



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Desarrollo de un entorno de programación visual para el
aprendizaje de Java

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Carrillo Albero, Guillermo

Tutor/a: Vega Gisbert, Oscar

CURSO ACADÉMICO: 2022/2023

Resumen

Se pretende desarrollar un entorno en el que se pueda programar de manera visual en el lenguaje Java. La mecánica base sería un “*drag and drop*” de bloques que representen las distintas funcionalidades de Java. Por la complejidad de Java, lenguaje que lleva más de 25 años evolucionando, no se adaptará en su totalidad sino las funciones más importantes como la POO.

Este código visual será posteriormente traducido a Java por el entorno y ejecutado. También se tratará, previamente a la ejecución el indicar errores, a pesar de que el objetivo también será no dejar cabida a que el usuario cometa errores.

Con esto se pretende añadir una fase previa más amigable al aprendizaje del lenguaje Java, que puede resultar complejo para un nuevo aprendiz. Esto podrá resultar en una entrada más sencilla a este robusto lenguaje.

Palabras clave: Entorno, Java, programación visual, bloques, facilitar.

Abstract

The goal is to develop an environment to program Java in a visual way. The main mechanic will be “drag and drop” functional blocks featuring the iconic functions of Java language. We aren’t going to adapt the whole language since it has been evolving for more than 25 years. Instead we will be adapting only the more important features like OOP.

Lately this visual code will be translated to Java code and executed by the environment. An objective will be to make the language almost error proof. Even with that the idea is to mark the errors done previous to execution.

With this project we pretend to add a friendlier previous phase in the Java learning process. This is because it might be a little hard language to learn for a person who is getting into programming. This could result into an easier way to get into this robust language.

Keywords: Environment, Java, visual programming, blocks, ease.

Glosario

- Entorno de programación: Programa diseñado con el fin de ayudar a la programación.
- *Framework*: Conjunto de conceptos, prácticas y criterios estandarizados con los que se trabaja para resolver una problemática particular.
- Librería: Similar a un *framework*, ayuda a solucionar problemas específicos. Sin embargo, en vez de brindar las herramientas, brinda soluciones.
- Java: Lenguaje de programación orientada a objetos.
- JavaFX: *Framework* consistente en un conjunto de librerías que se puede añadir a Java y que permite la programación de interfaz gráfica.
- AnchorPane: Tipo de Nodo de JavaFX que actúa de lienzo permitiendo contener otros Nodos en él.
- ScrollPane : Tipo de Nodo de JavaFX también de tipo Panel que además de permitir contener Nodos añade barras de scroll.
- HBox: Tipo de Nodo de JavaFX de tipo Panel que posiciona sus nodos en una línea horizontal
- VBox: Tipo de Nodo de JavaFX de tipo Panel que posiciona sus nodos en una línea vertical.
- Import: Sentencia típica de ciertos lenguajes de programación por la que se añaden ciertas funcionalidades externas al archivo.
- JVM: Máquina virtual que permite la interpretación de código binario del lenguaje Java.
- Snippet: En programación, pequeño fragmento de código reusables y fáciles de integrar.

Índice

| | | |
|----------------|---|-----------|
| 1 | Introducción..... | 8 |
| 1.1 | Motivación..... | 8 |
| 1.1.1 | Motivaciones personales..... | 8 |
| 1.1.2 | Motivaciones profesionales..... | 9 |
| 1.2 | Objetivos..... | 9 |
| 1.2.1 | Objetivos generales..... | 9 |
| 1.2.2 | Objetivos específicos..... | 10 |
| 1.3 | Impacto Esperado..... | 10 |
| 1.4 | Metodología..... | 11 |
| 2 | Estado del Arte..... | 12 |
| 2.1 | Drag & Drop..... | 12 |
| 2.2 | Programación Visual..... | 12 |
| 2.2.1 | Programación visual educativa..... | 13 |
| 2.2.2 | EMF..... | 15 |
| 2.3 | Conclusiones..... | 16 |
| 3 | Análisis del problema..... | 16 |
| 3.1 | Requerimientos software..... | 16 |
| 3.2 | Ánalisis de posibles soluciones..... | 17 |
| 3.2.1 | Primera aproximación..... | 18 |
| 3.2.1.1 | Solución desde cero..... | 18 |
| 3.2.1.2 | Solución basada en librerías existentes..... | 18 |
| 3.2.2 | Segunda aproximación..... | 18 |
| 3.2.2.1 | Solución para PC..... | 19 |
| 3.2.2.2 | Solución para dispositivos móviles..... | 19 |
| 3.2.3 | Tercera aproximación..... | 19 |
| 3.2.3.1 | Solución web..... | 20 |
| 3.2.3.2 | Solución con aplicación..... | 20 |
| 3.2.3.3 | Puntualización..... | 20 |
| 3.3 | Solución elegida..... | 20 |
| 3.3.1 | Primera aproximación..... | 21 |
| 3.3.2 | Segunda aproximación..... | 21 |
| 3.3.3 | Tercera aproximación..... | 21 |
| 3.3.4 | Conclusiones..... | 21 |
| 3.4 | Diagrama de casos de uso..... | 22 |
| 3.4.1 | Descripción de los casos de uso..... | 22 |
| 3.5 | Plan de trabajo..... | 24 |
| 3.5.1 | Búsqueda de la temática del TFG..... | 25 |
| 3.5.2 | Planificación del tiempo..... | 25 |
| 3.5.3 | Imprevistos..... | 25 |
| 4 | Tecnologías utilizadas..... | 26 |
| 4.1 | Java..... | 26 |

| | |
|--|-----------|
| 4.2 Framework de la interfaz visual..... | 27 |
| 4.2.1 AWT..... | 27 |
| 4.2.2 Swing..... | 27 |
| 4.2.3 SWT..... | 28 |
| 4.2.4 JavaFX..... | 28 |
| 4.2.5 Apache Pivot..... | 29 |
| 4.2.6 QT Jambi..... | 30 |
| 4.2.7 GWT..... | 30 |
| 4.2.8 Conclusión..... | 31 |
| 5 Diseño..... | 31 |
| 5.1 Arquitectura del sistema..... | 32 |
| 5.2 Diseño detallado..... | 32 |
| 5.2.1 Interfaz..... | 33 |
| 5.2.1.1 Bloques de programación..... | 33 |
| 5.2.1.2 Vista principal..... | 34 |
| 5.2.1.2.1 Área de programación..... | 34 |
| 5.2.1.2.2 Área de creación de bloques..... | 35 |
| 5.2.1.2.3 Icono de borrado de bloques..... | 36 |
| 5.2.1.2.4 Botón de traducción de código..... | 36 |
| 5.2.1.3 Segunda vista..... | 37 |
| 5.2.1.3.1 Cuadro de edición de código..... | 38 |
| 5.2.1.3.1 Botón de guardado..... | 38 |
| 5.2.2 Controladores..... | 39 |
| 5.2.2.1 Controlador de bloque..... | 39 |
| 5.2.2.2 Controlador de la vista principal..... | 39 |
| 5.2.2.3 Controlador de la segunda vista..... | 39 |
| 5.2.3 DraggableMaker..... | 40 |
| 5.2.4 Expander..... | 40 |
| 5.2.5 Main..... | 41 |
| 6 Desarrollo de la solución propuesta..... | 41 |
| 6.1 Estructura de archivos..... | 41 |
| 6.2 Interfaz..... | 43 |
| 6.2.1 Bloques..... | 43 |
| 6.2.2 Vistas..... | 44 |
| 6.3 Controladores..... | 44 |
| 6.3.1 Controladores de Bloques..... | 45 |
| 6.3.2 Controlador de la Vista Principal..... | 45 |
| 6.3.2.1 Selectores..... | 45 |
| 6.3.2.2 Recorrido del Árbol..... | 45 |
| 6.3.2.3 Funciones de escritura..... | 46 |
| 6.3.3 Controlador de la Vista Secundaria..... | 46 |
| 6.4 DraggableMaker..... | 47 |

| | |
|--|----|
| 6.4.1 Evento de clic..... | 47 |
| 6.4.2 Evento de arrastrado..... | 47 |
| 6.4.3 Evento de soltado..... | 48 |
| 6.4.4 Problemas de scroll..... | 49 |
| 6.5 Expander..... | 49 |
| 6.5.1 Expandir la clase..... | 49 |
| 6.5.2 Contraer la clase..... | 51 |
| 6.5.3 Expandir los imports..... | 52 |
| 6.5.4 Contraer los imports..... | 52 |
| 6.5.5 Expandir la ventana..... | 52 |
| 7 Resultados finales..... | 53 |
| 7.1 Paso a aplicación de escritorio..... | 53 |
| 7.2 Traza de la ejecución normal del programa..... | 54 |
| 8 Pruebas..... | 59 |
| 8.1 Pruebas de caja blanca..... | 59 |
| 8.2 Pruebas de caja negra..... | 60 |
| 9 Conclusiones..... | 61 |
| 9.1 Relación de trabajo hecho con estudios cursados..... | 61 |
| 9.2 Futuras ampliaciones..... | 62 |
| 10 Referencias..... | 62 |

Índice de figuras

- Figura 1 Desarrollo Iterativo**
- Figura 2 Drag and Drop en Google Maps**
- Figura 3 Dibujo de la interfaz del programa HyperCard**
- Figura 4 Programa en LabView**
- Figura 5 Programa en Alice**
- Figura 6 Programa en AppInventor**
- Figura 7 Diagrama de clases hecho con EMF**
- Figura 8 Diagrama de casos de uso del entorno**
- Figura 9 Logo de Java**
- Figura 10 Logo de JavaFX**
- Figura 11 Logo de Apache Pivot**
- Figura 12 Logo de QT**
- Figura 13 Logo de GWT**
- Figura 14 Diagrama de Componentes del programa**
- Figura 15 Vista principal**
- Figura 16 Área de programación de la vista principal**
- Figura 17 Selectores del tipo de bloque de la vista principal**
- Figura 18 Icono de papelera de la vista principal**
- Figura 19 Icono de acción de la vista principal**

- Figura 20** Segunda vista
Figura 21 Cuadro de texto de la segunda vista
Figura 22 Icono de guardado de la segunda vista
Figura 23 Estructura de archivos del proyecto
Figura 24 Archivos .java del proyecto (no todos)
Figura 25 Imágenes del proyecto
Figura 26 Archivos FXML del proyecto (no todos)
Figura 27 Arrastrado de un bloque
Figura 28 Soltado de un bloque
Figura 29 Expansión de bloques contenedores
Figura 30 Contracción de bloque contraído
Figura 31 Expansión de área de imports
Figura 32 Comando para ejecutar el .jar
Figura 33 Archivos para la ejecución
Figura 34 Código de ejemplo para la traza
Figura 35 Área de programación con los parámetros correctos
Figura 36 Primeras sentencias del ejemplo
Figura 37 Programa visual resultante del ejemplo
Figura 38 Archivo temporal generado del ejemplo
Figura 39 Código dentro del archivo temporal del ejemplo
Figura 40 Segunda visita del ejemplo
Figura 41 Edición en el código traducido del ejemplo
Figura 42 Ventana de selección de carpeta
Figura 43 Archivo Java resultante de la traza

1 Introducción

Este proyecto tiene como finalidad la creación de un entorno de trabajo basado en la programación visual que ayude a facilitar la introducción de los nuevos programadores al lenguaje Java mediante un entorno más simplificado que traducirá su código visual a código Java.

1.1 Motivación

A continuación, paso a exponer las razones que me han llevado a elegir esta temática para hacer el trabajo de final de grado. Estas razones se dividirán en dos grupos, motivaciones a nivel personal, qué son las inquietudes que me despierta el tema y las segundas son las motivaciones profesionales, que se refieren a que aprendo haciendo este trabajo y por qué me puede servir.

1.1.1 Motivaciones personales

La motivación de este proyecto viene dada por la propia vivencia del aprendizaje de Java. Si bien este no es el lenguaje más complejo que un estudiante puede aprender sí que es de los más populares y demandados, y una de las principales opciones elegida tanto por estudiantes independientes como por centros educativos para la introducción al mundo de la programación. Sin embargo, Java, comparado con otros lenguajes como Python, ofrece una entrada ardua.

Un programa Java se ve obligado a contar por necesidad de un paquete, una clase y si se desea ejecutar algo una función main. Si se compara de nuevo con Python podemos observar que mientras Java necesita todo eso con Python se puede empezar a escribir código directamente. No es solo eso, sino que además Java cuenta con una asignación de tipos estática frente al tipado dinámico de Python entre otras cosas.

Por otro lado, y evitando comparaciones, la escritura de código en formato de texto puede ser algo molesta al principio. Cuesta diferenciar bien donde está cada cosa, es más sencillo ensuciar el código y cometer errores de escritura, si no se mantiene bien el mismo se puede volver un jeroglífico, etc. Estos aspectos no son de gran preocupación para programadores avanzados, pues hay mil maneras de hacer la escritura de código menos tediosa, sin embargo, para alguien novato estas ventajas pueden ser favorecedoras.

Todas estas razones motivaron este proyecto con el cual se pretende facilitar la entrada al lenguaje de programación Java guiando a los nuevos programadores todo el rato y delimitando las opciones que tienen para que aprendan bien su funcionamiento.

1.1.2 Motivaciones profesionales

A nivel profesional lo que motiva la redacción de un trabajo así es el deseo de aprender sobre ciertos tópicos. Por una parte, tenía cierto interés en el uso de ciertas estructuras de datos y por otra sobre los compiladores e intérpretes. Sin embargo, no encontré nada con lo que poder innovar en ese aspecto con mis conocimientos actuales. Es por eso que me decanté por una opción más original y que no escondiera tanta complejidad como tratar de innovar en campos tan estudiados como pueden ser los de las estructuras de datos o los de los compiladores.

Estos temas son de los más relevantes en la informática de hoy en día. El conocimiento sobre estructuras de datos es altamente demandado por las empresas más punteras del sector y no es para menos teniendo en cuenta la cantidad de datos que se generan hoy en día.

Por otra parte, mi intención también era utilizar alguna librería que me permitiera trabajar con interfaz gráfica por ser algo que no es muy tratado en el grado que he estudiado y por fortalecer mis conocimientos en áreas que no domino tanto.

Además, este trabajo puede abrirme los ojos sobre cómo es enfrentarse a un problema y puedo aprender a cómo enfocar futuros desarrollos.

1.2 Objetivos

A continuación, se procede a enumerar los distintos objetivos planteados para este proyecto. Se dividirán en dos partes, los generales y los específicos. El primer tipo corresponde a metas del proyecto que deberían cumplirse al final del mismo. El segundo tipo corresponde más a objetivos más concretos que en conjunto ayudarán a cumplir los objetivos generales.

1.2.1 Objetivos generales

Con la creación de este entorno hay ciertos objetivos generales. El primero y principal es facilitar el aprendizaje tanto del lenguaje de programación Java como de la programación de manera general. Esta es la razón de ser del proyecto y la meta a alcanzar. También se pretende crear una experiencia más amena que la que ofrece actualmente la programación escrita. La mejor manera de aprender es divirtiéndose y por eso este objetivo, para mejorar la experiencia y por tanto el aprendizaje.

A partir de esos dos objetivos más principales hay otros objetivos relacionados. Por ejemplo, está el objetivo de crear un programa muy intuitivo y sencillo de usar. Se tiene que tratar que el usuario sea capaz de utilizar este entorno sin ningún manual previo.

Ya hablando de objetivos más bien personales uno de ellos sería tratar de corregir errores antes de avanzar a la siguiente parte. Esta metodología de trabajo queda mejor explicada en su correspondiente apartado, pero se basa en tener siempre una versión funcional a la que se le irá implementando cosas en pequeñas iteraciones. La idea es reducir al máximo los fallos antes de avanzar de iteración. El otro objetivo

personal es el de aprender una tecnología de diseño de interfaz ya que este es un aspecto también importante en la industria.

1.2.2 Objetivos específicos

Los objetivos específicos son los que ayudarán a cumplir los objetivos generales. Primeramente, para lograr realizar los primeros objetivos generales está el objetivo de crear una interfaz basada en bloques arrastrables. En posteriores puntos se defenderá el *drag and drop* como mecánica para entornos educativos. Otro objetivo que ayuda a cumplir los principales objetivos generales es el de limitar las posibilidades del usuario prácticamente a solo las correctas. Este conjuntamente con la indicación de los errores ayudarán al usuario a fallar menos y por tanto frustrarse menos. También con el fin de hacer aprender el lenguaje se pretende facilitar la memorización de las distintas sentencias que forman Java. Eso y traducir el lenguaje visual posteriormente a Java, incluso permitir que este sea guardado en archivos .java.

Con el fin de ayudar a que el programa sea simple e intuitivo de utilizar se pretende lograr una interfaz que sea intuitiva además del uso de la mecánica del *drag and drop*. Otro objetivo es el uso de colores llamativos y representativos que también guíen. Además, el añadido de pocos botones en la interfaz para tratar de confundir lo menos posible.

1.3 Impacto Esperado

En cuanto a cómo podría afectar este producto hay que verlo desde distintas perspectivas. Lo primero es tratar quienes serían los potenciales usuarios. Como se ha mencionado ya este sería un entorno pensado para la introducción de nuevos usuarios a la programación y más especialmente al lenguaje Java. Si nos ponemos a ver esto desde la perspectiva de un programador con cierto conocimiento o inclusive avanzado este producto carece de interés puesto que no estaría tan pensado para grandes desarrollos ni para atajos de teclado que facilitasen la maestría y la agilidad. Se concluye entonces que los potenciales usuarios de este programa son los estudiantes más novatos de programación, así como sus mentores, que necesitarían conocerlo para enseñar.

Ahora hay que plantear el modelo de negocio de este software. En caso de que llegase a un estado suficientemente pulido para ser comercializado y teniendo en cuenta que ninguna empresa querría usar este programa habría dos alternativas. La primera es la venta de licencias de uso, tanto si el programa fuese de licencia de software propietario como si fuese de software libre, pero de pago. Posiblemente más centrada en la venta al por mayor a centros educativos que a personas individuales. La otra alternativa sería hacer que fuese de licencia libre y el programa fuese gratuito, aceptando donativos. Esta segunda opción sería la elegida puesto que la idea de este desarrollo no tiene un fin tanto comercial como más bien el de ayudar en el aprendizaje (similar a el AppInventor) [1], además un programa como el que se plantea desarrollar carece de interés comercial por como está planteado.

1.4 Metodología

Con respecto a la metodología utilizada se podría decir que la que más se asemeja es la metodología iterativa [2]. Esta consiste en un primer análisis de requisitos que posteriormente serán repartidos en distintas iteraciones. Es entonces cuando comienza el proceso de iterar. En cada iteración se procederá a desarrollar tan solo la parte acordada. Una vez acabado el desarrollo de esa parte se le realizarán pruebas, se analizará el resultado en búsqueda de nuevos objetivos y se documentará el proceso. Así hasta acabar todas las iteraciones. Todas las iteraciones ofrecen como resultado una versión funcional del programa pero que no cumple todos los requisitos finales además de un aprendizaje ganado para la próxima iteración.

Se trata de un tipo de metodología ágil [3]. Estas metodologías se caracterizan por tener un alcance variable y unos objetivos no definidos inicialmente. De esta manera, el programa puede ir evolucionando según el cambio de los requisitos adaptándose siempre a las situaciones. Se suele trabajar de manera iterativa en todas sus variantes por como está planteada esta metodología. Siempre ofrece una versión funcional en cada iteración que permite al cliente analizarla y cambiar requisitos.

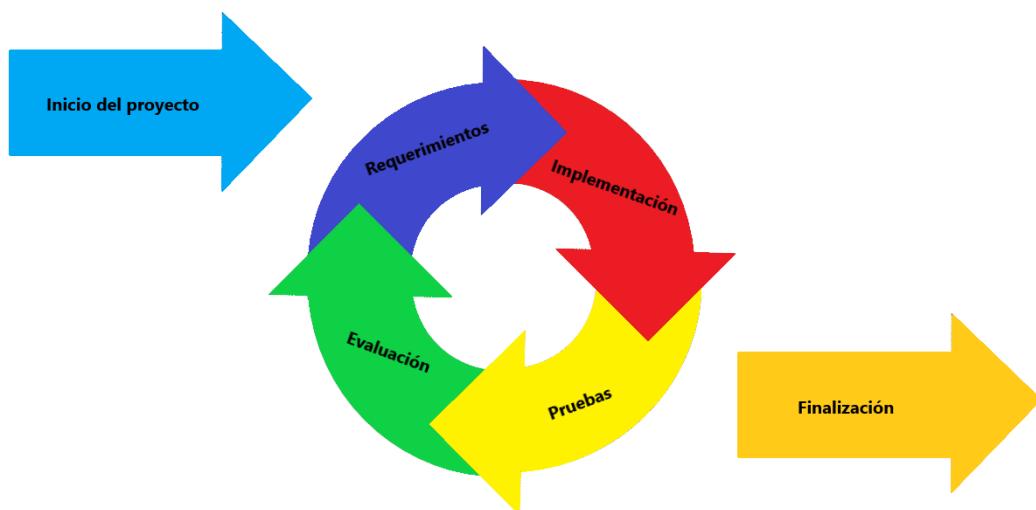


Figura 1: Desarrollo Iterativo

Este proceso es el más similar a como se ha afrontado el desarrollo de este proyecto. Primero desglosando las partes de este y posteriormente marcando objetivos de desarrollo independientes. Una vez acabados estos desarrollos haciendo pruebas para ver si cumplían el correcto funcionamiento y no siguiendo hasta que se arreglase la parte correspondiente. Inclusive documentando de manera simple lo desarrollado y lo que aún faltaba por desarrollar.

Gracias al uso de este tipo de desarrollo siempre se ha tenido un modelo funcional del programa en cada momento en vez de centrarse en partes individuales del código y tratar de acabarlas completamente. Esto ha beneficiado en encontrar errores tempranamente y así evitar bolas de nieve. También ha orientado en cómo debía proseguir el desarrollo de este software.

2 Estado del Arte

En este capítulo se hará un repaso de las distintas tecnologías y programas del área relacionada con este proyecto. Se hablará un poco de *drag and drop* y sobre todo de programas que utilicen la programación visual. Al final de este capítulo también se concluirá con una comparación con las tecnologías más similares a la idea del proyecto.

2.1 Drag & Drop

La existencia del *drag and drop* viene prácticamente ligada a la del ratón de ordenador. En los años 80 esta funcionalidad fue creada por los ingenieros de Apple [4] para simplificar y hacer más intuitivo el uso de los ordenadores de la época. Debido a su éxito, esta se ha ido añadiendo a distintos programas para facilitar su uso. Actualmente está muy presente a la hora de usar un pc. Los casos más familiares son las ventanas y los iconos del escritorio del PC. Otro caso muy conocido es la capacidad de arrastrar texto o imágenes desde ciertos programas a otros. Más ejemplos posibles de su uso actual los encontramos por ejemplo en Google Maps, que nos permite visualizar calles arrastrando y soltando un muñeco encima. Esta funcionalidad no ha cambiado mucho con los años a parte de añadirse a más aplicaciones.

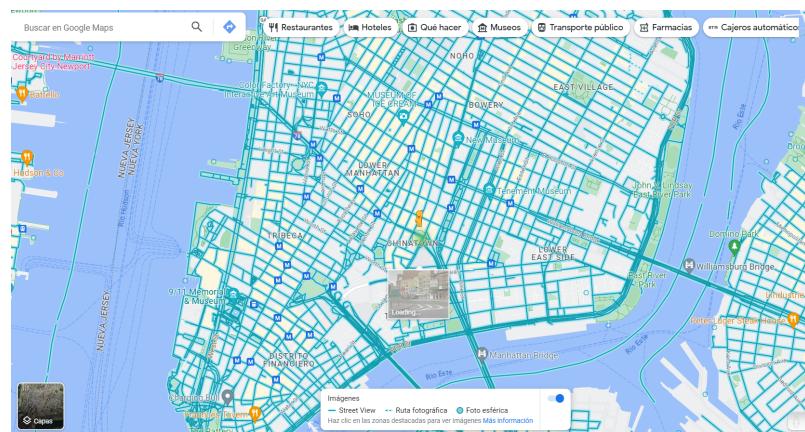


Figura 2: Drag and Drop en Google Maps

2.2 Programación Visual

La programación visual es una de las distintas maneras de crear programas con una semántica distinta. La diferencia es que con la programación visual se busca la facilitación de ciertos conceptos mediante su visualización y la gestión de eventos. Si bien la idea de programar de manera gráfica ya rondaba la cabeza de los programadores en las décadas de los 60 y 70, no sería hasta los 90 que este tipo de lenguajes aparecerían. En 1987 apareció lo que se podría considerar como el primer entorno de programación visual, HyperCard de Apple [5]. Hoy en día existen bastantes ejemplos distribuidos entre muchas funciones distintas.

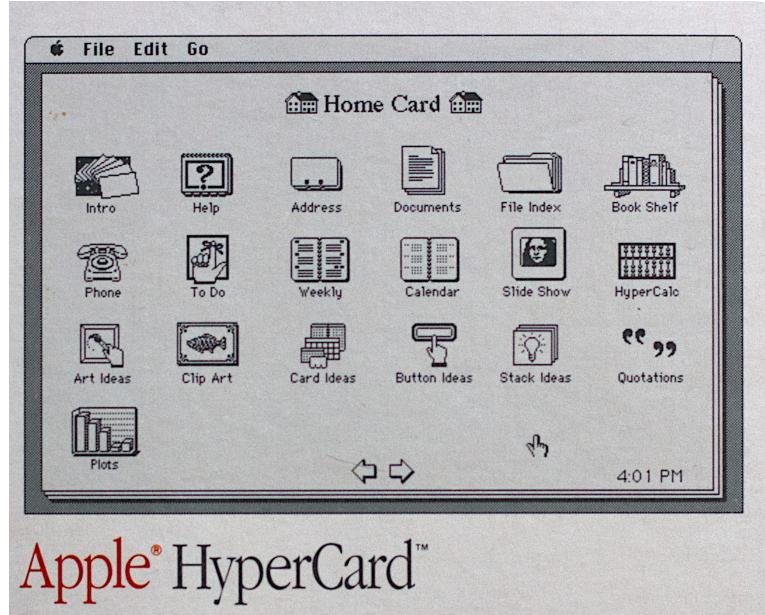


Figura 3: Dibujo de la interfaz del programa HyperCard [6]

Existen casos de programación visual en ámbitos como los videojuegos con Unity o Unreal Engine, que permite añadir ciertos componentes visualmente arrastrándolos, en entornos multimedia como Blender para modelado 3D, en entornos más centrados en la programación de microcontroladores como LabView o Ladder y sobre todo donde más lenguajes de este tipo encontraremos y en el área en la que nos vamos a centrar, en la educación [7].

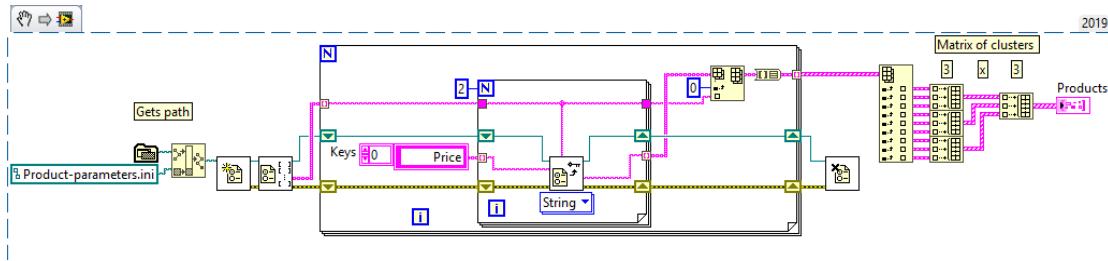


Figura 4: Programa en LabView [8]

2.2.1 Programación visual educativa

Al igual que la programación visual, los primeros programas educativos de este estilo aparecen alrededor de la década de los 90. Algunos lenguajes como Alice [9] o herramientas como AgentSheets [10] que nos permitían aprender a programar minijuegos o animaciones mediante personajes a los que se les aplicaba normas de comportamiento. Ya en estos primeros lenguajes se empezó a utilizar la fórmula más común en este sector, el *drag and drop* de bloques que representan acciones generalmente. Otro que surgió en la época como puede ser EToys [11] permitía incluso la personalización del personaje al que se maneja.

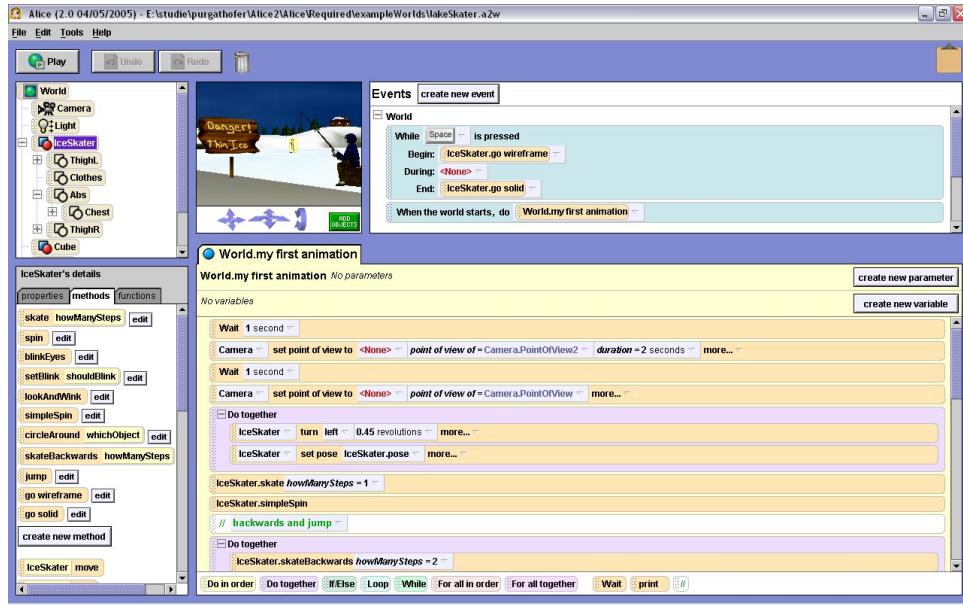


Figura 5:Programa en Alice [12]

En los tempranos 2000 apareció el lenguaje más popular de este estilo, Scratch [13]. Muy similar a lo que ya se ha comentado, Scratch también se basa en arrastrar bloques para programar juegos y animaciones, pero con un estilo más moderno y muchas más posibilidades.

No será hasta la siguiente década que surjan muchas nuevas propuestas novedosas. En el año 2010 AppInventor, un entorno de programación educativo de aplicaciones móviles verá la luz. Este permite arrastrar componentes de la aplicación y programarlos con una sintaxis mucho más similar a la de cualquier lenguaje de programación. En esta década también aparecen lenguajes visuales basados más en diagramas de estados que en bloques arrastrables como por ejemplo Flowgorithms [14] o Raptor. Otro tipo de lenguajes que surgieron fueron los que siendo educativos iban más orientados a el aprendizaje junto con microcontroladores como Arduino. Sin ir más lejos hay una extensión de Scratch para Arduino llamada MBlock. Inclusive saldrán entornos de programación educativos para plataformas móviles como Catrobat [15] o Hopscotch.

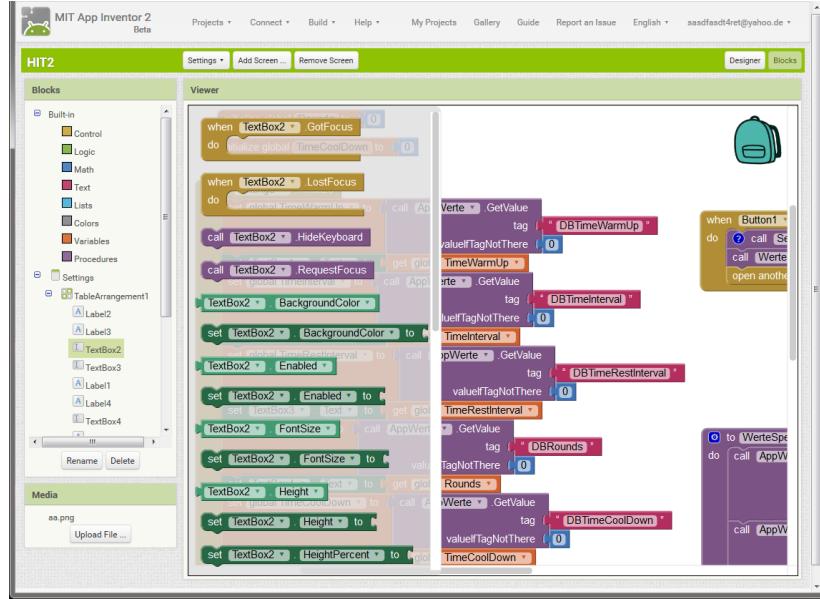


Figura 6: Programación en AppInventor [16]

2.2.2 EMF

Cabe la mención de una herramienta que no entra dentro del apartado de programa educativo. Esta herramienta es EMF [17] o el *framework* de modelado de Eclipse. Se trata de un *framework* de modelado que facilita la creación de código de Java. Esta se basa en especificaciones de modelos en XMI, un lenguaje de intercambio de modelos UML entre varias plataformas [18]. En base a estos modelos es capaz de traducir a clases Java.

Se está destacando este *framework* en concreto por la gran similitud con la idea propuesta. De una manera visual como pueden ser los diagramas UML nos permite programar clases, con sus métodos, sus atributos e incluso relaciones entre ellas. Luego todo esto sería traducido a Java tal y como se plantea este proyecto.

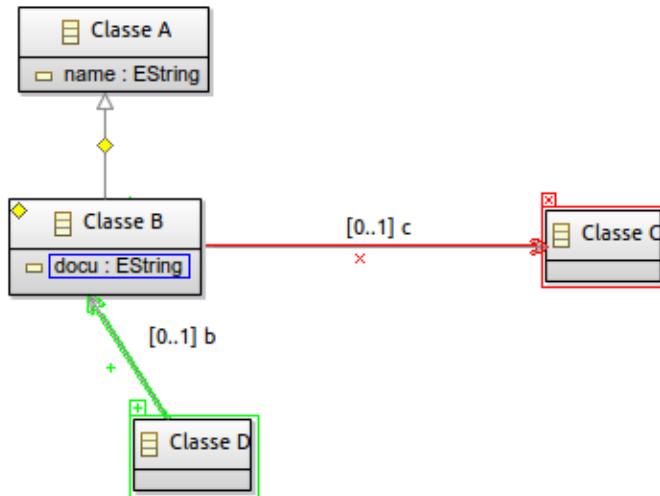


Figura 7: Diagrama de clases hecho con EMF [19]

2.3 Conclusiones

Se ha podido comprobar que en el ámbito de la educación la elección del drag and drop y de la programación visual no solo son una muy buena idea por todas las ventajas que ofrecen, sino que además son de las opciones más populares. En este apartado se ha recogido solo unos de los muchos ejemplos que se pueden encontrar para ayudar a dar contexto.

Además, estos lenguajes, plataformas, herramientas, etc servirán de referencia a la hora de elaborar el entorno objetivo. Sobre todo, AppInventor por su proximidad más a un lenguaje de programación y no a una serie de acciones además de por el diseño y también Flowgorithms puesto que, aunque no se haya comentado este entorno puede traducir los diagramas con los que se programa a distintos lenguajes de programación. Se tomará en cuenta también EMF como una de las aproximaciones más cercanas a nuestro objetivo en concreto.

De esta manera se podría llegar a entender este proyecto como una combinación entre el estilo de AppInventor y ciertas funcionalidades de Flowgorithms aplicado al lenguaje Java.

3 Análisis del problema

En este apartado se analizará el problema para posteriormente pasar a diseñar varias soluciones. Este análisis constará de varias partes que abordarán la temática desde distintos aspectos.

3.1 Requerimientos software

Para obtener detalles del problema vamos a recabar los distintos requerimientos del mismo. En este caso lo haremos teniendo en cuenta dos tipos, los funcionales y los no funcionales. Los requerimientos funcionales son la descripción de lo que el software debe hacer y no hacer. Los requerimientos no funcionales no tienen tanto que ver con el objetivo del proyecto sino con el sistema en sí.

Requisitos funcionales

El programa ha de contar con bloques que representen las distintas sentencias Java

Los bloques han de poder ser arrastrados y soltados con el ratón

Si un bloque se arrastra encima de otro y existe la posibilidad, este primero se ha de introducir dentro del segundo (anidado)

Ha de haber un selector que permitirá elegir el tipo de bloque que se quiere añadir, esto hará que este bloque aparezca en pantalla

Los bloques se deben poder borrar fácilmente, causando este acto que desaparezca de pantalla

La pantalla tiene que ser scrollable para poder ver bien todo el programa creado

| |
|---|
| Cuando un bloque se introduzca dentro de otro el bloque contenedor ha de crecer en dimensiones para abarcar al otro |
| Un bloque ha de poder ser arrastrado fuera de otro que lo contenga |
| Si un bloque es arrastrado fuera del otro que lo contenía este último ha de adaptarse a esto decreciendo lo necesario |
| El bloque arrastrado ha de estar siempre visible |
| Se han de crear divisiones en los tipos de bloques según su función y distinguirlos mediante su color |
| Habrá un botón que traduzca el código a Java |
| Se controlarán errores al momento de traducir el código |
| No se ha de permitir traducir el código si los errores son detectados |
| Marcar los errores detectados con un aviso |
| Una vez traducido el código este debe poder ser editable antes de guardarlo |
| Ha de haber un botón que permita guardar el código en un archivo Java |
| Aparecerá un menú emergente que permita seleccionar donde guardar el archivo |
| El código se ha de traducir indentado |

Requisitos no funcionales

| |
|---|
| El programa ha de funcionar correctamente y sin errores que impidan su uso |
| La velocidad de ejecución de las acciones en el programa no ha de ser muy elevada |
| Las acciones han de mostrar feedback |
| La funcionalidad de un bloque ha de ser reconocible a simple vista |
| La interfaz ha de ser intuitiva y simple |
| Los textos dentro del programa han de ser visibles y agradables |

3.2 Análisis de posibles soluciones

Una vez planteados todos los requisitos de nuestro entorno es el turno de pensar en soluciones que puedan cumplir estos requisitos. De este proceso surgirán varias soluciones y aproximaciones que tratarán de satisfacer los requisitos. Posteriormente de este grupo de soluciones se tendrá que decidir cuál es la mejor.

3.2.1 Primera aproximación

Para crear un entorno con las características y requisitos demandados no tan solo tenemos una manera de proceder. Existe la posibilidad de crear todo de cero o basarnos en alguna tecnología existente. Hay que valorar los pros y contras de ambas posibilidades.

3.2.1.1 Solución desde cero

En este tipo de solución se tiene que trabajar todos los aspectos requeridos sin ningún tipo de base. Estos aspectos abarcan la creación de los bloques, las mecánicas de *drag and drop*, el comportamiento implícito de un bloque cuando se hace el *drag and drop*, el anidado de bloques dentro de otros, etc. Dependiendo de las tecnologías elegidas este tipo de solución puede llegar a ser extremadamente complicada e incluso imposible. Sin embargo, con otras tecnologías podría resultar perfectamente posible y tan solo algo laboriosa.

La principal ventaja de este tipo de solución es la total y completa personalización en casi todos los aspectos. Tanto en el diseño de bloques como en el comportamiento detallado de las principales mecánicas. También provoca el mejor entendimiento del programa al final de su desarrollo ya que se ha tenido que desarrollar enteramente.

Como desventajas podemos encontrar que es una solución más complicada de realizar. Además, al tener que crear todo este proceso puede dar lugar a más fallos.

3.2.1.2 Solución basada en librerías existentes

Existen librerías y *frameworks* existentes que facilitan mucho el desarrollo de lenguajes visuales. El mayor ejemplo y referente en este proyecto serían el conjunto de librerías que conforma Blockly [20]. Este mismo se usó en el desarrollo de AppInventor y muchos otros lenguajes educativos. Cuenta con una apariencia muy similar a scratch.

Las ventajas de utilizar una tecnología así son el hecho de tener gran parte del trabajo hecho. Gracias a estas librerías el programador se puede centrar en la parte lógica del programa y evitar la gráfica. Además, como se comentó antes se pueden evitar errores provenientes de crear todo desde cero.

Como desventajas tendríamos la falta de una mayor capacidad de personalización. También la obligatoriedad de tener que adaptarse a las condiciones que demande esta librería.

3.2.2 Segunda aproximación

Otro enfoque para realizar este proyecto es en que plataforma se pretende utilizar. Hoy en día una gran mayoría de dispositivos electrónicos tienen la posibilidad de interactuar mucho con el usuario. Un ejemplo son las pantallas táctiles más modernas de los vehículos, que posibilitan muchas funciones añadidas y la visualización sencilla de los datos del vehículo. Si bien pensar en este programa para el caso de un vehículo no es muy realista, aún existen otras posibilidades que dan más debate como pueden

ser ordenadores, teléfonos móviles o tabletas. Se va a razonar para que plataforma sería más conveniente desarrollarlo.

3.2.2.1 Solución para PC

Cuando se habla de un ordenador personal se pueden dar por asumidas ciertas prestaciones. Una pantalla más grande, o incluso más de una pantalla. Una gran capacidad de disco duro. Unas mayores prestaciones y velocidad de procesador, de memoria, etc. Interacción mediante un teclado físico y un ratón físico.

Estos periféricos y prestaciones nos aportan ciertas ventajas. La primera más espacio para visualizar todo el programa. También unos periféricos más precisos que una pantalla táctil. Además, al hablar de una potencia mayor el programa puede ser más complejo y potente y no necesita vigilar tanto la optimización [21]. Otra ventaja es el trabajar directamente en ordenador, donde futuramente se programará.

Como desventajas se puede encontrar su mayor precio o su menor transportabilidad. Otra desventaja frente a los dispositivos móviles es que la pantalla táctil crea una interacción más cercana que el ratón.

3.2.2.2 Solución para dispositivos móviles

En cambio, cuando se habla de dispositivos móviles se presupone una pantalla más pequeña que un ordenador. Menor tamaño de disco, aunque actualmente no sea tan decisivo este factor en teléfonos modernos. Una menor capacidad de procesamiento y menor memoria. Además, la interacción con un dispositivo móvil actualmente se basa casi completamente en su pantalla táctil y puede que algún botón.

Las ventajas que pueden aportar son en su mayoría por la movilidad y disponibilidad de un teléfono móvil. Este tipo de dispositivos son muy comunes de tener y dominar por parte de la juventud en países desarrollados [22]. Además, la interacción con una pantalla táctil es más cercana.

Sin embargo, también tiene varias desventajas. La principal y más grande es el tamaño de pantalla. Al tratarse de una pantalla mucho menor los elementos han de aprovechar mejor el espacio e igualmente es muy probable que estorben en la pantalla. También hay que tener en cuenta las menores prestaciones de un móvil frente a un ordenador. Además, la precisión a la hora de manejar una pantalla táctil frente a un ratón y un teclado es mucho menor.

3.2.3 Tercera aproximación

La última aproximación que se va a tratar se centra en una temática que es tendencia. Se ha comentado anteriormente en el apartado de estado del arte que muchos lenguajes visuales no tienen programa, sino que se utilizan en web. Inclusive actualmente la tendencia se acerca a esa temática y podemos ver ejemplos como los programas de Microsoft Office que se pueden utilizar online actualmente. Es por eso que en esta aproximación se verán las ventajas de la web y de un programa.

3.2.3.1 Solución web

Una solución web implica la publicación del programa en un dominio web accesible a todo el mundo. Al entrar a este dominio web los propios usuarios interactuarían con el programa y crearían su código Java igualmente.

La gran ventaja de esta solución es la simple y rápida accesibilidad desde cualquier sitio al programa. Esto también evitaría tener que descargar el programa.

Los contratiempos de esta alternativa son varios. Hace falta alojar el sitio web en un servidor al que accederán los usuarios. Hay que controlar lo concurrido del sitio web y según eso adaptar las capacidades del servidor. La decisión web casi nos forzará a utilizar HTML, CSS y JavaScript en vez de poder buscar otras alternativas [23].

3.2.3.2 Solución con aplicación

Por otra parte, una solución con programa implica la creación de un programa descargable que los futuros usuarios deberán descargar e instalar en su dispositivo. Entonces el programa será de fácil acceso en el equipo.

Las ventajas de esto es que una vez instalado ya es muy sencillo de acceder y no depende de factores externos como la conexión a internet o que el servidor que aloja la web haya caído. Además, al no interactuar con elementos de la red es más seguro que la implementación web ya que todo se queda en local.

Como desventajas está la de tener que desarrollar para todos los sistemas operativos. Sin embargo, en este aspecto Java por ejemplo nos ofrece la ventaja de ser entendido por cualquier plataforma con la JVM. También la descarga e instalación, pasos que complican la primera toma de contacto con el programa.

3.2.3 Puntualización

Finalmente hay que comentar que estas dos últimas aproximaciones no son tan relevantes ya que se pueden crear distintas versiones para distintas plataformas e inclusive web. Es algo que este proyecto no cubrirá por su alcance, pero hay que tenerlo en cuenta como posibilidad. La problemática que podría plantear esto es que tecnologías se utilizarían para cada caso, esto obviamente complicaría su desarrollo por tener que usar una tecnología distinta para cada versión. Con respecto a las ventajas que nos ofrecería estarían todas las ventajas anteriormente dichas además de un mayor alcance y facilidad de acceder a utilizar el entorno.

3.3 Solución elegida

En el apartado anterior se han expuesto varias aproximaciones a seguir a la hora de analizar que solución era más conveniente. Además, se han dicho los pros y los contras de las distintas opciones. En este apartado se van a explicar y justificar cuáles han sido las soluciones elegidas. A pesar del punto de puntualización sí que se va a elegir opción en las últimas aproximaciones por considerar que cubrir todas las opciones sería abarcar más del alcance que se plantea para este proyecto.

3.3.1 Primera aproximación

En la primera aproximación se comparaba el hecho de utilizar *frameworks* y librerías preexistentes especializados con el de crear un entorno desde cero. Si bien se comentó que era muy popular el uso de ciertas librerías preexistentes y que estas podrían facilitar las cosas en este proyecto se prefiere la creación desde cero del entorno. Las razones son la mayor capacidad de personalización que puede otorgar al entorno una mayor personalidad y distintividad a la par de dar la posibilidad de hacer todas las alteraciones que se crean convenientes.

3.3.2 Segunda aproximación

En la segunda aproximación se hablaba del desarrollo para plataformas móviles o para ordenadores. Ambos ofrecían sus ventajas con respecto a campos distintos. Por ejemplo, el ordenador con facilitar el uso del programa mientras que el teléfono con la accesibilidad. Sin embargo, en este proyecto hay ciertos objetivos más importantes, además de que ya se ha comentado cuál sería el público objetivo por lo que el alcance no tendría tanto sentido. Es por eso que la opción de ordenador sería la elegida. Esta opción da más posibilidades y facilidades a la hora de su uso y de su desarrollo por su mayor tamaño y facilidad de uso.

3.3.3 Tercera aproximación

En la tercera aproximación se comentaba la posibilidad de un entorno web frente a una aplicación de escritorio o de móvil. Ambos con sus ventajas y desventajas a niveles de alcance. A nivel de usabilidad ambos pueden alcanzar niveles similares ya que la plataforma no afectaría tanto. Se va a preferir utilizar la versión de programa. Las razones son que a pesar de que el entorno web ofrezca mucha accesibilidad también trae ciertos niveles de complejidad que pueden distraer de los objetivos principales. Por otra parte, la versión de programa tan solo necesita una instalación que teniendo en cuenta el público objetivo no es tan dramático.

3.3.4 Conclusiones

Tras la elección de las soluciones que se han considerado más adecuadas se ha determinado que el resultado final ha de ser desarrollado desde cero sin uso de un *framework* como Blockly. Este desarrollo ha de dar como resultado un programa de escritorio que cumpla los requisitos especificados en pasos anteriores. Teniendo en cuenta que esta será la solución buscada los pasos siguientes del proyecto serán ejecutados teniendo en mente siempre esta meta final.

3.4 Diagrama de casos de uso

Los diagramas de casos de uso son un tipo de notación específica del lenguaje de modelado UML. Esta notación sirve para describir gráficamente los casos que se pueden dar a la hora de que un actor interactúe con un sistema. Esto puede servir para visualizar mejor el sistema que se ha de desarrollar y entender mejor los requisitos.

En este apartado se van a mostrar el diagrama de los distintos casos de uso que pueden ocurrir durante el uso del entorno. Para ello se tomará como base los requisitos definidos anteriormente.

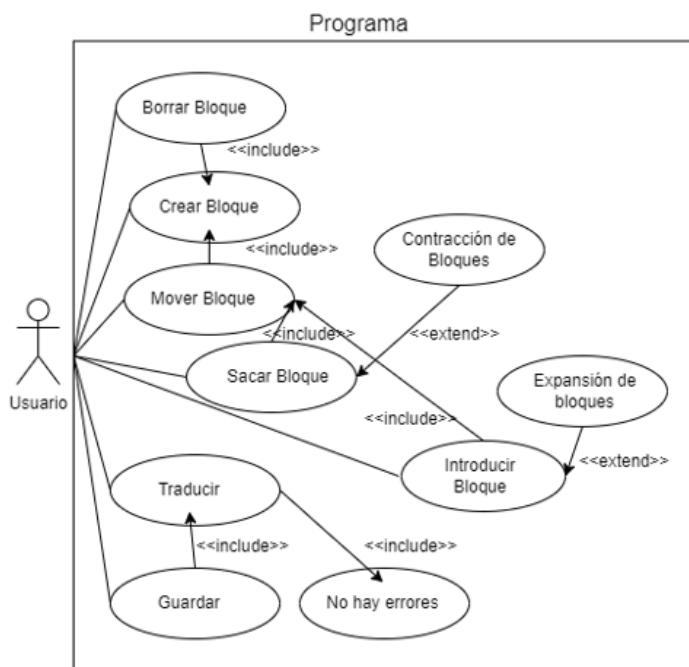


Figura 8: Diagrama de casos de uso del entorno

3.4.1 Descripción de los casos de uso

| | |
|-----------------------|---|
| Nombre | Crear Bloque |
| Evento | Elegir un bloque en el selector de bloques |
| Resumen | Selección de bloque para poder crearlo e interactuar con él |
| Pre-condición | - |
| Post-condición | Aparece el bloque arrastrable |
| Extensões | - |

| | |
|-----------------------|---|
| Nombre | Borrar Bloque |
| Evento | El bloque se arrastra a la basura |
| Resumen | Arrastrado de bloque a la basura para eliminarlo de la pantalla |
| Pre-condición | - |
| Post-condición | El bloque es borrado |
| Extensiones | - |

| | |
|-----------------------|--|
| Nombre | Mover Bloque |
| Evento | Clicar y arrastrar con el ratón el bloque |
| Resumen | Al clicar y arrastrar con el ratón el bloque se mueve con él |
| Pre-condición | - |
| Post-condición | Cambio de posición del bloque |
| Extensiones | - |

| | |
|-----------------------|--|
| Nombre | Introducir Bloque |
| Evento | El bloque se posiciona encima de otro con zona contenedora |
| Resumen | El bloque se introduce dentro de otro que tiene zona contenedora |
| Pre-condición | Mover el Bloque encima de otro con una zona contenedora |
| Post-condición | El bloque se introduce dentro de otro |
| Extensiones | El bloque contenedor se expande si es necesario |

| | |
|-----------------------|--|
| Nombre | Expansión de bloques |
| Evento | Se ha introducido un bloque en otro |
| Resumen | Expansión de los bloques contenedores para hacer hueco al nuevo bloque |
| Pre-condición | Se ha introducido un bloque |
| Post-condición | Los bloques contenedores se expanden |
| Extensiones | - |

| | |
|-----------------------|--|
| Nombre | Sacar Bloque |
| Evento | El bloque se retira de otro con zona contenedora |
| Resumen | El bloque se extraerá de otro con zona contenedora |
| Pre-condición | Que el bloque estuviese en un bloque contenedor. Que se arrastre fuera |
| Post-condición | El bloque se saca del que lo contenía |
| Extensiones | El anterior bloque contenedor se contraerá si es necesario |

| | |
|-----------------------|---|
| Nombre | Contracción de Bloques |
| Evento | Se ha sacado un bloque de otro |
| Resumen | Contracción de los bloques contenedores para reajustarse a los bloques contenidos |
| Pre-condición | Se ha sacado un bloque de otro |
| Post-condición | El bloque contenedor se contrae |
| Extensiones | - |

| | |
|-----------------------|---|
| Nombre | Guardar |
| Evento | Clic al botón de guardado |
| Resumen | Se guarda en un archivo .java el código traducido |
| Pre-condición | Que el código se haya traducido |
| Post-condición | Creación de archivo .java con el código traducido |
| Extensiones | - |

| | |
|-----------------------|---|
| Nombre | Traducir |
| Evento | Clic al botón de traducción |
| Resumen | El código visual se traduce a código java |
| Pre-condición | El código visual no tiene errores |
| Post-condición | Código java resultado del código visual |
| Extensiones | - |

| | |
|-----------------------|---------------------------------------|
| Nombre | No hay errores |
| Evento | Clic al botón de traducción |
| Resumen | La disposición de bloques es correcta |
| Pre-condición | - |
| Post-condición | Permite la traducción a código Java |
| Extensiones | - |

3.5 Plan de trabajo

Para realizar un proyecto serio debe existir una organización de recursos cuidadosamente planificada. Tanto los recursos temporales como económicos han de ser planificados con el fin de buscar la eficacia. En este apartado se explicará al detalle el plan de trabajo seguido durante los meses que ha durado este proyecto.

3.5.1 Búsqueda de la temática del TFG

Esta primera etapa duraría desde principios de febrero hasta su mitad. En primer lugar, se empezó el TFG buscando la temática del mismo. Se consultó profesores para ver sus ofertas, sin embargo, no se logró encontrar demasiado. Es por eso que, a mediados de febrero, después de buscar varias ideas posibles, se consultó con el profesor tutor la viabilidad de las mismas. Tras el visto bueno de una de ellas empezó el proceso de elaboración del TFG.

3.5.2 Planificación del tiempo

Lo primero que se hizo al empezar el TFG fue fijar un horario. El TFG comenzó el día 20 de febrero. La intención era entregarlo para presentar en julio lo que deja como fecha máxima el 7 de julio para entregarlo. Sin embargo, la idea era no acercarse tanto a la fecha límite. Es por eso que se contará con unos 130 días extensibles hasta 137 en caso de tener que llegar al último día.

La elaboración de un TFG son 12 créditos, es decir entre 300 y 360 horas. Estas horas serán divididas en tres partes del proceso. Primero la investigación, que se extenderá las primeras semanas con el fin de empezar sobre una base sólida. La segunda, el desarrollo del programa cuya idea era acabar entre el 1 y 15 de mayo. Y por último la elaboración del documento que contaría con el tiempo restante. A la investigación se le dedicaría 2h diarias llegando a sumar de unas 28 a 42 horas. Por otra parte, al desarrollo se le dedicaría 3 h diarias extensibles. Teniendo en cuenta los plazos acordados esto rondaría las 190 h. Finalmente quedarían aproximadamente unas 100 h para el documento al que se le dedicaría también 3h diarias.

3.5.3 Imprevistos

A pesar del plan comentado anteriormente durante el transcurso de un proyecto muchos imprevistos pueden surgir. Estos contratiempos sumarán tiempo a las partes del proyecto haciendo que aparezcan retrasos que lo lastrarán.

Sin ir más lejos la parte de la investigación, planificación y puesta a punto tardó un mes completo. Esto se debió a varios factores como las decisiones creativas a tomar sobre cómo enfocar el entorno, las múltiples opciones para hacer el desarrollo o los problemas e incompatibilidades al poner a punto el entorno de programación usados. Es por eso que en total tomó 60h toda esta sección.

El retraso anteriormente mencionado junto a más retrasos durante el desarrollo, al tratarse de una tecnología jamás usada anteriormente llevaron a más retrasos en esta parte. Además, durante este mismo desarrollo hubo cambios del enfoque planteado inicialmente sobre cómo llevar a cabo ciertas partes del entorno. Todo esto hizo que el desarrollo terminase el 31 de mayo. Esta fecha también se puso para no alargar de más este desarrollo. En total esta parte serían unas 215 h.

Finalmente queda el documento al cual habrá que dedicarle todo el mes de junio. Si tomamos el ritmo de trabajo de 3h anteriormente planteado y lo multiplicamos por los 30 días de junio esto nos daría unas 90 h, ligeramente inferiores a lo planteado inicialmente y justamente las necesarias para completar la duración natural de un TFG.

4 Tecnologías utilizadas

Una de las muchas decisiones que se deben tomar a la hora de hacer un desarrollo software es qué tecnologías o conjunto de tecnologías conformarán la solución al problema que se plantea. En este apartado se va a hacer una investigación de las distintas opciones y una elección teniendo en cuenta los requisitos, objetivos y fundamentos del proyecto.

4.1 Java

La decisión de uso de Java en este trabajo no viene motivada por ninguna razón técnica. Si bien es verdad que al tratarse de un lenguaje de programación orientado a objetos este aspecto nos podría resultar útil para ciertas clases, este no es el caso. El uso de Java para programar este entorno viene motivado por la naturaleza del proyecto. Al tratarse de un programa educativo con finalidad de enseñar a los programadores novatos el propio lenguaje Java de manera simplificada, ¿qué mejor lenguaje para inspirar a los estudiantes que el mismo que están aprendiendo?

Dicho eso, también hay que recalcar ciertas ventajas que nos ofrece Java [24]. Por ejemplo, su independencia de plataforma. Gracias a la JVM los programas son fáciles de trasladar de un ordenador a otro y de ejecutar. Otras ventajas son la anteriormente mencionada orientación a objetos. También su veteranía frente a lenguajes más modernos. Su muy extensa documentación. No hay que olvidar que java sigue siendo una opción muy elegida.

No se puede ignorar el hecho de que para diseño de interfaces hay opciones más populares como HTML junto con JavaScript o Dart con Flutter. Aun así, Java sigue siendo una opción sólida en el terreno de aplicaciones de escritorio. A continuación, se repasarán los distintos *frameworks* gráficos de Java para ver cual se decide utilizar.

Con respecto a qué versión de java se va a utilizar esta será la versión 17. Esta decisión se debe a que es de la versión más reciente y además es la última versión LTS, es decir long term support o con soporte a largo plazo. Se considera buena práctica elegir de las versiones más recientes ya que así tardará más tiempo en quedarse obsoleta y por tanto en tener que ser sustituida.

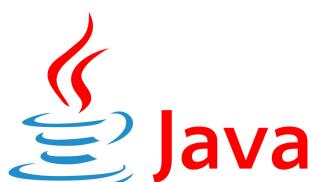


Figura 9: Logo de Java

4.2 Framework de la interfaz visual

Una vez decidido que el lenguaje de programación que se va a utilizar es Java lo que nos quedaría es decidir con qué *framework* se va a programar la interfaz del programa [25]. Estos *frameworks* son librerías de clases con funcionalidades de mostrar componentes gráficos que nos pueden servir. En este aspecto Java cuenta con varias alternativas que se van a repasar a continuación con la finalidad de encontrar la más conveniente para este proyecto y poder empezar a crear una solución para el mismo.

4.2.1 AWT

AWT significa Abstract Window Toolkit [26]. Es parte de la JFC, la Java Foundation Classes , que es la API estándar de Java para las interfaces gráficas. Surgió junto con Java en 1995 y es la predecesora de Swing. Esta librería proporciona una pequeña capa de abstracción entre el programa y el sistema operativo. Es por eso que al crear un componente de la interfaz con AWT este hace una llamada directa a la rutina del sistema para la creación de estos componentes, es decir, en cada sistema operativo el programa tendría un aspecto distinto. Esto puede resultar una ventaja si se quiere que el programa parezca nativo del sistema, sin embargo, también es una desventaja puesto que el programa no tendrá el mismo aspecto en dos sistemas operativos distintos. También es una limitación ya que solo se pueden crear aplicaciones con los componentes que se encuentren en todos los sistemas para los que se planea desarrollar la aplicación.

Con la salida de la J2SE 1.2 [27] AWT quedó obsoleto al ser sobrepasado ampliamente por las capacidades de Swing.

4.2.2 Swing

Como ya se ha comentado anteriormente hablando sobre AWT, Swing surgió para resolver las limitaciones de AWT y ofrecer un conjunto de componentes más sofisticado. También hemos comentado como AWT es la base de Swing [28] y es que Swing fue construido sobre las clases de AWT. Sin embargo, Swing no llama a las rutinas del sistema operativo, sino que usa Java 2D para llamar al subsistema local gráfico, logrando así la independencia del sistema.

Un programa de Swing tiene una estructura de componentes que heredan todos de la clase JComponent [29] y una serie de contenedores que heredan de la clase Container. Estas dos clases a su vez heredan de la clase Component que viene de AWT.

Ofrece grandes ventajas como un mayor número de componentes y más cantidad de parámetros personalizables de los mismos. Incluso se puede crear un aspecto totalmente personalizado de los componentes. Además, sigue el patrón Modelo Vista Controlador.

Nos permite cargar diferentes “pluggables look and feel”[30] que como sugeriría su nombre alteran sobre todo el aspecto de la interfaz y de los componentes (look) y también algo del comportamiento de sus componentes (feel). Entre ellos podríamos destacar JGoodies o FlatLaf por ejemplo.

De las principales ventajas de Swing es que al ser nativo en Java no hace falta incluir ninguna librería ya que todas sus clases se incluyen en Java. Además, muchos de los IDEs más populares de Java cuentan con algún tipo de editor gráfico o herramienta para desarrollar más fácilmente Swing.

Con respecto al *drag and drop* swing ofrece posibilidades de arrastrar texto entre diferentes campos y también imágenes.

4.2.3 SWT

SWT o Standard Widget Toolkit [31] es un conjunto de herramientas para crear interfaces gráficas en Java. Inicialmente fue desarrollado por IBM y actualmente es mantenido por Eclipse. Vuelve bastante a la idea de componentes nativos del sistema, como AWT. Una desventaja adicional de SWT aparte de que no es nativo de Java es que tiene menos componentes que Swing. También ocurre que la implementación de SWT es distinta para cada sistema operativo por lo que se ha de ajustar este aspecto para las distintas versiones.

4.2.4 JavaFX

Familia de productos software desarrollados por Oracle [32]. Inicialmente se desarrolló en el lenguaje JavaFX Script (anteriormente llamado F3) pero después pasó a estar escrito totalmente en Java para poder ejecutarlo en cualquier pc con java. Este proyecto se creó con la intención de ser el reemplazo de Swing y de competir con Flash. Esa fue la intención hasta el JDK 11 [33], en el cual JavaFX dejó de estar incluido y pasó a tener que ser añadido manualmente. Esta decisión se debió principalmente a dos razones. La primera era que el sector de mercado de JavaFX se vio afectado por la tendencia de desarrollar primero para teléfono móvil, el conocido como “mobile first” [34] y la segunda razón es por el Java Platform Module System [35], que promueve los módulos como manera de agrupar código añadidos de Java y propició la separación de JavaFX como un proyecto independiente. Actualmente Java FX recibe soporte de la compañía Gluon.

Los componentes de esta familia de productos son el principal SDK de JavaFX que cuenta con una serie de herramientas y componentes que ayudan en la creación de interfaces gráficas. Este JDK también llegó a incluir un compilador de JavaFX, cosa inútil hoy en día ya que JavaFX está íntegramente escrito en Java. También los distintos plugins o adaptaciones para IDEs que añaden la programación por *drag and drop* de componentes de JavaFX. También el Scene Builder que permite crear componentes de JavaFX también con *drag and drop* [36] y personalizarlos para después guardarlos en archivos FXML [37], escritos en xml que permiten hacer interfaces de manera similar a como se hace en HTML. Por último, también cuenta con muchas herramientas y plugins añadidos que aumentan sus funcionalidades, como por ejemplo los plugins para Adobe Photoshop y Adobe Illustrator.

Un programa de JavaFX cuenta con un escenario que equivale a la ventana mostrada. Este contiene una escena que ya empieza a contener nodos. Existe el nodo raíz que contiene a los demás y a partir de ahí están los nodos Hoja o rama. Los nodos hoja son botones o campos que no contienen otros nodos, los nodos rama son contenedores de otros nodos.

Con respecto al *drag and drop* de JavaFX este soporta también mover nodos. Esto se hace con eventos que permiten mover el elemento o copiarlo de un lado hacia otro. También existe la posibilidad de simular el *drag and drop* mediante eventos de ratón y movimiento de nodos basado en la posición de este mismo. Es una opción también muy popular.



Figura 10: Logo de JavaFX

4.2.5 Apache Pivot

Es una plataforma para el desarrollo de RIAs [38] o aplicaciones de internet enriquecidas, que son las aplicaciones web que tienen muchas de las características de una aplicación de escritorio [39]. Tiene la licencia apache y es de código abierto. Está enteramente escrita en Java por lo que es compatible con la máquina virtual de Java. Principalmente orientado a los desarrolladores que utilizan HTML, CSS y JavaScript.

Esta plataforma incluye más componentes que swing y Java FX conocidos como WKT. Permite creación de componentes personalizados e inclusive el uso de decoradores para cambiar propiedades de componentes o animarlos. También tiene el formato BXML similar a el FXML de Java FX que nos permite lo mismo en aspectos de jerarquía de componentes o de creación de interfaz.

Soporta el *drag and drop* pero parece que solo de imágenes y texto. Además, otra desventaja que presenta es que su última versión data del 2017 cosa que no da buenas señales sobre la continuidad del proyecto.



Figura 11: Logo de Apache Pivot

4.2.6 QT Jambi

Es una adaptación de las distintas APIs de QT para Java [40]. Funciona revisando las cabeceras de los archivos de C++ de la implementación de QT para luego hacer la correcta traducción de código para unirlo a Java. Inclusive se puede usar QT Jambi para crear más adaptaciones de otras APIs de QT. Empezó siendo desarrollado por la compañía de QT para pasar a ser mantenido por la comunidad.

La adaptación funciona de manera muy similar a como funcionan las APIs de QT por lo que los programadores que las conozcan no tendrán problemas utilizándola. Esta cuenta con ventajas como su modelo de programación o las herramientas multiplataforma. Sin embargo, hay mucha superposición con respecto a los frameworks gráficos de Java.



Figura 12: Logo de QT

4.2.7 GWT

Este es un *framework* creado por Google que pretende ocultar cierta complejidad del desarrollo web [41]. Permite desarrollar RIA al igual que Apache Pivot pero esta vez traducirá el código Java a JavaScript y HTML. Cuenta con un modo desarrollo que funciona con bytecode de Java y otro de web que ya funciona con el HTML y JavaScript compilado a partir del código Java. Cuenta con un compilador de Java a JavaScript, con un alojamiento web de la aplicación para desarrollo, con una cobertura para las principales librerías de Java y con una librería de desarrollo de interfaz de usuario.

Java Script sí que permite la programación de drag and drop así que técnicamente GWT también. El resultado del proceso de utilización de este *framework* es una web RIA o con capacidades enriquecidas en vez de una aplicación de escritorio.



Figura 13: Logo de GWT

4.2.8 Conclusión

Después de la recopilación de información sobre las distintas opciones que hay para poder trabajar con Java llega el momento de tomar una decisión sobre ello. Esta decisión no será sencilla debido a la gran cantidad de opciones que existen y al gran parecido que hay entre muchas ellas.

Para empezar, se descartarán los *frameworks* cuyo proyecto ya no esté siendo continuado. Estos serían Apache Pivot y AWT. Se descartan debido a que no es nada aconsejable elaborar software con tecnología que no avanza ya que este software puede llegar a quedar obsoleto y una migración a otra tecnología puede ser demasiado laboriosa. Además, en el caso de AWT este mismo ha sido reemplazado por una mejor opción como es Swing.

Otro *framework* a descartar es SWT. Esto se debe a la complicación que plantea a la hora de adaptarlo a varios sistemas operativos.

Un punto muy importante a la hora de descartar es el *drag and drop*. Al observar Swing este es de las mejores opciones a la hora de desarrollar interfaz en java por tratarse de la opción nativa. Sin embargo, su soporte con respecto al *drag and drop* no se parece a lo que se busca en el entorno a desarrollar. Con esta decisión se descartarían Swing, QT Jambi y Apache Pivot.

Esto dejaría tan solo con dos opciones, Java FX y GWT. Basado en los casos que se ha indicado en el estado del arte la opción web es muy común en estos casos. Sin embargo, previamente se ha decidido en el análisis del problema que la solución sería una aplicación de escritorio. Además, en cierto modo que el código Java se acabase traduciendo a HTML y JavaScript sería fallar un poco a la decisión de hacer el proyecto en Java.

Es por eso y por varios aspectos más que la elección final es JavaFX. Entre otros aspectos que benefician esta decisión están la existencia de los ficheros FXML que permiten crear componentes personalizados cargables y también la gran cantidad de ejemplos drag and drop que se puede encontrar en internet. Otra ventaja es la estructura con la que almacena los nodos que puede beneficiarnos a la hora de traducir el código visual por tener forma de árbol. También como se ha comentado antes, en este proyecto se prefiere mantener todo en lenguaje Java para inspirar a los futuros aprendices del lenguaje sus capacidades.

De esta manera termina este apartado habiendo clarificado cuáles serán las tecnologías utilizadas para tratar de poner solución a este problema.

5 Diseño

Una vez decididas tanto la solución concreta del problema como las tecnologías utilizadas es el momento de empezar el diseño. Tratar el diseño es tomar decisiones de cómo llevar a la realidad la solución decidida. Hay varios apartados que se deben repasar en búsqueda de soluciones. A continuación, se van a repasar y tomar todas las decisiones sobre el diseño.

5.1 Arquitectura del sistema

El programa planteado es muy sencillo. No necesita de una base de datos ya que todo lo que hacer lo guarda en local si el usuario lo desea, es prácticamente como un bloc de notas enriquecido. Se debe buscar entre arquitecturas más sencillas para poder clasificarlo y sobre todo fijarse en facetas distintas del programa.

La comunicación entre la interfaz y el programa será activada mediante eventos del ratón, ya sea el clicar un botón o interactuar con un desplegable. No obstante, no toda la comunicación interna del programa funcionará con eventos. Es por eso que no se le puede clasificar enteramente como una arquitectura basada en eventos [42].

Por otra parte, se pretende crear varios archivos con funciones claramente separadas y distinguidas. Es por eso que también en parte esta arquitectura será basada en componentes o módulos software que ejecutarán su función específica dentro del programa [43]. Otra vez es en parte porque los componentes no están tan fuertemente desacoplados entre ellos. Además, los componentes son relativamente sencillos en comparación con productos software más complejos.

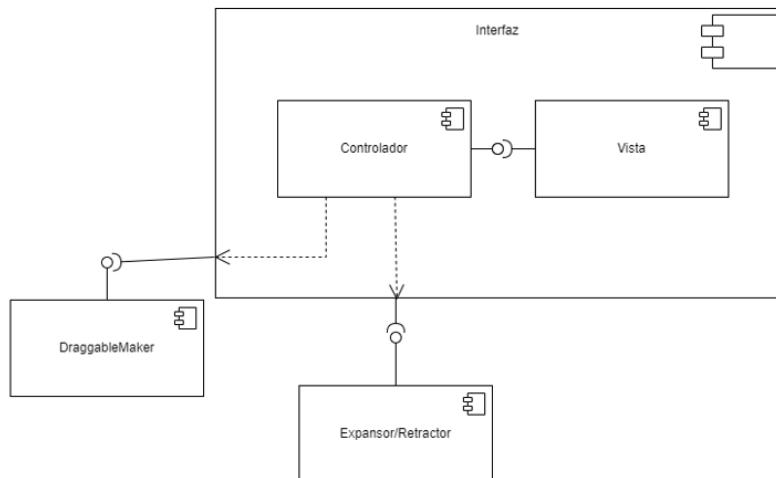


Figura 14: Diagrama de Componentes del programa

5.2 Diseño detallado

Una vez hablada de la arquitectura que sigue el programa es momento de entrar más en detalle sobre cada una de sus partes. En este punto se va a detallar cómo se ejecutarán cada una de las partes y todas las decisiones de diseño tomadas.

5.2.1 Interfaz

Teniendo en cuenta las bases de este proyecto, sus requisitos y objetivos se puede sacar en claro que hacía falta una interfaz gráfica. No solo eso, sino que queda claro que es una de las partes más importantes y ha de pensarse muy bien. Esta interfaz que comentamos constará de dos vistas: la vista de programación y la vista de edición de código. Estas vistas se encargan de cubrir las dos funciones más importantes que ofrece el entorno: crear código Java de manera visual y traducirlo para guardarlo en un fichero. Otros elementos que se cubrirán de la interfaz son los bloques de programación.

5.2.1.1 Bloques de programación

Como se ha estado comentando durante todo el documento la unidad mínima de programación será el bloque. Mediante su *drag and drop* es que se construirán los programas. También se ha comentado que los bloques representan las distintas funciones Java. Los bloques de programación serán representados por archivos FXML, al igual que las vistas. Serán como micro interfaces cargables en el programa. Por lo tanto, al ser así constarán de sus componentes o nodos respectivos necesarios y su controlador respectivo. El controlador servirá para aplicar funcionalidad a cada bloque o vista, para interpretar los eventos.

Los bloques han de ser identificables por su función así que una de las primeras medidas a tomar fue la de dividirlos en grupos. Los grupos responden a una división del tipo de sentencias Java.

Los grupos creados fueron:

- Variables: En este grupo encontraremos todo lo relacionado con la inicialización, asignación y llamada de variables. Desde los tipos de datos a posibles valores.
- Operaciones: En este grupo entran las operaciones aritmético lógicas además de los valores true y false.
- Bucles: En este apartado se añadieron los bucles tanto de condición como de iteración. En este caso solo se creyó necesario añadir el bucle While y el bucle For. No se añadió Do While por ser muy similar a while y replicable mediante este mismo.
- Condicionales: En este grupo se añadieron las estructuras condicionales. Están las sentencias If, If Else y Else. No se añadió Switch por ser replicable mediante varios ifs.
- Funciones: Un grupo dedicado a las funciones o métodos. En él se encuentra su declaración, sus sentencias específicas como el tipo de retorno void y la sentencia de retorno y sus llamadas tanto estáticas como dinámicas.
- Entrada/Salida: Por último, está el grupo de entrada y salida. Este se comporta más como un cajón de sastre ya que incluye todo el apartado de importación de otras clases. También incluye snippets para añadir la lectura y escritura. Finalmente, incluye sentencias típicas de la orientación a objetos.

Cada grupo tendrá un color asignado para reconocer y asimilar su función con más facilidad y facilitar su reconocimiento.

Para facilitar que el programa tenga código reutilizable lo que hay que hacer es que los bloques tengan un comportamiento muy similar entre ellos y una estructura similar también. Una de las partes que ha de tener cada bloque es un cuerpo, que compartirá con todos los bloques, otra es la zona arrastrable y otra, en algunos casos, la zona donde se puede soltar.

Con el fin de reducir los errores se pretende clasificar los bloques en dos grandes grupos. Los bloques que irán dentro de las asignaciones y condiciones y demás, y los bloques que irán dentro del contenido de bucles, estructuras condicionales, clases, funciones, etc. Esta distinción permite evitar que se introduzcan bloques que carecen de sentido dentro de otros bloques que no deberían contenerlos.

5.2.1.2 Vista principal

La vista principal es la primera que se ve cuando el programa se inicia. Es la misma vista con la que el usuario interactúa para programar. En ella se deben encontrar el sitio donde se programa, un elemento para seleccionar los bloques, un botón para traducir de bloques a códigos y por último algo que permita borrar bloques. Además, esta interfaz ha de ser *scrollable* en ambos sentidos ya que el programa puede crecer bastante y se debe poder navegar por él con total libertad. Es con estos requisitos que se ha desarrollado la interfaz además de la idea de facilitar todo el proceso de programar.

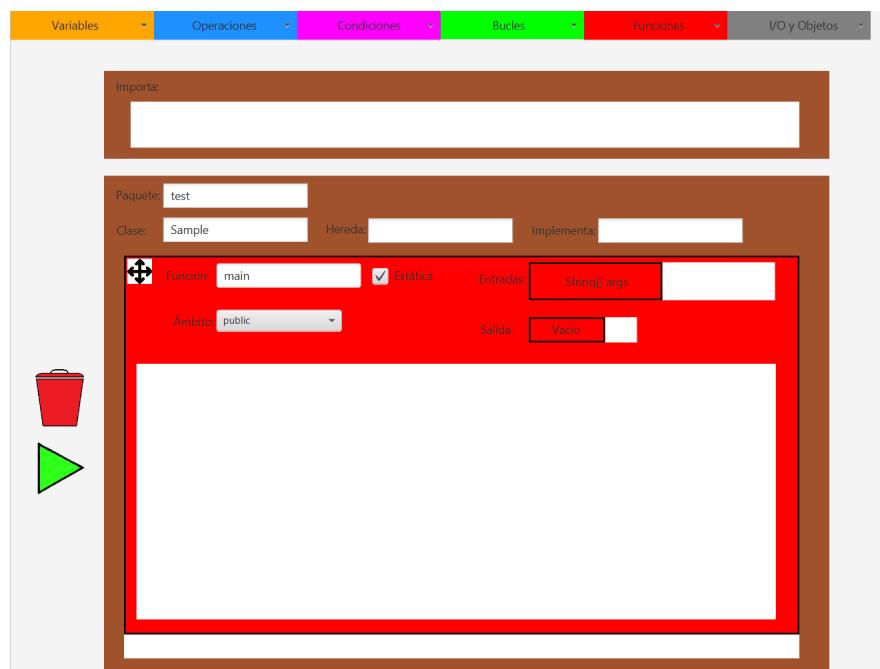


Figura 15: Vista principal

5.2.1.2.1 Área de programación

Se va a empezar por hablar sobre el área de programación. Esta zona, al tratarse de Java debería ser una clase. Es obligatorio en un archivo .java indicar un paquete y una clase. Entre estas declaraciones irían los imports del programa. Por la manera en

la que el programa acabará estando hecho se decidió separar la zona de importación de la zona de la programación.

De esta maneraaremos una zona de importación donde se añadirán bloques de importación y una zona donde se programará en la que se indicará la clase y el paquete. Además, se podrá indicar la clase de la que se hereda, si se hereda, y la interfaz que se implementa, si se implementa.

Por defecto y para facilitar el aprendizaje se añadirá por defecto los bloques de la clase main con toda su configuración directamente. Esto servirá para que los estudiantes aprendan ya que los bloques utilizados para crear la clase main serán los mismos que el usuario puede utilizar. No será así con la clase porque un archivo .java no tiene más de una clase, por lo menos no publica ya que sí que es posible albergar más de una clase en un archivo Java, sin embargo, esto ya es un nivel más avanzado de programación Java.

Por tanto, la función de esta zona será la de interacción con los bloques. Estos serán introducidos, movidos, cambiados en función de cómo queramos el programa. Se podría comparar con el cuadro de texto de un IDE que muestra y nos permite interactuar con el contenido del archivo.



Figura 16: Área de programación de la vista principal

5.2.1.2.2 Área de creación de bloques

Para poder interactuar con bloques se deben poder crear, como bien se especifica en los requisitos. Para cumplir esa función hace falta una área donde podamos seleccionar los bloques que necesitamos y donde se generen. Teniendo en cuenta la división anteriormente mencionada sobre los tipos de bloques la aproximación seguida

será la de crear varios selectores de bloques. Habrá uno por cada grupo funcional que nos mostrará todos los tipos de bloques disponibles de ese grupo. Siguiendo con la dinámica de los bloques estos selectores serán del color correspondiente y tendrán una etiqueta representativa del tipo de bloque.

Respecto a qué tipo de selectores serán, una buena manera de mostrar opciones y hacerlas seleccionables es un desplegable con ellas. Este responderá al evento de selección de una opción en concreto y creará el bloque en cuestión. Con respecto a donde aparecerán estos bloques lo mejor es en una zona donde sean visibles y no molesten demasiado.

De esta manera se crearán los bloques, mediante desplegables con opciones clasificados por grupos. Estos bloques se crearán en una zona cualquiera que cumpla los requisitos de no estorbar y serán desde ese momento arrastrables.



Figura 17: Selectores del tipo de bloque de la vista principal

5.2.1.2.3 Ícono de borrado de bloques

De la creación de bloques surge la necesidad de eliminarlos. Muchos bloques en pantalla pueden saturarla de contenido que, en algunos casos, por error o mala previsión, puede ser innecesario. Es por eso que siguiendo el espíritu del *drag and drop* se ha considerado que lo mejor sería arrastrarlos a un ícono en forma de cubo de basura. Este es un lenguaje visual familiar para los usuarios y bastante intuitivo.

El resultado sería un ícono de cubo de basura no clicable. Este se encontraría en un área reservada para iconos y botones. Al momento de arrastrar la parte arrastrable de un bloque y soltarlo encima de este ícono el bloque sería borrado. Lo mismo con grupos de bloques anidados.



Figura 18: Ícono de papelera de la vista principal

5.2.1.2.4 Botón de traducción de código

Por último, tendremos en la misma área de botones e iconos uno más con la función de traducir el código. Este ha de ser pulsado cuando el programa se haya finalizado. Su función será la de recorrer los bloques del programa traduciendo de bloques a Java. Este código será almacenado en un archivo de texto temporal que servirá para tener el código hasta que se decida guardarlo o se descarte.

Otra función de este botón es abrir la segunda vista del programa. Esta vista no cierra la primera, sino que coexisten. Esto permite volver a la anterior si algo no nos convence y cambiarlo para retraducir el código.

El icono de este botón ha de ser representativo. Sin embargo, nadie entiende que un botón traduce código de por sí. Es por eso que se ha optado por un botón más bien de acción. Será un play verde similar a los de ejecutar el programa de un IDE. Sin embargo, su significado será el de ejecutar el propósito de este programa.

Al contar con pocos iconos, de gran tamaño y bastante descriptivos de su acción la interfaz permanece simple e intuitiva. Además, este botón al abrir una ventana cuenta con cierto feedback. También se planea que si se detecta algún error al traducir código muestre feedback indicativo de error en vez de pasar al siguiente paso.

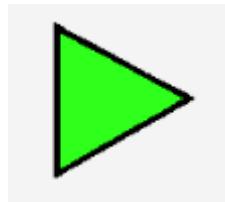


Figura 19: Icono de acción de la vista principal

5.2.1.3 Segunda vista

Esta segunda vista aparece cuando se presiona el botón de acción, que traduce el código. Su finalidad es la de editar el código y visualizarlo antes de guardarlo o descartarlo y seguir programando visualmente. Podremos encontrar solo dos elementos: el área de texto con el código traducido y un botón de guardado. También ha de ser *scrollable* por las mismas razones que la vista principal, por el tamaño del programa. A continuación, se detallarán estas partes.

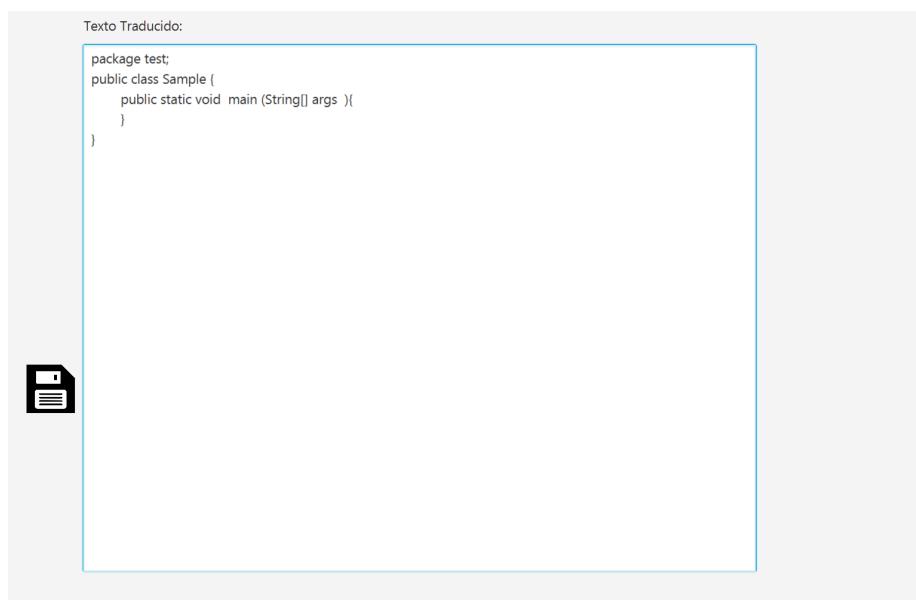
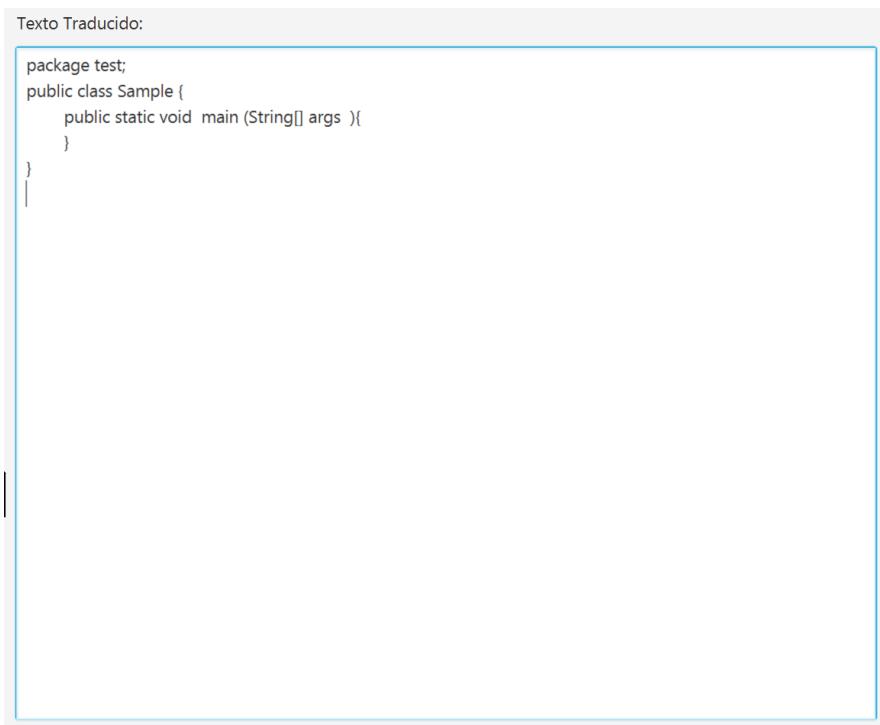


Figura 20: Segunda vista

5.2.1.3.1 Cuadro de edición de código

Esta parte se tratará de un área de texto que ocupará casi toda la vista. En ella se mostrará el contenido del archivo donde se guarda la traducción del código. Este contenido será editable por el usuario, aunque no es del todo su propósito. La idea es la visualización de la traducción del código visual programado. Así poco a poco se pretende que el usuario aprenda relacionando partes. La edición de su contenido estaría más pensada para una fase más avanzada del aprendizaje.

El contenido de este recuadro será el que se guardará posteriormente en un archivo .java. Esta vez sin realizar ninguna comprobación por lo que las ediciones realizadas por el usuario deben ser bajo su propia responsabilidad y conocimiento.



The screenshot shows a light gray window titled "Texto Traducido:" containing a single line of Java code. The code defines a class named "Sample" with a main method:

```
package test;
public class Sample {
    public static void main (String[] args ){
    }
}
```

Figura 21: Cuadro de texto de la segunda vista

5.2.1.3.1 Botón de guardado

Botón que se encuentra en la misma zona donde estaba el botón de acción en la anterior vista. Este tendrá un icono de disquete al igual que el típico botón de guardado en casi todos los programas. Su función será la de guardar el código finalmente en un archivo .java.

Cuando este botón es pulsado se abrirá un selector de carpetas para poder elegir donde se guardará el archivo. Para guardarlo el nombre del archivo será automáticamente el nombre de la clase principal del programa. El contenido guardado será el que el usuario haya dejado en el área de texto editable. Por tanto, no se comprobarán las ediciones que pueda realizar el usuario.



Figura 22: Icono de guardado de la segunda vista

5.2.2 Controladores

Un controlador es la parte de código que aplica lógica a la interfaz. Cada una de las interfaces de este programa cuentan con uno. Eso son todos los tipos de bloques y ambas vistas. En esta parte vamos a explicar el comportamiento de los controladores.

5.2.2.1 Controlador de bloque

Estos son los controladores más simples. Tan solo tienen una función. Su función es declarar la parte arrastrable de los bloques como un atributo y pasársela a el componente que hace que los bloques tengan drag and drop. Esto ocurre cuando el bloque es inicializado. Gracias a esto los bloques se convierten en arrastrables.

5.2.2.2 Controlador de la vista principal

Este es el controlador más importante. Su función es la de controlar los eventos de los selectores de bloques y el botón de acción. Al controlar los eventos de los desplegables ha de comprobar cuál ha sido la opción seleccionada y crear el bloque correcto además de colocarlo donde toca. Hay casos incluso en los que genera más de un bloque. También se encarga de inicializar las opciones posibles de los selectores de los distintos tipos de bloques.

Con respecto a controlar la acción del botón esta es la parte más compleja. Al clicar el botón ocurren dos cosas. La primera es toda la traducción completa del código mediante el recorrido de todo este y la introducción del código completo en el archivo temporal. La segunda es la invocación de la segunda vista con el código metido en ella.

5.2.2.3 Controlador de la segunda vista

Este controlador es más simple. Su función es la de controlar el evento de guardado. Si este botón es pulsado una ventana del sistema operativo se abrirá con el fin de seleccionar una carpeta. Una vez seleccionada un archivo .java con el contenido del cuadro de texto de esta vista y el nombre de la clase como nombre de archivo será guardado en el directorio elegido.

Una vez hecho esto no se cerrará ninguna ventana más que la de selección de carpeta. Se permite así guardar las veces que se desee en distintas localizaciones.

5.2.3 DraggableMaker

El componente DraggableMaker o componente que permite que los bloques sean arrastrados es una clase separada de Java. Esta tan solo cuenta con una función que permite que un bloque sea tanto arrastrable como soltable al actuar sobre cierta zona de él. Dentro de esta función que ejecutarán los bloques en su Controlador para poder ser arrastrables se detectan 3 eventos:

- Cuando se clica un bloque: El bloque se coje de donde estuviese. Si estaba fuera simplemente se coje. Si estaba dentro de algún otro bloque este es sacado fuera y cogido.
- Cuando el ratón se arrastra: La posición del bloque se recalcula todo el rato en base a la posición del ratón para así simular que el bloque está siendo arrastrado.
- Cuando el ratón suelta el bloque: En este caso lo que ocurre es que el bloque es soltado. Si se suelta encima de una parte contenedora de algún bloque entonces este se introduce dentro en la posición que se ha indicado. Sin embargo, si el bloque es soltado fuera simplemente se quedará fuera. Si el bloque es soltado encima del cubo de basura el bloque procederá a borrarse.

Gracias a la detección de estos eventos y la actuación en función a esto se puede simular un *drag and drop*.

5.2.4 Expander

El Expander es el último componente de este sistema. Su función es expandir componentes para hacer hueco al anidado de bloques dentro de otros bloques. También tiene la función contraria, haciendo pequeños los bloques cuando el anidado se reduce. Esto es logrado mediante la clase Expander que al igual que la clase DraggableMaker contiene funciones que permiten hacer esto.

Esta clase cuenta con 5 funciones. Estas son la expansión del contenido de la clase, la compresión del contenido de la clase, la expansión del contenido del apartado de los imports, la compresión del apartado de los imports y por último la expansión de la vista. Se ha dividido las expansiones y compresiones por las partes que expanden y contraen por la manera diferente en la que se expanden y contraen. También se considera que no hace falta contraer la vista principal por varias razones. La primera, que no es extraño visualmente debido al *scroll* y que no es un bloque estirado vacío. La segunda, que al tener *scroll* el usuario siempre se puede mantener en el área importante. Y la tercera, que el programa potencialmente volverá a crecer por lo que volvería a expandirse.

Esas son las funcionalidades del Expansor. La existencia de este elemento es estética y para facilitar que el programa sea intuitivo. Si los bloques no se expandiesen el resultado sería visualmente extraño mientras que si no decreciesen luego esto podría llevar a confusiones sobre qué hay realmente en el código.

5.2.5 Main

Esta parte como tal no es un componente, sino que se encarga de lanzar la primera vista y colocar todos los elementos en su lugar. Se trata de la clase principal con el método main. Clásico en cualquier programa Java al ser el método que se ejecuta automáticamente de ser encontrado en algún archivo Java.

Su función como se ha comentado es la de cargar la interfaz y darle dimensiones. Posteriormente carga un bloque de tipo función, lo configura para ser que por defecto sea un método main y lo introduce en la clase.

Por último, muestra la ventana del programa. De esta manera se tiene inicialmente un programa correcto con los datos más problemáticos de un aprendiz de Java llenados y el programa dispuesto para empezar a ser programado.

6 Desarrollo de la solución propuesta

Este apartado servirá para concretar más de qué manera se han desarrollado todas las partes específicas. Se explicarán todas las partes más interesantes del código [44] realizado y los motivos de este.

6.1 Estructura de archivos

El programa será un proyecto Java. Este vendrá estructurado por tanto con las librerías añadidas fuera y una carpeta con el contenido. En este caso particular dentro estará un paquete llamado application que contendrá ya los archivos .java, es decir, los controladores, el Expander y el DraggableMaker. Luego habrá dos paquetes internos más, uno con recursos como las imágenes y otro con los archivos .fxml.

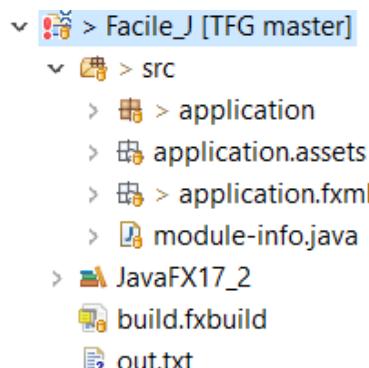


Figura 23: Estructura de archivos del proyecto

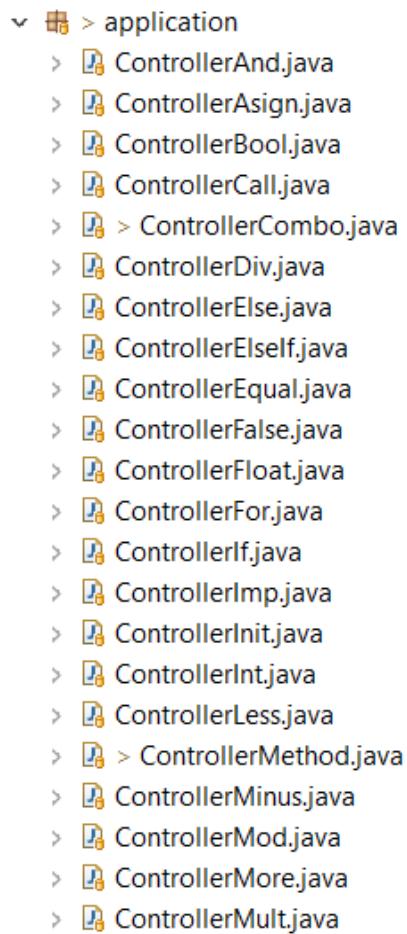


Figura 24: Archivos .java del proyecto (no todos)

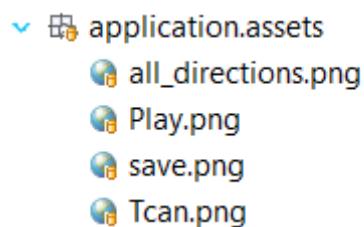


Figura 25: Imágenes del proyecto



Figura 26: Archivos FXML del proyecto (no todos)

6.2 Interfaz

A continuación, se detalla en apartados como se han construido los distintos elementos de la interfaz. Se hará hincapié más en aspectos del aspecto y detalles del elemento a nivel de interfaz y no del comportamiento ya que esto será dejado para otro apartado.

6.2.1 Bloques

Como se menciona en apartados anteriores se debe hacer una interfaz que siga unas pautas muy claras. Por eso todos los bloques deben tener ciertas cosas en común para que ciertas funciones del programa actúen sobre todos.

Para empezar todas las interfaces de los bloques se basarán en un AnchorPane, de esta manera se puede colocar los elementos en ellas como se deseé y además permitimos interactuar con todas igual. Para introducir bloques dentro contaremos con VBox si queremos que se apilen verticalmente como dentro de una clase o HBox si queremos que ocurra horizontalmente como en la condición de un if. Para los bloques de este tipo, que además pueden crecer se les pone un tamaño mínimo para luego tomarlo como referencia en el caso de decrecer.

Un bloque ha de tener obligatoriamente una zona de arrastrado. Esta se puede tratar en caso de los bloques con una zona de soltado (HBox o VBox) de una imagen

con unas flechas. En caso de bloques más pequeños se puede tratar de otro AnchorPane con las mismas dimensiones que el mismo bloque. Este sería el caso de un bloque +. El uso de imágenes indicativas de la zona de arrastrado nos permite que no todo el bloque sea donde tenemos que clicar para arrastrar. Eso evita superposiciones de zonas de arrastrado que harían funcionar mal el programa.

Todos los elementos importantes de un bloque deben contar con su id para poder ser localizados posteriormente. Por lo general serán distintivas de la función de ese elemento. En el caso de ser la zona de arrastrado serán Drag y algo distintivo del bloque por ejemplo.

Otro detalle importante son las clases que se les asociará a cada elemento. Las zonas de arrastrado tendrán la clase drag, las zonas de soltado tendrán la clase drop y la base de los programas la clase body. Además las zonas de arrastrado tendrán las clases V, H,O o imp. Los bloques con la clase V sólo podrán estar en VBoxes, los de H solo podrán estar en HBoxes, los de la clase O solo podrán estar solos a la vez en un HBox y los imp solo en el VBox de los imports. En este último caso sólo se podrán introducir dentro de los contenedores con la clase one.

Por último y como ya se mencionó anteriormente todos los bloques contarán con cosas básicas como un color identificativo de su función. También un Controlador que posteriormente será analizado en detalle.

6.2.2 Vistas

Las vistas en este aspecto no tienen mucha complejidad. Los principios seguidos son similares a los de los bloques. En ellas empezamos usando como base un ScrollPane que automáticamente añade scroll si nota que la ventana crece más que el tamaño de la escena. Luego encima va un AnchorPane y en él todo el contenido. Todos los elementos relevantes de la escena cuentan con una ID para poder encontrarlos luego. Además se editarán el estilo de los elementos sobre todo los elementos selectores o ChoiceBox con propiedades CSS para quitar sus bordes redondeados. También se colocará tamaño mínimo a los elementos que han de crecer y posteriormente decrecer.

Con respecto a la sección de los imports y la clase estos a pesar de tener la misma estructura que un bloque no tienen ni las mismas clases ni sigue el mismo protocolo que los bloques arrastrables. Esto se debe a que en ningún momento serán arrastrables.

6.3 Controladores

Los controladores contendrán la lógica de manejo de los eventos. También contendrán la lógica de traducción y guardado. Es por eso que vale la pena detenerse más a hablar de estas partes

6.3.1 Controladores de Bloques

Como se ha mencionado en el capítulo anterior un controlador de bloque, en este proyecto suele tener solo una función. Esta es hacer que el bloque sea arrastrable. Sin embargo, se va a explicar cómo se ha hecho concretamente.

Lo primero es mencionar que al trasladar elementos con id específica de los archivos FXML a la realidad hay que etiquetarlos como tal. Todos los componentes con id se etiquetan, siempre que se vayan a usar.

Lo que se utilizará será la función *initialize* resultante de que todos los controladores implementarán la interfaz *Initializable* de JavaFX. En el método *initialize* se llamará a la función *makeDraggable* del *DraggableMaker* importado y a esta función se le pasará la parte arrastrable del bloque. Para diferenciarla siempre le hemos puesto la palabra *Drag* en el id en cada bloque.

Esto hará que el bloque sea arrastrable. Para el caso del bloque función este cuenta con un desplegable cuyas opciones son colocadas también en el *initialize* de su controlador pasándole una lista de strings.

6.3.2 Controlador de la Vista Principal

Este controlador es de las clases más grandes. Esto sucede porque contiene la lógica de traducción del código visual a Java. Además, también contiene la lógica de selección de opciones de los ChoiceBox.

6.3.2.1 Selectores

Primeramente, se inicializan las variables que representan los selectores de bloques. En el método *initialize* descrito anteriormente se dará valores a estos desplegables. Luego cada uno tendrá un *listener* en el evento acción. Al detectar este evento que se da cuando se interactúa con los desplegables procederemos a obtener en forma de String la opción seleccionada. Posteriormente en base a ella cargaremos el bloque correcto. Se le indicará una posición y se añadirá al AnchorPane principal. Finalmente se borra la opción seleccionada para que no permanezca ahí dificultando seleccionar una nueva opción.

6.3.2.2 Recorrido del Árbol

Como se mencionó en el capítulo de Estado del Arte javafx guarda sus elementos como un árbol con jerarquías. Hay elementos padres y elementos hijos. Esto será muy conveniente en este caso ya que de la manera en la que se ha planteado el problema la solución a la hora de traducir será sencilla.

A la hora de recorrer un árbol [45] de n hijos hay varios recorridos a seguir, cada uno con sus propiedades y ventajas. Sin embargo, para este caso lo mejor es el preorden. El preorden es el recorrido de un árbol en el que primero se visita un nodo, después su nodo izquierdo y luego su nodo derecho (en caso de ser binario). En este lenguaje de programación visual que se ha creado los bloques están contenidos dentro de otros que empezarán antes, por eso mismo primero conviene visitar a un

nodo y luego sus hijos, que siempre estarán después de su padre. Esa es la razón de usar ese recorrido.

Para implementarlo se ha optado por la opción más intuitiva, la recursividad. De esta manera y después de tratar con los datos de la clase, que no es un bloque, se empezará con el método que recorre el árbol. Su comportamiento es muy sencillo. Si el elemento que le pasamos es un HBox o VBox hará llamadas recursivas con sus hijos, sinó llamará a métodos de posibles nodos hoja. Hay que aclarar que son tan solo posibles ya que entre ellos puede encontrarse un método que vuelva a llamar a la función que viaja por el árbol.

Eventualmente el nodo raíz se queda sin hijos que revisar y la llamada a la función finaliza. Es entonces cuando el programa acaba de ser traducido.

6.3.2.3 Funciones de escritura

En el apartado anterior se mencionó unas funciones a las que se llamaba en caso de ser un nodo “hoja”. Estas tienen la funcionalidad de escribir en el archivo temporal donde se almacena la traducción.

Cuando un nodo normal, es decir la base de un bloque, llega a el método de recorrido del árbol este no es clasificado ni como HBox ni como VBox, por tanto es un nodo “hoja”. Entonces se llama a la función pertinente, ya que cada bloque cuenta con la suya, para que escriba ese bloque en el archivo. Esta función es totalmente distinta para cada bloque. Para los bloques más simples sólo añade una pequeña cantidad de texto, como puede ser un +. Sin embargo para los bloques más complejos como el While esta llama de nuevo a la función de recorrido de árbol para cada nodo del contenido y de la condición. Por eso no se puede considerar que los nodos que no son VBox o HBox son nodos hoja realmente.

Una parte relevante de este proceso es que el nivel de indentación que maneja en esa parte el programa se va pasando entre nodos aumentando en cuanto el nivel de anidación aumenta o descendiendo cuando se acaba una parte anidada en otra. Este pase del valor nos sirve entonces para mantener la indentación correcta en cada momento.

6.3.3 Controlador de la Vista Secundaria

El controlador más sencillo después del de los bloques. Tan solo cuenta con el *listener* del botón de guardado además de la inicialización tanto del botón como del área de texto. Este *listener* se incluirá en el método *initialize* mencionado anteriormente.

El funcionamiento de este *listener* es el siguiente. Se obtendrá el *path* de la carpeta de destino del fichero .java gracias a un objeto de JavaFX llamado *DirectoryChooser* con el cual se puede abrir un diálogo en la escena para que se seleccione el directorio. Seguidamente manipulando el contenido del área de texto se obtendrá el nombre de la clase, el cual será el nombre del archivo final. Con este path se inicializará un objeto tipo *File* con el que gracias a la función *createFile* se puede comprobar si este ya existe o no además de rodear esta función con un try catch por si falla. Si todo sale

bien se añadirá el contenido del área de texto dentro del archivo y así finalizará el proceso.

6.4 DraggableMaker

Esta es una de las partes más complejas del código ya que tiene una extensión grande. Como ya se comentó su función es detectar los eventos relacionados con el *drag and drop* de bloques. Esto se hace haciendo que la parte arrastrable de cada bloque ejecute una función que tiene los tres eventos. El método añadirá tres *listeners* a la parte arrastrable de cada bloque. A continuación se repasarán los tres tipos de evento y exactamente que se hace en cada uno.

Cabe puntualizar antes que lo primero que hace la función es obtener el valor del bloque entero en base a la parte arrastrable de este mismo. Esta es la parte con la que vamos a interactuar principalmente.

6.4.1 Evento de clic

Se activa cuando el ratón hace clic encima de la parte arrastrable de un nodo. Cuando se activa primeramente manda el bloque entero al frente de la escena, para que nada se superponga a él. Luego calcula tanto la posición del ratón como la posible distancia que el usuario se desplaza con el *scroll*. Esto se hace debido a que el sistema usa dos maneras de calcular la posición de los elementos, una no tiene en cuenta la distancia desplazada con *scroll* sino que solo la parte visible de la pantalla y otra el total. Acto seguido obtendremos el nodo padre del bloque en el que estamos y convertimos su tipo a un VBox o a un HBox. Una vez hecho eso colocaremos el bloque en la posición del ratón y lo sacaremos de su nodo padre para meterlo en el AnchorPane principal de la vista, sacándolo fuera de donde estaba. Finalmente usaremos el Expander para disminuir el tamaño del padre en el que estaba.

6.4.2 Evento de arrastrado

Este evento es detectado cuando clicando un bloque este es movido sin soltar el ratón. En este *listener* se vuelven a calcular parámetros de posición de ratón y de *scroll* y en base a ellos se ajusta la posición del bloque.



Figura 27: Arrastrado de un bloque

6.4.3 Evento de soltado

Se activa cuando después de clicar en la zona arrastrable de un bloque y luego se suelta el clic. Este es el *listener* más complejo. Primero obtendrá los parámetros de posición del ratón además de una lista de los elementos en los que se puede introducir un bloque. Calculará la posición de nuestro bloque y la comparará primero con la del cubo de basura para borrar bloques. De coincidir el bloque sería borrado quitándolo de donde estuviese contenido y terminando la ejecución del *listener*.

Si no fuese el caso continuaría la ejecución normal del *listener* esta vez recorriendo todos los nodos en los que se puede introducir un bloque y comparando posiciones. Nos quedaremos entonces con el último coincidente por ser el más superior.

A partir de esta zona es donde se complica más el método ya que debe revisar la coincidencia de ciertas cosas. Hay 4 posibles casos. El primero es que sea un bloque que vaya dentro de un VBox y por tanto donde lo estemos soltando sea un VBox. El segundo que sea un bloque que vaya dentro de un HBox y por tanto donde lo estemos soltando sea dentro de un HBox. El tercer caso es el de un HBox que solo puede contener un bloque a la vez y lo indica si tiene la clase one, esto se reserva para tipos de datos, por ejemplo. Por último, hay un caso dedicado a los imports.

En todos los casos menos en el tercero se van a hacer las siguientes cosas. Se va a calcular el layout del bloque dentro de la clase. Esto se hará recorriendo todo el árbol sumando los layouts hasta el elemento con id #Class mediante un do while. Se comparará luego con todos los nodos hermanos para ver si el bloque se encuentra entre la parte más alta y la parte más baja de cada elemento. Si es así el bloque será introducido en el índice de este elemento, si no coincide con ninguno se introducirá al final del todo. Finalmente llamamos al expander para que haga crecer el bloque, en el caso de los imports el método del expandador será distinto.

En el tercer caso simplemente se comprobará que no hubiese ningún bloque previamente antes de introducir uno nuevo. Si si que lo hay no se introduce, sino si que se introduce. Así se logra que no haya más de un tipo a la vez por ejemplo.



Figura 28: Soltado de un bloque

6.4.4 Problemas de scroll

En JavaFX al lograr la posición de un elemento la posición obtenida es sobre el conjunto visible de pantalla. Sin embargo al ponerle la nueva posición se pone sobre el total de la vista. Esto nos trae un problema debido a que el programa puede tener *scroll* tanto horizontal como vertical. Es por eso que hay que arreglarlo de alguna manera. La solución propuesta es compensar el valor final sumándole la distancia que se ha desplazado con el *scroll*.

Calcularemos esta distancia en base a un valor que nos devuelve el método *.getHValue* o *.getVValue*, depende del eje. Este método nos devuelve el valor del desplazamiento entre 0 y 1. Posteriormente hemos de multiplicar este valor por la resta entre el tamaño total del panel *scrollable* menos el tamaño de la escena. Esto nos dará el valor que buscamos. Este valor ha de ser sumado al recalcular posiciones de bloques.

6.5 Expander

Como ya se mencionó anteriormente esta clase contiene 5 funciones. Dos de ellas se encargan de la expansión y compresión de los bloques de la clase. Otros dos hacen lo mismo con los imports y el último expande las dimensiones de la ventana. A continuación se explicará su funcionamiento.

6.5.1 Expandir la clase

Esta función se encarga de expandir todos los bloques desde el que es llamado hasta llegar de nodo padre en nodo padre hasta la clase, si esto fuese necesario para contener los nuevos bloques. Esta función se llamará siempre que un bloque sea introducido dentro de otro. Hay dos variantes de este método. Todo depende de si se le pasa un HBox o un VBox ya que no se expanden igual.

En el caso de un HBox, el cual apila los bloques horizontalmente tiene sentido que este crezca horizontalmente. Además crecerá hacia la derecha siempre ya que este es el sentido en el que apila bloques. Lo primero será calcular la altura máxima y el ancho máximo. Si el ancho es menor al calculado se expandirá al calculado más un margen. Este margen servirá para indicar que aún hay hueco para poner más bloques. Si el bloque introducido tiene la clase one entonces el ancho se le ajustará ya que ese bloque irá solo. Posteriormente se le suman unos valores para obtener el valor supuesto del AnchorPane padre, el cual se comparará con su valor actual y se ajustará. Si se expande esto se guardará en un booleano. Con respecto a la altura si la altura calculada es superior a la actual se expandirá la actual ajustándola a la calculada ya que bajo del bloque no va nada en un HBox. Posteriormente se buscará al elemento más alto dentro del nodo padre para ajustar su altura a esa. Si el nodo padre se expande ocurrirá lo mismo que con el ancho y esto se guardará en un booleano. Este booleano sirve para comprobar si ha habido expansión y por tanto vale la pena llamar al método recursivamente hasta llegar al AnchorPane de la clase. Cuando llega a la clase es cuando llama al método que expande la ventana de ser necesario.

El caso del VBox es más sencillo. Primero se calcula la altura y el ancho, la altura sumando las alturas de todos los bloques contenidos y el ancho calculando el ancho máximo. Luego procedemos a hacer lo mismo tanto para la altura como para el ancho, salvo que al ancho no le pondremos margen y al alto si nuevamente para indicar que se pueden poner más elementos. Compararemos este alto o ancho calculado con el actual, de ser mayor expandiremos el VBox en esa dimensión. Si se expande calcularemos una hipotética altura o ancho del nodo padre del VBox y la compararemos con la actual. Si resulta mayor expandiremos el nodo padre y marcaremos el booleano de expansión como verdadero. Si este booleano es verdadero y no estamos en el nodo clase volveremos a llamar a la misma función recursivamente. Si estamos en el AnchorPane clase entonces llamaremos a expandir la ventana.



Figura 29: Expansión de bloques contenedores

6.5.2 Contraer la clase

Esta función se encarga de contraer todos los bloques desde el que es llamado hasta llegar de nodo padre en nodo padre hasta la clase, si esto fuese necesario hasta un mínimo de tamaño. Esta función se llamará siempre que un bloque sea extraído de otro. Hay dos variantes de este método. Todo depende de si se le pasa un HBox o un VBox ya que no se contraen igual.

En el caso de un HBox, el cual apila los bloques horizontalmente y hacia la derecha. Por tanto si ha de decrecer ha de ser horizontalmente. Se calculará por tanto el ancho como la suma de todos los anchos de los bloques que contenga y el alto como el alto del bloque con más altura. Ahora se comparará el ancho que tiene el contenedor para ver si es mayor al calculado, si es así entonces decrecerá hasta este más un margen o hasta el ancho mínimo del contenedor. Luego se calculará el elemento más ancho del AnchorPane padre de este contenedor, esto se hace con el propósito de saber si se puede contraer este o no. Se comprobarán entonces todos los VBox y HBox del AnchorPane padre y el más ancho será guardado. Es entonces cuando se comprueba si este ancho es menor que el del padre más un margen. De ser así el padre decrecerá hasta esto o hasta su mínimo. Por otra parte, estas comprobaciones se tienen que dar también con la altura. Seguirán la misma lógica a la hora de empequeñecer el contenedor pero cambiarán a la hora de tratar con el padre. En este caso se mirará la diferencia guardada y se decrecerá en base a esta diferencia restándola o quedándose en el tamaño mínimo del padre. Si en algún momento se ha decrecido al padre se llamará recursivamente a esa función con el padre del padre hasta que se alcance el AnchorPane de la clase.

En caso de ser un VBox es ciertamente similar. Los VBox se expanden hacia abajo por el aplicado de bloques de manera vertical. Se calcula la altura sumando la altura de todos los bloques y el ancho obtenido el ancho más grande de todos los bloques contenidos. Con respecto a las comprobaciones que se harán posteriormente tanto en el caso de la altura como en el caso del ancho se buscará el elemento más alto o ancho del AnchorPane padre del VBox. Esto servirá para adaptar la altura o ancho a el tamaño de este elemento o en caso de ser lo suficientemente pequeño, al mínimo del AnchorPane. Posteriormente si se ha expandido durante la ejecución lo habremos marcado con un booleano para luego hacer llamadas recursivas hasta llegar al AnchorPane de la clase.



Figura 30: Contracción del bloque expandido

6.5.3 Expandir los imports

Esta función se encarga de expandir el cuadro de los imports y mover la clase para abajo para hacer hueco de ser necesario para contener los nuevos imports. A esta función la llama el DraggableMaker cuando introduce un import.

Lo que hace esta función es calcular la altura de todos los elementos de la parte contenedora de la zona de imports sumandola. Esta altura más un número, para dejar algo de espacio vacío bajo será la altura a comprobar. Si el tamaño actual de la zona contenedora de imports es menor a este valor entonces se actualiza a este valor. Luego en base a esto se suma la posición de la zona contenedora dentro de la zona de imports y un valor de margen para calcular si hace falta expandir el AnchorPane de la zona de imports. De ser así se expande , se mueve la zona de la clase para abajo, debido a que los imports están arriba de esta y además se expanden el ScrollPane y el AnchorPane que sirven de ventana. Esto es para poder contener el desplazamiento que se le ha realizado a la zona de la clase.

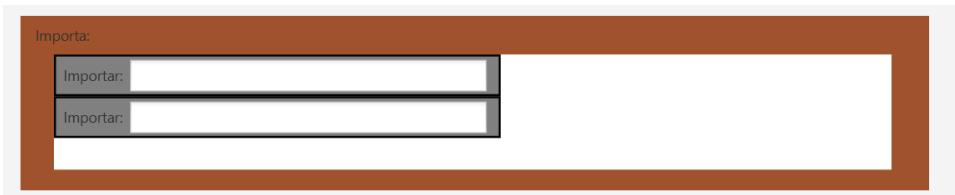


Figura 31: Expansión del área de imports

6.5.4 Contraer los imports

Esta función se encarga de contraer el cuadrado de los imports y mover la clase hacia arriba en caso de retirar imports de dentro hasta un tamaño mínimo. A esta función la llama el DragableMake cuando un bloque es extraído de los imports.

Su funcionamiento es similar al caso anterior. Primero se calculará la altura total de los bloques sumados que hay dentro de la zona contenedora de los imports. Esta altura más un margen para dejar espacio en blanco será comparada con su altura actual y si es menor entonces ha de decrecer. El decrecimiento ha de calcularse en función a si este provoca que el bloque sea menor a su altura mínima. Si ese es el caso entonces se decrecerá hasta la altura mínima y se guardará la diferencia. Sinó se decrecerá hasta la suma de las alturas de los imports más un margen y se guardará la diferencia. De la misma manera y gracias a la diferencia se calculará si es necesario decrecer el AnchorPane de la zona de imports. Si esto fuese necesario además se desplazaría la zona de la clase hacia arriba.

6.5.5 Expandir la ventana

Esta función se encarga de expandir tanto el ScrollPane como el AnchorPane que forman juntos la vista principal en caso de que el programa crezca demasiado. Es llamado cuando es necesario dentro del expand de la clase.

Su funcionamiento es simple. Si ha sido llamado es porque se ha de expandir la ventana por lo que primero obtiene la referencia al ScrollPane y al AnchorPane que forman la base de la ventana y luego incrementa su tamaño en función al tamaño del AnchorPane de la clase.

7 Resultados finales

La solución desarrollada acaba siendo convertida en una aplicación de escritorio. Por tanto su implantación no es tan compleja como la implantación de una aplicación web conectada a un servidor. Se tiene que pasar del código fuente del programa a una aplicación de escritorio. Además luego se comentará una traza del funcionamiento.

7.1 Paso a aplicación de escritorio

Lo que se hará es utilizar el IDE utilizando para convertir los archivos .java en archivos .jar. Este tipo de archivos pueden ser ejecutados, pero por lo general será necesario usar consola de comandos. El comando en cuestión sería el siguiente :

```
C:\Users\usuario1>jar -p "ruta/hacia/JavaFX" --add-modules javafx.controls,javafx.base,javafx.fxml,javafx.graphics,javafx.media,javafx.web -jar archivo.jar
```

Figura 32: Comando para ejecutar el .jar

Este comando es complicado de ejecutar por los usuarios, además la ruta hacia las librerías de JavaFX puede variar dependiendo de los usuarios. Por eso una mejor opción es convertir el .jar en un exe junto con las librerías de JavaFX. Para hacer esto hay varias maneras pero una de las más sencillas es mediante el comando Jpackage [46] al cual hay que indicarle tanto el jar como los módulos de los que depende. Este comando crearía el .exe e incluso nos permitiría más opciones dependiendo del sistema operativo.

Sin embargo, al no ser nuestra versión aún la más pulida se ha decidido optar por una aproximación similar más simple. Estos cuentan con archivos de configuración en .txt. En este archivo el usuario deberá poner la ruta de sus librerías de JavaFX. Seguidamente el ejecutable final será un archivo .bat, por lo menos para Windows, que tomará esta ruta y construirá en función a eso el comando a ejecutar para el funcionamiento del programa.

| | | | |
|--------------|------------------|-----------------------|----------|
| config.txt | 28/06/2023 19:52 | Documento de tex... | 1 KB |
| Facile_J.jar | 28/06/2023 19:41 | Executable Jar File | 8.754 KB |
| out.txt | 28/06/2023 19:54 | Documento de tex... | 1 KB |
| prog.bat | 28/06/2023 19:54 | Archivo por lotes ... | 1 KB |

Figura 33: Archivos para la ejecución

Esta será la estructura de archivos con el config.txt que contiene la ruta de las librerías JavaFX, el .jar con las clases compiladas del programa, el out.txt o archivo temporal donde se almacena el último resultado de la traducción de código y por último el .bat que ejecuta el comando anteriormente visto con la ruta que hay en el config.txt

Además se cambiará el comando de java a javaw. Este comando tiene la misma función que java pero cuenta con una ventaja específica. Permite el cierre de la consola de comandos sin que el programa cese su ejecución. Para un caso como estos es una gran ventaja no tener todo el rato la línea de comando abierta molestando.

7.2 Traza de la ejecución normal del programa

En este apartado procederemos a demostrar cómo sería un uso normal del programa. Para ello contaremos con un código de ejemplo en el que nos basaremos. La idea será replicar este código en lenguaje visual y traducirlo posteriormente.

```
1 package application;
2
3 public class Test{
4     public static void main (String[] args){
5         int a = 10;
6         for(int i = 0; i< a ; i++){
7             if(i % 2 == 0){
8                 System.out.println(i);
9             }else{
10                 System.out.println("No");
11             }
12         }
13     }
14 }
```

Figura 34: Código de ejemplo para la traza

Este código es un ejemplo de programa que se va a tratar de emular con el entorno. Para emularlo lo primero que se hará es abrir el programa. La primera imagen que se verá es la de la clase y la función main rellenas con nombres de ejemplo. Se modificará estos valores para adaptarse al ejemplo.

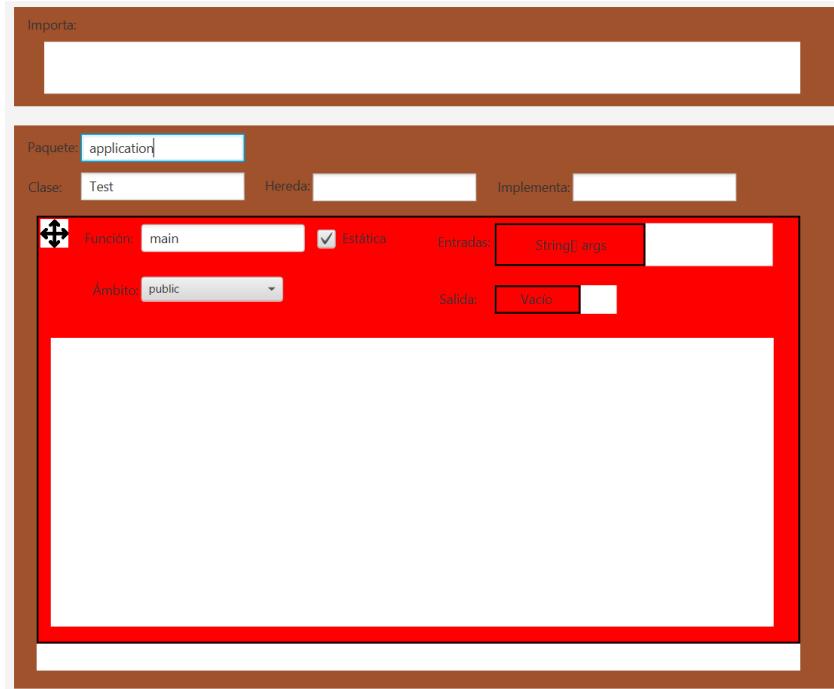


Figura 35: Área de programación con los parámetros correctos

Lo próximo que se debe hacer es ocuparse del código dentro de la función main. En este caso contamos con la inicialización de una variable y la asignación del valor 10 a ella. Además hay un bucle for que contiene más cosas. Se empezará por esto.

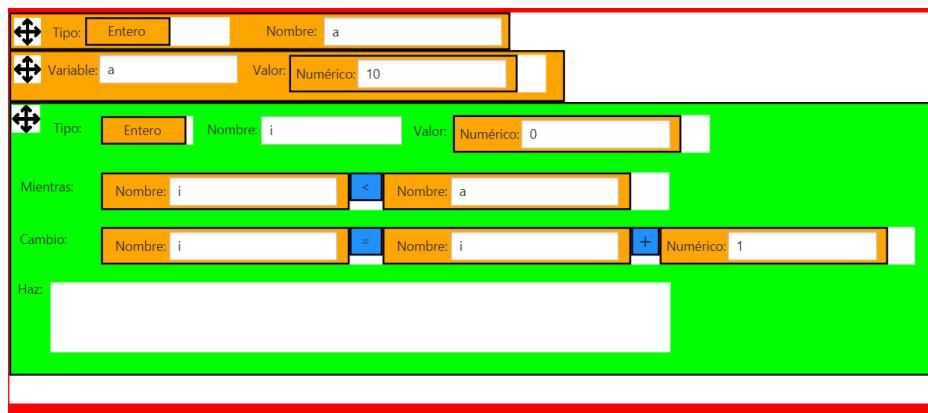


Figura 36: Primeras sentencias del ejemplo

Se ha rellenado la inicialización y la asignación por separado ya que así es como funciona el entorno. Además, se ha puesto el bucle for rellenando los campos con la configuración de arriba. Posteriormente dentro del bucle for se halla un if con su respectivo else con la condición de comprobar los números pares e imprimir en función del resultado. Se rellenará esta sección.

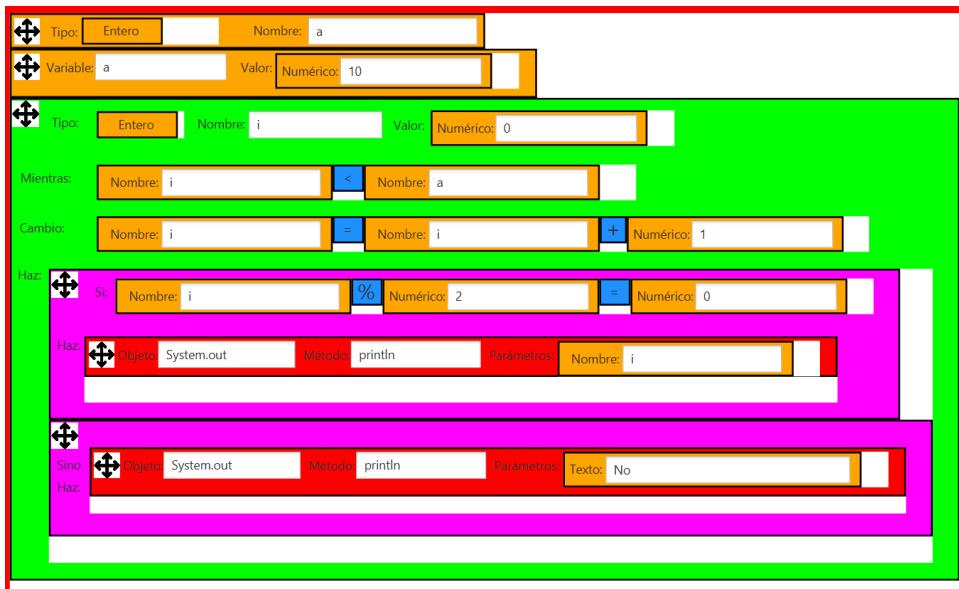


Figura 37: Programa visual resultante del ejemplo

Una vez rellenado los casos del if el programa ya estará terminado. Será momento de traducirlo para comprobar el resultado. Este se habrá escrito en un archivo de texto para conservarlo en caso de cerrar el programa sin guardarlo.



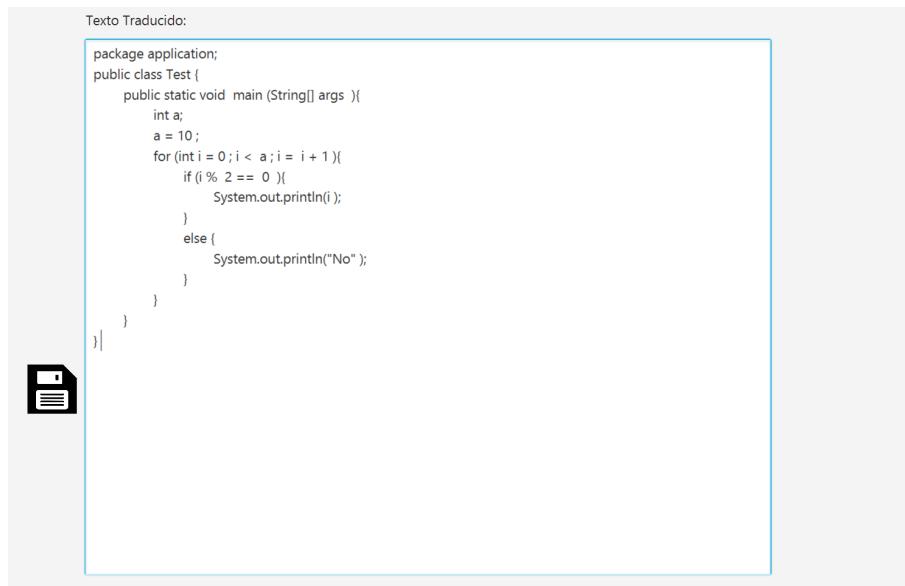
Figura 38: Archivo temporal generado del ejemplo

Este sería el archivo el cual tiene el siguiente contenido:

```
out.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
package application;
public class Test {
    public static void main (String[] args ){
        int a;
        a = 10 ;
        for (int i = 0 ; i < a ; i = i + 1 ){
            if (i % 2 == 0 ){
                System.out.println(i );
            }
            else {
                System.out.println("No" );
            }
        }
    }
}
```

Figura 39: Código dentro del archivo temporal del ejemplo

Se abrirá la segunda vista con el área de texto con el contenido del out.txt, es decir con la traducción.

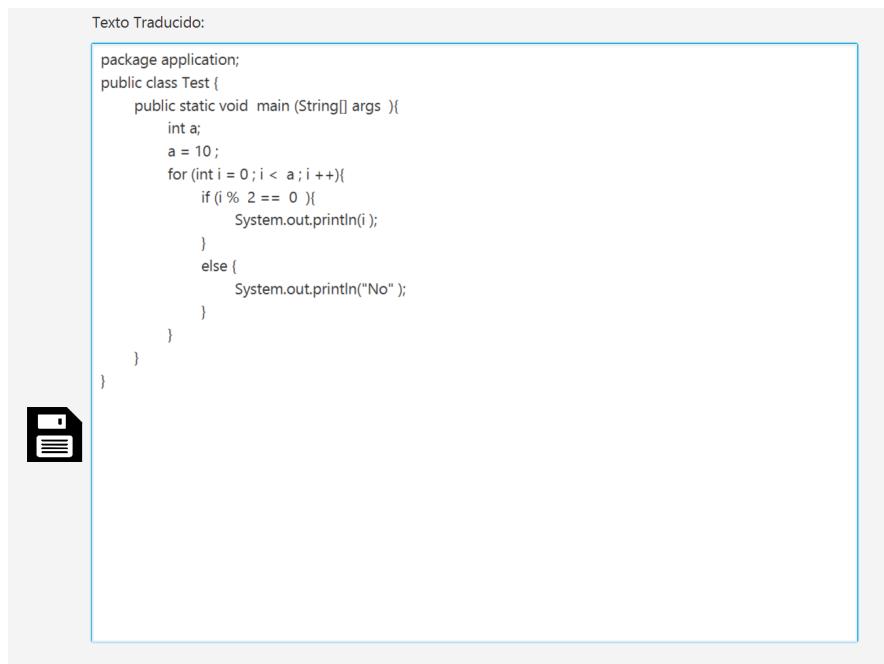


Texto Traducido:

```
package application;
public class Test {
    public static void main (String[] args ){
        int a;
        a = 10 ;
        for (int i = 0 ;i < a ;i = i + 1){
            if (i % 2 == 0 ){
                System.out.println(i );
            }
            else {
                System.out.println("No" );
            }
        }
    }
}
```

Figura 40: Segunda vista del ejemplo

Es aquí donde se puede cambiar por ejemplo la acción de iteración del bucle for. Al no contar en el programa con el operador ++ se ha tenido que optar por poner la versión larga. Si se cuenta con el conocimiento de Java suficiente para esto se puede editar el archivo y poner por ejemplo ++ manualmente.



Texto Traducido:

```
package application;
public class Test {
    public static void main (String[] args ){
        int a;
        a = 10 ;
        for (int i = 0 ;i < a ;i ++){
            if (i % 2 == 0 ){
                System.out.println(i );
            }
            else {
                System.out.println("No" );
            }
        }
    }
}
```

Figura 41: Edición en el código traducido del ejemplo

Finalmente se guardará el archivo pulsando el botón pertinente. Esto abrirá el selector de carpetas para elegir dónde se quiere guardar. El archivo se guardará automáticamente con el nombre de la clase que se haya puesto.

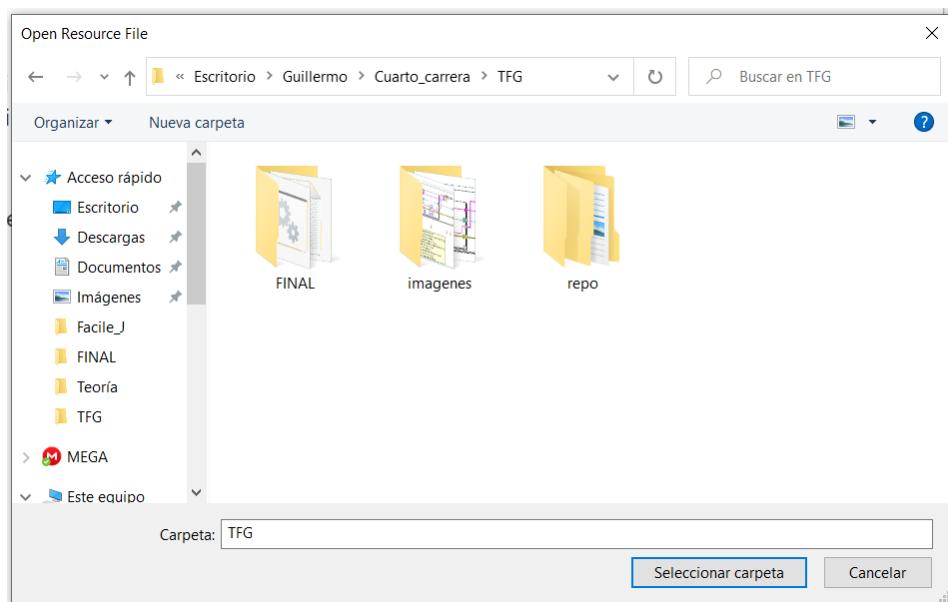


Figura 42: Ventana de selección de carpeta

Este sería el resultado final de la traza.

A screenshot of a Java code editor window. On the left, there is a preview icon of a spiral notebook labeled "Test.java". The main area shows the Java code for a class named "Test":

```
package application;
public class Test {
    public static void main (String[] args ){
        int a;
        a = 10 ;
        for (int i = 0 ; i < a ; i ++){
            if (i % 2 == 0 ){
                System.out.println(i );
            }
            else {
                System.out.println("No" );
            }
        }
    }
}
```

Figura 43: Archivo java resultado de la traza

8 Pruebas

En este capítulo se hablará de las pruebas realizadas para asegurar el correcto funcionamiento del programa. A pesar de la posición de este capítulo las pruebas se han ido realizando durante todo el desarrollo como parte de la metodología empleada. Estas pruebas se centran en comprobar que el funcionamiento del programa es el deseado. Se pondrá especial atención en que el programa cumpla las funciones que tiene de la manera esperada. La experiencia de utilizar este programa ha de ser satisfactoria para lograr un mejor aprendizaje.

Se van a realizar pruebas del tipo caja blanca para analizar más minuciosamente ciertas partes del programa. También se van a realizar pruebas de caja negra para centrarse más en la interfaz.

8.1 Pruebas de caja blanca

Estas pruebas son en las que se pretende estudiar minuciosamente todo el funcionamiento del sistema. Para eso se plantea seguir ciertos caminos de ejecución del programa para comprobar resultados.

Pruebas de caja blanca

Selección de todos los tipos de bloques para comprobar que se relacionó bien el bloque con su opción en el desplegable

Clic en un bloque fuera de un contenedor

Arrastrado de bloque

Soltar un bloque arrastrado fuera de un contenedor

Soltar un bloque arrastrado dentro de un contenedor

Comprobar que el contenedor se expanda siempre y cuando el bloque sobrepase la capacidad de este

Comprobar que sólo crezcan los contenedores y bloques que se vean forzados por su tamaño

Clic en un bloque dentro de un contenedor

Arrastrar bloque fuera de contenedor

Comprobar que si el contenedor no estaba en su tamaño mínimo este decrece

Comprobar que solo decrezcan los contenedores y bloques que no estén en su tamaño mínimo y que sean nodos ancestros del bloque movido

Comprobar que los tamaños crecidos y decrecidos están acorde al código

Borrar bloques

Comprobar desplazamiento hacia abajo al colocar bloque en import

Comprobar desplazamiento hacia arriba al extraer bloque en import siempre que esta área no haya alcanzado su tamaño mínimo

Intento de inserción de bloque H en un VBox

Intento de inserción de un bloque V en un HBox

Intento de inserción de bloques H y V en un HBox con etiqueta one

| |
|---|
| Intento de inserción de más de un elemento en HBox con etiqueta one |
| Intento de inserción de imports en HBox, VBox y HBox one sin etiqueta imp |
| Intento de inserción de elementos H, V y one en la zona de import |
| Arrastrado de bloques con bloques contenidos dentro |
| Uso de los checkboxes, campos de texto y desplegables de los bloques |
| Clic del botón de acción |
| Comprobación de funcionamiento de alerta en casos erróneos |
| Comprobación de traducción de cada bloque en el archivo de texto temporal |
| Realizado de todas las acciones anteriores con scroll horizontal |
| Realizado de todas las acciones anteriores con scroll vertical |
| Correcta navegación a la nueva ventana si no ha habido errores soportados en el código |
| Equivalencia entre el código creado y el código traducido en el text area |
| Modificado del contenido del text área |
| Cierre de ventana y vuelta a pulsar botón de acción |
| Guardado del archivo |
| Comprobación del contenido del archivo |
| Ejecución del archivo |
| Cierre total del programa |

8.2 Pruebas de caja negra

En estas pruebas ya no interesan todos los detalles de la ejecución del programa. El foco estará en entradas y salidas, como si el programa fuese una caja negra de la que no sabemos que hay dentro.

Pruebas de caja negra

| |
|--|
| Abrir el programa |
| Creación de bloques |
| Prueba de agarrar un bloque |
| Prueba de arrastrado |
| Prueba de soltado |
| Prueba de borrado de bloque |
| Prueba de creación de un programa arrastrando bloques |
| Uso del botón de acción |
| Prueba de modificación del texto de la traducción |
| Prueba de guardado del programa |
| Prueba de valores límite en la expansión |
| Prueba del funcionamiento del scroll |
| Cierre del programa |

9 Conclusiones

Este proyecto ha sido muy útil para enfrentarse por primera vez a un desarrollo algo más serio. Durante toda la etapa educativa se suele mantener al alumno en un entorno relativamente controlado y este trabajo deja probar un poco de la realidad.

Entre muchas de las cosas de las que ayuda a darse cuenta un proyecto así está la de que un gran desarrollo ha de basarse en unos cimientos sólidos y bien pensados. De no ser así la cantidad de errores será enorme. Incluso peor, se hará una bola de nieve que para el momento en el que se quiera arreglar será mejor rehacer todo el programa.

También ayuda a darse cuenta de lo complicado que es planificar cosas en función de unos recursos determinados, en este caso temporales. Es muy fácil caer en el error de intentar solventar un problema demasiado complejo o con un alcance muy amplio, sin embargo y a medida va pasando este tiempo será la propia realidad la que te devuelva a tu sitio. Por tanto, ocurrirá el caso de que habrá que recortar el alcance del proyecto.

Otra cuestión que ha afectado mucho en este desarrollo han sido las versiones de las herramientas utilizadas. Cuando uno crea software ha de tener en cuenta que este no quede obsoleto rápido. Sin embargo, hay veces que trabajar con lo más moderno no es compatible con las decisiones tomadas. Hay que priorizar siempre versiones estables y compatibles para evitar errores mayores.

Por otra parte, el desarrollo en sí ha radicado más bien en crear un entorno fiel a la idea original. Debido a esto se han tenido que descartar opciones más sencillas. Sin embargo, es satisfactorio poder ver un resultado que realmente se parezca a como se ha concebido un proyecto.

En resumidas cuentas, este ha sido un proyecto complejo, con objetivos ambiciosos tales como trasladar Java a un lenguaje visual inventado. Sin embargo, y a pesar de que el resultado final no es tan completo como Java este es suficiente, es funcional y puede llegar a cumplir su objetivo, que no olvidemos que es ayudar en el aprendizaje.

9.1 Relación de trabajo hecho con estudios cursados

El trabajo realizado ha sido en gran parte inspirado por mis estudios. Durante todo el grado prácticamente, sobre todo al inicio, se ha trabajado con Java para aprender sobre el mundo de la programación.

Por otra parte, al tratarse de traducción de código y muy inspirado en la idea de cómo funciona un compilador se pretendía trabajar con árboles. Gracias a asignaturas como Estructuras de Datos y Algoritmos se ha podido afrontar más sencillamente desde un inicio este problema.

Para crear una interfaz intuitiva se han seguido ciertas pautas dadas por la asignatura de Interfaces. Esto ha sido de gran importancia debido a que este era uno de los objetivos principales del proyecto.

Además, para llevar a cabo un proyecto así han hecho falta los conocimientos de Ingeniería de Software. En partes como la recopilación de requisitos, los diagramas de casos de usos y de componentes, los riesgos, etc. Inclusive en esta asignatura se llegó a programar interfaz en Java, cosa que en parte motivó este proyecto.

9.2 Futuras ampliaciones

El proyecto llega a su fin y con ello los resultados y los objetivos cumplidos. Desde un principio los objetivos han sido adaptados al tiempo de desarrollo de disponible teniendo en cuenta que este trabajo queda enmarcado en un TFG. Es por eso que fueran quedan varias posibilidades que llevarían este proyecto a un mayor nivel de completitud. A continuación, se enumeran una serie de características que por el tiempo han quedado fuera pero que completarían y mejorarían el proyecto.

- Añadir todos los bloques correspondientes a todas las funcionalidades que contiene Java.
 - Abarcar más tipos de errores e indicarlos de los que se vigilan ahora
 - Crear una interfaz que se adapte mejor al tamaño de la pantalla y de la ventana.
 - Añadir un sistema de versiones del entorno directamente relacionado con las versiones de Java.
 - Crear sistema de guardado y carga del propio lenguaje visual mediante formatos como el JSON para evitar perder el trabajo hecho.
 - Vigilar errores en la segunda vista.
 - Cambiar el acceso al programa en vez de un .bat . Esto se podría hacer convirtiendo el .jar en un exe o un msi instalable como se ha comentado anteriormente.
 - Permitir la ejecución del código desde el propio programa para comprobar su funcionamiento antes de guardarlo.

10 Referencias

[1] MIT AppInventor “Página de inicio de MIT AppInventor” 2021-2022 [En línea].

Disponible: <http://appinventor.mit.edu/>

[2] IBM “Desarrollo Iterativo” 2021 [En línea]

Disponible:

<https://www.ibm.com/docs/es/engineering-lifecycle-management-suite/lifecycle-management/6.0.3?topic=scenarios-iterative-development>

[3] Sandra Garrido Sotomayor “Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa” 2021 [En línea].

Disponible:

[https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/#:~:text=Por%20definici%C3%B3n%2C%20las%20metodolog%C3%ADas%20%C3%A1giles,las%20circunstancias%20espec%C3%BDficas%20del%20entorno.](https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/#:~:text=Por%20definici%C3%B3n%2C%20las%20metodolog%C3%ADas%20%C3%A1giles,las%20circunstancias%20espec%C3%ADficas%20del%20entorno.)

[4] Will Bernholz “Drag and Drop: The Other D&D” 2015 [En línea]

Disponible:

<https://www.dropsource.com/blog/drag-and-drop-the-other-dd/#:~:text=Originally%20called%20E2%80%9Cclick%20and%20drag,Xerox%20PARC's%20Star%20Information%20System.>

[5] Ossian Muscad “What Is Visual Programming & Why Is It Important? A General Guide” 2022 [En línea]

Disponible:

<https://datamyte.com/visual-programming/#:~:text=A%20Brief%20History%20of%20Visual%20Programming&text=The%20concept%20carried%20over%20to,released%20Visual%20Basic%20in%201991.>

[6] Adam Levine 2017 [En línea]

Disponible: <https://pxhere.com/es/photo/202494>

[7] Wikipedia “Visual programming language” 2023 [En línea]

Disponible: https://en.wikipedia.org/wiki/Visual_programming_language

[8] Aldhair.gsnt “File:Labview code example.png” 2021 [En línea]

Disponible: https://commons.wikimedia.org/wiki/File:Labview_code_example.png

[9] Wikipedia “Alice (software)” 2023 [En línea]

Disponible: [https://en.wikipedia.org/wiki/Alice_\(software\)](https://en.wikipedia.org/wiki/Alice_(software))

[10] Wikipedia “AgentSheets” 2023 [En línea]

Disponible: <https://en.wikipedia.org/wiki/AgentSheets>

[11] Wikipedia “Etoys (lenguaje de programación)” 2023 [En línea]

Disponible: [https://es.wikipedia.org/wiki/Etoys_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Etoys_(lenguaje_de_programaci%C3%B3n))

[12] Tek3327 File:Alice javaIDE.png” 2016 [En línea]

Disponible: https://commons.wikimedia.org/wiki/File:Alice_javaIDE.png

[13] Wikipedia “Scratch (programming language)” 2023 [En línea]

Disponible: [https://en.wikipedia.org/wiki/Scratch_\(programming_language\)](https://en.wikipedia.org/wiki/Scratch_(programming_language))

[14] Wikipedia “Flowgorithm” 2022 [En línea]

Disponible: <https://en.wikipedia.org/wiki/Flowgorithm>

[15] Wikipedia “Catrobat” 2023 [En línea]

Disponible: <https://en.wikipedia.org/wiki/Catrobat>

[16] Florian Schäffer “File:Mit app inventor 2 beta2.png” 2016 [En línea]

Disponible: https://commons.wikimedia.org/wiki/File:Mit_app_inventor_2_beta2.png

[17] Wikipedia “Eclipse Modeling Framework” 2022 [En línea]

Disponible: https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework

[18] Wikipedia “XML Metadata Interchange” 2019 [En línea]

Disponible: https://es.wikipedia.org/wiki/XML_Metadata_Interchange

[19] Dyhorus “File:Emf compare graphical.png” 2011 [En línea]

Disponible: https://commons.wikimedia.org/wiki/File:Emf_compare_graphical.png

[20] Bitelchux “Que es Blockly de Google” 2023 [En línea]

Disponible:

<https://applesana.es/formated/aprendeprogramando.es/cursos-online/blockly/introduccion-a-blockly/que-es-blockly/>

[21] David Ortiz “¿Puede un teléfono móvil reemplazar a un ordenador?” 2012 [En línea]

Disponible:

<https://www.xatakandroid.com/moviles-android/puede-un-telefono-movil-reemplazar-un-ordenador>

[22] Álvaro García “Este mapa muestra las horas de uso del móvil y el ordenador en cada país” 2023 [En línea]

Disponible:

https://www.larazon.es/tecnologia/este-mapa-muestra-horas-uso-movil-ordenador-cada-pais_20230503645211b52e790c0001a5b0b6.html#:~:text=Seg%C3%BAn%20un%20estudio%20reciente%2C%20el,el%2060%25%20utiliza%20computadoras%20personal

[23] W3Tech “Usage statistics of HTML for websites” 2023 [En línea]

Disponible:

https://w3techs.com/technologies/details/ml-html_any#:~:text=HTML%20is%20used%20by%2095.1,whose%20markup%20language%20we%20know

[24] IBM “Ventajas de Java” 2023 [En línea]

Disponible: <https://www.ibm.com/docs/es/aix/7.3?topic=monitoring-advantages-java>

[25] Netbrain “Java GUI frameworks. What to choose? Swing, SWT, AWT, SwingX, JGoodies, JavaFX, Apache Pivot?” 2013-2015 [En línea].

Disponible:

<https://stackoverflow.com/questions/7358775/java-gui-frameworks-what-to-choose-swing-swt-awt-swingx-jgoodies-javafx>

[26] Wikipedia “Abstract Window Toolkit” 2023 [En línea].

Disponible: https://en.wikipedia.org/wiki/Abstract_Window_Toolkit

[27] Wikipedia “Java version history” 2023 [En línea].

Disponible: https://en.wikipedia.org/wiki/Java_version_history

[28] Wikipedia “Swing (Java)” 2023 [En línea].

Disponible: [https://en.wikipedia.org/wiki/Swing_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java))

[29] JavaPoint “Java Swing Tutorial” 2021 [En línea].

Disponible: <https://www.javatpoint.com/java-swing>

[30] Wikipedia “Look and Feel” 2022 [En línea].

Disponible:

https://en.wikipedia.org/wiki/Look_and_feel#Look_and_Feel_in_Widget_Toolkits

[31] Wikipedia “Standard Widget Toolkit” 2023 [En línea].

Disponible: http://en.wikipedia.org/wiki/Standard_Widget_Toolkit

[32] Wikipedia “JavaFX” 2023 [En línea].

Disponible: <https://en.wikipedia.org/wiki/JavaFX>

[33] Oracle “JDK 11 Release Notes” 2018 [En línea].

Disponible: <https://www.oracle.com/java/technologies/javase/11-relnote-issues.html>

[34] Ionomos “Mobile First: el enfoque actual del diseño web móvil” 2019 [En línea].

Disponible:

<https://www.ionos.es/digitalguide/paginas-web/diseno-web/mobile-first-la-nueva-tendencia-del-diseno-web/>

[35] Wikipedia “Java Platform Module System” 2023 [En línea].

Disponible: https://en.wikipedia.org/wiki/Java_Platform_Module_System

[36] Jakob Jenkov JavaFX FXML” 2018 [En línea].

Disponible:

<https://jenkov.com/tutorials/javafx/fxml.html#:~:text=JavaFX%20FXML%20is%20an%20XML,rest%20of%20the%20application%20code.>

[37] Irina Fedortsova “Drag-and-Drop Feature in JavaFX Applications” 2013 [En línea].

Disponible: https://docs.oracle.com/javafx/2/drag_drop/jfxpub-drag_drop.htm

[38] Wikipedia “Rich web application” 2023 [En línea].

Disponible: https://en.wikipedia.org/wiki/Rich_web_application

[39] Apache Pivot “Frequently Asked Questions (FAQ)” 2023 [En línea].

Disponible: <https://pivot.apache.org/faq.html>

[40] QT “QT Jambi” 2021 [En línea].

Disponible: https://wiki.qt.io/Qt_Jambi

[41] Wikipedia “Google Web Toolkit” 20122 [En línea].

Disponible: https://es.wikipedia.org/wiki/Google_Web_Toolkit

[42] Wikipedia “Arquitectura dirigida por eventos” 2022 [En línea].

Disponible: https://es.wikipedia.org/wiki/Arquitectura_dirigida_por_eventos

[43] Wikipedia “Diseño estructurado” 2022 [En línea].

Disponible: https://es.wikipedia.org/wiki/Dise%C3%B1o_estructurado

[44] Oracle “Java Platform, Standard Edition (Java SE) 8” 2016 [En línea].

Disponible: <https://docs.oracle.com/javase/8/javase-clienttechnologies.htm>

[45] Matematicas Discretas “Capítulo 12: TEORIA DE ARBOLES BINARIOS” 2017 [En línea].

Disponible:

<https://medium.com/@matematicasdiscretaslibro/cap%C3%ADtulo-12-teoria-de-arboles-binarios-f731baf470c0>

[46] Oracle “The jpackage Command” 2023 [En línea].

Disponible: <https://docs.oracle.com/en/java/javase/17/docs/specs/man/jpackage.html>