Sistema de creación de tutoriales interactivos para el aprendizaje de lenguajes de programación

Rafael Caturla Torrecilla
Carlos Congosto Sandoval

Trabajo de fin de grado del Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Director: Enrique Martín Martín
Codirector: Manuel Montenegro Montes

AUTORIZACIÓN PARA LA DIFUSIÓN DEL TRABAJO FIN DE GRADO Y SU DEPÓSITO EN EL REPOSITORIO INSTITUCIONAL E-PRINTS COMPLUTENSE

Los abajo firmantes, alumnos y tutores del Trabajo Fin de Grado (TFG) en el Grado en Ingeniería Informática de la Facultad de informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

TÍTULO del TFG: Sistema de creación de tutoriales interactivos para el aprendizaje de lenguajes de programación

Curso académico: 2015 / 2016

Nombre del Alumnos:

Rafael Caturla Torrecilla

Carlos Congosto Sandoval

Tutores del TFG y departamento al que pertenecen:

Enrique Martín Martín

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación

Firma del alumno/s Firma del tutor/es

Índice

Resumen

En estos últimos años la curiosidad por el sector informático, y más específicamente en la programación, ha producido una gran demanda por los interesados en estas áreas. Esto ha provocado que el número de tutoriales y entrenadores haya aumentado de una forma considerable.

Debido a esta gran demanda, ha surgido la idea de desarrollar una herramienta con la que poder gestionar tutoriales de programación con una aplicación de escritorio. La finalidad de este proyecto es poder crear tutoriales interactivos que permitan a los alumnos adquirir conocimientos de una forma sencilla y atractiva. Esto se consigue mediante la posibilidad de enriquecer el formato del texto al mostrar las explicaciones o los enunciados de las preguntas y añadir imágenes.

Para evitar que sea un tutorial de sólo lectura, que resulta poco atractivo y no permite al estudiante evaluar los conocimientos adquiridos, con esta herramienta se podrán intercalar preguntas entre la teoría para enganchar más al alumno, y conseguir que el aprendizaje sea muy incremental, para no tener una cantidad abrumadora de información en poco tiempo.

Con estos tutoriales el alumno podrá comprobar los conocimientos adquiridos gracias al seguimiento continuo. El estudiante, según lea la teoría y realice los ejercicios con grado de dificultad creciente, tendrá más capacidad para resolver ejercicios nuevos y prepararse para continuar el tutorial.

El resultado de este proyecto ha sido una herramienta multiplataforma y con licencia abierta MIT, que puede ser descargada de https://github.com/Kherdu/TFG

Palabras clave: tutoriales interactivos, lenguajes de programación, herramienta de aprendizaje, YAML, Python, ANTLR

1. Introducción

La necesidad de programadores está creciendo exponencialmente en los últimos años por el gran aumento de uso de todo tipo de dispositivos que necesitan ser programados para las necesidades de la sociedad, especialmente aplicaciones para teléfonos móviles, *tablets* y páginas web. De ahí que cada vez haya más personas interesadas en cursar estudios relacionados con la computación, tanto *software* como *hardware*. Esto nos lleva al mundo del aprendizaje electrónico, más conocido como *e-learning*.

Tradicionalmente, la programación se ha aprendido como cualquier otra materia. Por un lado, el autoaprendizaje implicaba conseguir un manual del lenguaje, y una máquina con un editor de texto y un compilador para experimentar. Si, por el contrario, se tenía la suerte de que un profesor pudiera orientar al alumno, las herramientas más habituales eran la pizarra, tizas, lápiz y papel, para más tarde pasar a un ordenador para probar los conocimientos adquiridos. Ahora, la cosa es bastante diferente de como se planteaba.

Cada día aparecen nuevas formas de aprendizaje de los distintos lenguajes de programación. En la web podemos encontrar diversos tipos de tutoriales y entrenadores de muchos de estos lenguajes, sobre todo de los más modernos y populares hoy día, así como de los que más demanda de programadores hay. Se podría decir que se está cambiando el modelo de aprendizaje a un ritmo considerable.

Si nos fijamos en los cambios en la forma de aprendizaje hablando de programación, en estos últimos años se observa que hay una retroalimentación. A medida que se modernizan los métodos de aprendizaje es más fácil aprender, y más gente que sabe, que a su vez desarrolla nuevas herramientas o mejora las que hay.

El equipo que ha desarrollado esta aplicación como Trabajo Fin de Grado queremos aportar nuestro granito de arena en este campo. Por ello, el objetivo de esta herramienta es crear un sistema de tutoriales interactivos de lenguajes de programación, que sirva de ayuda tanto a los profesores de programación con su asignatura, ya sea en el colegio, instituto, o universidad, como a las personas autodidactas que quieran introducirse al mundo de la programación o a un lenguaje nuevo.

1.1. Antecedentes

Hay multitud de herramientas de ayuda al aprendizaje de lenguajes de programación. Aquí presentaremos algunas de ellas, más o menos conocidas, pero todas ellas relevantes.

Aunque seguramente nos dejemos algunas en el tintero, con esta exposición preliminar, fruto de la investigación, daremos una visión general del panorama actual.

Un ejemplo de herramientas de este tipo son las páginas web con entrenadores del lenguaje en el que se quiere aprender a programar. Con este tipo de herramientas, las personas interesadas tienen a su disposición un campo de texto en el que pueden escribir pequeños fragmentos de código relacionados con lo explicado en el tutorial y comprobar si su funcionamiento final es el deseado. Con estas características hay diferentes webs. Un ejemplo es la familia de páginas web "learn" (http://www.learnpython.org, http://www.learnjavaonline.org, http://www.learn-c.org, http://www.learn-js.org, http://www.learn-php.org, http://www.learnshell.org, http://www.learncs.org), que permiten ver ejemplos de códigos ya generados por el tutorial y modificarlos en el campo de código para hacer pruebas personales y ver cómo varía la salida en función de los cambios realizados.

Otra web muy parecida es www.w3schools.com, en la que se ofrece la posibilidad de estudiar distintos lenguajes de programación orientados a la web como JavaScript, HTML, CSS o PHP. Esta página está dividida en distintos apartados que permiten acceder de una forma directa a aquel en el que se está interesado, consiguiendo así una mayor fluidez a la hora de avanzar en temas específicos sobre un lenguaje de programación. También incorpora un entrenador en el que se puede modificar el código para que el usuario pueda realizar pruebas.

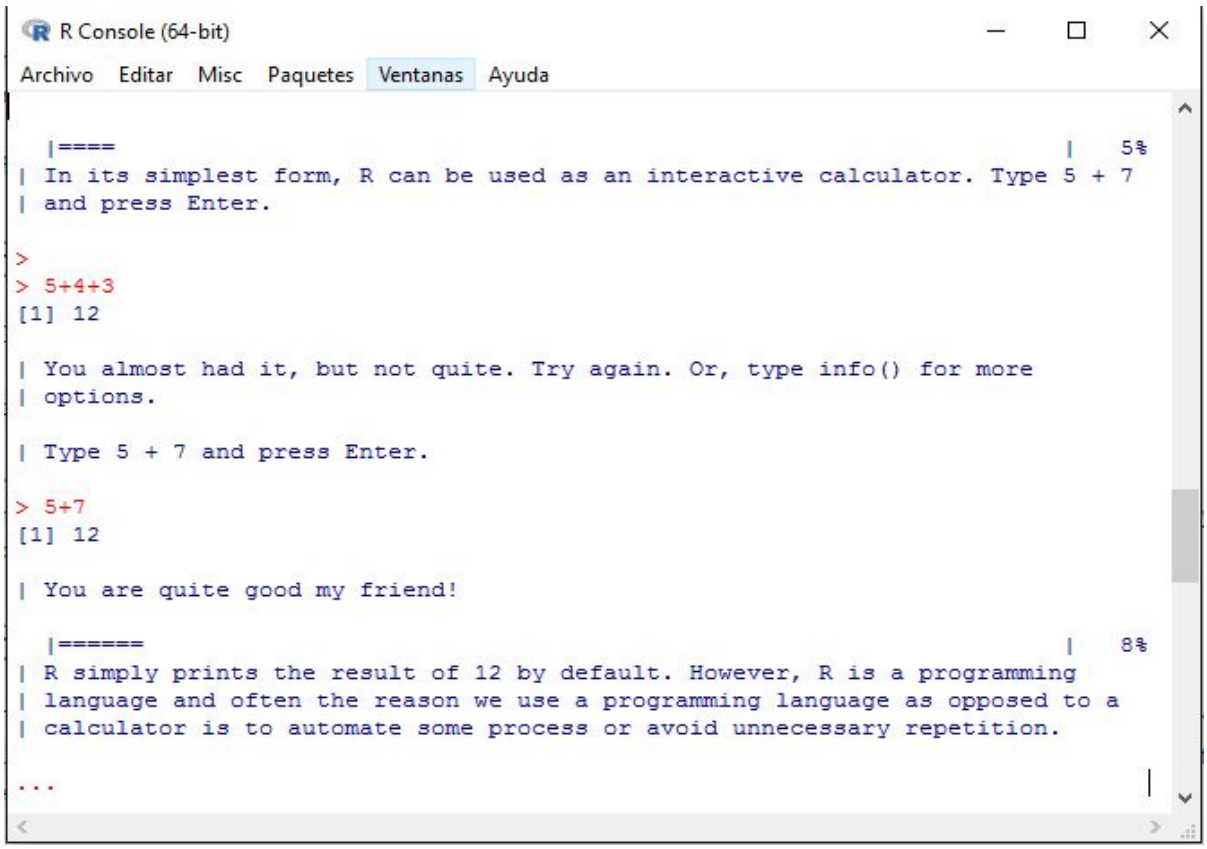
Si queremos un método de aprendizaje más dinámico tenemos videotutoriales, por ejemplo los presentados en http://unity3d.com/es/learn. Aquí se hace una demostración con videos y pequeñas explicaciones textuales sobre cómo desarrollar un proyecto desde cero con el motor de videojuegos multiplataforma Unity [1], mostrando en todo momento los pasos a seguir y los cambios producidos por las modificaciones realizadas en los elementos del proyecto.

A día de hoy se está estudiando la posibilidad de introducir la programación durante el período escolar como una asignatura más. Por eso se están creando "juegos" cuyos controles son sentencias de algún lenguaje de programación. Algunos ejemplos pueden ser la web https://code.org con tutoriales dirigidos a niños, en los cuales las sentencias ya vienen escritas y lo único que hay que hacer es ordenarlas arrastrándolas con el ratón. En el caso de que ya se haya pasado la etapa infantil, existe otra web, https://scratch.mit.edu, del mismo estilo en el que se pueden crear animaciones con distintos personajes conocidos o incluso programar una placa Arduino [2]. El uso de los comandos es similar al anterior.

Para público un poco más adulto, y si se prefieren los juegos de rol y estrategia, en la web www.codecombat.com se puede conducir a un guerrero a través de distintos mapas con sentencias de código en Python (lenguaje por defecto, aunque hay 6 disponibles).

En el contexto de tutoriales como aplicaciones de escritorio, existe Swirl[1], una herramienta modular creada para la realización de tutoriales interactivos del lenguaje R [3]. Este tutorial se ejecuta sobre un intérprete propio. Aquí se refleja tanto la información teórica como las preguntas en texto plano. La teoría se divide en varios fragmentos y para avanzar entre ellos es necesario pulsar la tecla INTRO. Según se avanza a lo largo del tutorial, van apareciendo preguntas a las que el usuario ha de contestar correctamente para avanzar.

Las preguntas a contestar son de tipo test o de escritura de código, estas últimas corregidas mediante funciones escritas en R, con el inconveniente de que hay que introducirlo tal cual se pide en el enunciado para poder seguir avanzando. Esto se puede ver a continuación:



En este ejemplo vemos que en el enunciado se pide introducir la suma 5 + 7, y el usuario introduce 5 + 4 + 3. La aplicación no deja continuar hasta que no se introduce la operación tal y como se muestra en el enunciado.

Otras herramientas web utilizadas por distintas universidades son *¡Acepta el reto!* https://www.aceptaelreto.com en la Universidad Complutense de Madrid (UCM) y *UVa Online Judge* https://uva.onlinejudge.org en la Universidad de Valladolid (UVa). En estas

[1] http://swirlstats.com

webs se propone a los alumnos distintos tipos de problemas de programación generados por los propios profesores de cada organismo. Así los alumnos pueden comprobar sus conocimientos de programación y los profesores ver dicho trabajo y orientar mejor sus clases, en función del progreso observado.

De libre acceso, pero de similar utilidad, existe *FLOP* http://problem-g.estad.ucm.es/FLOP/index.jsp. Aquí los estudiantes pueden subir la solución a los problemas planteados y comprobar si es correcta. Como utilidad para profesores, existe un método de generación de colecciones de ejercicios.

Tras probar Swirl y ver cómo funcionan el resto de herramientas *online*, se pensó en crear una aplicación de escritorio para evitar problemas como la necesidad de conexión a internet, y los derivados de tener ejecutores de código externo en un servidor de internet. Suponiendo que se pudiese construir algo similar, pero más completo y amigable tenemos una idea de la que partir.

Esta nueva herramienta debería ser capaz de solventar los problemas de Swirl como puedan ser la configuración previa cada vez que se ejecuta de nuevo y la imposibilidad de adaptación a otros lenguajes.

1.2. Objetivos

El objetivo del presente Trabajo Fin de Grado es desarrollar una herramienta gratuita y abierta para la gestión de tutoriales interactivos de lenguajes de programación, con el fin de ayudar a alumnos y profesores de todos los niveles de educación con las distintas asignaturas de programación. Consistirá en una aplicación de escritorio que permita la realización de tutoriales interactivos con las siguientes características:

- Presentación de tutoriales cortos y dinámicos, que resulten atractivos a los alumnos mediante la aparición de elementos visuales que faciliten su lectura.
- Posibilidad de crear y añadir dichos tutoriales de forma independiente a la aplicación.
- Soporte para varios tipos de pregunta diferentes, en principio tipo test, de código y de sintaxis (explicados posteriormente).
- Soporte de lenguajes interpretados[2], teniendo en mente Python como punto de partida. La aplicación debe admitir cualquier lenguaje interpretado sin necesidad de modificar la aplicación.
- Funcionamiento independiente del sistema operativo.

[2] Se entiende por lenguajes interpretados aquellos que no necesitan una fase de compilación para producir código máquina, sino que son ejecutados por un intérprete.

Los tutoriales de cada lenguaje estarán divididos en varios temas, y estos a su vez en distintas lecciones. Con esta organización se pretende que la información esté compartimentada y estructurada en relación a su contenido. Aunque se podrán añadir tutoriales de cualquier extensión, se pretende que las lecciones sean breves y se puedan completar en un periodo de tiempo corto. Las lecciones constarán de pequeñas explicaciones teóricas que se intercalan con preguntas. Estas últimas deben resolverse para completar la lección.

El hecho de que los tutoriales sean independientes a la aplicación permitirá que los profesores puedan crear tutoriales personalizados para sus alumnos siguiendo un estándar definido en este mismo proyecto.

A continuación se explican los distintos tipos de pregunta más en profundidad:

- **Tipo test:** preguntas cuyo cometido es que el alumno responda a la pregunta marcando una o varias respuestas propuestas. Las preguntas podrán ser de respuesta única o respuesta múltiple.
- **Tipo código**: preguntas en las que se comprueba que el código introducido por el alumno genera la salida esperada por el profesor. Se ejecutará el código del alumno y comprobará la salida de este.
- **Tipo sintaxis**: en éstas preguntas, a diferencia de las anteriores, se comprueba que el fragmento de código introducido por el alumno cumple la estructura deseada por el profesor. Otra diferencia a destacar, es que el código no se ejecuta en ningún momento, por lo que tampoco se comprueba la salida producida. La estructura se define independientemente mediante una gramática que será reconocida por un intérprete (en el apartado 2.3.2 se explicará cómo se hace).

Con esta variedad de preguntas, el alumno podrá comprobar y afianzar los conocimientos explicados en la teoría que haya leído recientemente. Para poder avanzar con una lección es necesario responder correctamente a la pregunta formulada. Se podrá volver atrás para consultar la teoría de la propia lección si se necesita.

De la forma en la que se ha construido la aplicación, se podrá añadir un tutorial de cualquier lenguaje interpretado. Con esta característica buscamos que la herramienta sea más flexible, completa, y se ajuste a las necesidades de un mayor número de personas.

1.3. Plan de trabajo

En esta sección se hablará de todos los aspectos relacionados con el desarrollo de la aplicación en distintos apartados, siendo estos metodología de trabajo y planificación y fases de desarrollo.

1.3.1. Metodología de trabajo y planificación

La metodología usada en este proyecto ha sido Scrum [4]. Se eligió por ser un método de desarrollo incremental, dinámico y flexible, diseñado para proyectos cuyos requisitos están expuestos a cambios, y por ello se pueden producir solapamientos en las distintas etapas de desarrollo. Además, está pensada para pequeños grupos de desarrollo como es el caso de este equipo.

Para poder llevar a cabo un seguimiento y poder incrementar la funcionalidad paulatinamente, en primer lugar se hizo una reunión donde se habló de qué y cómo se quería como producto final y, cada 2 ó 3 semanas, se fueron realizando reuniones en las que se revisaba el trabajo hecho desde la reunión anterior, se fijaban unos objetivos o hitos a cumplir dentro de lo posible para la siguiente reunión y se analizaban posibles modificaciones sobre lo ya realizado para mejorar la implementación. En período de exámenes se dejaba un margen mayor entre reuniones con el fin de tener tiempo de estudiar.

En cada reunión se ha tomado acta de lo tratado, estas actas se encuentran en el Anexo A.

1.3.2. Fases de desarrollo

Estas son las diferentes fases de desarrollo por las que se ha pasado, presentadas en forma esquemática y con sus fechas correspondientes:

1.- Diseño de los ficheros que definen los tutoriales (6/10/2015 - 3/11/2015)

- Decisión del lenguaje utilizado para su representación (YAML).
- Creación de ficheros de ejemplo con los campos necesarios para mostrar toda la información necesaria.
- Inclusión de marcadores Markdown a los campos de texto de los ficheros.
- Generación de métodos necesarios para interpretar la información.

2.- Creación de las vistas de la aplicación (3/11/2015 - 14/1/2016)

- Diseño de las ventanas de la aplicación.
- Generación código HTML con la información contenida en los ficheros YAML.
- Muestra de información obtenida de archivos YAML en las ventanas.

3.- Incorporación de las preguntas tipo test (17/11/2015 - 1/12/2015)

- Recogida de las respuestas que están marcadas por el usuario.

- Implementación de la función correctora.
- Actualización de la ventana con el resultado.

4.- Incorporación de las preguntas cuya corrección requiere ejecución en el lenguaje destino (14/1/2016 - 17/03/2016)

- Decisión de ejecutar el código mediante Python.
- Creación del fichero de corrección de Python.
- Definición del formato y contenido del fichero temporal donde se guardarán los resultados de la ejecución.
- Implementación de las funciones correctoras en Python.
- Actualización de la ventana con la información del resultado.

5.- Incorporación de preguntas corregidas por analizadores sintácticos (19/04/2016 - 27/05/2016)

- Elección del analizador sintáctico que se utilizará en este tipo de preguntas.
- Generación de gramáticas de prueba.
- Generación de las clases necesarias para la corrección.
- Implementación de la función correctora.
- Actualización de la ventana con el resultado de corrección.

6.- Creación del archivo de preferencias de usuario para mantener la configuración. (26/04/2016 - 27/05/2016)

- Búsqueda de información sobre almacenamiento de variables independientes a la aplicación.
- Decisión de usar variables de sistema frente a ficheros de configuración.
- Implementación de métodos de guardado y carga de dichas variables.
- Enlazado con la interfaz de la aplicación y prueba de funcionamiento.

Al finalizar cada fase del proyecto se comprobaba que todo funcionase correctamente y en caso necesario, los resultados de las fases podrían sufrir modificaciones posteriormente según las necesidades según se avanzaba en el desarrollo.

Los periodos de tiempo que se encuentran entre algunas fases se han dedicado a solucionar algunos fallos que se encontraban al acabar dichas fases y reestructurar los elementos de la aplicación.

1.4. Herramientas de desarrollo utilizadas

Se han utilizado múltiples herramientas para construir la aplicación y lo que la rodea, cada una de ellas teniendo un propósito específico y, en conjunto, han aportado comodidad a la hora de avanzar en el desarrollo de la aplicación y documentar los avances tras cada paso.

La herramienta de la que más se ha dependido para el desarrollo de la aplicación ha sido el entorno de desarrollo Eclipse [5] en su versión 4.5, con diversos *plug-ins* para facilitar la escritura de los archivos externos YAML[3] y la ejecución de código Python (ver más adelante). No entraremos mucho más en dichos *plug-ins*, pues tienen una relevancia menor en el proyecto.

Otra herramienta fundamental fue GitHub[4], que utilizamos como sistema de control de versiones para el código. Esta herramienta se presenta con más detalles en la sección 2.2.9.

Por último para las actas de reuniones, el intercambio de archivos de texto, almacenamiento de enlaces y citas de bibliografía relevantes se ha utilizado Google Docs aprovechando las cuentas de usuario provistas por la Universidad Complutense.

[3] La información acerca de esta tecnología se explicará en el apartado 2.2.3
[4] La información acerca de esta tecnología se explicará en el apartado 2.2.9

1.B. Introduction

The needs of programmers have been growing exponentially in the last few years because of the great increase of use of all kind of devices that needs to be programmed to fit the needs of society, specially smartphone applications, tablets and web pages. Hence every time there are more people interested in studies related to computer science, both software and hardware. This brings us to the world of e-learning.

Traditionally, programming has been studied as any other subject. On one side, self-learning implied getting a language manual, and a machine with a text editor and a compiler to experiment with. If, conversely, it was fortunate that a teacher could guide the pupil, then, the most common tools were the blackboard, chalks, pen and paper, to later go to a computer to test the knowledge acquired. Nowadays, the subject is quite different than then.

Everyday new ways of learning programming languages appear. In the web we can find various types of tutorials and trainers for many of these languages, above all (specially) for the most modern and popular ones, as well as the ones that have the highest professional demand. We could say that the model of learning is changing at considerable pace.

If we look at the changes in the learning techniques regarding programming, in the recent years it is noted that there is a feedback. As learning methods are modernized it's easier to learn, so there is more people who knows and, at the same time, they develop new tools or improve the existing ones.

The team who has developed this application as End of Degree project wants to contribute to leave our mark in this field. Thus, the objective of this tool it is to create an interactive programming language tutorial tool, which can be helpful for both teachers with their programming classes, either in school, highschool or college(university) as to help self-taught people who want to introduce themselves to the world of programming or new languages.

1.1.B. Background

There are plenty of tools to help learning programming languages. Here we shall introduce some of them, more or less known, but all of them relevant. Although we surely leave some in the pipeline, with this a preliminary statement, result of research, we will give an overview of the current situation.

A few examples of this kind of tools are the web pages with trainers for the language that we want to learn. With this kind of tools, interested people have at their disposal a text field in which they can type little code snippets related to the matter explained in the tutorial and check if its behaviour is the one they desired. With these characteristics there are different webs. An example is the "learn" family webs (http://www.learnpython.org, http://www.learnjavaonline.org, http://www.learn-c.org, http://www.learn-js.org, http://www.learn-php.org, http://www.learnshell.org, http://www.learncs.org), which allow to see code examples generated by the tutorial and modify them in the code text field to test it and see how the output changes depending on the input field variations.

Another similar web is www.w3schools.com, where It's offered the possibility to study different web oriented programming languages like JavaScript, HTML, CSS or PHP. This page is divided in different sections that allows direct access to the one we are interested in, thus achieving greater fluency when we try to advance to specific subjects about a language. It also integrates a trainer where the code can be modified so the user can test it.

If we want a more dynamic learning method we have video-tutorials, for example the ones presented in http://unity3d.com/es/learn. Here demonstrations are made through videos and brief texts that explain how to develop a project from scratch with the multiplatform video game engine Unity[1], showing constantly the steps to follow and the changes produced by the modifications done in project elements.

Nowadays it's being studied the possibility to introduce programming during the scholar period as a new class . Because of that, games are being developed whose controls are similar to programming language sentences. Some examples could be the web https://code.org with tutorials are directed toward children, where the sentences are already written, and the only thing to do is to sort them by dragging them with the mouse. For older children exists another web, https://scratch.mit.edu, which has a similar style, where animations can be created of different known characters, anybody could program an Arduino[2] board. The command usage is similar to the previous one.

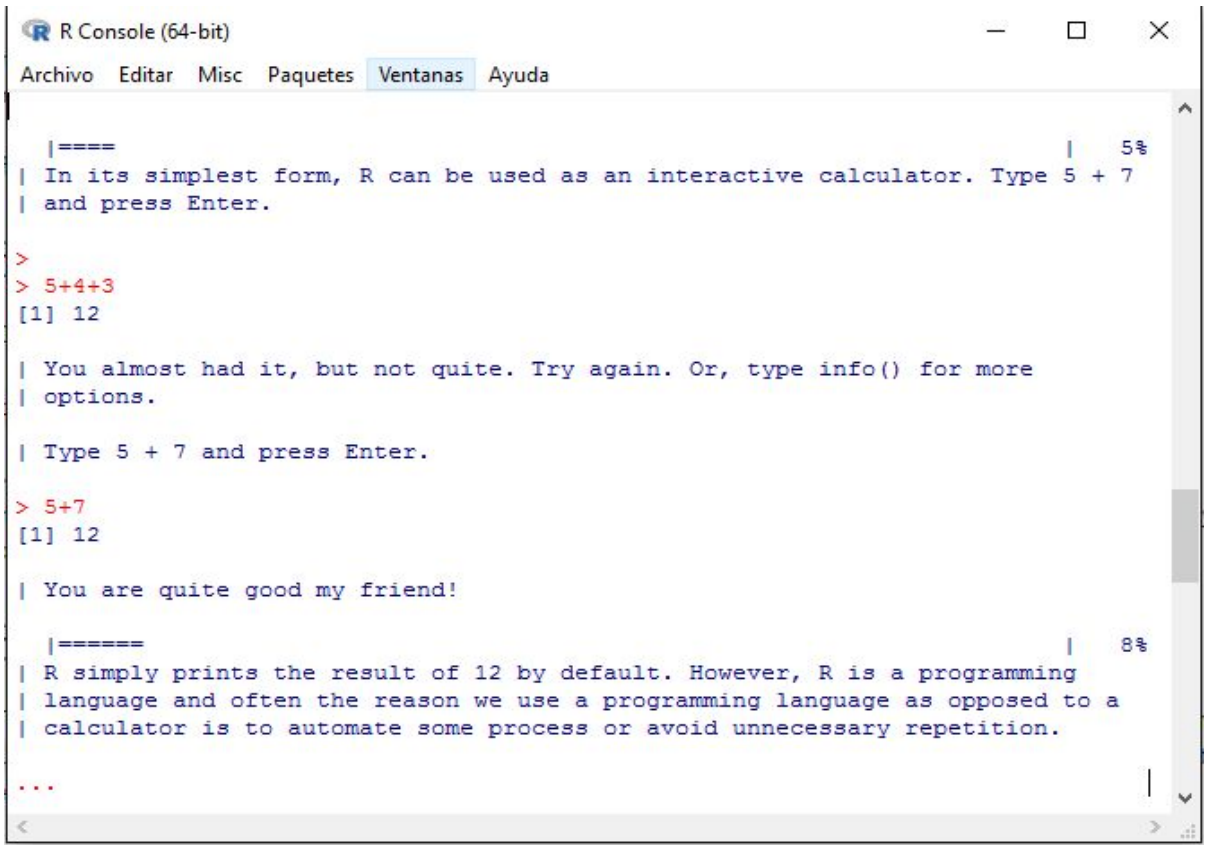
For a public a little bit older, that prefers roleplaying and strategy games, in www.codecombat.com you can guide a warrior through different maps with Python code sentences (it's one of six avaiable languages).

In the context of desktop application tutorials, we have Swirl[5], a modular tool designed to present interactive of R[3] language tutorials. This application it's executed through a non independent interpreter. In that interpreter it's displayed both the theory explanation and the questions as plain text. The theory part it's divided in some subsections,

[5] http://swirlstats.com

to advance through them it's needed to press the ENTER key. As the tutorial advances, questions appear that the user has to answer correctly.

The mentioned questions are test type or code writing type, the last ones are manually corrected with R[3] written functions, with the inconvenience that you have to introduce them exactly as the statement requires to be able to advance further. We can see that in this image.



In this example we can see that in the statement it's asked to introduce the addition 5 + 7, and the user introduces 5 + 4 + 3. The application doesn't allow to continue unless you introduce the operation exactly as the statement asks.

Other web tools used by different universities are *¡Acepta el reto!* https://www.aceptaelreto.com in the Complutense University of Madrid (UCM) and *UVa Online Judge* https://uva.onlinejudge.org in the Universidad of Valladolid (UVa). These webs, proposes the students different types of programming problems generated by the own teachers of the organization. That way, the students can try out their programming knowledge and the teachers can see that work and direct better their classes according to that progress.

A similar utiliy free alternative is FLOP http://problem-g.estad.ucm.es/FLOP/index.jsp. Here the students can upload their solutions to

the proposed problems and check if it's correct. And as teachers utility it exist a method to generate sets of exercises.

After trying out Swirl and see how the rest of the online tools work, we thought about creating a desktop application to evade issues like internet connection necessity, and the ones derivatives from running external code inside an internet server. Supposing something similar could be built, but more complete and friendly. We have an idea to start from.

This new tool should be able to solve Swirl issues like having to configure the program each time we have to launch it and the impossibility to adapt to other languages.

1.2.B. Targets

The main target of this End of Degree Project is to develop a free open source tool to manage programming languages interactive tutorials, with the objective to help students and teachers across any educational level with their different programming subjects. It will consist on a desktop application that allows the realization of interactive tutorials with the following features:

- Short, dynamic tutorials, attractive to students by visual elements that ease their lecture.
- The possibility to create and add those tutorials independently of the application
- Different question type support, firstly test, code and syntax type (later explained)
- Interpreted languages[6] support, having Python in mind as a start point. The application should be able to admit any interpreted language without the needs of any modification in the application
- Operative System independent usability .

Each language tutorials is divided in some subject-matters, and these at the same time are divided in different lessons. This is intended to compartmentalize information and structure it according to its content. Although tutorials with any extension could be added, it is expected that lessons are brief to be completed in a short period of time. The lessons will feature little theory explanations which are interleaved with questions. The latter must be resolved to complete the lesson.

[6] Interpreted languages understood as those that do not need a compilation phase to produce machine code, but are executed by an interpreter.

The fact that tutorials are independent from the application will allow the teachers to create them personalized to their students by following a standard which is defined in this project.

Now the different types of question are explained more in depth:

- **Test type:** questions whose task is to make the student to respond by marking one or more proposed answers. Questions can be multiple-answer or single-answer.
- **Code type**: questions which the code written by the student generates the expected output according to the teacher. The code will be executed and the output will be compared.
- **Syntax type**: these questions, unlike the others, check that the code fragment introduced satisfy the structure the teacher wants. Another difference to emphasize is that the code will not be executed in any time, so the exit will not be checked. The structure will be defined independently through a grammar that will be recognized by the interpreter (In section 2.3.2 will be explained how it's done).

With this variety of questions, the student will be able to check and demonstrate everything explained in the theory he has recently read. To advance with a lesson the user is expected to answer to the raised question correctly. You can also go backwards to consult the theory of the current lesson if needed.

1.3.B. Work plan

In this section we will discuss all the aspects related to the development of the application in various sections, those being: work methodology and planning, development phases and development tools used.

1.3.1.B. Work methodology and planning

In this project the work methodology used has been Scrum[4]. It was selected because it is an incremental, dynamic and flexible method, designed for projects whose requirements are exposed to changes and in which overlaps could happen between the different development phases. Also it is designed for little groups, which is the case of this team.

To keep track and increase gradually the functionality, in first instance a meeting was made where we talked what and how the final product would be and, each 2 or 3 weeks, meetings were made where it was checked the progress from the previous meeting, new targets were locked to accomplish as far as possible for the next meeting and potential

modifications were analysed to improve implementation. At exams period a wide margin between those meetings were made to have enough time to study.

A record of each meeting was saved. This information is in the Annex A.

1.3.2.B. Development phases

These are the different stages of development, schematically presented and with their corresponding dates.

1.- Design of the files that define the tutorials (6/10/2015 - 3/11/2015)

- Selection of the language that represents it (YAML).
- File creation with examples with the necessary fields to hold all the information needed.
- Markdown markers inclusion in the file text fields.
- Generation of the needed methods to read the information.

2.- Creation of the application views (3/11/2015 - 14/1/2016)

- Window design.
- HTML code generation with the information contained in the YAML files.
- Display in the windows of information obtained from YAML files.

3.- Test questions incorporation (17/11/2015 - 1/12/2015)

- Capture of the marked user answers.
- Corrector function implementation.
- Window refresh indicating result.

4.- Incorporation of questions which correction requires execution in destination language (14/1/2016 - 17/03/2016)

- Decision of executing the code through Python.
- Python correction file creation.
- Format definition and content of the temporal file where the execution data will be stored.
- Python corrector functions implementation
- Window refresh indicating result.

5.- Incorporation of questions corrected by syntax analyzers (19/04/2016 - 27/05/2016)

- Selection of the syntax analyzer to be used in this type of questions.
- Test grammars generation.
- Correction needed classes generation.

- Corrector function implementation.
- Window refresh indicating result.

6.- Preferences file creation to save user configuration (26/04/2016 - 27/05/2016)
- Information search about application independent variables storage.
- Decision to use system variables versus configuration files.
- System variables save and load methods implementation.
- Interface linked and operation test.

At the end of each project phase it was checked that everything operated as expected and, if necessary, the results of previous phases could be later modified according to the needs as the development as it advanced.

The time periods between some of these phases have been used to solve some problems that were found when they were finished and to restructure the application elements.

1.4.B Development tools used

Multiple tools have been used to build the application and everything surrounding it. Each one of them had a specific purpose and, as a pack, they have contributed to ease the development and documentation during and following each step done.

The tool which we depended the most for the development was the integrated development environment Eclipse[5] in its version 4.5 with diverse plugins to help the writing of external YAML[7] files and the Python execution (see ahead). No more information about this will be provided about these plugins because they have less relevance in the project.

Another fundamental tool was GitHub[8], which was used as version control system for the code.

Ultimately, for the meeting records, text files exchange, link storage and bibliographic relevant citations we have used Google Docs, taking advantage of the accounts provided by the Complutense University.

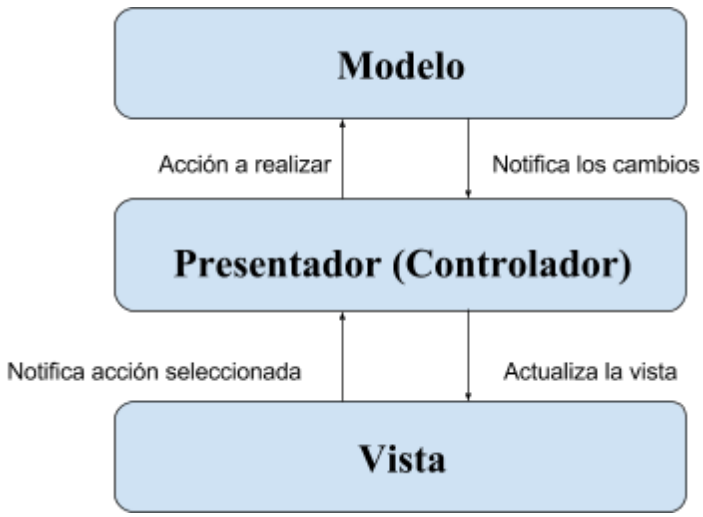
[7] Information about this technology in section 2.2.3
[8] Information about this technology in section 2.2.9

2. Diseño de la aplicación

En este apartado se explicará en profundidad cómo se ha construido la aplicación, su funcionamiento y las tecnologías y herramientas utilizadas para su desarrollo.

2.1. Estructura de clases

En el diseño de esta aplicación se ha utilizado una variación del Modelo-Vista-Controlador, en concreto Modelo-Vista-Presentador, con el fin de facilitar la ampliación de posibles nuevas funcionalidades de una forma muy sencilla y casi automática y a su vez complementandose con la metodología Scrum. La implementación se divide en distintos paquetes que constituyen las 3 partes de este patrón.

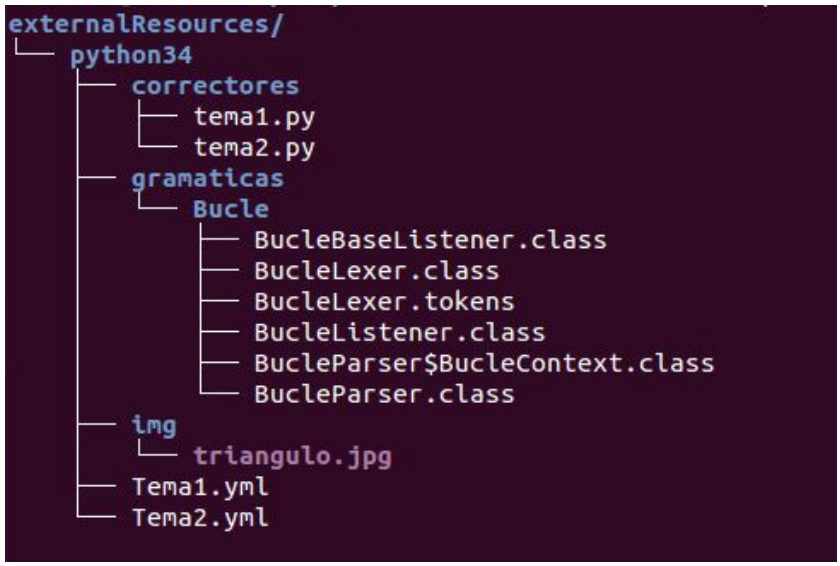


En este esquema se puede ver claramente cómo el `Modelo` no notifica directamente los cambios a la `Vista` como en el Modelo-Vista-Controlador. Para llevar a cabo dicha función los cambios realizados en el Modelo se notifican a la Vista a través del `Controlador`, que en este caso se denomina `Presentador`.

En el paquete `modelo` se encuentra la información a tratar por la aplicación. En el paquete `vista` se encuentran todas las clases correspondientes a las distintas interfaces gráficas de la aplicación para mostrar la información al usuario. Por último, en el `presentador`, se encuentra la única clase controlador de la aplicación. Esta clase es la encargada de establecer una conexión entre ambas partes permitiendo mostrar la información del modelo en la vista.

Aparte de lo mencionado, se ha creado un paquete adicional llamado `Utilities`, en el cual se encuentran las clases necesarias para cargar la información de los archivos de los tutoriales, el código necesario para generar las vistas con código HTML y obtener la información de las variables de entorno o 'preferencias' del sistema.

En cuanto a elementos externos al código de la aplicación, es necesario un directorio externo donde tendremos los recursos de la aplicación, es decir, los archivos YAML de los tutoriales, las posibles imágenes de enunciados y explicaciones, el código de corrección de las preguntas y las gramáticas compiladas. Este directorio se podrá llamar de cualquier forma. En el ejemplo se nombrará como '*externalResources'* y obligatoriamente tiene la siguiente estructura:



En este ejemplo se puede comprobar que el directorio raíz es `externalResources`, donde se encuentran las carpetas con los lenguajes disponibles. En nuestro caso sólo existe `python34`. Cada carpeta de lenguaje debe contener las carpetas `correctores` y `gramaticas` y los ficheros YAML que contienen la información de los temas. Si además se hace uso de alguna imagen, existirá una carpeta `img` que las contendrá.

En la carpeta `correctores` están los archivos de código de corrección de preguntas, y en la carpeta `gramaticas` tenemos un directorio por cada pregunta de este tipo que tengamos entre todos los temas del lenguaje. En estas carpetas estarán ubicados las clases java generadas con ANTLR precompiladas .

Cabe destacar también el uso de variables de entorno del sistema operativo con Java Preferences API, donde guardaremos la ruta del directorio del que hemos hablado anteriormente, junto con las rutas de los intérpretes del lenguaje.

2.2. Tecnologías utilizadas

En este apartado vamos a hablar sobre todas las tecnologías que hemos utilizado a lo largo del desarrollo de la aplicación, profundizando lo suficiente en cada una de ellas para aportar una comprensión global de por qué, cómo y dónde se utiliza cada una de ellas.

2.2.1. Java [6]:

Es un lenguaje de programación orientado a objetos que permite desarrollar para cualquier plataforma que tenga una Máquina Virtual de Java[9] instalada. Por esta razón y porque los miembros se sienten cómodos con este lenguaje se ha decidido implementar esta aplicación con este lenguaje. Como versión se ha elegido la 1.8, versión más actual del lenguaje que permite nuevas funcionalidades respecto de la versión previa como pueden ser las funciones lambda[10] y nuevas funcionalidades de JavaFX del que se hablará a continuación.

2.2.2. JavaFX [7]:

La interfaz gráfica ha sido implementada mediante tecnología JavaFX. En primer lugar se descartó utilizar la librería gráfica AWT[11] por estar prácticamente obsoleta. Entre JavaFX y Swing se eligió el primero principalmente porque es más moderno, y, aunque Swing[12] aún no está obsoleto, será desechado en un futuro no muy lejano. Además, JavaFX nos proporciona una serie de características muy interesantes para nuestra herramienta:

- Permite una migración a interfaz web bastante sencilla.
- Proporciona un componente que permite mostrar elementos web en código HTML.
- Admite la implementación de animaciones en caso de que sean necesarias.
- Soporta dispositivos de pantalla táctil.
- Los elementos visuales tienen propiedades asociadas. Esto permite un mayor control sobre ellos, como por ejemplo asignar una hoja de estilo CSS[8] para cambiar su aspecto.

En conclusión, los elementos en JavaFX son más manejables y con más posibilidades que las que ofrece Swing.

[9] Más información en la cita [6] de bibliografía
[10] Para más información visitar
http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html
[11] Más información en la cita [6] de bibliografía
[12] Más información en la cita [6] de bibliografía

2.2.3. YAML (YAML Ain't Another Markup Language)[13]:

Es un formato de representación de datos de forma legible para humanos basada en un árbol de pares clave-valor, al que se le pueden introducir diversos tipos de elementos por cada clave, como por ejemplo listas, bloques de texto que preservan retornos de línea y tabulaciones, y *arrays* asociativos del tipo clave-valor. De cara a este proyecto se ha pensado utilizar YAML para la representación de la información de los tutoriales debido a su fácil lectura y escritura en comparación con XML o JSON. Los que más se beneficiarán de esto serán los profesores, pues serán éstos quienes escriban los tutoriales.

Un ejemplo de fichero YAML conteniendo un valor asignado a una clave y una lista de elemento con un valor definido sería:

```
clave: valor
lista:
 - elemento: elemento uno
 - elemento: elemento dos
```

A modo de comparativa, otras formas de representación de datos descartadas debido a su mayor dificultad de leer la información son XML y JSON. Por ejemplo, la información anterior se representaría en XML y JSON de la siguiente manera:

Ejemplo XML:

```
<clave> valor </clave>
<lista>
      <elemento>elemento uno</elemento>
      <elemento>elemento dos</elemento>
</lista>
```

Ejemplo JSON:

```
{
      clave: valor
      lista: [{elemento: "elemento uno"}, {elemento: "elemento dos"}]
}
```

Como conclusión, se decidió que YAML era una manera muy adecuada de representar los contenidos de los tutoriales tanto por su facilidad de escritura como de interpretación y su posibilidad de introducir variedad de contenidos.

[13] http://yaml.org/ , http://symfony.com/legacy/doc/reference/1_3/en/02-YAML

A la hora de incorporar YAML en nuestro proyecto hemos escogido la librería SnakeYAML[14], que permite convertir el texto de un archivo YAML a un árbol clave-valor de objetos Java. Esta librería ha sido elegida por su gran número de usuarios y su mayor cantidad de documentación frente a otras librerías como por ejemplo, Jyaml[15]. Además podrá permitir en el futuro construir los archivos YAML.

2.2.4. Markdown[16]:

Es un lenguaje de marcas ligero para proporcionar estilo a textos, muy conocido por su uso en repositorios como GitHub (ver en la sección 2.2.9). Este lenguaje permite escribir de forma sencilla documentos de texto con elementos de estilo como negritas, títulos, subtítulos, bloques de código y algunos más complejos como tablas y listas. Un ejemplo claro sería:

Ejemplo de Markdown

```
# Nivel 1
## Nivel 2
### Nivel 3
El **comienzo** de la lista

* El primero
* El segundo
* El tercero
    1. 3.1
    1. 3.2

Y esto el final de la lista
First Header | Second Header |        Third Header
:----------- | :------------: | --------------------:
First row    |      Data     | Very long data entry
Second row   |    **Cell**   |               *Cell*
```

Resultado formato HTML (interpretado)

Nivel 1

Nivel 2

Nivel 3

El **comienzo** de la lista

- El primero
- El segundo
- El tercero
    1. 3.1
    2. 3.2

Y esto el final de la lista

First Header	Second Header	Third Header
First row	Data	Very long data entry
Second row	**Cell**	*Cell*

Para incorporar Markdown a nuestra herramienta se ha utilizado la librería pegdown[17]. Esta librería permite generar código HTML a partir de las etiquetas de Markdown contenidas en el texto. La decisión de usar esta librería frente a otras como por ejemplo txtmark[18], es debido a que pegdown ofrece distintas extensiones para generar más elementos HTML. A su

[14] https://github.com/asomov/snakeyaml
[15] http://jyaml.sourceforge.net/
[16] https://guides.github.com/features/mastering-markdown/
[17] https://github.com/sirthias/pegdown
[18] https://github.com/rjeschke/txtmark

vez estas extensiones se pueden gestionar según las necesidades según el uso que se le vaya a dar. Algunos ejemplos de extensión son:

- ALL: añade todas las extensiones
- TABLES: Permite generar tablas HTML
- SUPPRESS_HTML_BLOCKS: Suprime los bloques HTML

Como ejemplo de pegdown, el código Markdown presentado anteriormente se transformaría en su correspondiente HTML:

```
<h1>Nivel 1</h1>
<h2>Nivel 2</h2>
<h3>Nivel 3</h3>
<p>El <strong>comienzo</strong> de la lista</p>
<ul>
	<li>El primero</li>
	<li>El segundo</li>
	<li>El tercero
		<ol>
			<li>3.1</li>
			<li>3.2</li>
		</ol>
	</li>
</ul>
<p>Y esto el final de la lista</p>
<table>
	<thead>
		<tr>
			<th>First Header</th>
			<th>Second Header</th>
			<th>Third Header</th>
		</tr>
	</thead>
	<tr>
		<td>First row</td>
		<td>Data</td>
		<td>Very long data entry</td>
	</tr>
	<tr>
		<td>Second row</td>
		<td><strong>Cell</strong>
		</td><td><em>Cell</em></td>
	</tr>
</table>
```

2.2.5. ANTLR (Another Tool For Language Recognition) [9]:

Es una librería que permite generar e interpretar gramáticas independientes del contexto. Esto servirá para comprobar que el código cumple con ciertos requisitos sintácticos. Para ello se escribe un archivo '.g4' con la gramática deseada y a continuación se compila para obtener las clases Java (`Lexer y Parser`) encargadas del reconocimiento del patrón establecido.

La clase `Lexer` es la encargada de generar los Tokens de reconocimiento a partir de los caracteres de la gramática definida. La clase `Parser` es la que lleva a cabo la comparación.

Un ejemplo sencillo de gramática es la que define la cadena `"hello " + una cadena de sólo letras.`

A continuación se explica cómo definir esta gramática:

```
grammar Hello;
r    : 'hello ' ID ;
ID : [a-z]+ ;
```

2.2.6. Python [10]:

Lenguaje de programación dinámico y sencillo de aprender, con una sintaxis que ayuda a hacer un código legible, multitud de librerías disponibles y un nivel de abstracción alto, lo que consigue que sentencias que en otro lenguaje son largas se puedan realizar en pocas líneas de código. La versión elegida de este lenguaje es la 3.4, que era la más actual y estable cuando comenzó este proyecto. Actualmente la más reciente es la 3.5, pero la diferencia es pequeña y no provoca ningún cambio sustancial en el funcionamiento de esta aplicación. Este lenguaje podría variar en función del lenguaje que se vaya a enseñar con la herramienta. En este caso, como base y por sugerencia de los directores, va dirigida a Python.

2.2.7. JSON (JavaScript Object Notation)[19]:

Es un formato de texto pensado para el intercambio de datos u 'objetos' con un esquema de atributos tipo clave-valor. Además, es un formato independiente del lenguaje que lo vaya a interpretar, y tiene librerías para su uso en casi todos los lenguajes.

[19] http://www.json.org/

Se utiliza para la corrección del tipo de preguntas de 'código', comunicando Java y Python (o cualquier otro lenguaje interpretado que se vaya a utilizar) mediante un archivo temporal de este tipo.

2.2.8. Maven [11]:

Es una herramienta software concebida para construir y gestionar las dependencias de proyectos Java. Con esta herramienta se facilita la importación y el uso de librerías externas. Su característica fundamental es definir la estructura del proyecto con un archivo POM (Project Object Model)[20], donde se encuentran las dependencias y el orden de construcción del proyecto. Se ha utilizado para importar y empaquetar el proyecto con las librerías externas.

2.2.9. GitHub[21]:

Plataforma de desarrollo en paralelo basada en el sistema de control de versiones *open source* git [12], que permite trabajo colaborativo. Esta plataforma se aloja en la nube y es muy popular entre programadores de todo el mundo, lo que permite dar gran difusión a la aplicaciones que aloja. Se ha utilizado debido a su facilidad de uso para el desarrollo en paralelo, permitiendo así a los miembros del equipo trabajar independientemente con el mismo repositorio.

2.3. Definición de archivos externos

Ya se ha explicado cómo se estructura el directorio de archivos externos. En este apartado explicaremos cómo son estos archivos en cuanto a su contenido y estructura.

2.3.1. Archivos YAML

Los archivos donde se encuentra la información de los tutoriales son de tipo YAML, y su extensión es '.yml'. Para que la carga de tutoriales funcione correctamente es necesario que se mantenga la estructura y el nombre de las distintas etiquetas, que explicaremos con un ejemplo.

[20] Explicación en el la cita [11] de la bibliografía
[21] https://github.com/

Primero vamos a explicar la estructura del tema y las lecciones con un ejemplo:

```
Tema: 2
Titulo: "#Listas"
Archivo: "python34/correctores/tema2.py"
Introduccion: |
 ##Python utiliza varios tipos de datos compuestos,
 ##que se utilizan para agrupar otros valores.
 ##El más versátil es la lista.
Lecciones:
 - Leccion: 1
   Titulo_Leccion: Listas
   Intro_leccion: >
   "Los elementos de una lista se declaran entre corchetes y
separados por ','."
   Elementos:
    - Elem:
      .
      .
      .
```

Estos campos expresan lo siguiente:
Tema: Numero del tema (número)
Titulo: Titulo del tema (texto)
Archivo: archivo de corrección para las preguntas de tipo código (lenguaje/correctores/temaX.ext) donde X es el número del tema y ext la extensión del archivo, por ejemplo para python será py.
Introduccion: texto con información del tema (texto)
Lecciones: lista de lecciones (vacío, el elemento es la lista de lecciones)
Leccion: número de la lección (número)
Titulo_Leccion: título de la lección (texto)
Intro_leccion: introducción de la lección (texto)
Elementos: lista de elementos de la lección lista de elementos (vacío)
Elem: hay dos tipos de elemento, se escribirá cual es (pregunta o explicacion)

Hay que destacar que un tema puede tener todas las lecciones que sean necesarias y una lección también puede tener todos los elementos que se quiera.

Ahora explicaremos qué contiene un elemento y cómo se estructura, comenzamos con las explicaciones:

```
- Elem: explicacion
  Contenido: >
```

```
      Los elementos de una lista no tienen por qué ser todos del
      mismo tipo. En la misma lista puede haber números enteros y
      cadenas.
      .
      .
      .
```

Las explicaciones sólo tienen un campo, en el que tendremos texto con la propia explicación, imágenes y los elementos markdown representados.

Contenido: texto donde escribiremos la explicación. Se comienza con '>', que es una etiqueta que indica que el siguiente texto ignorará los saltos de línea, alineando así el texto. En caso de poner bloques de código se puede escribir con '|', que conservará dichos saltos de línea tal y como estén en el archivo.

Ahora veremos las preguntas de tipo opciones:

```yaml
- Elem: pregunta
  Numero: 2
  Enunciado: >
        ¿que opción es correcta para extraer
        los dos nombres de la lista l=["Rafael", "Carlos", 32, 27]?
  Tipo: Opciones
  Pista: La numeración de los elementos comienza en 0
  Multiple: yes
  Opcion_correcta: 1,2,3
  Opciones:
       - 1) l[0:2]
       - 2) l[:2]
       - 3) l[0:-2]
       - 4) l[1:3]
      .
      .
      .
```

Enunciado: el enunciado de la pregunta (común a todas las preguntas)
Tipo: Opciones, aquí se indicará (común a todas las preguntas)
Pista: campo opcional con una pista que pueda ayudar a resolver la pregunta (común a todas las preguntas)
Multiple: (solo para test) yes | no Indica si la pregunta tiene una o más soluciones, es en inglés porque al interpretar el árbol traduce directamente este nodo a *booleano*

Opcion_correcta: (solo para test) número o números (en caso de tener más de una opción correcta) de la opción correcta. Siempre hay que poner los valores entre comillas. Ej: "1" ó "1,2,3".
Opciones: (solo para test) lista en la que se enumeran las opciones. El número de opciones puede variar.

Vamos ahora con las de tipo código:

```
 - Elem: pregunta
   Numero: 1
   Enunciado: Crea una lista 'l' con dos nombres y los números 32 y
27
   Tipo: Codigo
   Resultado: pregunta11
      .
      .
      .
```

Resultado: en el único campo específico de estas preguntas se escribirá el nombre de la función que comprobará el código. Esta función estará implementada en el archivo corrector.

Y por último las de sintaxis:

```
- Elem: pregunta
  Numero: 3
  Enunciado: >
        Escribe el código para introducir en la lista 'l'
        los numeros del 1 al 5 utilizando un bucle
  Pista: Consulta el tema de bucles si tienes problemas
  Tipo: Sintaxis
  Gramatica: CadRep
  Resultado: l=[1,2,3,4,5]
```

Y sus campos son:

Gramática: nombre del archivo correspondiente a la gramática a la que tenga que ajustarse la respuesta.
Resultado: **(opcional)** Si queremos tener comprobación del resultado. Aunque esta opción, como explicaremos más adelante no se completó, nos pareció interesante mantener este campo.

En los campos de texto del fichero YAML se pueden poner etiquetas tipo Markdown para introducir imágenes, que estarán en el directorio `img` de la carpeta del lenguaje del

tutorial en cuestión, así como resaltar partes del texto o introducir fragmentos de código y que sean mostrados con una fuente distinta al texto normal. Esto facilitará la lectura del alumno y le ayudará a mantener la atención a las partes más fundamentales.

En el caso de querer introducir imágenes con Markdown, la ruta que se especifique en la etiqueta tiene que ser la relativa al directorio `externalResources`.

Otro dato fundamental para escribir estos archivos es que hay que respetar las tabulaciones y espacios para anidar correctamente los elementos, o se producirá un error al tratar de interpretar el archivo. Igualmente ocurre si no se respetan los acentos de las etiquetas. Hay que escribirlas tal y como se presentan. Para más información recomendamos ver los ejemplos del repositorio GitHub del proyecto.

2.3.2. Archivos ANTLR

Estos archivos son los que contienen la gramática definida por el profesor. En el caso de este proyecto se ha utilizado la versión 4.X de ANTLR, por lo que la extensión de los archivos es '.g4'. En versiones anteriores sería '.g'.

Para que funcione correctamente con ANTLR y con nuestra aplicación el archivo ha de cumplir una serie de requisitos:

- El nombre del archivo debe coincidir con el de la gramática (mayúsculas y minúsculas también)
- El nombre de la regla debe coincidir con el de la gramática (todo en minúsculas)

Un ejemplo de gramática sería el siguiente:
El contenido del archivo `Triangulo.g4` es:

```
grammar Triangulo;
triangulo: 'h = 'NUMBER '\nb = ' NUMBER '\n' FORMULA ;
NUMBER: [0-9]+ ;
FORMULA: 'a=h*b/2' ;
```

En este archivo se describe una gramática que espera recibir un fragmento de código necesario para hallar el área 'a' de un triángulo con asignación de valores a dos variables:
h: altura
b: base

Con el ejemplo anterior, el fragmento de código aceptado sería:

```
h = 2
b = 3
```

```
a=h*b/2
```

Por el contrario una el fragmento de código incorrecto sería:

```
h = hola   → Falla porque se espera que a la variable h se le asigne un número
b = 3
a=3        → Falla porque espera la fórmula del área del triángulo
```

2.3.3. Archivos Python

Los archivos Python contendrán las funciones correctoras de las preguntas de tipo código. Se creará un archivo '.py' por cada tema que tenga el tutorial. Cada uno de estos archivos tiene una estructura similar a la que vamos a presentar con un ejemplo:

Primero vamos a ver los *import* necesarios en dicho archivo. Por defecto se usan fundamentalmente tres para la corrección pero, como es evidente, se podrá añadir cualquier otra librería necesaria para la corrección de preguntas, siempre y cuando el intérprete la tenga disponible. Estas librerías son:

sys: Esta librería nos permite obtener las variables que nos llegan desde Java y acabar la ejecución.
json: Nos permite tratar el archivo json.
tempfile: Nos permite operar con el archivo temporal.

Ahora vamos a ver un ejemplo del código Python correspondiente al fichero de corrección indicado en el YAML mostrado anteriormente:

```
def pregunta11(code, filename):
  try:
    exec(codigo)
    dicc = {}
    r=["Rafa","Carlos", 32,27]
    if set(locals()['l']) == set(r) :
      value1 = True
      value2 = ''
      value3 = ['']
    else:
      value1 = False
      value2 = 'Error de contenido'
      value3 = ['Puede que hayas escrito mal los elementos']
    dicc['isCorrect'] = value1
    dicc['typeError'] = value2
```

```
    dicc['hints'] = value3
    with open(filename, 'w') as outfile:
      json.dump(dicc, outfile)
    sys.exit(0);
  except Exception as e:
    e.print_exc()
    sys.exit(1)


def main():
  question = sys.argv[1] #1
  code = sys.argv[2] #2
  temp_name = next(tempfile._get_candidate_names())
  default_tmp_dir = tempfile._get_default_tempdir()
  filename = default_tmp_dir + '/' + sys.argv[3]

  if question == "pregunta11":
    pregunta11(code, filename)

if __name__ == "__main__":
  main()
```

Seguramente habrá muchas más formas de construir estos archivos que funcionen correctamente. Esta es la que se utiliza en los tutoriales de ejemplo que se han creado. Vamos a pasar a explicar qué hace:

Comencemos por el método main. Aquí sacamos los argumentos que la aplicación Java le pasa al intérprete de python, estos argumentos los sacamos del *array* sys.argv[]. El argumento 1 es el nombre de la pregunta a corregir, el 2 es el código que tenemos que probar y el número 3 el nombre del archivo temporal donde tenemos que escribir el resultado para que Java sepa lo que ha ocurrido en la ejecución. En el siguiente apartado explicaremos qué hay en este archivo.

Lo primero que hacemos es preparar el nombre del archivo temporal. Obtenemos la ruta hasta el directorio de archivos temporales y le concatenamos el nombre. Después comprobamos que el nombre de la pregunta que nos han pasado es alguno de los que tenemos en este archivo. En caso de que haya varias preguntas se pondrán todas las opciones con un bloque *if-else.*

Ahora pasamos a la pregunta en cuestión. A estas funciones correctoras se les pasará como argumentos el código que tendremos que comprobar y el archivo donde tenemos que escribir la solución. Para asegurarnos de que el código es ejecutable usamos un bloque *try-catch.* La función que ejecuta el código es exec(). Una vez ejecutado tendremos que buscar que las variables pedidas en la pregunta estén creadas y con sus valores

correspondientes disponibles. Estén o no tendremos que rellenar el resultado. Para eso están los siguientes campos:

- **value1:** El resultado de la pregunta, siendo éste *true* o *false*
- **value2:** Tipo de error. Este será el mensaje que se mostrará en caso de fallar
- **value3:** Lista de pistas adicionales. Estas pistas varían en función del error que cometa el alumno, siendo sólo visibles cuando el alumno falle.

Una vez creadas estas variables las introducimos en un diccionario *clave-valor* en con los campos correspondientes a estas 3 variables, nombrados 'isCorrect', 'typeError' y 'Hints', para luego volcarlos en el archivo temporal.

Cabe mencionar que la forma de finalizar este código representa el estado en el que acaba la ejecución. Esto tiene una interpretación en Java. Si el proceso acaba con '0' es que se ha ejecutado correctamente. Sin embargo si acaba con '1' se interpretará que ha habido un fallo de ejecución, posiblemente por sintaxis.

2.3.4. Archivos temporales JSON

Los archivos JSON serán el método de comunicación entre Java y Python. Decidimos utilizar archivos temporales para que no quede basura una vez acabada la ejecución de los tutoriales. Estos archivos tienen una estructura muy básica, que veremos a continuación:

```
{ 'isCorrect': True, 'typeError': 'correcto', 'hints': ['prueba1', 'prueba2']}
```

Esto representa el diccionario que creamos en el archivo Python, y será interpretado por Java para mostrar el resultado al usuario.

2.4. Interacción de componentes siguiendo ejemplo de ejecución

Para explicar cómo se enlazan todos los componentes y se utilizan las librerías veremos un ejemplo paso a paso. Tomaremos como punto de partida el archivo YAML de la sección 2.3.1 que representa un tema modelo.

Cuando arranca la aplicación, la clase controlador busca utilizando la Java Preferences API[22], en variables de entorno entorno del sistema operativo, la ruta del

[22] http://docs.oracle.com/javase/7/docs/technotes/guides/preferences/

directorio *'externalResources'* y los ejecutables de los lenguajes que tengamos disponibles en la aplicación. En el caso de estar configurado se muestra la ventana del menú principal. En caso contrario, se muestra la ventana de configuración para que el usuario pueda establecer los parámetros correctos y, con ello, hacer uso de la herramienta.

Una vez seleccionados el lenguaje y el tema a mostrar, el archivo YAML de dicho tema se interpreta con la librería snakeyaml, obteniendo un árbol clave-valor con el contenido del archivo. Una vez hecho esto se crean objetos java con los atributos del árbol. Todo este proceso se lleva a cabo en la clase `YamlReaderClass` implementada por los desarrolladores.

A la hora de mostrar la información en las distintas ventanas, los campos de texto se interpretan con la herramienta Markdown para interpretar las posibles etiquetas y conseguir mostrar el texto en la vista de la forma deseada.

El procesamiento de las preguntas de tipo *test* es trivial. Se crea un *array* de enteros con las respuestas seleccionadas y se compara con el *array* solución de la clase Opciones.

Para la comprobación de las preguntas de tipo código, en primer lugar se crea un archivo temporal, que utilizaremos para comunicar el resultado de la ejecución escribiendo una estructura JSON. Una vez creado el archivo, creamos un proceso llamando al intérprete del lenguaje, y pasándole el archivo corrector, la pregunta a corregir, la respuesta del alumno, y el nombre del fichero temporal para escribir en él.

En función del resultado obtenido tras la comprobación de este tipo de preguntas se rellena el fichero temporal JSON como se explica en el apartado 2.3.4.

Para las preguntas de tipo sintaxis, la comprobación se lleva a cabo mediante clases generadas y precompiladas previamente con ANTLR, que se explica en el siguiente apartado. Estas clases se encuentran en el directorio de `gramaticas` como se explica en el apartado 2.1. Como son clases que externas a la aplicación es necesario crear instancias para ser usadas internamente. Esto se consigue con URLClassLoader[23]. Así podemos ejecutar los métodos necesarios para la comprobación.

2.5. Integración y modificación

Los profesores deben ser capaces de crear nuevos tutoriales, añadir o eliminar temas de un tutorial ya creado o incluso corregir alguna posible errata que se haya podido encontrar en alguno de los ficheros ya publicados. En esta sección explicamos cómo llevar a cabo estas acciones.

[23] Librería de java que permite el acceso a clases externas a un proyecto en tiempo de ejecución.

La integración de nuevos tutoriales no es una tarea muy complicada. Sólo hay que tener en cuenta unos cuantos puntos con respecto a la creación de nuevos archivos.

Los pasos a seguir son los siguientes:

1. Suponiendo que el directorio con el nombre del lenguaje no exista, se crea el directorio con el nombre deseado. Por ej. (Python, PHP, Ruby, Perl...) y se añade a `externalResources`.

2. Se escribe el archivo YAML que contiene la información del tutorial. En este paso hay que tener en cuenta la estructura del fichero y los nombres de las claves explicado en el apartado 2.3.1. Estos archivos se guardan en la raíz del directorio del paso anterior.
3. En el caso de que se hayan incluido imágenes en alguno de los campos del tutorial, se creará un directorio llamado `img` con las imágenes del tutorial. Este directorio se encuentra en la raíz del directorio del nombre del lenguaje.

4. Se generan otros dos directorios en el directorio raíz del lenguaje:
    a. `Correctores`: aquí se encuentran los ejecutables encargados de llevar a cabo la corrección de las preguntas de tipo código.
    b. `Gramaticas`: aquí se encuentran los archivos ya compilados encargados de la corrección de las preguntas de tipo sintaxis.

5. Se generan los ficheros correctores de las preguntas de tipo código y se guardan en el directorio correctores.

6. Se generan las gramáticas para las preguntas de tipo sintaxis. Debido a que este paso es más engorroso se explicarán los pasos de manera detallada:
    a. Se escribe la gramática con las restricciones explicadas en el apartado 2.3.2 en un fichero con extensión *.g4* (supongamos Gramatica.g4).
    b. Se generan las clases de Java con ese fichero con el siguiente comando a través de consola:

    ```
    > java -cp antlr-4.5.3-complete.jar
    org.antlr.v4.Tool Gramatica.g4
    ```

    Ahora se van a explicar cada una de las partes del comando:
        i. *antlr-4.5.3-complete.jar*: herramienta encargada de compilar la gramática. Si no está en el mismo directorio que el archivo con la gramática es necesario escribir la ruta completa.
        ii. *Gramatica.g4:* archivo que hemos escrito en el paso a).
    c. Ahora llega el momento de compilarlos. Para ello se escribe el comando:

```
> javac -cp antlr-4.5.3-complete.jar Gramatica*.java
```

d. Se copian los archivo con extensión *.class* en un directorio con el mismo nombre que el archivo de la gramática generada antes. Este directorio ha de ser un subdirectorio de la carpeta `gramaticas`.

7. El último paso sería distribuir el directorio del nuevo lenguaje entre los alumnos y que estos lo guarden en sus equipos en la dirección correcta para poder llevar a cabo su uso.

Si por el contrario se quisieran añadir nuevos temas los pasos a seguir serían los mismos salvo el paso número 1.

En el caso de que se encuentre alguna errata, lo único que se debería hacer es localizar el fichero YAML del tema concreto, llevar a cabo su modificación y la de los elementos involucrados y volver a distribuirlo.

3. Conclusiones

En este apartado se hablará de los resultados obtenidos frente a los esperados, los problemas encontrados para llevar a cabo los objetivos y las soluciones que se pusieron para subsanarlos. Además, se sugerirán posibles ampliaciones que se pueden llevar a cabo para mejorar la aplicación. Algunas de ellas no se realizaron debido al escaso tiempo disponible para un proyecto de tal envergadura; otras son ideas que implican grandes cambios, pensadas para implementar a largo plazo.

3.1. Resultados obtenidos frente a resultados esperados

En esta sección nos centraremos en comprobar si los objetivos marcados desde el principio se han cumplido satisfactoriamente o no.

A continuación hablaremos de los resultados obtenidos con respecto a cada objetivo:

3.1.1. Presentación de tutoriales

Este objetivo se ha conseguido de forma satisfactoria, debido a que se ha logrado el principal objetivo, la presentación de tutoriales cortos, consiguiendo una interacción muy dinámica gracias a la posibilidad de intercalar preguntas de distintos tipos con las que el alumno puede comprobar de una forma rápida los conocimientos aprendidos.

Debido a la escasez de tiempo y a algunos problemas que se tuvieron para poder mostrar ciertos elementos visuales, la interfaz de usuario no se ha podido hacer más agradable a la vista debido a que primaba más la funcionalidad de la herramienta.

3.1.2. Creación y adición de tutoriales.

Desde el principio se pensó en la posibilidad de poder gestionar tutoriales para mostrarlos con la herramienta. En primera instancia, para utilizar estos tutoriales, deberían estar como recursos de la implementación de la herramienta. Este hecho suponía que cada vez que se quisiera añadir, borrar o modificar alguno de estos elementos habría que volver a compilar y empaquetar el proyecto por completo, siendo una tarea engorrosa para el profesor.

Para aliviar este cometido, se permitió que tanto los ficheros YAML como los recursos que fuesen necesarios para dicho tutorial estuvieran en un directorio externo a la aplicación. Así los profesores sólo tendría que crear o modificar los tutoriales y distribuirlos entre los alumnos.

3.1.3. Soporte para diferentes tipos de pregunta

La aplicación ofrece varios tipos de preguntas a las que los alumnos que quieran completar estos tutoriales deben responder correctamente.

Como ya se ha hablado anteriormente, se ofrecen preguntas de tipo código, test y sintaxis, consiguiendo los objetivos fijados con los dos primeros tipos. Con respecto a las preguntas de tipo sintaxis, se pensó en un principio hacer un chequeo de la salida obtenida con el código del alumno. Pero debido a la falta de tiempo y diversos problemas encontrados antes de esta fase, sólo se comprueba el formato del código introducido.

3.1.4. Soporte de lenguajes interpretados

La herramienta se pensó para que estuviera destinada a lenguajes de programación interpretados. Esto se debe a su facilidad a la hora de ejecutar las sentencias introducidas por el usuario. Este problema en potencia sólo se encuentra en las preguntas de tipo código, ya que son las únicas en las que se comprueba mediante una ejecución del código introducido por parte del alumno.

Para conseguir este objetivo, la comprobación se hace desde el intérprete del lenguaje seleccionado instalado en el equipo del usuario. De esta forma se hace completamente independiente de la aplicación sin importar el lenguaje sobre el que se está trabajando.

3.1.5. Funcionamiento multiplataforma

Desde que se empezó el desarrollo se pensó en que fuese una herramienta destinada a los alumnos. Como cada persona tiene sus preferencias con respecto al Sistema Operativo[24], que eligen por gusto o adecuación a sus necesidades, se pensó que la aplicación debería funcionar en todos los SO. Gracias a haber construido la aplicación con Java, este objetivo se ha cumplido.

Sin embargo, en SO como *Linux*, la interfaz de la aplicación puede sufrir pequeñas anomalías en la parte visual, por ejemplo no permitiendo ver algunos elementos de forma correcta.

3.2. Problemas encontrados y sus soluciones

A lo largo del desarrollo, han ido apareciendo algunos problemas que han dificultado en mayor o menor medida la continuidad del proyecto. Aunque no han imposibilitado la finalización del proyecto, sí han supuesto un cierto retraso a la hora de llevar a cabo la finalización de cada una de las fases por separado.

A continuación se describen algunos de los problemas encontrados más relevantes.

3.2.1. Herramienta SceneBuilder

Cuando se comenzó la implementación de la interfaz de la aplicación con JavaFX se pensó en crear el diseño con la herramienta externa `SceneBuilder`. Pensada para generar interfaz gráfica con JavaFX, esta herramienta genera un archivo `FXML` con los elementos y sus características de difícil comprensión, lo que provocaba dificultades a la hora de implementar la funcionalidad y tratar de modificarlas a mano.

Solución tomada:

Se buscó información sobre cómo generar las vistas mediante código para implementarlo así.

[24] En adelante SO.

3.2.2. Rutas de recursos externos

La gestión de acceso a directorios externos a la aplicación supuso un quebradero de cabeza para los desarrolladores. A pesar de saber que era necesario lidiar con esto desde una fase temprana del desarrollo, se comenzó haciendo que los archivos externos estuviesen integrados al empaquetar la aplicación, lo que, aunque facilitaba enormemente la gestión de rutas, provocaba que cada vez que se quisiera integrar nuevos tutoriales a la aplicación hubiese que recompilar y reempaquetar todo el proyecto.

Solución tomada:

Como alternativa a este problema se ha decidido utilizar Preferences API de Java, que nos permite almacenar la ruta del directorio externo definido anteriormente. Esto también provocó un ligero retraso debido a que se tuvo que buscar información acerca de esta tecnología.

3.2.3. Ajuste de elementos al tamaño de la ventana

Una vez creada la interfaz gráfica de la aplicación, se descubrió que los elementos que formaban parte de la vista no se ajustaban al tamaño de la ventana en el caso que se redimensionase. Esto se debía a que no se estaba utilizando el tipo de panel adecuado y además en el momento de mostrar los elementos se hacía de forma errónea.

Solución tomada:

Buscar información acerca de los paneles disponibles en JavaFX y seleccionar el que se ajustaba a las necesidades. Con esta solución se reveló el problema que había en el código con respecto a cómo se mostraban los elementos, que también fue resuelto aunque se tuvo que dedicar más tiempo del esperado.

3.2.4. Mostrar imagen en WebView

Otro problema relacionado con las rutas de los recursos externos fue conseguir mostrar las imágenes ligadas al código HTML. Esto estaba causado porque para mostrar imágenes mediante HTML hacía falta especificar la ruta absoluta al recurso. Este problema fue acompañándonos a lo largo de la mayor parte del proyecto debido a que no encontrábamos la razón del problema.

Solución tomada:

La solución a este problema fue investigar durante mucho tiempo la posible razón y e ir probando con las distintas soluciones que íbamos encontrando. Para evitar retrasos en la entrega del proyecto, se decidió continuar con la aplicación y trabajar en la solución en paralelo.

3.2.5. Compilación externa de gramáticas

Para las preguntas de sintaxis son necesarias las clases creadas por ANTLR. Para este apartado en un principio se decidió tener como externo solamente el archivo donde se escribe la gramática, lo que suponía tener que construir las clases con ANTLR en cada ejecución y además hacer una compilación en tiempo de ejecución. Esto suponía un problema por la necesidad de tener JDK[25] en vez de JRE[26] instalado en la máquina donde se fuese a usar la aplicación, por tener que utilizar librerías de compilación, no disponibles en JRE.

Solución tomada:

Crear en el directorio de recursos externos un directorio con las clases de ANTLR compiladas previamente. Esto propició el tener que buscar información acerca de cómo hacer uso de estas clases de forma externa a la aplicación.

[25] JDK(Java Development Kit) es el kit de desarrollo de Java, que incluye las herramientas de compilación, construcción de proyectos y el propio JRE

[26] JRE(Java Runtime Environment) es la propia máquina virtual de Java, que permite la ejecución de aplicaciones java

3.B. Conclusions

In this section we will discuss the results obtained against the expected results, the problems found to reach the objectives and the solutions taken to solve them. Besides, we will suggest possible extensions that could be brought to improve the application. Some of them were not made because the limited time available for a project of this scope. The last section consists of ideas that involve big changes, intended to be implemented in the long term.

3.1.B. Obtained results before expected results

In this section, we are going to concentrate on checking if the marked objectives from start were achieved in a satisfying way or not.

Then we will talk about the obtained results against each objective:

3.1.1.B. Tutorial display

This objective has been achieved, because the main objective has been completed, the presentation of short tutorials, obtaining the expected dynamic interaction, thanks to the possibility to alternate different kind of questions in which the student can check in a quick way the acquired knowledge.

Because of the limited time and some issues to show certain visual elements, the user's interface has not been made more appealing, because we advanced in favor of the tool's functionality.

3.1.2.B. Creation and addition of tutorials

From the beginning we considered the possibility to handle tutorials to be displayed with this tool. In first instance, to be able to use these tutorials, they had to be resources of the tool implementation. This fact implied that each time someone wanted to add, delete or modify any of these elements, it had to recompile and package the entire project. This task would be tedious for the teacher.

To ease this, it was allowed that both YAML files and the necessary resources for the tutorial were placed inside an external directory to the application. This way, teachers would just have to create or modify the tutorials and distribute them between the students.

3.1.3.B. Support to different kinds of questions

The application offers different kind of questions which students who want to complete these tutorials must answer correctly.

As it has been mentioned before, the different types of questions offered are: code, test and syntax. The objectives are achieved with the first two types. Regarding the last one, it was intended in first instance to check the output. But because of limited time and different issues that appeared before this phase, it's just checked the format of the input code.

3.1.4.B. Support of interpreted languages

The tool was designed to work with interpreted programming languages. This is because of the ease to execute the sentences written by the user. The potential trouble is found in the code type questions because these are the ones in which checking is made through the execution of the code that was introduced by the student.

To achieve this objective, checking is made from the selected language interpreter installed in the user's system. This way the execution is application independent so it doesn't matter which language we are working with.

3.1.5.B. Multiplatform operation

Since development started, we expected the application to be dedicated to the students. Because the fact that people have different preferences talking about Operating Systems[27] which are chosen because of need or preference, that implied the OS system independent operation. Thanks to the fact of building the application with Java, this target has been completed.

However, in OS and *Linux*, the interface can show little visual defects, for example not showing properly some elements.

[27] Further OS

3.2.B. Problems found and their solutions

Over development, some issues have appeared that have obstructed the progress of the project to a greater or lesser extent. Nevertheless it has not made the termination of the project impossible, but it has implied some delay during each of the development phases.

Next we will explain some of the most relevant issues.

3.2.1.B. SceneBuilder tool

When the interface development started, using JavaFX we tried to build the design of the views with the external tool `SceneBuilder`. This tool was designed to generate graphic interfaces with JavaFX. It generates an FXML file with the elements from which the interface is composed . This file format is barely understandable . This introduced lots of extra difficulties to implement its functionality and to modify them by hand.

Solution taken:

Search for information to build these views by means of code, and to implement them in that way.

3.2.2.B. External resources paths

The management of application external directories was a headache for the developers. Despite knowing it was a key feature to deal with since an early development phase, initially external files were packaged together with the application, which, even though easen path management, it led to recompiling and repacking the entire project each time we wanted to include new tutorials to the application.

Solution taken:

As an alternative to this issue we decided to use the Java Preferences API, which allows us to store the path to the external directory defined before. This fact also involved a little delay because of the information search this technology required to implement it.

3.2.3.B. External compilation of grammars

Once the application's graphic interface was created, we found out that the elements that composed it didn't adjust at all to the windows shapes if it was resized. That was caused because of the kind of panels chosen was not appropriate and it did not display correctly the elements in the stage.

Solution taken:

Research for further information about JavaFX available panels and choose one according to the needs of each case. With this solution it appeared another issue regarding the element display. It was also solved but it took more time than expected.

3.2.4.B. WebView image display

Another issue related to the pathing of external resources was to display correctly the images linked to the HTML code. This was caused due to the need to specify the absolute path to the resource. This issue was present throughout most of the development phase because we did not find out the source.

Solution taken:

We solve this by investigating for a long time the possible causes and trying out all the different possible solutions we found. To avoid more delays on the project deadline, we decided to go on with all the development and work in the solution at the same time.

3.2.5.B. External grammar compilation

For the syntax questions, classes created by ANTLR are needed. For this section the first idea was to have as external file the one where the grammar was defined, which implied to build the ANTLR classes each time we needed to execute the involving questions, and also to make an execution time compilation. The last one implying the needs to have JDK[28] instead of JRE[29] installed in the host machine, because compilation libraries were needed, not included in the JRE distribution.

[28] JDK(Java development Kit) it includes compilation and project building tools and also JRE

[29] JRE (Java Runtime Environment) is the Java Virtual Machine itself, who provide Java application execution.

Solution taken:

Creating the external resource directory with the ANTLR classes previously compiled. This involved searching the information about how to make usage of these classes externally to the application.

4. Posibilidades de ampliación

Dado el potencial de aplicación de este proyecto en el ámbito académico cabe pensar en cómo se puede mejorar y ampliar en un futuro para darle más utilidades y comodidades para los usuarios.

A continuación se describen ideas para poder ampliar esta aplicación:

4.1. Coloreado de sintaxis

En el campo de texto destinado a escribir el fragmento de código se muestra el texto plano tal y como lo escribe el usuario. Una posible modificación muy atractiva de cara al usuario sería añadir un coloreado a la sintaxis resaltando las palabras reservadas del lenguaje en cuestión. De esta manera, el componente de introducción de código tendría un comportamiento similar a un editor de texto enriquecido o un entorno de desarrollo.

Añadir esta funcionalidad supone tener que lidiar con la interfaz de JavaFX en la que se representa el código, haciendo que tras cada pulsación de tecla se revise el texto que tenga escrito y se compruebe si tiene palabras reservadas para cambiarles el color.

4.2. Interfaz gráfica

El desarrollo de esta aplicación se ha centrado más en la funcionalidad de la herramienta. Por eso, los elementos de la interfaz aparecen de una forma sencilla y resultan poco atractivos de cara a los usuarios.

Una clara mejora sería otorgar animaciones a los elementos como pueden ser los botones o buscar una combinación de colores y una distribución más atractiva de cara a los principales usuarios, los alumnos.

4.3. Gestión de usuarios

Debido a que es una herramienta que puede ser usada tanto por profesores para crear tutoriales, como por diferentes alumnos para aprender sobre un lenguaje de programación, en un futuro se podría añadir un gestor de usuarios que muestre distintos tipos de interfaces y

funcionalidades dependiendo del rol del usuario. Junto con la gestión de usuarios, es posible habilitar varias sesiones en un mismo equipo y mostrar datos estadísticos de los resultados obtenidos.

Para construir esta parte se tendría que integrar una base de datos que guarde la información de los distintos usuarios.

4.4. Guardar estado del tutorial

Por el momento, dado que la herramienta está pensada para lecciones de corta duración, no se contempla la idea de guardar el estado en el que se encuentra la lección. En un futuro sería interesante permitir almacenar respuestas correctas, elementos habilitados, tutoriales completados, etc.

Al igual que en la propuesta de gestión de usuarios descrita anteriormente, la implementación de esta funcionalidad también dependería de una base de datos que guarde el estado de cada uno de los usuarios.

4.5. Constructor de ficheros YAML

En el caso de que un profesor quiera generar un tutorial para esta herramienta, y una vez construido el módulo de gestión de usuarios, al acceder a la interfaz de profesor se podría mostrar una opción de generar tutorial donde se mostrarán los campos a rellenar y la aplicación generará un fichero YAML de forma automática.

Otro posible enfoque es implementar esta herramienta como otra aplicación externa, más sencillo que la integración en la aplicación, y con posibilidad de hacer la integración posteriormente, lo que lo convierte en una opción más atractiva. La segunda opción de esta modificación implicaría hacer una pequeña aplicación desde cero, utilizando la misma librería que hemos usado en el proyecto (snakeyaml), lo cual no plantea un reto demasiado complejo *a priori*. Sin embargo, la primera opción depende de disponibilidad de una base de datos y de la implementación previa de la funcionalidad de gestión de usuarios.

4.6. Ejecución de preguntas de tipo sintaxis

A la hora de corregir las preguntas de tipo sintaxis, se puede ser más flexible en la corrección de la entrada de código introducido por el alumno, centrándose sólo en búsqueda de ciertas estructuras, para después llevar a cabo una ejecución al igual que se hace en las preguntas de tipo código. Para llevar a cabo este proceso, se debería añadir el campo resultado a las preguntas de tipo sintaxis con el nombre de la función correctora en el archivo YAML e implementar dicha función en el script corrector.

4.7. Lenguajes compilados

Hasta el momento sólo se ha pensado en esta herramienta como gestor de tutoriales de lenguajes de programación interpretados como Python o R. En un futuro, se podría ampliar este ámbito a todo tipo de lenguajes compilados como C, C++, Java, etc. Para ello haría falta añadir las tecnologías necesarias propias de cada lenguaje.

Siendo más precisos, las modificaciones se aplicarían en las preguntas de código, en las que tendríamos que llamar a un compilador del lenguaje en cuestión desde la aplicación, y éste trataría de compilar el código. Si este compila, ejecutarlo y comprobar que el resultado es el pedido. Si no compila, buscar el error y mostrárselo al usuario.

4.8. Migración a la web

Actualmente, la herramienta desarrollada es una aplicación de escritorio que cada alumno debe descargar e instalar en su equipo. Sería una buena idea publicar/hacer disponible la aplicación vía web y así eliminar la etapa de instalación. Además, esto agilizaría la distribución de actualizaciones o nuevos tutoriales ya que el alumno no necesitará descargarse nuevos ficheros. También abriría la posibilidad de que el alumno notifique sus progresos al profesor de forma inmediata.

También se podría enlazar el sistema de tutoriales con el Campus Virtual, presentando las lecciones como ejercicios para los alumnos, de tal forma que los ejercicios de evaluación continua de una asignatura sean estos tutoriales, haciendo así que se automatice y facilite la evaluación de alumnos.

Sin duda, este es el cambio más complejo y grande, y a su vez el que más posibilidades y retos propone. En primer lugar, la migración a web implica el uso de una base de datos y, lo peor de todo, ejecución de código potencialmente dañino en un servidor. Esto llevaría a tener que implementar un control de código introducido por el usuario, o como solución rápida, crear máquinas virtuales específicas para ejecutar el código para que, en caso de recibir un ataque, no se produzcan grandes daños.

4.9. Paquete de idiomas

A pesar de tener el texto de las interfaces escrito en el código directamente, no sería excesivamente complicado ni trabajoso añadir paquetes de idioma. Tan sólo habría que añadir una clase de constantes por cada idioma donde tendríamos el texto a mostrar, cambiar las interfaces donde tenemos texto escrito con estas variables y hacer un pequeño método de selección en la ventana de ajustes. Esto podría aportar mayor difusión a la herramienta.

Por otro lado, al estar separados los tutoriales de la aplicación, sin necesidad de traducir ésta última se pueden traducir los tutoriales a otros idiomas de manera muy sencilla, siempre y cuando en su construcción se respete el esquema explicado.

5. Aportaciones personales

5.1. Rafael Caturla Torrecilla

En primer lugar me gustaría decir que el trabajo realizado junto con mi compañero, a pesar de la cantidad de horas dedicadas y la frustración de ciertos momentos por no conseguir algunos de los objetivos en el plazo designado ha sido una experiencia gratificante que ha tenido como resultado este Trabajo Fin de Grado como nuestra primera aportación real.

La mayor parte del trabajo lo hemos hecho a medias. Uno buscaba información, el otro hacía pequeñas pruebas sobre lo encontrado, uno construía los archivos externos mientras el otro preparaba el *parser,* en resumen, que hemos trabajado sobre lo mismo simultáneamente pero con enfoques ligeramente distintos. Este método de trabajo nos ha dado una visión global del proyecto, y nos ha llevado a tomar decisiones que, de trabajar en solitario, seguramente habrían sido diferentes.

Comenzando con las aportaciones personales, cuando comenzamos el desarrollo, y aprovechando que estaba aprendiendo Python en otra asignatura de la carrera, comencé haciendo algunas pruebas de programación en este lenguaje cuando no estaba con mi compañero, mientras que cuando estaba con él me dedicaba a buscar información sobre YAML, para fijar la estructura de este archivo, que resultó en el esquema que se ha explicado.

Cuando decidimos incluir Markdown dentro de la estructura YAML pensé que iba a ser complicado introducir un lenguaje de marcas dentro de otro lenguaje de marcas, asique me dediqué a buscar cómo hacerlo y probar si podían surgir problemas. Me encontré con una gran variedad de librerías para interpretarlo, decantándonos al final por pegdown por ser la que tenía más actividad en el repositorio de github.

Después de esto me dediqué a dar retoques a los archivos y al parser que había hecho Carlos para cerciorarme de que todo estaba correctamente.

Pasamos entonces a la construcción de las clases Java del proyecto, para lo que cambié la estructura base a Maven. Una vez introducidas todas las clases, pasamos a probar la interfaz gráfica con JavaFX, los primeros pasos fueron desastrosos, porque no nos gustaba a ninguno de los dos la herramienta de diseño gráfico '*Scene Builder*', por lo que acabamos mirando la API de JavaFX para escribir el código a mano. Carlos hizo las ventanas de la interfaz mientras yo probaba componentes para saber cuáles se adaptaban mejor a nuestras necesidades, con esto descubrimos infinidad de *features* de esta librería.

Una vez la interfaz estaba más o menos hecha, comencé a crear el código de corrección de las preguntas de opciones. Además, en este momento surgió un pequeño problema con las extensiones del intérprete de markdown, que provocaba que algunos elementos no se leyeran de forma correcta. Una vez solucionado se nos ocurrió la posibilidad de añadir imágenes con etiquetas Markdown, que también fue un quebradero de cabeza. Al final se perdió bastante tiempo con esto, pero se corrigió unos días más tarde.

Luego pasé a intentar integrar el intérprete de Python en Java. Lo primero fue crear los correctores en Python, con ayuda de mi compañero. Después, como primera aproximación conseguí integrarlo mediante una librería que simulaba el propio intérprete dentro de la JVM, pero acabamos desechando este módulo por ser muy lento e ineficiente. Tras una reunión con los directores, pasé a investigar cómo ejecutar el propio intérprete de Python desde Java, sin necesidad de que fuese dentro de la JVM sino de forma independiente, en otro hilo, para además tener la posibilidad de darle un *'timeout'* por si el alumno introducía un bucle infinito. Al final se utilizó el ProcessBuilder de Java con un poco de paciencia. Además también tuve que investigar el tema de creación de archivos temporales, y el uso de JSON en dichos archivos para la comunicación del proceso de la corrección con la aplicación.

Después de esto empezamos a necesitar una ampliación de la interfaz gráfica, en la que colaboré asiduamente, porque no estaba previsto y era necesario urgentemente, junto con algún que otro retoque a lo hecho. Esto incluye el paginador de la lección, un elemento gráfico que ayuda bastante al alumno a saber dónde está y cuánto le queda.

Por último investigué la Preferences API para guardar en variables de sistema las rutas de los archivos externos y los intérpretes, Así como dar numerosos retoques necesarios para empaquetar el proyecto, tener una jerarquía de clases adecuada, solucionar pequeños errores, etc.

5.2. Carlos Congosto Sandoval

Al tratarse de un grupo reducido de sólo dos personas, el soporte básico de la aplicación lo he llevado a cabo en conjunto con mi compañero de proyecto. Dependiendo de la fase de desarrollo me he centrado en unos hitos más que en otros como explico a continuación:

En la fase de generación de los archivos YAML, investigué el formato que deben tener los distintos elementos que forman el archivo como son los bloques de texto o las listas de elementos. Además, como los directores nos sugirieron la idea de añadir etiquetas de Markdown, busqué información acerca de ello e introduje las distintas etiquetas que veía interesantes para la aplicación y donde podían resultar más útiles.

Mientras se estaban llevando a cabo los últimos retoques de los ficheros, empecé a pensar en una posible jerarquía de clases, necesaria para almacenar la información de cada elemento que constituía el tutorial. Gracias a esto, cuando se concluyeron las modificaciones de los ficheros YAML, pude debatir con mi compañero la viabilidad de la idea de estructura de clases.

Tras tener elegido el intérprete de los ficheros YAML y comprobar su funcionamiento, me encargué de crear el modelo de la aplicación con los distintos elementos que forman el árbol generado por el intérprete de YAML. Esto pudo ser llevado a cabo de una manera muy rápida gracias a que me pude adelantar pensado en la posible jerarquía de clases del modelo.

Durante la fase de creación de vistas desarrollé las distintas clases encargadas de generar los paneles, basándome en la función escrita por mi compañero. Más tarde añadí la función necesaria para interpretar las posibles etiquetas Markdown que pueda haber en el archivo YAML, en código HTML y mostrarlo en el elemento encargado de mostrar contenido web, así como las hojas de estilo (CSS), tanto para los elementos web como de la ventana generada con JavaFX.

Para las preguntas de tipo test, me centré más en el modo de poder visualizar las opciones en la ventana, y generar la función correctora para determinar si la respuesta es correcta o no. Para esta fase fue muy importante la comunicación con mi compañero y el trabajar junto a él para llegar a un convenio sobre qué tipo de datos se iban a utilizar en esta función.

En la fase de preguntas tipo código, tuve que implementar las funciones necesarias para obtener el código introducido por el usuario. Este cometido tuvo que ser realizado por

mí debido a que yo conocía mejor los elementos de cada ventana y podríamos avanzar de una forma más rápida. La parte encargada de llevar a cabo la corrección de dichas preguntas fue realizada junto a mi compañero debido a su mayor conocimiento sobre Python. Busqué información sobre dicho lenguaje y me hice cargo de escribir un pequeño esqueleto del archivo que contiene las funciones correctoras a cada pregunta del tema. Más tarde le expliqué la idea de ejecución que se lleva a cabo para que pudiera avanzar con el trabajo mientras yo me centraba en arreglar pequeños fallos de fases anteriores y avanzar con los elementos visuales.

Para las preguntas de tipo sintaxis, debido a que mi compañero estaba encargado del desarrollo de otros elementos del proyecto, busqué información acerca de la herramienta necesaria para generar las gramáticas. Una vez tuve una idea general de cómo se podían hacer gramáticas personalizadas empecé a hacer pequeñas pruebas y cuando comprobé que funcionaban correctamente, empecé a escribir las necesarias para este tutorial concreto, así como su compilación y su funcionamiento desde dentro de una aplicación Java.

En nuestro proyecto se propuso la idea de que las clases que realizan la comprobación mediante sintaxis fueran independientes a la aplicación. Por lo tanto, para hacer uso de ellas, había que generar las clases fuera del proyecto y hacer la llamada externa desde el mismo. Por eso, consulté con los directores para que me guiaran cómo hacerlo, y empecé a probar hasta conseguir el éxito de este cometido.

Como encargado de la parte visual, me vi en la necesidad tener que hacer una actualización de la ventana tras contestar a las distintas preguntas, así que añadí un elemento más a los paneles, que muestran las preguntas para indicar si la respuesta introducida por el usuario en cualquiera de los formatos es correcta o no.

6. Agradecimientos

Queremos dar las gracias a los directores Enrique y Manuel, por estar ahí cuando hacía falta un empujón para continuar con el proyecto.

También queremos dar las gracias a la Asociación Socio Cultural de Ingenieros en Informática (ASCII) por estar siempre ahí para animarnos en nuestros momentos de bajón e incitarnos a cambiar el trabajo por unos ratos de juego y diversión, y darnos consejos brillantes con algunos trocitos de código, en especial a Miky y Pont.

No nos olvidamos de nuestras familias, que siempre han estado presentes en las ganas que le hemos echado al desarrollo en las innumerables horas de dedicación, y que nos han soportado y apoyado durante tantos años de carrera.

Tenemos que mencionar también a la web http://stackoverflow.com/ y su comunidad, por ayudarnos en numerosas ocasiones aportando ideas y ejemplos de código que nos han ayudado a acabar el proyecto.

Por último queremos mencionar a Will, por echarnos una manilla con la traducción, y a título personal algunos agradecimientos más.

Bibliografía

[1] Lavieri, Dr Edward. *Getting Started with Unity 5*. Packt, 2015.

[2] Margolis, Michael. *Arduino Cookbook*. Sebastopol, CA: O'Reilly, 2012.

[3] Crawley, Michael J. *The R Book*. Chichester, England: Wiley, 2007.

[4] Sutherland, Jeffrey Victor. *Scrum: The Art of Doing Twice the Work in Half the Time*.

[5] Vogel, Lars. *Eclipse IDE: Java Programming, Debugging, Unit Testing, Task Management and Git Version Control with Eclipse*. Leipzig: Vogella, 2013.

[6] Eckel, Bruce. *Thinking in Java*. Upper Saddle River, NY: Prentice Hall, 2014.

[7] Horstmann, Cay S. *Java SE 8 for the Really Impatient*, 2014.

[8] Meyer, Eric A., and Eric A. Meyer. *CSS: The Definitive Guide*. Beijing: O'Reilly, 2007.

[9] Parr, Terence. *The Definitive ANTLR 4 Reference*, 2013.

[10] Lutz, Mark. *Python*. Cambridge: Ed. O'Reilly, 2010.

[11] *Maven*. S.l.: Alphascript, 2012.

[12] Loeliger, Jon. *Version Control with Git:*. Beijing: O'Reilly, 2009.

Anexos

A) Actas de las reuniones

Nº DE ACTA: 1

FECHA: 06-Octubre-2015

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Decisiones sobre tecnologías a utilizar

DECISIONES TOMADAS:
- Lenguaje de programación: Java
- Tipo de archivo de tutoriales: YAML
- Lenguaje dirigido el tutorial: Python 3.4

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Crear los archivos YAML

Nº DE ACTA: 2

FECHA: 20-Octubre-2015

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar archivos YAML

DECISIONES TOMADAS:

- Cambiar aspectos del YAML
- Añadir campo pista para algunas preguntas
- Añadir etiquetas Markdown a los campos de texto
- Uso de SNAKEYAML como intérprete de los archivos YAML

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN

- Llevar a cabo las modificaciones propuestas
- Elegir tecnología para interpretar Markdown

Nº DE ACTA: 3

FECHA: 03-Noviembre-2015

ASISTENTES:

- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:

- Comunicar a los profesores tecnologías elegidas
- Comprobar que los cambios tienen éxito
- Comprobar estructura de clases Java

DECISIONES TOMADAS:

- Se acepta el uso de MAVEN para las dependencias de librerías externas

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN

- Comprobar la interpretación de las etiquetas Markdown
- Mostrar información por ventana

Nº DE ACTA: 4

FECHA: 17-Noviembre-2015

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Probar elementos web en las vistas

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Cambiar path de recursos
- Probar elementos complejos con pegdown
- Prototipo de preguntas tipo test

Nº DE ACTA: 5

FECHA: 01-Diciembre-2015

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar los objetivos de la reunión anterior

DECISIONES TOMADAS:
- Seguir con los problemas planteados

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Incluir hojas de estilo (CSS) en las vistas
- Cambiar formato de opciones para preguntas de opciones (checkbox o radioButton)
- flujo de ventanas en papel

Nº DE ACTA: 6

FECHA: 17-Diciembre-2015

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar avances
- Debatir el diseño de flujo entre ventanas

DECISIONES TOMADAS:
- Cambiar el flujo de ventanas para poder mostrar preguntas intercaladas con la teoría

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Construir ventana única para la teoría y las preguntas
- Mostrar el flujo de una lección con preguntas de tipo test

Nº DE ACTA: 7

FECHA: 14-Enero-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar el flujo de la lección

DECISIONES TOMADAS:
- El flujo es correcto
- Continuar con el proyecto

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Ejecución de preguntas tipo código

- Generar fichero python corrector
- Probar en linux
- Modificar el YAML para añadir el campo con el nombre del fichero corrector

---

Nº DE ACTA: 8

FECHA: 18-Febrero-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar avances del proyecto
- Debatir cómo ejecutar Python

DECISIONES TOMADAS:
- Ejecutar Python a través del intérprete
- Ejecución de Python con un hilo independiente para que no haya riesgo de que la aplicación se cuelgue
- Uso de funciones de generación/lectura de salida

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Llevar a cabo la ejecución de Python con todas las instrucciones decididas

---

Nº DE ACTA: 9

FECHA: 08-Marzo-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:

- Modificar salida de Python
- Debatir sobre el flujo a través de una lección (esperar respuesta correcta o no)

DECISIONES TOMADAS:

- Se decide no dejar avanzar al usuario hasta que no de la solución correcta a una pregunta
- Utilizar archivo JSON para devolver la salida de la función correctora
- Formato del archivo JSON
- Añadir un elemento Paginador para el flujo entre elementos de una lección

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN

- Crear el archivo JSON y su flujo de modificación para la comunicación entre la aplicación y la consola de Python.
- Modificar elementos visuales y añadir otros nuevos

Nº DE ACTA: 10

FECHA: 17-Marzo-2016

ASISTENTES:

- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:

- Comprobar que el paginador funciona
- Hablar redimensionado de componentes

DECISIONES TOMADAS:

- Cambiar tipo de paneles de las vistas para que la redimensión funcione
- Meter un coloreador de gramáticas en el campo de código

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN

- Llevar a cabo el redimensionado de elementos en las vistas
- Buscar información acerca de librerías de grmáticas

Nº DE ACTA: 11

FECHA: 31-Marzo-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Debatir sobre posibles librerías de gramáticas a usar
- Comprobar que todo funciona según lo deseado

DECISIONES TOMADAS:
- Usar librería ANTLR

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Reestructurar el código
- Cambiar rutas de los elementos

Nº DE ACTA: 12

FECHA: 19-Abril-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Problemas al generar el jar

DECISIONES TOMADAS:
- Estructura del proyecto

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Buscar información sobre Java Preferences API
- Hacer modificaciones e código (conflictos con linux)

Nº DE ACTA: 13

FECHA: 4-Mayo-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Hablar acerca de la memoria del proyecto
- Solucionar problemas

OBJETIVOS A CUMPLIR PARA LA PRÓXIMA REUNIÓN
- Tener el primer apartado de la memoria hecho
- Acabar las implementaciones

Nº DE ACTA: 14

FECHA: 27-Mayo-2016

ASISTENTES:
- Rafael Caturla
- Carlos Congosto
- Enrique Martín
- Manuel Montenegro

ORDEN DEL DÍA:
- Comprobar que todo está correcto
- Organizar mejor la memoria

DECISIONES TOMADAS:
- Todo está correcto
- No hacer más reuniones

B) Ejemplo de uso de la herramienta

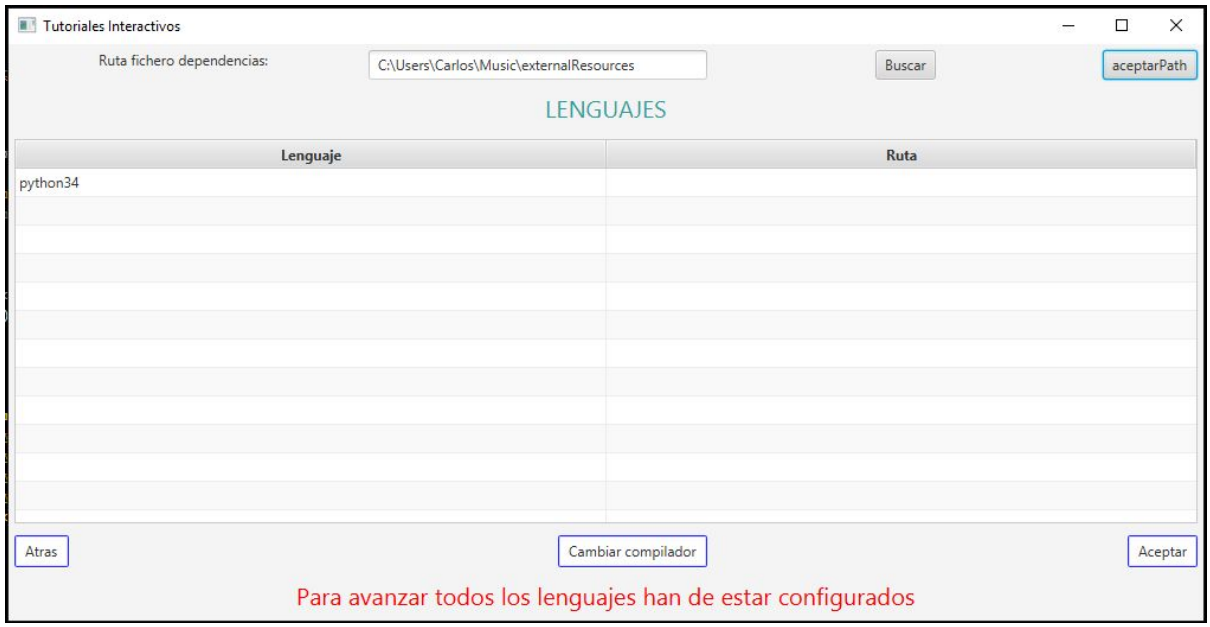
Tras la explicación sobre la implementación de la herramienta y las herramientas necesarias para llevar a cabo el producto final, ahora se va a pasar a explicar con un pequeño ejemplo cómo se utilizaría bien nuestra aplicación.

Partiendo de la base que la aplicación está instalada en el equipo y existe un fichero con los recursos externos a dicha aplicación, empezamos por arrancar la aplicación.

En el caso de ejecutar la aplicación por primera vez en el equipo o tener lenguajes sin configurar se mostrará la ventana de configuración.

Ventana de Configuración

Como se puede ver, no hay lenguajes disponibles y el campo con la ruta al fichero de dependencias está vacío. Para seleccionar el directorio de las dependencias basta con pulsar el botón de “Buscar” y nos aparecerá una ventana para seleccionar dónde está dicho directorio. Para confirmar que ese es el directorio se pulsa “aceptarPath”. Entonces aparecerán los lenguajes disponibles en la tabla.



Esta ventana también aparecerá cuando se arranque la aplicación con el fichero de dependencias seleccionado pero haya algún lenguaje que no lo esté. En ese caso aparecerán los que ya lo estén con su configuración y los que no como se muestra en este ejemplo con Python34.

Para configurar cada lenguaje hay que seleccionar el que se quiera configurar y a continuación pulsar “Cambiar compilador”. Nos volverá a aparecer otra ventana similar a la de selección del fichero de dependencias, pero esta vez se selecciona el compilador de dicho lenguaje que haya instalado en el equipo. Tras este paso la ventana se mostrará del siguiente modo:

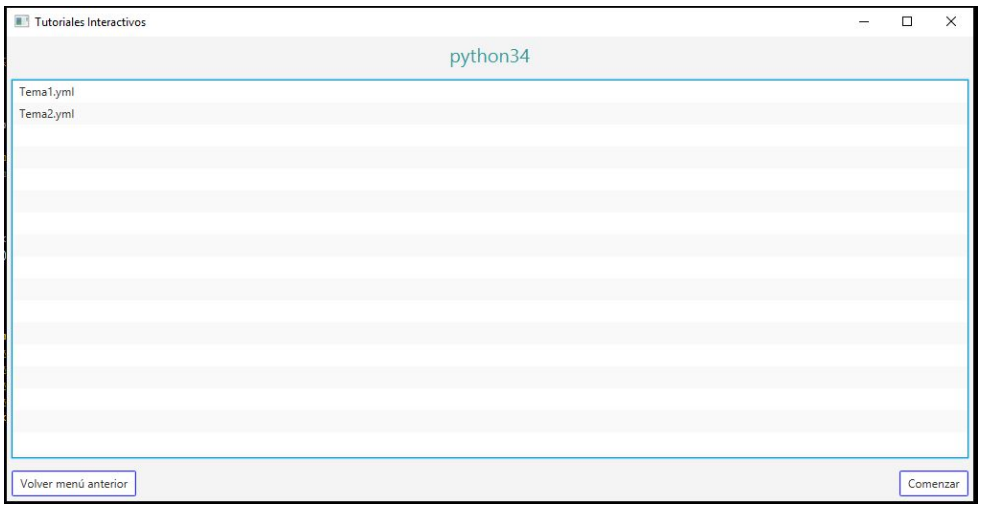
Una vez se tengan todos configurados, pulsando “Aceptar” se guardarán todas las configuraciones y se volverá a la pantalla de inicio. Si alguno de los lenguajes no está configurado no se podrá salir de esta ventana.

Si por el contrario la aplicación está bien configurada, cuando se lance aparecerá la ventana de inicio donde se muestra una lista de los lenguajes disponibles por la aplicación.

Ventana de inicio

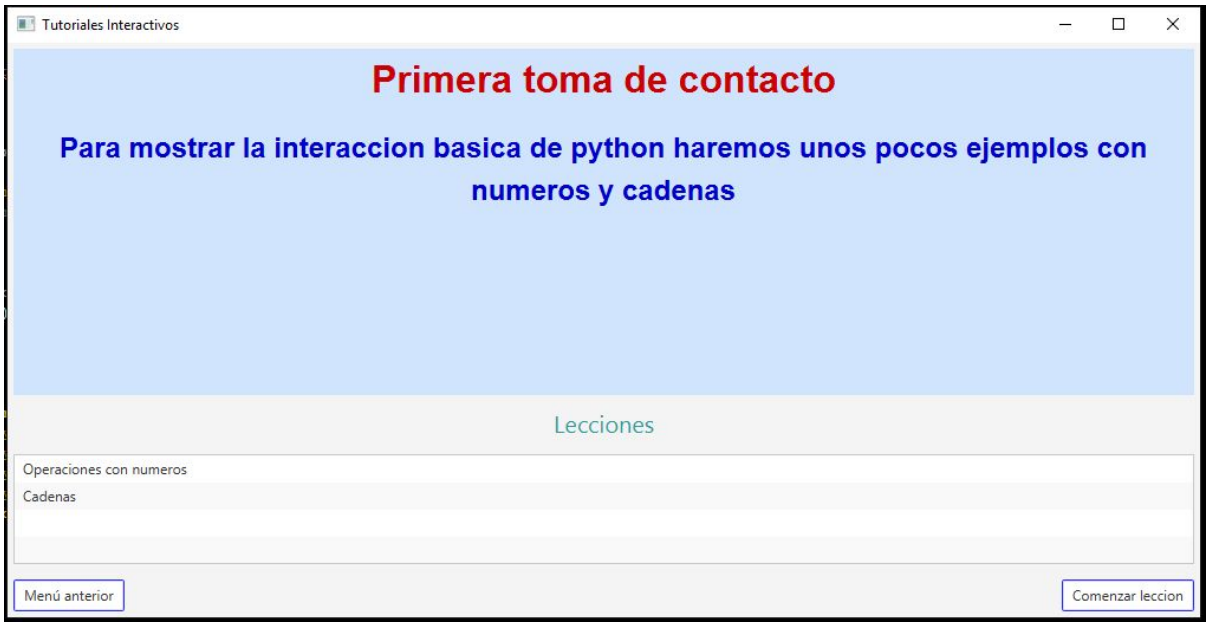
En esta ventana podemos ir a la ventana de ajustes o comenzar el tutorial tras tener seleccionado un lenguaje.

Tras la selección del lenguaje se mostrará una lista con los temas que componen el tutorial de este lenguaje.



Menú de temas

Una vez seleccionado el tema, se llega al menú de las lecciones.

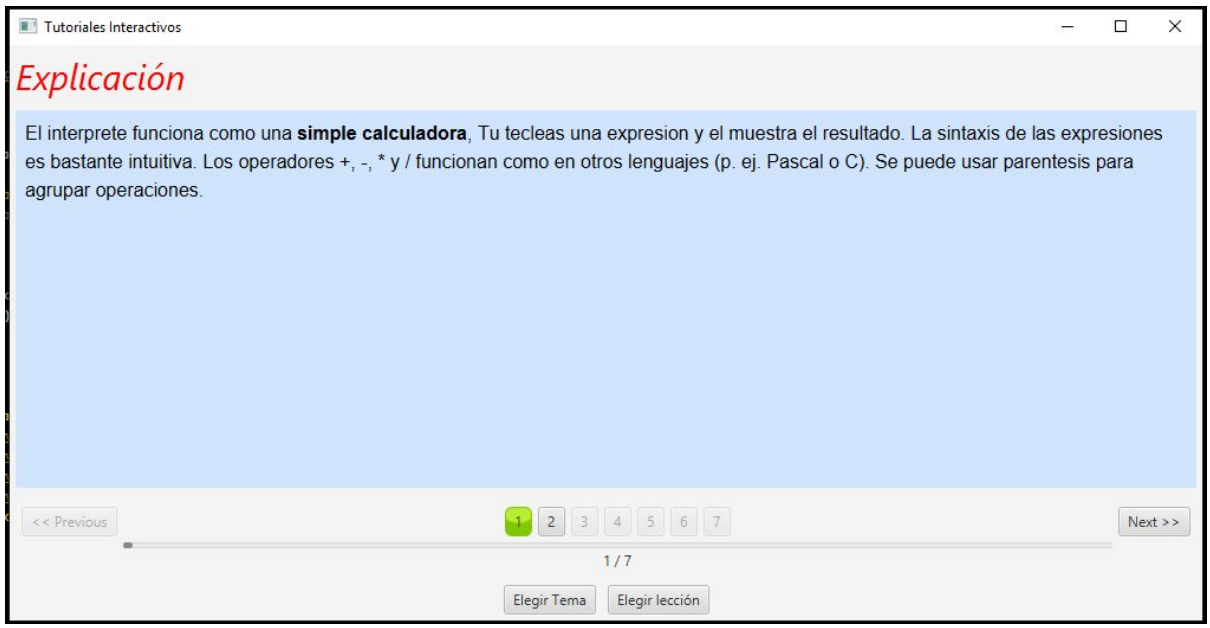


Menú lecciones

En esta ventana se muestra el título del tema junto a una pequeña introducción del tema, seguido de la lista de las lecciones.

Tras seleccionar una lección, y pulsar “Comenzar lección”, se muestra el contenido de dicha lección. El contenido consta de explicaciones o preguntas.

Si el contenido es una explicación se mostrará como en la siguiente imagen.



Contenido de explicación

Si por el contrario el contenido es una pregunta variará del tipo de pregunta que se trate. En caso de ser de tipo test:

Pregunta tipo test

Si las preguntas son de tipo código o tipo sintaxis tendrán la misma apariencia:

Preguntas tipo código o sintaxis

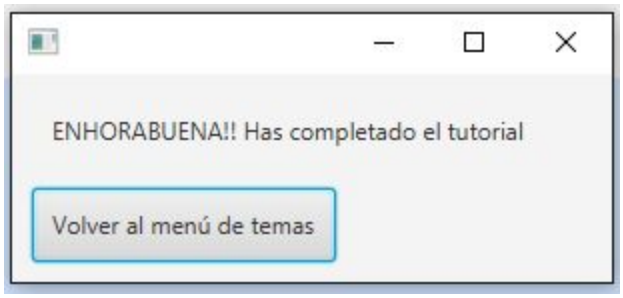
Ambas vistas de preguntas tienen un formato similar. En la parte central se muestra el enunciado de la pregunta y debajo el campo de respuesta que varía en función del tipo: lista de opciones en el caso del tipo test y en el resto un campo de texto donde escribir el código. Además, a la derecha del campo de respuesta siempre aparecen los mismos botones.

- **Resolver:** Para comprobar la respuesta introducida.
- **Ayuda:** Muestra un consejo en caso de existir para responder de forma correcta a la pregunta.

Tras contestar se mostrará un mensaje con el resultado de la pregunta. En caso de ser correcta, y mostrar el correspondiente mensaje, se activarán los botones necesarios para poder continuar con la lección. En caso contrario se mostrará el mensaje indicando que se ha fallado y no se activarán dichos botones.

Si la pregunta es de tipo código, además de aparecer el mensaje comunicando el error, se mostrará un botón con más información para ayudar a responder correctamente a la pregunta en función del error producido.

Tras acabar la lección por completo, se mostrará una ventana felicitando el trabajo y después la aplicación volverá al Menú de temas.



Felicitación

C) Obtención y compilación de la herramienta

Como ya se ha dicho anteriormente, esta herramienta está disponible en https://github.com/Kherdu/TFG/ bajo licencia MIT, para utilizarla es necesario descargarla y compilarla.

Una vez descargado como '.zip' se descomprime. A continuación se abre el terminal y desde la raíz del proyecto se ejecuta el comando `mvn package`. **mvn** es el nombre del ejecutable de maven en el equipo. Si no está configurado como una variable del sistema habrá que escribir la ruta completa de donde se encuentra.

Tras el paso anterior, en el directorio desde donde se ejecutó la instrucción aparecerá una carpeta '*target*', la cual contiene 2 archivos con la extensión '.jar'.

```
-TutorialesInteractivos-0.0.1-SNAPSHOT.jar
-TutorialesInteractivos-0.0.1-SNAPSHOT-jar-with-dependencies.jar
```

Para utilizar la herramienta es necesario ejecutar el segundo con la sentencia:

```
java -jar <nombre>
```