Tecnología de la Programación

Curso 2015-2016

Práctica 4: *Ataxx*

Descripción de la práctica

- **Objetivos: Genéricos y colecciones. Extensión de una aplicación existente. Iniciación a las técnicas de diseño de programas.**
- En esta práctica se va a extender una aplicación existente que se proporciona como parte del enunciado.
- La utilización de código existente tiene una función doble:
  - Permite la asimilación de algunos conceptos de diseño sin necesidad de hacer una aplicación desde cero.
  - Por otra parte el alumno deberá comprender y modificar un programa escrito por otro equipo de programación, que es el escenario habitual de un entorno de trabajo real.
- Este esquema básico lo utilizaremos en todas las prácticas de este cuatrimestre.

Estructura del programa inicial

- La aplicación que se proporciona contiene una implementación básica en modo texto de un programa para jugar a juegos de tablero.
- En particular, permite jugar a tres juegos muy sencillos: *Connect-N*, *Tic-Tac-Toe* y *Tic-Tac-Toe avanzado*.
- Se ha diseñado el programa para que sea fácilmente extensible con nuevos juegos, que es el objetivo de esta práctica.
- La aplicación proporcionada como base está contenida en el paquete `es.ucm.fdi.tp.basecode`.
- El método de inicio del programa está en la clase `es.ucm.fdi.tp.basecode.Main.java` y el resto del código tiene la siguiente estructura de paquetes:

Estructura del programa inicial

- `bgame`: Contiene las clases relacionadas con la gestión del juego, que son comunes a todos los juegos del programa. en particular, tiene tres componentes fundamentales:
  - `bgame.control`: Es el *controlador* del programa. En la aplicación actual en modo texto hay dos controladores: el primero utiliza el esquema tradicional que se ha visto en el primer cuatrimestre, y el segundo sigue el patrón de diseño *Modelo-Vista-Controlador* (*MVC*).
  - `bgame.model`: Contiene todas las clases relacionadas con el *modelo* de MVC: el tablero (`Board`), la partida (`Game`), además de diversos interfaces y clases abstractas.
  - `bgame.views`: Contiene las clases relacionadas con las *vistas* del patrón MVC. En la aplicación proporcionada solo hay una clase genérica para las operaciones de consola.
- `connectN`, `ttt` y `attt`: Estos paquetes contienen las clases relacionadas con tres juegos que se proporcionan en la aplicación: *Connect-N*, *Tic-Tac-Toe* y *Tic-Tac-Toe avanzado*.

La clase `Main`

- El programa inicia su ejecución en un método `main` definido en la clase `es.ucm.fdi.tp.basecode.Main.java`.
- En esta clase se incluye código para tratar los argumentos pasados en la línea de órdenes.
- En concreto, el usuario podrá utilizar los siguientes argumentos:
  - `-h` o `--help`: solicita a la aplicación que muestre la *ayuda* sobre cómo utilizar los argumentos de la línea de órdenes.
  - `-d` o `--dim`: indica el tamaño del tablero (si el juego lo permite). A continuación se debe indicar el tamaño utilizando el formato `ROWSxCOLS`. Si no se indica, se creará un tablero de un tamaño por defecto que depende del juego.
  - `-g` o `--game`: indica el tipo de juego. Debe ir seguido del tipo de juego: `cn` para jugar a *Connect-N*, `ttt` para *Tic-Tac-Toe* y `attt` para *Tic-Tac-Toe avanzado*. Si no se indica nada, se jugará a *Connect-N*.

La clase `Main`

- `-m` o `--multiviews` indica si se utilizarán múltiples vistas. No aplicable en esta práctica.
- `-v` o `--view` indica el interfaz de usuario. No aplicable en esta práctica. Debe ir seguido de `console` para interfaz de texto, o `window` para interfaz gráfica. Por defecto, `console`.
- `-p` o `--players` seguido de una lista de jugadores.
  - ★ Esta lista estará formada por una secuencia de elementos separados por comas (sin espacios en blanco).
  - ★ Cada elemento puede ser bien un carácter `A` que corresponde con el identificador de las fichas del jugador,
  - ★ o bien un par `A:B` en el que el primer elemento es el identificador de las fichas del jugador y el segundo elemento es el *modo de juego* de ese jugador.
  - ★ Los modos de juego pueden ser `m` (manual), `r` (*random*) o `a` (*AI*, jugador automático inteligente).

La clase `Main`

```
usage: es.ucm.fdi.tp.basecode.Main [-d <arg>] [-g <game identifier>]
       [-h] [-m] [-p <list of players>] [-v <view identifier>]
```

`-d,--dim <arg>`	The number of rows in the board (if allowed by the selected game). It must has the form ROWSxCOLS.
`-g,--game <game identifier>`	The game to play (cn [for ConnectN] ttt [for Tic-Tac-Toe] attt [for Advanced Tic-Tac-Toe] ataxx [for Ataxx]). By default, cn.
`-h,--help`	Print this message
`-m,--multiviews`	Create a separate view for each player (valid only when using the window view)
`-p,--players <list of players>`	A player has the form A:B (or A), where A is sequence of characters (without any whitespace) to be used for the piece identifier, and B is the player mode (m [for Manual] r [for Random] a [for AI]). If B is not given, the default mode 'm' is used. If this option is not given a default list of pieces from the corresponding game is used, each assigmed the mode 'm'.
`-v,--view <view identifier>`	The view to use (window [for Swing] console [for Console]). By default, console.

La clase `Main`

- Por ejemplo, para jugar a *Connect-N* en un tablero 7x7 con dos jugadores, siendo el primero un jugador aleatorio con fichas identificadas por `'W'` y el segundo un jugador manual con fichas identificadas por `'B'`, se debe utilizar la siguiente secuencia de argumentos:

```
-g cn -d 7x7 -p W:r,B:m
```

- Para el análisis de los argumentos se utiliza *Apache Commons CLI*, una de las biblioteca disponibles para tratar la línea de órdenes en Java. Puedes encontrar toda la documentación que necesites sobre esta biblioteca en:

```
https://commons.apache.org/proper/commons-cli/
```

Descripción de las clases de un juego: *Connect-N*

- Para poder extender el programa base con un juego nuevo, se deben definir (o reutilizar) algunas clases que implementan los distintos elementos que forman el código del juego. Lo vemos por ejemplo con el juego *Connect-N*.
- Las clases que definen el juego están contenidas en el paquete `es.ucm.fdi.tp.basecode.connectN` y son las siguientes:
  - `ConnectNFactory`: Es lo que se denomina una *factoría*, una clase que contiene métodos para crear los distintos objetos del juego. Debe ser subclase (directa o indirectamente) de `GameFactory`, definida en `es.ucm.fdi.tp.basecode.bgame.control`.
  - `ConnectNMove`: Es la clase que implementa un movimiento de un jugador sobre un tablero. Debe ser subclase de `GameMove`.
  - `ConnectNRandomPlayer`: Es la clase que implementa el funcionamiento de un jugador que realiza movimientos aleatorios para este juego. Debe ser subclase de `Player`.

Descripción de las clases de un juego: *Connect-N*

  - `ConnectNRules`: Contiene las reglas del juego y realiza la actualización del tablero como resultado de la aplicación de un movimiento. Debe ser subclase de `GameMove`.

- Cuando se crea un nuevo juego, se deben considerar estas clases, implementándolas completamente o adaptando (mediante herencia) las clases de otros juegos existentes.
- También se pueden reutilizar directamente clases de otros juegos. Por ejemplo, el juego *Tic-Tac-Toe* reutiliza directamente las clases `ConnectNMove` y `ConnectNRandomPlayer` del juego *Connect-N*.
- El juego *Tic-Tac-Toe avanzado* adapta mediante herencia algunas clases de *Connect-N* (`ConnectNMove` y `ConnectNRandomPlayer`) y otras de *Tic-Tac-Toe* (`TicTacToeFactory` y `TicTacToeRules`).

Descripción del juego *Ataxx*

- El objetivo de la práctica consiste en la implementación de un nuevo juego sobre el código proporcionado, que se debe utilizar como base.
- El juego a implementar es *Ataxx*, un juego de tablero creado en los años 90 para diversos ordenadores domésticos.
- Se puede encontrar una descripción del juego en la siguiente página web: `https://en.wikipedia.org/wiki/Ataxx`.
- El juego que se debe implementar en esta práctica es una variante en la que pueden jugar hasta cuatro jugadores y el tamaño del tablero puede ser diferente.

Disposición inicial del tablero

- En la versión básica del juego hay dos jugadores, cada uno con fichas de un color diferente (blanco y negro, identificados por los caracteres 'X' y 'O').
- Los jugadores se enfrentan en un tablero cuadrado 7x7.
- Para esta práctica, el tamaño del tablero puede cambiar (mínimo 5x5, dimensiones con valor impar) y podrán jugar dos o más de dos jugadores (hasta 4).
- Inicialmente, cada jugador tiene colocadas en el tablero dos fichas en esquinas opuestas: el jugador blanco en las esquinas superior izquierda e inferior derecha.

Disposición inicial del tablero

- El tablero inicial, por tanto, se puede representar de la siguiente forma (las casillas vacías se representan con un punto):

```
   +---------------+
0  | X . . . . . O |
1  | . . . . . . . |
2  | . . . . . . . |
3  | . . . . . . . |
4  | . . . . . . . |
5  | . . . . . . . |
6  | O . . . . . X |
   +---------------+
     0 1 2 3 4 5 6
```

Disposición inicial del tablero

- Si participan 3 o 4 jugadores, el tablero inicial tendrá las siguientes disposiciones (las fichas de los nuevos jugadores se representan con los caracteres 'R' y 'B'):

```
   +---------------+      +---------------+
0  | X . . . . . O |   0  | X . . B . . O |
1  | . . . . . . . |   1  | . . . . . . . |
2  | . . . . . . . |   2  | . . . . . . . |
3  | R . . . . . R |   3  | R . . . . . R |
4  | . . . . . . . |   4  | . . . . . . . |
5  | . . . . . . . |   5  | . . . . . . . |
6  | O . . . . . X |   6  | O . . B . . X |
   +---------------+      +---------------+
     0 1 2 3 4 5 6          0 1 2 3 4 5 6
```

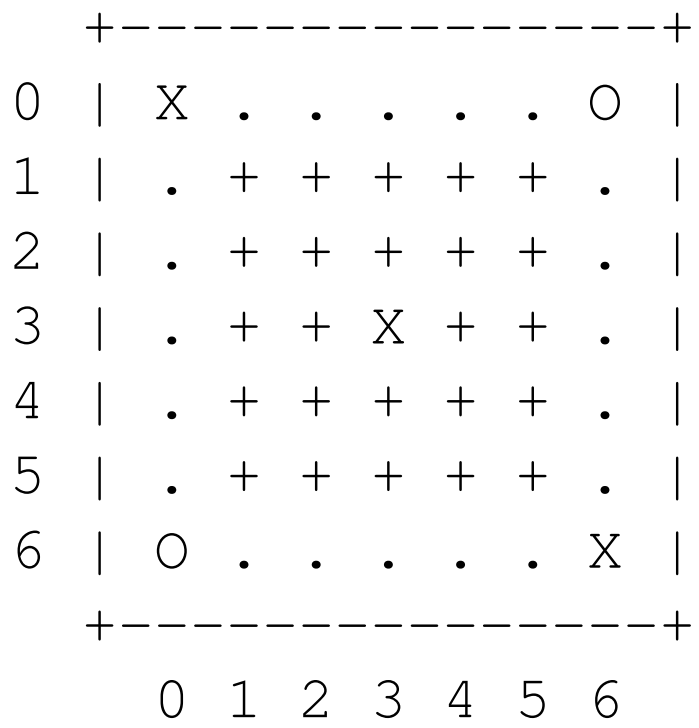
- En los siguientes apartados utilizaremos dos jugadores, aunque las reglas son extensibles a 3 o 4 jugadores.

Movimientos

- El primer jugador de la lista de jugadores es el primero en mover, y se alterna el turno de juego entre los jugadores mientas sea posible.
- Si hay más de dos jugadores:
  - El turno irá rotando entre los jugadores en el orden que se fija en el parámetro `-p` de la línea de órdenes.
  - Después del último jugador de la lista pasará el turno al primero.
  - Si un jugador no puede mover, se pasa el turno al siguiente jugador en el mismo orden.
- En cada turno, los jugadores mueven una ficha una o dos casillas en cualquier dirección. En total, una ficha se puede mover a un máximo de 8+16 posiciones distintas: las 8 casillas adyacentes, más las 16 casillas situadas a una distancia máxima de dos casillas.

Movimientos

- En la siguiente figura se muestran con + las casillas a las que puede mover una ficha blanca situada en el centro del tablero:

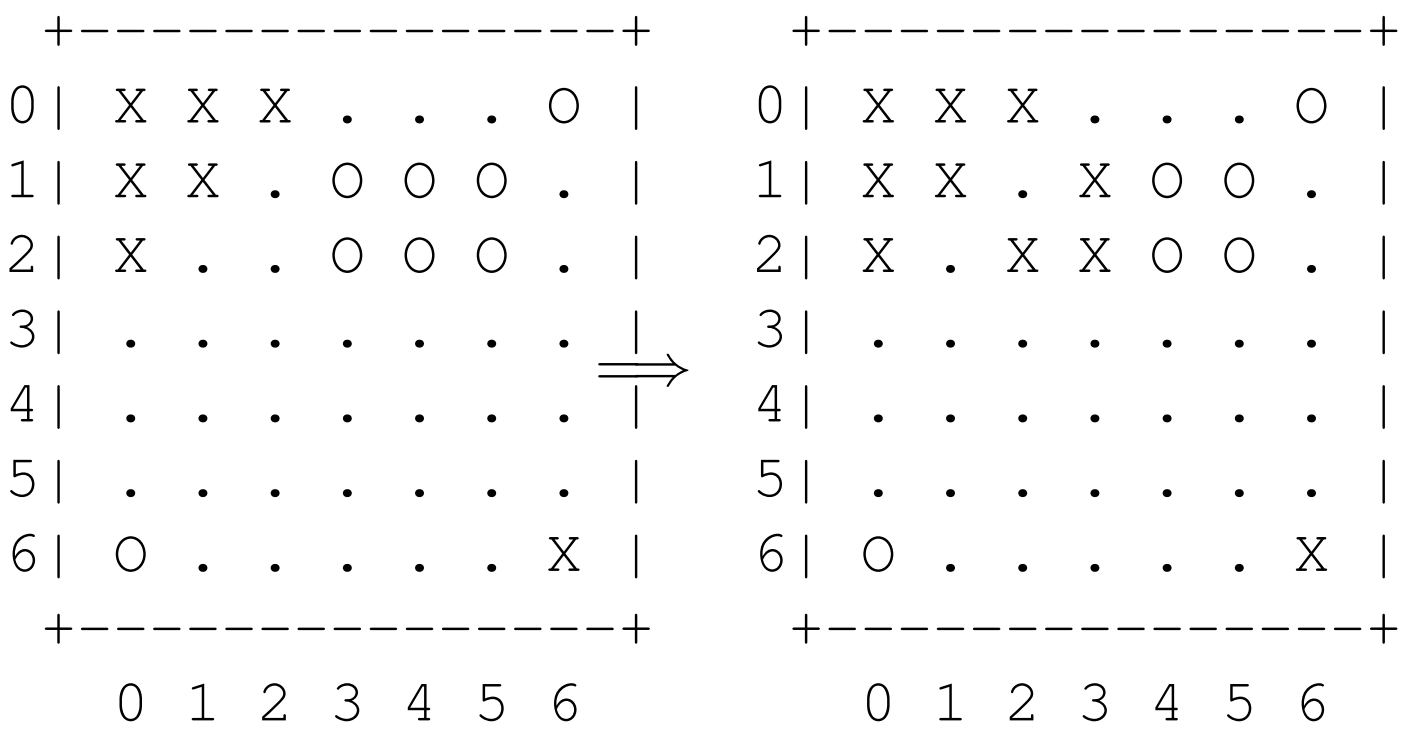


Movimientos

- Si la casilla de destino es adyacente a la casilla de origen, se crea una nueva ficha en la casilla de destino. En caso contrario, la ficha se mueve a la casilla de destino.
- La casilla de destino debe estar vacía, pero las casillas intermedias entre el origen y el destino pueden estar ocupadas.
- Después del movimiento, todas las fichas de los oponentes adyacentes a la casilla de destino se convierten al color del jugador.
- Los jugadores deben mover obligatoriamente siempre que sea posible. En caso contrario, deben pasar el turno al siguiente jugador.
- El juego termina cuando todos los jugadores excepto uno se han quedado sin fichas, o bien cuando ningún jugador puede mover.
- El ganador es el jugador con más fichas. Si hay dos o más jugadores con el máximo número de fichas la partida termina en tablas.

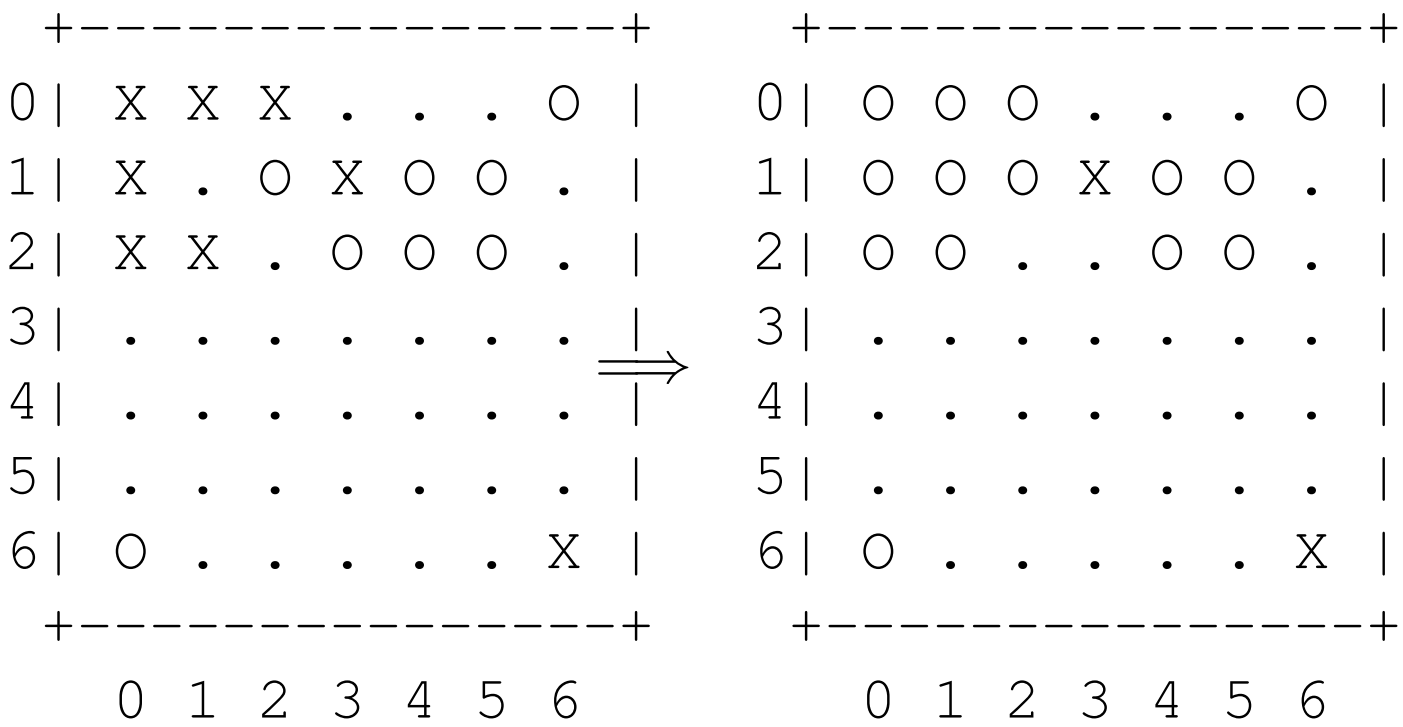
Movimientos

- En el siguiente ejemplo de movimiento, la ficha blanca ('X') que está situada en la posición $\langle 1, 1 \rangle$ se mueve a la posición $\langle 2, 2 \rangle$, cambiando de color las fichas de las posiciones $\langle 1, 3 \rangle$ y $\langle 2, 3 \rangle$, y creándose una ficha en la posición de origen:



Movimientos

- A continuación se muestra otro ejemplo en el que la ficha negra ('○') situada en la posición $\langle 2, 3 \rangle$ se mueve a la posición $\langle 1, 1 \rangle$, cambiando de color las fichas de las posiciones $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$, $\langle 0, 2 \rangle$, $\langle 1, 0 \rangle$, $\langle 2, 0 \rangle$, y $\langle 2, 1 \rangle$. En este caso no se crea una ficha en la casilla de origen, pues la casilla de destino no es adyacente a ella:

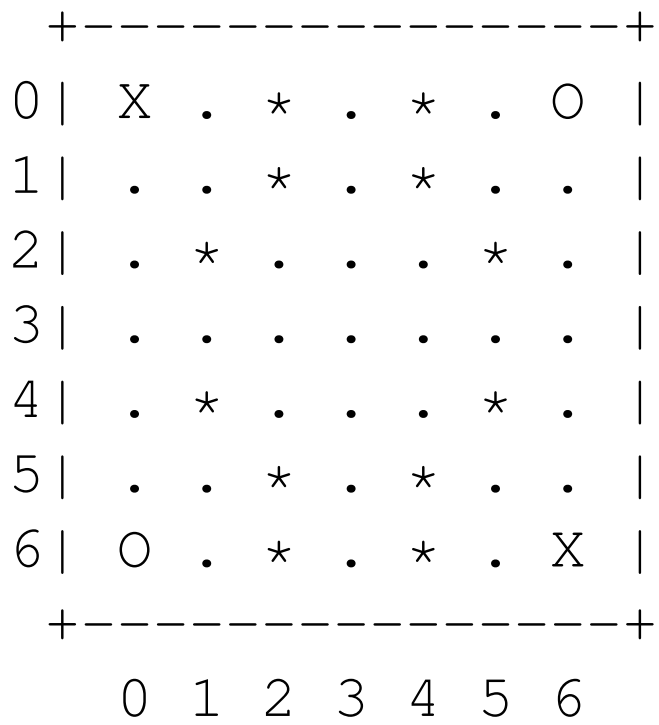


Obstáculos en el tablero

- En los ejemplos anteriores todas las casillas están disponibles para colocar fichas.
- La configuración del tablero puede modificarse considerando ciertas casillas ocupadas (por fichas que no pertenecen a ningún jugador) y con diferentes tamaños del tablero.
- En el juego que se debe implementar se de poder especificar con un nuevo argumento ($-\circ$) el número de casillas ocupadas por obstáculos. Estas casillas serán elegidas aleatoriamente.
- Las casillas iniciales ocupadas se mostrarán mediante el carácter '$\star$'.
- Una alternativa es elegir aleatoriamente algunas de las casillas y replicarlas de forma que produzcan una figura geométrica.

Obstáculos en el tablero

- Por ejemplo, si se eligen aleatoriamente 3 casillas para situar obstáculos, se puede obtener la siguiente figura replicando estas casillas en los cuatro cuadrantes:



Clases que hay que implementar o modificar

- En esta práctica se debe extender el programa proporcionado con un nuevo juego: *Ataxx*.
- Crea un nuevo paquete `es.ucm.fdi.tp.practica4.ataxx` en el que se deben incluir las clases necesarias para que funcione el nuevo juego.
- Investiga cómo están implementados los demás juegos para utilizar el mismo diseño.
- Además, debes adaptar la clase `Main` para que se pueda jugar al nuevo juego con la opción `-g ataxx` (y se pueda seguir jugando a todos los demás).
- Copia la clase `es.ucm.fdi.tp.basecode.Main.java` en el paquete `es.ucm.fdi.tp.practica4` y modifica esta nueva clase para que se pueda jugar al nuevo juego solicitándolo desde la línea de órdenes.
- Esta nueva clase debe incluir la posibilidad de utilizar la opción `-o` (o `--obstacles`) para especificar el número de obstáculos que aparecerán en el tablero.

No se debe modificar ninguna clase de los paquetes que están en `es.ucm.fdi.tp.basecode`.

Entrega de la práctica

- La práctica debe entregarse utilizando el mecanismo de entregas del campus virtual.
- La práctica debe entregarse no más tarde de la fecha y hora indicada en la cabecera de la práctica: **11 de marzo de 2016 a las 16:00h.**
- El fichero debe tener al menos el siguiente contenido:
  - Directorio `src` con el código de todas las clases de la práctica.
  - Fichero `alumnos.txt` donde se indicará el nombre de los componentes del grupo.
  - Directorio `doc` con la documentación generada automáticamente sobre la práctica.