(EJERCICIO 10 - CATAN) PROGRAMACIÓN ORIENTADA A OBJETOS BÁSICA

Angel Maroto Chivite
24/01/2023

Índice

Contenido

Índice 1
1. Java & Kotlin 2
2. Ejecución de la simulación 2
2.1. Instanciar objetos 2
2.2 Impresión en pantalla 2
2.3 Decisiones Jugadores 3
2.4 Lanzar Dados 4
3. Pruebas unitarias 5
4. Diseño implementado 6
4.1 Factories Class 6
4.2 Models Class 6
4.3 Enum Class 6

1. Java & Kotlin

Primero realicé la práctica en Java.

Para la versión de Kotlin me he apoyado del conversor del IntelIJ, en este caso no he tenido que corregir ningún error.

2. Ejecución de la simulación

2.1. Instanciar objetos

Instanciamos el Mapa gracias a mis factories, está compuesta de una Casillas (Box), donde se generará aleatoriamente el mapa, siempre creando dos casillas mínimo de 2 por cada recurso.

Los dos jugadores que jugarán, en este caso un Jugador Humano que tome decisiones y un segundo Jugador Máquina que tomará las decisiones aleatoriamente.

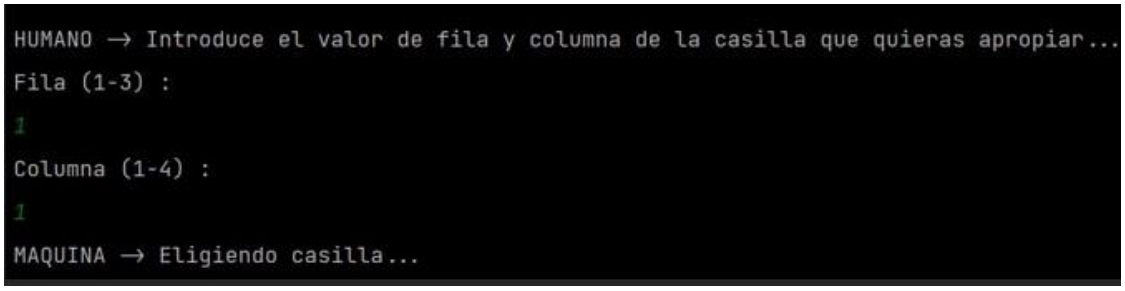
2.2 Impresión en pantalla

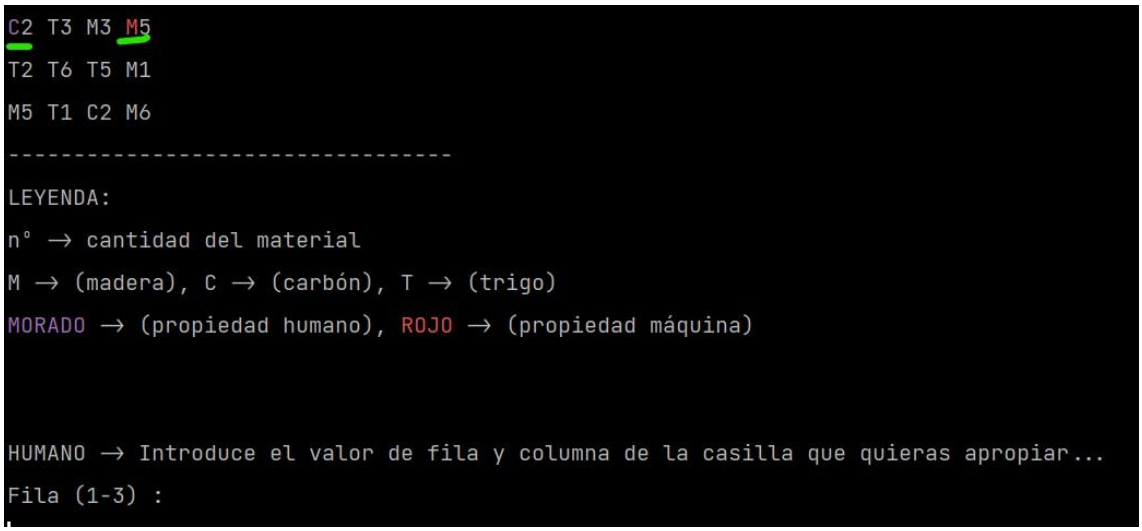
En todo momento se informará sobre el estado de la partida y el mapa con su correspondiente leyenda:

2.3 Decisiones Jugadores

En primer lugar, elegirá el Jugador Humano, la casilla que quiera poseer (la entrada se encuentra filtrada para que solo acepte valores numéricos con el valor menor y mayor del mapa generado, y que se asigne una casilla sin ningún poseedor)

En segundo lugar, si todo es correcto, la máquina decidirá qué casilla poseer de manera aleatoria con los mismos filtros o muy similares.



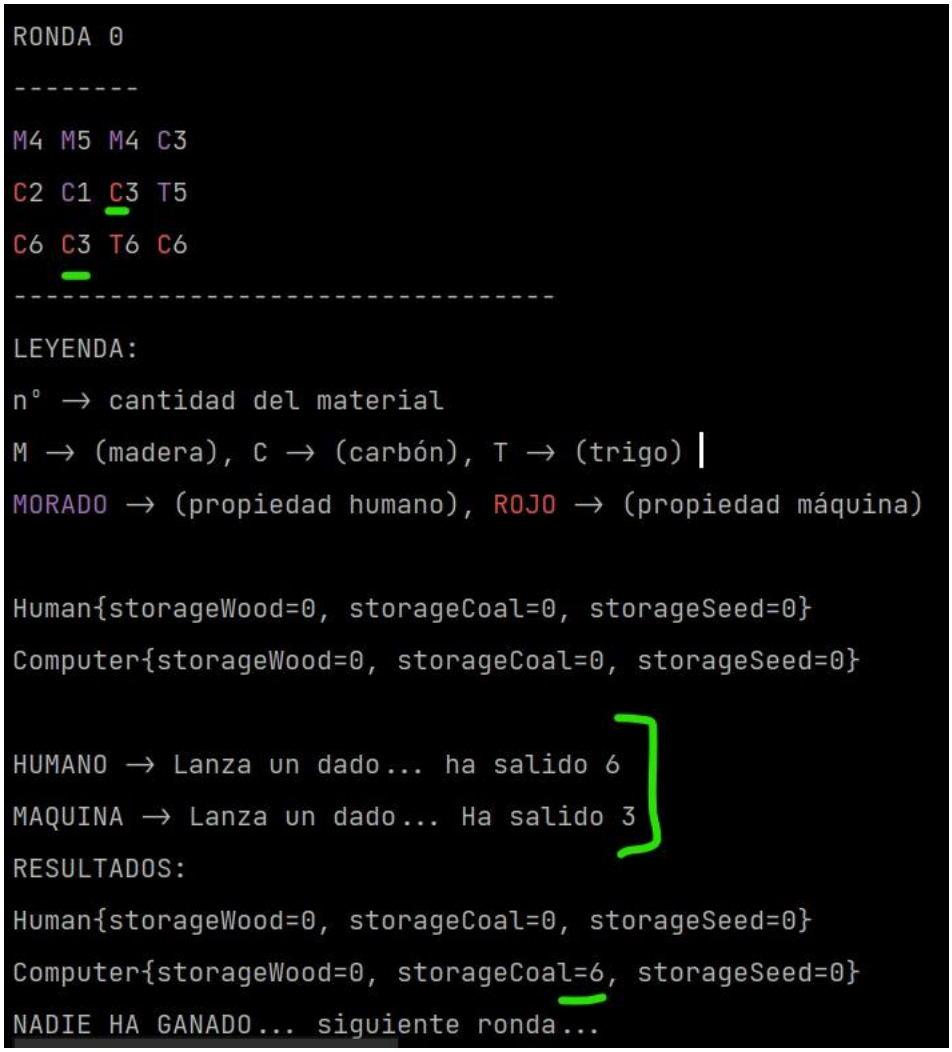


Se repetirá esta acción hasta que todas las casillas se encuentren poseídas e iniciaría la tirada de dados.

2.4 Lanzar Dados

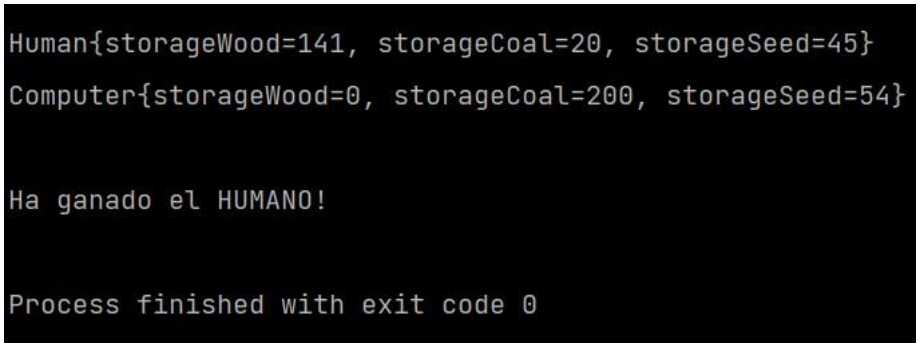
Se lanzarán dado de manera aleatoria y se verificará, si es humano o máquina para introducir el valor del dado que sea igual a la casilla del mapa que corresponda con la posesión de cada uno.

En primer lugar, será el Humano y después la Máquina.



Se repetirá estas acciones hasta que uno de los dos alcance, el valor de 20 o más en madera, carbón y trigo.

En este caso, en el juego la máquina no seleccionó ninguna casilla de madera, gracias a eso ya gané, porque nunca aumentará su contador de madera.



3. Pruebas unitarias

El código de impresión en pantalla no se encuentra probado, y tampoco alguna función puntual que se me ha complicado de más, el resto del código está probado:

factories	100% (8/8)	100% (8/8)	100% (26/26)
BoxFactory	100% (1/1)	100% (1/1)	100% (3/3)
ComputerFactory	100% (1/1)	100% (1/1)	100% (1/1)
HumanFactory	100% (1/1)	100% (1/1)	100% (1/1)
MapFactory	100% (1/1)	100% (1/1)	100% (8/8)
models	100% (12/12)	72% (16/22)	36% (30/82)
Box	100% (2/2)	100% (4/4)	100% (11/11)
Computer	100% (1/1)	50% (1/2)	20% (1/5)
Human	100% (1/1)	50% (1/2)	20% (1/5)
Map	100% (1/1)	50% (1/2)	5% (1/19)
Player	100% (1/1)	100% (1/1)	100% (1/1)
utils	100% (4/4)	73% (22/30)	47% (134/280)
funciones	100% (1/1)	71% (10/14)	45% (62/135)
randomData	100% (1/1)	100% (1/1)	100% (5/5)

Para la versión en Java he utilizado el gestor de dependencias Maven mediante la dependencia de Test Junit5.

Para la versión de Kotlin he utilizado el gestor de dependencias Gradle mediante la dependencia de Test Junit5.

4. Diseño implementado

4.1 Factories Class

Creación de objetos aleatorios mediante factories, en el caso del número de recurso y tipo que habrá en cada casilla (aunque obligando que haya mínimo 2 casillas de cada tipo).

4.2 Models Class

Objetos que instanciaré a lo largo de la ejecución del programa.

Donde aplico una herencia en Humano y Máquina de Persona, esta última siendo abstracta así actúa como molde inicial.

4.3 Enum Class

Mediante:

Tipo de recursos así podré mantener un tipo distinto por cada casilla.

Tipo de posesión, así cada casilla tendrá asignada si está en posesión de un jugador, otro o vacía..

Colores para mejorar la interfaz