¿QUIÉN ES QUIÉN?

PL1 – SWI Prolog – 9 de marzo de 2017

Conocimiento Y Razonamiento Automatizado

Miguel Romeral

Contenido

Objetivo

El problema que se implementa en esta práctica es el conocido juego ¿Quién es quién? en el lenguaje de alto nivel Prolog. El juego consiste en adivinar de una lista de personajes el elegido por parte del rival. Para ello haremos preguntas acerca de sus rasgos físicos para ir descartando diferentes personajes y quedarnos únicamente con la solución. En la implementación, se simulará el comportamiento del juego a través del ordenador, esto es, el ordenador conocerá las características físicas de cada jugador y obtendrá un personaje solución de entre todos los personajes e irá haciendo preguntas hasta obtener dicha solución.

Detalles de implementación

Hechos

Al tratarse de un programa diseñado en Prolog, se deberá tener en cuenta los diferentes hechos que componen la base de conocimiento en el momento inicial del juego. Éstos consisten en una lista de los diferentes rasgos que poseen cada uno de los personajes. La lista completa en el código se encuentra en el fichero *hechos.pl*.

La primera distinción que hacemos es la de género, separando a los hombres y a las mujeres mediante la siguiente proposición:

```
genero(hombre, albert).
genero(mujer, tiffany).
```

Existen un total de 21 personajes, de los cuales 14 son hombres y 7 son mujeres. La siguiente característica se refiere al color del pelo, en la que existen los personajes con el pelo negro y los del pelo rubio.

```
% color_de_pelo(X,Y): Y tiene el color de pelo X.
```

La siguiente concierne al color de la ropa. En el juego únicamente tendremos personajes con ropa roja o ropa verde. Esta característica es en la que más cerca están la cantidad de personajes con ropa roja y ropa verde, con 11 y 10 personajes respectivamente. Este dato es importante en el modo inteligente, el cual pasaremos a detallar más adelante.

```
% color_de_ropa(X,Y): Y lleva ropa de color X.
```

El estado de ánimo varía entre feliz o triste y se define de la siguiente manera:

```
% estado_de_animo(X,Y): Y tiene de estado de ánimo X.
```

Algunos personajes llevan gafas y otros no, sin embargo, la siguiente proposición únicamente indica a los personajes que sí la llevan, mientras que los que no llevan gafas no tienen ningún hecho que indique que no las llevan.

```
% lleva_gafas(X): X lleva gafas.
```

Podría haberse implementado la regla de tal manera que indicara explícitamente si lleva gafas o no, sin embargo, no se ha considerado necesario durante el diseño del programa.

El color de los ojos, la cual es la última característica incluida, vuelve a tener dos estados, ojos azules o marrones.

```
% color_de_ojos(X,Y): Y tiene color de ojos X.
```

Una vez vistos todos los hechos, se pasan a definir las preguntas que seleccionarán las diferentes reglas de los dos modos de juego:

```
pregunta(X,esChico):-genero(hombre,X).
pregunta(X,esChica):-genero(mujer,X).
pregunta(X,llevaGafas):-lleva_gafas(X).
pregunta(X,esRubio):-color_de_pelo(rubio,X).
pregunta(X,esMoreno):-color_de_pelo(negro,X).
pregunta(X,estaFeliz):-estado_de_animo(feliz,X).
pregunta(X,estaTriste):-estado_de_animo(triste,X).
pregunta(X,ropaRoja):-color_de_ropa(roja,X).
pregunta(X,ropaVerde):-color_de_ropa(verde,X).
pregunta(X,ojosAzules):-color_de_ojos(azules,X).
pregunta(X,ojosMarrones):-color_de_ojos(marrones,X).
```

Definimos las preguntas como `esChico`, `esChica`, `llevaGafas`, `esRubio`, `esMoreno`, `estaFeliz`, `estaTriste`, `ropaRoja`, `ropaVerde`, `ojosAzules` y `ojosMarrones` de tal manera que el resto del código hará mención a las preguntas de esa manera.

Por último, se adjuntan las preguntas que se realizarán como hechos, y la lista de personajes que se pueden seleccionar.

Reglas

En el fichero *reglas.pl* se incluyen diferentes reglas que son comunes en los dos modos de juego. Es importante explicar cómo será capaz el ordenador de llevar la lista de personajes que pueden ser solución o las preguntas que ha hecho o hará. Al inicio del juego, todos los personajes son añadidos a la memoria dinámica mediante el comando `assertz(seleccionable(X)).` donde X es el personaje que se añade. Así, la lista de hechos de tipo `seleccionable(X)` indica los personajes que pueden ser solución y aún no han sido descartados. Cuando el ordenador realiza una pregunta, en función de la respuesta, (detallaremos más adelante cómo lo hace), elimina mediante `retract(seleccionable(X))`. los diferentes personajes que no cumplen con las características del personaje solución en función de esa pregunta. De igual manera sucede con las preguntas a realizarse y las que ya han sido realizadas, con `assert` y `retract`. Las preguntas que ya se han hecho, se almacenan en la memoria dinámica mediante el hecho `pregunta_hecha(X,Y).` donde X es la pregunta e Y la respuesta en función del personaje solución. Las preguntas que aún no ha realizado se guardan con `pregunta_sin_hacer(X).` Al final, pueden o no quedar preguntas por hacer, pero únicamente debe haber un personaje como seleccionable, que será la solución, momento en el que podremos comprobar si se ha alcanzado el final del juego.

Las reglas `insertarSeleccionables` e `insertarSeleccionablesAux` tienen como fin añadir a la memoria dinámica la lista de los personajes que se pueden seleccionar. Estas reglas se ejecutan al inicio del juego. `insertarSeleccionables` obtiene una lista de los personajes que existen mediante `findall(X,personaje(X),L).` en la variable L. Esa variable es pasada a `insertarSeleccionablesAux` que añade a la memoria dinámica uno a uno de la lista hasta que no queda ninguno más.

De igual manera ocurre con `insertarPreguntas` e `insertarPreguntasAux`, sin embargo, las preguntas que aún no han sido realizadas estarán en `pregunta_sin_hacer(X).`

Hemos obtenido diversas reglas con el fin de otorgar funcionalidad al juego. Alguna de ellas es `borrarDeLista(Y, [Y|Xs], Zs),` que obtiene se encarga de borrar un elemento en concreto de una lista. `buscar([Y|Xs], N, X)` encuentra el índice de un elemento en una lista.

`seleccionado(Y)` obtiene un personaje al azar de entre todos los posibles con el fin de encontrar una solución para el juego. Esta regla se ejecuta también al comenzar dicho juego.

`buscarAzar(X, L)` se encarga de, mediante la regla `random(A, B, X),` un índice aleatorio para ser buscado en una lista, la cual será de personajes para encontrar una solución o encontrar preguntas al azar (aunque en el caso del modo inteligente, encontrará preguntas de manera más sofisticada).

La regla juego, genera todas las preguntas que se pueden realizar y los personajes que pueden ser solución. La manera en la que se inicia el juego depende del modo de juego, por lo que su implementación no está incluida en el fichero *reglas.pl*.

La regla preguntar será explicado con más detalle en cada modo de juego, sin embargo, es importante mencionar que esa regla, tras haber seleccionado una pregunta, realiza una acción en función de la solución, de modo que existen dos reglas definidas para la acción a llevarse a cabo. `accion(S,P)` recibe la solución y, si la respuesta es falsa (a si la solución lleva ropa de color roja, por ejemplo), eliminará de todos los seleccionables los personajes que estén en la memoria dinámica los que sí respondan afirmativamente a la pregunta (en el ejemplo, se eliminarán los personajes con ropa roja). Si por el contrario la respuesta es afirmativa, serán eliminados los que no cumplan con dicha pregunta (si se pregunta si la solución es chico, entonces se recogerán todos los personajes, se les quitará todos los que sean chicos, y la lista que queda, todos los personajes que son chicas, serán eliminados de entre los seleccionables). En esta regla, además de `findall`, nos apoyamos en `subtract`, que se encarga de restar dos listas.

`eliminarDeLista([X|L])` elimina de entre los seleccionables el elemento X si se encuentra entre los seleccionables.

La regla `resuelto`, compara si en la lista de seleccionables coincide con la solución, devuelve `true`, momento en el que el juego ha acabado satisfactoriamente.

La regla `iguales(L,M)` compara si dos listas son iguales. Es utilizada por `resuelto`, ya que la solución es almacenada en una lista para compararse con la lista de personajes seleccionables.

Se puede reiniciar el juego mediante la regla `reiniciar` y así crear nuevamente la lista de personajes seleccionables, se eliminan todas las preguntas hechas, se ponen como preguntas sin hacer y se obtiene una solución nueva aleatoriamente.

`mostrarPreguntasHechas` tiene como fin listar todas las preguntas que se han hecho durante el juego. El motivo por el que se incluyó esta regla, en lugar de utilizar `listing(pregunta_hecha(X))` es que en Prolog, si no se ha realizado ninguna pregunta y se intentan listar se obtiene un mensaje de error de Prolog, indicando que no existen hechos (ni estáticos ni dinámicos) de `pregunta_hecha(X)`, por lo que se inserta una pregunta auxiliar y se elimina, de modo que el usuario no obtiene nunca el error si lista las preguntas mediante esta regla. Es más ineficiente que `listing` pero se evita que el error aparezca al usuario.

Por último, se consideró oportuno añadir una regla de ayuda al usuario en que únicamente se muestran las diferentes reglas que el usuario puede realizar en la consola de Prolog.

Modo aleatorio

El modo aleatorio tiene como fin obtener preguntas al azar, sin importar cuántos descartes realizará. El mayor problema que existe, además de realizar más preguntas que por otra manera, es que el ordenador de esta forma puede hacer una pregunta cuya solución previamente ya conoce de manera implícita, como puede ocurrir si pregunta si la solución es un chico, de tal forma que ya conoceríamos el género de la solución, el ordenador podría preguntar después si es chica, lo cual no nos proporcionaría información de cara a la resolución del juego.

Las reglas del fichero *aleatorio.pl* incluyen la regla juego en la que únicamente añade los personajes a ser seleccionados y las preguntas a ser realizadas. Sin embargo, lo que distingue a este fichero del fichero del modo inteligente es su definición de la regla preguntar. En esta definición, se obtiene una pregunta de manera aleatoria, por lo que pueden darse los problemas que se han mencionado.

Modo inteligente

Este modo tiene en cuenta la cantidad de personajes que proporciona una pregunta y obtiene, cuando se quiere preguntar, la pregunta que mayor probabilidad de descarte tenga. La probabilidad de descarte está definida como:

$$\frac{\text{RT}}{\text{Total}} \times \frac{\text{Total} - \text{RT}}{\text{Total}} = \text{Probabilidad Descarte}$$

donde RT es la cantidad de personajes que cumplen con la pregunta (albert cumple con la pregunta esChico). De esta manera, la probabilidad de descarte es la probabilidad de que la solución esté entre las respuestas afirmativas a una pregunta determinada multiplicado por la cantidad de personajes que no cumplen con la pregunta. Así, las preguntas que obtienen mayor probabilidad de descarte son las preguntas cuyo número de respuestas correctas se aproxima al de respuestas erróneas. El mayor índice estaría dado por 0.25 (La mitad cumple y la otra mitad no, 0.5 * 0.5 = 0.25, por lo que ningún índice será mayor de 0.25). El número total está determinado por la cantidad actual de personajes que pueden ser solución para evitar que los índices de probabilidad no se actualicen.

La pregunta que se escoge cuando se desea preguntar es la que mayor probabilidad de descarte tenga. Si varias preguntas tienen el mismo índice, se obtiene de manera aleatoria una de ellas para que el juego no sea idéntico en todas las instancias. Cuando se realiza una pregunta, se vuelven a calcular todos los índices de probabilidad de cada pregunta que aún queda por hacer.

Interacción con el usuario

Para cargar el juego se debe estar en la carpeta donde reside el código fuente e introducir:

```
-? [hechos], [reglas], [aleatorio].
```

si se desea iniciar un juego en modo aleatorio, o

```
-? [hechos], [reglas], [inteligente].
```

si se desea iniciar un juego en modo inteligente.

Por último, se indica la lista de diferentes reglas que el usuario puede introducir durante el transcurso del juego:

`-? ayuda.`	Muestra las diferentes reglas que se pueden aplicar por pantalla.

`-? juego.`	Inicia el juego. No se puede iniciar el juego si ya está otro en curso, de lo contrario, no se obtendrían los resultados esperados.
`-? preguntar.`	Realiza una pregunta y descarta personajes en función de la respuesta y la solución.
`-? resuelto.`	Determina si el juego ya está resuelto.
`-? reiniciar.`	Reiniciar el juego con una nueva solución.
`-? listing(seleccionable).`	Muestra en una lista los personajes que pueden ser solución.
`-? listing(pregunta_sin_hacer).`	Muestra la lista de preguntas que aún no se han realizado.
`-? solucion(X).`	Indica en X cuál es la solución escogida por el ordenador.
`-? listing(probabilidad).`	Muestra la probabilidad de descarte de cada pregunta que aún no se ha realizado. Solo puede ser usada en el modo inteligente.
`-? mostrarPreguntasHechas.`	Muestra una lista de las preguntas que el ordenador ya ha realizado.