Análisis de código y pruebas unitarias

Enlace al repositorio GitHub: https://github.com/JaviGomez02/JuegoJaca.git

Índice:

Puedes hacer click en cualquiera de los títulos para llevarte a la página directamente.

Proyecto conjunto. EEDD y Programación

1. Análisis estático de código.

Gracias a la herramienta SonarLint he podido realizar un análisis de mi código y he podido identificar algunos errores o malas prácticas de programación, las cuales he registrado en la siguiente tabla:

Tipo de problema	Criticidad	Descripción	Solución aplicada

Número random	Critical	Estoy creando un número random en un método, por lo que no es eficiente:
		
		private int random(int tamanno) {
		Random r = new Random();
		return r.nextInt(tamanno);
		}
			Sacar el Random del método y declararlo como un atributo:
			private Random r = new Random();

Nombres de métodos	Minor	Hay métodos que están nombrados de forma errónea, como por ejemplo CambiaJugadorPosicion	Nombrarlos con el estándar para que el código sea más eficiente: cambiaJugadorPosicion

Sentencias IF	Major	Tengo una sentencia IF detrás de otra en vez de estar las dos condiciones dentro del mismo IF
		if (e instanceof Jugador j) {
		if (j.getDinero() == Constantes.DINERO) {
		resultado = true;
		}
		}
			Juntamos las dos condiciones dentro del mismo IF:
			if (e instanceof Jugador j && j.getDinero() == Constantes.DINERO) {
			resultado = true;
			}

Definir constante	Critical	En la clase TestJugador uso la frase "Error si no salta la exception" en repetidas ocasiones, por lo que es muy redundante.	Creamos una constante que contenga esa frase y usamos la constante: private static final String error="Error si no salta la exception";
Nombrar constante	Critical	Al crear la constante la he nombrado en minúscula: error	Según las reglas de refactorización debemos nombrar las constantes en mayúsculas: ERROR

Nota: Solo he tenido en cuenta los problemas de las clases Jugador, Coordenadas y Juegos, ya que son las únicas que he tenido que crear o modificar.

2. Definición de pruebas unitarias:

Las siguientes definiciones han sido creadas en una hoja de cálculo, y posteriormente han sido exportadas a este documento, por lo que puede que algunas palabras se encuentren entrecortadas debido al limitado tamaño de la página. Aún así, he intentado hacer lo mejor que he podido para que quede lo mejor posible.

Clase Jugador:

ID Caso Prueba	Nombre	Descripción	Precondiciones	Resultado esperado	Resultado obtenido	Comentarios
P_01	fuerzaParaLucharElfo	Comprobamos que los valores de fuerza son correctos cuando el jugador es un Elfo	Crear un jugador del tipo Elfo	Valor de fuerza entre 0 y 5	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_02	fuerzaParaLucharOgro	Comprobamos que los valores de fuerza son correctos cuando el jugador es un Ogro	Crear un jugador del tipo Ogro	Valor de fuerza entre 0 y 7	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_03	fuerzaParaLucharGuerrero	Comprobamos que los valores de fuerza son correctos cuando el jugador es un Guerrero	Crear un jugador del tipo Guerrero	Valor de fuerza entre 0 y 6	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_04	fuerzaParaLucharMago	Comprobamos que los valores de fuerza son correctos cuando el jugador es un Mago	Crear un jugador del tipo Mago	Valor de fuerza entre 0 y 4	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_05	velocidadParaLucharElfo	Comprobamos que los valores de velocidad son correctos cuando el jugador es un Elfo	Crear un jugador del tipo Elfo	Valor de velocidad entre 0 y 7	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_06	velocidadParaLucharOgro	Comprobamos que los valores de velocidad son correctos cuando el jugador es un Ogro	Crear un jugador del tipo Ogro	Valor de velocidad entre 0 y 4	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos

P_07	velocidadParaLuchar Guerrero	Comprobamos que los valores de velocidad son correctos cuando el jugador es un Guerrero	Crear un jugador del tipo Guerrero	Valor de velocidad entre 0 y 5	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_08	velocidadParaLuchar Mago	Comprobamos que los valores de velocidad son correctos cuando el jugador es un Mago	Crear un jugador del tipo Mago	Valor de velocidad entre 0 y 6	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_09	magiaParaLucharElfo	Comprobamos que los valores de magia son correctos cuando el jugador es un Elfo	Crear un jugador del tipo Elfo	Valor de magia entre 0 y 6	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_10	magiaParaLucharOgr o	Comprobamos que los valores de magia son correctos cuando el jugador es un Ogro	Crear un jugador del tipo Ogro	Valor de magia entre 0 y 4	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_11	magiaParaLucharGu errero	Comprobamos que los valores de magia son correctos cuando el jugador es un Guerrero	Crear un jugador del tipo Guerrero	Valor de magia entre 0 y 5	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_12	magiaParaLucharMa go	Comprobamos que los valores de magia son correctos cuando el jugador es un Mago	Crear un jugador del tipo Mago	Valor de magia entre 0 y 7	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_13	setDineroCorrecto1	Comprobamos que el método nos deja asignar la mayor cantidad de dinero	Crear un jugador de cualquier tipo	True	OK	Le hemos introducido un valor de 4
P_14	setDineroCorrecto2	Comprobamos que el método nos deja asignar la menor cantidad de dinero	Crear un jugador de cualquier tipo	False	OK	Le hemos introducido un valor de 0

P_15	setDineroIncorrecto1	Comprobamos que el método nos salta una exception al asignarle una cantidad de dinero por encima de su límite mayor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de 6
P_16	setDineroIncorrecto2	Comprobamos que el método nos salta una exception al asignarle una cantidad de dinero por encima de su límite menor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de -1
P_17	setPocionesCorrecto 1	Comprobamos que el método nos deja asignar la mayor cantidad de pociones	Crear un jugador de cualquier tipo	True	OK	Le hemos introducido un valor de 3
P_18	setPocionesCorrecto 2	Comprobamos que el método nos deja asignar la mayor cantidad de pociones	Crear un jugador de cualquier tipo	True	OK	Le hemos introducido un valor de 0
P_19	setPocionesIncorrecto1	Comprobamos que el método nos salta una exception al asignarle una cantidad de pociones por encima de su límite mayor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de 4
P_20	setPocionesIncorrecto2	Comprobamos que el método nos salta una exception al asignarle una cantidad de pociones por encima de su límite mayor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de -1
P_21	setGemasCorrecto1	Comprobamos que el método nos deja asignar la mayor cantidad de gemas	Crear un jugador de cualquier tipo	True	OK	Le hemos introducido un valor de 5
P_22	setGemasCorrecto2	Comprobamos que el método nos deja asignar la menor cantidad de gemas	Crear un jugador de cualquier tipo	True	OK	Le hemos introducido un valor de 0

P_23	setGemasIncorrecto1	Comprobamos que el método nos salta una exception al asignarle una cantidad de gemas por encima de su límite mayor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de 6
P_24	setGemasIncorrecto2	Comprobamos que el método nos salta una exception al asignarle una cantidad de gemas por encima de su límite menor	Crear un jugador de cualquier tipo	Exception	OK	Le hemos introducido un valor de -1
P_25	encuentraGema	Comprobamos que el método funciona correctamente y asigna una gema al jugador	Crear un jugador de cualquier tipo sin gemas	Gemas=1	OK	
P_26	encuentraDinero	Comprobamos que el método funciona correctamente y asigna dinero al jugador	Crear un jugador de cualquier tipo sin dinero	Dinero=1	OK	
P_27	encuentraPocion	Comprobamos que el método funciona correctamente y asigna una pocion al jugador	Crear un jugador de cualquier tipo sin pociones	Pociones=1	OK	
P_28	rompeRocaConGema	Nos aseguramos de que un jugador rompe una roca cuando este jugador lleva una gema o más	Crear un jugador de cualquier tipo y asignarle una gema o más	Rompe roca con gema	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos
P_29	rompeRocaResultado	Nos aseguramos de que un jugador gana o pierde contra una roca cuando el jugador no lleva ninguna gema	Crear un jugador de cualquier tipo sin gemas	Gana o pierde contra la roca	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los valores sean correctos

Clase Coordenadas:

ID Caso Prueba	Nombre	Descripción	Precondiciones	Resultado esperado	Resultado obtenido	Comentarios
P_01	coordenadaIncorrecta1	Comprobar que la coordenada se crea correctamente si metemos valores negativos.		Coordenada=(0,0)	OK	Hemos introducido x=-1, y=1
P_02	coordenadaIncorrecta2	Comprobar que la coordenada se crea correctamente si metemos valores negativos.		Coordenada=(0,0)	OK	Hemos introducido y=-1, x=1
P_03	goDownCorrecto	Comprobamos que el método goDown nos funciona Correctamente, es decir, nos devuelve true	Crear una coordenada donde la y<9	True	OK	Hemos introducido la coordenada (1,1)
P_04	goDownIncorrecto	Comprobamos que el método goDown nos devuelve False al aplicarlo a una coordenada que no puede ser Aumentada	Crear una coordenada donde la y=9	False	OK	Hemos introducido la coordenada (1,9)
P_05	goRightCorrecto	Comprobamos que el método goRight nos devuelve True al aplicarlo a una coordenada	Crear una coordenada donde la x<9	True	OK	Hemos introducido la coordenada (1,1)
P_06	goRightIncorrecto	Comprobamos que el método goRight nos devuelve False al aplicarlo a una coordenada específica	Crear una coordenada donde la x=9	False	OK	Hemos introducido la coordenada(9, 1)
P_07	goLeftCorrecto	Comprobamos que el método goLeft nos devuelve true Al aplicarlo a una coordenada	Crear una coordenada donde la x>0	True	OK	Hemos introducido la coordenada(1, 1)

P_08	goLeftIncorrecto	Comprobamos que el método goLeft nos devuelve False al aplicarlo a una coordenada en específico	Crear una coordenada donde la x=0	False	OK	Hemos introducido la coordenada(0, 1)
P_09	goUpCorrecto	Comprobamos que el método goUp nos devuelve True al aplicarlo a una coordenada	Crear una coordenada donde la y<0	True	OK	Hemos introducido la coordenada(1, 1)
P_10	goUpIncorrecto	Comprobamos que el método goUp nos devuelve False al aplicarlo a una coordenada en específico	Crear una coordenada donde la y=0	False	OK	Hemos introducido la coordenada (1, 0)

Clase Juego:

ID Caso Prueba	Nombre	Descripción	Precondiciones	Resultado esperado	Resultado obtenido	Comentarios
P_01	isTerminado	Comprueba que el método funciona correctamente. Este método devuelve verdadero cuando un jugador consigue todo el dinero	Crear un juego nuevo	True	OK	Le he asignado a un jugador el máximo de dinero
P_02	isTerminadoIncorrecto	Comprueba que el método devuelve falso cuando todavía existen más de un jugador y ninguno de ellos tiene todo el dinero	Crear un juego nuevo	False	OK	
P_03	getNextPositionN	Comprueba que el método devuelve correctamente la posición a la que se tiene que mover cuando se introduce “N”	Crear un juego nuevo	La nueva posición debe ser una celda más hacia el norte	OK	Hemos realizado un bucle con 10 iteraciones para asegurarnos de que los Valores sean correctos

P_04	getNextPosition S	Comprueba que el método devuelve correctamente la posición a la que se tiene que mover cuando se introduce “S”	Crear un juego nuevo	La nueva posición debe ser una celda más hacia el sur	OK	Hemos realizado un bucle con 10 iteraciones para asegurarnos de que los Valores sean correctos
P_05	getNextPosition E	Comprueba que el método devuelve correctamente la posición a la que se tiene que mover cuando se introduce “E”	Crear un juego nuevo	La nueva posición debe ser una celda más hacia el este	OK	Hemos realizado un bucle con 10 iteraciones para asegurarnos de que los Valores sean correctos
P_06	getNextPosition O	Comprueba que el método devuelve correctamente la posición a la que se tiene que mover cuando se introduce “O”	Crear un juego nuevo	La nueva posición debe ser una celda más hacia el oeste	OK	Hemos realizado un bucle con 10 iteraciones para asegurarnos de que los Valores sean correctos
P_07	getNextPosition Exception	Comprobamos que salta la excepción cuando se introducen valores erróneos	Crear un jugador del tipo Guerrero	Valor de velocidad entre 0 y 5	OK	Hemos realizado un bucle con 50 iteraciones para asegurarnos de que los Valores sean correctos

3. Implementación de pruebas unitarias:

He creado las pruebas unitarias basándome en las definiciones anteriormente mostradas en el apartado 2.
Dichas pruebas pueden ser observadas y analizadas dentro del mismo proyecto. Se puede acceder a él mediante el enlace que he facilitado en la primera página.