Práctica 6: Servidor de Juego

Fecha de entrega: 20 de mayo de 2016 a las 16:00

Objetivo: Aplicación cliente-Servidor utilizando sockets. Serialización de objetos. Programación multihebra.

Antes de comenzar, crea un paquete practica6 y copia Main.java de practica5 a practica6. Todo el código de esta práctica debe estar definido dentro de practica6. Te recordamos que está estrictamente prohibido modificar ningún fichero del paquete basecode. Además, te recomendamos que no modifiques nada de practica4 o practica5, excepto si vas a corregir algún error de programación.

1. Servidor de juego

En esta práctica debes extender la aplicación de juegos de tablero para proporcionar la posibilidad de jugar a través de la red, de manera que varios jugadores desde distintos ordenadores puedan jugar entre ellos a un juego determinado. En particular, debes en primer lugar desarrollar un servidor de juego que se inicie en una máquina y dé un servicio "*para jugar a un juego específico*", y en segundo lugar, desarrollar clientes que se puedan conectar al servidor desde otras máquinas para jugar a ese juego (en modo ventana) entre ellos. La nueva funcionalidad debe ser completamente transparente a las vistas desarrolladas en la práctica 5, de forma que el código de la práctica 5 no se modifique en absoluto.

La aplicación se inicia en modo cliente o modo servidor dependiendo de las opciones de la línea de órdenes. Por ejemplo, para iniciar el servidor que permita jugar a *Ataxx* con dos jugadores podemos utilizar la siguiente línea de órdenes:

```
java -jar pr6.jar -g ataax -p X,O --app-mode server --server-port 4000
```

Este comando inicia un servidor de juego en el puerto 4000 para jugar a *Ataxx* con dos jugadores `X` y `O` (si se omite la lista de jugadores se utiliza la lista por defecto, como en las prácticas anteriores). En cuanto se inicia el servidor, los clientes pueden conectarse a él para jugar al mismo juego utilizando la línea de órdenes:

```
java -jar pr6.jar --app-mode client --server-host hostname --server-port 4000
```

donde `hostname` es el nombre de la máquina en la que se está ejecutando el servidor. A cada cliente se le asigna una única ficha de la lista de fichas (que se proporciona en la opción `-p` que inicia el servidor). En cuanto el número de clientes conectados al servidor coincide con el número de jugadores esperados (es decir, el número de fichas), el servidor inicia el juego. A continuación, los clientes pueden jugar entre ellos de la siguiente forma: (1) los clientes envían sus acciones (por ejemplo, sus movimientos) al servidor, que a su vez los reenvía al juego correspondiente; y (2) las notificaciones de GameObserver que lanza el juego deben ser reenviadas por el servidor a los clientes. Una vez comenzado el juego, si algún cliente intenta conectarse al servidor debe ser rechazado. Cuando termina el juego, el servidor debe permitir de nuevo conectarse a los clientes para comenzar un nuevo juego. El servidor debe mostrar una ventana Swing sencilla en la que se muestren mensajes informativos cuando se conectan los clientes, cuando se inicia el juego, etc. Esta ventana también debe tener un botón que permita parar el servidor y salir de la aplicación.

Debes utilizar las siguientes opciones de la línea de órdenes para la nueva funcionalidad:

- --app-mode o -am: indica el modo en el que se quiere iniciar la aplicación de juego. Toma uno de los siguientes valores: `normal`, `client` o `server`. Si no se proporciona esta opción, por defecto debe ser `normal`.
- --server-port o -sp: indica el puerto sobre el que debe iniciarse el servidor (cuando se utiliza con la opción -am server) o el puerto en el que estará escuchando el servidor y al que debe conectarse el cliente (cuando se utiliza con la opción -am client). Si no se proporciona esta opción, el valor por defecto debe ser 2000.
- --server-host o -sh: requiere un parámetro que corresponde con el nombre (o dirección IP) de la máquina sobre la que se está ejecutando el servidor (es decir, se debe utilizar con la opción -am client). Si no se proporciona esta opción, el valor por defecto debe ser "localhost".

2. OPCIONAL: Jugadores automáticos en la vista de ventana

2.1. Parte I

En la versión anterior de basecode los jugadores automáticos estaban programados para esperar unos milisegundos y después devolver un movimiento aleatorio (de ahí el nombre DummyAIPlayer). La nueva versión de basecode tiene una nueva funcionalidad que permite utilizar el algoritmo `MinMax` para realizar movimientos automáticos. Este algoritmo explora el espacio de estados y decide el mejor movimiento para el jugador al que le toca el turno – se basa en el método evaluate de la clase GameRules correspondiente para evaluar lo bueno que es un estado del juego. Puedes ver MinMax.java para ver los detalles. Observa que basecode ya tiene una versión de evaluate implementada para *Connect-N* (que también se utiliza para *Tic-Tac-Toe* y para *Advanced Tic-Tac-Toe*). Se deben utilizar las siguientes opciones de la línea de órdenes para activar esta funcionalidad:

- --ai-algorithm o -aialg: indica el algoritmo que queremos utilizar para el jugador automático. Puede ser minmax para utilizar el algoritmo `MinMax`, minmaxab para utilizar el algoritmo `MinMax` con poda $\alpha-\beta$ (que es más eficiente), o none para utilizar el jugador "dummy" automático. Si no se proporciona esta opción se debe considerar por defecto none.

- --minmax-depth o -md: indica la profundidad máxima del árbol de búsqueda construido por los algoritmos `MinMax`. Valores más altos resultan en algoritmos más "*listos*" pero menos eficientes (pues el árbol que se debe recorrer es más grande). El valor por defecto es 3.

En esta parte debes copiar esta nueva funcionalidad de Main.java de basecode al fichero Main.java que está en el paquete de tu práctica (sólo debes copiar los métodos que procesan las opciones de línea de órdenes anteriores). También debes implementar el método evaluate para el juego *Ataxx*. Por ejemplo, puedes utilizar la siguiente estrategia *naïf*:

```
public double evaluate(Board board, ..., Piece p) {

   double m = 0;
   double n = 0;
   ...
   // 1. 'n' contiene el numero de fichas de
   //    tipo 'p' en el tablero 'board'
   // 1. 'm' contiene el numero de fichas de
   //    tipo distinto de 'p' en el tablero 'board'
   //    (sin contar los obstaculos)

   double total = n+m;
   return (n / total) - (m / total);
}
```

2.2. Parte II

Cuando en la práctica 5 se realiza un movimiento automático puedes observar que la ventana deja de responder a las acciones del usuario hasta que termina el movimiento. Esto es así porque el jugador automático se "*duerme*" durante un tiempo (para simular un cálculo que tarda algún tiempo) y después devuelve un movimiento aleatorio. Mientras está durmiendo, la hebra de gestión de eventos de Swing se bloquea por lo que el interfaz gráfico deja de responder a las acciones del usuario.

En esta parte debes mejorar este comportamiento, cambiando la vista de ventana de forma que las llamadas a ctrl.makeMove(p) se realicen en una hebra que sea diferente de la hebra de gestión de eventos de Swing. Esto se puede realizar creando una nueva hebra que ejecute la llamada a ctrl.makeMove(p), o preferiblemente utilizando un *SwingWorker* (para evitar crear una nueva hebra en cada llamada).

2.3. Parte III

En esta tercera parte debes cambiar el comportamiento de los movimientos automáticos (solo en la vista de ventana) para que hagan lo siguiente: si el jugador automático no devuelve un movimiento en X segundos, se lanza un error y el modo del jugador que tiene el turno se cambia de *automático* a *manual*. El valor de X puede estar predefinido (por ejemplo, 5 segundos) o puedes añadir un componente Swing a la vista donde el usuario pueda controlar este valor.

Para la implementación de esta parte puedes necesitar saber que los jugadores *automáticos* (el jugador "*dummy*" y el que utiliza el algoritmo `MinMax`) devuelven null inmediatamente si la hebra en la que se ejecutan es interrumpida –puedes verlo en DummyAIPlayer.java y AIPlayer.java.

3. Entrega de la práctica

La práctica debe entregarse en un único archivo comprimido utilizando el mecanismo de entregas del campus virtual, no más tarde de la fecha y hora indicada en la cabecera de la práctica. El fichero debe tener al menos el siguiente contenido:

- Directorio src con el código fuente del programa.
- Fichero alumnos.txt donde se indicará el nombre de los componentes del grupo.
- Directorio doc con la documentación generada automáticamente sobre el código que has implementado.