Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

PRÁCTICA 1 – AKINATOR DE PELÍCULAS

Conocimiento y Razonamiento Automatizado

Grado en Ingeniería Informática – Curso 2018/2019

Eduardo Graván Serrano – 03212337L

Marcos Barranquero Fernández – 51129104N

INTRODUCCIÓN

La práctica consiste en realizar un programa similar al juego Akinator, centrado exclusivamente en películas.

El programa tiene un menú, que invita a jugar. Si se elige jugar, se comienza a hacer una serie de preguntas establecidas, en relación a las características de las películas. Conforme se responden estas preguntas, se va haciendo criba de posibles películas candidatos. Cuando queda sólo una, se pregunta si es esa. De acertar, se muestra por pantalla y se lanza el menú. Si no es esa, significa que la base de datos no contiene esa película, y por tanto se debe añadir a ésta.

El proyecto está dividido en dos ficheros Prolog:

- películas.pl: que contiene los predicados que guardan información sobre las películas que hay en la base de datos.
- main.pl: que contiene toda la lógica de la aplicación.

PELICULAS.PL

La funcionalidad y el funcionamiento de este primer archivo es bastante simple. Se encarga simplemente de almacenar sentencias de Prolog que tienen dos componentes.

El primero de ellos es el título de la película en sí, que prácticamente sólo se usa a la hora de mostrar los resultados del juego. El segundo componente es una lista de tamaño variable que almacena las características de cada película.
La lista de características que se almacenan es:

- Década: década en la que se estrenó la película.
- Productora: la productora de la película.
- País: país en el que se grabó la película. (En coproducciones, el principal).
- Protagonista: el género del protagonista (hombre o mujer).
- Mundo: si está ambientada en un mundo ficticio o real.
- Género: género principal, es decir, acción, comedia, drama…
- Ambientación: si está ambientada en el pasado (<1960), futuro o contemporánea.
- Saga: en caso de que sea parte de una saga.
- Guerra: si la película contiene o es parte de una guerra.
- Nazis: si en la película hay nazis.
- Corta: si la película dura menos de 2 horas se considera corta.

Ya que la mayoría de estas características son a su vez términos de prolog, permite agrupar ciertas características bajo un mismo nombre.

Por ejemplo, en vez de almacenar cada década por separado, podemos agruparlas haciendo que la década se encuentre dentro de un predicado de prolog.

De esta forma conseguimos normalizar la forma de acceder a estas características. Esta aproximación se ha usado con características que pudiesen tener diferentes valores, como las décadas, productoras, etc.

Se ha definido que el predicado películas/2 sea dinámico, para poder crear nuevas entradas en este archivo desde el propio juego.

El archivo de películas es consultado y cargado desde el archivo principal nada más empezar la ejecución para poder ser accedido.

MAIN.PL

FUNCIONAMIENTO GENERAL

1. Lanzar menú y preguntar si se quiere jugar. En caso de que no, salir del programa.
2. Leer archivo de películas y cargarlo en la lista de películas candidatas, y mostrar instrucciones.
3. Ir realizando preguntas sobre características, y descartar posibles películas candidatas.
4. Para cada pregunta, comprobar qué elementos de la lista de candidatos cumplen o no cumplen la sentencia, y eliminarlos de la lista en función de ello.
5. Cuando sólo queda 1 candidato, se intenta adivinar. Si se acierta, se lanza el menú (paso 1).
6. Si no se acierta, se descarta y se sigue preguntando hasta el final, a fin de recolectar todas las características posibles antes de escribir la película en el archivo.
7. Se pide el título de la película y se escribe en la base de datos de películas. Después, se lanza el menú (paso 1).

Podemos dividir el archivo main en varias funcionalidades:

MENÚ

El menú es el comienzo del juego. Muestra el nombre Akinator y pregunta si se quiere jugar.

Si la respuesta es sí, se carga el fichero de películas, se muestran las instrucciones del programa y se empieza a preguntar. Si la respuesta es no, se sale de la ejecución del programa (halt).

CARGA DE PELÍCULAS

Se tienen dos métodos:

- listaPeliculas: abre el archivo de películas y le asigna un file decryptor, y llama al método de lectura de archivo. Devuelve la lista de películas, que se cargará futuramente en la lista de películas candidatas.
- leerArchivo: se lee línea a línea (película a película) el archivo, y se van añadiendo estas líneas a una lista, que se rellena en sucesivas llamadas recursivas. Si se llega al final de archivo, se detiene la llamada recursiva, y se devuelve la lista.

PREGUNTAS

Se tienen varios métodos en relación a las preguntas:

- hacerPreguntas/3: recibe la lista de candidatos y características, carga la lista de preguntas y llama a hacerPreguntas/5.
- hacerPreguntas/5: ejecuta la 1ra pregunta de la lista de preguntas (hacerPregunta_aux) y se llama recursivamente con el resto de las preguntas.
- hacerPregunta_aux: en este método se hacen las preguntas como tal y se llama a las funciones de filtro según la respuesta. Lo primero que hace es transformar el elemento de la lista de preguntas que entra como parámetro a una pregunta de verdad, que está compuesta de 3 variables. La primera de ellas es la característica por la que se está preguntando en formato adaptado para poder compararlo con la lista de películas. El segundo es una condición que se ha implantado para evitar que se hagan preguntas que no tienen sentido si ya han sido respondidas antes. Por ejemplo, si ya hemos respondido que una pregunta es de los años 60, no tiene sentido seguir preguntando por décadas. Esto se consigue haciendo que, si la variable “Repetido” se encuentra en la lista de características, simplemente actualizamos las listas de Candidatos y

Características salientes, y no hacemos la pregunta. Si la pregunta es pertinente, sacamos la parte de mensaje de la pregunta y esperamos un input. Si la respuesta es sí, filtramos acordemente. Si es no, hacemos lo mismo. Si por algún casual la respuesta no fuese valida, se vuelve a preguntar lo mismo.

COMPROBAR CARACTERÍSTICAS

Se tienen 2 principales funciones para comprobar las características y eliminar candidatos, ambas funcionan de forma similar.

- Tiene: recibe la lista de candidatos y el dato a comprobar. El dato viene dado por la pregunta realizada: década, género, etc. Para cada película, comprueba si ese dato está en sus características. Si está, la añade a una lista auxiliar y se llama recursivamente pasando los candidatos restantes y la lista auxiliar. Si no está, hace lo mismo, pero sin añadirla a la lista auxiliar. Cuando se llama con una lista de candidatos vacía, se devuelve la lista auxiliar y se escribe como nueva lista de candidatos, con los candidatos que no tenían el dato eliminados.
- No tiene: similar a tiene: para candidato, si no tiene el dato en sus características, se descarta.

La llamada a cada uno de estos métodos viene dada por la respuesta del usuario a la pregunta que se haga:

- Responde si -> Se ejecuta tiene.
- Responde no -> Se ejecuta no tiene.

ESCRIBIR A ARCHIVO

Esta funcionalidad esta implementada con una sola función, que es llamada si la película en que está pensando el jugador no se encuentra en la base de datos.

Lo primero que hace es pedir un input envuelto en comillas simples (para poder almacenar nombres que empiezan por mayúscula), que corresponde al titulo de la nueva película. Se comprueba que la película realmente no esté en el archivo y, si realmente no lo está, se añade el predicado que corresponde a la película con el titulo que introduce el usuario y la lista de características que se han ido recopilando a través de las preguntas. Una vez hecho esto se cargan todas las películas en el archivo para no perder datos usando las sentencias tell y told.

BIBLIOGRAFÍA

- Stack overflow
- Apuntes de clase
- Filmaffinity para las características de películas
- Adrián Montesinos

ANEXO – CÓDIGO

Se incluye el código fuente del fichero de lógica del programa, se considera que el que almacena los datos no es necesario.

MAIN.PL

```
% Se carga el fichero que guarda las peliculas
:-['peliculas.pl'].

% Recupera el archivo peliculas.pl en forma de lista para poder trabajar con el
facilmente
listaPeliculas(Lista_peliculas):-
    % Abre el archivo en modo lectura
    open('peliculas.pl', read, Id_archivo),
    % llama a la funcion auxiliar leerArchivo y almacena los contenidos del archivo en
la variable Lista_peliculas
    leerArchivo(Id_archivo, Lista_peliculas),
    % cierra el archivo
    close(Id_archivo).

% Funcion auxiliar que se encarga de leer el archivo
leerArchivo(Id_archivo, Lista):-
    % Leo la primera línea del archivo, guardándola en Linea
    read_term(Id_archivo, Linea, []),
    % Si Linea es el final del archivo, L es una lista vacia
    (   Linea = end_of_file -> Lista = [];
        % En caso contrario, pongo Linea en la cabeza de la lista
        % y llamo recursivamente a la funcion con el resto del archivo
        Lista = [Linea|Resto], leerArchivo(Id_archivo, Resto)
    ).


% Función que muestra el menú por la pantalla.
menu:-
    writeln('----------------------------'),
    writeln('-----Akinator peliculas-----'),
    writeln('----------------------------'),
    nl, nl,
    writeln('Deseas jugar? (si/no): '),
    read(Respuesta_usuario),
    verificar(Respuesta_usuario).

    % Si el usuario introduce no, se sale del juego.
    verificar(no):-
        nl, writeln('Hasta luego!'), nl, halt.

    % Si el usuario introduce si, se juega una nueva partida.
    verificar(si):-
        % se muestran instrucciones por pantalla.
```

```
        nl,
        writeln('Para jugar, introduce si o no, sin comillas: '),
        %writeln('s = si'), writeln('n = no'), nl,
        % Se carga la lista de todas las películas
        listaPeliculas(Peliculas),
        % Y se comienza a hacer preguntas.
        preguntar(Peliculas, Candidatos, Caracteristicas),
        %
        conclusion(Candidatos, Caracteristicas),
         menu.

preguntas([
    % Décadas
    pregunta(decada(1950), decada(_), 'Es de la decada de los 50s?: '),
    pregunta(decada(1960), decada(_), 'Es de la decada de los 60s?: '),
    pregunta(decada(1970), decada(_), 'Es de la decada de los 70s?: '),
    pregunta(decada(1980), decada(_), 'Es de la decada de los 80s?: '),
    pregunta(decada(1990), decada(_), 'Es de la decada de los 90s?: '),
    pregunta(decada(2000), decada(_), 'Es de la decada de los 00s?: '),
    pregunta(decada(2010), decada(_), 'Es de la decada de los 10s?: '),

    % Países
    pregunta(pais('EEUU'), pais(_), 'La pelicula es de EEUU?'),
    pregunta(pais('UK'), pais(_), 'La pelicula es de UK?'),
    pregunta(pais('New Zealand'), pais(_), 'La pelicula es de New Zealand?'),
    pregunta(pais('Italia'), pais(_), 'La pelicula es de Italia?'),
    pregunta(pais('Espana'), pais(_), 'La pelicula es de Espana?'),
    pregunta(pais('Irlanda'), pais(_), 'La pelicula es de Irlanda?'),
    pregunta(pais('Japon'), pais(_), 'La pelicula es de Japon?'),

    % Género
    pregunta(genero(drama), genero(_), 'Es de drama?: '),
    pregunta(genero(accion), genero(_), 'Es de accion?: '),
    pregunta(genero(ciencia_ficcion), genero(_), 'Es de ciencia ficcion?: '),
    pregunta(genero(animacion), genero(_), 'Es de animacion?: '),
    pregunta(genero(fantasia), genero(_), 'Es de fantasia?: '),
    pregunta(genero(comedia), genero(_), 'Es una comedia?: '),
    pregunta(genero(intriga), genero(_), 'Es de intriga?: '),
    pregunta(genero(crimen), genero(_), 'Es de crimen?: '),
    pregunta(genero(terror), genero(_), 'Es de terror?: '),
    pregunta(genero(aventuras), genero(_), 'Es de aventuras?: '),

    % Mundo
    pregunta(mundo(ficcion), mundo(_), 'Esta ambientada en un mundo ficticio? (o mundo
real con elementos de ficcion): '),
    pregunta(mundo(real), mundo(_), 'Esta ambientada estrictamente en el mundo real?:
'),

    % Hay guerra
```

```
    pregunta(guerra, guerra, 'Hay alguna guerra en la pelicula?: '),

    % Hay nazis
    pregunta(nazis, nazis, 'Hay algun nazi en la pelicula?: '),

    % Es cortita
    pregunta(corta, corta, 'La pelicula dura menos de 2 horas?: '),

    % Protagonista
    pregunta(protagonista(hombre), protagonista(_), 'El protagonista es un hombre?: '),
    pregunta(protagonista(mujer), protagonista(_), 'La protagonista es una mujer?: '),

    % Saga
    pregunta(saga, saga, 'La pelicula es parte de una saga?: '),

    % Ambientación
    pregunta(ambientacion(pasado), ambientacion(_), 'Esta ambientada en el pasado
(anterior a 1960)?: '),
    pregunta(ambientacion(contemporaneo), ambientacion(_), 'Esta ambientada en una epoca
contemporanea? (desde 1960): '),
    pregunta(ambientacion(futuro), ambientacion(_), 'Esta ambientada en el futuro?: '),

    % Productora
    pregunta(productora('Pixar'), productora(_), 'Es de la productora Pixar?: '),
    pregunta(productora('Warner'), productora(_), 'Es de la productora Warner?: '),
    pregunta(productora('Paramount'), productora(_), 'Es de la productora Paramount?:
'),
    pregunta(productora('Disney'), productora(_), 'Es de la productora Disney?: '),
    pregunta(productora('Wingnut Films'), productora(_), 'Es de la productora Wingnut
Films?: '),
    pregunta(productora('DreamWorks'), productora(_), 'Es de la productora DreamWorks?:
'),
    pregunta(productora('Metro'), productora(_), 'Es de la productora Metro?: '),
    pregunta(productora('Melampo'), productora(_), 'Es de la productora Melampo?: '),
    pregunta(productora('New Line'), productora(_), 'Es de la productora New Line?: '),
    pregunta(productora('Coproduccion'), productora(_), 'Es una coproduccion?: '),
    pregunta(productora('Miramax'), productora(_), 'Es de la productora Miramax?: '),
    pregunta(productora('Zoetrope'), productora(_), 'Es de la productora Zoetrope?: '),
    pregunta(productora('Orion'), productora(_), 'Es de la productora Orion?: '),
    pregunta(productora('Oz'), productora(_), 'Es de la productora Oz?: '),
    pregunta(productora('Fox'), productora(_), 'Es de la productora Fox?: '),
    pregunta(productora('Cartel'), productora(_), 'Es de la productora Cartel?: '),
    pregunta(productora('Sogetel'), productora(_), 'Es de la productora Sogetel?: ')
]).

% recupera la lista de preguntas disponibles y llama a preguntar/5
preguntar(Candidatos, Candidatos_out, Caracteristicas_out):-
    % Tomo las preguntas
    preguntas(ListaPreguntas),
```

```
    % Y llamo a preguntar/5.
    preguntar(ListaPreguntas, Candidatos, [], Candidatos_out, Caracteristicas_out).

% llama a la funcion auxiliar donde se hacen las preguntas y se llama a si misma con el
resto de preguntas para recorrer toda la lista
preguntar([Pregunta|Resto], Candidatos_in, Caracteristicas, Candidatos_out,
Caracteristicas_out):-
    realizar_pregunta(Pregunta, Candidatos_in, Caracteristicas, Candidatos_aux,
Caracteristicas_aux),
    preguntar(Resto, Candidatos_aux, Caracteristicas_aux, Candidatos_out,
Caracteristicas_out).

% Funcion de gestion de la lista vacia, que ocurre cuando ya hemos agotado las preguntas
preguntar([], Candidatos, Caracteristicas, Candidatos, Caracteristicas).

% Si solo queda 1 peli candidata, se intenta adivinarla.
realizar_pregunta(Pregunta, [P|[]], Caracteristicas, Candidatos_out,
Caracteristicas_out):-
    intentaAdivinar([P|[]]),
    % Si no era esa, se sigue preguntando para recolectar info.
    realizar_pregunta(Pregunta, [], Caracteristicas, Candidatos_out,
Caracteristicas_out).

% Funcion que hace las preguntas y llama a las funciones de filtro segun la respuesta
realizar_pregunta(Pregunta, Candidatos, Caracteristicas, Candidatos_out,
Caracteristicas_out):-
    pregunta(Dato, Repetido, Mensaje) = Pregunta,
% Convierto el elemento de la lista de preguntas a una pregunta
    (memberchk(Repetido, Caracteristicas) -> Candidatos_out = Candidatos,
Caracteristicas_out = Caracteristicas;                % Si ya hay una respuesta para la
pregunta que estamos evaluando, la saltamos
        writeln(Mensaje), read(Input),
% Si no, sacamos el mensaje por pantalla y esperamos input
                        (Input = si -> tiene(Candidatos, Dato, Candidatos_out),
Caracteristicas_out = [Dato|Caracteristicas];    % Si la respuesta es si, filtramos por
las peliculas que cumplen la condicion
                            Input = no -> no_tiene(Candidatos, Dato, Candidatos_out),
Caracteristicas_out = Caracteristicas;        % Si la respuesta es no, filtramos por las
peliculas que no cumplen la condicion
                            realizar_pregunta(Pregunta, Candidatos, Caracteristicas,
Candidatos_out, Caracteristicas_out)         % Si la respuesta no es valida, se vuelve a
formular la misma pregunta
                        )
    ).

intentaAdivinar(Candidatos):-
    Candidatos = [pelicula(Titulo, _)|[]],
    %Pelicula = pelicula(Titulo|_),
    write('Tu pelicula es '), write(Titulo), write('?\n'),
```

```
        read(Input),
        (
            Input = si -> writeln('Gracias por jugar.'), nl, menu;     % Si la respuesta
es si, gana el programa, se salta al menú.
            Input = no -> writeln('Habia que intentarlo. ')     % Si no, se sigue
preguntando características para almacenar la película.
        ).

% Si la lista de candidatos esta vacia, añadimos una nueva pelicula
conclusion([], Caracteristicas):-
    anadirPelicula(Caracteristicas).

conclusion(Candidatos, Caracteristicas):-
    Candidatos = [pelicula(Titulo, _)|Resto],                   % Coge la primera
pelicula de la lista de candidatos y almacena el resto por si acaso
    write('Tu pelicula es '), write(Titulo), write('?\n'),
    read(Input),
        (
            Input = si -> writeln('Gracias por jugar.'), nl;     % Si la respuesta es
si, gana el programa
            Input = no -> conclusion(Resto, Caracteristicas)     % Si no, llama a
recursion hasta que la lista este vacia
        ).

% Añade una nueva película al documento, con las características respondidas.
anadirPelicula(Caracteristicas):-
    nl, writeln('No conozco esa pelicula, introduce el titulo para guardarla(entre
comillas simples ''''): '),
    read(Input),
    (\+ pelicula(Input, _) ->
        assertz(pelicula(Input, Caracteristicas)),
        tell('peliculas.pl'),
        listing(pelicula),
        told
    ).

% FUNCIONES DE FILTRO
no_tiene(Peliculas_in, Dato, Peliculas_out):-
    % Llamo a la fun. aux con lista vacía.
    no_tiene_aux(Peliculas_in, Dato, [], Peliculas_out).

no_tiene_aux([Pelicula|Otras], Dato, Peliculas_aux, Peliculas_out):-
    % Si la película contiene el dato y no está en la lista aux.
    (
        % Tomo película
        pelicula(_, Caracteristicas) = Pelicula,
        % Compruebo que NO tiene el dato
        \+ memberchk(Dato, Caracteristicas),
        % Y que no la he tomado previamente
```

```
        \+ memberchk(Pelicula, Peliculas_aux)
        % Entonces
        ->
            % La añado
            Peliculas_new_aux = [Pelicula|Peliculas_aux],
            % y continúo verificando el resto de pelis
            no_tiene_aux(Otras, Dato, Peliculas_new_aux, Peliculas_out)
    ;
        %Si la película tiene el dato, me la salto y continúo:
        no_tiene_aux(Otras, Dato, Peliculas_aux, Peliculas_out)
    ).

no_tiene_aux([], _, Peliculas_aux, Peliculas_aux).

tiene(Peliculas_in, Dato, Peliculas_out):-
    % Llamo a la fun. aux con lista vacía.
    tiene_aux(Peliculas_in, Dato, [], Peliculas_out).

tiene_aux([Pelicula|Otras], Dato, Peliculas_aux, Peliculas_out):-
    % Si la película contiene el dato y no está en la lista aux.
    (
        % Tomo película
        pelicula(_, Caracteristicas) = Pelicula,
        % Compruebo que tiene el dato
        memberchk(Dato, Caracteristicas),
        % Y que no la he tomado previamente
        \+ memberchk(Pelicula, Peliculas_aux)
        % Entonces
        ->
            % La añado
            Peliculas_new_aux = [Pelicula|Peliculas_aux],
            % y continúo verificando el resto de pelis
            tiene_aux(Otras, Dato, Peliculas_new_aux, Peliculas_out)
    ;
        % Si la película no tiene el dato, me la salto y continúo:
        tiene_aux(Otras, Dato, Peliculas_aux, Peliculas_out)
    ).

tiene_aux([], _, Peliculas_aux, Peliculas_aux).
```