Reto 8

Estudiante: Jose Javier Bailón Ortiz
Asignatura: Programación de servicios y procesos
Convocatoria:

Sumario

Concepto general del juego....4
Experiencia de usuario....4
Menú inicial....4
Partida....5
Instrucciones....6
Elementos del juego....7
Nave del jugador....7
Enemigos....7
Naves enemigas....7
Asteroides....7
Tipos de premios....7
Disparos....8
El fondo....8
Otros recursos y elementos que forman parte del juego....9
Atribuciones....10
Casos de uso....11
CU1 - Salir....11
CU2 – Empezar partida....11
CU3 – Ver instrucciones....12
CU4 – Pausar partida....12
CU5 – Disparar....12
CU6 – Mover la nave....13
CU7 – Coger premio....13
Diagramas....14
Diagrama de herencias....14
Diagrama de implementación de interfaces....15
Arquitectura del programa....16
Concepto general de la estructura del programa....16
Motor y el ciclo de un fotograma....17
Concurrencia en el programa....18
Clases principales....20
Controlador....20
Escena....20
EscenaInicial....20
Instrucciones....20
Partida....20
Dibujo....21
Interfaces Colisionable, Disparable y clase Disparo....21
Nave....21
Meteorito....22
Enemigo....22
Premio....22
Recursos....23
Clases de configuración....23
Ejecución del programa....23

Concepto general del juego

El juego, llamado SPACE ATTACK, consiste en un arcade de scroll vertical cuya temática es la lucha en el espacio. El jugador maneja una nave que puede mover por la pantalla y con la que puede disparar a los enemigos. Los enemigos aparecen por la parte superior y van descendiendo mientras disparan. Durante la partida aparecen asteroides que el jugador puede destruir para recibir premios en forma de puntos o mejoras.

Conforme se avanzan los niveles los enemigos son mas peligrosos y difíciles de matar. Cuando el jugador pasa todos los niveles habrá ganado el juego. Si pierde toda su vida habrá terminado el juego.

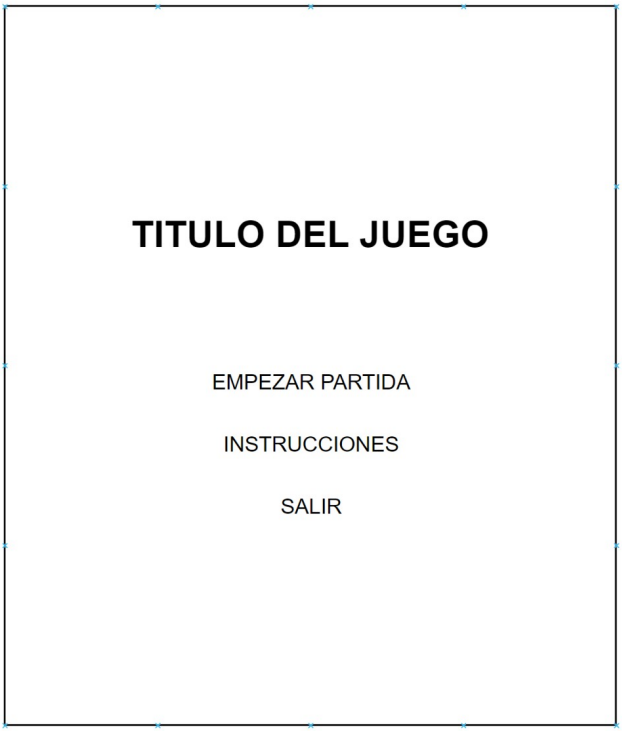
Experiencia de usuario

El usuario vera un menú al comenzar el juego. En ese menú podrá elegir entre empezar una partida, ver las instrucciones y salir del juego.

Las pantallas y sus mockups son los siguientes:

Menú inicial

Con los cursores verticales se puede mover por el menú y con espacio o intro acepta la selección.



Partida

Al comenzar la partida aparecerá la nave del jugador por la parte inferior. Tras una animación de entrada podrá moverla y disparar. Aparecerán oleadas de enemigos y asteroides que debe destruir disparándoles. Al destruirlos dejan premios que el jugador puede recoger pasando sobre ellos con la nave. Conforme pasan los niveles se necesitan mas disparos para destruir los enemigos y las colisiones y disparos enemigos hacen mas daño. El jugador debe compensar eso recogiendo premios de salud y mejoras en el arma principalmente. Cuando se pase de un nivel a otro se informa al usuario de que entra en un nuevo nivel.

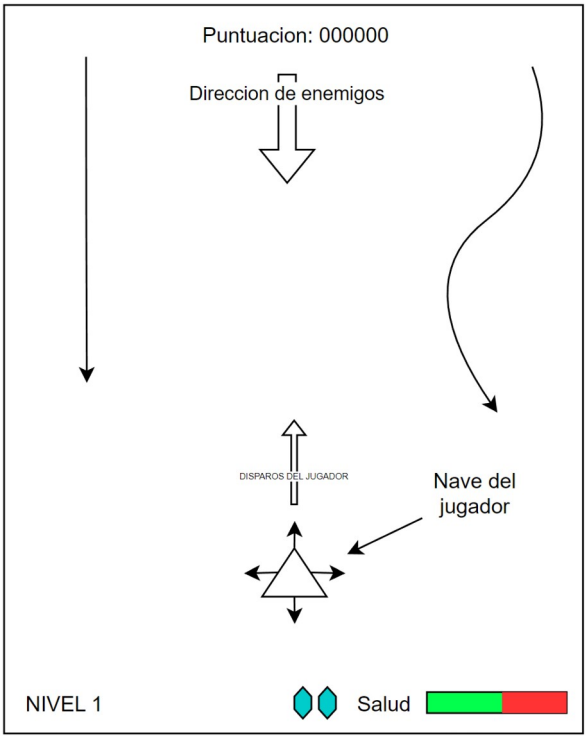
Una vez se agote la salud de la nave y no queden vidas extras disponibles habrá perdido la partida.

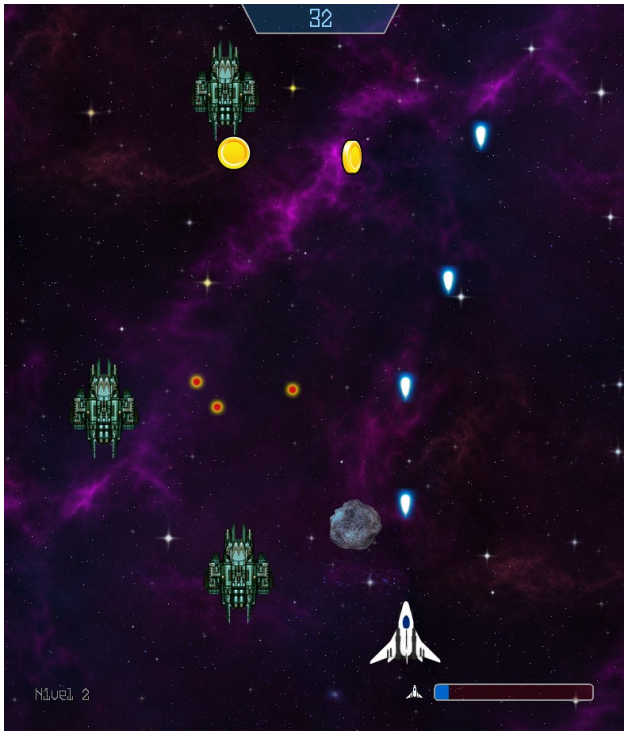
Si consigue superar el ultimo nivel (nivel 10) habrá ganado la partida y la nave efectúa una animación desapareciendo por la parte superior.

Una vez terminada la partida se informa de la victoria o el fracaso y se vuelve al menú inicial.

Durante la partida el jugador puede obtener información del estado de la partida en las siguientes zonas:

- Arriba: puntuación
- Abajo izquierda: nivel actual
- Abajo derecha: cantidad de vidas extra y estado de la salud de la nave



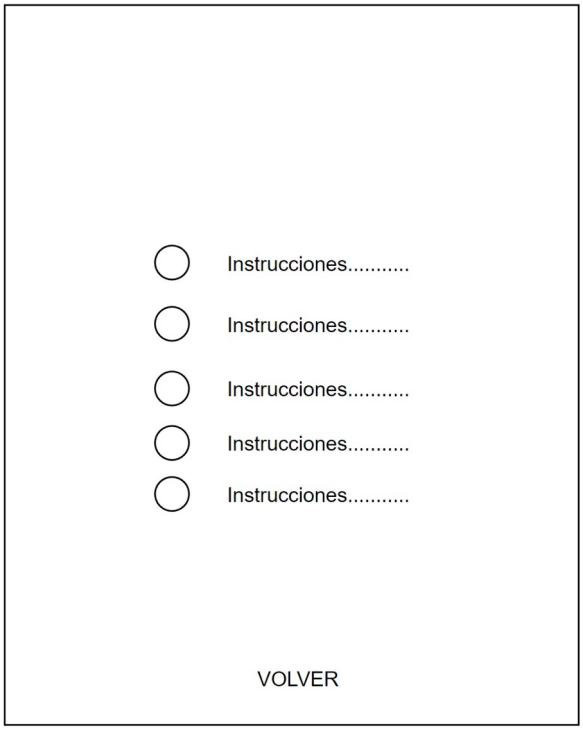


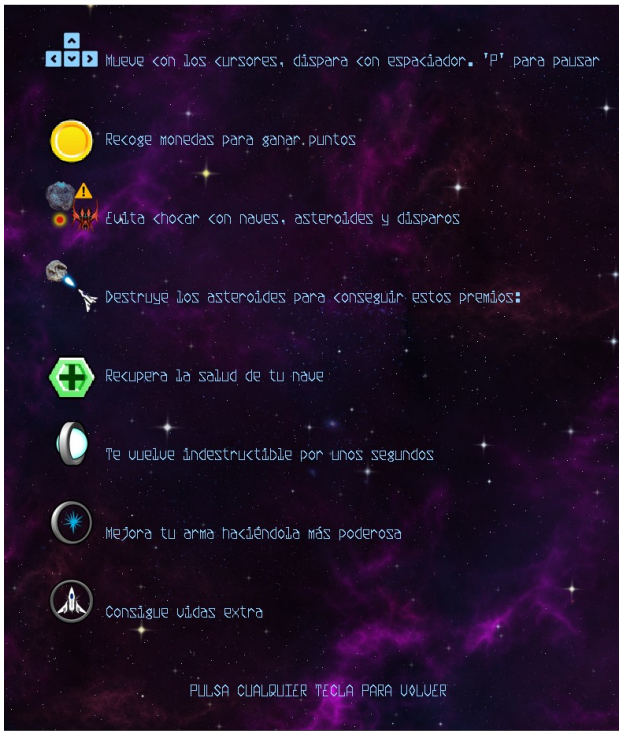
Instrucciones

Al entrar en la pantalla de instrucciones el jugador puede ver información sobre como controlar la nave y los objetivos durante el juego así como una explicación de los efectos de los premios.

Las instrucciones son:

- Mueve con los cursores, dispara con espaciador. “P” para pausar
- Recoge monedas para ganar puntos
- Evita chocar con naves, asteroides y disparos
- Destruye los asteroides para conseguir estos premios(se muestran iconos):
  - Recupera la salud de tu nave
  - Te vuelve indestructible por unos segundos
  - Mejora tu arma haciéndola más poderosa
  - Consigue vidas extra





Elementos del juego

Nave del jugador

La nave se puede mover en cualquier dirección y disparar. Propiedades de la nave:

- Potencia de disparo: Aumenta con premios
- Salud: Tiene una salud inicial que decrementa con los impactos enemigos. Se puede recuperar con premios.
- Escudo: Hace que sea inmune a disparos y colisiones. Un premio lo activa temporalmente.

Enemigos

Naves enemigas

Pueden moverse de manera vertical ondulada o lineal. Disparan hacia posición de la nave del jugador en el momento del disparo. El daño que hacen y su salud se incrementa conforme avanzan los niveles. Al ser destruidas premian con puntos.



Asteroides

Se mueven de manera lineal en dirección descendente aleatoria. Cuando son destruidos premian con cualquiera de los premios disponibles.



Tipos de premios

- **Puntos:** Se reciben puntos para el marcador.
- **Recarga de salud:** Se recarga una cantidad de salud de la nave
- **Escudo:** Una burbuja rodea la nave haciéndola inmune a los disparos y colisiónes
- **Mejora de disparo:** Aumenta la fuerza de los disparos del jugador
- **Vida extra:** Añade una vida extra al jugador

Premios de izquierda a derecha: puntos, salud, escudo, mejora disparo, vida extra





Disparos

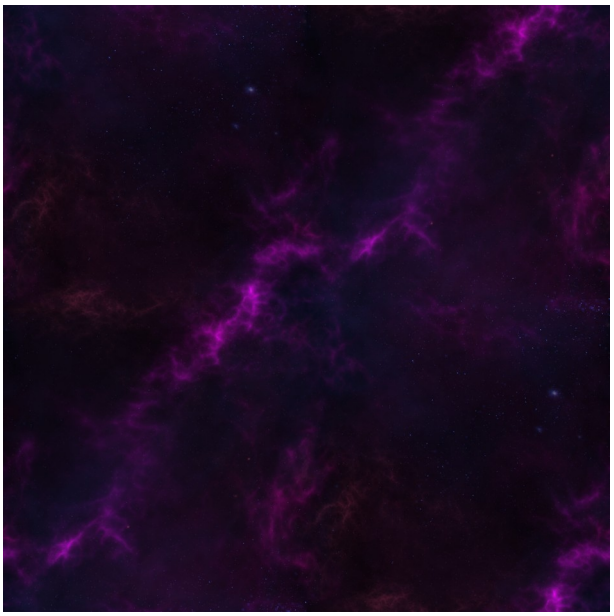
Los disparos son efectuados por las naves enemigas y por la nave del jugador. Cuando un disparo enemigo impacta con la nave del jugador la daña. Lo mismo sucede con los disparos del jugador cuando colisionan con naves enemigas o asteroides. Los disparos del jugador son verticales ascendentes. Los disparos de las naves enemigas son en dirección a donde esté el jugador en ese momento con respecto a la nave que dispara.

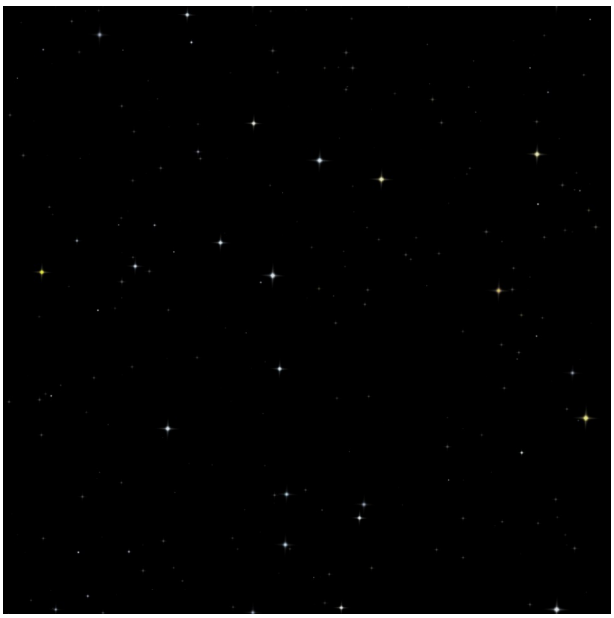
El disparo del jugador puede tener mas potencia y dañar mas a los enemigos conforme se consiguen premios de mejoras de arma. Eso se refleja también en el color del disparo:

Los disparos de las naves enemigas son cada vez más potentes conforme pasan los niveles pero no son diferentes en aspecto:

El fondo

Durante todo el juego hay un fondo estelar a modo decorativo que no influye sobre el juego. Esta formado por un fondo nebuloso que se desplaza verticalmente dando sensación de avance y varias capas de estrellas a diferentes velocidades generando una ilusión de paralaje y profundidad del fondo.





Otros recursos y elementos que forman parte del juego

Ademas de los recursos citados anteriormente en el juego se usan otros como fuentes tipográficas o imágenes de explosiones.

Las fuentes usadas son “COMPUTERRobot” y “Plasmatic”.

La nave del jugador, los premios y las explosiones están compuestos por una sucesión de imágenes que forman animaciones.

Nave:

Explosión de meteorito:

Explosión de nave:

Premio puntos:

Premio salud:

Premio Escudo

Premio potencia de arma:

Vida extra:

Atribuciones

Fuentes TTF freeware:

COMPUTER Robot: Wino S Kadir

Plasmatic: Larabie Fonts

Imágenes:

Naves enemigas (CC BY-SA 3.0): Usuario SCORPIO de opengameart.com

Iconos de premios (royalty free for personal and commercial projects): Razeware LLC

Fondos estelares (CC0 1.0): Brain Studios

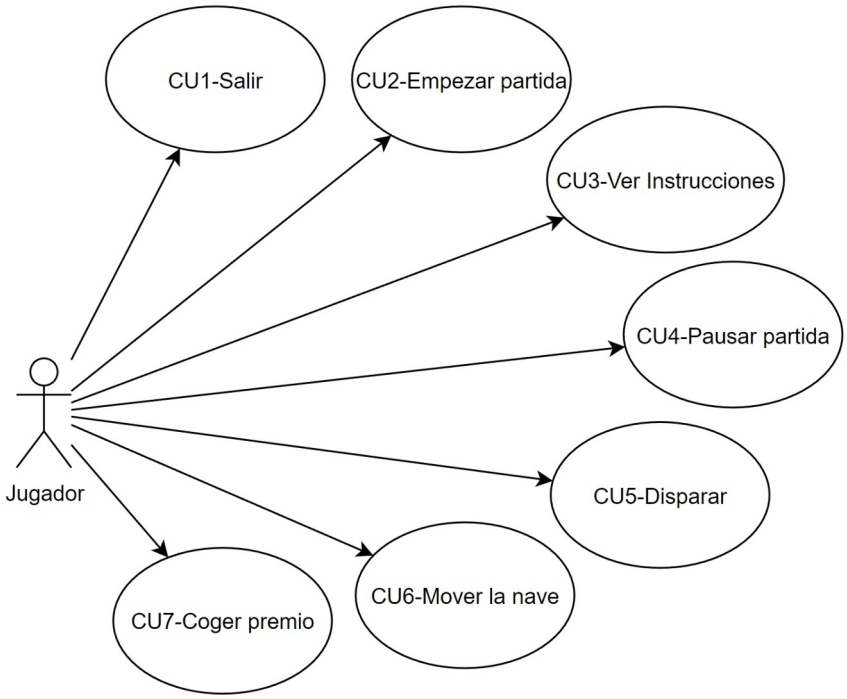
Icono de teclas (Attribution): Vecteezy

Nave del jugador (personal and commercial use): Wobblegut Studio

Disparos del jugador(public domain): Usuario Wenrexa de Itch.io

Explosiones(permitted to use for personal and commercial projects): Craftpix.net

Casos de uso



CU1 - Salir		
Descripción	Salida del programa	
Precondicion	El programa está iniciado	
Secuencia Normal	PASO	ACCIÓN
	1	El usuario va al menu inicial y solicita terminar el programa
	2	El programa termina
Postcondición	El programa se habrá cerrado	

CU2 – Empezar partida		
Descripción	Iniciar una partida	
Precondición	El programa está iniciado	
Secuencia Normal	PASO	ACCIÓN
	1	El usuario va al menú inicial y solicita empezar una partida
	2	La partida empieza
Postcondición	El usuario puede jugar la partida	

CU3 – Ver instrucciones		
Descripción	Ver las instrucciones del juego	
Precondición	El programa está iniciado	
Secuencia Normal	PASO	ACCIÓN
	1	El usuario va al menú inicial y solicita ver las instrucciones
	2	Se muestran las instrucciones
	3	El usuario solicita dejar de ver las instrucciones y se vuelve al menu inicial
Postcondición	El ha podido leer las instrucciones	

CU4 – Pausar partida		
Descripción	Detener una partida temporalmente	
Precondición	Hay una partida en curso	
Secuencia Normal	PASO	ACCIÓN
	1	El usuario solicita pausar la partida
	2	La partida se pausa
	3	El usuario solicita continuar la partida
	4	La partida continúa
Postcondición	Se habrá pausado la partida temporalmente	

CU5 – Disparar		
Descripción	Disparar con la nave durante la partida	
Precondición	Hay una partida en curso	
Secuencia Normal	PASO	ACCIÓN
	1	El jugador acciona el control de disparo
	2	Se genera un disparo que avanza
	3	Si el disparo llega al final de la pantalla desaparece
		Si el disparo impacta sobre una nave enemiga o un meteorito lo daña. Al ser destruido se genera un premio
Postcondición	Se habrá disparado con la nave y provocado las consecuencias del disparo	

CU6 – Mover la nave		
Descripción	Mover la nave durante la partida	
Precondición	Hay una partida en curso y la nave está en pantalla en un estado controlable. No está en mitad de ninguna transición de llegada o salida.	
Secuencia Normal	PASO	ACCIÓN
	1	El jugador acciona el control de movimiento
	2	La nave toma la dirección indicada por el control accionado
	3	Mientras no llegue a los bordes de la pantalla se mueve
		Si llega al borde de la pantalla. Deja de avanzar hacia fuera de la pantalla pero puede seguir avanzando a lo largo del borde.
Postcondición	Se habrá desplazado la nave según las instrucciones del jugador.	

CU7 – Coger premio		
Descripción	Conseguir un premio durante la partida	
Precondición	Hay una partida en curso y hay premios en pantalla	
Secuencia Normal	PASO	ACCIÓN
	1	El jugador mueve la nave hasta la posición del premio
	2	El premio desaparece
	3	Los beneficios del premio son aplicados.
Postcondición	Se habrá aplicado el beneficio del premio.	

Diagramas

Diagrama de herencias

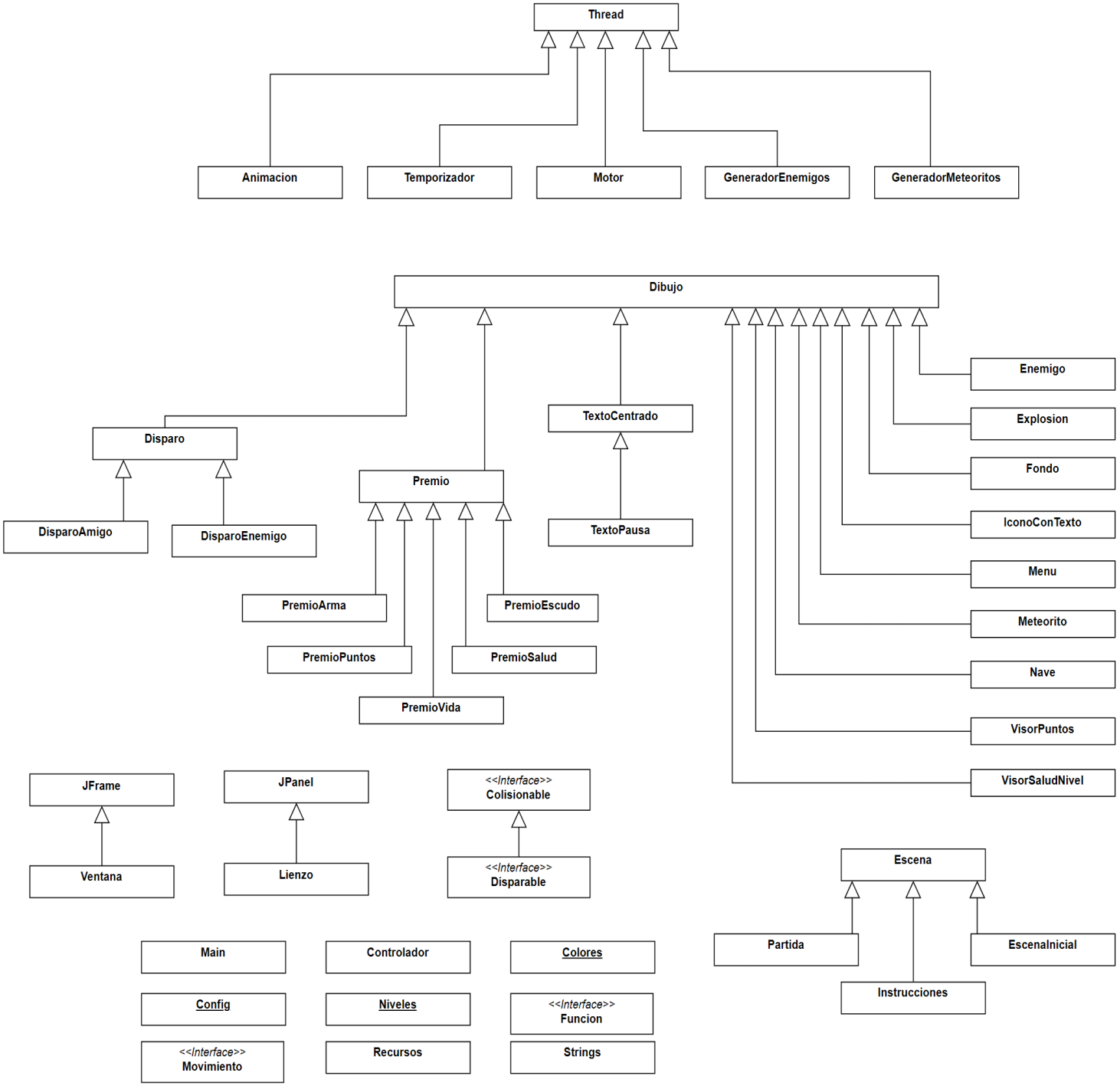
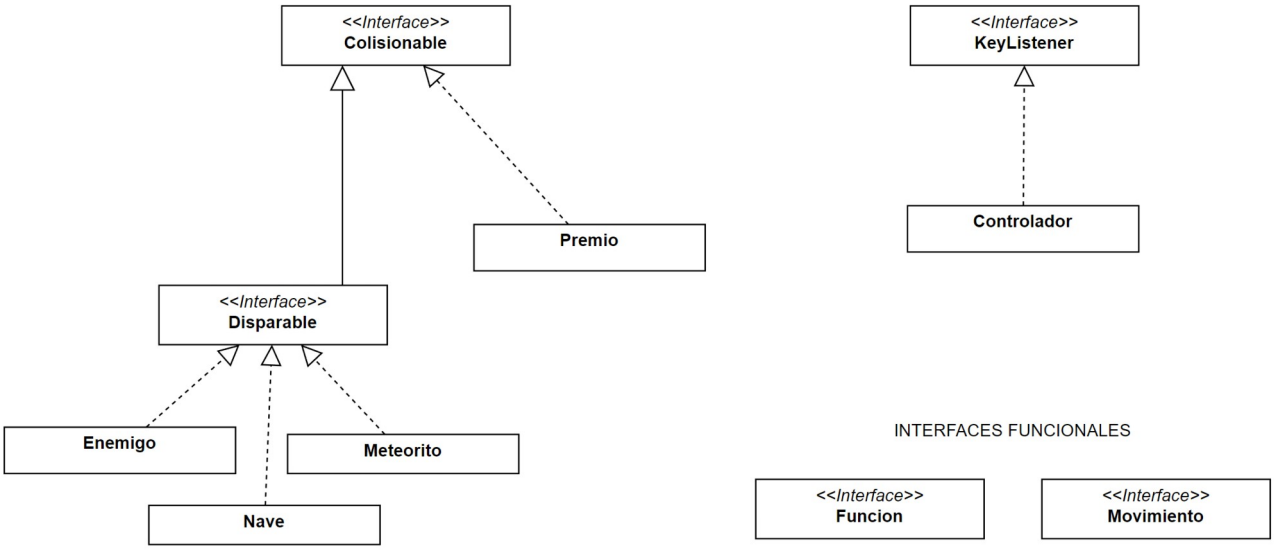


Diagrama de implementación de interfaces

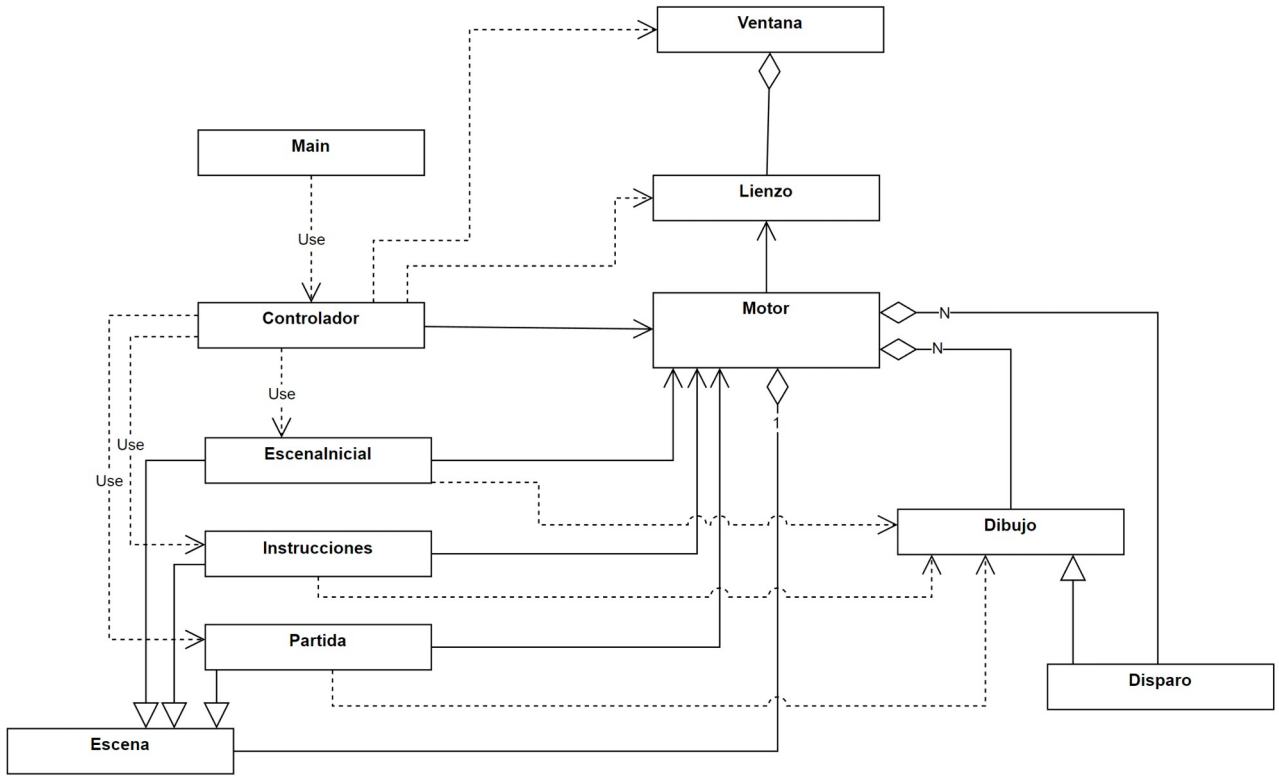


Arquitectura del programa

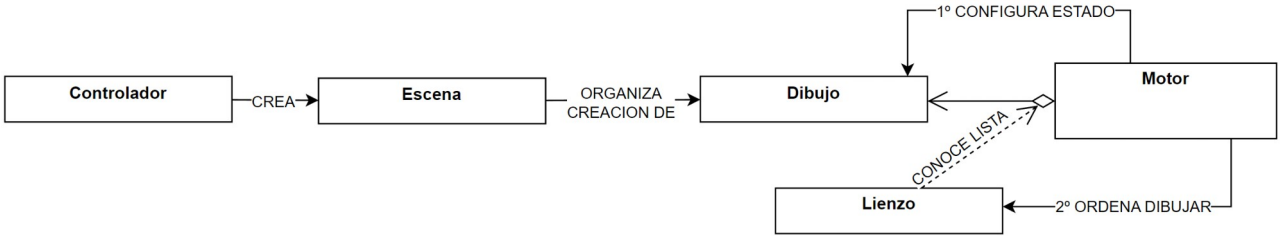
Concepto general de la estructura del programa

Dependiendo de en qué pantalla se esté, el *Controlador* crea un objeto subclase concreta de *Escena* que se encarga de manejar la generación de objetos subclase de *Dibujo* e introducirlos en el objeto *Motor.* Los objetos *Dibujo* son objetos que contienen la manera en que un elemento del juego se dibuja.

El objeto *Motor* tiene listas de objetos *Dibujo* agrupándolos según su uso y orden de dibujado. Además el objeto *Motor* se encarga de ejecutar el bucle de fotograma en el cual primero se calcula el estado de los objetos *Dibujo* y después ordena al objeto *Lienzo(*extiende JPanel*)* que dibuje los objetos *Dibujo.*



Esquema del flujo de creación hasta llegar a la imagen en pantalla:



Motor y el ciclo de un fotograma

La clase *Motor*, que sigue el patrón singleton, es en un hilo con carrera consistente en el ciclo de fotograma.

Motor tiene atributos LinkedBlockingDeque en las cuales almacena los objetos *Dibujo* que forman parte de la escena en curso. Esas listas conforman capas tanto gráficas como lógicas de la escena. Las capas son: fondo, disparos amigos, disparos enemigos, nave del jugador, meteoritos, naves enemigas y premios.

Durante cada iteración del ciclo de fotograma se actúa siguiendo las siguientes fases y terminando cada una antes de pasar a la siguiente:

1. **Comprobar si el motor esta en pausa.** Si esta en pausa el hilo del *Motor* queda esperando a espera de ser notificado para que continúe cuando deje de estar en pausa.
2. **Calcular el tiempo delta** (tiempo transcurrido desde el anterior fotograma). Este tiempo es usado como proporción durante la fase 4 para tener consistencia espacial en las animaciones de los objetos independientemente de los fotogramas por segundo con los que se trabaje.
3. **Limpiar las capas** eliminando los dibujos establecidos como no vivos. Cuando un *Dibujo* debe desaparecer se marca como no vivo para que en esta fase el motor lo elimine. Un pool de hebras se encarga de hacer este trabajo.
4. **Animar los *Dibujos*:** Se recorren todas las capas y usando un pool de hebras se calculan los cambios de posición, rotación, transparencia, fotograma de animación u otro aspecto que necesite cada Dibujo para actualizar su estado.
5. **Detectar las colisiones favorables**(premios, disparos amigos): Un pool de hebras calcula si la nave ha colisionado con objetos de la capa premios o si los objetos de la capa disparos amigos han impactado en objetos de la capas meteoritos y naves enemigas.
6. **Detectar las colisiones desfavorables**: Un pool de hebras calcula si hay colisiones de la nave con objetos de las capas meteorito, naves enemigas o disparos enemigos.
7. **Ordenar a *Lienzo* que ejecute el dibujado** de los elementos de las capas.
8. **Esperar para la limitación de fotogramas** en caso de ser necesario. Al final del bucle hay un sistema para limitar los FPS y que no se sobrepase el especificado en la configuración si el equipo tiene potencia de cálculo como para sobrepasarlo.

Concurrencia en el programa

El programa divide su trabajo en las siguientes hebras:

Hebra inicial: Al iniciar el programa se crea esta hebra. Durante su ejecución se crea el *Controlador* que en su constructor ordena la carga de archivos de recursos. Después inicia el *Motor* lo que provoca que se inicie la hebra del Motor. Seguidamente crea la *Ventana* y con ello el Event Dispatch Thread. Tras ello lanza la escena inicial y este hilo termina.

Event Dispatch Thread: Hebra activa durante todo el programa. La hebra de manejo de eventos de Swing ejecuta 2 acciones principalmente. Ejecutar el repintado cuando la hebra Motor lo ha solicitado, y por otro lado ejecuta la acción vinculada a la pulsación de una tecla. Las acciones vinculadas a la pulsación de teclas son cortas limitándose a alterar estados booleanos de pulsación de teclas en el caso de las partidas o a la modificación de un contador y/o creación de un nuevo objeto escena en otros casos.

Hebra Motor: Activa durante todo el programa. Se encarga de orquestar las fases del ciclo de fotograma. Solo realiza directamente las fases 1,2,7 y 8 y crea los pool de hebras que se encargan de hacer el trabajo en el resto de fases. En las fases con pool de hebras espera la finalización del trabajo del pool antes de pasar a la siguiente fase. Comparte con otras hebras las listas de Dibujos que conforman las capas tanto para lectura como para escritura. Por eso están implementadas con LinkedBlockingDeque. También comparte con el Event Dispatch Thread la variable que determina si el motor está funcionando (atributo booleano play) pero este solo lo usa para lectura y solo es escrito/leido por la Event Dispatch Thread cuando se pulsa una tecla.

Otro atributo que comparte con otras hebras (en su mayoría las hebras de animaciones que describo más adelante) es el atributo AtomicLong tiempo. Este atributo almacena el tiempo de trabajo del motor(descontando los tramos en los que ha estado pausado) y es usado por hebras *Animacion* que se encargan de animar valores de manera autónoma y paralela a la animación general realizada en la fase 4 del ciclo de fotograma.

Pools de hebras: Existen solo durante la fase del ciclo de fotograma en la que trabajan. Son creados durante las fases 3, 4, 5 y 6 del ciclo de fotograma y acceden a las listas LinkedBlockingDeque de objetos *Dibujo* que hay en *Motor*. Modifican parámetros de los objetos *Dibujo* que hay en ellas. En la fase de animación no hay concurrencia de varias hebras sobre el mismo *Dibujo* por lo que los parámetros de animación de los objetos *Dibujo* no tienen control alguno para concurrencia. Sin embargo en las fases de colisiones si se puede dar el caso de que varias hebras intenten modificar la salud de un mismo objeto (por ejemplo una nave colisiona con dos disparos en el mismo fotograma) por lo que ese atributo si está implementado usando un AtomicInteger.

Animaciones autónomas: Para algunos casos se usan objetos *Animacion.* Estos objetos son una hebra que anima un valor a lo largo del tiempo y tras terminar ejecutan un callback especificado con una interfaz funcional. No aplican el valor animado a ningún otro objeto si no que lo tienen como atributo. Otras hebras acceden al valor del atributo en ese momento para recogerlo y aplicarlo donde corresponda. Por tanto los objetos Animaciones son las únicas hebras que se encargan de modificar el valor propio y las otras hebras se encargan de leer el estado actual del valor. Esas otras hebras que

recogen el valor de animación y lo aplican a un caso concreto son las hebras del pool de animación durante la fase 4 del ciclo de fotograma.

Por ejemplo: Cuando la nave del usuario entra en pantalla hay un hilo *Animacion* generando un valor adecuado a la posición que debería tener la nave durante esa animación de entrada. El valor progresa a lo largo del tiempo de A a B. A su vez una hebra del pool de hebras se encarga de actualizar la nave en la fase 4 del ciclo de fotograma y coge el valor de esa *Animacion* y la aplica a la posición Y de la nave.

Estas hebras *Animacion* están pensadas para generar animaciones concretas al margen del control del usuario y pueden ser usadas para coordinar el mismo valor animado para varios objetos. Es el caso de la salida de texto cuando termina la escena de *Instrucciones*. Todas las lineas de texto tienen aplicado el mismo objeto animación el cual es usado en la fase 4 del ciclo de fotograma para determinar la posición X de los textos.

NOTA: Las animaciones de los *Dibujos* son principalmente calculadas directamente por el pool de hebras del punto 4 del ciclo de fotograma. Las animaciones autónomas descritas solo intervienen en casos concretos donde se quiere coordinar varios objetos y/o se quiere ejecutar un callback al terminar la animación. Es el caso de la entrada y salida de textos y la entrada y salida de la pantalla de la nave del jugador.

Temporizador: Activa temporalmente. Utilizada para programar ejecuciones futuras. Consiste en una hebra que duerme durante un tiempo definido en su constructor tras el cual ejecuta la *Funcion* suministrada en el constructor.

Generadores: El último tipo de hebras que intervienen en el programa. Estas hebras son las encargadas de generar los meteoritos y las naves enemigas. Están implementadas en las clases *GeneradorMeteoritos* y *GeneradorEnemigos.* Ambas clases son hebras y tienen una relación de concurrencia con el resto de hebras similar: durante las partidas generan objetos de subclase de *Dibujo* que envían a las LinkedBlockingDeque que hay en *Motor.*

GeneradorMeteoritos es creado una sola vez al inicio de cada partida y su vida se extiende durante una la partida al completo. En intervalos determinados genera objetos *Meteorito* subclase de *Dibujo* y los envía a una lista del *Motor.* Cuando termina la partida la hebra de *GeneradorMeteoritos* termina.

GeneradorEnemigos es una clase de objeto creado varias veces durante la partida. A lo largo del juego llegan oleadas de enemigos. Cada oleada es generada por un objeto *GeneradorEnemigos* que es una hebra que se encarga de ir generando en el tiempo adecuado un *Dibujo* de clase *Enemigo* que envía a *Motor.* Cuando termina de generar la oleada el hilo termina.

Clases principales

Además de la clase *Motor* estas son las principales clases:

Controlador

Se encarga de establecer el estado inicial ordenando la carga de recursos, iniciar la Ventana, iniciar el Motor y cargar la primera Escena.

A partir de ahí sus funciones son:

- Iniciar otra Escena cuando se le solicite. Esto sucede cuando al terminar la Escena actual pide a Controlador que inicie una nueva Escena.
- Terminar el programa cuando se le solicite.
- Es el KeyListener de la *Ventana* y recoge los eventos de teclado para seguidamente enviarlos a la Escena actual. Esa escena es la que decide qué hacer con los eventos de teclas que se reciben. Controlador solo sirve de puente hacia la escena actual.

Escena

Los objetos escena se encargan de manejar cómo y qué objetos Dibujo se generan. También contiene la lógica referente a la escena actual y decide qué acción tomar tras la pulsación de teclas. Hay tres escenas: *EscenaInicial, Instrucciones, Partida.*

EscenaInicial

Esta escena agrega al *Motor* el titulo del juego y un objeto *Menu*. También se encarga de ordenar al menú que cambie su opción seleccionada o que ejecute la selección actual para ir a otra escena según se pulsen las teclas.

Instrucciones

Esta escena agrega al motor objetos *IconosConTexto* a los cuales aplica una *Animacion* en su entrada y tras pulsarse cualquier tecla también a la salida. Al terminar ordena a *Controlador* que inicie la *EscenaInicial.*

Partida

Esta escena se encarga de llevar la lógica de la partida. Lleva un seguimiento de vidas extra, nivel y puntos. En su inicio agrega al *Motor* la *Nave* (a la que aplica una *Animacion* de entrada), elementos de la GUI como el *VisorPuntos* y *VisorSaludNivel* e inicia un *GeneradorMeteoritos.* También inicia el ciclo de niveles generando oleadas de enemigos según indica la configuración especificada en la clase estática *config.Niveles*. A cada oleada crea un *GeneradorEnemigos.* Cuando no quedan vidas extra y la nave es destruida o se ha pasado el último nivel ordena la finalización del *GeneradorMeteoritos* y ordena a *Controlador* que cargue la *EscenaPrincipal.*

Dibujo

De esta clase extienden todos los elementos que se ven en pantalla. Cada subclase agrega el comportamiento que necesita aprovechando la base de la clase *Dibujo*. Esta clase tiene como atributos la posición en X,Y, la velocidad X,Y, la rotacion, la velocidad angular, la opacidad. También tiene tres atributos donde referenciar 3 objetos *Animación* que son opcionales. Cada uno esta destinado a animar, X, Y y opacidad.

Dibujo tiene dos métodos principales: dibujar y nuevoFotograma.

El método *dibujar* es abstracto y debe ser implementado por las subclases definiendo qué dibujar. Este es el método ejecutado por *Lienzo* durante el dibujado en pantalla.

El método *nuevoFotograma* determina el estado del objeto tras pasar un nuevo fotograma. Por defecto la clase *Dibujo* implementa este método aplicando las animaciones autónomas de X,Y y opacidad si existen pero es sobreescrito por las subclases para establecer el comportamiento que necesitan más allá de ese.

Los objetos *Dibujo* tienen un atributo boolean *vivo* que es usado por el *Motor* en la fase de limpieza para eliminar los que tengan este atributo a False.

Interfaces Colisionable, Disparable y clase Disparo

La clase *Disparo* define los parámetros de un disparo, posicion, vector velocidad y fuerza del daño que hace al impactar. Las interfaz *Colisionable* define métodos utilizados durante la detección de colisión entre dos *Colisionable* y la interfaz *Disparable* define un metodo para calcular el impacto de un *Disparo* en un *Disparable.*

La interfaz *Disparable* hereda de *Colisionable.* Estas interfaces son implementadas por los objetos *Nave* (la del jugador), *Meteorito, Enemigo* (naves enemigas) y *Premio*. Los disparos efectuados son de la clase *Disparo.* Los pool de hebras de las fases 5 y 6 del ciclo de fotograma usan estos métodos para calcular las colisiones y daños de los impactos de disparo. Las colisiones son calculadas en 2 fases. Primero se comprueba si las cajas contenedoras de cada *Colisionable* se solapan. De ser así se calcula la distancia entre ambos colisionables y se comprueba si la suma de sus radios es menor o mayor que esa distancia. Para el calculo de los impactos de los disparos se simplifica presuponiendo un radio 0 al disparo.

Nave

Esta clase que hereda de *Dibujo* es la nave del jugador. Su método *dibujar* consiste en dibujar la imagen de la nave que toca de entre sus imagenes. Además el objeto es modificado por la escena *Partida* según el estado de las teclas lo que cambia los atributos de velocidad de la nave. En su método *nuevoFotograma* recalcula su posición usando esos atributos velocidad limitando la posicion cuando se intenta salir de la pantalla.

Cuando *Partida* se lo notifica también ejecuta un disparo agregando un nuevo objeto *Disparo* al *Motor.*

Esta clase implementa la interfaz *Disparable* y el atributo *salud* va modificándose según los impactos. Cuando llega a 0 notifica a *Partida* que ha muerto y esta decide si la partida continua con

una nueva nave o termina. En caso de que muera también crea y envía a *Motor* un objeto de clase *Explosion* que muestra sucesivamente los fotogramas de una explosión y muere.

Meteorito

Los objetos de esta clase, que hereda de *Dibujo,* son generados por *GeneradorMeteoritos* como se describió anteriormente. Implementa la interfaz *Disparable.* Estos objetos tienen una velocidad constante y mueren al ser disparados, si colisionan o salen de la pantalla. Cuando mueren generan un objeto subclase de *Premio* (descrito más adelante) que envían al *Motor*. También genera un objeto de clase *Explosion* que también envía a *Motor.*

Enemigo

Esta clase también hereda de Dibujo e implementa la interfaz *Disparable.* Los objetos e esta clase son generados por *GeneradorEnemigos* como se describió anteriormente. El calculo de su movimiento en el método *nuevoFotograma* puede ser de velocidad vertical lineal o vertical ondulada y están definido en lambda que cumplen la interfaz funcional *Movimiento*. En el caso de movimiento ondulado su posicion X se recalcula cada fotograma con el seno de una proporción de su posición y. Cuando sale de la pantalla o su salud llega a 0 muere.

Los enemigos ademas de salud tienen un numero máximo de disparos que pueden efectuar. Ambos valores se incrementan con los niveles. Cuando *Motor* ejecuta la fase 4 del bucle de fotograma también decide si genera un disparo. De ser así crea un nuevo objeto *Disparo* y lo envía al *Motor.*

Cuando *Enemigo* muere genera un nuevo objeto *Premio* de la subclase *PremioPuntos* y otro de clase *Explosion* que envía a *Motor.*

Premio

Los objetos de la clase *Premio* implementan la interfaz *Colisionable*. Son tenidos en cuenta para la colisión con la nave del jugador. Cuando la nave del jugador colisiona con un *Premio* se ejecuta su método abstracto *aplicarEfecto.* Este método debe ser implementado por las subclases definiendo cual es el efecto de haber cogido el premio.

Las subclases de premio son:

PremioArma: Incrementa el valor del atributo *fuerzaDisparo* de la *Nave.*

PremioEscudo: Activa el escudo de *Nave* que inhabilita las colisiones desfavorables. Tras recogerse se inicia un *Temporizador* que ejecuta con posterioridad la desactivación del escudo.

PremioPuntos: Agrega puntos a la *Escena Partida.*

PremioSalud: Incrementa la salud de *Nave.*

PremioVida: Agrega una vida extra a *Escena Partida.*

Recursos

La clase *Recursos* es la encargada de cargar las imágenes y fuentes tipográficas desde disco. Sigue un patrón singleton y es accedida desde subclases de *Dibujo* para obtener imágenes y fuentes.

Cuando *Controlador* se crea, inicia la carga de recursos. Conforme carga los recuros los va almacenando en unos *HashMap* que posteriormente pueden ser accedidos para recuperar un recurso cargado.

Los archivos de recursos están en la ruta: reto8juego/recursos. Ahí se pueden encontrar dos carpetas una para las imágenes y otra para las fuentes.

Clases de configuración

Las clases de configuración se encuentra en el paquete *reto8juego.config*

Colores: Esta clase contiene definiciones de colores usadas en el programa.

Config: En esta clase hay configuraciones sobre el motor, valores de visualización y sobre la partida.

Niveles: En esta clase se puede configurar la sucesión de oleadas de enemigos de cada nivel o modificar la cantidad de niveles que hay modificando la cantidad de paquetes de oleadas. La información está almacenada en una matriz de tres dimensiones. Los elementos de la primera dimensión se corresponden con la cantidad de niveles que se quiere.

Ejecución del programa

Paquete: reto8juego

Clase main: reto8juego.Main

Compilación: javac reto8juego/Main

Ejecución: java reto8juego/Main

Configuración en: reto8juego.config.Config.java y reto8juego.config.Niveles.java