

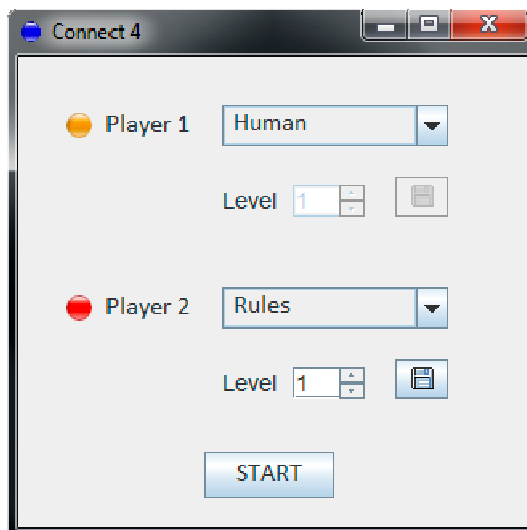
INTERFAZ

CUATRO EN RAYA - Por: Daniel Compán López de Lacalle


Se ha desarrollado una sencilla e intuitiva interfaz grafica para interactuar con el programa. Aunque también se puede jugar por consola usando el método `consolePlay()`:

```
IA - 4 en Raya
=====
Escoja jugador 1:
 1 - Human.
 2 - Random.
 3 - Rules.
 4 - Minimax.
¿Eleccion? (1~4)
1
Jugador 1 listo.
Escoja jugador 2:
 1 - Human.
 2 - Random.
 3 - Rules.
 4 - Minimax.
¿Eleccion? (1~4)
3
¿Nivel? (1~3)
2
Archivo de reglas "rules.txt" (37 líneas) leído.
Elementos de ficha propia (0) y ficha del oponente (X) intercambiados en las reglas.
Jugador 2 listo.
1234567
+-----+
|         |
|         |
|         |
|         |
|         |
|         |
|         |
+-----+
1234567
Primer movimiento para el jugador 1 (Humano) [0].
¿Columna? (1~7)
4
1 - El jugador 1 (Humano) [0] juega en la columna 4.
1234567
+-----+
|         |
|         |
|         |
|         |
|         |
|         |
|         |
+-----+
1234567
PUNTUACIONES:
Columna 1: -50
Columna 2: -40
Columna 3: -40
Columna 4: 0
Columna 5: -40
Columna 6: -40
Columna 7: -50
Puntuacion mas alta: 0
2 - El jugador 2 (Ruler) [X] juega en la columna 4.
1234567
+-----+
|         |
|         |
|         |
|         |
|         |
|         |
|         |
+-----+
1234567
¿Columna? (1~7)
```

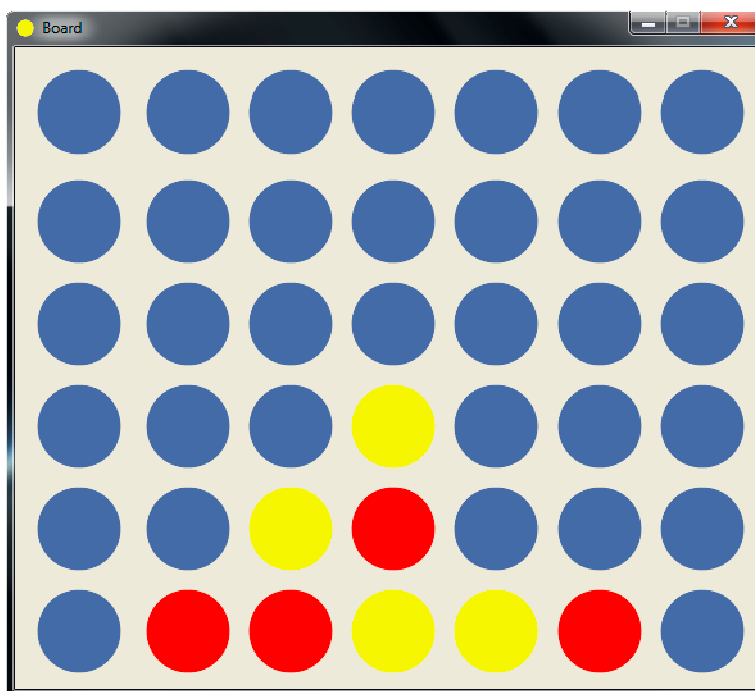
En la ventana principal se presenta el menú:



Según el jugador que escojamos se habilitaran las opciones de seleccionar nivel (si tiene varios) y archivo de reglas (en el caso del jugador minimax para la heurística).

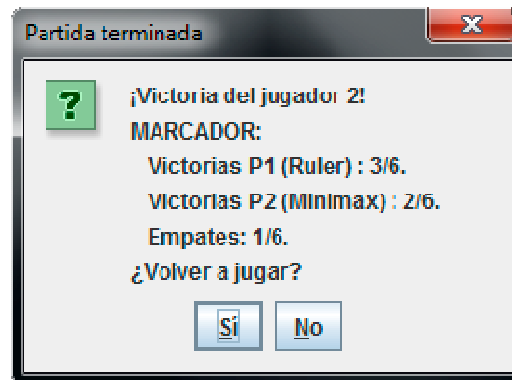
El botón  abre el explorador de archivos que nos permite elegir el que queramos para las reglas. Si no seleccionamos ninguno, se usarían los que hay por defecto. Los archivos elegidos son independientes para los jugadores 1 y 2. También para el tipo de jugador. Esto es, en caso de que elijamos uno, y luego cambiemos de tipo de jugador, no se tomará el archivo antes seleccionado a no ser que se vuelva a elegir el jugador con el que se seleccionó.

Pulsando "START" comienza la partida, y se abre una nueva ventana con el tablero.



El icono de la parte superior izquierda cambia de color en función del jugador al que le toque mover. Si se trata de un jugador humano, basta con hacer clic con el ratón en la columna deseada.

Cuando la partida termine se mostrará un cuadro de dialogo en el que aparecerá el marcador y se preguntará si se desea volver a jugar.

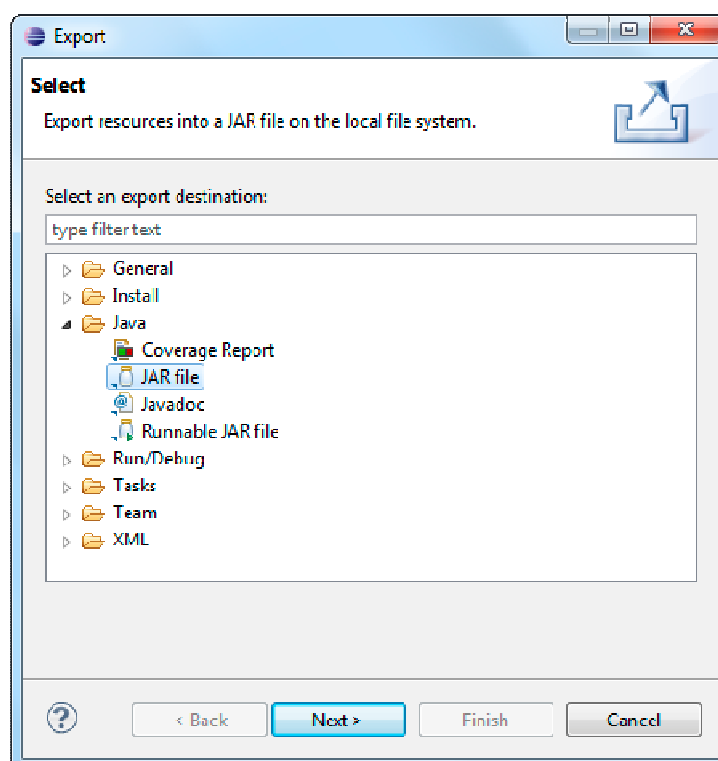


Generar archivo JAR:

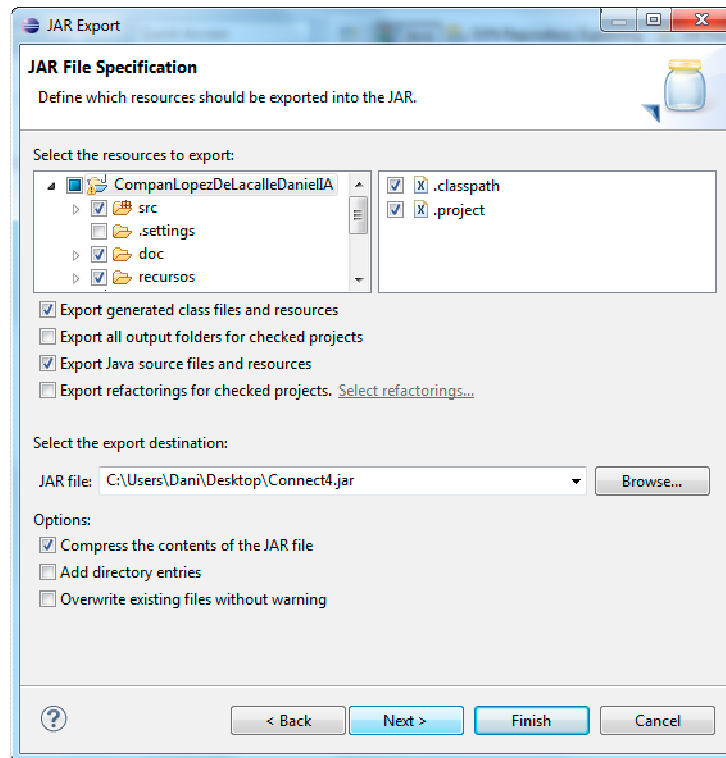
Los archivos de imágenes y reglas por defecto, se encuentran empaquetados en el archivo .JAR. Cuando esto se hace así la forma de acceder a ellos en el código cambia (se llaman como recursos). Para no tener que hacer muchas modificaciones en el código, si se desea obtener el archivo .JAR, se ha creado una constante a la que hay que darle el valor `true`. Se llama TOJAR y aparece en las clases `Connect4`, `WindowsBoard` y `Rule`. En todas ellas hay que cambiarla:

```
32 public class Connect4 extends JFrame {  
33  
34     /** Si se va a empaquetar en un archivo .JAR ha de ponerse a <code>true</code> */  
35     public static final boolean TOJAR=true;  
36
```

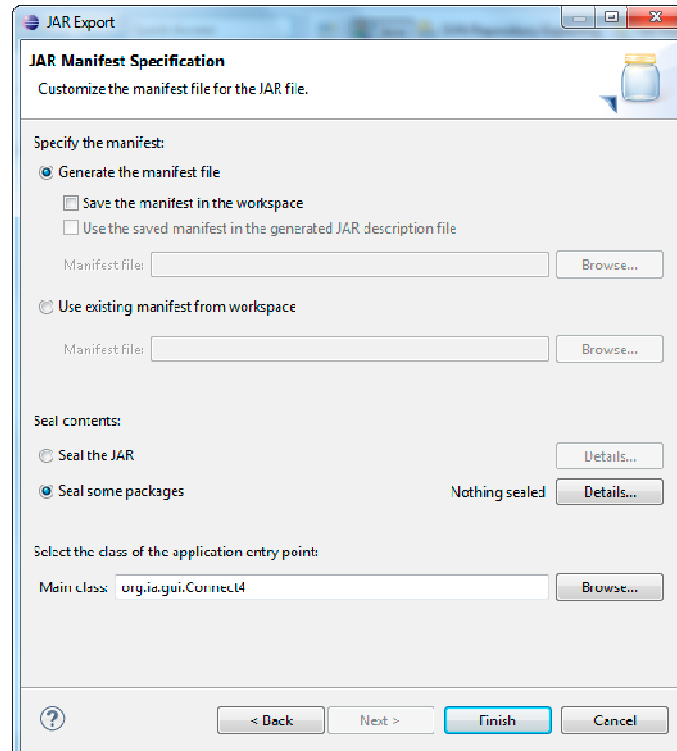
Seguidamente en Eclipse vaya a File > Export...



Seleccionar "*JAR File*" (y no "*Runnable JAR File*") porque esta opción nos permite añadir recursos. Añadimos los archivos fuente y los recursos:



Por ultimo, especificamos el nombre de la clase con el método main que iniciará el programa:



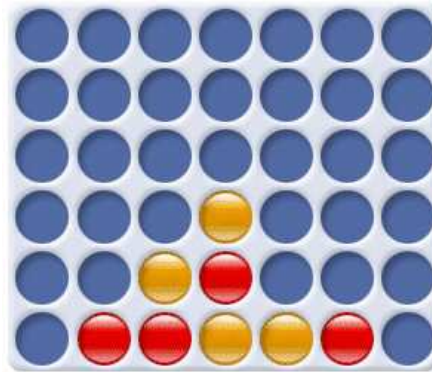
Con esto nos generará un sencillo archivo MANIFEST.MF:

```
Manifest-Version: 1.0
Class-Path: .
Main-Class: org.ia.gui.Connect4
```

PRACTICA 1

CREACIÓN DE UN SISTEMA DE PRODUCCIÓN

El sistema de producción se diseñará para el juego de las 4 en ralla:



El objetivo del juego es colocar 4 fichas del mismo color en línea. Los jugadores van colocando su fichas alternativamente en una de las columnas y estas caen hasta la última posición libre.

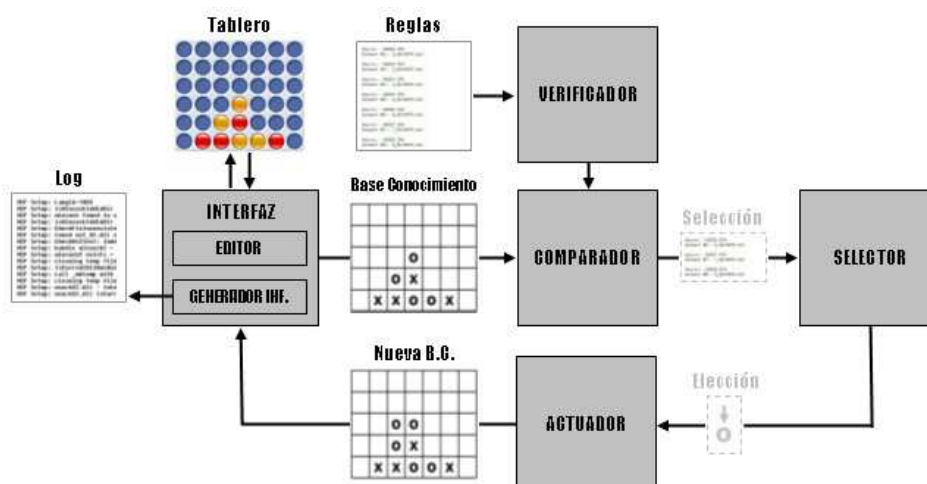
El sistema de producción esta formado por los siguientes componentes:

- Un Conjunto de reglas.
- Una base de conocimiento.
- Una estrategia de control.
- Un aplicador de reglas.

Además, también ha de implementarse:

- Interfaz para jugadas.
- Editor de tablero.
- Generador de informe.

Un esquema simplificado:



BASE DE CONOCIMIENTO

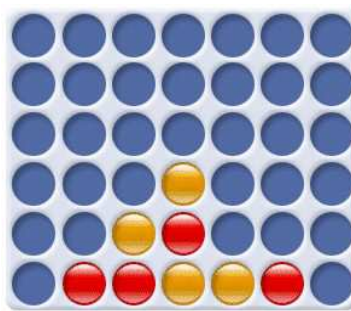
En la BC (base de conocimiento) está contenida toda la información del sistema necesaria para poder tomar decisiones y actuar sobre él. Los elementos que integra el sistema están representados en ella de forma simbólica con el objetivo de facilitar la computación.

La base de conocimiento puede implementarse con una matriz de tamaño 6×7, ya que el tablero de juego tiene 6 filas y 7 columnas.

Los elementos que pueden aparecer en el tablero son:

- O** : Ficha propia.
- X** : Ficha del oponente.
- E** : Casilla vacía.

De manera que para este tablero:



Tendríamos la siguiente base de conocimiento para la interfaz:

	0	1	2	3	4	5	6
0	E	E	E	E	E	E	E
1	E	E	E	E	E	E	E
2	E	E	E	E	E	E	E
3	E	E	E	O	E	E	E
4	E	E	O	X	E	E	E
5	E	X	X	O	O	X	E

Si embargo, con las reglas usaremos una base de conocimiento aumentada (BCA):

	0	1	2	3	4	5	6	7	8
0	I	I	I	I	I	I	I	I	I
1	I	Z	Z	Z	Z	Z	Z	Z	I
2	I	Z	Z	Z	Z	Z	Z	Z	I
3	I	Z	Z	Z	_	Z	Z	Z	I
4	I	Z	Z	_	O	Z	Z	Z	I
5	I	Z	_	O	X	_	_	Z	I
6	I	_	X	X	O	O	X	_	I
7	I	I	I	I	I	I	I	I	I

REGLAS

Una regla es una línea de texto que tiene 3 partes básicas separadas por, al menos, un espacio en blanco:

Secuencia [Dirección] Valor

Determinar una tirada consiste en elegir una columna. En nuestro caso escogeremos la que tenga mayor **puntuación**. Para determinar la puntuación de cada columna usaremos las reglas. Cada regla añadirá su *valor* a la puntuación de las columnas que cumplan su *secuencia*.

Elementos:

Las reglas están pensadas para encontrar una secuencia de elementos en la BC. Estos elementos pueden ser los que están presentes en la BCA:

- O** : Ficha propia.
- X** : Ficha del oponente.
- _** : Casilla vacía jugable.
- z** : Casilla vacía no jugable.
- I** : Pared (limite del tablero).

Y también estos otros:

- E** : Casilla vacía (de cualquier tipo: '_', 'z').
- #** : Casilla vacía jugable elegida.
- =** : Nueva casilla vacía jugable elegida (por el oponente).
- ?** : Cualquier elemento.

Secuencia:

Una *secuencia de búsqueda* es una serie de elementos consecutivos indicados en una regla. Para que una secuencia se cumpla ha de haber coincidencia entre sus elementos y los de otra *secuencia de prueba* tomada de la BCA. Aunque la *secuencia de prueba* sea mayor, solo se comparan los elementos que se corresponden con los de la *secuencia de búsqueda*.

Cuando evaluamos nuestra jugada:

En la *secuencia de búsqueda* aparece una sola vez el elementos '#' (y nunca '='). Este elemento ha de coincidir con un el elemento '_' de la *secuencia de prueba*.

La *secuencia de prueba* se toma de la BCA. Además, el elemento '_' de la columna que se esta evaluando ha de estar en la misma posición que '#' en todas las secuencias que se tomen de la misma columna. A esto lo llamaremos *posición de sincronización* de la *secuencia de búsqueda* con la *secuencia de prueba*.

Ejemplos:

Secuencia de búsqueda:	#zz	#Oz	_#_	_#E	EEE#	EEE#	#?O	#_E	E_#O	##	#=
Secuencia de prueba:	<u>zz</u>	<u>Oz</u>	<u> </u>	<u> z</u>	<u>z </u>	<u>zzz</u>	<u>_XO</u>	<u>_zz</u>	<u>OO_X</u>		

Cuando evaluamos la próxima jugada de nuestro oponente:

Cuando queremos ver que pasaría si yo coloco mi ficha en una columna y mi oponente juega en la misma columna que yo uso el elemento '=' en vez de '#':

En la *secuencia de búsqueda* aparece una sola vez el elementos '=' (y nunca '#'). Igual que antes, ha de coincidir con el elemento '_' de la *secuencia de prueba*.

La secuencia de prueba se obtiene también de la misma base de conocimiento aumentada (BCA). Si bien se tienen presentes las dos únicas posiciones que cambian según la columna que se este evaluando. En ella, el elemento '_' de la columna se sustituye por 'O' (nuestra hipotética jugada) y el elemento 'z' de encima por '_'. Siguiendo con el ejemplo anterior, para la columna 5 tendríamos:

	0	1	2	3	4	5	6	7	8
0	I	I	I	I	I	I	I	I	I
1	I	z	z	z	z	z	z	z	I
2	I	z	z	z	z	z	z	z	I
3	I	z	z	z	_	z	z	z	I
4	I	z	z	_	O		z	z	I
5	I	z	_	O	X	O	_	z	I
6	I	_	X	X	O	O	X	_	I
7	I	I	I	I	I	I	I	I	I

Ahora, del mismo modo que en el caso anterior, se busca una correspondencia del elemento '=' en la secuencia de búsqueda con el elemento '_' de la columna en la que se haya jugado. Esta correspondencia nos la indica la *posición de sincronización* de la secuencia.

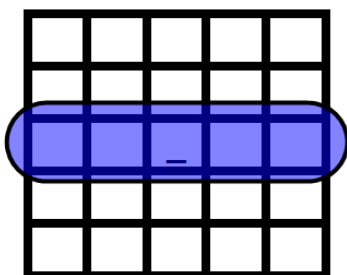
Ejemplos:

Secuencia de búsqueda: =zz =Xz _=_ E=E EOX= EEE= =?? =_E E_=O == =#
 Secuencia de prueba: _zz _Xz _ _z _OX_ zOX_ _z _XX OO_X

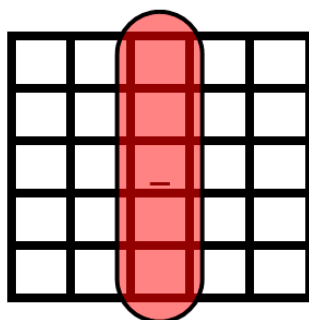
Dirección (de la secuencia):

Se puede especificar una o varias direcciones en las que se ha de buscar la secuencia, en la base de conocimiento aumentada (BCA). Estas pueden ser:

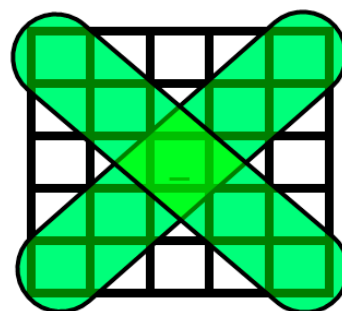
HORIZONTAL



VERTICAL



DIAGONAL



Los *especificadores de dirección* son los siguientes:

H : Secuencia horizontal.

V : Secuencia vertical.

D : Secuencia diagonal.

Se colocan después de la secuencia de búsqueda dejando, al menos, un espacio en blanco. Si hay más de un especificador se colocan juntos (el orden es indiferente).

Ejemplos:

Búsqueda en horizontal:

EO# H

Búsquedas para las diagonales y en vertical:

OOO# DV

Si no apareciese ninguno la búsqueda se realizaría en todas direcciones, es decir, sería lo mismo que colocar **HVD**.

Aunque las reglas permiten especificar la dirección, no se indica el sentido de la búsqueda ya que se realiza en ambos sentidos. Esto otorga "simetría" a las reglas, de manera que, a efectos prácticos, la secuencia **E#O** es igual que **O#E**. Es decir, la búsqueda horizontal se hace de izquierda a derecha y viceversa, la vertical de arriba a abajo y viceversa (aunque, en este caso, no tenga sentido), y en las dos diagonales de forma ascendente y descendente.

Valor (de una regla):

El *valor* de la regla es un número entero con o sin signo, que esta separado de la dirección (si la hay) por, al menos, un espacio en blanco.

El uso que se le de al *valor* de la regla depende de la estrategia de control. En algunos casos puede utilizarse para otorgar una prioridad a la regla, en otros simplemente se ignora. Nosotros lo usaremos para calcular como de buena (o mala) es una situación.

Ejemplos:

Situación ventajosa (valor alto):

#OO +300

Una mala elección (valor muy bajo):

|??#X H -500

Los espacios en diagonal nos interesan (valor positivo):

#E D 50

Límites (de los valores):

El valor máximo y mínimo que se le puede asignar a una regla viene determinado por la implementación. Para poder asignar a una regla el valor mas alto o el más bajo sin necesidad de saber los detalles de la programación se usan los límites:

MAX : Valor máximo (también **Max** y **max**).

MIN : Valor mínimo (también **Min** y **min**).

Se colocan en la regla en lugar que ocuparía el *valor*. "Max" tiene un valor superior a cualquier valor que se pueda representar numéricamente. Igual ocurre con "Min" pero al contrario.

Ejemplos:

La mejor situación: **#OOO MAX**

La peor elección: **=XXX Min**

La inclusión de los límites en las reglas puede ser de utilidad cuando el valor de las reglas se usa como prioridad, pero puede plantear problemas si el valor se usa de otra forma, como puede ser, calcular como de buena es una posición. En estos casos se suelen considerar el límite superior (MAX) como $+\infty$ y el límite inferior (MIN) como $-\infty$, con lo que las operaciones que se realicen con ellos no modifican su *valor*. Sin embargo, también han de considerarse situaciones en las que se cumplan al mismo tiempo condiciones valoradas con MAX y MIN, o que haya varias evaluadas como MAX.

BNF (Backus-Naur Form):

A continuación mostramos la sintaxis de una regla en BNF. Los símbolos terminales están resaltados en negrita mientras que los no terminales están colocados con comillas tipográficas (<>). Entre los no terminales no hay espacios en blanco, cuando ha sido necesario introducirlos se ha hecho con el no terminal <E> que indica uno o más espacios en blanco () así como tabuladores, pero no saltos de línea (↵).

```
+ , -
<REGLA> ::= <SECUENCIA> <E> <VALOR> | <SECUENCIA> <E> <DIRECCIÓN> <E> <VALOR>
<SECUENCIA> ::= <ELEMENTO> | <ELEMENTO> <SECUENCIA>
<ELEMENTO> ::= O | X | _ | z | I | E | # | = | ?
<VALOR> ::= <NUMERO> | <SIGNO> <NUMERO> | <LIMITE>
<NUMERO> ::= <DIGITO> | <DIGITO> <NUMERO>
<DIGITO> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<SIGNO> ::= + | -
<LIMITE> ::= MAX | Max | max | MIN | Min | min
<DIRECCION> ::= <ESPECIFICADOR> | <ESPECIFICADOR> <DIRECCION>
<ESPECIFICADOR> ::= H | V | D
<E> ::= | <E>
```

Comentarios:

Además de las reglas se añadirá la posibilidad de introducir comentarios de línea precedidos por doble barra:

// : Comentario de línea.

Podrán colocarse detrás de una regla, separados de ella por, al menos, un espacio en blanco y su ámbito durará hasta el siguiente salto de línea.

Ejemplo:

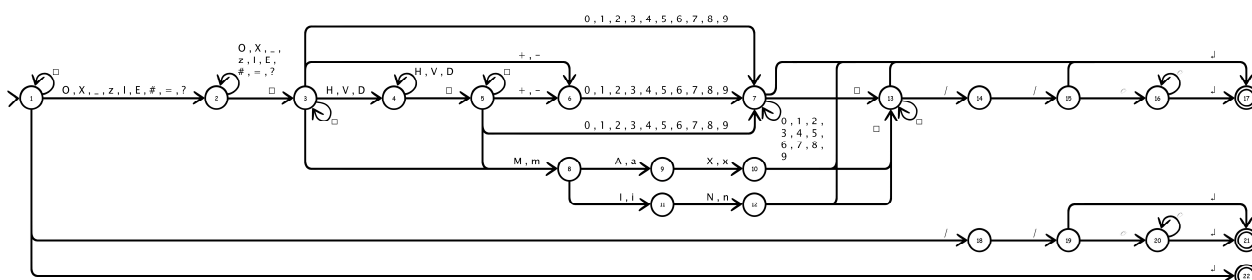
Regla con comentario: **#000 MAX // Regla principal**

ESTRATEGIA DE CONTROL

En este apartado trataremos aspectos fundamentales para el sistema de producción. Aunque ya hemos definido las reglas y tenemos una base de conocimiento, aún tenemos que decidir como se van a usar. Se tratarán cuestiones como: asegurarnos de que una regla este bien escrita, la manera en que se buscan las secuencias en la BCA, reducción del número comprobaciones, elección de la próxima jugada, etc.

AFD (Automata Finito Determinista):

Para verificar que una regla esté bien escrita usaremos un AFD ya que es fácil de crear y sencillo de implementar:



Nótese que el AFD también acepta los comentarios, y reconoce líneas en blanco así pues esta pensado para evaluar una línea completa del archivo de reglas.

Estructura de las reglas:

Aquí se describe la estructura de implementación en la que se guarda una regla.

REGLA

Línea	SECUENCIA	INVERSA	Direcciones[]	VALOR	Comentario	Estado	Error
-------	-----------	---------	----------------	-------	------------	--------	-------

Línea: Entero con el número de línea en el que aparecía la regla en el archivo de reglas.

SECUENCIA: Estructura con los datos de la secuencia de búsqueda.

INVERSA: Estructura (misma que SECUENCIA) con los datos de la secuencia de búsqueda invertida.

Direcciones[]: Array booleano de tamaño 3 que indica en que direcciones se aplica la regla (H, V, D).

VALOR: Estructura con el valor de la regla. Dispone de NaN, +INF y -INF.

Comentario: Cadena con el comentario (si lo había).

Estado: Entero que indica el estado de la regla. Estos pueden ser:

- Error: La regla tiene algún error que se indica en el campo "Error".
- Ninguno: La regla esta sin inicializar (este estado no permanece después de la generación).
- Blanco: Se ha leído una línea en blanco.
- Valida: Es una regla valida.

Error: Cadena con el mensaje de error (si ha habido alguno).

SECUENCIA

Secuencia	Posición
-----------	----------

Secuencia: Cadena de caracteres con la secuencia de elementos.

Posición: Entero con la posición de sincronización del elemento de casilla vacía jugable.

VALOR

Valor	+INF	-INF	NaN
-------	------	------	-----

Valor: Entero largo con el valor numérico.

+INF: Booleano que indica un valor de +infinito (en cuyo caso "valor" y "-INF" no se usan).
-INF: Booleano que indica un valor de -infinito (en cuyo caso "valor" y "+INF" no se usa).
NaN: Booleano que indica que el número no es válido (en cuyo caso no se usa ninguno de los anteriores).

Evaluación de las reglas:

Valga el siguiente ejemplo para ilustrar como se comprueba una regla.
 Supongamos que lo estamos haciendo para la columna 3:

#OE HVD 10

	0	1	2	3	4	5	6	7	8
0	I	I	I	I	I	I	I	I	I
1	I	z	z	z	z	z	z	z	I
2	I	z	z	z	z	z	z	z	I
3	I	z	z	z	_	z	z	z	I
4	I	z	_	_	O	_	z	z	I
5	I	z	O	O	X	X	_	z	I
6	I	_	X	X	O	O	X	_	I
7	I	I	I	I	I	I	I	I	I

Primero obtenemos todas las secuencias de prueba que pasan por el elemento casilla vacía jugable '_' de la columna 3:

#OE HVD 10

	0	1	2	3	4	5	6	7	8
0	I	I	I	I	I	I	I	I	I
1	I	z	z	z	z	z	z	z	I
2	I	z	z	z	z	z	z	z	I
3	I	z	z	z	z	z	z	z	I
4	I	z	_	_	O	_	z	z	I
5	I	z	O	O	X	X	_	z	I
6	I	_	X	X	O	O	X	_	I
7	I	I	I	I	I	I	I	I	I

H : Iz__O_zzI

V : Izzz_OXI

Da: I_O__zzI

Dd: Iz z_XOI

Para cada secuencia de prueba es importante guardar el índice del elemento '_' que estaba en la columna que estábamos evaluando. Esto es lo que hemos llamado la posición de sincronización:

H : Iz__O_zzI

012345678

V : Iz z_OXI

01234567

Da: **I_O__zzI**
 012**3**4567
 Dd: **Izz_XOI**
 012**3**456

Y para cada una de las secuencias de prueba hacemos la comparación con la secuencia de búsqueda de la regla, primero con la secuencia de búsqueda original y luego con la secuencia de búsqueda invertida. En ambos casos la posición de sincronización de la secuencia de búsqueda viene dada por el elemento '#'.
 #

<p>012 #OE H : Iz_O__zzI 012345678</p>	<p>012 EO# H : Iz_O__zzI 012345678</p>
<p>012 #OE V : Izzz_OXI 01234567</p>	<p>012 EO# V : Izzz_OXI 01234567</p>
<p>012 #OE Da: I_O__zzI 01234567</p>	<p>012 EO# Da: I_O__zzI 01234567</p>
<p>012 #OE Dd: Izz_XOI 0123456</p>	<p>012 EO# Dd: Izz_XOI 0123456</p>

Simplificando la parte significativa de la secuencia de prueba y haciendo la comparación obtenemos:

<p>#OE H : _O_</p>	<p>EO# H : _z_</p>
<p>#OE V : _OX</p>	<p>EO# V : zz_</p>
<p>#OE Da: __z</p>	<p>EO# Da: _O_</p>
<p>#OE Dd: __XO</p>	<p>EO# Dd: zz_</p>

Como la sentencia coincide en 2 ocasiones, sumamos 2 veces el valor de la regla (10x2=20) a la puntuación total que ya tenía esa columna.

Una vez evaluadas todas las reglas en todas las columnas (exceptuando las que estén llenas) se elige jugar en la que tenga mayor puntuación. En caso de empate se elige una al azar.

Archivo (de reglas):

Las reglas han de guardarse en un archivo de texto (por defecto "*rules.txt*") en el cual cada línea podrá ser:

- Una línea vacía.
- Un comentario de línea.
- Una regla (con o sin comentario de fin de línea).

Este es el contenido del archivo "*rules.txt*" que se usa por defecto:

```
[ 1] // IA - P1 - Archivo de reglas para el juego de las 4 en raya
[ 2] // =====
[ 3]
[ 4] // 1 casilla adyacente (Valoraciones bases)
[ 5] #   H   1
[ 6] #z HVD  2 // =   3
[ 7] #_   H   4 // =   5
[ 8] #_   D   9 // =  10
[ 9] #O HVD 50 // =  51
[10]
[11] // 2 casillas adyacentes
[12] #Oz HVD  10 // =  61
[13] O#z HVD   8 // =  61
[14] #O_ HVD  20 // =  71
[15] O#_ HVD  20 // =  71
[16] #OO HVD 100 // = 151
[17] O#O HVD 100 // = 151
[18]
[19] // 3 casillas adyacentes utiles
[20] =OOO   HD  -200
[21] #OzO   HD  240 // = 301
[22] O#zO   HD  240 // = 301
[23] #XX_   HVD  500
[24] X#X_   HVD  500
[25] #OO_   HVD  550 // = 701
[26] =XXX   HVD -1000
[27] X=XX   HVD -1000
[28] _O#O_   HVD 1500
[29] #XXX   HVD 2000
[30] X#XX   HVD 2000
[31] #OOO   HVD  MAX
[32] O#OO   HVD  MAX
[33]
[34] // Proximidad a la pared
[35] #I   HVD -10
[36] #EI  HVD -10
[37] #EEI HVD -10
```

Explicaciones:

- [5] La puntuación inicial de una columna es NaN (también es el valor devuelto si la columna está llena). Con esta regla le asignamos un valor de 1 a cualquier columna en la que se pueda colocar una ficha. Solo la evaluamos en una dirección para que el valor no sea 4 (H+V+Da+Dd). Si esta fuese la única regla del archivo, el comportamiento sería idéntico al de un jugador *Random* que elige su jugada al azar.
- [6] Cuando una regla tiene la secuencia de otra como prefijo, su valor se le suma. Esto es, a la regla [6] se le suma el valor de [5]. En realidad, el valor de la regla prefijo se suma a la columna, pero en la práctica el valor de la regla que la contiene se convierte en un incremento.
- [8] Al haber incluido en nuestras reglas la posibilidad de especificar la dirección, podemos darle mayor valor a las diagonales, como es el caso de esta regla respecto a [7]. Esto se traduce en que el programa prefiera colocar sus fichas encima de las

del oponente (ya que estas posiciones disponen de más diagonales). Aunque está estrategia pueda resultar extraña, a la larga resulta efectiva.

- [9] Las posiciones con fichas propias adyacentes deben tenerse en consideración y estar bien valoradas. Lo mismo hace la regla [16], entre otras.
- [13] Esta regla complementa a la anterior [12]. Es decir, si lo que realmente queremos valorar que la situación final sea OOz hemos de considerar las dos posibilidades.
- [14] Es muy similar a la regla [12] pero dándole mas valor si el espacio en blanco es una casilla jugable. También tiene su complementaria en [15].
- [20] Esta regla tiene en consideración lo que pasaría después de poner nuestra ficha. Si se cumple lo que se indica en la secuencia, tendríamos 3 fichas propias consecutivas y, en la misma columna, una casilla libre jugable. Podríamos pensar a priori que es una situación que nos acerca mucho a la victoria, pero, en realidad, lo más probable es que el oponente se de cuenta y, dado que el siguiente turno es suyo, nos quitaría la victoria moviendo en la misma columna que nosotros. Es una mejor estrategia esperar a que él se vea obligado a jugar primero en esa columna. En consecuencia le damos un valor negativo.
- [21] Expresa una situación en la que las fichas no están consecutivas pero que no deja de ser una posibilidad de victoria futura cuando se pongan mas fichas en la columna que las separa. [22] la complementa.
- [26] Victoria del oponente en la próxima jugada, si mueve en la columna que se esta evaluado. Una situación muy peligrosa que no debe pasar desapercibida. Afortunadamente nuestras reglas contemplan esa posibilidad, tal y como se muestra en esta con un valor negativo grande que hace que la columna quede casi descartada. Decimos "casi" porque puede darse la posibilidad de que nosotros tengamos antes una jugada ganadora en esa columna. Si se diese el caso una regla con valor MAX ([31] o [32]) haría posible jugar en esa columna. También se le podría haber dado a esta regla un valor de MIN, siempre y cuando el motor del juego de preferencia a MAX sobre MIN (el nuestro lo hace). [27] la complementa.
- [28] Una situación muy interesante, victoria en dos jugadas (si el oponente no gana antes). Es raro que se de, pero merece la pena tenerla presente.
- [29] ¡Cuidado! El oponente esta a punto de hacer 4 en raya, a no ser que tengamos una jugada ganadora, estamos obligados a mover ahí. [30] la complementa.
- [31] La victoria es nuestra, hemos encontrado una jugada ganadora. Hemos de darle la puntuación más alta: MAX. [32] la complementa.
- [35] Las jugadas cerca de la pared dan menos juego así que son de menor interés. Aunque haya espacios en blanco de por medio estos no suelen ser de utilidad. Es justo que les quitemos un poco de valor a estas posiciones. [36] y [37] son del mismo tipo.

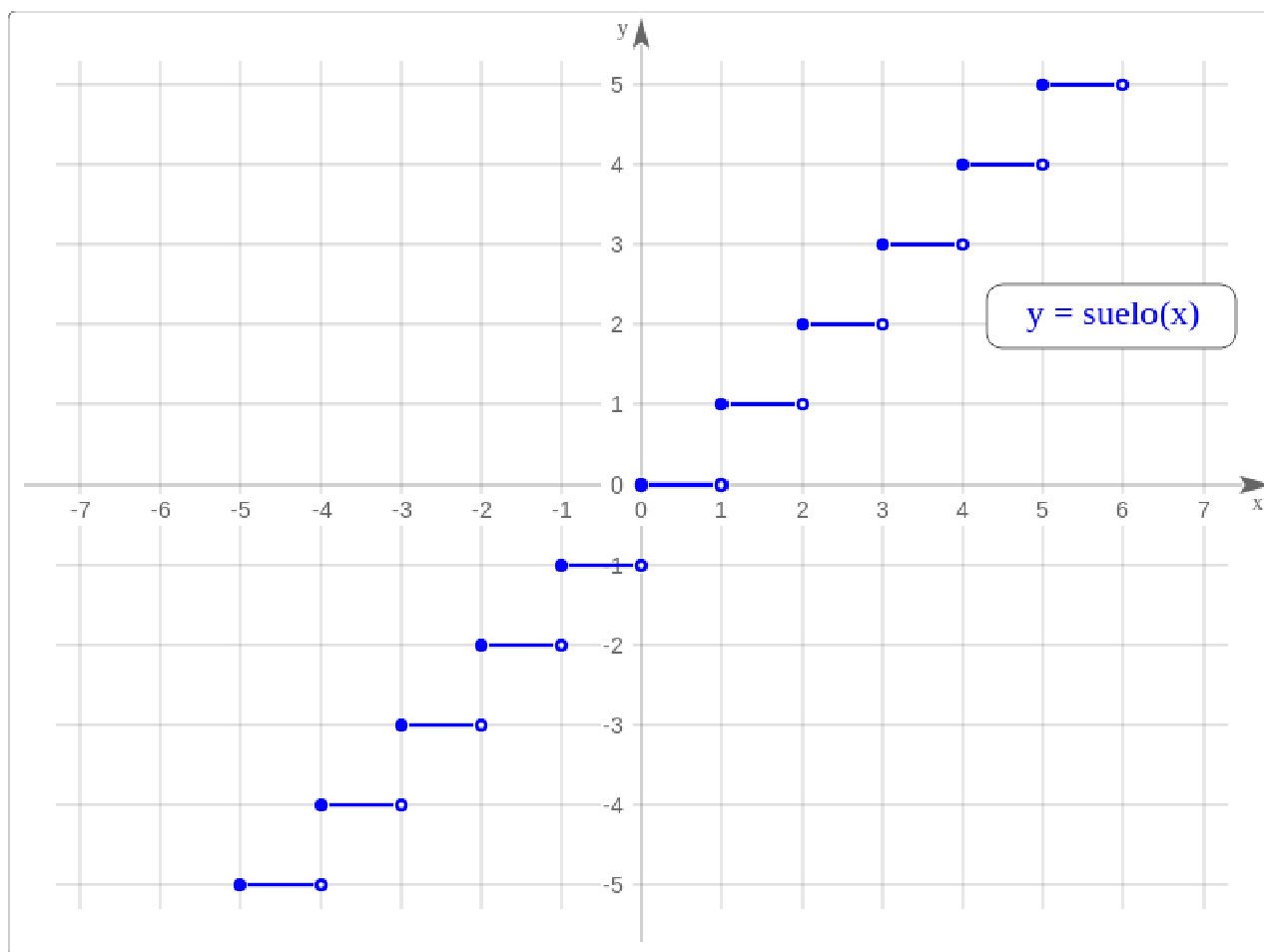
Nivel de las reglas:

A la hora de implementar niveles de juego para un jugador basado en reglas, se nos plantean 2 opciones. La primera es hacer varios archivos de reglas que hagan que el jugador juegue mejor o peor. La segunda consistiría en modificar ligeramente la toma de decisiones para que se de la posibilidad de elegir jugadas no tan buenas.

Nosotros hemos elegido implementar la segunda opción puesto que pensamos es la que más se ajusta a los requerimientos de la práctica. Además, por la naturaleza de nuestras reglas, modificar los valores de las reglas o las puntuaciones de las columnas es algo relativamente sencillo.

Como ya hemos dicho, si después de evaluar las columnas, hay varias que tienen la puntuación más alta, se elige una de ellas al azar. Conforme avanza el juego es más difícil encontrar columnas con igual puntuación, y mucho más si se trata de la más alta. La

idea consistirá en hacer que el selector considere como iguales puntuaciones parecidas. Vamos a transformar las puntuaciones en base a una función escalonada parecida a $E[x]$ (parte entera).



Si bien en nuestro caso el intervalo no puede ser la unidad, y tampoco tienen por que ser necesariamente intervalos iguales; de hecho, para que haya varios niveles es deseable que esto se pueda modificar. Téngase en cuenta que cuantos más intervalos haya y más pequeños sean, mejor jugará nuestro jugador de reglas; podrá discernir entre más valores y por tanto su comportamiento se aproximará más al original. Y viceversa, cuantos menos intervalos y de mayor tamaño peor jugará.

Teniendo esto en cuenta y después de haber observado cuales son los rangos de puntuaciones más comunes que suele generar nuestro archivo de reglas hemos escogido 2 secuencias de intervalos que se corresponderán con los niveles 1 y 2:

Nivel 1:

```
{-2000,-1500,-1000,-500,-300,-100,-50,-20,0,20,50,100,300,500,1000,1500,2000}
```

Nivel 2:

```
{-1000,-900,-800,-700,-600,-500,-400,-300,-200,-100,-90,-80,-70,-60,-50,-40,-30,-20,-10,0,10,20,30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900,1000}
```

Nivel 3:

Sin intervalos. Evaluación normal de las reglas.

PRACTICA 2

EL PROCEDIMIENTO MINIMAX

CONSIDERACIONES

- Se ha implementado el algoritmo Minimax en su forma simple y también con poda alfa-beta. Está última es la que se usa con el jugador minimax.
- La clase que implementa el procedimiento minimax tiene los siguientes métodos principales:

Minimax(): Crea el nodo raíz.
setDepth(): Establece la profundidad del árbol ($Niveles = Profundidad \times 2 + 1$).
setHeuristic(): Establece las reglas heurísticas (si ya se disponía de ellas).
loadHeuristic(): Carga el archivo con las reglas heurísticas.
setValueHeuristic(): Establece el valor de un nodo hoja usando una heurística.
expand(): Expande el árbol hasta la profundidad requerida.
alphaBeta(): Genera el árbol realizando la poda alfa-beta.
propagateValues(): Propaga los valores de los nodos hoja hasta el nodo raíz.
- La profundidad del árbol se puede modificar pero no se recomienda que sea mayor de 3 (7 niveles). Indica un turno de MAX y otro de MIN (además del nodo raíz).
- El nivel del jugador viene determinado por la profundidad del árbol minimax (1 a 3).
- En cada movimiento se vuelve a generar el árbol minimax, no se parte de una rama de árbol ya existente.
- Cuando el estado del tablero es simétrico, no se generan los nodos hijos para todas las columnas, se elige al azar entre las columnas 1 y 7, 2 y 6, 3 y 5.
- Cuando una columna está llena, no se genera el nodo hijo que se corresponde con una jugada en dicha columna.
- Cuando un nodo tiene un tablero lleno, no se evalúan sus hijos, esté en el nivel que esté.
- Cuando un nodo tiene un tablero con victoria de alguno de los jugadores, este no genera más nodos hijos, esté en el nivel que esté.
- A un nodo con un estado de victoria se le asigna directamente el valor +INF/-INF según se trate de un nodo MIN/MAX respectivamente. No se evalúa la heurística para asignarle un valor a estos nodos. Y tampoco genera descendientes, esté en el nivel que esté.
- El valor de un nodo es la suma de los valores que ha obtenido cada columna (en las que se puede mover) después de que se haya obtenido para cada una un valor al evaluarse en ellas las reglas heurísticas.

- Las reglas heurísticas se escriben igual que las descritas en la practica 1. La principal diferencia con las reglas del jugador de reglas y el jugador minimax, es que en este caso se evalúa la situación actual (no la próxima jugada), y en consecuencia no se elige el valor más alto, si no que toma la suma.
- El archivo de reglas heurísticas por defecto se llama "heuristic.txt" y tiene este aspecto:

```
[ 1] // IA - P2 - Archivo de heurística para el juego de las 4 en raya
[ 2] // =====
[ 3]
[ 4] #000 HVD 4
[ 5]
[ 6] #00E HVD 3
[ 7] #0EO HVD 3
[ 8] #E00 HVD 3
[ 9]
[10] #0EE HVD 2
[11] #EOE HVD 2
[12] #EEO HVD 2
[13]
[14] #EEE HVD 1
[15]
[16] #XXE HVD 2
[17] #XEX HVD 2
[18] #EXX HVD 2
[19]
[20] #XXX HVD 3
```