Othello en Java

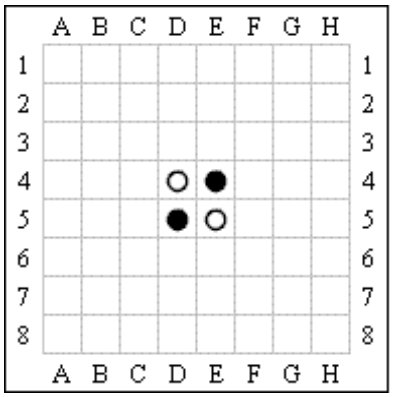
Se trata de realizar unha aplicación Java que permita a dúas ***persoas*** xogar ao Othello entre elas, contra o ordenador ou que o ordenador xogue contra si mesmo.

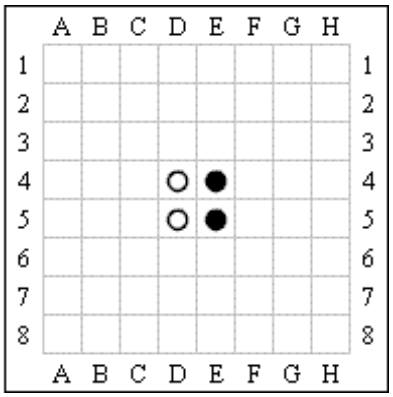
As regras do Othello se poden consultar na Wikipedia (https://es.wikipedia.org/wiki/Reversi):

Reglas

En el othello, al igual que en el reversi, se emplea un tablero de 8 filas por 8 columnas y 64 fichas idénticas, redondas, blancas por una cara y negras por la otra (u otros colores). Las casillas se denotan numerando las columnas, comenzando por la esquina superior izquierda del tablero, con letras de la A a la H, e igual con las filas, pero con números del uno al ocho. A un jugador se le asigna un color y se dice que lleva las fichas de ese color, lo mismo para el adversario con el otro color.

En el otelo, de inicio, se colocan cuatro fichas tal como se ve en el diagrama de la izquierda: dos fichas blancas en D4 y E5, y dos negras en E4 y D5. En el reversi estas cuatro casillas comenzarían vacías y se irían ocupando alternativamente, típicamente comenzando por el jugador que lleva las fichas negras. Ésta es la primera diferencia entre las reglas del reversi y del otelo: una única restricción en la apertura (los diagramas de abajo muestran las dos posibles aperturas, no siendo posible la segunda en el otelo).



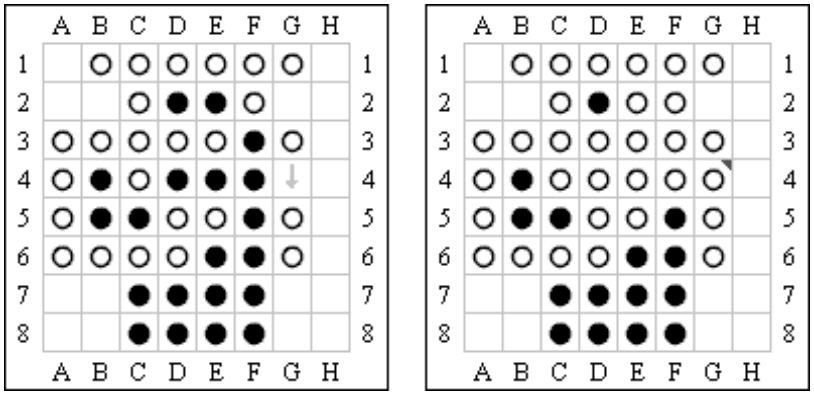


La segunda y última diferencia entre el reversi y el otelo estriba en el número de fichas del que dispone cada jugador para hacer sus movimientos. **En el otelo se comparten las sesenta y cuatro fichas**, mientras que en el reversi cada jugador dispone de sólo treinta y dos fichas (sólo existirán diferencias si uno de los jugadores pasa, puesto que entonces el otro jugador tendrá que mover de nuevo gastando una de sus fichas y al final de la partida no le quedarán suficientes para hacer sus últimos movimientos, viéndose pues obligado a pasar).

Empezando por quien lleva las fichas negras los jugadores deben hacer un movimiento por turno, a menos que no puedan hacer ninguno, pasando en ese caso el turno al jugador contrario. *El movimiento consiste en colocar una ficha de forma que flanquee una o varias fichas del color contrario y voltear esas fichas para que pasen a mostrar el propio color.*

Se voltean todas las fichas que se han flanqueado en ese turno al colocar la ficha del color contrario. Esas fichas, **para que estén flanqueadas, deben formar una línea continua recta (diagonal u ortogonal) de fichas del mismo color entre dos fichas**

del color contrario (*una de ellas la recién colocada y la otra ya presente*). En el siguiente ejemplo juegan las blancas donde indica la flecha y se puede ver qué fichas se voltean.



La partida finaliza cuando ningún jugador puede mover (normalmente cuando el tablero está lleno de fichas) y gana quien en ese momento tenga sobre el tablero más fichas mostrando su color.

Análise da Aplicación

Como se comentou na documentación de "*Introdución á Programación Orientada a Obxectos I*", unha aplicación orientada a obxectos consiste nun conxunto de obxectos (cada un cunhas características concretas – atributos – e capaz de levar a cabo unhas accións determinadas – métodos –) que se van indicando qué se ten que facer para levar a cabo os distintos *casos de uso* (respostas da aplicación as distintas situacións) da aplicación.
Polo tanto, o máis importante para deseñar unha aplicación é identificar os obxectos que van a formar parte da mesma os seus atributos e os métodos dos que deben dispoñer, para deseñar as distintas Clases as que pertencen.
Unha vez feito isto, debemos identificar os distintos *casos de uso* que presenta a aplicación e a resposta que se debe dar a cada un de eles.
Normalmente, isto xera diagramas UML que representan as clases (diagrama de clases) e os casos de uso (diagrama de casos de uso). A partir deles se deseñan outros diagramas como o diagrama de secuencia… etc.

Neste módulo, que é de introdución a programación, non realizaremos os diagramas UML, pero a identificación das clases e dos casos de uso é imprescindible.

Identificación e deseño das Clases

Os obxectos que forman parte da aplicación os seus atributos e os métodos dos que constan se poden deducir **empregando o sentido común** a partir do enunciado examinando:

- **Os substantivos**: Os substantivos poden ser **obxectos**, **atributos** ou non ser relevantes para a aplicación por múltiples razóns.
- **Os verbos**: Os verbos son accións levadas a cabo por un suxeito. Polo tanto os verbos poden ser **métodos** de algún obxecto.

Partindo do enunciado e das regras do xogo podemos ver:

1. **Persoas –** Son as que xogan… podemos chamalas ***Xogadores***. No xogo existen dous e poden ser persoas reais, ou un obxecto da aplicación…. (as persoas reais xogan elas e non é necesario programar nada, pero si queremos que xogue a aplicación necesitamos un Obxecto Xogador). Os Xogadores deberían ser capaces de :

    a) **realizar xogadas** – Si temos un obxecto Xogador, debe ter a capacidade de ***xogar***. Si o xogador é humano, se lle debería solicitar a **posición (**fila - número do 1 ao 8 - e columna - letra entre A e H -) da xogada. Si o xogador é a aplicación debería elixir a **posición** seguindo unha estratexia de xogo

2. Temos un **taboleiro** de 8x8 posicións. Cada posición almacenará unha **ficha**, que podemos representar con 0, 1 ou 2. O taboleiro debería ser capaz de:

    a) ***representarse*** (por exemplo na pantalla de texto, ou nunha pantalla gráfica),

    b) ***poñer unha ficha*** nunha **posición** si a xogada é posible lanzando un erro en caso contrario

    c) **indicar qué ficha** se atopa nunha posición indicada

    d) **devolver o número de fichas** que ten o taboleiro dunha cor determinada

3. Temos 64 **fichas**. Si lemos o enunciado vemos que en realidade temos un máximo de 64 xogadas entre os dous xogadores. Pero as fichas serven para calquera dos dous xogadores (son das dúas cores) e temos 64 celas, polo que en realidade se xoga ata que o taboleiro queda cheo ou ningún xogador ten xogada posible. *Se xoga mentres algún dos dous pode xogar*. As fichas poden ser brancas ou negras, pero as podemos representar numericamente, con 0, 1 ou 2.

4. Temos o Xogo (***Othello***), que consiste nun **taboleiro** e dous **xogadores**. O Xogo debería ser capaz de **xogar**. Xogar consiste en comezar sorteando as cores branco (1) ou negro (2) e poñendo no taboleiro as fichas iniciais. Unha vez feito isto, comezaría a xogar o xogador con negras. Mentres se poda xogar, o xogador en quenda realizaría a súa xogada e se cambiaría de quenda si o xogador contrario ten algunha xogada posible, se non, continuaría o mesmo xogador.

5. Cando se realiza unha xogada, se debe comprobar si é válida segundo as regras do xogo. Aquí podemos tomar dúas aproximacións:

    a) ***As regras do xogo son asunto do taboleiro***: Nese caso, o Xogo se limitaría a poñer a ficha (1,2) na posición indicada, e cambiaría de quenda si todo vai correctamente e o xogador contrario pode xogar. O taboleiro debería ter:

    - un método que realice a xogada segundo as regras do xogo e lance un erro en caso de que non sexa correcta.
    - un método que indique si un xogador ten algunha xogada posible ou non.
    - un método que poña o taboleiro na posición inicial do xogo

    Esta aproximación liga o taboleiro ao xogo, facendo que non se poda utilizar este obxecto en outros xogos (como Reversi ou 4 en raia...).

b) ***As regras do xogo son asunto do xogo***: Con esta aproximación o taboleiro sería máis simple, limitándose as tarefas básicas descritas anteriormente. O xogo debería ter :

- un método que realice a xogada segundo as regras do xogo e lance un erro en caso de que non sexa correcta.
- un método que indique si un xogador ten algunha xogada posible ou non.
- un método que poña o taboleiro na posición inicial do xogo
- un método que indique si é posible seguir xogando (cando algún dos dous pode xogar)

6. A **Posición** da xogada simplemente agrupará nun obxecto a fila e columna onde se desexa xogar. En todo caso, podería facer o axuste de numero (1 - 7) , letra (A – H) a coordenada (0-7),(0-7). Tamén podería proporcionar mediante sobrecarga a posibilidade de crear posicións das dúas formas. Internamente almacenaría sempre números a partir de 0 para as coordenadas.

Podemos deducir polo tanto as seguintes clases:

```
// Xogador – Os obxectos desta clase poden realizar xogadas
class Player {
        private String name;            // Nome do xogador
        private int color;              // Neste xogo pode ser 1 ou 2

        public Player(String name);     // Constructor
        public String getName();        // devolve o Nome
        public int getColor();          // devolve o Color
        public void setColor(int c);    // asigna un color ao xogador
        public Position doMovement();   // xogada, devolve a posición elexida
}

// Posición – Os obxectos de esta clase representan unha posición agrupando (fila e columna)
class Position {
        private int row;
        private int column;

        public Position(int row,char column);   // Constructor da forma (1,'A')
        public Position(int row,int column);    // Constructor sobrecargado da forma (0,3)
        public int getRow();                    // Devolve a fila
        public int getColumn();                 // Devolve a columna
}

// Taboleiro – Eleximos a a proximación de facer o tableiro simple, e poñer a lóxica na clase do xogo
class Board {
        private int[][] cells;          //  Para que sexa flexible, permitirá crear taboleiros de calqueira tamaño

        public Board(int nr,int nc);    // Constructor. Crea un taboleiro de nr x nc
        // Pon a cela indicada pola posición á cor indicada por color, si a posición é ilegal, lanza un erro
        public void put(Posicion p,int color) throws Exception;
        // devolve a cor da cela da posición p.
        public int get(Posicion p);
        public void print();            // Visualiza o tableiro
        public int count(int color);    // Devolve o número das fichas que ten o taboleiro da cor indicada
}
```

```
// Xogo – Implementa as regras do xogo, e conten o máin da aplicación
class Game {
        private Board b;
        private Player[] p;
        private int colorTurn;     // 1 ou 2 ...

        // Constructor. Crea o xogo para os xogadores indicados
        // - crea o tableiro e o pon na posición inicial
        // - establece o turno inicial
        public Game(Player p1, Player p2);
        public void runGame();  // Comenza o xogo: sortea as cores, inicializa o tableiro ... etc
        // Realiza a xogada na posición p do tableiro
        // lanza o erro en caso de xogada non legal segundo as normas do xoto
        public void play(Position p) throws Exception;
        public boolean canPlay(Player p);  // true si o xogador p pode xogar
        public boolean end();      // true si ningun xogador ten xogada posible (o xogo finaliza)

        private void init();    // Inicializa o taboleiro. E privada, porque so se usa internamente dende o constructor

}
```

A aplicación se lanzaría mediante o seguinte método main():

```
class RunOthello {
        public static void main(String[] args) {
                Player p1=new Player("Xogador 1");
                Player p2=new Player("Xogador 2");
                Game g=new Game(p1,p2);
                g.runGame();
        }
}
```

A aplicación presenta un *caso de uso* único, que é xogar unha partida:

- **O inicio da aplicación** (runGame) producirá o seguinte efecto: (poderíamos considerar como actor ao sistema, ou a persoa que lanza a aplicación)

  - Se crea un taboleiro de 8x8 e se pon a posición inicial

  - Se sortean as cores dos xogadores. A quenda inicial son as negras (1 ou 2, da igual...)

  - Mentras exista a posibilidade de xogar, o xogador da quenda actual realiza a xogada e se cambia de quenda

  - Se contan as fichas de cada cor e se determina o gañador.

- **A realización dunha xogada** por parte dun xogador prantexa dous casos de uso:
  - Que a xogada sexa correcta, en ese caso, se cambia de quenda
  - Que a xogada non sexa correcta, en ese caso se notifica e se continúa na mesma quenda.
- **O cambio de quenda**, tamén prantexa dous casos de uso distintos:
  - Que o xogador contrario non poda xogar, nese caso non se cambia de quenda
  - Que o xogador contrario si poda xogar, nese caso se cambia de quenda

O deseño completo en UML da aplicación sería máis longo e detallado e requeriría de varios diagramas, pero isto é suficiente para poder realizar a aplicación con garantías.

Si nos fixamos ben no deseño anterior, podemos darnos conta de que en realidade este esquema é válido para numerosos xogos de tableiro, variando únicamente pequenos detalles. O xogo do 4 en raia, por exemplo sería prácticamente idéntico, variando so a implementación das regras de xogo, ou o reversi, ou incluso o xadrez ou as damas. Simplemente os métodos se realizan de xeitos distintos, salvo o método “canPlay” que non sería necesario en xogos como o 4 en raia, damas... etc.

O deseño das clases sen indicar a implementación se denomina ***interface*** e se verá máis adiante no curso.