Informe de la 2ª práctica programación II

Autor:

Jordi Lazo Florensa

Grado en Ingeniería Informática

Programación II

Grupo 2

Universitat de Lleida

Descripción del desarrollo de la práctica

1. Position (1 punto)

La clase “Position” es la segunda clase que más fácil me ha sido programar. Cabe destacar en la función “colinear” decidí crear un condicional que compara las posiciones de *x, y* y que gracias al entorno de desarrollo de IntelliJ me permite automáticamente simplificar la función devolviendo la condición escrita en el “if” pero para mí entendimiento decidí dejarlo tal cual.

Por lo que respecta a la función “distance”, al principio decidí aplicar dos condicionales if para comparar el valor de position 1 y 2 y multiplicarlos por *-1 para obtener el valor absoluto, pero comentado este resultado con un compañero de clase me comentó que en java existe una función math.abs que permite retornar el valor absoluto y decidí aplicarlo ya que me ahorraba varias líneas de código.

Finalmente para la función “middle” tuve una duda de como devolver 2 objetos (x,y) pero me di cuenta que había que devolver 1 punto creando un nuevo objeto.

2. Direction (1 punto)

Esta clase también es bastante sencilla pero me hizo que pensar con el eje x,y para el UP y DOWN ya que es diferente que el típico eje x,y porque para subir debes añadir 1 unidad a y y para bajar debes restar 1 a y y al tratase de programación POO en java es al revés (los dibujos de la clase “Geometry” me ayudaron a entender mucho el sistema de representación gráfica).

Para la función “apply” necesité entender mejor el getter && setter de java ya que aquí lo utilizé por primera vez al llamar from.getX() de la clase “Position” para poder dar los nuevos valores al objeto creado dentro de la función y finalmente devolverlo. El hecho de dedicarle tiempo a este clase me ayudó posteriormente en todas las demás clases porque tanto el apply como el array Direction [] ALL más adelante las usaré en varias ocasiones.

3. Cell (1 punto)

La clase “Cell” ha sido la clase más sencilla de programar de toda la práctica 2. Simplemente hay que decir si una casilla del tablero esta vacía, prohibida u ocupada, para ello he decidido crear los condicionales más típicos aprendidos en la asignatura de Programación I que compararán el status con los atributos estáticos declarados arriba del constructor (FORBIDDEN, FILLES, EMPTY). Como apunte ItelliJ permite automáticamente reducir los condicionales.

4. Board (1 punto)

En esta clase tuve problemas posteriormente ya que la clase “Game” me daba error al pasar los tests de modo que “Board” era decisiva para el funcionamiento correcto.

En el constructor de la clase “Board” crearemos el tablero del juego. Para hacerlo tendremos que “romper” el String que se nos introduzca en forma de tablero (FORBIDDEN=`#`, FILLED=`o`, EMPTY=`·`) en “fichas” con la función StringTokenizer, posteriormente recorreremos con 2 bucles el array cells para introducir cada ficha en cada posición del array con el método charAt. Así crearemos el tablero.

En las funciones posteriores para comprobar si las celdas están llenas, vacías o prohibidas crearemos condicionales que comprobaran si las posiciones x,y están dentro o fuera de tablero para ver si las jugadas son posibles o no.

5. Game (2 punto)

La clase “Game” ha sido la clase más complicada de programar y por eso la explicaré detalladamente que hace función. He creado una función auxiliar “verifyDirection” que me permite recorrer las direcciones posibles que comprueba si se puede matar las fichas adyacentes. La usaré en la función “isValidFrom” que determinará si es posible. En la función “isValidTo” es imprescindible usar la función anteriormente programada para ver si es válido o no el movimiento de “matar” las piezas que se encuentran a 2 posiciones des de la posición inicial hasta la final siempre que en medio no exista una posición vacía. Estas 2 funciones son las más importantes de la clase ya que más adelante se usarán en reiteradas ocasiones.

En la función “Position move” se realiza la acción de mover la ficha, es decir, se elimina la ficha de la posición intermedia (se deja vacía), se llena la posición final y se vacía la posición inicial de movimiento.

En la función “hasValidMovesFrom” hay que recorrer el tablero con 2 bucles y comprobar con la función “isValidFrom” si existe alguna posición con algún movimiento posible desde el inicio de la partida.

Finalmente, para las funciones “countValidMovesFrom” i “countValidMovesTo” recorreremos con 2 bucles las posiciones validas desde el inicio y desde el final, respectivamente, y les sumaremos +1 a cada contador para averiguar el total de jugadas válidas.

6. Geometry (1 punto)

Por lo que respecta a esta clase simplemente hay que implementar las dimensiones del tablero interno dentro de la pantalla, las celdas las cuales dividen el tablero interno y finalmente los óvalos (círculos) que forman cada celda. Todo ello los obtendremos dividiendo, sumando y multiplicando el ancho, largo, columnas y filas del tablero y celdas correspondientes.

Conclusiones

Gracias a este proyecto he entendido mejor los fundamentos de las funciones y clases ya que para poder realizar el proyecto es esencial tener un buen dominio. Los “getter” y “setters” de cada clase se han usado en otras haciendo que se sincronicen de manera perfecta y sean dependientes entre ellas.

También he aprendido a mejorar en funcionamiento de los objetos creados en todas las clases junto a sus atributos públicos, privados, estáticos y finales.

También el uso de los paquetes de “graphics” me ha ayudado a entender más y mejor la POO en java.

Finalmente, el uso de los “tests” en esta práctica me ha ayudado muchísimo a ver mis errores cometidos en las clases y en como solventarlos