Sistemas Inteligentes

— oOo —

Desarrollo de un agente inteligente para el juego *Conecta-4*

Geneviève Cirera
genevieve.cirera@alumnos.upm.es

César Pérez
cperez@alumnos.upm.es

26 de enero de 2014

Índice general

1 Introducción

La meta de esta práctica es hacer un sistema inteligente con lógica borrosa y/o algoritmos evolutivos.
Para esto, eligimos hacer una inteligencia de un conecta-4. Es decir, construir un juego del conecta-4 donde uno de los dos jugadores sea virtual, que elija sólo que columna jugar.
Eligimos hacer el programa en java, porque es un lenguaje objeto muy conocido por toda la gente.

En este documento será desarrollado los datos de entrada elegidos y el proceso de elección de la columna a jugar con cada dato intermedio.
El jugador inteligente es el FuzzyPlayer.java que crea un Process.java para elegir una columna. Este proceso está detallado en las páginas siguientes.

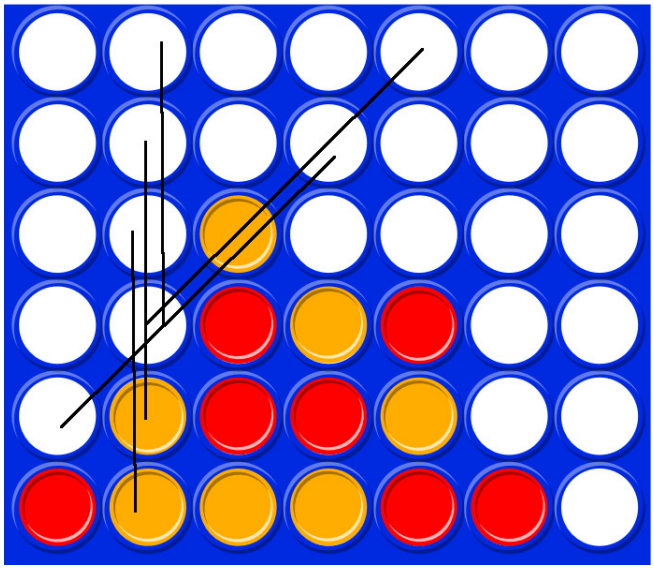
Cada clase es probada por JUnit tests en el repositorio "tests".

2 Los datos

Al principio, tenemos que coger los datos para poder analizarlos y devolver una solución al problema. Estos datos será la entrada de nuestro proceso para elegir la mejora columna.

2.1. Entrada 1

Eligimos tomar el número de posibilidades que si el jugador pone una ficha en la columna n, tendrá m posibilidades de hacer una línea de 4 fichas.
Por ejemplo :



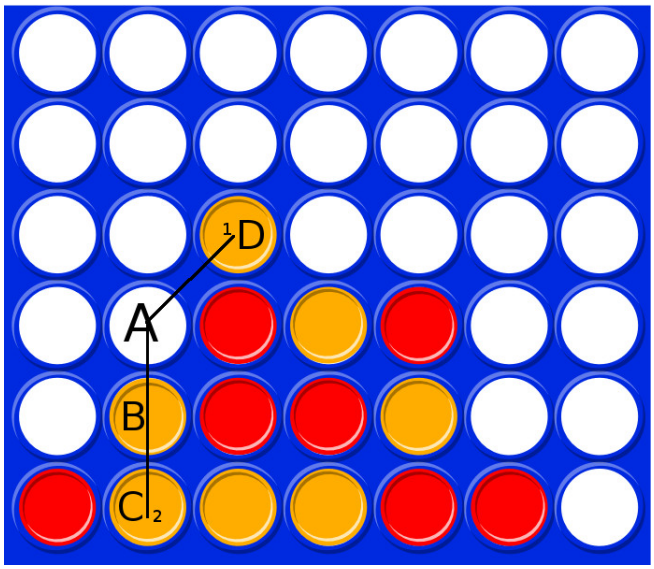
Aquí, si el jugador amarillo pone una ficha en la segunda columna, tendrá 3 (en vertical)

más 2 en diagonal derecha, es decir 5 oportunidades de hacer una línea (las oportunidades son subrayadas).
Lo mismo con el jugador rojo, si pone una ficha en la sexta columna, tendrá 2 (en vertical) más 2 en diagonal izquierda, es decir 4 oportunidades.
Estas oportunidades son contadas por el método "countLine"de FuzzyPlayer.java.

2.2. Entrada 2

2.2.1. Coger los datos

Eligimos también coger la suma de las fichas al lado de la ficha que quisieramos poner, mezclado al número de directiones por donde se van.
Por ejemplo :



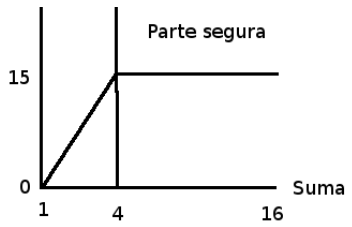
Tenemos 4 fichas, la A, la B, la C y la D. Así, si contamos la suma será la A, la B y la C : 3 más la A y la D : 2, es decir una suma de 5 para 2 direcciones.
Las direcciones son contadas por el método "countDirectionz la suma por "sum"de FuzzyPlayer.java.
Esta segunda entrada es una mezcla entre la suma y el número de direcciones calculado en el método "addWeight"de la clase FuzzyPlayer.java.

2.2.2. Mezclar suma y dirección

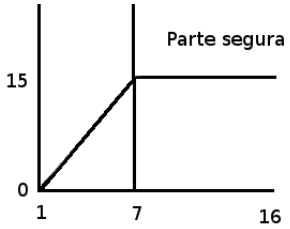
Sabemos que según el número de direcciones y la suma, la columna es más o menos buena.
En total, hay 4 direcciones posibles (vertical, horizontal y las dos diagonales), y la suma puede ir de 6 en horizontal máximo, más 4 en vertical, más 10 para cada diagonal. Es decir 34, pero este caso es casí imposible porque el jugador haría todas las líneas más grandes del tablero. Pues, vamos a tomar un número máximo de 16 que corresponde en la oportunidad de hacer 4 líneas de 4 o una línea de 6 con 2 líneas de 4 y una línea de 2 añadiendo una ficha.
Ya es mucho, si occure una suma más alta, se pondrá un peso muy alto para mostrar lo significativo.

El método "addWeight", mezcla la suma y el número de direcciónes. Pues para cada número de direcciónes de 1 hasta 4, una funccion devuelve un número entre 0 y 15 mientras que las oportunidades (entrada 1) puede devolver un número entre 0 y 13. Esto para añadir un peso a la entrada 2 respecto a la entrada 1. En efecto, más vale una suma de 4 y una oportunidad de 1 a una suma de 1 y una oportunidad de 4.
Así creamos 4 funcciones que llamaremos funcción "transform":

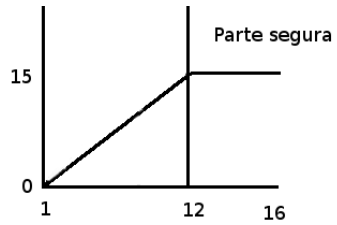
- Para 1 dirección, si la suma es de 4 o más, es seguro que el jugador ganará, así devuelve el máximo : 15. Más la suma es baja, menos es bueno y devuelve un valor más bajo.



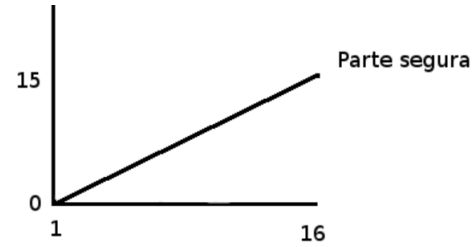
- Para 2 direcciones, lo mismo para esta funcción excepto que lo seguro es con 7 en suma (4 + 3 en suma). Se puede seguro con un 6 también si está repartido en 4 + 2 pues se devolverá un poco menos.



- Para 3 direcciones, lo mismo pero seguro con un 12 y a lo mejor con un 8 y más.



- Para 4 direcciones, lo mismos empezando a estar seguro con un 10 y estando seguro con el 16.



- Si no hay ninguna direcciónes, se pondrá el valor 0.

3 El processo de elección

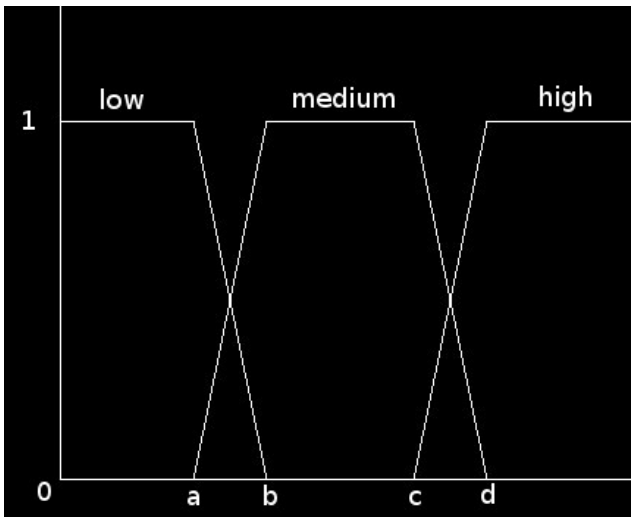
El processo se define en varias partes : Fuzzificación, Inferencia (que incluye las reglas y la agregación) y Defuzzificación.
Tenemos también el symbol table donde se almacena todos los datos.
En los párafos siguientes, vamos a desarrollar cada una de estas partes.
El proceso se realiza gracias a la clase Proceso.java que crea todos los objectos correspondiendo a cada parte del proceso, almacena los datos en el SymbolTable.java y puede devolver el valor numérico que queremos al final gracias a la funcción "chooseColumn".

Como lo vemos en este squema, el difusor corresponde a la fuzzificación, el mecanismo de inferencia que carga las reglas, las aplica y efectua la agregación entre todos los resultados de las reglas y el desdifusor que corresponde a la defuzzificación.
Los datos de entrada es el número de oportunidades (entrada 1) y la mezcla de la suma y las direcciones (entrada 2) vistos en el apartado anterior para cada columna es decir 7 veces.
La salida es el número de la columna elegida.
Cada vez que sale un número de columna, se limpian todas las listas del proceso para estar listo para acoger nuevos datos.

4 El proceso de fuzzificación

4.1. Los funcciones

Hemos elegido de hacer 3 funcciones para tener más detalles y que el sistema sea más preciso. Pues, hemos creado una clase "Fuzzification"para este proceso.
Esta clase define las 3 funciones low, medium e high tomando en parámetro un valor y devolviendo el valor correspondiente por la función.
El método "fuzzification"tomando las dos entradas correspondientes al número de oportunidades y suma/direcciones de fichas devuelve una tabla de 2 líneas y 3 columnas correspondientes a los resultados de las 2 entradas por cada una de las 3 funcciones.
Se puede ver aquí a que parecen las funcciones low, medium e high.



En el aje horizontal los valores van de 0 hasta 15, como lo vimos en la parte "Datos". Se queda definir los cuatro valores a, b, c y d que definen el tamaño de cada función de fuzzificación.

4.2. Los valores a, b, c y d

Como lo vimos anteriormente, la entrada 2 da un valor entre 0 y 15, pero a partir de un certo rango, se puede que sea muy bien o no según la repartición de las fichas. Después de este rango, es seguro que el jugador ganará.

Para 2 direcciones, si la suma es 6 se puede que da la línea de 4 o no.
Para 3 direcciones, si la suma es entre 8 y 12 se puede que da la línea de 4 o no.
Para 4 direcciones, si la suma es entre 10 y 16 se puede que da la línea de 4 o no.
Por la funcción transform (vista previamente) tenemos :
transform(6) = 12,5
transform(8) = 9,55
transform(10) = 9
Pero estos valores son consideradas como buenas (medium) o muy buenas (high), pues tienen que estar donde los dos funcciones medium y high se cruzan, es decir entre c y d. Por eso, pondremos c = 8 y d = 13.
Ademaás el número de oportunidades no puede superar 13, pero no tiene tanto peso que la entrada 2, pues es normal que casí siempre tendrá valores de low y medium.

Para los valores de a y b, podemos decir que sólo tener una oportunidad de hacer una línea de 4 es muy poco, tener 2 también porque el adversario puede quitar las posibilidades muy rápidamente. Pues, tomaremos como valores a = 2 y b = 5.

5 Las reglas

5.1. Explicaciones

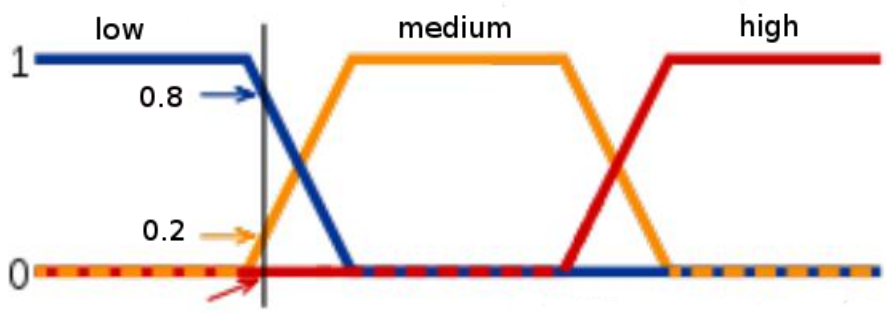
Las reglas son definidas en un fichero ressources/Rules.txt de la manera siguiente.
HIGH MEDIUM:MEDIUM
Esto para decir que si la primera variable (entrada 1) es HIGH y la segunda (entrada 2) MEDIUM entonces la columna de donde se trata es buena de manera MEDIUM.

El método "applyRules", aplica las reglas para un conjunto de datos de una columna, es decir para cada table de 2 líneas y 3 columnas generadas por la fuzzificación y devuelve un HashMap<String, Float>donde se definirá la altura deducida para cada conjunto.

Para encontrar estos datos, la clase Rules.java analiza los datos de la salida de la fuzzificación, es decir, hace un producto cartesiano entre la primera variable y la segunda para todos los datos diferentes de 0, sabiendo que la primera columna corresponde al resultado de la funcción low de fuzzificación y la última a la funcción high, sale un HashMap de tipo <"LOW MEDIUM", 0.2>. El 0.2 correspondiendo al mínimo entre los dos datos de la funcción low para la primera variable y medium para la segunda porque es un "et" pues tomamos el mínimo.

Después con este HashMap se puede aplicar las reglas con analyseRules. Este método busca en el HashMap donde se almacenaron todas las reglas del fichero y devuelve en otro HashMap<String, Float>el resultado. Este último HashMap define la altura deducida para cada funcción.

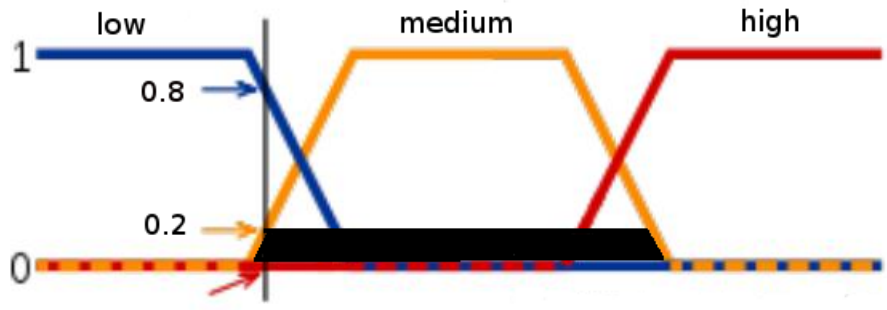
5.2. Ejemplo



En este grafo, por el valor de entrada 1 dado, tendremos un resultado de tipo `[0.8, 0.2, 0]` a la fuzzificación de este valor. Imaginamos que tengamos otro valor (entrada 2) que da `[1, 0, 0]`, pues tendremos :
`[0.8, 0.2, 0]`
`[1, 0, 0]`
como salida de fuzzificación. Lo que da un elemento de la primera HashMap de tipo <"LOW LOW", 0.8>. "LOW LOW" correspondiendo a la primera columna y 0.8 al mínimo entre 1 y 0.8.
Tendremos también un segundo elemento <"MEDIUM LOW", 0.2>.

Se buscará después si existe alguna regla correspondiente a "LOW LOW" y a "MEDIUM LOW" y devolverá el resultado de la regla en la segunda HashMap.

Si el resultado de la regla "MEDIUM LOW" es "MEDIUM", tendremos un elemento de la segunda HashMap : <"MEDIUM", 0.2>que quiere decir una altura de 0.2 sobre la funcción medium :



El polígono en negro de altura 0.2 y delimitado por la funcción medium.

Si por las reglas, se devuelve varias veces el mismo conjundo (LOW, MEDIUM o HIGH) se toma el valor máximo entre ellos.

Si hay varios elementos de la primeras HashMap que tiene un resultado por una regla, habrá varios elementos en la segunda HashMap. Después se habrá hacer el proceso de agregación en el proceso de inferencia para juntar todas las superficies.

6 El proceso de inferencia

Representado por una clase, el método tiene una lista de tablas, el conjunto total de las entradas ya pasadas en el proceso de fuzzyficación en parámetro. Así, cada elemento de la lista representa los datos para una columna, es decir, cada elemento es una tabla de 2 líneas y 3 columnas que representan los 2 vectores resultados de la fuzzificación de las 2 entradas para una columna.

El motor de inferencia carga las reglas (crea un objeto Rule) y aplica las reglas para cada conjunto fuzzificado gracias al método "applyRules"de la clase Rules"por cada elemento de la lista y almacena el resultado para cada columna en la lista resultRules". Cada elemento de esta lista es un HashMap<String, Float>, explicaciones en la sección "Las reglas".

Después, el motor de inferencia puede aplicar el proceso de agregación llamando el método "agregation"de su clase y almacena el resultado en la lista "resultAgregation". La meta de la agregación es hacer un conjunto global del resultado de todas las reglas para cada columna. Es decir por ejemplo de los dos elementos HashMap resultados de las reglas para una columna vamos a hacer un polígono definido por el conjunto de todas la area del HashMap.
Este método encuentra los puntos de los polígonos creados con las reglas y crea un objeto Polygon asociado.
Así tenemos una lista de polígonos definidos gracias a los puntos deducidos de la HashMap precedente para cada columna.

7 El proceso de defuzzificación

Este proceso es sencillo, se trata de convertir los datos almacenados en los polígonos en un número. Este proceso occure después del proceso de inferencia.
La clase Defuzzyfication permite encuentrar el centroide de cada polígono definidos anteriormente.
Así sólo queda elegir la columna, será el que tiene el x del centroide lo más elevado porque más cerca del high.

8 Conclusión

Con esta práctica hemos aprendido a aplicar la lógica borrosa a algo concreto y ver los resultados.
Este conecta-4 reflecciona efectivamente a la manera de la lógica borrosa, podemos darse cuenta de que el FuzzyPlayer intenta elegir la columna que está lo más cerca de sus otras fichas y en segundo lugar donde hay lo más oportunidades de hacer una línea.

Sin embargo, el juego no toma en cuenta las fichas del adversario, es decir que para una versión futura, en lugar de tener 2*número de columna de entradas, deberá tener 2*2*número de columna de entradas, teniendo en cuenta que lo malo para el adversario, es lo bien para el otro jugador e inversamente.