Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Herramienta para el estudio de la propagación de ondas de radio en interior

Estudante: Diego González Morandeira
Dirección: Óscar Fresnedo Arias
José P. González Coma

A Coruña, setembro de 2022.

A mi abuelo

Agradecimientos

Quisiera agradecer a mis dos supervisores, Óscar Fresnedo Arias y José Pablo González Coma, por la oportunidad que me han dado al realizar este proyecto junto a mi. También dales las gracias no solo por la paciencia y comprensión con la que siempre me han tratado durante todo el proyecto, sino también por haber logrado transformar este proyecto en una experiencia tan agradable y enriquecedora. En segundo lugar, también agradecer tanto a mi familia como amigos por el cariño que he recibido a lo largo del proyecto y que me ha dado las fuerzas para conseguirlo y cerrar este capítulo de mi vida.

Resumen

Las redes inalámbricas son cada vez más importantes en nuestras vidas lo que implica que nuestros conocimientos sobre ellas también deben avanzar.

Nuestro proyecto consiste en la elaboración de una herramienta capaz de simular la propagación de ondas en un entorno de interior y que nos permite analizar el rendimiento de nuestras redes inalámbricas y lo óptimo que puede resultar la distribución de nuestro espacio de trabajo.

El objetivo principal de esta aplicación es permitir a cualquier usuario, independientemente de sus conocimientos dentro de la materia, poder utilizarla en sus respectivos sectores para poder mejorar sus resultados o incluso aprender más sobre este campo de estudio.

Abstract

Wireless networks are increasingly important in our lives, which means that our knowledge about them must also advance.

Our project consists of developing a tool capable of simulating the propagation of waves in an indoor environment and that allows us to analyze the performance of our wireless networks and how optimal the distribution of our workspace can be.

The main objective of this application is to allow any user, regardless of their knowledge of the subject, to be able to use it in their respective sectors in order to improve their results or even learn more about this field of study.

Palabras clave:

- Comunicación inalámbrica
- Simulación en interior
- Ondas electromagnéticas
- Propagación
- Trayectoria
- Atenuación
- Leyes físicas
- Interfaz gráfica

Keywords:

- Wireless communication
- Indoor simulation
- Electromagnetic waves
- Propagation
- Path
- Path loss
- Physical laws
- Graphic interface

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este proyecto consiste en desarrollar una herramienta de simulación que nos permita recrear el comportamiento de una onda electromagnética propagándose en un entorno cerrado y que nos aporte la posibilidad de analizar el rendimiento de nuestras redes inalámbricas y calcular como de óptima es la distribución de nuestro espacio de trabajo.

1.1 Contextualización

En la actualidad, las comunicaciones inalámbricas están muy presentes en la vida diaria de las personas convirtiéndose, de esta forma, en un elemento fundamental e imprescindible en nuestro día a día, del cual ya dependemos. Tanto en el ámbito laboral como en los hogares, hemos integrado las redes inalámbricas de tal forma que ya hacemos un uso constante de ellas no solo para comunicaciones, sino también en temas de Internet Of Things (IoT), en sistemas domóticos, control de presencia, etc. Por esta razón, consideramos que es muy importante diseñar estos sistemas de comunicación inalámbricos fiables, que permitan transmitir a alta velocidad y dar servicios a un gran número de usuarios de forma simultánea. Para ello, muchas compañías emplean este tipo de herramientas para simular la propagación de ondas electromagnéticas y de esta manera poder perfeccionar la calidad de sus sistemas.

Uno de los principales problemas de modelar estos sistemas de comunicaciones es el grado de complejidad que implica llevar a cabo las transmisiones de ondas electromagnéticas, teniendo en cuenta la cantidad de factores que influyen en la propagación de las mismas. Por tanto, a la hora de modelar estos sistemas en nuestra herramienta, debemos valorar todos los elementos involucrados en el proceso de propagación y sus características. Dependiendo de cada escenario *indoor* considerado, contaremos con ambos nodos de comunicación inalámbrica (nodo transmisor y nodo receptor), la propia señal que se propaga, las paredes del plano y con distintos obstáculos representados por figuras geométricas. Estos objetos serán uno de los puntos más importantes a tener en cuenta en nuestro proyecto, puesto que son los que

provocarán la mayor parte de los cambios en nuestras ondas, produciendo desviaciones en las trayectorias y múltiples rebotes debido al fenómeno de reflexión. La atenuación en la potencia es otro de los efectos que sufrirá la señal, debido a la distancia que deba recorrer hasta alcanzar al nodo receptor y al material de construcción de las paredes de la habitación o de los objetos que haya en el entorno por el entorno que se propague la señal. Por otra parte, estos obstáculos también provocarán, en algunos casos, el efecto de *shadowing*, por el cual las ondas atravesarán el objeto o, por el contrario, quedarán bloqueadas por él mismo dependiendo del material del que estén formados.

Como podemos observar, adentrarnos en el campo de las redes inalámbricas no es trivial. Por tanto, consideramos que el uso de herramientas de este estilo pueden también tener un objetivo didáctico que nos ayude a entender mejor ciertos conceptos relacionados con la propagación de ondas electromagnéticas, desde los elementos que intervienen en el proceso de comunicación hasta que leyes pueden afectar a la onda durante la propagación de la misma. Por esta razón, creemos que la mejor manera es a través de una herramienta visual e intuitiva que nos permita observar el comportamiento de las ondas en distintos escenarios, y comprender como estas se ven afectadas por las características de su entorno.

1.2 Motivación

Como bien sabemos, hoy en día, estamos completamente rodeados de redes inalámbricas, las cuales necesitamos para poder realizar tareas que ya consideramos básicas en nuestra vida cotidiana. La importancia que están adquiriendo estos sistemas es algo que debemos tener en cuenta y, por ello, debemos entender como funcionan, no solo para ser capaces de utilizarlas correctamente, sino también para poder aplicar estos conocimientos a otros ámbitos. De manera personal, el mundo de las comunicaciones siempre me ha parecido muy interesante y aprovechando que la informática y las redes inalámbricas van de la mano, escogí realizar este proyecto para fusionar ambos campos en una herramienta. Por ello, comenzamos este proyecto con el objetivo de aplicar mis conocimientos adquiridos en el grado y, a la vez, ampliar mis conocimientos acerca del proceso de propagación de ondas.

Todos sabemos que, en muchas ocasiones, para entender mejor alguna materia vista en clase de teoría, es preferible realizar ejercicios prácticos para consolidar ciertos conceptos que puedan ser más complejos. Por ello, consideramos que esta herramienta podría servir de gran ayuda a aquellos estudiantes que se estén iniciando en el campo de las redes inalámbricas, principalmente como complemento práctico a la hora de estudiar la propagación de ondas y así comprender mejor los factores que intervienen en la misma. Por otro lado, a un usuario en su ámbito laboral, podría ayudarle en el diseño, planificación y despliegue de un sistema de comunicaciones como, por ejemplo, de la red *wifi* de las oficinas de una empresa. En este

caso, resultaría útil poder conocer cual sería la colocación idónea de los nodos o puntos de acceso y las características necesarias de los mismos para conseguir una cobertura adecuada de la señal.

Por último, a todos nos gusta que nuestra conexión a internet en nuestros hogares vaya lo mejor posible, tanto a la hora de ver una película, como a la hora de trabajar. Sin embargo, no siempre es así. Por ello, creemos que esta aplicación podría ser muy útil para que cualquier persona en su casa fuera capaz de escoger la mejor estancia para colocar su *router*, con el objetivo de conseguir una cobertura óptima de la señal, sobre todo en aquellas habitaciones donde hubiera dispositivos de conexión importantes.

1.3 Objetivos

Los objetivos principales que deseamos completar con el desarrollo de este proyecto son los siguientes:

- Comprender la teoría básica referente a la propagación de las ondas electromagnéticas y los fenómenos físicos asociados a ella.
- Definir un modelo sencillo para estos fenómenos físicos en función de los objetos que nos podemos encontrar en un escenario *indoor*.
- Implementar la lógica necesaria para representar los distintos eventos (rebotes, atenuaciones, bloqueos, ...) que pueden suceder durante una transmisión inalámbrica en un escenario concreto de acuerdo a este modelo.
- Implementar una interfaz gráfica que permita al usuario configurar todos los parámetros del escenario que quiere simular y visualizar los resultados obtenidos de una forma cómoda e intuitiva.

1.4 Visión General de la herramienta

Esta herramienta gráfica se ha diseñado con el objetivo de proporcionar al usuario un software sencillo e intuitivo, con el cual sea capaz de crear escenarios realistas, pudiendo configurar las dimensiones de las habitaciones, los materiales, la posición de los objetos que estén involucrados, o incluso la forma de los mismos haciendo uso de figuras geométricas. También podrá parametrizar diferentes aspectos relacionados con las señales inalámbricas (potencia, frecuencia, dirección de salida), con el fin de simular diferentes casos reales de propagación de ondas. Por último, también se podrá escoger la posición de los diferentes nodos implicados en la comunicación. En la figura 1.1, se muestra la interfaz gráfica de nuestra aplicación donde el usuario podrá configurar los distintos escenarios y visualizar los datos.

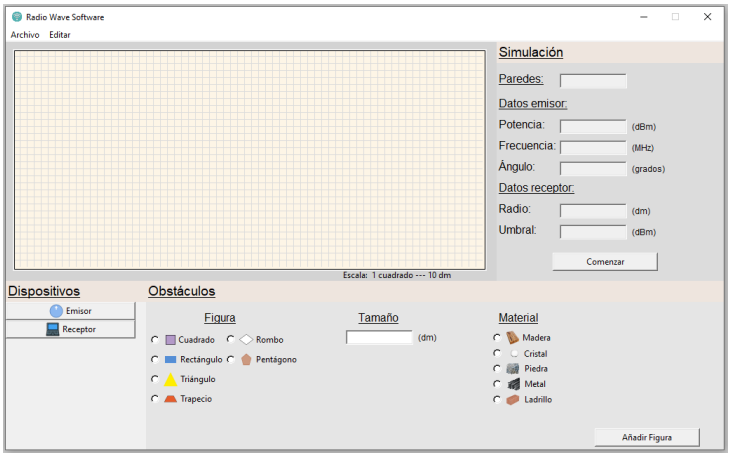


Figura 1.1: Vista Principal de la herramienta

1.5 Estructura de la memoria

Esta memoria está dividida en diferentes apartados, los cuales avanzan desde un punto de vista más genérico del proyecto, englobando puntos importantes como la preparación del mismo, hasta alcanzar un punto más cercano al desarrollo y la implementación de la propia herramienta. A continuación, mostramos cada uno de los capítulos que componen este documento.

- **Introducción:** En el primer capítulo, describimos el contexto en el que se desenvuelve nuestro proyecto y los objetivos que deseamos alcanzar.

- **Revisión del estado del arte:** En este segundo capítulo, realizamos una investigación acerca de las aplicaciones existentes en el mercado que sean similares a la nuestra y una comparativa para explicar las mejoras que nuestra herramienta puede aportar.

- **Fundamentos teóricos:** Este apartado está dedicado a explicar los conceptos teóricos necesarios para llevar a cabo nuestro proyecto, mostrando al mismo tiempo la complejidad del mismo.

- **Fundamentos tecnológicos:** En este cuarto capítulo de la memoria, ofrecemos una explicación de todas las herramientas y tecnologías utilizadas en la elaboración de nuestro proyecto y una justificación de el motivo por el que se eligieron.

- **Metodología, recursos y planificación:** En este capítulo, describiremos tanto la metodología escogida para acometer el desarrollo de las diferentes etapas de nuestro proyecto, como los recursos empleados en el mismo.

- **Análisis del proyecto:** En el sexto capítulo, describiremos los requisitos que se han identificado para la aplicación que se va a desarrollar, junto con la arquitectura escogida para crear el diseño de la misma.
- **Desarrollo de la herramienta:** En el séptimo capítulo, abordaremos más en profundidad la implementación llevada a cabo en cada una de las iteraciones del proyecto.
- **Funcionamiento de la herramienta:** En este penúltimo capítulo, describiremos el funcionamiento de nuestra herramienta a modo de manual de usuario para mostrar cada uno de los pasos a seguir.
- **Conclusiones:** En el último capítulo de la memoria, mostraremos las conclusiones que hayamos sacado en claro tras la finalización del proyecto, así como las posibles mejoras que se puedan añadir en un futuro.

Capítulo 2

Revisión del estado del arte

Una vez comprendido el contexto en el que se desenvuelve nuestro proyecto, es interesante explicar como hemos analizado el mercado en busca de herramientas software que tuvieran una cierta semejanza con la herramienta que pretendemos construir. El objetivo de dicha investigación fue intentar adquirir ideas para nuestro desarrollo e identificar potenciales mejoras que incorporar en la herramienta a implementar. En nuestro caso, encontramos varias aplicaciones muy prometedoras, de las cuales hablaremos a continuación.

2.1 CINDOOR

La primera herramienta que nos encontramos es un software en desarrollo llamado CINDOOR [6], el cual fue creado por la Universidad de Cantabria para mejorar la planificación y el diseño de sistemas inalámbricos reales. En concreto, nos permite modelar en un plano 3D entornos reales, tanto *indoor* como *outdoor*, para así poder simular la cobertura y el rendimiento del sistema inalámbrico, siendo dos funciones básicas para el análisis eficiente de las redes inalámbricas.

Características:

Este software es bastante complejo por la cantidad de valores que se pueden modificar al preparar cada una de las simulaciones. A la hora de utilizar esta aplicación, podemos seleccionar distintos tipos de análisis sobre la propagación de ondas, también nos permite escoger en que puntos evaluar la señal que se esté transmitiendo en ese momento, consiguiendo así una mayor precisión en sus resultados. Otros aspectos que se pueden configurar usando esta herramienta son las características de las antenas utilizadas en la simulación, así como las características de las paredes del plano, ya que consideran que es un punto muy importante en la transmisión inalámbrica de señales.

Esta herramienta está diseñada para ser utilizada en sistemas operativos como Windows, contando con una interfaz gráfica y sencilla que convierte un software complejo en una he-

rramienta intuitiva y fácil de usar. Podemos observar un ejemplo de dicha interfaz en la figura 2.1, donde se muestra como se configurarían las propiedades del receptor.

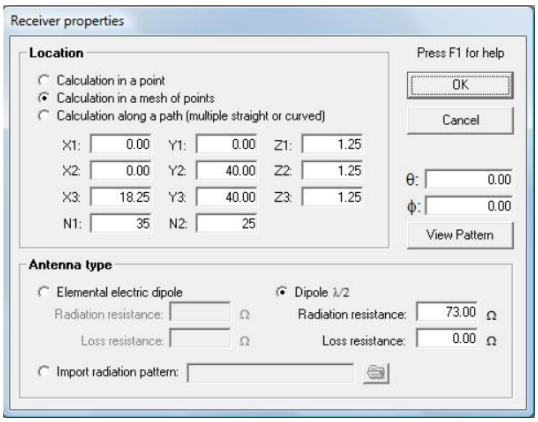


Figura 2.1: Interfaz gráfica de CINDOOR: propiedades receptor

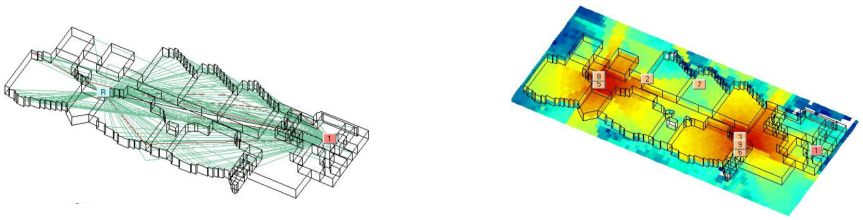
Visualización de datos:

Otro punto interesante de esta herramienta es que permite al usuario visualizar y analizar la información que se genera a través de diferentes gráficos y ventanas emergentes, ayudando de esta forma al usuario a sacar mejores conclusiones acerca del proceso de propagación de ondas en los escenarios creados. Esta aplicación dispone de un gran conjunto de gráficos y ventanas de resultados. Entre todas ellas, destacan tres fundamentales:

- El trazado de rayos, para visualizar las distintas trayectorias que sigue el haz de la onda emitida por el transmisor. Esta gráfica se puede visualizar a continuación en la figura 2.2a.
- La ventana donde se muestran las estadísticas de la señal, en la cual se muestran diferentes datos relevantes tras la simulación, y gracias a los cuales podremos apreciar como la onda fue afectada por el entorno.
- Por último, el mapa de cobertura es otra gráfica importante a tener en cuenta, ya que nos muestra los puntos de calor en el plano donde la señal llega con más intensidad, tal y como se ve en la figura 2.2b.

2.2 RemCom

RemCom [7] es una empresa estadounidense dedicada al desarrollo de un amplio abanico de productos software innovadores centrados principalmente en la simulación electromagnética y la propagación inalámbrica. Esa empresa proporciona una gran variedad de herramientas versátiles y ágiles de utilizar por los diferentes clientes a los que da soporte, entre ellos,



(a) Trazado de rayos

(b) Mapa de cobertura

Figura 2.2: Gráficos de pantalla en CINDOOR

diferentes patrocinadores del gobierno de los Estados Unidos y grandes multinacionales de gran prestigio, cubriendo de esta forma diversas necesidades. Dentro de la lista de productos que RemCom ha desarrollado, podemos destacar tres productos dirigidos especialmente a la simulación y propagación de ondas de radar dentro de espacios cerrados, por lo que pueden presentar funciones muy similares a las que nosotros buscamos.

2.2.1 Wireless InSite

El primer producto que destacamos es Wireless InSite [8], una herramienta corporativa cuya función principal es modelar las trayectorias de los rayos y analizar diferentes sistemas de comunicación inalámbrica. Este software de propagación de ondas proporciona predicciones precisas de las características del canal de comunicación en diversos entornos complejos, como pueden ser entornos urbanos, interiores, rurales o mixtos.

Aunque existan diferentes tipos de entornos cada uno con sus características particulares, existen algunos desafíos comunes, incluyendo el uso de antenas y la existencia de estructuras y obstáculos que provocarán la aparición de múltiples trayectorias y una dispersión significativa de la señal. En el caso de de aplicar una herramienta como esta en entornos *indoor*, nos debe permitir modificar las opciones correspondientes a este tipo de escenarios, como por ejemplo, las dimensiones de las habitaciones, las características de las estructuras u obstáculos que allí nos encontremos, todo con el fin de conseguir resultados más precisos y fieles a la realidad.

Wireless InSite predice con exactitud los efectos provocados por las múltiples trayectorias que sigue una señal en ambientes interiores, ofreciendo una poderosa herramienta para la simulación en la que se detallan las características del canal, incluyendo los efectos de absorción y dispersión, las pérdidas incurridas por la propagación a través de las paredes y el intenso *multipath* causado por interacciones con superficies y muebles dentro de las habitaciones, tal y como podemos observar en la imagen de la figura 2.3.

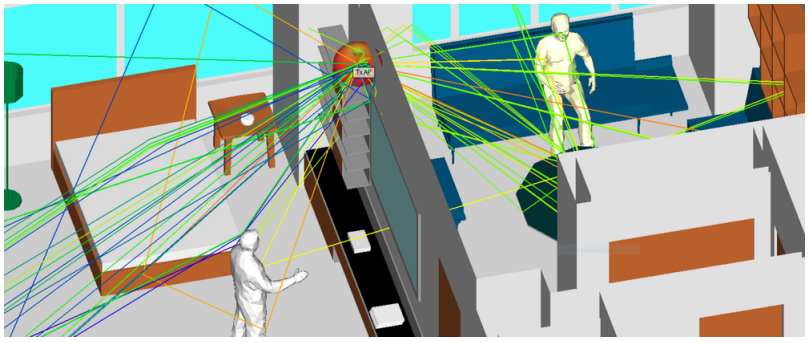


Figura 2.3: Wireless InSite: Ejemplo de reflexiones y difracciones

Versiones y requisitos del sistema:

RemCom ha decidido adoptar en cuanto al desarrollo de los diferentes productos un mecanismo de *multiversion*, elaborando para cada uno de los productos de la compañía diferentes versiones del mismo, distribuyendo las diferentes funciones que conforman el producto entre distintas opciones y creando, de esta forma, un abanico de utilidades entre las cuales los clientes podrán escoger. De esta forma, Remcom permite a los usuarios poder adquirir aquella versión del producto que más se ajuste al presupuesto de la empresa y al problema que desean abordar, evitando en muchos casos comprar un producto más sofisticado y caro sin ninguna necesidad. En este caso, Wireless InSite consta de las siguientes versiones:

- **Wireless InSite Standard:** Esta versión contiene un conjunto básico de modelos de propagación sobre trazado de rayos bidimensional y tridimensional de gran exactitud. Esto permite cubrir las operaciones principales del programa y le permite al cliente poder analizar sistemas de comunicación inalámbricos que se ajusten a las características que esta versión pueda configurar. Podemos observar en la figura vista anteriormente (figura 2.3), un ejemplo de modelado de propagación 3D sin restricciones en cuanto a la geometría o la colocación de las antenas en el plano.

- **Wireless InSite Professional:** La segunda versión agrupa las funciones y configuraciones disponibles en la versión anterior, agregando nuevos modelos empíricos y basados en rayos rápidos. También incluye nuevas funciones para que el cliente pueda desarrollar configuraciones y escenarios más personalizados.

- **Wireless InSite MIMO:** Por último, la compañía crea una tercera versión para añadir a las operaciones de la herramienta las capacidades necesarias para poder soportar la tecnología Multiple-input Multiple-output (MIMO).

Otro aspecto interesante que debemos comentar en este punto son los requisitos del sistema necesarios para poder utilizar esta aplicación en la práctica. En la mayoría de casos,

aplicaciones de este nivel suelen necesitar ser ejecutadas en equipos con unas prestaciones elevadas, ya que son herramientas muy potentes. Para este producto en cuestión, la compañía recomienda contar con un procesador intel i7 y cuatro procesadores lógicos, a parte de necesitar 1TB de espacio en disco, 16 GB de RAM y 6 GB de GPU RAM, contando que disponemos de una tarjeta gráfica con capacidad CUDA y compatible con NVIDIA OptiX 4.1.0. Como se puede ver, estas características conllevan una inversión cuantiosa para poder adquirir un equipo en condiciones.

Por otro lado, tras observar las especificaciones del software, vemos que dispone de una interfaz gráfica muy compleja que solo es compatible con sistemas Windows y algunos sistemas Linux, dejando de esta forma poco margen de maniobra a usuarios que dispongan de un equipo con un sistema operativo de otra distribución. Los cálculos que este software debe realizar son muy complejos y, como es de esperar, se requiere una capacidad de cómputo superior a otras aplicaciones del mercado, lo que provoca que sea solo compatible con algunos sistemas operativos.

Estos factores son realmente importantes ya que limitan mucho el acceso por parte de algunos usuarios a este tipo de productos, convirtiéndose de esta manera en herramientas destinadas al uso por parte de grandes empresas y organizaciones, dejando de lado a particulares que también se dediquen a la investigación o formación en estos campos.

Características y aplicaciones de uso:

A pesar de disponer de diferentes versiones del producto, la versión estándar ya cuenta con una gran cantidad de operaciones que nos permiten analizar en profundidad la comunicación inalámbrica que se esté produciendo en los diferentes escenarios *indoor*. Una vez nos hemos informado sobre las características de esta aplicación, podemos destacar algunas de las que nos han parecido más interesantes y que más se asemejan al resultado que queremos conseguir en nuestro proyecto.

Comenzaremos comentando algunas de las funcionalidades que nos parecen fundamentales en una aplicación que trata sobre propagación de ondas. En primer lugar, vemos que el usuario tendrá la opción de ajustar parámetros importantes de la simulación como pueden ser el coeficiente de reflexión o valores de configuración de la propia transmisión. También tendrá la oportunidad de crear o de alguna manera manipular los escenarios predefinidos. En este caso, el software además de incluir modelos predefinidos de interiores, también permite al usuario editar gráficamente el aspecto del plano para ajustarlo al diseño que él quiera mediante operaciones como rotar o escalar los obstáculos o incluso generar nuevos elementos para incluirlos en el escenario, modificar las propiedades de los materiales de construcción y alternar diferentes antenas de transmisión con diferentes configuraciones.

Una vez el usuario haya terminado de preparar su entorno de simulación, este podrá comenzar el análisis completo de la propagación de la onda en múltiples trayectorias y obtener

los resultados. Esta herramienta es perfecta para determinar la ubicación ideal de los puntos de acceso de nuestra red inalámbrica en un entorno complejo y conseguir una conexión óptima en todo momento.

Visualización de datos:

Cuando valoramos una herramienta, es fundamental analizar el tipo de datos que la aplicación es capaz de generar como resultado tras las simulación realizada y como se puede visualizar esta información para poder analizarla posteriormente. Esta herramienta nos permite obtener datos muy relevantes sobre la propagación de ondas que se produce en la comunicación inalámbrica como, por ejemplo, datos sobre las características del canal y de los sistemas de comunicación. A continuación, señalamos algunos de los datos más interesantes.

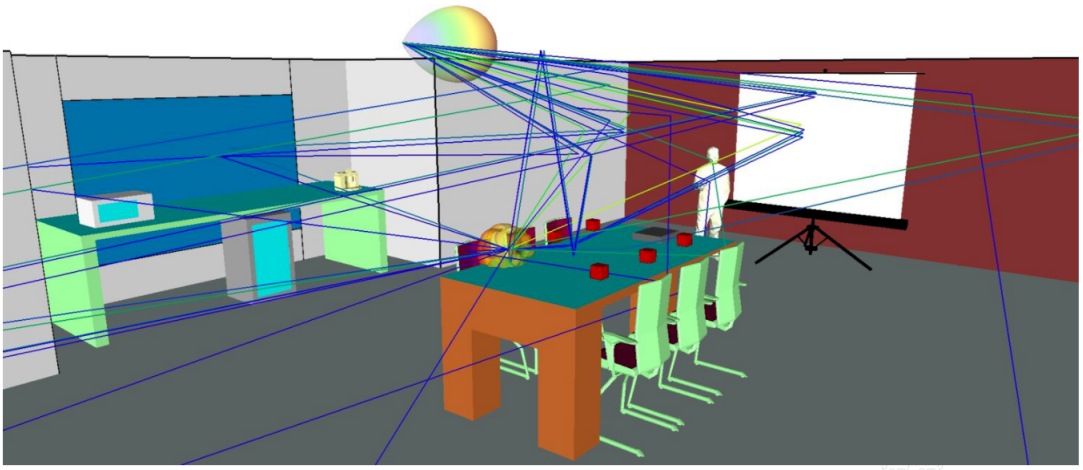


Figura 2.4: Wireless InSite: Modelo de propagación 3D

En primer lugar, comentaremos aquellos datos que la aplicación muestra visualmente porque, a primera vista, pueden resultar más sencillos de comprender al verlos en el contexto del escenario. Esta herramienta nos muestra la mayoría de resultados con diferentes colores para diferenciar bien las diferentes trayectorias que toma la onda a través de nuestro entorno. Un ejemplo de esta funcionalidad se puede observar en la figura 2.4.

Por otro lado, Wireless InSite dispone de un modelo de recuento de muros que aplica en los cálculos para entornos de interior. Los rayos directos se construyen entre el transmisor y el receptor, y cada pared con la que impacta la señal es contada, lo que provocará una pérdida adicional de potencia sumada a la pérdida producida por la transmisión en espacio libre. A mayores del recuento, esta opción también nos mostrará un gráfico visual o mapa de calor, como el de la figura 2.5, donde podremos ver las zonas del espacio donde se producen más perdidas.

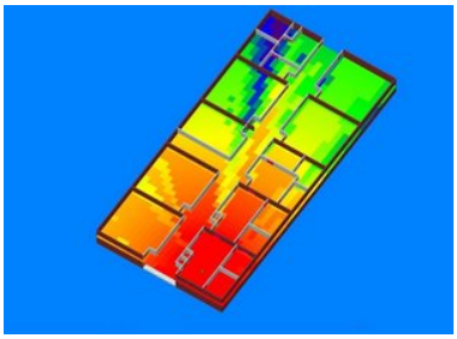


Figura 2.5: Wireless InSite: Contador de muros intermedios

Por último, disponemos también de una extensa lista de resultados numéricos que son fundamentales en el análisis de nuestras simulaciones ya que mantendrán una cierta sinergia con las representaciones visuales. En concreto, este software nos aporta todo tipo de información acerca de lo ocurrido durante las transmisiones. Algunos de estos datos pueden ser las pérdidas de potencia producidas por el trayecto, tasa de error de bits (BER) o la calidad recibida respecto a la señal de referencia (RSRQ). También nos muestra una comparativa final entre la potencia total emitida y la potencia recibida, o las direcciones de salida de la onda desde el transmisor y dirección de llegada al receptor, etc.

2.2.2 WaveFarer® y XGtd®

En segundo lugar, tenemos dos productos creados por RemCom para completar el análisis de redes inalámbricas, su función principal consiste en detectar y analizar obstáculos que se encuentren próximos al nodo transmisor, es decir, ideal para el estudio de escenarios *indoor*. Ambas herramientas son muy importantes ya que disponen de funcionalidades diferentes que, al combinarse, nos permiten conseguir resultados más completos.

WaveFarer®:

Por un lado, tenemos WaveFarer [9], un simulador de radar de alta fidelidad creado para realizar estudios rápidos y precisos. La especialidad de esta herramienta consiste en el análisis de las múltiples trayectorias que se generan cuando la onda principal se dispersa al colisionar con las distintas estructuras del escenario. Este software de simulación está diseñado para ser capaz de soportar otras aplicaciones relevantes de simulación y análisis como, por ejemplo, el siguiente producto que nos ofrece esta compañía.

XGtd®:

XGtd® [10] es una herramienta de análisis electromagnético que nos aporta, por un lado, métodos muy útiles para realizar un rastreo de rayos como pueden ser el disparo y el rebote de rayos, y por otro, métodos para incluir la influencia de leyes físicas en sus cálculos multitra-

yecto, desde la reflexión, difracción o la propia interacción de la onda con muebles, paredes u otros materiales. A parte de estas funcionalidades, esta tecnología también permite a los usuarios generar animaciones de trayectorias de rayos para mejorar la visualización de los resultados en las simulaciones.

2.3 Conclusiones sobre el estado del arte

En este capítulo, hemos presentado dos herramientas existentes en el mercado, las cuales fueron creadas para el análisis de sistemas inalámbricos con el objetivo de poder estudiar el rendimiento de los canales de comunicación e incluso examinar como de óptimo puede ser el diseño de un sistema de comunicación en función del entorno. Ambas aplicaciones tienen funcionalidades muy parecidas que nos permiten tanto modificar diferentes elementos de un escenario *indoor*, como también configurar las características de los dispositivos y elementos involucrados en la simulación ajustando cada prueba a la realidad. Por otro lado, también vemos que el tipo de resultados que se obtienen en ambos casos son muy similares, tanto la información numérica como los gráficos. Para la elaboración de nuestro programa, hemos considerado muchas similitudes de estas dos herramientas para así conseguir desarrollar una aplicación útil para nuestros usuarios y también competitiva dentro del mercado, ya que nuestra herramienta va a disponer de aquellas funcionalidades necesarias para el correcto análisis de entornos *indoor*, que es nuestro objetivo principal.

A continuación, mencionaremos las principales ventajas de nuestro software que nos diferencian respecto a las aplicaciones anteriores y que nos podrían permitir destacar dentro del mercado.

En primer lugar, nuestra herramienta dispone de las principales funcionalidades tanto para configurar las características de entornos complejos como también para realizar todos los cálculos y operaciones necesarias para conseguir simulaciones fieles a la realidad. Por esta razón, nuestra aplicación es una buena elección para ser empleada como herramienta de apoyo a la hora de analizar sistemas de comunicaciones inalámbricos y averiguar como de óptima es la distribución de nuestro espacio de trabajo, consiguiendo un punto de vista diferente. A pesar de ello, nuestra aplicación fue desarrollada con un diseño más intuitivo que el resto, lo que implica que es más fácil de comprender como se debe manejar. Este es el principal motivo por el cual consideramos que nuestra herramienta podría tener un uso también didáctico y ser utilizada por usuarios interesados en el campo de las comunicaciones inalámbricas que, independientemente de sus conocimientos dentro de la materia, pudieran emplear este software como apoyo práctico para afianzar sus conocimientos sobre la propagación de ondas.

En segundo lugar, nuestra aplicación nos permite diseñar escenarios complejos por los diversos elementos con los que podemos trabajar, pero a diferencia de las otras aplicaciones, no se necesita tener conocimientos de diseño o modelado 3D para generar los planos del entorno, puesto que en nuestro caso simplemente debemos configurar que elementos deseamos incluir y luego distribuirlos sobre el plano. Esto también reduce el tiempo que el usuario debe dedicar a la creación de los planos y comenzar antes con la simulación y el análisis de resultados.

Por último, al igual que en las aplicaciones del mercado, en nuestras simulaciones también se realizan resoluciones de problemas complejos y cálculos avanzados para conseguir los resultados más realistas posibles. Sin embargo, nuestro programa es capaz de mantener el mismo rendimiento independientemente de las prestaciones del equipo donde se esté ejecutando. Esta característica es muy importante porque permite a una mayor cantidad de personas poder utilizar nuestra herramienta sin necesidad de realizar cuantiosas inversiones.

Capítulo 3

Fundamentos teóricos

En este capítulo, explicaremos los conceptos teóricos que hemos necesitado aprender para poder desarrollar nuestro proyecto, manteniéndonos fieles a los fenómenos físicos que influyen en toda comunicación inalámbrica y que se deben tener en cuenta para realizar los cálculos correctamente y obtener unos resultados cercanos a la realidad.

3.1 Comunicación inalámbrica

La comunicación inalámbrica es un tipo de comunicación entre dos o más nodos que no se encuentran unidos por ningún medio de propagación físico, sino que utiliza la modulación de ondas electromagnéticas a través del propio espacio [11].

3.1.1 Elementos de una comunicación

Para una mayor comprensión del funcionamiento de la comunicación inalámbrica, también debemos estudiar que tipo de elementos básicos componen dicha comunicación. En 1948, Shannon y Weaver lanzaron una teoría matemática de la comunicación [12], en la cual elaboraban su modelo sobre el proceso de comunicación basado en cinco elementos: fuente, transmisor, canal, receptor, destinatario. Des estos cinco elementos podríamos destacar tres que serían fundamentales para nuestra herramienta: el transmisor, el receptor y el canal. Los otros dos elementos consideramos que no son relevantes para nuestro modelo de propagación de ondas, puesto que su implicación sería anterior (fuente) y posterior (destinatario) a la propia propagación inalámbrica. A continuación, describimos brevemente los tres elementos que tienen importancia en nuestro modelo para la creación de la herramienta.

Canal:

Se conoce por canal al medio físico que debe transportar las señales entre dos nodos de comunicación inalámbrica. Este tipo de comunicaciones no se encuentran unidas por un medio de propagación físico dedicado, si no que se utiliza la modulación de ondas electromagnéticas

a través del espacio libre. En nuestro escenario, el canal viene determinado por la configuración del entorno y por los fenómenos físicos que contribuyen a la transmisión o atenuación de las ondas electromagnéticas radiadas desde el transmisor.

Transmisor:

Un transmisor en el campo de las comunicaciones es un equipo que emite una señal, código o mensaje a través de un medio. En el caso de nuestro proyecto, trabajaremos con un caso particular de transmisor conocido como transmisor de radio o radiotransmisor y que se define como un dispositivo electrónico que, mediante una antena, radia ondas electromagnéticas que transportan información. Una variable importante a tener en cuenta en un transmisor es la potencia de transmisión, que consiste en el nivel de potencia que un equipo inalámbrico puede emitir. Por otro lado, también tenemos el ángulo de salida que nos indica la dirección hacia la cual se emitirá la onda.

Receptor:

Un receptor en el área de las comunicaciones inalámbricas es un dispositivo electrónico que, mediante la ayuda de una antena, es capaz de recoger las ondas electromagnéticas que viajan por el espacio libre transportando todo tipo de información y filtrar aquellas que están realmente destinadas para él. Una variable asociada al receptor es la sensibilidad (potencia mínima en dBm con los que debe llegar la señal para poder capturarla). También en el receptor dispondremos de un ángulo, pero esta vez de llegada, que nos indicará en que direcciones es capaz de captar la señal.

Antena:

Una antena es un dispositivo, normalmente conductor metálico, capaz de emitir (o recibir) ondas electromagnéticas hacia el espacio libre. Algunas de las características fundamentales que tendremos en cuenta durante el desarrollo de nuestra aplicación para las antenas serán la ganancia y la eficiencia de una antena:

- Por un lado, la ganancia de una antena se define como el beneficio de potencia en la dirección de máxima radiación, es decir,

$$G_{[\text{dBi}]} = 10\log_{10}((4\pi EA)/L^2), \tag{3.1}$$

  donde E es la eficiencia, L la longitud de onda y A es el área de apertura física.

- En segundo lugar, también consideraremos la eficiencia de una antena que se puede definir como la relación entre la potencia radiada por el emisor $P(r)$ y la potencia entregada al receptor $P(\text{in})$ en ausencia de pérdidas de propagación, es decir,

$$e = P(r)/P(\text{in}) = G/D, \tag{3.2}$$

donde G y D representan la ganancia de la antena y la potencia disipada en la emisión, respectivamente.

Teniendo en cuenta estos elementos en la comunicación, resulta interesante comentar las características de los dispositivos que consideraremos en nuestra herramienta con el objetivo de simplificar el modelo de propagación de ondas. Por una parte, hemos decidido que en el caso de nuestro transmisor estará compuesto por una antena muy direccional que emitirá las ondas electromagnéticas en la dirección que el usuario le indique como parámetro. Además, el usuario podrá especificar la potencia de transmisión con la que se emitirán las ondas electromagnéticas. Por el contrario, el receptor de nuestras simulaciones será omnidireccional, es decir, será capaz de recoger las ondas electromagnéticas que viajen por el entorno de simulación alrededor de un área específica de captación que, de nuevo, será configurable al igual que la sensibilidad mínima para poder adquirir una señal.

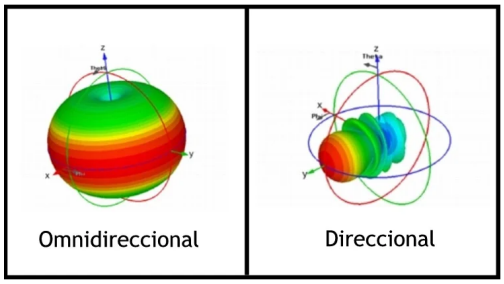


Figura 3.1: Comparación entre antena omnidireccional y direccional [1]

3.2 Radiación electromagnética

La radiación electromagnética es una combinación de campos eléctricos y magnéticos oscilantes que se propagan a través del espacio libre como ondas transportando energía de un lugar a otro. Dentro del contexto de nuestro proyecto, esta radiación son las propias ondas generadas por las fuentes del campo electromagnético y que se propagan a la velocidad de la luz. Tanto la generación como la propagación de estas ondas concuerdan con el modelo de ecuaciones matemáticas definido por Clerk Maxwell [13].

A diferencia de otros tipos de onda, como las del sonido, este tipo de ondas electromagnéticas no necesitan de un medio material para propagarse ya que tienen la capacidad de propagarse en el vacío. Como anécdota, en el siglo XIX se pensaba que existía una sustancia invisible, llamada éter, que ocupaba el vacío y servía de medio de propagación de las ondas electromagnéticas.

3.2.1 Ondas electromagnéticas

Una onda [14] o mejor dicho, un movimiento ondulatorio, es un tipo de movimiento que

transporta energía y momento lineal, pero no transporta materia. El ejemplo típico serían las oscilaciones que se forman al lanzar una piedra al centro de un estanque. Las ondas pueden clasificarse de diversas forma:

- Según el sentido de vibración:
  - **Ondas trasversales:** En las cuales la vibración es perpendicular a la dirección de propagación.
  - **Ondas longitudinales:** En estas segundas, la vibración es en la misma dirección que la dirección de propagación.
  - **Ondas mixtas:** Estas ondas son una combinación de las dos anteriores.
- En función de si necesitan un medio material para propagarse:
  - **Ondas mecánicas:** En este tipo de ondas lo que oscila es un medio material.
  - **Ondas no mecánicas:** Por el contrario, estas ondas no necesitan de ningún medio material para propagarse.

Por tanto, en relación a las ondas electromagnéticas que son las que nos conciernen en el desarrollo de nuestro proyecto, podemos decir que son ondas no mecánicas al no depender de ningún medio físico y transversales como se puede demostrar a partir de las ecuaciones de Maxwell.

Por otra parte, dos parámetros muy importantes relacionados con las ondas que debemos tener en cuenta también en nuestras simulaciones, son:

- La frecuencia, que consiste en el tiempo que tarda la onda en completar una oscilación, es un parámetro muy importante ya que influye decisivamente en las atenuaciones debidas a la propagación en espacio libre, y en las pérdidas por reflexión o por dispersión junto al coeficiente de penetración de los materiales.
- La longitud de onda, que se define como la distancia que hay entre dos puntos que están en el mismo estado de vibración, es una variable que también tiene su importancia en el fenómeno de *shadowing* ya que menor longitud de onda implica menor posibilidad de superar determinados obstáculos. Estos dos parámetros se relacionan de forma inversamente proporcional, es decir, menor longitud de onda implica una mayor frecuencia de la señal.

3.2.2 Propagación de ondas

Una vez que ya sabemos lo que son las ondas electromagnéticas, debemos comprender como se propagan y porque esto es tan importante para nuestro estudio.

La propagación de ondas de radio es el comportamiento de las ondas electromagnéticas cuando se desplazan por el espacio libre. Todo sistema de telecomunicación debe diseñarse para que en el receptor pueda obtener un rendimiento mínimo que garantice su funcionamiento, pero a veces la propia propagación y sus efectos asociados pueden provocar que la señal transmitida se atenúe y que no pueda ser captada ya que el receptor tiene una cierta sensibilidad de captación. En la siguiente sección, veremos algunos de los fenómenos físicos que pueden afectar al comportamiento en la propagación de las ondas electromagnéticas.

3.3 Efectos asociados a la propagación de las ondas

Llegados a este punto, tenemos que explicar los diferentes fenómenos físicos que afectan a las ondas electromagnéticas durante su propagación, ya que debemos tener en cuenta todas estas leyes físicas para elaborar el modelo que va a ser usado durante el desarrollo de nuestras simulaciones y ser así fieles a la realidad.

3.3.1 Pérdida de potencia en el espacio libre

En los sistemas de comunicación inalámbricos, la pérdida de potencia en el espacio libre es la atenuación de energía que sufre la onda electromagnética cuando se propaga entre dos nodos de comunicación. Dicha atenuación está asociada a la propia pérdida de energía que sufren las ondas electromagnéticas cuando se propagan en el espacio y depende de la frecuencia de las mismas y de la distancia recorrida. Esta atenuación no tendrá en cuenta las pérdidas de potencia en las propias antenas debido a imperfecciones en las mismas [15]. En muchos diseños de sistemas, las ganancias que proporcionan las antenas se utilizan para compensar en parte estas pérdidas.

Fórmula de Friis

Aunque existen otros modelos, la fórmula de Friis es el modelo más habitual para tratar las pérdidas de potencia en espacio libre. Esta pérdida se puede calcular con la siguiente ecuación:

$$P(d)_{[\text{dB}]} = -32.5 - 20\log_{10}(d) - 20\log_{10}(f), \tag{3.3}$$

donde d es la distancia recorrida y f es la frecuencia de la onda.

3.3.2 Reflexión

El fenómeno de reflexión será fundamental para la implementación de los rebotes que se produzcan en las simulaciones con nuestra aplicación. Cuando las ondas alcanzan una pared o un obstáculo, se debe recalcular la trayectoria que sigue dicha onda tras la colisión. La ley de

la reflexión describe el comportamiento de un rayo cuando alcanza una superficie reflectante ya que, en ese caso, el rayo será reflejado con el mismo ángulo con el que incide, tomando siempre como referencia la normal a la superficie con la que colisiona [16]. Podemos ver un ejemplo visual de este fenómeno físico en la siguiente figura 3.2.

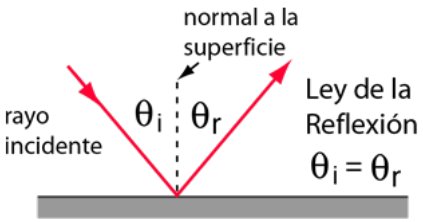


Figura 3.2: Ley de reflexión [2]

3.3.3 Refracción

La refracción es el cambio de dirección que experimenta una onda al pasar de un medio físico a otro con distinto índice refractivo. Solo se produce si la onda incide oblicuamente sobre la superficie de separación de los dos medios y si estos tienen índices de refracción distintos. En concreto, la refracción se origina por el cambio de velocidad de propagación de la onda correspondiente al cambiar de medio [17]. Este fenómeno lo podemos ver representado en la figura 3.3.

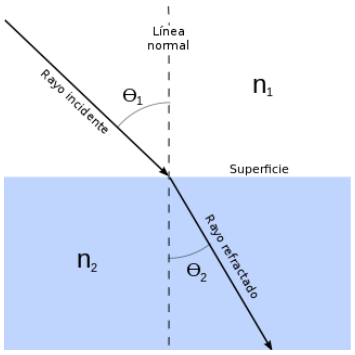


Figura 3.3: Fenómeno de refracción [3]

Algunas de las causas que pueden ocasionar este fenómeno durante una comunicación inalámbrica en un entorno de interior puede ser la colisión de las ondas sobre un objeto de algún tipo de vidrio, como son algunos de los objetos que podemos añadir en la aplicación especificando ese material. El resultado de este fenómeno puede acabar afectando a la onda que se está propagando dando lugar a disminuciones de la velocidad de propagación y un desvío del ángulo de la onda, de acuerdo a la ley de Snell.

3.3.4 Difracción

La difracción es un término que se atribuye a varios fenómenos que ocurren cuando una onda se encuentra con un obstáculo. Puede definirse como la curvatura y dispersión de una señal de radiofrecuencia cuando esta se encuentra un determinado objeto en su trayectoria con un tamaño similar al de su longitud de la onda y con unas características físicas o geometría particular que causan el efecto de difracción. Cuando esto ocurre, las ondas afectadas se doblan alrededor del objeto en cuestión, tomando caminos más largos y diferentes, lo que provocará una reducción de la potencia [18]. En la figura 3.4, mostramos un ejemplo de dicho fenómeno.

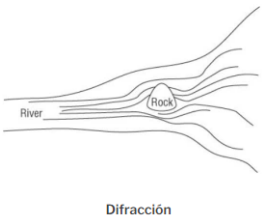


Figura 3.4: Ejemplificación del fenómeno de difracción [4]

3.3.5 Dispersión

En física, se denomina dispersión al fenómeno de separación de las ondas en distintas frecuencias al atravesar un determinado material. El ejemplo más clásico de este fenómeno es la dispersión de un haz de luz blanca en luz de distintos colores (distintas frecuencias) al atravesar un prisma de cristal, creando una situación similar a la que se muestra en la figura 3.5. La señal original, por tanto, se dispersará en múltiples señales secundarias, lo que además disminuirá la potencia de la ondas resultantes [18]. Este fenómeno resulta muy interesante a la hora de modelar la propagación de ondas electromagnéticas en nuestra herramienta ya que puede suponer recibir varias réplicas de la señal original en el receptor. Nuestra onda electromagnética se puede ver afectada por este fenómeno físico cuando colisiona con algún tipo de superficie irregular que no es completamente sólida y se refleja en múltiples direcciones dependiendo del material/es del objeto y de la forma geométrica.

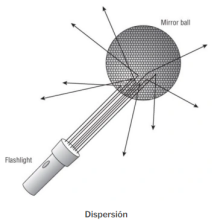


Figura 3.5: Fenómeno de dispersión [4]

3.3.6 Shadowing

El efecto de *shadowing* o también conocido como sombreado, consiste en un área inalcanzable donde la señal correspondiente a nuestra onda electromagnética queda bloqueada por un obstáculo sólido [19]. Como se ha comentado anteriormente, la capacidad de que una onda pueda atravesar un obstáculo viene determinada por la longitud de onda de la misma, el tamaño del objeto considerado y por el coeficiente de penetración del material del mismo.

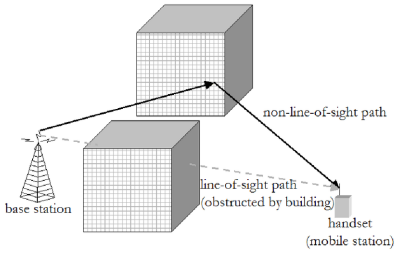


Figura 3.6: Efecto de *shadowing* [5]

Si la antena del receptor está en un área de sombra, como podemos ver en el ejemplo de la figura superior 3.6, donde las estaciones móviles quedan tapados por la cobertura de los edificios dentro de la ciudad, la señal que llega es demasiado débil para recibirla. De nuevo, es importante destacar que el impacto de este fenómeno físico se puede modular en función de las características de los objetos que tengamos, las cuales determinarán si la onda puede atravesarlos o no.

3.3.7 Fenómenos físicos implementados

Para acabar esta sección, comentaremos la manera en que hemos incorporado cada uno de los fenómenos físicos en la implementación de nuestra herramienta, consiguiendo de esta forma resultados más realistas.

En primer lugar, tanto la pérdida de trayectoria en el espacio libre como también la ley de reflexión han sido las más evidentes puesto que, a medida que la onda se propaga por nuestro escenario, iremos aplicando una atenuación en la potencia de la señal acorde a la distancia recorrida y a la frecuencia de la señal. Por otra parte, en caso de que la onda se encuentre con algún obstáculo con capacidad de reflexión o con alguna pared del escenario, aplicaremos la ley de reflexión variando la trayectoria de la onda de acuerdo a esta ley.

En segundo lugar, los fenómenos de refracción y difracción no se pueden apreciar visualmente en nuestra herramienta, pero sí hemos implementados estos dos fenómenos a través de las operaciones que se realizan cuando calculamos las atenuaciones provocadas por los impactos en los obstáculos y paredes ya que, dependiendo de la geometría y material de construcción, se aplicarán en la señal los coeficientes de atenuación adecuados en función de estas características del objeto.

En penúltimo lugar, también hemos implementado el fenómeno de dispersión en nuestro programa considerando el potencial grado de dispersión de los objetos, el cual depende sobre todo del tipo de material que estemos considerando. Para ello, se fija para cada objeto un coeficiente de dispersión en función del material seleccionado por el usuario y luego, dependiendo de dicho coeficiente, se generarán un determinado número de réplicas de la onda con ángulos aleatorios, y atenuaciones determinadas por el material y la frecuencia de la onda. De esta forma, si empleas un material muy dispersivo que contenga, por ejemplo, un coeficiente de dispersión de 0.8 se generarán varias réplicas aleatorias de la onda que habrá que manejar.

Por último, debemos comentar que el efecto de sombreado también está presente en nuestras simulaciones. En caso de que el usuario decida esconder el receptor detrás de obstáculos suficientemente grandes como para que ninguno de los rayos del haz de la onda lo puedan alcanzar, la herramienta comunicará al usuario que la potencia de la señal recibida es insuficiente para tener la calidad requerida por el receptor. Sin embargo, la intensidad de este fenómeno viene determinada por otro coeficiente con el que también se parametrizan los objetos, que hemos denominado coeficiente de penetración y que determina el grado de atenuación que sufren las señales cuando atraviesan un objeto. Si este coeficientes toma un valor próximo a 0, la señal simplemente se verá reflejada mientras que, si este coeficiente toma valores próximos a 1, la señal podrá atravesar el objeto con una atenuación muy pequeña.

3.4 Unidades métricas

En nuestra aplicación, utilizaremos distintas métricas para medir distintos parámetros importantes en el modelado de los escenarios de comunicación y en la presentación de los resultados. Entre ellos podemos destacar:

- Las distancias dentro de nuestro escenario se medirán siempre en decímetros (dm), tanto a la hora de diseñar nuestro escenario como en la visualización de las distancias recorridas por cada rayo del haz de nuestra onda. Los cálculos que realicemos internamente cumplirán con el estándar de cada fórmula matemática obteniendo de esta forma resultados razonables. Sin embargo, una vez obtenidos los resultados de la simulación, se realizarán las conversiones necesarias y se volverá a trabajar con estas unidades de longitud para ser consecuentes con el diseño que haya elaborado el usuario y obtener unos resultados acordes. Tal y como podemos ver en la imagen global de la herramienta de la figura 1.1 del primer capítulo, nuestro plano dispondrá de una cuadrícula, donde cada cuadrado se corresponde con un tamaño en dm de 10x10. De acuerdo al tamaño elegido para el lienzo que representa el escenario, el plano tendrá un tamaño máximo de 670x310 (dm), espacio suficiente para realizar simulaciones *indoor*. Durante la ejecución de nuestro programa, en ningún momento el usuario deberá trabajar en píxeles,

por lo que no consideramos necesario representarlos ni hacer mención explícita a ellos ya que estos serán transparentes a nivel de usuario.

- También hacemos uso de otras unidades de medida, como los grados, para especificar los ángulos de salida de los diferentes rayos del haz de la onda que se genera desde el emisor, así como también para determinar los ángulos de incidencia y reflexión/dispersión de los rayos que colisionan en los diferentes obstáculos y paredes del escenario.

- Otra de las métricas más interesantes que debemos explicar por su relevancia en trabajos sobre ondas electromagnéticas es el decibelio-milivatio (dBm). Esta unidad de medida relaciona la potencia de una señal que podemos expresar en decibelios (dB) relativa a un milivatio (mW) [20]. Por ejemplo, 15 dBm de potencia son igual a 32mW, potencia típica de transmisión de WiFi en portátiles.

  De esta forma, por un lado tenemos la fórmula que expresa la relación entre dos potencias en escala logarítmica en decibelios:

  $$P_{[\text{dB}]} = 10\log_{10}(P_1/P_2), \tag{3.4}$$

  donde P_1 y P_2 son las potencias de las dos señales que se comparan expresadas en mW.

  En segundo lugar, tenemos la fórmula que relaciona el decibelio-milivatio (dBm) con la potencia de una señal en milivatios (mW). Supongamos que una señal tiene un nivel de potencia P expresada en mW. Entonces, la potencia de la señal expresada en dBm se obtendrá directamente de la siguiente manera:

  $$S_{[\text{dBm}]} = 10\log_{10}(P). \tag{3.5}$$

  Es importante destacar que, si la potencia de la señal está expresada en cualquier otra unidad, es preciso convertirla primero a mW antes de aplicar la fórmula anterior. En el caso de que a alguien le extrañe que nuestras simulaciones puedan dar algún resultado negativo en las potencias recibidas, debemos comentar que una señal de 1 mW tiene un nivel de 0 dBm y que, por tanto, las señales más débiles que 1 mW tienen valores de dBm negativos. Por el contrario, las señales con un nivel de potencia mayor a 1 mW, tendrán valores de dBm positivos.

  Esta segunda métrica es empleada en nuestra herramienta para representar la potencia emitida por nuestro transmisor (tx), el umbral de captación (sensibilidad) de nuestra antena receptora y los resultados de la simulación referentes a la potencia recibida (rx) y a las atenuaciones sufridas por parte de la onda durante su propagación.

- Por último, utilizaremos el megahercio (MHz) para representar la frecuencia de las ondas electromagnéticas que generamos desde el emisor.

Capítulo 4

Fundamentos tecnológicos

Este capítulo consistirá en describir las diferentes tecnologías que hemos utilizado durante el desarrollo de nuestro proyecto para cada una de las tareas que debemos abordar. Para ello, explicaremos primero el lenguaje de programación empleado, en concreto Python 3.0, junto con las diferentes librerías complementarias añadidas a nuestro código para elaborar algunas funciones. Otra tecnología que hemos escogido es Tkinter, usada para la implementación de la interfaz gráfica de nuestra aplicación. Por último, se explicarán también las herramientas necesarias para gestionar diferentes tareas como, por ejemplo, la redacción de la memoria o la gestión de las versiones del proyecto.

4.1 Python 3.0

Una de las primeras decisiones que hemos tomado a la hora de plantear nuestro proyecto fue qué lenguaje de programación se iba a utilizar para el desarrollo de nuestra aplicación. En nuestro caso, tras barajar varias posibilidades, optamos por Python, un lenguaje de programación que actualmente se encuentra entre los más conocidos y usados por los desarrolladores de todo el mundo [21].

4.1.1 Características

Entre sus numerosos atributos, podemos destacar que Python es un lenguaje multiparadigma, lo que significa que se pueden adoptar diferentes estilos de programación como, por ejemplo, la orientación a objetos, programación imperativa y, en algunos casos, programación funcional. La ventaja principal de esta propiedad es que, al utilizar un lenguaje de este tipo, podremos acceder a las diferentes herramientas de cada uno de los paradigmas que admita, consiguiendo solucionar una mayor cantidad de problemas y aportando soluciones más complejas y eficientes.

Otra característica que observamos de Python es la portabilidad del mismo hacia diferentes sistemas operativos. Estamos ante un lenguaje interpretado, lo que le permite ser independiente de la máquina donde se esté ejecutando, siempre que el código fuente de nuestro programa no contenga instrucciones propias de un procesador en concreto y se utilicen llamadas a funciones que cualquier intérprete pueda conocer. Esta propiedad en concreto nos permite generar un software multiplataforma, el cual podrá ser ejecutado en diferentes equipos independientemente del sistema operativo que utilice el usuario.

Por último, debemos comentar dos características más sobre el lenguaje escogido para nuestra aplicación. Por un lado, Python es un lenguaje que permite usar tipado dinámico, lo que le convierte en una opción más interactiva para el desarrollador y nos permite elaborar un código más flexible en cuanto a la asignación de distintos tipos de valores a las variables. Si logramos gestionar adecuadamente dichas variables, podremos aportar agilidad a nuestro programa reduciendo los tiempos de compilación. Por otro lado y no menos importante, debemos comentar los beneficios a la hora de desarrollar un software de código abierto, es decir, una aplicación cuyo código fuente esté disponible para los usuarios y desarrolladores que la usen, con el objetivo de que estos también puedan aportar nuevas ideas y agregar aquellas funciones que crean convenientes según las necesidades que identifiquen. De esta forma, se pueden ir optimizando diferentes partes del software, alcanzando así un producto de mayor calidad. También conseguiremos mantener actualizada nuestra herramienta al gusto de los clientes y con una mayor cobertura de sus necesidades. Esta es probablemente la razón principal por la que las empresas eligen software de código abierto, a parte de ser a menudo más barato o libre de cargos, por lo que estará disponible para todo tipo de empresas independientemente del presupuesto.

A parte, hemos escogido Python en nuestro proyecto porque es un lenguaje de programación ideal para trabajar de forma eficiente con vectores o matrices y nos da la posibilidad de incluir un gran número de librerías útiles para la implementación de ciertas partes de la herramienta.

4.1.2 Tipos y estructuras de datos

En los sucesivos apartados de esta sección, procederemos a describir en profundidad los diferentes tipos y estructuras de datos utilizadas en el desarrollo de nuestro código para manejar toda la información, crear los diferentes elementos necesarios en la herramienta e implementar los distintos casos de uso.

Módulos:

Los módulos son archivos con extensión ".py" que albergan un conjunto de funciones, variables y clases que pueden usar o ser usados por otros módulos a su vez. El uso de estos módulos nos aporta la ventaja de poder reutilizar código y organizarlo mejor.

En nuestro caso, creemos que es suficiente el uso de diferentes clases para organizar correctamente nuestro código fuente y, por ello hemos decidido albergar todo nuestro código fuente en un único módulo, llamado "herramienta.py". Sin embargo, podemos decir que utilizamos los módulos para otro objetivo diferente ya que, en nuestro caso a través de las sentencias "from" e "import", podemos importar algunos atributos o incluso todo el contenido de aquellos módulos que estemos interesados en añadir a nuestro fichero para poder acceder a sus utilidades desde nuestro código sin la necesidad de programarlas de nuevo, es decir, nos permite mejorar nuestro programa añadiendo nuevas funcionalidades a partir de archivos externos. En nuestro caso, se ha utilizado, por ejemplo, el módulo de matemáticas "math" para realizar todo tipo de cálculos. Una vez añadido este módulo, podemos utilizar el valor de la constante "pi" o la función "sqrt" que nos devuelve la raíz cuadrada de cualquier número.

Funciones:

Las funciones son bloques de código compuestos por una secuencia de instrucciones que encadenamos con el objetivo de que realicen una tarea u operación en concreto. Estas funciones llevan asociado un nombre para identificarlas y, en algunos casos, será necesario aportarles ciertos argumentos cuando sean utilizadas. El uso de estas funciones está muy ligado a la programación estructurada y nos aportan grandes ventajas a la hora de elaborar nuestro código. Por ejemplo, nos permiten dividir nuestro programa complejo en segmentos más simples y fáciles de manipular tanto en la programación como en la depuración. Otro de los beneficios de emplear funciones es la reutilización de las mismas, ya que en muchos casos nos libran de emplear código redundante en situaciones semejantes.

```
1 def mi_funcion(self, param1, param2):
2     param3 = param1 + param2
3     return param3
```

Listing 4.1: Ejemplo de función con parámetros y valor de retorno

Operadores:

Los operadores son símbolos presentes en todos los lenguajes de programación que constituyen herramientas básicas que permiten a los desarrolladores realizar distintas operaciones sobre las variables de nuestro programa. Existen distintos tipos de operadores, sin embargo, para este tipo de proyectos los más comunes son los siguientes:

- Operadores de asignación: Este tipo de operadores nos permiten asignar valores a las variables de nuestro programa. También existen otras variantes de este operador que, a parte de realizar su función de asignación, pueden realizar operaciones aritméticas.
- Operadores aritméticos: Podemos emplear este tipo de operadores para realizar operaciones aritméticas básicas entre las variables de nuestro programa como, por ejemplo, sumas, restas, divisiones, ..., todo con el objetivo de obtener un resultado final.

- Operadores relacionales: Por último, estos operadores sirven para comparar la igualdad entre los valores de nuestras variables y dar paso a realizar operaciones condicionales según el resultado booleano que obtengamos.

```
# Operador de asignación
>>> valor = 5; valor += 10; valor
15
# Operador aritmético
>>> 2 * 6
12
#Operador relacional
>>> 5 != 3
True
```

Listing 4.2: Ejemplo de operadores

Tipos de datos:

En cualquier lenguaje de programación de alto nivel se manejan tipos de datos, los cuales definen un conjunto de valores que tienen una serie de características y propiedades determinadas. Todo valor que pueda ser asignado a una variable de nuestro código tiene asociado un tipo de dato dependiendo del objetivo que deba cumplir dicha variable en nuestra implementación. Por ello, podemos afirmar que un tipo de dato establece qué valores puede tomar una variable y qué operaciones se pueden realizar sobre la misma.

En primer lugar, contamos con una serie de tipos de datos básicos que son muy útiles para operar con ellos en situaciones sencillas. En nuestro caso, Python 3.0 nos permite trabajar tanto con datos booleanos, como con datos numéricos y cadenas de caracteres.

Debido a los conceptos matemáticos y físicos con los que tenemos que trabajar en este proyecto, en nuestro código predominan los tipos de datos más asociados a la necesidad de realizar cálculos y operaciones numéricas frente al resto. Sin embargo, también observamos situaciones en las que debemos hacer uso de las cadenas de caracteres. Por ello, en los dos siguientes apartados explicaremos en profundidad en qué consisten ambos tipos de datos.

Datos numéricos:

Python define tres tipos de datos numéricos y básicos: enteros, números de punto flotante y los números complejos. De estas tres opciones, nosotros hemos utilizados las dos primeras para realizar los cálculos necesarios y, de esa forma, generar los resultados finales que desea obtener el usuario a través de nuestra aplicación.

- Números enteros: Este tipo de datos se representan en el lenguaje de programación como “int” y comprende el conjunto de todos los números enteros. Sin embargo, como dicho conjunto es infinito, en Python el conjunto está limitado realmente por el número

de bits utilizados para representar el tipo de dato que en la mayoría de los casos es más que suficiente para que el usuario pueda realizar su objetivo.

- Números de punto flotante: Por el contrario, este otro tipo de dato es conocido en Python con el término "float" y se emplea para representar cualquier número real. A diferencia de los anteriores, los números de punto flotante son utilizados en aquellas situaciones donde se requiere una aproximación matemática lo más precisa posible.

- En ambos casos, se emplea una doble precisión (64 bits) para representar el tipo de dato.

En nuestro proyecto, hemos empleamos ambos tipos de datos, ya que consideramos que los dos aportan funcionalidades diferentes y muy útiles. En el primer caso, los números enteros son empleados en situaciones donde se requiera el uso de algún tipo de contador como, por ejemplo, para conocer el número de obstáculos desplegados en el plano. También se puede dar la posibilidad de necesitar este tipo de dato en alguna comparativa de valores y que ambos sean de tipo entero. En segundo lugar, tenemos los números de punto flotante que son utilizados tanto para representar las distintas magnitudes de nuestras ondas, como también para cada una de las operaciones matemáticas que realizamos para obtener resultados precisos en las simulaciones.

Cadenas de caracteres:

Otro tipo de dato básico e imprescindible de Python son las cadenas de caracteres, las cuales representan distintas secuencias de caracteres de longitud variable. Este tipo de dato es conocido como *string* y se representa con la palabra clave "str" dentro del código fuente de nuestro programa. A continuación, vemos un sencillo ejemplo de como inicializar variables con este tipo de dato.

```
1 # Cadenas de caracteres
2 mi_string = "Hola"
3 mi_otroString = "25"
```

Listing 4.3: Ejemplo de inicialización de cadenas de caracteres

Llegados a este punto, resulta interesante explicar algunas de las funciones más interesantes que nos permiten trabajar cómodamente con los distintos tipos de datos, y que aportan sustento y flexibilidad a nuestro programa.

En primer lugar, debemos conocer en todo momento el tipo de datos que albergan las variables de nuestro programa para poder trabajar correctamente con ellas y no cometer errores de tipado. Para ello, hacemos uso de la función "type" que nos devuelve el tipo de dato del objeto que le pasemos como parámetro. Una vez que ya conocemos de qué tipo son las variables, ya sabremos qué operaciones podremos aplicar sobre ellas sin problema.

Cuando creamos las variables de nuestro programa les asignamos un tipo de dato concreto porque tenemos previsto realizar ciertas operaciones sobre ellas. Sin embargo, también existe la posibilidad de que algunas de estas variables de nuestro programa tengan que cambiar su tipo de dato en algún punto de la ejecución, ya que nos surge la necesidad de realizar otro tipo de operaciones. Esta solución es posible en Python gracias a algunas funciones que nos permiten cambiar de un tipo de dato a otro. Por una parte, tenemos "str" que, según el objeto que le pasamos por parámetro, nos devuelve su representación como cadena de caracteres. En segundo lugar, tenemos las funciones "int" y "float" que devuelven un número entero o punto flotante, respectivamente, a partir del número o secuencia de caracteres que le pasemos.

```
>>> type(numero_entero)
<class 'int'>
>>> type(numero_puntoFlotante)
<class 'float'>
>>> type(mi_string)
<class 'str'>

>>> type(mi_otroString)
<class 'str'>
c
>>> nuevo_numero = int(mi_otroString)
>>> type(nuevo_numero)
<class 'int'>
```

Listing 4.4: Ejemplo de operaciones con tipos de datos

A pesar de contar con una gran variedad de tipos de datos básicos, también podemos hacer uso de los tipos de datos complejos, entre los que destacan las listas que comentamos en el siguiente apartado.

Listas:

En determinadas situaciones de nuestro programa, debemos trabajar con varios tipos de datos diferentes para solucionar un caso en particular cuyas condiciones varían según la iteración del bucle en la que nos encontremos. En ese caso, debemos hacer uso de las listas. Las listas son un tipo de dato compuesto que se usan para almacenar conjuntos de elementos relacionados pero que pueden ser de diferentes tipos. A parte de poder ser heterogéneas, las listas son dinámicas, permitiéndonos añadir o eliminar elementos de ellas después de haber sido creadas. Por otra parte, estas listas también tienen la capacidad de ser mutables, por lo que nos permiten modificar sus elementos en un determinado momento. Gracias a todas estas características, podemos decir que las listas son la estructura de datos más versátil del lenguaje, siendo muy útiles para la programación con Python.

Para poder manejar correctamente las listas y aprovechar bien sus características, disponemos de una serie de operaciones que nos permiten trabajar cómodamente con ellas. A

continuación, veremos un ejemplo de lista y mostraremos algunas de las operaciones más utilizadas con ellas.

```
>> mi_lista = ['string', 10, 3.5]
>> mi_lista
['string', 10, 3.5]
>> mi_lista[2]
3.5
>>len(mi_lista)
3
>> mi_lista[2]='word'
>> mi_lista
['string', 10, 'word']
>> mi_lista.append(4.2)
>> mi_lista
['string', 10, 'word', 4.2]
>> mi_lista.pop(2)
['string', 10, 4.2]
```

Listing 4.5: Ejemplo de operaciones con listas

4.1.3 Programación Orientada a Objetos

La Programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos" como base para el diseño del software, en lugar de funciones y lógica sobre esas funciones [22].

En el desarrollo de nuestra aplicación contaremos con este paradigma, puesto que su uso tiene grandes ventajas tanto a nivel colaborativo como organizativo. El uso de este paradigma para el desarrollo de nuestra herramienta nos permite percibir mejor cuales son los distintos componentes que la forman y poder tratarlos de manera independiente, con el objetivo de dividir la aplicación general en partes más simples y sencillas de implementar. Por tanto, este método es perfecto para abordar proyectos grandes que se quieran mantener actualizados y con un correcto mantenimiento.

Otro de los beneficios que conlleva utilizar este paradigma sucede cuando los desarrolladores proponen aumentar el número de elementos o sumar nuevas funcionalidades al programa. En este caso, como los elementos se estructuran y mantienen de forma independiente desde un principio, añadir nuevas opciones no supone ningún reto, puesto que estas no influirán en el resto. Este factor se conoce como escalabilidad.

A continuación, explicaremos los cuatro fundamentos sobre los cuales se sostiene la programación orientada a objetos.

Encapsulamiento:

El encapsulamiento es la primera característica que presenta la programación orientada a objetos y es el principal motivo por el cual usamos este paradigma en nuestro trabajo. Esta propiedad nos permite aislar la implementación y el estado de cada elemento de nuestra aplicación de forma independiente mediante límites conocidos como "clases". Por tanto, una vez creada la clase de un objeto en concreto, los demás elementos del programa no tendrán acceso a los datos ni métodos de dicha clase. Sin embargo, si esa clase contiene funciones o métodos públicos, entonces el resto de objetos podrán utilizarlos para las tareas asignadas. Esta propiedad de la POO nos permite ocultar mejor la información contenida en cada uno de los elementos, y así mantener alta la seguridad del programa y evitar la corrupción de los datos. A continuación, explicaremos más en profundidad en que consisten las clases ya que han sido un factor importante en la organización e implementación de nuestro código.

Clases y objetos:

Una clase es una especie de "plantilla" que contiene los atributos y métodos que definen un tipo de objeto de nuestro programa. Estas plantillas nos permiten crear fácilmente instancias de los distintos elementos de nuestra aplicación, con el objetivo de que el usuario pueda tanto leer como recuperar los datos y métodos de dichos objetos, y trabajar con ellos.

Un objeto es, por tanto, una entidad provista de unas propiedades (atributos) que reflejan las características del propio objeto y cuyos valores le aportan el estado actual en el que se encuentra dicho objeto y lo diferencian del resto de objetos de la misma clase. También dispone de unas funcionalidades (métodos) que definen el comportamiento del objeto ante ciertos eventos que sucedan en la aplicación. Estos métodos se pueden interpretar como algoritmos que se ejecutan cuando el objeto correspondiente recibe un mensaje por parte del sistema, y pueden verse como las acciones que un objeto puede realizar para producir cambios, no solo en sus propiedades, si no también para generar nuevos eventos que afecten a otros objetos de la aplicación.

En el siguiente recuadro, podemos apreciar un ejemplo de como se definiría una clase con sus atributos y métodos. Se muestra también como crearíamos un objeto a partir de dicha clase y como llamaríamos a sus funcionalidades correspondientes.

```
1  class Humano():
2      def __init__(self, nombre, edad):
3          #Atributos de la clase
4          self.nombre = nombre
5          self.edad = edad
6
7      #Métodos de la clase
8      def Hablar(self):
9          print("Hola")
10
```

```
    def Caminar(self):
        print("Estoy caminando")

#Creamos la instancia de una clase (objeto) y llamamos a un método
>> persona = Humano("Manolo",23)

>> persona.Hablar()
Hola
```

Listing 4.6: Ejemplo de clase

Abstracción:

La abstracción es la propiedad por la cual los distintos elementos del programa no revelan sus métodos internos a excepción de que sean necesarios para el correcto funcionamiento de otro elemento. Esta característica nos permite ocultar cualquier código de implementación innecesario, aumentando así la seguridad del programa y evitar la corrupción de datos.

Herencia:

La herencia nos permite asignar relaciones entre los elementos del programa de tal forma que podemos mantener una clase como principal, la cual es conocida como "superclase" y que será común a otras subclases que heredarán sus atributos y métodos. Es decir, esta propiedad nos da la posibilidad de reutilizar la lógica común. Esta característica de la POO nos obliga a realizar un análisis más profundo de los elementos que conforman nuestro programa pero también nos aporta una programación más precisa.

Polimorfismo:

El polimorfismo es la propiedad de la POO por la cual los distintos elementos de un programa pueden adoptar más de una forma según el contexto. Esto se resume en que métodos diferentes que pertenezcan a distintos elementos u objetos del programa pueden tener el mismo nombre asociado. A pesar de compartir el mismo nombre, cuando se produzca la llamada a dicho método se detecta el objeto en cuestión que se está empleando y así el método invocado cumplirá con el comportamiento esperado.

4.2 Tkinter

En esta sección, explicaremos en qué consiste la herramienta Tkinter y como se convirtió en un punto a favor a la hora de escoger el lenguaje de programación de nuestro proyecto. Tkinter [23] es un *binding* de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python y podríamos decir que es un estándar para la elaboración de interfaces gráficas de usuario (GUI). Por otra parte, destacar que es un paquete que viene ya instalado con Microsoft Windows, aunque también está disponible en la mayoría de plataformas de Unix y macOS.

4.2.1 ¿En que consiste Tcl/Tk?

Tcl/Tk es una biblioteca compuesta por tres módulos muy bien diferenciados, cada uno con sus propias funcionalidades y características, lo que le aporta consistencia y variedad al mismo tiempo. A continuación, describiremos cada uno de estos tres módulos:

- Tcl: Es un lenguaje de programación interpretado dinámico muy poderoso pero fácil de aprender, al igual que Python. Este lenguaje es adecuado para una amplia gama de usos. En algunos casos, se puede usar por sí solo como un lenguaje de programación para aplicaciones web o de escritorio, aunque generalmente se suele integrar en aplicaciones desarrolladas en C como un motor de secuencias de comandos o como una interfaz para el conjunto de herramientas Tk.

- Tk: Es un conjunto de herramientas para crear interfaces gráficas de usuario que lleva el desarrollo de aplicaciones de escritorio a un nivel más alto, ya que nos permite agregar comandos personalizados para construir y manipular *widgets* de GUI.

- Themed Tk (Ttk): Por el contrario, a diferencia del módulo anterior, Themed Tk es una nueva versión que contiene herramientas más actualizadas, pero más complejas de implementar y configurar.

El hecho de que esta biblioteca gráfica viniera incluida en nuestro sistema operativo nos permite ahorrar tiempo de búsqueda y centrarnos directamente en profundizar en la documentación de Tkinter y observar todas las funciones y posibilidades que nos ofrece. Una vez realizada la investigación pertinente, vemos que esta biblioteca en concreto dispone de ciertos elementos que son realmente útiles para el desarrollo de nuestra aplicación, puesto que cumplen algunos de los requisitos y objetivos que deseamos alcanzar.

En concreto, para nuestro proyecto hemos decidido emplear únicamente Tk, que es el módulo principal de la biblioteca gráfica de Tkinter. Tal y como hemos comentado anteriormente, este módulo nos aporta las funciones necesarias para elaborar y personalizar correctamente nuestra herramienta. Esta herramienta dispone de una gran cantidad de *widgets* muy personalizables que nos aportan el medio para crear una interfaz con un aspecto simple e intuitivo, ayudando al usuario a navegar con mayor fluidez por la herramienta y así comprender mejor nuestro software y diferenciar claramente cada una de las partes y funcionalidades del sistema. En algunas ocasiones, construir un software con elementos sobrecargados, le aporta a la aplicación una cierta complejidad visual que puede derivar en confusiones por parte de los usuarios.

La razón principal por la cual escogimos Tkinter frente otras alternativas es porque esta herramienta dispone de *widgets* muy útiles para recrear un simulador de propagación de

ondas, como por ejemplo, el lienzo "Canvas" y a mayores disponemos de un conjunto de operaciones como "tagbind" que nos permite unir los eventos que ocurran dentro del plano a las distintas funciones que implementemos en Python.

A continuación, procederemos a describir algunos de los elementos de Tkinter que consideramos más importantes y que hemos escogido para elaborar la interfaz de usuario.

4.2.2 Widgets

Un *widget* es un componente visual que puede ser reutilizado por los programadores en el desarrollo de sus aplicaciones con el objetivo de combinarlos entre sí para construir una GUI compleja y elaborada. Cada uno de estos *widgets* se representa dentro del código de implementación como un objeto de Python instanciado a partir de una clase.

Entre estos *widgets* existe una jerarquía encargada de organizar mejor los diferentes elementos dentro de la interfaz gráfica y de mantener la relación entre los distintos componentes que la construyen. De esta forma, unos elementos pueden contener a otros que, a su vez, están incluidos en una ventana principal conocida como raíz. Para que un *widget* dependa de otro, simplemente debemos pasarle como primer parámetro el *widget* principal al cual pertenecerá.

Una vez escogidos los diferentes *widgets* que vamos a utilizar para elaborar nuestra interfaz gráfica, decidimos el orden y la distribución de los mismos dentro del diseño de la vista de nuestra aplicación. Tras todo ese proceso, lo último que nos falta por hacer para acabar de configurar los *widgets* será establecer las opciones de los distintos elementos empleados, para cambiar sus aspectos y sus comportamientos, con el objetivo de darles una apariencia más personalizada.

Ahora que sabemos en qué consisten y para qué sirven, a continuación mostraremos distintos tipos de *widgets* que hemos empleado en nuestra implementación y que nos han servido de mucha ayuda.

***Widgets* más comunes:**

En primer lugar, tenemos estos cuatro tipos de *widgets* que son los que predominan en el diseño de nuestra herramienta y que representan el primer paso para conseguir interactuar directamente con el usuario.

Botones y *Radiobutton*:

Por un lado, tenemos el botón que es por lo general el elemento de interacción más común en una aplicación con interfaz gráfica. Este elemento está representado por un recuadro que contiene un texto o una imagen representativa de la función que realiza y que, al ser presionado por el usuario, ejecuta una operación definida. En Tkinter, este *widget* pertenece a la clase *tk.Button* y, desde el punto de vista del código, su utilización es muy sencilla.

```
boton = ttk.Button(text="¡Hola, mundo!")
```

Listing 4.7: Instancia de un botón

En segundo lugar, dentro de los *widgets* más comunes tenemos un derivado del anterior que se conoce como *radiobutton*. Este elemento implementa un botón pero con múltiples opciones, es decir, es una forma de ofrecer al usuario muchas elecciones posibles a través de un único *widget*. Por ejemplo, en nuestro proyecto es utilizado para ofrecer distintas opciones acerca de la forma o material de los obstáculos. La sintaxis de este elemento es muy simple, tal y como vemos a continuación, ya que la única condición es que todos los botones del *widget* deben pertenecer a la misma variable.

```
boton=ttk.Radiobutton(frame, image, text, variable, value, options)
```

Listing 4.8: Instancia de un *radiobutton*

Entrada:

En tercer lugar, tenemos las entradas de texto, que es otro *widget* básico de Tkinter utilizado para obtener cadenas de texto introducidas por el usuario a través de la interfaz. Estos campos de texto generalmente se usan para que el usuario pueda escribir valores de un cierto tipo de dato y comprendidos dentro de un rango de valores. Sin embargo, gracias a los métodos de los que dispone este elemento podemos darle otros usos como, por ejemplo, bloquearlos para que el usuario no pueda escribir en ellos y utilizarlos para mostrar los distintos parámetros que el cliente va escogiendo a lo largo de la ejecución y presentar los resultados finales de nuestras simulaciones. Este *widget* está representado a través de la clase *tk.Entry* y se instancia de la siguiente manera.

```
entry = tk.Entry(root)
```

Listing 4.9: Instancia de una Entrada de texto

Etiqueta:

En cuarto lugar, tenemos las denominadas etiquetas que son otro de los elementos fundamentales en una interfaz gráfica. Este tipo de *widget* nos permite mantener informados a nuestros usuarios para poder guiarlos cómodamente a través de nuestra aplicación, mostrándoles cuales son las distintas secciones que conforman la herramienta, así como las distintas utilidades y elementos disponibles a la hora de utilizar el programa. El método para conseguir esto consiste en crear cuadros de texto mediante los cuales se aporta toda la información necesaria para que el cliente pueda manejarse con fluidez. En Tkinter, este *widget* pertenece a la clase *tk.Label* y dispone de una gran variedad de opciones de configuración.

```
1 EtiquetaEjemplo = tk.Label(root, text="Esto es una Etiqueta")
```

Listing 4.10: Instancia de una Etiqueta

Frames:

Figura 4.1: Delimitando nuestra interfaz gráfica en secciones mediante *frames*

Otro componente muy utilizado es el *frame*, que es un tipo de *widget* que se utiliza como contenedor para albergar otros elementos de nuestra interfaz gráfica y distribuir mejor la ventana principal o las ventanas secundarias que se generen durante la ejecución de nuestro programa. Este *widget* es la herramienta más importante en el proceso de agrupar y organizar los distintos elementos de nuestra vista con el objetivo de crear una interfaz de usuario amigable y cómoda para el cliente.

Su uso es muy sencillo ya que simplemente consiste en crear distintas áreas rectangulares en las ventanas de nuestra aplicación para organizar mejor el diseño, como podemos observar en la figura 4.1. Haciendo uso del método "config" podemos modificar las características de estos marcos, dotándolos de diferentes tamaños, colores o comportamientos con el fin de diferenciar las distintas zonas de nuestra herramienta.

Canvas:

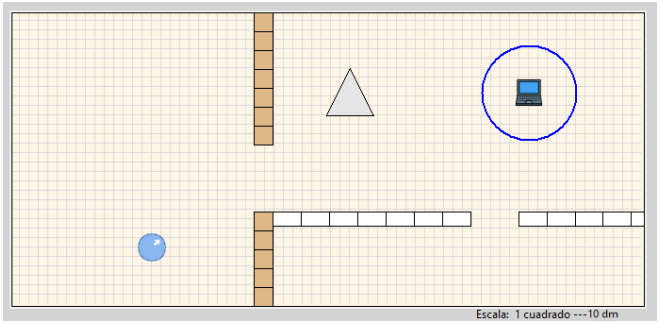


Figura 4.2: Utilización del *widget Canvas* como lienzo

En último lugar, tenemos el *widget* más importante y que se convirtió en la razón principal por la cual escogimos Tkinter para el desarrollo de nuestra interfaz gráfica, ya que este elemento nos aporta el factor visual diferencial que queríamos conseguir. El *widget Canvas* nos brinda la posibilidad de crear una superficie rectangular destinada a cumplir el papel de lienzo en nuestro programa sobre el que crear el escenario de simulación. Esta herramienta nos ofrece las funciones gráficas y creativas necesarias para poder plasmar a través de nuestro programa escenarios *indoor* reales o crear nuestros propios diseños tan complejos como queramos. En el ejemplo de la figura 4.2, vemos como las propiedades de *Canvas* nos permiten modificar el plano haciendo uso tanto de imágenes como de figuras geométricas para simular los distintos elementos que influyen en nuestra simulación.

Al igual que el resto de herramientas que forman parte de Tkinter y que hemos utilizado, podemos decir que emplear la clase *tk.Canvas* es relativamente sencillo. En primer lugar, debemos crear una instancia de dicha clase para formar el plano en cuestión que utilizaremos como lienzo de nuestro programa. Para ello, llamaremos a la función que mostramos en el siguiente recuadro. Como se puede observar, la sintaxis es muy sencilla ya que simplemente debemos facilitarle los parámetros que se necesitan para la configuración del plano.

```
1 Lienzo=Canvas(frame, width, height, background, bd, relief)
```

Listing 4.11: Funciones para crear y posicionar el lienzo

Como podemos ver en esta función, el dato obligatorio y que debemos proporcionarle como información principal a dicha función es el identificador de la ventana o *frame* donde queremos añadir el plano. En nuestro caso, será uno de los *frames* creados anteriormente y que hemos utilizado para diseñar y distribuir mejor los elementos de la aplicación, tal y como se ha mostrado en la figura 4.1. Adicionalmente, le pasaremos a la función otras opciones de configuración para ajustarlo a nuestra necesidades y darle un aspecto más estético. Una vez creado el *widget canvas*, trabajar con él es muy cómodo ya que disponemos de una serie de funciones con las que crearemos los elementos del escenario y los situaremos en nuestro plano.

Representación de obstáculos:

En primer lugar, tenemos las funciones necesarias para crear las distintas figuras geométricas que usaremos en nuestro plano para representar los obstáculos. Existen varios tipos de funciones con las cuales se pueden dibujar figuras geométricas en *Canvas*. Aunque algunas de estas figuras tienen sus propias funciones, nosotros usaremos únicamente dos funciones para generar los diferentes obstáculos en nuestro escenario.

De las dos funciones que observamos en el siguiente recuadro, la primera resulta ser la más genérica de todas ya que nos permite crear cualquier tipo de polígono, desde un mínimo de tres lados hasta el número máximo que queramos. Por ello, es la más utilizada en nuestra aplicación

a la hora de dibujar los distintos obstáculos, ya que solo con esta función podemos crear todo tipo de figuras geométricas: triángulos, cuadrados, rombos, etc. Para crear correctamente nuestras figuras, debemos pasarle a dicha función como primer parámetro una lista con todas y cada una de las coordenadas de los distintos vértices que contenga. Luego, simplemente nos queda la posibilidad de añadir algunas opciones adicionales para modificar su aspecto, como el color del relleno, el color de los lados o su grosor.

```
Forma1=Lienzo.create_polygon(coordinates,outline,fill,width,tags)
Forma2=Lienzo.create_oval(coordinates,outline,fill,width,tags)
```

Listing 4.12: Funciones para dibujar figuras geométricas en *Canvas*

Por otra parte, tenemos la segunda función que nos permite crear un círculo o una elipse en las coordenadas dadas. Su funcionamiento es idéntico a la función anterior, sin embargo, la única diferencia es que solo toma dos pares de coordenadas: la esquina superior izquierda y la esquina inferior derecha del rectángulo delimitador del círculo que queramos dibujar. Esta función es fundamental en la implementación de nuestra aplicación ya que nos permite crear obstáculos con forma circular, pero también la utilizaremos para delimitar visualmente el área de influencia de nuestra antena receptora.

Representación de dispositivos:

Para completar nuestro escenario y poder comenzar la simulación, necesitamos añadir los dos últimos elementos implicados en la comunicación: el transmisor con su antena emisora y el receptor con su correspondiente antena receptora. Para ello, hemos buscado una función muy útil de la clase *tk.Canvas*, la cual se llama "create_image", que nos permitirá añadir imágenes a nuestro plano y de esta forma representar estos dispositivos de una manera más fiel a la realidad. El uso de esta función lo podemos observar en el siguiente recuadro junto con el proceso previo que debemos realizar para trabajar correctamente con ella y que explicaremos a continuación.

```
ImagenReceptor = Image.open('portatil.png')
ImagenReceptor = ImagenReceptor.resize((30,30),Image.BOX)
ImagenReceptor = ImageTk.PhotoImage(ImagenReceptor)
NumeroReceptor=Lienzo.create_image(coordinates,image,anchor,tags)
```

Listing 4.13: Funciones para añadir imágenes en *Canvas*

Para empezar, tenemos que entender que esta función debe cumplir unos requisitos en cuanto a los datos que recibe como parámetros para funcionar correctamente. En primer lugar, no podemos pasarle la imagen directamente a este método ya que solo admite como parámetro un objeto de tipo "PhotoImage" que será el dato que le indicará el contenido gráfico en cuestión que deseamos situar sobre el plano. Sin embargo, para instanciar un objeto de esta clase, debemos trabajar en formato GIF y, en nuestro caso, estamos trabajando con formato

PNG, lo que implica que debemos instalar previamente una librería externa para realizar esta conversión que explicaremos más adelante. Una vez que tenemos todo instalado y preparado, comenzamos abriendo la imagen que hayamos escogido desde nuestro código con la primera instrucción que vemos en el recuadro anterior. Una vez tenemos nuestra variable "ImagenReceptor" inicializada con la imagen que deseamos incorporar en el plano, podemos modificar su tamaño para ajustarla a las características deseadas. De esta forma, ya habremos creado una variable que sí acepta la función "PhotoImage" y, por tanto, ahora ya podremos crear nuestra representación del dispositivo receptor en el plano pasándole el objeto resultante como parámetro a nuestra función "create_image".

Representación de la onda:

Una vez que ya sabemos como crear tanto los obstáculos como también los dispositivos de comunicación, nos falta poder representar la onda viajando por nuestro escenario. Para ese caso, hemos encontrado una función perfecta para dibujar las distintas trazadas que representan la onda moviéndose por nuestro escenario.

```
1 Onda=Lienzo.create_line(source,target,fill,width,tags)
```

Listing 4.14: Método para dibujar la onda sobre el lienzo

Como podemos observar en el recuadro anterior, la función que utilizamos para dibujar las trayectorias de la onda mediante rectas es sencilla ya que solo necesita dos parámetros: el punto de origen y el punto de destino de la recta en cada caso. Por tanto, a lo largo de cada simulación básicamente necesitamos calcular en cada momento cual es la dirección que debe tomar la recta y cuales son su nuevo origen y destino. Con esta información, podremos simular una onda avanzando por el plano y colisionando tanto con los obstáculos como también con las paredes.

Es interesante comentar también que esta función tiene otra aplicación muy interesante y estética en nuestra herramienta, puesto que se utiliza para generar la cuadrícula de nuestro plano para facilitar al usuario poder distribuir mejor los espacios y ser más preciso con la colocación de los elementos a la hora recrear o diseñar sus escenarios.

4.2.3 Administrador de geometría

Otro concepto que debemos tratar si hablamos de Tkinter son los administradores de geometría y su importancia en el desarrollo de la aplicación. Estos mecanismos se utilizan a la hora de programar para especificar la posición relativa de todos y cada uno de los *widgets* que usamos para la creación de nuestra interfaz. Existen diferentes administradores de geometría, de los cuales hemos escogido "place" a pesar de que se considera un poco engorroso de usar en comparación con otros, ya que la mayoría pueden tomar la relación cualitativa de los elementos y resolver sus coordenadas para el propio usuario. Por el contrario, con "place" eso no

ocurre ya que somos nosotros mismos los que debemos especificar las coordenadas exactas de la posición donde queremos colocar nuestros *widgets* dentro de otro contenedor cuyo tamaño estará determinado por el tamaño de los elementos que contenga en su interior.

Esta característica de nuestro gestor de geometría implica que el usuario no podrá variar las dimensiones de nuestra herramienta porque el resto de *widgets* no se ajustarían dinámicamente tras el cambio. Este factor nos beneficia en el sentido de que impide que el usuario pueda dimensionar la herramienta y desajustar la estética que hemos escogido para nuestra interfaz y que creímos conveniente al principio del proyecto. Por último, debemos comentar que el uso de un administrador de geometría es obligatorio, ya que sin él los distintos *widgets* que hayamos utilizado no aparecerían en la interfaz de la herramienta. Es muy común cometer el error de omitir el uso de estos mecanismos y luego no comprender porque no se visualizan los elementos que hayamos agregado.

4.2.4 Acoplamiento de variables

Otra característica de Tkinter que hemos empleado en nuestra herramienta trata sobre el valor actual de algunos *widgets*, es decir, como podemos dotar de valor a algunos de estos elementos que hemos usado en nuestra interfaz y luego recuperar ese valor o incluso reconocer si ha variado en el transcurso de la ejecución. La implementación actual de Tkinter nos impide de poder asociar directamente una variable arbitraria a un *widget*. La única manera de poder ligar ambos elementos es si dicha variable deriva de la clase "Variable" definida en Tkinter. En concreto, existen varias subclases útiles de "Variable" que nos permiten dotar a nuestros *widgets* de diferentes tipos de datos. En nuestra implementación, hemos utilizado la subclase "IntVar" para asociar un número entero a cada una de las opciones del *widget radiobutton* que utilizaremos para escoger el material de las paredes y, en los obstáculos, su forma y material.

4.2.5 Enlaces y eventos

En Tkinter también existe un método para vincular *widgets* de nuestra interfaz gráfica a ciertos eventos y, de esta forma, activar una función de devolución cuando el evento suceda. Existen distintos tipos de métodos de los cuales en nuestra implementación hemos escogido "tag_bind" de la clase *tk.Canvas* porque nos permite vincular un controlador de eventos a un elemento o grupo de elementos de nuestro lienzo a través de sus "tags".

```
1 Lienzo.tag_bind("emisor", "<ButtonPress-1>", self.presion_boton)
2 Lienzo.tag_bind("emisor", "<ButtonPress-3>", self.presion_boton)
3 Lienzo.tag_bind("emisor", "<Button1-Motion>", self.mover)
```

Listing 4.15: Como vincular controladores de eventos con *widgets*

En el cuadro anterior, vemos un ejemplo de nuestra implementación donde empleamos este método para conectar el emisor de nuestro lienzo con tres eventos diferentes del ratón. En primer lugar, tenemos el clic izquierdo del ratón que llamará a nuestra función "presion_boton" para recopilar la posición actual del objeto dentro del plano. En segundo lugar, tenemos el botón derecho del ratón que llamará a la misma función pero con un resultado distinto ya que, esta vez, nos permitirá escoger entre dos opciones: editar el objeto o eliminarlo del plano. En último lugar, ligaremos el movimiento del ratón con el clic izquierdo presionado con la función "mover", que nos permite mover los objetos por nuestro lienzo actualizando constantemente su posición.

4.3 Librerías Estándar y Externas

Una de las ventajas que hace destacar a Python frente otros lenguajes de programación es su facilidad de aprendizaje y su flexibilidad para el desarrollo de diversos tipos de software como, por ejemplo, interfaces web o diferentes herramientas de escritorio. Esta característica viene dada por la posibilidad de disponer de una librería estándar que aportan al lenguaje una gran variedad de funciones que hacen que sea uno de los más utilizados en la actualidad. Por ello, vemos que Python no es solo utilizado para introducirse en el mundo de la programación, si no que se aplica en el desarrollo de aplicaciones importantes.

Por otro lado, al ser un lenguaje tan conocido y utilizado en todo el mundo, provoca que sus usuarios quieran mantenerlo fresco y actualizado, desarrollando diferentes librerías externas que puedan aportar nuevas funcionalidades y, de esa forma, perfeccionar la programación con este lenguaje y ayudar a otros desarrolladores a elaborar nuevos y mejores productos.

A continuación, mostraremos las principales librerías externas que hemos seleccionado para el desarrollo de nuestra herramienta, al igual que el módulo principal de la librería estándar usado para numerosos cálculos.

4.3.1 Math

La "librería estándar" de Python está dotada de numerosos módulos, entre los cuales se incluye la librería Math que nos ofrece una gran cantidad de funciones útiles para su uso en el campo de los números reales. A parte de las operaciones básicas de suma, resta, multiplicación y división, este módulo nos aporta operaciones matemáticas que nos servirán de gran ayuda en la realización de nuestros cálculos como, por ejemplo, funciones trigonométricas y logarítmicas, la obtención del valor exacto del número "pi", trabajar con valores absolutos y muchos más casos. En la mayor parte de las operaciones que realizamos en nuestra herramienta, se hace uso de este módulo [24].

4.3.2 PIL/Pillow

Otra de las librerías externas que hemos añadido al código de nuestra herramienta ha sido Python Imaging Library (PIL), una librería gratuita que nos permite editar imágenes directamente desde Python consiguiendo, de esta forma, poder manipular e interactuar con ellas sin hacer uso de ningún software adicional. Una de las grandes ventajas de esta librería es que puede soportar una amplia variedad de formatos, incluyendo los más utilizados como pueden ser GIF, JPEG y PNG. Además, la mayor parte de su código está escrito en C, logrando un buen rendimiento del sistema. Por todo ello, es una librería muy útil y demandada por los usuarios en los casos donde se requieran manipular imágenes [25].

En un principio, esta librería simplemente fue creada para ser compatible hasta la versión 2.7 de Python. Sin embargo, al convertirse en una librería tan utilizada por la comunidad, Alex Clark junto con más colaboradores decidieron desarrollar por su cuenta Pillow, una bifurcación más intuitiva para los usuarios que pretende mantener esta librería actualizada y operativa para que se adapte a las nuevas tecnologías como es en nuestro caso con Python 3.

Esta librería nos ha resultado muy útil en el desarrollo de nuestra herramienta, permitiéndonos manipular diferentes imágenes, mejorando de esta forma la interfaz gráfica y logrando un aspecto más elegante e intuitivo para el usuario, puesto que hemos añadido diferentes iconos a los botones para que el cliente pudiera distinguir mejor cada una de las opciones de nuestra aplicación. Por otra parte, nos ha permitido no solo abrir imágenes y añadirlas, si no también dimensionarlas para cuadrarlas dentro de nuestra interfaz y, en algunos casos, rotar estas imágenes para poder guiar mejor al usuario en algunas situaciones como la dirección de salida de las ondas.

4.4 Herramientas de apoyo

4.4.1 Google Drive

Una de las herramientas de apoyo que hemos utilizado en el desarrollo de nuestra aplicación, en concreto, para almacenar el código fuente de nuestro software y todos aquellos archivos necesarios, fue Google Drive, ya que esta opción también nos permitía realizar un control de las distintas versiones del software que íbamos generando en cada una de las iteraciones de nuestra metodología, donde podemos ver también que partes fueron modificadas respecto a la versión anterior. De esta forma, podemos realizar distintas pruebas en cada una de las iteraciones, manteniendo siempre una versión anterior a la que volver en caso de error. Esta funcionalidad es similar al control de versiones que tenemos en otras herramientas como subversion o git.

4.4.2 Latex+Overleaf

La opción escogida para elaborar correctamente la documentación del proyecto ha sido Latex, ya que es una de las herramientas mejor preparadas para la ocasión. Consiste en un sistema avanzado de composición de textos que permite crear documentos técnicos de alta calidad y que está preparado para ser capaz de maquetar muy bien textos que incluyan formulación matemática. Por otro lado, lo hemos combinado con Overleaf, un editor de texto colaborativo y *online* que no necesita que instalemos nada para poder utilizarlo y que nos permite editar el texto simultáneamente.

4.4.3 Violet UML Editor

Violet es un editor UML muy sencillo de utilizar que nos permite generar todo tipo de diagramas atractivos: diagramas de clases, casos de uso, diagramas de secuencias,etc. Es una herramienta destinada a desarrolladores y estudiantes que necesitan producir diagramas UML de una manera fácil y eficiente.

Capítulo 5

Metodología, recursos y planificación

5.1 Metodología

Antes de comenzar a elaborar un proyecto, es primordial escoger una metodología adecuada puesto que esta nos marcará las pautas que debemos seguir en cada momento con el objetivo de realizar un buen trabajo y cumplir con los requisitos establecidos. En proyectos de un cierto calibre, ya sea por la importancia del mismo o por el valor económico que supone, es fundamental encontrar una metodología apropiada. Una buena metodología debe definir claramente las distintas etapas a seguir para completar el proyecto, permitiéndonos realizar en cada paso las distintas tareas establecidas en el proceso de planificación, y habilitar un mecanismo para asegurar que el trabajo en cada etapa se ha realizado correctamente antes de pasar al siguiente.

Para nuestra herramienta hemos escogido una **metodología basada en un desarrollo iterativo e incremental** con el fin de dividir el ciclo de vida del producto en diferentes etapas, en las cuales consigamos alcanzar los objetivos y requisitos planteados previamente en el proceso de planificación. Este método de trabajo nos permite seguir un proceso evolutivo de nuestra herramienta, ya que iremos abordando en cada iteración aspectos cada vez más específicos. De esta forma, conseguimos refinar nuestra aplicación añadiendo más funcionalidades hasta alcanzar el grado de complejidad que buscamos. Este enfoque permite un desarrollo más ágil y flexible del producto frente a metodologías más tradicionales, puesto que nos permite también proponer cambios y mejoras en las sucesivas iteraciones. Otro punto a destacar es que esta metodología nos permite diseñar para cada iteración sus pruebas correspondientes y verificar que las funcionalidades desarrolladas presentan el comportamiento esperado. Mediante estas pruebas comprobamos que todo funciona como es debido con el fin de evitar arrastrar fallos a los siguientes pasos, donde solucionarlos supondría un coste más elevado.

Desde el primer momento estábamos seguros de que esta metodología era la que mejor se adaptaba a las necesidades de nuestro proyecto, puesto que desde el inicio teníamos más o menos claro cuales iban a ser los requisitos indispensables a desarrollar, lo que nos permitió establecer una planificación más estricta. Sin embargo, como ya hemos comentado antes, este tipo de metodología también nos permite proponer cambios y mejoras entre las distintas iteraciones, manteniendo así una cierta flexibilidad respecto la planificación de referencia. Otro de los aspectos claves que determinó la elección de esta metodología fue la posibilidad de ir incorporando cada vez funcionalidades más complejas en la herramienta a medida que el alumno iba mejorando sus conocimientos técnicos de las herramientas utilizadas.

5.2 Recursos

Los recursos son componentes que se necesitan gestionar y asignar para desarrollar el proyecto y completar sus tareas. En la gestión de nuestro proyecto, hemos identificado dos tipos diferentes de recursos: humanos y materiales.

5.2.1 Humanos

El ejemplo perfecto para este tipo de recurso en un proyecto son los diversos profesionales o empleados y miembros del equipo de trabajo que desempeñan una función concreta dentro del proceso. Entre ellos, contamos con:

- **Supervisores:** Este papel está desempeñado por los tutores del proyecto que tratan de guiar al alumno a través de todo el proceso, con la intención de evitar que este cometa errores en la elaboración de la aplicación y de la documentación asociada.
- **Analista y desarrollador:** Por otro lado, el alumno tendrá que cumplir con las funciones tanto de analista como también de desarrollador de la aplicación, siendo este el que deba tomar las decisiones necesarias para completar con éxito el proyecto.

5.2.2 Materiales

A este grupo de recursos podemos atribuir cualquier herramienta necesaria para la finalización del proyecto, tanto software como hardware, según las características del mismo. Por tanto, en nuestro caso podremos incluir en este tipo de recurso tanto las herramientas software que hemos mencionado en el capítulo 4 de esta memoria, empleadas para la elaboración del proyecto, como también el propio ordenador personal del alumno utilizado durante todo el proceso de desarrollo.

5.3 Planificación inicial

Después de realizar una reunión inicial con los tutores del proyecto donde se acotó el alcance del mismo y los requisitos que debía cumplir la herramienta a desarrollar, se estableció una planificación inicial considerando los recursos disponibles y las características del trabajo. En esta sección, detallaremos las diferentes fases en la que se dividió el desarrollo en esta planificación inicial.

5.3.1 Iteraciones planificadas para el desarrollo

- **Iteración 1:** La primera iteración de nuestro proyecto la hemos reservado para adquirir los conocimientos teóricos necesarios sobre comunicaciones inalámbricas para desarrollar nuestra aplicación correctamente.
- **Iteración 2:** Esta segunda iteración está pensada para explorar y decidir las herramientas software adecuadas para poder desarrollar una aplicación que tuviera todos los elementos necesarios para poder recrear distintos entornos *indoor* y simular las ondas propagándose por ellos.
- **Iteración 3:** En la tercera iteración, comenzaremos a construir la base de nuestro programa, implementando la primera versión de la interfaz gráfica que contendrá los elementos imprescindibles para nuestras simulaciones.
- **Iteración 4:** En la cuarta iteración de este proceso, implementaremos toda la lógica necesaria para simular el escenario más simple posible, es decir, una comunicación inalámbrica directa entre dos dispositivos. Además, se ajustará la interfaz para permitir que el usuario pueda realizar una configuración adecuada de los elementos ya implementados.
- **Iteración 5:** En la quinta iteración, nuestro objetivo es aumentar la complejidad asociada a la transmisión del haz de ondas que, en este caso, deberá considerar las colisiones con las paredes del plano antes de lograr alcanzar al receptor.
- **Iteración 6:** En esta penúltima iteración, acabaremos con la implementación de nuestra simulación añadiendo la posibilidad de que el usuario introduzca obstáculos en el entorno, lo que puede provocar cambios significativos en el comportamiento de la onda.
- **Iteración 7:** Por último, esta iteración está dedicada a implementar la representación visual de los resultados obtenidos tras la simulación y acabar de completar la interfaz gráfica añadiendo también nuevas funcionalidades que mejoran la fluidez del programa y completen las posibilidades de simulación.

5.3.2 Estimación temporal

Una vez definidas las diferentes iteraciones de las que constaría el proyecto y las tareas asociadas a cada una de ellas, se estableció que la duración del proyecto sería aproximadamente de 7 meses, teniendo en cuenta que el estudiante dedicaría entorno a 15 horas semanales debido a la dificultad que supone compaginar el desarrollo del proyecto con otras actividades. Tras realizar esta planificación temporal del proyecto, estimamos que la fecha de inicio sería el día 07/10/2021 y la fecha de finalización el día 20/04/2022. Además, se estableció que al final de cada iteración se realizaría una reunión entre los diferentes roles involucrados en el proyecto para revisar el trabajo realizado en esa iteración, aclarar posibles dudas y decidir los pasos para continuar con el trabajo en la siguiente iteración. Toda esta información podemos verla representada tanto en el diagrama de Gantt de la figura 5.1 como en la tabla de la imagen 5.2, donde se detallará en más profundidad la información temporal y económica del proyecto considerando las distintas etapas de nuestra planificación.

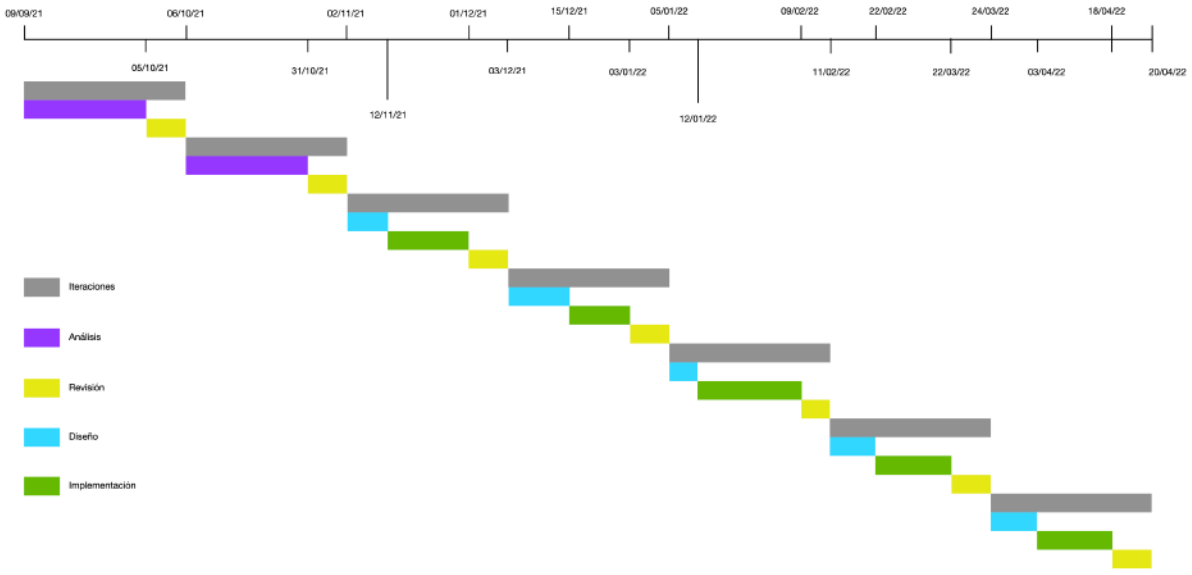


Figura 5.1: Diagrama de Gantt

En la figura 5.2, se completa la información mostrada en el diagrama de Gantt, indicando las fechas planificadas para cada una de las tareas del proyecto, las horas estimadas para poder completarlas, los recursos encargadas de realizarlas y el coste correspondiente, que explicaremos en la siguiente sección. Como se puede observar, el número total de horas que se planificaron para completar el proyecto fue de 463, de las cuales 20 horas corresponden al rol ejercido por los supervisores y el resto son asignadas al alumno en su rol de analista y desarrollador.

5.4 Estimación de costes

En esta sección, mostraremos un análisis de los costes asociados al trabajo realizado para la creación de nuestro programa y una estimación del tiempo consumido desde la fecha de su comienzo hasta su finalización.

En este sentido, a partir de la planificación inicial del proyecto, también hemos realizado una estimación económica de lo que habría supuesto la creación de nuestra aplicación en un contexto comercial. Para ello, nos hemos informado de la banda salarial actual de un desarrollador de Python y de un supervisor de proyectos en España [26]. Tras este estudio, se ha concluido que la media salarial para un desarrollador se encuentra aproximadamente en 15 €/hora y para un supervisor en 10 €/hora. En base a estas cifras, y teniendo en cuenta las horas dedicadas al desarrollo del programa y los recursos laborales comentados en la sección anterior, obtenemos un coste en recursos humanos aproximado de 7.905 € para el proyecto. En la figura 5.2, podemos observar el reparto de los costes asociados a cada una de las iteraciones que conforman el desarrollo completo de nuestra aplicación.

Iteraciones	Comienzo:	Fin:	Horas:	Asignado a:	Coste:
Iteración 1	09/09/2021	06/10/2021	50		910
Análisis	09/09/2021	05/10/2021	42	Analista	630
Revisión	05/10/2021	06/10/2021	8	Supervisores;Analista	280
Iteración 2	06/10/2021	02/11/2021	40		780
Análisis	06/10/2021	31/10/2021	31	Analista	465
Revisión	31/11/2021	02/11/2021	9	Supervisores;Analista	315
Iteración 3	02/11/2021	03/12/2021	54		890
Diseño	02/11/2021	12/11/2021	16	Desarrollador	240
Implementación	12/11/2021	01/12/2021	34	Desarrollador	510
Revisión	01/12/2021	03/12/2021	4	Supervisores;Desarrollador	140
Iteración 4	03/12/2021	05/01/2022	72		1160
Diseño	03/12/2021	15/12/2021	24	Desarrollador	360
Implementación	15/12/2021	03/01/2022	44	Desarrollador	660
Revisión	03/01/2022	05/01/2022	4	Supervisores;Desarrollador	140
Iteración 5	05/01/2022	11/02/2022	90		1470
Diseño	05/01/2022	12/01/2022	21	Desarrollador	315
Implementación	12/01/2022	09/02/2022	63	Desarrollador	945
Revisión	09/02/2022	11/02/2022	6	Supervisores;Desarrollador	210
Iteración 6	11/02/2022	24/03/2022	107		1765
Diseño	11/02/2022	22/02/2022	32	Desarrollador	480
Implementación	22/02/2022	22/03/2022	67	Desarrollador	1005
Revisión	22/03/2022	24/03/2022	8	Supervisores;Desarrollador	280
Iteración 7	24/03/2022	20/04/2022	50		930
Diseño	24/03/2022	03/04/2022	16	Desarrollador	240
Implementación	03/04/2022	18/04/2022	25	Desarrollador	375
Revisión	18/04/2022	20/04/2022	9	Supervisores;Desarrollador	315
			Total:		Total:
			463		7905

Figura 5.2: Análisis temporal y económico

Por otro lado, en relación a los recursos materiales, software y hardware, se ha asumido que no suponen ningún coste adicional para el proyecto ya que todas las herramientas utilizadas son libres y se ha usado el ordenador personal del estudiante. De esta forma, el coste estimado final para todo el proyecto coincide con el gasto asociado a los salarios de los recursos humanos y asciende a un total de 7.905 €.

5.5 Seguimiento

Como se puede observar, la fecha inicial fijada para la finalización del proyecto estaba estimada para el día 20/04/2022. Sin embargo, podemos afirmar que se han sufrido algunas desviaciones durante el desarrollo del mismo que han ido retrasando esta fecha. En primer lugar, debido a problemas de salud del alumno, se decidió mantener el proyecto parado durante varios meses (desde principios de febrero hasta mediados de abril) por la imposibilidad de continuar con el desarrollo. Es importante destacar que, al mantener parado el proyecto, esto no implicó un aumento significativo del número de horas de trabajo por lo que el coste también se mantuvo relativamente estable.

Por otro lado, las horas reales dedicadas al proyecto son unas pocas más que las planificadas inicialmente. En concreto, se estima que fueron unas 38 horas a mayores que fueron invertidas, sobre todo, en la penúltima iteración del proyecto con el objetivo de desarrollar correctamente las leyes físicas que intervienen en el comportamiento de la señal puesto que su implementación requirió más tiempo del estimado en un principio. En el resto de iteraciones, las desviaciones ocurridas fueron mucho menores y, de hecho, en algunas se consiguió recuperar parte de las horas acumuladas a mayores en iteraciones previas. El hecho de dedicar esas 38 horas más al desarrollo supone también un aumento del coste real del proyecto, alcanzando un total de 8.475 €, lo que implica un coste extra de 570 € respecto al coste estimado.

Capítulo 6

Análisis y diseño

En este capítulo, trataremos de describir los requisitos del sistema identificados en la fase de análisis previa a la implementación, y se describirá además el patrón de diseño escogido para elaborar la arquitectura de nuestra aplicación.

6.1 Requisitos del sistema

Un requisito se puede definir como una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio. En nuestro caso, el desarrollo software, es muy importante reunir todos los requisitos del sistema correctamente ya que en muchos proyectos es la principal fuente de fracaso. Para ello, es necesario realizar un análisis previo de las necesidades que se pretenden cubrir con el producto que se va a desarrollar y establecer los requisitos correspondientes. En general, en ingeniería del software podemos encontrarnos dos tipos de requisitos: funcionales y no funcionales.

6.1.1 Requisitos funcionales

En primer lugar, nos encontramos los requisitos funcionales de la aplicación que establecen el comportamiento de la misma, es decir, definen las funcionalidades de nuestro software. Normalmente, un analista de requisitos puede detectarlos en la fase de análisis empleando diagramas de casos de uso, que son generados tras haber realizado una serie de reuniones con el cliente. Sin embargo, la metodología escogida para el desarrollo de este proyecto nos permite detectar nuevos requisitos si fueran necesario, o actualizar otros ya existentes, en otras fases del proyecto para cubrir perfectamente las necesidades del usuario. En el análisis de requisitos de nuestra aplicación, hemos detectado claramente los siguiente requisitos funcionales:

- **Configuración del escenario:** El primer requisito que detectamos en nuestro proyecto fue la necesidad de poder diseñar nuestros propios escenarios, personalizarlos y

configurarlos de acuerdo a las características reales que deseábamos simular. En este sentido, nuestra herramienta nos debe permitir diseñar escenarios personalizados con la posibilidad de ajustar su tamaño y emplear figuras geométricas para representar posible obstáculos que pueda haber en un entorno *indoor*, además de situar por el plano nuestros nodos de comunicación, creando así una red inalámbrica. Esta funcionalidad también nos permite ajustar los atributos de los distintos elementos del plano, tanto de los dispositivos, obstáculos e incluso las propias paredes del escenario, para conseguir una mayor flexibilidad en nuestras simulaciones. En esta misma línea, es también importante poder parametrizar las propiedades de las ondas generadas en el transmisor.

- **Simulación de la propagación de la onda transmitida:** La segunda funcionalidad consiste en la implementación de todos aquellos cálculos necesarios para recrear matemáticamente la propagación de una onda electromagnética en un entorno de interiores previamente diseñado, incluyendo las interacciones con los elementos presentes en el mismo. Por un lado, esto provocará que la onda generada se vea afectada y sufra cambios en la dirección de propagación, ya que cambiará constantemente de trayectoria al colisionar con los diferentes obstáculos siendo fieles al fenómeno de reflexión. También veremos como se producirá una atenuación en la potencia de la onda transmitida provocada por las pérdidas de propagación en espacio libre, entre otros fenómenos, y que también se puede ver afectada por posibles efectos de dispersión.

- **Visualización de resultados:** La tercera funcionalidad que hemos identificado para nuestro programa es la capacidad de mostrar los resultados de la simulación al usuario de un modo tanto visual como numérico. El usuario contará además con una lista de estadísticas y valores que podrá emplear para analizar el rendimiento del sistema de comunicaciones en el escenario considerado y sacar sus propias conclusiones.

- **Interfaz gráfica:** Una de las funcionalidades que hemos priorizado en la elaboración de nuestro proyecto fue la creación de una interfaz que le permita al usuario poder configurar todos los parámetros de la simulación de forma cómoda.

- **Asistente de configuración:** En penúltimo lugar, la herramienta también contará con un asistente de configuración capaz de guiar al usuario mostrándole los pasos que debe seguir para completar correctamente una simulación.

6.1.2 Requisitos no funcionales

En segundo lugar, tenemos los requisitos no funcionales de nuestra aplicación que no hacen referencia directamente a las funciones específicas, si no que representan características generales y restricciones de la aplicación o sistema que estemos desarrollando. Con frecuencia,

los requisitos no funcionales son subestimados en la fase de análisis, lo que termina originando complicaciones que se identifican en la fase de implementación, y cuya solución implica más trabajo y coste. Por ello, nunca debemos ignorar la importancia de estos requisitos. En nuestro caso, hemos tenido en consideración cuatro requisitos no funcionales que debía cumplir nuestra aplicación:

- **Rendimiento:** Esta característica se basa en la cantidad de trabajo realizado por nuestro sistema para llevar a cabo una determinada acción. Dentro de nuestro contexto, sugiere que nuestra aplicación debe conseguir tiempos de respuesta cortos en la obtención de resultados, baja utilización de recursos computacionales y una alta disponibilidad del sistema.

- **Fiabilidad:** Nuestra aplicación también debe tener la capacidad de mantener su nivel de prestación bajo condiciones definidas y durante un periodo de tiempo establecido. Es importante que los resultados proporcionados se ajusten a la realidad de acuerdo al modelo utilizado.

- **Usabilidad:** Debemos conseguir que nuestra herramienta sea fácil de utilizar por aquellas personas que quieran emplearla. En este sentido, la interfaz debe ser intuitiva, limpia, comprensible y sencilla de utilizar por los potenciales usuarios. De igual modo, los resultados se deben presentar de forma clara y ordenada para mejorar la comprensión de los mismos.

- **Extensibilidad:** La implementación de nuestro programa debe ser capaz de considerar y facilitar un crecimiento futuro para incorporar nuevas funcionalidades.

6.2 Arquitectura

El estilo de arquitectura de software que hemos escogido para nuestra herramienta se basa en el patrón de diseño conocido comúnmente como Modelo Vista Controlador (MVC). El objetivo principal de este diseño es separar la lógica de la aplicación, la interfaz de usuario y la lógica de control, ya que esto nos proporciona una mejor división del trabajo a realizar, mejorar la modularidad de la aplicación y simplifica el mantenimiento. Por tanto, tal y como podemos observar en la figura 6.1, este patrón de diseño establece tres componentes claramente diferenciados para la arquitectura software:

- **Modelo:** Este elemento contiene una representación de los datos que maneja el sistema, su lógica de negocio y sus mecanismos de persistencia. En nuestra aplicación, este componente será el responsable de almacenar la información relevante de cada uno de los elementos que hayamos incluido en el diseño de nuestro escenario, además de

proporcionar los cálculos que se requieran a lo largo de la simulación para obtener los resultados finales, los cuales no se almacenarán de forma persistente puesto que nuestra intención era añadir en un futuro la posibilidad de que el usuario fuera capaz de exportar e importar escenarios ya creados.

- **Controlador:** Este elemento actúa como intermediario entre el modelo y la vista, gestionando los flujos de información que se envían entre ellos y adaptando los datos a las necesidades de cada uno. Este componente es posiblemente el más importante en nuestra aplicación ya que es el que recibirá las notificaciones por parte de los eventos de entrada desde la interfaz y ejecutará la simulación solicitando al resto de componentes los datos y los cálculos necesarios para completarla con éxito.
- **Vista:** Este elemento se encarga del diseño y la presentación de la información. En nuestra aplicación, este componente será empleado para proporcionar al usuario los puntos de entrada a través de los cuales podrá ajustar los parámetros y aportar los datos necesarios para el correcto funcionamiento de la simulación y, una vez completada esta, presentar los resultados finales para poder ser analizados. También desempeñará la función de aplicar todos aquellos cambios visuales que se deban aplicar en la interfaz de usuario como resultado de una simulación o de la interacción con el usuario.

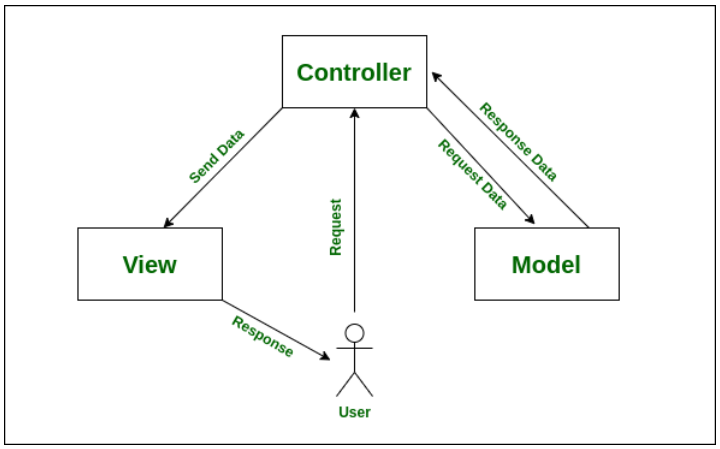


Figura 6.1: Patrón de diseño MVC

6.3 Diagrama de clases

Como resultado del análisis de requisitos y de la arquitectura definida para el sistema, se ha realizado el diagrama de clases que se muestra en la figura 6.2 de la siguiente página, con el objetivo de comprender mejor cómo estructurar la aplicación que se va a desarrollar. Como se puede apreciar, la clase "Controller" es la encargada de gestionar los eventos del sistema y solicitar al resto de componentes los datos y los cálculos que necesitaremos para realizar correctamente la simulación. Para ello, en el caso de necesitar información relevante acerca de algún elemento incluido en el escenario, llamará a su respectiva clase que forma parte del modelo, mientras que para realizar los cálculos que se requieran durante la simulación llamará a la clase "Calculo". Esta clase contiene las tres funciones más importantes del sistema, puesto que son las encargadas de realizar los cálculos principales que determinan los resultados de nuestras simulaciones. En primer lugar, tenemos la función "trayectoria" que determina el camino que sigue la onda durante su propagación y también se encarga de averiguar si la señal logra alcanzar al nodo receptor. En segundo lugar, la función "distancia" se encarga de contabilizar la longitud que recorre la señal. Esta información es bastante relevante ya que nos permite calcular las diversas pérdidas de potencia de la onda sufridas durante la propagación empleando la función "checkPotencia". Finalmente, cuando la capa "Controller" obtiene todos los resultados finales provenientes de la clase "Calculo", podrá establecer los cambios sobre la vista de nuestro programa. Para ello, llamará a la clase "View" que contiene las funciones necesarias para presentar los cambios correspondientes en el lienzo de la aplicación y mostrar las ventanas emergentes con todos los resultados finales.

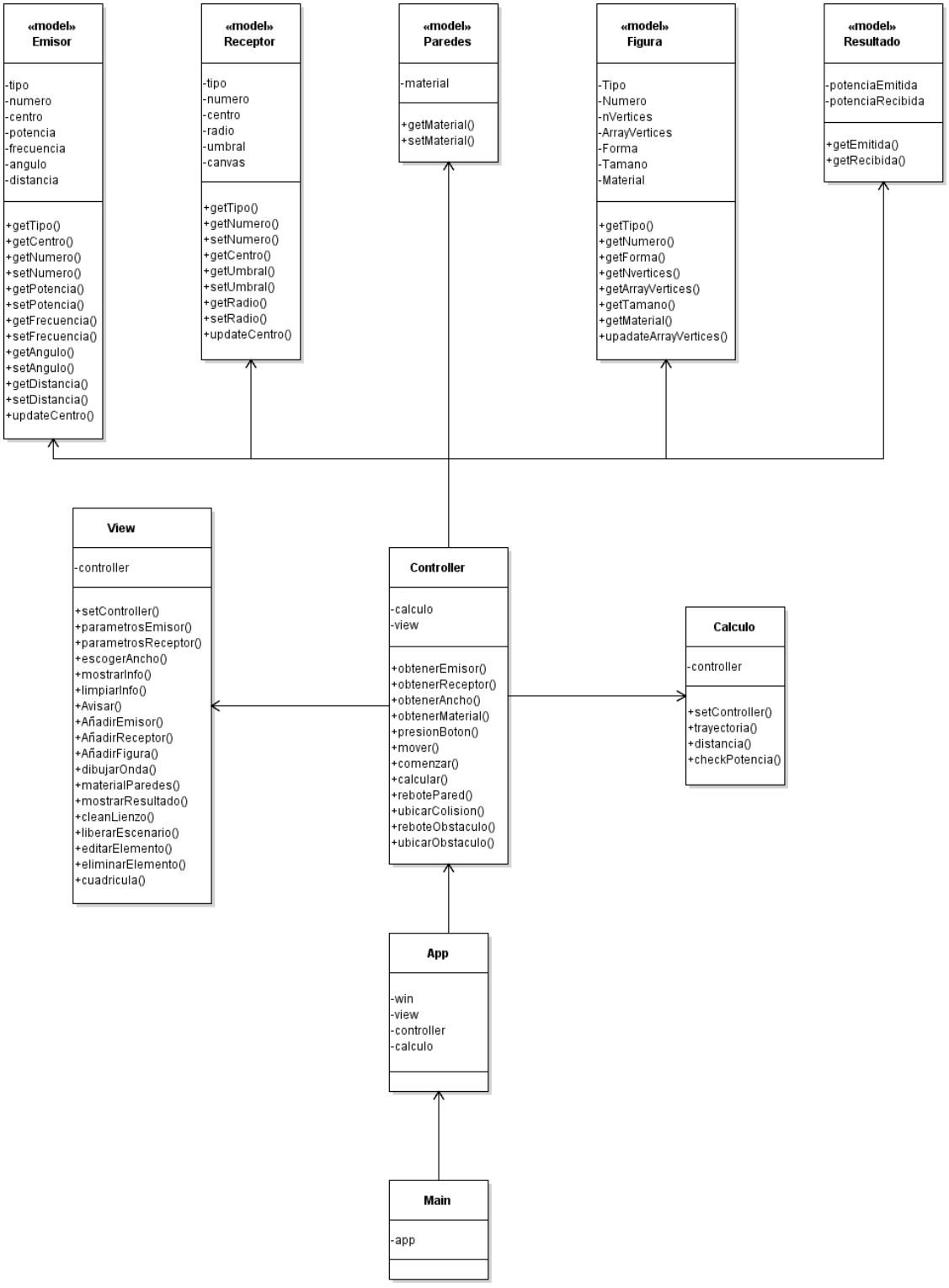


Figura 6.2: Diagrama de clases del programa

Capítulo 7

Desarrollo de la herramienta

En este capítulo, describiremos las tareas realizadas en cada una de las fases en que hemos dividido el desarrollo de nuestra aplicación cumpliendo en todo momento con la metodología descrita en el capítulo 5.

7.1 Iteración 1

En esta primera iteración, nuestro principal objetivo fue adquirir todos los conocimientos necesarios para poder comprender mejor el funcionamiento de las comunicaciones inalámbricas y representar la propagación de ondas electromagnéticas correctamente.

Por tanto, en esta fase del desarrollo hemos identificado todos los elementos involucrados en las transmisiones inalámbricas y escogido aquellos que consideramos necesarios e indispensables en nuestras simulaciones. También hemos estudiado qué leyes físicas podrían operar en el proceso de propagación y afectar a nuestra señal produciendo cambios en los resultados.

Una vez que ya conocíamos tanto los componentes que debían participar en la propagación de ondas y las leyes que debían intervenir, solo nos faltaba deducir la manera de introducirlos en nuestro programa. En las siguientes iteraciones, explicaremos como hemos implementado todos y cada uno de estos elementos en nuestro programa.

7.2 Iteración 2

Una vez adquiridas las bases conceptuales sobre las que se sostiene nuestra aplicación, hemos empleado el tiempo de esta segunda iteración con dos objetivos.

En primer lugar, hemos identificado los casos de uso y requisitos que nuestro programa debe contener para que el usuario pueda hacer un uso adecuado de él y obtener los resultados deseados. En segundo lugar, hemos realizado una búsqueda de aquellas herramientas y

tecnologías que se adaptaban mejor a las necesidades de nuestra aplicación.

Por un lado, sabíamos que al desarrollar una herramienta dentro del campo de las redes inalámbricas, debíamos buscar un lenguaje de programación que nos permitiera realizar todos los cálculos necesarios. Por ello, hemos escogido Python para el desarrollo de nuestra aplicación, ya que nos aporta la posibilidad de trabajar de forma eficiente con vectores o matrices y también con librerías muy útiles, como la librería Math, que nos ofrece una gran cantidad de operaciones y funciones trigonométricas para la implementación de ciertas partes de la herramienta, como por ejemplo, la resolución de una ecuación de segundo grado para comprobar si la onda alcanza el receptor. Otra situación podría darse cuando necesitemos hallar el gradiente de Friis y calcular la pérdida de potencia de nuestra señal al desplazarse por el espacio libre, donde debemos hacer uso de funciones logarítmicas.

Por otro lado, hemos empleado Tkinter para el desarrollo de la interfaz gráfica de nuestra aplicación. Principalmente, porque está diseñada para poder trabajar perfectamente con Python y, además, nos aportaba las herramientas necesarias para poder dotar a nuestro programa de un lienzo donde el usuario pudiera recrear los escenarios de sus simulaciones. También dispone de una extensa lista de *widgets* personalizables que nos aportan flexibilidad a la hora de diseñar nuestra interfaz e implementar todos los casos de uso y requisitos identificados anteriormente.

7.3 Iteración 3

En la tercera fase del desarrollo de nuestro proyecto, comenzamos a construir la primera versión de la interfaz gráfica y, de esta forma, incorporar los elementos principales con los cuales el usuario podrá interactuar y configurar las simulaciones.

Una vez identificados los casos de uso y los requisitos que debíamos cumplir en nuestro programa, comenzamos a diseñar la interfaz gráfica conforme a ellos. En primer lugar, hemos redistribuido el espacio de nuestra interfaz en cuatro sencillas secciones haciendo uso del *widget Frames* que nos proporciona Tkinter. Luego, como podemos observar en la figura 7.1 en la parte superior izquierda de la interfaz, empleando el *widget Canvas*, añadimos el lienzo de nuestro programa donde el usuario podrá diseñar sus escenarios *indoor*.

En segundo lugar, incluimos en la parte superior derecha de la interfaz un espacio destinado a visualizar los parámetros que se han configurado durante la preparación de la simulación. Para ello, hemos creado en la clase "View" dos funciones llamadas "mostrarInfo" y "limpiarInfo" para poder mostrar dicha información en la vista del programa o eliminarla en caso de que se proceda a realizar una nueva simulación. Como esta zona de la interfaz estaba destinada a previsualizar los ajustes del escenario, también agregaremos un botón "comenzar" con el propósito de iniciar la simulación cuando esté lista, ya que se llamará a una función con el mismo

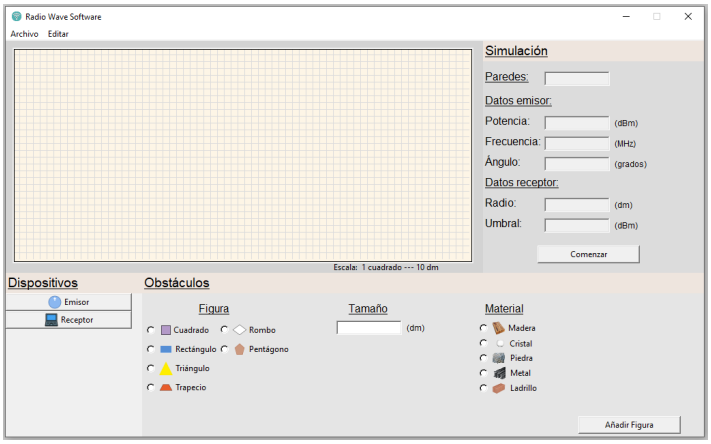


Figura 7.1: Distribución de espacios en la interfaz

nombre encargada de comprobar si se han añadido lo elementos necesarios para comenzar la simulación de la propagación de la onda, es decir, un nodo transmisor y un nodo receptor.

En tercer lugar, hemos añadido a la interfaz gráfica una zona donde dispondremos de dos botones para la creación de ambos dispositivos de comunicación. Por otra parte, hemos implementado las clases “Emisor” y “Receptor” que nos permitirán crear ambos dispositivos y almacenar toda la información referente a ellos. También hemos creado las funciones “parametrosEmisor” y “parametrosReceptor” en la clase “View”, las cuales mostrarán al usuario una ventana emergente donde podrá introducir los valores de configuración del dispositivo en cuestión. En el caso del emisor, contará con tres sencillos parámetros. La potencia de la señal emitida, la frecuencia de la onda y el ángulo de salida hacia donde se emitirá la onda generada. Por el contrario, el receptor contará con dos atributos: el radio de captación que nos indica el área hasta la cual es capaz de recibir la señal y el umbral de potencia que será la mínima potencia que deberá tener la onda para poder ser detectada. En ambos casos, es evidente que una vez el usuario introduzca los valores de configuración, se llamará a la función correspondiente de la clase “Controller” que se encarga de comprobar que los datos son correctos y coherentes. En este caso, estas funciones se llaman “obtenerEmisor” y “obtenerReceptor”. Finalmente, para que los dispositivos creados aparezcan en el lienzo de la herramienta, se hará uso de las funciones “AñadirEmisor” y “AñadirReceptor”.

Una vez contábamos con la forma de agregar ambos dispositivos al plano, debíamos buscar la manera de poder desplazarlos y situarlos donde el usuario quisiera. Para ello, creamos la funciones, “presionBoton” que es la encargada de identificar el elemento del plano con el que estamos interactuando y “mover” la cual nos permite desplazar los diferentes objetos por el plano. Para elaborar estas dos funciones, hemos empleado algunos métodos que nos proporciona el *widget Canvas*, en concreto, en la primera función hemos utilizado los métodos “findwithtag” y “gettags” para identificar el elemento en cuestión que estamos seleccionando

en el plano, mientras que en la segunda función hemos utilizado el método "move" para poder desplazar el objeto por el plano y el método "coords" para identificar la nueva localización del objeto dentro del plano. De esta forma, además si el usuario desplaza algún objeto fuera de él, éste será eliminado automáticamente. Por último, empleando la función "tagbind", hemos podido conectar los diferentes elementos que se introducen en el plano con distintos eventos del ratón, pudiendo así interactuar con ellos y llamar a las funciones que hemos comentado anteriormente.

Por último, como sabíamos que más adelante implementaríamos los obstáculos para añadirlos en nuestras simulaciones, decidimos ya en este punto del desarrollo agregar a nuestra interfaz una sección donde poder configurar y agregar las distintas figuras geométricas a nuestro lienzo. En dicha zona, el usuario podrá ajustar tres aspectos diferentes de los obstáculos: la forma geométrica, el tamaño y el material.

7.4 Iteración 4

En la cuarta fase del desarrollo del proyecto, comenzamos a abordar la propagación de ondas y a implementar como sería una conexión directa entre los dos nodos de comunicación. Como podemos ver en la figura 7.2, estamos ante el escenario más simple con el que nos podríamos encontrar en nuestras simulaciones.

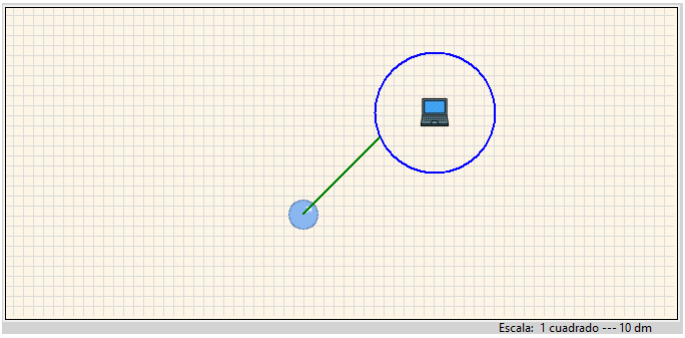


Figura 7.2: Primera versión de la simulación: Conexión directa

En esta iteración, creamos en la clase "Controller" una función llamada "calcular", la cual será la primera función a la que llamemos cuando comience la simulación. Este método será el encargado de llamar al resto de funciones que harán todos los cálculos pertinentes. En primer lugar debemos comprender que, en nuestras simulaciones, la onda se representará como una recta que se desplaza por el escenario y que, por tanto, la mayoría de nuestros cálculos se basan en resolver problemas trigonométricos. Una vez comience la simulación, llamamos a la función "trayectoria" que es el método encargado de comprobar si la onda alcanza en algún punto el área de captación del nodo receptor. Por tanto, el problema que debemos resolver aquí

es una ecuación de segundo grado para averiguar si la onda (una recta) corta en algún punto el área de captación del receptor (una circunferencia). Para realizar esta operación, los datos que consideramos son: las coordenadas del nodo emisor que representa el punto de origen de la recta, el ángulo de salida de la onda que nos aportará la pendiente de la recta, las coordenadas del receptor que nos indicará el centro de la circunferencia y el radio de captación de la antena receptora para conocer el propio radio de la circunferencia. Por último, según el resultado que obtengamos de resolver dicha ecuación, sabremos si la onda llega a alcanzar al nodo receptor o no.

En segundo lugar, llamaremos a la función "dibujarOnda", encargada de dibujar en el plano la recta que representa la onda. Para ello, solo necesitamos saber el punto de origen que, en este caso, será el centro del transmisor, el ángulo de salida que fue indicado por el usuario y también el punto exacto donde la onda corta la circunferencia del receptor, dato que hemos calculado previamente con la función "trayectoria". Finalmente, se hará uso del método "createLine" del *widget Canvas* que nos permite agregar líneas en nuestro lienzo.

En último lugar, debemos comentar que en este punto del desarrollo nuestro principal objetivo era obtener la manera de comprobar si la onda conseguía llegar al nodo receptor. Sin embargo, también comenzamos a implementar la función "distancia", encargada de determinar la distancia recorrida por la onda y la función "checkPotencia", en la cual realizamos los cálculos para obtener la potencia con la que la onda llega al receptor después de propagarse por el espacio libre una cierta distancia. En la función de la distancia, simplemente realizamos el cálculo de la hipotenusa siguiendo el teorema de Pitágoras para hallar cuánta distancia se ha desplazado la onda hasta llegar al receptor. Luego, en la segunda función, sabiendo la distancia que la onda ha recorrido y la frecuencia de la misma, podemos aplicar la fórmula de Friis para calcular la pérdida de potencia de la señal debida a la distancia recorrida y restársela a la potencia inicial de la onda. Estos resultados se almacenarán en la clase "Resultado" que hemos creado para posteriormente mostrárselos al usuario.

7.5 Iteración 5

En la quinta iteración de la implementación de nuestro programa, hemos abordado el caso en el cual la onda no es capaz de alcanzar al receptor mediante una conexión directa y, por tanto, va colisionar en alguna de las paredes del plano. En primer lugar, hemos creado la clase "Paredes", la cuál nos permitirá dotar a los bordes del plano de un material en concreto, según la decisión del usuario. Para ello, hemos implementado la función "materialParedes" que le mostrará al usuario una ventana emergente donde podrá escoger uno de los materiales de la lista. Esta elección será obligatoria antes de comenzar la simulación puesto que el material de dichas paredes será relevante para el cálculo de la pérdida de potencia de la señal.

Llegados a este punto, debíamos buscar la manera de hacer interactuar la onda con las paredes. Para ello, en la función "calcular" ya mencionada anteriormente, hemos implementado un bucle que comprobará en cada una de las iteraciones si la onda alcanza o no al nodo receptor. Si se confirma a través de la función "trayectoria" que la onda llega al área de recepción, entonces el bucle termina porque se ha completado la comunicación pero, en el caso de que la onda no alcance la área de captación del nodo receptor, entonces mediante la función "rebotePared" calcularemos con cual de los bordes impactaría la onda. Para ello, solo necesitaremos saber las coordenadas del nodo transmisor y el ángulo de salida, ya que debemos resolver una ecuación simple para averiguar el punto exacto en el que se cortan la recta que representa la onda y la recta que representa el borde del plano. Una vez obtenido el resultado de dicha ecuación y sabiendo con cual de las paredes colisiona la onda, almacenaremos dichas coordenadas como nuevo punto de partida de la onda para la siguiente iteración del bucle y podremos calcular el nuevo ángulo de salida de la onda. Para ello, necesitaremos saber únicamente el ángulo con el cual incide la recta sobre el borde del plano y, cumpliendo con la ley de reflexión vista en el capítulo 3, la recta será reflejada con el mismo ángulo con el que incide, tomando siempre como referencia la normal a la superficie con la que colisiona, tal y como podemos observar en el ejemplo de la figura 7.3.

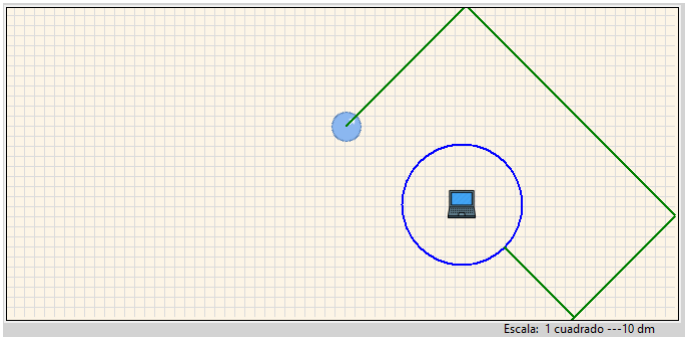


Figura 7.3: Segundas versión de la simulación: Rebotes en las paredes

Por último, hemos modificado alguna de las funciones desarrolladas en las iteraciones previas para poder realizar nuevos cálculos teniendo en cuenta que las ondas pueden rebotar en las paredes del plano. Por tanto, ahora en cada una de las iteraciones del bucle se medirá la distancia recorrida en ese tramo haciendo uso de la función "distancia" para sumarla al total y, finalmente, calcular también la pérdida de potencia provocada por la propagación de la onda en el espacio libre mediante la función "checkPotencia". A mayores, modificaremos esta última función para que ahora también tenga en cuenta los impactos con las paredes y que, dependiendo del material de las mismas, se aplique un porcentaje de pérdida por la colisión debido a un coeficiente de penetración que dependerá del material seleccionado. Esto quiere decir que si la onda se transmite con una potencia de 10 dBm y el material es madera,

por ejemplo, se asume que se perderá un 5%, es decir 0.5 dBm y quedarán 9.5 dBm en total. También modificaremos esta última función para que en cada iteración, tras calcular todas las pérdidas de energía de la señal que se producen en ese fragmento de trayectoria, podamos comprobar si la potencia de la onda en dicho instante de la propagación resulta inferior al umbral de captación del nodo receptor y, por tanto, detener la simulación.

7.6 Iteración 6

La penúltima iteración de la implementación fue destinada a introducir los obstáculos en las simulaciones consiguiendo, de esta forma, construir un escenario más complejo, tal y como podemos observar en la figura 7.4. En este caso, los obstáculos estarán representados por figuras geométricas que el usuario podrá seleccionar en la parte inferior de la pantalla principal de la herramienta, donde podrá escoger la forma, el material e introducir un tamaño. Para poder generar los obstáculos, hemos creado la clase "Figura" que guardará toda la información relativa a cada objeto que queramos añadir mediante la función "añadirFigura". Esta función permite añadir los objetos en nuestro lienzo siempre y cuando todos los parámetros de configuración de los obstáculos estén cubiertos y no haya valores erróneos.

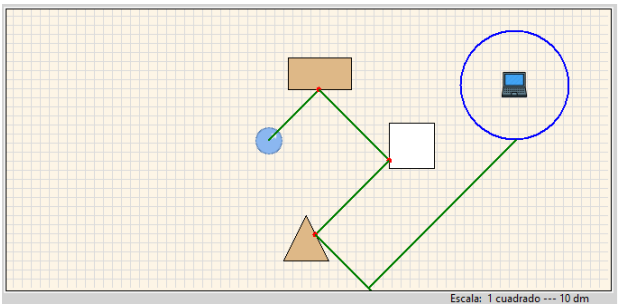


Figura 7.4: Tercera versión de la simulación: Interacción con obstáculos

Para poder simular correctamente este tipo de escenarios, hemos añadido a la función "calcular" una nueva comprobación. Si el usuario ha decidido agregar algún obstáculo al plano y la onda no alcanza de primeras al nodo receptor, antes de colisionar en alguna de las paredes, debemos averiguar si alguno de estos objetos se encuentra en la trayectoria de la onda. Por tanto, lo que hace ahora nuestra función en cuanto inicia la simulación es comprobar si hay conexión directa. Si no la hay, entonces comprobará si el usuario agregó obstáculos y, en caso negativo, el proceso sería idéntico al comentado en la iteración anterior. Sin embargo, si existe alguna figura geométrica, la función entrará en un bucle que recorrerá todas las figuras que se hayan introducido en el plano, comprobando si alguna de ellas se encuentra en la trayectoria de la onda. De esto, se encargará la función "ubicarColision", la cual mediante el punto de origen de la recta en esa iteración, la pendiente de la recta y el tipo de obstáculo,

puede comprobar mediante un bucle *while* y resolviendo ecuaciones simples entre la recta y cada arista de la figura, si la onda impacta y ubicar el punto exacto en el que colisiona. En este tipo de escenarios con obstáculos, a veces se puede dar el caso de que el obstáculo y el nodo receptor, se encuentren al mismo tiempo en la trayectoria de la onda. Para resolver esta situación, simplemente hacemos uso de la función "ubicarObstaculo" que mediante el teorema de Pitágoras es capaz de determinar cual de ellos está por delante y colisiona con la onda.

En el caso de la onda acabe impactando en alguno de los obstáculos antes de alcanzar el receptor o una pared, se tomarán las coordenadas de la colisión como nuevo punto de origen de la recta para las siguientes iteraciones y llamaremos a la función "reboteObstaculo" para determinar el nuevo ángulo de salida de la onda, el cual volverá a ser igual al ángulo con el que incide, tomando como referencia la normal a la superficie con la que colisiona. Por ello, esta función necesitará saber tanto el ángulo que llevaba la recta, como también la arista de la figura con la que haya colisionado.

Por último, también debemos realizar los cambios correspondientes en la función "check-Potencia" para poder abordar las nuevas pérdidas de potencia provocadas por el impacto de la señal con los obstáculos. A mayores de calcular las pérdidas de potencia tanto por colisionar con paredes como también por el propio recorrido en el espacio libre, ahora debemos tener en cuenta las colisiones con los obstáculos que también implicarán una cierta pérdida de energía en la señal. Para ello, cada vez que la onda colisiona con un objeto, se almacenará la información del mismo para llegados a este punto poder calcular dicha pérdida producida por la ley de reflexión que dependerá de la frecuencia de la onda y del coeficiente de penetración de los propios materiales. En segundo lugar, también hemos implementado el fenómeno de shadowing de tal forma que dependiendo de la longitud de onda y del material del obstáculo, la señal podría incluso atravesarlo, con la proporcional perdida de potencia. Por último, también implementamos la posibilidad de otorgar un cierto coeficiente de dispersión a las figuras geométricas que dependerá fundamentalmente del tipo de material. Finalmente, dependiendo de este coeficiente y de la frecuencia de la señal la onda puede generar algunas réplicas aleatorias, implicando también una cierta pérdida de potencia.

7.7 Iteración 7

En la última fase del desarrollo, hemos implementado en la clase "View" la función "mostrarResultado", que será la encargada de presentar los resultados de la simulación. Esta función mostrará tres pantallas diferentes, una por cada rayo del haz, con los correspondientes resultados y datos de la simulación. A mayores, veremos en el lienzo las tres trayectorias seguidas por cada uno de los rayos, bien diferenciadas por colores para poder observar como se ha comportado cada onda durante la simulación.

Además, hemos incorporado otras funcionalidades para conseguir mayor flexibilidad tanto en la configuración de nuestros escenarios como también en los resultados.

En primer lugar, es importante destacar que en esta iteración hemos añadido a nuestras simulaciones dos rayos más de igual potencia que el rayo principal como podemos observar en la figura 7.5, debido a los propios perfiles de radiación de las antenas. De esta forma, creamos un haz de tres rayos bien diferenciados y separados por un ángulo escogido arbitrariamente de 45 grados que provocará que cada uno de los rayos tome una trayectoria diferente de salida y obtenga resultados diferentes. Por ello, también hemos realizado cambios en nuestra vista para poder mostrar los resultados por separado y diferenciar correctamente los datos asociados a cada uno de ellos. El objetivo de formar este haz es evitar la estricta restricción en la direccionalidad de la antena del nodo transmisor y observar en un mismo escenario varios caminos que la onda podría tomar hasta llegar al receptor. Para realizar este cambio, hemos modificado la función "comenzar", nombrada en la iteración 3, a la cual hemos añadido un bucle que realizará tres repeticiones, es decir, una simulación para cada uno de los rayos.

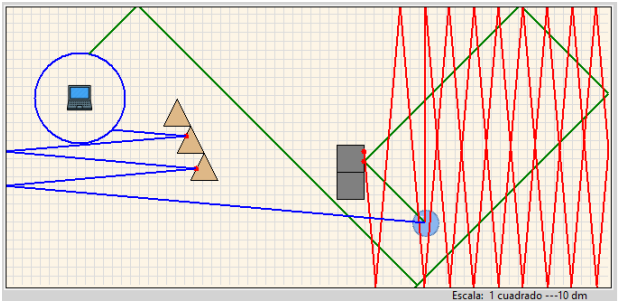


Figura 7.5: Versión Final de la simulación: Haz de tres rayos

En segundo lugar, hemos decidido añadir nuevas funcionalidades que le aportan mayor flexibilidad a nuestra herramienta a la hora de preparar las simulaciones como, por ejemplo, poder modificar el ancho del plano a gusto del usuario. Para ello, hemos creado las funciones "escogerAncho" que le permiten al usuario disminuir el ancho del plano, como también la función "obtenerAncho" que se encarga de comprobar que los valores introducidos cumplen con lo establecido. A parte, también hemos añadido una cuadrícula al plano mediante la función "cuadricula" para facilitar que el usuario pueda gestionar mejor el espacio a la hora de diseñar sus escenarios. Debemos comentar que las dimensiones iniciales de nuestro lienzo son 670x310 píxeles (px), siendo este el máximo espacio con el que podemos contar. En este caso, hemos definido la escala del plano de tal forma que cada una de las cuadrículas están formadas por 10 píxeles y se traducen en 100 decímetros cuadrados (dm^2) de longitud, consiguiendo unas dimensiones totales para el plano de 670x310 (dm), las cuales consideramos que son suficientes para que el usuario pueda diseñar sus entornos *indoor* de simulación. Las distancias dentro de nuestro escenario se medirán siempre en decímetros (dm), a la hora de

diseñar nuestro escenario, en la visualización de las distancias recorridas por cada rayo del haz de nuestra onda o a la hora de escoger el tamaño que deseamos para nuestros objetos. Los cálculos que realicemos internamente cumplirán con el estándar de cada fórmula matemática obteniendo de esta forma resultados razonables. Sin embargo, una vez obtenidos los resultados de la simulación, se volverán a trabajar con estas unidades de longitud para ser consecuentes con el diseño que haya elaborado el usuario y obtener unos resultados acordes.

En tercer lugar, en esta fase del proyecto también se decidió cambiar la implementación del lienzo para que, cuando la simulación se iniciara, el plano se bloqueara e impedir así que los elementos introducidos fueran modificados o desplazados por parte del usuario sin plena intención. Esta restricción se impone con el fin de mantener una consistencia entre los resultados numéricos obtenidos en la simulación y el resultado visual que aparecerá en el escenario configurado tras acabar la simulación. Tras finalizar la simulación, en caso de que el usuario quiera modificar su diseño y repetir la prueba, deberá escoger la opción editar/escenario de la barra superior de tareas del programa.

En penúltimo lugar, hemos añadido un menú en la barra superior de la herramienta que nos permitirá realizar acciones como crear un escenario nuevo mediante la función "cleanlienzo", editar el escenario actual permitiéndonos mover nuevamente los elementos por el plano con la función "liberarEscenario", o incluso volver a cambiar el ancho del plano o el material de las paredes llamando nuevamente a sus correspondientes funciones. También hemos añadido las funcionalidades que nos permitan editar o eliminar alguno de los elementos ya introducidos dentro del lienzo. Para ello, hemos modificado la función "presionBoton" para que sea capaz de diferenciar entre los eventos del ratón, botón izquierdo para seleccionar el elemento y desplazarlo o botón derecho y que nos emerja una nueva pantalla dándonos la opción de editar con la función "editarElemento" o eliminar con la función "eliminarElemento".

Por último, también hemos añadido a nuestra vista la función "Avisar" que se encarga de notificar al usuario en caso de que cometa algún error en la configuración de los escenarios o intente realizar alguna acción que no esté permitida como, por ejemplo, añadir más de un dispositivo de comunicación del mismo tipo al escenario. El objetivo de esta función es guiar al usuario a través de la aplicación y conseguir completar la simulación correctamente.

Capítulo 8

Funcionamiento de la herramienta

En este capítulo explicaremos nuestra herramienta al completo mostrando las diferentes funcionalidades implementadas. Así, este apartado constituye una guía para aportar al usuario una visión general de la aplicación facilitando de este modo que, gracias a las orientaciones aportadas, pueda manejar la herramienta de una manera sencilla y fluida.

8.1 Ventana Principal

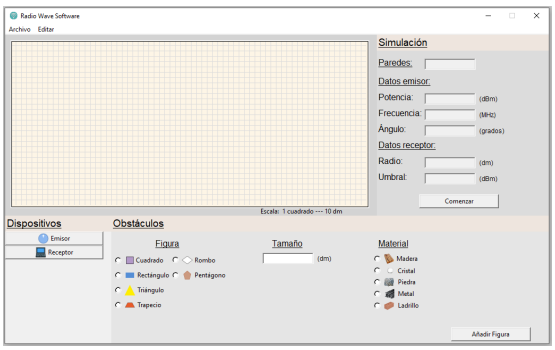


Figura 8.1: Ventana principal de la herramienta

En primer lugar, explicaremos como está distribuida la ventana principal de la herramienta y qué funciones nos encontraremos en ella. Como podemos observar en la figura 8.1, distinguimos tres zonas bien diferenciadas. La primera de ellas, situada en la parte superior izquierda, donde nos encontramos el plano utilizado para representar los escenarios. En la parte superior derecha vemos un panel que nos permitirá configurar información relevante para la simulación, tanto de las paredes del escenario como de los dispositivos de comunicación. Además, se presenta el botón "comenzar", que debemos pulsar para iniciar la simulación cuando tengamos todo preparado. Por último, la zona situada en la parte inferior de la ventana contiene los distintos elementos para emplear en nuestros diseños, los dos dispositivos de comunicación

que serán obligatorios en toda simulación y las opciones de configuración de los obstáculos que queramos añadir.

8.2 Pasos a seguir

Una vez nos hemos familiarizado con la apariencia de nuestra herramienta, debemos conocer los pasos que debemos seguir para configurar correctamente nuestra simulación y conseguir un escenario como el de la figura 8.2.

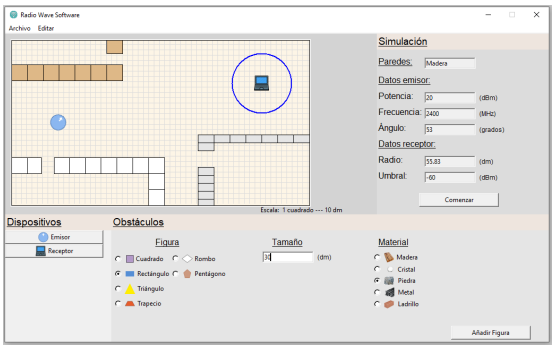
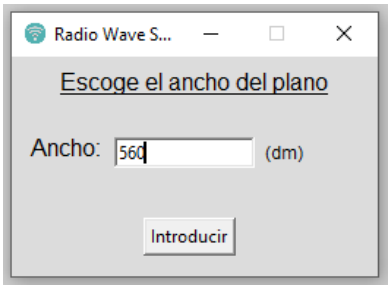
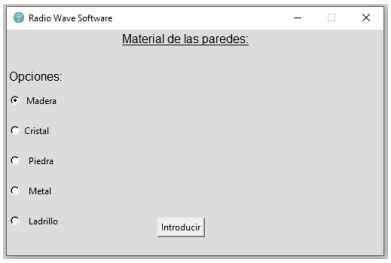


Figura 8.2: Ejemplo de escenario antes de la simulación

En primer lugar, abriremos nuestro programa y nos aparecerán dos ventanas secundarias, a parte de la ventana principal que acabamos de explicar en la sección anterior. Podemos observar ambas ventanas en la figura 8.3.



(a) Ventana para cambiar el ancho



(b) Ventana para cambiar el material de las paredes

Figura 8.3: Ventanas de configuración del plano

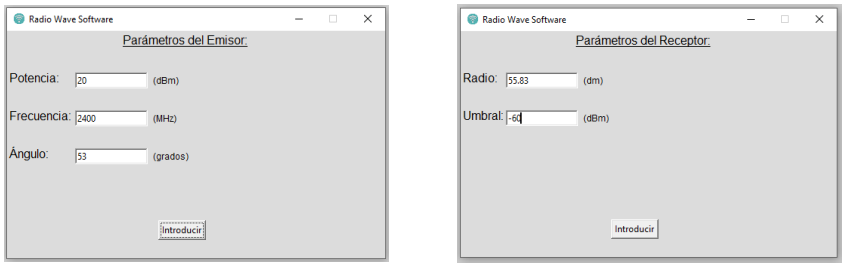
Por una parte, en la figura 8.3a, vemos representada la ventana que nos permitirá modificar el ancho de nuestro plano para ajustarlo a la geometría del escenario que nosotros queramos crear. Debemos comentar que las dimensiones iniciales de nuestro lienzo son 670x310 píxeles (px), siendo este el máximo espacio con el que podemos contar. Sin embargo, si observamos la escala del plano en la figura 8.2 podemos ver que cada una de las cuadrículas que están formadas por 10 píxeles cada una, representan 10 decímetros cuadrados (dm^2) puesto que

consideramos que un plano de esas dimensiones es suficiente para realizar simulaciones en espacios cerrados. Por otro lado, manteniendo la altura como una dimensión fija, el usuario podrá reducir el ancho del plano hasta un mínimo de 310 (dm) consiguiendo así un plano totalmente cuadrado.

Por otra parte, la segunda figura 8.3b nos muestra un listado de posibles materiales de los cuales debemos escoger una opción para las paredes de nuestro plano, ya que es una característica importante que influirá en los resultados y, por ello, es una condición obligatoria para poder comenzar nuestra simulación.

En cualquier caso, si decidimos no efectuar ninguno de estos cambios y cerramos ambas ventanas, no habrá ningún problema puesto que se tomarían los valores por defecto y estos se podrían modificar también desde la opción editar que tenemos en la barra superior de nuestra aplicación.

Ahora, es el turno de añadir los dispositivos de comunicación ya que son dos elementos fundamentales de nuestra simulación. Una vez que seleccionamos con el botón correspondiente el dispositivo que queremos añadir a nuestro escenario, aparecerá una ventana de configuración en la que añadiremos los valores necesarios para la emisión o recepción de la señal inalámbrica. En las dos imágenes siguientes, veremos un ejemplo de cómo configurar ambos dispositivos.



(a) Configuración del emisor (b) Configuración del receptor

Figura 8.4: Ventanas de configuración para ambos dispositivos de comunicación

Tal y como podemos ver en la figura 8.4a, en el caso del emisor su configuración consta de tres parámetros: potencia transmitida, frecuencia de la onda y ángulo de salida. Emplearemos tanto la potencia como la frecuencia para realizar los cálculos en la simulación, y el ángulo de salida para marcar la dirección hacia donde se emitirá el rayo principal. La representación de la antena emisora está configurada para que, según el ángulo que el usuario añada, su imagen rote hacia esa dirección y sea más sencillo visualizar hacia donde se emitirá el rayo principal en el plano. Podemos ver un ejemplo de esta característica si observamos el ángulo que añadimos en la imagen 8.4a y observamos como la flecha del emisor de la imagen 8.5 indica hacia esa misma dirección.

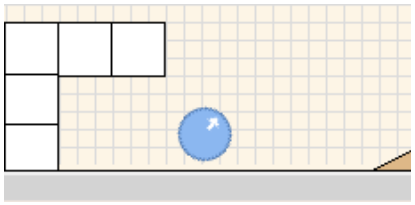


Figura 8.5: Rotación de la flecha del emisor indicando la dirección

En segundo lugar tenemos el receptor, cuya configuración está compuesta básicamente por dos parámetros: radio de adquisición y umbral o sensibilidad de recepción. Comenzamos con el radio que nos muestra el área receptora donde nuestra antena será capaz de captar la onda emitida. Este atributo está representado en el escenario por una circunferencia alrededor de nuestro receptor, como la que observamos en la figura 8.2. En segundo lugar, tenemos el umbral que nos indica la potencia mínima con la que los rayos de nuestra onda deben llegar para poder ser recuperados por el receptor. En caso de no alcanzar dicha potencia, esa contribución o réplica de señal será descartada.

Llegados a este punto tenemos dos alternativas, iniciar la simulación ya que nuestro escenario dispone de los elementos necesarios para hacerlo, o dar rienda suelta a nuestra imaginación y añadir a nuestro plano distintos obstáculos para darle forma a nuestro diseño. Como podemos observar en la imagen de la figura 8.6, en la parte inferior de la ventana principal disponemos de un menú compuesto por tres atributos para crear nuestros obstáculos. Por tanto, para cada figura geométrica que queramos añadir debemos escoger su forma, tamaño y material. En caso de haber creado demasiados objetos, si queremos eliminar algún objeto o dispositivo, podemos arrastrarlo fuera del plano o realizar un clic derecho sobre el elemento y el programa nos dará esta opción. Igualmente, si queremos modificar alguno de los atributos de los dispositivos, podremos realizarlo también haciendo clic derecho sobre él y ya nos presentará la opción de editarlo.

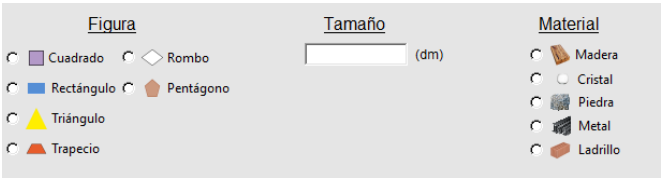


Figura 8.6: Configuración de obstáculos

Finalmente, debemos iniciar la simulación accionando el botón de “comenzar” y esperar a obtener los resultados tanto visuales como los numéricos. Tal y como observamos en la figura 8.7, en el escenario se dibujarán las distintas trayectorias de los tres rayos que conforman nuestra señal propagándose hacia el receptor en función de la configuración que se ha realizado para el escenario. Es decir, se muestra visualmente al usuario lo ocurrido en la transmisión,

para así comprender mejor los resultados numéricos que se aportan.

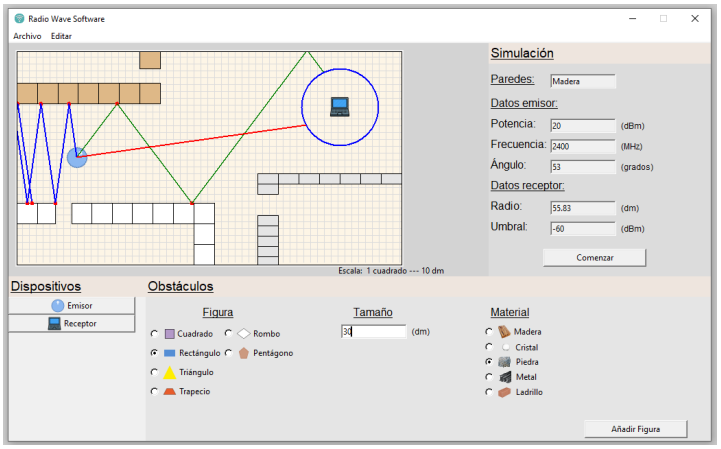
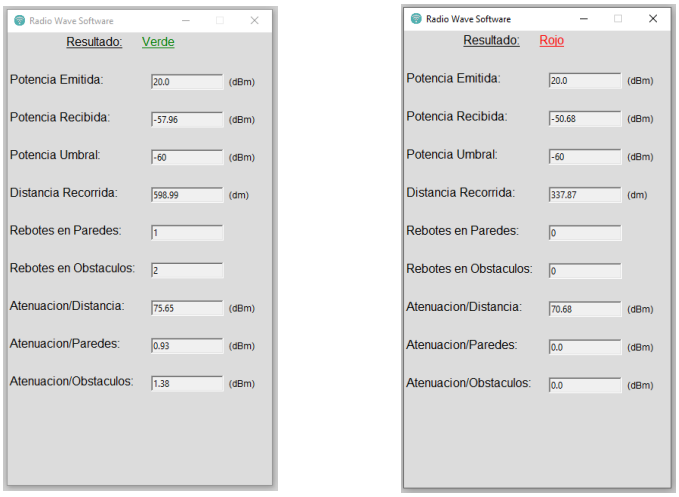


Figura 8.7: Trayectorias de la onda

8.2.1 Ejemplo de uso

Hemos preparado la simulación de la figura 8.7 con el objetivo de que nos diera los dos tipos de resultados que podemos obtener tras emitir los diferentes rayos que componen nuestra onda por el escenario. Como se puede observar en la figura anterior, dos de los rayos que se han enviado han alcanzado la antena receptora con la suficiente potencia como para ser detectados (rayo verde y rayo rojo). Por el contrario, el rayo marcado en color azul no ha alcanzado el receptor, ya que las diferentes atenuaciones sufridas han provocado que su potencia caiga por debajo del umbral establecido para el receptor. En las dos siguiente pantallas que se muestran en la figura 8.8, podemos ver dos ejemplos de los resultados numéricos que se nos mostrarán en caso de sí alcanzar al nodo receptor.

Como podemos observar en la figura 8.8, son los resultados correspondientes a los rayos verde y rojo de la onda que consiguieron alcanzar el área de recepción de nuestra antena con suficiente potencia. Estas ventanas emergentes nos muestran todo tipo de datos acerca del transcurso de dicha propagación. Si nos paramos a analizar la información que se nos aporta y la comparamos con los trazados de los rayos que vimos en la imagen 8.7, veremos que existe correlación entre los resultados numéricos y visuales. Esta información contiene muchos datos interesantes acerca de la simulación, vemos como se realiza un recuento de los rebotes que realiza cada rayo por separado, junto a la atenuación sufrida en cada caso tanto por las colisiones con los materiales como por la propia propagación en espacio libre, y observamos además como ello impacta sobre el valor numérico de la potencia recibida por la antena. Por último, vemos también el total de la distancia recorrida por cada rayo y su paralelismo con el trazado efectuado en el lienzo.



(a) Resultados del rayo rojo

(b) Resultados del rayo azul

Figura 8.8: Resultados cuando la antena receptora obtiene los rayos

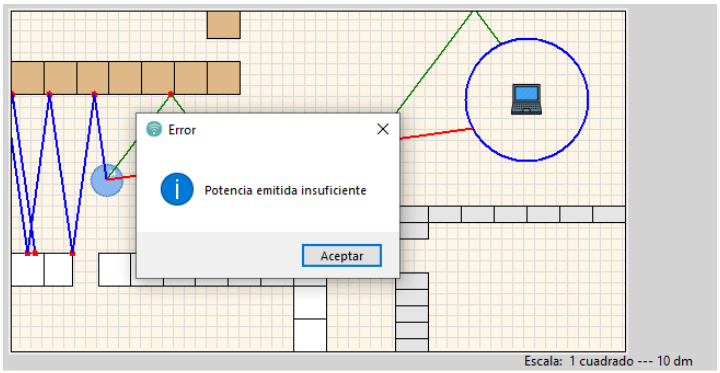


Figura 8.9: Resultado por potencia insuficiente

Por otro lado, tenemos en la figura 8.9 un ejemplo del segundo tipo de resultado que podríamos obtener en alguna simulación. En este caso, el rayo azul de nuestra onda no logra alcanzar el área de nuestra antena con una potencia superior al umbral de nuestro dispositivo receptor. Por ello, obtenemos una advertencia por parte de nuestra simulación donde nos indica que la potencia resultante siguiendo esa trayectoria en cuestión no es suficiente.

En esta sección, hemos visto un ejemplo de los pasos que debemos seguir para utilizar correctamente nuestra aplicación de una manera sencilla y fluida. En caso de querer realizar una nueva simulación desde cero sin cerrar el programa, debemos usar la opción archivo/nuevo que tenemos en la barra superior de nuestra ventana principal. En otro caso, si queremos repetir la simulación actual pero editando algún aspecto del escenario, utilizaremos la opción editar/escenario de la cual disponemos también en la barra superior de nuestra interfaz.

8.3 Control de errores

En esta sección, explicaremos algunos de los controles de errores que hemos implementado en el desarrollo de nuestra aplicación y que nos encontraremos a lo largo de la ejecución de nuestro programa. Estos mensajes de advertencia han sido creados con el fin de guiar al usuario a través de la ejecución impidiendo de alguna manera que se comprometa el correcto funcionamiento de la simulación o del propio programa.

Durante el proceso de desarrollo de la herramienta hemos implementado una gran cantidad de mensajes que nos guiarán a través del programa. A continuación, mostraremos algunas de las situaciones más relevantes en las que podríamos encontrarnos estas advertencias.

Posiblemente la situación más importante en la que nos podemos encontrar una advertencia sucede cuando el usuario intenta iniciar la simulación sin haber introducido antes todos los elementos obligatorios en el plano. En el siguiente ejemplo 8.10, mostramos el mensaje de advertencia que nos saldrá cuando nos olvidamos de añadir un dispositivo emisor en el escenario. En el caso del receptor o del material de las paredes del escenario, obtendríamos un aviso similar con la información correspondiente.

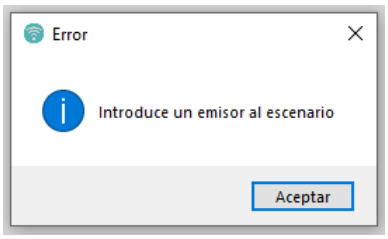


Figura 8.10: Ventana de error al no introducir un emisor

Una situación análoga a la que acabamos de comentar sucede cuando intentamos añadir un obstáculo nuevo al plano utilizando sus opciones de configuración, y el usuario no se percata de que dejó algún ajuste sin cubrir. En este caso, el programa le avisará a través de una ventana de error sobre que opción se olvidó de cubrir, tal y como vemos en la figura 8.11, donde se muestra un ejemplo de cuando nos olvidamos de añadir la forma del objeto.

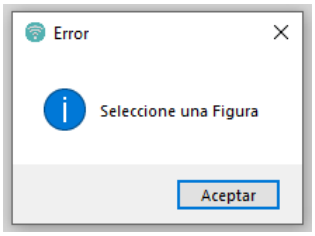


Figura 8.11: Ventana de error al no seleccionar la forma del obstáculo

Por otro lado, podremos añadir a nuestro escenario tantos obstáculos como nuestro diseño requiera mientras que, en cuanto a los dispositivos de comunicación, la simulación está preparada para admitir únicamente un dispositivo de cada tipo. Por ello, si el usuario decide añadir más de un emisor o receptor al plano, el programa le advertirá de que es una operación incorrecta puesto que ya existe un dispositivo de ese tipo en el escenario, tal y como vemos en la figura 8.12 para el caso del emisor.

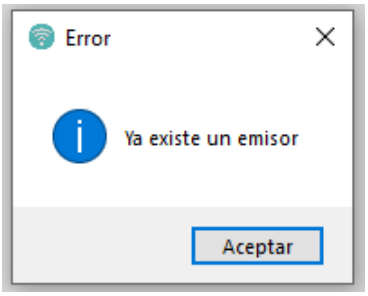
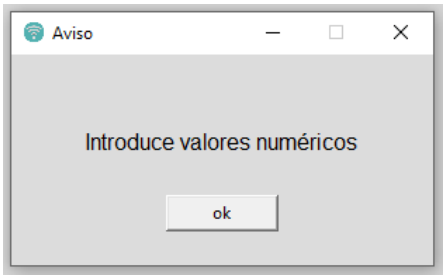
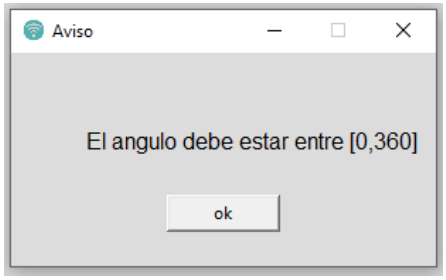


Figura 8.12: Ventana de error al crear un dispositivo ya existente

El siguiente tipo de control de datos es muy típico de aquellos programas donde el usuario debe tomar partida e introducir manualmente ciertos valores por teclado. En nuestro caso, existen diferentes momentos en la configuración de la simulación donde el usuario deberá aportar ciertos valores para crear su escenario. En concreto, en las dos siguiente imágenes vemos dos ejemplos de advertencia por introducir incorrectamente valores de configuración.



(a) Ventana de error al introducir valores no numéricos



(b) Ventana de error al introducir un ángulo fuera del rango

Figura 8.13: Control de errores de los valores de entrada.

En la figura 8.13a, vemos un mensaje de error pensado para avisar al usuario cuando deje algún campo en blanco o introduzca algún carácter no cuantificable. En segundo lugar, tenemos la figura 8.13b, donde observamos un error también acerca de los datos de entrada introducidos por parte del usuario, pero más específico al atributo donde cometió el error, en este caso la dirección de salida de la onda.

En último lugar, debemos comentar una característica de nuestro escenario ya que resulta que, en el momento en que comenzamos nuestra simulación, todos y cada uno de los elementos situados en el lienzo quedan bloqueados y, por tanto, el usuario tras finalizar la simulación no podrá editar su diseño sin antes escoger la opción editar/escenario de la barra de tareas del programa. Esto se debe a que se quiere evitar perder consistencia entre el escenario y los resultados de la simulación.

Por tanto, cuando el usuario quiera modificar alguna característica del lienzo como puede ser el ancho o el material de las paredes, el programa mostrará un aviso como el de la figura 8.14.

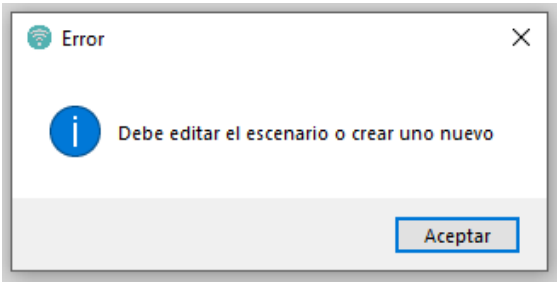


Figura 8.14: Error al intentar modificar el plano tras la simulación.

Capítulo 9

Conclusiones y líneas futuras

En este capítulo, explicaremos a qué conclusiones hemos llegado tras finalizar el proyecto y cuáles han sido nuestras sensaciones durante su realización. Además, se comentarán varias líneas de trabajo futuro con el objetivo de mejorar y completar la herramienta.

9.1 Conclusiones

Este proyecto partió con la idea principal de profundizar más en el sector de las comunicaciones inalámbricas, un campo que está permanentemente evolucionando y ganando protagonismo, y que no puede estar ya más ligado actualmente a la informática. Por esta razón, hemos aprovechado esta oportunidad para poner en práctica todos los conocimientos aprendidos durante el grado y para ampliar significativamente la información en el campo de las comunicaciones, todo con el objetivo de conseguir crear una herramienta relacionada con la propagación de ondas. El desarrollo de esta herramienta implica conocer en detalle las propiedades de los elementos que están presentes en el proceso de comunicación, como también las leyes físicas involucradas en todo el transcurso de la onda viajando hasta alcanzar su destino. Fue muy importante comprender y analizar todos estos detalles con el fin de diseñar un modelo de propagación simplificado pero realista que se ajuste a las características de nuestro proyecto. Por otra parte, este proyecto también nos da la oportunidad de aprender nuevas tecnología que no habíamos visto hasta el momento, y que fueron escogidas porque cumplían con los requisitos necesarios para el desarrollo de nuestra herramienta.

Finalmente, no podemos estar más satisfechos del resultado obtenido en nuestro proyecto, puesto que hemos conseguido elaborar una herramienta con una interfaz gráfica sencilla e intuitiva que permite a los potenciales usuarios, independientemente de sus conocimientos dentro del campo, comprender mejor la lógica de la propagación de las ondas en una red inalámbrica. Además, para aquellos usuarios que trabajen en un sector cercano a la comunicación inalámbrica, pueden emplear este programa como una herramienta más de apoyo,

que les sirva para crear o incluso mejorar sus diseños, optimizar el despliegue de sus redes y conseguir un punto de vista diferente.

Por último, desde mi punto de vista personal, realizar un proyecto de este estilo por primera vez supone un gran esfuerzo puesto que implica comenzar de cero y ser capaz de organizar el trabajo de manera eficiente. No solo implica, adentrarse en un campo de estudio diferente y aprender conceptos complejos por tu cuenta, sino también aprender a emplearlos con propiedad con la finalidad de cumplir unos objetivos. En nuestro caso hemos logrado cumplir con los objetivos establecidos al comienzo del proyecto y conseguir desarrollar una herramienta cuya principal funcionalidad consiste en simular la propagación de ondas en un entorno cerrado y que nos permite analizar y optimizar el rendimiento de nuestras redes inalámbricas.

9.2 Trabajo futuro

Desde el principio del proyecto, hemos desarrollado un programa que tuviera una estructura modular y que nos permitiera mantener una gestión sostenible y flexible ante nuevas ideas a implementar. Por ello, hemos pensado en nuevas propuestas que se puedan incorporar en un futuro para mantener el software actualizado y en constante mejora. Entre estas líneas de trabajo futuro, podemos destacar:

- Estudiar la posibilidad de aumentar la complejidad de nuestro modelo de propagación de ondas y de esta forma perfeccionar nuestras simulaciones y conseguir resultados todavía más fieles a la realidad. En este caso, una opción factible sería poder aumentar el número de antenas en los dispositivos, con el objetivo de conseguir una propagación de la señal menos direccional y más acertada a la realidad.

- Aumentar el número de elementos, tanto en nuestra lista de dispositivos como de obstáculos, para mantener una variedad que nos permita diversificar nuestros diseños.

- Incluir nuevas funcionalidades en nuestra herramienta para facilitar el trabajo al usuario. La opción de poder exportar e importar escenarios ya creados es una utilidad muy eficaz que le ayudaría al usuario a evitar perder tiempo en la configuración de escenarios ya empleados anteriormente. Otra capacidad muy interesante de implementar sería dotar a la herramienta de la opción de sugerir zonas óptimas donde colocar los dispositivos de comunicación, en base al escenario elaborado y a los resultados obtenidos.

Apéndices

Lista de acrónimos

BER Bit Error Ratio. 13

GIF Graphics Interchange Format. 41, 45

GUI Graphical User Interface. 35–37

IoT Internet Of Things. 1

JPEG Joint Photographic Experts Group. 45

MIMO Multiple-input Multiple-output. 10

MVC Model View Controller. 55

PIL Python Imaging Library. 45

PNG Portable Network Graphics. 42, 45

POO Programación orientada a objetos. 33–35

RSRQ Reference Signal Received Quality. 13

Bibliografía

[1] "Figura de antenas: direccional y omnidireccional." [En línea]. Disponible en: https://www.geektopia.es/es/technology/2015/10/07/articulos/antenas-conoce-como-funcionan-aprende-colocar-tu-router-repetidor-senal-wi-fi.html

[2] "Imagen de reflexión." [En línea]. Disponible en: http://hyperphysics.phy-astr.gsu.edu/hbasees/phyopt/Fermat.html

[3] "Figura de refracción." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Índice_de_refracción

[4] "Imagen de difracción y dispersión." [En línea]. Disponible en: https://dot11ap.wordpress.com/cwna/radio-frequency-rf-technologies/reflection-refraction-diffraction-and-scattering/

[5] "Figura de shadowing." [En línea]. Disponible en: https://www.researchgate.net/figure/Geometry-of-non-line-of-sight-error-occurring-in-urban-environments_fig1_26547905

[6] "Herramienta cindoor." [En línea]. Disponible en: https://www.gsr.unican.es/es/CINDOOR.html

[7] "Remcom inc." [En línea]. Disponible en: https://es.remcom.com

[8] "Remcom inc, software wirelessinsite." [En línea]. Disponible en: https://es.remcom.com/wireless-insite-em-propagation-software

[9] "Remcom inc, wavefarer." [En línea]. Disponible en: https://www.remcom.com/wavefarer-automotive-radar-software

[10] "Remcom inc, xgtd." [En línea]. Disponible en: https://es.remcom.com/high-frequency-em-analysis-software-xgtd

[11] Real Academia de Ingeniería de España, "Diccionario español de ingeniería," 2014.

[12] Shannon y Weaver, "La teoría de la información."

[13] "Radiación electromagnética." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Radiación_electromagnética

[14] "Ondas electromagnéticas." [En línea]. Disponible en: https://blog.facialix.com/wp-content/uploads/2022/04/Ondas_Electromagneticas.pdf

[15] "Pérdida de trayectoria en el espacio libre." [En línea]. Disponible en: https://en.wikipedia.org/wiki/Free-space_path_loss

[16] "Ley de reflexión." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Reflexión_(física)

[17] "Leyes de refracción." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Refracción

[18] "Fenómeno de difracción." [En línea]. Disponible en: https://dot11ap.wordpress.com/cwna/radio-frequency-rf-technologies/reflection-refraction-diffraction-and-scattering/

[19] "Efecto del shadowing." [En línea]. Disponible en: https://www.zcom.com.tw/index/faq_qd/detail?id=105

[20] "What is decibels relative to one milliwatt (dbm)?" [En línea]. Disponible en: https://www.techtarget.com/whatis/definition/decibels-relative-to-one-milliwatt-dBm

[21] "Python." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Python

[22] "Programación orientada a objetos." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Programación_orientada_a_objetos

[23] "Tkinter." [En línea]. Disponible en: https://docs.python.org/3/library/tkinter.html

[24] "Math." [En línea]. Disponible en: https://docs.python.org/3/library/math.html

[25] "Pil/pillow." [En línea]. Disponible en: https://pillow.readthedocs.io/en/stable/

[26] "Salario desarrollador python galicia." [En línea]. Disponible en: https://es.talent.com/salary?job=desarrollador