Proyecto 1

Java

Generación 33

Versión básica en consola de texto de un juego de batallas

Flujo del programa

El juego consiste en un combate entre 2 contrincantes que estará definido por turnos.

Cada uno de los contrincantes cuenta con 6 monstruos distintos que se enfrentarán. Dichos monstruos pelearán de uno en uno en la arena de combate y en cada turno los jugadores les darán órdenes.

El flujo de un turno es el siguiente:

- El programa imprime información de los dos monstruos combatientes en pantalla (nombre, puntos de vida y nivel).

- Contrincante 1 elige darle una orden a su monstruo (representado en el programa como un menú de consola).

- Contrincante 2 elige darle una orden a su monstruo.

- El monstruo más rápido ataca primero, y luego el otro. Los ataques mandados a otro monstruo también pueden fallar, tienen un 20% de probabilidad de fallo (1 en 5).

- El programa imprime los resultados de los ataques de los monstruos o de las órdenes recibidas por sus jugador.

Todos los monstruos pertenecen a un tipo elemental (Agua, Fuego, Hierba y Eléctrico) que determina a qué ataques son débiles y a cuáles son fuertes, y tienen dos movimientos de ataque diferentes. Estos monstruos también poseen un nivel, el cual va a determinar qué tan fuertes son.

Los contrincantes también pueden elegir en vez de atacar dos opciones:

1. Cambiar de monstruo que tienen en batalla por otro disponible en su equipo.
2. Usar un tipo pócima que mejorará las características del monstruo.

Si un contrincante realiza alguna de esas acciones, el contrario debe atacar a su monstruo sin que este pueda responder.

Si los puntos de vida de un monstruo son reducidos a 0, este es inhabilitado y su propietario debe mandar a otro monstruo de su equipo para tomar su lugar. Un contrincante conseguirá la victoria cuando logré derrotar a todos los monstruos de su contrincante.

Funcionalidad de clases

Clase Contrincante:

Clase que va a representar a un contrincante. Un contrincante tiene un nombre y dos ArrayLists, uno que contiene a su equipo de monstruos, y otro que contiene las pócimas.

Los métodos son los siguientes:

- guardarMonstruo - Se usa cuando un jugador toma al monstruo que tiene en batalla y lo saca de combate para guardarlo (en su ArrayList equipo). Esto pasa cuando el jugador cambia de monstruo al ser derrotado otro y tiene que guardarlo.

- elegirMonstruo - Este método se usa cuando un contrincante da la orden de cambiar monstruo a la mitad de la batalla o cuando uno de sus monstruos es derrotado.

- usarPocima - Este método se usa cuando un jugador saca una pócima y la usa en su monstruo en batalla. Después de ser usada la pócima se desecha.

- listarMonstruo - Método que se usa en el menú para listar el equipo de un jugador. Se muestra de cada monstruo nombre, nivel, puntos de vida y estado.

Clase Pocima

La clase Pocima es una clase abstracta que representa las características básicas de una pócima usable en juego. Sólo solicita que sus clases herederas implementen el método de usar el objeto en un monstruo.

Las clases herederas de Pocima son tres:

PocimaVida: Recupera en un 20% los puntos de vida del monstruo.

PocimaAtaque: Incrementa en un 10% el ataque de un monstruo en batalla.

PocimaDefensa: Incrementa en un 15% la defensa de un monstruo en batalla.

Al empezar el duelo, cada jugador tendrá 2 PocimaVida, 2 PocimaAtaque y 2 PocimaDefensa.

Clases Monstruo

Clase abstracta que representa las características y comportamientos generales de un monstruo.

Los atributos de esta clase son los siguientes:

- hp: Puntos de vida de un monstruo, si llega a 0, el monstruo es derrotado.

- apodo: Opcionalmente, un monstruo puede tener un apodo.

- nivel: Número entero de 1 al 100 que va a determinar que tanta experiencia tiene un monstruo. Se usa en los cálculos para establecer el ataque, defensa, velocidad y hp de un monstruo.

- ataque: Que tanta fuerza tiene un monstruo. Usado para calcular el daño de sus ataques a otro monstruo.

- defensa: Que tanto daño puede absorber un monstruo. Usado para calcular la cantidad de daño que se puede anular de un ataque de un monstruo enemigo.

- velocidad: Determina en un turno qué monstruo ataca primero.

- estado: String que describe el estado de un monstruo. Si es "ok", el monstruo está bien, si es "fuera de combate" el monstruo ya no puede pelear, si es "paralizado" el monstruo no se puede mover, entre otros.

Los métodos de esta clase son:

- recibirDaño: Este método determina que si un monstruo llega a 0 hp, muestre "el monstruo está fuera de combate" y ponga su atributo estado a "fuera de combate".

- recibirHp: Método que sólo se usa para incrementar el hp de un monstruo. Usado por pócimas que recuperen hp.

- recibirAtaque: Método que sólo se usa para incrementar el ataque de un monstruo. Usado por pócimas que auementen ataque.

- recibirDefensa: Método que sólo se usa para incrementar la defensa de un monstruo. Usado por pócimas que recuperen defensa.

- getters y setters para ataque, defensa y velocidad.

- multiplicadorElemental: Método abstracto que pide como es que las clases herederas van a definir multiplicadores de daño para ataques elementales (explicado más adelante)

- ataque1: Método abstracto para pedir que las clases herederas definan un primer movimiento de ataque.

- ataque2: Método abstracto para pedir que las clases herederas definan un segundo movimiento de ataque.

Clases Agua, Fuego, Hierba, Eléctrico y sus subclases.

Las clases siguientes heredan de Monstruo que se les denomina "clases elementales", y son clases abstractas que determinan a qué elemento pertenece un monstruo, sólo serán: Agua, Fuego, Hierba y Eléctrico.

En esta clase se definen dos métodos pedidos por la clase Monstruo:

- multiplicadorElemental: El juego contiene la mecánica que si un monstruo realiza un ataque de un cierto elemento, el monstruo que recibe ese ataque va a variar su daño dependiendo al elemento al

que pertenezca éste: Si un monstruo fuego recibe un ataque de agua, le va a hacer mucho daño, pero recibe uno de tipo hierba, apenas lo va a sentir.

En este programa hacemos la suposición que si un monstruo pertenece a un elemento, todos sus ataques van a ser de dicho elemento. Este multiplicador de daño funciona de la siguiente manera:

- Ataque de fuego:

    x2 de daño a monstruo tipo Hierba, x1/2 a monstruo tipo Agua o Fuego, x1 a otro tipo.

- Ataque de agua:

    x2 de daño a monstruo tipo Fuego, x1/2 a monstruo tipo Agua o Hierba, x1 a otro tipo.

- Ataque de hierba:

    x2 de daño a monstruo tipo Agua, x1/2 a monstruo tipo Hierba o Fuego, x1 a otro tipo.

- Ataque de eléctrico:

    x2 de daño a monstruo tipo Agua, x1/2 a monstruo tipo Hierba o Eléctrico, x1 a otro tipo.

- ataque1: Primer movimiento de ataque de un monstruo. Todos los monstruos que pertenezcan a un elemento en particular, tendrán ese ataque. Este ataque obligatoriamente funcionará de la siguiente manera:

Daño inflingido a monstruo defensor = (MonstruoAtacante.Ataque - MonstruoDefensor.Defensa) * (Multiplicador elemental)

Si la diferencia del ataque y defensa es 0 o menor, el monstruo defensor no recibe daño.

Cuando se realiza este movimiento, se debe mostrar en pantalla un mensaje con el nombre del monstruo que realiza el ataque y el nombre del movimiento (es de su invención, pero debe corresponder con el tipo de elemento).

Dicho ataque tiene 20% de probabilidad de fallo.

Clases de monstruos específicos

Estas clases heredan de las clases elementales y representan a los monstruos que pueden usarse en el juego. Por cada clase elemental deben haber 3 clases herederas, dando un total de 12 monstruos distintos. Esto es para que cada contrincante tenga 6 monstruos distintos.

Estas clases van a tener dos características importantes: La definición del segundo ataque y la implementación del constructor.

- ataque2: El segundo movimiento de ataque de un monstruo que va a ser único para cada clase de monstruo y debe ser distinto en uso al ataque1. Este ataque puede ser una variación del primer ataque, pero que haga más daño, o que en vez de realizar un ataque, el monstruo trate de paralizar al otro.

Independiente del movimiento que hagan, este movimiento también tendrá 20% de probabilidad de fallo.

Constructor del monstruo

Para crear un objeto monstruo y ponerlo en el equipo del jugador, lo único que se va a pedir son apodo del monstruo y su nivel.

Si un jugador no quiere ponerle apodo al monstruo, en este parámetro debe pasar una cadena vacía, a lo cual el constructor asigna el nombre predeterminado de la clase del monstruo (escrito en la constante "nombreMonstruo" de esta clase).

El parámetro nivel se va a pasar para hacer los siguientes cálculos para determinar el ataque, defensa, hp y velocidad de un monstruo:

ataque (de clase Monstruo) = nivel*ataqueBase (constante de la clase del monstruo especifico)

defensa (de clase Monstruo) = nivel*defensaBase (constante de la clase del monstruo especifico)

velocidad (de clase Monstruo) = nivel*velocidadBase (constante de la clase del monstruo especifico)

hp (de clase Monstruo) = nivel*hpBase (constante de la clase del mosntruo específico)

Las constantes base van a ser distintas para cada clase de Monstruo, se recomienda que se manejen los siguientes valores:

hpBase : 15 - 25

ataqueBase: 10 - 20 (establecer valores más altos que defensaBase)

defensaBase: 10 - 20

velocidadbase: 10 - 20

En este constructor también se inicializa el estado del monstruo.

Especificaciones adicionales

El manejo del menú, las batallas y la interacción entre jugadores y monstruos debe hacerse por medio de una clase Torneo. También se pide que todas las clases de Monstruo y sus derivadas están en un solo paquete llamado "equipo". En la implementación de los métodos descrita pueden hacerles cambios ligeros, o agregar métodos que ustedes consideren de ayuda mientras que no alteren los atributos de las clases, la estructura de las clases o el funcionamiento especificado del programa.

Entrega y aspectos a evaluar

La entrega se realizará a mi correo personal con lo siguiente en un archivo comprimido:

- Archivos .java y paquetes
- Documentación en javadoc
- Diagrama de clases en UML
- Guía de uso con impresión de pantallas

Se le preguntará a cada integrante la funcionalidad y lógica del código, me tendrán que mostrar el uso de git y un repositorio remoto (github o bitbucket) en su proyecto, una vez revisado personalmente a cada equipo se enviará a mi correo.
Tomaré en cuenta también la originalidad, que sea óptimo el código y la buena identación y convención de nombres.

Recuerden que los programas deben de ser únicos entre los equipos. Si no se presenta el día de entrega se reprobará el curso.

La fecha de entrega será el sábado 15 de Octubre después del curso de C- Linux

“Hard work beats talent when talent doesn’t work hard” - Tim Notke