CONNECT4

José Miguel Avellana López

18068091G

GM1

**Problemas**

En Player.isEqualTo, this.id solo puede valer ‘1’ o ‘2’, debido a que el constructor de Player es privado, si vamos a comparar con otra variable Player que podría valer por ejemplo ‘.’ esta tendrá que ir como parámetro, es decir, entre paréntesis. Si no saltará el error NullPointerException.

Otro error que tenia era en Board.maxConnected, ya que en los while primero preguntaba si eran iguales las dos celdas y después si esa celda se encontraba en el tablero.

Esto podia dar error ya que en el caso de que la celda que varia (la que tiene i) no existiera en el tablero igualmente se metia en el isEqualTo y daba error.

Se corrige simplemente cambiando el orden, ya que si no se cumple que i esté en el tablero, sale directamente del while.

La primera vez que hice el maxConnected no hice el método para contar la diagonal inversa y el test me lo paso igual.

**Board.maxConnected**

Lo que hace maxConnected es comparar las celdas conectadas en cada una de las cuatro direcciones de pos, y retorna el valor más alto de estos cuatro.

La comparación se realiza con el método Math.max, como solo se pueden meter dos parámetros primero se obtiene:

-El máximo entre diagonal y diagonal invertida.

-Y el máximo entre horizontal y vertical.

Y entonces retorna el máximo de los dos máximos anteriores.

Para saber el máximo de celdas conectadas en cada dirección hay un método privado para cada una, los cuatro tienen la misma estructura.

Tenemos una variable ‘c’ que cuenta las celdas conectadas en esa dirección, y que es el valor que se retornará.

También tenemos dos variables ‘i’ y ‘j’ que cuentan la variación de columnas y filas respectivamente. Para las direcciones horizontal y vertical solo se usa una ya que la otra permanece constante.

De esta forma, las variables ’i’ y ‘j’ comienzan marcando pos, en el primer while vamos en una dirección, y se va aumentando ‘c’ y aumentando o disminuyendo ‘i’ o ‘j’ dependiendo de la dirección, en el momento en que salgamos del tablero o la casilla sea diferente saldremos del while.

Ahora se cambia el valor de ‘i’ y ‘j’ al de la casilla al lado de pos en la dirección del método y sentido inverso al del anterior while, y se hace otro while para el sentido contrario al anterior pero con la misma estructura.

De esta forma la casilla pos solo la cuenta en el primer while, y por tanto el valor que se retornará será siempre igual o mayor que uno.

**Game.play**

Lo primero que se comprueba es a qué jugador le toca, como la partida no ha acabado solo hay esos dos estados posibles.

El código se repite en el if de cada jugador, pero como el constructor Player es privado no se puede crear una variable y simplificar el código.

La estructura de cada if es:

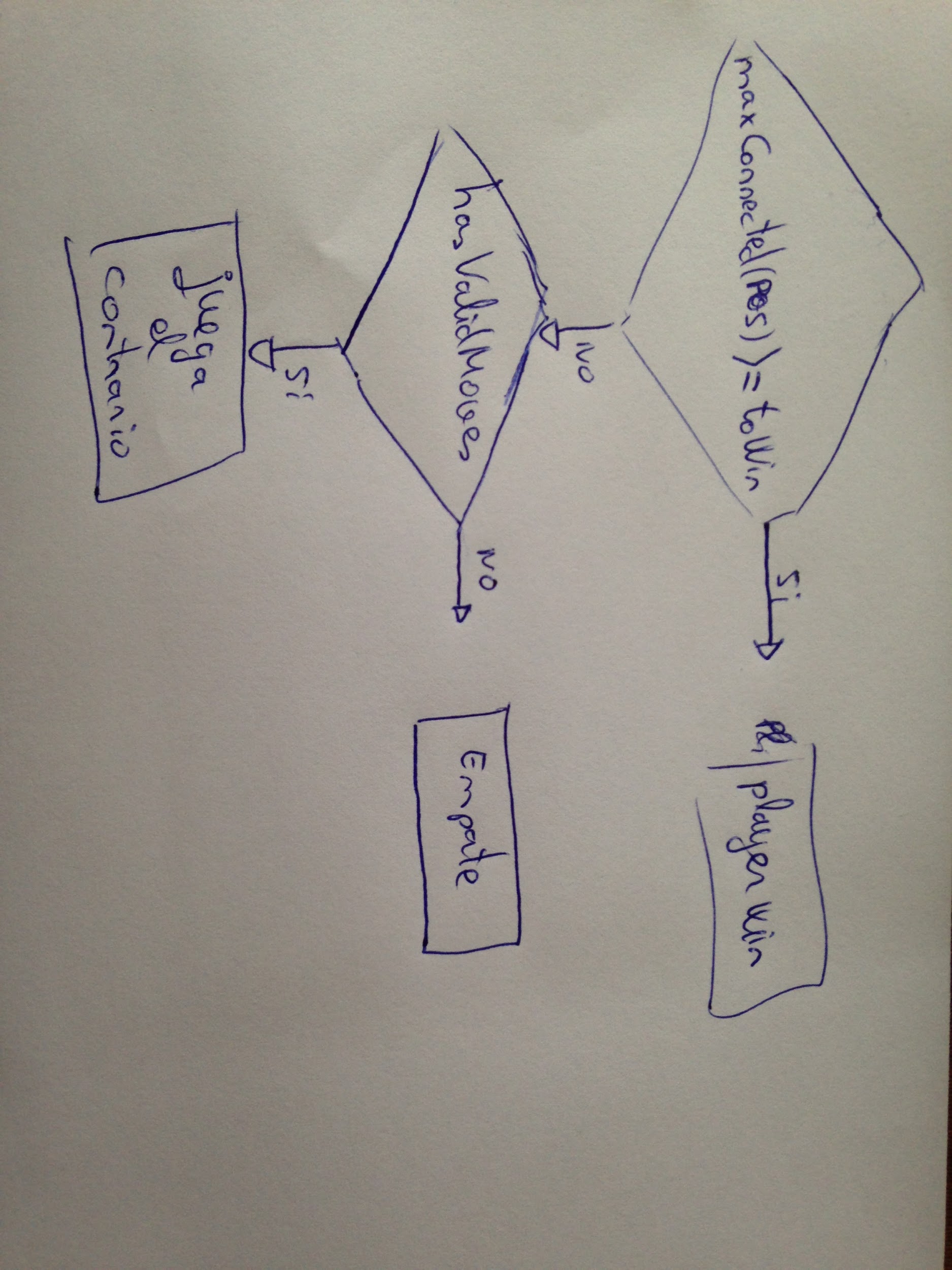
Primero se obtiene la posición pos donde cae la ficha del jugador en la columna que se indica como parámetro, al mismo tiempo la clase Board.play, que es la que retorna la posición, cambia el valor del id de esa celda al de player.

Después se comprueba si el máximo de celdas conectadas del jugador en esa posición es igual o mayor al necesario para ganar, si lo es, el estado indicará que el jugador ha ganado.

Si no se cumple comprueba si ya no quedan movimientos válidos con hasValidMoves y si no hay cambiará el estado a empate.

Si el jugador ha ganado y aún quedan movimientos cambiará el estado a que le toca jugar al contrario.

Finalmente retornará un Move con el jugador y la posición como parámetros.



**Board.play**

Usando el método lasEmptyRow obtenemos la última casilla sin ocupar, y se almacena en la variable row.

Al igual que con Game.play, al ser el constructor de Player un constructor privado, no se puede crear una variable player que contenga el id del jugador, por lo que uso un if y repito el código para cada jugador.

Dentro del if se cambia el id de cells[row][column] al del jugador, y se retorna la posición de esa casilla. En el caso de que row valga -1, que es porque la columna está llena, no entrará en ninguno de los dos ifs y retornará null.

**Conclusiones**

En esta práctica he intentado optimizar el código mejor que en la anterior: no repetir código, usar una estructura descendiente o usar métodos privados. He comprendido mejor la forma en la que se conectan las clases de un programa y cuando un método o instancia de una clase ha de ser pública o privada.