

blindg sol.pdf



Damaga



Metodología de la Programación



1º Grado en Ingeniería Informática en Tecnologías de la Información



**Escuela Politécnica de Ingeniería de Gijón
Universidad de Oviedo**



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.




ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN
METODOLOGÍA DE LA PROGRAMACIÓN

Control Práctico de Mayo. Vienes 13 de Abril de 2018.

Introducimos un nuevo tipo de monstruo el “guardian ciego”. Realmente es un tipo de momia ciega que guarda la mazmorra (desde ahora **BlindGuardian**). Al no poder ver, se guía por el ruido que hacen el resto de personajes. Se siente atraída por el ruido y allí es donde se dirigirá cuando se mueva.

Se pide:

1. **[0,5 puntos]** Define un monstruo nuevo, el **BlindGuardian**, que se diferencia del resto de momias sólo en su comportamiento, tal y como se detallará más adelante. Su icono es:  y su representación como carácter es la “m” minúscula.

*Creamos la clase **BlindGuardian** que hereda de la clase **Momia**. Implementamos la interface **Ficha** reescribiendo el método **toChar()** y, la interface **ElementoGrafico** sobrescribiendo **getImagen()**.*

2. **[1,5 puntos]** Algunos tipos de personajes son considerados **Ruidosos**. Aunque en cada versión del juego pueden variar, en la versión actual se consideran ruidosos a los héroes. Los personajes ruidosos se caracterizan por tener los siguientes métodos:

- **int getRuidoAcumulado()**: retorna el nivel de ruido acumulado del personaje.
- **void addRuidoMovimiento(int n)**: añade al nivel actual de ruido acumulado del personaje el ruido de moverse **n** casillas.
- **void addRuidoCombate()**: añade al nivel actual de ruido acumulado del personaje el ruido de combatir, da igual que esté defendiendo o atacando.
- **void resetRuido()**: fija a cero el nivel de ruido acumulado del personaje.

*Debemos crear la Interface **Ruidoso** con los prototipos de los métodos indicados. La interface debe ser implementada por la clase **Heroe**.*

3. **[2 puntos]** Un personaje Ruidoso comienza la partida con un nivel 0 de ruido acumulado y se va incrementando según realiza acciones:

*La clase **Heroe** necesita un atributo **protected int ruidoAcumulado** para registrar el ruido y poder modificarlo en las subclases (Nota: también sería válido declararlo **private** y crear un método **setRuidoAcumulado()** **protected**). Se inicializa en el constructor con valor cero o llamando a **resetRuido()**. Se implementa **Heroe.resetRuido()** que fija a cero el atributo anterior, y **Heroe.getRuidoAcumulado()** que retorna el valor actual de dicho atributo.*

- **Al moverse una casilla**: los enanos incrementan en 1 unidad su nivel de ruido acumulado, los bárbaros no incrementan su ruido acumulado (no hacen ruido al moverse).

*La clase **Heroe** sobrescribe **accionMovimiento()** invocando a **x = super.accionMovimiento()** e invocando a **addRuidoMovimiento()** con el valor **x** retornado por la llamada anterior.*

*En la clase **Enano** se sobrescribe **addRuidoMovimiento(int n)** para que incremente en cantidad **n** el ruido acumulado. En la clase **Barbaro** se hace de forma similar a **Enano**, pero el método realmente no incrementa el ruido.*

- **Al combatir (tanto como atacante como defensor)**: los enanos incrementan en 5 unidades su nivel de ruido acumulado y los bárbaros 7 unidades (gritan más que los enanos).

*En la clase **Heroe** sobrescribimos **combatir()**, invocando en el método a **super.combatir()** y a **this.addRuidoCombate()**. Para la defensa modificamos **Heroe.defender()** e incluimos la llamada a*

this.addRuidoCombate(). (Nota: Alternativamente también admite identificar en *Heroe.combatir()* que el defensor *p* es *Ruidoso*, en cuyo caso invocamos a *p.addRuidoCombate()*)

En la clase *Enano* sobrescribimos el método *addRuidoCombate()* de forma que incremente el ruido en 5 unidades. En la clase *Barbaro* se haría lo mismo, pero incrementando el ruido en 7 unidades.

NOTA: Se recomienda actualizar el método *toString()* en las clases de personajes ruidosos para que retornen también el nivel de ruido acumulado. Ej: "...cuerpo:2/7) [Ruido: 3]"

4. **[1 punto]** Los *BlindGuardian* tienen como enemigo a los personajes ruidosos, además de los que tendría cualquier otro monstruo.

Se sobrescribe *BlindGuardian.esEnemigo(Personaje p)*, en él se debe retornar la expresión:
(p instanceof Ruidoso) || super.esEnemigo()

5. En su turno el *BlindGuardian*:

- i. **[0,5 puntos]** Primero se mueve y luego combate.

Se sobrescribe *BlindGuardian.resuelveTurno()* invocando primero *accionMovimiento()* y después *accionCombate()*.

- ii. **[3,5 puntos]** Al moverse se dirige siempre hacia el personaje que más ruido acumulado tenga, y si lo alcanza finalizará su movimiento. Consideramos que lo alcanza cuando la distancia euclídea entre sus casillas es igual a 1. Si no hay personajes ruidosos en el tablero el *BlindGuardian* no se mueve.

Para esta tarea definiremos dos métodos auxiliares:

- **Personaje masRuidoso(Personaje[] pjs):** retorna el personaje más ruidoso vivo del array *pjs*, o null tanto si ninguno de los personajes vivos son ruidosos como si el más ruidoso tiene *ruidoAcumulado=0* (es decir, que no hizo ruido).

Primero debemos localizar el primer personaje de *pjs* que esté vivo y sea *Ruidoso* (via *instanceof*), para a continuación aplicar el algoritmo de cálculo del máximo del resto de elementos. Si ningún personaje del vector es *Ruidoso* o el nivel del más ruidoso es 0, entonces se retorna *null*, sino se retorna el personaje más ruidoso encontrado.

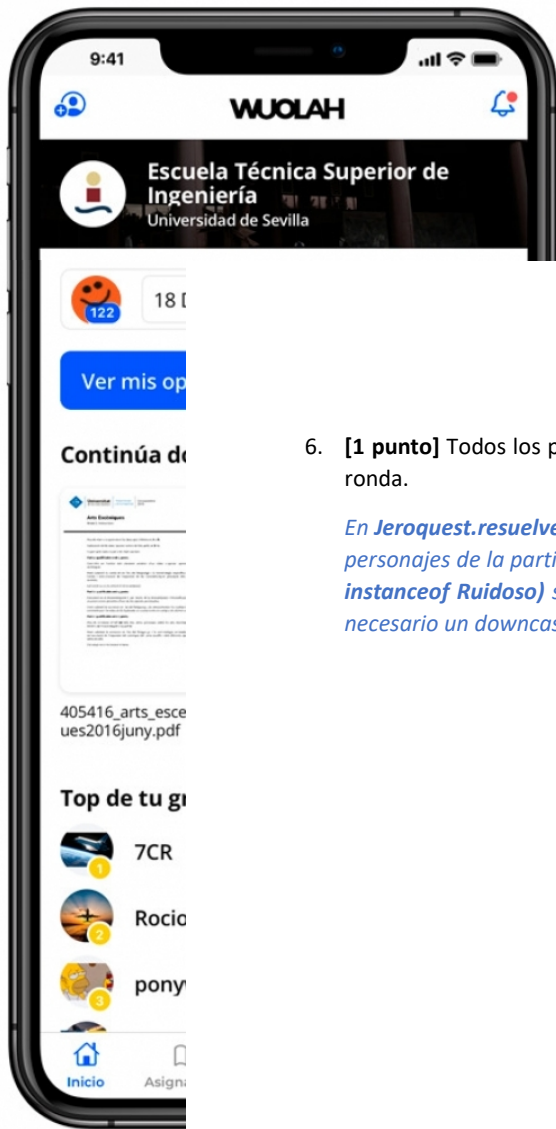
- **XYLocation masCercana(VectorDinamicoXYLocation v, Personaje p):** retorna de todas las posiciones del vector *v*, la más cercana a la posición del personaje *p*.

Aplicamos directamente el algoritmo de cálculo del mínimo de un vector, en este caso será aquel cuya distancia a *p* sea la más pequeña.

Nota: Es conveniente revisar las clases antes de definir nuevos métodos, la clase *XYLocation* dispone del método estático *distancia()* para calcular la distancia euclídea.

Tras moverse, atacará a cualquier enemigo que tenga accesible, como cualquier otro monstruo.

En *BlindGuardian* sobrescribimos *accionMovimiento()*, el código es similar al heredado, simplemente fijamos, al principio del método, como objetivo del movimiento aquel personaje que retorne *this.masRuidoso(partidaActual.getPersonajes())*. Si retorna *null*, directamente finalizamos el método retornando 0 (casillas movidas). En otro caso reutilizamos el bucle de movimiento de *Personaje*, simplemente cambiando que al escoger la siguiente casilla del movimiento utilizamos *masCercana(casillasValidas)*, en vez de tirar un dado para escoger al azar. Y además, que a la condición del bucle *while* se añade que la distancia al objetivo ha de ser mayor a 1.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



6. [1 punto] Todos los personajes ruidosos vivos reinician a cero su nivel de ruido acumulado al inicio de cada ronda.

En `Jeroquest.resuelveRonda()` antes del bucle de resolución de turnos ejecutamos otro similar que recorra los personajes de la partida, y compruebe para cada personaje `p` del vector que si se cumple `(p.estaVivo())` && `(p instanceof Ruidoso)` se resetee su ruido acumulado vía el método `resetRuido()`, para ello previamente será necesario un downcast: `((Ruidoso)p).resetRuido()`.