TPI Objetos - Yellow Submarine

Polimorfismo: ¿Cuál es el mensaje polimórfico? ¿Qué objetos lo implementan? ¿Qué objeto se aprovecha de ello?

El mensaje polimórfico es ***proximaPosicion(unObjeto)***, lo implementan los objetos sentidos (arriba, abajo, izquierda, derecha). Los objetos que se aprovechan de ello son los instanciados a partir de la clase ***MovimientoRecto*** que ejecutan ***sentido.proximaPosicion(unObjeto)*** para obtener una nueva posición en el sentido correspondiente pero sin importar qué sentido tenga asociado el objeto movimientoRecto porque los trata polimórficamente.

La ventaja está en que se pueden agregar sentidos sin modificar absolutamente nada del código de la clase ***Movimiento Recto***, sólo hay que agregar un objeto o clase que implemente el método ***proximaPosicion(unObjeto)***.

Colecciones: ¿Qué operaciones de colecciones se utilizan? ¿Usaron mensajes con y sin efecto? ¿Para qué?

Las operaciones de colecciones con efecto que utilizamos son distintos ***forEach*** para agregar los obstáculos correspondientes a cada dificultad al juego a traves del metodo ***generarCambios()***. Por ejemplo:

```
method generarCambios(){
        self.bombas().forEach({unaBomba =>
                game.addVisual(unaBomba)
                yellowSubmarine.agregarBomba(unaBomba)})
}
```

Las operaciones de colecciones sin efecto que utilizamos son ***intersection()*** y ***isEmpty()*** en los métodos ***noChocaConPiedra(nuevaPosicion)*** y ***noChocaConBomba(nuevaPosicion)***.

```
method noChocaConPiedra(nuevaPosicion) =
self.piedrasDelJuego().intersection(game.getObjectsIn(nuevaPosicion)).isEmpty()
```

Estos métodos son utilizados por ***noChocaConBombaNiPiedra(nuevaPosicion)*** para averiguar si en una posición determinada del tablero no hay un objeto que impida el movimiento o aparición de otro. El método ***intersection()*** calcula la intersección entre los objetos posicionados en la ***nuevaPosicion*** y las piedras que ya fueron agregadas al juego anteriormente, luego analiza con el método ***isEmpty()*** si la colección resultante de la intersección es vacía. Esto indica que el objeto se puede mover a ***nuevaPosicion***.

Clases: ¿Usan clases? ¿Por qué? ¿Dónde o cuándo se instancian los objetos?

Usamos una clase ***Moneda*** porque cada vez que el submarino agarra una moneda se genera una nueva automáticamente. Todas las monedas tienen el mismo comportamiento por lo tanto sería ilógico crear muchos objetos moneda distintos sin implementar una clase.

Cuando el objeto submarino colisiona con una moneda se ejecuta el método:

```
method teChocoElSubmarino(){
        submarino.agarrarMonedas(1)
        self.agregarMoneda(new Moneda())
        self.borrarMoneda(self)
}
```

Este método ejecuta ***self.agregarMoneda(new Moneda())*** el cual instancia un objeto de la clase ***Moneda*** y lo agrega al juego junto con su método de movimiento.

```
method agregarMoneda(unaMoneda){
        game.addVisual(unaMoneda)
        game.onTick(
            yellowSubmarine.dificultad().velocidadMoneda(),
            "mover la moneda" + unaMoneda.toString(),
            { unaMoneda.moverse()})
}
```

Composición: ¿Qué objetos interactúan? ¿Dónde se delega? ¿Por qué no herencia?

Utilizamos composición cuando interactúan los objetos instanciados a partir de la clase ***Tiburon*** y los objetos instanciados a partir de la clase ***movimientoRecto*** cuando se instancia un nuevo objeto ***Tiburon***.

```
new Tiburon(
    movimiento = new MovimientoRecto(sentido = derecha),
    distancia = 3,
    velocidad = 100)
```

No utilizamos herencia en este caso porque no queremos instanciar todos los objetos ***Tiburon*** con el mismo ***movimientoRecto***, con distinto ***sentido***, ***distancia*** y ***velocidad***.

Herencia: ¿Entre quiénes y por qué? ¿Qué comportamiento es común y cuál distinto?

Utilizamos herencia con los obstáculos del juego, entre la clase madre ***Obstaculo*** de la cual hereda la clase ***ObstaculoConPoder*** y a su vez de esta última heredan las clases ***Piedra***, ***Bomba*** y ***Tiburon***. De la clase ***Obstaculo*** también hereda la clase ***Pulpo***.

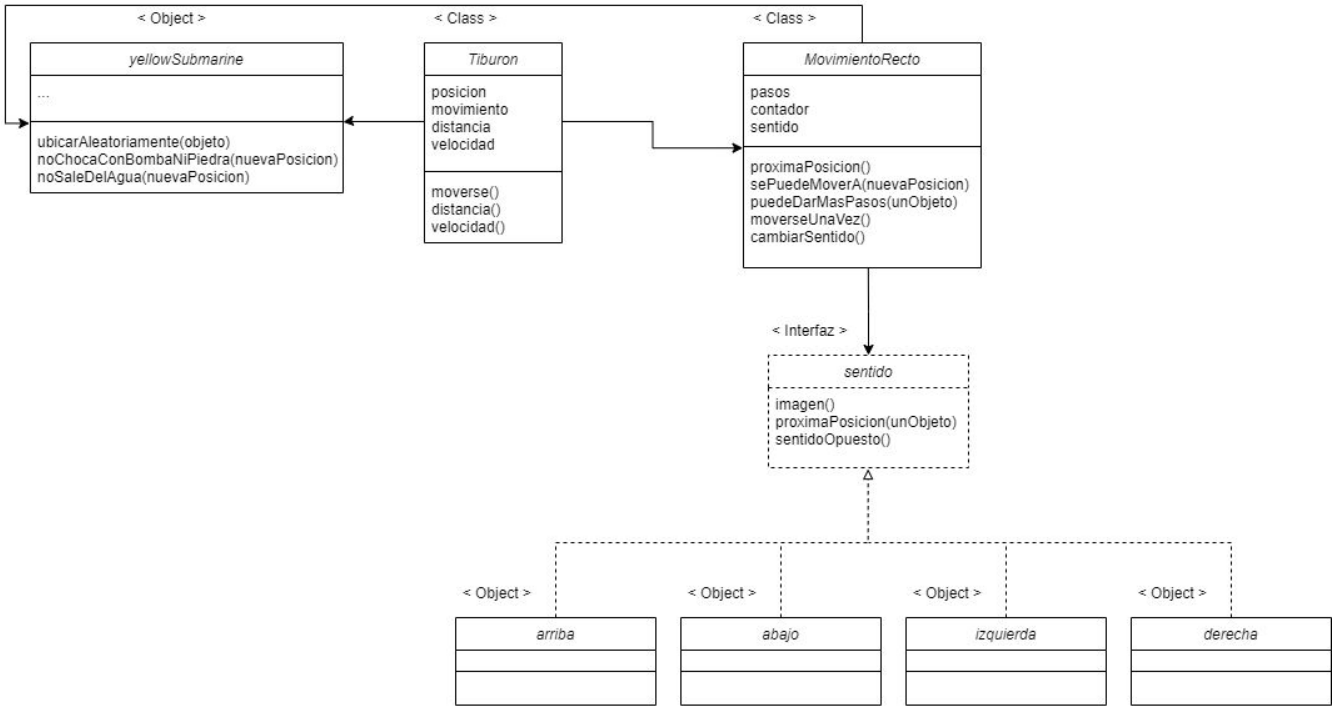
El comportamiento común para todos los obstáculos es analizar si el submarino tiene escudo cuando choca con estos, en caso de que tenga escudo se lo rompen y si no tiene escudo cada clase realiza una acción distinta que depende del tipo de obstáculo. Este comportamiento se ejecuta a traves del metodo ***teChocoElSubmarino()*** definido en la clase ***Obstaculo***.

```
method teChocoElSubmarino(){
        if(submarino.tieneEscudo())
            submarino.perderEscudo()
        else
            self.accionAlChocar()
}
```

El comportamiento distinto es la ***accionAlChocar()*** que se ejecuta cuando el submarino colisiona con un obstáculo, este método se encuentra definido como un método abstracto en la clase madre ***Obstaculo*** y redefinido en las demás clases.

La diferencia es que los ***ObstaculosConPoder*** dañan al submarino quitando una cantidad determinada de vidas, la cual depende de la clase, por ejemplo las piedras le quitan 1 vida y las bombas 2 vidas. En cambio al chocar con el pulpo este no le hace daño al submarino, pero le roba todas sus monedas.

Movimiento de los tiburones (Diagrama de Clases):



Con este diagrama explicaremos cómo funciona el movimiento de cada objeto de la clase ***Tiburon***, el cual es un movimiento recto en línea horizontal o vertical.

Cuando se instancia un objeto de la clase ***Tiburon*** se le asigna un objeto de la clase ***movimientoRecto*** en su atributo ***movimiento***. Al ***movimientoRecto*** a su vez se le asigna alguno de los 4 objetos sentidos posibles de movimiento (arriba, abajo, izquierda, derecha) en su atributo ***sentido***. También se inicializa su atributo ***posicion*** con una posición aleatoria la cual es generada por el método ***ubicarAleatoriamente(objeto)*** del objeto ***yellowSubmarine***.

La imagen del ***Tiburon*** es calculada a partir de su atributo ***movimiento***, éste calcula la imagen correspondiente en función de su sentido actual enviando el mensaje ***imagen()*** al ***sentido***.

Cada vez que el ***Tiburon*** quiere dar un paso (cada ***Tiburon*** tiene asignado un evento OnTick en el que se ejecuta el método ***moverse()*** cada determinado tiempo, este tiempo está determinado por su atributo ***velocidad***) le pide una nueva posición al movimiento enviándole el mensaje ***moverseUnaVez()***.

El método ***moverseUnaVez()*** verifica si puede dar el próximo paso el tiburón con los métodos ***sePuedeMoverA(nuevaPosicion)*** y ***puedeDarMasPasos(unObjeto)***. En caso de poder dar el paso le retorna la nueva posición al ***Tiburon*** y sino cambia el ***sentido*** de ***movimiento*** al ***Tiburon*** con el método ***cambiarSentido()***.

El método ***cambiarSentido()*** cambia el atributo ***sentido*** del ***movimientoRecto*** por el sentido opuesto, y retorna la misma posición en la que se encontraba el ***Tiburon***.