Trabajo de Fin de Grado

Grado en Ingeniería Informática

Proyecto de detección y análisis de caracteres en imágenes (OCR): <MCard Analyzer (OCR - Application)>

Aduanich Rodríguez Rodríguez

La Laguna, 30 de enero de 2019

D. **JOSÉ LUIS, GONZÁLEZ ÁVILA**, con N.I.F. 78.677.390 - W profesor Asociado de la Universidad de La Laguna, adscrito al Departamento de Ingeniería Informática y de Sistemas, como tutor:

CERTIFICA:

Que la presente memoria titulada:

Proyecto de detección y análisis de caracteres en imágenes (OCR): <MCard Analyzer (OCR - Application)>

Ha sido realizada bajo su dirección por:

D. **ADUANICH RODRÍGUEZ RODRÍGUEZ**, con N.I.F. 54.061.584 - S

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 30 de enero de 2019.

Resumen

El objetivo principal de este trabajo ha sido trabajar con una librería que permita implementar un sistema de detección de caracteres en imágenes (Optical Character Recognition). Debido a lo poco demostrativo y claro de los posibles resultados, se ha decidido implementar una aplicación que haga uso de dicho sistema OCR permitiendo así a su vez dar un posible ejemplo de uso para los resultados de este proyecto y aumentando a su vez el alcance del mismo.

Sabiendo esto, el tema sobre el que se trabajará con el OCR será el juego de mesa: "Magic The Gathering" (MTG). Se trata de un juego de cartas que gracias a la forma en la que está creado y a la manera poco ortodoxa de intercambiar dichas cartas con otros jugadores, ha generado un mercado de compra-venta que cambia constantemente. Tras un pequeño estudio y una serie de búsquedas en los foros de dicho juego, se ha logrado deducir cuáles son las características necesarias para que dicha aplicación logre cumplir con las expectativas de los usuarios del juego cubriendo ciertos campos que otras a día de hoy aún no han trabajado.

Se creará así y como ya se citó, una aplicación multiplataforma escrita en Java que nos permita analizar el texto en las imágenes de las cartas de dicho juego (usando el OCR previamente entrenado y almacenado en un servidor Linux al que se accederá para su uso) y devolver los resultados de mercado de una de las páginas europeas más importantes de compra-venta de las susodichas (www.cardmarket.com). Esta permitirá además al usuario, almacenar en una base de datos (Apache Derby), todo el contenido de su colección con los detalles obtenidos en el momento de búsqueda de cada una de las que haya decidido analizar, así como si la tiene reservada para comerciar en un futuro o si está haciendo uso de ella. Se busca pues analizar el nombre de estas y generar plantillas con datos recolectados a través del OCR y de las fuentes de internet, para hacer un almacenamiento de dichos modelos generados, en función de los parámetros indicados por el usuario que haga uso de la aplicación.

Palabras Clave: OCR, multiplataforma, JAVA, base de datos, tratamiento de información, MTG, precios de mercado.

Abstract

The main objective of this project is to work with a library that allow to implement an OCR system (Optical Character Recognition). Due to the unclear possible demonstration of the results we can get with this, it has been decided to make an application thanks to which we can see a better use of the OCR and at the same time allow us to increase the extent of the work field.

Then, first of all we need to know the working topic for the OCR, this will be the board game: "Magic The Gathering" (MTG). Being it a card game with a lot of complex mechanics and a weird way to exchange the cards, it has developed his own market that changes constantly. After a bit of looking-for and a few of searching into the forums of the game, it has been deducted which are the characteristics required by the application to fulfill the needs by the players of the game, covering fields that nowadays no other application has even think of.

It'll be created therefore and as commented, a Java multi-platform application that allow us to read the text printed into the aforementioned cards (by using the OCR previously trained and stored in a Linux server used then when we need it to) and return all the market results from the most important european buy-sell internet page (www.cardmarket.com). This will also allow to the user to gather in a Apache Derby Database all the information about his collection with all the data that he got at the moment of searching each one as well as if he have anyone of those reserved to exchange it on the future or if he is using it on one of his decks.

Summarizing: we seek to analyze the name of cards and generate templates with data collected by the OCR and the internet sources, to make a storage of these generated models according to the parameters indicated by the one that makes use of the application.

Key Words: OCR, multi-platform, JAVA, database, data manipulation, MTG, market prices.

Índice general

Índice de figuras

El primer número del índice de una imagen o tabla indica el capítulo donde aparece y el segundo, el orden de aparición (Entre paréntesis se indica la página de aparición)

Índice de tablas

Introducción

Breve introducción a los objetivos

Como ya se comentó brevemente en el sumario/resumen, el objetivo de este proyecto va a ser trabajar con un sistema de detección de caracteres que permite como su nombre dice, detectar caracteres en imágenes. Se sabe que el resultado del entrenamiento de esta red neuronal, que posteriormente se explicará con mayor detalle, devuelve como resultado una cadena de caracteres en lenguaje natural que representa aquello detectado por el sistema. Se trata de un resultado poco demostrativo puesto que las conclusiones nos permitirían básica y únicamente deducir cuál es el texto que se encuentra en las imágenes analizadas.

Sabiendo esto, se ha decidido ampliar el alcance del proyecto. Se alojará para ello el OCR entrenado en un servidor Linux del que se hará uso y a su vez de una aplicación multiplataforma que conectará por SFTP y SSH al mismo para obtener resultados de análisis sobre imágenes que tienen un estereotipo similar a las ya entrenadas. Este parecido entre las imágenes a analizar es necesario puesto que el entrenamiento del OCR se realiza sobre una serie de caracteres que poseen un determinado formato y tamaño de letra lo que limita bastante la posible variedad entre las imágenes a utilizar. Aun así y como ya se verá más adelante, aunque se trabaje con imágenes similar, el entrenamiento intensivo del OCR ha devuelto unos resultados bastante prometedores para caracteres que incluso poseen un formato diferente al ya entrenado, pudiéndose así hacer uso (si se continuase entrenado hasta terminar de perfeccionar) de las bases de entrenamiento ya obtenidas para imágenes que cubren otro tipo de campos y de formatos.

Por último y con la información ya predispuesta, se generarán plantillas que se utilizarán como modelo para la aplicación y se creará una base de datos local (empleando Apache Derby DB) que nos permita almacenar todas las que se vayan generando durante el uso normal y corriente de la aplicación arriba nombrada.

Temática del trabajo

Se necesita pues una temática para trabajar con el OCR. Se ha elegido para este trabajo, el juego de mesa de cartas: “Magic The Gathering” (MTG para abreviar).

Se añade con ello la precisión sobre el campo y el tipo de imágenes que se van a utilizar para entrenar el sistema de detección. Se emplearán las susodichas cartas del juego y se analizarán sus campos para lograr obtener una plantilla con toda la posible información de la carta. Dicho esto, y entrando un poco más en detalle, las imágenes que se usarán tienen el siguiente formato:

Remarcado en rojo tenemos las áreas más importantes de las cartas:

- Comenzamos por el nombre de la carta (remarcado con un (1) sobre la imagen 1.1) es probablemente el campo de mayor importancia a analizar. Es un atributo único y que permite distinguir de forma indiscutible una carta de otra.
- El símbolo que indica la expansión de la carta (remarcado con un (2) sobre la imagen 1.1). Indica el conjunto de cartas que salieron a la venta en un momento concreto formando un “set”.
- El contenido de la carta (remarcado con un (3) sobre la imagen 1.1). Indica su forma de uso, es decir, las capacidades de juego de la carta. Este punto, aunque no sea determinante para el OCR, resulta sin duda un factor a tener en cuenta para el mercado de compra-venta del juego puesto que gracias al potencial de uso de la misma se determina el precio que se le pone a la carta (y que cambia constantemente) llegando a veces a precios que para personas no asiduas al juego resultan desorbitados.
- La rareza de la carta (remarcado con un (4) sobre la imagen 1.1). Se distinguen entre común (C), poco común (U), rara (R), y mítica (M) en función de la facilidad con la que se encuentran en los sobres en venta.

Objetivos

Campos de trabajo del OCR

Teniendo en cuenta la información sobre las imágenes a analizar, explicadas en el apartado anterior, se puede deducir un orden de prioridades entre los campos de texto que se encuentran en las cartas del juego en cuestión. Así, se genera la siguiente disposición:

Orden	Campo
1	Nombre
2	Expansión
3	Rareza
4	Contenido

Nuestra prioridad como se puede observar a simple vista es el nombre. Este será pues el objetivo a analizar para el OCR. Con él se podrá conseguir mucha información de la carta, incluso los otros campos que no se lleguen a analizar con el sistema de detección, y nos permitirá además, hacer una búsqueda por los mercados de compra-venta para encontrar los precios de las cartas analizadas.

Enlazando con lo recién dicho, pasaremos a comentar los campos a buscar en los mercados de cartas. Para el proyecto, se usará como ejemplo la página www.cardmarket.com y se empleará su sistema de organización para obtener los precios que se tomarán como referencia para completar la plantilla con la información que se pretende generar. Basándonos por tanto en su estructura conseguiremos los siguientes parámetros:

- Precio mínimo. Se trata del valor mínimo que se paga por una carta.
- Tendencia de precio. Promedio generado entre las compras y el valor establecido para todas las cartas a la venta. Suele ser el valor de referencia usado comúnmente por los jugadores.
- Precio mínimo “foil”. Es el valor mínimo que se paga por la misma carta pero en una versión especial (brillante).

Todos estos valores junto a extraídos de las lecturas de las cartas, nos permite ya intuir cómo van a ser las plantillas que se usarán durante desarrollo del proyecto.

Así pues, antes de hablar de su estructura y desarrollo, toca comentar los antecedentes referentes al juego.

Antecedentes

Antes de comenzar a hablar sobre el estado en el que se encuentra la comunidad en relación al juego de mesa con el que tratamos, es necesario hablar de otras investigaciones por parte de otros proyectos presentados en años anteriores para poder así comparar los campos trabajados.

Existe un proyecto de TFG presentado el año pasado que se sustenta en la idea de crear un aparato físico que permite ordenar y clasificar las cartas según sus características. Es necesario por tanto comentar, que este proyecto también posee una porción que cumple con un objetivo similar a ese. Se pretende pues, al igual que el trabajo ya presentado, identificar las cartas para trabajar con ellas y poder recopilarlas en nuestras plantillas particulares.

Sin embargo, resulta imprescindible resaltar el hecho de que nuestro objetivo difiere del de ese proyecto. Para el desarrollo de este TFG se empleará un sistema de detección de caracteres diferente al de Tesseract de Google (más adelante se comentará con más detalle) y nos permitirá alojarlo en un servidor para que las aplicaciones que estén preparadas para ello (aquella desarrollada en este trabajo es un ejemplo) puedan acceder y hacer uso de esta cuando así se considere pertinente. Esto dará una mayor libertad a otros posibles proyectos de trabajar con un sistema de análisis no dependiente con tan solo conectarse a dicho servidor. También, se desarrollará la ya nombrada aplicación con el objetivo de trabajar con los precios de mercado de las cartas analizadas para llevar un registro que tiende al carácter económico además del meramente analítico que por defecto se presenta en el tema con el que se trabaja. Y es que aunque para este proyecto, sólo se empleen los resultados de una web en concreto, con tiempo y un enfoque adecuado, resulta imposible descartar la idea de hacer lo mismo con otras bases de mercados y lograr algún tipo de aplicación que haga uso de lo ya implementado para mejorar los resultados que se obtendrán en este TFG.

Otro punto del que es necesario hablar es que como ya se dijo al con anterioridad, el proyecto implementa una base de datos local (en Apache Derby) que permitirá al usuario guardar su colección de una forma más cómoda a la habitual, teniendo siempre a mano una lista de las cartas que reserve para otros jugadores así como aquellas que esté usando actualmente. Se trata de un añadido de importancia a la aplicación que se va a desarrollar en este proyecto puesto que es una característica que tras el breve estudio que se comentó al principio en el informe, se logró discernir como imprescindible al no existir una página o programa de referencia que cubra con eficacia este campo.

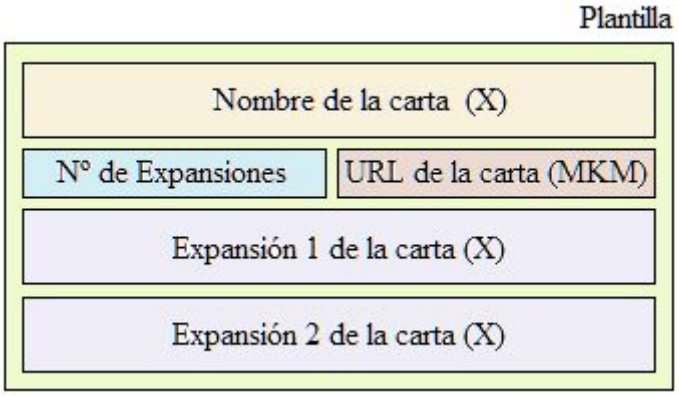
Esta última adición nos permitirá así introducir un poco el estado actual de la comunidad del juego así como el de las tecnologías usadas diariamente. Actualmente no existe ninguna aplicación de escritorio de uso conocido, así que todo lo disponible para los usuarios son páginas web con alguna temática definida y fácilmente identificable. Basándonos así en su enfoque se pueden distinguir y clasificar en tres tipos:

- Páginas y/o aplicaciones que permiten generar una lista con el mazo que el usuario desee comprarse en un futuro (Un ejemplo muy claro es https://deckstats.net/).
- Páginas y/o aplicaciones que permiten consultar información así como todas las normativas referentes a una carta (Un ejemplo sería https://scryfall.com/).
- Páginas de compra-venta (como la usada en este proyecto - www.cardmarket.com) en la que los usuario adquieren aquellas que deseen.

En lo relacionado a aplicaciones móviles sólo existe una relacionada con el mercado americano llamada TCGMarket. Salió al público en agosto de 2017 y presentaba una serie de problemas en lo relacionado a la detección de cartas y en la conexión con sus servidores de mercado que a día de hoy han logrado parcialmente resolver. Aun así su uso sigue sin estar demasiado extendido puesto que al trabajar con la economía americana (que difiere de forma severa de la europea) no sirve ni siquiera como programa a tener de ejemplo para nuestro proyecto.

Estructura y organización de los datos (Plantillas)

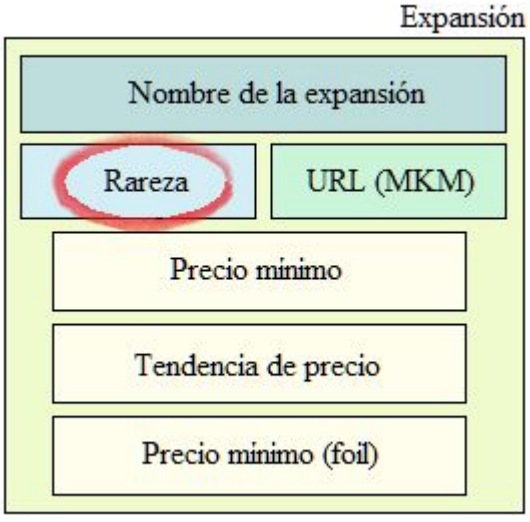
Ya se ha visto gran parte de la información referente al juego así como los antecedentes del mismo. Teniendo esto en cuenta, toca pasar a ver la configuración de las plantillas que se usarán. El esquema explicativo es el siguiente:



(La URL de la carta es necesaria puesto que como ya se verá más adelante, la información se extrae mediante "web scraping")

Dentro de cada plantilla, se puede observar el nombre de la carta que es el parámetro principal a analizar por el OCR y un vector o "array" de bloques correspondientes a las expansiones en las que se ha impreso dicha carta. El tamaño de dicho vector viene definido por el número almacenado en la variable "número de expansiones". Además, como se acaba de comentar, se trata de un bloque, lo que nos lleva a que se trata de otra estructura contenedora que tiene a su vez dentro la siguiente información:

Es importante recordar que la rareza no es genérica para una carta con nombre concreto. Debido a políticas de la empresa que desarrolla el juego, una misma carta puede tener dos rarezas diferentes en dos expansiones diferentes si su utilidad desvaría demasiado con su frecuencia en los sobres que se ponen a la venta. Así, la variable correspondiente a la rareza (marcada en rojo) es dependiente del "set" en el que se ha impreso.



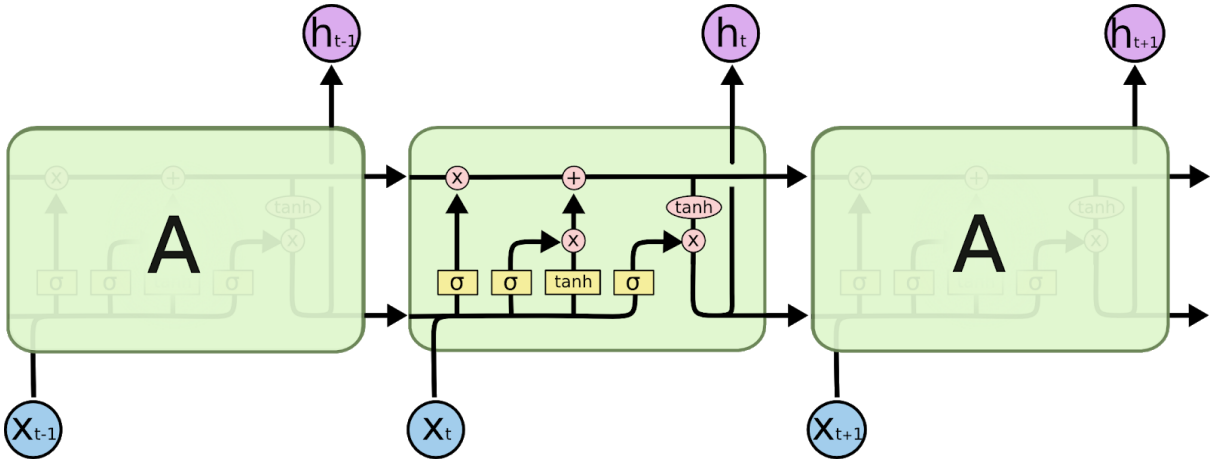
Desarrollo del proyecto - OCR

Librería OCR usada

Para el desarrollo de este proyecto, se ha elegido como la librería Ocropus/Ocropy OCR. Escrita en Python, trabaja con redes neuronales de tipo LSTM sin entrenar. Estas están especialmente diseñadas para trabajar con elementos menores que varían drásticamente o que pueden tener una cantidad bastante elevada de combinaciones entre ellos. Es por tanto perfecta para trabajar con lenguaje natural puesto que permite abarcar con simpleza una combinación de letras considerablemente variada.

Funcionamiento de la librería OCR

Expliquemos con un mayor detalle el funcionamiento de la red neuronal LSTM. Para ello, se usará el siguiente esquema:

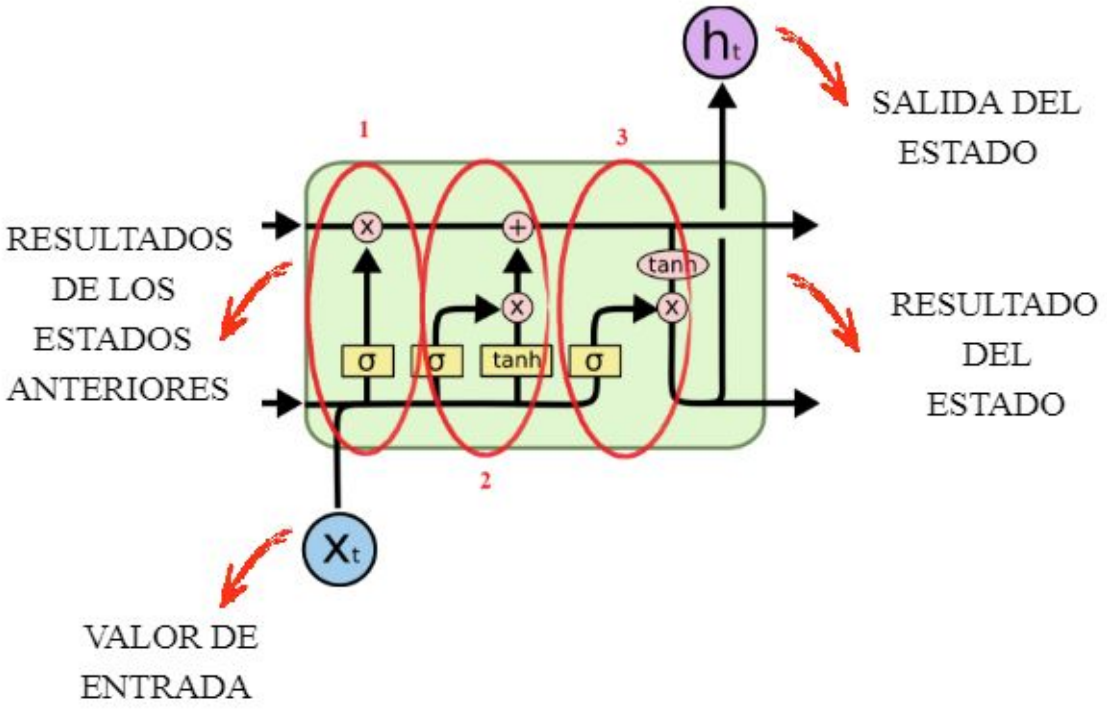


Red Neuronal LSTM

Como se puede observar en la imagen superior, se muestran las relaciones entre cada estado. Estos se ven representados por un cuadrado (pintado de verde) que almacena durante cierto tiempo y devuelve como salida el valor que resulte de las operaciones internas especificadas para el mismo usando como parámetros dos tipos de datos:

- Por un lado el valor de entrada (en nuestro caso las letras de la cadena separadas y divididas en celdas individuales - Xt).
- Por el otro el resultado devuelto por el estado anterior (Ht).

Para entender con mayor facilidad el funcionamiento interno de un estado lo dividiremos en tres secciones operativas que realizan una función concreta de la siguiente forma:



Dónde:

1. Lo primero es distinguir las variaciones las variaciones entre el estado actual y el anterior. Estas se determinan en función de los parámetros comentados arriba.
2. Acto seguido se especifica cuáles son los elementos de los que se han recibido que varían de la forma elegida al principio.
3. Para acabar se realizan los susodichos cambios.

Así, las transformaciones realizadas en la zona o sección (3) de un estado se transmite al siguiente además de devolverse como resultado de las operaciones del mismo. Logramos con ello, encadenar en nuestro problema dado la salida de todos los estados para generar un texto en lenguaje natural.

¿Por qué se eligió Ocropus/Ocropy sobre otros OCR?

Antes de comenzar a explicar el por qué se ha elegido Ocropus/Ocropy como OCR a usar para el proyecto, es necesario comentar que existe otro OCR desarrollado por Google llamado Tesseract-OCR que junto a la que se usa en este trabajo, son las dos librerías que abarcan todo el campo de las mismas para el reconocimiento de caracteres.

Sabiendo que ambas son de licencia libre (todo el mundo puede hacer uso de ellas cuando desee) y ambas están distribuidas bajo una licencia Apache 2.0, entonces ¿Por qué se eligió Ocropus y no Tesseract?

Existen varios motivos por los cuales la toma de esta decisión pero solo dos merecen ser destacados:

1. Ocropus/Ocropy es más simple de almacenar en un servidor como el que se busca para este proyecto. Solo requiere de una instalación con pocos comandos lo que lo hace fácil de manejar.
2. Durante la fase de estudio y desarrollo, se probaron ambos sistemas OCR empezando por el Tesseract y debido a una serie problemas de procesamiento y ejecución que presentó durante su uso, resultó imposible trabajar con este. Esto problemas se presentan en las versiones más recientes de la librería y no fueron corregidos hasta pasado un tiempo en el desarrollo de este proyecto.

Fase de desarrollo del OCR.

Antes de comenzar a explicar cómo se hace uso del OCR y cuáles son los comandos a usar es necesario indicar que requisitos previos se ha de tener para poder hacer uso del sistema en cuestión. Al tratarse de una librería escrita en Python lo primero que será necesario instalar será la última versión de este de la siguiente forma:

```
$> sudo apt-get install python2.7
$> python --version (Para comprobar la versión instalada)
```

Una vez hecho esto, se necesitará preparar también los módulos que el OCR necesita para realizar operaciones matemáticas, cálculos varios y estructuras de datos generalmente compuestas entre otras. Para ello, se hará uso del instalador de módulos y paquetes "Pip" de Python. Se instala con el siguiente comando:

```
$> python get-pip.py
```

Gracias a esta herramienta podremos preparar todos los módulos que han sido nombrados. Estos se encuentran listados a continuación:

Módulo	Comando
Numpy	*$> sudo pip install numpy*
Plt	*$> sudo pip install plt*
Matplotlib	*$> sudo pip install matplotlib*
Scipy	*$> sudo pip install scipy*
Lxml	*$> sudo pip install lxml*

Acabado esto se puede pasar a instalar la librería en cuestión. Es necesario comentar que una vez descargada con el "wget" (herramienta libre que permite la descarga de contenidos desde servidores web) se moverán los modelos a la carpeta que servirá de almacén durante su uso (*). Esto también se aplica durante la preparación del servidor como se verá más adelante. Dicho esto, su preparación consta de los siguientes tres comandos:

```
$> sudo wget -nd http://www.tmbdev.net/en-default.pyrnn.gz
$> mv en-default.pyrnn.gz /usr/local/share/ocropus/ (*)
$> sudo python setup.py install
```

Para obtener una mejor calidad en la imagen que vamos a analizar, se va a trabajar con ella previamente. Aquí entra en juego la última instalación que se realizará para el OCR. Se emplearán las herramientas que nos proporciona

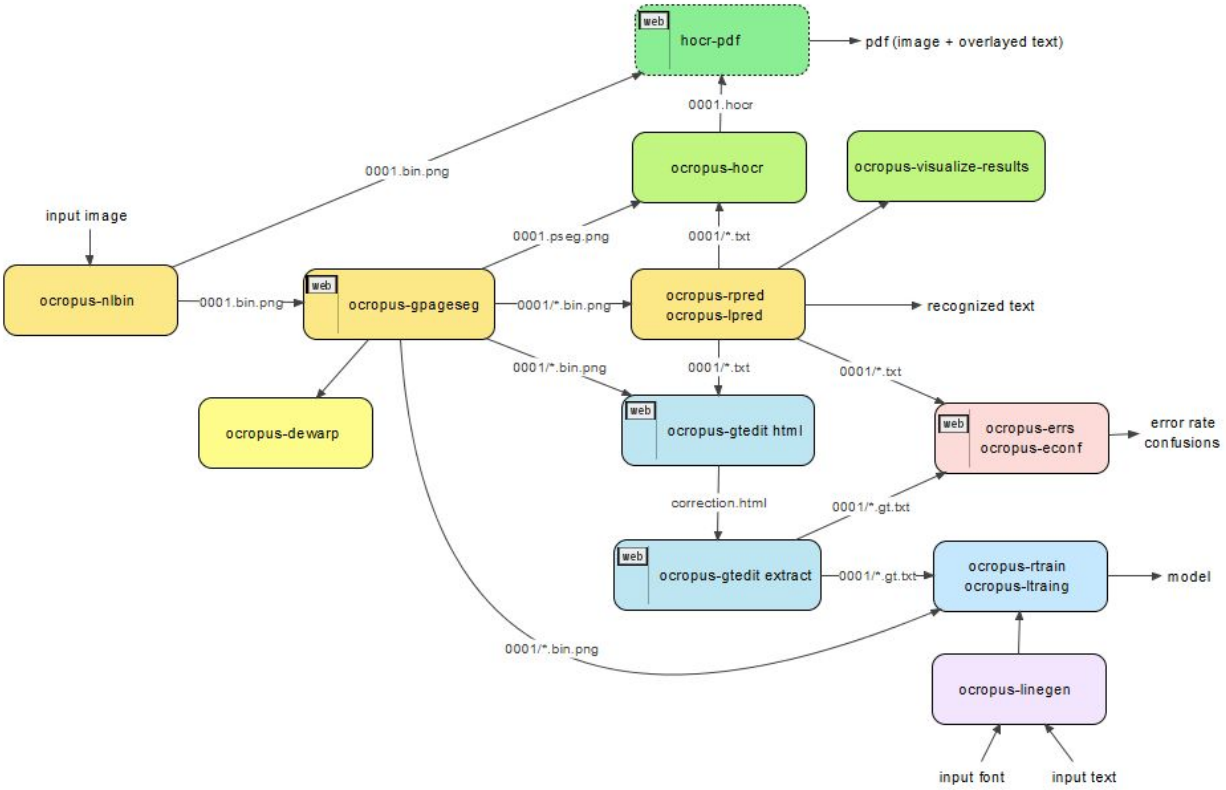
“ImageMagick”, librería que permite realizar operaciones básicas sobre estas. El comando necesario para ello es el siguiente:

```
$> sudo apt-get install imagemagick
```

Funciones como “convert” serán de gran utilidad para recortar y enfocar la imagen y obtener resultados como el siguiente:

Una vez preparado todo, toca hacer uso del OCR. Como ya se comentó anteriormente, se trata de un sistema de detección de tipo LSTM sin entrenar. Este último dato es extremadamente importante puesto que aunque por defecto venga con una base de entrenamiento para usar, no proporciona los resultados esperados, lo que nos lleva a tener que generar una que sea capaz de detectar los campos deseados en la imagen basándonos en las características de nuestro problema. Se entrena por tanto a la red neuronal para que detecte textos con el formato que se presenta en las cartas. Es necesario comentar que el estado de las imágenes a emplear durante todo el proyecto es perfecto o casi perfecto. Sin embargo, los resultados obtenidos con dicha base siendo empleada sobre cartas en estado no tan idóneo resultan como se verá en los resultados bastante prometedores.

En lo referente a este OCR existe una gran cantidad de comandos con diferentes opciones según qué y en qué formato se desea obtener. El siguiente esquema muestra todas las posibles aplicaciones de este sistema de detección:



Y es que aunque sean muchas las posibles opciones de uso para este OCR nos centraremos en explicar con detalle sólo aquellas de las que se usan:

$> ocropus-nlbin -n [imagen.jpg] -o book

Dónde:

- ***[imagen.jpg]*** es la imagen con la que se va a trabajar.

Antes de que el sistema de detección pueda trabajar con la imagen es necesario pasarla a binario para que se pueda apreciar con un mayor contraste las variaciones entre el fondo de la imagen y los caracteres (El resultado se guarda en el directorio "book"). Una imagen binaria no es más que una imagen que tiene únicamente dos posibles valores para cada píxel.

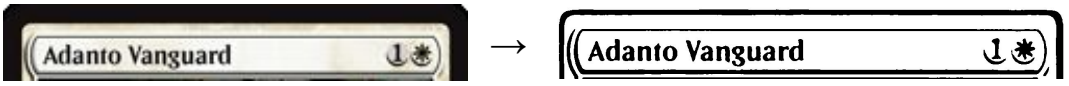
La forma más usual y simple de pasar una imagen a binario es aplicándole un filtro de grises similar al siguiente:

Gris(x,y) = (0.299 * r(x,y)) + (0.587 * g(x,y)) + (0.114 * b(x,y))

Donde el valor de gris para el píxel (x,y) viene determinado por los valores de RGB de ese píxel por unas determinadas constantes.

Se transforma por tanto la imagen de forma que para cada píxel, su valor R, G y B son iguales permitiéndonos trabajar con un único valor que representa los tres colores para cada punto de la imagen.

Tras aplicar el filtro de grises, se elige un umbral propio que sirve como separador entre píxel de fondo y de elemento detectado. En caso de que el valor representativo de gris obtenido supere el umbral se le asigna un valor (por ejemplo "1") y en caso contrario el otro posible ("0") de forma que queda una imagen separada por dos colores únicos (por lo general blanco y negro) de la siguiente forma:



Es necesario tener en cuenta que factores externos como el brillo, el estado de la imagen, el enfoque, el tipo de formato de imagen y en nuestro caso el estado de las cartas puede provocar que el resultado no sea el esperado desencadenando muy probablemente en un análisis con resultados erróneos. Aun así esto no afectará a los resultados obtenidos en este proyecto puesto que como se indicó con anterioridad, el estado de las imágenes de las cartas que se usa es prácticamente perfecto.

$> ocropus-gpageseg -n book/[0001.bin.png]

Dónde:

- ***[0001.bin.png]*** es la imagen pasada a binario usando el comando anterior.

Este comando realiza un análisis del diseño de la imagen a analizar. Determina en qué secciones de lo que Ocropus determina "página" se encuentran los caracteres a procesar y les asigna una etiqueta para su posterior tratamiento.

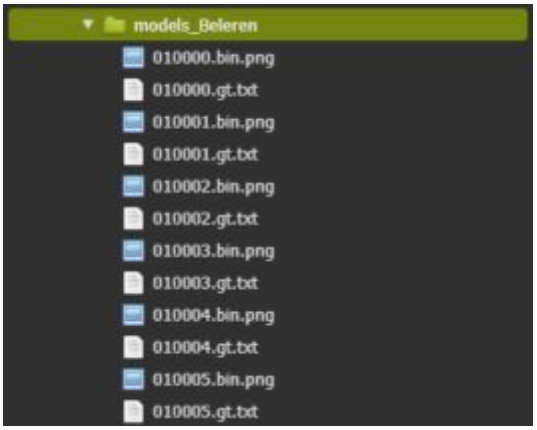
$> ocropus-rtrain -o [baseConocimiento] [imagendir/*.png]

Dónde:

- ***[baseConocimiento]*** es el nombre que se le va a dar a nuestra base de conocimiento.
- ***[imagendir/*.bin.png]*** es el directorio que contiene todas las imágenes que se usarán para el entrenamiento de la red neuronal. Estas tienen que estar pasadas a binario.

Este punto es el de mayor relevancia para el trabajo con el OCR para este proyecto. Como ya se comentó, el sistema de detección parte de una situación sin entrenamiento alguno, por lo que se tiene que crear una base de conocimiento desde cero empleando imágenes del juego.

Durante la ejecución del comando, el sistema de detección buscará un fichero con el mismo nombre que la imagen que esté utilizando para aprender en ese momento solo que con terminación ".gt.txt" (proveniente de "Get true" u "Obtener la verdad" haciendo referencia a que en su interior está lo que se supone que tiene que ser correcto). En ese fichero se ha de introducir manualmente el texto de lo que se supone que tiene que estar escrito en la imagen que está analizando. Así, la carpeta con el contenido a tratar por el OCR está organizada por pares de ficheros con el siguiente patrón:



Al acabar, generará un fichero "my-model-xxxxxxxx.pyrnn.gz" que es la base de conocimiento que ha generado según lo aprendido. Así pues, cuanta mayor sea la cantidad de imágenes y la variación entre las palabras contenidas en ella mejor serán los resultados del entrenamiento y de los análisis posteriores usando dicha base.

$> ocropus-rpred -m [baseConocimiento] [dirImagenes/img.bin.png]

Dónde:

- ***[baseConocimiento]*** es la base de conocimiento entrenada basándonos en lo dicho en el punto anterior.
- ***[dirImagenes/img.bin.png]*** imagen a analizar previamente pasada a binario.

Realiza el reconocimiento de caracteres por líneas en la imagen que se pasa por parámetros usando como referencia lo aprendido con la base de

entrenamiento que se le indique (Es necesario que la imagen esté en binario). Devolverá como resultado, un fichero de texto plano (.txt) con lo que ha logrado entender escrito dentro.

Una vez explicado cómo trabajar con el OCR en cuestión, tocará hablar de nuestro caso particular. Para el desarrollo de este trabajo, la base de conocimiento cumple con las siguientes características:

- Más de 1.500 cartas diferentes de las últimas 6 expansiones usadas como ejemplo para el aprendizaje. Estas fueron preparadas junto a su correspondiente fichero ".gt.txt" con el supuesto texto de la imagen escrito dentro.
- El OCR entrenó con estas 1.500 cartas durante más de 2.275.000 iteraciones para obtener la susodicha base usada en este proyecto.
- Se tardó aproximadamente un mes y quince días en preparar y procesar todos los datos para dejarlo en el estado actual.

Con estos datos, los resultados obtenidos de los análisis posteriores usando el sistema de detección ya entrenado son bastante prometedores:

1. ~98% de acierto sobre el resto de cartas (Aproximadamente 50~60) elegidas de forma aleatoria (2). Es necesario mencionar que del total de fallos que se obtuvieron en este apartado la mayoría eran letras "t" que debido al formato de las letras de las cartas el OCR tiende a confundirlo con otras letras como la "l" o con signos de puntuación como el ";".

2. ~74% de acierto sobre cartas del apartado anterior en un estado con mayor cantidad de desperfectos (problemas de brillo y enfoque).

errors	51
missing	51
total	2125
err	2.400 %
errorserrnomiss	2.400 %
2.4	

Desarrollo del proyecto - Servidor Linux

Preparación

Durante el desarrollo del proyecto, se creará y simulará un servidor que devuelva como resultado el texto de la carta que se le pase mediante transferencia de ficheros “cliente-servidor” una vez esta haya sido analizada por el OCR. Para la realización de este apartado, se ha pedido a la universidad que nos dé acceso a una máquina virtual alojada en el IAAS de la ULL para que puede usarse para estos fines.

Se usará el lenguaje bash para trabajar con una consola Linux y ejecutar en esta un script que ejecute el sistema de detección sobre la imagen con la que se esté trabajando. Para que funcione según se ha preparado en este trabajo, es necesario tener instalado en el servidor bash y python, así como todos los módulos y librerías señalados en el apartado cuatro del capítulo anterior. Estos son esenciales puesto que sin ellos no se podría hacer funcionar al OCR correctamente.

Por último será necesario enviar y almacenar en el servidor la base de conocimiento que se ha entrenado previamente. Hecho esto, se tendría todo lo necesario para dejar preparado el servidor listo para su uso.

Fase de desarrollo del servidor Linux

Pasamos pues a explicar cómo hacer uso del mismo. Se preparará un script en Bash al que se pueda llamar de forma directa cuando sea necesario hacer uso del OCR de forma remota. Este preparará una imagen con un nombre con patrón determinado (pasada previamente por transferencia de ficheros) para ser analizada (empleando las herramientas de “ImageMagick”) y se la pasará al OCR, devolviendo el texto de la misma como resultado.

Dicho esto, procedemos a enseñar el contenido de dicho script con el objetivo de poder observar y explicar con mayor detalle lo que este hace paso por paso:

```
#!/bin/bash

    convert -size 175x24 $1 +repage -crop 175x24+17+17 weop.jpg &> /dev/null
    convert weop.jpg -resize 1248x180 weop2.jpg &> /dev/null
    rm -rf weop.jpg
    ./ocropus-nlbin -n weop2.jpg -o book &> /dev/null
    rm -rf weop2.jpg
    ./ocropus-gpageseg -n 'book/????.bin.png' &> /dev/null
    ./ocropus-rpred -m soloclear-00080000.pyrnn.gz 'book/????/??????.bin.png' -n &> /dev/null

    cat book/0001/010001.txt
    rm -rf ./book/*
```

Donde, en las dos primeras líneas se recorta y enfoca la imagen en la sección que nos interesa (convert) y como ya se dijo, posteriormente se pasa al OCR para que convierta la imagen en binario, genere el diseño de texto de la imagen y la lea guardando el resultado en un fichero llamado 010001.txt (último apartado del capítulo 3). Este se muestra con el "cat", para que se pudiese comprobar durante el desarrollo del TFG que los resultados que se devolvía eran los correctos.

En caso de ejecutar el script desde la consola del servidor, nos mostraría todos los procesos que ejecuta, de forma similar a la siguiente:

```
INFO:  writing
/usr/lib/python2.7/dist-packages/scipy/ndimage/measurements.py:431: FutureWarning: Conversion of the second argument of
issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).t
ype`.
  safe = ((np.issubdtype(dt, int) and dt.itemsize <= int_size) or
INFO:
INFO:  ########## /usr/local/bin/ocropus-gpageseg -n book/????.bin.png
INFO:
INFO:  book/0001.bin.png
/usr/lib/python2.7/dist-packages/scipy/ndimage/measurements.py:431: FutureWarning: Conversion of the second argument of
issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).t
ype`.
  safe = ((np.issubdtype(dt, int) and dt.itemsize <= int_size) or
INFO:  scale 42.249260
INFO:  computing segmentation
INFO:  computing column separators
INFO:  considering at most 3 whitespace column separators
INFO:  computing lines
INFO:  propagating labels
INFO:  spreading labels
INFO:  number of lines 2
INFO:  finding reading order
INFO:  writing lines
INFO:       1  book/0001.bin.png 42.2 2
INFO:
INFO:  ########## /usr/local/bin/ocropus-rpred -m /mnt/c/Users/User/Desktop/P
INFO:
INFO:  #inputs: 2
# loading object /mnt/c/Users/User/Desktop/Programa/ocropy/soloclear-00080000.pyrnn.gz
```

Aunque todo esto no es necesario mostrarlo, motivo por el cual, todos los resultados devueltos por consola han sido desviados a "/dev/null" lo que nos permite eliminar los mensajes innecesarios, dejando visible solamente el texto de la carta que se haya analizado.

Desarrollo del proyecto - Aplicación

Introducción

Toca pues hablar del programa desarrollado durante el proyecto. Se trata de una aplicación multiplataforma (Actualmente está preparada para su uso en Windows y en Linux) escrita en Java. Se ha elegido este lenguaje puesto que su punto fuerte reside en su fácil forma de portarlo a otras plataformas.

Para trabajar en este apartado se ha empleado el IDE de Eclipse Java. Para su correcto funcionamiento, es necesario tener instalado el JRE y el JDK en su versión más actual (1.8). Estas herramientas pueden obtenerse fácilmente a través de la web de Oracle Java:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

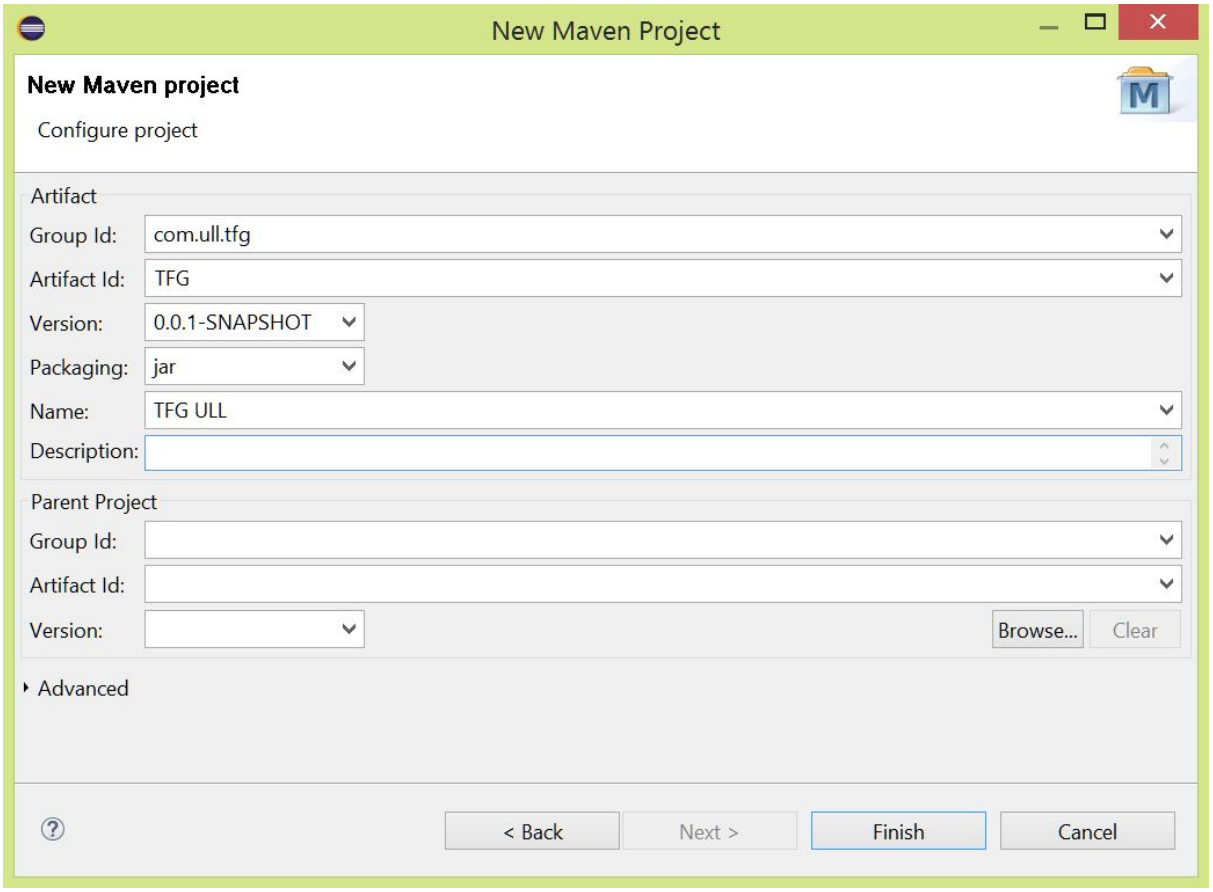
Maven en Eclipse

¿Qué es Maven y para qué nos será útil?

Maven es una herramienta de software para la gestión y construcción de proyectos Java. Nos permitirá servir como repositorio (en este caso integrado en Eclipse) para alojar nuestro proyecto bajo el principio de Convención sobre Configuración (CoC), paradigma de programación que busca minimizar el número de decisiones que un desarrollador necesita hacer.

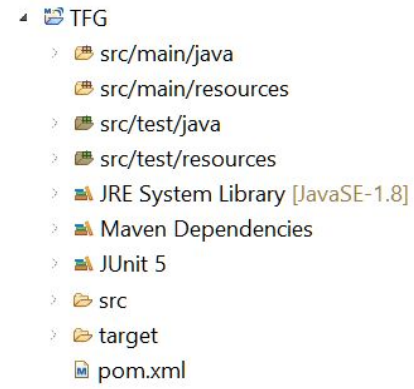
Se trata de un entorno con un sistema controlado de directorios que permite programar y trabajar con recursos con facilidad. Entre las características de mayor utilidad de Maven, la introducción de las librerías al proyecto resulta ser la más ventajosa logrando añadir cualquiera sin siquiera descarga alguna. Además nos proporciona un contenedor dedicado específicamente para almacenar pruebas que se usarán más adelante para asegurarnos del correcto cubrimiento del código.

Se procede pues a su creación. Para ello, vamos al menú de navegación de Eclipse a través de la ruta: “File” → “New” → “Other” y buscamos la opción “Maven Project” y le damos al botón “Siguiente”. En la siguiente ventana, activamos la opción de crear un proyecto simple (usando una selección de arquetipo por defecto) y continuamos hasta que aparezca el panel de características como muestra la imagen:



Se rellenarían pues los campos como se encuentran indicados, teniendo en cuenta que:

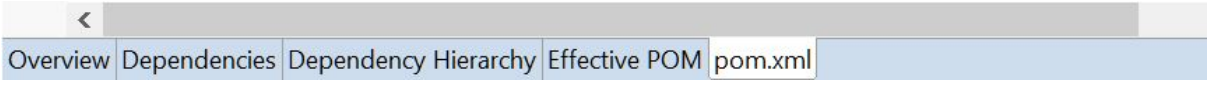
- El “Group Id” es el nombre de un dominio que tu controlas. Al tratarse de un proyecto, se inventará uno para el caso que nos ocupa.
- El “Artifact Id” es el nombre del “.jar” que se va a generar sin incluir la versión del mismo.
- La versión es la por defecto.
- El empaquetamiento es tipo “.jar” como se comentó.
- Y el nombre del proyecto: “TFG ULL”.



Una vez configurado, se generará un árbol de directorios como el de la derecha en el que se distinguen por orden:

- Las carpetas del código y sus recursos.
- Las carpetas de los tests y sus recursos.
- Las librerías.
- Carpetas del explorador con todo el contenido nombrado en los puntos anteriores.
- El fichero "pom.xml", que responde a las siglas de "Project Object Model" y antes llamado "project.xml" (con Maven 1) es un fichero XML que actúa como la unidad principal de un proyecto Maven. En su interior se guarda toda la información acerca del mismo, así como las fuentes, tests, dependencias, plugins, versiones y más, de forma que antes de ejecutarse cualquier tarea o proyecto, Maven comprueba el contenido de este fichero y actúa en función de la información guardada.

Echemos un vistazo con mayor detalle al contenido del "pom.xml". Para ello lo abrimos desde eclipse y seleccionamos la última pestaña llamada igual que el fichero:



Desde ahí se puede observar el aspecto de la configuración que se preparará a lo largo del proyecto. Cuando sea necesario añadir una dependencia o librería externa, así como definir parámetros propios del proyecto e incluso perfiles para otras herramientas usadas en el desarrollo del mismo se ha de configurar el XML que aparece en dicha pestaña.

A medida que se avance con la explicación se irán indicando con detalle todas las adiciones necesarias en este fichero. Así, excluyendo el contenido desde la marca "dependencies" hasta el final, que son características recomendadas de configuración inicial, el contenido inicial debería ser:

```
<project xmlns="..." xmlns:xsi="..." xsi:schemaLocation="...">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.ull.tfg</groupId>
	<artifactId>TFG</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>TFG ULL</name>
	<dependencies> </dependencies>
	<profiles> </profiles>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.6.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

[ ] Necesario/Básico
[ ] Recomendado

Por último toca hablar del ciclo de vida de un proyecto Maven. Es el conjunto de acciones o metas que se busca realizar con los ficheros del proyecto con los que se esté trabajando. Dentro de todas las partes de un proyectos, Maven identifica como propias del ciclo de vida del mismo (aquellas que en medida de lo posible tienen que ser realizadas) las siguientes.

Parte de ciclo de vida	**Acción que se realiza**
*$> mvn build **	Genera y organiza la estructura y configuración del proyecto Maven para que el resto de partes del ciclo de vida puedan llevarse a cabo (Propio de Eclipse* - No puede ser considerado parte del ciclo de vida como tal - Se accede con la ruta: “click” derecho sobre el proyecto → Run As → Maven build).
$> mvn compile	Genera los ficheros “.class” compilando los fuentes “.java”.

$> mvn test	Ejecuta los test Junit existentes (aborta si alguno falla).
$> mvn package	Genera el fichero ".jar" con los ".class" compilados.
$> mvn install	Copia el fichero .jar a un directorio del ordenador donde Maven deja todos los .jar para poder usarse en otros proyectos Maven del mismo ordenador.
$> mvn deploy	Copia el fichero ".jar" a un servidor remoto poniéndolo disponible para cualquier proyecto Maven con acceso ha dicho servidor.

```
validate
generate-sources
process-sources
generate-resources
process-resources
compile
process-test-sources
process-test-resources
test-compile
test
package
install
deploy
```

Así, cuando se ejecuta cualquiera de las acciones arriba listadas, Maven, irá verificando fase por fase todo el ciclo de vida según el orden de la tabla realizando hasta la elegida en el comando todas aquellas que no se hayan ejecutado previamente.

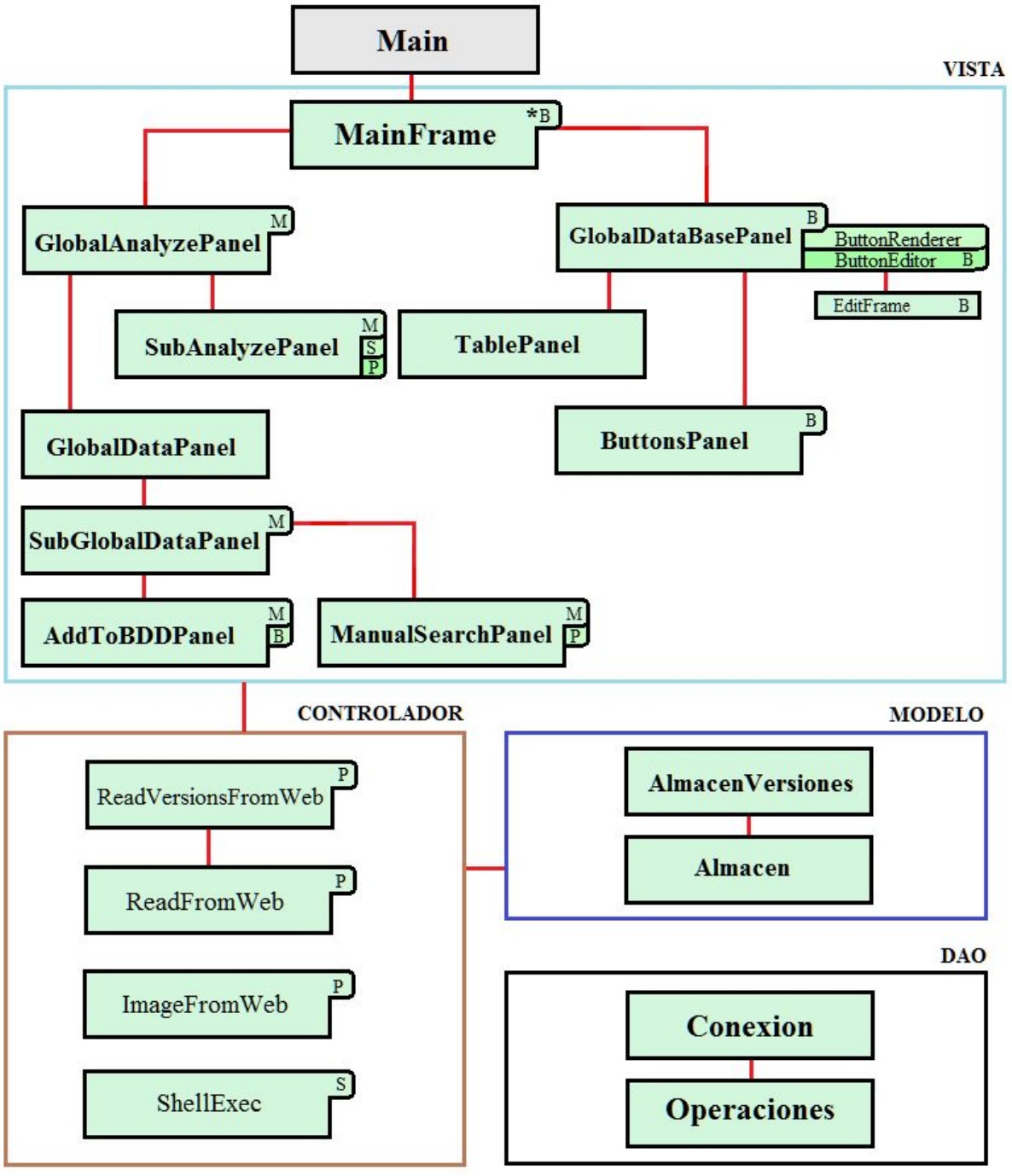
Las acciones llevadas a cabo en un ciclo de vida completo se encuentran listadas a la izquierda.

Estructura del programa

Es el momento pues, de hablar de la organización del programa. Esta se ha decidido estructurar basándonos en el estilo de arquitectura "Modelo Vista Controlador" (MVC). Este propone la construcción de tres componentes diferentes que son los que le dan nombre al patrón:

- El modelo, que contiene la información con la que el sistema trabaja y la manera de gestionarla correctamente.
- La vista, que es el elemento visual del sistema, presenta la información del modelo en un formato adecuado.
- El controlador, que se encarga de manejar peticiones y eventos generalmente producidos por acciones llevadas a cabo por la interfaz que se presenta en la vista. Sirve como capa intermedia entre las dos citadas arriba.

Dicho esto, el siguiente esquema muestra todas las clases que han sido creadas para la aplicación, así como a qué capa pertenece cada una:



Por motivos organizativos y por la facilidad que implica, como se puede observar, se ha creado una capa extra llamada “DAO” que incluye la pareja de clases con métodos encargados de trabajar con la base de datos. Realizan todo tipo de acciones, desde crearla y conectarse al sistema embebido de Apache Derby DB hasta hacer operaciones de inserción o borrado entre otras.

En relación al resto de capas, todas están conectadas basándonos en la premisa MVC y se interrelacionan entre ellas según las acciones que realicen o los campos que abarquen. Junto a cada clase hay una etiqueta (en caso de que sea necesario) indicando cuales son los recursos de los que hace uso basándonos en la siguiente clasificación:

- "B" (Base de Datos): Son aquellas clases que se conectan a la base de datos y hacen algún tipo de operación empleando los métodos contenidos en las clases de la capa "DAO". La clase "MainFrame" (*) es una excepción puesto que solamente se dedica a crear la base la primera vez que se ejecute el programa y a cargarla en el resto de ocasiones.

```
private transient Conexion cn;
public MainFrame(){
    cn = new Conexion();
    cn.accederBD(true);
}
```

Crear BD. Si ya está creada, cargarla.

- "M" (Modelo): Aunque resulte obvio que el modelo esté conectado de una forma u otra a la vista, este marcador indica qué clases exactamente hacen uso de la información guardada en las instancias creadas. Como se comentó al principio, un objetivo del proyecto era crear plantillas con la información recopilada. Así pues, se han creado ambas clases en función del esquema mostrado en el último apartado del segundo capítulo de forma que "AlmacenVersiones" corresponde a la plantilla en sí y "Almacen" es una de todas las versiones que se guardan en la clase anterior.

```
public class AlmacenVersiones {
    private String nombreCarta;
    private String generalUrl;
    private int numVersiones = 0;
    private ArrayList<Almacen> versiones = new ArrayList<>();
                        ...
}
public class Almacen {
    private String url;
    private String expansion;
    private double tendenciaPrecio;
    private double minimoPrecio;
    private double foilPrecio;
    private String rarity;
                ...
}
```

Plantilla

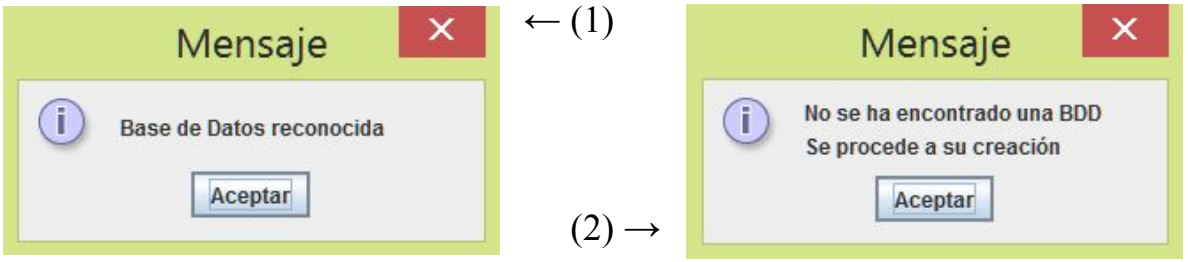
Versión/Expansión

- “P” (Scraping): Indica qué clases (de la vista) hacen uso de los métodos del controlador encargados de hacer “scraping”, así como cuáles son las clases de dicha capa que contienen estos métodos. Por definición el “web scraping” se define como: *técnica usada mediante programas de software para extraer información de sitios web*. En nuestro caso, se trabaja con el código HTML directo de la página para extraer la información. Se optó por esta opción al no existe API para Java que permitiese obtener los datos que se buscaban para el proyecto.
- “S” (Servidor): Señala únicamente a la clase “SubAnalyzePanel” pues es la encargada de emplear los métodos de la clase “ShellExec” para acceder correctamente al servidor Linux que se explicó en el capítulo anterior. Los procesos que llevados a cabo son: transferir una imagen, ejecutar el script “Bash” y obtener el resultado esperado para posteriormente analizarlo y crear una plantilla con dichos datos y los extraídos del scraping anteriormente citado.

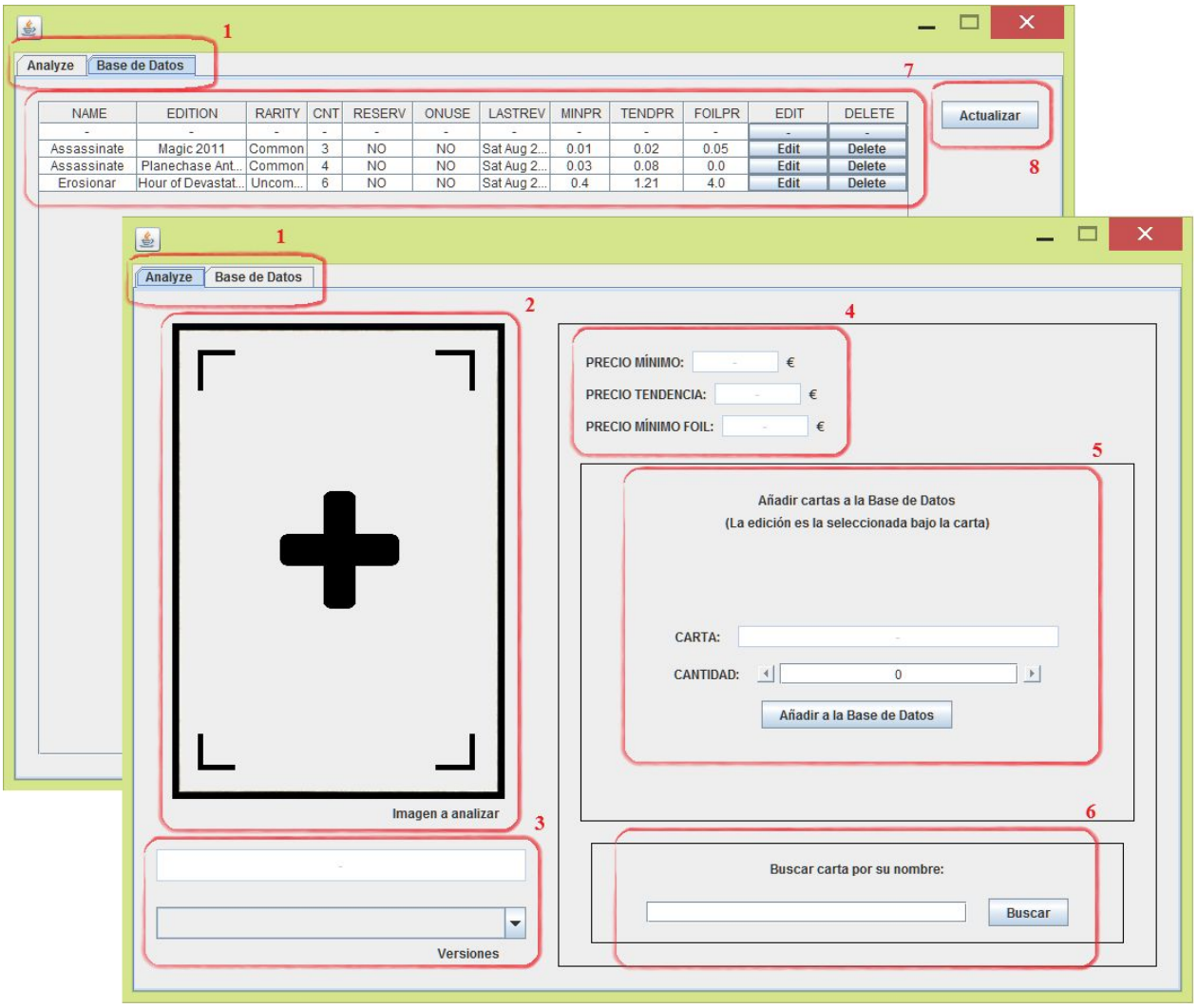
Interfaz de la aplicación

Pasemos a ver qué elementos componen la interfaz del programa Java y cuál es la función de cada uno. Empezamos arrancando la aplicación.

Dependiendo de si en la carpeta local donde está alojado el programa está creada la base de datos Apache Derby DB (1) o no (2), este cargará la existente o creará una vacía y lista para su uso.



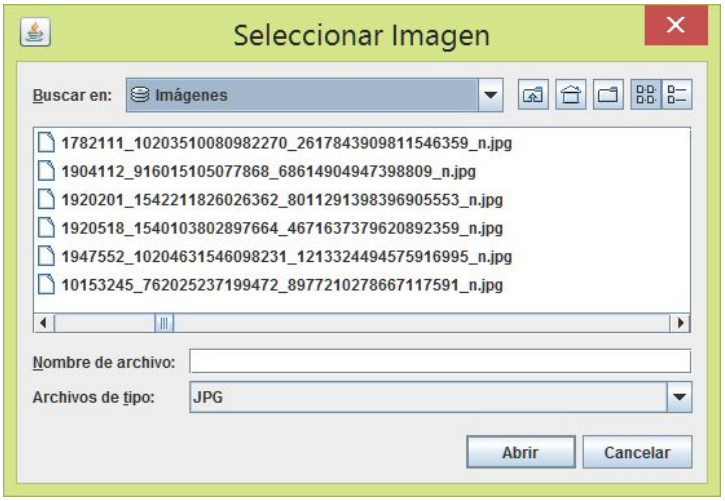
Si todo ha salido correctamente, la aplicación habrá cargado toda la información necesaria y nos mostrará la ventana principal de la misma. Su aspecto es el siguiente:



Dónde:

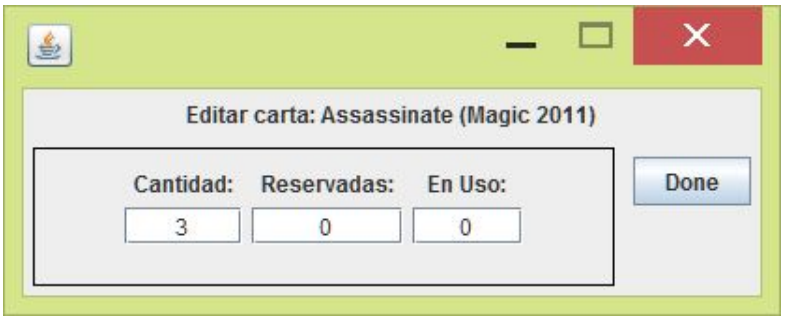
1. Es la sección donde se indica cuál de las dos pestañas disponibles se desea usar. Como se puede apreciar en la imagen, por un lado se tiene la pestaña de análisis en la que se trabaja con la búsqueda y el tratamiento de las cartas, sus precios y otros tipos de información. Como se verá más adelante, en este panel es donde se llama al servidor para hacer uso del sistema de detección. Por otro lado se tiene la pestaña que aloja la base de datos embebida con las cartas y la información que se guarda sobre ellas.
2. Es el recuadro donde se muestra la imagen de la carta a analizar por el OCR o la imagen encontrada usando la búsqueda manual que se explicará más adelante.

Es necesario comentar que se trata de una sección interactiva sobre la que se puede hacer “click” para desplegar un navegador, donde se puede elegir una imagen a tratar por el sistema de detección. Está además, preparado para soportar ficheros en formato “.png” y “.jpg”.



3. Contiene el nombre de la carta detectada/analizada y un desplegable con todas las versiones disponibles para la misma. Debido a que el precio de mercado de estas varían según de qué versión se trate, cuando se seleccione una en concreto para la carta con lo que se esté trabajando, el contenido de los campos 4 y 5 variarán en consecuencia.
4. Muestra los diferentes precios de mercado de la carta analizada (o buscada) basándonos en la expansión seleccionada en el campo comentado en el apartado anterior.
5. Es la sección de adición de una carta a la base de datos. Cuando se busque alguna carta, mostrará cuántas de ellas se tiene en la base de datos y permitirá o bien añadir una nueva entrada a la misma o modificar la existente. Al igual que ocurre con el apartado anterior, este panel también es dependiente de los valores en los campos ya citados.
6. De forma opcional se puede buscar una carta (y toda la información pertinente vinculada a ella) de forma manual introduciendo el nombre de la misma en inglés o español.
7. Muestra el contenido almacenado en la base de datos. Se muestra la información obtenida durante el análisis y el scraping además de otros campos como la rareza y la fecha de adición de cada entrada a esta.

Además cada TUpla contiene un botón para borrarla de la base de datos directamente y otro para editar su contenido de forma que el usuario que haga uso de la aplicación pueda decidir si en su colección las cartas las tiene en uso o reservadas.



8. Por último un pequeño botón añadido que permite actualizar la tabla de la aplicación por si el contenido de la base de datos cambia de alguna forma.

Una vez mostrada la estructura y aspecto del programa, es el momento de hablar con mayor calidad de detalles sobre cada herramienta usada en este.

(Herramientas) Web Scraping - JSoup

Empecemos por el "scraping" del que se habló en el apartado anterior. Java tiene limitación a la hora de leer cadenas con demasiados caracteres. Esto desembocó en una serie de problemas que impedía leer el HTML de la web con la que se trabajaba.

Para solucionar estos inconvenientes se empleó la herramienta JSoup. Es una librería preparada específicamente para trabajar con elementos web, por lo que es ideal para el caso que nos ocupa. Así por ejemplo se puede captar el código HTML de una página con tan solo dos líneas de la siguiente forma:

```
Document doc = Jsoup.connect(Url).get();
String line = doc.toString();
```

Para añadir esta librería a nuestro proyecto, es necesario introducir las siguientes líneas en la sección de "dependencies" de nuestro "pom.xml":

```
<dependencies>
	<dependency>
		<groupId>org.jsoup</groupId>
		<artifactId>jsoup</artifactId>
		<version>1.11.3</version>
	</dependency>
</dependencies>
```

(Herramientas) Conexión con el servidor Linux - Encriptación

Con el fin de que la conexión con el servidor sea segura, se ha evitado mostrar información como la contraseña de acceso al mismo o la dirección IP en el código fuente de la aplicación. Se ha decidido trabajar con encriptación para este apartado. Es necesario para desarrollar este punto tener incluida la librería "Commons-codec" al proyecto. Para ello se añadirá al fichero "pom.xml" como con el caso anterior, con las siguientes líneas:

```
<dependencies>
	<dependency>
		<groupId>commons-codec</groupId>
		<artifactId>commons-codec</artifactId>
		<version>1.11</version>
	</dependency>
</dependencies>
```

Dicho esto, se han creado dos ficheros que contendrán esta información dispuesta de la siguiente forma:

- Por un lado "adr.config" que almacena las posibles direcciones IP del servidor que se quiera usar. El contenido de este fichero no estará encriptado puesto que no es considerado información relevante.
- Por otro lado "client.sign" que guarda todas las contraseñas válidas de acceso a dicho/s servidor/s (en nuestro caso solo es uno). Este punto si requiere ser encriptado puesto que de lo contrario podría generar fácilmente una vulnerabilidad.

Para la encriptación del contenido de este último fichero se hace uso del patrón MD5 (Message-Digest Algorithm 5) que codifica el mensaje en 128 bits representados comúnmente como un número hexadecimal de 32 cifras. Además, la llave de cifrado usada, ha sido tratada basándonos en el esquema de "cifrado por bloques" AES (Advanced Encryption Standard) que es considerado como una de los algoritmos más eficaces y reconocidos en el campo de la criptografía simétrica (Trabaja bajo la premisa de usar una misma clave para cifrar y descifrar mensajes en el emisor y receptor).

De esta forma, cada vez que se intenta acceder al servidor, se llama a un método que le envía a este la cadena correspondiente a la contraseña decodificada bajo dichos patrones como resultado.

```
//Desencriptado MD5
public static String desencriptar(String textoEncriptado) {
    String base64EncryptedString = "";
    try {
        byte[] message = Base64.decodeBase64(textoEncriptado.getBytes(UTF8));
        MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] digestOfPassword = md.digest(getSecretKey().getBytes(UTF8));
        byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
        SecretKey key = new SecretKeySpec(keyBytes, "AES");
        Cipher decipher = Cipher.getInstance("AES");
        decipher.init(Cipher.DECRYPT_MODE, key);

        byte[] plainText = decipher.doFinal(message);

        base64EncryptedString = new String(plainText, "UTF-8");

    } catch (Exception ex) { LOGGER.log(Level.INFO, "Error al desencriptar"); }
    return base64EncryptedString;
}
```

Desencriptado MD5

Descifrado AES

(Herramientas) Conexión con el Servidor Linux - JSch

Pudiendo trabajar ya con la contraseña encriptada, será necesario generar una conexión con el servidor para poder usar correctamente el OCR instalado en el mismo. Para ello, se emplearán dos tipos de protocolos que cumplirán cada uno con un papel diferente:

- Por un lado un protocolo SFTP (SSH File Transfer Protocol) con el objetivo de traspasar la imagen que se vaya a analizar en un momento dado.

- Por otro lado una conexión SSH (Secure Shell) para poder ejecutar el script “Bash” que se ha preparado previamente y obtener el resultado que se haya generado con el OCR.

Para trabajar con una mayor facilidad con este tipo de protocolos usando el lenguaje Java se instalará la librería “JSch” (Java Secure Channel) añadiendo para ello las siguientes líneas al fichero “pom.xml”:

```
<dependencies>
	<dependency>
		<groupId>com.jcraft</groupId>
		<artifactId>jsch</artifactId>
		<version>0.1.54</version>
	</dependency>
</dependencies>
```

Una vez añadida la biblioteca a nuestro proyecto, seguiremos los siguientes pasos para conectarnos al servidor:

1. Generamos una sesión JSch (objeto “Session”) con el servidor Linux. Para ello es necesario indicar el nombre de usuario, la dirección IP y el puerto (que en nuestro caso es el 22) además de la contraseña desencriptada de la siguiente forma:

```
JSch jsch = new JSch();
Session session = jsch.getSession(USERNAME, host, port);
session.setConfig("StrictHostKeyChecking", "no");
session.setPassword(desencriptar(psw));
session.connect();
```

2. Crearemos a continuación un canal por cada tipo de conexión que se quiera establecer con el servidor.

   2.1. Empezaremos por el protocolo SFTP. Nos situaremos con un comando de cambio de directorio (CD) en la carpeta que contiene los ficheros del OCR y desde ahí se transferirá el fichero que se haya seleccionado.

```
Channel channel= session.openChannel("sftp");
channel.connect();

ChannelSftp channelSftp = (ChannelSftp)channel;
channelSftp.cd("/home/usuario/Programa/ocropy/");

channelSftp.put(urlIMG, "a.jpg");
channelSftp.disconnect();
```

2.2. Al haber terminado de enviar la imagen, creamos otros canal, esta vez con el protocolo SSH para ejecutar el fichero "Bash" y recuperar el resultado devuelto por el sistema de detección.

```
ChannelExec channelExec = (ChannelExec)session.openChannel("exec");
InputStream in = channelExec.getInputStream();

channelExec.setCommand("cd Programa/ocropy && bash Analizaimagen.sh a.jpg");
channelExec.connect();
```

(Herramientas) Base de Datos - Apache Derby DB

Para este punto, se usará en la aplicación la herramienta Apache Derby DB, un sistema gestor completamente diseñado en Java, que nos permite añadir una base de datos embebida a nuestro programa realizando cualquier tipo de operaciones directamente desde el código fuente. Para hacer uso de esta, lo primero que se tiene que hacer es añadir sus dependencias al fichero "pom.xml" de la siguiente forma:

```
<dependencies>
        <dependency>
                <groupId>org.apache.derby</groupId>
                <artifactId>derby</artifactId>
                <version>10.14.2.0</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Hecho esto, es necesario importar la librería en los ficheros que contendrán operaciones sobre la base de datos tal que así:

```java
package dao;

import java.nio.file.Path;
import java.nio.file.Paths;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

public class Conexion {
```

Apache Derby DB

Si todo ha salido correctamente, crear una base de datos y generar una tabla en ella resulta tarea fácil añadiendo las siguientes líneas a nuestro código:

```java
1  Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
   Path currentRelativePath = Paths.get("");
2  String s = currentRelativePath.toAbsolutePath().toString();
3  conn = DriverManager.getConnection("jdbc:derby:" + s +"\\DB\\Derby.DB;create=true");
4  String creartabla = "CREATE TABLE Cartas(NAME varchar(50), ..., PRIMARY KEY (NAME,...))";
5  op.executeCmm(creartabla, conn);
```

1 Elegimos el sistema embebido
2 Obtenemos la ruta del proyecto
3 Creamos la base de datos en dicha ruta
4 Creamos el modelo de una tabla
5 Creamos la tabla con dicho modelo

Esto generará un fichero “derby.log” en la ruta especificada con los registros de conexión a la base de datos y una carpeta (en el ejemplo de la imagen sería “DB/Derby.DB”) que la contendrá a ella y a todas las tablas con los modelos que se generen. En el caso que nos ocupa, sólo se trabajará con una tabla que contiene los siguientes campos y su correspondencia:

- **Name**: El nombre de la carta (PRIMARY KEY).
- **Edition**: La edición de la carta (PRIMARY KEY).
- **Rarity**: La rareza de la carta (Común, …) (PRIMARY KEY).
- **Cnt**: La cantidad de cartas de un tipo.
- **Reserv**: Indica si la/s carta/s está/n reservada/s (PRIMARY KEY).
- **OnUse**: Indica si la/s carta/s está/n en uso (PRIMARY KEY).
- **LastRev**: La última vez que se revisó los valores de la tabla.
- **MinPr**: El precio mínimo para dicha carta.
- **TendPr**: La tendencia de precio para dicha carta.
- **FoilPr**: El precio mínimo para dicha carta en versión especial (“foil”).

Evaluación del código generado

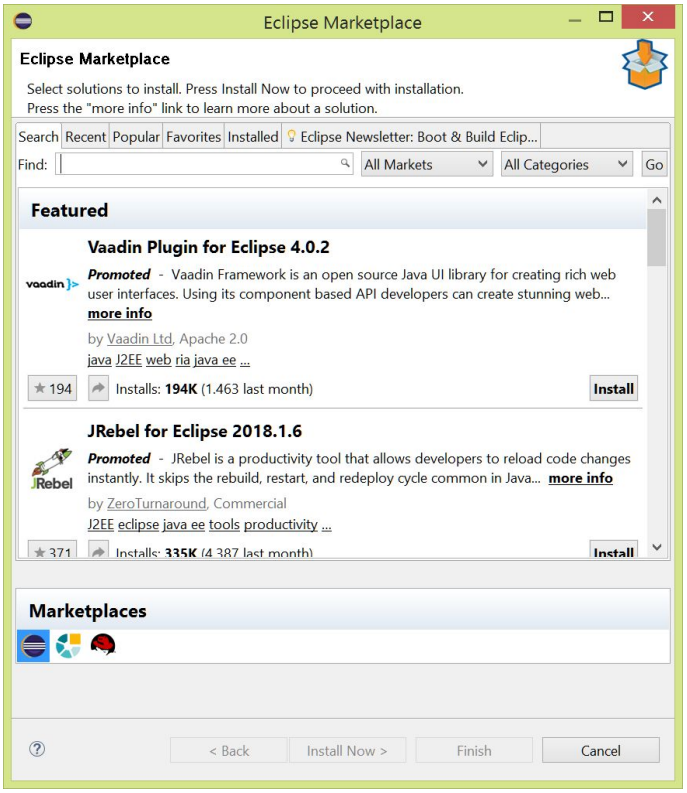
Sonar (SonarQube y SonarLint)

Con el objetivo de evaluar la calidad del trabajo hecho y asegurarnos de evitar posibles errores en el código generado, se ha utilizado la plataforma de evaluación SonarQube y su herramienta para Java integrada en Eclipse SonarLint. Además, presenta la facilidad de permitir su integración con proyectos Maven como es nuestro caso. Nos informará sobre código duplicado, estándares de codificación, pruebas unitarias, cobertura de código, complejidad del mismo, potenciales errores, y diseños de software en general.

Comenzaremos con la instalación de SonarLint, para ello será necesario acceder a la siguiente ruta dentro de Eclipse:

Help > Eclipse Marketplace

Esto nos abrirá el centro de descargas de Eclipse donde se pueden buscar todo tipo de herramientas para trabajar con nuestro proyecto.



Así, en la sección de "Find" introducimos el nombre que buscamos ("SonarLint") y le damos al botón de "Install" para añadirla a nuestro IDE. Desde este momento, todos los proyectos, a menos que se especifique en las opciones lo contrario, estarán bajo el análisis efectuado por esta herramienta. Se identificarán los errores detectados con un subrayado (generalmente verde) sobre ellos en el código.

```
System.out.println("Hi");
```

Con esto ya se tienen localizados los errores y es posible solucionarlos pero para entrar con mayor detalle y hacer un "estudio" de los errores de programación que se encontraban en el código, se utilizó SonarQube. Para hacer uso de esta herramienta con mayor detalle en nuestro repositorio Maven será necesario añadir su dependencia a nuestro fichero "pom.xml":

```
<dependencies>
	<dependency>
		<groupId>org.codehaus.mojo</groupId>
		<artifactId>sonar-maven-plugin</artifactId>
		<version>3.4.0.905</version>
		<type>pom</type>
	</dependency>
</dependencies>
<profiles>
	<profile>
		<id>sonar</id>
		<activation>
			<activeByDefault>true</activeByDefault>
		</activation>
		<properties>
			<sonar.host.url> http://localhost:9000 </sonar.host.url>
		</properties>
	</profile>
 </profiles>
```

Además añadiremos un perfil [ ] en el mismo fichero que indicará dónde se "hosteará" (mostrará) los resultados del análisis del código ya comentado. Para este proyecto se usará el indicado por defecto y veremos los resultados del análisis en local usando el navegador para acceder al SonarQube arrancado desde el puerto 9000.

Por otro lado, hará falta descargar "Apache Maven" directamente desde la web, al igual que los ficheros de SonarQube. Los enlaces son los siguientes:

	https://www.sonarqube.org/downloads/
	https://maven.apache.org/download.cgi

Por último, para empezar a hacer uso del SonarQube, es necesario tener configurado lo siguiente:

1. Primero las variables de entorno. En Windows que es el sistema operativo usado para este proyecto, se accede a través de la siguiente ruta:

   Panel de Control > Sistema (Configuración avanzada del sistema) > Variables de entorno > Variables del Sistema

   Ahí, se crearán las siguientes variables:

Variable	***Ruta***
JAVA_HOME	C:\Program Files (x86)\Java\jdk[x.x.x]
M2	%M2_HOME%\bin
M2_HOME	C:\...\apache-maven-[x.x.x]
MAVEN_HOME	C:\...\apache-maven-[x.x.x]

Y modificamos la variable "Path" para que incluya lo siguiente:

Ruta directa a los ficheros del SonarQube

C:\...\sonarqube-[x.x.x]\bin\windows-x86-64;

Ruta a las nuevas variables de entorno que se crearon arriba

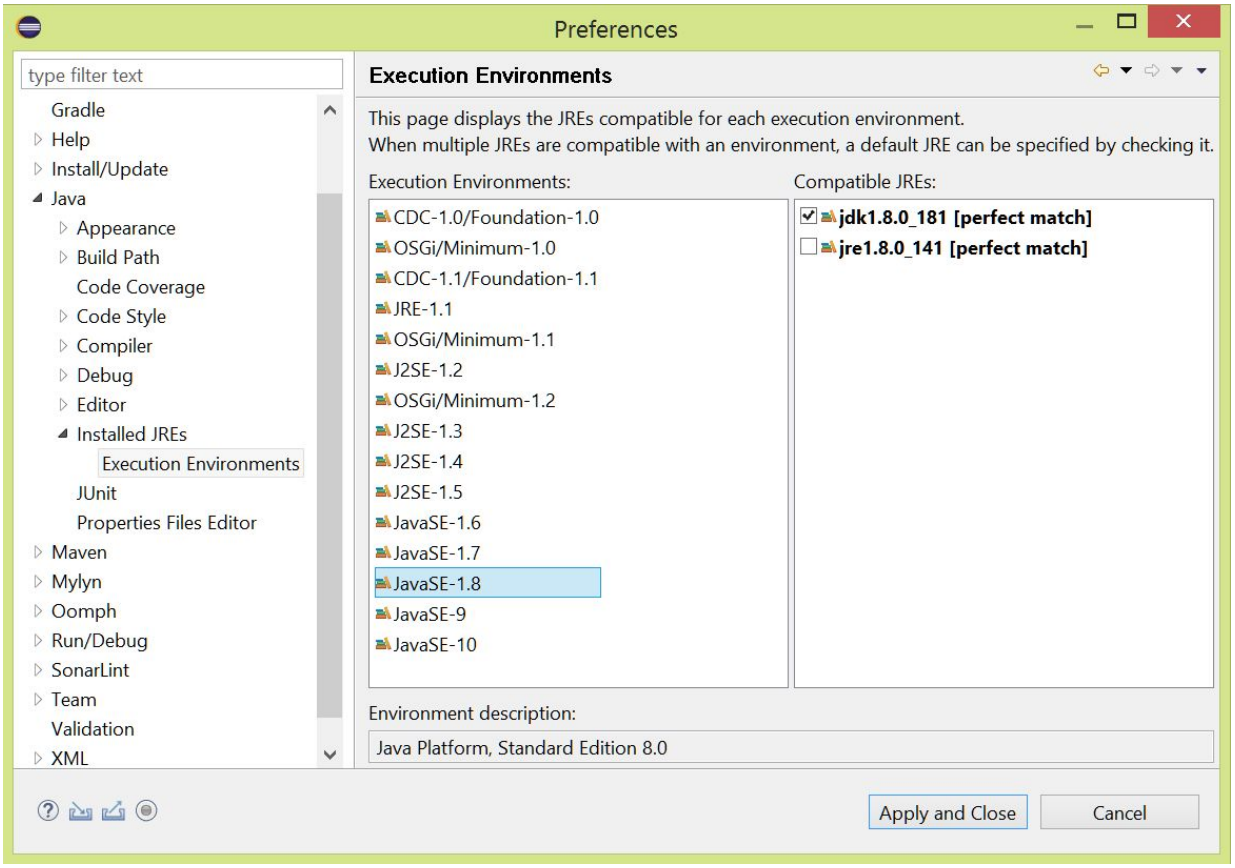
%M2%; %M2_HOME%\bin

No olvidar: [x.x.x] indica la versión descargada de cada herramienta.

2. Por último la configuración del Eclipse. Para evitar errores, es necesario elegir JDK como entorno de desarrollo. Para ello seguimos la siguiente ruta dentro del programa:

Window > Preferences > Java > Installed JREs > Execution Environments

Ahí buscamos el entorno llamado "JavaSE-x.x", donde las "x" son la versión más actual del JDK que se haya instalado anteriormente (Debería estar instalado si se siguieron los pasos del capítulo 5 apartado 1) y seleccionar el JDK en vez del JRE como se muestra en la siguiente imagen:



Con todo esto hecho, nos dirigiremos a la carpeta con el SonarQube que se descargó en su momento y en la subcarpeta "bin", se elegirá el directorio con la versión del sistema operativo que corresponda (en nuestro caso x86-64) y ejecutaremos el servidor "StartSonar" que iniciará el cmd de Windows. Sabremos que el Sonar está preparado cuando salga la frase: "SonarQube is up"

y dejaremos la ventana abierta en segundo plano (En caso de cerrarse, cesaría su ejecución y se tendría que volver a arrancar la aplicación).

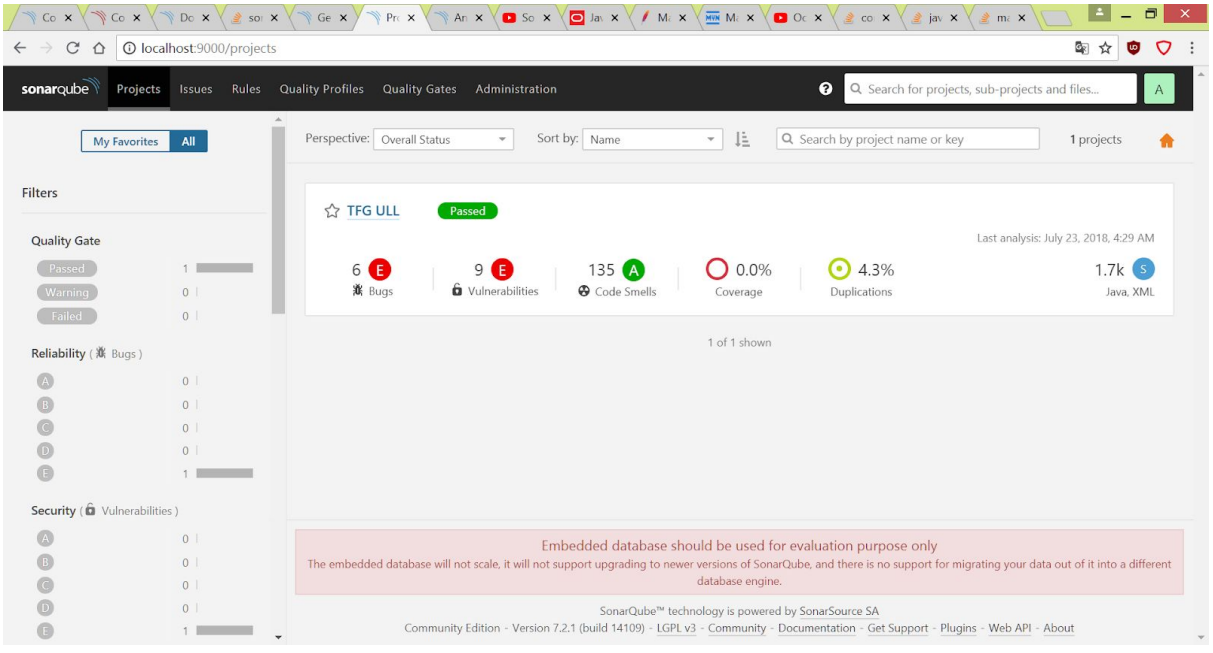
Ya está el SonarQube ejecutándose en el puerto 9000 de nuestro ordenador. Para acceder, abrimos el navegador y entramos en la dirección: http://localhost:9000/. A continuación accederemos con el botón "Log in" usando las credenciales por defecto:

- Nombre: admin
- Contraseña: admin

Nos mostrará el panel con los proyectos analizados por Sonar. Por ahora no debería haber ninguno, asique toca añadir el código de nuestro proyecto al servidor. Para ello será necesario acceder a la carpeta que contiene la versión que se descargó antes de "Apache Maven" y en la carpeta "bin" ejecutamos al igual que con Sonar, el fichero "mvn". Hecho esto, toca dirigirse con la consola de Windows ("cmd") al directorio donde se encuentra el proyecto y ejecutaremos los siguientes comandos:

$> nvm clean
$> nvm package sonar:sonar

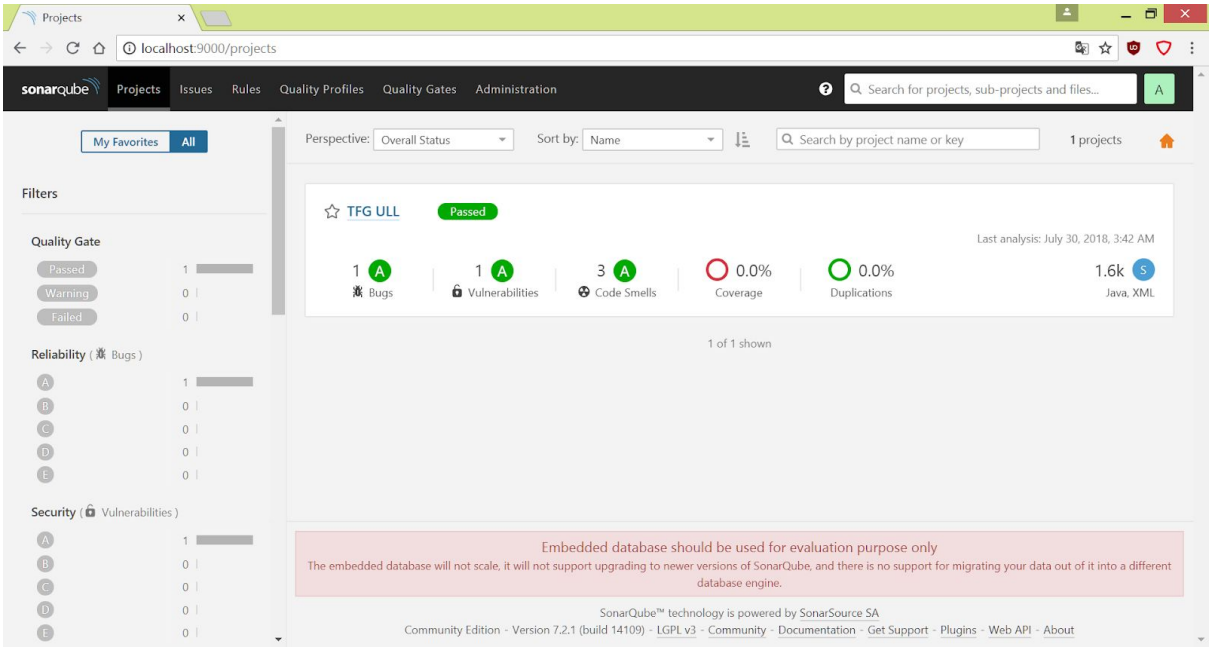
Si todo ha salido correctamente, ya se habrá añadido nuestro proyecto al servidor Sonar. En el caso de este proyecto. El resultado inicial fue el siguiente:



El código contenía 6 “bugs), 9 vulnerabilidades y 135 “code smells” además de un 4,3% del código duplicado. De entre los errores encontrados los más usuales eran los siguientes:

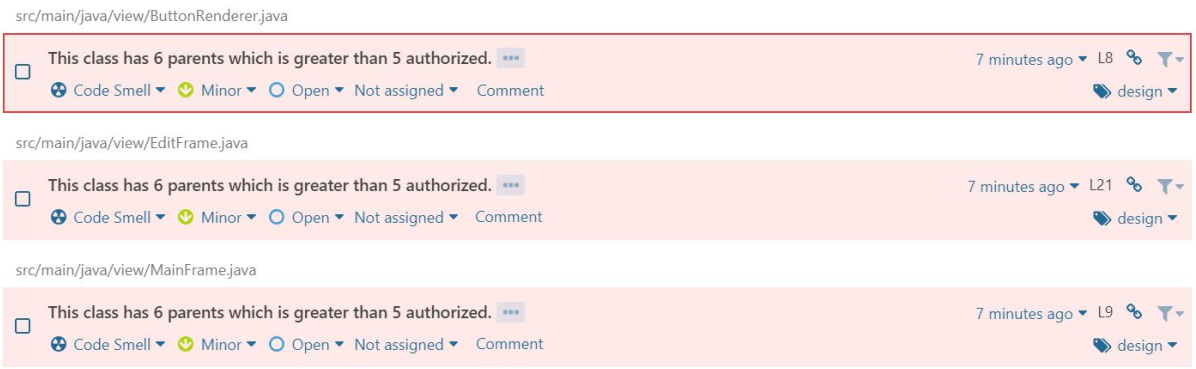
Vulnerabilidades	- Use a logger to log this exception. - Make this IP "10.6.129.157" address configurable. - Use the recommended AES (Advanced Encryption Standard) instead. - Use a variable binding mechanism to construct this query instead of concatenation.
Bugs	- Use try-with-resources or close this [[item]] in a "finally" clause.
Code Smells	- Remove unused local variable. - Reorder the modifiers to comply with the Java Language Specification. - Rename this local variable to match the regular expression '^[a-z][a-zA-Z0-9]*$'. - Remove this empty statement. (Unused catchs) - Define a constant instead of duplicating this literal " [[STRING]] " [[x]] times. - Add a private constructor to hide the implicit public one. - Remove this expression which always evaluates to "true". - Replace this use of System.out or System.err by a logger. - Add the "@Override" annotation above this method signature. - Make this anonymous inner class a lambda.

Tras una semana de arreglos, los resultados de las correcciones fueron bastante prometedores. Se pasó a tener un solo “bug”, así como una sola vulnerabilidad y sólo tres “code smells”. Además se eliminó por completo toda línea de código duplicado.



Estos errores restantes han sido correctamente identificados y se ha logrado determinar por qué son imposibles de corregir:

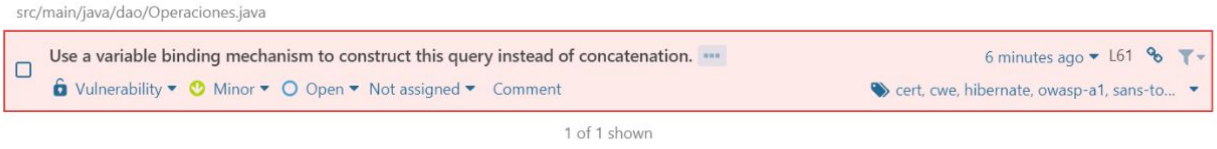
- En el caso de los "code smells" el problema reside en el mismo sitio. En SonarQube, existe una variable que determina el número de "padres" del que una clase puede heredar. Por defecto está establecida a 5 pero las clases predefinidas de Java (JButton…) heredan de 6 o más clases. Y es que aunque los desarrolladores se están planteando cambiar el valor de esta variable, por ahora esta no puede ser modificada lo que hace que nos muestre esos tres errores que "supuestamente" están sin corregir. Aun así, no suponen problema alguno y no generan ningún tipo de vulnerabilidad en nuestra aplicación.



- Por otro lado, el “bug” y la vulnerabilidad se generan porque SonarQube señala que las consultas SQL embebidas en el código (aquellas usadas para las operaciones con la base de datos Apache Derby DB) deben ser únicas y fijas en una misma cadena sin concatenar valores que varían dentro de ellas. Esto resulta casi imposible de solventar puesto que las sentencias SQL necesarias para la aplicación deben tener valores “dinámicos” en función de con qué y cómo se desee trabajar dentro de ella. Un ejemplo sería el siguiente:

```
String q = "SELECT " + fields + " FROM Cartas";
```

  Aun así y al igual que con el caso anterior, se tratan de dos supuestos errores sin mayor importancia y que no provocan problema de ineficiencia alguno.



Cubrimiento del código - Tests JUnit

Por último, se han preparado pruebas unitarias que nos permiten probar nuestro código y generar el cubrimiento necesario sobre las líneas de código de mayor relevancia. Para escribir dichos tests se ha empleado la herramienta integrada en el Eclipse IDE: “JUnit”. Su uso es bastante simple, se crearán ficheros (que como ya se comentó, se almacenarán en un directorio que Maven nos provee) con métodos que hagan uso de nuestras líneas de código y se les pondrá la etiqueta “@Test” sobre ellos para clasificarlos como pruebas y que JUnit los detecte. En caso de requerir la creación de variables/métodos antes o el borrado/tratamiento de estos al final, se escribe “@After” o “@Before” sobre dichas líneas en los ficheros que contienen estos tests, mientras que si se requiere la ejecución de varias de ellas antes o después de cada test se usa @AfterClass y @BeforeClass.

```java
@Test
public void testEncriptar() {
    String text = ShellExec.encriptar("EncryptTest");
    assertEquals("emdOrI/TO8vflRoACqjsgg==", text);
}
```

Junto a las etiquetas arriba mencionadas, JUnit hace uso de los conocido como "asserts" que permiten comparar los resultados obtenidos con los esperados de diferentes formas (un ejemplo está en la imagen anterior). Las más usuales suelen ser:

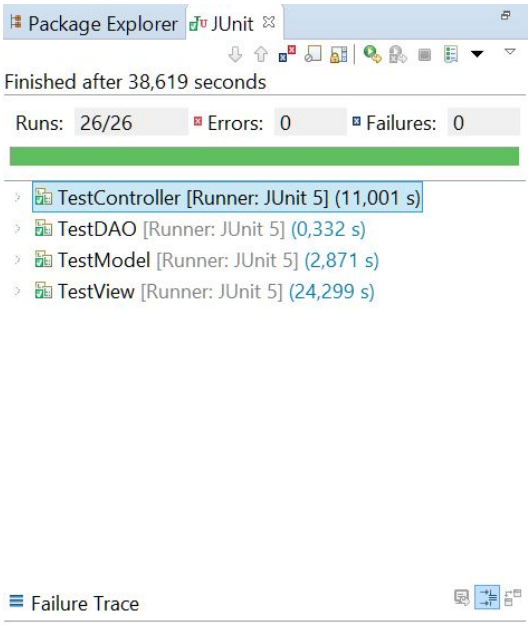
AssertEquals	Comprueba si dos elementos son iguales.
AssertNotEquals	Comprueba si dos elementos son diferentes.
AssertNull	Comprueba si un elemento es igual a "null" (nulo).
AssertNotNull	Comprueba si un elemento no es "null" (nulo).
AssertTrue	Comprueba si un elemento es igual a "true" (Boolean).
fail	Obliga un test a fallar.

En relación a la preparación necesaria para su uso en el repositorio Maven, será necesario añadir, al igual que con el resto de herramientas, sus dependencias en el fichero "pom.xml":

```
<dependencies>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Así pues, una vez alojadas las pruebas en el directorio correspondiente (*src/test/java*) ya se podrían ejecutar en Eclipse IDE desde la siguiente ruta:

(Click derecho sobre el directorio Maven) > Run As > JUnit Test



Para obtener los resultados del cubrimiento del código, la ruta usada es similar:

(Click derecho sobre el directorio Maven) > Coverage As > JUnit Test

En el caso que nos ocupa, se ha generado un fichero con tests, por cada capa que se indicó tenía la estructura de nuestro proyecto intentando cubrir todos los métodos relevantes del código de la aplicación y el resultado del cubrimiento ha sido el siguiente:

Coverage

(default package) (04-ago-2018 23:52:13)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instru...
TFG	83,7 %	5.205	1.015	6.220
src/main/java	81,5 %	4.288	973	5.261
view	82,7 %	3.210	673	3.883
controller	74,0 %	561	197	758
dao	79,6 %	401	103	504
model	100,0 %	116	0	116

Se ha obtenido que los tests generados cubren un 83,7% del código escrito para la aplicación. Es necesario comentar que gran parte del porcentaje restante son las zonas del código que cubren condiciones de error en caso de que problemas como la pérdida de conexión a internet ocurran.

Independientemente de este detalle, los resultados obtenidos son bastante favorables (aseguran el correcto funcionamiento de la aplicación) y nos sirve como un indicador que mejora la fiabilidad de que el código desarrollado funcione.

Conclusiones y líneas futuras

Se puede concluir que los resultados obtenidos de los apartados realizados han sido bastante exitosos. Nunca antes había trabajado ni los campos que se emplearon para este proyecto, ni gran parte de las herramientas para el desarrollo de los mismos, pero me ha permitido aprender con mayor detalle el funcionamiento general de los OCR y las redes neuronales que estos emplean. Son sin duda punto que merecen una revisión y un uso con mayor profundidad puesto que se trata de tecnologías que pueden generar un gran aporte a elementos que se ven o usan en nuestro día a día. Sólo con pensar en aplicar redes neuronales y sobre todo sistemas de detección se abre un amplio abanico de posibilidades para la automatización de tareas.

Pasando a hablar de mi experiencia y resultados personales se podría decir que he obtenido un nivel bastante estable en el campo de la detección de caracteres pero se ve a simple vista que todavía me quedan mucho puntos en los que ampliar mi conocimiento y mejorar lo que he podido aprender gracias a mi trabajo.

En lo referente a lo desarrollado en el TFG y como ya se citó al principio, el conjunto de resultados obtenidos han sido bastante favorecedores. Los análisis generados con nuestra base de conocimiento son muy prometedores, incluso con imágenes que difieren de las usadas para el entrenamiento. Además, la creación de la aplicación, así como las conexiones generadas con el servidor son factores que me pueden servir como un ejemplo funcional cuando intente trabajar con estos campos en un futuro próximo.

Y es que la lista de posibles mejoras futuras e implementaciones a llevar a cabo es larga, pero el tiempo de trabajo es bastante limitado. Intentar llevar la aplicación a las plataformas móviles, generar comparativas con otros mercados, acelerar el procesamiento por parte de las herramientas usadas, así como el de la aplicación y mejorar la base de conocimiento del OCR son puntos que en mi opinión sería ideal de llevar a cabo, pero que por falta de tiempo, no se han podido realizar, dejando el tema abierto de forma que permite un estudio más amplio del mismo e incita a mejorar en un futuro mi desarrollo personal.

Summary and Conclusions

I'm going to finish the document asserting that the results i got were much better than expected. I've never worked before with all the points we attend on this project, neither the tools we use to the development of those. However, thanks to this TFG, i can now have a better understanding about how the OCR and the neural networks operate. It's without any doubt a topic that needs a deeper study because it works with technologies we use on daily base. We can get a lot of new possibilities, just by thinking about how to use neural networks and detection systems to do automatic tasks.

If i talk about my personal experience on this project, i can confirm i have now a nice enough level on the character detection field but i can guarantee there is still a lot of points that with some work on it i can improve even more.

Now, about the development of this FGP (Final Grade Project) and retaking what i said first, the obtained results have been quite favorable. All the analysis generated with our knowledge base were good enough even with images that are remarkable different from those we use to train the OCR system. Also, i can now use all the information that i have obtained during the work i did to create the application and to connect it to the server on future tasks i may get.

And before ending i need to remember that there is a lot of possibles improvements such as export the application to mobile platform, perform analysis taking other markets to compare them one each other, improve the speed of the tools used on this project and to train even further the knowledge base for the OCR system but they were impossible to implement due to the lack of time. Anyways, the topic is still open to be treated and it allows us to research more of it on the nearly future to ensure my own personal development.

Presupuesto

En lo referente al presupuesto, se ha empleado en cada apartado, software libre, lo que reduce considerablemente el precio final necesario para llevar a cabo este proyecto. Una estimación de los elementos necesarios para el completo desarrollo de lo ya explicado sería:

Item	Descripción	Valor por unidad	Unidades	Precio Final
Servidor básico	Necesario para la implementación del apartado del Servidor Linux	Aproximadamente: 300 - 400 € Lo estimaremos en: 350€ (1)	1	~350€
Software y herramientas usadas	Ocropus/Ocropy (Python), Eclipse IDE (Java), JSoup, Commons-codec, ...	0€ (Software Libre)	1	0€
Trabajo personal	Valoración del tiempo personal empleado durante el desarrollo del proyecto	15€ por cada hora de trabajo	> 500 (*)	~7500€

Total:	~7850€

(1) Es una estimación de lo que costaría un servidor básico, sin embargo, gracias a la Universidad de La Laguna, se nos proporcionó una máquina virtual en el IAAS que cumple dicho trabajo sin suponer gasto alguno.

(*) Para este apartado, se tiene en cuenta el tiempo que se tomó en entrenar la base de conocimiento para el sistema de detección, así como el periodo que comprende el aprender a usar correctamente todas las herramientas, así como su funcionamiento.

Bibliografía

Información Ocropus OCR - https://en.wikipedia.org/wiki/OCRopus
Repositorio Ocropus OCR - https://github.com/tmbdev/ocropy
Redes LSTM - https://en.wikipedia.org/wiki/Long_short-term_memory
Python - https://www.python.org/
Herramienta Pip (Python) - https://pypi.org/project/pip/
ImageMagick - https://www.imagemagick.org/script/index.php
Cloud9 (Usado como entorno de pruebas) - https://c9.io/login
IAAS ULL (Sólo si estás conectado a la red de la ULL) - http://iaas.ull.es/
JRE/JDK Java - http://www.oracle.com/technetwork/es/java/javase/downloads/
Información Maven - https://en.wikipedia.org/wiki/Apache_Maven
Maven - https://maven.apache.org/
Eclipse IDE - https://www.eclipse.org/downloads/
StackOverflow - https://es.stackoverflow.com/ (Consultas/soluciones de algunos errores)
Información Apache Derby DB - https://es.wikipedia.org/wiki/Apache_Derby
Apache Derby DB - https://db.apache.org/derby/
MVN Repository - https://mvnrepository.com/ (Descarga y dependencias de las herramientas usadas)
Algoritmo de encriptación MD5 - https://es.wikipedia.org/wiki/MD5
Cifrado AES - https://es.wikipedia.org/wiki/Advanced_Encryption_Standard
SFTP - https://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol
SSH - https://en.wikipedia.org/wiki/Secure_Shell
JSch - http://www.jcraft.com/jsch/
JSoup - https://jsoup.org/
Commons-codec - https://commons.apache.org/proper/commons-codec/
SonarQube - https://www.sonarqube.org/
SonarLint - https://www.sonarlint.org/
Información JUnit - https://es.wikipedia.org/wiki/JUnit
JUnit - https://junit.org/junit5/

Repositorio GitHub: https://github.com/alu0100818130/TFGULL