle***



- Se debe retornar el valor “NULL”.

### ***EstadoGuardianAttack::EstadoGuardianAttack***

Si el objeto obj existe, se realiza lo siguiente:

- Se guarda la posición del objeto obj en la variable centro.
- Se activa el bloqueo (lock) en la variable g.
- Se establece el número actual de fotogramas (frames\_actual\_ani) como 0.
- Se establece el número máximo de fotogramas permitidos (frames\_maxim\_ani) como 8.

### ***void EstadoGuardianAttack::on\_salir***

- No se hace nada.

### ***void EstadoGuardianAttack::on\_update***

Si tanto el objeto "obj" como el candado "g.lock" están activados, se realiza lo siguiente:

- Se reduce la vida del objeto "obj" restando la cantidad de ataque del objeto "g". Por ejemplo, si la vida actual del objeto "obj" es de 100 y el ataque del objeto "g" es de 20, la vida del objeto "obj" se actualizará a 80.
- Luego, se verifica si la vida del objeto "obj" ha llegado a cero o menos. Si es así, se muestra un mensaje indicando que el objeto ha sido derrotado.
- Después de eso, se cambia el estado del objeto "g" al estado IDLE ("EstadoGuardianIDLE()").

Se reproduce el primer fotograma del sprite en el objeto g utilizando una lógica basada en el cálculo  $\text{frames\_actual\_ani} \% \text{frames\_maxim\_ani}$ .

Si el valor de frame\_dt es mayor que 5, se realizan las siguientes acciones:

- Se reinicia el contador frame\_dt a 0.
- Se incrementa el número de fotograma actual (frames\_actual\_ani).
- Si el valor de frame\_dt no es mayor que 5, se establece la variable g.lock en 0.

Se incrementa el contador frame\_dt en 1.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez

Hay principalmente un objeto estático utilizado en el juego: las plataformas. Se crean plataformas en la clase atlas para la creación del mapa y para identificar qué plataformas representan una gema, planta y el artefacto. Estas últimas 3 actúan de la misma forma pero se realizan acciones distintas dependiendo con cual colisione el jugador, por lo que la clase Plataformas contiene un atributo llamado ID, que identifica qué es lo que representa la plataforma. Estas IDs se establecen en la clase Atlas mediante constantes: GEMAS 1, PLANTAS 2, ARTEFACTO 3, ATTACK\_ZONE 4, PISO 5, MURO 6.

## 8-Capturas del DEMO Final



Imagen 1. Mensaje “Press L To Start”. Al momento de correr el juego, le pedirá que presione L para iniciar el juego, esto hará que la cámara y el enemigo comience a seguir al jugador y se muestran el estado de los atributos relevantes de los objetos.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez



Imagen 2: Mensaje “Attack Now”. Se puede ver como le indica al jugador que ya es posible atacar al enemigo con las plantas. Se le restara plantas al jugador y puntos vida al Enemigo cuando el jugador Presione J.



Imagen 3. Mensaje “Game Over” . El jugador ya no tiene puntos de vida por lo que se acaba el juego.



Maribel Itzel Méndez Ascencio  
Milka Yamil Trinidad Gutiérrez



Imagen 5. Mensaje “You Win” . El jugador encuentra el artefacto y entra en contacto con él, por lo que termina el nivel y gana.