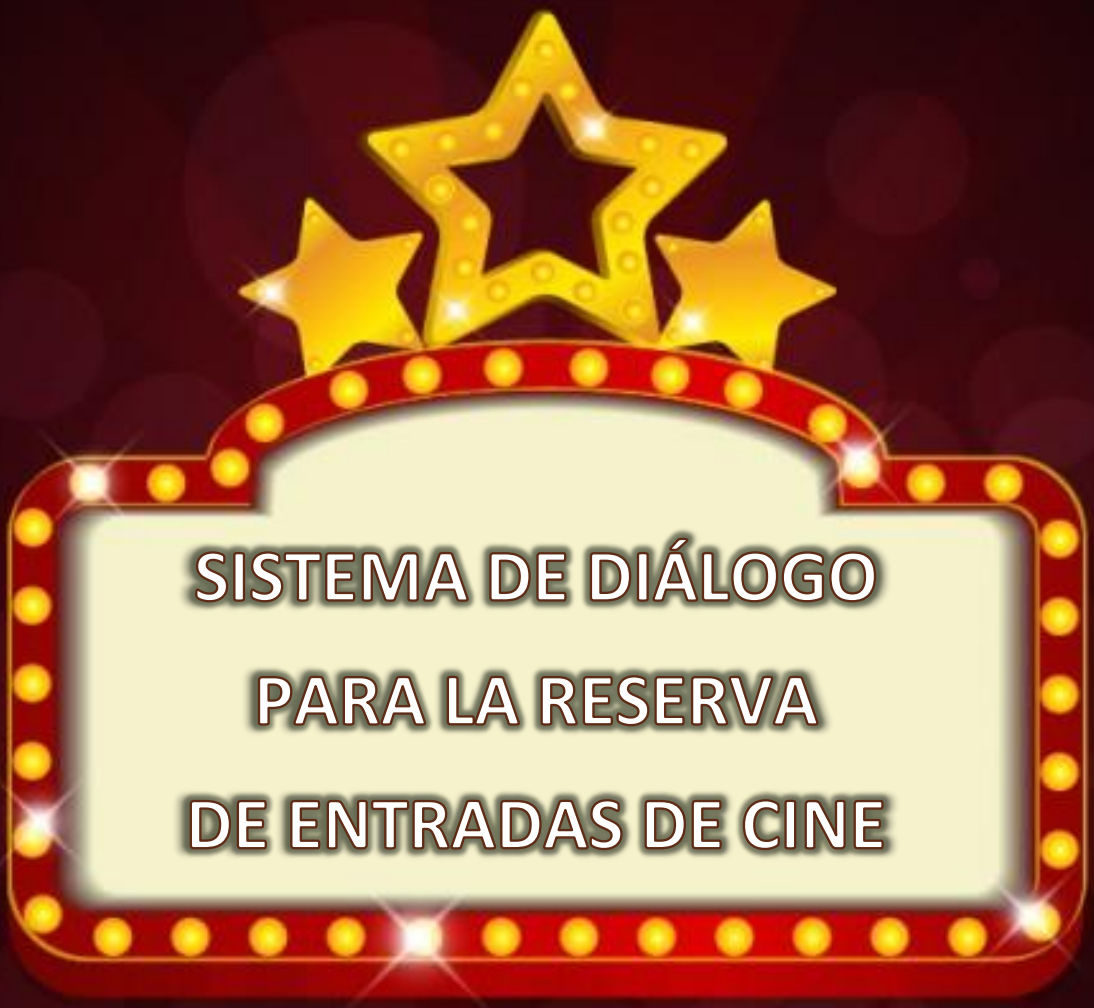

SISTEMA DE DIÁLOGO PARA LA RESERVA DE ENTRADAS DE CINE

TRABAJO DE FINAL DE GRADO

Grado en Ingeniería Informática 2016-2017

Alumno	Eric Romeu Andreu
Tutor de la UPV	Carlos David Martínez Hinarejos

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Resumen

Los sistemas de diálogo son aplicaciones en las que un ordenador interactúa de forma oral o escrita, usando diálogo, con un ser humano con el fin de conseguir un cierto objetivo. El trabajo realizado en este TFG ha consistido en recrear un sistema de dialogo escrito basado en estados, para obtener toda la información necesaria para realizar una reserva de entradas en cualquier cine de la provincia de Valencia. Su función es descargar los datos necesarios de la web de los cines de la provincia de Valencia, para almacenar una base de datos local y poder recrear una conversación con el usuario con un reconocimiento reducido de lengua escrita, con el fin de que obtenga la información necesaria para poder ir al cine que mejor le convenga.

Palabras clave: cine, reconocimiento de lengua escrita, Python, provincia de Valencia

Abstract

Dialogue systems are applications in which a computer interacts orally, using dialogue, with a human being in order to achieve a certain goal. The work done in this TFG has consisted of recreating a system of dialogue based on states, to obtain all the information necessary to make a reservation of tickets in any cinema in the province of Valencia. Its function to download the necessary data from the web of the cinemas of the province of Valencia, to store a local database and to recreate a conversation with the user with a reduced recognition of written language, in order to obtain the necessary information to be able to go to the cinema that suits him/her best.

Keywords: cinema, recognition of written language, Python, province of Valencia

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor Carlos David Martínez Hinarejos por su dedicación, rapidez de respuesta y sus consejos durante todo el proyecto.

Quisiera agradecer a la Universidad Politécnica de Valencia por aceptarme como alumno y haberme dado la valiosa oportunidad de realizar mis estudios universitarios con un profesorado siempre atento a mis necesidades.

Por último, pero no por ello menos importante, agradecer a todas las personas que están día a día a mi lado, haciendo que no decaiga en seguir siempre creciendo como persona y avanzando en mis estudios de la mejor manera posible.

Gracias

Tabla de Contenidos

1. Introducción

Este Trabajo de Final de Grado (TFG) titulado: "Sistema de diálogo para la reserva de entradas de cine", está dirigido por Carlos David Martínez Hinarejos, profesor de la Escuela Técnica Superior de Informática (ETSINF) en el Departamento. de Sistemas Informáticos y Computación (DSIC) en la Universidad Politécnica de Valencia (UPV) [1] y desarrollado desde el principio por mí, Eric Romeu Andreu, alumno de la Escuela Técnica Superior de Informática (ETSINF) de Valencia.

Breve descripción del proyecto

El proyecto se basa en la recopilación, organización y uso de la información necesaria para hacer saber al usuario que utiliza el programa la programación de una película cualquiera en uno de los cines de toda la provincia de Valencia.

El proyecto realiza un reconocimiento de lengua escrita humana dirigido solamente a la jerga cinematográfica, como pueden ser cines, películas o incluso horas de proyección o intervalos de proyección de horas, haciendo uso de una interfaz gráfica básica para el muestreo de la conversación con el usuario, con la intención de mejorar la experiencia del mismo al utilizar la aplicación.

Planteamiento

La realización del mismo se divide en dos grandes bloques de programación:

El primero de ellos se encarga de descargar la información necesaria de la web, como información de los cines, películas proyectadas en los mismos y horas de proyección de dichas películas. Actualiza la base de datos local para un funcionamiento casi instantáneo de las respuestas del programa hacia el usuario y crea todo lo necesario para que el funcionamiento del reconocimiento de la consulta del usuario sea el adecuado.

El segundo bloque es el que realiza el reconocimiento de la consulta del usuario, identificando sobre qué tipo de información está preguntando y recreando una conversación con el usuario, mostrando información para guiarlo hasta la satisfacción de saber la información necesaria para disfrutar en su cine habitual de la película deseada a la hora que mejor le sirva.

Objetivos

Este TFG se realizó con la intención de mejorar la experiencia del usuario respecto a la consulta de información a la hora de seleccionar películas en los distintos cines de la cartelera de la provincia de Valencia.

Uno de los principales objetivos era tener una base de datos local adecuada, simple y eficaz, para posteriormente mostrar toda la información guardada de la manera más rápida y sencilla posible. Por tanto, había que realizar la descarga de datos de la web, con más de 20 cines reconocidos en la provincia de Valencia, de la mejor forma posible, sencilla y eficaz para el usuario.

Por otra parte, otro objetivo sería hacer el reconocimiento del texto de las consultas del usuario rápidamente, para poder recrear así una conversación instantánea con el mismo usuario con respuestas coherentes y que al mismo tiempo guíen al usuario al conocimiento que busca, de la manera más intuitiva posible.

2. Componentes y tecnologías

Descripción de los componentes y tecnologías utilizadas en el desarrollo del programa

El lenguaje de programación elegido para el desarrollo de este proyecto en su totalidad es *Python* [3].

Python es un lenguaje de programación cuya filosofía hace hincapié en una sintaxis que favorezca un código legible y trasparente. Esto, junto con el uso del lenguaje en distintas asignaturas del grado de Informática, así como de la rama de Computación, cursada este mismo año por mí, hacen que fuese la herramienta de trabajo perfecta para desarrollar este proyecto de la manera más productiva.

Área de trabajo

En el sistema operativo *Windows* existen portales web de ejecución de código *Python*, al igual que existen programas para la ejecución de código *Python*. Pero gracias a que en mi ordenador tengo instalada una versión de *Linux* en forma de partición del disco, la misma terminal de *Linux*, con *Python* base instalado en el ordenador, fue la elección adecuada para realizar las pruebas y ejecuciones de cada modificación del programa.

La distribución de *GNU/Linux* desde la que trabajé es *Apricity OS*, arquitectura que se basa *ArchLinux* y utiliza una filosofía simplista con aplicaciones de *GNOME* para su completo desarrollo. Aunque a día de hoy los desarrolladores del mismo sistema operativo han anunciado el abandono del proyecto de desarrollo del sistema operativo por falta de tiempo, y por lo tanto queda obsoleto, aun así, sirve para realizar todas las pruebas y modificaciones necesarias.

Como *Python* es un lenguaje que no necesita ningún portal de trabajo como puedan ser *Eclipse* o *Visual Studio*, y con un simple editor de texto y la consola de *Linux* es suficiente para programar en *Python*, elegí un editor de textos para realizar las modificaciones del programa de la forma más eficaz posible. Este editor es Atom.

Atom [4] es un editor de texto que es moderno y accesible, pero modificable hasta el núcleo, una herramienta que puedes personalizar para hacer cualquier cosa, pero también usarla de manera productiva sin tocar un archivo de configuración. Por tanto, para navegar entre los archivos y modificar cada uno de ellos la interfaz de Atom fue la herramienta perfecta.

Bibliotecas utilizadas

Python tiene también muchísimas bibliotecas que hacen de su programación más sencilla [8]. El uso de ellas para el tratamiento de documentos, así como la ejecución de las instrucciones de sistema o la misma interfaz gráfica del programa, hacen que se simplifique el trabajo y mejore la experiencia de programación.

A continuación, se describen las bibliotecas utilizadas en el proyecto y la justificación de su uso.

Biblioteca *OS*

Este módulo [9] proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo.

Si sólo desea leer o escribir un archivo se utiliza el método *open ()*. Para manipular rutas o comprobar si existen algunos ficheros se utiliza el modulo *os.path*.

El método que durante el proyecto se utiliza se llama `os.path.isfile(ruta)`, que nos sirve para saber si existe un fichero o no.

Biblioteca *Math*

Este módulo está siempre disponible. Proporciona acceso a las funciones matemáticas definidas por el estándar C.

El módulo *Math* [10] proporciona un sinfín de funciones, pero en nuestro proyecto solo utilizaremos `math.log10(x)`, para hacer el logaritmo en base 10 en el cálculo de la Frecuencia Inversa del Documento (IDF) [18] para un término, que se explica posteriormente.

Biblioteca *subprocess*

El módulo de *subprocess* [11] permite generar nuevos procesos, conectarse a sus canales de entrada/salida/error y obtener sus códigos de retorno.

Con el método `subprocess.call(args)` se ejecuta por terminal la orden indicada en *args*. Para nuestro proyecto, será crucial a la hora de eliminar carpetas, eliminar la base de datos o incluso ejecutar órdenes de descarga de distintas webs.

Biblioteca *SQlite3*

SQLite3 [15] es una biblioteca *C* que proporciona una base de datos ligera basada en disco, que no requiere un proceso de servidor independiente y permite acceder a la base de datos utilizando una variante no estándar del lenguaje de consulta SQL.

Es una manera sencilla de crear una base de datos local y hacer consultas sobre ella, por lo que SQlite3 es la biblioteca utilizada para crear la base de datos simple para este proyecto.

Biblioteca *difflib*

Este módulo [12] proporciona clases y funciones para comparar secuencias. Puede ser utilizado, por ejemplo, para comparar archivos, y puede producir información de diferencia en varios formatos, incluyendo HTML y contexto.

La clase *difflib.SequenceMatcher*, que es la que se utiliza en el proyecto, es una clase flexible para comparar pares de secuencias de cualquier tipo. La idea es encontrar la sub-secuencia coincidente contigua más larga dentro de la misma palabra. Esa misma idea se aplica entonces recursivamente a las piezas de las secuencias a la izquierda y a la derecha de la sub-secuencia coincidente.

En este proyecto, esta clase se utiliza para encontrar un porcentaje de similitud entre palabras que forma parte de un algoritmo más completo, descrito posteriormente.

Biblioteca *time*

El módulo *time* [13] proporciona varias funciones relacionadas con el tiempo. En nuestro caso se utilizará `time.strftime("%d/%m/%y")` para obtener en formato estándar la fecha exacta en la que se ejecuta el proyecto y así poder identificar si los datos están actualizados o no.

Biblioteca *datetime*

El módulo *datetime* [14] proporciona clases para manipular fechas y horas de formas sencillas y complejas. Como en nuestro caso guardamos las fechas en formato de texto, sumar y restar horas sería muy complejo, pero gracias a la clase *datetime* y el formato de fecha estándar se puede realizar de manera más sencilla ese incremento y decremento de horas.

Biblioteca *shlex*

La clase *shlex* [16] facilita la escritura de analizadores léxicos para sintaxis simples. Esto a menudo será útil para escribir mini lenguajes (por ejemplo, en archivos de control de ejecución para aplicaciones *Python*) o para analizar cadenas entre comillas.

La función utilizada en este proyecto es `shlex.split (s)`, útil para dividir la cadena `s` usando la sintaxis de tipo shell.

Biblioteca *unicodedata*

Este módulo [17] proporciona acceso a la base de datos de caracteres Unicode que define las propiedades de carácter para todos los caracteres Unicode. Los datos de esta base de datos se basan en el archivo `UnicodeData.txt` versión 5.2.0 que está disponible públicamente desde ftp://ftp.unicode.org/.

Este módulo se utiliza para normalizar en formato NFC (Forma Normal Compuesta) algunas palabras para su posterior comparación de similitud entre palabras en el reconocimiento de lenguaje escrito.

Bibliotecas de interfaz gráfica para *Python*

Entre las bibliotecas de interfaz gráfica que existen en *Python* había varias que servían para realizar una interfaz sencilla para hacer la interacción con el usuario. En las tablas 1, 2 y 3 se describen algunas de las que contemplé previamente a elegir la adecuada.

Tabla 1: Ventajas y desventajas de la biblioteca WxPython [7].

WxPython	
Ventajas	Desventajas
-Completo conjunto de elementos gráficos (listados, arboles, etc.). -No se cierra en el mínimo denominador común; soporta las características comunes de *Windows*, y las emula en *Linux*/*Mac OS* cuando no se pueden hacer nativamente (y viceversa). -Permite separar completamente el diseño de la interface en *XML* del código *Python*. -Es fácil armar componentes personalizados, tanto que incorpora widgets que no están en *wxWidgets* mismo, ya que están escritos en *Python* -Documentación completa y ejemplos extensivos.	-No viene preinstalado con *Python*, se debe instalar un paquete. -Relativamente más complejo de aprender. -Al tener un desarrollo bastante rápido y sostenido, se liberan versiones frecuentemente, lo que en la práctica le confiere cierto nivel de "volatilidad" y problemas de compatibilidad si se deben mantener varias versiones para el mismo código. -Las características emuladas de otras plataformas no siempre se ven bien.

Tabla 2: Ventajas y desventajas de la biblioteca PyQt [7].

PyQt	
Ventajas	Desventajas
-Completo conjunto de elementos gráficos (listados, arboles, etc.) -Posee un mecanismo de conexión de señales y eventos simple. -Se puede definir los eventos más sencillos en un diseñador de interfaces graficas (ejemplo: al pulsar este botón, borrar este campo de texto) y en el código *Python* definir las acciones más avanzadas. -Se puede separar el diseño de la interfaz. -Arquitectura opcional para Modelo/Vista para las tablas, listas y árboles.	-No viene preinstalado con *Python*, se debe instalar por separado. -Relativamente más complejo de aprender. -En ocasiones emerge la implementación en *C++* subyacente, teniendo que hacer *castings* entre tipos de datos, etc. -No hay mucha documentación específica en *Python*, ya que el lenguaje en sí no es demasiado considerado.

Tabla 3: Ventajas y desventajas de la biblioteca PyGTK [7].

PyGTK	
Ventajas	Desventajas
-Completo conjunto de elementos gráficos (listados, arboles, etc.). -Flexible y potente control del comportamiento de la interfaz. -Es estable, y los mensajes de error son correctos.	-No viene preinstalado con *Python*, se debe instalar por separado. -Relativamente más complejo de aprender. -Relativamente lento en *Windows* (dibuja cada botón, etiqueta, menú, etc.), lo que le da una apariencia "extraña". -En *Windows* es la librería que tiene más dependencias, y se instalan por separado. -Aparentemente tiene la documentación más precaria de todos.

Biblioteca *Tkinter*

Tabla 4: Ventajas y desventajas de la biblioteca Tkinter [6]

Tkinter	
Ventajas	Desventajas
-Preinstalado con *Python* en casi todas las plataformas. -Relativamente simple y fácil de aprender. -Documentación completa.	-Pocos elementos gráficos (sin listados, arboles, etc.). -Limitado control del comportamiento de la interface (recomendado para proyectos "triviales"). -Apariencia "extraña" (no se parece a las aplicaciones nativas) **

Nota **: cabe aclarar que las últimas versiones de *Tkinter* mejoran varios de estos puntos, dibujando con las funciones nativas de la plataforma, lo que acelera y mejora la apariencia.

Tkinter [6] es el paquete estándar de interfaz gráfica de usuario de *Python* elegido para dar forma a la interfaz gráfica de este proyecto.

Tkinter no es el único kit de herramientas de programación gráfica de interfaz de usuario para *Python*. Sin embargo, es el más utilizado. Las características de *Tkinter* se muestran en la tabla 4.

Al ser entre todos los paquetes de interfaz gráfica vistos el más sencillo de utilizar y tener la ventaja de venir preinstalado en *Python* en casi todas las plataformas, hace la herramienta perfecta para implementar la sencilla interfaz gráfica del programa.

3. Implementación del sistema

La clase main.py (Estructura y orden de la información)

La clase `main.py` es la que se encarga de ejecutar las distintas clases del programa en el orden adecuado, para el correcto funcionamiento del proyecto. Para ello, ejecuta en primer lugar una clase que prepara todas las carpetas y subcarpetas necesarias para que el programa funcione correctamente, y crea, en caso de no existir, el contenedor de la base de datos.

Organización de datos en disco

Para tener la información organizada en carpetas y que tengan una coherencia a la hora de buscar la información, elegí la organización mostrada en la Figura 1.

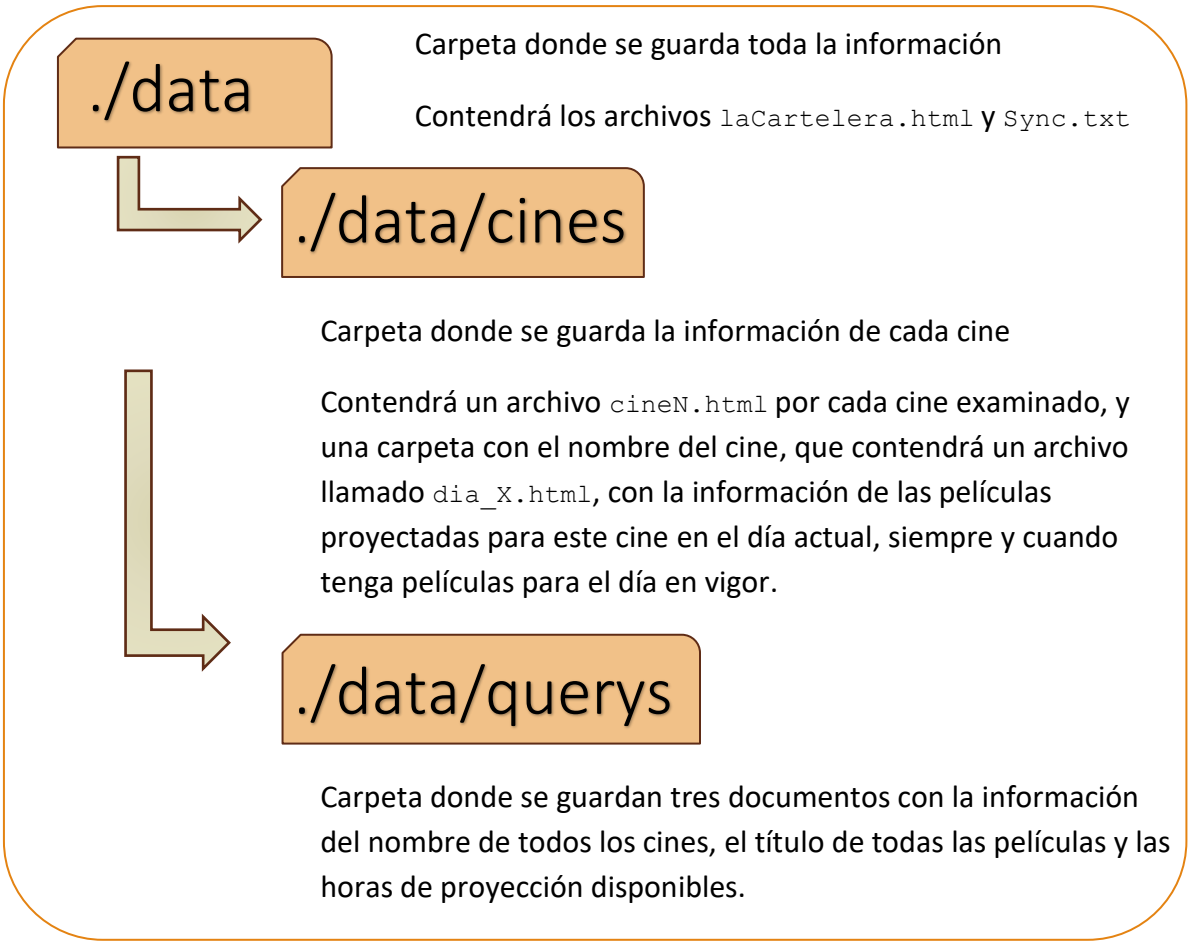

Figura 1: Organización de los datos en disco.

Creación y estructura de la base de datos

Elegí *SQLite* [15], entre varios, como sistema de gestión de base de datos debido a que utiliza llamadas simples a subrutinas y funciones, lo que reduce la latencia de acceso a la base de datos. Otra razón es que el conjunto de datos guardados por la base de datos, como definiciones, tablas, índices y los mismos datos, son guardados como un solo fichero estándar. Entonces, para la creación de la base de datos sencilla para mi programa, fue la elección perfecta.

La estructuración de la base de datos local se ciñe a dos tablas, que se relacionan entre ellas, guardando toda la información pertinente tanto de cines como películas.

Las tablas creadas y sus correspondientes campos internos se muestran en las tablas 5 y 6.

Tabla 5: Descripción de los campos de la tabla cinemas de la base de datos.

cinemas	
id	Campo entero, para hacer única la referencia al cine.
name	Campo de texto, con el nombre del cine en referencia.
direction	Campo de texto, con la dirección indicada por la web del cine.
coordX	Campo decimal, con la latitud en el mapa de coordenadas de la ubicación del cine.
coordY	Campo decimal, con la longitud en el mapa de coordenadas de la ubicación del cine.
price	Campo de texto, con la información de precios del cine.

Tabla 6: Descripción de los campos de la tabla cinema_movie de la base de datos.

cinema_movie	
id	Campo entero, que hace la función de identificador único para no repetir películas en el mismo cine a la misma hora.
cinema_id	Campo entero, que hace referencia al *id* del cine donde se proyecta la película.
name	Campo de texto, con el nombre de la película.
duration	Campo entero, con la duración de la película.
type	Campo de texto, donde se indica si la película es en 3D o Digital.
hour	Campo de texto, con la hora de proyección de la película en el cine con identificador *cinema_id*. El texto guardado tendrá el formato "HH:mm".

La clase parserCines.py

Función

Es la base de nuestro programa. `parserCines.py` es el nombre de la clase donde se descarga y organiza toda la información de los distintos cines de la provincia de Valencia.

Recoge información de más de 20 cines, que entre todos suelen proyectar más de 45 películas diarias, tanto por la mañana como por la tarde o noche.

En el funcionamiento del programa, lo primero es descargar la información en formato HTML de la web de donde se extrae toda la información. Para ello se hace uso de la función `call()` de la biblioteca *subprocess* [11] para ejecutar por consola la orden:

```
subprocess.call(shlex.split(
        'wget http://www.ecartelera.com/cines/0,48,0.html –o ./data/laCartelera.html -nv'))
```

Cabe destacar que se utiliza la función `split()` de la biblioteca *shlex* [16] para separar cada término de la orden dentro de un *array*, para que la función `call()` funcione correctamente.

La orden `wget` descarga de la web en formato HTML la dirección web que se le indique; las dos opciones adicionales que se indican en la orden significan:

-o [dir]: indica la ruta donde guardar el documento descargado y el nombre que se le quiera asignar. En nuestro caso, desde el directorio raíz, la carpeta `/data` y el nombre `laCartelera.html`

-nv: --no-verbose, para que no muestre toda la información de la descarga.

Razones de uso de www.ecartelera.com

La web es enorme, y en ella existen un sinfín de webs con la información que necesitábamos. Entonces, ¿por qué elegir www.ecartelera.com?

Revisé unas cuantas webs que contenían información acerca de los cines de la provincia de Valencia. Entre ellas se encuentra www.todoCine.com, donde se recoge toda la información que quería, pero además recoge información de cines de toda España. Para nuestro caso no nos hacía falta, por lo que descarté esa web por tener demasiada información y por existir otras webs con la información más centralizada en mi objetivo.

También revisé algunas webs de diarios como "Levante-EMV" o "Las Provincias" [5], que tenían carteleras propias actualizadas diaria o semanalmente y donde la información era adecuada; pero en comparativa de sencillez una candidata era la web de www.ecartelera.com.

www.ecartelera.com tenía información actualizada diariamente, y cuando está en mantenimiento de actualización de los horarios de los cines lo notifica, por lo que para nuestra aplicación (que muestra información solo del día actual) es perfecta. Además, tiene la información de los distintos cines estructurada en diferentes webs, por lo que permite descargar y guardar de forma sencilla.

Por tener toda la información de cines de la provincia de Valencia y ser una web en mantenimiento continuo diario, fue la elección adecuada para continuar adelante con el trabajo.

Métodos utilizados

internet_on(name)

Es un método creado con el fin de asegurarse que toda descarga se realiza de forma adecuada.

El método utiliza un archivo de texto llamado `Sync.txt` como guarda de la fecha de la última descarga hecha del archivo `name` que se pasa como parámetro. En caso de que en el archivo `Sync.txt` dicho nombre figure como actualizado, la descarga no se realizará, haciendo así más rápido el inicio del programa si ya se ha iniciado una vez y descargado los datos anteriormente.

En caso de no existir la entrada en el fichero con el campo `name`, o que la fecha indicada en ese fichero sea antigua, significa que la descarga debe realizarse, pero no sin antes comprobar que el ordenador que ejecuta el código esté conectado a internet. Si la comprobación de internet es exitosa, se guarda en el fichero `Sync.txt` una entrada con el nombre del archivo a descargar junto con la fecha del día en vigor, para no volver a realizar la descarga en caso de volver a ejecutar el programa. En caso de que el ordenador que ejecuta el programa no estuviese conectado a la web, se da a elegir al usuario entre dos opciones: conectarse a internet y seguir con el curso normal del programa, o proseguir con los datos antiguos del programa, asumiendo el riesgo de que los datos están desactualizados y las indicaciones que se van a mostrar serán de días anteriores.

actualizarArchivoSync()

En caso de ser necesario, ya sea porque no hay datos creados anteriormente o porque el día actual no es el mismo que el de la fecha de sincronización de los datos, este método actualiza la primera línea del archivo `Sync.txt` para mantener la fecha de sincronización actualizada en todo momento.

elimina_tildes(s)

Utiliza la biblioteca *unicodedata* [17] para normalizar en formato NFC (Forma Normal Compuesta) cada una de las palabras `s` que se le pasan como parámetro.

Es decir, elimina toda tilde, acento, diéresis, etc., de una palabra.

Ejecución de la clase

Una vez se hayan creado todas las carpetas donde guardar la información, la ejecución del programa prosigue de la siguiente manera:

1. Se descarga el archivo `laCartelera.html`.
2. Se extraen las direcciones web de las distintas webs de todos los cines de la provincia de Valencia de dicho documento, así como el nombre del cine y la ciudad donde éste se sitúa.
3. Se descarga, por cada web extraída del documento anterior, el HTML pertinente para obtener la información de ese cine.
4. Se extrae toda la información del cine, como dirección, precio de las entradas y, como no, toda la información referida a películas proyectadas por el cine.
5. Cine a cine, va guardando ordenadamente toda la información obtenida en una base de datos local.
6. Por último, rellena tres documentos, uno con los nombres de todos los cines que proyectan alguna película, otro con el nombre de todas las películas proyectadas en los distintos cines y un tercer documento con todas las horas de proyección posible para los distintos cines; estos documentos de guardan en una carpeta previamente creada llamada `QUERYS`, o la crea en caso de que no exista dicha carpeta. Estos documentos servirán posteriormente para el reconocimiento de texto de las consultas del usuario.

El estado de las carpetas y archivos después de la ejecución sin errores de `parseCines.py` será el mostrado en la Figura 2.

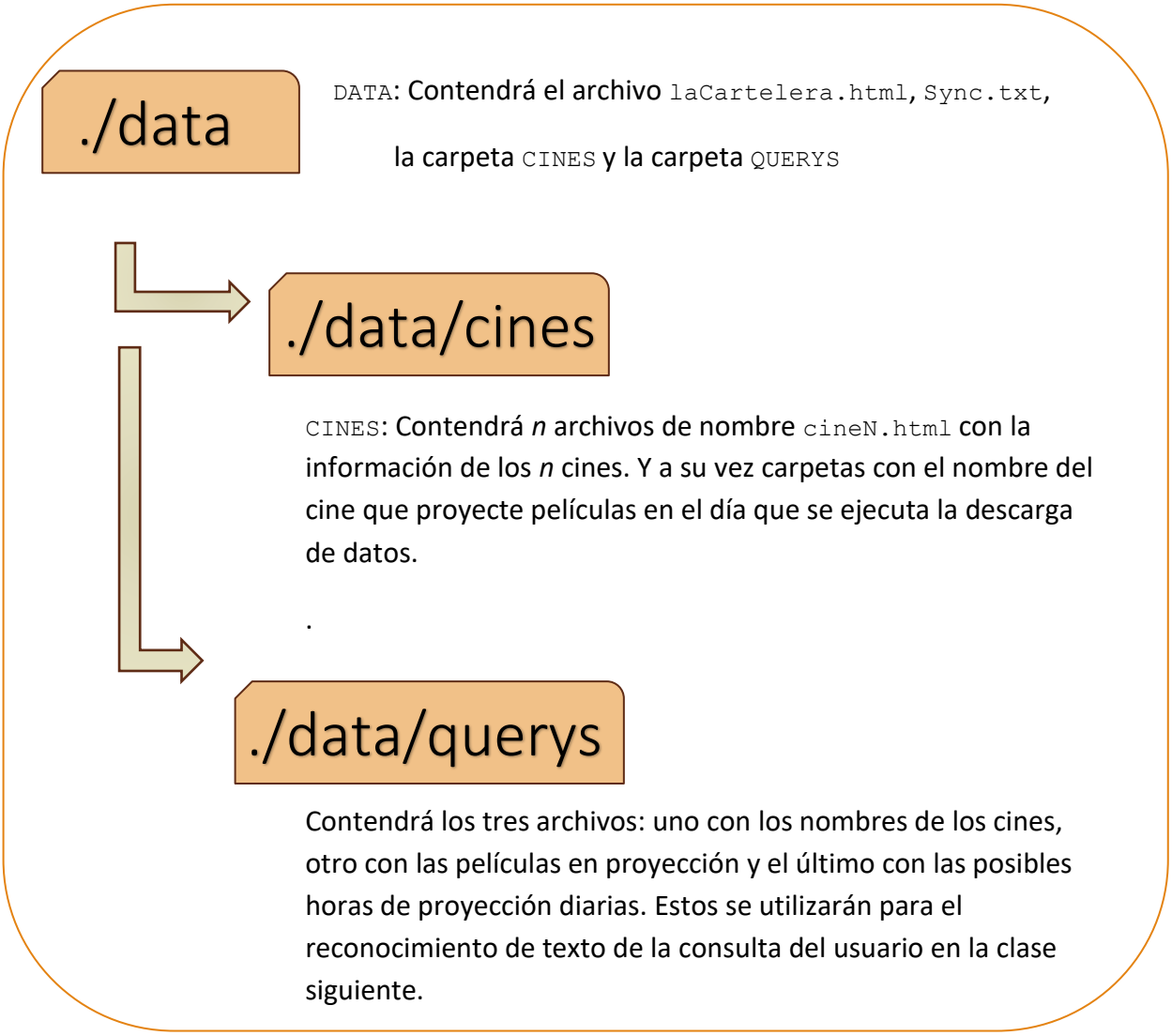

Figura 2: Almacenamiento de datos en disco.

La clase parseQuery.py

Función

Es la encargada de gestionar toda la interacción con el usuario. Gestiona las consultas del usuario y muestra los resultados dependiendo de la consulta realizada.

Además, es la clase que mantiene la interfaz gráfica activa para que la experiencia del usuario sea agradable.

Es una clase que funciona como reconocimiento de lenguaje natural, mediante consultas realizadas por el usuario en formato de texto. Como el sistema está reducido tan solo a la jerga cinematográfica, esta reducción llega al punto donde se pueden reconocer tres casos iniciales por los que el usuario puede preguntar: que el usuario referencie algún nombre de cine, que pregunte por alguna película o que su intención venga guiada por aproximación horaria o intervalo de horas de proyección. Por tanto, nuestro objetivo será identificar cuál es el caso por el que el usuario consulta.

Reconocimiento del lenguaje natural

Cómo saber sobre qué va dirigida la consulta del usuario es uno de los principales objetivos de este proyecto. Por tanto, tener una buena metodología para ser lo más preciso posible y abarcar el máximo número de consultas a identificar es crucial en este trabajo.

La metodología utilizada para el reconocimiento de lenguaje natural se centra en tres intenciones:

1. Un algoritmo que relaciona las palabras de la consulta con los distintos documentos creados anteriormente, es decir, con los nombres de los cines, títulos de películas y horas de proyección.
2. Una asignación de pesos por palabra por número de apariciones en los documentos para evitar palabras con menos importancia.
3. La eliminación de palabras de la consulta, basándose en el documento `menospreciables.txt` creado con experiencias reales de usuarios.

Algoritmo de similitud y asignación de pesos

En una consulta cualquiera, al estar tan relacionada con la jerga cinematográfica, debe aparecer uno de los tres casos iniciales que indicábamos anteriormente. Por tanto, esperamos el nombre de un cine, el título (o parte de él) de una película o alguna referencia a horario de proyección.

Por lo tanto, nuestro objetivo es calcular en dicha consulta cuál es el porcentaje de similitud mayor entre los nombres de cines almacenados, los títulos de las películas almacenadas o las horas posibles con los tres documentos creados anteriormente.

El algoritmo de similitud funciona de la siguiente manera:

1. Inicialmente se crean 3 variables que indicarán el máximo índice de similitud entre la consulta y alguna línea de cualquiera de los 3 documentos.
2. Para cada documento, encontrar el máximo grado de similitud en cualquiera de las líneas del documento, emparejando así la consulta y cada una de las líneas de los documentos.
3. Para eliminar algunas imperfecciones en el reconocimiento de lengua escrita, antes de hacer cualquier cálculo de similitud entre las dos palabras a comparar, se convierten las dos palabras a mayúsculas y se quitan todos los acentos y tildes que puedan hacer de la palabra una similitud menos adecuada.
4. Para cada línea del documento, la similitud con la consulta se realiza analizando palabra a palabra la consulta; calculando cuál es el máximo grado de similitud con todas las palabras de la línea del documento. Dicho grado se suma a un total que luego se divide por el número de palabras de la consulta inicial.
5. Al terminar de analizar los tres documentos con la consulta, el que mayor índice de similitud muestre respecto a los demás será la apuesta por la que nuestro programa seguirá adelante.

El algoritmo funcionaba, pero en ciertos casos, donde se incluían en la consulta conectores (como puede ser el conector "la") con muchas apariciones en algunas de las líneas de los documentos, hacían que se disparase el grado de similitud. Por tanto, quise dar un aporte más al algoritmo al cálculo de la frecuencia de aparición de un término en el documento.

Para hacer el cálculo de un IDF (*Inverse Document Frequency* = Frecuencia Inversa del Documento) [18] para un término solo se necesita contar el número de palabras de un documento y las veces que aparece una palabra en concreto en dicho documento. En la Tabla 7 se pueden ver las características principales del IDF.

Tabla 7: Características principales del Factor IDF.

Factor IDF [18]	
Denominación	*Inverse Document Frequency* = Frecuencia Inversa del Documento para un término
Descripción	Es el coeficiente que determina la capacidad discriminatoria del término de un documento con respecto a la colección. Es decir, distinguir la homogeneidad o heterogeneidad del documento a través de sus términos.
Finalidad	Discriminatoria
Casos	Poder discriminatorio bajo. El término es genérico y aparece en la mayoría de los documentos.
	Poder discriminatorio medio.
	Poder discriminatorio alto. El término es especializado y aparece en pocos documentos.

La fórmula del cálculo del peso para cada una de las diferentes palabras se calcula como se muestra en la Figura 3.

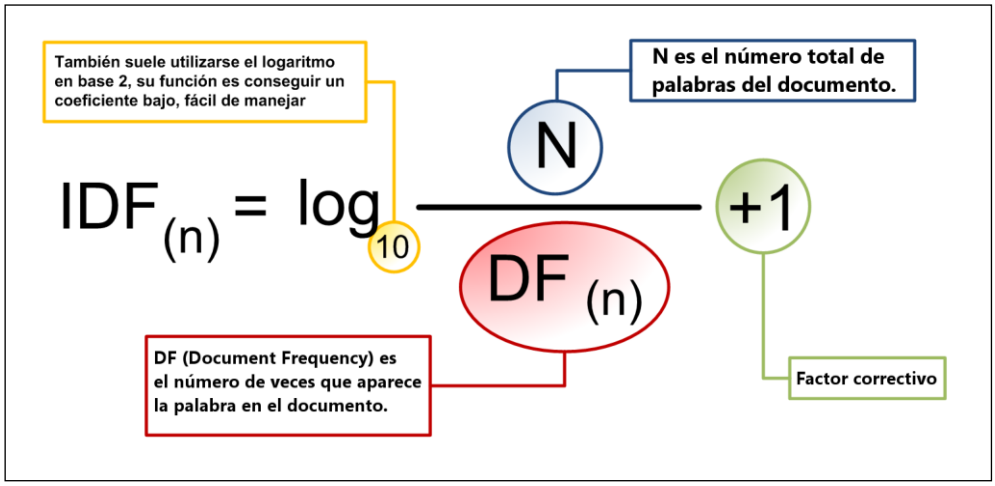

Figura 3: Descripción de los elementos de la fórmula de un IDF.

Entonces, para llevar el cálculo de los distintos pesos por palabra, al iniciar el programa se crean tres diccionarios: uno para el documento de cines, otro para el de películas y el último para las horas. En todos ellos aparecen cada una de las palabras de los documentos con el peso que se asigna a dicha palabra, dependiendo del número de apariciones de dicha palabra en los documentos.

El algoritmo de cálculo de pesos se vio en asignaturas del grado y es un algoritmo que soluciona gran parte del problema que se presentaba.

Una vez calculados los pesos para cada documento, tan solo quedaba, a la hora de asignar el grado de similitud de cada palabra, tener en cuenta su peso en el diccionario adecuado previamente creado. Así, si la palabra era importante (tiene pocas ocurrencias en el documento), al multiplicar el grado de similitud con el peso se daría más importancia a esa palabra que a otras que su peso fuese menor (tienen mayor número de ocurrencias en el conjunto de documentos).

Una vez realizada la asignación de similitudes junto el cálculo de los pesos IDF, el algoritmo mostraba un mayor porcentaje de aciertos. Aun así, a veces la longitud de la frase inicial, a comparar con un título de película, hacía que algunas palabras, por similitud, contasen más de la cuenta. Así pues, tras muchas pruebas con distintos usuarios, donde siempre se observaban las mismas palabras menospreciables, decidí eliminarlas de la consulta antes de realizar el cálculo del peso de similitud.

Por tanto, se creó un archivo llamado `menospreciables.txt` con la finalidad de almacenar todas las palabras que son innecesarias para el cálculo de similitud con los nombres de los cines, los títulos de las películas o las horas de proyección; previamente a realizar dicho cálculo se eliminan las palabras de la consulta que coinciden con alguna de las palabras del documento `menospreciables.txt`.

Los resultados, una vez se eliminaban las palabras adecuadas, conseguían un porcentaje elevado de aciertos respecto a la selección del nodo inicial por el que el usuario consultaba.

Intervalos de tiempo

En un caso de consulta normal no se suele indicar la hora exacta de la película que quieres ver. Los usuarios suelen mostrar interés por intervalos de tiempo como puede ser "sobre las 22" o simplemente "por la noche".

Por tanto, buscar en la base de datos local solo la información de una hora en concreto limitaba mucho las consultas del usuario. Por este motivo se amplió el proyecto para incluir dichos intervalos en el cálculo y muestreo de información hacia el usuario.

Siempre y cuando la similitud máxima no hiciese referencia a una hora en concreto no solo se muestran las películas proyectadas a esa hora, sino que se tiene en cuenta un rango de quince minutos antes y cuarenta y cinco minutos después.

De la misma manera, al incluir la posibilidad de indicar los intervalos (mañana, tarde y noche), se asignan los intervalos de horas de la siguiente manera:

- Mañana: De 11:00h a 13:00h, siendo así porque no hay nunca películas antes de las 11:00h ni más tarde de las 13:00h proyectadas en cualquiera de los cines.
- Tarde: de 15:00h a 21:00h.
- Noche: de 21:00h a 01:00h; cabe destacar que las 21:00h está elegida como hora máxima de representación de tarde y principio del intervalo de la noche por ser la hora en la que puedes llegar a cenar después de ver cualquier película, pero es más inusual cenar antes de ella; por tanto, para lo que la mayoría de personas entiende como normal, las 21:00h es una hora que separa la tarde y la noche de manera adecuada.

Gestión del diálogo e interacción con el usuario

La interacción con el usuario en forma de diálogo sigue un modelo basado en un autómata de estados finitos, donde cada estado indica toda la información ya recopilada sobre la consulta del usuario. Nuestro objetivo sería hacer llegar al usuario al estado final, mientras recopilamos la información necesaria para hacer el muestreo final de toda la información que se haya solicitado durante el proceso.

La parte inicial del autómata (las transiciones desde el estado inicial) se muestran en la Figura 4.

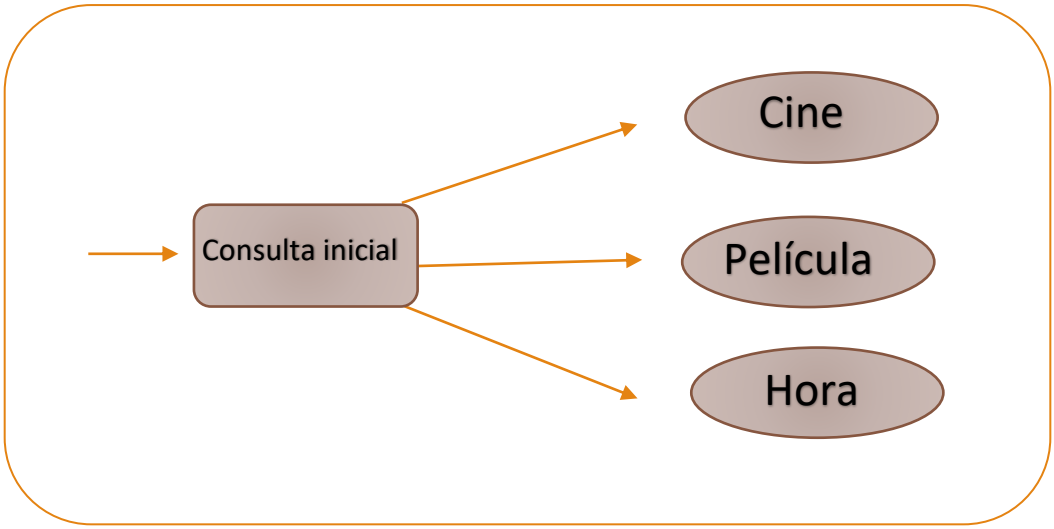

Figura 4: Grafo de los nodos iniciales del autómata del sistema de diálogo.

Una vez avanzado desde el nodo inicial a uno de los tres primeros nodos, se hace una consulta al usuario que referencie la información que nos falta, y se repite el proceso de reconocimiento de la consulta, para saber esta vez sobre cualquiera de las dos informaciones que faltan cuál es la que nos ha indicado. Esta estructura se puede ver en la Figura 5.

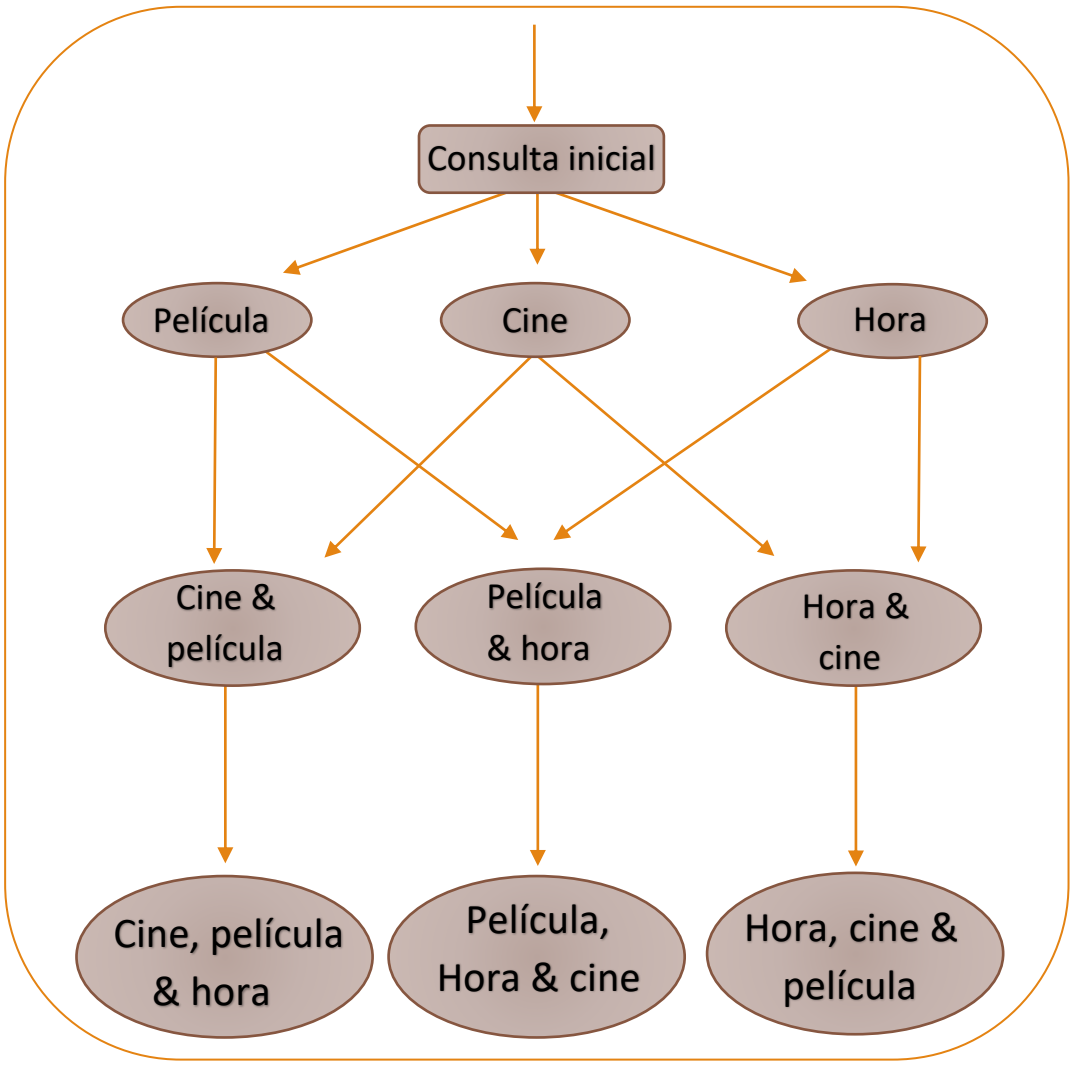

Figura 5: Grafo de los nodos completo del autómata del sistema de diálogo.

La conclusión final se centra en un muestreo completo de la información dependiendo de las consultas que el usuario ha ido indicando a lo largo del proceso.

Interfaz gráfica

La clase mantiene en bucle una interfaz gráfica implementada en `tkinter` [18], simple pero que mejora la visualización de datos masivos con respecto a la terminal del sistema.

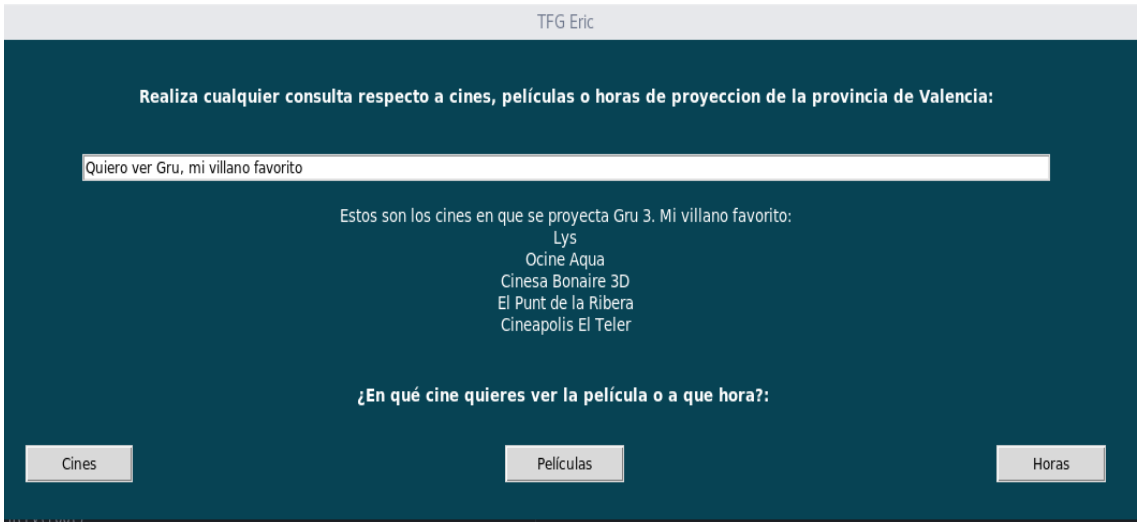

Figura 6: Interfaz gráfica del proyecto después de realizar una consulta.

La interfaz gráfica, visible en la Figura 6, está compuesta por un título que indica al usuario dónde puede realizar la consulta, seguida de un elemento de entrada de texto para que el mismo usuario pueda escribir la consulta ahí. Cabe destacar que la ejecución de todos los métodos que extraen información de la consulta se realiza en el mismo instante en el que el usuario pulsa la tecla `ENTER` como indicación de fin de su consulta.

Seguido del campo de entrada de texto, un gran campo de texto que se expande y contrae dependiendo de la cantidad de líneas que se muestren en él, hace de base para escribir toda la información por la que el usuario ha preguntado. Bajo él aparece otro campo de texto (que muestra el texto en negrita para resaltar la frase escrita), donde se escribirá la información más importante, como la siguiente pregunta que el programa hace al usuario o el resultado final buscado.

La interfaz gráfica incluye 3 botones adicionales en la parte inferior, con el nombre de "Cines", "Películas" y "Horas", que muestran, respectivamente, todos los cines disponibles separados por ciudades, todas las películas proyectadas en toda la provincia de Valencia en orden alfabético, y todas las horas de proyección disponibles de las películas. Estos botones son muy útiles por si el usuario no sabe qué cines hay en su ciudad o qué películas se están proyectando a día de hoy, así como el intervalo de horas de proyección de los cines de nuestra provincia de Valencia.

El color elegido para el fondo de la interfaz gráfica es un azul oscuro. Este color indica tranquilidad y bienestar a la vez que seguridad, y junto con los botones en blanco, la línea de entrada de texto blanca y que todo el muestreo de información se realiza con letra blanca, hace que la combinación de colores sea adecuada.

Ejecución del programa

Para iniciar el programa y todas sus funcionalidades solo bastaría con estar dentro de la carpeta donde residen todos los archivos y ejecutar en consola la orden:

```
python main.py
```

4. Validación

Cuando realizas una interacción con el usuario, este mismo te sorprende cada vez que interactúas con él. Por esta razón, decidí que la mejor manera de probar que todo funciona y realimentar el programa para un mejor funcionamiento, era probarlo con gente real con preguntas que ellos mismos realizarían sin ninguna información previa del funcionamiento y límites de la aplicación.

Así que, en mi círculo de amigos y familiares cercanos, realicé pruebas poco a poco uno por uno. Y así mismo, mediante un formulario de *Google* [19], cada uno evaluaba la experiencia personal respecto a los resultados que el programa mostraba.

Los resultados fueron favorables, ya que en ciertos casos me sirvió para encontrar errores que no tuve en cuenta, por la gran cantidad de estados que presenta el proyecto; otras me sirvieron para añadir palabras menospreciables, de las consultas que hacían, al documento de `menospreciables.txt` y otras veces solo me sirvió para validar que todo funcionaba correctamente.

5. Resultados y conclusiones

Los resultados, conforme a las valoraciones de los usuarios que prueban la aplicación, son favorables, por lo que supone unos resultados buenos.

Los resultados obtenidos se muestran a continuación en la tabla 8.

Tabla 8: Resultados obtenidos en la encuesta, a los usuarios que prueban el programa.

El programa era intuitivo y el contacto visual era adecuado									
1	0%	2	0%	3	11,1%	4	33,3%	5	55,6%
Las respuestas del programa se corresponden con mis preguntas									
1	0%	2	0%	3	0%	4	44,4%	5	55,6%
Ha sido útil el programa para mi									
1	0%	2	0%	3	0%	4	22,2%	5	77,8%

Propuestas de mejoras y ampliaciones

En cambio, todo programa puede mejorar y ampliarse de muchas maneras, por las que, en este proyecto, las mejoras y ampliaciones que yo propondría en visión de futuro serían:

- Guarda de varios días. Tan solo se hace la descarga de películas y cines con fecha del día en que se hace la ejecución, por lo que una descarga masiva en visión de futuro a una semana haría el programa más funcional.
- Mejora de la interfaz. La interfaz mejora la visión de terminal básico de Linux, pero es una interfaz básica con elementos simples de muestreo de la información. Una mejora de la misma sería una buena ampliación de futuro.
- Ampliación de estados iniciales de consulta. En este proyecto se centran tres grandes estados iniciales por los que el usuario puede preguntar; estos estados podrían ser ampliados a más incluyendo nuevos filtros de búsqueda, como genero de películas, duración de la misma o precio del cine.
- Reconocimiento de más de un estado inicialmente. Es decir, que se pudiese preguntar desde el principio por más de un campo de búsqueda para simplificar así el número de consultas a realizar por el usuario.
- Hacer del proyecto una aplicación portable, para que sea más fácil su utilización en distintos dispositivos.

6. Referencias y bibliografía

1. Universidad Politécnica de Valencia

<https://www.upv.es/entidades/ETSINF/info/934549normalc.html>

[Última consulta: 30 de junio de 2017]

2. Portal de Trabajos de Fin de Grado

<https://riunet.upv.es/>

[Última consulta: 30 de junio de 2017]

3. Lenguaje de programación

https://www.Python.org/

4. Herramienta de Trabajo (Atom):

<https://atom.io/>

[Última consulta: 30 de junio de 2017]

5. Webs de distintos cines:

<www.todocine.com >

[Última consulta: 30 de junio de 2017]

<http://ocio.levante-emv.com/cine/cartelera/valencia/>

[Última consulta: 30 de junio de 2017]

<http://www.ecartelera.com/cines/0,48,0.html>

[Última consulta: 30 de junio de 2017]

6. Información interfaz gráfica(tkinter)

<http://tkinter.unPythonic.net/wiki/>

[Última consulta: 30 de junio de 2017]

<https://wiki.Python.org/moin/TkInter>

[Última consulta: 30 de junio de 2017]

7. Comparativa de distintas interfaces gráficas

<http://www.Python.org.ar/wiki/InterfacesGraficas>

[Última consulta: 30 de junio de 2017]

8. Librerías

<https://docs.Python.org/2/library>

[Última consulta: 30 de junio de 2017]

9. OS

https://docs.Python.org/2/library/os.html>

[Última consulta: 30 de junio de 2017]

10. math

https://docs.Python.org/2/library/math.html>

[Última consulta: 30 de junio de 2017]

11. subprocess

https://docs.Python.org/2/library/subprocess.html#module-subprocess>

[Última consulta: 30 de junio de 2017]

12. difflib

https://docs.Python.org/2/library/difflib.html>

[Última consulta: 30 de junio de 2017]

13. time

https://docs.Python.org/2/library/time.html>

[Última consulta: 30 de junio de 2017]

14. datetime

https://docs.Python.org/2/library/datetime.html>

[Última consulta: 30 de junio de 2017]

15. SQlite3

https://docs.Python.org/2/library/sqlite3.html>

[Última consulta: 30 de junio de 2017]

16. shlex

https://docs.python.org/2/library/shlex.html

[Última consulta: 4 de julio de 2017]

17. unicodedata

https://docs.python.org/2/library/unicodedata.html

[Última consulta: 4 de julio de 2017]

18. Información IDF

http://ccdoc-tecnicasrecuperacioninformacion.blogspot.com.es/2012/11/frecuencias-y-pesos-de-los-terminos-de.html

[Última consulta: 3 de julio de 2017]

19. Formulario de Google

https://goo.gl/forms/poSPdCLz2EdGW6B12

[Última consulta: 04 de julio de 2017]