

CIIE

Bermo Taibo, Alejandra. D'angelo Sabín, Luca.
Fandiño Dovalo, David. García Fernández, Alejandro.

February 2023

Índice

1. Desarrollo artístico	3
1.1. Antecedentes	3
1.1.1. Ambientación	3
1.1.2. Historia	3
1.2. Personajes	5
1.2.1. Historia, motivaciones, bocetos, modelos en 3D, etc.	5
1.3. Otras características de la ambientación	7
1.3.1. Objetos destacados	7
1.3.2. Lugares	7
1.4. Guion	7
1.4.1. Desarrollo y transcurso de la acción en el videojuego.	7
1.4.2. Desarrollo total del juego.	8
2. Desarrollo técnico	9
2.1. Videojuego en 2D	9
2.1.1. Descripción	9
2.1.1.1. Personajes	9
2.1.1.2. Enemigos	10
2.1.1.3. Objetos	12
2.1.1.4. Diseño	13
2.1.2. Metodología de desarrollo	17
2.1.3. Escenas	18
2.1.3.1. Escena 1: Menú Inicial	26
2.1.3.2. Escena 2: Menú opciones	26
2.1.3.3. Escena 3: Nivel 1	26
2.1.3.4. Escena 4: Nivel 2	27
2.1.3.5. Escena 5: Nivel 3	27
2.1.4. Detalles de implementación	28
2.1.4.1. Persistencia de datos entre niveles	28
2.1.4.2. Sprint del jugador	29
2.1.4.3. Comportamiento de los enemigos	30
2.1.5. Aspectos destacables	31
2.1.5.1. Generación procedural	31
2.1.5.2. Mecánica de regreso en el tiempo	32
2.1.5.3. Abstracción de los controles de pygame	35
2.1.5.4. Diseño de una estructura CSV para representar información de entidades	35
2.1.6. Manual de usuario	37
2.1.6.1. Menú principal	37
2.1.6.2. Menú de configuración	38
2.1.6.3. Interfaz de usuario	39
2.1.6.4. Controles	40
2.1.7. Reporte de bugs	41

1. Desarrollo artístico

1.1. Antecedentes

1.1.1. Ambientación

El juego se ambienta en un lugar lejano de la civilización, donde reinan la paz y la tranquilidad hasta el fatídico momento en el cual comienza el juego. Se ha tomado cierta inspiración de la mitología griega para desarrollar una de las principales mecánicas del juego y ciertos personajes importantes.

1.1.2. Historia

En los comienzos del universo, la personificación del tiempo, Chronos, era capaz de utilizar su reloj para controlar el tiempo a su antojo. Sin embargo, tal cantidad de poder corrompió la mente de este Dios, que ante una caliente disputa aniquiló a todo su parentesco. Lleno de tristeza y remordimiento, Chronos comprendió el peligro de albergar semejante poder en una única entidad. Así, tomó la decisión de fracturar el reloj en 7 partes esparcidas en distintos puntos del universo, para así evitar otra tragedia.

Pasaron los años y los rumores se formaron, eran muchos los que anhelaban semejante poder. Y, en consecuencia, muchos emprendieron una búsqueda con un propósito oscuro.

Uno de ellos, el dios de la codicia, Mammón, había decidido conseguir dicho poder sin importar los medios utilizados para ello.

Las 7 partes se dividieron en 3 planetas separados por todo el universo. Tres partes cayeron en un planeta de la Galaxia Andrómeda, otras 3 partes cayeron en un planeta de la Galaxia Circinus, y la última parte, el núcleo del reloj, cayó en la Vía Láctea, concretamente en el planeta Tierra.

En cada punto de los planetas mencionados se forma un templo con las correspondientes partes del reloj. En él se hallan unas inscripciones, narrando la historia y los peligros de unificar el reloj para obtener el poder del tiempo. Dichos templos se hallan protegidos por criaturas varias y diferentes puzzles.

Sin embargo, el templo de la tierra carecía de protecciones. El personaje principal, la abuela, una exploradora interplanetaria ya jubilada, encontró dicho templo cerca de uno de los montes donde vivía. Su curiosidad nata e interés por las antiguas culturas y artefactos la impulsó a investigar el templo. Al descubrir la historia del reloj de Chronos, ella decidió tomar la pieza más importante que yacía en el templo, el núcleo del reloj. Dedujo que dada la falta de vigilancia, la mejor opción para evitar que cayese en las manos equivocadas era asumir la responsabilidad de protegerlo.

Casualmente, Mammón aparece en la Tierra para comenzar su búsqueda con la pieza principal. Es la única pieza que por sí sola permite un control sobre el tiempo, pero es muy limitado por la falta de sus partes adicionales.

Mammón descubrió que la abuela poseía el reloj, pero sabía que, aún conteniendo solo una fracción de su poder, el artefacto era muy peligroso, entonces formuló un plan para hacerse con él. Provocó un accidente haciendo que el abuelo se cayese de un árbol, causando su muerte, con las esperanzas de que la abuela usara el reloj para retroceder en el tiempo y salvarlo. Esto haría que el reloj agotase su poder, haciéndolo vulnerable momentáneamente.

Sin embargo, la abuela comprendió la trampa y, muy a su pesar, permitió la muerte de su esposo. Al ver que su plan había fallado, Mammón decidió recolectar primero las demás partes del reloj antes de intentar hacerse con el núcleo. La abuela, después de este incidente, decidió que tal poder no debía estar en manos de nadie y comenzó su propia búsqueda para reunir las demás partes del reloj antes que Mammón.

1.2. Personajes

1.2.1. Historia, motivaciones, bocetos, modelos en 3D, etc.



Figura 1: Paleta de colores principal para el planeta Tierra

- Gato Floripondio: gato negro de La abuela. Le gusta estar en casa y no hacer nada.



Figura 2: Sprite correspondiente a la posición estática del gato Floripondio.

- La abuela: valiente y experimentada exploradora intergaláctica, jubilada en una humilde cabaña en medio del bosque con su esposo El abuelo. Se trata del personaje principal de esta historia, su objetivo es poder recuperar todos los fragmentos del reloj de Chronos para poder salvar a su esposo. Para ello hará uso de sus habilidades y destrezas aprendidas en su carrera de exploración.

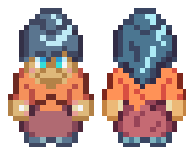


Figura 3: Sprite correspondiente a la posición estática de la abuela.

- El abuelo: un humilde hombre de campo, creció trabajando desde los 13 años. Conoció a La abuela cuando eran jóvenes en la frutería de sus padres en la que el trabajaba y desde entonces han tenido una conexión mágica.

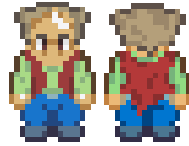


Figura 4: Sprite correspondiente a la posición estática del abuelo.

- Mammón: Un rey demonio del planeta Nibiru, perteneciente a la constelación de Aquila, que busca el poder absoluto para doblegar a todos los seres de la galaxia.
- Chronos: conocido como Dios del tiempo, nació de Hydros y Thesis para darle estabilidad al universo. Protagoniza un evento fatídico y desencadenante en la historia.

1.3. Otras características de la ambientación

1.3.1. Objetos destacados

- Cada nivel es generado proceduralmente y único cada vez que se juega.
- El Reloj de Chronos: el reloj originariamente del Dios del tiempo, fracturado en 7 partes y codiciado por todos, alberga gran parte de su poder en el núcleo el cual es el único que permite retroceder en el tiempo. Una vez reconstruido otorga un poder incommensurable.
- El bastón galáctico de la abuela: puede parecer un simple bastón de anciano pero realmente está altamente equipado y le ha acompañado en cada una de sus aventuras. Posee múltiples funcionalidades, incluso la retropropulsión.
- La zapatilla de la abuela: considerada inofensiva por algunos ignorantes, esta herramienta es una de las armas a distancia más letales de todo su repertorio.

1.3.2. Lugares

- Planeta Tierra:
 - Portal mágico.
 - Estructuras varias.
- Planeta Tatooine:
 - Portal mágico.
 - Estructuras varias.
- Planeta Enceladus:
 - Portal mágico.
 - Estructuras varias.

1.4. Guion

1.4.1. Desarrollo y transcurso de la acción en el videojuego.

La abuela, equipada con el núcleo del reloj y sus artefactos, sigue pistas para encontrar las piezas restantes. Utiliza extraños portales que conectan los templos en distintas galaxias para llegar a los planetas que tienen los fragmentos. En esos planetas diversos enemigos ya habían puesto sus manos en las piezas,

por lo que tendrá que recuperarlos a la fuerza.

En los dos últimos fragmentos te encuentras con Mammón, desatando peleas. La abuela consigue derrotarlo y es capaz de reconstruir el reloj de Chronos y activar su poder. Ella consigue manipular el tiempo para retroceder hasta el momento del accidente del abuelo para salvarlo, sin alterar los acontecimientos posteriores.

Después, intentan destruir el reloj, pero el poder desatado en el intento es incontrolable, y lo único que se ve a continuación es una explosión como el Big Bang, de lo que no se sabe nada más.

1.4.2. Desarrollo total del juego.

Al comenzar el videojuego la abuela se encontrará en los alrededores del templo situado en el planeta Tierra. Dentro del templo se encuentra el primer portal a una nueva galaxia. También podrá explorar el entorno, en el que se puede encontrar powerups y objetos que le serán de utilidad en la jornada.

Al atravesar el portal, la abuela aparece en el templo situado en el nuevo planeta, con dos portales; uno para regresar al mundo anterior, y el otro hacía el siguiente mundo, el cual se hará accesible al conseguir las piezas del reloj presentes en ese planeta.

En el primer planeta se derrotará a un jefe, enemigos, y se resolverá un par de puzzles para obtener los 3 fragmentos. Opcionalmente también se podrá explorar y hacer más puzzles para conseguir objetos. También se fortalecerá la habilidad de controlar el tiempo conforme se recuperan más piezas del reloj, de manera que surjan nuevas mecánicas o variaciones del poder.

Después de avanzar por el portal se llega al siguiente planeta. Igual que el anterior, se derrotará enemigos y se completará puzzles para conseguir los fragmentos e items opcionales, pero con el añadido de dos enfrentamientos contra Mammón. El primero es una pelea parcial de la que huirá, y el segundo es el enfrentamiento final, en el que vendrá más equipado y será más difícil.

Por último se juega una última sección en la que La abuela caminará en el tiempo para evitar la muerte del abuelo. En seguida habrá una escena en la que los dos intentan destruir el reloj, causando una gran explosión.

2. Desarrollo técnico

2.1. Videojuego en 2D

2.1.1. Descripción

La idea principal del equipo de desarrollo era crear un videojuego de acción y aventuras en el que el usuario pudiera disfrutar de un vasto entorno para explorar. Por lo tanto, se decidió que la mejor manera de proporcionar esta experiencia de juego era mediante una vista cenital (top-down) ligeramente inclinada, lo que aumentaría la sensación de profundidad. Otra ventaja de este tipo de vista es que permite al jugador moverse en diferentes direcciones dentro del plano horizontal, proporcionando así más libertad de movimiento y una experiencia de juego más inmersiva.

Se implementó un sistema de generación procedural para los niveles del juego

2.1.1.1. Personajes

■ La Abuela:

El personaje principal del juego es la Abuela, que posee una amplia variedad de habilidades que le permiten interactuar con el entorno de manera efectiva. Puede caminar y correr, lo que le da una gran movilidad para explorar los diferentes escenarios del juego. Además, puede atacar a los enemigos con su bastón, lo que le da una ventaja en combate, y también tiene la capacidad de lanzar zapatillas a distancia para infligir daño a los enemigos.

Pero lo que hace que la Abuela sea un personaje verdaderamente interesante es su habilidad especial para retroceder en el tiempo utilizando el reloj de Chronos. Esta habilidad es invaluable en situaciones en las que se necesita corregir un error o volver a una posición anterior en el juego. Con esta habilidad, la Abuela puede enfrentar difíciles desafíos con mayor facilidad y completar misiones de manera más eficiente.

En definitiva, la combinación de habilidades de combate y movilidad de la Abuela, junto con su capacidad única de retroceder en el tiempo, la convierten en un personaje poderoso y estratégico que los jugadores disfrutarán controlar.

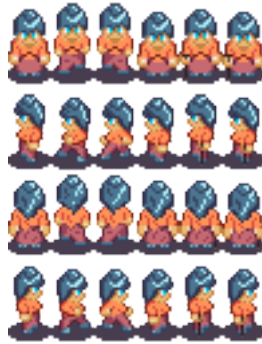


Figura 5: Sprite sheet correspondiente al movimiento básico de la Abuela.

2.1.1.2. Enemigos

Se han implementado enemigos variados que se distribuyen en los distintos mundos del juego. Con el fin de proporcionar una experiencia de juego más inmersiva, se ha diseñado un enemigo exclusivo para cada nivel.

- Avispa: es un enemigo común que habita en el planeta Tierra. Utiliza un tipo de ataque cuerpo a cuerpo. Es capaz de volar a la misma velocidad a la que camina el personaje principal. Para encontrar el camino hacia sus enemigos, utiliza el algoritmo A* que permite encontrar un camino corto evitando obstáculos. Para derrotar a la avispa se requiere de un único golpe. Así mismo, cada ataque de la avispa le quita medio corazón al jugador.



Figura 6: Sprite sheet correspondiente a la animación estacionaria, movimiento, ataque y fallecimiento de la Avispa.

- Minotauro: se localiza en el segundo nivel del juego, perteneciente al planeta Tatooine. Al igual que la Avispa, utiliza el ataque cuerpo a cuerpo.

A diferencia de la Avispa, este enemigo tiene un rango de visión menor, por lo que requerirá que el jugador se acerque más para que este le siga y lo ataque. Sin embargo, el minotauro es un contrincante más robusto, por lo cual dispone de más vida y más daño en sus ataques.



Figura 7: Sprite sheet correspondiente al movimiento, ataque y fallecimiento del Minotauro.

- Fantasma: este enemigo pertenece al tercer nivel del juego ubicado en el planeta Enceladus. Este enemigo dispone de un rango visión mayor, incluso que el de la Avispa, sin embargo tiene una velocidad de desplazamiento menor. Una característica destacable es su capacidad para atravesar elementos colisionables. Para realizar su ataque se desvanece momentáneamente para reaparecer en otra posición causando daño al jugador. Su ataque causa medio corazón de daño

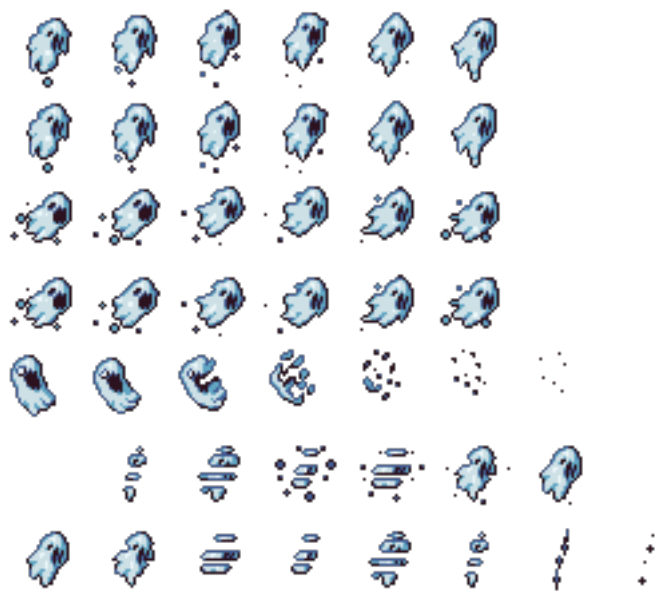


Figura 8: Sprite sheet correspondiente al movimiento, ataque y fallecimiento del Fantasma.

2.1.1.3. Objetos

- Reloj de Chronos:

Este objeto es considerado el más codiciado por aquellos seres que conocen su existencia. Su poder es tal, que en manos equivocadas, podría extinguir toda forma de vida conocida. El objetivo de la Abuela es recolectar las piezas faltantes, para poder avanzar por los diferentes mundos y salvar a su amado.



Figura 9: Reloj de Chronos.

- Zapatillas:

Las zapatillas son objetos que estarán dispersos a lo largo del mapa y podrán ser recogidos para recargar el ataque "Zapatillazo", el jugador tendrá un número limitado de zapatillas para lanzar.

- Corazones:



Figura 10: Sprite correspondiente a un frame de una zapatilla.

Al recoger este objeto, la Abuela recuperará un corazón de vida, hasta llegar al máximo de vida, de tres corazones.



Figura 11: Sprite correspondiente a un frame de un corazón.

- Portal:

El portal es la puerta a un nuevo mundo, y nivel, una vez finalizados todos los objetivos del planeta actual.



Figura 12: Sprite correspondiente al portal.

2.1.1.4. Diseño

- Descripción:

El juego comienza en el planeta Tierra. La Abuela, armada con su bastón y las zapatillas que va recolectando, debe recuperar todas las piezas de reloj esparcidas, y liberar al mundo de un determinado número de enemigos para poder avanzar hasta el siguiente planeta.

- Fase 1: El Planeta Tierra

En este nivel, la Abuela se enfrentará a varias Avispas cuyo objetivo es defender las tan preciadas piezas del Reloj de Chronos. Estas se encuentran cuidadosamente escondidas en algún lugar del planeta. Una vez que la Abuela haya conseguido todas las piezas del reloj, abrirá automáticamente el portal hacia el próximo planeta.



Figura 13: Screenshot nivel 1.

- Fase 2: Planeta Tatooine

Este escenario es completamente árido, caracterizado por la ausencia total de agua, ofreciendo una gran semejanza a los desierto terrícolas. El jugador debe estar en guardia, ya que resulta fácil desorientarse y ser aniquilado por los guardianes de las piezas del reloj: los Minotauros, misteriosas criaturas extremadamente agresivas pero con una visión no envidiable. La Abuela deberá derrotar un número determinado de Minotauros y recuperar las piezas del reloj para poder avanzar al siguiente planeta.



Figura 14: Screenshot nivel 2.

- Fase 3: Planeta Enceladus:
Este es un mundo extraterrestre con una topografía característica, está compuesto por una superficie azul y mares de un morado profundo. Los principales habitantes de este planeta son los fantasmas, los cuales pueden desplazarse con gran libertad, representando una amenaza constante para la Abuela.
Una vez más, la Abuela deberá derrotar cierto número de fantasmas y recuperar las piezas del reloj para completar finalmente el juego.



Figura 15: Screenshot nivel 3.

- Mecánicas:
Las mecánicas del videojuego, agrupadas en categorías, son las siguientes:
 - Desplazamiento: el personaje principal puede desplazarse en cuatro direcciones (arriba, abajo, izquierda y derecha) sobre el mundo bidimensional, ya sea caminando o corriendo.
 - Ataque: como el videojuego cuenta con una serie de enemigos que intentan matar a La Abuela, esta dispone de dos ataques:
 - Bastonazo: se trata de un ataque tipo *melee* (cuerpo a cuerpo) en el que la abuela utiliza su bastón. Cada golpe de este ataque le costará al enemigo un punto de vida.

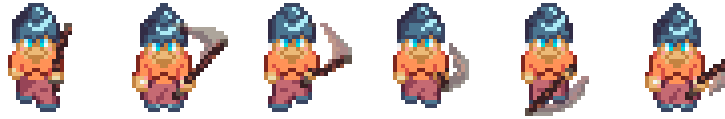


Figura 16: Sprite sheet correspondiente al bastonazo orientado hacia abajo.

- Zapatillazo: ataque a distancia. La Abuela lanza una zapatilla con el fin de dañar a sus enemigos a distancia. Cada golpe de la zapatilla restará un total de dos puntos de vida al enemigo. Al principio, la abuela solo dispone de dos zapatillas. Sin embargo, durante su travesía por los mundos, podrá recoger las zapatillas que encuentre esparcidas para utilizarlas en su defensa.

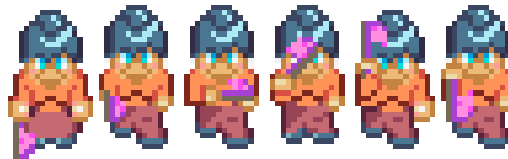


Figura 17: Sprite sheet correspondiente al zapatillazo orientado hacia abajo.

- Mecánica del tiempo: la mecánica del tiempo es una pieza característica del videojuego. Su objetivo principal es permitir que el usuario pueda retroceder en el tiempo, siempre y cuando tenga suficiente maná, para poder corregir ciertas acciones que hayan tenido lugar en el pasado. Esta mecánica tiene un impacto directo en todas las entidades del juego, permitiendo que el usuario pueda tomar decisiones más informadas y estratégicas, a la vez que permite al jugador restaurar su cantidad de vida en instantes previos.

Además, para aumentar la inmersión del jugador en el mundo, se ha implementado una funcionalidad complementaria en la mecánica del tiempo: la música de ambiente también retrocede en el tiempo junto con la acción del jugador. De esta forma, la banda sonora del juego cambia dependiendo de las acciones del usuario, creando una experiencia auditiva más cohesiva y envolvente.

2.1.2. Metodología de desarrollo

Como podemos ver reflejado en la literatura[LM12] el uso de una Scrum beneficia la organización y eficacia de equipos pequeños o medianos. Es por esto por lo que nos hemos decantado por dicha metodología. Scrum se basa en la creación de sprints, que son ciclos cortos de trabajo en los que el equipo se enfoca en tareas específicas.

En nuestro caso, Alejandra se encargó de la gestión de todos los menús, implementando las diferentes maneras de interacción. Conjuntamente con David fue responsable de todo el apartado artístico del videojuego, incluyendo el diseño de sprites y la música. Además, David también se encargó de implementar la lógica de control de retroceso musical y la implementación de la interfaz de usuario. Alejandra también es la responsable de todo el código para realizar la carga de sprites y el manejo de estos en el `Sprite_handler`. Además, realizó la implementación de ambos ataques de la abuela, incluyendo la gestión de múltiples zapatillas en la clase `WeaponPool`.

Luca, por su parte, trabajó en el algoritmo de generación procedural, y asumió todo el trabajo de diseño e implementación de enemigos, incluyendo ataques, pathfinding, posicionamiento en el mapa, etc. También es el responsable de toda la estructuración de los niveles, la lógica de los portales y de toda la implementación de items. Mientras tanto, entre Alejandro y Luca desarrollaron la implementación de la cámara, la del jugador y la estructuración de control del director. Además, Alejandro desarrolló la abstracción y la implementación de los controladores, la carga de tiles en función del mapa usando bitmasking, la mecánica de retroceder en el tiempo y el estándar CSV para cargar sprites.

En nuestro proyecto, cada sprint tenía una duración de una semana natural, lo que nos permitió tener un enfoque constante y una buena gestión del tiempo. Para organizar las tareas y el progreso de cada sprint utilizamos una tabla tipo Scrum en Trello. Esta tabla nos permitió visualizar las tareas pendientes, en proceso y completadas, facilitando así la comunicación y la colaboración en equipo.

El uso de esta herramienta fue fundamental para llevar un registro de las tareas y el progreso de cada miembro del equipo, esto nos ayudó a tener una mejor gestión de tiempo y a cumplir con los objetivos de cada sprint.

En resumen, la metodología Scrum y la utilización de herramientas como Trello nos permitieron trabajar de manera eficiente, organizada y tener una comunicación constante y una visión clara de las tareas a realizar en cada sprint. En general, el uso de Scrum nos permitió trabajar de manera más productiva y organizada.

2.1.3. Escenas

Debido a la naturaleza de la generación procedural de los niveles del juego, estructuraremos todos los elementos en común de las escenas en el apartado más general de Escenas, y los elementos que son variantes serán los que se documentarán en cada apartado correspondiente a la escena.

El flujo de control de las escenas del juego recae en la clase Director (fig. 20), el cual es una implementación del patrón Singleton[Wika]. Este patrón restringe el número de instancias que se pueden crear de una clase a una única instancia, esto nos permite guardar un estado global al que podemos acceder desde cualquier parte simplemente instanciando la clase. El Director posee una estructura de datos de tipo cola en la que guarda las escenas del juego que se van cargando. También es el responsable de ejecutar el *game loop* o bucle del juego.

Todas las escenas heredan de un interfaz en común llamado SceneInterface (fig. 18), y el Director es el encargado de pasar los datos de una escena a la otra cuando se cambia de escena utilizando los métodos *get_player_data()* y *set_player_data()* de las escenas. (véase sec. 2.1.3.5).

Las escenas jugables son creadas a partir de la clase Level (fig.21), que también hereda de SceneInterface. Como la generación de los niveles es procedural, prácticamente todas las escenas comparten la misma estructura y flujo, esencialmente usando una variante del patrón plantilla[Wikb] en la que se concretiza unos pocos parámetros para cada escena. Después se detallará algunos aspectos que se ven en el diagrama de clase, como la generación procedural (sec. 2.1.5) y el objeto Clock (sec. 2.1.5.1).

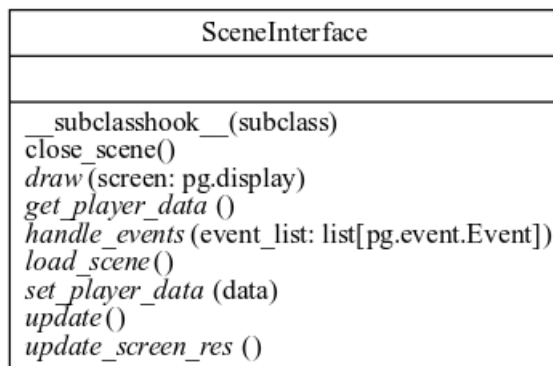


Figura 18: Interfaz *SceneInterface* que es el interfaz expuesto al Director.

El diagrama de flujo de las escenas y el Director se podría representar como la figura 19.

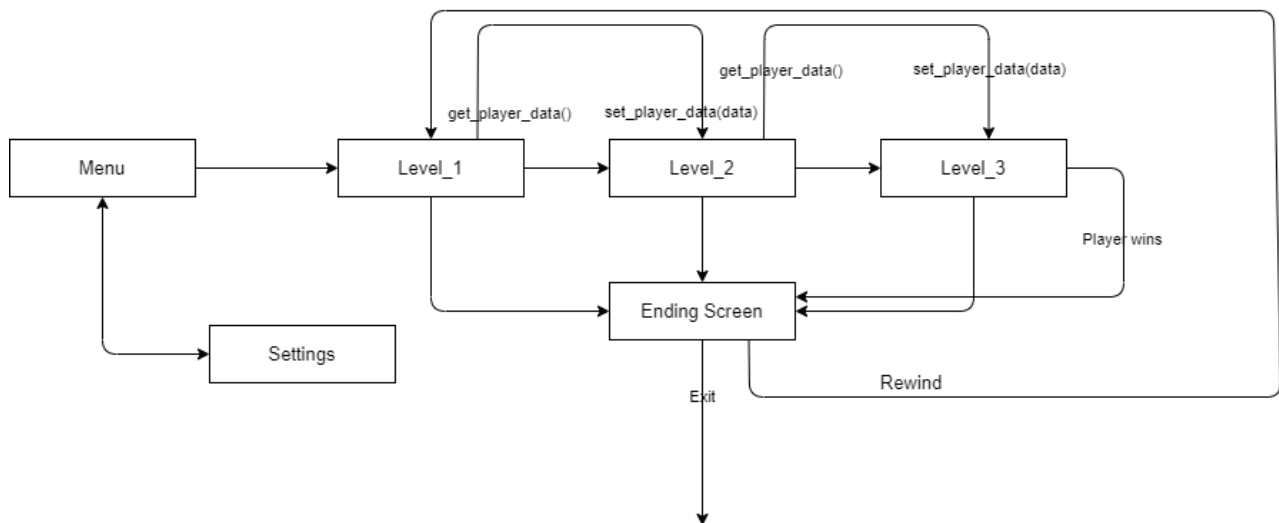


Figura 19: Diagrama de flujo de las escenas del juego. La pantalla de muerte y final es la misma, y existen 3 niveles.

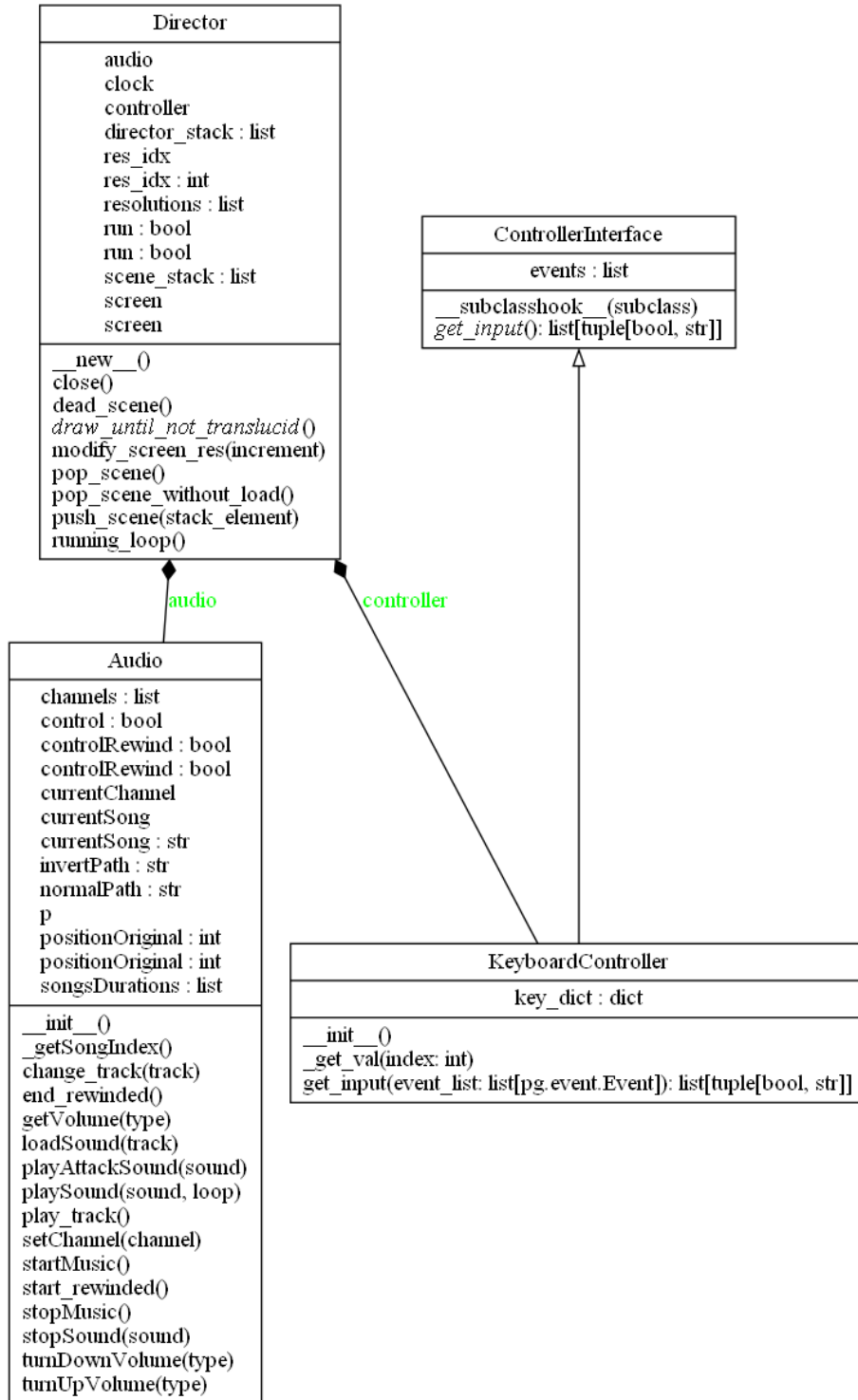


Figura 20: Clase Director que implementa patrón Singleton.

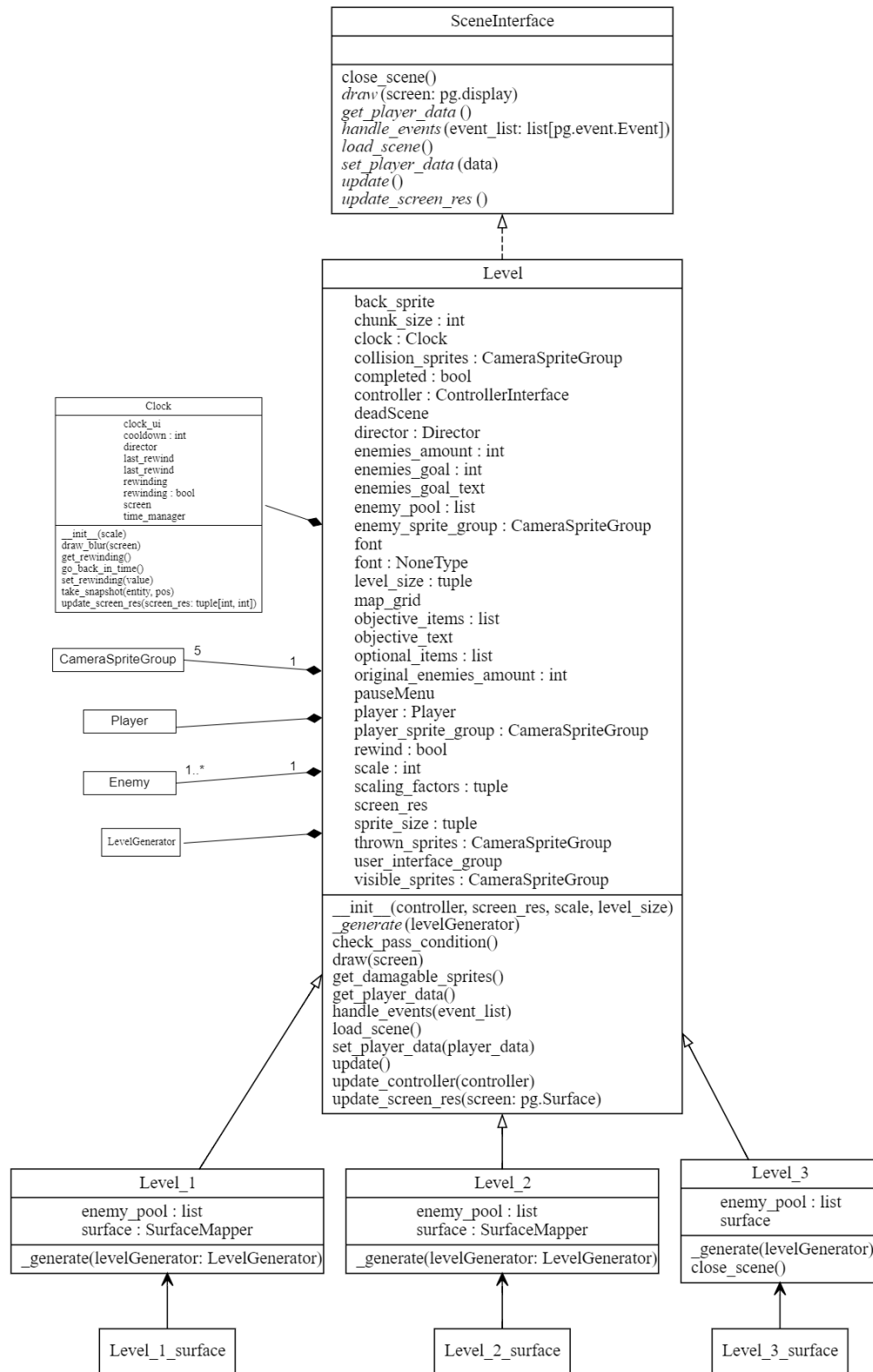
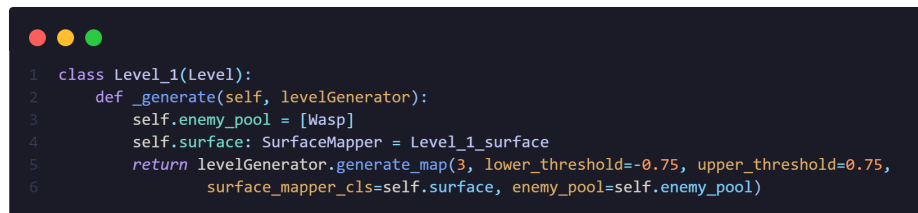


Figura 21: Clase Level, que es la encargada de manejar y pintar todos los objetos en escena como el jugador, los enemigos, el mapa, los items, proyectiles, UI...

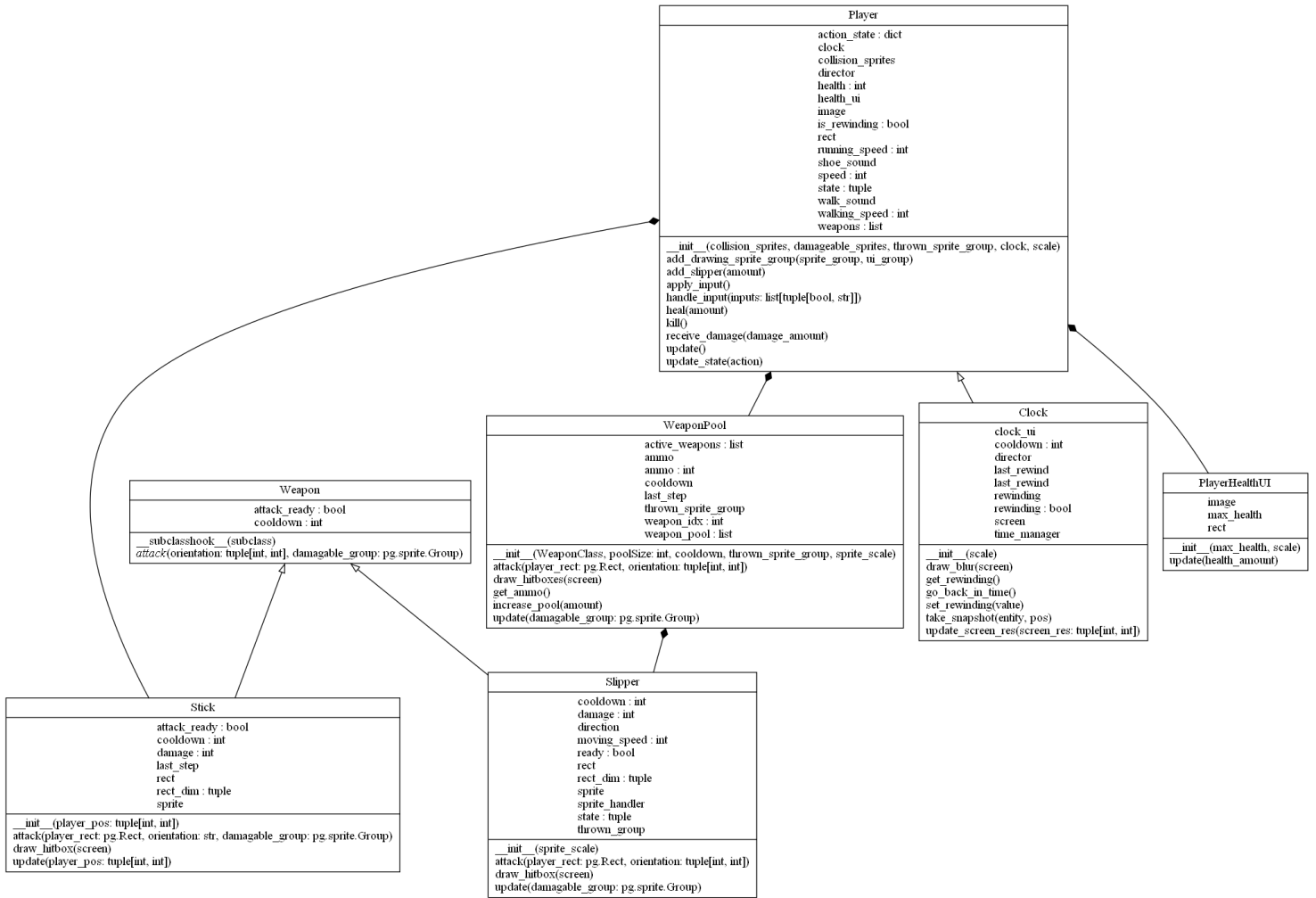
Para la creación de un nivel lo único que se tiene que hacer es definir qué clase de SurfaceMapper (detallada en la sec. 2.1.5) se tiene que usar y cuál es el pool de enemigos que el nivel tendrá (fig. 22).

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and defines a class named `Level_1` that inherits from `Level`. The class has a method `_generate` that initializes an enemy pool with `Wasp` and creates a `SurfaceMapper` object. The method then calls `levelGenerator.generate_map` with specific parameters and returns the result.

```
1 class Level_1(Level):
2     def _generate(self, levelGenerator):
3         self.enemy_pool = [Wasp]
4         self.surface: SurfaceMapper = Level_1_surface
5         return levelGenerator.generate_map(3, lower_threshold=-0.75, upper_threshold=0.75,
6             surface_mapper_cls=self.surface, enemy_pool=self.enemy_pool)
```

Figura 22: Clase de Level_1, que hereda de Level y define los atributos necesarios para la creación del nivel concreto.

Cada nivel contiene a un objeto Player, correspondiente al jugador. fig. 2.1.3 Este en cada ciclo de juego recibirá una lista de acciones acotadas por la variable *events* de la clase ControllerInterface la cual podrá usar para actualizar su estado interno. Esta lista es calculada dentro del Level que recibe una lista de *pygame events*[Pyg] proveniente del Director. De manera complementaria, en cada ciclo, se llamará a la función *update()* del jugador, para que lleve a cabo las acciones correspondientes a su estado interno.



Tanto el jugador (Player) como los enemigos (Enemy) heredan de una clase en común llamada Entity, que posee atributos comunes como la vida, velocidad de movimiento, etc . También incluye funciones para el manejo de sus spritesheets y animaciones (clase Sprite_handler), y un enumerado de acciones para definir los distintos estados y orientaciones en los que se puede encontrar una entidad. fig. 23.

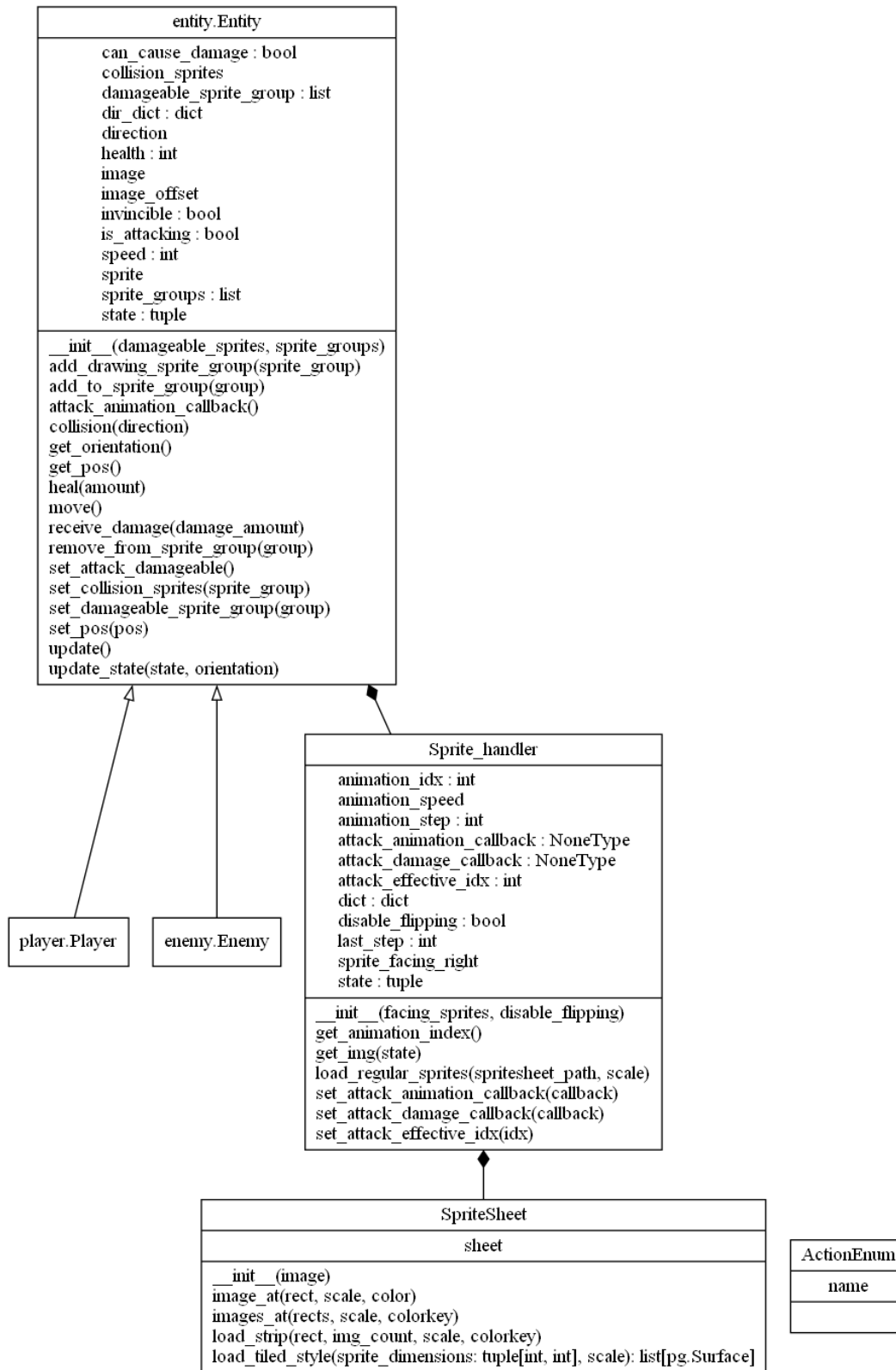


Figura 23: Clase Entity, de la que heredan Player y Enemy. Hace uso del ActionEnum, y el estado de una entidad se compone de una acción de ese enumerado con una orientación, que son strings que pueden tomar los valores "up", "down", "left" y "right".

Cada nivel también contiene sus correspondientes enemigos (fig. 24). Los enemigos tienen su arma (clase MonsterWeapon), que hereda de la clase Weapon, con la que pueden hacer daño. El arma como tal no tiene texturas y se basa en una hitbox que se hace AdHoc para cada enemigo que se desplaza en función de la hit box del enemigo atacando.

Cada enemigo que hereda de la clase general Enemy define su spritesheet y colisión a la superclase. Además de eso también sobrescribe otros valores de los que hereda de Entity como la vida y la velocidad. También define atributos de su "arma", como la hitbox, cooldown, y daño.

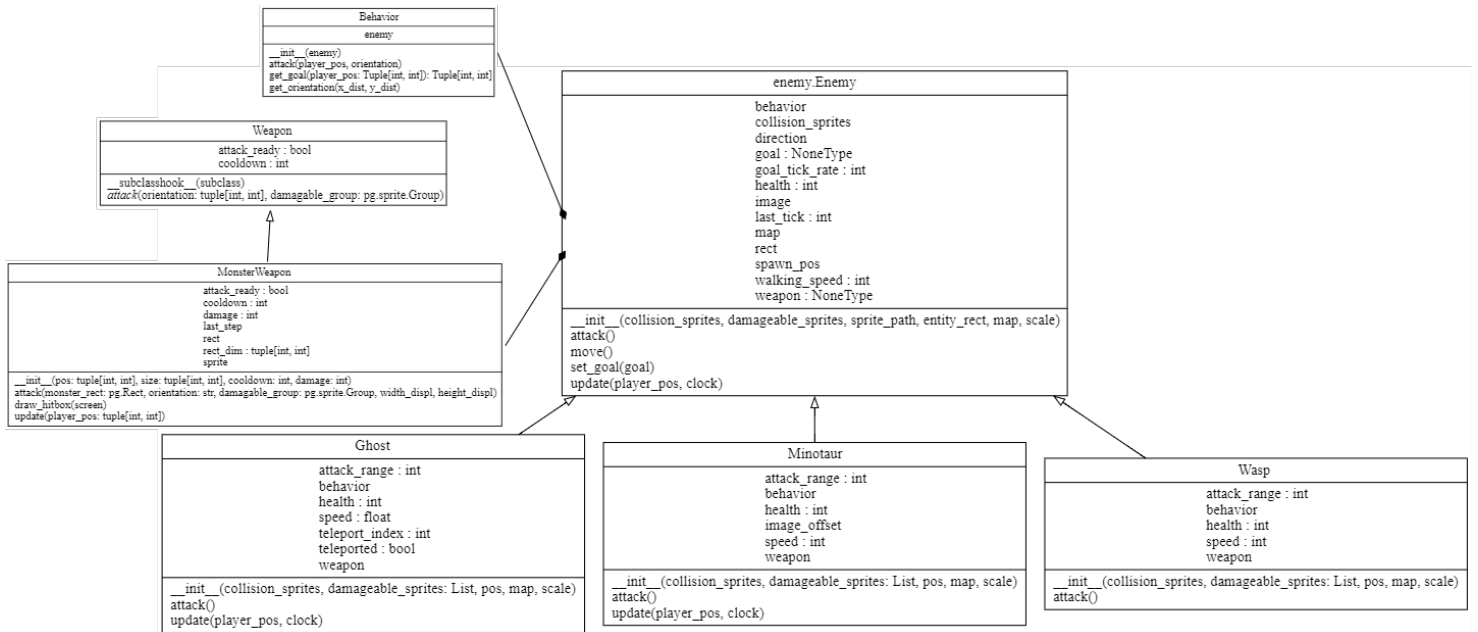


Figura 24: La clase Enemy contiene una clase Behavior, de la que pide instrucciones a su Behavior con el método get.goal().

```

1 class Wasp(Enemy):
2
3     def __init__(self, collision_sprites, damageable_sprites: List, pos, map=None, scale=1):
4         entity_rect = Rect(pos[0], pos[1], 54, 54)
5         sprite_path = '../sprites/enemies/wasp'
6         super().__init__(collision_sprites, damageable_sprites, sprite_path, entity_rect, map, scale, facing_sprites='left')
7         self.health = 1
8         self.attack_range = 60
9         self.behavior = ChaseBehavior(self, self.attack_range, 450)
10        self.speed = 4
11
12        # Wasp specific "weapon"
13        weapon_hitbox = (self.rect.size[0] * 0.9, self.rect.size[1] * 0.6)
14        weapon_damage = 1
15        weapon_cooldown = 7 * 120
16        self.weapon = MonsterWeapon(self.rect.center, weapon_hitbox, weapon_cooldown, weapon_damage)
17        self.sprite.set_attack_effective_idx(3)
18
19    def attack(self):
20        ...

```

Figura 25: Ejemplo de una clase de un enemigo concreto.

2.1.3.1. Escena 1: Menú Inicial

- Descripción

En la primera escena tenemos el menú inicial, desde el que podemos acceder al primer nivel, al menú de opciones o salir del juego.

2.1.3.2. Escena 2: Menú opciones

- Descripción

Nos permite ajustar los valores de sonido del juego y el tamaño de pantalla. Desde esta escena únicamente se puede volver al menú inicial.

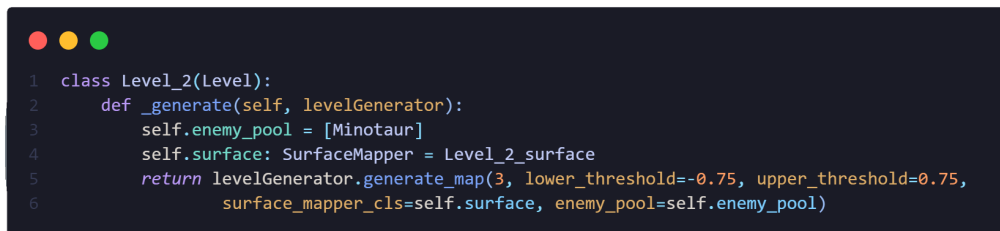
2.1.3.3. Escena 3: Nivel 1

En el primer nivel nos encontramos en el planeta Tierra, con un tileset color verde correspondiente al césped, y los obstáculos son lagos esparcidos por el mapa. La clase correspondiente se vio previamente en la fig. 22, la cual implementa el SurfaceMapper Level_1.surface.

La pool de enemigos de este nivel consta solo de avispas (fig. 25), que implementan un comportamiento de ChaseBehaviour sencillo, basado en seguir al jugador en línea recta a partir de un rango, y atacar a partir de otro. Tienen una velocidad mediana y hacen 1 de daño al jugador.

2.1.3.4. Escena 4: Nivel 2

En el segundo nivel nos encontramos en un desierto en el planeta Tatooine, cuyos obstáculos son casas del desierto esparcidas por el mapa. La clase `Level_2` (fig. 26) es la encargada de generarlo, y usa como pool de enemigos a los Minotauros (fig. ??), que son enemigos lentos, pero que tienen ataques contundentes que hacen 2 de daño al jugador.



```
1 class Level_2(Level):
2     def _generate(self, levelGenerator):
3         self.enemy_pool = [Minotaur]
4         self.surface: SurfaceMapper = Level_2_surface
5         return levelGenerator.generate_map(3, lower_threshold=-0.75, upper_threshold=0.75,
6             surface_mapper_cls=self.surface, enemy_pool=self.enemy_pool)
```

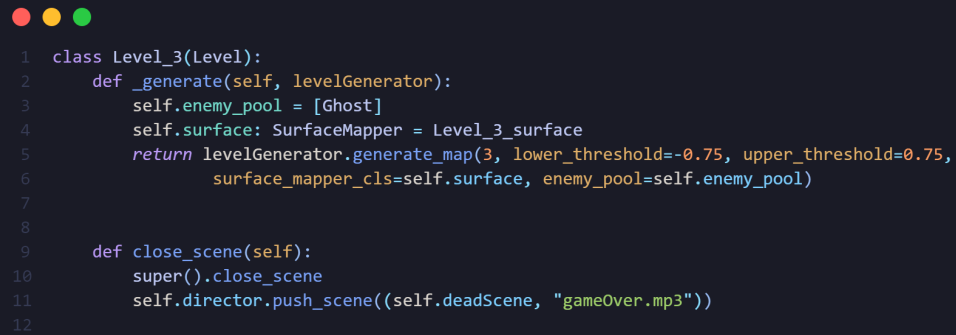
Figura 26: La clase `Level_2`, que le da la implementación a la clase `Level` para generar la escena del segundo nivel. Usa la clase de `Level_2_surface` para su creación.

Cabe destacar que se hacen algunas operaciones de más en los métodos de `update` y `attack` de la clase `Minotaur` porque su spritesheet es demasiado grande y llena de espacio vacío. Otra característica del Minotauro es que utiliza un comportamiento de `PatrolBehavior`, que hace que patrulle por donde aparece, y limita la distancia a la que se puede alejar de su punto de patrulla (detallado en la sec. 2.1.4.2).

2.1.3.5. Escena 5: Nivel 3

En el tercer nivel nos encontramos en un desierto en el planeta Enceladus. Este planeta es un tanto peculiar, con un suelo de color cian, estructuras de color verde y lagos llenos de un líquido morado desconocido. La clase que implementa ese nivel la de `Level_3` (fig. 27), e tiene como enemigos a los fantasmas (fig. ??). Los fantasmas implementan el mismo comportamiento que las Avispas del nivel 1, pero están modificados para ataquen teletransportándose al jugador.

Este nivel tiene la peculiaridad de sobrescribir el método de `close_scene()`, ya que es el último nivel.



```

1  class Level_3(Level):
2      def _generate(self, levelGenerator):
3          self.enemy_pool = [Ghost]
4          self.surface: SurfaceMapper = Level_3_surface
5          return levelGenerator.generate_map(3, lower_threshold=-0.75, upper_threshold=0.75,
6              surface_mapper_cls=self.surface, enemy_pool=self.enemy_pool)
7
8
9      def close_scene(self):
10         super().close_scene
11         self.director.push_scene((self.deadScene, "gameOver.mp3"))
12

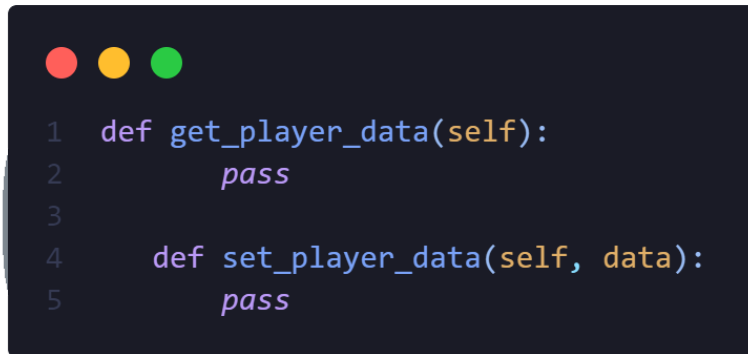
```

Figura 27: La clase Level_3, que le da la implementación a la clase Level para generar la Escena del tercer nivel. Implementa el Level_3_surface para su creación.

2.1.4. Detalles de implementación

2.1.4.1. Persistencia de datos entre niveles

Inicialmente los métodos *get_player_data()* y *set_player_data()* del SceneInterface se llamarían de manera más genérica (como por ejemplo *get/set_data()*), pero el único caso de uso que le damos es para pasar la información del jugador entre niveles (véase sección 2.1.7) jugables, por lo que se dejó ese nombre por claridad. Solo la clase Level (y las que realmente utilicen información del jugador) implementan ese método.



```

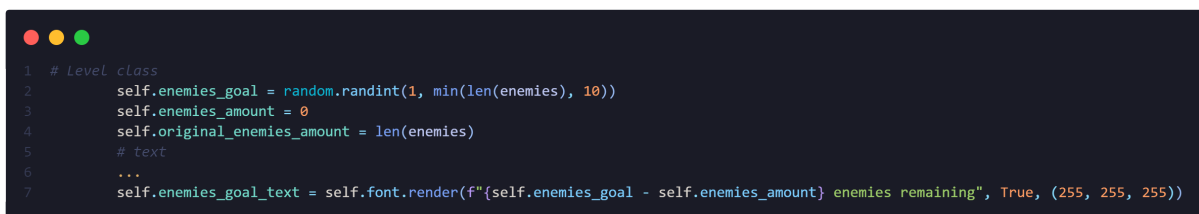
1  def get_player_data(self):
2      pass
3
4  def set_player_data(self, data):
5      pass

```

Figura 28: Implementación por defecto de los métodos `get_player_data()` y `set_player_data(data)` en el interfaz de `SceneInterface`.

2.1.4.2. Sprint del jugador

Mientras probábamos el juego vimos que el jugador podía simplemente correr por la escena recogiendo los objetos clave para pasar al siguiente nivel sin necesidad de enfrentarse a cualquier enemigo o hacer uso de alguna mecánica a mayores. Por eso decidimos implementar otra condición para pasar a la siguiente fase, que es matar un número aleatorio de enemigos del nivel actual (fig. 29). Eso hace que el jugador tenga que entranarse más en el juego.



```

1  # Level class
2  self.enemies_goal = random.randint(1, min(len(enemies), 10))
3  self.enemies_amount = 0
4  self.original_enemies_amount = len(enemies)
5  # text
6  ...
7  self.enemies_goal_text = self.font.render(f"{self.enemies_goal - self.enemies_amount} enemies remaining", True, (255, 255, 255))

```

Figura 29: Se define un número de enemigos a matar entre 1 y 10 como un requisito para pasar de nivel

Otro problema que avistamos mientras desarrollábamos es que la acción de correr del jugador no posee ningún tipo de penalización o limitación como la stamina. Eso hace que correr siempre sea algo beneficioso para el jugador, haciendo que la acción de andar tenga prácticamente ningún uso.

Con todo, no tuvimos tiempo suficiente para implementar un sistema de estimina para limitar la acción de correr, y pensamos que la velocidad a la que anda el jugador, aunque lenta para el ritmo de los enemigos, es más fácil para controlarlo al recoger items; por lo que decidimos dejar la acción de correr incorporada en la versión final.

2.1.4.3. Comportamiento de los enemigos

Para el comportamiento de los enemigos se utilizo un patrón estrategia, con el que permite la modularidad e independecia de los enemigos de su comportamiento. Funciona mediante la agregación de una clase Behavior dentro de la clase Enemy (véase el diagrama de la clase Enemy en la fig. 24), a la que el enemigo le llama el método *get_goal* a cada determinada cantidad de ticks (definido como un atributo de clase en Enemy) para obtener la siguiente acción a realizar.

La clase de Behavior se encarga de modificar al enemigo, que es definido como su atributo en su creación, llamando a métodos como *enemy.update_state(...)* y *enemy.set_goal(...)* para hacer que el enemigo se mueva o ataque. Cada subclase de Behavior además lleva algunos atributos a mayores que son específicos de cada comportamiento, ejemplificado en la fig.??.

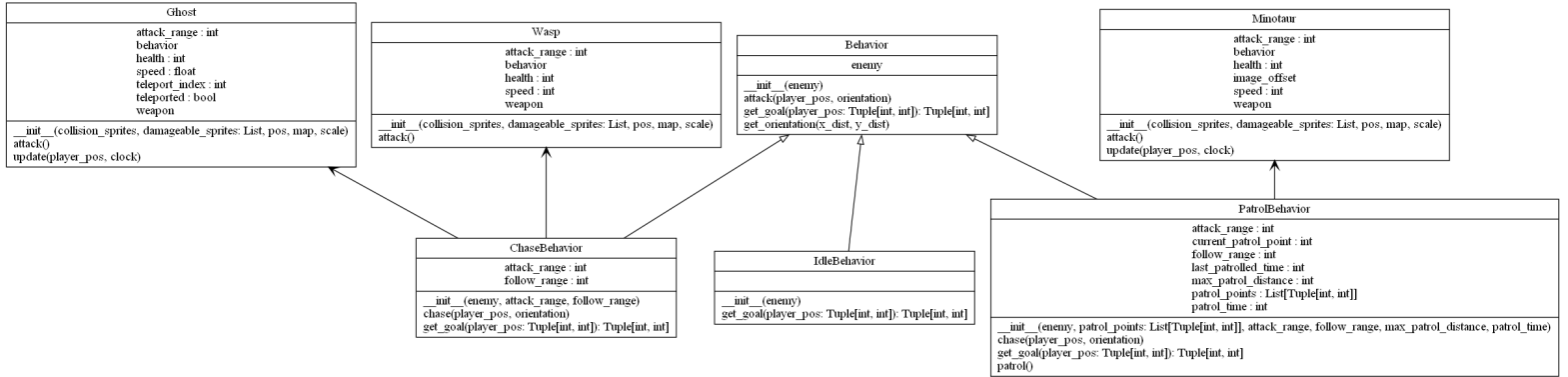


Figura 30: Diagrama de clases de Behavior, donde se ve qué enemigos implementan qué comportamiento.

A priori los enemigos siempre seguían la misma ruta, y acababan aglomerados en un mismo espacio. Para arreglarlo se añadió un pequeño offset aleatorio al punto objetivo para hacer que la ruta de cada enemigo sea ligeramente diferente.

Cabe destacar que para moverse solamente se utiliza un vector de dirección que apunta hacia el destino, por lo que los enemigos pueden quedarse atascados en obstáculos fácilmente. Se podría arreglar ese comportamiento utilizando un algoritmo de A* en los enemigos. Sería un cambio trivial, ya que está implementado y se usa para la creación del mapa, y el enemigo ya recibe la información

del mapa para saber donde están los obstáculos. Con todo, nos inclinamos por un enfoque más simplista.

2.1.5. Aspectos destacables

2.1.5.1. Generación procedural

Una de las características principales del videojuego es la generación procedural de los niveles. El algoritmo funciona en varias fases, empezando por crear un mapa de ruido de Perlin y aplicar una doble umbralización. Esto crea zonas distribuidas en el mapa que futuramente serán asignadas como zonas de obstáculos. Después se hace una división y discretización del mapa en zonas denominadas *chunks*, concluyendo la fase de generación de chunks y mapeado.

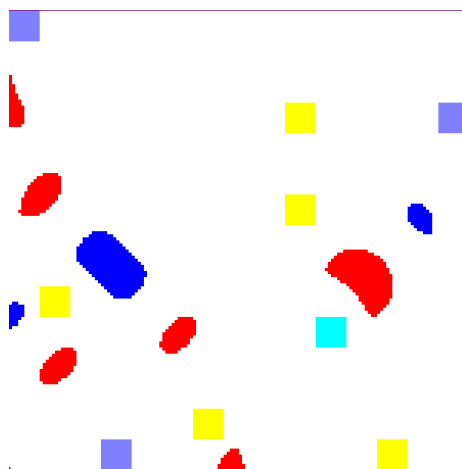


Figura 31: Áreas designadas como obstáculos (en rojo y azul) mediante el ruido de Perlin, junto a los objetivos (violeta) y puntos de interés (amarillo) asignados en el mapa de chunks. El *spawn* o punto de aparición del jugador es marcado en turquesa.

La siguiente fase, denominada la de posicionamiento, se encarga de asignar chunks en el mapa, intentando espaciar los chunks asignados con un radio para que estén bien distribuidos por el mapa. Esta metodología, aunque bastante consistente generando mapas válidos, no tiene garantizada la accesibilidad del mapa, sobretodo en sus zonas críticas como los objetivos principales. Por lo tanto se usa un algoritmo de búsqueda de A* para garantizar que los chunks objetivo son alcanzables desde el spawn. Si en alguna de las generaciones no se cumple ese requisito, el mapa es generado otra vez, hasta un máximo de 10 veces.

```

1 class Level_2_surface(SurfaceMapper):
2     def __init__(self, map_matrix, scale):
3         super().__init__(map_matrix, scale)
4
5         grnd_spritesheet = SpriteSheet(image.load('../sprites/environment_tileset/level2/ground.png'))
6         self.ground_sprite_pool = grnd_spritesheet.load_tiled_style((16,16), scale=scale)
7
8         obst_spritesheet = SpriteSheet(image.load('../sprites/environment_tileset/level2/desert_house.png'))
9         obst_sprites = obst_spritesheet.load_tiled_style((16,16), scale=scale)
10        self.obst1_dict = {
11            "center": obst_sprites[4],
12            ...
13        }
14
15        obst2_spritesheet = SpriteSheet(image.load('../sprites/environment_tileset/level2/desert_house.png'))
16        obst2_sprites = obst2_spritesheet.load_tiled_style((16,16), scale=scale)
17
18        self.obst2_dict = {
19            "left": obst2_sprites[3],
20            ...
21        }

```

Figura 32: Ejemplo de clase SurfaceMapper para el Nivel 2 (usada en Level_2), reducida para simplicidad. Dentro se define qué tilesheets usará el nivel y sus relaciones con diccionarios de Python. El resultado se puede ver en los lagos y edificios que se generan a lo largo del nivel.

La siguiente fase se realiza con la clase de SurfaceMapper (fig. 32), la cual es la que define qué tilesets se usarán en el mapa y cómo se utilizarán las zonas de obstáculos designadas por el algoritmo, y la que se instancia al crear el Level. Esta clase es la encargada de seleccionar correctamente el *sprite* necesario para cada posición del mapa, utilizando un algoritmo de *bitmasking*, en el cual se tienen en cuenta la información de la posición a considerar, y la de sus ocho vecinos. También es la encargada de determinar y generar los elementos que darán lugar a las colisiones del mapa.

La última fase se encarga de poblar el mapa con enemigos e items en las zonas de interés del mapa. Hace uso de una *pool* de enemigos que es definida en la clase Level concreta que se utiliza, y luego pasa sus referencias a la clase Level para que pueda manejar y pintar en pantalla los enemigos e items del nivel.

2.1.5.2. Mecánica de regreso en el tiempo

Esta mecánica está mayoritariamente implementada dentro de la clase Time-Manager. Dicha clase posee una lista *snapshot_list* que se trata como si fuese una lista circular, es decir, cuando queremos insertar un elemento en la siguiente posición del final de la lista, se insertará en el primer elemento. Complementariamente si queremos acceder a la posición -1, accederemos a la última posición de la lista.

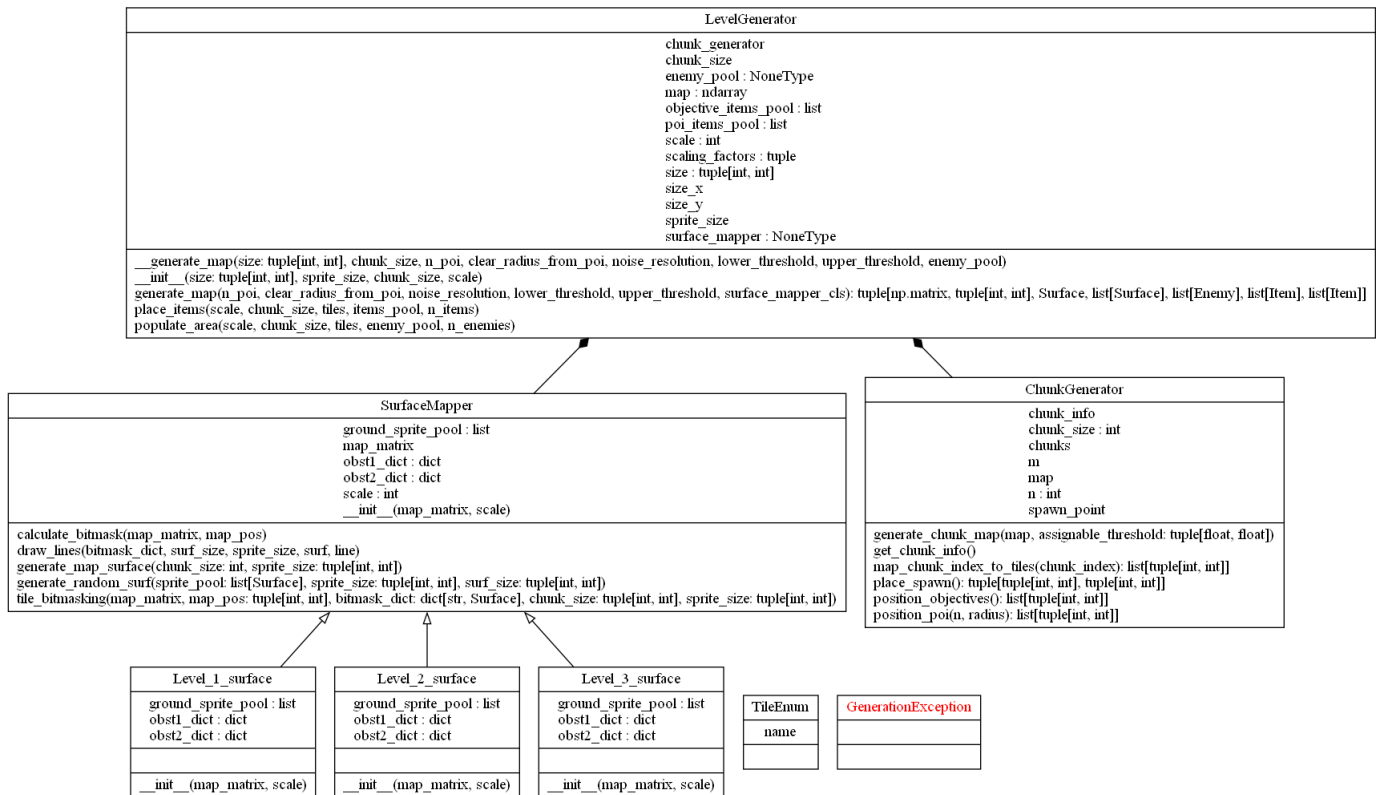


Figura 33: Diagrama de clases del módulo de generación de nivel. La clase LevelGenerator está agregada dentro de la clase Level, visto en el diagrama de la figura 21. La fase de generación de chunks se encuentra en la clase ChunkGenerator.

Cada elemento de esta lista será una lista de tuplas, que indicaran para una entidad cual era su posición. Destacar que la tupla que representa al jugador también almacena información sobre la vida de la que disponía en ese momento, para poder también restaurarla.

Para utilizar esta funcionalidad hay que llamar a la función *take_snapshot()* de manera regular, la cual hará uso de dos *cooldowns*. Uno para saber si a dicha entidad se le ha sacado una *snapshot* recientemente, y otro para saber si tenemos que avanzar en nuestra lista general *snapshot_list*.

La combinación de parámetros *snapshot_cooldown* y *snapshot_len* determinan la longitud de tiempo que se puede retroceder, y la frecuencia con la cual registramos las posiciones de las entidades.

Una vez queremos retroceder en el tiempo necesitaremos llamar a la función *get_snapshot()* la cual nos devolverá *snapshots* con la misma frecuencia que se tomaron. Posteriormente, desde el Clock, podemos utilizar dichos *snapshots* para reposicionar las entidades afectadas.

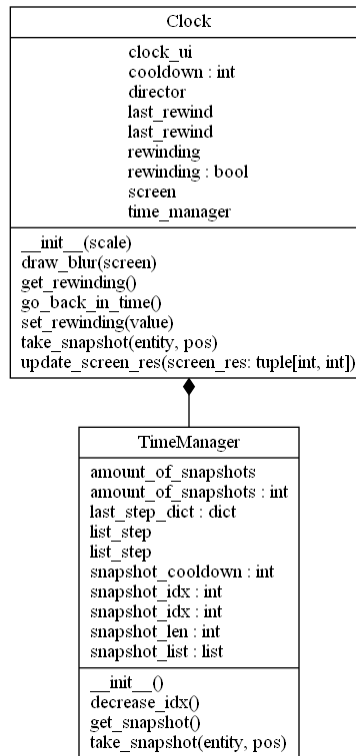


Figura 34: Diagrama de la clase Clock

2.1.5.3. Abstracción de los controles de pygame

Según los proyectos aumentan en tamaño, la falta organización y a abstracción supone un serio decremento en el avance del mismo. Es por esto por lo que en las últimas décadas se ha hecho especial hincapié en la estructuración y abstracción dentro de los programas de gran tamaño[Gam+94].

Siguiendo esa temática hemos decidido desacoplar nuestra representación de los eventos generados por el usuario principalmente por dos motivos:

- Centralización de los eventos utilizados: en el caso de que pygame cambiase el nombre o la manera que tiene de suministrar eventos, la mayoría de nuestro código no se vería afectado. Simplemente tendríamos que modificar nuestras clases de controladores. También en el caso de querer modificar la tecla para una acción concreta, tendremos un punto centralizado que almacena esta información
- Capacidad de implementar diversos controladores: gracias al desacoplamiento de las especificidades de cada dispositivo, podemos permitir que nuestro videojuego disponga de múltiples controladores, tan sólo con el hecho de que se adhieran a las acciones definidas en la clase ControllerInterface. Un claro ejemplo de esto es el JoystickController, el cual es una clase no implementada por completo, que permite el uso de un mando para interactuar con el juego.

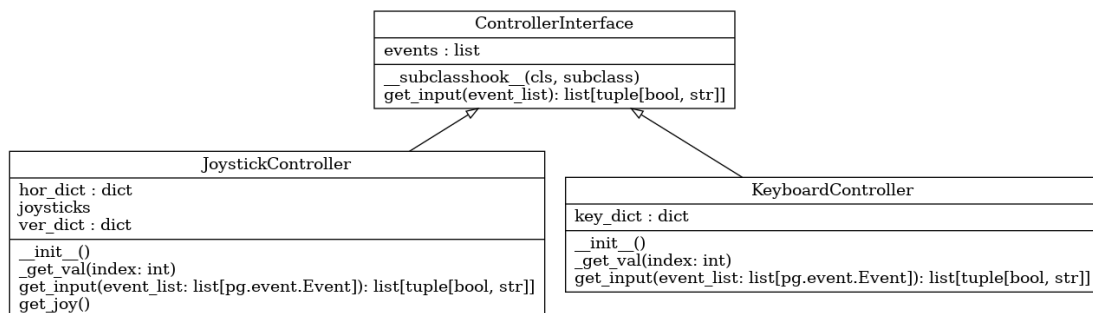


Figura 35: Diagrama de clases de la clase Controller

2.1.5.4. Diseño de una estructura CSV para representar información de entidades

Durante el desarrollo del juego, y con el uso de diferentes animaciones para cada acción de las entidades surgió la problemática de conseguir imágenes asociadas a su información de acción.

Es por ello por lo que optamos por diseñar una estructura específica en el formato *comma separated values* o CSV, que albergase dicha información.

En la primera fila se designa el tamaño en píxeles de cada sprite.

El resto de filas siguen el siguiente formato:

- Nombre de la acción que se va a especificar
- Velocidad de cambio de sprite entre animaciones
- Una orientación (de las cuatro posibles)
- Una serie de valores x e y, que designan la posición relativa de un sprite dentro del spritesheet tomando como unidad el tamaño definido al principio.
- La parte restante de la fila representa las orientaciones restantes.

```
1 width,32,height,48
2 attack_1,50,down,0,0,1,0,2,0,3,0,4,0,5,0,right,0,1,1,2,1,3,1,4,1,5,1,left,0,2,1,2,2,3,2,4,2,5,2,up,0,3,1,3,2,3,3,4,3,5,3
3 attack_2,50,down,0,10,1,10,2,10,3,10,4,10,5,10,left,0,8,1,8,2,8,3,8,4,8,5,8,right,0,9,1,9,2,9,3,9,4,9,5,9,up,0,11,1,11,2,11,3,11,4,11,5,11
4 walk,150,down,0,4,1,4,2,4,2,4,left,0,5,1,5,2,5,up,0,6,1,6,2,6,6,right,0,7,1,7,2,7
5 idle,350,down,0,4,3,4,4,left,0,5,3,5,up,0,6,3,6,6,right,0,7,3,7
6 long_idle,400,down,4,4,5,4,4,5,4,left,4,5,5,5,up,4,6,5,6,6,right,4,7,5,7
7
```

Figura 36: CSV de gestión de sprites de la Abuela

Posteriormente, la clase `Sprite_handler` puede almacenar esta información en un diccionario que utiliza como clave la acción y que, contiene como valor una tupla de dos elementos.

Siendo el segundo elemento la velocidad de animación de la acción, y el primer elemento un diccionario, el cual tiene como clave las cuatro posibles orientaciones y como valor la lista de sprites de la animación.

2.1.6. Manual de usuario

2.1.6.1. Menú principal

El menú principal del juego es crucial para causar una buena primera impresión en los jugadores, por lo que se ha diseñado cuidadosamente con una apariencia atractiva e intuitiva. Con el fin de facilitar la navegación, se ha optado por un diseño sencillo y fácil de usar que permita a los jugadores acceder rápidamente a las opciones que necesitan.

Para asegurar una experiencia visual atractiva, el menú principal cuenta con una paleta de colores cautivadora que se adapta al tema del juego, así como con una banda sonora adecuada para mejorar la inmersión del usuario en el juego.

Al iniciar el juego, los jugadores serán recibidos por el menú principal, que les permitirá acceder a diferentes opciones para comenzar su aventura. El menú principal consta de tres botones claramente destacados:

- Play: Nos permite comenzar la partida.
- Settings: accederemos al menú de configuración. Donde podremos modificar el volumen del audio del juego y la resolución de la ventana.
- Exit: abandonaremos el videojuego de forma segura.



Figura 37: Screenshot menú principal.

2.1.6.2. Menú de configuración

El menú personalizado del juego ha sido diseñado cuidadosamente para brindar al usuario una experiencia agradable y funcional. El menú se presenta de forma clara y organizada, con una apariencia visual atractiva y coherente con el estilo del juego.

En el menú, se pueden encontrar distintas opciones de configuración para ajustar la experiencia de juego a las preferencias del usuario. Una de las opciones disponibles es la de ajustar el volumen de la música, lo que permite al usuario disfrutar de la banda sonora del juego a su gusto. Además, también es posible ajustar el volumen de los efectos de sonido, lo que permite al usuario personalizar aún más su experiencia de juego.

Otra de las opciones disponibles en el menú es la de ajustar la resolución de la pantalla, con cinco opciones diferentes para elegir. Esto permite al usuario adaptar la visualización del juego a las características de su monitor y obtener una experiencia de juego óptima.

En este menú, podrás desplazarte utilizando las teclas del teclado. Para seleccionar uno de los distintos apartados, utiliza las teclas "arriba" y "abajo". Además, podrás ajustar los valores de cada configuración utilizando las teclas "izquierda" y "derecha".

- Sound:
configuración del volumen de los efectos de sonido del videojuego. Se configura de 0 a 10.
- Music:
configuración del volumen de la música del videojuego. Se configura de 0 a 10.
- Size:
configuración del tamaño de la ventana. Los tamaños disponibles son: [1280x720], [1366x768], [1600x900] y [1920x1080]
- Back:
volvemos al menú principal.



Figura 38: Screenshot menú de configuración.

2.1.6.3. Interfaz de usuario

En este apartado conoceremos todos los elementos relevantes para que el jugador disfrute de una mejor experiencia del videojuego.

- Corazones:
Indicador de la vida actual del personaje principal.



Figura 39: Sprites del indicador de vida.

- Reloj:
Indicador de la energía que posee el jugador para volver atrás en el tiempo.



Figura 40: Sprites del indicar del indicador de maná del reloj.

- Objetivos del nivel:
Abajo a la derecha, en la 41, podemos ver los objetivos de cada nivel.

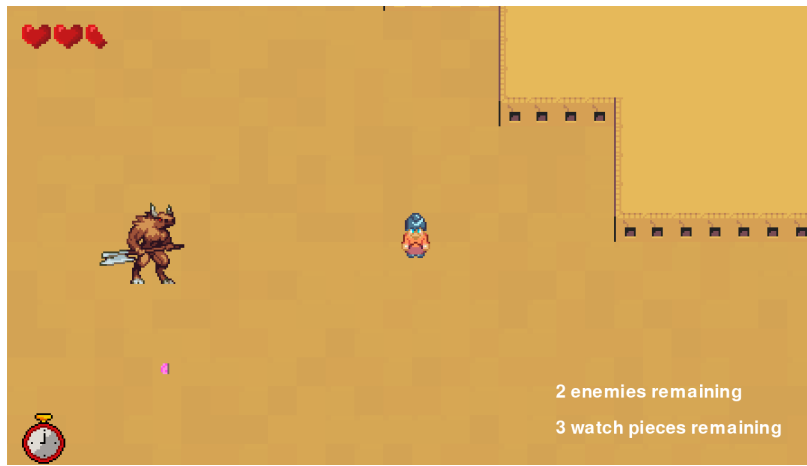


Figura 41: Screenshot en el nivel 2 con los objetivos sin finalizar.

2.1.6.4. Controles

En el mundo de los videojuegos, los controles son la clave para disfrutar de una experiencia de juego fluida y satisfactoria. Saber cómo controlar nuestro personaje y realizar las acciones necesarias en el momento justo es esencial para lograr el éxito en el juego. En este apartado, nos centraremos en explicar de manera detallada los controles de nuestro videojuego para que puedas dominarlos y disfrutar al máximo de la experiencia de juego.

■ Movimiento

- Tecla W o Flecha hacia arriba:
Caminar hacia arriba.
- Tecla S o Flecha hacia abajo:
Caminar hacia abajo.
- Tecla A o Flecha hacia la izquierda:
Caminar hacia la izquierda.
- Tecla D o Flecha hacia la derecha:
Caminar hacia la derecha.
- Tecla Shift + alguna de las opciones anteriores:
Correr en la dirección indicada.

■ Ataque

- Tecla N:
Bastonazo.
- Tecla M:
Zapatillazo.



Figura 42: Menú de pausa

- Habilidades especiales:
 - Espacio:

Mientras se tenga energía y se pulse la tecla, el tiempo volverá hacia atrás.
- Menú de pausa:
 - Tecla Escape:

Abrir menú de pausa.

2.1.7. Reporte de bugs

- La vida del jugador se restablece en cada nivel:

El paso de información del jugador entre niveles con los métodos `get/set_player_data` del `SceneInterface` no funcionan adecuadamente. Después de varias iteraciones descubrimos que el único valor realmente que se tiene que actualizar es la vida del personaje.

En la iteración actual intentamos cambiar el atributo directamente porque no tenemos implementados getters y setters para la vida, y creemos que esa es la causa de que no se actualice. Intentamos también moverlo a un atributo de clase pero la vida del jugador continua siendo restablecida en cada nivel nuevo que se entra.

- Las zapatillas son invisibles a partir del segundo nivel:

A partir del segundo nivel los proyectiles de las zapatillas se vuelven invisibles, pero siguen existiendo y haciendo daño a los enemigos. No sabemos exactamente la causa del bug, pero creemos que puede ser a como se pasa

el `thrown_sprite_group` del `Level` al `WeaponPool` del `Player` cuando este es creado.

- A veces se puede ver el borde del mundo cuando se mueve o cambia la ventana de tamaño:

Cuando se cambia el tamaño de la pantalla (especialmente con ratón) no se registran los eventos y handlers que se encargan de actualizar los límites de la cámara, por lo que puede salirse de los límites del mundo.

- En ciertas ocasiones al volver atrás en el tiempo los enemigos pueden teletransportarse:

Esto se puede deber principalmente a dos cosas, la manera en que estamos cogiendo la información de posición de las entidades, y a que el tiempo de cooldown de una entidad encaje de una manera no deseada con el cooldown de registrar snapshot.

- La zapatilla puede causar daño en área:

Este comportamiento se debe a que en cuanto una zapatilla colisiona con una entidad enemiga, daña a todas las entidades con las que esté colisionando. Por lo tanto si colisiona con dos hitboxes en el momento exacto dañará a ambas.

Referencias

- [Gam+94] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1.^a ed. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1.
- [LM12] Katarzyna Lukaszewicz y Jakub Miler. “Improving agility and discipline of software development with the Scrum and CMMI”. En: *IET software* 6.5 (2012), págs. 416-422.
- [Pyg] Pygame_ocs. *Pygame Events*. URL: <https://www.pygame.org/docs/ref/event.html> (visitado 22-03-2023).
- [Wika] Wikipedia. *Singleton design pattern*. URL: <https://es.wikipedia.org/wiki/Singleton> (visitado 14-03-2023).
- [Wikb] Wikipedia. *Template method pattern*. URL: https://en.wikipedia.org/wiki/Template_method_pattern (visitado 15-03-2023).