PARADIGMAS Y LENGUAJES DE PROGRAMACIÓN

LABOTORIO I - PROLOG - BLACKJACK

Cátedra:
Prof: Lic. Stickar, Romina
JTP: Lic. Pecile, Lautaro
Integrantes:
Toledo Margalef, Pablo Adrían
Universidad Nacional de la Patagonia San Juan Bosco

May 3, 2018

ENUNCIADO

Objetivos Preliminares

Dada la base de datos cards.pl que contiene la representación de las cartas, y sin alterar la misma, se deben implementar el siguiente conjunto de reglas para asistir al razonamiento del juego:

```
value(card(Number, Suit), Value)
```

Dada una carta, indica el valor de la misma. Para las númericas su mismo valor, para la figuras (J, Q, K) siempre 10 y para el As 11 o 1.

Una mano es representada como una lista de cartas. A partir de aquí, definimos Hand para hacer referencia a una mano de la forma [card(Number,Suit)], donde Number indica el número o figura de la carta y Suit el palo.

```
hand(Hand, Value)
```

Dada una mano, indica el valor o valores de la mano.

```
twentyone(Hand)
```

Indica si la mano suma exactamente 21.

```
over(Hand)
```

Indica si la mano se pasó de los 21.

```
blackjack(Hand)
```

Indica si la mano es un blackjack. O sea, suma 21 con sólo dos cartas.

Objetivos intermedios

En función de las reglas básicas del juego, es necesario darle entidad al crupier. El mismo tiene una política muy estricta de juego que respeta las siguientes reglas:

- Con una mano de 16 o menos, juega.
- Con una mano de 17 o más, se planta.

Existen dos posibles variantes a estas reglas cuando el crupier obtiene un 17 soft, esto es, su mano suma 17 con un As otorgándole el valor de 11.

Se deben implementar entonces dos reglas:

```
soft_dealer(Hand)
```

Se planta con una mano de 17 suave. Esto es, el As valiendo 11.

```
hard_dealer(Hand)
```

Sigue jugando con una mano de 17 suave. Donde Hand es la mano del crupier.

Objetivo principal

Implementar una regla:

```
play(Hand, Crupier, Cards)
```

que reciba tres listas de cartas. Hand y Crupier representan las manos del jugador y crupier respectivamente, Cards representa la lista de cartas que ya fueron puestas en lo que va de la ronda, tanto para todos los jugadores como para el crupier. Por simplicidad, se establece de base que el juego se realizará con un único mazo de cartas a diferencia de lo que se acostumbra en los casinos al utilizar múltiples barajas.

Esta regla debe realizar una deducción lógica indicando verdadero en el caso de seguir jugando y pedir más cartas o falso si decide plantarse. Es requisito primordial que el razonamiento se realice utilizando el total de parámetros indicados (Mano del jugador, mano del crupier y cartas que salieron en todo el juego) y cada uno de ellos debe hacer un aporte significativo a la lógica de juego. Este razonamiento además puede considerar la lógica del crupier, pero no es necesario que se haga una diferencia entre una política soft o hard.

DESARROLLO

Elementos Fundamentales de Juego

En primera medida se implementaron las reglas pedidas en los **Objetivos Preliminares** (value, hand, twentyone, over, blackjack), como puede observarse en los fuentes adjuntos a este informe.

Luego se desarrollaron las políticas para ambos croupiers de la siguiente manera:

- Se dictó la lógica del croupier que dice si pide otra carta o no (menor o igual a 16 pide otra carta, si no, se planta).

```
dealer(Value) :- Value =< 16.
```

- Se implementó la lógica que dictaba el valor del as para cada tipo de croupier. Es decir, para el softa el as vale 11 y para el hard vale 1.

```
% Para el soft dealer
soft_value(card(a,_),11):-!.
soft_value(Card,Value):- value(Card, Value).

% Para el hard dealer
hard_value(card(a,_),1):-!.
hard_value(Card,Value):- value(Card, Value).
```

- Luego se reimplementó el cálculo del valor de la mano para el croupier pero utilizando la nueva distinción en el valor del as.

```
% Hard dealer
% NOTA: El del Soft dealer es identico pero utilizando la regla soft_value en
    lugar de hard_value
hard_hand([Card|Resto],Value) :-
    hard_hand(Resto, Aux),
    hard_value(Card, CardValue),
    Value is CardValue + Aux.
```

La regla *Play*

Para poder discernir si uno debe (o no) pedir otra carta se puede tomar el siguiente enfoque.

Sabiendo qué cartas salieron hasta el momento, puedo calcular la probabilidad de que la siguiente carta que me den me haga pasarme de 21 y perder la mano

Por lo tanto debemos tener una forma de tomar todas las cartas que se jugaron hasta el momento (incluídas las del croupier) y sopesarlas de alguna forma para decidir. Ese "pesado" de las cartas ya jugadas es perfectamente posible de hacer y existen diversas maneras de hacerlo. En este caso utilizaremos una forma llamada Card Counting, en este caso utilizaremos el Conteo de Wong por mitades.

Counting Systems

En pocas palabras, un Sistema de Conteo le asigna a cada carta un valor y a medida que las cartas se van jugando se va manteniendo la suma de todos los valores de dichas cartas. Una convención es que las figuras, el 10 y el as tienen valores negativos y el resto valores positivos, de esta manera mientras más alta se encuentre la cuenta hay una mayor probabilidad de que la próxima carta sea una carta de alto valor, y aumentando así la probabilidad de Blackjack

(generalmente mejor pagado en las apuestas).

Por lo tanto debemos calcular la cuenta de las cartas que ya salieron y aplicarlas a nuestra situación de juego.

Para poder hacerlo primero debemos extablecer los valores que nos provee el Sistema que utilizaremos. Dichos valores se reflejan en la siguiente tabla.

2	3	4	5	6	7	8	9	10	J, Q, K, A
+0.5	+1	+1	+1.5	+1	+0.5	0	-0.5	-1	-1

Entonces se establecen las siguientes reglas en Prolog para corresponder cada carta con su valor.

```
halves(card(Number,_),1.5):- member(Number,[5]).
halves(card(Number,_),1):- member(Number,[3,4,6]).
halves(card(Number,_),0.5):- member(Number,[2,7]).
halves(card(8,_),0).
halves(card(9,_),-0.5).
halves(card(Number,_),-1):- member(Number,[10,j,q,k,a]).
```

Y se crea la regla halvesTotal que recibe una lista de cartas y devuelve su valor en Halves

```
halvesTotal([],0):-!.
halvesTotal([Card|Rest],Halves):-
    halvesTotal(Rest, Aux),
    halves(Card,Value),
    Halves is Aux + Value,!.
```

Ya tenemos una forma de sopesar las cartas que ya se jugaron, ahora estamos en condiciones de plantear el funcionamiento de la regla *play*

La regla en sí

Gracias al conteo de las cartas que aparecieron anteriormente la decisión se reduce a evaluar lo que hay sobre la mesa.

Se calcula la suma total de los valores de las cartas del Croupier, las cartas en la mano del jugador y las que ya aparecieron y se piensa de la siguiente forma.

```
play(Hand,Croupier,Cards) :- halvesPlay(Hand,Croupier, Cards).

halvesPlay(Hand,Croupier,Cards) :-
    hand(Hand, Value),
```

```
    halvesTotal(Croupier, HCards),
    halvesTotal(Cards, HCroupier),
    halvesDecide(Value,HCroupier,HCards).
```

- Si la mano del jugador es menor o igual a 11 y la cuenta es mayor que 0 se pide otra carta.

  Por lo siguiente: la cuenta mayor a cero indica que ya salió una cantidad interesante de números, por lo cual se elevan la posibiliadades de que salga una figura y así sumar 21 o cerca con la mano del jugador cerca de 11.

```
halvesDecide(Value,HCroupier,HCards):-
    Value =< 11,
    HTotal is HCroupier + HCards,
    HTotal > 0.
```

- Si la mano es mayor a 11 y la suma es menor a 0 el jugador se planta.

  Ya salió una cantidad interesante de números y la mano del jugador se encuentra en un número alarmante. La aparición de un número alto o de una figura nos deja autáticamente fuera del juego

```
halvesDecide(Value,HCroupier,HCards):-
    Value > 11,
    HTotal is HCroupier + HCards,
    HTotal < 0.
```