Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Cálculo de la irradiación de paneles solares en matriz

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Víctor Espinosa Mateu

Tutor: Roberto A. Vivó Hernando

2017-2018

Resumen

En este trabajo se ha desarrollado una aplicación que permite diseñar centrales fotovoltaicas, las cuales pueden generar energía eléctrica utilizando paneles solares. Tras el diseño de las centrales, la aplicación ofrece la posibilidad de realizar una simulación en un intervalo de tiempo determinado. En esta simulación, considerando las sombras que se proyectan los paneles solares entre sí, se puede calcular la eficiencia de las centrales mediante el uso de métodos gráficos. Así pues, la intención de la aplicación desarrollada es poder ser una herramienta de apoyo a la hora de construir y optimizar centrales fotovoltaicas. El proyecto se ha realizado haciendo uso de metodologías ágiles y utilizando el entorno de desarrollo Qt y la librería gráfica Threejs.

Palabras clave: energía solar, centrales fotovoltaicas, paneles solares, diseño 3D, simulación, cálculo de sombras, métodos gráficos, eficiencia, Qt, Threejs

Abstract

The purpose of this project has been to develop an application that allows to design photovoltaic power plants, which can generate electricity through solar panels. After the design of the plants, the application offers the possibility of performing a simulation in a specific time interval. In this simulation, considering the shadows that are projected by solar panels, the efficiency of photovoltaic power plants can be calculated using graphical methods. Therefore, the intention of the developed application is to be able to serve as a support tool when constructing and optimizing photovoltaic power plants. This project has been made by using agile methodologies, Qt development environment and Threejs graphics library.

Keywords: solar energy, photovoltaic power plants, solar panels, 3D design, simulation, calculation of shadows, graphic methods, efficiency, Qt, Threejs

Resum

En aquest treball s'ha desenvolupat una aplicació que permet dissenyar centrals fotovoltaiques, les quals poden generar energia elèctrica utilitzant panells solars. Després del disseny de les centrals, l'aplicació ofereix la possibilitat de realitzar una simulació en un interval de temps determinat. En aquesta simulació, considerant les ombres que es projecten els panells solars entre si, es pot calcular l'eficiència de les centrals mitjançant l'ús de mètodes gràfics. Aleshores, la intenció de l'aplicació desenvolupada és poder ser una eina de suport a l'hora de construir i optimitzar centrals fotovoltaiques. El projecte s'ha realitzat fent ús de metodologies àgils i utilitzant l'entorn de desenvolupament Qt i la llibreria gràfica Threejs.

Paraules clau: energia solar, centrals fotovoltaiques, panells solars, disseny 3D, simulació, càlcul d'ombres, mètodes gràfics, eficiència, Qt, Threejs

Dedicatoria

Me gustaría dedicar este trabajo a mis padres, Ampa y Agus, a mi hermana Andrea y a mi pareja, Irene. Los cuatro siempre me han apoyado y ayudado durante el desarrollo de todos mis proyectos. Por otro lado, me gustaría mencionar a mi amigo Fran, por haber sido un gran compañero durante todo el máster, y a mi tutor Roberto, por haberme guiado durante todo el trabajo.

Índice de contenido

Índice de figuras

1. Introducción

En este apartado se comentará cuál es la motivación para realizar este trabajo. Tras ello, se indicarán los objetivos del proyecto a rasgos generales. Después se explicará el trabajo previo que ha habido a este trabajo. Luego se mencionará qué metodología se ha utilizado para la realización del proyecto. Y finalmente se indicará la estructura que va a tener este documento.

1.1. Motivación

El uso de energías renovables es cada vez mayor y va a continuar aumentando. Este tipo de energías presentan una ventaja clara respecto a las energías no renovables, ya que las segundas son limitadas y aparte son perjudiciales para el planeta en el que vivimos. Por lo tanto, la tendencia es alcanzar un futuro donde podamos abastecernos solamente de las fuentes de energía inagotables. Por ello, es necesario investigar en el ámbito de la energía y tratar de mejorar las herramientas y mecanismos que utilizamos para extraer energía de las fuentes renovables.

La informática tiene un papel muy importante en este proceso de mejora continua, pues puede ayudar a los ingenieros a diseñar soluciones de manera eficiente. Dentro de las distintas energías renovables que hay, este trabajo se ha centrado en la energía que podemos obtener gracias al Sol, la energía solar. Este tipo de energía se obtiene mediante el uso de paneles solares, los cuales se agrupan en centrales fotovoltaicas. La cantidad de energía generada depende de la cantidad de radiación solar que reciben los paneles solares. Por lo tanto, la producción puede variar en función de factores como el lugar, la fecha o las sombras que pueden proyectarse los paneles entre sí.

En concreto, en este trabajo se ha querido implementar una aplicación capaz de calcular la eficiencia de una central fotovoltaica. La idea es calcular, mediante una simulación, cuánta luz recibe un panel en un lugar y fecha determinados. Esta simulación incluye el cálculo de las sombras que se proyectan los paneles entre sí. Por lo tanto, el objetivo de esta aplicación es poder ayudar a los ingenieros encargados de diseñar una central fotovoltaica. Con ella podrían estudiar cómo de productivos serían los paneles solares con una determinada configuración. Esto les permitiría optimizar el uso de los paneles solares para conseguir la máxima energía posible con el mínimo uso de los recursos.

1.2. Objetivos

El objetivo de este trabajo es crear una aplicación que permita simular una central fotovoltaica, para posteriormente calcular la eficiencia de los paneles solares a lo largo de la simulación. Esto divide el funcionamiento de la aplicación en dos bloques muy diferenciados. Por una parte, se debe poder crear una central fotovoltaica y establecer su configuración. Y tras ello, sobre la central creada, se debe permitir establecer el periodo de simulación y calcular la eficiencia de los paneles a lo largo de ese periodo.

En el diseño de la central fotovoltaica, primero deberá establecerse el terreno donde se ubicarán los paneles solares. Este terreno podrá ser creado a partir de un mapa de alturas, permitiendo emular un terreno real. Aparte, dicho terreno deberá ser ubicado en una latitud y longitud específicas. Posteriormente, en la simulación, el Sol se posicionará en función de las coordenadas establecidas. Tras la configuración del terreno, se instalarán los paneles solares. Para ello se indicarán las dimensiones de los paneles, así como sus movimientos permitidos. Después, el programa deberá calcular el número máximo de paneles que pueden ubicarse en el terreno. Finalmente, el usuario establecerá el número de paneles que quiere añadir.

Una vez esté completamente configurada la central fotovoltaica, se podrá establecer el periodo de simulación. Conforme avance la simulación, el Sol se moverá acorde a la ubicación y las fechas establecidas. Los paneles, a su vez, se orientarán automáticamente hacia el Sol lo máximo que puedan. La capacidad de movimiento de los paneles dependerá de los parámetros establecidos previamente. Por otro lado, la simulación gráfica podrá activarse y desactivarse siempre que se desee. Tras acabar la ejecución, se obtendrá una función de eficiencia y su integral en base al intervalo simulado. La eficiencia de los paneles dependerá de la cantidad de luz solar que reciban, teniendo en cuenta que los paneles no siempre podrán estar orientados completamente hacia el Sol y que, en algunos momentos, se pueden tapar entre sí.

1.3. Trabajo previo

Antes de comenzar con el proyecto, el tutor de este trabajo, Roberto A. Vivó, ya había realizado un primer prototipo de la aplicación planteada. Este prototipo era una aplicación web que utilizaba la librería Threejs para hacer uso de gráficos 3D. El prototipo construía un terreno plano, y colocaba una serie de paneles sobre él. La configuración de la central fotovoltaica era fija y no podía ser personalizada por el usuario. Al iniciar la aplicación, una luz se movía alrededor del terreno simulando el Sol. Sin embargo, la luz se movía de forma circular

sin tener en cuenta ninguna localización específica ni la trayectoria real del Sol. Por otro lado, los paneles se orientaban siempre para estar mirando al Sol.

La parte importante del prototipo no era la escena descrita en el párrafo anterior, sino el hecho de que se permitía calcular la eficiencia de los paneles. Al pulsar sobre un panel con el ratón, en la pantalla se indicaba la eficiencia de ese panel en el momento de pulsarlo. Este valor dependía de si en la simulación era de día y de las sombras que proyectaban el resto de los paneles. La eficiencia se obtenía analizando la parte visible de los paneles mediante el uso de *shaders*, que se podrían definir como programas que permiten realizar cálculos gráficos [1]. En la Figura 1 se puede ver cómo era el prototipo. En la parte superior derecha de la pantalla se indica la eficiencia del primer panel de la segunda fila tras haber pulsado sobre él.

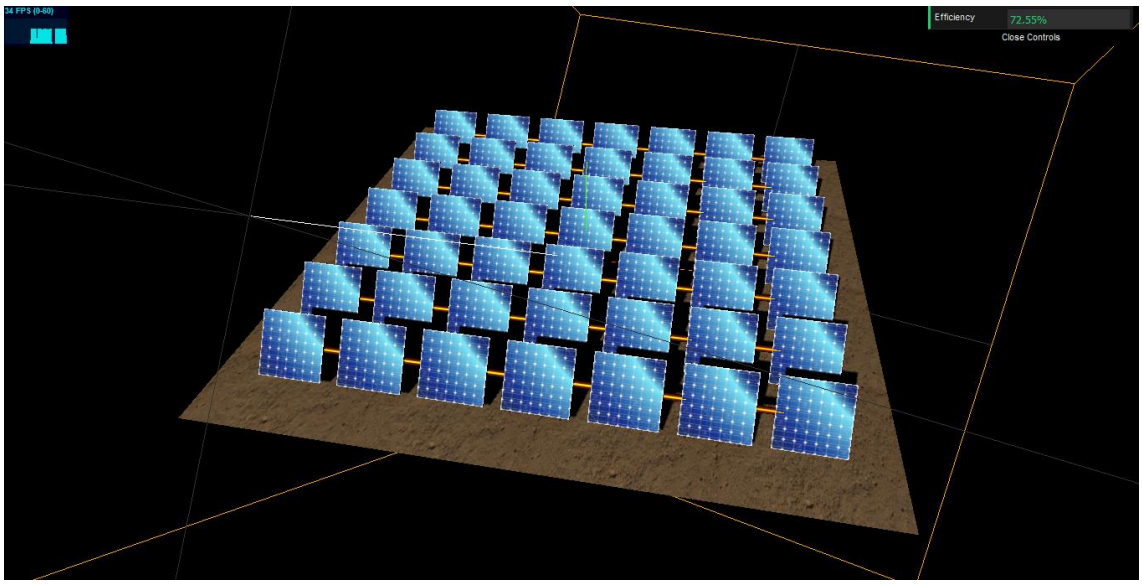

Figura 1. Prototipo inicial de la aplicación

La forma de calcular la eficiencia es la parte principal de este trabajo y motivó a realizar una aplicación que aportara valor al mercado actual haciendo uso de los algoritmos planteados. Todo el prototipo consta de 300 líneas de código aproximadamente. Este es solamente una aplicación sencilla que sirve para demostrar que los cálculos funcionan correctamente. A partir de él, se ha pretendido hacer una aplicación que pueda ser de utilidad a la hora de diseñar centrales fotovoltaicas. Esto implica que el usuario pueda diseñar una central fotovoltaica y establecer un periodo de simulación. A su vez, el programa debe ubicar la luz del Sol en función de la localización del terreno y simulando la trayectoria real del Sol. Finalmente, el programa debe registrar los resultados de los cálculos y estos deben poder visualizarse en la aplicación.

1.4. Metodología

La metodología utilizada en este proyecto se ha basado en las metodologías ágiles. La principal característica de este tipo de metodologías es que el desarrollo de los proyectos se plantea en iteraciones. Estas iteraciones son de corta duración, ya que suele oscilar entre las dos semanas y los dos meses. En cada iteración se desarrolla un bloque funcional de la aplicación, o se realiza toda la aplicación con un uso básico. Conforme se realizan iteraciones, se va ampliando la funcionalidad respecto a las iteraciones anteriores [2]. Al principio y al final de cada iteración se hace una revisión de los requisitos, con la posibilidad de cambiarlos si es necesario. Esta forma de trabajar tiene sus ventajas respecto a las metodologías tradicionales. En las segundas se intenta especificar todo desde el principio y se documenta mucho antes de desarrollar nada. En cambio, en las metodologías ágiles se asume que los requisitos pueden variar a lo largo del desarrollo y se prefiere aportar funcionalidad en fases tempranas para que el cliente pueda valorar [3].

Las metodologías más tradicionales suelen ser más útiles cuando los roles en el equipo de desarrollo están muy diferenciados. Es habitual que el analista hable con el cliente, y tras ello, el analista plasme las necesidades del cliente en una serie de documentos para que los programadores realicen la aplicación. Aquí estos roles se mezclan entre solamente dos personas. El cliente podría considerarse tanto el tutor del trabajo como el alumno. El primero es el que plantea el trabajo, pero el segundo también puede aportar nuevas ideas y enfocar la aplicación de otra manera distinta a la inicial. Luego, las responsabilidades típicas del analista y los programadores recaen todas en el alumno. Por ello, antes que crear muchos documentos para que todo el equipo sepa qué hacer, en este proyecto se ha focalizado en la continua revisión de los requisitos y del diseño de la aplicación.

Aparte de lo explicado sobre los roles, hay más justificaciones para haber escogido metodologías ágiles en este proyecto. Por un lado, la aplicación que se pidió era de gran envergadura. Debido al poco tiempo que se disponía para realizar el trabajo, era más prioritario documentar lo imprescindible y poder reaccionar de forma ágil a cambios en el planteamiento. Por otro lado, a priori no se sabía claramente cómo afrontar a nivel técnico ciertos problemas que planteaba la aplicación. Aparte, los requisitos eran genéricos, y siempre ha habido cierta tolerancia a hacer algunas modificaciones durante el desarrollo del proyecto. Y, como se puede observar en la filosofía de las metodologías ágiles, todo esto es más fácil de gestionar haciendo uso de ellas.

1.5. Estructura

En esta memoria se indicará todo el proceso realizado para el desarrollo de la aplicación propuesta. Primero, se comentará el estado del arte, que incluye un análisis del sector de la energía solar y una perspectiva de las aplicaciones actuales similares al proyecto propuesto. En segundo lugar, se hará un análisis del problema, lo que incluye la especificación de requisitos, el diagrama de casos de uso y la planificación del proyecto. Luego, se explicará el diseño de la aplicación, empezando por una perspectiva más general y acabando por los esbozos. Tras ello, se indicará la tecnología utilizada para el desarrollo del proyecto, así como la justificación de por qué se ha escogido dicha tecnología. Después, se indicará cómo se ha desarrollado la aplicación y cómo se ha implantado, así como las pruebas que se han realizado para verificar el correcto funcionamiento de la aplicación. Finalmente, se comentarán las conclusiones de este proyecto y las propuestas para trabajos futuros. Como anexo, al final del documento se ha añadido un glosario de términos.

2. Estado del arte

En este apartado se hará un análisis de contexto sobre los paneles solares. El objetivo de esta sección es poder concretar cómo debería ser un simulador de centrales fotovoltaicas que aporte valor en el mercado actual. Primero se mostrarán estadísticas sobre el uso de las energías renovables y en concreto, de la energía solar. Tras ello, se explicarán las características más relevantes de las centrales fotovoltaicas. Y finalmente se hará un estudio sobre el mercado actual y se concretará qué aplicaciones hay en este ámbito.

2.1. Uso de las energías renovables

Las energías renovables son aquellas que provienen de una fuente prácticamente inagotable. Los distintos tipos de energía son la eólica, geotérmica, hidroeléctrica, mareomotriz, solar, undimotriz, la biomasa y los biocarburantes [4]. Por otra parte, las energías no renovables son aquellas que provienen de recursos limitados. Entre ellos están los combustibles fósiles, que incluyen al carbón, el petróleo y el gas natural; y los combustibles nucleares, donde los más utilizados son el uranio y el plutonio [5]. Debido a que las energías no renovables son limitadas y a que sus procesos de obtención generan residuos y emisiones de gases contaminantes, la demanda de las energías renovables aumenta con la intención de alcanzar sostenibilidad y proteger el medioambiente [6].

Se estima que, en el año 2020, el 26% de la energía que se consume en el mundo provendrá de fuentes renovables. Dentro de estas, la energía solar es la segunda fuente que está en mayor crecimiento, por detrás de la energía eólica [7]. Respecto al uso de la energía solar en el mundo, en la Figura 2 se puede ver una gráfica que indica el porcentaje de uso de energía solar respecto a la energía total consumida. Dado el continuo crecimiento de la energía solar y de la imperiosa necesidad de utilizar energías renovables, la aplicación que se propone en este trabajo puede ser un producto útil y valorado en el mercado actual.

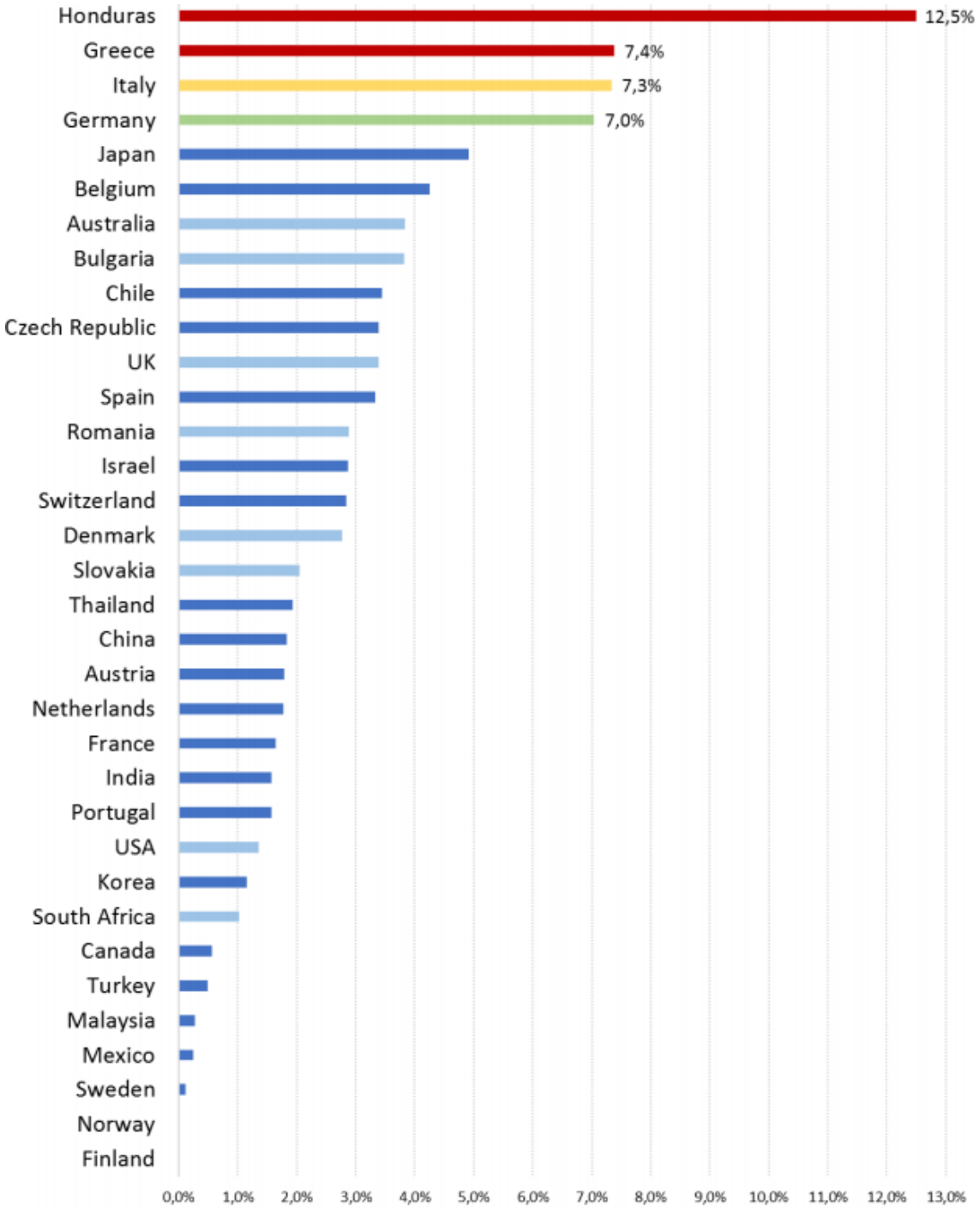

Figura 2. Porcentaje de uso de energía solar por países [8]

2.2. La energía solar y las centrales fotovoltaicas

La energía solar se basa principalmente en utilizar la radiación solar para transformarla en energía. Para realizar este proceso se utilizan los paneles solares, o también conocidos módulos solares. Hay dos tipos de paneles solares, los colectores solares (Figura 3) y los paneles fotovoltaicos (Figura 4). Los primeros se utilizan para calentar el agua en hogares y suelen ubicarse en los tejados de las casas. Los segundos sirven para generar electricidad y se sitúan en centrales fotovoltaicas [9]. Este proyecto se va a centrar principalmente en los paneles fotovoltaicos debido a que estos tienen un abanico más grande de posibles configuraciones. Los colectares solares, al estar sobre los tejados de las casas, no tienen tanta flexibilidad de configuración. Igualmente, ambos tipos de paneles solares se sustentan en los mismos principios y la aplicación propuesta para este trabajo podría servir también para los colectores solares.

Figura 3. Colectores solares

Figura 4. Paneles fotovoltaicos

En una central fotovoltaica, los paneles solares se posicionan mediante estructuras, generalmente hechas de aluminio [10]. Cada estructura puede contener uno o más paneles solares organizados en forma de matriz. En cada estructura puede haber mecanismos para inclinar y orientar los paneles solares. Posicionar los paneles correctamente es muy importante, ya que para aprovechar al máximo la luz solar, los paneles deben estar orientados perpendicularmente a los rayos del Sol [11].

2.2.1. Características de los paneles solares

Así pues, las estructuras de los paneles solares pueden configurarse para permitir uno o dos grados de libertad. Estos grados son la orientación acimut y la orientación cenit. La primera se refiere a la orientación de los paneles sobre la superficie terrestre; donde la orientación al Sur se indicaría con 0º, al Este con -90º, al Norte con 180º y al Oeste con +90º. Mientras que la orientación cenit se podría interpretar como la inclinación del panel solar; donde el tener el panel totalmente vertical mirando al horizonte serían 0º y el tenerlo horizontalmente mirando hacia el cielo serían 90º [11]. Como se ha comentado, estos movimientos se pueden hacer solamente si la estructura está preparada para ello. Los ajustes se pueden hacer un número de veces al año para ir ajustándose a las estaciones [12], o también se pueden hacer continuamente mediante el uso de sensores que capturen dónde está ubicado el Sol en cada instante [11].

Así que para aprovechar al máximo el Sol lo óptimo es disponer de un rastreador solar que permita actualizar la posición de los paneles continuamente. En caso de que este comportamiento quiera simularse, habría que calcular previamente cuál sería la posición del Sol en un determinado lugar y en una fecha concreta. Para calcular esto, el informe referenciado en [13] explica cómo calcular la posición. Dados una longitud, una latitud, una

fecha y una hora, el algoritmo permite calcular los ángulos que se visualizan en la Figura 5. Finalmente, otro aspecto relacionado con la posición de los paneles es el hecho de que los paneles pueden taparse entre sí, provocando que no todos reciban la misma cantidad de radiación solar. Este factor hay que tenerlo muy en cuenta y es la base de la aplicación propuesta en este trabajo.

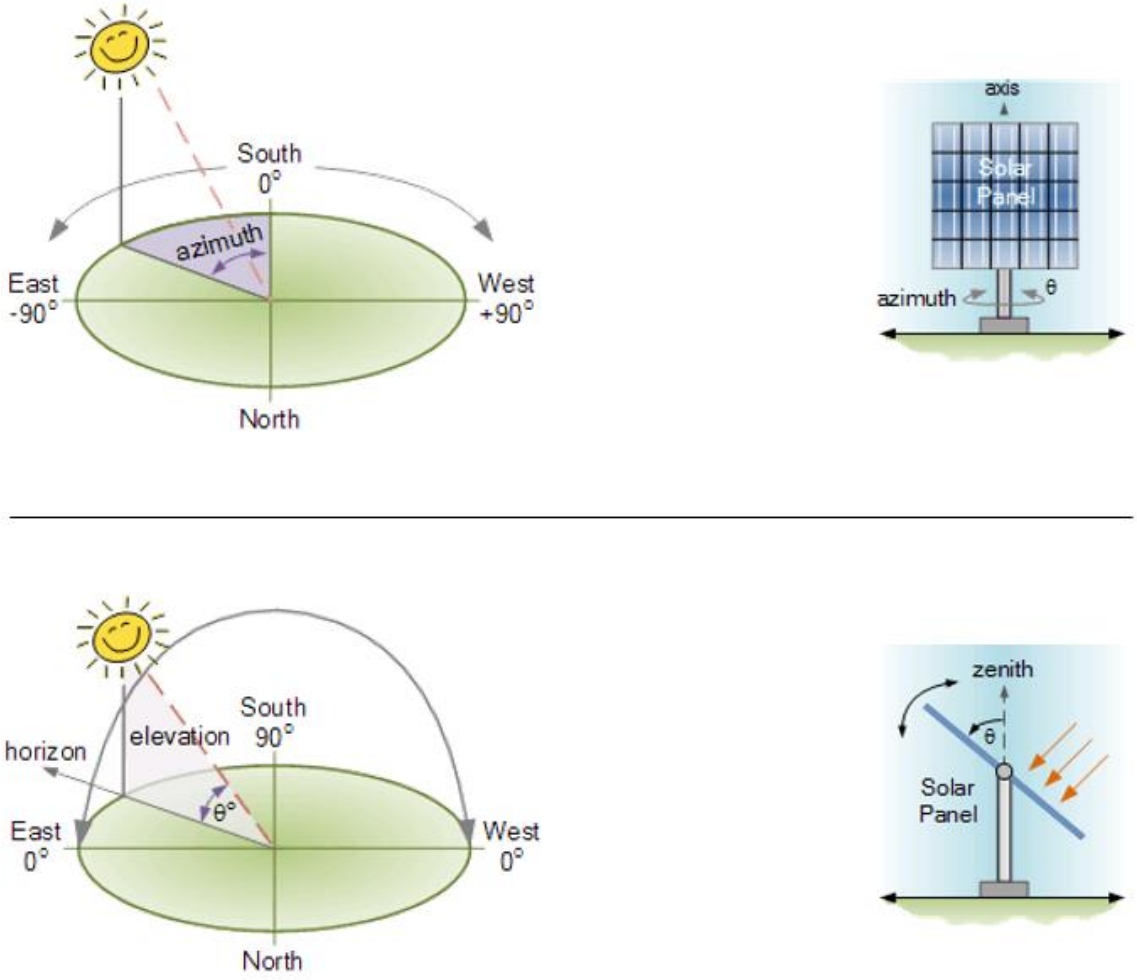

Figura 5. Grados de libertad de los paneles solares [11]

Junto al posicionamiento de los paneles, la potencia de los módulos es un factor clave. Dicha potencia se suele medir bajo unas condiciones óptimas y refleja el pico de potencia que puede llegar a generar. Estas condiciones son suponiendo una radiación del Sol de 1000 Vatios por metro cuadrado y una temperatura en los paneles de 25ºC [14]. Actualmente los módulos solares consiguen aprovechar la radiación del Sol con un 20% de eficiencia, aunque las investigaciones van avanzando y se están consiguiendo paneles de hasta 44,5% de eficiencia [15]. Así pues, los paneles solares suelen tener entre unos 200 o 300 Vatios de potencia, tal y como se puede observar en [16]. Para poder estimar cuánto puede aportar un panel solar, hay que tener en cuenta que una casa en España puede necesitar alrededor de unos 5.000 Vatios [17].

Aparte de la potencia, hay más parámetros relacionados con los paneles solares, como por ejemplo la tolerancia de potencia o el coeficiente de temperatura [18]. Pero estos factores ya no son tan relevantes y no están dentro del alcance de este documento. Así que, para este trabajo, lo más importante es conocer cómo puede aprovecharse la luz solar al máximo posible. Y para ello hay que pensar en la orientación de los paneles y en cómo se posicionan en el terreno para minimizar el hecho de que se tapen entre sí.

2.3. Aplicaciones para diseñar centrales fotovoltaicas

Una central fotovoltaica es una infraestructura compleja que requiere bastantes cálculos y simulaciones. Así que los ingenieros necesitan ayudarse de distintas herramientas para poder diseñar una central fotovoltaica. A continuación, se enumeran las herramientas más relevantes del mercado, las cuales se comentarán en el siguiente subapartado. Salvo las dos primeras herramientas, el resto se describen en mayor detalle en [19].

- **Microsoft Excel**
- **Etap**
- **Homer Pro**
- **PV F-Chart**
- **PVPlanner**
- **PVSyst**
- **Retscreen**
- **System Advisor Model (SAM)**
- **Solar Pro**

2.3.1. Crítica de las aplicaciones actuales

Respecto a Microsoft Excel, aunque sea una herramienta de cálculo genérica, se puede aprovechar su potencia para realizar todos los cálculos necesarios. Es una buena opción si los cálculos no son de mucha envergadura. Igualmente, todos los cálculos deben introducirse a mano, aumentando la probabilidad de cometer errores. Otro inconveniente es que es más costoso interpretar una central fotovoltaica utilizando solamente tablas y gráficas, sin disponer de un visor en 2D o en 3D de la central.

En lo que se refiere a Etap, es una herramienta orientada a cálculos eléctricos, la cual no solamente sirve para centrales fotovoltaicas. Es un programa complejo que permite todo tipo de cálculos y diseñar las infraestructuras eléctricas mediante esquemas. El precio de esta herramienta depende de la funcionalidad que se necesita, y para concretarlo hay que solicitárselo a la empresa.

Como se ha comentado antes, el resto de las herramientas están explicadas en la referencia [19] y en la Figura 6 se puede ver una gráfica comparativa de las mismas. Por ello, solamente

se van a comentar los aspectos más relevantes para este trabajo. Por ejemplo, algunas herramientas como Retscreen permiten calcular el coste de la central fotovoltaica y las características técnicas. Por otro lado, el cálculo de sombras es una funcionalidad interesante que aporta PVPPlanner. Aparte, esta herramienta permite ubicar la central utilizando un mapa vía satélite. Otro dato para tener en cuenta es que las aplicaciones Retscreen y SAM son gratuitas, lo que puede ser determinante a la hora de elegir un programa.

Finalmente está Solar Pro, que tal vez sea la mejor elección en lo que a sistemas fotovoltaicos se refiere. En la Figura 6 se puede apreciar que es el mejor valorado en [19], aparte de que es un programa exclusivo solo para energía solar. Su mayor potencial es que ofrece una simulación en 3D que hace cálculos en tiempo real y que permite calcular sombras. Esta última característica está presente en otras herramientas, pero no de una forma tan completa como Solar Pro. En la Figura 7 se puede observar su interfaz.

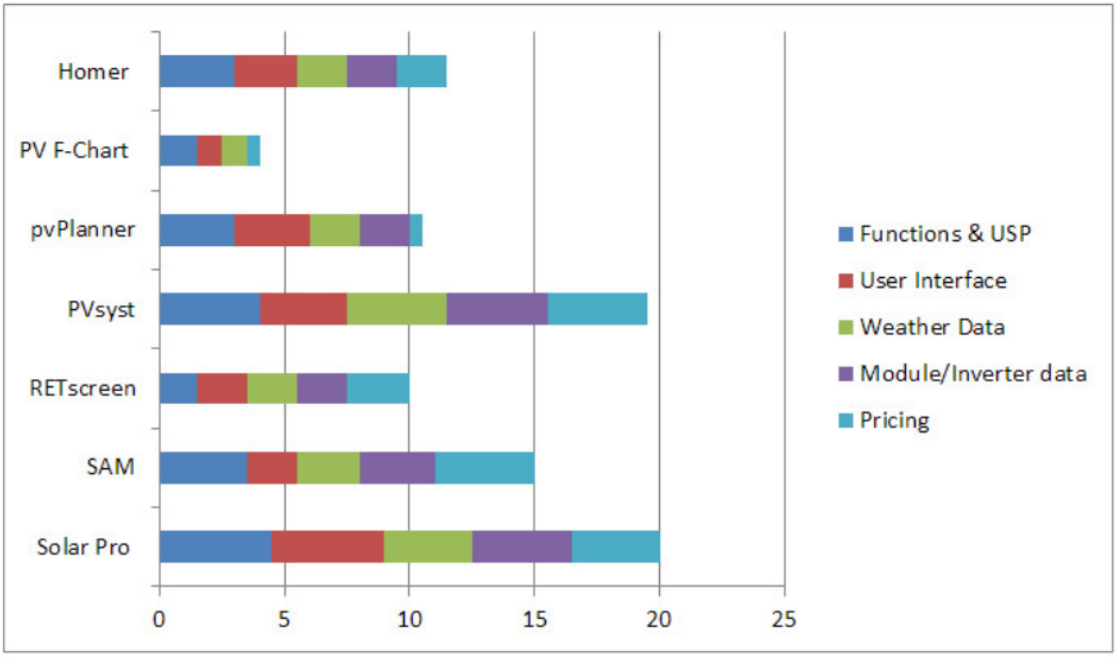

Figura 6. Gráfica comparativa de herramientas para centrales fotovoltaicas [19]

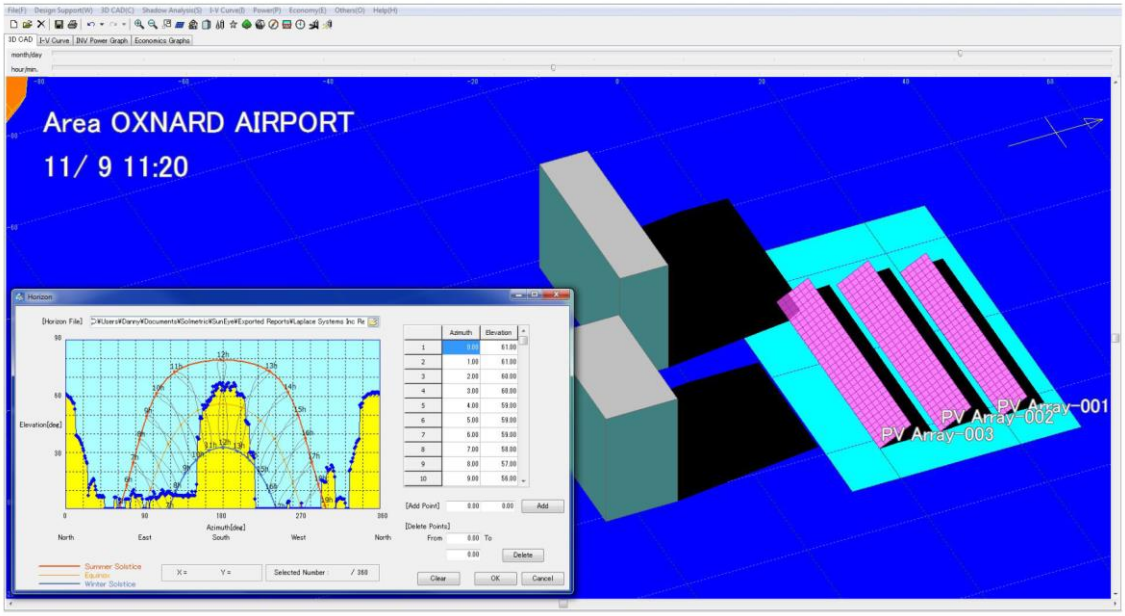

Figura 7. Interfaz de Solar Pro

2.3.2. Propuesta

En este proyecto se ha planteado una aplicación que permita hacer simulaciones de centrales fotovoltaicas y que calcule la eficiencia de los paneles solares teniendo en cuenta las sombras. La herramienta Solar Pro realiza esta funcionalidad y de forma más completa. Sin embargo, con pocos meses de desarrollo, la aplicación planteada en este trabajo podría empezar a competir con un programa realizado por una empresa dedicada al ámbito de la energía solar [20]. Por otro lado, el programa de este proyecto pretende ser más sencillo y más moderno que Solar Pro. En la Figura 7 se puede observar que Solar Pro dispone de una interfaz más clásica y más compleja que las aplicaciones que se desarrollan actualmente. Y esto no solamente se aplica a los menús, sino también a los gráficos que se utilizan para representar las centrales fotovoltaicas.

Por último, también hay que tener en cuenta el algoritmo para calcular la eficiencia. En la página web de Solar Pro no entran en mucho detalle de cómo realizan estos cálculos, pero comentan que los consiguen gracias a los gráficos 3D y a sofisticados algoritmos [21]. Aunque no se pueda saber qué tipo de algoritmos se utilizan realmente en Solar Pro, en este proyecto sí que se puede afirmar que el fundamento es muy básico y es escalable a cualquier complejidad de la escena. Es decir, que haya más sombras no complica el cálculo de la eficiencia, ya que esta se obtiene analizando la parte visible de los paneles mediante el uso de *shaders*, tal y como se ha comentado anteriormente.

3. Análisis del problema

En este bloque, se va a explicar el análisis del problema. Primero se mostrarán los requisitos de la aplicación. Luego se mostrará el diagrama de casos de uso. Tras ello se explicarán las alternativas posibles para realizar el trabajo. Y finalmente se indicará la solución propuesta, junto a la planificación y el presupuesto del proyecto.

3.1. Requisitos

En esta sección se especifican los requisitos que se definieron antes de la realización de la aplicación. Para documentar dichos requisitos, se utilizaron los estándares IEEE 830 y IEEE 29148, siguiendo los consejos que se encuentran en [22] y [23]. Algunos aspectos de este capítulo ya se han comentado anteriormente, pero se repiten aquí para ajustarse a los estándares.

3.1.1. Introducción

Propósito

Este capítulo está orientado desde una perspectiva docente y se dirige principalmente a los evaluadores de este trabajo, pero también puede ser útil para posibles futuros desarrolladores de la aplicación.

Ámbito del sistema

La aplicación pretende ser una herramienta de ayuda a la hora de diseñar centrales fotovoltaicas. La función de esta herramienta será calcular el rendimiento de los paneles solares con una disposición determinada, en un lugar y fecha concretos; mediante una simulación. Para ello se tendrá en cuenta la posición del Sol y las sombras que se proyectan entre los distintos paneles. La aplicación solamente se centrará en el rendimiento, y no se tendrán en cuenta aspectos como la energía que se puede llegar a generar o los costes que puede suponer realizar la central.

3.1.2. Descripción general

Funciones del producto

La aplicación debe permitir:

- Configurar el terreno donde se situará la central fotovoltaica
- Dimensionar y posicionar los paneles
- Establecer los movimientos permitidos de los paneles
- Indicar el periodo de simulación
- Realizar la simulación
- Obtener una función de rendimiento y su integral
- Visualizar en una gráfica el rendimiento de la central para un intervalo de tiempo

Características de los usuarios

La aplicación que se propone estará dirigida principalmente a ingenieros que trabajen diseñando centrales fotovoltaicas. Por lo que se espera que los usuarios del programa tengan un alto conocimiento de la energía solar y con capacidad para aprovechar al máximo una herramienta informática.

Restricciones

La única restricción que hay en este proyecto es que se cuenta con un tiempo limitado, por lo que hay que acotar considerablemente el alcance.

Suposiciones y dependencias

La aplicación debería poder ejecutarse en Windows. El uso de la aplicación en otros sistemas operativos (Linux, macOS) y en dispositivos móviles no es un requisito obligatorio, pero sería un aspecto que le añadiría valor a la aplicación.

Requisitos futuros

En el ámbito de la energía solar se requieren muchos cálculos y hay que tener en cuenta varios factores. Por ello, la aplicación puede extenderse considerablemente. La funcionalidad que podría añadirse es la siguiente:

- Soporte específico para simulaciones de colectores solares, que sirven para calentar el agua en hogares
- Cálculo de cuánta energía se puede producir
- Cálculo de costes

- Base de datos para almacenar distintos tipos de paneles solares
- Función para encontrar automáticamente la mejor configuración dado un terreno y unos parámetros económicos
- Mejoras en la interfaz
- Posibilidad de seleccionar un terreno mediante vista satélite

3.1.3. Requisitos específicos

- Se desarrollará una aplicación de escritorio que funcione para el sistema operativo Windows.
- La aplicación contará con un visor para poder visualizar la central fotovoltaica en 3D. El usuario podrá manejar el visor para ver la escena desde cualquier ángulo.
- La aplicación permitirá configurar el terreno donde se ubicará la central fotovoltaica. Dicho terreno se podrá visualizar en el visor una vez haya sido configurado. Los parámetros que se podrán especificar serán:
    - Ancho y largo del terreno.
    - Forma del terreno, que se podrá determinar indicando que el terreno es plano o a través de un mapa de alturas.
    - Longitud y latitud del terreno.
- La aplicación permitirá configurar los paneles de la central fotovoltaica. Los paneles se podrán visualizar sobre el terreno una vez se hayan configurado. Los parámetros que se podrán especificar son:
    - Número de filas y columnas que tendrán las matrices de paneles. Cada matriz representa una sola estructura que agrupa un número de paneles determinado.
    - Ancho y largo de cada panel solar.
    - Grados de libertad de las matrices de paneles:
        - Orientación
        - Inclinación
    - Disposición de los paneles en el terreno
- La aplicación permitirá configurar el intervalo de simulación que se desea realizar. Dicho intervalo no tendrá en cuenta los años, solamente se podrá especificar día y mes. Esto es debido a que los cálculos que se desean realizar son independientes del año.
- La simulación podrá visualizarse en el visor 3D o podrá ejecutarse solamente mostrando una barra de progreso. Ambos tipos de simulaciones podrán ser pausadas, reanudadas o paradas por completo en cualquier momento. Para la simulación con visualización se podrá establecer la velocidad a la que se ejecutará la simulación.
- La simulación recolectará el rendimiento de cada panel solar en cada instante simulado. Se calculará dicho rendimiento cada un número determinado de minutos.

Este rendimiento se calculará en función del ángulo en el que se proyectan los rayos solares en el panel y de las sombras que se proyecten entre sí los paneles. Con los valores extraídos, se construirá una función para predecir el rendimiento de los paneles en cualquier momento comprendido entre el tiempo simulado. A partir de dicha función, se calculará la integral.

- Con la función calculada, la aplicación permitirá visualizar una gráfica indicando el rendimiento a lo largo del tiempo. Los parámetros de entrada serán:
    - Un rango de fechas, sin tener en cuenta los años, que esté comprendido dentro de la etapa simulada.
    - Qué rendimiento se desea analizar: el de una matriz de paneles concreta, o el de la central en general.
- Con la función calculada, la aplicación también permitirá exportar una tabla en formato CSV indicando el rendimiento a lo largo del tiempo.
- De entre todas las características de calidad, las más relevantes para esta aplicación son: funcionalidad, fiabilidad, robustez, eficiencia, usabilidad y extensibilidad.

3.2. Diagrama de casos de uso

En la Figura 8. Diagrama de casos de uso se muestra el diagrama de casos de uso de la aplicación. Este diagrama se definió acorde a las normas del lenguaje de modelado UML (Unified Modeling Language) y haciendo uso de la herramienta Modelio. Aquí se puede observar que solamente hay un actor implicado: el ingeniero que utiliza la aplicación para diseñar centrales fotovoltaicas. Dicho actor debe poder realizar las acciones que se aprecian en la figura. A rasgos generales, el usuario de la aplicación debe poder: diseñar una central fotovoltaica, navegar por la escena 3D, simular la central en un intervalo determinado, obtener los resultados de la simulación y visualizar dichos resultados. Aparte, también se permite el guardado y cargado de las centrales diseñadas y los resultados.

Aparte de las acciones directamente relacionadas con el actor, se enriqueció el diagrama con el uso de las relaciones de inclusión y extensión que proporciona UML. Estas asociaciones se indican utilizando las etiquetas “<<Include>>” y “<<Extend>>”, respectivamente. Las inclusiones indican una acción que forma parte de la acción principal. Por ejemplo, la acción de diseñar una central fotovoltaica incluye el diseño del terreno y de los paneles. Por otro lado, la extensión sirve para especificar que una acción puede incluir opcionalmente a otra acción. Por ejemplo, el diseño del terreno puede ser extendido con la acción de cargar un mapa de alturas.

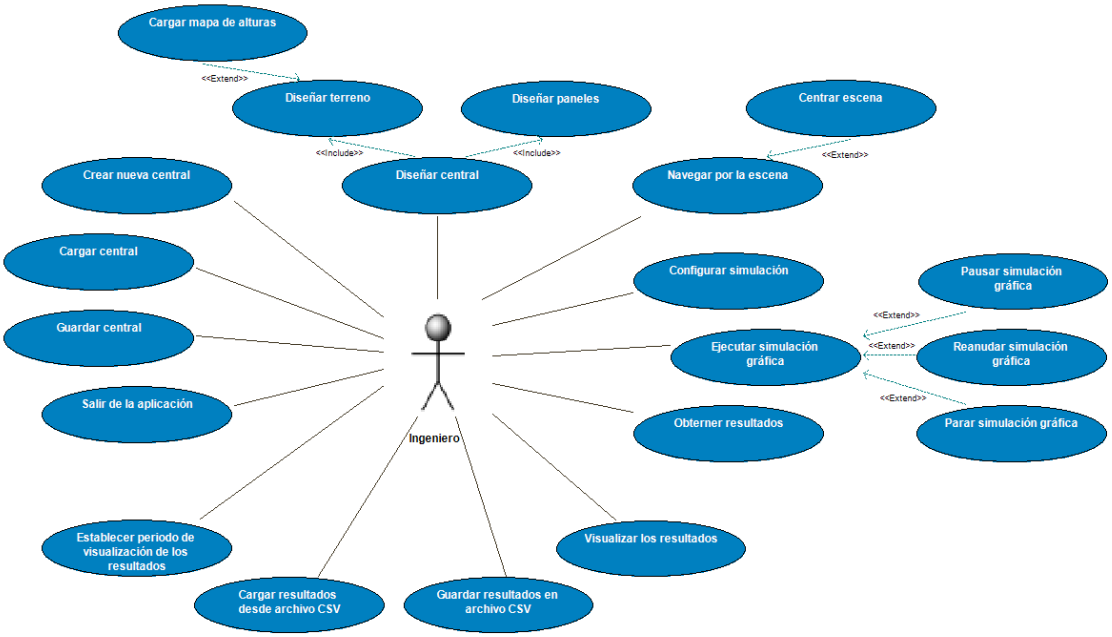

Figura 8. Diagrama de casos de uso

3.3. Identificación y análisis de soluciones posibles

El problema que tenía que resolverse en este trabajo era calcular la eficiencia de los paneles solares en una central fotovoltaica. Ante esto, surgieron dos soluciones claras. La primera era calcular de forma numérica cómo se tapan los paneles entre sí y cuál es el ángulo que forman los rayos solares con los paneles. Esta solución parece factible y los cálculos serían rápidos. Sin embargo, los algoritmos serían costosos de implementar. Aparte, si se quieren añadir detalles a la central fotovoltaica, como por ejemplo el hecho de crear el terreno a partir de un mapa de alturas, los algoritmos se complicarían más todavía.

La segunda alternativa, utilizada en el prototipo, era simular gráficamente la central fotovoltaica para determinar cuánta luz reciben los paneles solares. Esta opción es más costosa a nivel computacional, pero el algoritmo es sencillo de implementar y es capaz de adaptarse a cualquier cambio en el entorno. Esto quiere decir que, aunque el entorno de la central sea muy complejo (terreno escabroso, otros elementos que hagan sombra, etc.), el algoritmo no tendría que cambiarse y funcionaría exactamente igual. Por este motivo, y dado que actualmente se dispone de mucha potencia gráfica, se escogió la segunda opción frente a la primera.

3.4. Solución propuesta

Por lo tanto, la solución propuesta fue analizar la parte visible de los paneles mediante métodos gráficos. Aparte, esto tenía que englobarse en una aplicación usable que permitiera plantear varios escenarios. En el análisis ya se concretó que la aplicación tendría seis menús. El primero permitiría guardar y cargar centrales fotovoltaicas creadas por los usuarios. El segundo y el tercero estarían dedicados a la configuración del terreno y los paneles, respectivamente. El cuarto y el quinto permitirían realizar la simulación y visualizar los resultados obtenidos. Por último, el sexto menú simplemente sería para poder salir de la aplicación.

3.5. Plan de trabajo

En concreto, para gestionar las tareas de este proyecto se ha utilizado la metodología ágil Kanban. Se ha escogido esta metodología porque es sencilla y es adecuada para un solo desarrollador. Esta técnica consiste básicamente en disponer de un tablero donde se indican las tareas a realizar. El tablero está dividido en columnas, donde cada columna representa un estado en el que puede estar una tarea. Las columnas más comunes representan que una tarea está pendiente de hacer, que está realizándose o que ya se ha realizado; aunque se pueden establecer las columnas que se deseen. Este método permite organizarse fácilmente y ver de forma visual y rápida el estado del proyecto.

Para este trabajo en particular, se ha utilizado la herramienta Trello, que permite crear y gestionar los tableros de la metodología Kanban. En la Figura 9 se puede ver el tablero del proyecto, donde se han definido las columnas “Pendiente”, “Desarrollo”, “Pruebas” y “Hecho”. Aquí, aparte de los estados básicos, se ha añadido la categoría “Pruebas”. Esta columna solamente se ha utilizado para tareas de desarrollo de *software* y ha servido para indicar que se estaba probando la funcionalidad implementada. Para planificar el proyecto, cada tarea contaba con una fecha final para su realización, facilitando la gestión del tiempo. Además, estas fechas se han ido actualizando en función de los imprevistos que iban surgiendo.

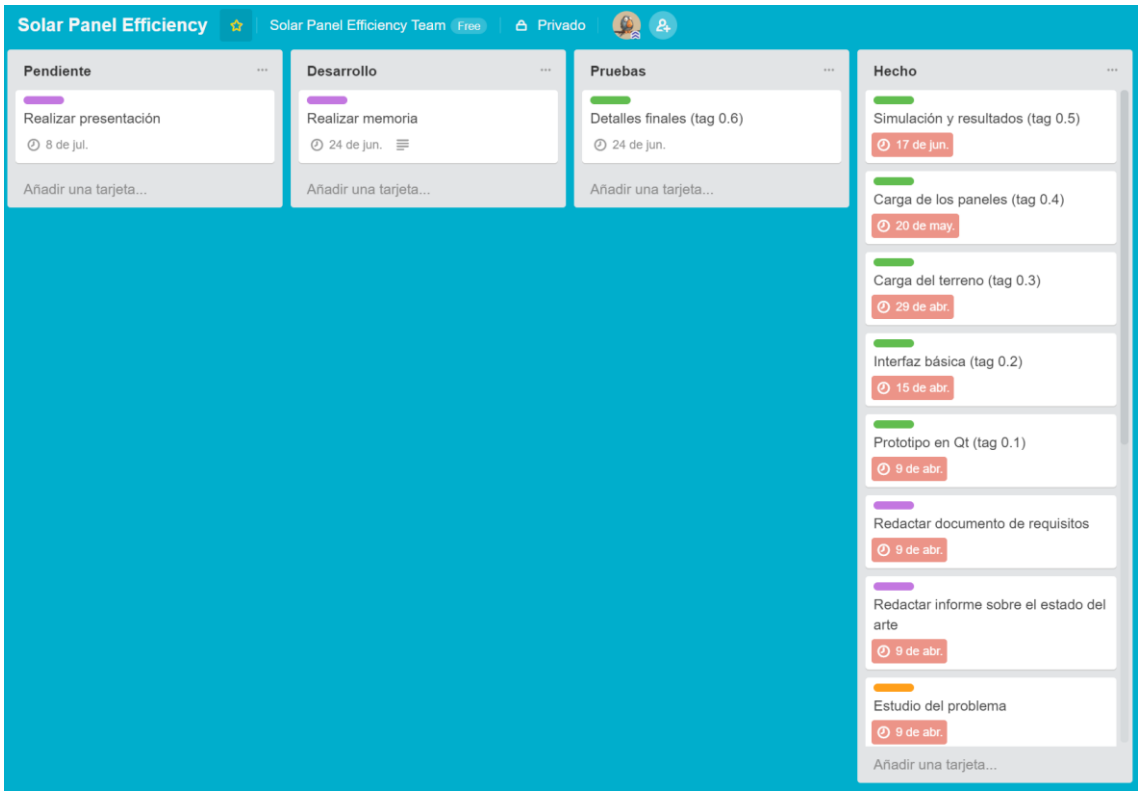

Figura 9. Tablero Kanban del proyecto

Aparte de las fechas, en la imagen se puede observar que las tareas se han clasificado por colores. El verde representaba los bloques funcionales de la aplicación que se tenían que implementar, el morado se utilizaba para tareas de documentación, y el naranja se reservaba para otros propósitos que no encajaran con las dos categorías anteriores. De aquí, lo más importante son las tareas verdes, donde cada bloque funcional implementado correspondería con una iteración, concepto propio de las metodologías ágiles. Con cada iteración, la funcionalidad se probaba, se lanzaba una nueva versión del programa y se revisaban los requisitos y el diseño de la aplicación.

3.6. Presupuesto

Respecto al presupuesto del proyecto, solamente se tuvieron en cuenta las horas que iba a realizar el único desarrollador de la aplicación, ya que no se necesitaba ningún material específico ni comprar ninguna licencia. Como se ha comentado, el desarrollo de la aplicación se ha dividido en diferentes bloques funcionales, los cuales son:

- **Prototipo en Qt.** Emular el prototipo de la aplicación con la tecnología elegida para el proyecto.

- **Interfaz básica.** Implementar la interfaz principal de la aplicación.
- **Carga del terreno.** Implementar la carga del terreno por parte del usuario, tanto un terreno plano como un terreno a partir de un mapa de alturas. Aparte, este bloque incluye la implementación de guardado y cargado de las centrales fotovoltaicas diseñadas por el usuario.
- **Carga de los paneles.** Implementar la carga de paneles por parte del usuario.
- **Simulación y resultados.** Implementar la configuración de la simulación, así como la simulación en sí y la visualización de los resultados obtenidos.
- **Detalles finales.** Pulir detalles que mejoren el valor de la aplicación y su usabilidad.

Tal y como se ha comentado, cada bloque funcional se ha considerado una iteración. Aunque algunos bloques sean más costosos que otros, se estableció un tiempo medio de dos semanas por iteración, dedicando aproximadamente 30 horas de trabajo por cada semana. Por lo tanto, dado que hay 6 bloques funcionales, se planteó una duración de 12 semanas, es decir, 360 horas.

4. Diseño de la solución

En este apartado se va a especificar el diseño de la solución. Primero se mostrará la arquitectura del sistema desde una perspectiva general. Tras ello se entrará en más detalle y se enseñarán una serie de diagramas y diseños que permiten ver cómo se ha enfocado el desarrollo de esta aplicación.

4.1. Arquitectura del sistema

En la Figura 10 se puede ver la arquitectura del sistema. Al tratarse de una aplicación de escritorio, la arquitectura es sencilla. En el gráfico se ha reflejado que la aplicación hace uso de tres librerías. La primera, Three.js, proporciona los métodos necesarios para poder trabajar con gráficos 3D. Luego está la librería OrbitControls.js que aumenta la funcionalidad de Three.js aportando controles básicos para que el usuario pueda moverse por la escena 3D. Finalmente, se ha utilizado Suncalc.js, que permite calcular la posición del Sol en función de una fecha y un lugar determinados. Aparte de las librerías, la aplicación se conecta con una base de datos en la cual se almacenan los resultados calculados.

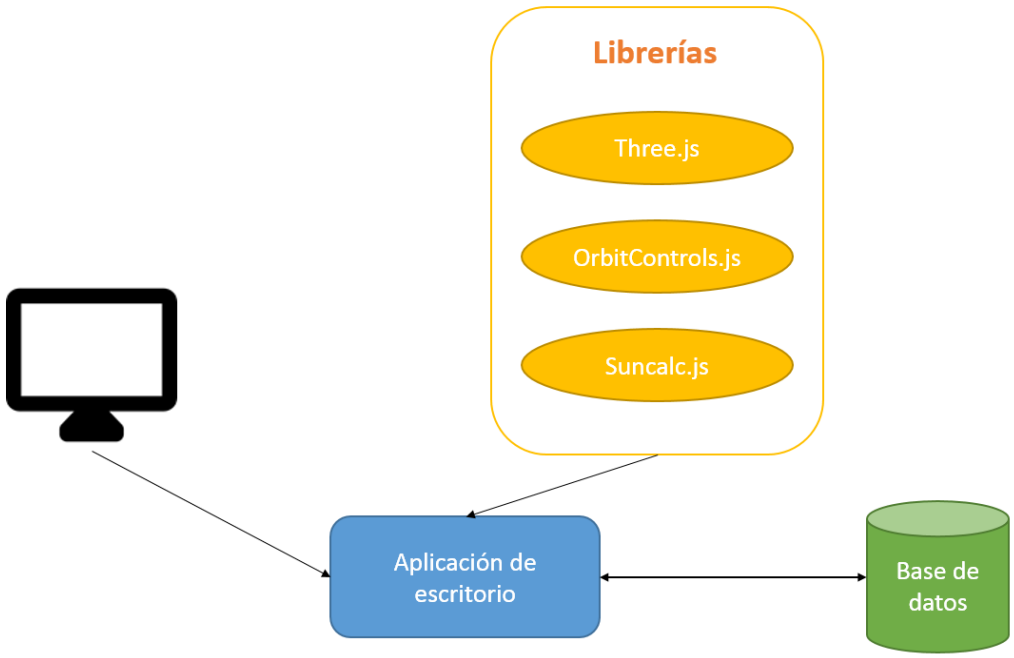

Figura 10. Arquitectura del sistema

4.2. Diseño detallado

Tras haber mostrado la arquitectura general de la aplicación, ahora se entrará en más detalle sobre el diseño de la aplicación. Esto incluye el diagrama de clases de la aplicación, la estructura de la base de datos, las ventanas planteadas para la aplicación, la organización de los ficheros y los diseños de la interfaz de la aplicación.

4.2.1. Diagrama de clases

En la Figura 11 se muestra el diagrama de clases, el cual ayuda a vislumbrar los elementos principales de la aplicación y su relación entre ellos. En este diagrama se puede observar que la entidad principal es la escena 3D de la aplicación. Al iniciar la aplicación, la escena solamente dispone de una luz que representa el Sol. Tras ello, el usuario puede añadir un solo terreno a la escena. Si se quiere definir la irregularidad del terreno, este puede disponer de un mapa de alturas. Luego, se pueden añadir los paneles, los cuales se estructuran en matrices. Una vez está toda la escena construida, se puede realizar una simulación en un intervalo determinado. Finalmente, de esta simulación se pueden obtener una serie de resultados.

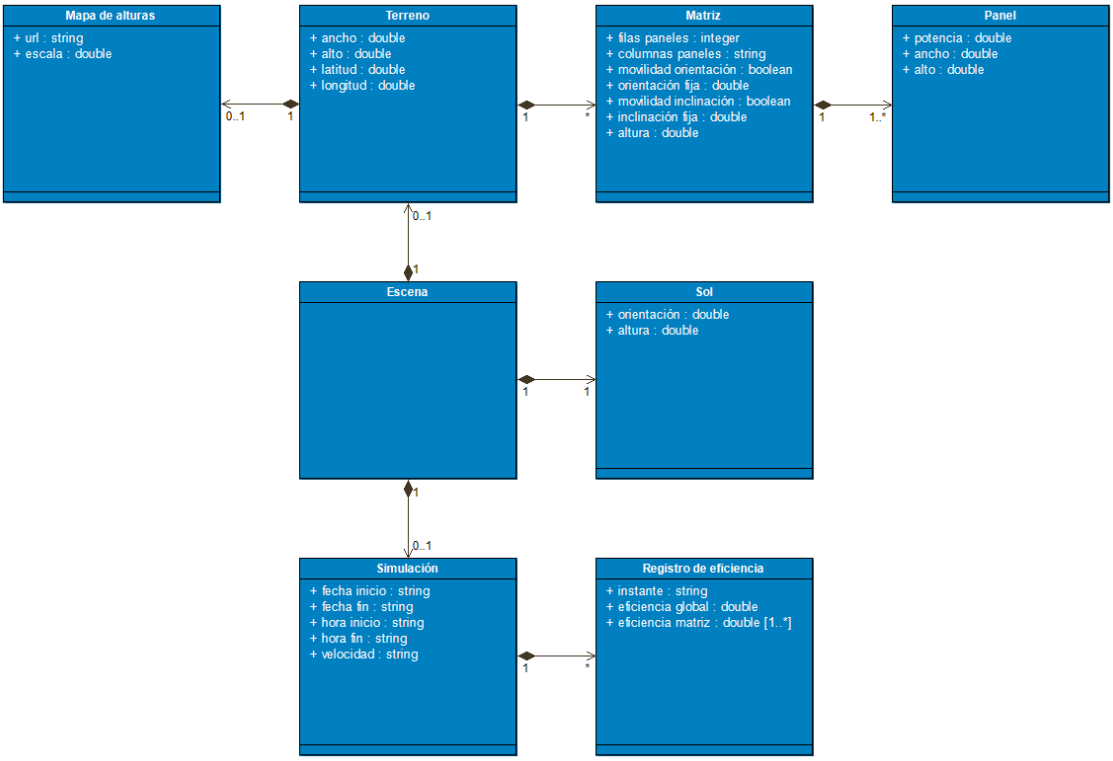

Figura 11. Diagrama de clases

4.2.2. Estructura de la base de datos

En principio, la aplicación no necesita ninguna persistencia. Los datos que pueden guardarse son las centrales fotovoltaicas y los resultados de las simulaciones, y esto se gestiona mediante ficheros. Sin embargo, durante la ejecución de la aplicación, la cantidad de resultados que se pueden generar es demasiado grande como para almacenarlos en la memoria RAM del ordenador. Por lo tanto, para poder almacenar tanta información, se ha preferido utilizar una base de datos. Conforme se van obteniendo resultados, estos se van almacenando en dicha base de datos. Una alternativa a esto podría haber sido utilizar un fichero temporal, pero esto habría dificultado las consultas a los resultados. En cambio, una base de datos dispone de un lenguaje específico para extraer información.

Por lo tanto, como la base de datos solamente es necesaria para almacenar resultados, su estructura es sencilla. Solamente se ha diseñado una tabla donde cada fila representa un instante de tiempo. Así pues, en cada fila se almacena: el instante de tiempo, la eficiencia de cada panel en ese momento, y la media de las eficiencias para disponer de la eficiencia global. En la Figura 12 se puede observar un ejemplo de cómo sería esta tabla. Hay que tener en cuenta que los instantes de tiempo se representan utilizando *timestamps*. Los *timestamps* son

números enteros para representar una fecha. En concreto, el número representa los milisegundos que han pasado desde las 00:00:00 del 1 de enero de 1970. Por otro lado, la eficiencia se guarda considerando 1 la eficiencia máxima y 0 la mínima.

timestamp	media	panel_1	panel_2	panel_n
993074400000	0	0	0	0
993075600000	0.8	0.8	0.8	0.8
993076800000	0.8	0.7	0.8	0.9
993078000000	1	1	1	1

Figura 12. Estructura de base de datos

4.2.3. Ventanas de la aplicación

En la Figura 13 se pueden observar las distintas ventanas que se han planteado para cumplir con los requisitos de la aplicación. Aquí se puede apreciar que la navegación de la aplicación es simple. Hay seis funcionalidades muy claras y cada una de ellas está ubicada en una ventana. Dos de ellas son típicas en una aplicación: la gestión de ficheros y la posibilidad de salir de la aplicación. Las otras cuatro son completamente específicas de la aplicación y consisten en: la carga del terreno, la carga de los paneles, la simulación y la visualización de los resultados.

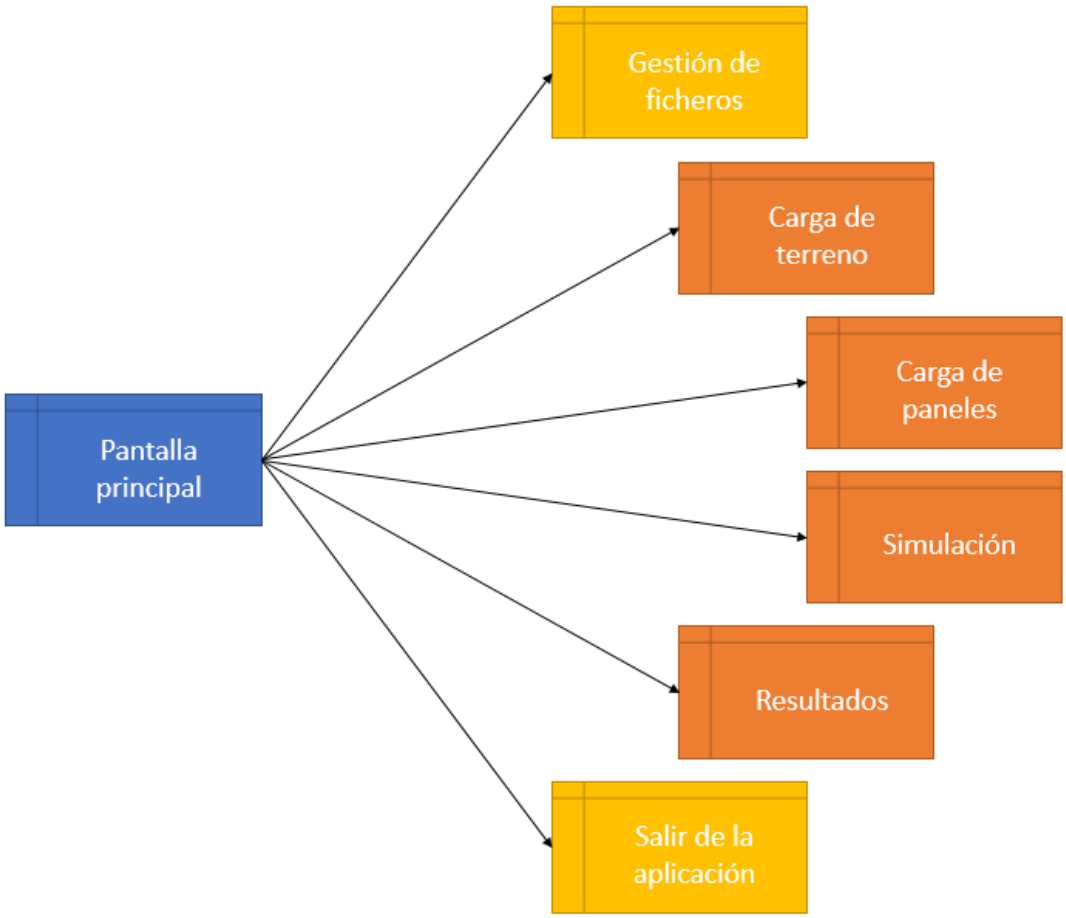

Figura 13. Diagrama de ventanas

4.2.4. Estructura de ficheros

En un primer nivel, la estructura de ficheros de este proyecto se ha planteado siguiendo la estructura típica de aplicaciones informáticas. Así pues, las carpetas y archivos del proyecto son:

- **doc/:** Carpeta que contiene la documentación del proyecto, como por ejemplo el análisis de requisitos.
- **img/:** Carpeta que contiene las imágenes del proyecto.
- **lib/:** Carpeta que contiene las librerías del proyecto. Estas librerías se han indicado en el apartado 4.1.
- **src/:** Carpeta que contiene el código del proyecto.
- **README.md:** Fichero que explica de qué va el proyecto.

Aquí, la carpeta más relevante es “src”, por lo que a continuación se va a explicar qué directorios se han utilizado para organizar el código. Cabe mencionar que los ficheros que están directamente dentro de la carpeta “src” son archivos relacionados con la configuración del proyecto.

- **canvas/:** Carpeta que contiene los archivos relacionados con el diseño 3D y el cálculo de la eficiencia. Por lo tanto, la parte más importante del código está en este directorio.
- **components/:** Carpeta que contiene componentes personalizados para su reutilización en toda la aplicación. Por ejemplo, aquí se definen cómo son los botones o los campos de texto.
- **global/:** Carpeta que contiene ficheros que afectan de forma global a toda la aplicación. En estos archivos se incluyen configuraciones de la aplicación y funciones de utilidad.
- **io/:** Carpeta que contiene el código que se encarga de la gestión del guardado y cargado de ficheros.
- **main/:** Carpeta que contiene el código de la interfaz de la pantalla principal.
- **submenus/:** Carpeta que contiene el código del resto de interfaces.

Dentro de las carpetas “components”, “main” y “submenus”, cada interfaz cuenta con dos ficheros, uno que describe el aspecto visual y otro que se encarga de la funcionalidad de esa interfaz.

4.2.5. Esbozos de la interfaz

Antes de desarrollar nada, se realizaron una serie de esbozos para diseñar cuál sería el aspecto de la aplicación. Estos esbozos se hicieron con Pencil, una herramienta específica para realizar diseños, también conocidos como *mockups*. Aunque esta herramienta permita diseños que se asemejen completamente al resultado final de la aplicación, se ha preferido utilizar los elementos de dibujo que tienen un formato de esbozo. Esto es debido a que solamente se quería concretar qué campos serían necesarios para cumplir con los requisitos de la aplicación.

Los colores, el diseño de los campos y cualquier aspecto puramente estético no se ha reflejado en estos documentos. Este tipo de diseños más detallistas son más útiles para un cliente sin perfil técnico, que quiere saber de antemano cómo va a quedar realmente la aplicación. Pero en este proyecto, y dado que se han utilizado metodologías ágiles, era preferible hacer un esbozo meramente funcional sin perder tiempo en hacer una maquetación perfecta. Aparte, con cada iteración en la planificación, estos diseños se han revisado y modificado para que se

ajusten a las nuevas necesidades. Por lo tanto, haberse preocupado de la estética durante todo el proyecto habría sido contraproducente.

En los diseños, manteniendo la sencillez, no hay ventanas que se abran a partir de otras; por lo tanto, todo se gestiona desde la pantalla principal. En concreto, para establecer un estilo moderno, los menús no son ventanas emergentes, típicas de las aplicaciones clásicas. Sino que son menús laterales que aparecen y desaparecen con una animación y que no cubren del todo la pantalla principal. Este tipo de diseño aporta un aspecto más unificado que permite una navegación más clara y sencilla. Aparte, cabe añadir que el texto utilizado en la aplicación está escrito en inglés. El motivo de esto es que se ha preferido realizar una aplicación que utilice el idioma más común en el ámbito de la informática. En futuras iteraciones se podría traducir a más idiomas.

Por último, hay que comentar cómo funcionan los campos que permiten al usuario introducir los datos. Cada uno de los campos cuenta con una explicación que aparece cuando se mantiene el ratón sobre él. Aparte de esta descripción, si el campo está vacío, se muestra un ejemplo o un patrón de cómo debe ser el valor. Además, mientras se escribe en ese campo, el programa controla continuamente si la entrada es correcta o puede ser correcta en un futuro. Por ejemplo, si se espera un valor numérico, las teclas que representan letras se ignoran. Finalmente, los aspectos de los campos que no se pueden controlar al instante, son validados cuando van a ser utilizados. Por ejemplo, si el ancho máximo de terreno permitido es de 50 metros y se ha introducido 60, la aplicación indica el error. Para ello, cada campo que no cumple con la validación se rodea de color rojo. Y ahora, tras explicar todo esto, se van a mostrar los diseños de la aplicación. Por cada uno de los siguientes subapartados, habrá una figura y su explicación. Por claridad, estas figuras no se han referenciado en el texto.

Pantalla principal

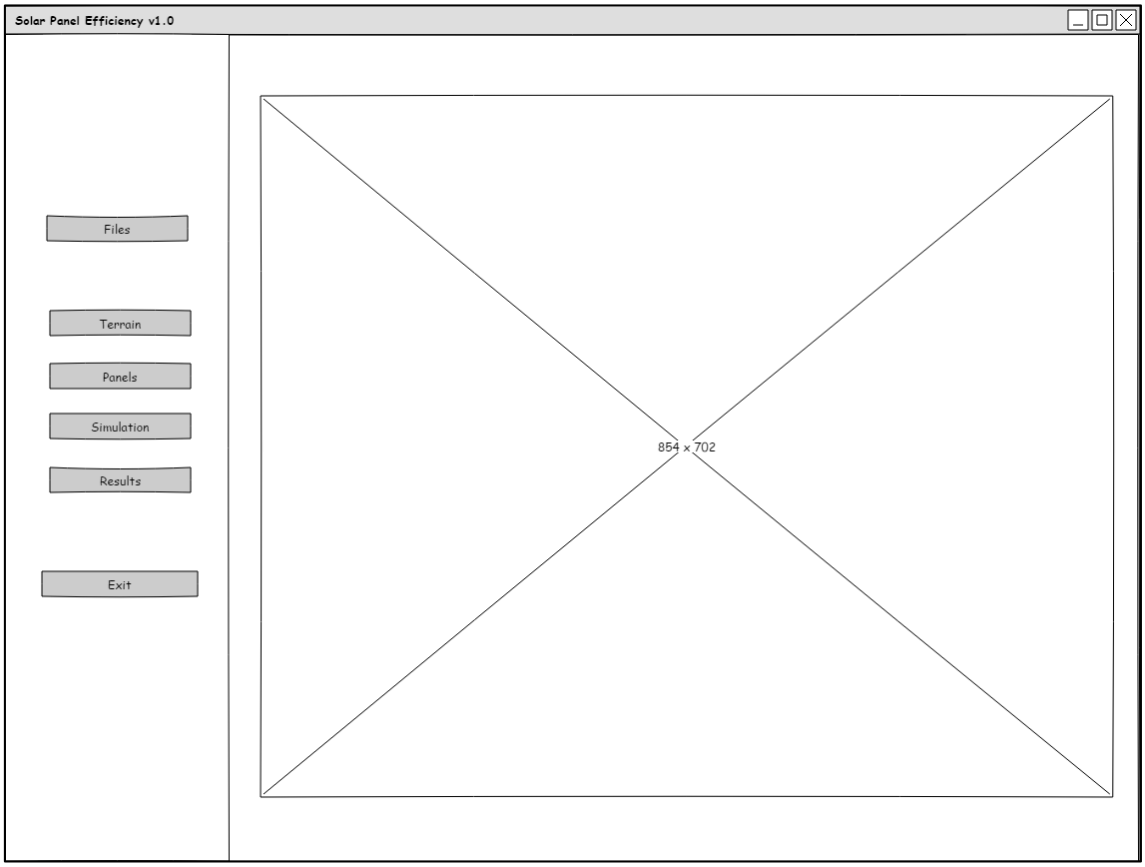

Figura 14. Diseño de la pantalla principal

La pantalla principal se compone de un menú a la izquierda y de una escena vacía a la derecha. El menú contiene 6 botones, los cuales abren cada una de las ventanas indicadas en el apartado 4.2.3. Aunque, como ya se ha mencionado, estas ventanas no son las ventanas convencionales, sino que son menús desplegables que tapan parte de la escena. A pesar de que los botones mostrados tienen texto, en la aplicación real estos botones son iconos. Esto también implica que el menú es más estrecho de lo que se visualiza en el diseño. En los siguientes esbozos, estos botones se han omitido para que los diseños sean más claros, pero en la aplicación real siempre está visible este menú principal. Respecto a la escena, al estar vacía, simplemente se ve un rectángulo blanco.

Menú de gestión de ficheros

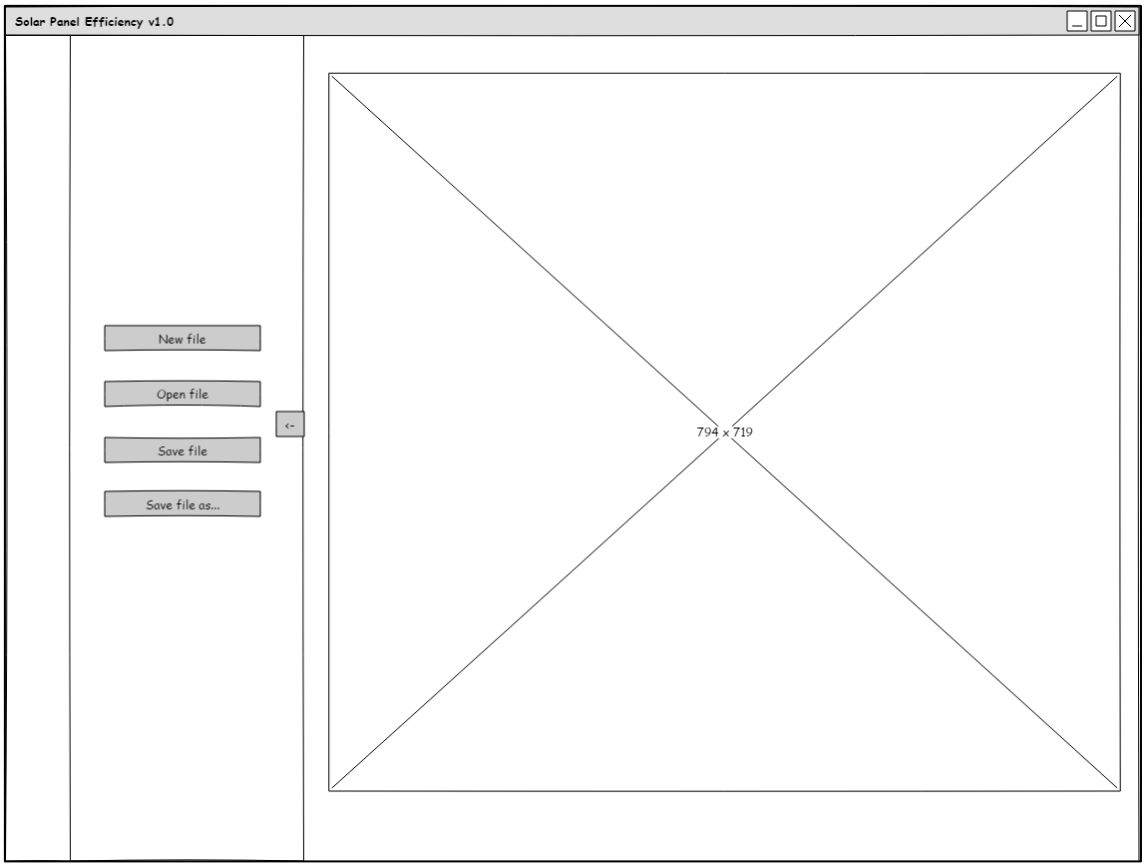

Figura 15. Diseño del menú de gestión de ficheros

El primer botón del menú principal corresponde con la gestión de ficheros. Al pulsarlo, se despliega un menú como el que se visualiza en el esbozo. Aquí solamente hay cuatro botones, los cuales permiten: crear una central fotovoltaica nueva, abrir una, guardar y guardar en un directorio específico. Aparte de estos cuatro botones, también se puede apreciar un pequeño botón a la derecha del menú desplegable. Este botón está en todos los menús y sirve para cerrarlos, acción que también se puede realizar pulsando la tecla “Escape” o abriendo otro menú.

Menú de carga del terreno

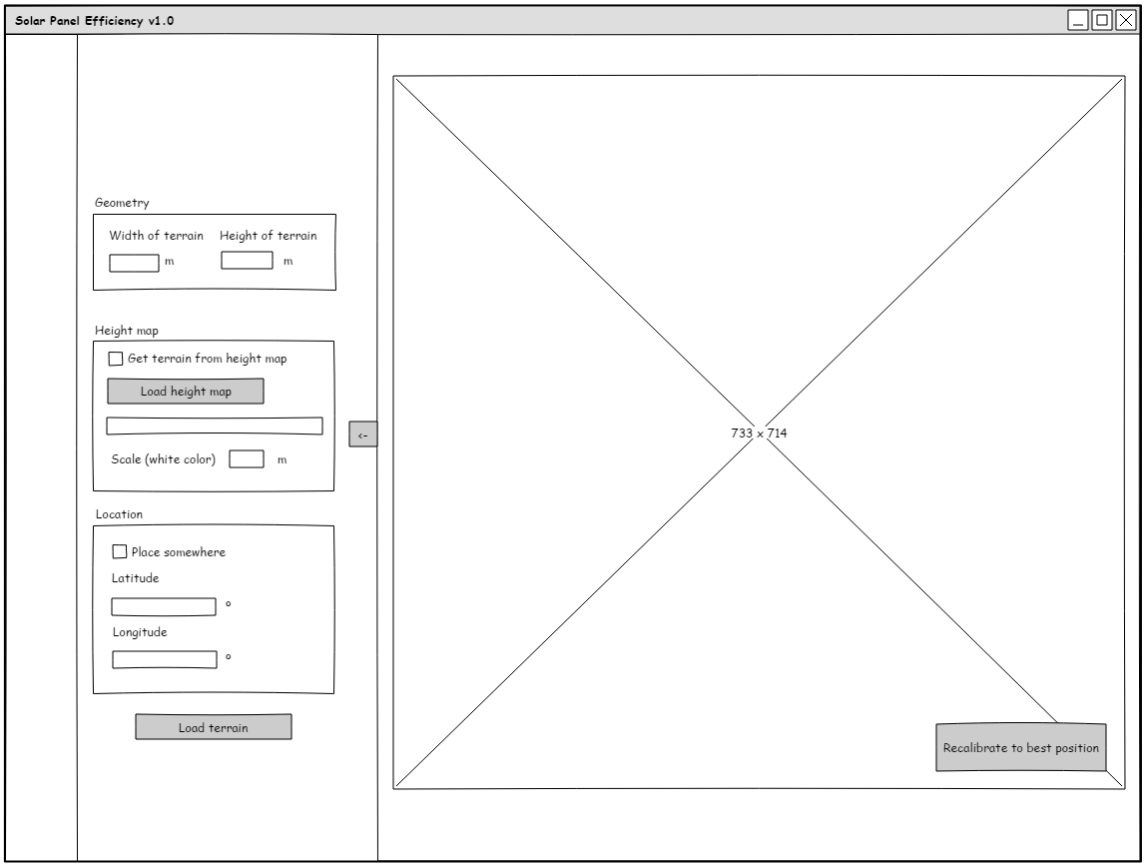

Figura 16. Diseño del menú de carga del terreno

El segundo botón del menú principal corresponde con la carga del terreno. Aquí se pueden especificar todas las características del terreno, que son: ancho y alto del terreno, mapa de alturas, escala del mapa de alturas, latitud y longitud. Salvo las dos primeras características, el resto son opcionales. Si no se utiliza ningún mapa de alturas, el terreno se crea totalmente plano; y si no se indican las coordenadas de latitud y longitud, el terreno se ubica en las coordenadas 0º y 0º, que coinciden con el centro del mapa del mundo.

Tras especificar los parámetros, se puede pulsar el botón de cargar el terreno. Como ya se ha comentado anteriormente, si hay campos que no se han especificado o que tienen valores incorrectos, la aplicación muestra el campo rodeado en rojo y no carga ningún terreno. Una vez están todos los valores correctos, la aplicación carga el terreno en la escena con las características indicadas. La visualización se ajusta automáticamente para que el terreno se pueda ver desde una posición adecuada. Aparte, al cargar el terreno, aparece un botón en la escena que permite volver a dicha posición. En la aplicación real, este botón es un icono.

Menú de carga de paneles

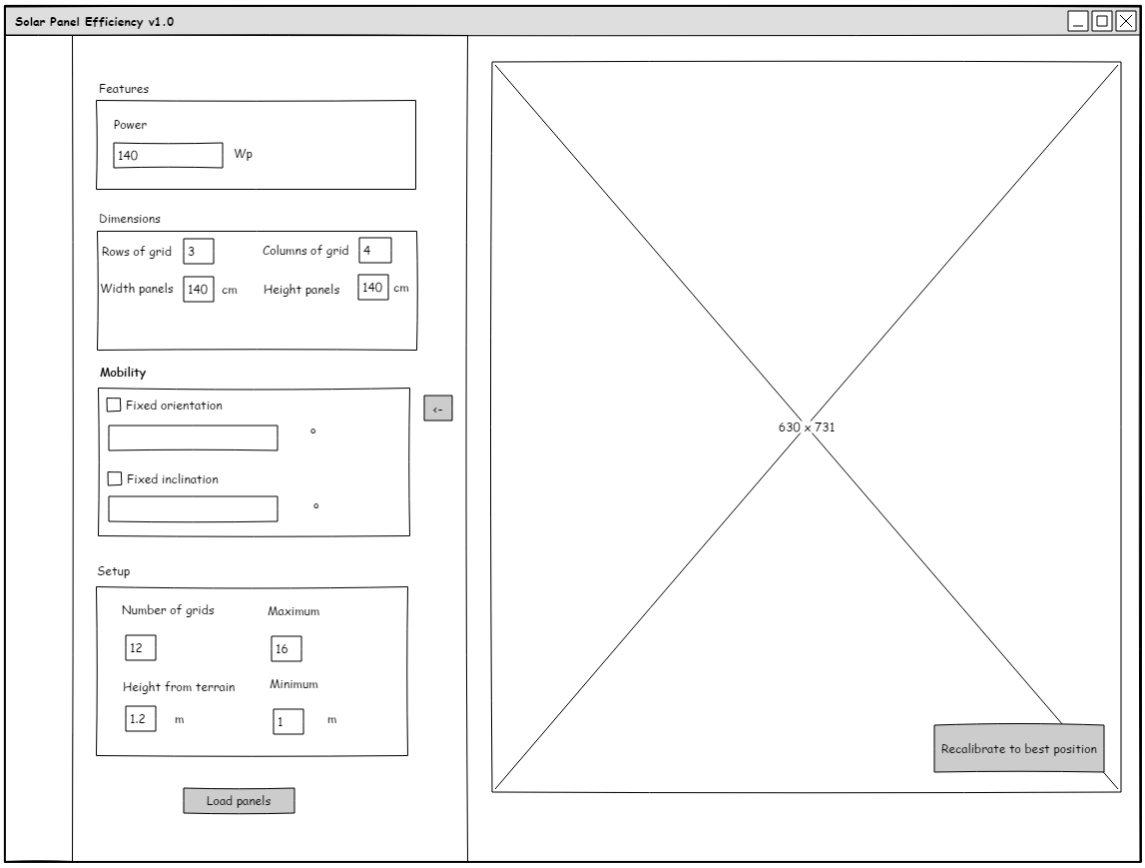

Figura 17. Diseño del menú de carga de paneles

El tercer botón del menú principal corresponde con la carga de los paneles. Aquí se pueden especificar todas las características relacionadas con los paneles, que son: potencia de los paneles, número de filas de paneles en cada matriz, número de columnas de paneles en cada matriz, ancho y alto de los paneles, movilidad de las matrices, número de matrices en el terreno y la elevación de las matrices. Cabe mencionar que el campo para indicar la potencia no estaba planteado en un principio. Pero conforme se fue haciendo la aplicación, se apreció que añadir este campo y ampliar ligeramente los cálculos aportaba mucho valor a la aplicación, ya que se podía calcular la energía generada en un intervalo de tiempo.

Respecto a la movilidad, se pueden indicar tanto la orientación, como la inclinación. Para cada uno de estos grados de libertad, se puede elegir si se tiene movilidad total, o si siempre habrá un valor fijo, el cual se puede especificar. Aparte de los campos que rellena el usuario, el programa calcula, siempre que puede, el número máximo de matrices que caben en el terreno y la altura mínima a la que deben estar las matrices para que no colisionen con el suelo. Tras

indicar todos los campos correctamente, se pueden cargar los paneles y estos se ubican uniformemente repartidos por todo el terreno.

Menú de simulación

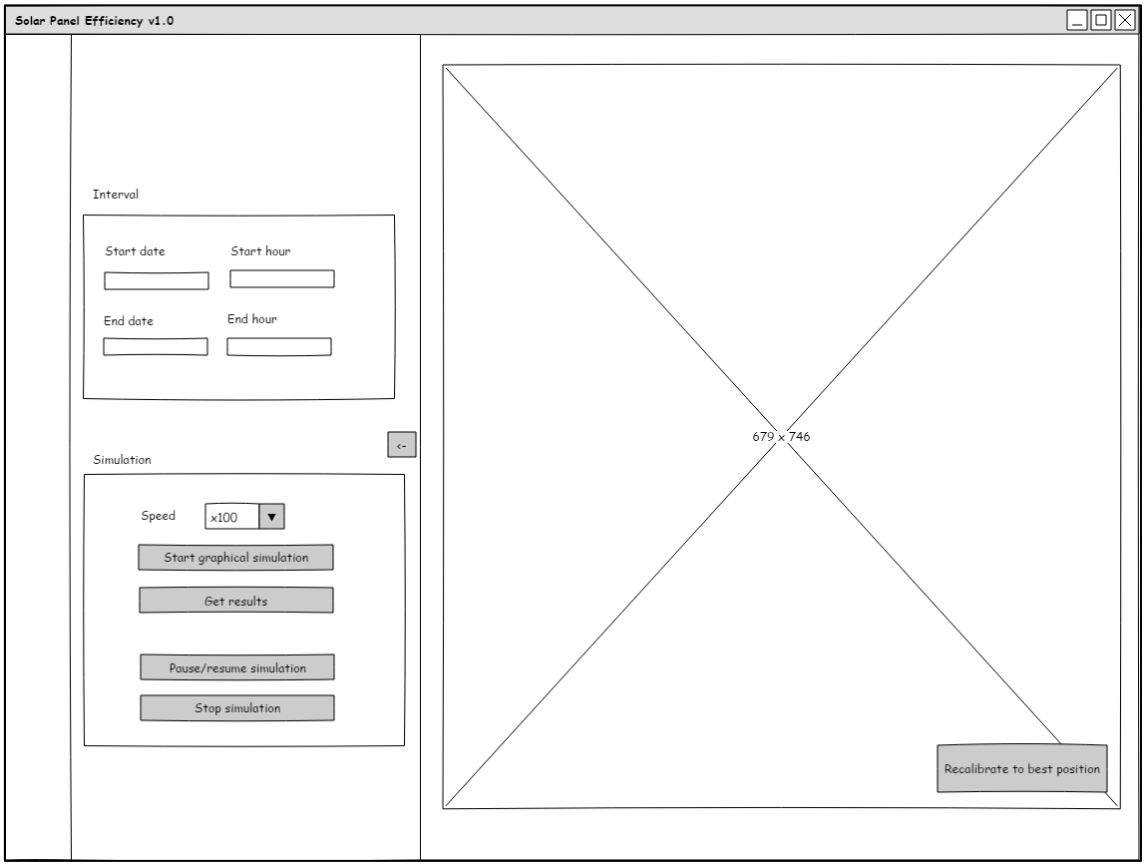

Figura 18. Diseño del menú de simulación

El cuarto botón del menú principal corresponde con la simulación. Aquí se especifica el intervalo de tiempo en el cual se quiere realizar la simulación. Por lo tanto, hay que poner unas fechas y horas de inicio y fin. Hay que tener en cuenta que las fechas solamente incluyen día y mes. Indicar el año no es necesario porque la posición del Sol no varía cada año. Por ejemplo, la ubicación del Sol es la misma a las 12:00 del 1 de enero de 2017 que a las 12:00 del 1 de enero de 2018. Aparte del intervalo, también se puede especificar la velocidad a la que se ejecuta la simulación.

Tras indicar los parámetros, el usuario dispone de una serie de botones. El primero es para comenzar la simulación gráfica. Al pulsarlo, el Sol se ubica en el inicio del intervalo indicado y se mueve a lo largo del tiempo con la velocidad indicada. Los dos últimos botones sirven para controlar la simulación, permitiendo pausarla y reanudarla, o incluso pararla por completo. En la simulación gráfica, por temas de rendimiento, no se calcula ninguna eficiencia. Para obtener los resultados, hay que utilizar el segundo botón. Al pulsarlo, la escena se oculta y aparece una barra de carga indicando el progreso. Aunque no se visualice nada, ocurre lo mismo que en la simulación gráfica y los cálculos se realizan por métodos gráficos. A la hora de obtener los resultados se ignora la velocidad establecida y los cálculos se realizan de la forma más rápida posible.

Aspecto durante la simulación

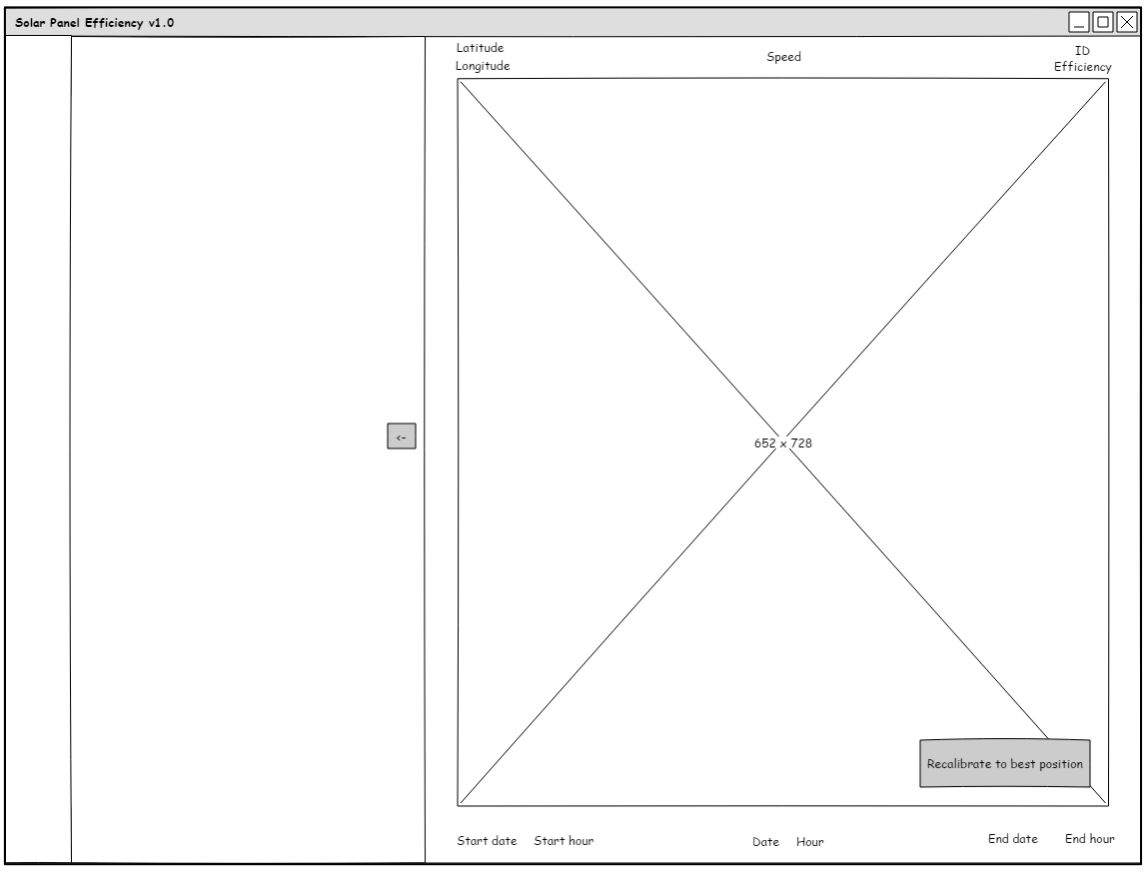

Figura 19. Diseño del aspecto durante la simulación

Durante la simulación gráfica, el usuario puede pulsar sobre los paneles y arriba a la derecha se indica el identificador de ese panel junto a la eficiencia en el momento de pulsar. Aparte de esto, la aplicación también muestra otra información de utilidad durante la simulación. Esta

información incluye: las coordenadas del terreno, la velocidad a la que está siendo ejecutada la simulación, el periodo de simulación y el instante que se está simulando en el momento.

Aspecto durante la obtención de resultados

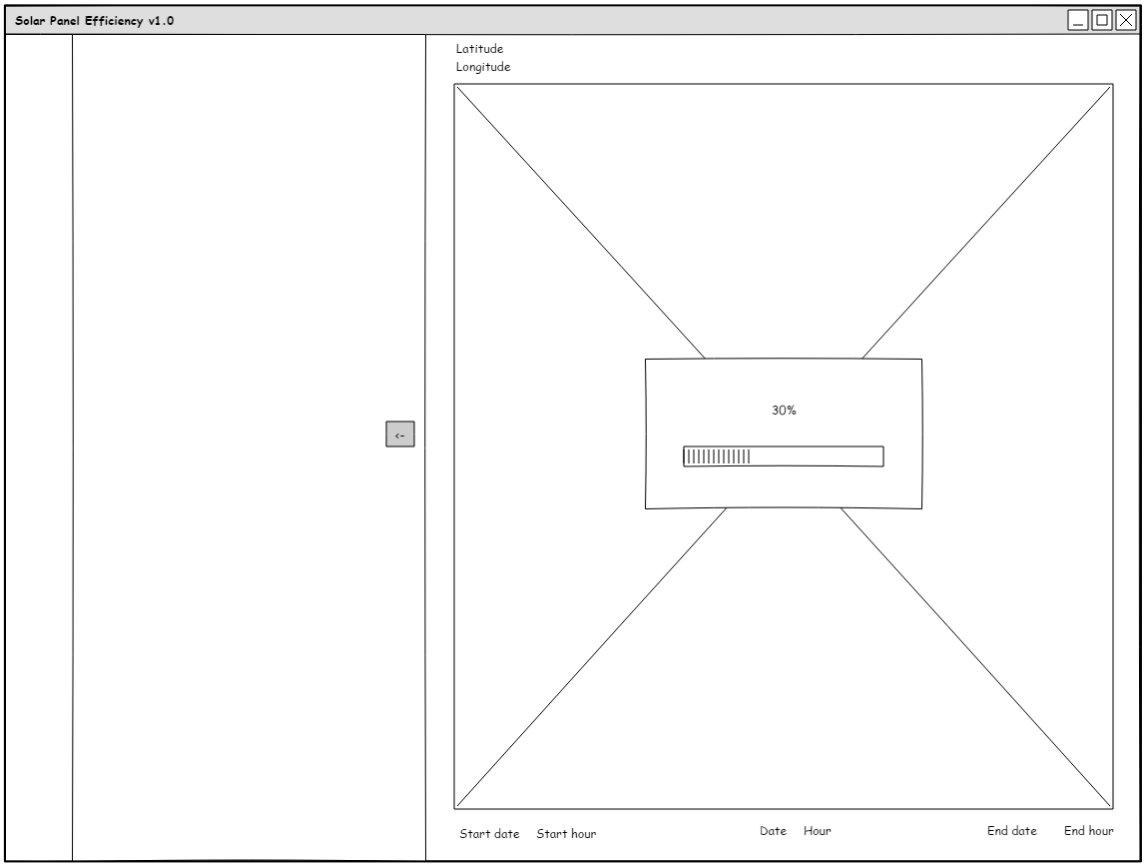

Figura 20. Diseño del aspecto durante la obtención de resultados

Como ya se ha comentado, durante la obtención de resultados la escena desaparece, y en su lugar, se muestra una barra de progreso que indica cuánto queda para acabar la simulación. Aquí, como ya no se puede pulsar sobre ningún panel, no se indica el campo sobre la eficiencia. Por otro lado, dado que la simulación sin gráficos se ejecuta a la velocidad máxima posible, tampoco tiene sentido indicar el valor de la velocidad.

Menú de resultados

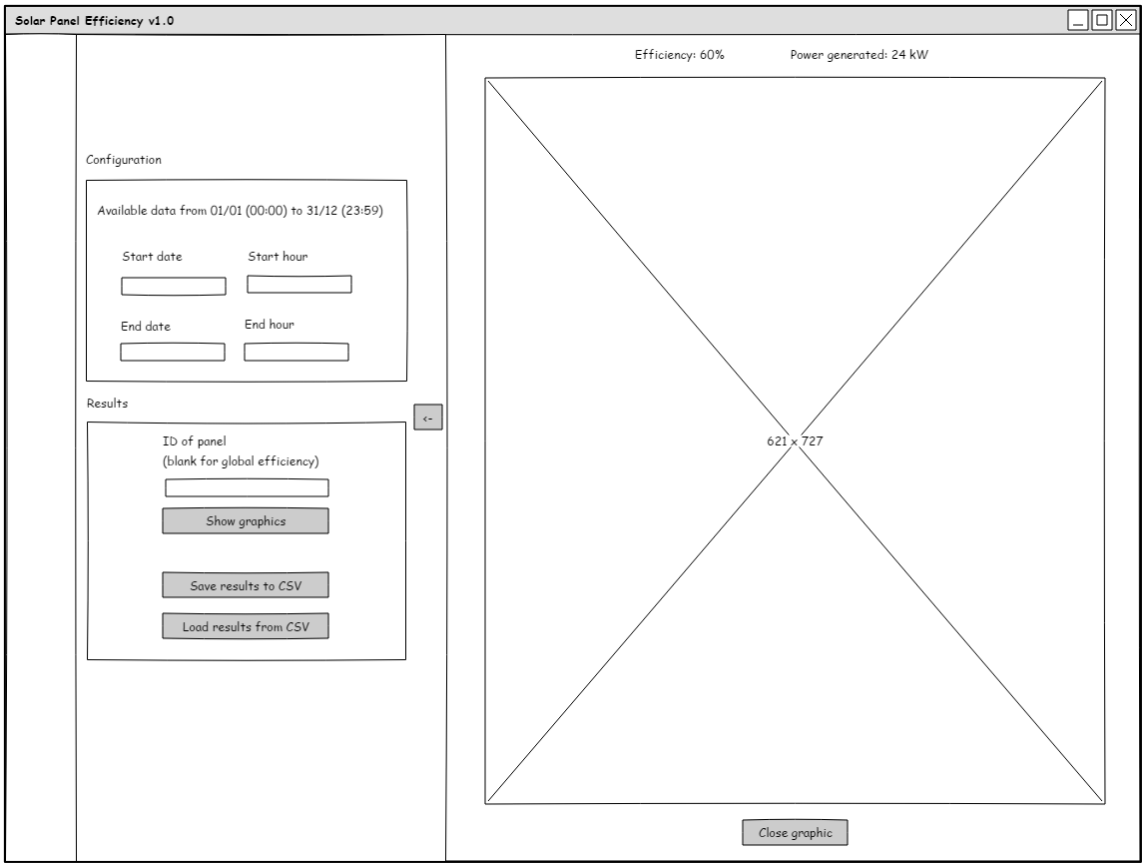

Figura 21. Diseño del menú de resultados

El quinto botón del menú principal corresponde con la visualización de los resultados. Si hay resultados disponibles, se indica el intervalo de tiempo que se puede visualizar. Dentro de dicho intervalo, el usuario puede especificar qué periodo desea ver. Tras ello, se puede indicar un panel en concreto mediante su identificador para ver la eficiencia de ese panel. Este identificador se puede obtener fácilmente pulsando sobre el panel en la escena 3D, tal y como se ha comentado antes. Si en lugar de la eficiencia de un panel, se prefiere visualizar la eficiencia global, simplemente hay que dejar el campo del identificador en blanco.

Tras indicar estos parámetros, se puede dibujar una gráfica que represente los resultados. En esta gráfica, el eje horizontal representa el tiempo transcurrido, y el eje vertical representa la eficiencia obtenida. Al mostrar la gráfica, la escena se oculta; por ello, hay disponible un botón para poder cerrar la gráfica. Aparte de la visualización de la gráfica, la aplicación también permite guardar los resultados en un fichero para después poder cargarlos. En el fichero se

guardan todos los resultados disponibles, ignorando el intervalo establecido para la visualización.

Menú para salir de la aplicación

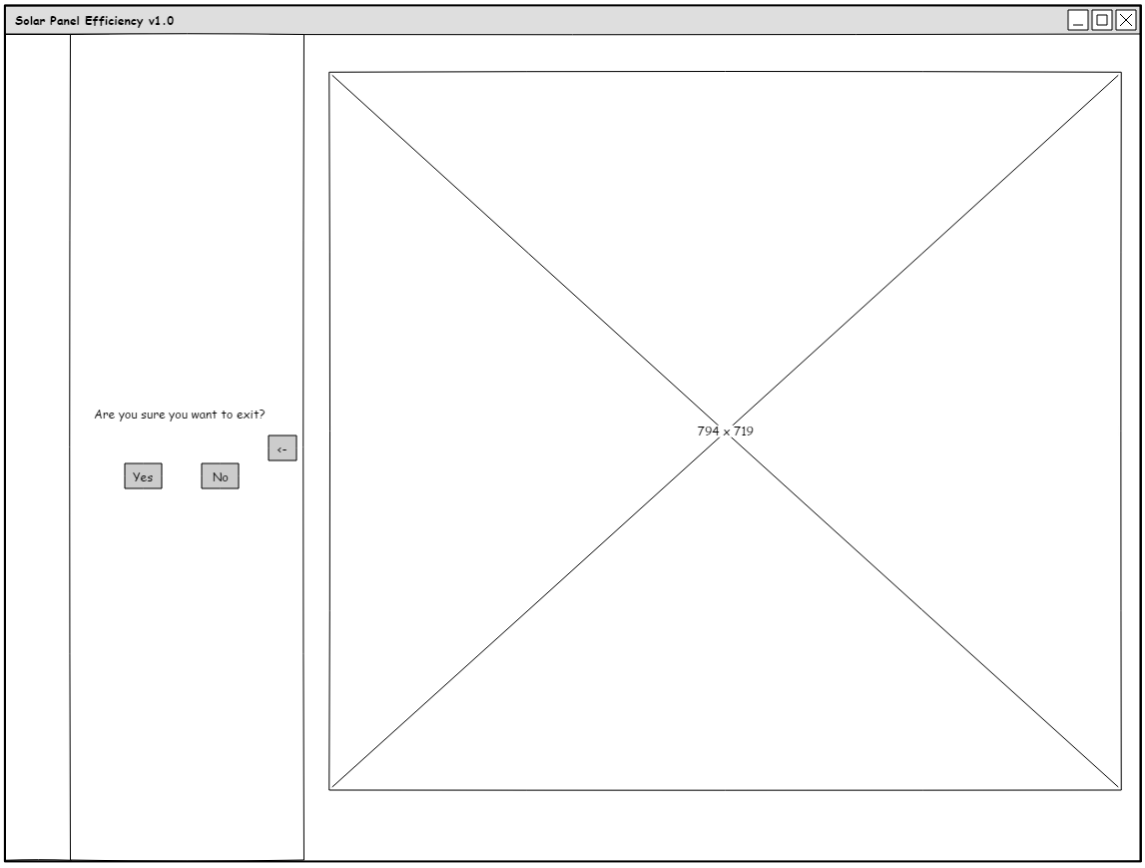

Figura 22. Diseño del menú para salir de la aplicación

El sexto botón del menú principal sirve para salir de la aplicación. Esta acción es una opción típica de las aplicaciones informáticas. Cuando se pulsa el botón, aparece un mensaje de confirmación preguntando si realmente se desea salir de la aplicación.

5. Tecnología utilizada

En este bloque se va a comentar la tecnología utilizada durante el proyecto. Primero se hablará de la tecnología que se ha escogido para el desarrollo de la aplicación y se justificará su elección frente a otras opciones. Tras ello, se enumerarán las herramientas que se han utilizado para otros fines distintos al desarrollo.

5.1. Tecnología de desarrollo

El prototipo inicial de la aplicación estaba realizado en los lenguajes HTML y JavaScript. Pero la gran mayoría de líneas de código eran JavaScript, ya que HTML solamente se utilizaba para indicar la ventana donde se dibujaba toda la escena. Aparte, para poder programar gráficos 3D con JavaScript se había usado la librería Threejs, basada en WebGL. Por ello, una opción muy razonable era realizar la aplicación utilizando la misma tecnología que se había utilizado. Sin embargo, se prefirió valorar otras alternativas.

5.1.1. Alternativas a la tecnología del prototipo

Las ventajas principales de utilizar HTML y WebGL (mediante Threejs) eran que el prototipo ya estaba programado con esos lenguajes, y que dicha aplicación sería multiplataforma al tratarse de un desarrollo web; es decir, podría funcionar en cualquier dispositivo y sistema operativo. Pero ambas ventajas se podían obtener con otra tecnología que fuera multiplataforma y que permitiera utilizar OpenGL. Hay que tener en cuenta que WebGL se basa en OpenGL [24], por lo tanto, emular el comportamiento de una tecnología con la otra no era excesivamente complicado. Así que, tras analizar que era factible utilizar una tecnología distinta, había que valorar qué podía aportar usar una alternativa.

Una ventaja de no utilizar tecnologías web era que la aplicación planteada en este proyecto encajaba más como una aplicación que se instala en un dispositivo, más que utilizar un navegador web para poder utilizarla. Usando tecnologías web, la aplicación se puede enfocar de dos formas: como una serie de archivos que se puedan usar en el ordenador del usuario mediante un navegador, o permitiendo ejecutar la aplicación en un servidor y que dicho servidor transmita al navegador el trabajo que va realizando. Esta segunda opción es muy interesante, dado que el uso de la nube permitiría que el usuario no necesitase un dispositivo

muy potente para poder utilizar la aplicación con un buen rendimiento. Sin embargo, esta opción era demasiado ambiciosa para un trabajo de fin de máster y se salía del alcance inicial.

Junto a la ventaja de utilizar una aplicación instalable en el dispositivo, se puede añadir la preferencia personal por una tecnología concreta. En concreto, para este proyecto, las tecnologías más conocidas por el alumno y con las que tenía más experiencia eran Java y Qt. Java es un lenguaje de programación muy popular que puede funcionar en distintos dispositivos gracias al uso de la máquina virtual Java [25]. Por otro lado, Qt es un entorno de desarrollo que permite desarrollar aplicaciones multiplataforma de dos maneras distintas. En la primera, la más antigua, se utiliza el lenguaje de programación C++. En cambio, en la segunda se utilizan los lenguajes QML y JavaScript. QML sirve para definir el aspecto de la interfaz, mientras que JavaScript se encarga de la funcionalidad.

Tanto Java como Qt eran buenas opciones, y ambas permiten realizar interfaces de forma más sencilla que en las tecnologías web. Pero entre estas dos, Qt tenía una clara ventaja, y es que permitía utilizar JavaScript, y por tanto Threejs. Así pues, con Qt se podía realizar una aplicación multiplataforma y que usase la misma tecnología que el prototipo. Aparte, el alumno tenía experiencia con el entorno y sabía aprovechar el potencial del lenguaje QML. Por lo tanto, al final se decidió utilizar Qt con QML y JavaScript. En la Figura 23 se puede ver cómo es el aspecto del entorno de desarrollo Qt.

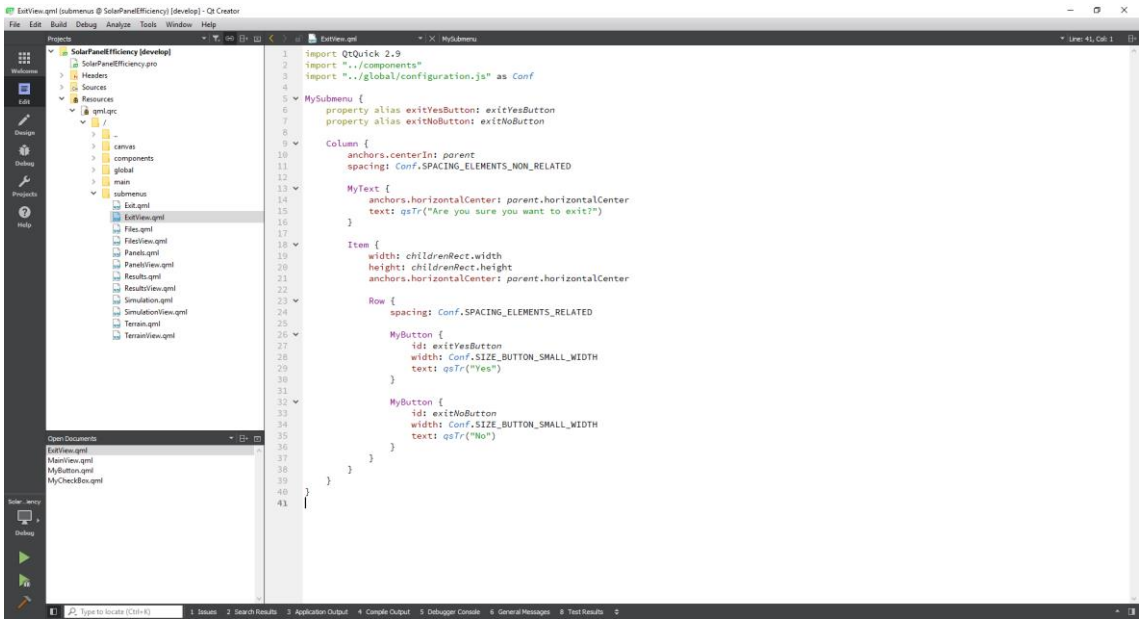

Figura 23. Aspecto del entorno de desarrollo Qt

5.1.2. Qt utilizando QML y JavaScript

Dado que programar con QML y JavaScript no es tan usual como con Java o C++, se va a explicar brevemente cómo son estas tecnologías. QML es un lenguaje declarativo que sirve para describir los elementos que componen una ventana en una interfaz de usuario. Este lenguaje cuenta con la librería Qt Quick donde hay gran cantidad de elementos que se pueden utilizar para crear una ventana. Dentro de un archivo QML, se puede implementar la lógica del programa utilizando el lenguaje de programación JavaScript. A continuación, en la Figura 24 y la Figura 25, se muestra un pequeño ejemplo de una aplicación hecha en QML.

```
import QtQuick 2.5
import QtQuick.Controls 1.4

ApplicationWindow {
    visible: true
    width: 300
    height: 300
    title: qsTr("Hello World")

    Rectangle {
        id: rectangle
        anchors.fill: parent
        color: "blue"

        Button {
            id: button
            anchors.centerIn: parent
            text: "Change!"
            onClicked: rectangle.color = "red"
        }
    }
}
```

Figura 24. Ejemplo de aplicación con QML y JavaScript. Código.

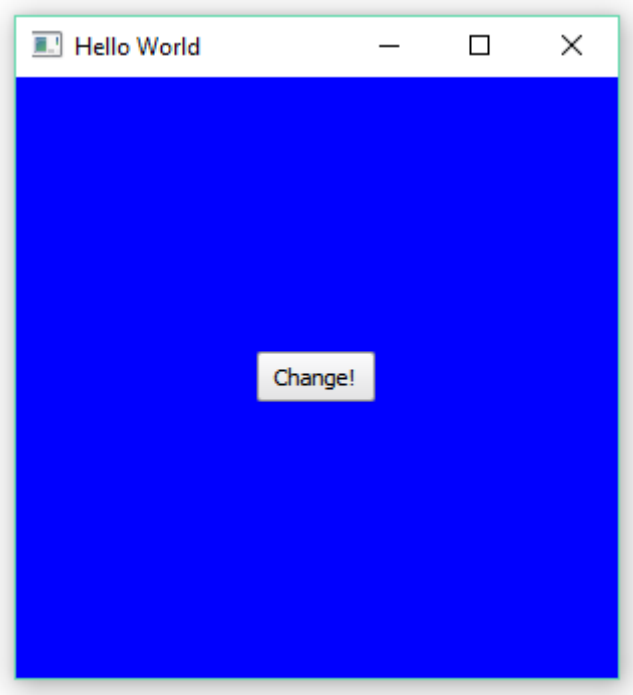

Figura 25. Ejemplo de aplicación con QML y JavaScript. Ventana.

En la primera figura se puede observar cómo se programa una ventana sencilla mediante el lenguaje QML. Primero se define la ventana, luego se indica un rectángulo azul, y finalmente se ubica un botón en el centro del rectángulo. Mediante JavaScript se programa que, al pulsar el botón, el color del rectángulo pase a ser rojo. Aquí se puede observar que el desarrollo de aplicaciones con Qt es claro y sencillo. Sin embargo, juntar el aspecto visual con la funcionalidad es una práctica que no es nada recomendable en ningún lenguaje. A continuación, se explica cómo evitar esto.

5.1.3. Separación de código QML y código JavaScript

Partiendo del ejemplo anterior, ahora se va a separar la parte visual de la parte funcional. En la Figura 26 y la Figura 27 se puede apreciar cómo realizar esto.

```
import QtQuick 2.5
import QtQuick.Controls 1.4

ApplicationWindow {
    visible: true
    width: 300
    height: 300
    title: qsTr("Hello World")

    property alias rectangle: rectangle
    property alias button: button

    Rectangle {
        id: rectangle
        anchors.fill: parent
        color: "blue"

        Button {
            id: button
            anchors.centerIn: parent
            text: "Change!"
        }
    }
}
```

Figura 26. Ejemplo de separación de código. Parte visual

```
import QtQuick 2.5

MyApp {
    button.onClicked: rectangle.color = "red"
}
```

Figura 27. Ejemplo de separación de código. Parte funcional

Ahora se ha creado un archivo llamado “MyApp.qml”. Aquí solamente se indica cómo es la ventana a nivel visual y se utiliza la cláusula “property alias” para registrar el identificador de los subcomponentes utilizados. Con esto, “MyApp” pasa a ser un componente disponible por otros archivos. Así pues, en el archivo principal se agrega el componente “MyApp”. Una vez definido, se accede al botón con identificador “button” para indicar que, al ser pulsado, debe cambiar el color del rectángulo “rectangle”.

Por lo tanto, hay dos archivos QML, uno con la interfaz y otro con la lógica. Qt ofrece la posibilidad de crear automáticamente un archivo “qml.iu” para diseñar el aspecto de la aplicación. Este archivo en un principio solo debe editarse utilizando una interfaz del entorno Qt, donde se pueden coger y arrastrar elementos visualmente. Sin embargo, esta forma de diseñar el aspecto es limitada, así que para esta aplicación se ha preferido describir las ventanas solamente mediante código. Así pues, en lugar de utilizar un fichero con extensión “qml.iu”, se ha usado la extensión “qml”.

5.2. Otras herramientas utilizadas

Aparte de las tecnologías mencionadas para el desarrollo de la aplicación, en este proyecto se han utilizado las siguientes herramientas:

- **Trello.** Esta herramienta ya se ha explicado anteriormente. Se ha utilizado para organizar las tareas del proyecto de forma visual.
- **Modelio.** Se ha utilizado para desarrollar los diagramas UML de este proyecto. Se eligió esta herramienta porque era gratuita y disponía de un mayor grado de personalización que otras aplicaciones similares.
- **Pencil.** Esta herramienta permite realizar esbozos de interfaces. Se ha utilizado para diseñar los *mockups* de la aplicación.
- **Git.** Se ha utilizado para el control de versiones de la aplicación. En concreto, se ha seguido el modelo de ramas que se especifica en [26]. En este modelo se usa principalmente la rama "develop" para desarrollar y la rama "master" para registrar las versiones de la aplicación.
- **Bitbucket.** Esta herramienta permite almacenar el repositorio de Git en la nube, lo cual es muy recomendable para disponer de una copia de seguridad del código. Aparte, Bitbucket ofrece una interfaz para gestionar el repositorio de forma sencilla. Se eligió esta herramienta porque permite tener repositorios privados de forma gratuita, al contrario que otras aplicaciones.
- **SonarQube.** Esta herramienta analiza el código e informa de errores y malas prácticas. Con esto, el desarrollador puede realizar aplicaciones de mayor calidad, aumentando su fiabilidad y mantenibilidad. En el apartado de pruebas se comentará más sobre esta utilidad.
- **Cloc.** Esta herramienta permite contar las líneas de código de una aplicación. Posteriormente se indicarán estadísticas extraídas haciendo uso de Cloc.

6. Desarrollo de la solución propuesta

En este bloque se va a comentar cómo ha sido el desarrollo de la aplicación, explicando los aspectos más relevantes y mostrando el resultado final. Para ello, se ha creado una sección por cada iteración que se ha hecho. Estas iteraciones se han comentado en los apartados 3.4 y 3.5, donde se hablaba sobre la planificación y el presupuesto del proyecto. Se recuerda que cada una de estas iteraciones ha correspondido con un lanzamiento de versión en la herramienta Git. Cabe añadir que en este apartado solamente se hablará sobre temas relacionados con la implementación. El funcionamiento básico de la aplicación ya se ha especificado en los diseños del apartado 4.

6.1. Prototipo en Qt

Para comprobar que era factible utilizar Qt para la aplicación de este trabajo, se rehízo el prototipo inicial usando los lenguajes de Qt. Como se ha comentado en el apartado 5, Qt se puede utilizar con JavaScript. Por lo tanto, debería poder usarse Threejs, una librería basada en WebGL y escrita en JavaScript para poder trabajar con gráficos. Pero, la librería como tal no sirve para Qt, pues está pensada para desarrollo web y el uso de HTML. Sin embargo, Qt dispone de una adaptación de Threejs para que funcione en las aplicaciones diseñadas por este entorno. Internamente, contiene algunas diferencias, pero cara al desarrollador, se trabaja exactamente igual.

Inevitablemente, aunque Threejs funcionara igual, el prototipo no se pudo traspasar de tecnología sin realizar nada. Se tuvo que hacer una serie de adaptaciones, la mayoría relacionadas con que en lugar de trabajar sobre HTML se trabajaba sobre una ventana de Qt. Aparte, había un algoritmo del Threejs de Qt que no coincidía con el especificado en el Threejs normal. Pero una vez se detectó, se copió el algoritmo y funcionó sin problemas. Cuando se solventaron todas estas incompatibilidades, se consiguió un prototipo prácticamente igual. En la Figura 28 y la Figura 29 se puede comparar el prototipo inicial con el prototipo realizado en Qt.

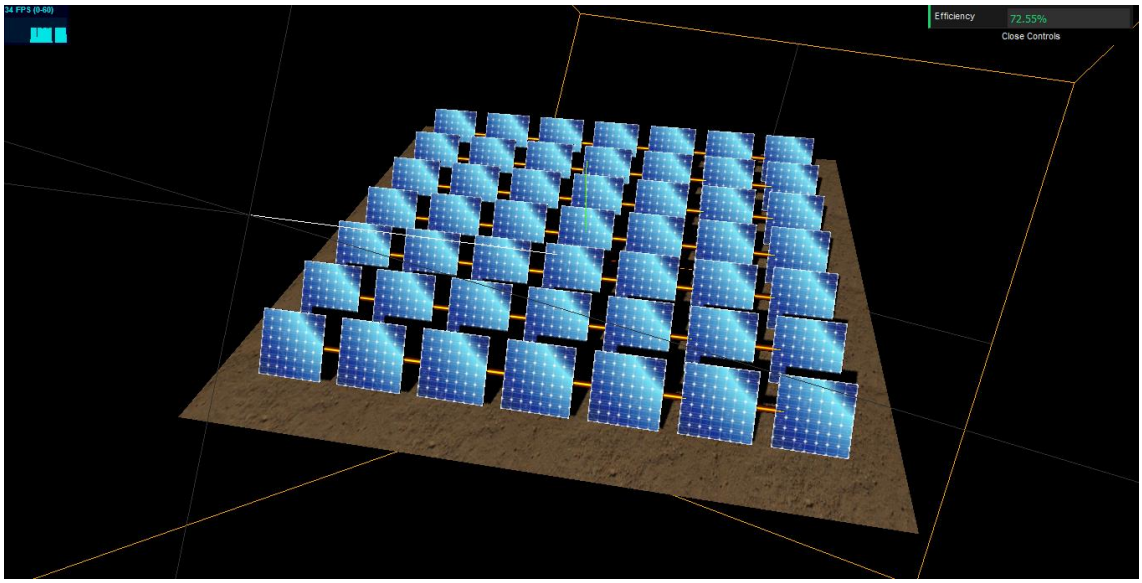

Figura 28. Comparación entre prototipos. Prototipo inicial

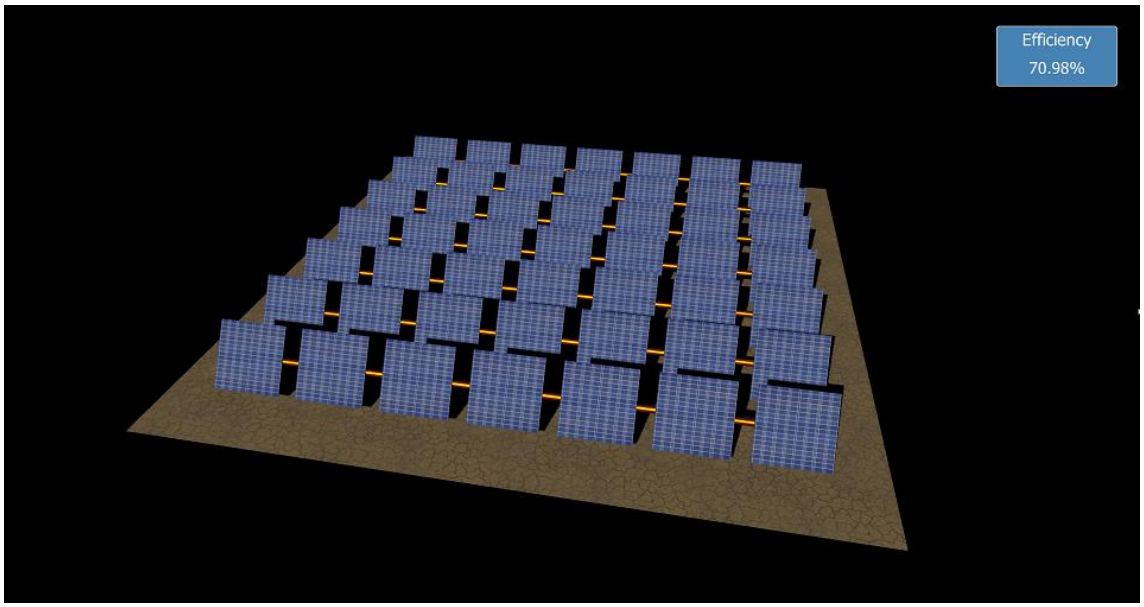

Figura 29. Comparación entre prototipos. Prototipo con Qt

Los dos prototipos contaban con exactamente la misma funcionalidad. A nivel estético se pueden apreciar diferencias, pero todas ellas tienen explicación. Por una parte, en el prototipo de Qt no se añadieron líneas auxiliares ni la gráfica que muestra los FPS (fotogramas por segundo), debido a que no se utilizarían en la aplicación final. Por otro lado, para mostrar la eficiencia, en el prototipo inicial se usaba una librería adicional para visualizar el campo de texto. En cambio, en el prototipo hecho en Qt se utilizó el lenguaje QML, añadiendo un

rectángulo que estuviera por encima del lienzo de dibujo. Finalmente, las texturas del terreno y de los paneles son distintas entre los prototipos.

6.2. Interfaz básica

Tras comprobar que la aplicación era completamente factible usando Qt, se hizo una estructura básica de cómo sería la interfaz de la aplicación. En concreto, se implementó: el menú lateral comentado en el apartado de diseño, el lienzo para los gráficos, y unas barras horizontales para poder mostrar información de la central y la simulación. Aparte, al pulsar los botones del menú principal, se implementó que apareciese un submenú vacío mediante una animación. Tal y como se ha indicado en el diseño, dicho submenú se puede cerrar con el botón de la flecha o pulsando la tecla “Escape”.

Aparte de las animaciones para abrir y cerrar los submenús, se implementaron otros detalles que le dan un aspecto moderno y fluido a la aplicación. Por ejemplo, los botones cambian de color si son pulsados o si el ratón pasa por encima. Otra característica destacable es que, en caso de que la pantalla sea muy pequeña, el menú y los submenús se puedan desplazar verticalmente. Esto se hizo pensando en que la aplicación podría llegar a ejecutarse en dispositivos pequeños. Una vez comentado esto, en la Figura 30 se puede ver la aplicación tras realizar la interfaz básica. Como se ha comentado, los submenús se dejaron vacíos, salvo el correspondiente para salir de la aplicación, el cual se puede apreciar en la Figura 31.

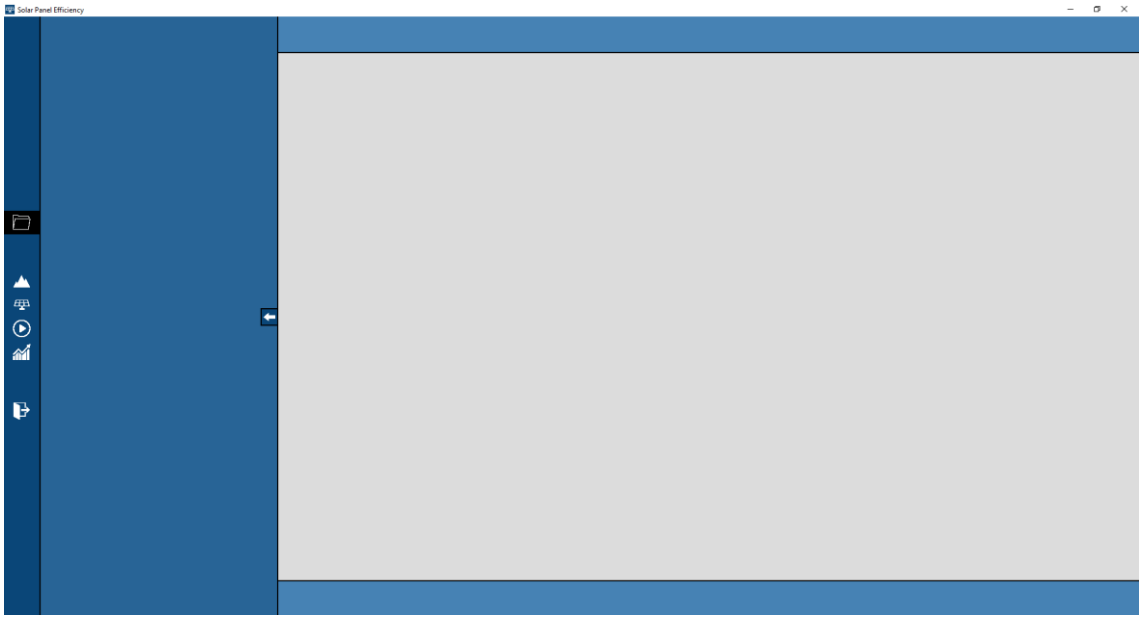

Figura 30. Interfaz básica

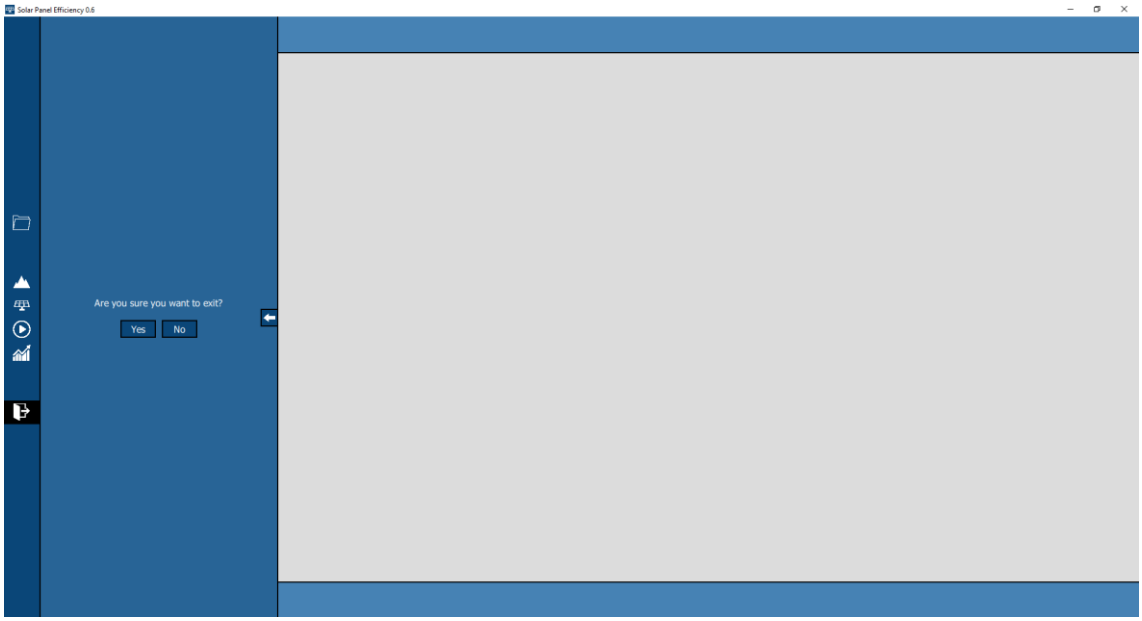

Figura 31. Interfaz para salir de la aplicación

6.3. Carga del terreno

6.3.1. Campos de entrada

Una vez se dispuso de la interfaz básica, se comenzó a implementar la funcionalidad relacionada con la carga del terreno. Lo primero fue crear los campos del submenú, los cuales se definieron en los diseños de la aplicación. Esto implicó que se tuvieron que diseñar los elementos típicos de un formulario, como por ejemplo los campos de texto o los menús desplegables. Qt cuenta con este tipo de elementos, pero tienen un aspecto por defecto que no encajaba con el diseño general de la aplicación. Así pues, se tuvieron que redefinir estos elementos para poder personalizarlos.

Aparte del aspecto visual de los componentes, también se tuvo que programar el control de la entrada de usuario. Como se ha comentado en el apartado 4, los campos tenían que contar con distintos mecanismos para que la entrada del usuario fuera correcta. En primer lugar, los campos cuentan con un texto que indica qué valores se espera en dicho campo, incluyendo el valor mínimo y máximo en caso de que sea un campo numérico. Hay que tener en cuenta que este texto indicativo solo aparece si el campo está vacío. En segundo lugar, cada vez que se escribe en un campo, se comprueba si el texto es un valor correcto. Esto implica que, si un campo es numérico, no se pueden escribir letras. En último lugar, a la hora de requerir los valores, el programa hace una verificación de que el dato es correcto. En caso de que no lo sea, el campo se marca en rojo. En la Figura 32 se pueden ver los campos creados para el submenú de carga del terreno, así como los mecanismos que se han comentado. En concreto, el campo que está marcado en rojo es incorrecto porque se ha superado el máximo indicado, que es 50.

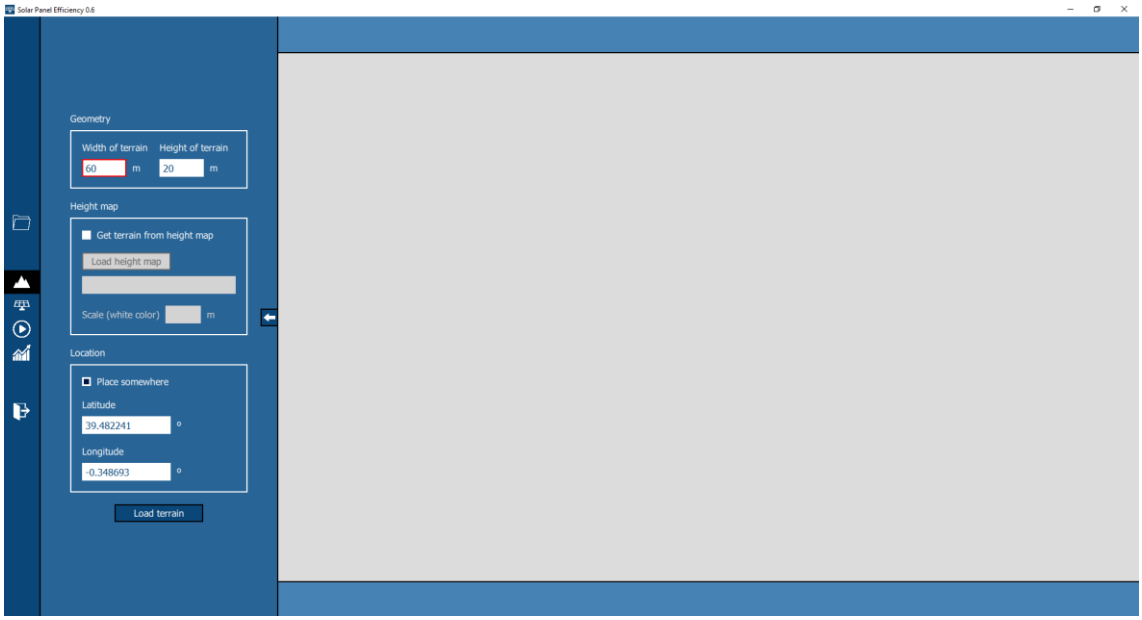

Figura 32. Campos del submenú de carga del terreno

6.3.2. Carga de un terreno plano

Tras implementar esto, se comenzó con la carga del terreno. Primero se programó el diseño de un terreno plano, sin tener en cuenta ningún mapa de alturas. En la Figura 33 se puede apreciar el aspecto de la aplicación tras cargar un terreno con las dimensiones indicadas en el submenú. Las coordenadas del terreno se pueden visualizar en la barra superior de la pantalla, pero estas no se utilizarán realmente hasta que no se haga la simulación. En la figura se ha mostrado el terreno tras haberlo cargado. La cámara que visualiza la escena se ajusta en función de las dimensiones del terreno para que este se pueda visualizar correctamente. Tras ello, el usuario puede mover la cámara para ver el terreno desde el ángulo que desee. En la Figura 34Figura 36 se puede ver el terreno creado desde otra perspectiva. Si luego se desea volver a la posición de la cámara por defecto, se puede pulsar el botón que se muestra en la parte inferior derecha de la pantalla.

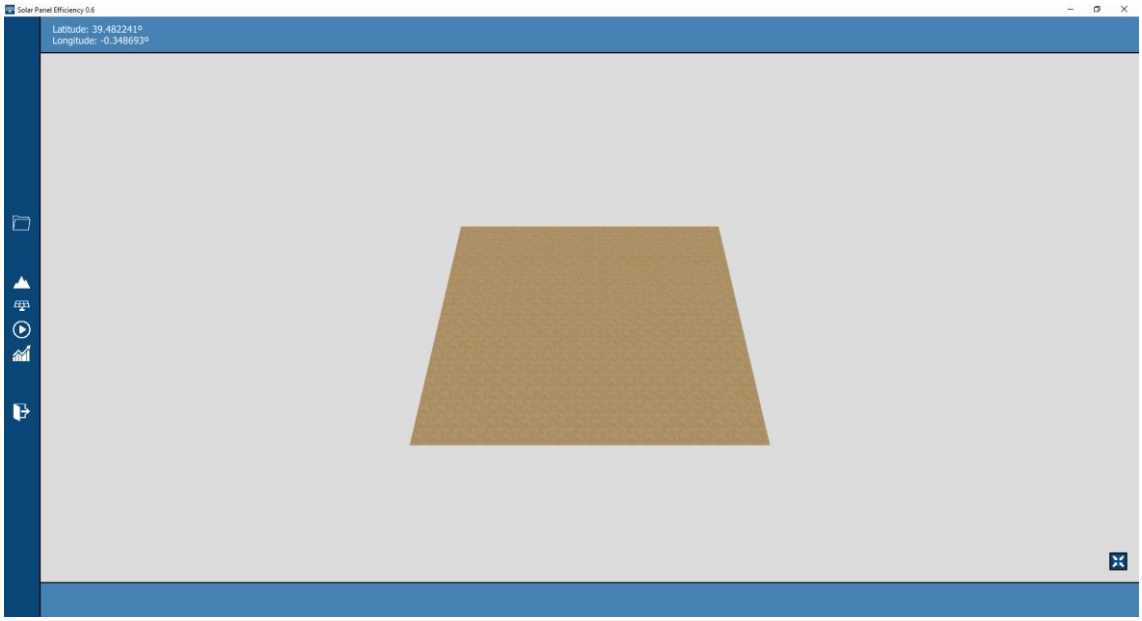

Figura 33. Terreno visto desde la perspectiva por defecto

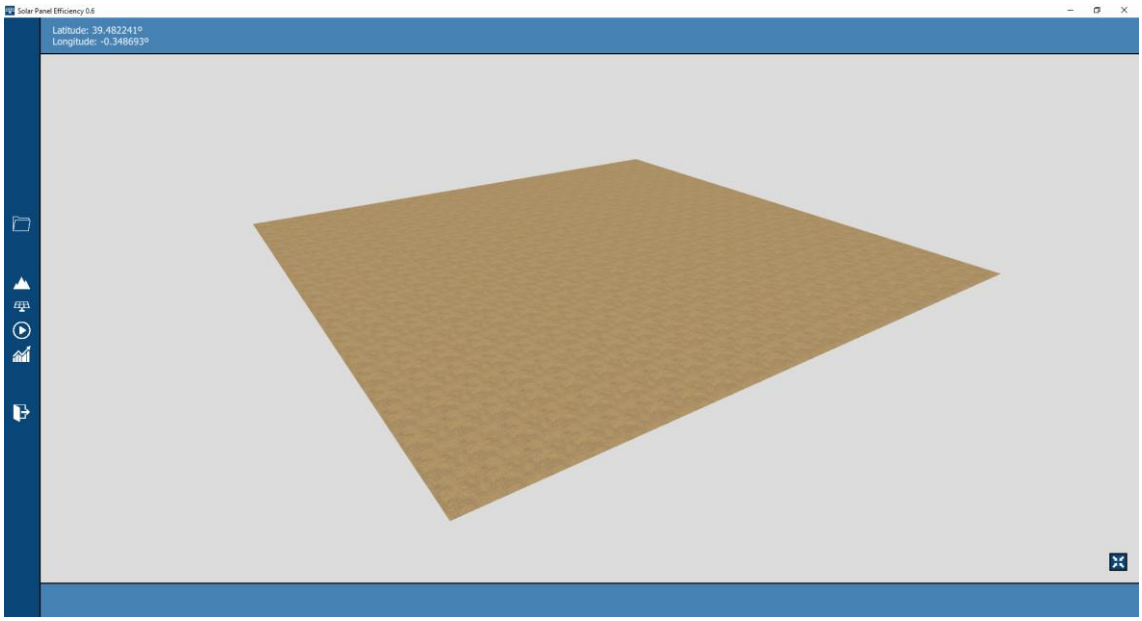

Figura 34. Terreno visto desde otra perspectiva

6.3.3. Carga de un terreno a partir de un mapa de alturas

Tras comprobar el correcto funcionamiento de la carga de un terreno plano, quedaba poder crearlo a partir de un mapa de alturas. Esta fue una de las características más complicadas del desarrollo de la aplicación; aparte de que luego, a la hora de cargar los paneles, también

dificultó el hecho de que el terreno no fuera plano. En Threejs, el proceso de modificar un plano en función de una imagen es como se explica a continuación. Primero, se extraen los valores de los píxeles de la imagen, los cuales pueden ser de 0 a 255. Para ello, se cogen los tres valores que definen un color (rojo, verde y azul) y se hace la media. Finalmente, el terreno se divide en distintos vértices y se cambia la altura de estos vértices en función de los valores extraídos. Este proceso es relativamente sencillo si se indican tantos vértices en el plano como píxeles tiene la imagen. Sin embargo, las imágenes suelen tener demasiados píxeles como para establecer esta relación sin afectar al rendimiento de la aplicación. Por ello, había que plasmar la información de todos los píxeles en un número inferior de vértices.

Para realizar esto, se tuvo que implementar un algoritmo que calculase la media de los píxeles que afectaban a un determinado vértice. Es decir, si un vértice corresponde con 6 píxeles de la imagen, por cada pixel se calcula qué porcentaje pertenece a ese vértice y se hace una media ponderada de todos los vértices. En otras palabras, se tiene más en cuenta el color de los píxeles que están completamente dentro del vértice que los que no. El desarrollo de este algoritmo no fue trivial, pero finalmente se consiguió con éxito. Por lo tanto, la aplicación puede cargar cualquier imagen como mapa de alturas. Cabe mencionar que también se tiene en cuenta el ancho y largo de la imagen y del terreno.

En la Figura 35 se puede ver un mapa de alturas y el terreno cargado a partir de ese mismo mapa. Hay que tener en cuenta que en el submenú se debe indicar la escala en la que está el mapa de alturas. En concreto, se debe indicar cuánta altura representa el valor 255, que corresponde con el color blanco. Para el ejemplo, se ha utilizado un mapa de alturas poco realista para ver claramente cómo se corresponde la imagen con el terreno. En la Figura 36 se puede apreciar un terreno más coherente.

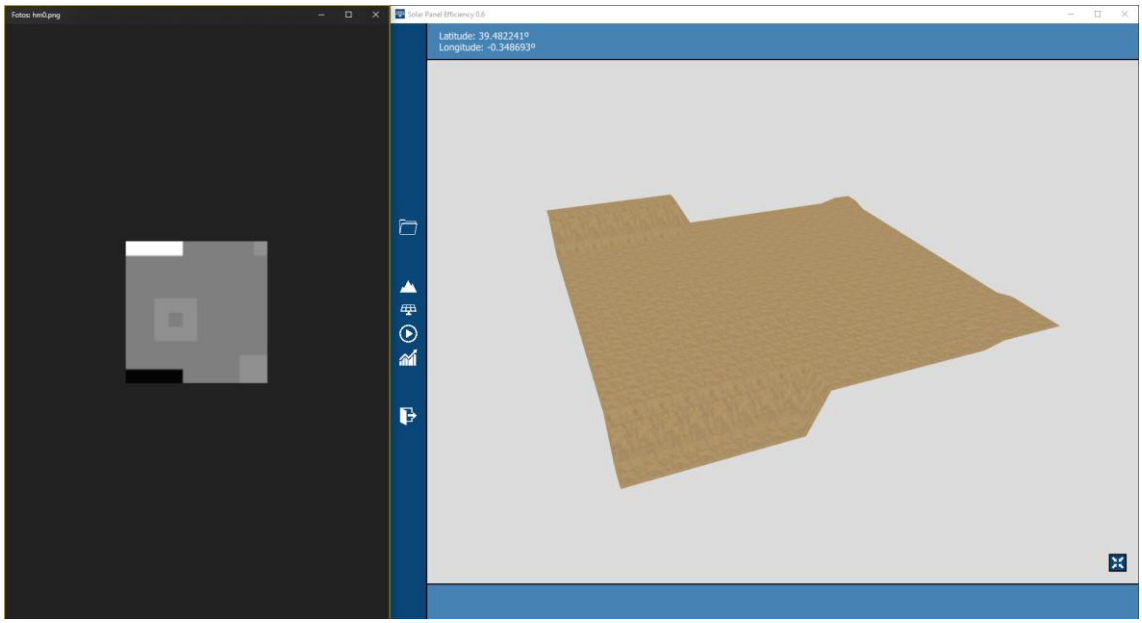

Figura 35. Terreno a partir de un mapa de alturas. Ejemplo básico

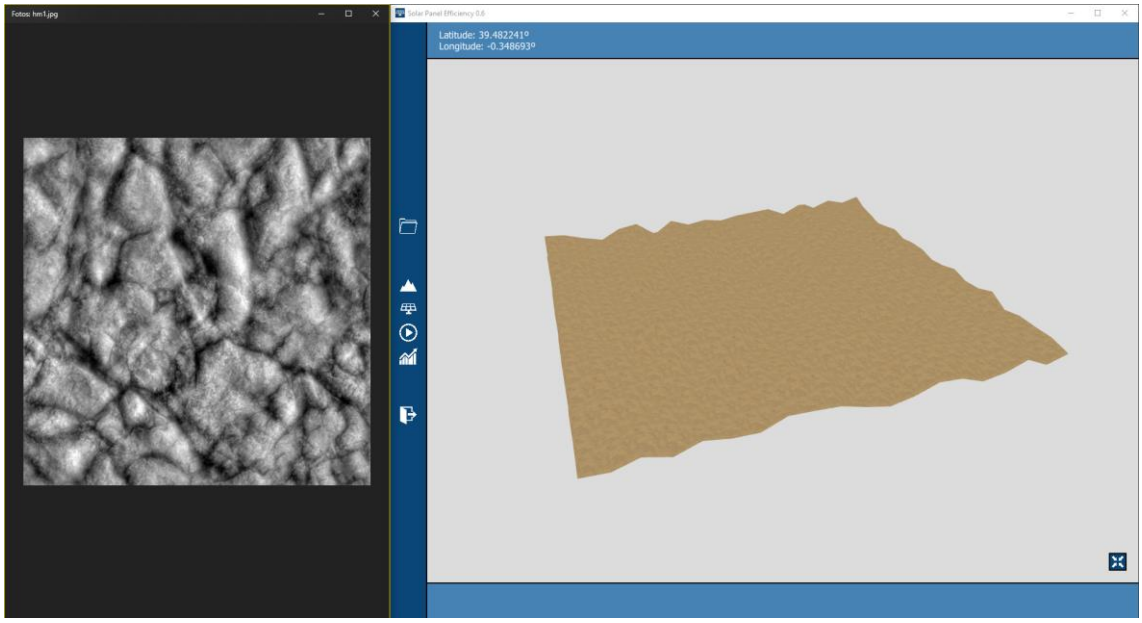

Figura 36. Terreno a partir de un mapa de alturas. Ejemplo realista

6.3.4. Gestión de ficheros

Como en esta fase de la aplicación ya se introducían datos para el diseño de la central fotovoltaica, surgió la necesidad de implementar la gestión de ficheros para poder guardar el estado en el que se encontraba la aplicación. Esto permitiría ir almacenando distintos diseños

de centrales fotovoltaicas para poder cargarlos más adelante. Así pues, se diseñó el submenú para la gestión de archivos y se implementó el guardado y cargado de la aplicación. En la Figura 37 se puede ver dicho submenú.

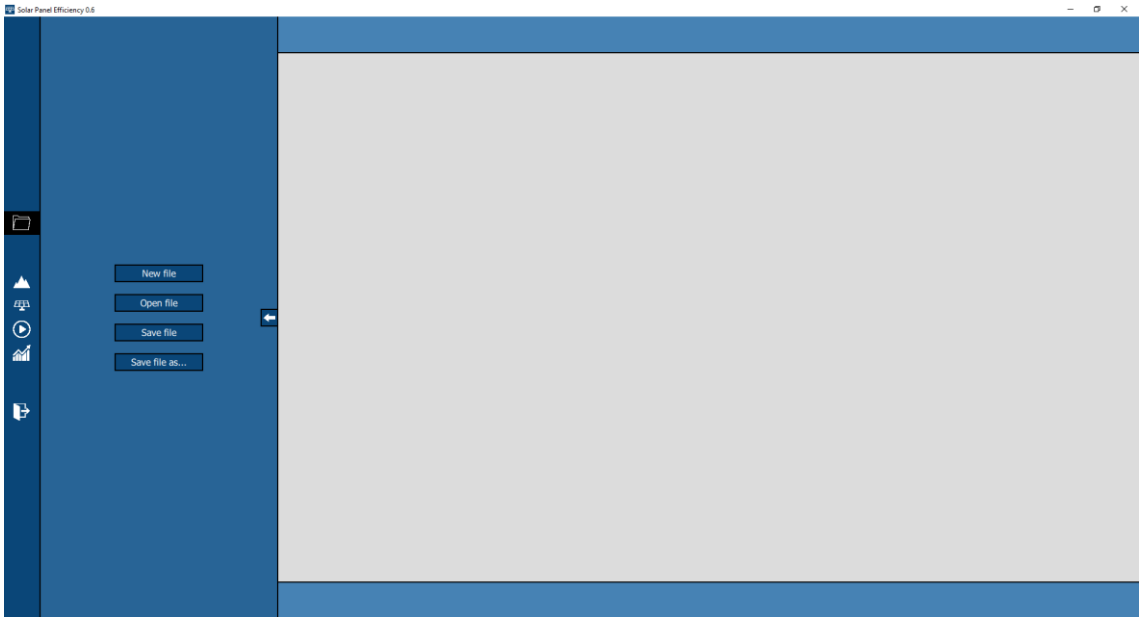

Figura 37. Submenú para la gestión de ficheros

6.4. Carga de los paneles

Tras la carga del terreno, el siguiente paso fue implementar la carga de los paneles. Aquí, el diseño del submenú fue más rápido debido a que la mayoría de los componentes ya se programaron en el submenú de la carga del terreno. La mayor dificultad de esta parte fue el posicionar los paneles sobre del terreno de forma correcta. Por un lado, había que ubicar los paneles de forma uniforme sobre el terreno. Esto es trivial cuando el terreno es cuadrado y el número de paneles puede repartirse en el mismo número de filas y columnas. Sin embargo, si no se cumplen estas condiciones, el algoritmo se complica ligeramente. Pero realizando unas cuantas operaciones, se consiguió resolver con éxito. En la Figura 38 se puede ver cómo se reparten 9 paneles sobre un terreno y luego en la Figura 39 se muestra la reubicación con 8 paneles. De paso, se puede apreciar cómo es el submenú para los paneles.

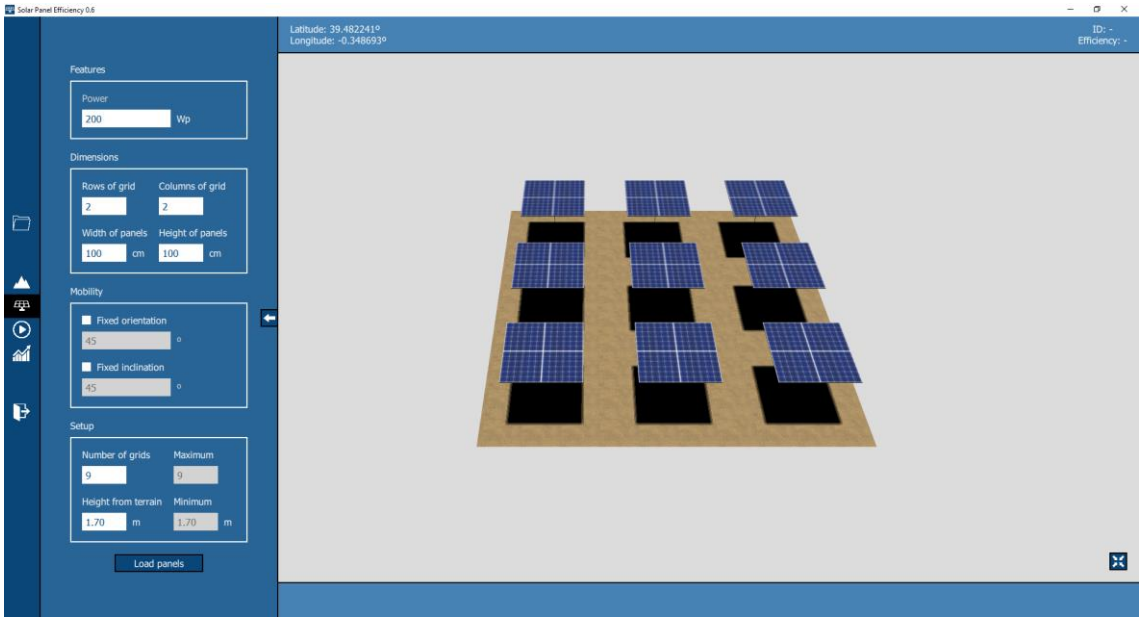

Figura 38. Ubicación de 9 paneles sobre el terreno

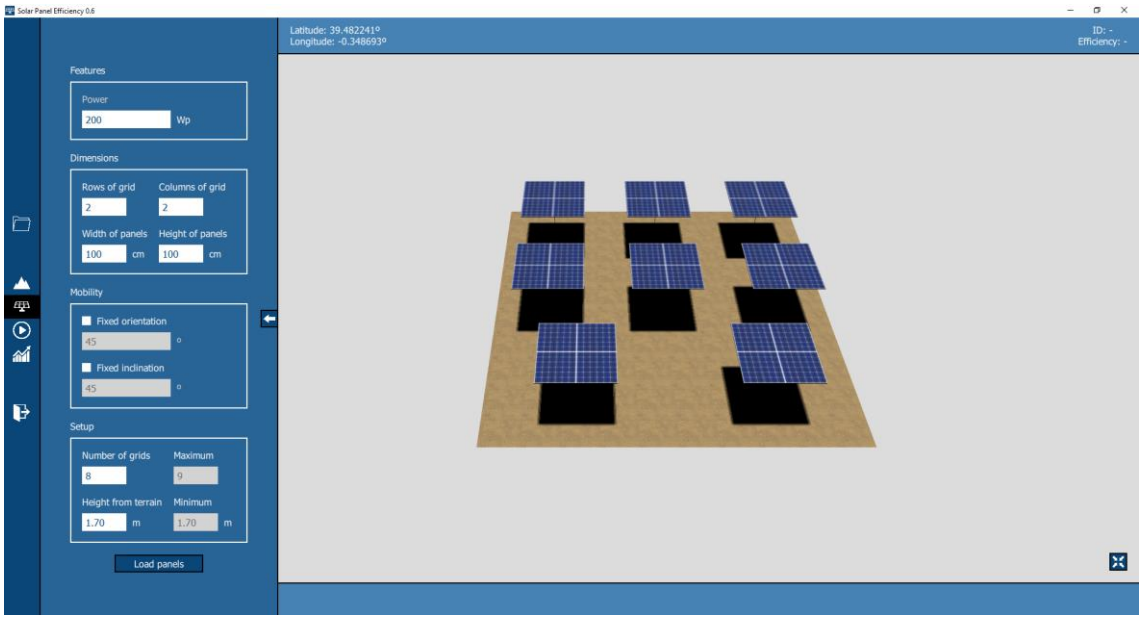

Figura 39. Ubicación de 8 paneles sobre el terreno

Aparte de la ubicación a lo largo y ancho del terreno, también hay que tener en cuenta si el terreno se ha creado a partir de un mapa de alturas. Hay que ubicar cada panel a la altura que le corresponde según el punto del terreno sobre el que se alza. En programación con gráficos, existe el concepto de que un elemento de la escena sea “hijo” de otro elemento. Hacer uso de esto permite que los “hijos” se vean afectados por las modificaciones en la traslación, rotación

y escala del “padre”. Sin embargo, aunque los paneles se indiquen como “hijos” del terreno, el cambiar los vértices del terreno no afectará a donde se posicionan los paneles. Por lo tanto, para concretar exactamente a qué altura tenía que estar un panel en un punto cualquiera del terreno, se tuvo que hacer uso de trazado de rayos. Threejs permite definir rayos para calcular intersecciones con otros objetos. Así pues, con cada panel, se calculaba en qué punto del terreno iba a estar, y tras ello se trazaba un rayo vertical en esas coordenadas para ver en qué punto de la escena se colisionaba con el terreno. De este punto, se obtenía su altura, y esta es la que se asignaba al panel. En la Figura 40 se puede ver cómo los paneles adoptan distintas alturas en un terreno irregular.

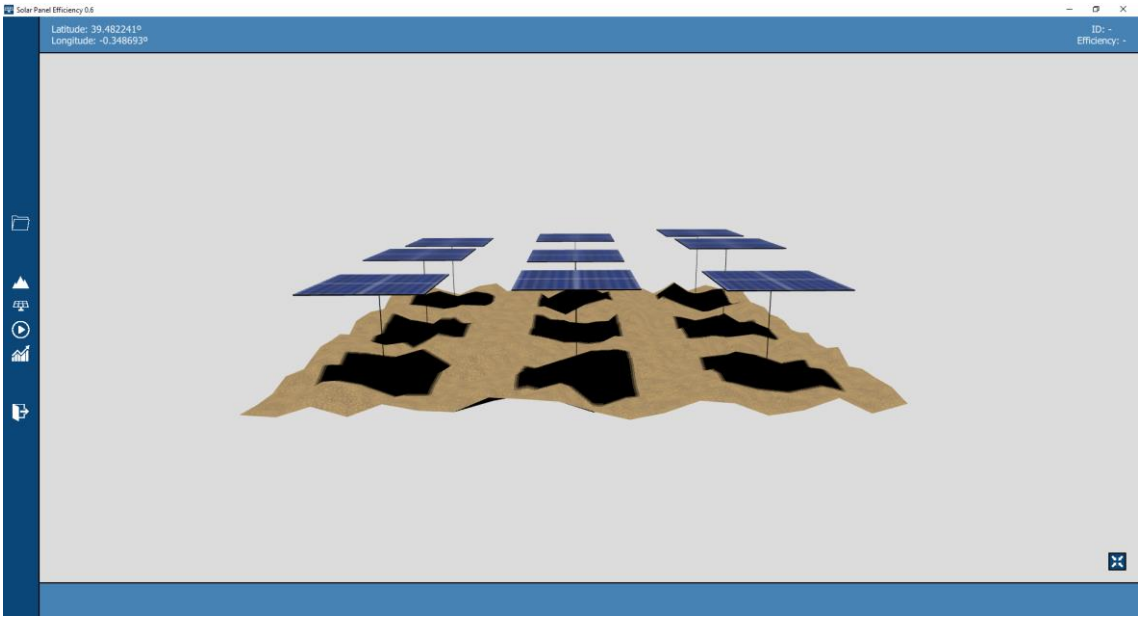

Figura 40. Paneles sobre un terreno irregular

6.5. Simulación y resultados

Tras la creación del terreno y los paneles, el siguiente paso fue implementar la simulación y la visualización de resultados. Para ello se implementaron los submenús indicados en el apartado 4.

6.5.1. Simulación gráfica

En el primer submenú se permite establecer un intervalo de tiempo en el cual se debe simular la central fotovoltaica teniendo en cuenta el movimiento del Sol y de los paneles. La simulación se puede realizar gráficamente para poder ver cómo se comportan los paneles, o se puede hacer sin gráficos para obtener resultados de la forma más rápida posible y visualizarlos posteriormente. En la Figura 41 se puede ver una simulación en proceso.

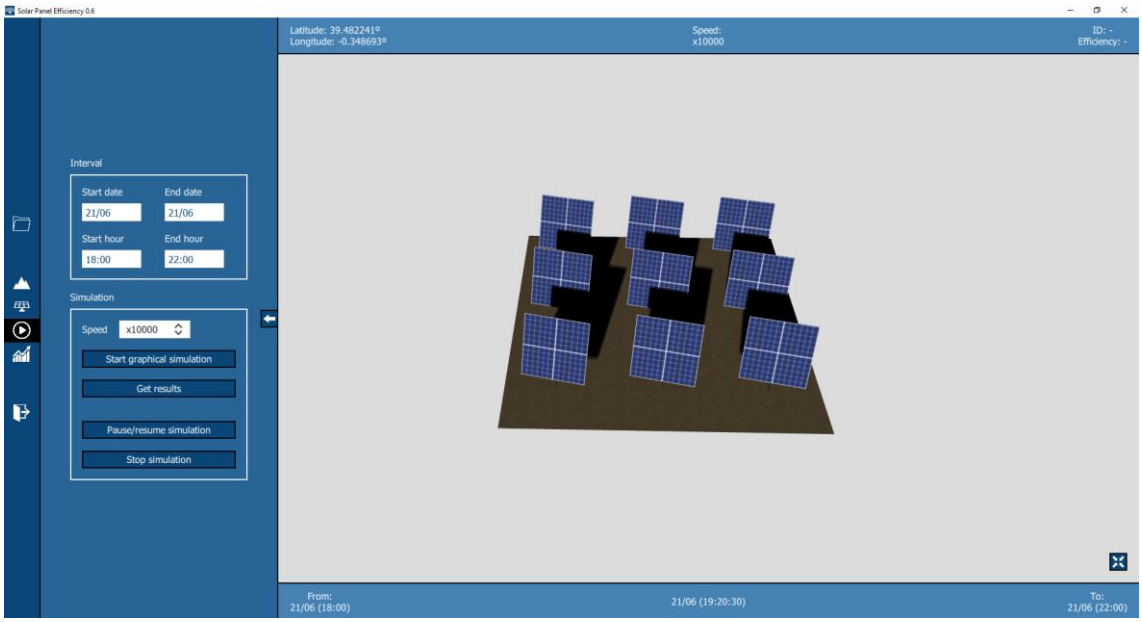

Figura 41. Simulación de una central fotovoltaica

Nótese que durante la ejecución se muestra información en las barras que todavía no se había visto. Hasta ahora solamente se habían visualizado las coordenadas del terreno y la eficiencia de un panel. Tal y como se ha comentado antes, los paneles se pueden pulsar para ver la eficiencia del panel seleccionado en ese momento. Pero ahora, aparte se incluye esta información: la velocidad a la que se está ejecutando la simulación, la fecha y hora de inicio, la fecha y hora que está ocurriendo en el momento, y la fecha y hora de fin.

Aparte de la imagen mostrada, se puede ver la trayectoria que sigue la iluminación del Sol cambiando una variable en el código. Estas guías se añaden fácilmente con Threejs y han ayudado notablemente en la depuración de la aplicación. En la Figura 42 se muestra el terreno visto de lejos, y el contenedor que correspondería con el efecto del Sol. Las dimensiones de este contenedor tuvieron que analizarse bien, en lugar de poner valores grandes sin más. Esto es debido a que cuanto más ancha es la zona que se puede alumbrar, las sombras se

distorsionan más. Por ello, el Sol se ajustó para que ocupara exactamente las dimensiones necesarias para alumbrar el terreno más grande que puede crear la aplicación, el cual es de 50x50 metros. Cabe añadir, que la posición del Sol se obtiene utilizando la librería Suncalc.js, la cual permite ubicar el Sol en función de unas coordenadas y una fecha.

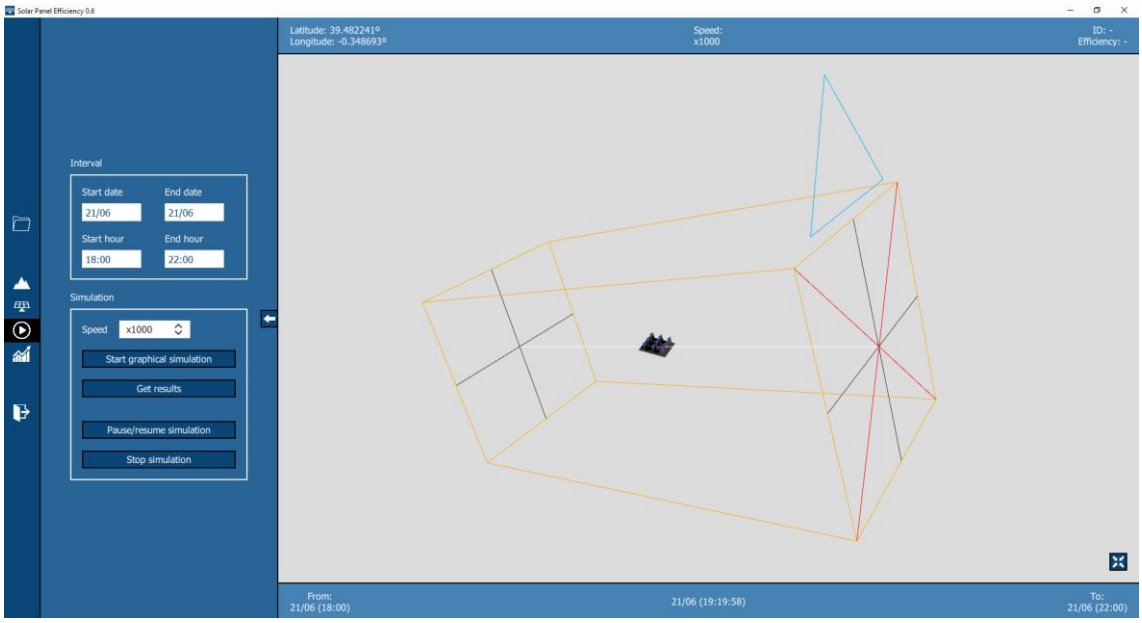

Figura 42. Espacio iluminado por el Sol

6.5.2. Obtención de resultados

Si lo que se quiere es obtener resultados y visualizarlos, se puede pulsar el segundo botón del submenú. En la Figura 43 se puede observar que, al darle a este botón, la escena dibujada desaparece y se muestra un porcentaje de progreso. Cuando la aplicación acaba de calcular los resultados, la escena aparece de nuevo y ya se pueden visualizar los cálculos desde el submenú de resultados. La forma de calcular los resultados se basa en indicar una cámara que enfoque directamente al panel y que se encargue de obtener la parte visible del panel. A continuación, se va a explicar en más detalle este proceso.

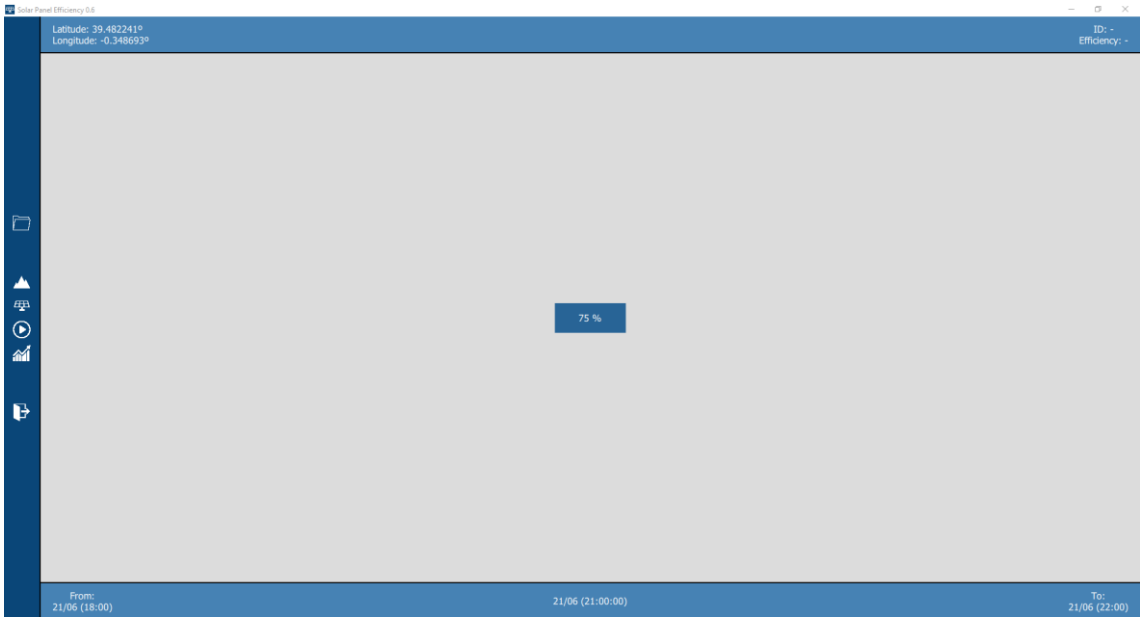

Figura 43. Obtención de resultados

Lo primero que se realiza es ubicar la cámara mencionada enfrente del panel pertinente. Tras ello, se borra la textura del panel y se pinta completamente de blanco. Después, mediante la cámara y el uso de *shaders,* se calcula la media de cada uno de los tres colores (rojo, verde y azul) de los píxeles del panel. Luego, se hace otra media de los tres colores obtenidos, con lo que se consigue un número entre 0 y 255. Este número se podría considerar como la cantidad de color blanco que hay, que se traduce a cuánta luz recibe el panel.

Si el número es 255, el panel se ve totalmente blanco y por lo tanto está recibiendo toda la luz posible. Por el contrario, si el número es 0, el panel se ve totalmente negro y significa que no está nada iluminado. Hay que tener en cuenta que, aunque no se tapen los paneles entre sí, si el Sol no está iluminando de forma perpendicular a los paneles, la luminosidad baja, y por lo tanto el número indicado es menor. Tras calcular el número, este se normaliza para que esté en un rango de 0 a 1, y ese valor será la eficiencia del panel, el cual se puede multiplicar por 100 para visualizarlo en porcentaje. Nótese entonces que, si la media de colores es 255, la eficiencia será 100%. Tras realizar todo este proceso, se vuelve a añadir la textura al panel y se repite lo mismo para el siguiente panel. En la Figura 44 se pueden apreciar las guías de la cámara encargada de obtener los píxeles de los paneles. Estas guías, al igual que las del Sol, solamente se han utilizado para depurar el programa, pero no se ven en una ejecución normal de la aplicación.

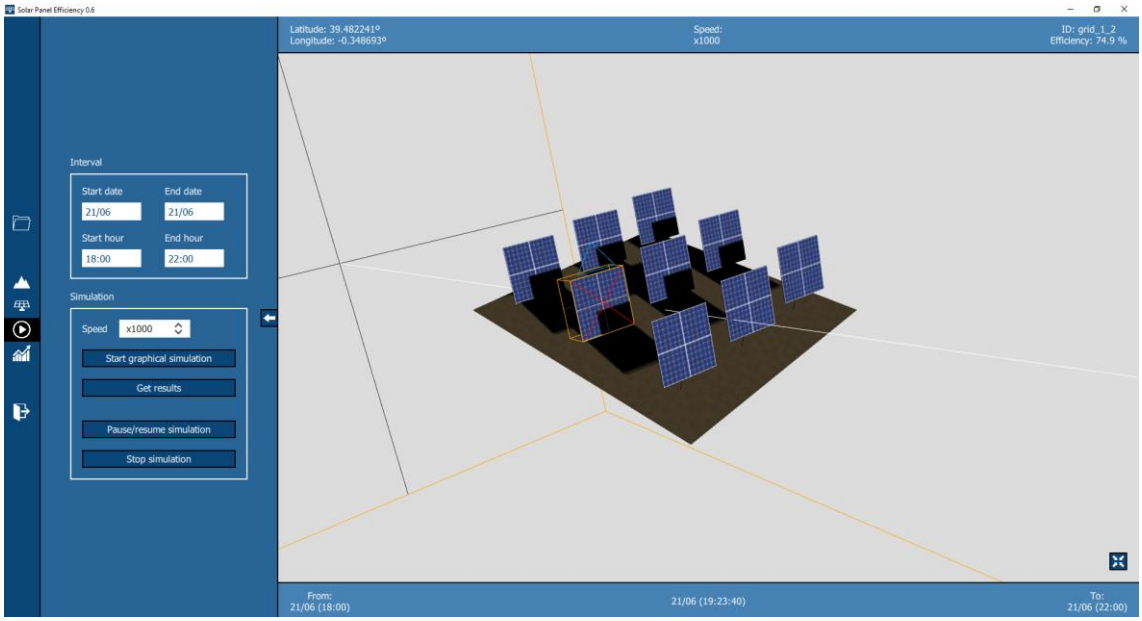

Figura 44. Cámara que obtiene la eficiencia de los paneles

6.5.3. Visualización y exportación de los resultados

Una vez se han obtenido los cálculos, estos se pueden visualizar o exportar desde el submenú de resultados. Como se ha comentado en el apartado de diseño, los cálculos se van guardando en una base de datos debido a que son muchos datos para hacer uso de la memoria RAM. Así que, a la hora de visualizarlos o exportarlos se hacen consultas a dicha base de datos. En la Figura 45 se puede ver el submenú de resultados. En primer lugar, se muestra qué intervalo de datos hay disponible, que dependerá del periodo que se haya simulado. Tras esto, el usuario puede indicar un subintervalo dentro de dicho intervalo para visualizar los resultados. Aparte, se puede indicar el identificador de un panel para ver la eficiencia de un panel concreto. En caso de que el campo destinado a ese identificador se deje en blanco, se muestra la eficiencia media de los paneles. Una vez indicado todo, se puede pulsar el botón de mostrar gráficos y aparece una gráfica como la que se puede observar en la Figura 46.

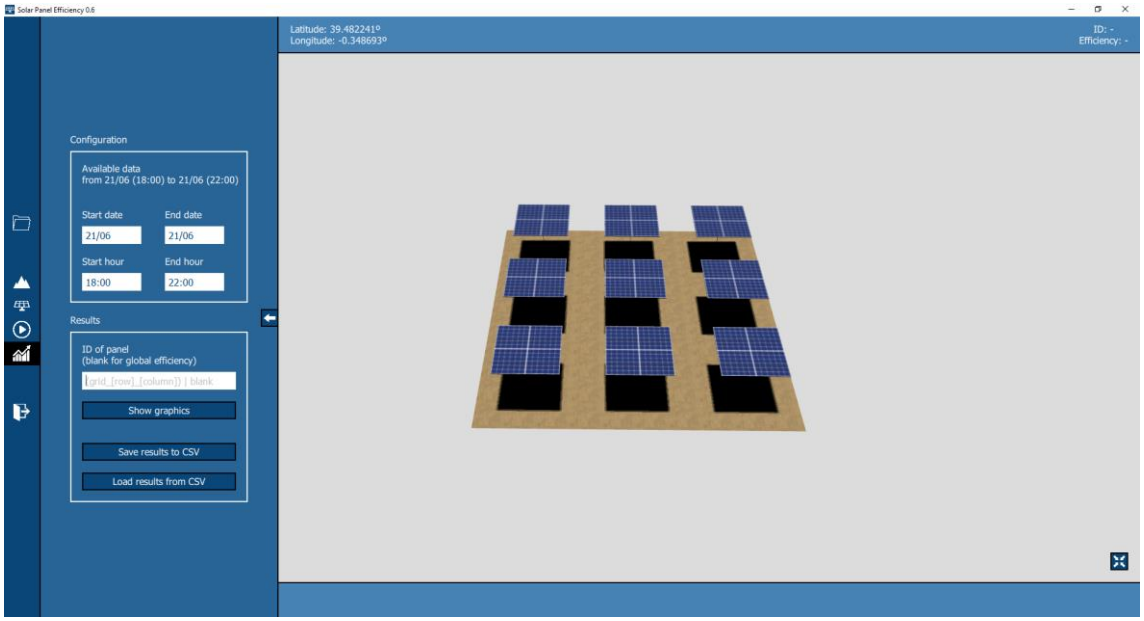

Figura 45. Submenú de resultados

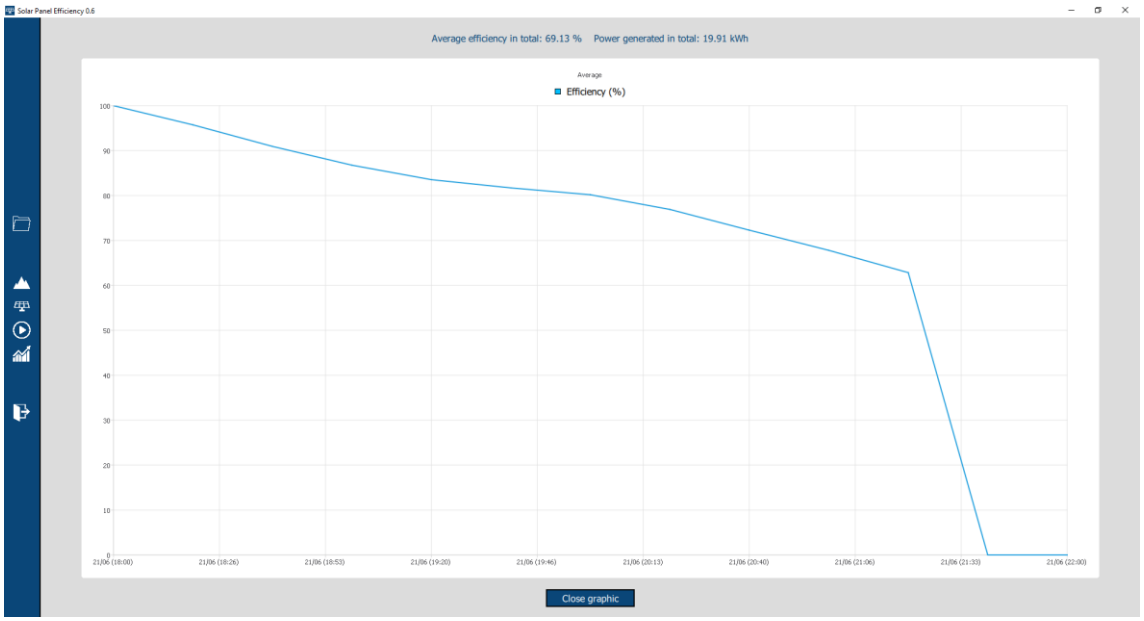

Figura 46. Gráfica de eficiencia

En el eje horizontal de la gráfica se muestran los instantes de tiempo a lo largo del intervalo especificado, y en el eje vertical se indica la eficiencia en porcentaje. En la parte superior de la gráfica se puede observar la eficiencia media de ese intervalo, así como la potencia generada, calculada en función de la eficiencia y la potencia de los paneles. Una vez se han visualizado los resultados, la gráfica se puede cerrar pulsando el botón de la parte inferior. En el submenú,

también se ofrece la posibilidad de exportar los resultados en un fichero CSV. Al exportar, la aplicación vuelca todos los datos de la base de datos en el fichero. Esto quiere decir que no se tiene en cuenta el intervalo que se ha indicado para visualizar la gráfica. Se prefiere guardar todo directamente para su uso posterior. Abriendo la aplicación de nuevo, y sin la necesidad de cargar ninguna central, estos resultados se pueden importar y visualizar de nuevo.

En los requisitos iniciales de la aplicación, se incluía el hecho de que había que obtener una función aproximada a partir de los resultados para poder calcular de forma infinitesimal cualquier punto del intervalo. Tras ello, dado un intervalo dado, se quería calcular la integral de esta función. Sin embargo, esto no se ha realizado por las siguientes razones. Al poder simularse la central durante todo un año, hay una variación clara en la eficiencia: cada noche la eficiencia es 0. Por otro lado, conforme se avanza el año, el comportamiento del Sol varía mucho, por lo que intentar obtener una función de todas estas anomalías era inabordable y poco realista. Dado que se extraen resultados cada un periodo de 20 minutos, es más preciso mostrar los cálculos realizados que si se consiguieran a través de una función aproximada. Por esto, se decidió almacenar los resultados y añadir la funcionalidad de poder exportarlos e importarlos posteriormente.

6.6. Detalles finales

En la última iteración de la aplicación se añadieron una serie de detalles para que la aplicación fuera más usable. Los más destacables son los que se van a comentar a continuación. El primero es que se añadió un indicador de carga mientras se añaden los elementos a la escena, o mientras se cargan los resultados desde el archivo CSV. En la Figura 47 se puede ver dicho indicador. Por otro lado, para ayudar al usuario a la hora de manejar la aplicación, se añadió una etiqueta informativa para cada campo, la cual se mostraba si el ratón se mantenía encima. En la Figura 48 se puede ver la etiqueta de un campo de texto.

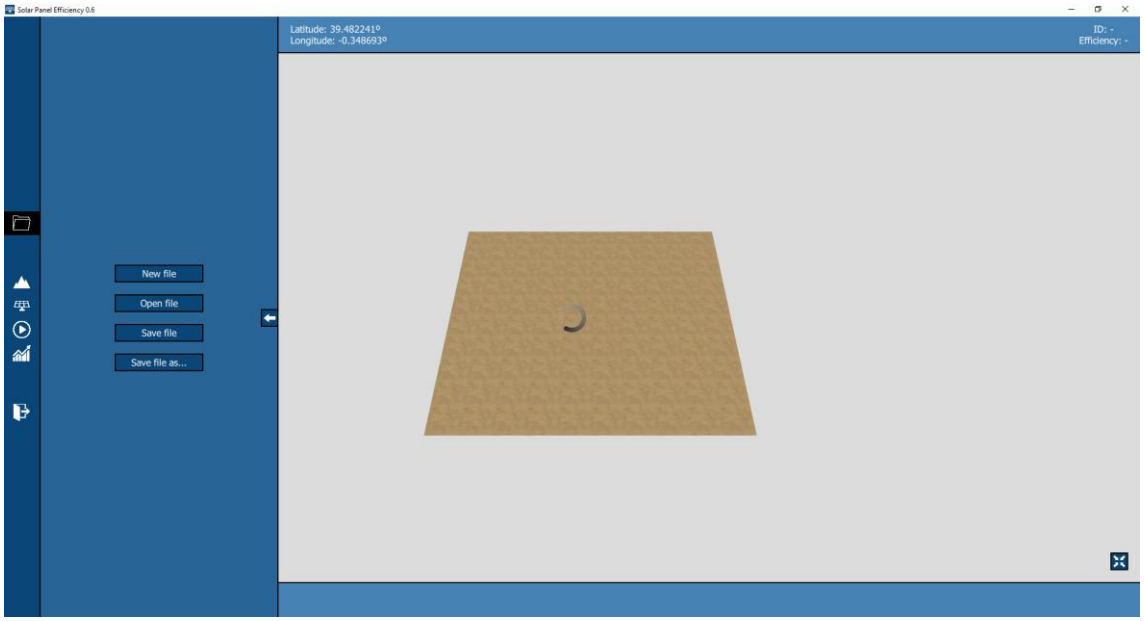

Figura 47. Indicador de carga

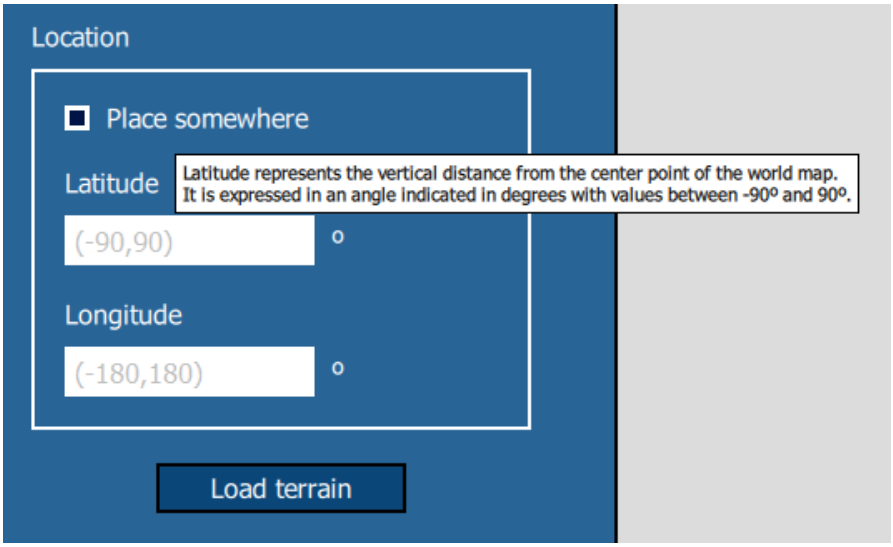

Figura 48. Etiqueta informativa

6.7. Envergadura de la aplicación

Una vez acabado el proyecto, para estimar cuál era la envergadura de la aplicación, se usó la herramienta Cloc, mencionada en el apartado de tecnologías utilizadas. Esta herramienta permite contar los archivos y las líneas de código de un directorio. Así pues, se hizo uso de ella para el directorio “src” de la aplicación, el cual excluye las librerías utilizadas, dejando

solamente el código desarrollado por el alumno. Las estadísticas obtenidas reflejan que hay 49 ficheros y 4416 líneas de código, donde no se tienen en cuenta líneas en blanco ni comentarios. Con estos valores se puede apreciar que la aplicación tiene una envergadura considerable.

7. Implantación

La implantación en un proyecto se refiere a la puesta en producción del sistema desarrollado. En aplicaciones web, esto implicaría configurar los servidores web y los servidores de bases de datos. Sin embargo, en este caso este proceso se simplifica, ya que el sistema desarrollado es una aplicación que simplemente se tiene que instalar en el dispositivo del usuario. Como ya se ha comentado anteriormente, Qt permite desarrollar aplicaciones multiplataforma. Por lo que el propio entorno aporta las herramientas necesarias para poder empaquetar todos los archivos necesarios en el formato correcto para que la aplicación pueda ser ejecutada en el dispositivo del usuario. Así pues, la aplicación desarrollada podría funcionar en cualquier dispositivo con cualquier sistema operativo. Esto incluye Windows, macOS, Linux, Android, iOS y otros sistemas menos conocidos, como los utilizados por otros dispositivos como televisiones o relojes [27].

Sin embargo, aunque Qt proporcione las herramientas para desplegar en todas las plataformas, cada una tiene sus peculiaridades. Por lo tanto, en este proyecto solamente se ha probado con los sistemas operativos Windows y Android. Si en futuras iteraciones se querría disponer de la aplicación para más plataformas, habría que analizar cómo realizar la configuración para la plataforma deseada, pero no se necesitaría un desarrollo extra. Respecto a los sistemas probados, en el proyecto se ha focalizado más en la plataforma Windows. Cuando se realizó el primer prototipo se realizaron pruebas tanto en Windows como en Android y funcionaba correctamente.

No obstante, conforme fue avanzando la aplicación, esta se hacía más compleja y requería más potencia para la visualización de los gráficos y los cálculos. Así que, utilizar la aplicación en dispositivos móviles dejó de ser recomendable. Por ello, no se ha verificado que la aplicación final funcione a la perfección en dispositivos móviles. Si llegase a ser una opción viable, se podría llegar a una aplicación completamente usable en poco tiempo, pues el código ya está escrito. A continuación, se explicará cómo se pueden generar todos los archivos necesarios para que la aplicación se pueda ejecutar en los entornos de Windows y Android. El siguiente paso sería subir dichos archivos a una página web o a las diferentes tiendas de aplicaciones de los sistemas.

7.1. Implantación en Windows

La configuración para usar la aplicación en Windows es sencilla. En la Figura 49 se pueden ver los comandos requeridos para ello. Los dos primeros siempre se configuran por defecto para que la aplicación pueda compilar. Sin embargo, el último hay que indicarlo manualmente. En este, se ejecuta la herramienta "windeployqt.exe" con los parámetros que se pueden ver en la figura. Aquí, lo más relevante es que se debe indicar la ubicación de los ficheros QML y el directorio donde deben generarse los archivos. Tras realizar esto, se obtiene un archivo ejecutable acompañado de todas las librerías necesarias. Este archivo ejecutable permite arrancar la aplicación en cualquier dispositivo con Windows, aunque no se tenga Qt instalado.

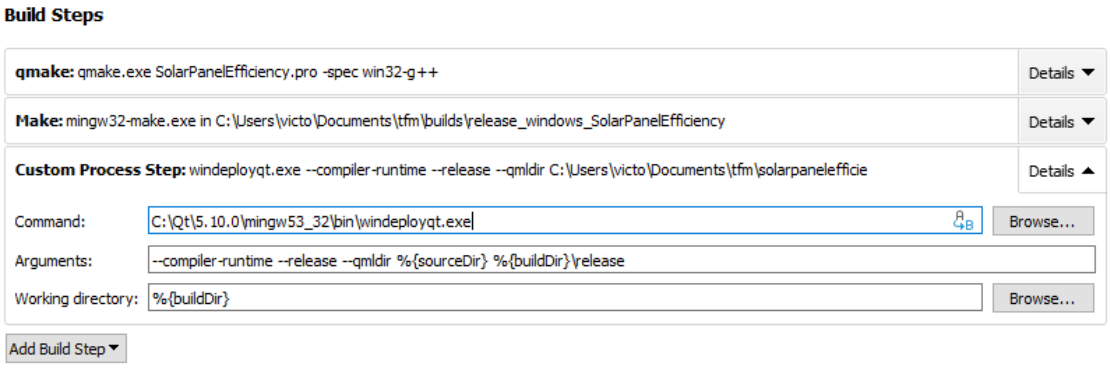

Figura 49. Configuración para implantación en Windows

7.2. Implantación en Android

La implantación en Android es más complicada que la de Windows porque es necesario instalar una serie de herramientas. En concreto se requiere una instalación de Java, los paquetes SDK y NDK de Android, y el AVD Android Virtual Device. Java es necesario debido a que las aplicaciones en Android funcionan con este lenguaje. SDK es el paquete principal de Android, mientras que NDK se requiere si se ha programado con otros lenguajes distintos a Java. Finalmente, AVD Android Virtual Device permite crear máquinas virtuales de dispositivos móviles para probar la aplicación sin la necesidad de un dispositivo real. Tras instalar todo esto, se deben indicar las rutas pertinentes en la configuración de Qt. Dicha configuración se puede apreciar en la Figura 50.

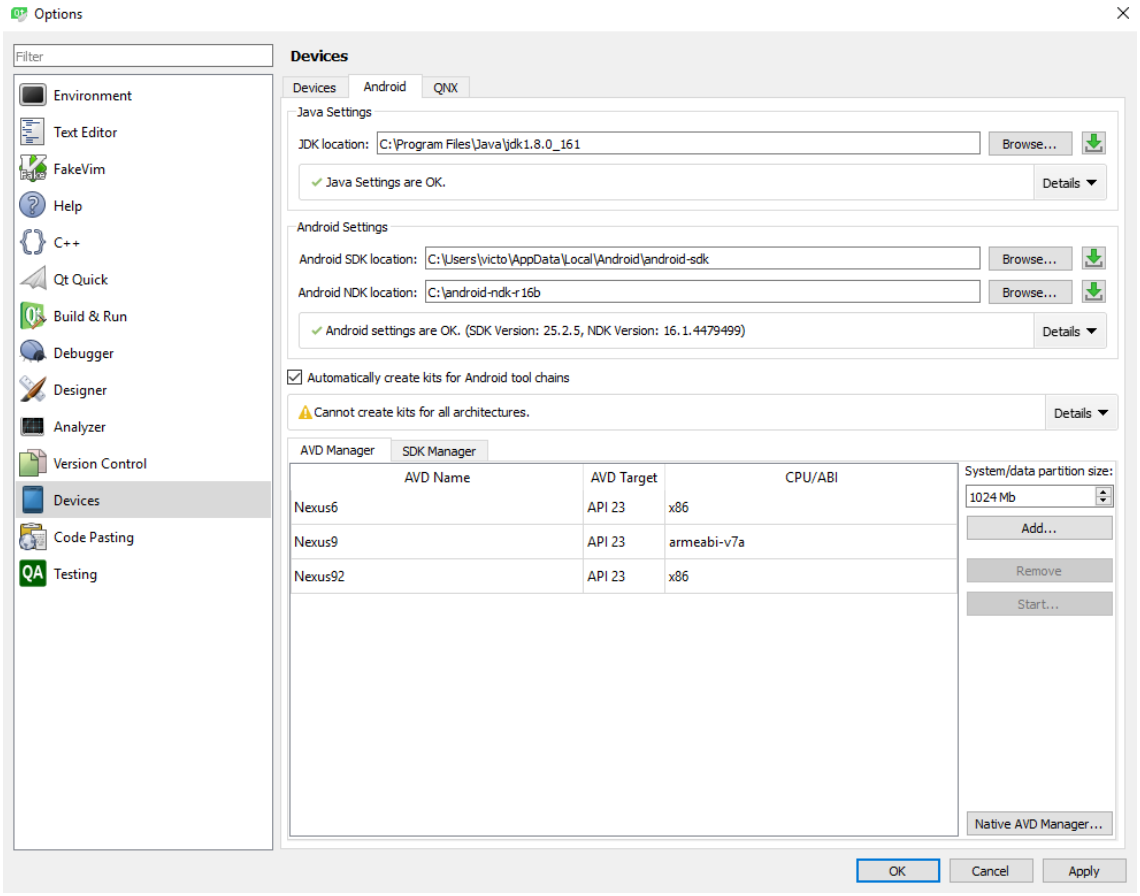

Figura 50. Configuración de las herramientas Android

Mientras que en Windows se genera un ejecutable junto a los archivos necesarios para que este funcione, en Android todos los archivos se empaquetan en un archivo con extensión APK. Tras instalar todas las herramientas mencionadas, se pueden ejecutar una serie de comandos similares a los que había en Windows para compilar la aplicación y generar el archivo APK. Concretamente para Android, estos comandos ya vienen dados por defecto en Qt. En la Figura 51 se pueden observar los pasos que se ejecutan. Tras esto, se genera el APK y aparte se lanza la aplicación en la máquina virtual que se haya configurado. En la Figura 52 puede verse la aplicación dentro del simulador.

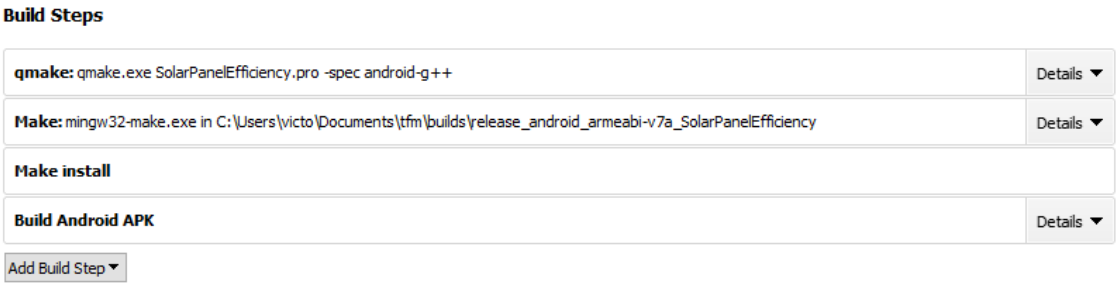

Figura 51. Configuración para implantación en Android

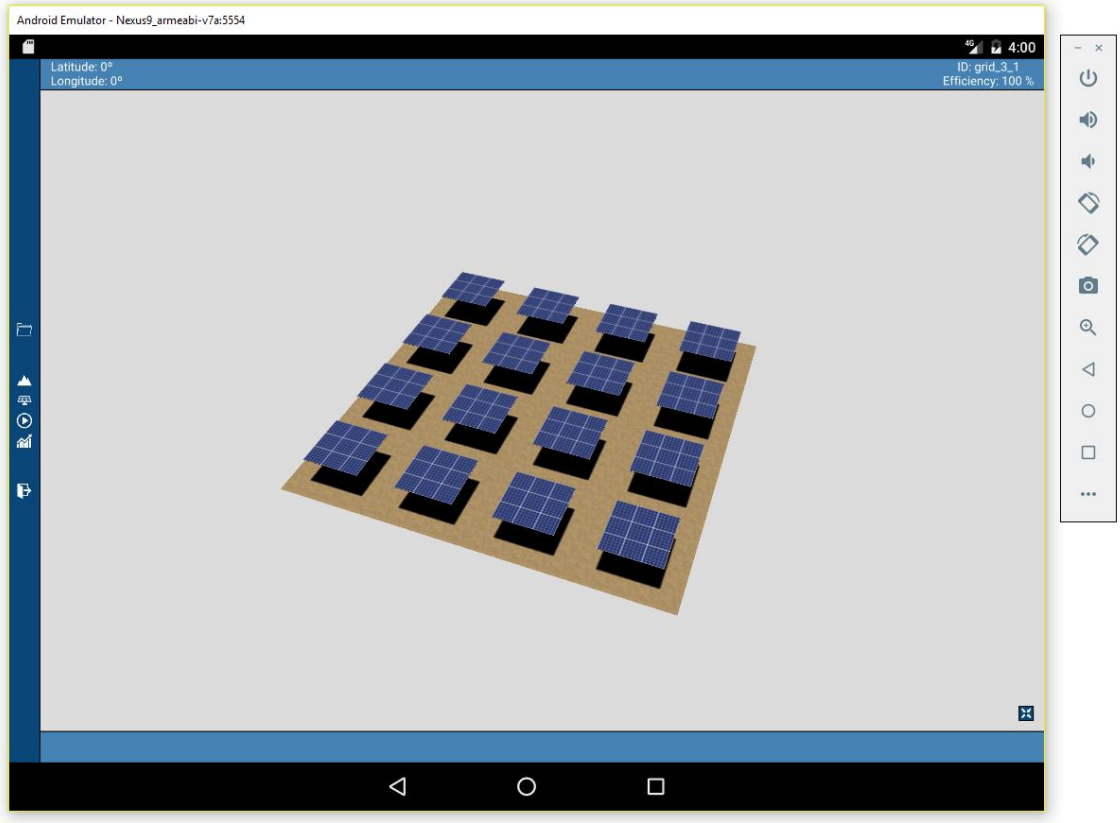

Figura 52. La aplicación en una máquina virtual de un dispositivo móvil

8. Pruebas

Para asegurar el correcto funcionamiento de la aplicación y velar por su calidad, se han realizado distintos tipos de pruebas. Estas pruebas se han realizado con cada iteración del proyecto, por lo que cada vez que se comenzaba una nueva iteración, se partía de un producto estable, probado y verificado. Las primeras pruebas que se hacían eran las pruebas estáticas. Con ellas, se revisa el código de la aplicación por si este se puede mejorar. Hay que tener en cuenta, que en este tipo de pruebas no se ejecuta la aplicación, de ahí que se llamen estáticas. Tras ello, se hacían pruebas funcionales, donde básicamente se probaba el correcto funcionamiento de la aplicación. Después, se comprobaba que el rendimiento de la aplicación, el consumo de la memoria RAM y los tiempos de espera eran adecuados para una aplicación eficiente y usable. Finalmente, cada versión se enviaba al tutor junto a un vídeo de demostración y una explicación de la funcionalidad desarrollada. Una vez enviado, el tutor verificaba que la aplicación cumplía con las expectativas, lo que podría considerarse como pruebas de aceptación.

Como se ha mencionado antes en el apartado de tecnologías utilizadas, para el análisis estático del código se ha hecho uso de SonarQube. Esta herramienta indica los fallos y malas prácticas en el código y sugiere cómo debería solucionarse. Cada lenguaje de programación dispone de una serie de reglas que deben cumplirse. Un ejemplo se puede ver en la Figura 53. En esta regla, básicamente se indica que el uso del operador "==" es menos restrictivo que "===", por lo que es recomendable utilizar el segundo. Otra característica interesante de SonarQube aparte de las reglas es el hecho de que analiza la complejidad de las funciones. Para ello, con cada sentencia que altere el flujo de la ejecución (como un condicional o un bucle), SonarQube va sumando puntos de complejidad. Si se sobrepasa un umbral establecido, se indica que debe reducirse la complejidad de dicha función. Por último, SonarQube también detecta el código duplicado, el cual es muy común durante un desarrollo.

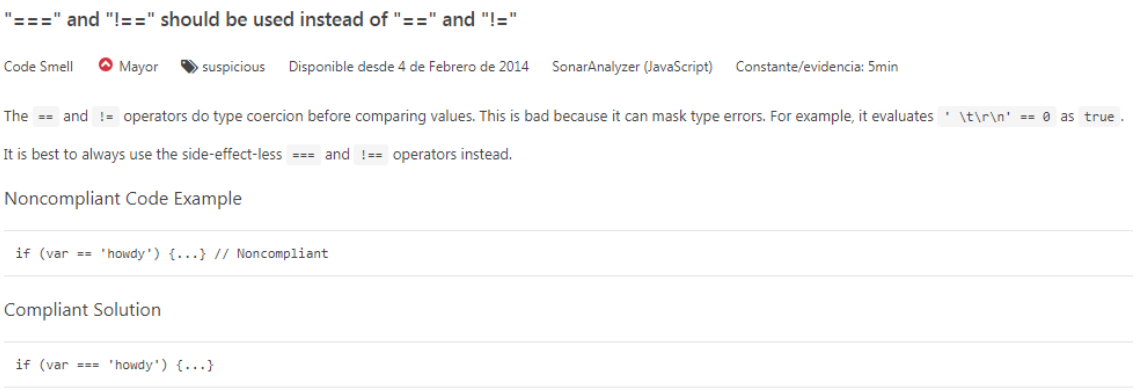

Figura 53. Regla de SonarQube

Todas estas incidencias que registraba SonarQube se han ido subsanando, lo que ha permitido que la calidad del código de la aplicación aumentara. Esto se puede visualizar en el panel de mandos que proporciona SonarQube, donde se hace un resumen del cumplimiento de la calidad del código. En este resumen se tienen en cuenta tres métricas: fiabilidad, seguridad y mantenibilidad. Aparte de esto, también se indican otros parámetros, como el porcentaje de código duplicado o el tiempo estimado necesario para arreglar todas las malas prácticas. En la Figura 54 se puede observar el panel de la aplicación, donde no hay ninguna incidencia registrada porque ya se han subsanado todas. Hay que mencionar que SonarQube no puede analizar el código QML, pero esto tampoco es crítico, ya que QML solamente se utiliza para describir la interfaz.

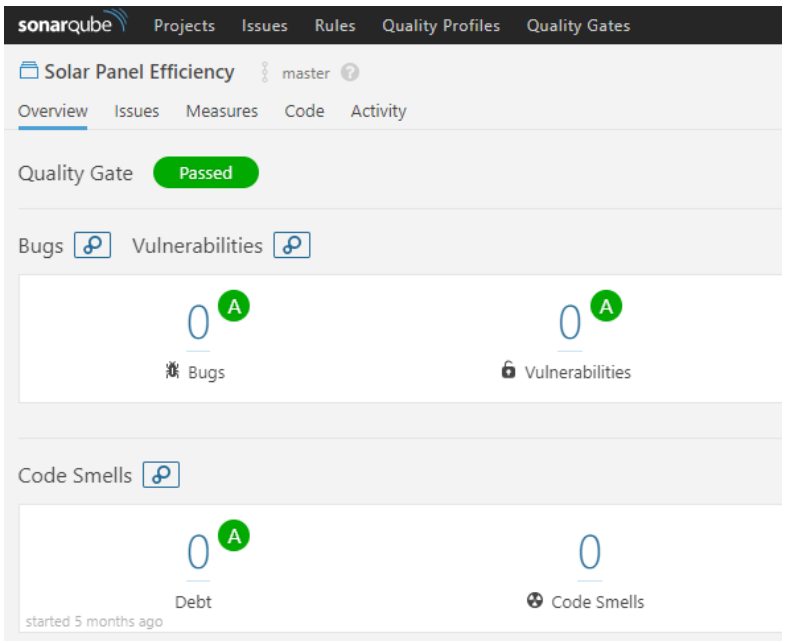

Figura 54. Panel de mando de SonarQube

Respecto al resto de pruebas que se han realizado, estas han permitido asegurar el correcto funcionamiento de la aplicación y verificar que el programa cumplía con los requisitos. En esta aplicación se ha focalizado especialmente en la usabilidad y el rendimiento de la aplicación. Esto ha llevado a cuidar los detalles que facilitan el uso del programa. Por ejemplo, tras probar la aplicación en un estado intermedio, surgió la necesidad de añadir las etiquetas informativas de cada campo, comentadas en el apartado 6. Por otro lado, las pruebas de rendimiento han permitido ajustar los valores límite de la aplicación, como por ejemplo el número máximo de vértices que puede tener un terreno.

8.1. Pruebas finales

Una vez se acabó la aplicación, se realizaron una serie de pruebas para validar que la aplicación realmente podía aportar valor al usuario. Para ello, se pensó en obtener los resultados de eficiencia un día de verano y compararlos con los resultados de un día de invierno. Como en verano hay más horas de luz que en invierno, se quería comprobar que la eficiencia en la primera estación era superior a la que se podía obtener en la segunda. Por lo tanto, se creó una central fotovoltaica ubicada en España y se simuló dos veces, una durante el día 21/06 y la otra durante el día 21/12. En la Figura 55 y la Figura 56 se puede ver los resultados obtenidos en el día de verano y en el día de invierno.

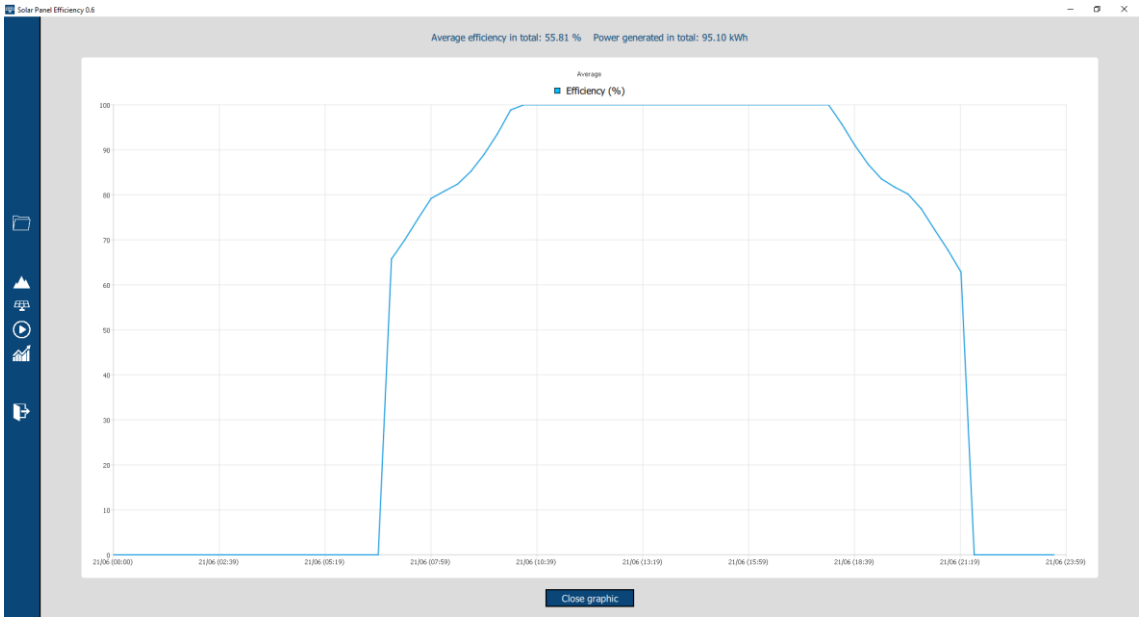

Figura 55. Eficiencia en un día de verano

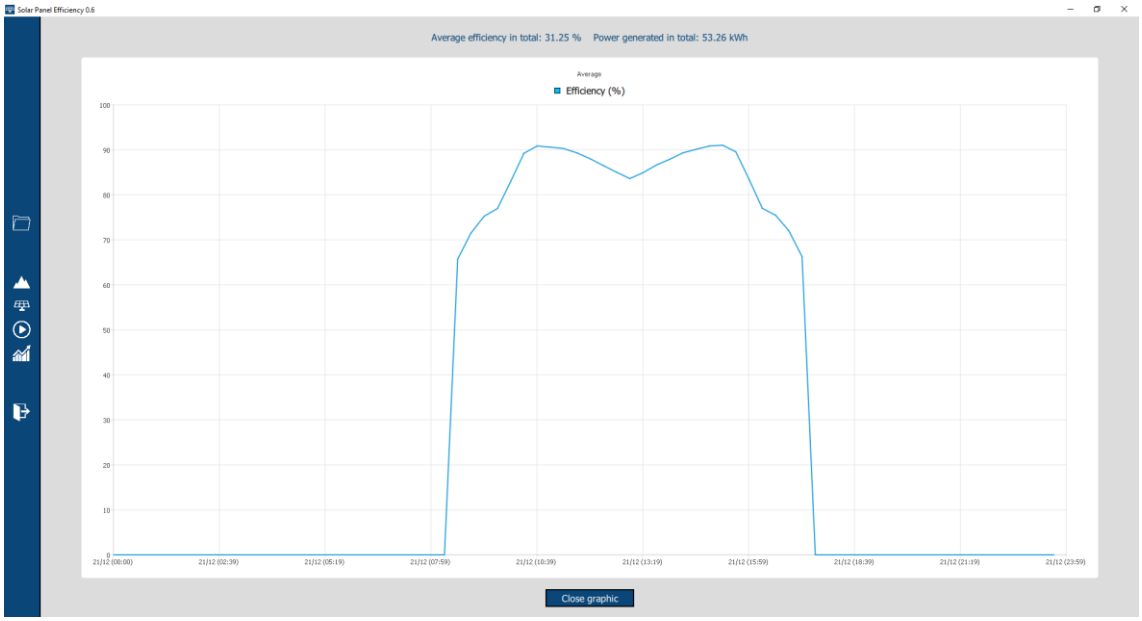

Figura 56. Eficiencia en un día de invierno

Como se puede observar, en el día de invierno se capta luz solar desde las 8:00 hasta las 18:00 aproximadamente, mientras que en el día de verano se capta desde las 7:00 hasta las 21:00. Esto deriva en que en invierno la eficiencia es de 31.25% y la potencia generada es 53.26 kWh. En verano, la eficiencia asciende hasta 55.81% y la potencia es 95.10 kWh. Por lo tanto, la aplicación funciona como se esperaba. Igualmente, al realizar estar pruebas, en la gráfica de invierno se detectó una anomalía. En la Figura 56 se puede ver que sobre las 13:00 la eficiencia baja y luego vuelve a subir. Esto resultaba un poco extraño y en la gráfica de verano no se podía apreciar nada similar. Por ello, se utilizó la simulación gráfica para ver qué ocurría exactamente a esa hora. Con esto se demostró que la aplicación podía ser muy útil a la hora de trabajar con centrales fotovoltaicas, ya que se permite obtener resultados de eficiencia, pero además se puede comprobar cómo se han obtenido dichos resultados. Tras la simulación, se pudo observar lo que se muestra en las siguientes figuras. Las imágenes enseñan la central fotovoltaica a las 12:00, a las 13:00 y a las 14:00 respectivamente.

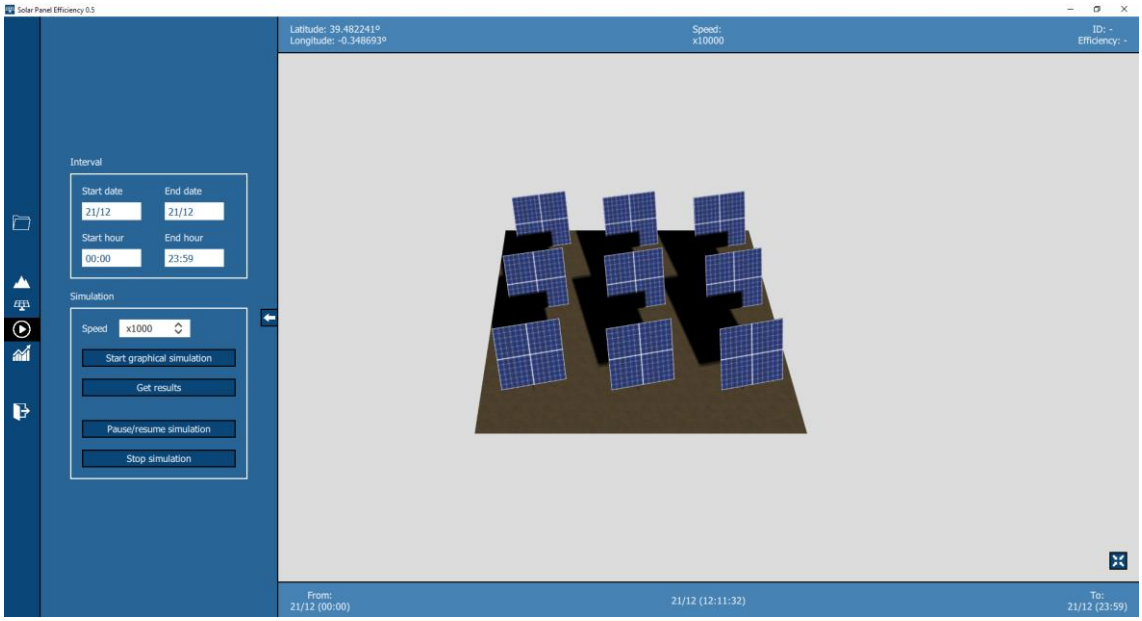

Figura 57. Central fotovoltaica en invierno a las 12:00

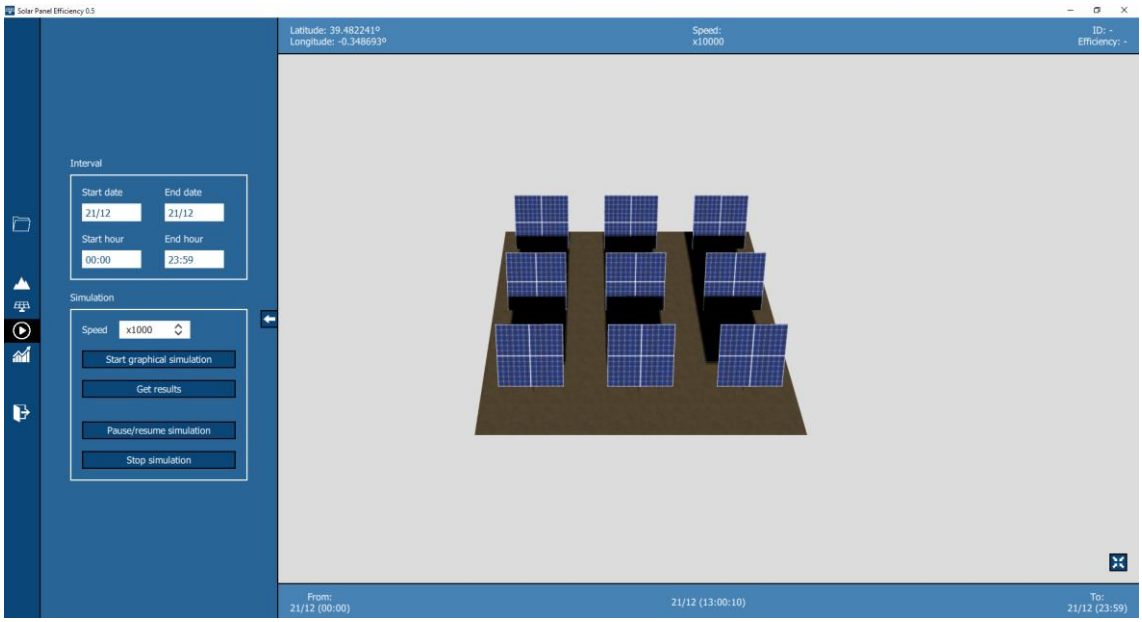

Figura 58. Central fotovoltaica en invierno a las 13:00

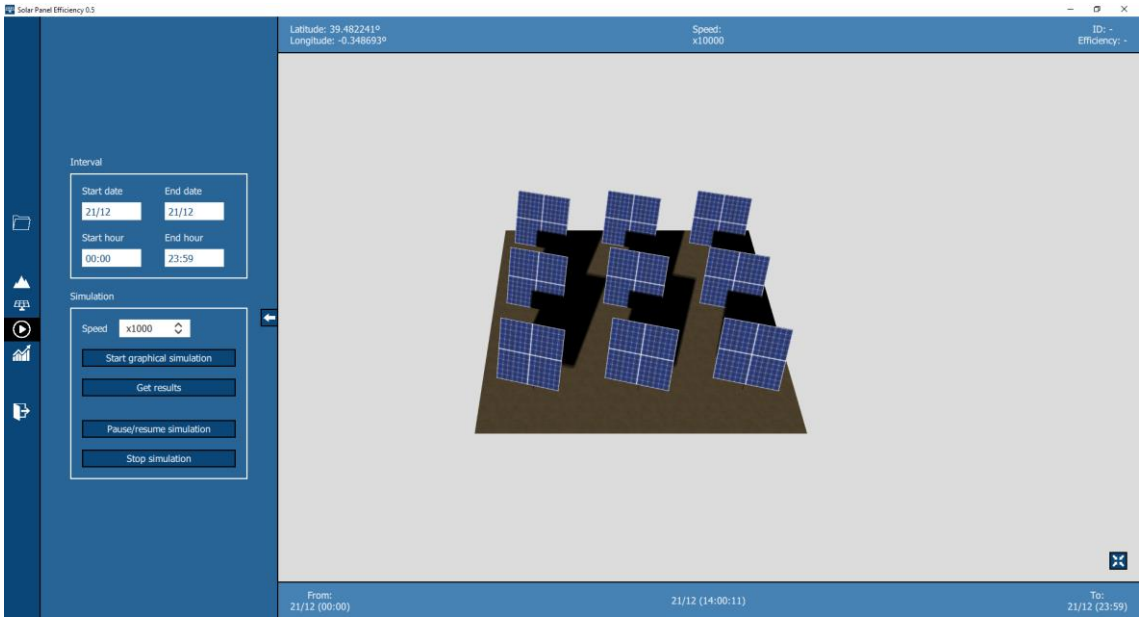

Figura 59. Central fotovoltaica en invierno a las 14:00

Aquí se puede observar que, justamente a las 13:00, los paneles se orientan de una forma que provoca que se tapen más entre sí que a las 12:00 o a las 14:00. Con esto, se puede explicar esa bajada en la eficiencia. Por otro lado, también se puede observar qué ocurre a estas horas en verano. En la Figura 60 se muestra la central fotovoltaica a las 13:00 el día 21/06. Aquí se puede observar que el Sol se posiciona más alto que en invierno y por ello no se tapan los paneles entre sí. Así que, tras estas pruebas se pudo comprobar que la aplicación funcionaba correctamente.

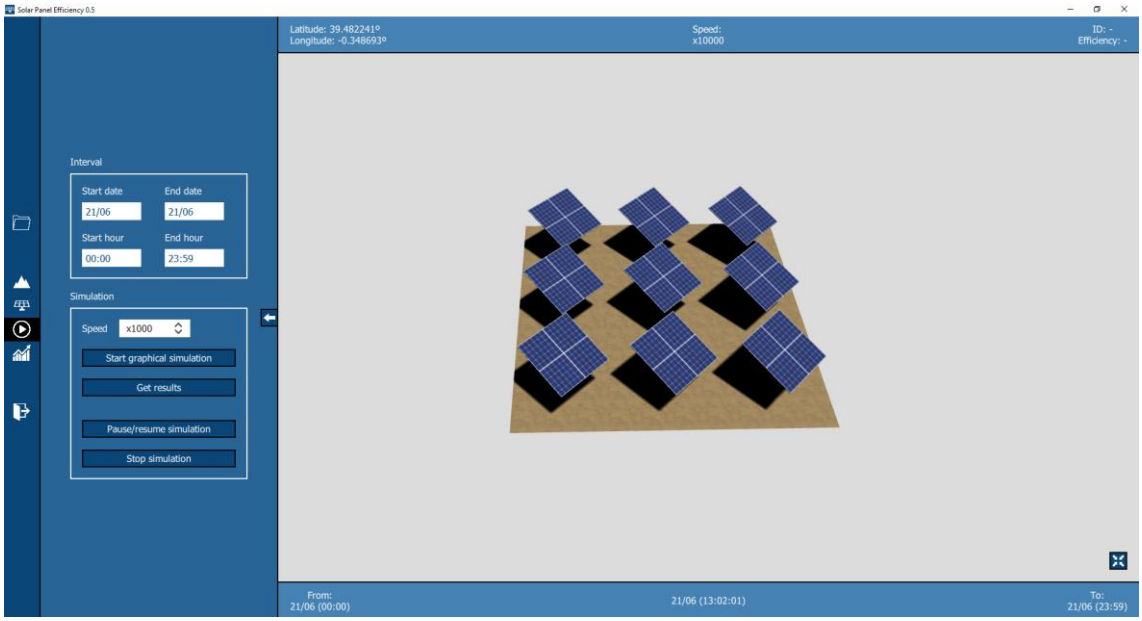

Figura 60. Central fotovoltaica en verano a las 13:00

9. Conclusiones

Tras la realización de este trabajo se puede ver que los objetivos planteados al inicio del proyecto se han cumplido satisfactoriamente. La aplicación desarrollada permite crear una central fotovoltaica y hacer una simulación para obtener la eficiencia de los paneles solares en un intervalo de tiempo. También se han cumplido los objetivos más específicos, como el poder crear un terreno a partir de un mapa de alturas. En concreto, el requisito de obtener una función a partir de los resultados y después obtener la integral no se ha realizado. Pero esto se debe a que no era factible y tampoco era necesario, tal y como se ha explicado en el apartado 6.5.

No solo se han cumplido los objetivos establecidos, sino que además se han hecho una serie de ampliaciones que no estaban especificadas en los requisitos. Un ejemplo es haber añadido la potencia de los paneles para luego poder calcular la energía generada final. Por otro lado, también se incluyó el poder exportar e importar los resultados haciendo uso de ficheros CSV. Aparte, la aplicación se ha complementado con funcionalidades transversales como la capacidad de poder guardar y cargar las centrales fotovoltaicas, o el hecho de que cada campo cuente con una etiqueta informativa.

También cabe comentar que la elección de la tecnología fue un acierto. Combinar la potencia de Threejs con la sencillez de QML ha permitido realizar una aplicación muy vistosa y eficaz. Realizar el proyecto con las otras tecnologías planteadas en el apartado 4 habría sido factible, pero seguramente no se habría conseguido el mismo resultado. Por otro lado, el resto de las herramientas utilizadas también han sido de mucha utilidad para el desarrollo del proyecto. Por ejemplo, el uso de Git ha permitido trabajar con la seguridad de poder ir guardando los cambios que se iban realizando en la aplicación. Y la aplicación Pencil ha sido muy útil a la hora de realizar los diseños de forma ágil y sencilla.

Durante el desarrollo de la aplicación se han encontrado distintos retos. Uno de ellos fue crear un terreno a partir de un mapa de alturas, donde se tenía que tener en cuenta que no se podía dividir el plano en muchos vértices por temas de rendimiento. Otro fue distribuir los paneles de forma uniforme por todo el terreno. Y la simulación y la obtención de resultados en general fue compleja, pero al final se consiguió resolver todo con éxito. Por otra parte, aunque Threejs se dé en el máster, no era una tecnología en la que se tenía mucha experiencia, por lo que se tuvo que aprender un poco más sobre ella y consultar continuamente la documentación oficial.

A nivel personal, este proyecto me ha permitido conocer mejor el campo de la computación gráfica, así como las tecnologías utilizadas. Por otro lado, he conseguido planificar y diseñar mejor este proyecto que los realizados a lo largo del grado universitario. También cabe mencionar que este proyecto me pareció interesante desde que vi la propuesta y que he disfrutado mucho realizando la aplicación. Aparte, es motivador trabajar sobre energías renovables, un tema tremendamente importante y en el que se debe promover más su uso e investigación. Respecto a la aplicación desarrollada, opino que los resultados han sido totalmente satisfactorios, especialmente los explicados en el apartado 8.1, donde se habla sobre las pruebas finales. Para desarrollos futuros, esta aplicación podría evolucionar de forma considerable, pues cuenta con un gran potencial para realizar muchas mejoras e implementar nueva funcionalidad.

9.1. Relación del trabajo desarrollado con los estudios cursados

El trabajo realizado está relacionado con varios ámbitos que se dan en el máster. En primer lugar, se han aplicado conocimientos obtenidos sobre planificación y dirección de proyectos. De aquí, se han aprovechado las metodologías ágiles, la estimación de tareas y el uso de herramientas como Trello. En segundo lugar, los temas sobre calidad se han utilizado durante el desarrollo de todo el proyecto, permitiendo obtener una aplicación fiable, eficiente y usable. Finalmente, las asignaturas más relevantes para el desarrollo del trabajo han sido las relacionadas con las aplicaciones gráficas, donde se han utilizado los conocimientos aprendidos continuamente.

10. Trabajos futuros

Como se ha comentado en otros apartados, la aplicación desarrollada tiene mucho potencial para ser ampliada. En este apartado se enumeran las características nuevas que podrían implementarse. Algunas de ellas ya se mencionaron en el apartado de requisitos futuros y otras han surgido mientras se desarrollaba la aplicación. La lista se muestra a continuación:

- Soporte específico para simulaciones de colectores solares, que sirven para calentar el agua en hogares
- Cálculo de costes
- Base de datos para almacenar distintos tipos de paneles solares

- Función para encontrar automáticamente la mejor configuración dado un terreno y unos parámetros económicos
- Mejoras en la interfaz
- Posibilidad de seleccionar un terreno mediante vista satélite
- Capacidad para ubicar cada panel en un punto específico del terreno
- Posibilidad de añadir otros elementos que puedan provocar sombras, como por ejemplo edificios
- Ampliación de las características de los paneles para ser más precisos con los cálculos
- Inclusión de predicciones de meteorología en la simulación, la cual puede afectar al rendimiento de los paneles
- Posibilidad de cargar elementos auxiliares necesarios en una central fotovoltaica y capacidad de añadir cálculos relativos a estos elementos, como por ejemplo cálculos para los cables que conectan la central
- Soporte específico para simulaciones de usos alternativos de la energía solar, como carreteras o tejas solares
- Aumento de la eficiencia de la aplicación y de los límites establecidos en el desarrollo actual
- Generación de documentos y planos para facilitar la implantación de los paneles solares
- Desarrollo de la aplicación para que pueda ser ejecutada en la nube

11. Referencias

[1] Microsoft. (2018). *Trabajar con sombreadores* [online]. Consultado el 01/06/2018. Disponible en: https://msdn.microsoft.com/es-es/library/hh873117.aspx

[2] Conectart. (2018). *Metodologías ágiles* [online]. Consultado el 01/06/2018. Disponible en: https://blog.conectart.com/metodologias-agiles/

[3] K Beck, M. Beedle et al. (2001). *Principles behind the Agile Manifesto* [online]. Consultado el 01/06/2018. Disponible en: http://agilemanifesto.org/principles.html

[4] J. M. Casas, F. Gea et al. *Educación medioambiental*. Alicante (España): Editorial Club Universitario. 2007.

[5] J. Puig, J. Corominas. *La ruta de la energía*. Barcelona (España): Editorial Anthropos. 1990.

[6] Remica. (2018). *Tipos de energía: Diferencias entre renovables y no renovables* [online]. Consultado el 04/06/2018. Disponible en: https://msdn.microsoft.com/es-es/library/hh873117.aspx

[7] "Renewable energy, Meidum-Term Market Report, 2015" [online], International Energy Agency, Paris (France), Exe. Sum. 2015. Consultado el 04/06/2018. Disponible en: https://www.iea.org/Textbase/npsum/MTrenew2015sum.pdf

[8] "Snapshot of global photovoltaic markets" [online], International Energy Agency, Paris (France), Rep. 2016. Consultado el 04/06/2018. Disponible en: http://www.iea-pvps.org/fileadmin/dam/public/report/statistics/IEA-PVPS_-_A_Snapshot_of_Global_PV_-_1992-2016__1_.pdf

[9] I. Mártil, "Evolución y perspectivas para la energía solar fotovoltaica" [online], *Público*, abril 2016. Disponible en: http://blogs.publico.es/econonuestra/2016/04/01/evolucion-y-perspectivas-para-la-energia-solar-fotovoltaica/

[10] Catálogo, Estructura Paneles Solares [online], AutoSolar, Valencia (España). Consultado el 04/06/2018. Disponible en: https://autosolar.es/estructura-paneles-solares

[11] Alternative Energy Tutorials. (2015). *Solar Panel Orientation* [online]. Consultado el 04/06/2018. Disponible en: http://www.alternative-energy-tutorials.com/solar-power/solar-panel-orientation.html

[12] Solar Panel Tilt. (2017). *Optimum Tilt of Solar Panels* [online]. Consultado el 04/06/2018. Disponible en: http://www.alternative-energy-tutorials.com/solar-power/solar-panel-orientation.html

[13] I. Reda, A. Andreas, “Solar Position Algorithm for Solar Radiation Applications” [online], National Renewable Energy Laboratory, Colorado (U.S.A.), Rep. 2008. Consultado el 05/06/2018. Disponible en: https://www.nrel.gov/docs/fy08osti/34302.pdf

[14] E. Sebastian. (2012). *Potencia y Energía con los paneles FV* [online]. Consultado el 05/06/2018. Disponible en: http://eliseosebastian.com/potencia-y-energia-en-paneles-solares-fotovoltaicos/

[15] T. Ward, “This is the most efficient solar panel ever made” [online], *Word Economic Forum*, Aug 2017. Consultado el 06/06/2018. Disponible en: https://www.weforum.org/agenda/2017/08/this-is-the-most-efficient-solar-panel-ever-made

[16] Catálogo, Directorio de Paneles [online], ENF Solar, Woking (U.K.). Consultado el 06/06/2018. Disponible en: https://es.enfsolar.com/pv/panel

[17] Europapress. (2016). *Un 11% de los hogares tiene contratada una potencia eléctrica superior a la necesaria, según Holaluz* [online]. Consultado el 06/06/2018. Disponible en: http://www.europapress.es/economia/energia-00341/noticia-11-hogares-tiene-contratada-potencia-electrica-superior-necesaria-holaluz-20161028144624.html

[18] TheSolarPlanner. (2015). *PV Module Specs and Features* [online]. Consultado el 06/06/2018. Disponible en: http://www.thesolarplanner.com/pv_panel_features.html

[19] E. Allam. (2017). *7 Most Popular Solar PV Design and Simulation Software* [online]. Consultado el 07/06/2018. Disponible en: https://www.linkedin.com/pulse/7-most-popular-solar-pv-design-simulation-software-eslam-allam/

[20] Laplace Systems. (2014). *About Laplace Solar* [online]. Consultado el 08/06/2018. Disponible en: http://www.laplacesolar.com/about/

[21] Laplace Systems. (2014). *Shading Analysis* [online]. Consultado el 08/06/2018. Disponible en: http://www.laplacesolar.com/photovoltaic-products/solar-pro-pv-simulation-design/shading-analysis/

[22] G. Mendez. (2008). *Especificación de Requisitos según el estándar de IEEE 830* [online]. Consultado el 10/06/2018. Disponible en: https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

[23] Smartsheet. (2018). *Building Valuable Software Requirement Specifications for Better Software Development* [online]. Consultado el 08/06/2018. Disponible en: http://www.laplacesolar.com/about/

[24] Khronos Group. (2018). *OpenGL ES for the Web* [online]. Consultado el 16/06/2018. Disponible en: https://www.khronos.org/webgl/

[25] Oracle. (2018). *¿Qué es la tecnología Java y para qué la necesito?* [online]. Consultado el 16/06/2018. Disponible en: https://java.com/es/download/faq/whatis_java.xml

[26] V. Driessen. (2010). *A successful Git branching model* [online]. Consultado el 18/06/2018. Disponible en: https://nvie.com/posts/a-successful-git-branching-model/

[27] The Qt Company. (2018). *Supported Platforms* [online]. Consultado el 18/06/2018. Disponible en: https://doc.qt.io/qt-5/supported-platforms.html

Anexo. Glosario de términos

En este anexo, se van a explicar brevemente una serie de términos específicos que se han utilizado en el documento. La mayoría de ellos se han definido en el momento en el que aparecían en el texto, pero se ha preferido agruparlos todos. No se han tenido en cuenta los conceptos más básicos relacionados con la informática, pues se prevé que el lector ya los conoce.

Panel solar. Un panel solar es un dispositivo que genera energía gracias a la radiación que produce el Sol.

Central fotovoltaica. Una central fotovoltaica es un conjunto de paneles solares instalados para suministrar energía eléctrica.

Iteración. Una iteración en este trabajo se ha referido concretamente a las iteraciones propias de las metodologías ágiles. En cada iteración se desarrolla un bloque funcional de la aplicación en un periodo corto de tiempo.

***Shader*.** Un *shader* es un programa que permite realizar cálculos gráficos, interactuando directamente con la unidad de procesamiento gráfico del ordenador.

***Mockup*.** Un *mockup* en informática se utiliza para referirse a los diseños de las interfaces de una aplicación.