MEMORIA DE TRABAJO

José Luis Bellosta Manchón
Javier García Rubio

DESCRIPCIÓN DEL JUEGO

El juego que hemos creado es un TBS (Turn-Based Strategy), lo que viene a ser un juego de estrategia por turnos, en este caso es de combates, en el cual tu seleccionas tu personaje (en este caso un alumno) y tienes que ir venciendo a los profesores de las carreras, al ser una carrera está dividida en 4 años y cada año tiene 8 peleas (4 del primer cuatrimestre y otras 4 del segundo).

El objetivo es completar las máximas carreras posibles, pero la dificultad es el enfrentarte a profesores los cuales cada vez son más difíciles y van aprendiendo de tus estrategias en las clases y los exámenes.

Cuantas más carreras tengas más difícil será completar otra, debido a que tienes demasiados conocimientos en tu cabeza y te cuesta más almacenar nuevos datos.

MANUAL DEL JUEGO

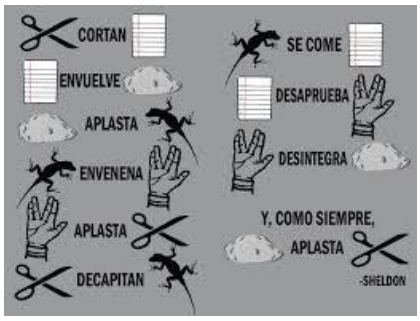
El juego es muy sencillo de jugar. Lo primero de todo es la creación de tu personaje, en el cual le indicas el nombre y la clase que va a ser tu personaje (Informático, científico, pintor o filósofo), cada clase tiene unas ventajas contra otras, según la clase que selecciones se crearan unos valores al azar para el poder (IQ, investigación, creatividad y carisma), cada clase tiene una franja de valores distintos.

La segunda parte es la selección del tipo de carrera a realizar (Informática, física o química, filosofía o arte), dependiendo de tu clase seleccionada y la carrera tendrás un bonificación o penalización dentro de la carrera, ya que es algo que se te puede dar mejor o peor dependiendo de la clase seleccionada.

La última fase es la batalla, como bien se ha indicado antes cada carrera tiene 4 años y cada año 8 batallas (4 por cuatrismestre), por lo tanto, para terminar una carrera hacen falta 32 batallas. Cada 8 batallas, va aumentando el poder de los profesores (y además dependiendo de la especialización del profesor podrás tener una ventaja contra él o una desventaja), debido a que cada año de carrera es más difícil y otra dificultad añadida es en los años 2 y 4 te pueden tocar profesores doblemente especializados por lo tanto tienen más poder que los profesores normales, hay un total de 44 profesores distintos, de los cuales 32 son profesores normales y 12 doblemente especializados. Al comenzar la batalla seleccionas un arma de las 6 disponibles (calculadora, apuntes, boli, chuleta, papel o Nokia3310), cada arma te da una bonificación distinta y una penalización dependiendo lo potente que sea el arma seleccionada.

La pelea contra los profesores está basada en piedra, papel, tijera, lagarto y Spock, en tu caso los ataques a poder realizar son:

1. Levantar la mano (Ataque)
2. Jugar al pc en clase (No hacer nada)
3. Presentar un proyecto (Defender)
4. Mirar apuntes en el examen (Ataque rápido)
5. Dormirse (Cegar)



Con esos ataques tendrás que vencer a los profesores que irán aprendiendo de tu comportamiento en clase. Siempre podrás ver las stats de tu personaje mientras estés en la batalla.

Al terminar la fase 3 y completar una carrera vuelves a la fase dos, así en bucle hasta perder, recordando que cada carrera va a ser más difícil que la anterior.

EASTER EGGS DEL JUEGO

El juego contiene un total de 8 Easter eggs, los cuales 2 son en la creación de personaje, 5 cuando pierdes contra determinados profesores y 1 cuando ganas a un determinado profesor.

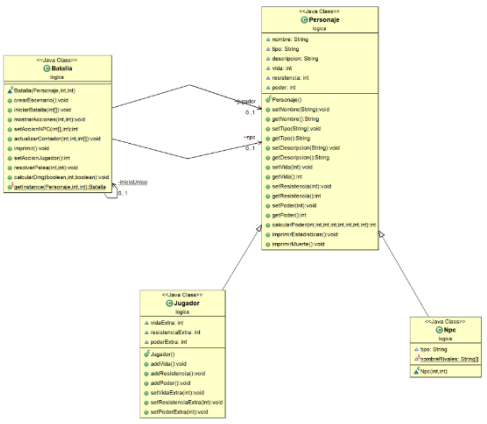
¿Cuántos eres capaz de encontrar?

PATRONES

Para mayor claridad de las imágenes en el archivo del programa hay un diagrama específico para cada clase explicada a continuación.

1. (1 punto) Se deberá utilizar el patrón Strategy para gestionar el conjunto de estrategias posibles y la estrategia que en cada combate adopte el enemigo de turno.

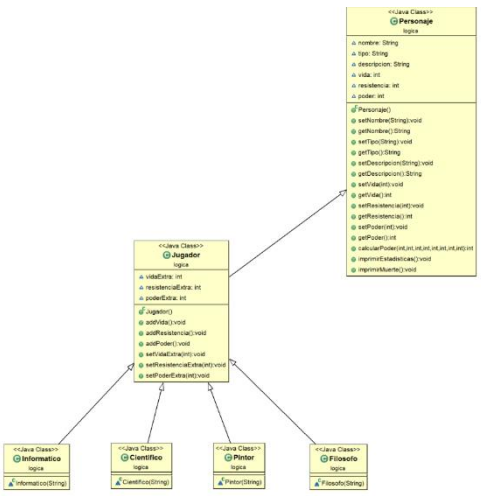
El patrón Estrategia (Strategy) se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.



El patrón se cumple en las clases **Jugador** y **Npc** que heredan de **Personaje** permitiendo así crear un Npc o un Jugador dependiendo de qué tipo de Personaje queramos crear según nos convenga.

2. (1 punto) Se deberá emplear un patrón Decorator para gestionar las distintas acciones a realizar por los personajes: se decorará la acción básica con los modificadores adecuados dependiendo de la estrategia y las características del personaje. Por ejemplo, si el personaje realiza un 'ataque con espada', este ataque habrá que decorarlo para hacerlo un 'ataque poderoso con espada' si el personaje tiene un alto valor de su atributo Fuerza.

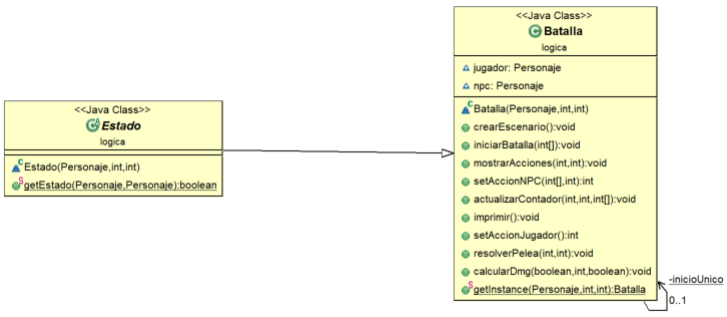
El patrón Decorator responde a la necesidad de añadir dinámicamente funcionalidad a un Objeto. Esto nos permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.



El patrón se cumple en las clases **Informático**, **Científico**, **Pintor** y **Filosofo** que heredan de **Jugador** y este a su vez de **Personaje** permitiendo así que la clase Jugador tenga diferentes características simplemente asociándose con las que le siguen.

3. (1 punto) Se deberá emplear un patrón State para controlar el estado de los personajes durante el combate, de forma que los personajes vayan pasando de un estado a otro dependiendo de las incidencias del combate.

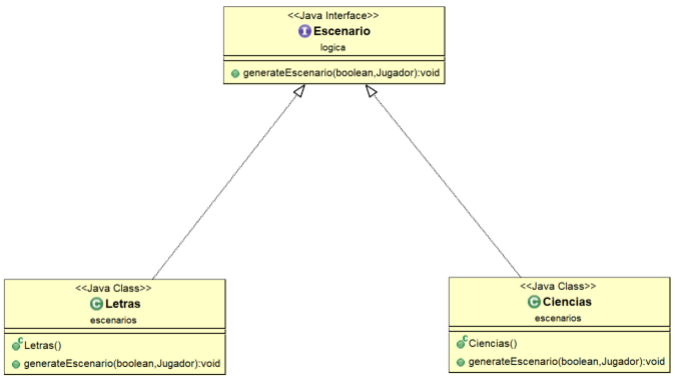
El patrón de diseño State se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo. Por ejemplo: una alarma puede tener diferentes estados, como desactivada, activada, en configuración. Definimos una interfaz Estado_Alarma, y luego definimos los diferentes estados.



Como indica el propio nombre de la clase **Estado** cumple la funcionalidad del patrón State estableciendo el estado en el que se encuentran las instancias jugador o npc en la clase **Batalla**.

4. (2 puntos) Se deberá emplear un patrón Abstract Factory para crear los diferentes enemigos. Existirán varios tipos de enemigos, comunes a todos los mundos/escenarios, pero adaptados a cada mundo/escenario.

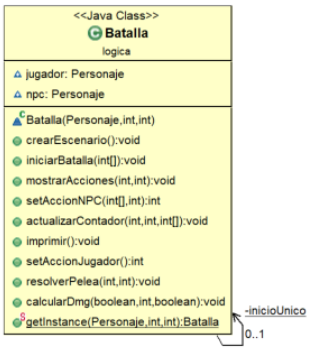
El patrón Abstract Factory nos permite crear, mediante una interfaz, conjuntos o familias de objetos (denominados productos) que dependen mutuamente y todo esto sin especificar cuál es el objeto concreto.



La interfaz **Escenario** nos permite crear, como su propio nombre indica, un escenario de tipo **Letras** o **Ciencias**. Dependiendo de que tipo de escenario queremos generar utilizaremos una clase u otra cumpliendo así con la condición del patrón.

5. (1 punto) Se deberá emplear un patrón Singleton en la clase encargada de hacer los cálculos sobre el resultado de los ataques, para que sólo haya una instancia de dicho 'calculador' en el sistema.

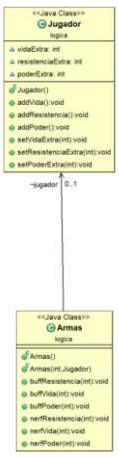
El patron Singleton o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.



Podemos observar que la clase **Batalla** tiene un método getInstance() que restringe el constructor de dicha clase para que solo pueda existir un objeto Batalla simultáneamente.

6. (1 punto) Se deberá emplear un patrón Template Method para implementar en cada clase Enemigo el algoritmo para decidir cuál será la siguiente acción para realizar.

El template method es un patrón de diseño de comportamiento que define el esqueleto de programa de un algoritmo en un método, llamado método de plantilla, el cual difiere algunos pasos a las subclases. Permite redefinir ciertos pasos seguros de un algoritmo sin cambiar la estructura del algoritmo.



Una de las características del Template Method es que la subclase pueda modificar ciertos parámetros de la clase principal sin cambiar la estructura de este. Como podemos observar la clase **Armas** tiene varios métodos tales como buffVida(), nerfVida(), buffPoder(),… que modifican variables de la clase **Jugador** sin modificar nada de su estructura cumpliendo así con las características del patrón.