

Cuervo_Solucion.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.



Introducimos un nuevo tipo de personaje el “cuervo”. Este personaje, que no es ni héroe ni monstruo, se dedica a robar las monedas del resto de personajes y escapar con ellas fuera del tablero. Por lo tanto, nos encontramos con que varios tipos de personajes actúan como “tesoreros” al poder llevar encima un número de monedas.

Se pide:

1. **[0,75 puntos]** Define un personaje nuevo, el **Cuervo**, que se caracteriza por tener los siguientes atributos

iniciales (MOVIMIENTO = 7, ATAQUE = 0, DEFENSA = 1, CUERPO = 1). Su icono es:  y su representación como carácter es la “c” minúscula.

Los cuervos no se defienden de los ataques, es decir, todos los impactos se convierten en heridas.

Creamos una nueva clase Cuervo que hereda de la clase Personaje, con las constantes referidas a los valores iniciales de sus atributos. Su constructor recibe simplemente el nombre del Cuervo como String, y se apoya en el constructor de Personaje vía super(). Para ser una clase concreta es necesario implementar todos los métodos abstractos heredados, concretamente Personaje.defender(). Este método se limitará a convertir los impactos en heridas que deben reducir los puntos de cuerpo del cuervo.

2. **[1,5 puntos]** Algunos tipos de personajes se comportan como **Tesoreros**. Aunque en cada versión del juego pueden variar, en la versión actual se consideran tesoreros a los héroes y los cuervos. Los personajes tesoreros se caracterizan por tener los siguientes métodos:

- `int getMonedas()`: retorna el total de monedas que tiene el personaje.
- `void setMonedas(int monedas)`: fija el número de monedas que tiene el personaje
- **void** `transfiereTodo(Tesorero p)`; todas las monedas del tesorero `p` pasan al tesorero actual.

Debemos crear la Interface Tesorero con los prototipos de los métodos indicados. La interface debe ser implementada por la clase Heroe y Cuervo. En la implementación de transfiereTodo() debemos recordar que estamos quitándole el dinero a `p`: `this.setMonedas(this.getMonedas()+p.getMonedas()); p.setMonedas(0);` NOTA: Es fundamental recordar a partir de este punto qué si hablamos de personajes con dinero, no podemos suponer que sean héroes o cuervos, solo podemos interpretar que son Tesoreros. Ya que además nos indican que en futuras versiones del juego los tipos de Personaje que son Tesorero pueden variar, y no queremos que eso implique modificar el código donde se utilicen “personajes con dinero”.

3. **[0,5 puntos]** Los cuervos comienzan la partida con cero monedas. Los héroes por su parte comienzan cada uno de ellos con un número aleatorio entre 10 y 100 monedas.

Tanto Heroe, como Cuervo, tienen un atributo privado o protegido `int monedas`, que debe iniciarse en el constructor de la clase con valor 0, en el caso de Cuervo, y con `9+Dado.tira(91)` en el caso de Heroe.

4. **[1 punto]** Si en el combate los héroes eliminan a un tesorero, se quedan con todo su dinero.

Se sobrescribe Heroe.combatir() que primero llama a super.combatir(), ya que el combate no varía, y a continuación comprueba si el objetivo es Tesorero y está muerto para saquear todo su dinero:

If ((objetivo instanceof Tesorero)&&(!objetivo.estaVivo())) this.transfiereTodo((Tesorero)objetivo);

Es necesario el downcasting para usar objetivo como argumento de transfiereTodo().

5. **[1 punto]** Los Cuervos tienen como enemigo a cualquiera que posea al menos una moneda y que no sea otro Cuervo.

*Sobrescribimos Cuervo.esEnemigo(Personaje p), donde debemos mirar que el objetivo no sea Cuervo: **!(p instanceof Cuervo)**, que tenga monedas implica que debe ser Tesorero para poder acceder al método `getMonedas()`: **(p instanceof Cuervo) && ((Tesorero)p).getMonedas() > 0***

6. En su turno el Cuervo:

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

Se sobrescribe **Cuervo.resuelveTurno()** invocando primero **accionMovimiento()** y después **accionCombatir()**.

- ii. **[2,5 puntos]** La acción de movimiento de un cuervo depende de si ya ha robado dinero o no:

- Si no tiene dinero: Se moverá hacia el personaje más rico.
- Si tiene dinero tratará de escapar, es decir, se mueve hacia la **casilla de salida**, la que tiene coordenadas (0,0), y se detiene si la alcanza.

Se sobrescribe **Cuervo.accionMovimiento()**, el código es similar al heredado, simplemente cambia que a la hora de escoger la casilla **destino** a la que moverse:

- Si no tiene dinero será aquella de las posibles que le acerque más al personaje más rico:
Personaje objetivo = masRico(partidaActual.getPersonajes());
destino = masCercana(posicionesValidas, objetivo.getPosicion());
Debemos tener la precaución de que **objetivo** puede ser **null**, en cuyo caso ya no nos debemos mover más.
- Si tiene dinero será aquella de las posibles que le acerque más a la casilla de salida.
destino = masCercana(posicionesValidas, new XYLocation(0,0));
Debemos tener la precaución de que la posición actual ya sea la (0,0) en cuyo caso ya no nos debemos mover más.

Para ello se definirán dos métodos auxiliares en Cuervo:

- **XYLocation masCercana(VectorDinamicoXYLocation v, XYLocation destino):** retorna de todas las posiciones del vector **v**, la más cercana a la casilla de coordenadas **destino**.

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.

- **Personaje masRico(Personaje[] pjs):** retorna el personaje vivo, **enemigo**, con más monedas del array **pjs**, o **null** si ningún personaje lleva monedas o su bolsa está vacía.

Aplicamos directamente el algoritmo de cálculo del máximo de un vector, en este caso será aquel que siendo **Tesorero** tenga más monedas. El inconveniente es que no todos los personajes son tesoreros, así que habrá que comprobarlo en primer lugar vía: **(p instanceof Tesorero)**, antes de tratar de acceder a sus monedas vía **((Tesorero)p).getMonedas()**.

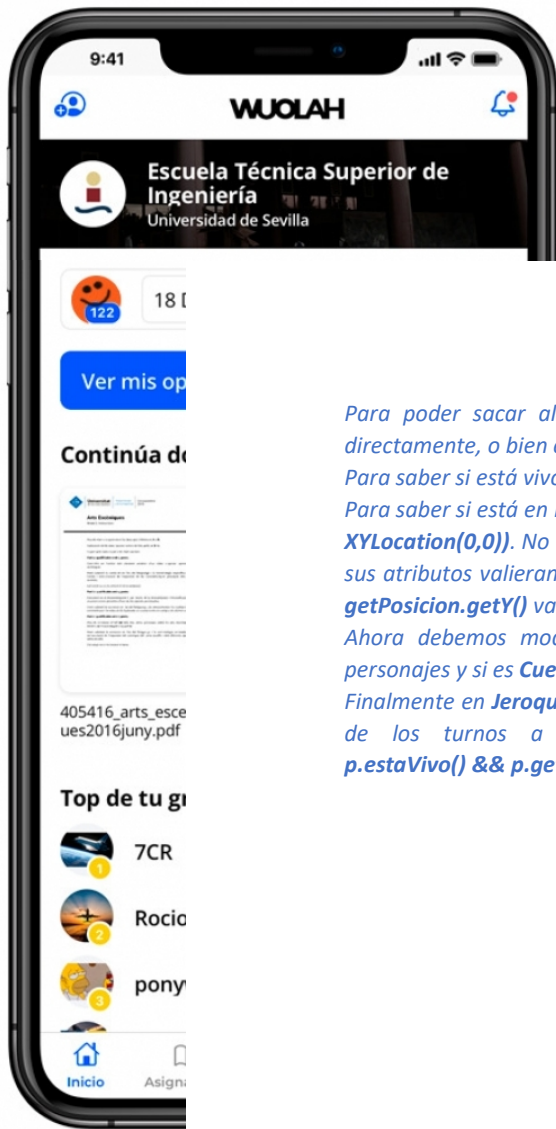
Al final, tanto si no hay ningún **Tesorero**, como si el máximo tiene cero monedas, se debe retornar **null**.

- iii. **[0,75 puntos]** La acción de combatir de los cuervos se limita a robar todo el dinero de su objetivo, si es que tiene alguno.

Se debe sobrescribir **Cuervo.accionCombatir()** y si tiene algún objetivo válido, escoger uno de ellos y si es **Tesorero** transferir todo su dinero. Como sus enemigos son solo **Tesoreros** podemos obviar la comprobación de que el objetivo lo sea: **this.transfiereTodo((Tesorero)objetivo)**.

Nota: no necesitamos sobrescribir **Cuervo.combatir()**, ya que no estamos en el combate, sino la acción de combatir.

7. **[1,5 puntos]** Al final de la ronda, todos los cuervos con dinero tratarán de escapar. Para ello se definirá en la clase **Cuervo** el método **escapaSiPuedes(<argumentos_que_consideréis_necesarios>)** : Si el cuervo está vivo, tiene alguna moneda y está en la **casilla de salida** se saca del tablero. Como consecuencia en la resolución de cada ronda, sólo deben jugar los personajes vivos que estén en el tablero.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.



Para poder sacar al **Cuervo** del tablero, es necesario que el método reciba el tablero de la partida directamente, o bien el objeto **partidaActual**.

Para saber si está vivo y tiene dinero utilizaremos **estaVivo()** y **getMonedas()**>0, respectivamente.

Para saber si está en la casilla de salida debemos utilizar el operador **equals()**. **This.getPosicion().equals(new XYLocation(0,0))**. No vale el operador **==** ya que miraría que fuesen el mismo objeto en memoria, no solo que sus atributos valieran lo mismo. También se podría comprobar que ambas coordenadas **getPosicion().getX()** y **getPosicion().getY()** valgan ambas cero.

Ahora debemos modificar **Jeroquest.siguienteRonda()** y, tras ejecutar la ronda, volver a recorrer los personajes y si es **Cuervo**, realizar el downcast a **Cuervo** e invocar a **escapaSiPuedes()**.

Finalmente en **Jeroquest.resuelveRonda()** modificamos el condicional para que solo se invoque la resolución de los turnos a personajes que además de estar vivos estén en el tablero, es decir: **p.estaVivo() && p.getPosicion() != null**