Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Inteligencia Artificial:

Blackjack-Prolog

Ginés Cruz Chávez
(alu0101431079@ull.edu.es)

Índice:

1. Introducción **2**
    1.1. Reglas del Blackjack 2

2. Desarrollo **2**
    2.1. Prolog 3
    2.2. JavaScript 7

3. Dificultades encontradas **9**

4. Conclusión **10**

5. Enlaces de interés **10**

1. Introducción

En esta práctica se ha desarrollado una implementación del famoso juego de cartas Blackjack utilizando Prolog y JavaScript. Para facilitar la comunicación entre ambos lenguajes, se ha utilizado la librería Tau-Prolog. El objetivo del proyecto es crear una versión funcional del juego jugable a través de una interfaz web, utilizando a su vez una base de conocimiento desarrollada en Prolog. En este informe se describirán los detalles técnicos del proceso de desarrollo del proyecto, así como los desafíos y soluciones encontrados durante el proceso de creación.

1.1. Reglas del Blackjack

Blackjack es un juego de azar que se juega con una o más barajas de 52 cartas. El objetivo del juego es obtener una mano de 21 puntos o una mano lo más cercana a 21 puntos posible sin pasarse. Al comienzo del juego, el jugador recibe dos cartas y el crupier recibe una carta visible y otra oculta. Las cartas numeradas del 2 al 10 tienen un valor igual a su número, mientras que las figuras (jotas, reinas y reyes) valen 10 puntos y el as vale 1 o 11 puntos, según sea conveniente para el jugador. El jugador tiene la opción de pedir más cartas hasta que decida plantarse o hasta que se pase de 21. Una vez que el jugador se plante, el crupier muestra su segunda carta y juega según un conjunto de reglas predeterminadas. Si el crupier se pasa de 21, el jugador gana, en caso contrario, ganará la persona que haya llegado más cerca a 21. Si tanto el jugador como el crupier obtienen la misma puntuación, se produce un empate.

2. Desarrollo

El desarrollo del proyecto se estructura en dos componentes principales: Prolog, y JavaScript. A continuación se describirá el proceso de desarrollo en detalle, incluyendo la implementación de ambos de los componentes mencionados.

2.1. Prolog

La primera parte que se ha desarrollado de esta práctica ha sido la base de conocimiento utilizando Prolog. Tanto la lógica de coger cartas, el comportamiento de la inteligencia artificial del crupier y las reglas que determinan la victoria y derrota del juego han sido implementadas como reglas o predicados Prolog que, junto a la interfaz gráfica, construyen la totalidad del juego.

```
:- use_module(library(random)).
:- use_module(library(lists)).
```

Al comienzo de la base de conocimiento se incluyen los módulos Prolog a utilizar en esta práctica: el generador de números y permutaciones aleatorias y las funciones que nos permiten manipular listas.

```
% cards/2 devuelve todas las cartas en una baraja de cartas
cards(Deck) :-
    Suits = [spades, hearts, clubs, diamonds],
    Cards = [ace, two, three, four, five, six, seven, eight, nine, ten,
jack, queen, king],
    findall(card(Suit, Card), (member(Suit, Suits), member(Card,
Cards)), Deck).
```

El primer predicado es "cards/2". Este predicado toma una lista (llamada "Deck") y la rellena con todas las cartas en una baraja de cartas. La baraja se construye a partir de cuatro palos (spades, hearts, clubs, diamonds) y trece tipos de cartas (ace, two, three, four, five, six, seven, eight, nine, ten, jack, queen, king). El predicado "findall/3" es utilizado para emparejar todos los tipos de carta y palo posible, y colocarlos en la lista Deck. En este caso, las cartas se han implementado como una estructura tipo **card(Suit, Card)**, donde Suit es el palo de la carta y Card el tipo de carta.

```
% value/2 devuelve el valor de una carta
value(card(_, ace), 11).
value(card(_, two), 2).
value(card(_, three), 3).
value(card(_, four), 4).
value(card(_, five), 5).
value(card(_, six), 6).
```

```
value(card(_, seven), 7).
value(card(_, eight), 8).
value(card(_, nine), 9).
value(card(_, ten), 10).
value(card(_, jack), 10).
value(card(_, queen), 10).
value(card(_, king), 10).
```

El segundo predicado es "value/2", que toma una carta y devuelve su valor. El valor de las cartas numeradas del 2 al 10 es igual al número de la carta, mientras que las figuras (jack, queen, king) y el 10 valen 10 puntos y el as puede valer 1 o 11 puntos.

```
% shuffle/2 baraja una lista de cartas de manera aleatoria
shuffle(Cards, Shuffled) :-
    random_permutation(Cards, Shuffled).
```

El tercer predicado es "shuffle/2", que baraja una lista de cartas de manera aleatoria utilizando el predicado "random_permutation/2" de la librería random. De esta manera, se genera una permutación aleatoria de la lista de cartas pasada por parámetro.

```
% draw_card/2 elimina la primera carta de una baraja y la devuelve
draw_card([Card | Rest], Card, Rest).
```

El cuarto predicado es "draw_card/2", que elimina la primera carta de una lista de cartas y la devuelve junto con la lista restante sin la carta eliminada.

```
% draw_card_into_hand/3 agrega una carta a una mano de cartas
draw_card_into_hand(Deck, NewDeck, Hand, NewHand) :-
    draw_card(Deck, Card, NewDeck),
    append(Hand, [Card], NewHand).
```

El tercer predicado es "draw_card_into_hand/3", que agrega una carta a una mano de cartas. Utiliza "draw_card/2" para sacar una carta de la baraja y luego utiliza el predicado "append/3" para agregar esa carta a la mano de cartas.

```
% score/2 calcula el puntaje de una mano de cartas
score(Hand, Score) :-
    score(Hand, 0, Score).
```

```
% score/3 es un predicado auxiliar para score/2
score([], Acc, Acc).
score([card(_, Type) | Rest], Acc, Score) :-
    value(card(_, Type), Value),
    (Type = ace, Acc + 11 > 21 ->
        NewAcc is Acc + 1
    ;   NewAcc is Acc + Value),
    score(Rest, NewAcc, Score).
```

El cuarto predicado es "score/2", que calcula el puntaje de una mano de cartas. Utiliza la recursión para sumar el valor de cada carta en la mano y devolver el resultado, además de un predicado auxiliar "score/3" para realizar la recursión.

El predicado auxiliar "score/3" comienza con un caso base, cuando la mano de cartas está vacía. En este caso, simplemente devuelve el acumulador de puntos "Acc". En el caso recursivo, el predicado obtiene el valor de la primera carta en la mano y lo agrega al acumulador. Si la carta es un as, entonces se verifica si agregar 11 puntos al acumulador haría que el puntaje total sea mayor que 21. Si es así, entonces el as solo vale 1 punto. Luego, se llama recursivamente al predicado con la mano restante y el acumulador actualizado.

```
% initial_setup/3 inicializa el juego de blackjack repartiendo dos
cartas a cada jugador
% y devuelve las cartas de cada jugador y el resto de la baraja
initial_setup(PlayerCards, DealerCards, Rest4) :-
    cards(Deck),
    shuffle(Deck, ShuffledDeck),
    draw_card(ShuffledDeck, PlayerCard1, Rest1),
    draw_card(Rest1, DealerCard1, Rest2),
    draw_card(Rest2, PlayerCard2, Rest3),
    draw_card(Rest3, DealerCard2, Rest4),
    PlayerCards = [PlayerCard1, PlayerCard2],
    DealerCards = [DealerCard1, DealerCard2].
```

El quinto predicado es "initial_setup/3", que inicia el juego repartiendo dos cartas a cada jugador. Utiliza "cards/2" para obtener una baraja de cartas y luego la baraja con "shuffle/2". A continuación usa "draw_card/2" para sacar cuatro cartas de la baraja y asignar dos cartas a cada jugador. Devuelve las cartas de cada jugador y el resto de la baraja.

```
% dealer_turn/2 juega el turno del crupier hasta que su puntaje sea
mayor o igual a 17
dealer_turn(Deck, DealerCards, NewDealerCards) :-
    score(DealerCards, DealerSum),
    DealerSum < 17,
    draw_card_into_hand(Deck, NewDeck, DealerCards, NewDealerCards),
    dealer_turn(NewDeck, NewDealerCards, NewDealerCards).
dealer_turn(Deck, DealerCards, DealerCards).
```

El sexto predicado es "dealer_turn/2", que juega el turno del crupier. Mediante la recursión, se sigue pidiendo cartas mientras que el puntaje del crupier sea menor que 17. Una vez superada esa puntuación, se devuelve la nueva lista de cartas después de haber jugado al turno. Para calcular el puntaje del crupier se utiliza “score/2” y "draw_card_into_hand/3" para agregar una nueva carta a la mano del crupier.

```
% determine_winner/3 determina el ganador del juego
determine_winner(PlayerCards, DealerCards, Winner) :-
    score(PlayerCards, PlayerScore),
    score(DealerCards, DealerScore),
    (PlayerScore > 21 ->
        Winner = dealer
    ;   DealerScore > 21 ->
        Winner = player
    ;   PlayerScore > DealerScore ->
        Winner = player
    ;   DealerScore > PlayerScore ->
        Winner = dealer
    ;   Winner = push).
```

El séptimo predicado es "determine_winner/3", que determina el ganador del juego. Utiliza "score/2" para calcular los puntajes del jugador y del crupier, y verifica varias condiciones para determinar quién ha ganado el juego. Si el puntaje del jugador es mayor que 21 (se ha pasado), entonces el crupier gana. Si los puntajes son iguales, entonces hay empate. En el caso restante, ganará aquella persona con puntaje mayor.

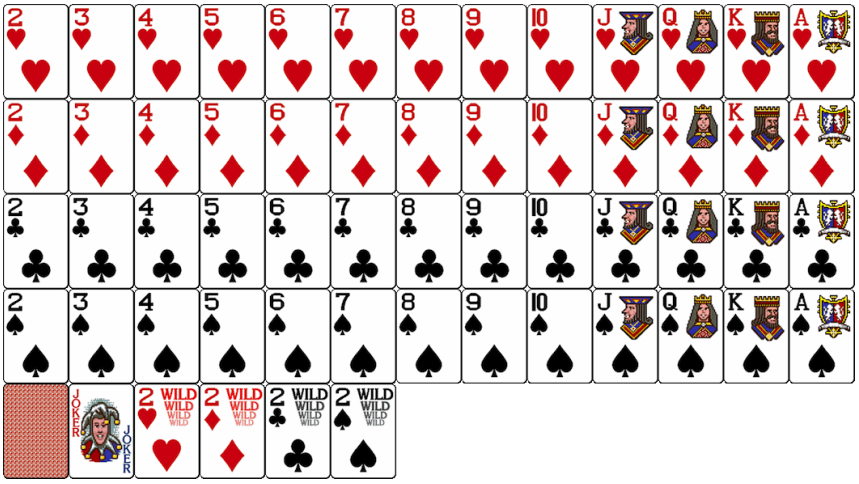
```
% has_player_busted/1 devuelve true si el jugador se pasó de 21
has_player_busted(PlayerCards) :-
    score(PlayerCards, PlayerSum),
    PlayerSum > 21.
```

El último predicado es "has_player_busted/1". Utiliza "score/2" para calcular el puntaje del jugador y luego verifica si es mayor que 21. Si es así, entonces devuelve verdadero, de lo contrario devuelve falso. Este predicado se utiliza para determinar si el jugador ha "estallado" en el juego y ha perdido automáticamente.

2.2. JavaScript

Para la implementación de la parte JavaScript, se ha optado por el uso de la librería Tau-Prolog. Esta librería permite la comunicación entre JavaScript y Prolog en aplicaciones web, y en este caso hemos conectado la base de conocimiento detallada anteriormente con la aplicación web que cumplirá la función de interfaz gráfica. Tal implementación permite aprovechar la potencia y flexibilidad de Prolog para el manejo del juego y el cálculo de las jugadas, mientras que al mismo tiempo es capaz de crear una interfaz atractiva y fácil de usar para el usuario final.

Para dibujar los elementos en la pantalla, se ha utilizado un Canvas de HTML5, que nos permite dibujar tanto los gráficos de las cartas como los textos que indican el estado de la partida. Para dibujar cada carta en la pantalla, se obtuvo un atlas de texturas donde se podían encontrar todos los gráficos necesarios para dibujar cada carta posible de la baraja francesa.



Para tener un funcionamiento básico de la interfaz gráfica se necesitaron dos funciones, una que dibuje una carta particular en pantalla, y otra que dibuje el estado actual de la partida, mostrando las cartas que tiene actualmente el jugador y el crupier.

Con respecto a la primera función para saber qué carta debemos dibujar, convertimos el id de la carta obtenido en Prolog a un índice numérico, que utiliza para buscar en el atlas de texturas el gráfico correspondiente a esa carta. Dado que todas las cartas tienen el mismo tamaño en píxeles, es muy sencillo encuadrar en un rectángulo las coordenadas de la carta que se quiere dibujar en el atlas de texturas, utilizando la función `ctx.drawImage`.

En el caso de dibujar el estado de la partida, se dibujan dos textos, uno especificando el nombre del jugador y otro el del crupier. A continuación, se dibuja para cada persona las cartas que tiene en la mano actualmente, recorriendo las listas de cartas que se le hayan pasado a la función por parámetro. Opcionalmente, podemos escoger si ocultar la primera carta del crupier o no para hacer el juego más realista, ya que el jugador no debe conocer una de las cartas de la mano del crupier hasta que su turno se haya finalizado, y el crupier comience a tomar cartas de la baraja.

Al cargar la página, se crea una sesión de Prolog y se carga la base de conocimientos del juego de Blackjack. Luego se obtienen las cartas iniciales para el jugador y el crupier y se muestran en la interfaz gráfica.

Se establecen manejadores de eventos para los botones "hit" y "stand" que permiten al jugador pedir una carta o finalizar su turno, respectivamente. Cuando se hace clic en el botón "hit", se utiliza `playerDrawCard` para pedir una carta y se muestra en la interfaz gráfica. Se utiliza `playerHasBusted` para verificar si el jugador se ha pasado de 21 y en ese caso se muestra un mensaje en la pantalla para mostrar que ha perdido. Cuando se hace clic en el botón "stand", se utiliza `dealerTurn` para que el crupier juegue su turno y se muestran las cartas del crupier en la interfaz gráfica. Finalmente, se utiliza `determineWinner` para determinar quién ha ganado la partida y se muestra un mensaje en la pantalla.

Para almacenar las listas de cartas del jugador, del crupier y de la baraja, se han utilizado arrays de JavaScript. Sin embargo, a la hora de dibujar las cartas, la aplicación debe convertir el formato card(Suit, Card) a algo más fácil con lo que trabajar para saber qué carta dibujar, por lo que también se ha implementado una función auxiliar que trabaja con listas de cartas para facilitar el dibujado de estas por pantalla.

```
// Determina si el jugador se ha pasado o no
async function playerHasBusted(session, hand) {
    const goal = "has_player_busted(" + hand.toString() + ").";
    await session.promiseQuery(goal);
    const answer = await session.promiseAnswer();
    return answer;
}
```

Como ejemplo de realización de consulta utilizando Tau-Prolog, se comentará este fragmento de código. El primer paso es especificar qué función de Prolog queremos consultar, y los parámetros que queremos pasar. En este caso, se convierte el array de cartas pasado por parámetro a una cadena para poder incluirlo en la consulta.

A continuación, con promiseQuery podemos esperar a que se consulte la base de conocimientos y obtengamos la respuesta a la regla que queremos consultar. Por último, utilizaremos esa respuesta obtenida como valor de retorno de la función.

De esta manera, podemos hacer funciones en JavaScript análogas a las de la base de conocimiento en Prolog, para así gestionar toda la lógica e inteligencia artificial de la partida de manera sencilla e intuitiva. Además, si en algún momento se quisiera cambiar de Prolog a otro sistema de funcionamiento lógico, esta estructura de código permite su intercambio fácilmente.

3. Dificultades encontradas

Una de las principales dificultades encontradas durante el desarrollo de este proyecto fue la elección del lenguaje de programación y la librería adecuados. Al principio se investigaron varias opciones como Python o SWI-Prolog, pero no contaban con la capacidad necesaria para implementar este tipo de interfaz gráfica o simplemente no funcionaban de manera correcta.

Otra dificultad fue la búsqueda de un atlas de textura adecuado para la baraja de cartas. Muchos de los atlas disponibles en internet no tenían una calidad suficientemente alta o no tenían todas las cartas necesarias.

La lógica involucrada en el cálculo de la puntuación del Blackjack es relativamente compleja, ya que debemos tener en cuenta el valor de cada carta individual y cómo se combinan entre sí. Además, las reglas del juego establecen que los ases pueden valer tanto 1 como 11, lo que aumenta aún más la

complejidad del cálculo. Sin embargo, se terminó implementando una solución recursiva que calculaba de manera precisa la puntuación de cada mano.

4. Conclusión

En conclusión, el proyecto Blackjack-Prolog ha demostrado ser una implementación efectiva del juego de Blackjack utilizando Prolog y JavaScript. Al utilizar Tau-Prolog como intermediario, se aprovecha la potencia y flexibilidad de Prolog para manejar el lógico del juego mientras que la interfaz gráfica es atractiva y fácil de usar para el usuario final.

El resultado final del proyecto es el siguiente:

5. Enlaces de interés

1. Tau-Prolog: http://tau-prolog.org/
2. HTML Canvas: https://developer.mozilla.org/es/docs/Web/HTML/Element/canvas