Práctica 5: Representación de conocimiento mediante reglas
Inteligencia Artificial

Autores (Grupo 5):
José Javier Cortés Tejada
Pedro David González Vázquez

1. Descripción del funcionamiento

El recomendador de aplicaciones se basa en una GUI implementada en Java siguiendo el diseño de un MVC junto con un observer del API de Java, además de un sistema experto implementado en Jess el cual se encarga de generar las recomendaciones para un usuario en base a varios datos que se le han solicitado, tales como su fecha de nacimiento o el país en el que viven.

Entrando en materia, la GUI contiene un panel donde se informa al usuario de los datos básicos de las aplicaciones que se le están recomendando, tales como el nombre o el precio, un segundo panel donde se informa del usuario que está usando la aplicación junto con un pequeño registro donde se le permite añadir un nuevo usuario o cambiar a un usuario ya existente en la memoria de trabajo. Para añadir un nuevo usuario basta con escribir los datos del mismo en dichos campos y pulsar el botón correspondiente, de manera que si estos son inválidos se mostrará un mensaje de error. Por otro lado, para cambiar de usuario basta con poner su nombre en el campo correspondiente y si este está en la MT se actualizará la GUI automáticamente con sus datos y una recomendación nueva en base a él. De cara a poder iniciar la aplicación sin tener que añadir ningún usuario, se han incluido dos usuarios en la memoria de trabajo con lo que se podrá probar el recomendador sn problema. Estos son:

- Pedro: del 1995, sexo masculino y de España.
- Javi: del 2004, sexo masculino y de España.

Por último tenemos en panel de las aplicaciones, que queda representado por una imagen de la misma en la que el usuario podrá clicar, con el fin de incrementar su valor de recomendación hacia dicho tipo de aplicaciones. Una vez se haya clicado en ésta, la vista hará que el controlador mande al modelo (la clase *Engine*) que aserte la nueva información en la MT para así incrementar el valor de recomendación y poder mostrar nuevas recomendaciones.

2. Cambios respecto de la práctica 4

2.1. Comocimiento sobre las *apps*

Este apartado ha sufrido variaciones en cuanto al conocimiento usado sobre las mismas, pues se han desechado parte de éste ya que hemos considerado que era innecesario para efectuar las recomendaciones o que eran poco útiles, como es el caso de la versión mínima del S.O. que corre cada aplicación o el peso de la misma. También se ha desechado la idea de permitir a los usuarios comprar aplicaciones pues supone un gran incremento en la complejidad del proyecto, el cual aporta escasos beneficios al objetivo de la aplicación en sí.

Por otro lado también se ha visto modificada la siguiente clasificación:

Libros	**Cine**	**Juegos**	**Música**
▪ Fantasía.	▪ Comedia.	▪ Puzzle.	▪ Rock.
▪ Terror.	▪ Suspense.	▪ Plataformas.	▪ Pop.
▪ Romance.	▪ Animación.	▪ Estrategia.	▪ House.
▪ Drama.	▪ Acción.	▪ Aventura.	▪ Vocaloid.
▪ Manga.	▪ Romance.	▪ Deporte.	▪ Indie

donde exponíamos como ibámos a agrupar las aplicaciones de nuestro dominio. Esto se ha llevado a cabo pues en parte debido a una definición poco útil por nuestra parte pues en el caso de *Libros* y *Música* no nos resulta demasiado útil hacer distinciones entre tipos de libros y géneros de música básicamente porque el ámbito que engloba este proyecto son las aplicaciones, no canciones ni libros en función de su género literario (aún así es cierto que muchos otros sistemas si recomiendan este tipo de contenido a pesar de estar pensados para una *store* como Google Play, aunque hemos considerado que se escapaba del objetivo de la práctica). La clasificación empleada en el desarrollo es la siguiente:

- Libros.
- Juegos.
  - Acción.
  - Aventura.
  - Puzzle.
  - Deportes.
  - Estrategia.
- Compras.
- Redes sociales.
- Navegación.
- Noticias.

3. Conocimiento utilizado y estructura modular

El conocimiento usado es el descrito en el apartado anterior, de manera que de los usuarios se solicita su nombre, su fecha de nacimiento, su sexo y país de origen, datos con los que inferiremos mediante *defrules* su edad, nivel socio-económico y también le asignaremos un perfil con unos gustos determinados, luego incrementaremos el conocimiento sobre el usuario en gran medida.

De cara a las aplicaciones, todo el conocimiento es proporcionado por el sistema, de manera que este es usado para recomendarle al usuario aplicaciones cotejando la categoria de la app con los intereses del mismo.

De cara a la estructura modula, esta es simplemente un módulo que contiene todo el código en Jess, luego tanto la información inferida de los datos del usuario como las recomendaciones se generan en el mismo sitio.

3.1. Conocimiento sobre las relaciones

En el prototipo descrito en la práctica 4 expusimos relaciones existentes entre distintas categorías y subcategorías de aplicaciones, aunque en el recomendador han sido reflejadas como perfiles de usuarios y no como relaciones dentro de la parte de Jess, de manera que a un usuario se le asigna un determinado perfil el cual se centra principalmente en aplicaciones de un mismo tipo, es decir, en aplicaciones presente dentro de una relaión.

4. Estructura del código Jess

El sistema implementado en Jess se basa en tres *deftemplates*, dos de ellas *User* y *App* son declaradas en base a las respectivas clases en Java y representan a los usuarios y las aplicaciones, mientras que la *deftemplate* Like nos permite relacionar usuarios con aplicaciones y estableerles el valor *fav*, que indica la afinidad del usuario hacia un determinado tipo de aplicaciones.

De cara al conocimiento inferido sobre el usuario, hemos creado varias *defrules* las cuales le asignan al mismo su idioma correspondiente en base al país en el que residen. También tenemos *defrules* que determinan el nivel socio-económico del usuario (este dato se aparece poco en el .clp, aunque es usado en la clase Engine.java que gestiona la comunicación entre Java y Jess con el fin de meter en la lista de recomendaciones un porcentaje bajo de aplicaciones de precios altos, incluso para usuarios con un nivel económico menos con el fin de tratar de ampliar el tipo de recomendaciones posibles) en base al país en el que viven y su edad, *defrules* y *deffunctions* que determinan la edad del usuario en base a su fecha de nacimiento y varias *defrules* extras que determinan el perfil de un usuario en base a su país y su edad.

También hemos usado *defquerys* que nos permiten hacer peticiones a la MT con el fin de obtener una lista de aplicaciones a recomendar.

5. Estructura del código Java

De cara al desarrollo de la GUI nos hemos basado en el prototipo presentado en la práctica 4, de manera que la hemos implementado con un MVC y un observer del API de Java con el fin de simplificar los cambios de la GUI tras la interacción del usuario.

En cuanto a la parte estética, hemos desechado gran parte de la idea original, que se basaba en un panel de registro de usuarios, un panel de imágenes con las recomendaciones y varias listas de imágenes con recomendaciones específicas. De todo esto hemos mantenido la idea del panel de imágenes para la interacción con el usuario y también, aunque en menor medida, el panel de registro de usuario pues ahora permite cargar un número de usuarios indeterminado (en el otro prototipo la idea era tener un usuario por cada ejecución de la aplicación, aunque no quedó reflejado correctamente en la memoria) y alternar entre ellos en cualquier momento. Además se ha añadido un panel con información complementaria sobre las aplicaciones para que sea más fácil ver el funcionamiento de la aplicación. También cabe destacar el uso del la clase *Observer* del API de Java, lo cual facilita enormemente la gestión de los cambios en la interfaz y aporta un también una gran facilidad de cara al incremento de la misma, luego estamos ante un sistema fácilmente ampliable de cara a incrementar el número de aplicaciones, usuarios o funcionalidades de la misma

Respecto de la lógica interna, esta se incluye en el controlador, encargado de comunicar el modelo y la vista, pero sobre todo en el modelo, el cual está formado por la clase ***Database*** la cual almacena el conocimiento que será introducido en la memoria de trabajo (las aplicaciones y los usuarios precargados en el sistema), además de la clase Engine encargada de comunicar la aplicación Java con el código en Jess.

6. Ejemplos de ejecución

Las ejecuciones las hemos realizado sobre dos usuarios de cuyos datos hacen que esten en perfiles diferentes, de manera que presentan la siguiente forma:

- usuario1: 1995, 'm', "Spain".
- usuario2: 1992, 'f', "Germany".

de manera que Usuario1 corresponde al perfil *Otaku*, mientras que usuario2 corresponde al perfil *hipster*.

Partimos con una larga lista de aplicaciones que se encuentra en el fichero Database.java, donde tenemos aplicaciones de todos los tipos indicados en la nueva clasificación del apartado 2, de manera que tras lanzar la aplicación vemos que las recomendaciones encajan con los perfiles que se habían inferido anteriormente, pues estos son los resultados que hemos obtenido para usuario2 y usuario1 espectivamente:

```
Name: EA SPORTS UFC
Category: Sports
Prize: 5.0$

Name: FIFA 16 Ultimate Team
Category: Sports
Prize: 5.0$

Name: Trial Xtreme 3
Category: Sports
Prize: 5.0$

Name: EMT Madrid
Category: Navigation
Prize: 0.0$

Name: Google Maps
Category: Navigation
Prize: 0.0$

Name: Casa del Libro
Category: Books
Prize: 0.0$

Name: DC Comics
Category: Books
Prize: 2.0$
```

```
Name: Messenger
Category: RRSS
Prize: 0.0$

Name: Skype
Category: RRSS
Prize: 0.0$

Name: Mobile Strike
Category: Strategy
Prize: 6.0$

Name: Casa del Libro
Category: Books
Prize: 0.0$

Name: DC Comics
Category: Books
Prize: 2.0$

Name: Ebook Reader
Category: Books
Prize: 0.0$
```