PROTOTIPO DE SIMULACIÓN DE CREATIVIDAD COMPUTACIONAL

DAVID LOSILLA CADENAS

ROBERTO FERNÁNDEZ CORREA

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

CURSO 2018-2019

DIRECTOR:

JUAN PAVÓN MESTRAS

A nuestros padres, hermanos y abuelos.

AGRADECIMIENTOS

En primer lugar, nos gustaría dar las gracias a nuestro director, Juan Pavón Mestras, ya que este proyecto ha sido posible gracias a su ayuda y apoyo constante.

Así mismo queremos agradecer a nuestra familia todo su apoyo y compresión, ya que siempre han estado cuando les hemos necesitado y nos han animado a seguir cuando el dia solo tenia 24 horas y teníamos que trabajar, asistir a la universidad y sacar adelante este proyecto.

Por último nos gustaría dar las gracias al arquitecto Luis de Garrido, el cual ha aportando sus conocimientos sobre la materia y nos ha aportando la visión de un usuario final.

RESUMEN

Este trabajo tiene como objetivo la ayuda al estudio de la creatividad humana. Se apoya en los estudios del arquitecto Luis de Garrido, el cual ha investigado una serie de cualidades que se encuentran presentes en las diferentes estructuras arquitectónicas. El cerebro humano, inconscientemente reconoce estas cualidades para determinar si una construcción es considerada “bella” o “adecuada”.

Para ello se ha desarrollado un asistente creativo, el cual es capaz de generar diseños de estructuras arquitectónicas en base a una serie de condiciones, lo que puede ayudar en gran medida a expertos sobre la materia a estimular su creatividad y obtener una solución creativa.

Para su implementación, se ha desarrollado una aplicación en Java, la cual es capaz de generar planos de estructuras en base a una serie de reglas e información proporcionada por el usuario.

La creatividad de esta herramienta se basa en la combinación de reglas, por lo que está ligada directamente a la creatividad de estas. Se ha diseñado un mecanismo mediante el cual desarrolladores con conocimientos sobre arquitectura y creatividad puedan desarrollarlas de una manera sencilla, dedicando así sus esfuerzos en mejorar la creatividad de las mismas.

Palabras clave

Creatividad, reglas, plano, prototipo, desarrollo inteligente

SUMMARY

This work aims to help the study of human creativity. It is based on the studies of the architect Luis de Garrido, who has investigated a series of qualities found in a different architectural structure. The human brain unconsciously recognizes these qualities to determine if construction is considered "beautiful" or "adequate."

that is why a creative assistant has been developed, which is capable of generating architectural structure designs based on a series of conditions, which can greatly help experts on the subject, stimulate their creativity and obtain a creative solution.

its implementation, an application has been developed in Java, powerful enough to generate structure drawings based on a series of boundaries and information provided by the user.

The creativity of this tool is based on the combination of guidelines, so it is directly linked to their creativity. A mechanism has been designed where developers with knowledge about architecture and creativity can develop them in a simple way, thus dedicating their efforts to improve their creativity.

Keywords

Creativity, rules, plane, prototype, intelligent development

ÍNDICE

Índice de figuras

Figura 57: Diagrama de trabajo individual Roberto Fernández

1. INTRODUCCIÓN

En esta sección se va a proceder a explicar cuál ha sido la motivación del proyecto desarrollado, sus objetivos y el estado del arte.

1.1. MOTIVACIÓN

La creatividad, es la capacidad de generar nuevas ideas o conceptos, que habitualmente producen soluciones originales.

La motivación principal de este proyecto es la estimulación de la creatividad humana.

Para ello se planteó la idea de diseñar un prototipo que fuese capaz de diseñar planos con una un base de conocimiento dada mediante reglas y tuviese como entrada una serie de requisitos.

Este aportaría computacionalmente un gran número de soluciones y el estudio de las mismas, lo que podría ayudar al usuario a alcanzar diseños innovadores y despertar ideas que antes no hubiese tenido.

1.2. OBJETIVOS

El objetivo principal del proyecto es el desarrollo de un agente creativo, para ello ha sido desarrollada una aplicación cuyo ámbito es la arquitectura y más concretamente la generación de planos de planta.

Como caso de estudio se ha realizado una aplicación para diseñar basílicas cristianas y así demostrar su funcionamiento.

Los objetivos específicos son los siguientes:

- Desarrollo de una aplicación que permita la interactividad con el usuario y sea capaz de generar diferentes tipos de planos, teniendo en cuenta las exigencias del usuario.
- Posibilidad de exportar los diseños a formatos estándar como PNG y SVG para que puedan ser visualizados con cualquier visor de imágenes o para que puedan ser importados en herramientas como Autocad, para poder trabajar con ellos.
- Desarrollo de un sistema que, aunque en este caso se ha usado para la generación de planos de basílicas, se pueda usar para la generación de cualquier tipo de planos.
- Desarrollar un agente que sea capaz de generar soluciones de mayor o menor creatividad en función de las reglas que le han sido proporcionadas.
- Intentar alcanzar un notable grado de creatividad, el cual pueda incluso llegar a sorprender al usuario.

1.3. ESTRUCTURA DE LA MEMORIA

Esta memoria cuenta con la estructura que se detalla a continuación.

En el primer apartado podemos encontrar una primera aproximación al trabajo realizado, donde podemos encontrar cuál ha sido la motivación, los objetivos, la estructura de la memoria y el estado del arte.

En el segundo apartado encontraremos una explicación de las principales tecnologías que han sido utilizadas durante el desarrollo de este proyecto.

En el tercer apartado podemos encontrar una descripción de la interacción del usuario y la aplicación. En este punto podremos obtener una idea general del funcionamiento de la aplicación y que veremos como usuarios.

En el cuarto apartado se detallan los procesos para poder exportar planos a formatos svg y png. También se detalla cómo se puede exportar un estado concreto de la ejecución para reanudarla más tarde.

En el quinto apartado encontraremos un visión de la estructura del proyecto a un nivel más técnico, donde se podrá ver qué decisiones y arquitecturas se han tomado para el desarrollo.

En el sexto apartado está explicado en mayor profundidad el funcionamiento y arquitectura del motor de reglas.

En el séptimo apartado se explican las pautas para poder llevar a cabo una aplicación y así poder utilizarla para cualquier tipo de construcción.

En el octavo apartado se pasan a detallar las conclusiones obtenidas durante el desarrollo de este proyecto.

En el noveno apartado se encuentran las expectativas de trabajo futuro, con el cual se podría avanzar este proyecto y explotar su potencial.

En el décimo punto se explica la distribución del esfuerzo empleado en el desarrollo del proyecto.

En el undécimo punto se encuentran desglosadas las aportaciones individuales de cada uno de los participantes en este proyecto.

Por último en el punto doce encontramos las instrucciones para el acceso y el uso del proyecto.

1.4. ESTADO DEL ARTE

Actualmente no existe ninguna herramienta semejante o relacionada directamente, que hayamos conseguido encontrar.

En lo referente al software relacionado con la arquitectura podríamos destacar Autocad.

Autocad es uno de los programas más usados en el mundo de la arquitectura. Se trata de un software de pago que proporciona un gran abanico de herramientas para el diseño de planos en dos y tres dimensiones.

Sin embargo, no tiene ninguna herramientas para la creación de planos automáticamente, todo tiene que ser diseñado por el usuario y la creatividad tiene que ser proporcionada por el mismo.

Aunque Autocad no es capaz de crear planos automáticamente, si ofrece grandes características para su diseño, como pueden ser la capacidad de dibujo a mano alzada ya sea con el ratón o con un lápiz en dispositivos como tablets y Ipad, añadir anotaciones en las figuras diseñadas, o el cálculo de áreas, ángulos, radios...

Durante los últimos años, en lo referente al proceso creativo han surgido grandes avances en este sector, gracias al auge de tecnologías como inteligencia artificial o machine learning, uno de los logros más famosos fue la creación del primer cuadro diseñado por un algoritmo.

Actualmente hay diversos estudios que intentan fomentar la creatividad. Según estos estudios, hay diferentes tipos de creatividad. La creatividad plástica, es la que se relaciona con los colores, formas, volúmenes, texturas y proporciones, por tanto es la más ligada a la arquitectura.

Se utilizan diferentes técnicas para fomentar la creatividad, las más relevantes son las siguientes.

- **Lluvia de ideas(*brainstorm*):** las fases de este método son: planteamiento del problema, elección del grupo que realizará la lluvia de ideas y por último la evaluación de las ideas.

  La idea principal de esta técnica, es aprovechar las derivaciones de una idea general y así poder llegar a la resolución del problema.

  La aplicación, realiza algo parecido a un *brainstorming* ya que es capaz de aportar un gran número de planos (ideas), los cuales ayudan al usuario a encontrar la solución a su problema (diseño final)

- **Estimulación por imágenes:** Este proceso consiste en escoger un grupo de personas, y enseñarles una serie de imágenes, cada imagen deberá ser relacionada con la información del problema a resolver y de esta forma se intenta crear nuevas ideas.

1.INTRODUCTION

In this section, we will proceed to explain the motivation of the project developed, its objectives and the state of the art.

1.1. MOTIVATION

Creativity is the ability to generate new ideas or concepts, which usually produce original solutions.

The main motivation of this project is the study and stimulation of human creativity, providing a large number of computationally solutions and the study of them, in order to achieve innovative designs or a series of specifications.

A tool has been developed which attempts to simulate human creativity.

1.2.OBJECTIVES

The main objective of the project is the development of a creative agent that is capable of generating creatively plans for any kind of structure.

As a case of study, an application has been made on this to design basilicas and thus demonstrate its operation.

These are the specific objectives:

- Development of an application that allows interactivity with the user and hashing different types of plans, taking into account the user's requirements.
- Possibility to export the designs to standard formats such as PNG and SVG so that they can be visualized with any image viewer or so, it also can be imported into tools such as Autocad, in order to work with them.

- Development of a system that, although in this case has been used for the generation of basilica plans, can be used for the generation of any type of plans.

1.3. MEMORY STRUCTURE

This report has the structure detailed below.

In the first section, we can find a first approach to the work done, where we can find what has been the motivation, the objectives, the structure of the memory and the state of the art.

In the second section, we will find an explanation of the main technologies that have been used during the development of this project.

In the third section, we can find a description of the user's interaction and the application. At this point we can get a general idea of the application's operation and what we will see as users.

The fourth section details the processes to export plans to SVG and png formats. It also details how a specific state of execution can be detailed later.

In the fifth section, we will find a vision project's structure at a more technical level, where you can see what decisions and architectures have been taken for development.

In the sixth section, the operation and architecture of the rule engine is explained in greater depth.

The seventh section explains the guidelines to be able to carry out an application, therefore, be able to use it for any type of construction.

In the eighth section, the conclusions obtained during the development of this project are detailed.

In the ninth section are the expectations of future work, so this project could be taken to the next level and exploit its potential.

In the tenth point, the distribution of the effort used in the development of the project is explained.

In the eleventh point, the individual contributions of each of the participants in this project are broken down.

Finally, in section twelve we find the instructions for access and use of the project.

1.4. STATE OF THE ART

Currently, there is no such tool we have managed to find.

Regarding software related to architecture, we could highlight Autocad.

Autocad is one of the most used programs in the world of architecture. It is a nonfree software that provides a wide range of tools for designing plans in two and three dimensions.

However, it does not have any tools for creating plans automatically, everything has to be designed and provided by the user, so is creativity.

Regarding the creative process, we have not found any software that attempts to emulate human creativity.

There are currently several studies that try to encourage creativity. According to these studies, there are different types of creativity. The plastic creativity is the one that relates to colors, shapes, volumes, textures, and proportions, therefore it is the closest to architecture.

There are some different techniques to increase creativity, these are some of the most relevant ones will be working with.

- **Brainstorm:** The phases of this method are: problem statement, choice of the group that will brainstorm ideas and finally the evaluation of ideas.

  The main idea of this technique is to take advantage of the derivations of a general idea and thus be able to reach the resolution of the problem.

  The application performs something similar to a brainstorming since it is capable of providing a large number of plans (ideas), which help the user find the solution to his problem (final design)

- **Image stimulation**: This process consists of choosing a group of people, and showing them a series of images, each image must be related to the problem to be solved and in this way, we try to create new ideas.

2. TECNOLOGÍAS EMPLEADAS

En esta sección se van a resumir las tecnologías utilizadas en el desarrollo del proyecto.

2.1. JAVA

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Java fue el lenguaje elegido debido a la facilidad que aporta la clase swing para crear interfaces de usuario, y la que aporta la clase graphics2D para el dibujo de los planos, además de ser el lenguaje de programación con el que más experiencia tienen los integrantes del proyecto.

2.2. SWING

Swing es una biblioteca gráfica para java, con ella ha sido desarrollada la interfaz gráfica de la aplicación, la arquitectura Swing se basa en un Modelo-Vista-Controlador y su principal ventaja es la rapidez con la que permite crear interfaces de usuario.

Existen diferentes alternativas a la hora de crear interfaces gráficas en Java, como por ejemplo SWT o Java FX. Se eligió utilizar Swing debido a la familiaridad con este y la numerosa documentación existente tras su largo recorrido.

2.3. GRAPHICS2D

Esta es la clase fundamental para el renderizado 2D en Java. Esta clase extiende la clase original Graphics para proporcionar un control más sofisticado sobre la geometría, la transformación de coordenadas, la gestión del color y el diseño del texto.

Graphics2D es la clase usada para el diseño de los planos, fue elegida debido a la rapidez en el renderizado de planos.

2.4. BATIK APACHE

Batik es una librería de Java que permite obtener un archivo SVG o PNG a partir de un objeto de tipo Graphics2D.

Debido a que la calidad de la imagen que ofrece la aplicación es muy baja se decidió transformarlo en SVG Y PNG, ya que los planos contienen pequeños detalles en los cuales es necesario ampliar bastante para que puedan ser observados, además, el formato SVG se puede importar en distintos entornos como Autocad para seguir trabajando con el plano.

2.5. GIT

Es el sistema de control de versiones usado en el desarrollo del proyecto, permite trabajar en paralelo a todos los integrantes del equipo, la principal ventaja de Git frente a otros es su modelo de ramificación, ya que permite tener múltiples ramas independientes entre sí que más tarde pueden ser unidas.

Además el plugin Egit aporta una fácil integración con eclipse.

2.6. ECLIPSE

Eclipse ha sido el IDE utilizado para el desarrollo, las ventajas de este IDE son, un modo de depuración muy completo, autocompletado, resaltado de sintaxis y gran variedad de plugins que facilitan el desarrollo del proyecto.

3. INTERACCIÓN CON EL USUARIO

En esta sección, se van a explicar las diferentes funcionalidades de la aplicación que interactúan con el usuario. Esta explicación será basada sobre la aplicación de basílicas generada, pero sería extrapolable a cualquier aplicación realizada.

3.1. PANTALLA INICIAL

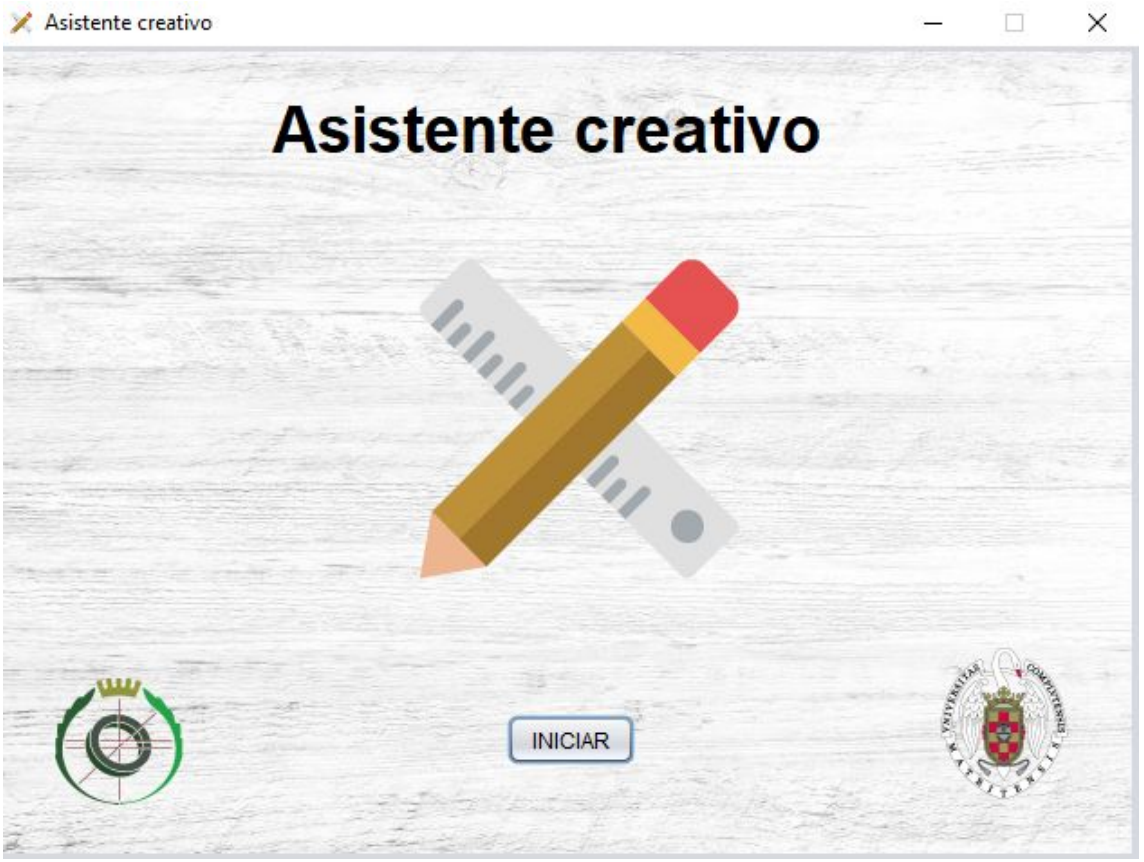


Figura 1: Pantalla inicial de la aplicación

Esta es la primera pantalla que verá el usuario al iniciar la aplicación, una vez haga click en el botón iniciar se mostrarán los siguientes tipos de ejecución.

3.2. MENÚS EJECUCIÓN

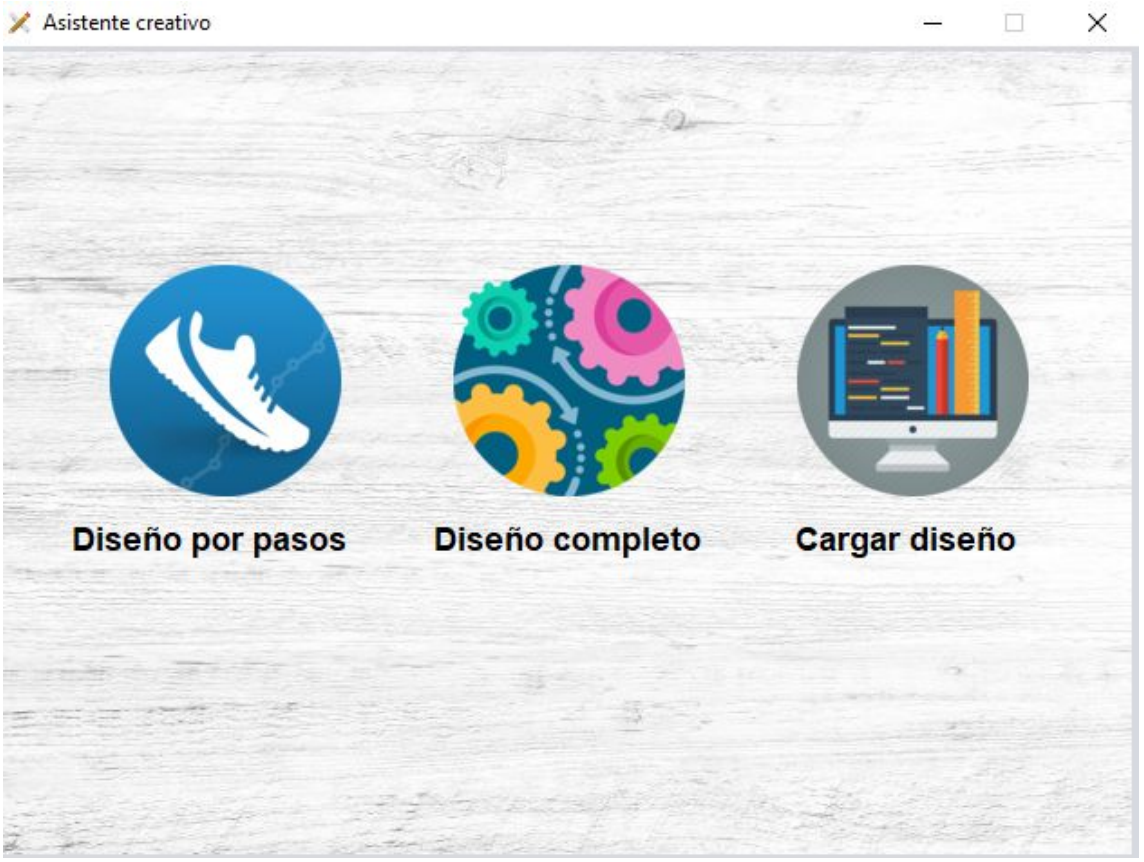


Figura 2: Modos de ejecución

En esta sección se van a explicar los diferentes modos de ejecución que tiene la herramienta.

3.2.1 DISEÑO POR PASOS

En este modo, se puede observar el proceso de creación de la basílica paso a paso. En en cada paso, se pueden observar los nuevos elementos arquitectónicos creados y elegir, o bien continuar con la forma en la que se han creado dichos elementos, o bien descartarlo y hacer que se construyan de diferente manera.

3.2.2 DISEÑO COMPLETO

En este modo, se ejecutarán todos los pasos de la basílica en una sola ejecución, y se mostrará al usuario un plano de la construcción completo, sin necesidad de que evalúe ninguna de las fases o etapas.

3.2.3 CARGAR DISEÑO

Cuando se esté ejecutando alguno de los modos anteriores, se permite guardar el progreso para continuar más adelante en el punto en el que se estaba anteriormente. Este modo permite cargar ese plano y seguir trabajando con él sin necesidad de repetir ningún proceso, manteniendo la configuración y todos los datos que se hubiesen introducido.

3.3. DATOS PARA LA EJECUCIÓN

La información proporcionada por el usuario es muy importante durante la ejecución, ya que esta condicionará en gran medida el producto final obtenido.

El sistema realizará una serie de preguntas al usuario, estas preguntas serán utilizadas para determinar cómo tienen que ser los elementos arquitectónicos que constituyen la basílica.

Una vez el usuario haya iniciado el modo paso por paso o diseño completo, se le mostrarán la siguiente ventana, donde determinará cuánta información desea introducir para la ejecución.

Destacar que siempre a mayor información proporcionada por el usuario se dará una menor creatividad, pero el producto final será más ajustado a sus necesidades.

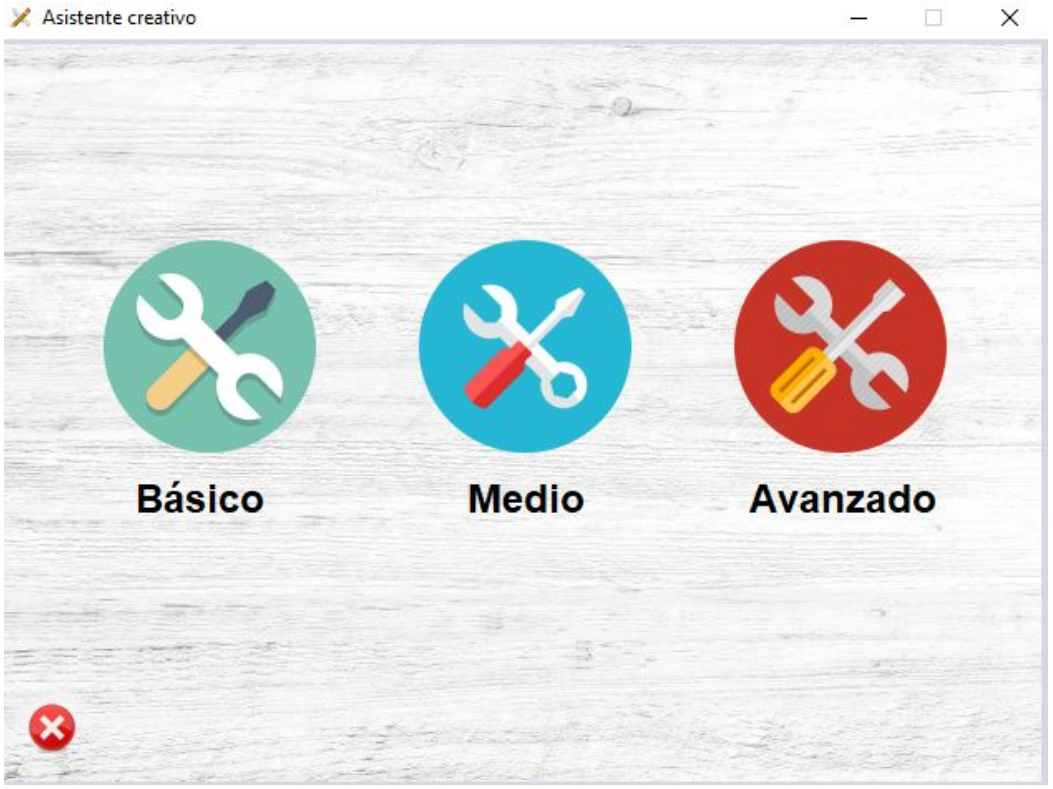


Figura 3: Niveles de preguntas

3.3.1 NIVEL BÁSICO

En este nivel, no se realizarán preguntas al usuario, por tanto la basílica será construida de forma totalmente aleatoria.

Este sería el punto donde más creativo podría llegar a ser el producto final debido a que no esta limitado por ningun requisito.

3.3.2 NIVEL MEDIO

En el nivel medio, serán realizadas una batería de preguntas generales que determinarán la forma de la basílica, estas preguntas pueden ser contestadas, o de lo contrario ignoradas.

En la esquina superior izquierda, se muestra el número de pregunta actual y el total de preguntas, los botones de la esquina superior derecha, son utilizados para avanzar o retroceder en las preguntas.

Figura 4: Niveles de preguntas medio

El botón con el dibujo de una X en la esquina inferior izquierda, es utilizado para salir, el botón con el dibujo de una papelera sirve para borrar la opción seleccionada y por último el botón de la esquina inferior derecha, se utiliza para iniciar la ejecución de la basílica,

3.3.3 NIVEL AVANZADO

La diferencia entre el nivel avanzado y el medio, es que además de las preguntas realizadas en el nivel medio, serán realizadas preguntas más específicas sobre los elementos de la basílica.

Se ha decidido dividir la batería de preguntas en estas dos secciones para no saturar al usuario, ya que un exceso de preguntas desde un primer momento puede provocar que el usuario pierda el interés.

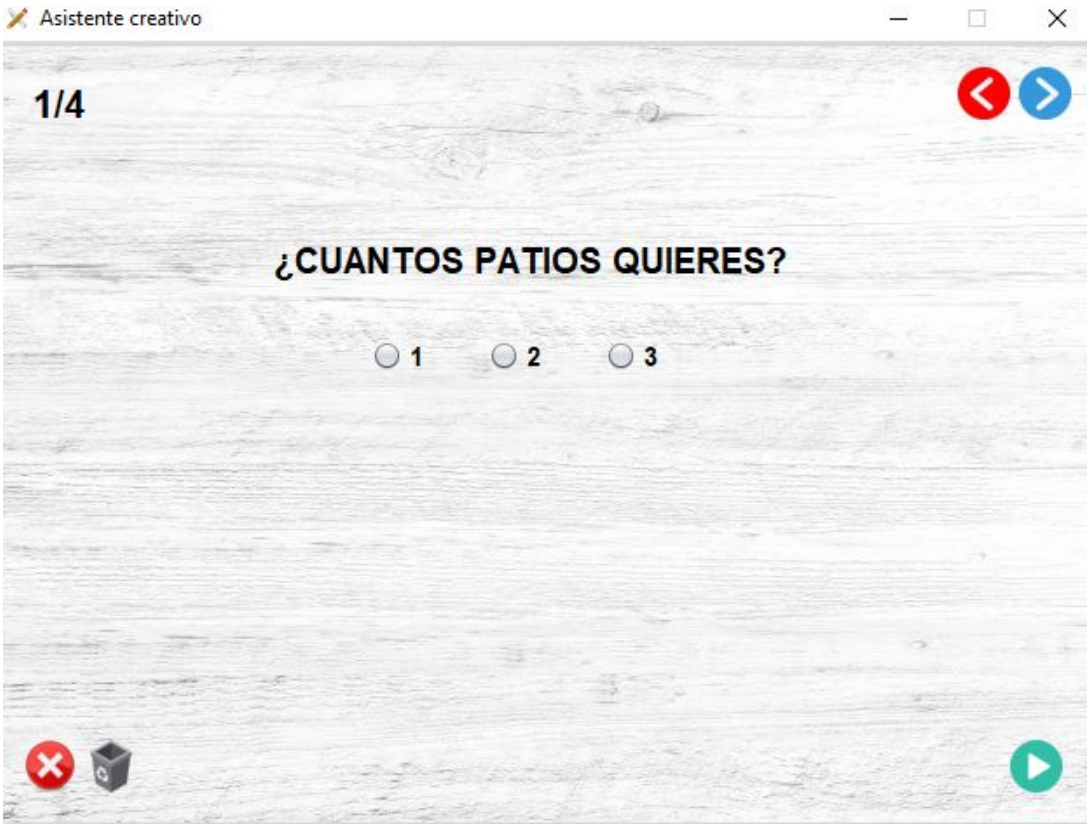


Figura 5: Niveles de preguntas avanzado

3.4. VENTANAS EJECUCIÓN

3.4.1 MODO PASO POR PASO

En esta ventana, como se ha mencionado anteriormente, se mostrará cada fase de la ejecución, donde se puede ir viendo el estado del plano.

La creación de una basílica, está constituida por diferentes fases, cada una de estas fases contiene un conjunto de reglas aplicables, el sistema, teniendo en cuenta las respuestas del usuario a las preguntas, elegirá la regla más adecuada en cada caso.

El usuario tendrá siempre la posibilidad de descartar los cambios realizados en cualquier fase, así como de guardar el estado actual o exportar los resultados a SVG o PNG.

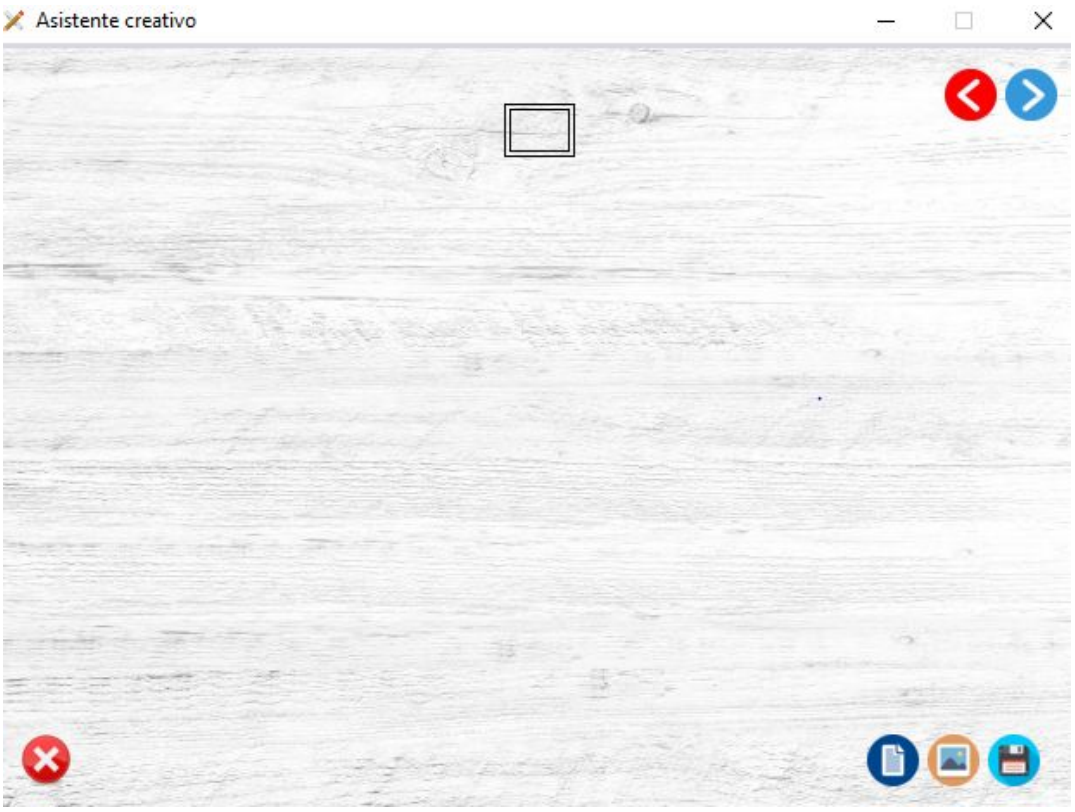


Figura 6: Ventana de ejecución en modo paso por paso

Figura 7: Botón que guarda el estado actual de la basílica

El botón seleccionado es utilizado para guardar el estado actual de la basílica, será abierta una ventana de diálogo donde el usuario podrá elegir donde guardar el archivo y más tarde poder cargarla para seguir trabajando con ella.

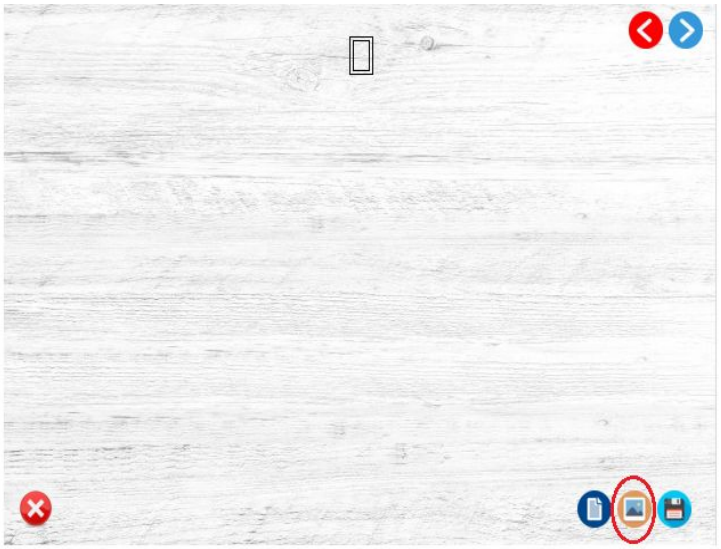


Figura 8: Botón que permite exportar el diseño a PNG y SVG

Este botón sirve para exportar el diseño actual en formato PNG o SVG, al igual que en el caso anterior, también se abrirá una ventana de diálogo donde el usuario podrá elegir la ubicación de destino en la cual serán guardados sus archivos.

Figura 9: Botón que muestra los estados y reglas ejecutadas

Este botón se utiliza para mostrar los estados y la regla ejecutada en cada estado. Una vez pulsado el botón será mostrada al usuario la siguiente ventana.

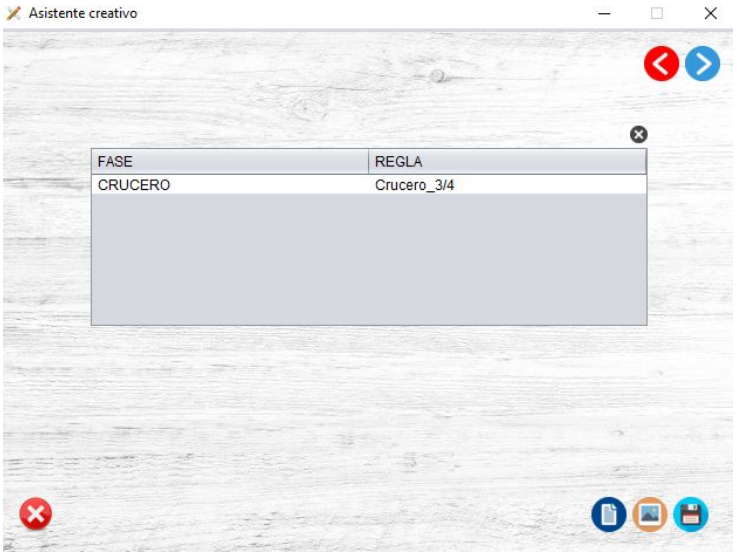


Figura 10: Ejemplo de fase y regla

En este caso, se ha ejecutado la fase crucero, y en ella la regla "Crucero_3/4", la cual ha creado un crucero con proporción ¾.

Figura 11: Botón encargado de avanzar de fase o cambiar de regla

Con el boton rojo se cambia la regla utilizada en esa fase, y con el botón azul se ejecuta la siguiente fase.

Si se pulsa el botón rojo, se aplica otra regla y por tanto se dibuja otro tipo de elemento o elementos arquitectónicos, en este caso será cambiado el crucero como se puede apreciar en el siguiente ejemplo.

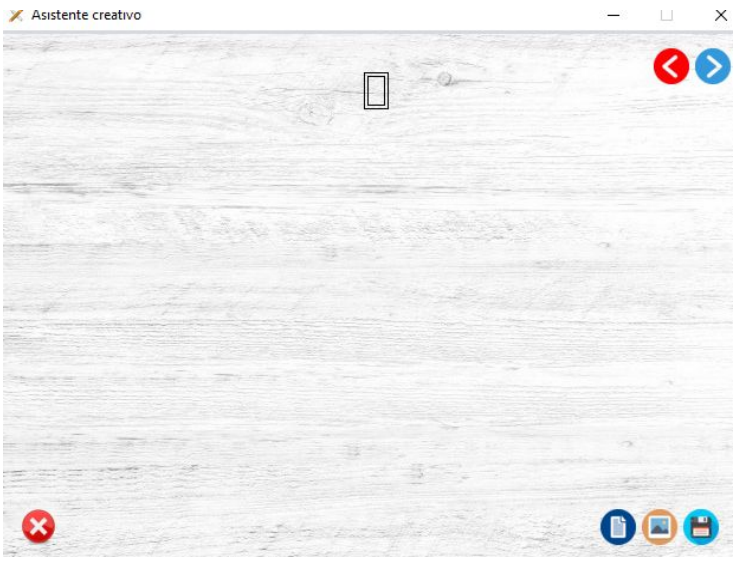


Figura 12: Ejemplo de cambio de regla

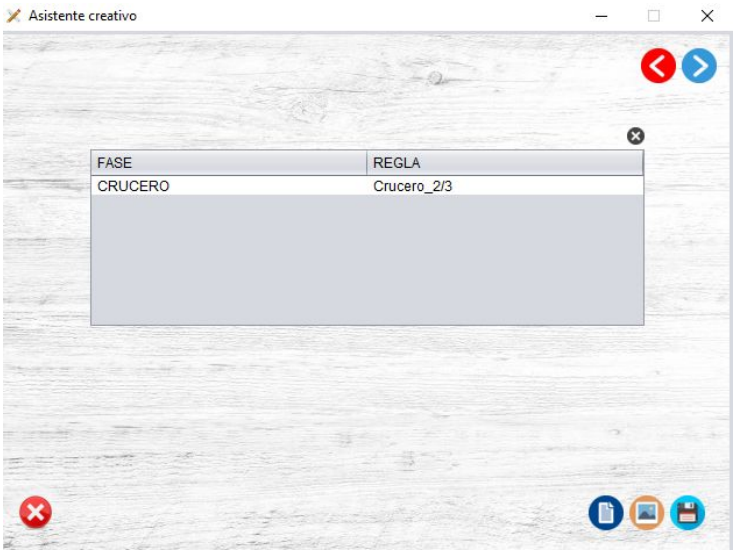


Figura 13: Pantalla que muestra el cambio de regla

En este caso, como podemos observar se ha aplicado la regla “Crucero_2/3” en la fase crucero.

Por otro lado, si hacemos click en el botón azul será ejecutada la siguiente fase.

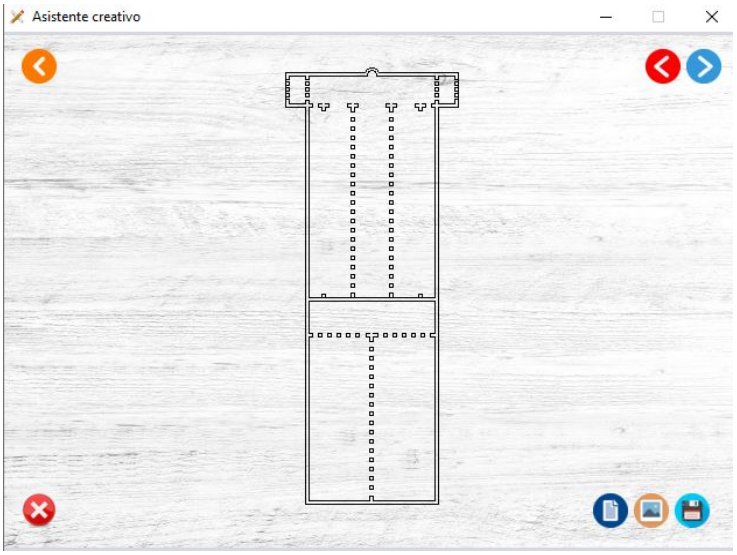


Figura 14: Imagen del modo de ejecución paso por paso

Como se puede apreciar en la imágen superior, han sido ejecutadas las siguientes fases del plano pulsando el botón azul. El botón naranja situado en la esquina superior izquierda, da la posibilidad de retroceder a la fase anterior

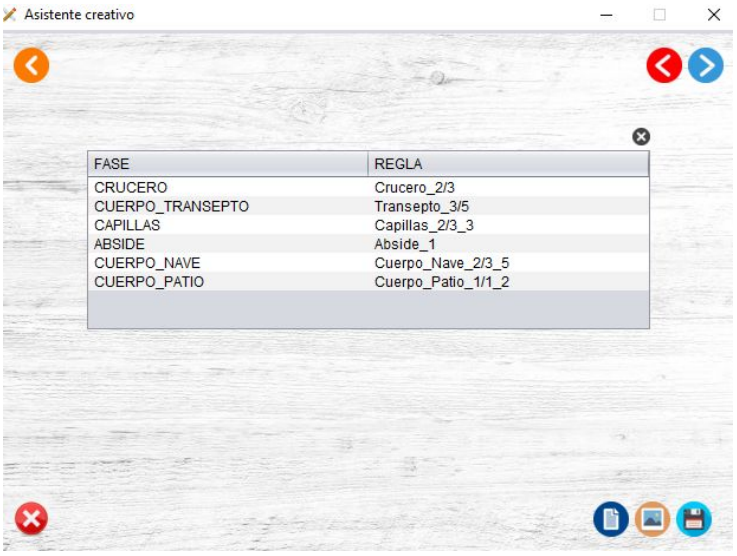


Figura 15: Ejemplo de salto hacia la fase anterior 1

En esta imágen, podemos ver las fases y reglas ejecutadas en el plano antes de que el botón naranja sea pulsado.

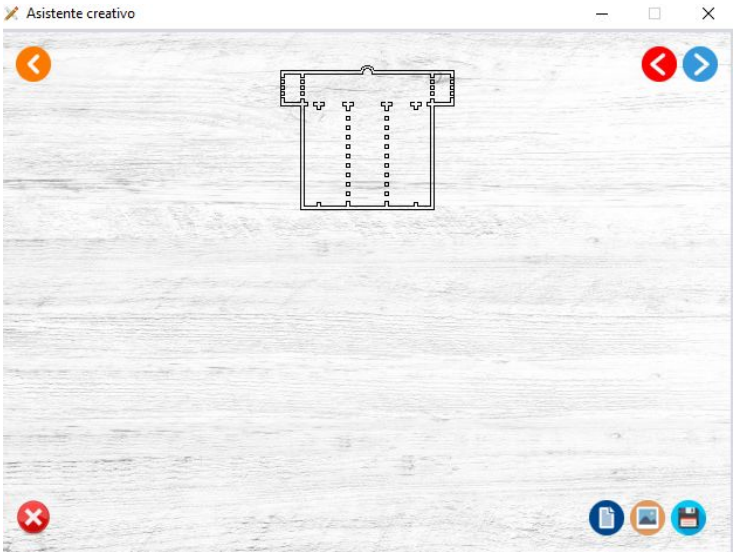


Figura 16: Ejemplo de salto hacia la fase anterior 2

Como se puede apreciar, al pulsar el botón, se ha vuelto a la fase anterior, ha desaparecido el cuerpo del patio de la basílica y actualmente está en la fase cuerpo de nave

Figura 17: Ejemplo de salto hacia la fase anterior 3

Al ver nuevamente las fases y reglas aplicadas en ellas, se puede observar como la fase “Cuerpo_Patio” ha sido eliminada.

3.4.2 DISEÑO COMPLETO

En este modo, después de responder a las preguntas, se generará un plano completo de la basílica. La ventana final se explicará más detenidamente en la siguiente sección.

3.5. PRODUCTO FINAL

Una vez se hayan ejecutado todas las fases en el diseño por pasos, o se haya creado el plano de una basílica directamente como en el diseño completo, se mostrará la ventana con el plano final de la basílica.

Figura 18: Plano final

Una vez el usuario haya llegado a esta ventana, tendrá las opciones que se detallan a continuación.

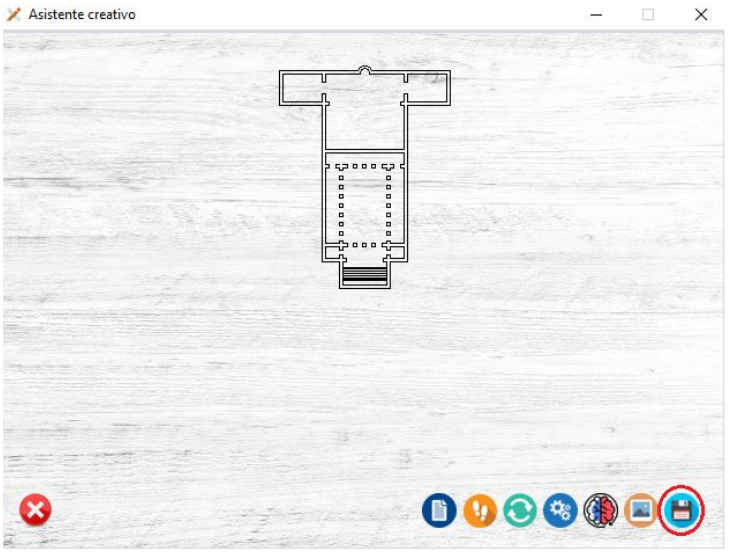


Figura 19: Botón que guarda el estado actual de la basílica

Como se ha mencionado anteriormente, este botón sirve para guardar el diseño. y poder seguir trabajando en él más tarde.

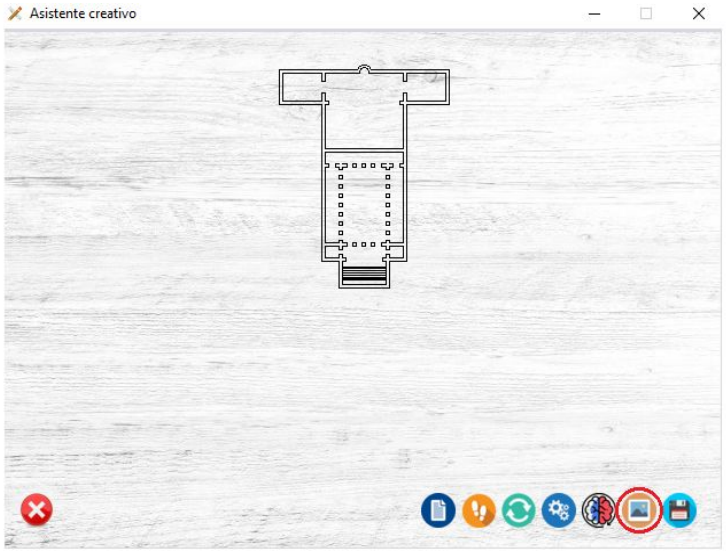


Figura 20: Exportación del diseño en formato PNG o SVG

Al igual que en el modo paso por paso, este botón permite exportar el diseño a los formatos PNG y SVG.

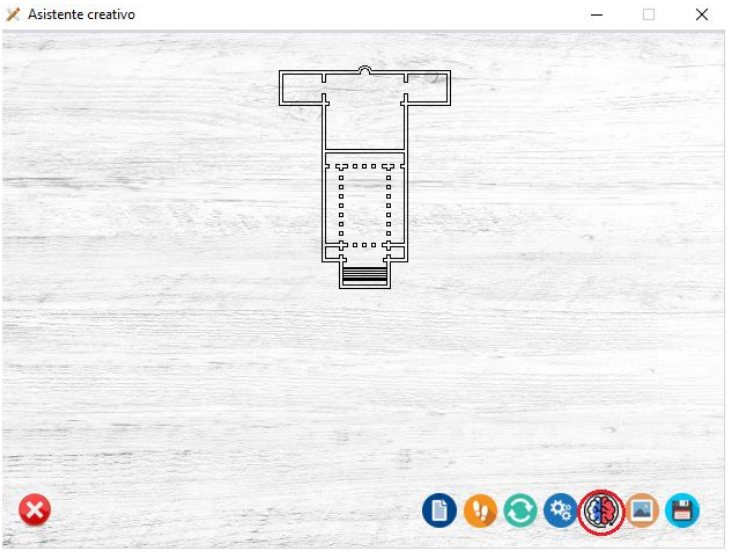


Figura 21: Botón que inicia las preguntas una vez el plano ha sido terminado

Esta opción realizará al usuario una serie de preguntas acerca del plano de la basílica, y una vez contestadas realizará otro plano ajustándose a las nuevas exigencias del usuario.

En la siguiente imágen, podemos ver un ejemplo de pregunta.

Figura 22: Ejemplo de pregunta

Figura 23: Botón que permite cambiar las respuestas a las preguntas iniciales

Cuando el usuario elija esta opción, se le mostrará el cuestionario que relleno antes de iniciar la ejecución del plano, para así poder modificar sus respuestas.

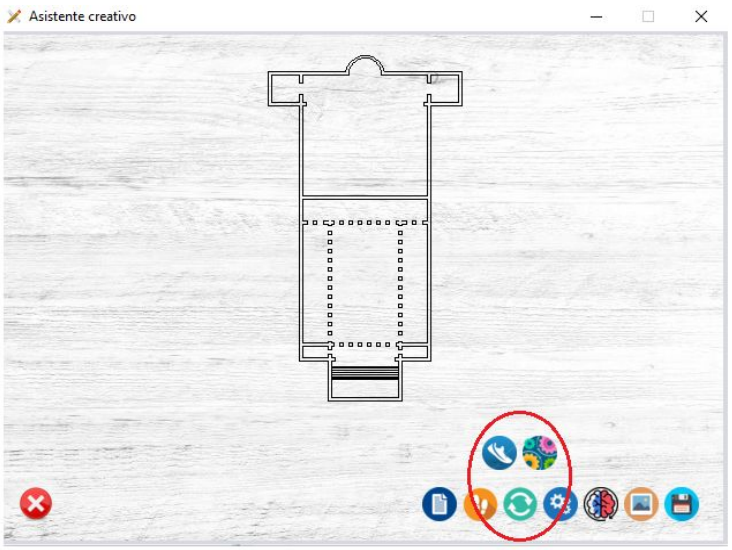


Figura 24: Botón que permite relanzar la ejecución en modo paso por paso, o diseño completo

Desde este botón se puede relanzar el diseño eligiendo el modo paso por paso o bien eligiendo diseño completo.

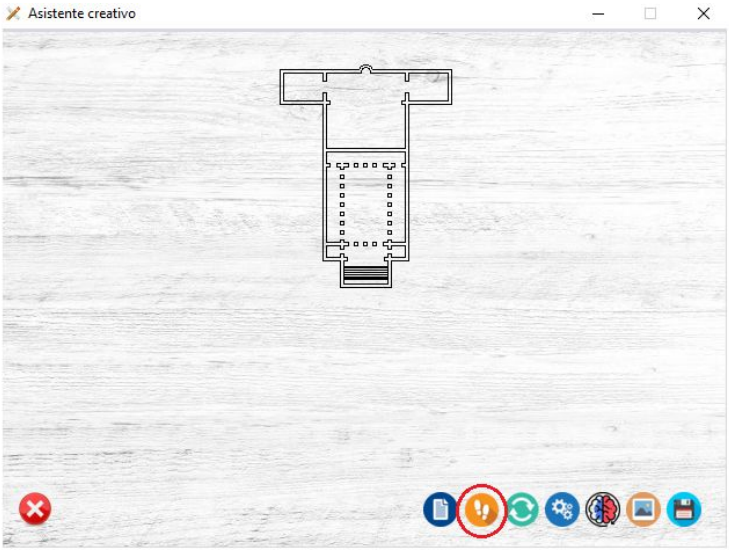


Figura 25: Botón que permite, una vez acabada la basílica modificar las reglas de sus fases

Este botón permite al usuario modificar las fases, de forma similar al diseño paso por paso. Una vez pulsado el botón, la ventana que verá el usuario será similar a la siguiente.

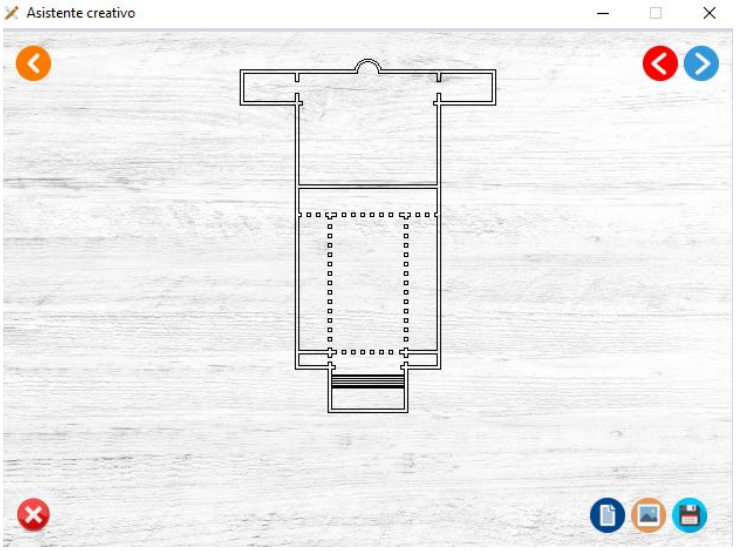


Figura 26: Ejemplo de ventana que permite modificar las reglas de las diferentes fases

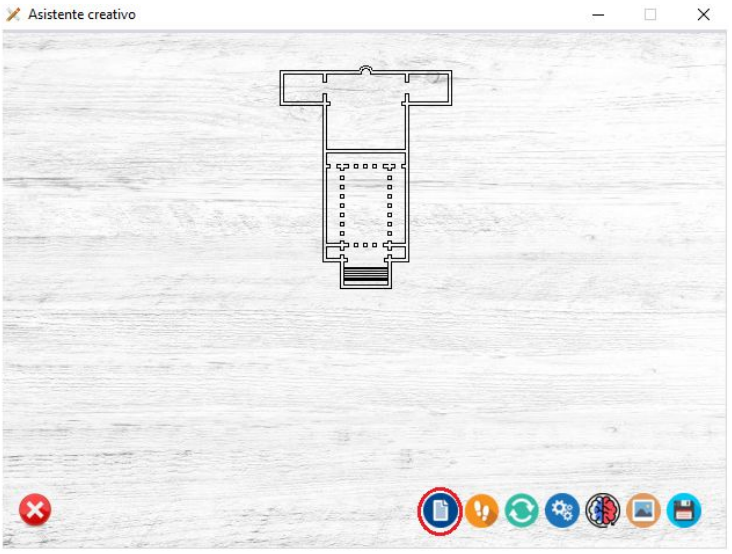


Figura 27: Botón que muestra las fases y regla ejecutada en cada fase

Por último, haciendo click en este botón, serán mostradas todas las fases de la basílica y la regla aplicada en cada una de ellas.

4. EXPORTACIÓN DE RESULTADOS

En este apartado se pasan a detallar las posibilidades de exportación de la herramienta, así como una visión general de su implementación.

4.1. EXPORTACIÓN DE PLANOS

Todos los planos generados, así como, los planos de fases intermedias pueden ser exportados tanto en formato PNG como en SVG.

El formato PNG permite ser visualizado en cualquier dispositivo ya que es un formato muy extendido y en cualquier dispositivo encontramos un visor para este instalado.

El problema de este formato es su baja calidad a la hora de ampliar para poder apreciar el plano con detalle.

EL formato SVG soluciona el problema de calidad de PNG, ya que este cuenta con una gran calidad y mantiene la nitidez a la hora de ampliar. SVG también puede ser importado en multitud de herramientas para editarlo y continuar trabajando con él, como por ejemplo, AutoCad.

El problema de SVG radica en que no todos los dispositivos cuentan con visores instalados para este formato.

Por las virtudes y problemas enumerados anteriormente, esta herramienta genera la exportación en ambos formatos para así intentar cubrir todas las necesidades de los usuarios.

En todas las ventanas donde visualizamos un plano, a través del icono seleccionado, podemos iniciar la exportación. Este nos abrirá un diálogo, a través del cual podemos elegir la ubicación y el nombre de los archivos que se van a generar.

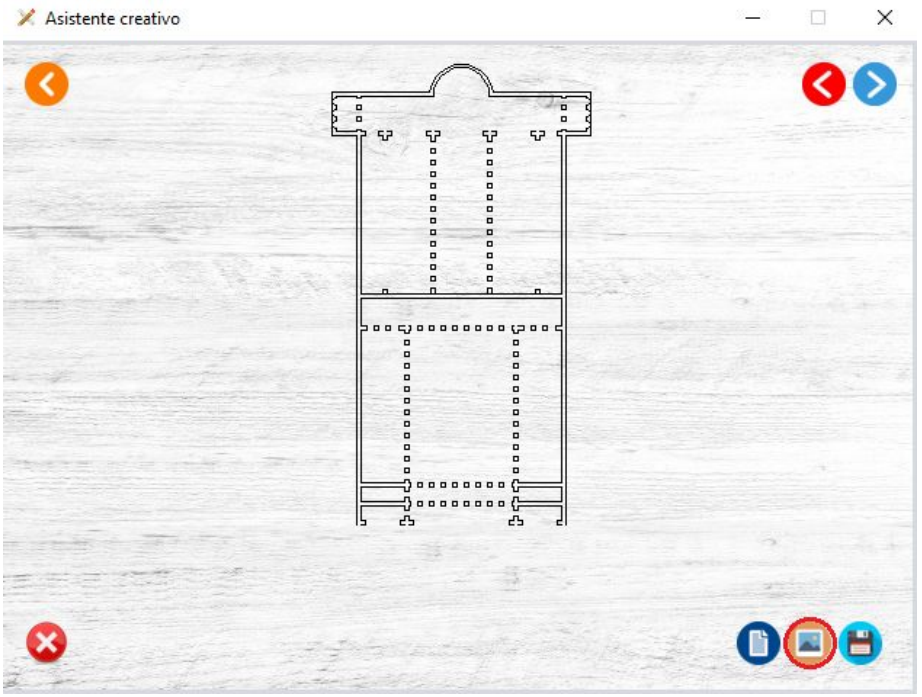


Figura 28: Plano de basílica

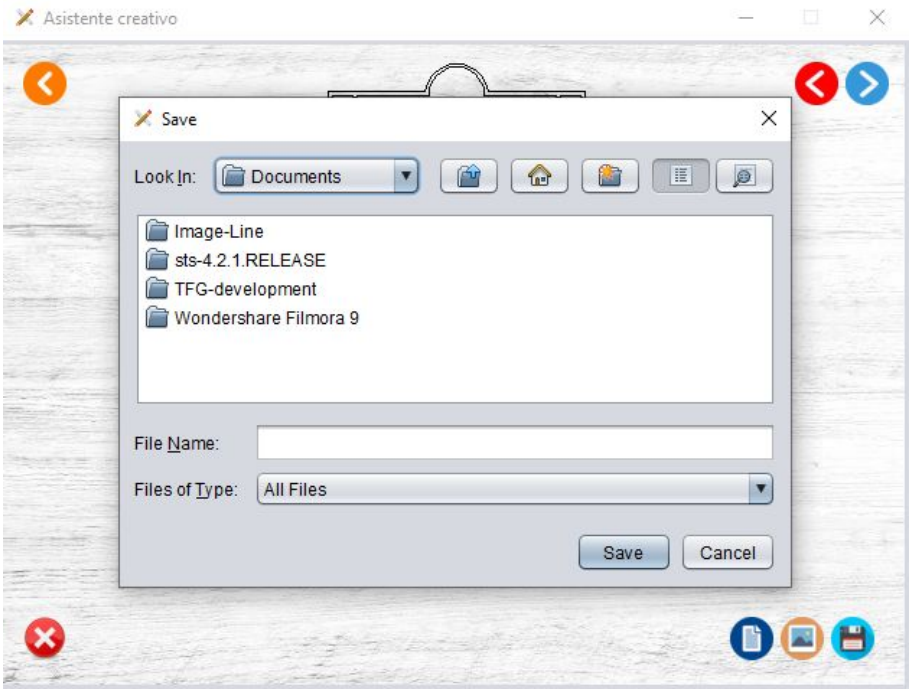


Figura 29: Ventana de diálogo para exportación del plano en formato PNG y SVG

4.2. EXPORTACIÓN DE ESTADOS DE LA EJECUCIÓN

La selección de un plano que se adecue a las necesidades del usuario es una tarea que puede llevar una gran cantidad de tiempo. Como se ha visto en otros puntos, en un plano se pueden modificar fases intermedias para poder ajustarlo a las necesidades requeridas.

Esta herramienta cuenta con la posibilidad de exportar un estado concreto de la ejecución, ya sea un producto final o una fase intermedia.

La funcionalidad de exportar el estado, otorga mucha flexibilidad al usuario, ya que tiene la posibilidad de guardar el punto en el que se encuentra para proseguir en otro momento sin perder todo su trabajo. También otorga la ventaja de poder guardar un diseño que no se adecua completamente pero que puede ser un gran candidato en otro momento.

La exportación se realiza mediante la serialización de objetos Java, lo que permite extraerlos en archivos, los cuales hemos dotado de la extensión .edificio, para más tarde cargarlos y proseguir con la ejecución.

En todas las ventanas donde visualizamos un estado concreto, a través del icono seleccionado, podemos iniciar la exportación. Este nos abrirá un diálogo, a través del cual podemos elegir la ubicación y el nombre del archivo que se van a generar.

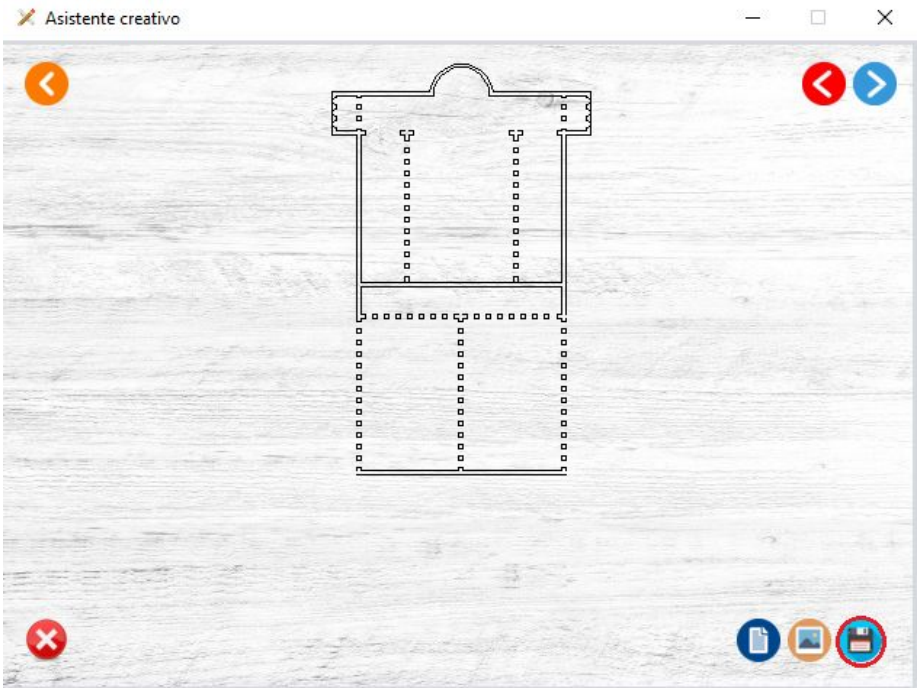


Figura 30: Plano de la basílica

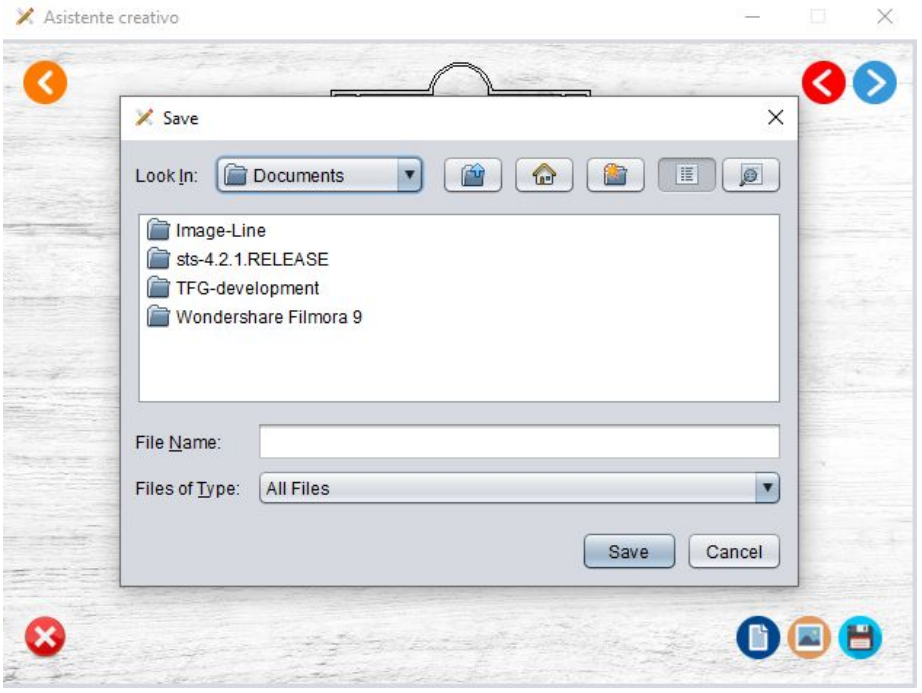


Figura 31: Ventana de diálogo para la exportación del diseño

5. ARQUITECTURA DE LA APLICACIÓN

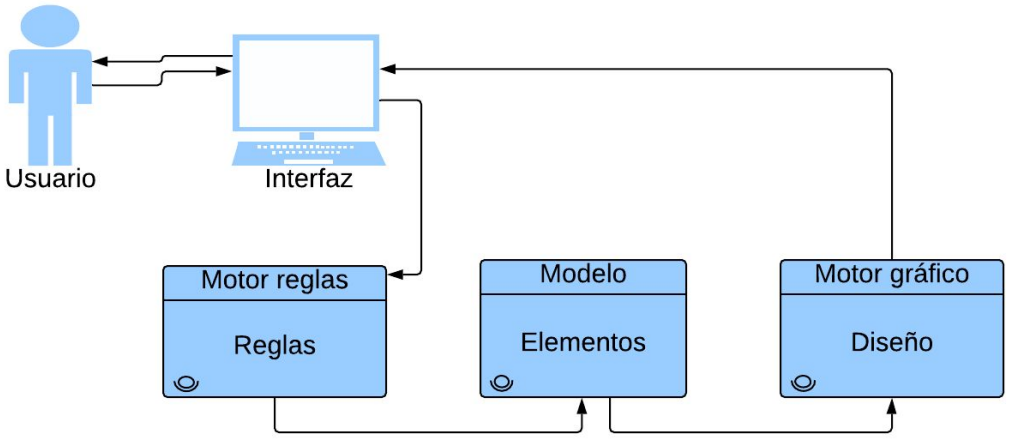
5.1. ARQUITECTURA GLOBAL

La arquitectura de la herramienta está formada por un conjunto de componentes independientes entre sí.

Podemos definir estos componentes como interfaz, motor de reglas, modelo y generador gráfico.

- **Interfaz**: este componente se encarga de interactuar con el usuario, es el encargado de recoger la información proporcionada por el usuario, así como, transmitir sus órdenes al resto de componentes de la herramienta.
- **Motor de reglas**: es el encargado de aplicar las diferentes reglas, dirigiendo la creación del plano en sus diferentes fases.
- **Modelo**: posee la definición de los objetos que constituyen el plano que se va a diseñar. Se encarga de implementar la representación de estos así como de determinar sus propiedades.
- **Generador gráfico**: la herramienta cuenta con un pequeño generador gráfico, el cual es capaz de generar salidas en diferentes formatos, permitiendo exportar los diferentes modelos generados.

Figura 32: Diagrama que muestra el hilo de ejecución de la aplicación



5.2. ARQUITECTURA DE LA INTERFAZ

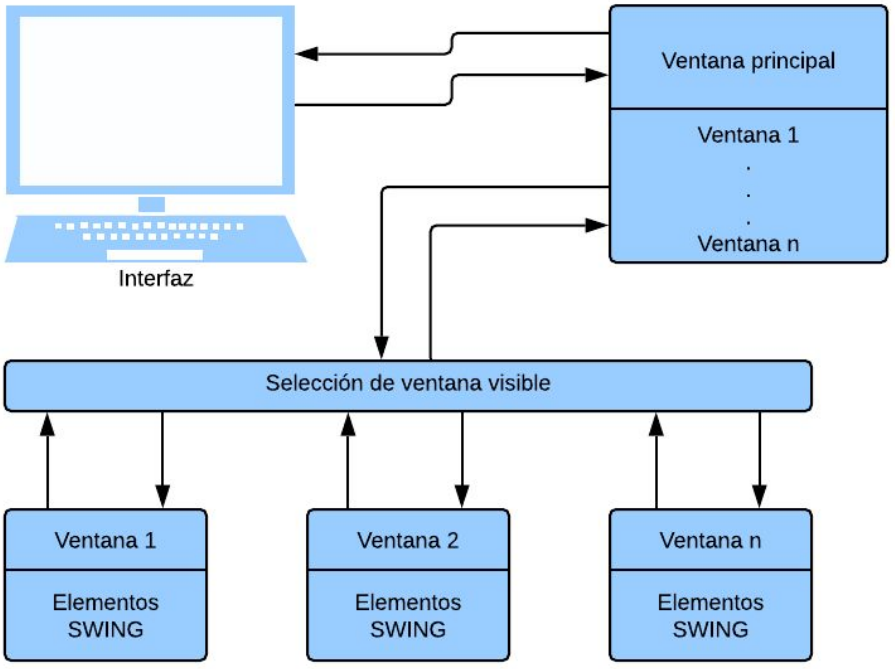
5.2.1. VENTANAS

Para la implementación de la interfaz uno de los aspectos más importantes es que sea lo más dinámica posible, evitando que sea necesario recargar ventanas o abrirlas y cerrarlas.

Para la implementación de la interfaz se empleó SWING, buscando crear una interfaz lo más dinámica posible.

Para conseguir este punto, se ha desarrollado una estructura de ventanas, mediante la cual solo se está visualizando una ventana (VentanaPrincipal), mientras que el resto de ventanas simplemente contienen los objetos que han de ser pintados.

Figura 33: Diagrama de ventanas



La ventana principal, posee definidas e identificadas, mediante un código, las ventanas secundarias.

Estas ventanas secundarias se encargan de definir los elementos SWING que van a ser visualizados, pero establece todos sus elementos en modo no visible y los mantiene identificados mediante claves en un mapa de elementos.

La ventana principal se encargará de notificar a las ventanas secundarias si deben ser visibles o no, ordenando a todas que establezcan su modo no visible a excepción de la única ventana que ha de ser visible.

Mediante este mecanismo conseguimos transiciones completamente fluidas entre ventanas, sin necesidad de cargar una ventana nueva y se proporciona un mecanismo para que desde otro punto de la aplicación, se pueda notificar qué ventana se debe mostrar de una manera sencilla.

Figura 34: Código de ventanas

```
private void crearSubVentanas() {

    Inicio inicio = new Inicio(this);
    subVentanas.put(CodigoVentana.INICIO, inicio);
    inicio.setVisible(true);

    Seleccion seleccion = new Seleccion(this);
    subVentanas.put(CodigoVentana.SELECCION, seleccion);

    Cargando cargando = new Cargando(this);
    subVentanas.put(CodigoVentana.CARGANDO, cargando);

    Mostrar mostrar = new Mostrar(this);
    subVentanas.put(CodigoVentana.MOSTRAR, mostrar);

    SeleccionMostrar seleccionar_mostrar = new SeleccionMostrar(this);
    subVentanas.put(CodigoVentana.SELECCIONAR_MOSTRAR, seleccionar_mostrar);

    SeleccionInicial seleccionar_inicial = new SeleccionInicial(this);
    subVentanas.put(CodigoVentana.SELECCION_INICIAL, seleccionar_inicial);

    MostrarPaso mostrar_paso = new MostrarPaso(this);
    subVentanas.put(CodigoVentana.MOSTRAR_PASO, mostrar_paso);

    PreguntasMedio preguntasMedio = new PreguntasMedio(this);
    subVentanas.put(CodigoVentana.PREGUNTAS_MEDIO, preguntasMedio);

    PreguntasDificil preguntasdificil = new PreguntasDificil(this);
    subVentanas.put(CodigoVentana.PREGUNTAS_DIFICIL, preguntasdificil);

    PreguntasInteligente preguntasInteligente = new PreguntasInteligente(this);
    subVentanas.put(CodigoVentana.PREGUNTAS_INTELIGENTE, preguntasInteligente);
}

public void cambiarVentana(CodigoVentana nuevaVentana) {
    subVentanas.entrySet().forEach(e -> {
        if(e.getKey() == nuevaVentana) {
            e.getValue().setVisible(true);
        }
        else {
            e.getValue().setVisible(false);
        }
    });
}
```

5.2.2. CONTROLADOR

En el apartado de la interfaz se encuentra definido un controlador denominado “ControladorEjecucion”. Este se encarga, como su nombre indica, de controlar la ejecución de la aplicación.

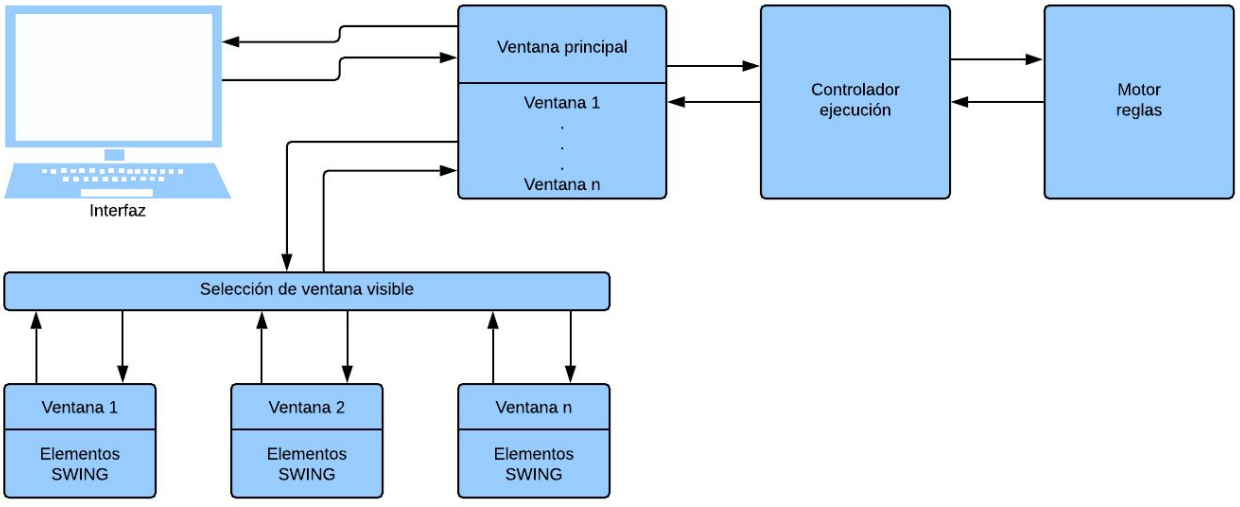
Este objeto se crea usando el patrón de diseño singleton, por el cual solo existe una instancia en la aplicación. Todas las ventanas y el motor de reglas, poseen una referencia a él.

Este se encarga de realizar la comunicación entre la interfaz y el motor de reglas, realizando las acciones necesarias indicadas por cualquiera de las partes.

A través de este controlador, la interfaz es capaz de generar un motor de reglas y ordenar un tipo concreto de ejecución, enviando al motor de reglas todos los datos necesarios para su ejecución, así como, la información introducida por el usuario.

Como se verá en el siguiente apartado, el motor de reglas actúa en un hilo de ejecución independiente. El controlador de ejecución, será el encargado de crearlo, comunicarse con él y finalizarlo de manera correcta.

Figura 35: Esquema controlador



5.3. ARQUITECTURA DEL MOTOR DE REGLAS

La herramienta cuenta con un motor de reglas propio, inicialmente, se planteó la idea de utilizar un motor ya existente, como por ejemplo Drools.

Existen varias opciones como Drools o Jess, las cuales tienen un gran recorrido y documentación, siendo herramientas muy maduras. El problema que se encontró con estas herramientas, fue que no conseguimos encontrar una forma mediante la cual reutilizar gran parte del código para futuras aplicaciones a estructuras. Por esto se decidió implementar un motor de reglas propio, el cual estuviese especializado en el proyecto y que permitiese reutilizar la mayor parte posible de una aplicación a otra.

Se puede dividir el motor en dos partes, las reglas y el motor en sí mismo.

5.3.1. REGLAS

Se ha diseñado una estructura de reglas basada en una interfaz, ReglaBase, la cual intenta acumular todo el trabajo posible común a todas las reglas, para que así se necesario programar una pequeña parte a la hora de diseñar una regla y que la mayor parte del tiempo sea invertido en la creatividad de la misma.

Cada regla, está constituida por una serie de atributos, los cuales podemos identificar como:

- **Predicados edificio**: lista de predicados referentes al edificio, los cuales se encargaran de validar si la regla puede ser aplicada o no, en función del estado en el que se encuentra el edificio sobre el que se va a aplicar.
- **Predicados usuario**: lista de predicados referentes a los datos introducidos por el usuario, los cuales se encargan de determinar

si la regla generaría un resultado que se ajustase a los requerimientos de este.

- **Estado**: determina en qué fase de la ejecución deberá ser aplicada esta regla.
- **Código**: código que permite identificar a las reglas durante la ejecución, para más tarde poder conocer qué reglas son responsables del producto final.

Cada regla tiene una serie de métodos, los cuales permiten desarrollar diferentes acciones:

- **InicializarEstadosPredicados**: este método se encarga de definir cuáles serán los predicados concretos de la regla, así como, determinar a qué estado pertenece. Este método debe ser implementado por las reglas que sean creadas.
- **Test**: este método se encarga de validar las diferentes listas de predicados, para así determinar si la regla puede ser aplicada o debe ser descartada. Este método no es necesario que sea implementado, recupera los predicados que fueron creados por “inicializarEstadosPredicados”;
- **AplicarRegla**: este método se encarga de llevar a cabo las modificaciones sobre el edificio que se le ha proporcionado. Este método debe ser implementado al crear una nueva regla.
- **Copiar**: este método se encarga, mediante serialización, de generar una copia del edificio proporcionado, para que las modificaciones no afecten en caso de descartar la regla (se verá en detalle en la explicación del motor). No debe ser implementada.
- **Execute**: este método se encarga de controlar todo el proceso de la regla. Primero comprueba mediante el método test si esta puede ser aplicada o no, para después aplicar la regla mediante

"aplicarRegla" devolviendo el resultado obtenido. No debe ser implementado.

- **Constructora**: se debe crear una constructora del objeto, en la cual se define el código de la regla y se llama a la constructora de "ReglaBase", la cual se encargará de llamar a "inicializarEstadoPredicados".
- **GetCodigo**: permite acceder al código de la regla. No debe ser implementado.
- **GetEstado**: permite acceder al estado al cual está asociado la regla. No debe ser implementado.

Figura 36: Ejemplo código de reglas

```
public abstract class ReglaBase  implements Serializable {

    private static final long serialVersionUID = -5096679464397378709L;

    protected List<Predicate<Edificio>> predicadosEdificio;
    protected List<Predicate<Map<String,String>>> predicadosUsuario;
    protected Estados estado;
    protected String codigo;

    public ReglaBase(){ 

    public Edificio execute(Edificio edificio,Map<String,String> datosUsuario){ 

    private boolean test(Edificio edificio, Map<String,String> datosUsuario){

    public Estados getEstado(){

    public String getCodigo() {     

    public abstract void inicializarEstadoPredicados();

    public abstract Edificio aplicarRegla(Edificio edificio, Map<String,String> datosUsuario);

    protected Edificio copiar(Edificio b) {

}


public class Abside_1  extends ReglaBase{

    private static final long serialVersionUID = -282717588829729533L;

    public Abside_1 (){

    public void inicializarEstadoPredicados() {     

    public Basilica aplicarRegla(Edificio edificio, Map<String, String> datosUsuario) {

}
```

5.3.2. MOTOR

El motor es el encargado de controlar toda la ejecución de las reglas. Este se encuentra definido en la clase “ControladorReglas”.

Este es independiente de la aplicación, es capaz de trabajar con cualquier conjunto de reglas que extiendan de la clase “ReglaBase” y cualquier representación que extienda de la clase “Edificio”.

Está implementado como un hilo de ejecución independiente. Este será creado y arrancado por la aplicación y se mantendrán en continua comunicación, intercambiando información y órdenes entre ellos.

Para el intercambio de información se intentó usar señales, pero tras leer documentación y realizar pruebas, se observó que muchas de ellas estaban deprecadas y su funcionamiento no era del todo fiable.

En base a esto se buscó documentación para desarrollar un sistema con el cual poder intercambiar órdenes correctamente y controlar tiempos de espera o finalización. Para ello se implementó un mecanismo de espera de parámetros, de tal forma que uno de los dos extremos, se puede mantener a la espera de recibir una orden a través de la consulta periódica de una variable.

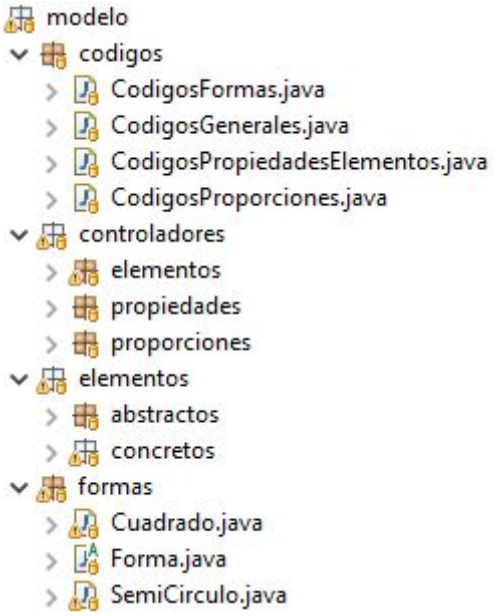
Se entrará más en profundidad acerca de la comunicación y el funcionamiento del algoritmo implementado en forma de árbol de decisión en el siguiente punto (6. MOTOR DE REGLAS)

5.4. ARQUITECTURA DEL MODELO

Para la creación de la arquitectura del modelo, se decidió dividir los diferentes elementos arquitectónicos de la basílica en niveles. De este modo, fue creada una clase abstracta por nivel a partir de la cual extendieron las clases pertenecientes a ese nivel.

Cada nivel es un conjunto de elementos arquitectónicos, cuanto más elevado sea el nivel significa que los elementos que contiene son más generales y al contrario con los niveles menores.

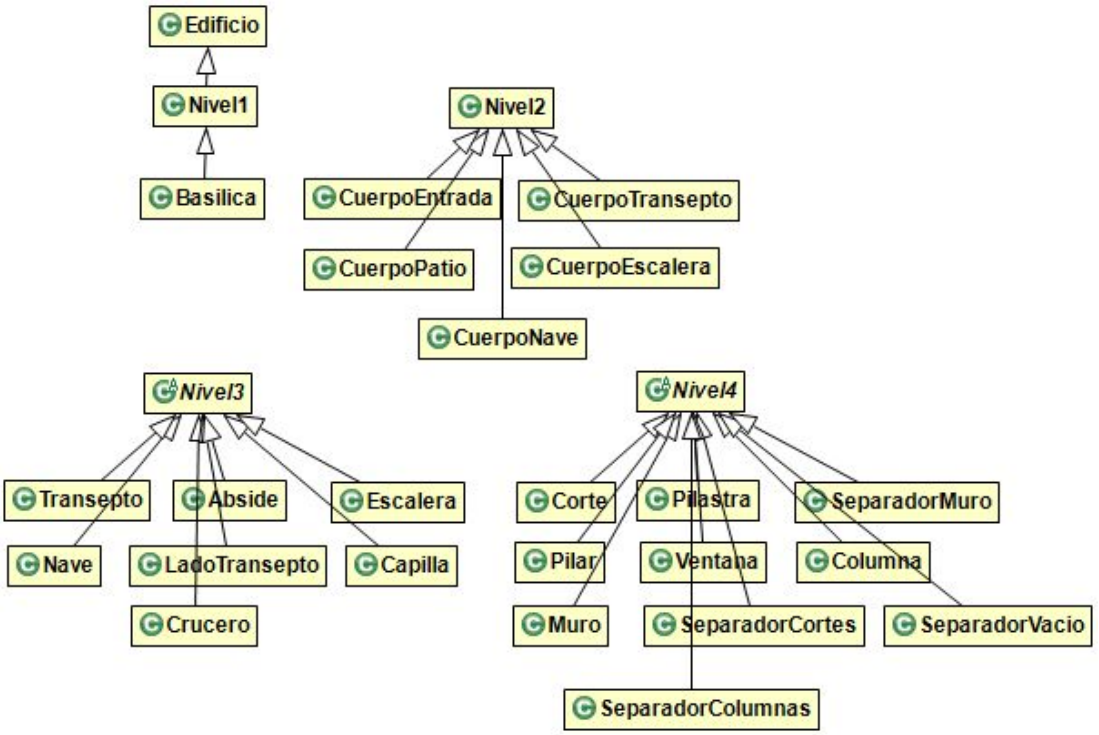
Figura 37: Modelo de objetos



En el caso de realizar una nueva aplicación, el modelo debe extender de la clase Edificio e implementar su método paint, el cual devolverá un objeto G2D con el contenido del plano.

En el siguiente diagrama se muestran los principales elementos arquitectónicos del modelo creados para el diseño del plano de la basílica cristiana.

Figura 38: Diagrama UML



6. MOTOR DE REGLAS

6.1. INTRODUCCIÓN

En este apartado se van a describir los puntos más importantes del motor de reglas.

El funcionamiento del motor de reglas está basado en un algoritmo de árbol de decisión.

Este está implementado mediante un método recursivo, el cual recibe como parámetros el edificio que se está construyendo, el estado en el que se va a ejecutar y las reglas que han sido ejecutadas en los estados anteriores.

Figura 39: Código función ejecutar estado

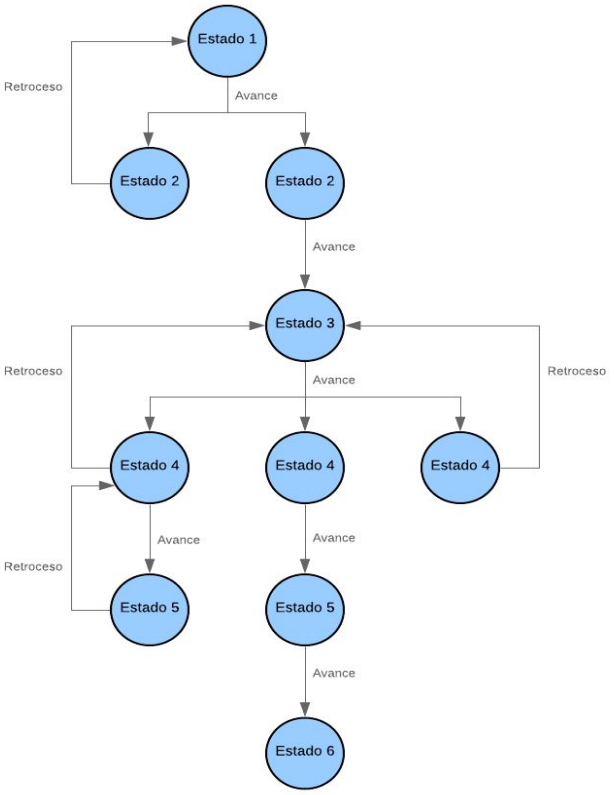
```
private Estados ejecutarEstado(Edificio edificio, Estados estadoActual, Map<Estados,String> reglasEjecutadas){
    if(finalizarEjecucion) {
        return Estados.FINALIZAR_EJECUCION;
    }

    if(estadoActual == Estados.TERMINADA){
        return resolverEstadoFinal(edificio, reglasEjecutadas);
    }

    List<ReglaBase> reglasEstado = new ArrayList<ReglaBase>();

    ((List<ReglaBase>)reglas.get(estadoActual)).forEach(e -> { reglasEstado.add(e); });

    while(reglasEstado.size() > 0 && !finalizarEjecucion){

        ReglaBase regla = null;

        if(controladorEjecucion.getTipoEjecucion() == TipoEjecucion.CARGA) {
            try {
                regla = reglasEstado.stream().filter(r -> r.getCodigo().equalsIgnoreCase(
                        controladorEjecucion.getCodigoReglaEstado(estadoActual))).findAny().get();
                if(esFinCarga(Estados.values()[estadoActual.ordinal()+1],
                            (List<ReglaBase>)reglas.get(Estados.values()[estadoActual.ordinal()+1]))) {
                    controladorEjecucion.setVentanaEjecucion(CodigoVentana.SELECCION_INICIAL);
                    controladorEjecucion.finDeCarga();
                }
            }catch(Exception e) {}
        }
        else {
            int reglaAleatoria = (int) Math.floor(Math.random()*((reglasEstado.size()-1)-0+1)+0);
            regla = reglasEstado.get(reglaAleatoria);
        }

        if(regla != null) {
            reglasEstado.remove(regla);
            Edificio edificioGenerado = regla.execute(edificio, datosUsuario);

            if(edificioGenerado != null){

                estadoUsuario = null;
                ResourcesUtils.getResourcesUtils().setEdificio(edificioGenerado);

                EstadoPaso pasoAceptado = EstadoPaso.APROBADO;

                HashMap<Estados, String> reglasEjecutadasIteracion = new HashMap<Estados,String>(reglasEjecutadas);
                reglasEjecutadasIteracion.put(estadoActual, regla.getCodigo());

                if(controladorEjecucion.getTipoEjecucion() == TipoEjecucion.PASOS) {
                    controladorEjecucion.setReglasEjecutadasPaso(reglasEjecutadasIteracion);
                    controladorEjecucion.cambiarVentana(CodigoVentana.MOSTRAR_PASO);
                    pasoAceptado = resolverPaso();
                }

                if(pasoAceptado == EstadoPaso.APROBADO) {
                    Estados estadoGenerado = ejecutarEstado(edificioGenerado,
                            Estados.values()[estadoActual.ordinal()+1], reglasEjecutadasIteracion);

                    if(estadoGenerado != estadoActual){
                        return estadoGenerado;
                    }
                }
                else if(retroceder) {
                    retroceder = false;
                    return Estados.values()[estadoActual.ordinal()-1];
                }

            }
        }
    }
    return Estados.values()[estadoActual.ordinal()-1];
}
```

6.2. FUNCIONAMIENTO GENERAL

El método *ejecutarEstado*, se encarga de obtener la lista de reglas disponibles a ejecutar en esa fase de la ejecución. Una vez obtenidas va seleccionando reglas aleatoriamente e intentando aplicarlas. Cuando ha conseguido aplicar una regla, se produce una llamada recursiva enviando el nuevo edificio generado, el siguiente estado y la lista de reglas actualizada. Este esperará como respuesta de la llamada recursiva un estado. Si el estado recibido es el actual, indica que debe seleccionar otra regla para crear una nueva rama, en caso contrario retorna el estado recibido.

En caso de que un estado no pueda aplicar ninguna regla (bloqueo), realiza un retroceso, indicando al estado anterior (el cual le ha llamado) que el árbol no pude continuar expandiéndose a través de esa rama, por lo que el estado anterior selecciona otra regla que pueda aplicar e intenta continuar el desarrollo por la nueva rama. Este proceso se realiza retornando el estado anterior al actual.

Figura 40: Esquema de estados

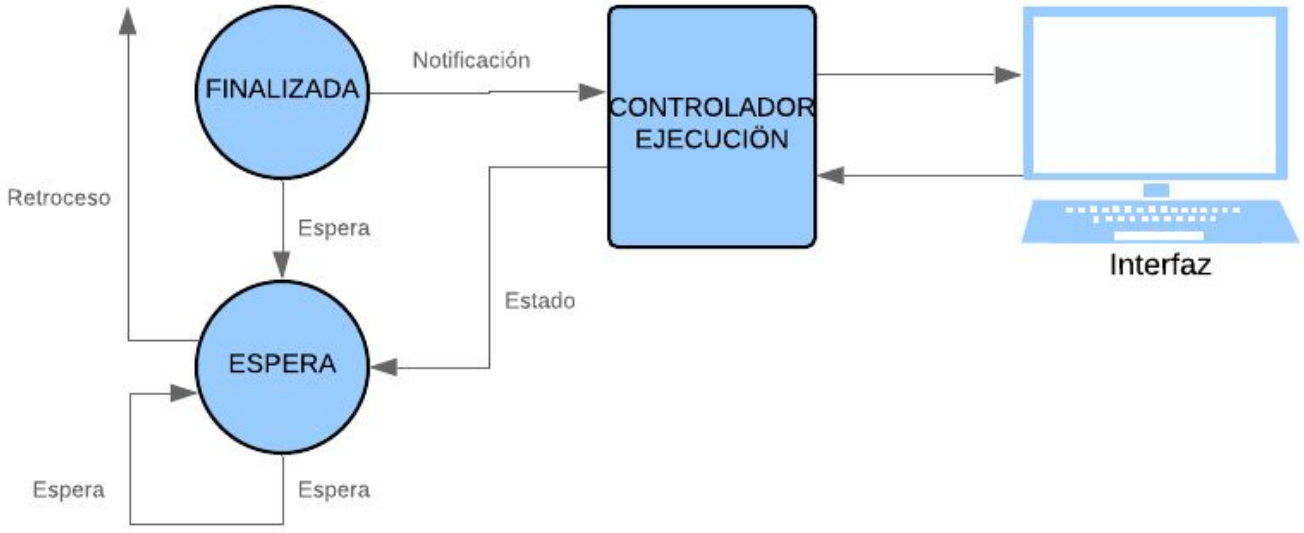


6.3. RESOLVER ESTADO FINAL

Una vez alcanzado el estado *FINALIZADA*, se procede a resolver el estado final. En este punto se notificará al controlador de ejecución que se ha generado un producto final y el motor de reglas pasará a un estado de espera, en el que consulta periódicamente una variable de tipo Estado, para realizar un retroceso en función de su valor. Pueden darse dos casos:

- ***FINALIZAR_EJECUCION***: en caso de recibir el estado *FINALIZAR_EJECUCION*, la ejecución ha terminado, el usuario no va a continuar trabajando con este diseño y se procede a finalizar este hilo de ejecución.
- **Estado**: en caso de recibir un estado diferente a finalizar ejecución, se debe propagar el estado recibido realizando un retroceso. Esto se produce cuando el usuario selecciona la opción de retroceso, para ir retrocediendo paso a paso (se recibiría el último estado de la ejecución) o bien cuando se lleva a cabo la mejora inteligente (se recibirá el estado, que ha determinado la mejora inteligente, a partir del cual se debe rehacer el diseño).

Figura 41: Resolver estado final



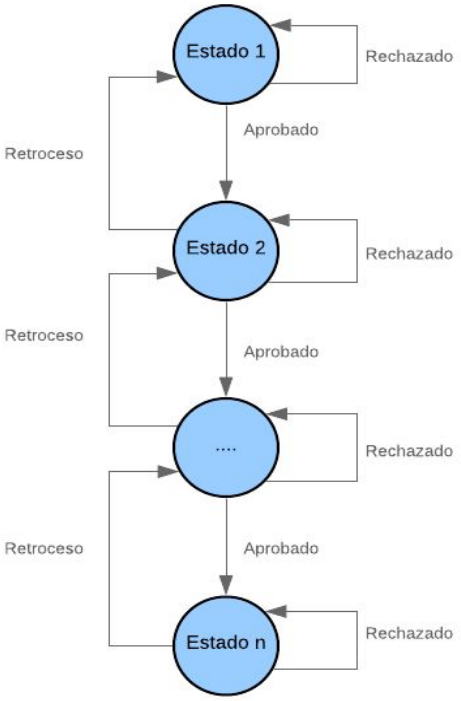
6.4. EJECUCIÓN POR PASOS

Hemos visto como es el funcionamiento general de una ejecución, pero el modo de ejecución por pasos tiene una peculiaridad.

En este modo de ejecución cada vez que el motor resuelve un estado, realiza una notificación al controlador de ejecución y pasa a modo espera, esperando uno de los siguientes resultados:

- **Aprobado**: si recibe una notificación de que el paso ha sido aprobado, realiza un llamada al siguiente nivel como sucedería en una ejecución normal.
- **Rechazado**: indica que el paso ha sido rechazado y por lo tanto la regla ejecutada no satisface al usuario. En este punto se descarta la regla y se selecciona otra de las disponibles.
- **Retroceso**: si se recibe una orden de retroceso, significa que el usuario se ha dado cuenta de que desea modificar un paso del nivel anterior. Se realizaria un retroceso tal y como hemos visto anteriormente, retornando el nivel anterior.

Figura 42: Esquema de estados



6.5. CARGAR Y ERROR DE AVANCE

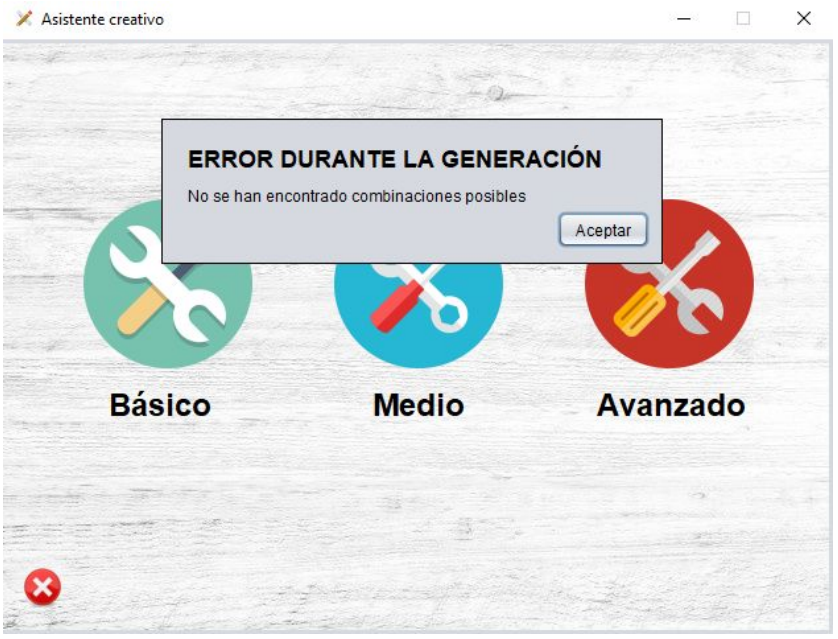
En el caso de realizar una carga, que haya sido exportada previamente, se arrancará el motor de reglas indicando que se trata de una carga. Este recibirá de que tipo de ejecución se trataba, los datos de usuario que estaban introducidos en el momento de la exportación, así como que regla fué ejecutada en cada estado.

Este comprobará en cada estado si posee una regla que fue ejecutada en el, en caso afirmativo la ejecutará y saltará al siguiente estado. En caso de que no exista una regla que fue ejecutada en ese estado indica que a partir de ese punto a finalizado la carga y debe continuar de manera normal.

Puede darse el caso, que debido a la configuración introducida por el usuario o al excesivo descarte, se llegue a un punto en el que sea inviable continuar la ejecución por falta de reglas que se puedan ejecutar en alguno de los estados.

LLegados a este punto el motor detectara que hay un error de avance y finalizará su ejecución informando al usuario de que no existían combinaciones posibles, por lo que el usuario debe modificar los filtros que ha introducido.

Figura 43: Imágen cargar error



7. IMPLEMENTACIÓN

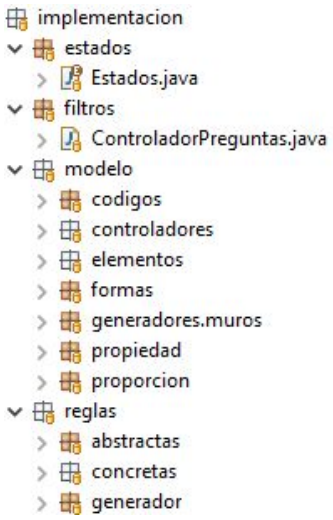
Uno de los principales objetivos de este proyecto, es diseñar una herramienta con la cual fuese posible desarrollar distintos tipos de plano. En este caso se ha desarrollado una aplicación para desarrollar planos de basílicas, pero podría ser empleada para desarrollar el diseño de planos de otro tipo de estructura arquitectónica..

En la estructura del proyecto, encontramos que se encuentra dividida en dos paquetes, “implementacion” y “agente_creativo”.

El paquete implementación contiene el modelo y las reglas específicas para crear el diseño del plano, en este caso para el diseño de basílicas cristianas. Si se quiere diseñar otro tipo de plano, sería necesario crear el modelo de la estructura arquitectónica a diseñar, sus estados y sus reglas, por tanto, todos los fuentes que deben ser modificados para la creación de otro plano se encuentran en “implementacion”.

El paquete “agente_creativo” contiene el resto del código de la aplicación, interfaz, motor de reglas y generador gráfico.

Figura 44: Paquete implementación



A continuación se van a enumerar y explicar los cambios que serían necesarios para poder llevar a cabo un elemento arquitectónico diferente de la basílica cristiana.

7.1. ESTADOS

Para el correcto desarrollo de la ejecución, están definidos una serie de estados o fases, gracias a los cuales se establecen las diferentes fases en las que se construye un edificio.

Las reglas que se ejecutarán están asociadas a un estado concreto para así determinar en qué momento de la ejecución han de utilizarse.

A la hora de realizar una nueva aplicación, se deben reemplazar estos estados por los de la propia aplicación.

Los únicos estados que se deben mantener obligatoriamente en las posiciones que se encuentran son “FINALIZAR_EJECUCION”, “ERROR_AVANCE”, “TERMINADA” y “RELANZAR”, ya que estos estados son usados internamente para detectar determinadas circunstancias.

No existe ningún límite de estados, se pueden definir tantos como se consideren oportunos o sean necesarios, siempre y cuando se respete el punto anterior.

Figura 45: Código de estados

```
public enum Estados {

    FINALIZAR_EJECUCION, ERROR_AVANCE,

    CRUCERO,
    CUERPO_TRANSEPTO,
    CAPILLAS,
    ABSIDE,
    CUERPO_NAVE,
    CUERPO_PATIO,
    ENTRADA,
    ESCALERAS,

    TERMINADA,RELANZAR

}
```

7.2. FILTROS

Uno de los puntos claves del proyecto es el descubrimiento de los requisitos o gustos del usuario.

Para recabar información acerca de estos, se realizan una serie de preguntas al usuario, las cuales determinan las decisiones que se tomarán a la hora de aplicar las reglas y desarrollar las fases de la ejecución.

La definición de estas preguntas se encuentra en la clase “ControladorPreguntas.java”. En este podemos encontrar diferentes Map en los cuales se encuentran definidas, mediante clave - valor, las diferentes preguntas.

- **preguntasSimples**: se trata del mapa que aloja las preguntas del apartado medio. Es necesario rellenar este mapa con las preguntas y respuestas que se desean realizar en el método “cargarPreguntas”.

Figura 46: Código de preguntas simples

```
// -- PREGUNTAS SIMPLES
List<String> listaPreguntasSimples2 = new ArrayList<String>();
listaPreguntasSimples2.add(CodigosPreguntas.DIMESIONES_ALARGADA);
listaPreguntasSimples2.add(CodigosPreguntas.DIMESIONES_NORMAL);
listaPreguntasSimples2.add(CodigosPreguntas.DIMESIONES_ANCHA);
preguntasSimples.put(CodigosPreguntas.DIMESIONES, listaPreguntasSimples2);

List<String> listaPreguntasSimples = new ArrayList<String>();
listaPreguntasSimples.add(CodigosPreguntas.NAVES_1);
listaPreguntasSimples.add(CodigosPreguntas.NAVES_3);
listaPreguntasSimples.add(CodigosPreguntas.NAVES_5);
preguntasSimples.put(CodigosPreguntas.NAVES,listaPreguntasSimples);
```

- **preguntasComplejas**: se trata del mapa que aloja las preguntas del apartado avanzado. Es necesario rellenar este mapa con las preguntas y respuestas que se desean realizar en el método “cargarPreguntas”.

```
// -- PREGUNTAS COMPLEJAS

List<String> listaPreguntasComplejas2 = new ArrayList<String>();
listaPreguntasComplejas2.add(CodigosPreguntas.PATIOS_1);
listaPreguntasComplejas2.add(CodigosPreguntas.PATIOS_2);
listaPreguntasComplejas2.add(CodigosPreguntas.PATIOS_3);
preguntasComplejas.put(CodigosPreguntas.PATIOS, listaPreguntasComplejas2);

List<String> listaPreguntasComplejas3= new ArrayList<String>();
listaPreguntasComplejas3.add(CodigosPreguntas.MUROS_ABIERTOS);
listaPreguntasComplejas3.add(CodigosPreguntas.MUROS_CERRADOS);
preguntasComplejas.put(CodigosPreguntas.MUROS, listaPreguntasComplejas3);
```

Figura 47: Código de preguntas complejas

- **preguntasInteligentes y estadosInteligentes**: se trata del mapa que aloja las preguntas del apartado mejora inteligente. En este caso "preguntasInteligentes" contiene las preguntas y respuestas de este apartado, mientras que "estadosInteligentes", contiene el estado asociado a cada respuesta, para así determinar que parte de la ejecución es válida y cual no se ajusta a los requerimientos del usuario.

  Es necesario rellenar este mapa con las preguntas y respuestas que se desean realizar en el método "cargarPreguntas".

Figura 48: Código de preguntas inteligentes

```
// -- PREGUNTAS INTELIGENTES

List<String> listaPreguntasInteligentes2 = new ArrayList<String>();
listaPreguntasInteligentes2.add(CodigosPreguntas.OPRESORA_SI);
estadosInteligentes.put(CodigosPreguntas.OPRESORA_SI, Estados.CRUCERO);
listaPreguntasInteligentes2.add(CodigosPreguntas.OPRESORA_NO);
estadosInteligentes.put(CodigosPreguntas.OPRESORA_NO, Estados.ENTRADA);
preguntasInteligentes.put(CodigosPreguntas.OPRESORA, listaPreguntasInteligentes2);

List<String> listaPreguntasInteligentes = new ArrayList<String>();
listaPreguntasInteligentes.add(CodigosPreguntas.LUMINOSA_SI);
estadosInteligentes.put(CodigosPreguntas.LUMINOSA_SI, Estados.CUERPO_NAVE);
listaPreguntasInteligentes.add(CodigosPreguntas.LUMINOSA_NO);
estadosInteligentes.put(CodigosPreguntas.LUMINOSA_NO, Estados.ENTRADA);
preguntasInteligentes.put(CodigosPreguntas.LUMINOSA, listaPreguntasInteligentes);

List<String> listaPreguntasInteligentes3 = new ArrayList<String>();
listaPreguntasInteligentes3.add(CodigosPreguntas.OPRESORA_SI);
estadosInteligentes.put(CodigosPreguntas.SOLEDAD_SI, Estados.CRUCERO);
listaPreguntasInteligentes3.add(CodigosPreguntas.OPRESORA_NO);
estadosInteligentes.put(CodigosPreguntas.SOLEDAD_NO, Estados.ENTRADA);
preguntasInteligentes.put(CodigosPreguntas.SOLEDAD, listaPreguntasInteligentes3);
```

7.3. REGLAS

Todas las reglas, tienen que extender de la clase abstracta ReglaBase e implementar los métodos “inicializarEstadoPredicados” y “aplicarRegla”.

El método *inicializarEstadoPredicados* implementado en cada regla, tiene en cuenta las respuestas introducidas por el usuario y la composición actual del edificio, será el encargado de decidir si la regla se puede ejecutar o no.

El método *aplicarRegla* creará los elementos arquitectónicos necesarios en dicha regla.

Figura 49: Código de la clase *ReglaBase*

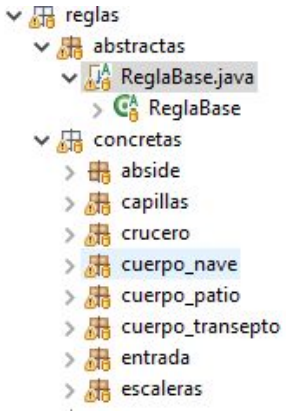
```
public abstract class ReglaBase  implements Serializable {

    private static final long serialVersionUID = -5096679464397378709L;

    protected List<Predicate<Edificio>> predicadosEdificio;
    protected List<Predicate<Map<String,String>>> predicadosUsuario;
    protected Estados estado;
    protected String codigo;

    public ReglaBase(){ ..

    public Edificio execute(Edificio edificio,Map<String,String> datosUsuario){ ..

    private boolean test(Edificio edificio, Map<String,String> datosUsuario){..

    public Estados getEstado(){..

    public String getCodigo() {     ..

    public abstract void inicializarEstadoPredicados();

    public abstract Edificio aplicarRegla(Edificio edificio, Map<String,String> datosUsuario);

    protected Edificio copiar(Edificio b) {..

}
```

A continuación se procede a explicar cómo está estructurado el ejemplo que encontrará el usuario, a modo de guia.

Para una mejor organización, debido a la relación entre los estados y las reglas, se decidió crear un paquete por cada estado y este contendrá todas las reglas aplicables al estado al que hace referencia dicho paquete.

Para una mejor reutilización de código, cada paquete contiene una regla abstracta encargada de implementar los métodos *inicializarEstadoPredicados* y *aplicarRegla.* Por tanto las demás reglas, sólo tienen que extender la regla abstracta mencionada anteriormente inicializando sus atributos según sea conveniente en cada regla.

Figura 50: Paquete reglas



En las siguientes imágenes se muestra un ejemplo con las reglas del estado Capillas.

Figura 51: Código de reglas

```
public abstract class ReglaCapillasAbstracta extends ReglaBase{

    private static final long serialVersionUID = -5895253040401514183L;

    protected Proporcion proporcion;
    protected int lado;
    protected Integer separacion;

    public ReglaCapillasAbstracta(){
        super();
        estado = Estados.CAPILLAS;
    }

    public Basilica aplicarRegla(Edificio edificio, Map<String, String> datosUsuario) {

    public  void inicializarEstadoPredicados() {

}

public class CapillaPropDosTres1Columnas extends ReglaCapillasAbstracta{

    private static final long serialVersionUID = 1L;

    public CapillaPropDosTres1Columnas() {
        super();
        codigo = "Capillas_2/3_1";
        proporcion = Proporcion.TRES_DOS;
        separacion = 7;
    }

}
```

7.4. MODELO

A continuación se procede a explicar cómo está estructurado el ejemplo de modelo que encontrará el usuario, a modo de guia.

Como se ha mencionado anteriormente, la arquitectura del modelo está dividida en niveles.

```
public class Nivel2 implements Serializable {

    private static final long serialVersionUID = -2151246373176235338L;

    protected List<Nivel3> elements  = new ArrayList<>();
    protected UUID id;
    protected Forma forma;
    protected String tipo;
    protected ControladorElementos elementosController;
    protected int anchuraMuros;
    protected Proporcion proporcion;
```

Figura 52: Código de la clase Nivel 2

Cada nivel, tiene una lista de elementos del nivel posterior.

Los elementos pertenecientes a los niveles superiores, son elementos generales que están formados por la unión de elementos del siguiente nivel.

Se continúa con esta premisa, hasta llegar al último nivel, en el cual los objetos pertenecientes a este nivel, son lo suficientemente simples como para poder dibujarse directamente.

Los elementos del último nivel son dibujados utilizando los objetos proporcionados por la clase Graphics2D

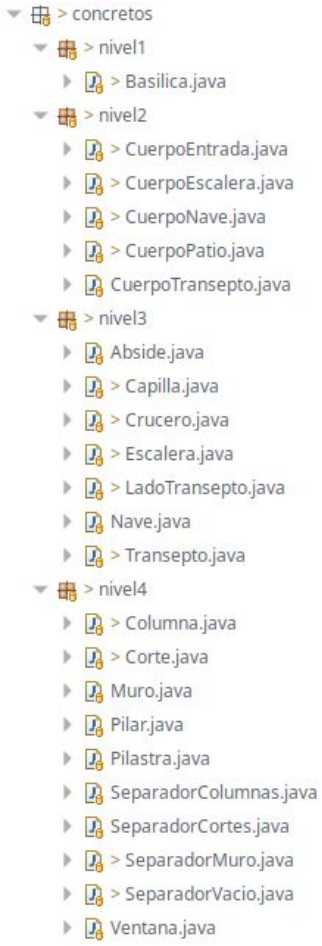


Figura 53: Elementos concretos

Por ejemplo, el objeto de nivel 2 “CuerpoNave” está compuesto entre otros. por la unión de los objetos Nave, pertenecientes al nivel 3 y estas a su vez están compuestas por muros, columnas…, estos últimos objetos pertenecen al nivel 4, por tanto sólo contienen los objetos de la clase Graphics2D necesarios para ser dibujados, como se ve en el ejemplo del objeto columna en la siguiente imagen.

Figura 54: Código de la clase Columna

```
public class Columna extends Nivel4{

    private static final long serialVersionUID = 1L;
    private Rectangle2D rectangulo;
    private Ellipse2D circulo;
    private double lado;

    public Columna(Point p1,double lado){

    public void paint(Graphics2D graphics) {

    public double getLado() {

    public void setLado(double lado) {

}
```

8. CONCLUSIONES

En el desarrollo de este proyecto se ha llevado a cabo una herramienta que intenta satisfacer algunas de las necesidades de personas interesadas en el mundo de la arquitectura y el estudio de la creatividad.

Se ha comprendido la importancia de la información proporcionada por el usuario, a la hora de ofrecerle un resultado, ya que la omisión de resultados que no cumplen sus expectativas, captará su atención y le incitara a continuar utilizando la herramienta.

Durante el desarrollo de este proyecto, se han puesto en práctica muchos de los conocimientos aprendidos durante la carrera y se ha aprendido la importancia de realizar una buena planificación del trabajo, organización de las tareas y ser realistas con los plazos de entrega de versiones. Ha sido necesario programar una serie de iteraciones para obtener diferentes productos intermedios, rectificando y avanzando en función de los objetivos conseguidos.

En lo referente a la creatividad, hemos observado que el sistema ayuda a estimular la creatividad ya que al aportar un gran número de planos, ayuda al usuario que interactúa con el sistema a aportar otros puntos de vista y de este modo conseguir estimular la creatividad del usuario para que sea capaz de desarrollar una solución creativa.

8. CONCLUSIONS

In the development of this project, the tool tries to satisfy some of the needs of people interested in the world of architecture and the study of creativity.

You understood the importance of the information provided by the user when offering a result, since the omission of results that do not meet your expectations, will capture your attention and encourage you to continue using the tool.

During the development of this project, many of the knowledge learned during the race has been put into practice and the importance of good work planning, tasks organization and being realistic with the delivery deadlines has been learned. It has been necessary to program a series of iterations to obtain different intermediate products, rectifying and advancing according to the objectives achieved.

Regard to creativity, we have observed that the system helps out to stimulate creativity by providing a large number of plans, it helps the user who interacts with the system to contribute other points of view and be able to stimulate the creativity of the user to be able to create a creative solution.

9. TRABAJO FUTURO

Como trabajo futuro, una de las principales ideas que tiene este proyecto, es que se lleven a cabo múltiples aplicaciones sobre este, desarrollando planos más complejos y enfocados a las construcciones actuales.

Aunque este proyecto a cumplido sus objetivos iniciales, hay varios puntos que se podrían mejorar en un futuro:

- **Visualización de planos**: sería interesante que la herramienta contase con un visor que permita ver con más detalle los planos. Actualmente estos son mostrados directamente en la aplicación, pero no permiten ser ampliados para apreciar el detalle, para esto es necesario exportarlo y visualizar el SVG. Por lo que aportaría gran valor el contar con un visor de mayor calidad integrado.

- **Mejora inteligente**: profundizar más en el apartado de mejora inteligente, sería otro punto de cara a futuro, recabando mayor información y empleando esta para sorprender al usuario.

- **Archivo de reglas:** actualmente las reglas están escritas directamente en el código, para que la modificación de estas fuese más sencilla, deberían estar en un archivo a parte y que dicho archivo sea leído y parseado.

- **Valoración de planos**: sería interesante añadir un sistema que almacene los gustos del usuario en función de su aprobación o descarte en el diseño por pasos, para así otorgar mayor o menor peso a unas reglas u otras.

- **Creatividad en reglas**: uno de los puntos más importantes de cara a un trabajo futuro, es diseñar reglas con un alto grado creativo. Debido a los escasos conocimientos de creatividad en la arquitectura, las reglas diseñadas para este prototipo son poco creativas y la creatividad tan solo proviene de la combinación de estas.

- **Control de versiones**: sería muy interesante para el usuario tener un control de versiones para los diseños generados. Para así poder retroceder a una versión anterior de un diseño guardado.

9. FUTURE WORK

As future work, one of the main ideas this project has is that multiple applications are shown, developing more complex plans and focused on current constructions.

Although this project has fulfilled its initial objectives, there are several points which could be improved in the future:

- **Visualization of plans**: it would be interesting if the tool had an interface that allows us to see more detailed plans. Currently, these are shown directly in the application but does not extend to appreciate the detail, that is why it is necessary to export it and view the SVG. So it would be of great value to have a viewer of higher quality integrated.

- **Smart improvement**: diving deeper into the smart improvement section, it would be another point for the future, gathering more information and using it to please the user.

- **Rule file**: currently the rules are written directly in the code so the modification was easier, they should be in a separate file and this one read and parsed.

- **Assessment of plans**: it would be interesting to add a system which stores the user's tastes based on their approval or discarding in the design in steps, in order to give more or less weight to some rules or others.

- **Creativity in rules**: one of the most important points for future work is to design rules with a high creative degree. Due to the limited knowledge of creativity in architecture, the rules designed for this prototype are not very creative and creativity only comes from the combination of these.

- **Version control**: it would be very interesting for the user to have version control for the generated designs. In order to go back to an earlier version of a saved design.

10. DISTRIBUCIÓN DE ESFUERZO

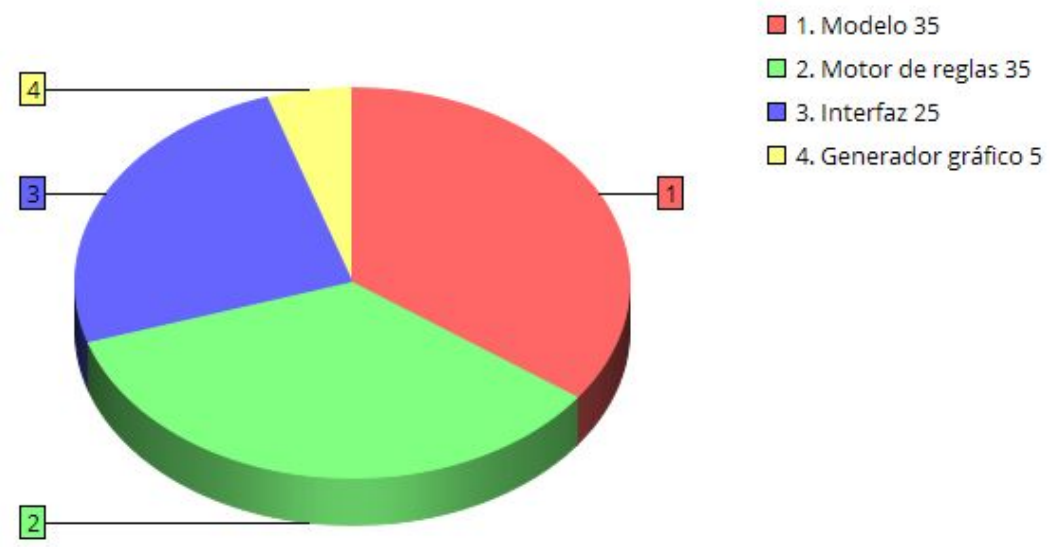


Figura 55: Distribución del esfuerzo

10.1 INTERFAZ

Fue invertido mucho tiempo en el estudio de la biblioteca gráfica swing, con la cual se diseñó la interfaz.

Gran parte del tiempo fue invertido en la búsqueda de un mecanismo para aportar la mayor fluidez posible a la interacción con el usuario.

Se probaron varias opciones hasta alcanzar una arquitectura con la que el grado de fluidez de la interacción era el deseado.

10.2 MOTOR DE REGLAS

La mayor parte del tiempo, fue invertido en el diseño del algoritmo encargado de llevar a cabo las fases y la ejecución de las reglas.

Además de ello, cabe destacar el tiempo invertido en controlar la ejecución de este algoritmo en un hilo de ejecución separado.

10.3 GENERADOR GRÁFICO

Como muestra el diagrama, en el generador gráfico fue donde menos tiempo fue invertido.

Para su creación fue necesario el aprendizaje de la librería BATIK la cual permite, a partir de un objeto Graphics2D crear una imagen de tipo SVG y PNG.

10.4 MODELO

Una gran parte del esfuerzo, fue destinado al desarrollo del modelo, debido a nuestros escasos conocimientos de arquitectura y más concretamente sobre basílicas.

Antes y durante el desarrollo del modelo, tuvimos que adquirir muchos conocimientos sobre cómo se crean los elementos arquitectónicos, como se unen entre sí, sus dimensiones, proporción…

Además cabe mencionar el tiempo invertido en el estudio de la librería Graphics2D con la cual se dibujan los planos.

11. APORTACIONES INDIVIDUALES

En el desarrollo de este proyecto, el peso del trabajo ha sido repartido de forma equitativa, ambos integrantes del equipo han trabajado en todas las partes del proyecto.

Desde un principio se han definido una serie de iteraciones con unos objetivos marcados, distribuyendo las tareas en entre ambos participantes y realizando rectificaciones en estas cuando alguna tarea era más compleja de lo esperado.

Aun así, cada uno de los integrantes a aportado un poco más de peso en alguna de las partes del proyecto.

Para llevar a cabo una valoración del trabajo, se va a dividir el proyecto en cuatro partes: motor de reglas, interfaz, modelo y creación de reglas, indicando simplemente en que partes ha existido una mayor contribución de un integrante.

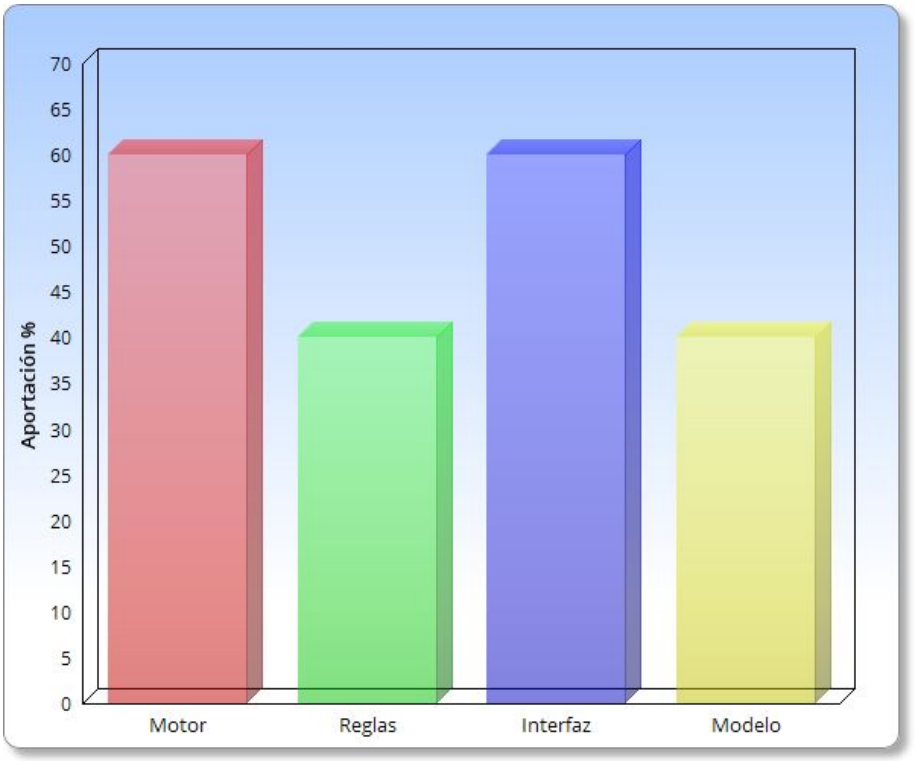
11.1. DAVID LOSILLA CADENAS

Como resumen de la contribución del integrante David Losilla, podemos destacar el desarrollo del motor de reglas, diseñado su estructura para definir su comportamiento.

También podemos destacar el diseño de la arquitectura de la interfaz, estableciendo el flujo de ventanas, la interacción con el usuario y la comunicación con el resto de componentes de la herramienta.

Por último podríamos destacar, el diseño de interfaces para buscar la menor dependencia entre aplicaciones, para un posible trabajo futuro sobre el proyecto.

Figura 56: Diagrama de trabajo individual David Losilla

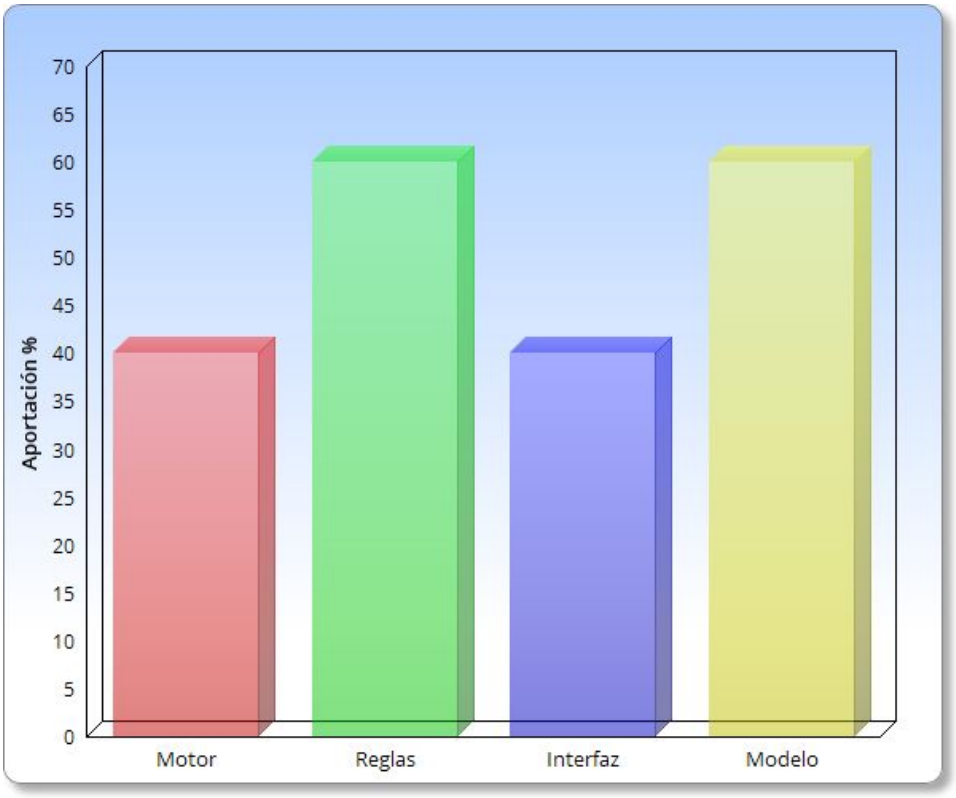


11.2. ROBERTO FERNÁNDEZ CORREA

Como resumen de la contribución del integrante Roberto Fernández, se puede mencionar la creación de la arquitectura de las reglas y del modelo.

Además cabe destacar el estudio de la clase Graphics2D usada para dibujar los elementos arquitectónicos de la basílica y el estudio de la librería Batik, que permite exportar el diseño creado con Graphics2D a formatos estándar como SVG y PNG.

Figura 57: Diagrama de trabajo individual Roberto Fernández



12. ACCESO Y USO DEL PROYECTO

Este proyecto ha sido desarrollado como una herramienta de código abierto. Se invita a cualquier persona que lo desee, a realizar sus propias aplicaciones o introducir las mejoras que pueda aportar al proyecto.

El único requisito que se solicita, es que se compartan los desarrollos realizados sobre este. Todas las mejoras o aplicaciones han de ser subidas al repositorio para entre todos conseguir una herramienta gratuita de calidad.

Este proyecto se encuentra en github, alojado como un repositorio público en la siguiente dirección:

https://github.com/PROTOTIPO-SIMULACION-CREATIVIDAD/PROTOTIPO

12. ACCESS AND USE OF THE PROJECT

This project has been developed as an open-source tool. Any person who wishes to do so is invited to make their own applications or introduce the improvements they can bring to the project.

The only requirement that is requested is that the developments made on it be shared. All improvements or applications have to be uploaded to the repository to get a free quality tool together.

This project is in GitHub, hosted as a public repository at the following address:

https://github.com/PROTOTIPO-SIMULACION-CREATIVIDAD/PROTOTIPO

BIBLIOGRAFÍA

- **Manual GIT**
  https://git-scm.com
- **Manual Java 8**
  https://www.java.com
- **Batik SVG**
  https://xmlgraphics.apache.org/batik/using/svg-generator.html
- **Api de Graphics2D**
  https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html
- **Java swing**
  https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html
- **Serialización de objetos**
  https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html
- **Documentacion de Autocad**
  https://www.autodesk.es/products/autocad/overview

  https://latinoamerica.autodesk.com/products/autocad-360/features/all
- **Diferentes artículos que investigan la creatividad humana**

  https://elcultural.com/Los-origenes-de-la-creatividad-humana

  https://es.wikipedia.org/wiki/Creatividad

  https://www.neuronilla.com/tecnicas-de-creatividad-carlos-churba/

  https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/166/mod_resource/content/1/la-creatividad/index.html

  https://www.xataka.com/robotica-e-ia/sale-a-subasta-primera-vez-obra-arte-generada-algoritmo