ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

APLICACIÓN PARA EL CÁLCULO DE INSTALACIONES SOLARES TÉRMICAS EN EDIFICIOS RESIDENCIALES

Francisco Javier Tirapu Francés

Pamplona, 7 de Febrero de 2011

Índice

Prólogo

Este proyecto fin de carrera, desarrollado en la empresa pública Natural Climate Systems S.A (MIYABI), se centra en el desarrollo de una aplicación para el cálculo de la contribución solar mínima exigida por el Código Técnico de Edificación (CTE) en edificios de nueva construcción.

El objetivo del proyecto es hacer una aplicación local enfocada a facilitar el trabajo de saber cuántos paneles solares y qué características deben tener para cubrir las exigencias normativas que desde 2007 afectan a todos los edificios de nueva construcción en nuestro país. Este trabajo se debe realizar durante el desarrollo del proyecto del edificio tanto para garantizar dicha cobertura solar mínima como para calcular la etiqueta de calificación energética del edificio. Actualmente existen algunas aplicaciones comerciales y gratuitas que realizan este trabajo, pero sin tener en cuenta las pérdidas por sombras que pueden afectar a los paneles solares dependiendo de su posición respecto a posibles obstáculos remotos, este trabajo actualmente se debe realizar aparte y calcular el resultado final. En nuestro caso queremos integrar toda esta funcionalidad en una aplicación independiente.

Por otro lado, nos urge disponer de módulos programados con esta funcionalidad debido a la futura aprobación del método simplificado de calificación energética “CES”, que incluirá un apartado donde debemos incorporar toda la funcionalidad de esta aplicación y que funcione integrada en el resto del sistema.

Capítulo 1: Introducción y Contexto

1.1 Arquitectura bioclimática

La arquitectura bioclimática consiste en el diseño de edificaciones teniendo en cuenta las condiciones climáticas, aprovechando los recursos disponibles (sol, vegetación, lluvia, vientos) para disminuir los impactos ambientales, intentando reducir los consumos de energía.

Una vivienda bioclimática puede conseguir un gran ahorro e incluso llegar a ser sostenible en su totalidad. Aunque el coste de construcción puede ser mayor, puede ser rentable, ya que el incremento de la vivienda se compensa con la disminución de los recibos de energía.

El hecho de que la construcción hoy en día no tenga en cuenta los aspectos bioclimáticos, se une al poco respeto por el ambiente que inunda a los países desarrollados y en vías de desarrollo, que no ponen los suficientes medios para frenar el desastre ecológico que dejamos a nuestro paso.

A pesar de que parece un concepto nuevo, se lleva utilizando tradicionalmente desde antiguo, un ejemplo de ello son las casas encaladas en Andalucía o los tejados orientados al sur en el hemisferio Norte, con objeto de aprovechar la inclinación del sol. También el

ejemplo de los chalets en los Alpes o las casas rurales en muchas partes del mundo. En estos dos tipos de vivienda señalados, el establo inferior servía de calefactor en invierno y se sacaban los animales en verano para pastar, sirviendo de aislamiento térmico. [9]

Adaptación a la temperatura

Es quizá en este punto donde es más común incidir cuando se habla de arquitectura bioclimática. Lo más habitual, es aprovechar al máximo la energía térmica del sol cuando el clima es frío, por ejemplo para calefacción y agua caliente sanitaria. Aprovechar el efecto invernadero de los cristales. Tener las mínimas pérdidas de calor (buen aislamiento térmico) si hay algún elemento calefactor.

Cuando el clima es cálido lo tradicional es hacer muros más anchos, y tener el tejado y la fachada de la casa con colores claros. Poner toldos y cristales especiales como doble cristal y tener buena ventilación son otras soluciones. En el caso de usar algún sistema de refrigeración, aislar la vivienda. Contar delante de una vivienda con un gran árbol de hoja caduca que tape el sol en verano y en invierno lo permita también sería una solución

Orientación

Con una orientación de los huecos acristalados al sur en el Hemisferio Norte, o al norte en el Hemisferio Sur, esto es, hacia el ecuador, se capta más radiación solar en invierno y menos en verano, aunque para las zonas más cálidas (con temperaturas promedio superiores a los 25°C) es sustancialmente más conveniente colocar los acristalamientos en el sentido opuesto, esto es, dándole la espalda al ecuador; de esta forma en el Verano, la cara acristalada sólo será irradiada por el Sol en los primeros instantes del alba y en los últimos momentos del ocaso, y en el Invierno el Sol nunca bañará esta fachada, reduciendo el flujo calorífico al mínimo y permitiendo utilizar conceptos de diseño arquitectónico propios del uso del cristal.

Soleamiento y protección solar

Las ventanas con una adecuada protección solar, alargadas en sentido vertical y situadas en la cara interior del muro, dejan entrar menos radiación solar en verano, evitando el sobrecalentamiento de locales soleados.

Por el contrario, este efecto es beneficioso en lugares fríos o durante el invierno, por eso, tradicionalmente, en lugares fríos las ventanas son más grandes que en los cálidos, están situadas en la cara exterior del muro y suelen tener miradores acristalados, para potenciar la beneficiosa captación de la radiación solar.

Aislamiento térmico

Los muros gruesos retardan las variaciones de temperatura, debido a su Inercia térmica.

Un buen aislamiento térmico evita, en el invierno, la pérdida de calor por su protección con el exterior, y en verano la entrada de calor.

Ventilación cruzada

La diferencia de temperatura y presión entre dos estancias con orientaciones opuestas, genera una corriente de aire que facilita la ventilación.

Una buena ventilación es muy útil en climas cálidos húmedos, sin refrigeración mecánica, para mantener un adecuado confort térmico.

Integración de energías renovables

Mediante la integración de fuentes de energía renovable, es posible que todo el consumo sea de generación propia y no contaminante. En este caso, hablamos de "edificios 0 emisiones". Puede llegarse incluso a generar más energía de la consumida -que podría ser vendida a la red-, en cuyo caso hablamos de "edificios energía plus".

Las fuentes más empleadas son la energía solar fotovoltaica, la energía solar térmica e incluso la energía geotérmica.

Como vemos el campo de la arquitectura bioclimática es muy extenso y tiene muchas ramas de aplicación, pero en definitiva su objetivo básico es el ahorro de energía en la edificación, construir edificios sostenibles que respeten el medio ambiente y que a largo plazo supongan un ahorro para el propietario, aunque esta última idea no siempre es posible.

1.2 El protocolo de Kioto

El Protocolo de Kioto sobre el cambio climático es un acuerdo internacional que tiene por objetivo reducir las emisiones de seis gases que causan el calentamiento global: dióxido de carbono (CO_2), gas metano (CH_4) y óxido nitroso (N_2O), además de tres gases industriales fluorados: Hidrofluorocarbonos (HFC), Perfluorocarbonos (PFC) y Hexafluoruro de azufre (SF_6), en un porcentaje aproximado de al menos un 5%, dentro del periodo que va desde el

año 2008 al 2012, en comparación a las emisiones al año 1990. Por ejemplo, si la contaminación de estos gases en el año 1990 alcanzaba el 100%, al término del año 2012 deberá ser al menos del 95%. Es preciso señalar que esto no significa que cada país deba reducir sus emisiones de gases regulados en un 5% como mínimo, sino que este es un porcentaje a nivel global y, por el contrario, cada país obligado por Kioto tiene sus propios porcentajes de emisión que debe disminuir.

El protocolo fue inicialmente adoptado el 11 de diciembre de 1997 en Kioto, Japón pero no entró en vigor hasta el 16 de febrero de 2005. En noviembre de 2009, eran 187 estados los que ratificaron el protocolo. EEUU mayor emisor de gases de invernadero mundial no ha ratificado el protocolo.

El instrumento se encuentra dentro del marco de la Convención Marco de las Naciones Unidas sobre el Cambio Climático (CMNUCC), suscrita en 1992 dentro de lo que se conoció como la Cumbre de la Tierra de Río de Janeiro. El protocolo vino a dar fuerza vinculante a lo que en ese entonces no pudo hacer la CMNUCC [9]

1.3 Calificación energética de edificios

Ligado al concepto de arquitectura bioclimática, podemos decir que un edificio eficiente es aquel que minimiza el uso de las energías convencionales (en particular la energía no renovable), a fin de ahorrar y hacer un uso racional de la misma. En los años 70 cuando ocurrió la primera gran crisis del petróleo la mayoría de los países desarrollados establecieron un control de la eficiencia energética edilicia, en particular Suecia, Alemania, Inglaterra y Francia. Estos además implementaron políticas activas para el ahorro de energía en edificios. Casi el 30 % del consumo de energía primaria es debido a los edificios, y por ello las normativas europeas han intentado incidir sobre el consumo energético de las construcciones, en este caso creando una herramienta similar a la ya empleada en el caso de los electrodomésticos. Cada edificio tiene su propia etiqueta de calificación energética y en función de las emisiones de CO2 anuales respecto a una serie de factores (dimensiones, tipología de edificio, zona climática en la que se encuentra…) se le da una calificación en la escala de las letras A a la G, siendo la A la calificación más alta posible, donde sin duda estaríamos hablando de un edificio de alta eficiencia energética.

La Directiva Europea 2002/91/CE tiene como objetivo fomentar la Eficiencia Energética de los Edificios y obliga a todos los estados miembro, entre otras cosas, a que todo edificio, tanto si se vende como si se alquila, vaya acompañado de un Certificado de Eficiencia Energética. Este certificado se presentaría a la persona interesada, el propietario o inquilino. Esta directiva en el estado español no se ha legislado hasta el año 2007, mediante la aprobación del Código Técnico de la Edificación (CTE). [11]

Herramientas de Calificación Energética

En España existe un programa informático denominado “Calener” que es el método general acreditado por IDAE (Instituto para el Desarrollo y Ahorro de Energía) para obtener la calificación energética de cualquier edificio. Para ello debemos introducir en el programa todos los datos necesarios del edificio en cuestión. El problema que plantea la calificación con “Calener” es que es un proceso lento y pesado por lo que se están desarrollando métodos alternativos simplificados que de una manera mucho más sencilla y rápida sea posible obtener una calificación energética, eso sí en ningún caso podrá ser tan exacta y precisa como con el método general. El desarrollo de estos métodos simplificados salió a concurso público, y MIYABI fue adjudicataria de dicho concurso con su proyecto “CES, calificación energética simplificada.”

Este proyecto fin de carrera planteado como una aplicación independiente es el comienzo de un módulo que estará disponible en CES tanto para el cálculo de la contribución solar mínima como para el cálculo de las sombras que afectan al edificio.

1.4 El Código Técnico de Edificación (CTE)

El Código Técnico de la Edificación (CTE) [13] es el marco normativo que establece las exigencias que deben cumplir los edificios en relación con los requisitos básicos de seguridad y habitabilidad establecidos en la Ley 38/1999 de 5 de noviembre, de Ordenación de la Edificación (LOE).

Las Exigencias Básicas de calidad que deben cumplir los edificios se refieren a materias de seguridad: seguridad estructural, seguridad contra incendios, seguridad de utilización; y habitabilidad: salubridad, protección frente al ruido y ahorro de energía.

El CTE también se ocupa de la accesibilidad como consecuencia de la Ley 51/2003 de 2 de diciembre, de igualdad de oportunidades, no discriminación y accesibilidad universal de las personas con discapacidad, LIONDAU.

El CTE pretende dar respuesta a la demanda de la sociedad en cuanto a la mejora de la calidad de la edificación a la vez que persigue mejorar la protección del usuario y fomentar el desarrollo sostenible. El CTE se aplica a edificios de nueva construcción, a obras de ampliación, modificación, reforma o rehabilitación y a determinadas construcciones protegidas desde el punto de vista ambiental, histórico o artístico.

Respecto al desarrollo sostenible que es el tema que nos ocupa en este proyecto, dentro del CTE encontramos el “Documento Básico HE” que tiene por objeto establecer reglas y procedimientos que permiten cumplir las exigencias básicas de ahorro de energía:

1. El objetivo del requisito básico "Ahorro de energía" consiste en conseguir un uso racional de la energía necesaria para la utilización de los *edificios*, reduciendo a límites sostenibles su consumo y conseguir asimismo que una parte de este consumo proceda de fuentes de energía renovable, como consecuencia de las características de su *proyecto, construcción, uso* y *mantenimiento.*

2. Para satisfacer este objetivo, los *edificios* se proyectarán, construirán, utilizarán y mantendrán de forma que se cumplan las exigencias básicas que se establecen en los apartados siguientes.

3. El Documento Básico "DB HE Ahorro de energía" especifica parámetros objetivos y procedimientos cuyo cumplimiento asegura la satisfacción de las exigencias básicas y la superación de los niveles mínimos de calidad propios del requisito básico de ahorro de energía.

Para normalizar todos estos conceptos el "Documento Básico HE" se compone de cinco capítulos en cada uno de los cuales se definen las normas básicas que deben cumplir los nuevos edificios:

Exigencia básica HE 1: Limitación de demanda energética
Exigencia básica HE 2: Rendimiento de las instalaciones térmicas
Exigencia básica HE 3: Eficiencia energética de las instalaciones de iluminación
Exigencia básica HE 4: Contribución solar mínima de agua caliente sanitaria
Exigencia básica HE 5: Contribución fotovoltaica mínima de energía eléctrica

1.5 Exigencia básica HE 4

Y ahora sí ya hemos llegado exactamente al enclave de este proyecto, la exigencia básica HE 4, del documento básico HE del código técnico de edificación y que se trata de la "Contribución solar mínima de agua caliente sanitaria".
En este documento se recoge toda la normativa a seguir en los *edificios*, con previsión de demanda de agua caliente sanitaria, una parte de las necesidades energéticas térmicas derivadas de esa demanda se cubrirá mediante la incorporación en los mismos de sistemas de captación, almacenamiento y utilización de energía solar de baja temperatura, adecuada a la radiación solar global de su emplazamiento y a la demanda de agua caliente del edificio. Los valores derivados de esta exigencia básica tendrán la consideración de mínimos, sin perjuicio de valores que puedan ser establecidos por las administraciones competentes y que contribuyan a la sostenibilidad, atendiendo a las características propias de su localización y ámbito territorial.

Contribución Solar Mínima Anual: es la fracción entre los valores anuales de la energía solar aportada exigida y la demanda energética anual, obtenidos a partir de los valores mensuales.

Zona Climática: Se definen en España 5 zonas climáticas en función de la Radiación Solar Global media diaria anual sobre superficie horizontal.

En la tabla se aprecian los valores de radiación para cada zona climática:

Tabla 3.2 Radiación solar global

Zona climática	MJ/m^2	kWh/m^2
I	H < 13,7	H < 3,8
II	13,7 ≤ H < 15,1	3,8 ≤ H <4,2
III	15,1 ≤ H < 16,6	4,2 ≤ H < 4,6
IV	16,6 ≤ H < 18,0	4,6 ≤ H < 5,0
V	H ≥ 18,0	H ≥ 5,0

En el siguiente mapa apreciamos como quedan las zonas climáticas distribuidas geográficamente:

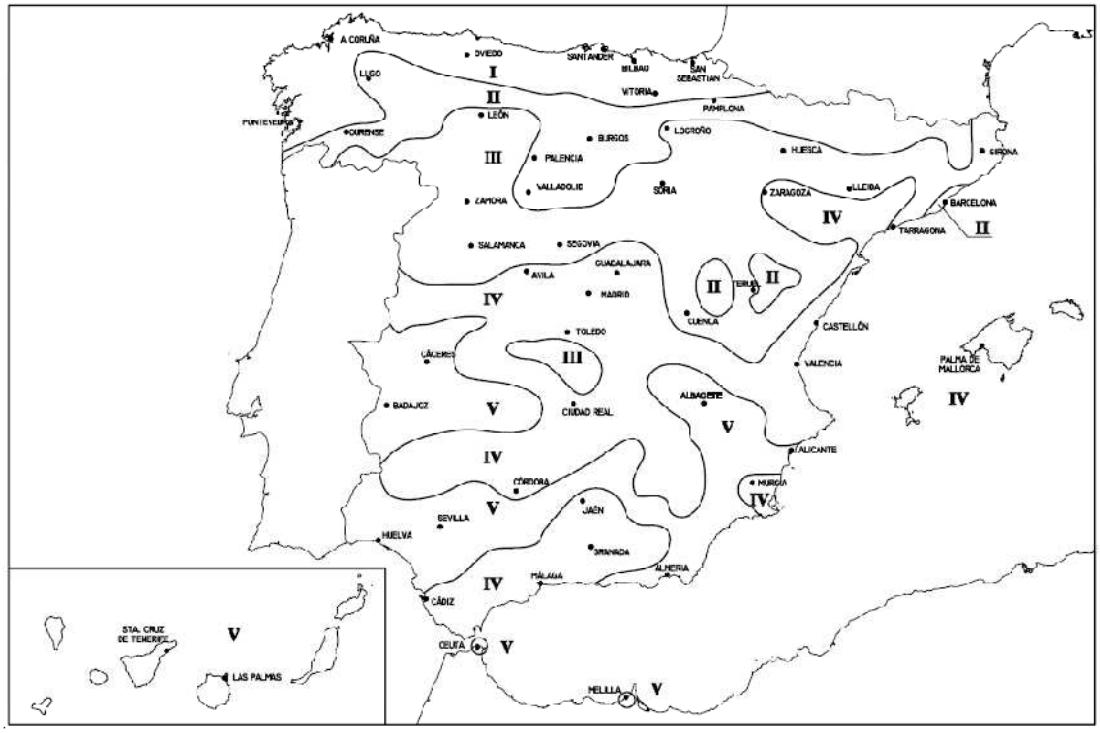


Fig. 3.1. Zonas climáticas

Para saber el porcentaje de contribución solar mínima de agua caliente sanitaria que debe tener un nuevo edificio nos valdremos de la siguiente tabla:

Tabla 2.1. Contribución solar mínima en %. Caso general

Demanda total de ACS del edificio (l/d)	Zona climática I	II	III	IV	V
50-5.000	30	30	50	60	70
5.000-6.000	30	30	55	65	70
6.000-7.000	30	35	61	70	70
7.000-8.000	30	45	63	70	70
8.000-9.000	30	52	65	70	70
9.000-10.000	30	55	70	70	70
10.000-12.500	30	65	70	70	70
12.500-15.000	30	70	70	70	70
15.000-17.500	35	70	70	70	70
17.500-20.000	45	70	70	70	70
> 20.000	52	70	70	70	70

Al final del presente proyecto se adjunta este documento “Exigencia Básica HE4”, ya que es el enclave a seguir para todo el desarrollo.

1.6 El cálculo F-chart

Para el dimensionado de las instalaciones de energía solar térmica se sugiere el método de las curvas *f* (F-Chart), que permite realizar el cálculo de la cobertura de un sistema solar, es decir, de su contribución a la aportación de calor total necesario para cubrir las cargas térmicas, y de su rendimiento medio en un largo período de tiempo.

Ampliamente aceptado como un proceso de cálculo suficientemente exacto para largas estimaciones, no ha de aplicarse para estimaciones de tipo semanal o diario. Para desarrollarlo se utilizan datos mensuales medios meteorológicos, y es perfectamente válido para determinar el rendimiento o factor de cobertura solar en instalaciones de calentamiento, en todo tipo de edificios, mediante captadores solares planos, ya que dicho cálculo es anual y como hemos dicho el método es aceptado para estimaciones largas en el tiempo.

Su aplicación sistemática consiste en identificar las variables adimensionales del sistema de calentamiento solar y utilizar la simulación de funcionamiento mediante ordenador, para dimensionar las correlaciones entre estas variables y el rendimiento medio del sistema para un dilatado período de tiempo.[12]

Capítulo 2: Requisitos

Durante el desarrollo de la aplicación se ha procurado tener en cuenta una serie de requisitos que se consideraban fundamentales para que el resultado permitiera una utilización correcta y eficaz de la misma por parte de los usuarios finales. A lo largo de este capítulo iremos presentando dichos requisitos que han sido tenidos en cuenta durante todo el proceso, tanto en la fase de análisis como de diseño de este proyecto. A continuación se presentan todos los requisitos ordenados según su naturaleza.

2.1. Requisitos Hardware

Los requisitos de hardware vienen determinados esencialmente por el entorno donde se desarrolla la aplicación.

1. Deseamos que se trata de de una aplicación “Desktop Aislada” y que sea portable de manera que pueda ser utilizada en cualquier tipo de plataforma. Por ello, se programará mediante un lenguaje que nos permita cumplir con este requisito y que sea gratuito, Y además la naturaleza de esta aplicación nos exige la utilización de funciones complejas de cálculo Se ha decidido optar por el lenguaje de cuarta generación “Python”, que dispone de los módulos necesarios para desarrollar el interfaz gráfico de usuario y las funciones de cálculo necesarias. Dentro de los editores gratuitos para este lenguaje hemos decidido utilizar “Boa constructor”.
2. La aplicación está pensada para ser utilizada por el mayor número de usuarios posible, dentro del campo en el que se ubica, y dado que el sistema operativo más extendido es Windows se ha optado por el mismo.

2.2. Requisitos Software

Los requisitos software vienen determinados principalmente por el tipo de aplicación a realizar. Son aquellos requisitos, tanto funcionales como no funcionales que la aplicación debe cumplir, para que todos los requerimientos sean implementados correctamente.

2.2.1 Requisitos de apariencia GUI’s

1. Las opciones disponibles para el usuario se mostrarán en todo momento de una manera clara.

2. Si alguna de las opciones en determinado momento no debe ser seleccionada ésta será desactivada para imposibilitar que el usuario la seleccione.

3. Los resultados que debe generar la aplicación será mostrados al usuario de manera clara de forma que éste pueda entenderlos e interpretarlos correctamente

4. La aplicación se apoyará en la muestra de mensajes para dar a conocer al, usuario situaciones de error o informarle de determinados eventos que se puedan producir durante la utilización de la misma.

Con los requisitos anteriormente definidos lo que se pretende es conseguir que la aplicación final sea intuitiva de modo que el usuario final, solo con los conocimientos básicos pueda utilizarla correctamente.

2.2.2. Requisitos Funcionales

2.2.2.1 Requisitos funcionales generales

1. El interfaz gráfico se compondrá de una ventana en la que el usuario verá disponible una barra de menú y dos pestañas

2. La barra de menú se compondrá un de dos elementos: “Archivo” y “Calcular”. En “Archivo” el usuario encontrará las opciones necesarias para crear un nuevo proyecto, guardar el proyecto actual, abrir proyecto o salir de la aplicación. En “Calcular” el usuario encontrará las opciones de cálculo y comprobación del CTE.

3. En caso de que el usuario intente calcular la cobertura solar y no haya introducido todas las características fundamentales del edificio y su instalación solar térmica se le informará mediante un mensaje.

4. Todos los campos del interfaz deben ser cumplimentados con el tipo de dato adecuado, en su defecto se informará al usuario con un mensaje indicando los campos incorrectos.

5. Hasta que todos los datos no hayan sido cumplimentados correctamente, la aplicación no devolverá ningún resultado.

2.2.2.2 Requisitos funcionales de la entrada del edificio

1. En esta pestaña el usuario encuentra los campos que debe rellenar para definir el edificio en estudio (tipología, localización, superficie…)

2. Al introducir la provincia donde se sitúa el edificio, la aplicación cargará por defecto los valores mensuales de "radiación solar por superficie horizontal" y "temperatura media del agua de la red" además de la latitud media, dejando que posteriormente el usuario los edite manualmente.

3. Una vez el usuario edita los campos provincia, tipología y superficie el programa calcula el consumo de agua y energético anual medio. Dejando también que el usuario lo edite manualmente.

4. El campo "Albedo" (tipo de superficie sobre la que se encuentra el edificio) lo puede editar el usuario o escoger el valor por defecto de cada tipo de superficie disponible.

5. Todos los campos de esta pestaña son obligatorios para realizar los cálculos. En caso de faltar alguno e intentar realizar el cálculo se mostrará el correspondiente mensaje.

2.2.2.3 Requisitos funcionales de la entrada de los captadores solares

1. En esta pestaña el usuario encuentra los campos que debe rellenar para definir la instalación solar térmica del edificio (tipo de colector, número de colectores, colocación…)

2. El campo tipo de colector nos ofrecerá una serie de modelos comerciales disponibles en el mercado.

3. Al seleccionar el tipo de colector se rellenarán automáticamente las características del equipo, sin dejar que el usuario las edite posteriormente.

4. El campo orientación debe estar en el rango (-180,180). Ya que se mide en grados y se considera que la orientación sur corresponde con 0 grados. Conforme el usuario edita el campo se muestra un mensaje de error si el valor introducido no se encuentra en el rango establecido.

5. El campo inclinación debe estar en el rango (0, 90). Ya que se mide en grados y se considera que los 0 grados corresponden el plano horizontal y 90º al plano vertical.

6. Al introducir los datos de orientación e inclinación el programa calcula automáticamente las pérdidas asociadas a los valores introducidos.

7. Las pérdidas por sombras pueden ser introducidas manualmente por el usuario o ser calculadas por el módulo de obstáculos remotos que estará disponible mediante un botón.

2.2.2.4 Requisitos funcionales de la entrada de obstáculos remotos

1. Para acceder a la definición de obstáculos remotos será necesario introducir los datos de orientación e inclinación de la instalación solar térmica. En caso contrario se informará con un mensaje.

2. La nueva ventana dispondrá del gráfico de trayectoria solar, sobre el que se irán introduciendo los obstáculos que sombrean el edificio por coordenadas de “Acimut” y “Elevación”.

3. Al introducir un nuevo obstáculo, el usuario verá la repercusión en el gráfico con un dibujo.

4. Debido a la complejidad del cálculo de las coordenadas de acimut y elevación, se incorporará una ayuda para introducir obstáculos paralelos al edificio.

5. En el diálogo de ayuda el usuario debe introducir las distancias del obstáculo respecto al edificio y en función de ellas se calcularán automáticamente las coordenadas reales de acimut y elevación.

6. Una vez el usuario haya definido todos los obstáculos remotos del edificio, presionando el botón aceptar, se calcularán las pérdidas por sombras asociadas a dichos obstáculos y se rellenará la casilla correspondiente en la pestaña de definición de los captadores solares.

2.3 Requisitos técnicos

El proyecto que se va a desarrollar implementa una serie de cálculos que se definirán en los próximos capítulos. Podemos adelantar que estos cálculos matemáticos no son excesivamente complejos y además están perfectamente definidos. La dificultad radica en la comprensión de los métodos de representación necesarios, ya que sin este estudio previo, sería completamente imposible el desarrollo de la aplicación.

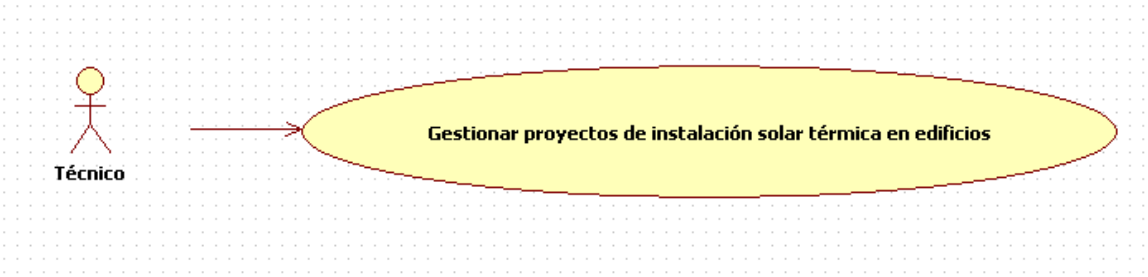
2.4 Conclusiones

Todos estos requisitos iniciales que se han desarrollado a lo largo de este capítulo suponen el punto de partida para empezar a trabajar en este proyecto. Respetar todos los requisitos en el análisis para la posterior implementación es la siguiente tarea que debemos realizar.

Capítulo 3: Análisis de requisitos

A lo largo de este capítulo mostraremos el análisis de requisitos orientado a objetos a partir de los requisitos especificados en el capítulo anterior, cuya finalidad es la de permitirnos diseñar e implementar el producto deseado de manera correcta. Para ello nos serviremos del modelo de casos de uso que podemos encontrar dentro de este proyecto. Mediante estos diagramas lo que pretendemos es especificar la funcionalidad y el comportamiento del sistema mediante su interacción con los usuarios, es decir, mostraremos las relaciones que se establecen entre los actores y los casos de uso del sistema.

3.1 Caso de Uso General (Interacción 0)



Actor Principal: Técnico instalador.

Actores Secundarios: No hay

Personal involucrado e intereses: El técnico que desea gestionar los proyectos de instalación solar térmica que todo edificio nuevo debe poseer para cumplir el mínimo normativo exigido por el CTE.

Precondiciones: El técnico ha debido realizar correctamente la medición del edificio para poder introducir en la aplicación todos los datos necesarios para realizar el cálculo de la cobertura solar.

Poscondiciones: el sistema devuelve si la instalación elegida por el técnico cumple o no el mínimo normativo, permitiendo salvar la información y modificación constante.

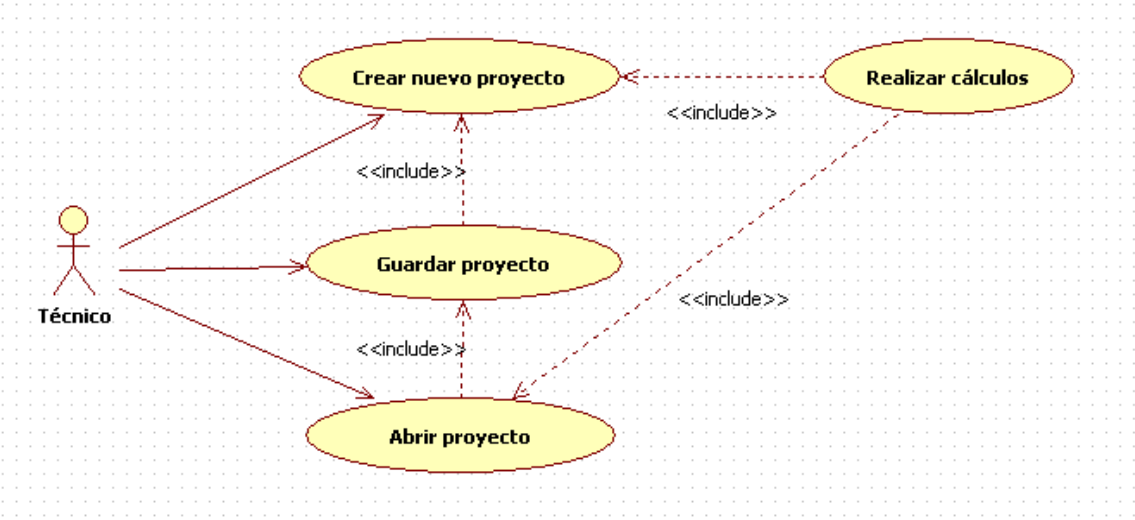
Flujo Básico:

1. El técnico arranca el sistema.
2. El técnico introduce todos los datos necesarios.
3. El sistema devuelve resultados.
4. El técnico salva el proyecto.

Flujos Alternativos:

1. Tratar de obtener resultados sin completar correctamente todos los datos de entrada

3.2 Uso de la herramienta para la gestión de proyectos



Actor Principal: Técnico instalador.

Actores Secundarios: No hay

Personal involucrado e intereses: El técnico que desea gestionar los proyectos de instalación solar térmica que todo edificio nuevo debe poseer para cumplir el mínimo normativo exigido por el CTE.

Precondiciones: El técnico ha debido realizar correctamente la medición del edificio para poder introducir en la aplicación todos los datos necesarios para realizar el cálculo de la cobertura solar en caso de que quiera crear un proyecto nuevo. Si ya había guardado uno, podrá abrirlo para recuperar la información guardada.

Poscondiciones: Si empieza a crear un nuevo proyecto en cualquier momento podrá guardarlo. Si abre uno anterior también podrá guardar los nuevos datos que. En cualquier caso hasta que todos los campos no estén correctamente completados no podrá obtener resultados.

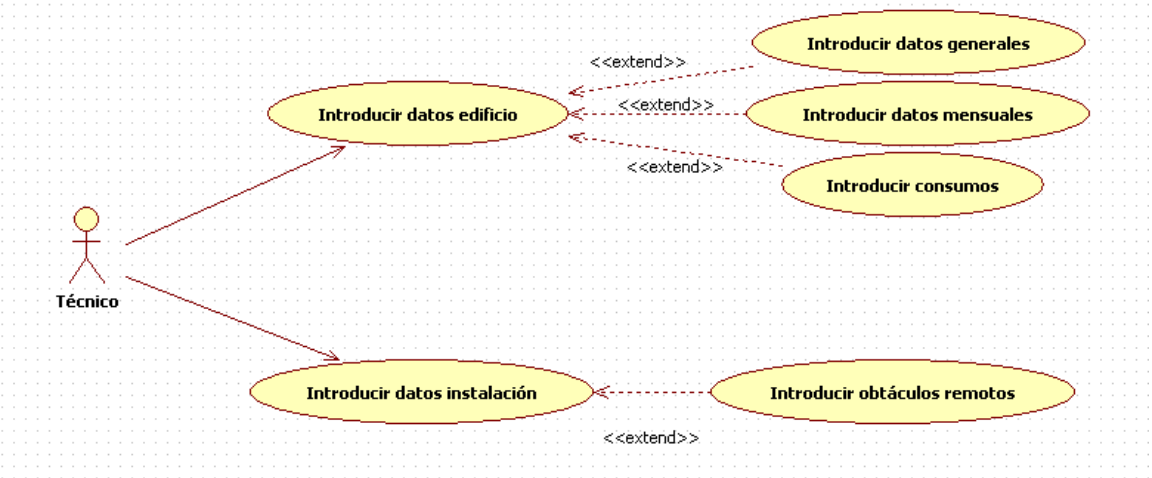
Flujo Básico:

1. El técnico arranca el sistema.
2. El técnico decide si comienza un nuevo proyecto o recupera una anterior.
3. El técnico introduce todos los datos necesarios.
4. El sistema devuelve resultados.
5. El técnico guarda los resultados obtenidos.

Flujos Alternativos:

1. Tratar de obtener resultados sin completar correctamente todos los datos de entrada.
2. Tratar de abrir un proyecto dañado, defectuoso o con extensión de archivo distinta a la necesaria...

3.3 Uso de la herramienta para crear un proyecto nuevo



Actor Principal: Técnico instalador.

Actores Secundarios: No hay

Personal involucrado e intereses: El técnico que desea crear un nuevo proyecto de instalación solar térmica que todo edificio nuevo debe poseer para cumplir el mínimo normativo exigido por el CTE.

Precondiciones: El técnico ha debido realizar correctamente la medición del edificio para poder introducir en la aplicación todos los datos necesarios para realizar el cálculo de la cobertura solar. Para ello debe introducir datos sobre el edificio y la instalación en estudio.

Poscondiciones: En función de los datos introducidos, la aplicación auto completa algunos campos, en algunos casos deja que el usuario edite los campos auto completados ya que se estima que puede disponer de información más precisa de la que aporta el sistema.

Flujo Básico: La aplicación está diseñada para que el usuario complete los campos que se le presentan en orden de aparición.

1. Introducción de los datos del edificio.

   Introducción de datos generales. Al completar dichos datos se completan automáticamente los datos mensuales y de consumos.

   Edición si es necesario de datos mensuales.

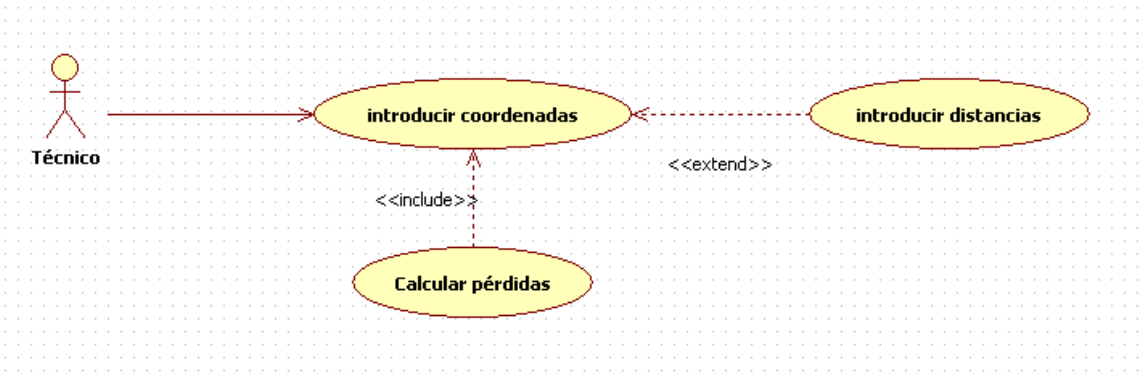
   Edición si es necesario de datos de consumos.

2. Introducción de los datos de la instalación
3. Introducción de obstáculos remotos.

Flujos Alternativos:

1. Introducir los campos de datos mensuales o de consumos antes que los generales del edificio.
2. Introducir datos de obstáculos remotos antes que los de la instalación.

3.4 Uso de la herramienta para introducir obstáculos remotos



Actor Principal: Técnico instalador.

Actores Secundarios: No hay

Personal involucrado e intereses: El técnico que desea medir las pérdidas por sombras que afectan a la instalación solar térmica del edificio. Aunque el sistema permite introducir directamente el dato de pérdidas, se recomienda utilizar el método implementado por la exactitud del mismo.

Precondiciones: El técnico ha debido realizar correctamente la medición de los obstáculos remotos que se pueden interponer en la trayectoria solar respecto a los paneles solares de la instalación. Además debe comprender la técnica de medición necesaria.

Poscondiciones: En función de los datos introducidos, la aplicación calcula las pérdidas por sombras anuales de la instalación. Este valor afecta tanto al rendimiento de la instalación como a los requisitos normativos exigidos por el CTE.

Flujo Básico:

1. Introducción de los obstáculos remotos
2. Guardar el patrón de sombras
3. Calcular las pérdidas anuales por sombras.

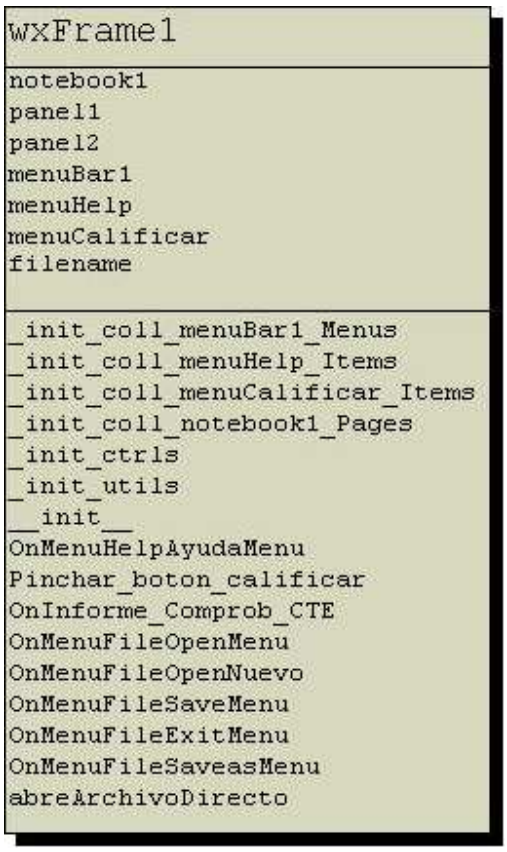
Capítulo 4: Clases

A lo largo de este capítulo se presentan los esquemas de cada una de las clases implementadas para que construyan nuestra aplicación. Se presenta la información esquematizada de cada clase así como una breve explicación de cada Atributo y método de las mismas. Por último aparece el diagrama general de clases de la aplicación.

4.1 Clases: Atributos y métodos

Clase wxFrame1 del fichero ventanaFchart.py

Clase que implementa la ventana principal del interfaz gráfico de la aplicación.



Atributo **notebook1**: instancia del "notebook" que compone la ventana.

Atributo **panel1**: instancia del primer panel del notebook que implementa la recogida de datos del edificio.

Atributo **panel2**: instancia del segundo panel del notebook que implementa la recogida de datos de la instalación.

Atributo **menuBar1**: instancia de la barra de menú de la aplicación.

Atributo **menuHelp**: instancia del elemento de menú "Archivo".

Atributo **menuCalificar**: instancia del elemento de menú "Resultados".

Atributo **filename**: cadena de texto que contiene el nombre del fichero actualmente en uso.

Método **_init_coll_menuBar1_Menus**(self, parent): inicia los elementos de la barra de menú.

Método **_init_coll_menuHelp_Items**(self, parent): inicia los elementos del menú "Archivo".

Método **_init_coll_menuCalificar_Items**(self, parent): inicia los elementos del menú "Resultados".

Método **_init_coll_notebook1_Pages**(self, parent): inicia las pestañas del "notebook"

Método **_init_ctrls**(self, prnt): inicia los "widgets" del interfaz.

Método **_init_utils**(self): inicia los menús de la aplicación.

Método **__init__(**self, parent): constructor de la clase.

Método **OnMenuHelpAyudaMenu**(self, event): manejador de evento de objeto de menú.

Método **Pinchar_boton_calificar**(self, parent): manejador de evento de objeto de menú.

Método **OnGenerarInforme**(self, parent) : manejador de evento de objeto de menú.

Método **OnInforme_Comprob_CTE**(self, parent) : manejador de evento de objeto de menú.

Método **OnMenuFileOpenMenu**(self, event) : manejador de evento de objeto de menú.

Método **OnMenuFileOpenNuevo**(self, event) : manejador de evento de objeto de menú.

Método **OnMenuFileSaveMenu**(self, event) : manejador de evento de objeto de menú.

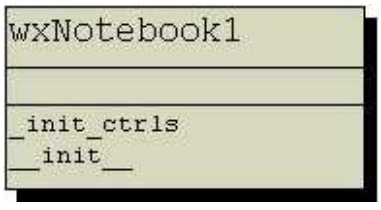
Método **OnMenuFileExitMenu**(self, event) : manejador de evento de objeto de menú.

Método **OnMenuFileSaveasMenu**(self, event) : manejador de evento de objeto de menú.

Método **abreArchivoDirecto**(self, filename) : manejador de evento de objeto de menú.

Clase wxNotebook1 del fichero notebookFchart.py

Clase que implementa el "notebook" del interfaz gráfico de la aplicación.

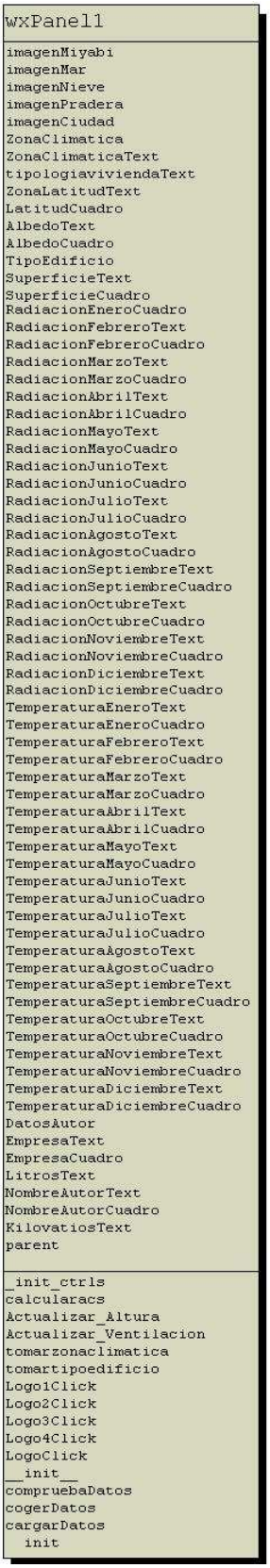


Método **_init_ctrls**(self, prnt): inicia elementos del "notebook"

Método **__init__(**self, parent): constructor de la clase

Clase wxPanel1 del fichero panel1Fchart.py

Clase que implementa el panel del interfaz gráfico de recogida de datos del edificio.



Atributo **imagenMiyabi**: imagen del símbolo de la aplicación.

Atributo **imagenMar**: imagen para selección del “Albedo”.

Atributo **imagenNieve**: imagen para selección del “Albedo”.

Atributo **imagenPradera**: imagen para selección del “Albedo”.

Atributo **imagenCiudad**: imagen para selección del “Albedo”.

Atributo **ZonaClimatica**: “choice” para introducir la provincia.

Atributo **ZonaClimaticaText**: “staticText” de provincia.

Atributo **tipologiaviviendaText**: “staticText” de tipología.

Atributo **ZonaLatitudText**: “staticText” de latitud.

Atributo **LatitudCuadro**: “textCtrl” para introducir el dato de latitud.

Atributo **AlbedoText**: “staticText” de Albedo.

Atributo **AlbedoCuadro**: “textCtrl” para introducir el dato del albedo.

Atributo **TipoEdificio**: “choice” para elegir el tipo de edificio.

Atributo **SuperficieText**: “staticText” de superficie.

Atributo **SuperficieCuadro**: “textCtrl” para introducir el dato de superficie.

Atributo **RadiacionFebreroText**: “staticText” de radiación para Enero.

Atributo **RadiacionEneroCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Enero.

Atributo **RadiacionFebreroText**: “staticText” de radiación para Febrero.

Atributo **RadiacionFebreroCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Febrero.

Atributo **RadiacionMarzoText**: “staticText” de radiación para Marzo.

Atributo **RadiacionMarzoCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Marzo.

Atributo **RadiacionAbrilText**: “staticText” de radiación para Abril.

Atributo **RadiacionAbrilCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Abril.

Atributo **RadiacionMayoText**: “staticText” de radiación para Mayo.

Atributo **RadiacionMayoCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Mayo.

Atributo **RadiacionJunioText**: “staticText” de radiación para Junio.

Atributo **RadiacionJunioCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Junio.

Atributo **RadiacionJulioText**: “staticText” de radiación para Julio.

Atributo **RadiacionJulioCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Julio.

Atributo **RadiacionAgostoText**: “staticText” de radiación para Agosto.

Atributo **RadiacionAgostoCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Agosto.

Atributo **RadiacionSeptiembreText**: “staticText” de radiación para Septiembre.

Atributo **RadiacionSeptiembreCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Septiembre.

Atributo **RadiacionOctubreText**: “staticText” de radiación para Octubre.

Atributo **RadiacionOctubreCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Octubre.

Atributo **RadiacionNoviembreText**: “staticText” de radiación para Noviembre.

Atributo **RadiacionNoviembreCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Noviembre,

Atributo **RadiacionDiciembreText**: “staticText” de radiación para Diciembre.

Atributo **RadiacionDiciembreCuadro**: “textCtrl” para introducir el dato de radiación para el mes de Diciembre.

Atributo **TemperaturaEneroText**: “staticText” de temperatura media para Enero.

Atributo **TemperaturaEneroCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Enero.

Atributo **TemperaturaFebreroText**: “staticText” de temperatura media para Febrero.

Atributo **TemperaturaFebreroCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Febrero.

Atributo **TemperaturaMarzoText**: “staticText” de temperatura media para Marzo.

Atributo **TemperaturaMarzoCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Marzo.

Atributo **TemperaturaAbrilText**: “staticText” de temperatura media para Abril.

Atributo **TemperaturaAbrilCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Abril.

Atributo **TemperaturaMayoText**: “staticText” de temperatura media para Mayo.

Atributo **TemperaturaMayoCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Mayo.

Atributo **TemperaturaJunioText**: “staticText” de temperatura media para Junio.

Atributo **TemperaturaJunioCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Junio.

Atributo **TemperaturaJulioText**: “staticText” de temperatura media para Julio.

Atributo **TemperaturaJulioCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Julio.

Atributo **TemperaturaAgostoText**: “staticText” de temperatura media para Agosto.

Atributo **TemperaturaAgostoCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Agosto.

Atributo **TemperaturaSeptiembreText**: “staticText” de temperatura media para Septiembre.

Atributo **TemperaturaSeptiembreCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Septiembre.

Atributo **TemperaturaOctubreText**: “staticText” de temperatura media para Octubre.

Atributo **TemperaturaOctubreCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Octubre.

Atributo **TemperaturaNoviembreText**: “staticText” de temperatura media para Noviembre.

Atributo **TemperaturaNoviembreCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Noviembre.

Atributo **TemperaturaDiciembreText**: “staticText” de temperatura media para Diciembre.

Atributo **TemperaturaDiciembreCuadro**: “textCtrl” para introducir el dato de temperatura media del agua de la red para el mes de Diciembre.

Atributo **AguaText**: “staticText” para el consumo de agua.

Atributo **AguaCuadro**: “textCtrl” para introducir el dato de consumo de agua.

Atributo **LitrosText**: “staticText” para los litros.

Atributo **EnergiaText**: “staticText” para el consumo de energía.

Atributo **EnergiaCuadro**: “textCtrl” para introducir el dato de consumo de energía.

Atributo **KilovatiosText**: “staticText” para los kilovatios hora año.

Atributo **parent**: instancia de la ventana de la aplicación.

Método **_init_ctrls(**self, prnt): inicia los “widgets” del interfaz.

Método **calcularacs** (self,event): manejador del evento de introducir texto en ”superficieCuadro ”.

Método **tomarzonaclimatica**(self, event): manejador del evento de elegir en “ZonaClimatica”.

Método **tomartipoedificio**(self, event): manejador del evento de elegir en “TipoEdificio”.

Método **Logo1Click**(self, event) : manejador del evento de hacer click sobre “imagenMar”.

Método **Logo2Click**(self, event): manejador del evento de hacer click sobre “imagenPradera”.

Método **Logo3Click**(self, event): manejador del evento de hacer click sobre "imagenNieve".

Método **Logo4Click**(self, event) : manejador del evento de hacer click sobre "imagenCiudad".

Método compruebaDatos(self): comprueba los datos que se introducen en el panel.

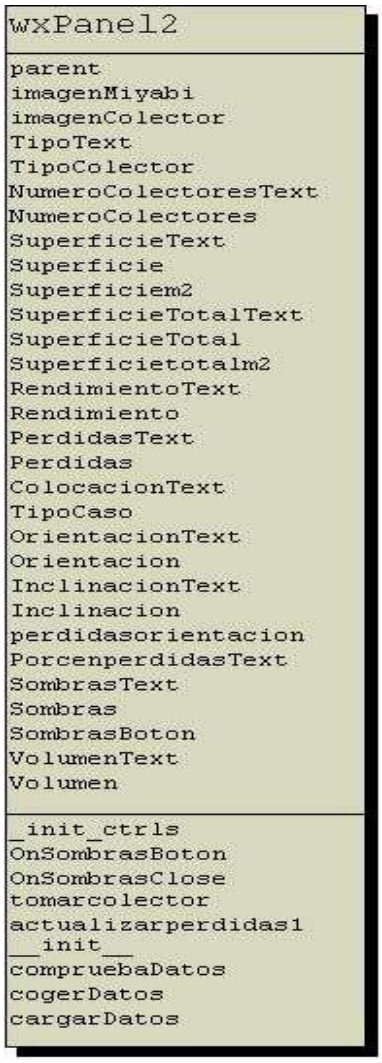
Método **cogerDatos**(self): devuelve los datos introducidos en el panel.

Método **cargarDatos**(self, datos): inicia los campos del panel.

Método **__init__**(self, parent, id, pos, size, style, name): constructor de la clase.

Clase wxPanel2 del fichero panel2Fchart.py

Clase que implementa el panel del interfaz gráfico de recogida de datos de la instalación del edificio.



Atributo **parent**: instancia de la ventana de la aplicación.

Atributo **imagenMiyabi**: imagen del símbolo de la aplicación.

Atributo **imagenColector**: imagen del colector.

Atributo **TipoText**: "staticText" de tipo de colector.

Atributo **TipoColector**: "choice" para elegir el colector.

Atributo **NumeroColectoresText**: "staticText" de nº de colectores.

Atributo **NumeroColectores**: "textCtrl" para introducir el dato de número de colectores.

Atributo **SuperficieText**: "staticText" de superficie unitaria.

Atributo **Superficie**: "textCtrl" para introducir el dato de superficie unitaria.

Atributo **Superficiem2**: "staticText" de unidades m2

Atributo **SuperficieTotalText**: "staticText" de superficie total.

Atributo **SuperficieTotal**: “textCtrl” para introducir el dato de superficie total.

Atributo **Superficietotalm2**: “staticText” de unidades m2.

Atributo **RendimientoText**: “staticText” de rendimiento óptico.

Atributo **Rendimiento**: “textCtrl” para introducir el dato del rendimiento óptico del colector.

Atributo **PerdidasText**: “staticText” delcoeficiente de pérdidas.

Atributo **Perdidas**: “textCtrl” para introducir el dato del coeficiente de pérdidas.

Atributo **ColocacionText**: “staticText” de colocación.

Atributo **TipoCaso**: “choice” para elegir el tipo de colocación.

Atributo **OrientacionText**: “staticText” de orientación.

Atributo **Orientacion**: “textCtrl” para introducir el dato de orientación.

Atributo **InclinacionText**: “staticText” de inclinación.

Atributo **Inclinacion**: “textCtrl” para introducir el dato de inclinación.

Atributo **perdidasorientacion**: “textCtrl” para introducir el dato de % pérdidas.

Atributo **PorcenperdidasText**: “staticText” de % pérdidas.

Atributo **SombrasText**: “staticText” de pérdidas por sombras.

Atributo **Sombras**: “textCtrl” para introducir el dato de pérdidas por sombras.

Atributo **VolumenText**: “staticText” de volumen de acumulación.

Atributo **Volumen**: “textCtrl” para introducir el dato del volumen de acumulación.

Método **_init_ctrls**(self, prnt): inicia los “widgets” del interfaz.

Método **OnSombrasBoton**(self, event): manejador del evento de hacer click en el botón “Calcular Pérdidas”.

Método **OnSombrasClose**(self, event): manejador del evento de cerrar la ventana de cálculo de pérdidas por sombras.

Método **tomarcolector**(self, event): manejador del evento de elegir en “TipoColector”.

Método **actualizarperdidas1** (self,event): manejador del evento de introducir texto en ”Inclinación” o en “Orientación”.

Método **__init__**(self, parent, id, pos, size, style, name): constructor de la clase.
Método **compruebaDatos**(self): comprueba los datos que se introducen en el panel.

Método **cogerDatos**(self): devuelve los datos introducidos en el panel.

Método **cargarDatos**(self,datos): inicia los campos del panel.

Clase Dialog2 del fichero dialogoConfirma.py

Clase que implementa un cuadro de diálogo de confirmación genérico para el usuario.

Atributo **botonAceptar**: botón para aceptar.

Atributo **botonCancelar**: botón para cancelar.

Atributo **staticText1**: texto que se muestra al usuario.

Atributo **dev**: booleano, que toma el valor “True” si el usuario acpta o “False” si cancela.

Método **_init_ctrls**(self, prnt,cad): inicia los “widgets” del interfaz.

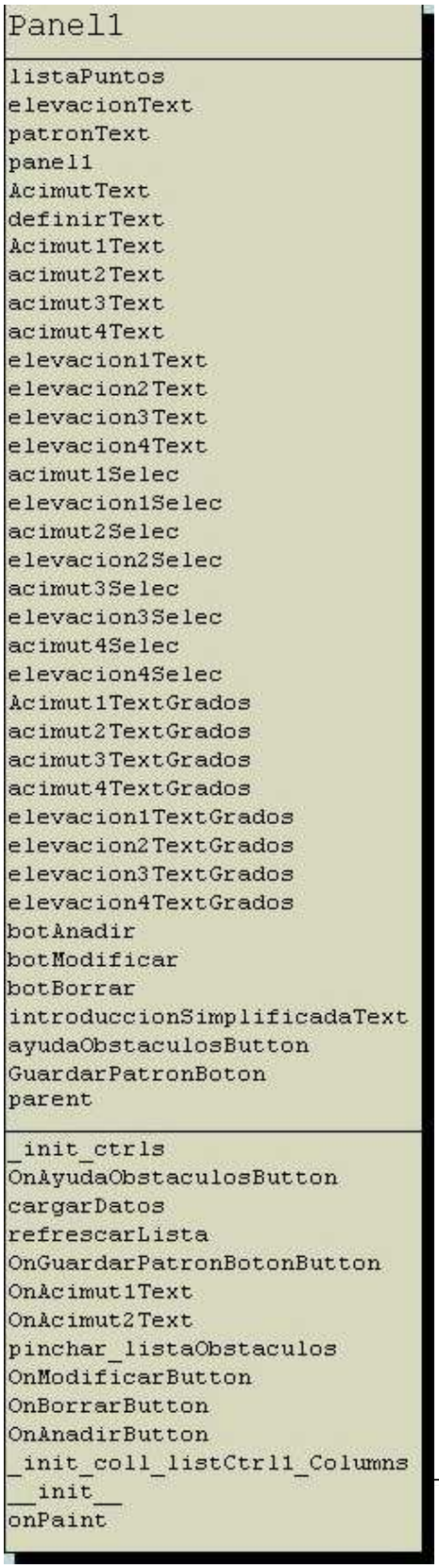
Método **__init__**(self, parent,cad): constructor de la clase.

Método **OnBotonAceptarButton**(self, event): manejador del evelto de hacer click en el botón “botonAceptar”.

Método **OnBotonCancelarButton**(self, event): manejador del evelto de hacer click en el botón “botonCancelar”.

Clase Panel1 del fichero menuObstaculosRemotos.py

Clase que implementa la ventana de recogida de datos de obstáculos remotos .



Atributo **listaPuntos**: lista de polígonos que el usuario a introducido.

Atributo **elevacionText**: “staticText” de elevación.

Atributo **patronText**: “staticText” del título de la ventana.

Atributo **panel1**: panel sobre el que se dibuja el patrón de sombras.

Atributo **AcimutText**: “staticText” de Acimut.

Atributo **definirText**: “staticText” de definir polígono.

Atributo **Acimut1Text**: “staticText” del acimut del primer punto.

Atributo **acimut2Text**: “staticText” del acimut del segundo punto.

Atributo **acimut3Text**: “staticText” del acimut del tercer punto.

Atributo **acimut4Text**: “staticText” del acimut del cuarto punto.

Atributo **elevacion1Text**: “staticText” de la elevación del primer punto.

Atributo **elevacion2Text**: “staticText” de la elevación del segundo punto.

Atributo **elevacion3Text**: “staticText” de la elevación del tercer punto.

Atributo **elevacion4Text**: “staticText” de la elevación del cuarto punto.

Atributo **acimut1Selec**: “textCtrl” para introducir el dato de acimut del primer punto.

Atributo **elevacion1Selec**: “textCtrl” para introducir el dato de elevación del primer punto.

Atributo **acimut2Selec**: “textCtrl” para introducir el dato de acimut del segundo punto.

Atributo **elevacion2Selec**: “textCtrl” para introducir el dato de elevación del segundo punto.

Atributo **acimut3Selec**: “textCtrl” para introducir el dato de acimut del tercer punto.

Atributo **elevacion3Selec**: “textCtrl” para introducir el dato de elevación del tercer punto.

Atributo **acimut4Selec**: “textCtrl” para introducir el dato de acimut del cuarto punto.

Atributo **elevacion4Selec**: “textCtrl” para introducir el dato de elevación del cuarto punto.

Atributo **Acimut1TextGrados**: “staticText” de los grados del primer punto.

Atributo **acimut2TextGrados**: “staticText” de los grados del segundo punto.

Atributo **acimut3TextGrados**: “staticText” de los grados del tercer punto.

Atributo **acimut4TextGrados**: “staticText” de los grados del cuarto punto.

Atributo **elevacion1TextGrados**: “staticText” de los grados del primer punto.

Atributo **elevacion2TextGrdos**: “staticText” de los grados del segundo punto.

Atributo **elevacion3TextGrados**: “staticText” de los grados del tercer punto.

Atributo **elevacion4TextGrados**: “staticText” de los grados del cuarto punto.

Atributo **botAnadir**: botón para añadir un polígono al dibujo.

Atributo **botModificar**: botón para modificar un polígono del dibujo.

Atributo **botBorrar**:botón para borrar un polígono del dibujo.

Atributo **introduccionSimplificadaText**: “staticText” de introducción simple.

Atributo **ayudaObstaculosButton**: botón para acceder a la ntroducción simplificada.

Atributo **GuardarPatronBoton**: botón para guardar el patrón creado y volver a la ventana principal de la aplicación.

Atributo **parent**: instancia del panel de recogida de datos de instalaciones.

Método **_init_ctrls**(self, prnt): inicia los “widgets” del interfaz.

Método **OnAyudaObstaculosButton**(self, event)

Método **cargarDatos**(self): inicia los campos del panel.

Método **refrescarLista**(self): refresca la lista de polígonos.

Método **OnGuardarPatronBotonButton**(self, event): manejador del evelto de hacer click en el botón "GuardarPatronBoton".

Método **OnAcimut1Text**(self, event): manejador del evento de introducir texto en "acimut1Selec".

Método **OnAcimut2Text**(self, event): manejador del evento de introducir texto en "acimut2Selec".

Método **pinchar_listaObstaculos**(self, event): manejador del evelto de hacer click en "listaPuntos**"**

Método **OnModificarButton**(self, event): manejador del evelto de hacer click en el botón "botModificar".

Método **OnBorrarButton**(self, event)): manejador del evelto de hacer click en el botón "botBorrar".

Método **OnAnadirButton**(self, event)): manejador del evelto de hacer click en el botón "botAnadir".

Método **_init_coll_listCtrl1_Columns**(self, parent): inicia la lista.

Método **__init__**(self, parent, id, pos, size, style, name, prov,alfa,beta): constructor de la clase.

Método **onPaint**(self, event=None): manejador del evento de dibujar en el panel.

Clase Comprueba del fichero compruebaCampos.py

Clase para comprobar los datos que el usuario introduce en el interfaz gráfico.

Atributo **dato**: dato introducido por el usuario.

Atributo **tipoObjeto**: "widget" del que proviene el dato que la clase debe analizar (choice, textctrl…)

Atributo **listaErrores**: lista inicial de errores posibles.

Atributo **valorMin**: en caso de que deba evaluar un valor numérico acotado, el valor mínimo que puede tomar.

Atributo **valorMax**: en caso de que deba evaluar un valor numérico acotado, el valor máximo que puede tomar.

Atributo **msgError**: mensaje de error que devuelve en caso de que encuentre algún fallo en el dato examinado.

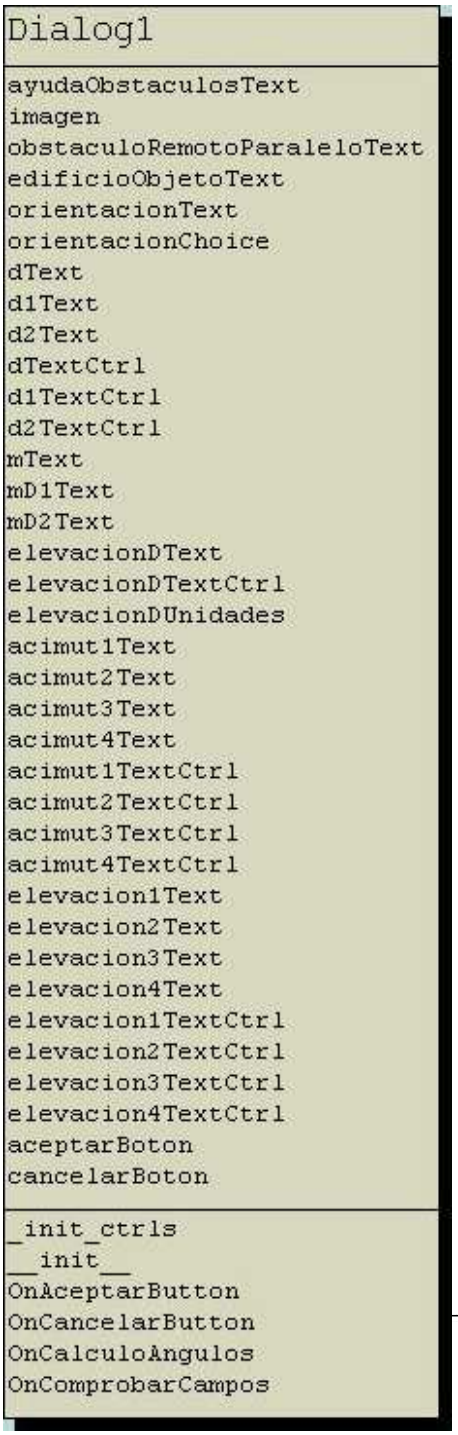
Atributo **ErrorDevuelto**: booleano que es "True" si hay algun error en el dato y "False" en caso contrario.

Método **__init__**(self,dato,tipoObjeto,lista,msgError,valorMin=None,valorMax=None): constructor de la clase.

Método **realizaComprobacion**(self): ejecuta las comprobaciones pertinentes en función de los Atributos de entrada.

Calse Dialog1 del fichero ayudaModificarObstaculos.py

Clase que implementa la ventana de recogida de datos de obstáculos remotos simplificados.



Atributo **ayudaObstaculosText**: “staticText” del título del cuadro de diálogo

Atributo **imagen**:

Atributo **obstaculoRemotoParaleloText**: “staticText” bajo la imagen.

Atributo **edificioObjetoText**: “staticText” bajo la imagen.

Atributo **orientacionText**: “staticText” de orientación.

Atributo **orientacionChoice**:

Atributo **dText**: “staticText” del campo “d”.

Atributo **d1Text**: “staticText” del campo “d1”.

Atributo **d2Text**: “staticText” del campo “d2”.

Atributo **dTextCtrl**: “textCtrl” para el dato “d”.

Atributo **d1TextCtrl**: “textCtrl” para el dato “d1”.

Atributo **d2TextCtrl**: “textCtrl” para el dato “d2”.

Atributo **mText**: “staticText” de unidades del campo “d”.

Atributo **mD1Text**: “staticText” de unidades del campo “d1”.

Atributo **mD2Text**: “staticText” de unidades del campo “d”.

Atributo **elevacionDText**: “staticText” del campo “elevación”.

Atributo **elevacionDTextCtrl**: “staticText” de unidades del campo “elevación”.

Atributo **elevacionDUnidades**: “staticText” de unidades del campo “elevación”.

Atributo **acimut1Text**: “staticText” del acimut del primer punto.

Atributo **acimut2Text**: “staticText” del acimut del segundo punto.

Atributo **acimut3Text**: “staticText” del acimut del tercer punto.

Atributo **acimut4Text**: “staticText” del acimut del cuarto punto.

Atributo **acimut1TextCtrl**: “textCtrl” para el dato de acimut del primer punto.

Atributo **acimut2TextCtrl**: “textCtrl” para el dato de acimut del segundo punto.

Atributo **acimut3TextCtrl**: “textCtrl” para el dato de acimut del tercer punto.

Atributo **acimut4TextCtrl**: “textCtrl” para el dato de acimut del cuarto punto.

Atributo **elevacion1Text**: “staticText” de la elevación del primer punto.

Atributo **elevacion2Text**: “staticText” de la elevación del segundo punto.

Atributo **elevacion3Text**: “staticText” de la elevación del tercer punto.

Atributo **elevacion4Text**: “staticText” de la elevación del cuarto punto.

Atributo **elevacion1TextCtrl**: “textCtrl” para el dato de elevación del primer punto.

Atributo **elevacion2TextCtrl**: “textCtrl” para el dato de elevación del segundo punto.

Atributo **elevacion3TextCtrl**: “textCtrl” para el dato de elevación del tercer punto.

Atributo **elevacion4TextCtrl**: “textCtrl” para el dato de elevación del cuarto punto.

Atributo **aceptarBoton**: botón para aceptar.

Atributo **cancelarBoton**: botón para cancelar.

Método **_init_ctrls**(self, prnt):): inicia los “widgets” del interfaz.

Método **__init__**(self, parent): constructor de a clase.

Método **OnAceptarButton**(self, event): manejador del evelto de hacer click en el botón "aceptarBoton".

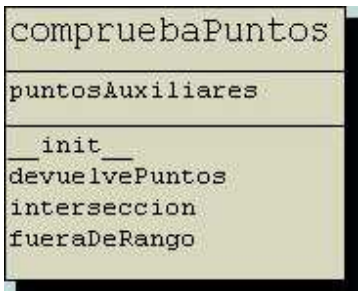
Método **OnCancelarButton**(self, event) : manejador del evelto de hacer click en el botón "cancelarBoton".

Método **OnCalculoAngulos**(self, event): manejador de todos los eventos de introducción de datos de la ventana. Si todos son correctos calcula los ángulos y completa los campos de acimut y elevación.

Método **OnComprobarCampos**(self): comprueba los datos introducidos por el usuario.

Clase compruebaPuntos

Clase para que si el usuario introduce valores de "Acimut" o "elevación" fuera de rango se corrijan automáticamente y se genere el o los polígonos necesarios para representar el obstáculo remoto.



Atributo **puntosAuxiliares**: valores que introduce el usuario y están fuera del rango común.

Método **__init__**(self,x1,y1,x2,y2,x3,y3,x4,y4): constructor de la clase.

Método **devuelvePuntos**(self): devuelve los puntos de el o los polígonos necearios para representar el obstáculo.

Método **interseccion**(self,poly): calculo la intersección en el plano de dos rectas definidas por dos puntos cada una.

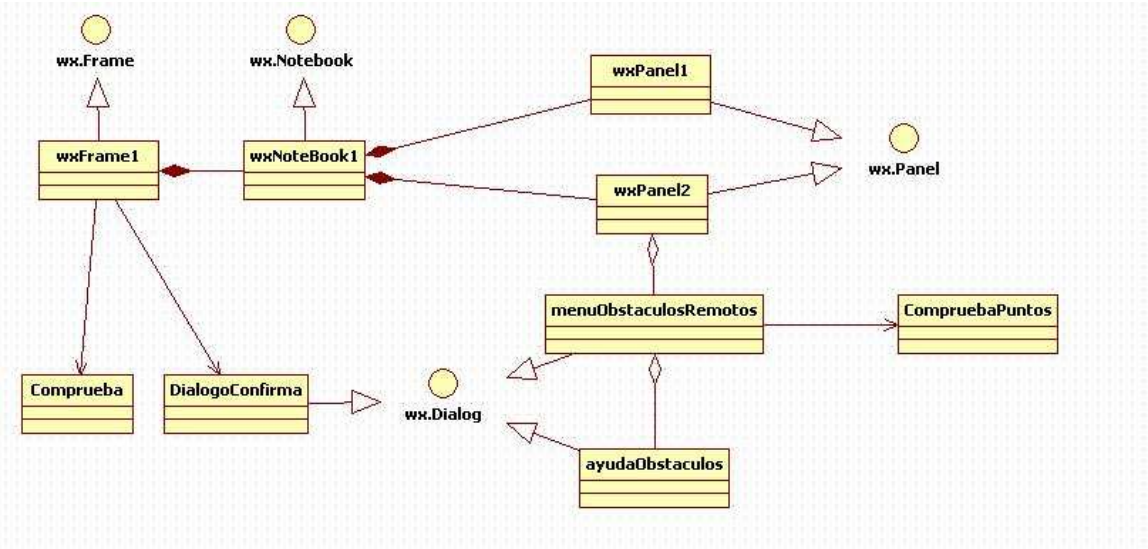
Método **fueraDeRango**(self,x,y): devuelve "True" si los puntos están fuera de rango y "False" en caso contrario.

4.2 Módulos de cálculo

Para implementar toda la funcionalidad de cálculo necesaria para este sistema, además de las clases explicadas en el apartado anterior son necesarias una serie de funciones que por razones de organización no entran en una clase concreta. Por ello todas estas funciones se han implementado en módulos aparte a modo de librerías de forma que pueden ser llamadas desde cualquier punto de la aplicación el cualquier momento. Estos módulos son:

- calcularPerdidasSombrass.py
- globalesFchart.py
- inicioNuevasSombras.py
- ayudaModificarObstaculos.py

4.3 Esquema general de clases de la aplicación



Capítulo 5: Implementación

A lo largo de este capítulo desarrollaremos todo el proceso de implementación de la aplicación que se ha desarrollado. A lo largo del mismo descubriremos la teoría que fundamenta los cálculos implementados, las técnicas de programación necesarias y la tecnología utilizada para el desarrollo.

5.1 Lenguaje de programación

El lenguaje elegido es Python [1] en su versión 2.5, ya que se trata de un lenguaje interpretado multiparadigma que se adapta perfectamente al modelo de diseño orientado a objetos desarrollado en capítulos anteriores. Aunque inicialmente la aplicación está diseñada para funcionar en plataforma Windows, Python y las librerías utilizadas son multiplataforma, por lo que no supondría mucho esfuerzo adaptar el código para que funcionase bajo otros sistemas operativos como Linux o MAC OS X. Otras características que hacen muy potente este lenguaje, es el tipado dinámico o que permite la herencia múltiple y el polimorfismo.

Los usuarios de Python se refieren a menudo a la “Filosofía Python”[10] que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el código opaco u ofuscado es bautizado como "no pythonico" ("unpythonic" en inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Ralo es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Aunque lo práctico gana a la pureza.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque *nunca* es a menudo mejor que *ya mismo*.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (*namespaces*) son una gran idea ¡Hagamos más de esas cosas!

Estos principios describen perfectamente las sensaciones que te da este lenguaje desde el primer momento que empiezas a trabajar con él. En mi caso se trataba de un lenguaje de programación que desconocía por lo que tuve que empezar desde cero con él. La parte positiva es que si que conocía la filosofía orientada a objetos y había trabajado ya con lenguajes como Java o .NET. Si a esto último sumamos la facilidad de sintaxis que otorga Python puedo asegurar que desde el primer momento conseguí resultados.

Otra ventaja que proporciona este lenguaje es la cantidad de librerías de las que dispone. En la versión 2.5 utilizada para el desarrollo, vienen incluidos en la distribución módulos para trabajar con tipos básicos de datos, el sistema, el sistema operativo, conexiones de red, serialización de objetos… Además existen multitud de librerías de sencilla instalación no incluidas en la distribución que aportan infinidad de funcionalidad a la hora de programar.

Para el desarrollo GUI en Python tenemos varias librerías disponibles, de entre las cuales hemos elegido "wxPython" [2]. Se trata de un "binding" de la biblioteca gráfica "wxWidgets" de C++ para el lenguaje de programación Python. La biblioteca" wxWidgets" se caracteriza por ser multiplataforma, por lo que su uso junto a Python permite el desarrollo rápido de aplicaciones gráficas multiplataforma.

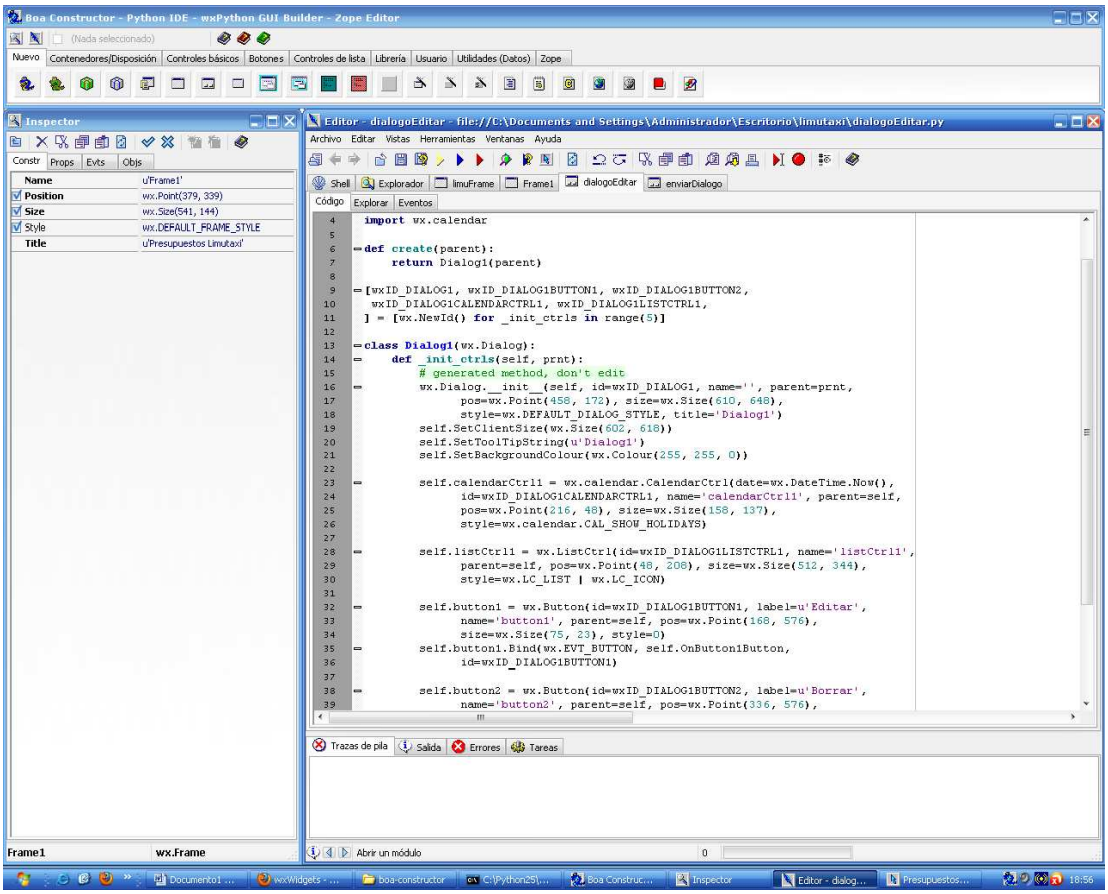
5.2 IDE y GUI Builder

IDE: Un entorno de desarrollo integrado (en inglés *integrated development environment*) es un programa informático compuesto por un conjunto de herramientas de programación.

GUI Builder: Un constructor de interfaz gráfica es una herramienta de programación que simplifica la creación de interfaces gráficas de usuario, permitiéndole al diseñador ordenar los widgets con un editor. Sin un constructor de interfaz, ésta se puede construir manualmente, especificando en el código fuente cada parámetro del widget que se quiere usar, pero sin obtener una previsualización del proceso, como sí lo permite el constructor.

Actualmente lo normal es que el IDE y el GUI Builder sean una misma aplicación que integra toda la funcionalidad. Para Python todos los programas con estas características y que además nos dan buenas prestaciones a la hora de programar son de pago, por lo que nos las hemos tenido que ingeniar para encontrar por un lado un GUI Builder que nos permita construir nuestra interfaz cómodamente y por otro un IDE que nos permita desarrollar lo más cómodamente posible.

En primer lugar hablaremos del "IDE y GUI Builder Boa Constructor", una aplicación libre y muy completa para el desarrollo en este lenguaje, pero con bastantes limitaciones a la hora de debuggear el código. Por ello hemos utilizado este software para construir nuestro interfaz gráfico para después elegir un buen IDE para continuar con el desarrollo. A continuación se muestra un pantallaza de la aplicación:



Como vemos se compone de tres ventanas independientes, el “wxPython GUI Builder” (arriba), el “Inspector” (izquierda) y el “Editor” (derecha).

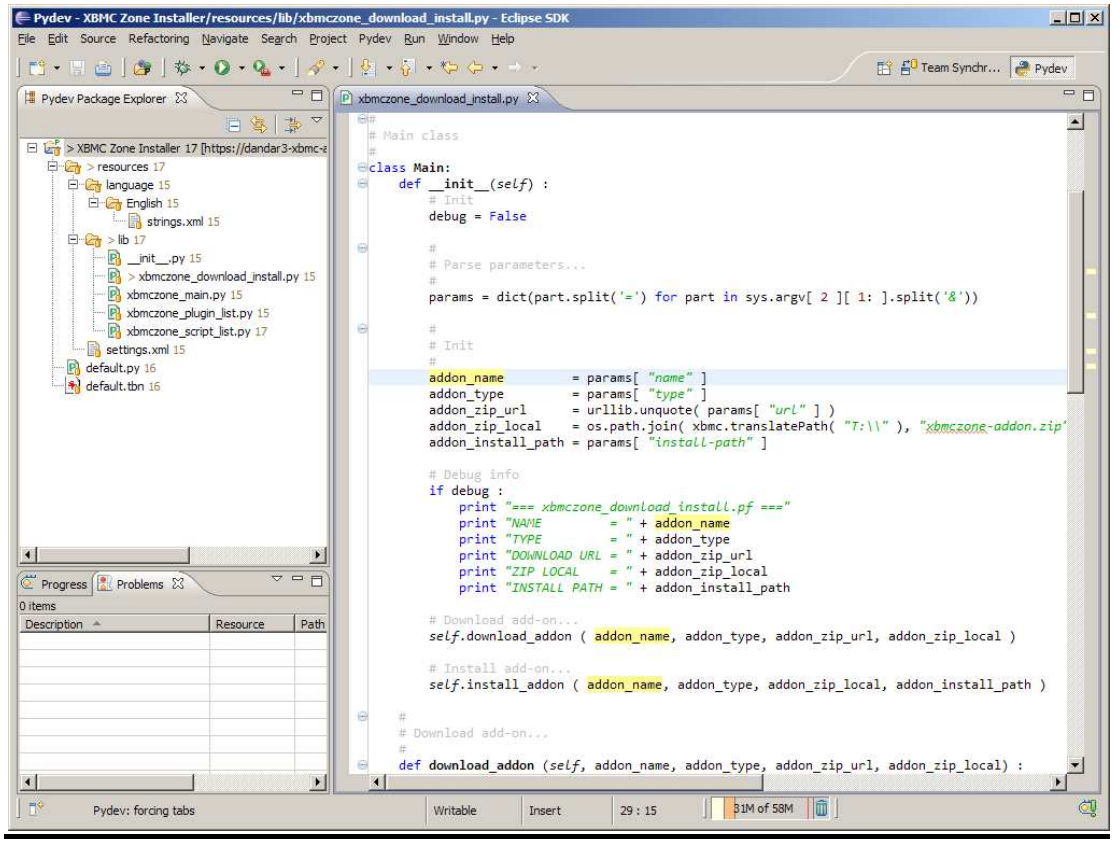
A lo largo de las pestañas del “wxPython GUI Builder” encontramos todos los “widgets” disponibles en la librería “wxPython” para que podamos situarlos en nuestro interfaz gráfico. Además nos permite la opción de crear nuestros propios controles de usuario y tenerlos disponibles en la pestaña “Usuario” para cuando nos hagan falta.

La ventana “Inspector” nos muestra la información de los objetos que vamos creando en el editor a partir de las selecciones en el “wxPython GUI Builder”. Nos permite modificar todos los argumentos de dichos objetos y nos muestra todas las opciones de eventos asociados al objeto creado.

Por último en la ventana “Editor” podemos ir desarrollando de forma sencilla el diseño del interfaz gráfico tanto en modo gráfico como en directamente sobre el código fuente que se ha ido generando.

El IDE elegido para desarrollar la aplicación es la archiconocida plataforma Eclipse junto con el plugin de desarrollo para Python, “Pydev” [14]. Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

“Pydev” junto a “Pydev Extensions” combinados con Eclipse dan como resultado un entorno de desarrollo ideal para Python al que solo le falta el constructor de interfaces gráficas. A continuación se presenta un pantallaza de la aplicación:



La principal característica que hace bueno a este IDE, es la filosofía de vistas configurables que posee. Hay distintas vistas que se pueden configurar como cada uno quiera para desarrollo, debbuging o testing. Además facilita encontrar las cosas con su procesador de textos que remarca en color amarillo las coincidencias encontradas o mediante el outliner que muestra en todo momento las partes que componen un fichero.

5.3 El cálculo F-chart

Como se ha citado en el primer capítulo de este proyecto, el método de cálculo de las curvas f (F-chart) ha sido el empleado para calcular la cobertura solar del edificio en estudio. A continuación se detalla el método de cálculo que se ha debido implementar.

La ecuación principal del cálculo viene determinada por la siguiente fórmula:

$$f = 1{,}029\, D_1 - 0{,}065\, D_2 - 0{,}245\, D_1^2 + 0{,}0018\, D_2^2 + 0{,}0215\, D_1^3$$

La secuencia que suele seguirse en el cálculo es la siguiente:

1. Valoración de las cargas caloríficas para el calentamiento de agua destinada a la producción de A.C.S. o calefacción.
2. Valoración de la radiación solar incidente en la superficie inclinada del captador o captadores.
3. Cálculo del parámetro *D*1.
4. Cálculo del parámetro *D*2.
5. Determinación de la gráfica *f*.
6. Valoración de la cobertura solar mensual.
7. Valoración de la cobertura solar anual y formación de tablas.

Las cargas caloríficas determinan la cantidad de calor necesaria mensual para calentar el agua destinada al consumo doméstico, calculándose mediante la siguiente expresión:

$$Q_a = C_e\, C\, N\, (t_{ac} - t_r)$$

donde:

*Q*a = Carga calorífica mensual de calentamiento de A.C.S. (J/mes)

*C*e = Calor específico. Para agua: 4187 J/(kgA°C)

C = Consumo diario de A.C.S. (l/día)

*t*ac = Temperatura del agua caliente de acumulación (°C)

*t*r = Temperatura del agua de red (°C)

N = Número de días del mes

El parámetro *D*1 expresa la relación entre la energía absorbida por la placa del captador plano y la carga calorífica total de calentamiento durante un mes:

*D*1 = Energía absorbida por el captador / Carga calorífica mensual

La energía absorbida por el captador viene dada por la siguiente expresión:

$$E_a = S_c\, F_r \mathrm{N(J")}\, R_1\, N$$

donde:

*S*c = Superficie del captador (m2)

*R*1 = Radiación diaria media mensual incidente sobre la superficie de captación por unidad de área (kJ/m2)

N = Número de días del mes

*F*rN(J") = Factor adimensional, que viene dado por la siguiente expresión:

$$F_rN(J") = F_r\,(J")_n\,[(J") / (J")_n]\,(F_rN/F_r)$$

donde:

*F*r (J")n = Factor de eficiencia óptica del captador, es decir, ordenada en el origen de la curva característica del captador.

(J") / (J")n = Modificador del ángulo de incidencia. En general se puede tomar como constante: 0,96 (superficie transparente sencilla) o 0,94 (superficie transparente doble).

*F*rN/ *F*r = Factor de corrección del conjunto captador-intercambiador. Se recomienda tomar el valor de 0,95.

El parámetro *D*2 expresa la relación entre las pérdidas de energía en el captador, para una determinada temperatura, y la carga calorífica de calentamiento durante un mes:

*D*2 = Energía perdida por el captador / Carga calorífica mensual

La energía perdida por el captador viene dada por la siguiente expresión:

$$E_p = S_c\,F_rN\,U_L\,(100 - t_a)\,\Delta t\,K1\,K2$$

donde:

*S*c = Superficie del captador (m2)

*F*rN *U*L = *F*r *U*L (*F*rN/ *F*r)

donde:

*F*r *U*L = Pendiente de la curva característica del captador (coeficiente global de pérdidas del captador)

*t*a = Temperatura media mensual del ambiente

Δ*t* = Período de tiempo considerado en segundos (s)

*K*1 = Factor de corrección por almacenamiento que se obtiene a partir de la siguiente ecuación:

$$K1 = [\text{kg acumulación} / (75\ Sc)]^{-0,25}$$

$$37,5 < (\text{kg acumulación}) / (\text{m2 captador}) < 300$$

K2 = Factor de corrección, para A.C.S., que relaciona la temperatura mínima de A.C.S, la del agua de red y la media mensual ambiente, dado por la siguiente expresión:

$$K2 = 11,6 + 1,18\ tac + 3,86\ tr - 2,32\ ta / (100 - ta)$$

donde:

tac = Temperatura mínima del A.C.S.
tr = Temperatura del agua de red
ta = Temperatura media mensual del ambiente

Una vez obtenido *D*1 y *D*2 , aplicando la ecuación inicial se calcula la fracción de la carga calorífica mensual aportada por el sistema de energía solar. De esta forma, la energía útil captada cada mes, *Q*u , tiene el valor:

$$Qu = f\ Qa$$

donde:

*Q*a = Carga calorífica mensual de A.C.S.

Mediante igual proceso operativo que el desarrollado para un mes, se operará para todos los meses del año. La relación entre la suma de las coberturas mensuales y la suma de las cargas caloríficas, o necesidades mensuales de calor, determinará la cobertura anual del sistema:

$$\text{Cobertura solar anual} = \sum Q_u \text{ necesaria} / Q_a \sum \text{necesaria}$$

5.4 Cobertura solar mínima exigida por el HE-4

En el documento anexo HE-4 correspondiente al CTE, se recoge toda la normativa que afecta a la cobertura solar mínima para edificios nuevos. De toda esa normativa, se extraen numerosos datos para implementar parte del modelo de la aplicación.

Cobertura solar mínima en función de la demanda de ACS y zona climática del edificio:

Tabla 2.1. Contribución solar mínima en %. Caso general

Demanda total de ACS del edificio (l/d)	Zona climática I	II	III	IV	V
50-5.000	30	30	50	60	70
5.000-6.000	30	30	55	65	70
6.000-7.000	30	35	61	70	70
7.000-8.000	30	45	63	70	70
8.000-9.000	30	52	65	70	70
9.000-10.000	30	55	70	70	70
10.000-12.500	30	65	70	70	70
12.500-15.000	30	70	70	70	70
15.000-17.500	35	70	70	70	70
17.500-20.000	45	70	70	70	70
> 20.000	52	70	70	70	70

Límites de pérdidas por sombras e inclinación/orientación:

Tabla 2.4 Pérdidas límite

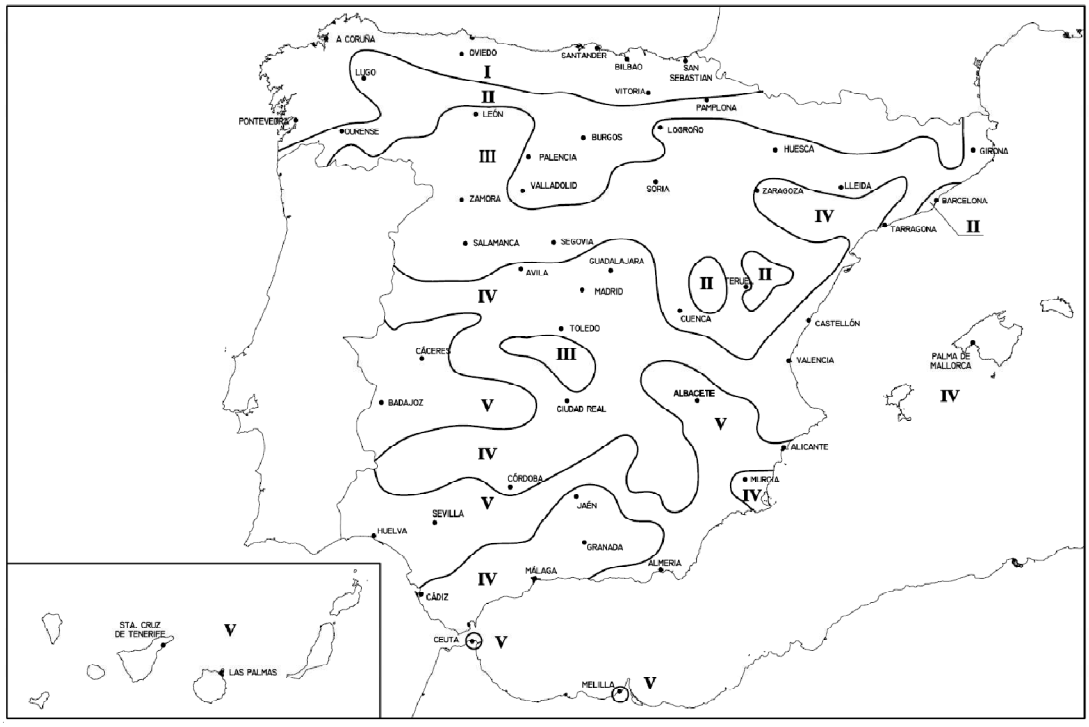
Caso	Orientación e inclinación	Sombras	Total
General	10 %	10 %	15 %
Superposición	20 %	15 %	30 %
Integración arquitectónica	40 %	20 %	50 %

Demanda de ACS de referencia en edificios y unifamiliares:

Tabla 3.1. Demanda de referencia a 60ºC (1)

Criterio de demanda	Litros ACS/día a 60º C	
Viviendas unifamiliares	30	por persona
Viviendas multifamiliares	22	por persona

Distribución geográfica de las zonas climáticas:



Fórmula de cálculo de las pérdidas por inclinación(β) y orientación(α):

$$Pérdidas\ (\%) = 100 \cdot [1{,}2 \cdot 10^{-4} \cdot (\beta - \beta opt)^2 + 3{,}5 \cdot 10^{-5} \alpha^2]$$

5.5 Implementación para calcular la contribución solar mínima

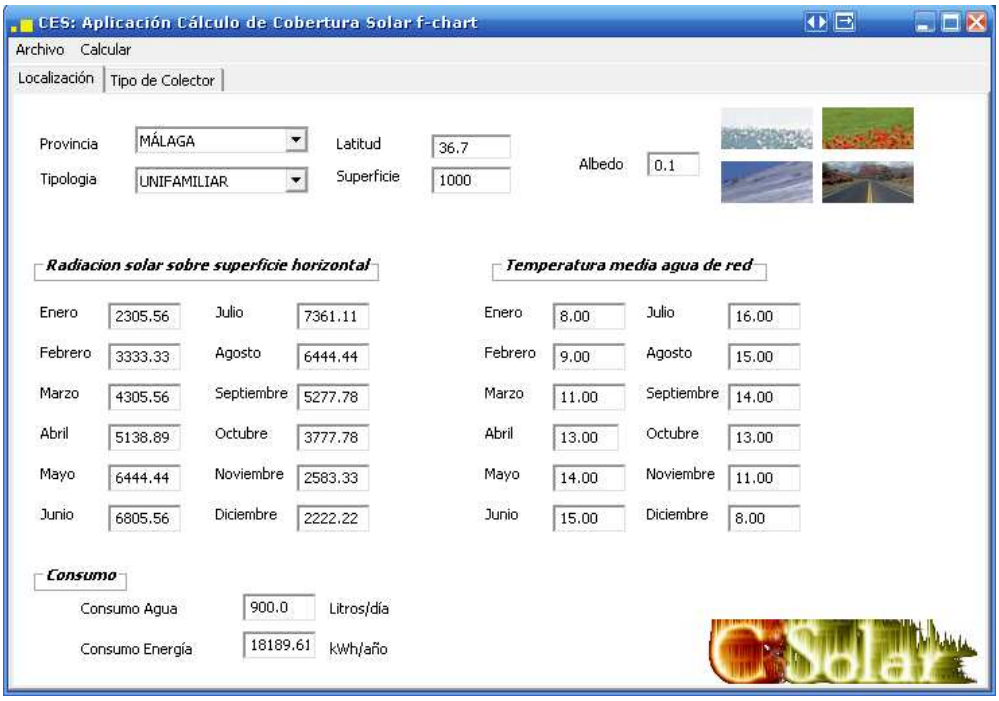
Todas estas fórmulas y normas se recogen en forma de código fuente en el fichero "globalesFchart.py". En dicho fichero se encuentran las funciones necesarias para realizar los cálculos y comprobaciones para asegurar un correcto cálculo de la cobertura solar en todos los casos posibles de edificios que el caso general del documento HE-4 exige, muchos cálculos nos los simplifica la librería "Numpy" [5]. Todas estas funciones las utiliza la aplicación para finalmente devolver al usuario si el edificio e instalación introducidos cumplen o no la normativa vigente.

Interfaces Gráficas para calcular la contribución solar mínima

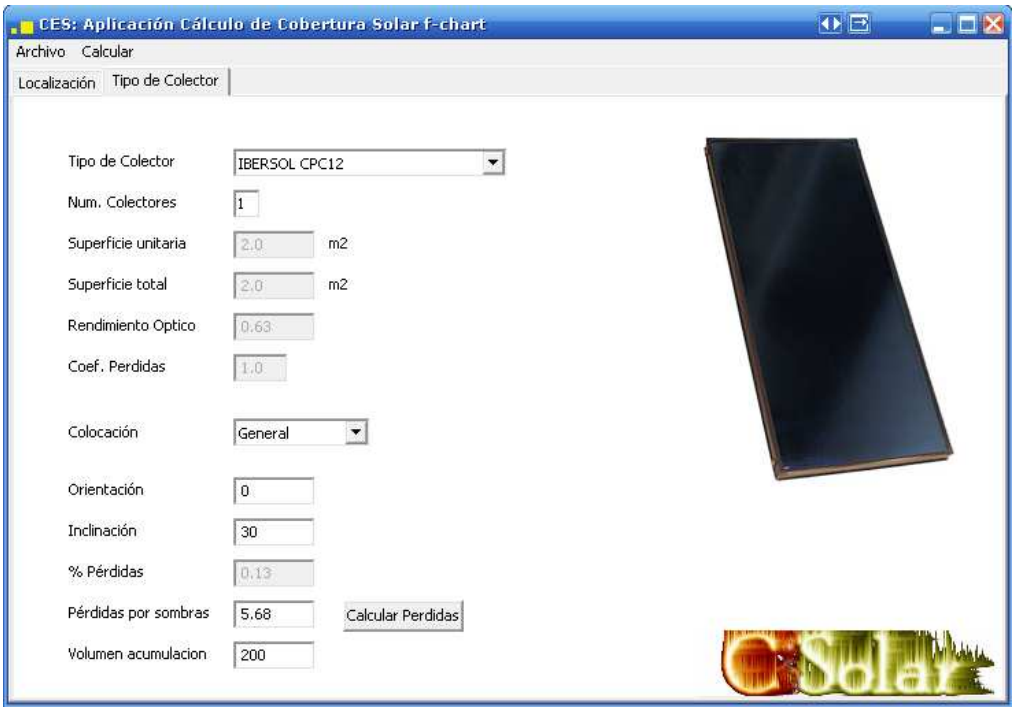
Para que el usuario pueda introducir todos los datos necesarios para este cálculo e interactúe con el sistema, se ha diseñado una pequeña ventana con una barra de menú y dos pestañas (Localización y Tipo de Colector).

La barra de menú consta de dos elementos: Archivo y Calcular. En el submenú "Archivo", el usuario encuentra las opciones típicas que se encuentran en cualquier aplicación como son "Nuevo", "Abrir", "Guardar" y "Salir". En el submenú "Calcular" el usuario encuentra las opciones "Calcular Cobertura" y "Comprobación CTE", que si los datos introducidos son correctos, devuelven la cobertura solar y el cumplimiento o no de la normativa respectivamente.

En la pestaña "Localización", se piden los datos relativos al edificio en estudio, en concreto la provincia donde se encuentra, el tipo de edificio que es, superficie útil del edificio y tipo de superficie sobre la que se encuentra (Albedo). Los datos relativos a la "Latitud", "Radiación solar sobre la superficie horizontal" y "Temperatura media del agua de la red" se rellenan automáticamente, con los valores por defecto cuando se introduce la provincia, aunque se deja que el usuario los edite si desea. Los datos relativos a los consumos, se rellenan automáticamente con los valores por defecto en función de la superficie del edificio, aunque se deja que el usuario los edite si lo desea. A continuación se presenta un pantallazo de esta pestaña.



En la pestaña “Tipo de Colector” se piden al usuario los datos que definen la instalación solar térmica del edificio. Se han obtenido datos de diferentes colectores comerciales, de forma que el usuario solo tenga elegir el tipo y número de colectores, de forma que el resto de parámetros se rellenen automáticamente. Además se le solicita al usuario la colocación, inclinación y orientación del colector. Por último es necesario introducir las pérdidas por sombras y el volumen de acumulación del depósito. Como se puede apreciar en el pantallazo de la pestaña que aparece a continuación, al lado del campo “Pérdidas por sombras” aparece el botón “Calcular Pérdidas”, que da acceso al interfaz para realizar el cálculo, que se explicará en el próximo apartado.



5.6 Cálculo de pérdidas de radiación solar por sombras

En este apartado se describe el método de cálculo de las pérdidas de radiación solar que experimenta una superficie debidas a sombras circundantes, que se explica en el documento HE-4 del CTE. Tales pérdidas se expresan como porcentaje de la radiación solar global que incidiría sobre la mencionada superficie, de no existir sombra alguna.

Para realizar el cálculo, es labor del usuario realizar el trabajo de campo de medir los obstáculos remotos que pueden incidir sobre la superficie de los paneles solares, para posteriormente introducir los datos obtenidos del perfil de los obstáculos en la aplicación. Los datos necesarios deben medirse en coordenadas angulares (acimut y elevación).

Acimut: ángulo de desviación con respecto a la dirección sur

Elevación: ángulo de inclinación con respecto al plano horizontal

Los datos recogidos sobre los obstáculos remotos deben representarse en el diagrama de trayectorias solares. Dicho diagrama representa la banda de trayectorias del sol a lo largo de todo el año. La línea inferior del diagrama representa el solsticio de invierno, es decir el día de menos horas de sol (22 de diciembre), y la superior el solsticio de verano (21 de junio). Por tanto la línea central representa a los equinoccios (20 de marzo y 22 de septiembre) La línea vertical central representa las doce del medio día (hora solar), es decir la hora en la que el sol está en el punto más alto de su trayectoria. El resto de líneas verticales representan una hora de desviación respecto a las doce hora solar.

Como vemos en la imagen, estas líneas forman una "cuadrícula", en la que cada casilla ha sido nominada (A1,A2,A3….D13,D14). Cada una de las porciones resultantes, representa el recorrido del sol en un cierto periodo de tiempo (una hora a lo largo de varios días) y tiene, por tanto, una determinada contribución a la irradiación solar global anual que incide sobre la superficie de estudio. Así, el hecho de que un obstáculo cubra una de las porciones supone una cierta pérdida de irradiación, en particular aquélla que resulte interceptada por el obstáculo.

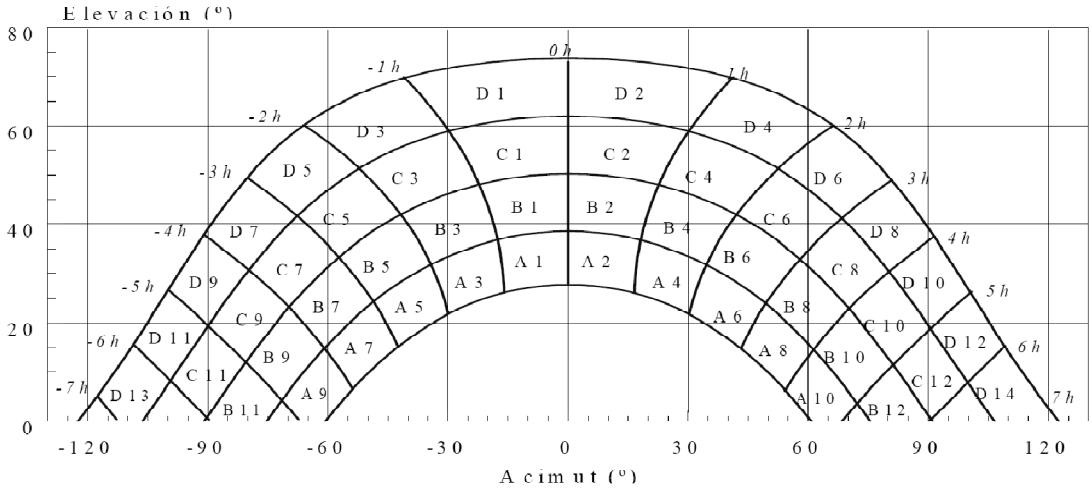


Figura 3.4 Diagrama de trayectorias del sol

De esta forma, si dibujamos el perfil de obstáculos remotos obtenido sobre el diagrama de trayectorias solares, una serie de casillas quedarán cubiertas total o parcialmente por el conjunto de obstáculos. A partir de aquí solo queda estimar el porcentaje de cada una de las casillas que está siendo cubierto y multiplicar dicho porcentaje por un coeficiente que nos lo dan las siguientes tablas:

Tabla B.1

	β=35° ; α=0°				β=0° ; α=0°				β=90° ; α=0°				β=35° ; α=30°			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
13	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,18	0,00	0,00	0,00	0,15	0,00	0,00	0,00	0,10
11	0,00	0,01	0,12	0,44	0,00	0,01	0,18	1,05	0,00	0,01	0,02	0,15	0,00	0,00	0,03	0,06
9	0,13	0,41	0,62	1,49	0,05	0,32	0,70	2,23	0,23	0,50	0,37	0,10	0,02	0,10	0,19	0,56
7	1,00	0,95	1,27	2,76	0,52	0,77	1,32	3,56	1,66	1,06	0,93	0,78	0,54	0,55	0,78	1,80
5	1,84	1,50	1,83	3,87	1,11	1,26	1,85	4,66	2,76	1,62	1,43	1,68	1,32	1,12	1,40	3,06
3	2,70	1,88	2,21	4,67	1,75	1,60	2,20	5,44	3,83	2,00	1,77	2,36	2,24	1,60	1,92	4,14
1	3,17	2,12	2,43	5,04	2,10	1,81	2,40	5,78	4,36	2,23	1,98	2,69	2,89	1,98	2,31	4,87
2	3,17	2,12	2,33	4,99	2,11	1,80	2,30	5,73	4,40	2,23	1,91	2,66	3,16	2,15	2,40	5,20
4	2,70	1,89	2,01	4,46	1,75	1,61	2,00	5,19	3,82	2,01	1,62	2,26	2,93	2,08	2,23	5,02
6	1,79	1,51	1,65	3,63	1,09	1,26	1,65	4,37	2,68	1,62	1,30	1,58	2,14	1,82	2,00	4,46
8	0,98	0,99	1,08	2,55	0,51	0,82	1,11	3,28	1,62	1,09	0,79	0,74	1,33	1,36	1,48	3,54
10	0,11	0,42	0,52	1,33	0,05	0,33	0,57	1,98	0,19	0,49	0,32	0,10	0,18	0,71	0,88	2,26
12	0,00	0,02	0,10	0,40	0,00	0,02	0,15	0,96	0,00	0,02	0,02	0,13	0,00	0,06	0,32	1,17
14	0,00	0,00	0,00	0,02	0,00	0,00	0,00	0,17	0,00	0,00	0,00	0,13	0,00	0,00	0,00	0,22

Tabla B.2

	β=90° ; α=30°				β=35° ; α=60°				β=90° ; α=60°				β=35° ; α= -30°			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
13	0,10	0,00	0,00	0,33	0,00	0,00	0,00	0,14	0,00	0,00	0,00	0,43	0,00	0,00	0,00	0,22
11	0,06	0,01	0,15	0,51	0,00	0,00	0,08	0,16	0,00	0,01	0,27	0,78	0,00	0,03	0,37	1,26
9	0,56	0,06	0,14	0,43	0,02	0,04	0,04	0,02	0,09	0,21	0,33	0,76	0,21	0,70	1,05	2,50
7	1,80	0,04	0,07	0,31	0,02	0,13	0,31	1,02	0,21	0,18	0,27	0,70	1,34	1,28	1,73	3,79
5	3,06	0,55	0,22	0,11	0,64	0,68	0,97	2,39	0,10	0,11	0,21	0,52	2,17	1,79	2,21	4,70
3	4,14	1,16	0,87	0,67	1,55	1,24	1,59	3,70	0,45	0,03	0,05	0,25	2,90	2,05	2,43	5,20
1	4,87	1,73	1,49	1,86	2,35	1,74	2,12	4,73	1,73	0,80	0,62	0,55	3,12	2,13	2,47	5,20
2	5,20	2,15	1,88	2,79	2,85	2,05	2,38	5,40	2,91	1,56	1,42	2,26	2,88	1,96	2,19	4,77
4	5,02	2,34	2,02	3,29	2,86	2,14	2,37	5,53	3,59	2,13	1,97	3,60	2,22	1,60	1,73	3,91
6	4,46	2,28	2,05	3,36	2,24	2,00	2,27	5,25	3,35	2,43	2,37	4,45	1,27	1,11	1,25	2,84
8	3,54	1,92	1,71	2,98	1,51	1,61	1,81	4,49	2,67	2,35	2,28	4,65	0,52	0,57	0,65	1,64
10	2,26	1,19	1,19	2,12	0,23	0,94	1,20	3,18	0,47	1,64	1,82	3,95	0,02	0,10	0,15	0,50
12	1,17	0,12	0,53	1,22	0,00	0,09	0,52	1,96	0,00	0,19	0,97	2,93	0,00	0,00	0,03	0,05
14	0,22	0,00	0,00	0,24	0,00	0,00	0,00	0,55	0,00	0,00	0,00	1,00	0,00	0,00	0,00	0,08

Como vemos estas tablas nos dan un coeficiente para cada casilla en función de los ángulos alfa y beta. El ángulo alfa oscila entre -60 y +60 grados y mide la orientación del panel respecto al punto cardinal sur. Por consiguiente los 0 grados representan una orientación a sur, los -60 grados representan una orientación suroeste y los 60 grados sureste. El ángulo beta oscila entre 0 y +90 grados y mide la inclinación del panel. Los 0 grados representan que el panel solar está situado en el plano horizontal y los 90 grados en el plano vertical. Con todo esto aplicamos la siguiente fórmula y obtenemos el porcentaje de pérdidas por sombras de nuestro panel.

$$\text{Pérdidas por sombras} = \sum Pc \cdot Cc$$

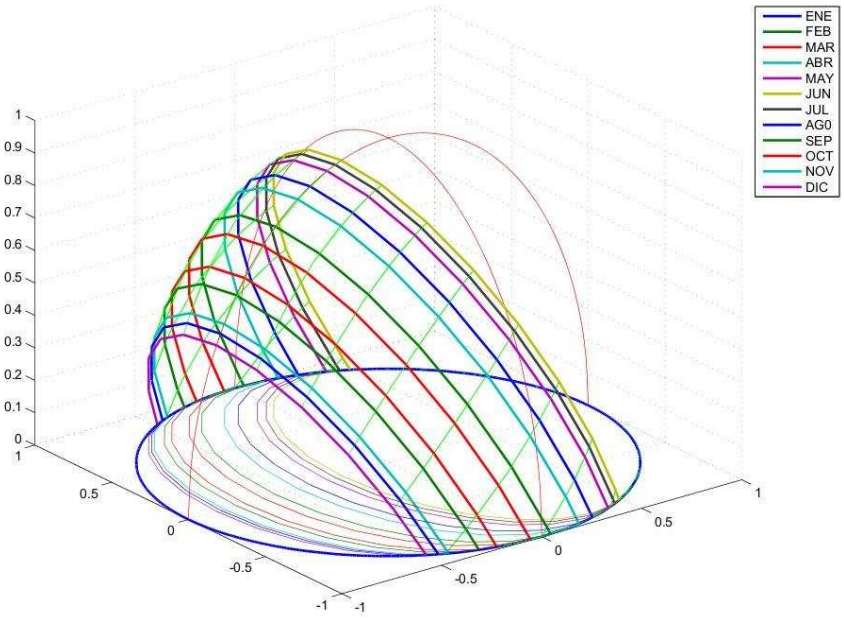
donde:

Pc: es el porcentaje cubierto de cada casilla.

Cc: es el coeficiente apropiado de las tablas para cada casilla.

5.7 Implementación para calcular las pérdidas de radiación solar por sombras

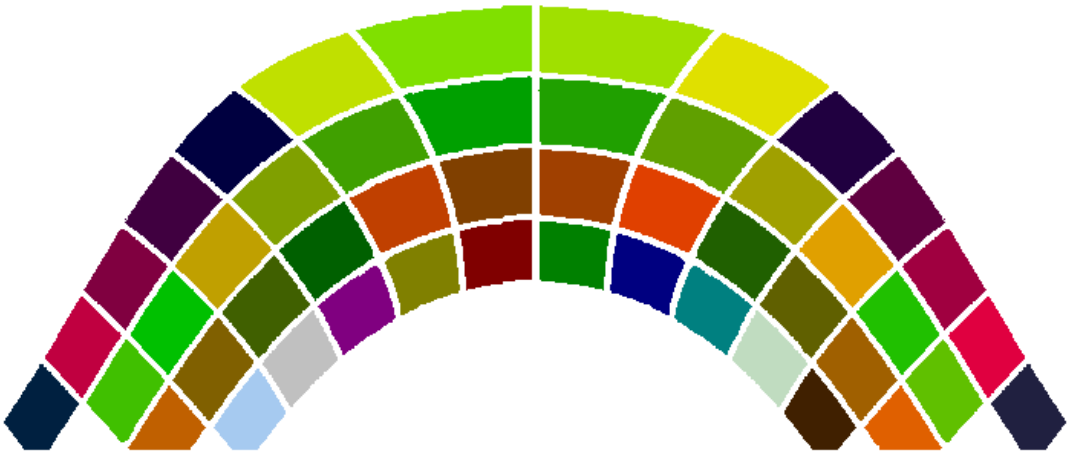
El primer problema que se nos plantea a la hora de implementar este cálculo es la dificultad de abstraer la explicación del mismo. No resulta para nada sencillo comprender el sistema de medición por coordenadas angulares que en el plano deriva en una proyección cilíndrica de la realidad que se representa. La siguiente imagen nos ayuda a comprender mejor la representación del diagrama de trayectorias solares de la figura 3.4.



Además de la trayectoria solar, debemos representar con coordenadas angulares los diferentes obstáculos sobre el diagrama. Como ya se ha citado es labor del usuario realizar el trabajo de campo, obtener las mediciones de acimut y elevación correspondientes. Para introducir estos datos en la aplicación, se ha pensado en un sistema de polígonos de cuatro lados, de forma que el usuario debe introducir las coordenadas de los cuatro vértices del polígono y de esta forma queda representado sobre el diagrama de trayectorias solares.

Una vez resuelto el problema de la representación de los obstáculos remotos, nos queda por resolver la forma de calcular el porcentaje de cada casilla que es cubierto por dichos obstáculos. Y la verdad que no es un problema sencillo de resolver, o al menos no lo es matemáticamente que es la primera solución que se puede venir a la cabeza. La resolución matemática de este problema supone realizar un montón de cálculos de intersecciones en un plano para hallar los puntos de corte de los polígonos con las líneas curvas del diagrama y para posteriormente hallar las superficies cubiertas en todos los casos posibles. Por todo esto se nos ocurrió otra forma de hacerlo, cuanto menos ingeniosa.

La técnica empleada se basa en un simple tratamiento fotográfico, para lo que nos ayudará la librería PIL (python image library [4]). En primer lugar se construye una imagen a 256 colores partiendo del diagrama de trayectorias solares, eliminando los bordes y pintando cada casilla de un color. El resultado es el siguiente:



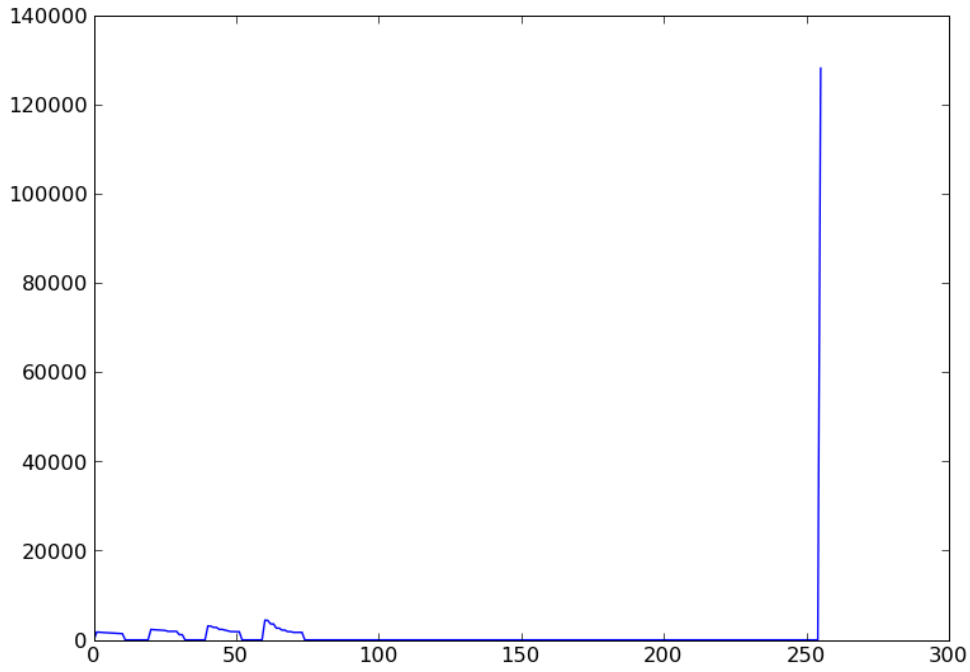
Esta imagen es constante, no cambia nunca y es necesaria como recurso externo de la aplicación. La razón de utilizar una imagen a 256 colores se debe a que es la opción más simple de color, que podemos utilizar para pintar de un color diferente, cada una de las 48 casillas que componen el diagrama. Los colores utilizados para cada una de las casillas, en notación ["Código de casilla" – "Código de color"], son:

[A1-1]	[A2-2]	[A3-3]	[A4-4]	[A5-5]	[A6-6]
[A7-7]	[A8-8]	[A9-9]	[A10-10]	[B1-21]	[B2-22]
[B3-23]	[B4-24]	[B5-25]	[B6-66]	[B7-27]	[B8-28]
[B9-29]	[B10-30]	[B11-31]	[B12 -32]	[C1-41]	[C2-42]
[C3-43]	[C4-44]	[C5-45]	[C6-46]	[C7-47]	[C8-48]
[C9-49]	[C10-50]	[C11-51]	[C12-52]	[D1-61]	[D2-62]
[D3-63]	[D4-64]	[D5-65]	[D6-66]	[D7-67]	[D8-68]
[D9-69]	[D10-70]	[D11-71]	[D12-72]	[D13-73]	[D14-74]

El siguiente paso es esperar a que el usuario defina en el interfaz gráfico los obstáculos remotos que afectan al panel solar a modo de polígonos. Tras esto se genera una nueva imagen dinámicamente, en concreto es una máscara para la imagen inicial, con los siguientes criterios:

1. Se genera una imagen a dos colores del mismo tamaño que la primera, iniciando todos los píxeles con el valor numérico 0.
2. Para cada polígono que introduce el usuario se comprueban todos los píxeles y se cambian a valor numérico 1 los que queden en el interior del polígono.

Llegados a este punto la solución se vuelve más intuitiva si se conoce el concepto de histograma de una imagen y de aplicar una máscara a una imagen. Un histograma nos da una lista con el número de píxeles de cada color que componen una imagen. La representación gráfica del histograma de la imagen a 256 colores es:



Como podemos observar sólo hay píxeles de los colores con los que hemos pintado cada una de las casillas y el color blanco de fondo que es el que predomina en la imagen. Aplicar una máscara a una imagen, consiste simplemente en multiplicar dos imágenes del mismo tamaño píxel a píxel. Por lo que aplicando la máscara generada a partir de los polígonos sobre la imagen a 256 colores, obtenemos una nueva imagen a 256 colores en la que únicamente aparecen coloreados los píxeles que quedan en el interior de los polígonos (el resto de píxeles quedan con el valor numérico 0, que corresponde al color negro). Para hallar los porcentajes cubiertos de cada casilla, aplicamos la siguiente fórmula:

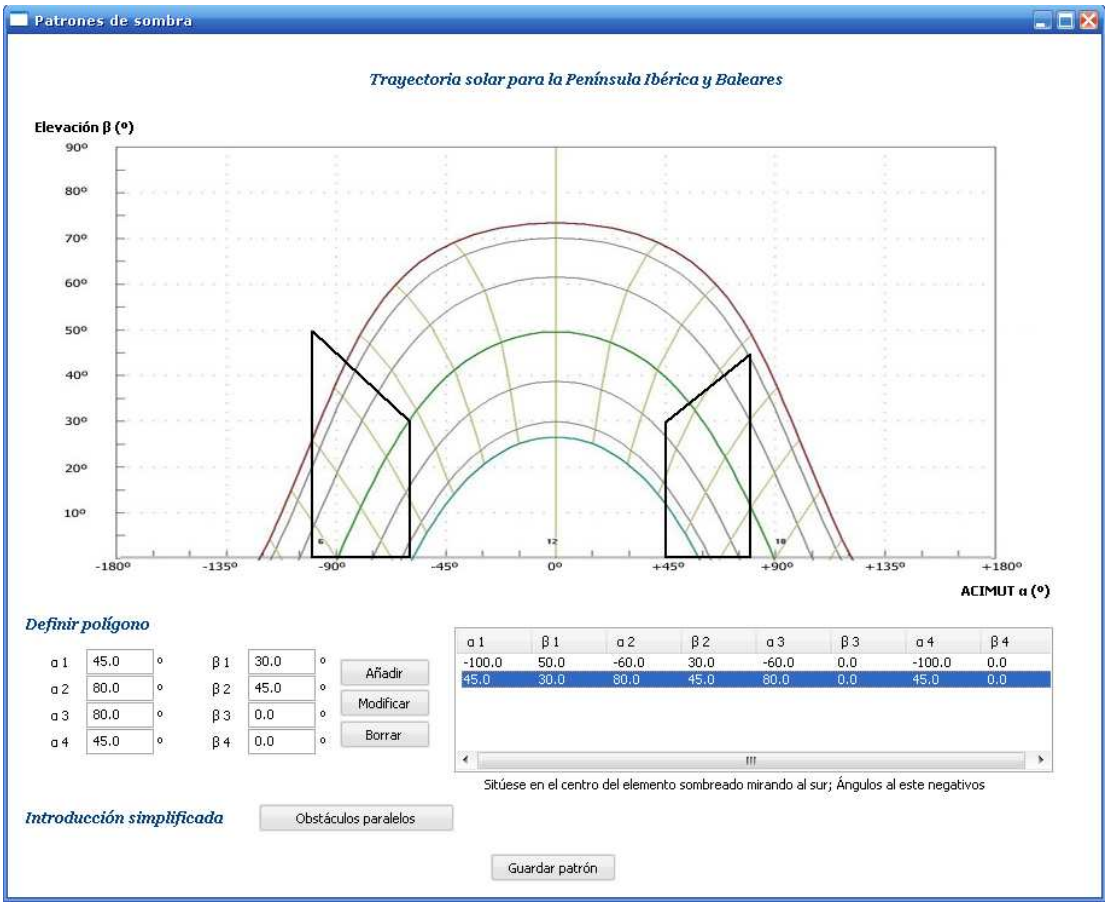
Porcentajes = Histograma(imagen 256 * mascara) * 100 / Histograma(imagen 256)

Todo este cálculo se implementa en el fichero “calcularPerdidasSombras.py” y para hacerlo ha sido necesario instalar dos librerías de Python que nos simplifican bastante las cosas. Por

un lado la librería “Polygon” [3], que integra funciones para trabajar con polígonos, y por otro la librería “Image”, que integra funciones para trabajar con imágenes.

Interfaz Gráfica para calcular las pérdidas por sombras

Como queda reflejado en el apartado *“Interfaces Gráficas para calcular la contribución solar mínima***”,** en la pestaña “Tipo de Colector” de la aplicación aparece el botón “Calcular Pérdidas”. Dicho botón da acceso al interfaz diseñado para este cálculo que se presenta a continuación.



Como vemos en la imagen, el interfaz presenta el diagrama de trayectorias solares y una serie de campos para definir los vértices de los polígonos que conforman los obstáculos

remotos. Las líneas más oscuras que aparecen sobre el diagrama son los polígonos que ya se han definido.

Los botones “Añadir”, “Modificar” y “Borrar” permiten al usuario gestionar los diferentes polígonos necesarios para la definición de los obstáculos remotos que desea introducir. En la tabla situada a la derecha de estos botones aparecen los datos de definición de los diferentes polígonos.

Una vez el usuario ha definido el patrón de sombras, con el botón “Guardar Patrón” se almacenan los datos de los polígonos introducidos para pasárselos al módulo de cálculo, se cierra esta ventana, se efectúa el cálculo de las pérdidas y se coloca el resultado de este cálculo en el campo “Perdidas por Sombras” de la pestaña “Tipo de Colector”.

El botón “Obstáculos paralelos” da acceso a otra ventana que se explica en el próximo apartado.

5.8 Simplificación para introducir obstáculos remotos paralelos

Dado que el método para hallar las coordenadas de “Acimut” y “Elevación” de los obstáculos remotos que afectan al panel resulta complejo, difícil de entender y que además es necesario disponer de equipos que midan ángulos como un sextante, se ha implementado otra forma de introducir los obstáculos remotos.

Para esta nueva definición solo son necesarios cinco datos:

Orientación; indica la orientación del plano del edificio objeto al cual se le va a aplicar el patrón de sombra.

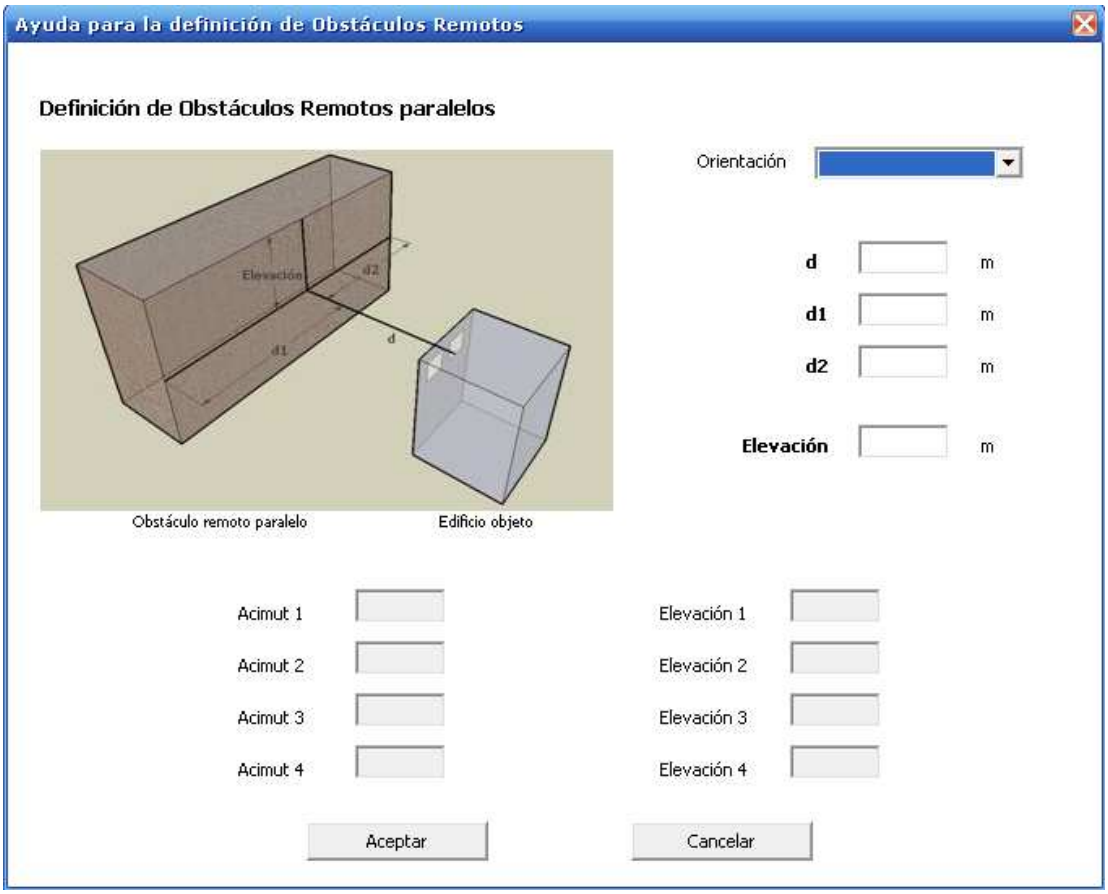
d (m); distancia o longitud de la línea perpendicular que une el plano al que se le aplicará el patrón de sombras del edificio objeto con el plano que provoca la sombra del objeto remoto paralelo.

d1 (m); situándose en el punto de cálculo del patrón de sombra del edificio objeto y observando desde él el obstáculo remoto, d1 es la distancia que hay desde la proyección de dicho punto sobre el obstáculo remoto hasta el final del obstáculo hacia la izquierda.

d2 (m); situándose en el punto de cálculo del patrón de sombra del edificio objeto y observando desde él el obstáculo remoto, d1 es la distancia que hay desde la proyección de dicho punto sobre el obstáculo remoto hasta el final del obstáculo hacia la derecha.

elevación (m); es la diferencia de cotas entre el punto de la superficie considerado para hallar el patrón de sombras y la elevación total del edificio que le proyecta la sombra, situado frente a él.

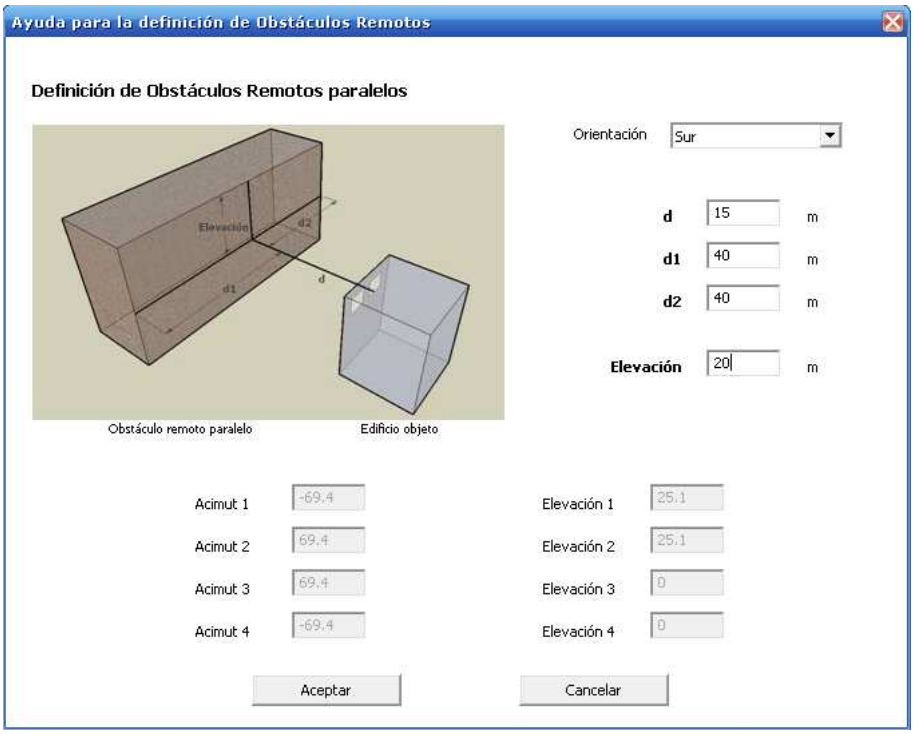
Con estos datos, se aplican cálculos vectoriales y trigonométricos para calcular los vértices del obstáculo y las coordenadas angulares de acimut y elevación de los mismos. Estos cálculos se recogen en el fichero "inicioNuevasSombras.py". El interfaz gráfico implementado es el siguiente:



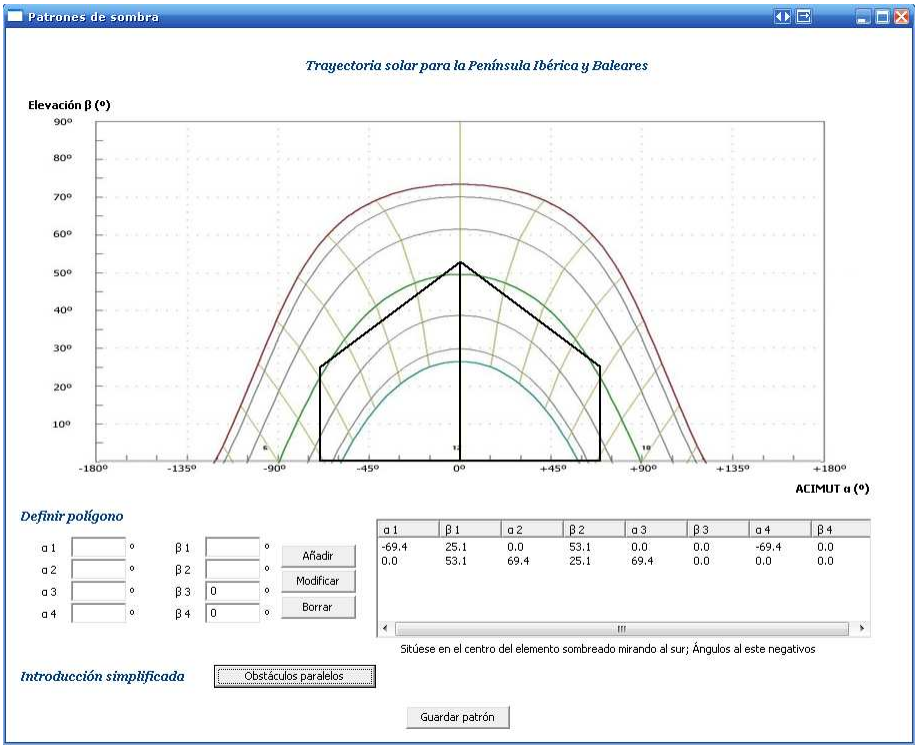
Al presionar el botón “Aceptar” se pasan los datos recogidos al módulo de cálculo, para que nos devuelva los polígonos necesarios para representar el obstáculo.

5.9 Ejemplo gráfico del método de cálculo del porcentaje cubierto por casilla

Supongamos una situación de obstáculos remotos paralelos

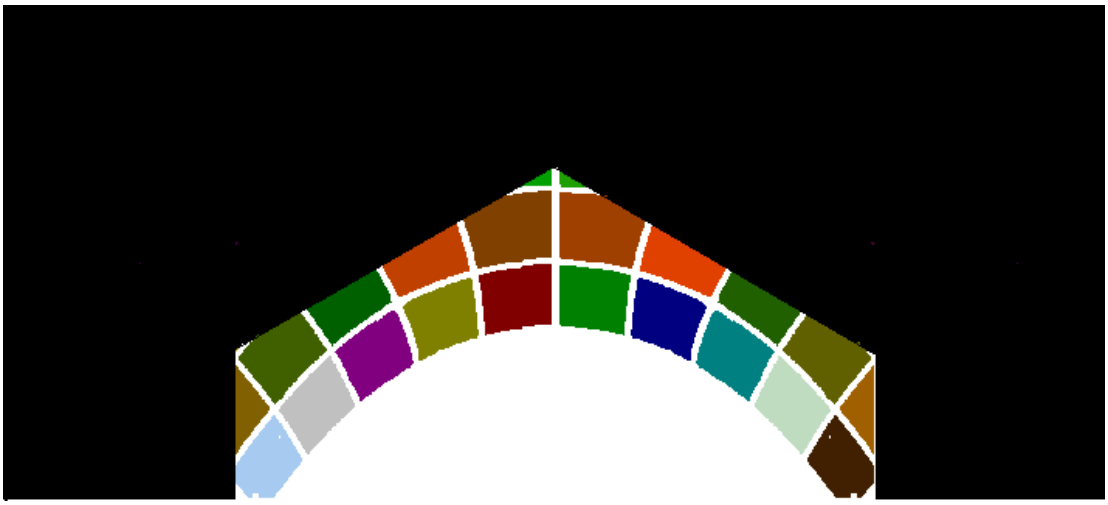


Que la aplicación traduce en los dos polígonos que aparecen sobre la carta solar:



Con la información de los polígonos se genera la máscara poniendo a 1 los píxeles que forman parte de los polígonos y dejando a 0 el resto. Para representarlo, el negro corresponde al 1 y el blanco al 1.

Se aplica la máscara sobre la imagen de 256 colores obteniendo como resultado:



Finalmente se aplica la técnica de los histogramas descrita en el apartado 5.7 para calcular el porcentaje de cada casilla que queda cubierto y se aplica la fórmula del apartado 5.6 para calcular las pérdidas por sombras anuales.

Capítulo 6: Análisis de código y pruebas

En todo desarrollo software resulta de vital importancia tener un código fuente fácil de leer e interpretar así como realizar pruebas sobre el mismo que verifiquen que actúa como nosotros queremos. Esta tarea debe realizarse durante todo el proceso de vida de creación del software para garantizar la máxima calidad. En este capítulo presentamos las herramientas utilizadas en el desarrollo de este proyecto para analizar el código fuente (PyLint) y realizar las pruebas (PyUnit).

6.1 Herramientas de análisis de código

Para mantener el ritmo cada vez mayor demanda de los clientes en la funcionalidad del software y el tiempo de las expectativas del mercado, los desarrolladores de software han tenido que evolucionar la manera de desarrollar un código que sea más rápido y de calidad superior. Por esto nacen las herramientas de análisis de código fuente (también llamadas herramientas de análisis estático o herramientas SCA) que analizan programas de software en las primeras etapas de desarrollo. Las herramientas SCA analizan un programa para calcular métricas y encontrar grietas potenciales y defectos en el código. De esta forma estas herramientas nos advierten de posibles fallos tales como las vulnerabilidades de seguridad, errores lógicos, análisis de la vulnerabilidad de código, los defectos de aplicación , violaciones de concurrencia, las condiciones de frontera raros, o cualquier número de otros tipos de problemas que causa el código.

6.2 La herramienta PyLint

Pylint [6] es la principal herramienta de análisis estático para Phyton. La verdad es que para ser una herramienta libre es bastante completa ya que hace muchas revisiones sobre el código fuente. Gracias a PyLint podemos realizar:

• Revisiones básicas
• Revisión mediante de inferencia de tipos
• Revisión de variables
• Revisión de clases
• Revisión de diseño
• Revisión de *imports*
• Revisión de conflictos entre viejo/nuevo estilo
• Revisión de formato
• Otras revisiones

Cabe destacar que Pylint utiliza el PEP8 como guía de estilo por defecto para realizar las comprobaciones de estilo. Podemos encontrar esta guía en los Anexos de este proyecto.

Revisiones básicas

La revisión básica abarca el siguiente tipo de comprobaciones:

• Presencia de cadenas de documentación (*docstring*).
• Nombres de módulos, clases, funciones, métodos, argumentos, variables.
• Número de argumentos, variables locales, retornos y sentencias en funciones y métodos.
• Atributos requeridos para módulos.
• Valores por omisión no recomendados como argumentos
• Redefinición de funciones, métodos, clases.
• Uso de declaraciones globales.

Revisión mediante inferencia de tipos

Python es un lenguaje dinámico, ya que las variables pueden cambiar su tipo en tiempo de ejecución. Por ello, no se pueden detectar errores de tipos en una fase de compilación.
Pylint intenta detectar errores asociados a los tipos de las variables utilizando para ello la inferencia de tipos, es decir, intentando averiguar qué tipos puede tener una variable determinada durante su ejecución. A partir de esta técnica, se hacen comprobaciones como las siguientes:

• Invocación de un miembro no existente sobre una variable.
• Llamadas a objetos “no invocables”.
• Asignación realizada a través de una función que no devuelve nada.

Revisión de variables

La revisión de variables abarca el siguiente tipo de comprobaciones:

• Comprobación de si una variable o *import* no está siendo usado.
• Variables no definidas.
• Redefinición de variables proveniente de módulos *built-in* o de ámbito externo.
• Uso de una variable antes de asignación de valor.

Revisión de clases

La revisión de clases abarca el siguiente tipo de comprobaciones:

• Métodos sin *self* como primer argumento.
• Acceso único a miembros existentes vía *self*.
• Atributos no definidos en el método *__init__*.

• Código inalcanzable.

Revisión de diseño

La revisión de diseño abarca el siguiente tipo de comprobaciones:

• Número de métodos, atributos, variables locales, entre otros.
• Tamaño, complejidad de funciones, métodos, entre otros.

Revisión de *imports*

La revisión de *imports* abarca el siguiente tipo de comprobaciones:

• Dependencias externas.
• *imports* relativos o importación de todos los métodos/variables mediante el uso de (*wildcard*).
• Uso de *imports* cíclicos.
• Uso de módulos obsoletos.

Revisión de conflictos entre viejo/nuevo estilo
• Uso de *property*, *__slots__*.
• Uso de *super*.

Revisiones de formato

La revisión de formato abarca el siguiente tipo de comprobaciones:

• Construcciones no autorizadas.
• Indentación estricta del código.
• Longitud de la línea.
• Uso de <> en vez de !=.

Otras revisiones

Adicionalmente, se realizan otras revisiones que abarcar las siguientes comprobaciones:
• Revisión de excepciones.
• Notas de alerta en el código como FIXME, XXX, TODO.
• Código sin tener una declaración de *encoding*. PEP-263
• Búsqueda por similitudes o duplicación en el código fuente.

Pylint clasifica las incidencias encontradas en las siguientes categorías:
• **Error (E)**. Errores de programación importantes, es probable que se trate de un *bug*.
• **Convención (C).** Incumplimientos del estándar de codificación (PEP8 por defecto).
• **Aviso (W).** Problemas de estilo o errores de programación menores.
• **Refactorización (R).** Incumplimientos de alguna buena práctica.

A continuación se listan las incidencias más comunes que se suelen cometer a la hora de codificar un sistema, y que Pylint nos detecta automáticamente:

E0203 – Acceso a variable miembro antes de asignar valor
PyLint reporta este tipo de mensajes cuando se detecta que una variable miembro es utilizada antes de haberle asignado ningún valor.
E0601 – Acceso a variable antes de asignar valor
PyLint reporta este tipo de mensajes cuando se detecta que una variable local es utilizada antes de haberle asignado ningún valor.
E0602 – Variable no definida
PyLint reporta este tipo de mensajes cuando detecta el uso de una variable que no está definida.
E1101 – Acceso a un miembro no existente de una variable
PyLint reporta este tipo de mensajes cuando se detecta que el código intenta acceder a un miembro que no existe en una instancia de una clase.
E1103 – Acceso a un miembro no existente de una variable (algunos tipos no pueden ser inferidos)
PyLint reporta este tipo de mensajes cuando se detecta que el código intenta acceder a un miembro que puede que no exista de una variable. Sin embargo, la herramienta no es capaz de inferir todos los posibles tipos de dicha variable.
R0201 – Un método puede ser una función
PyLint reporta este tipo de mensajes cuando un método no utiliza la referencia a la instancia (*self*) y por lo tanto podría ser escrito como una función.
R0901 – Demasiadas clases padre
PyLint reporta este tipo de mensajes cuando una determinada clase tiene demasiados antepasados, es decir, demasiadas clases padre. Es preferible reducir el número de clases base de tal manera que la clase sea más simple y, por lo tanto, más fácil de utilizar.
R0902 – Demasiados atributos
PyLint reporta este tipo de mensajes cuando una clase tiene demasiadas variables miembro. Es preferible reducir el número de variables miembro de tal manera que la clase sea más simple y, por lo tanto, más fácil de utilizar. Una forma de reducir el número de variables miembro es agrupando conjuntos de variables en estructuras o clases.
R0903 – Pocos métodos públicos
PyLint reporta este tipo de mensajes cuando una clase tiene demasiado pocos métodos públicos.
R0904 – Demasiados métodos públicos
PyLint reporta este tipo de mensajes cuando una clase presenta muchos métodos públicos.
R0911 – Demasiadas sentencias *return*
PyLint reporta este tipo de mensajes cuando un método o función tiene demasiadas sentencias *return* haciendo que el flujo de ejecución sea difícil de seguir.
R0912 – Demasiadas ramas de ejecución
PyLint reporta este tipo de mensajes cuando un método o función tiene demasiadas ramas de ejecución, haciendo que sea difícil de seguir el flujo de ejecución.
R0913 – Demasiados argumentos

PyLint reporta este tipo de mensajes cuando una función o método acepta demasiados argumentos.

R0914 – Demasiadas variables locales
PyLint reporta este tipo de mensajes cuando una función o método tiene demasiadas variables locales.
R0915 – Demasiadas sentencias en un método o función
PyLint reporta este tipo de mensajes cuando un método o función contiene demasiadas líneas de código. En este caso es recomendable "romper" el método grande en métodos más pequeños. De este modo se facilita la legibilidad del código.
W0102 – Argumentos con valores por defecto peligrosos
PyLint reporta este tipo de mensajes cuando se detecta el uso de un valor mutable como una lista o un direccionario como valor por defecto para un argumento.
W0104 – Sentencias que no tienen efecto
PyLint reporta este tipo de mensajes cuando se detecta una sentencia que no tiene ningún efecto.
W0107 – Sentencia *pass* innecesaria
PyLint reporta este tipo de mensajes cuando se detecta una sentencia *pass* que no es necesaria.
W0108 – Función Lambda probablemente innecesaria
PyLint reporta este tipo de mensajes cuando se detecta que el cuerpo de una expresión lambda consiste en una llamada a una función con la misma lista de argumentos que la expresión lambda. En estas condiciones y en la mayoría de los casos, la expresión lambda puede ser sustituida por una llamada directa a la función.
W0141 – Uso de funciones Python desaconsejadas
PyLint reporta este tipo de mensajes cuando se detecta el uso de una función desaconsejada.
W0201 – Atributos definidos fuera del método __init__
PyLint reporta este tipo de mensajes cuando una variable miembro de una instancia es definida fuera del método *__init__*(contructor de la clase).
W0212 – Acceso a un miembro protegido de una clase
PyLint reporta este tipo de mensajes cuando se accede a un miembro protegido de una clase (por ejemplo, un método de una clase que comienza por un carácter "_") desde fuera de la clase o de uno de sus descendientes.
W0221- Número de argumentos en método distinto al esperado
PyLint reporta este tipo de mensajes cuando se detecta un método que tiene un número de argumentos distinto al de la interfaz que implementa o del método que sobrescribe.
W0231 – Método *__init__* de la clase base no es llamado
PyLint reporta este tipo de mensajes cuando se detecta que una clase derivada no llama al método *__init__* de la clase base de la cual hereda.
W0301 – "Punto y coma" innecesario
PyLint reporta este tipo de mensajes cuando se detecta el uso de un "punto y coma" para finalizar una sentencia.
W0311 – Indentación inadecuada
PyLint reporta este tipo de mensajes cuando se detecta un número inesperado de indentaciones con tabuladores o espacios.

W0312 – Mezcla de tabuladores y espacios en un módulo
PyLint reporta este tipo de mensajes cuando detecta el uso de una mezcla de caracteres para realizar la indentación del código, o más concretamente cuando la indentación utilizada no es consistente con la opción "indent-string". Por defecto, esta propiedad utiliza un valor de 4 espacios, que es el estilo de indentación recomendado en el PEP8.
W0402 – Uso de un módulo *deprecated*
PyLint reporta este tipo de mensajes cuando se detecta la importación de un módulo marcado como *deprecated* (en desuso).
W0404 – Importación múltiple de un mismo módulo
PyLint reporta este tipo de mensajes cuando se detecta que un mismo módulo es importado múltples veces.
W0511 – Presencia de notas de aviso en el código
PyLint reporta este tipo de mensajes cuando se detecta en el código notas de aviso de tipo FIXME o similares.
W0601 – Variable global no definida en el ámbito de un módulo
PyLint reporta este tipo de mensajes cuando una variable ha sido definida mediante la sentencia *global* pero la variable no ha sido definida en el ámbito de un módulo.
W0602 – Uso de "*global*" para una variable que no es modificada
PyLint reporta este tipo de mensajes cuando se define una variable con la sentencia *global* pero no se realiza ninguna asignación a dicha variable.
W0611 – Módulo importado pero no usado
PyLint reporta este tipo de mensajes cuando un módulo que ha sido importado no es utilizado en el código.
W0612- Variable no usada
PyLint reporta este tipo de mensajes cuando se detectan variables que están definidas pero nunca son utilizadas.
W0613 – Argumento no usado
PyLint reporta este tipo de mensajes cuando se detecta un argumento de un método o función que nunca es utilizado en el código.
W0621 - Nombre redefinido (I)
PyLint reporta este tipo de mensajes cuando se detecta que una variable o función tiene el mismo nombre que una variable o función definida en un ámbito más exterior.
W0622 – Nombre redefinido (II)
PyLint reporta este tipo de mensajes cuando se detecta que una variable o función tiene el mismo nombre que una variable o función definida en una librería, sobrescribiendo así dicho nombre.
W0631 – Uso de una variable de bucle fuera del bucle
PyLint reporta este tipo de mensajes cuando una variable de bucle, es decir, una variable definida por un bucle *loop*, un *list comprehesion* o un generador de expresiones, es utilizada fuera del bucle.
W0702 – Tipo de excepción no especificado
PyLint reporta este tipo de mensajes cuando se detecta un bloque *except* que no especifica qué tipo de excepciones capturar.
C0302 – Demasiadas líneas en un módulo
PyLint reporta este tipo de mensajes cuando encuentra un módulo cuyo número de líneas supera el valor de 1000. Los módulos con muchas líneas de código son difíciles de leer y,

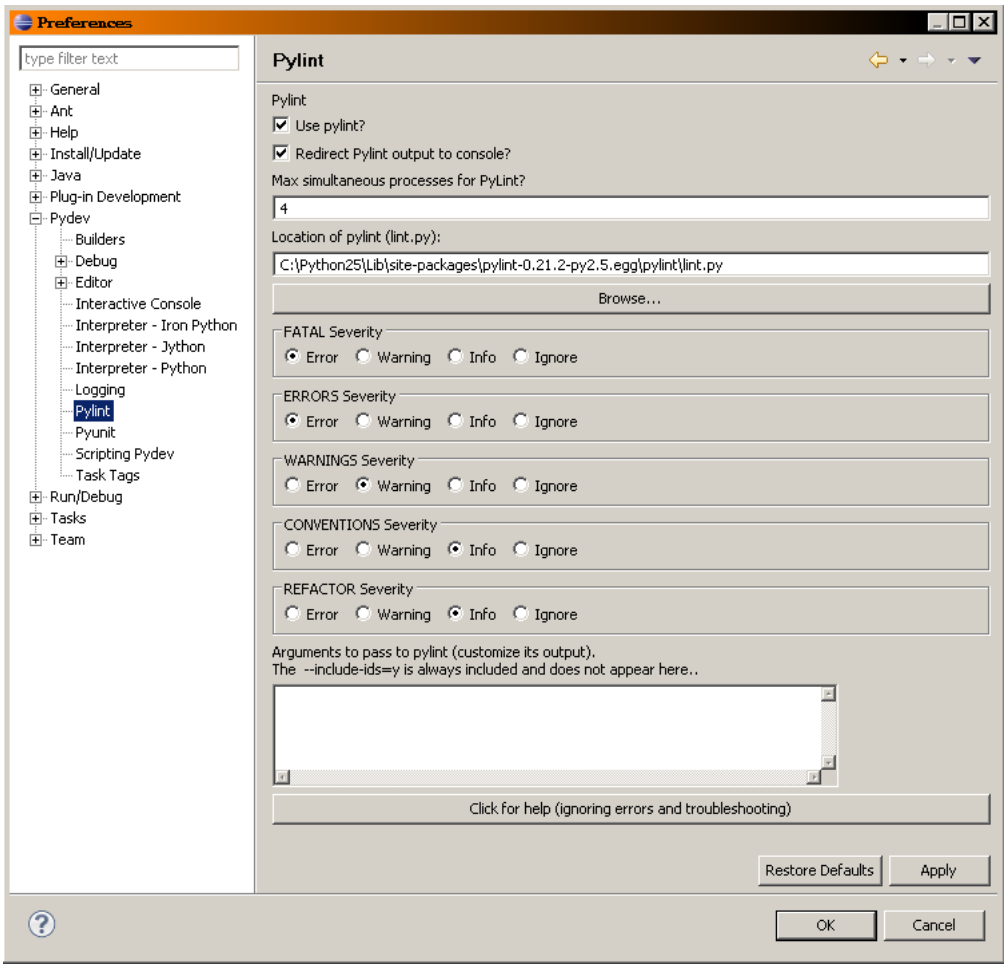
generalmente, suele ser un indicador de la existencia de componentes de código (clases, métodos) cuya cohesión es baja.

C0111 – *Docstring* no encontrado
Se recomienda documentar adecuadamente el código fuente mediante *docstrings*, especialmente todos los módulos, funciones, clases y métodos que sean públicos. No se considera estrictamente necesario usar *docstrings* en métodos no públicos, aunque es recomendable proporcionar algún tipo de documentación (por ejemplo, comentarios) que indique qué hace el método.
El PEP 257 proporciona las guías y convenciones necesarias para escribir *docstrings* adecuados.
C0103 - Nombres no válidos
PyLint reporta este tipo de mensajes cuando el nombre del algún elemento de código no coincide con las convenciones establecidas por la herramienta. Estas convenciones de nombres están basadas en el "*PEP 8 – Style Guide for Python Code*".
C0301 – Líneas demasiado largas
PyLint reporta este tipo de mensajes cuando una línea de código es más larga que un determinado número de caracteres. El umbral por defecto de la herramienta está establecido en 80, aunque el límite fijado por el PEP 8 está en 79 caracteres por línea.
C0321 – Más de una sentencia en una línea
PyLint reporta este tipo de mensajes cuando se encuentra más de una sentencia en la misma línea.
C0322 – Operador no precedido por un espacio
PyLint reporta este tipo de mensajes cuando se detecta un operador (!= | <= | == | >= | < | > | = | += | -= | *= | /= | %) que no está precedido por un espacio.
C0323 – Operador no seguido por un espacio
PyLint reporta este tipo de mensajes cuando se detecta un operador (!= | <= | == | >= | < | > | = | += | -= | *= | /= | %) que no está seguido por un espacio.
C0324 – "Coma" no seguida de un espacio
PyLint reporta este tipo de mensajes cuando se encuentra en el código fuente una coma (,) que no está seguida de un espacio.

6.3 Integración de PyLint en el IDE Eclipse

Aunque como herramienta de análisis estático está bastante lograda, Pylint a la hora de utilizarse para un proyecto completo es bastante desagradable de usar. Hay que ejecutarlo sobre línea de comandos fichero a fichero que quieras analizar. Además si desarrollas un módulo completo y después ejecutas el análisis de Pylint, lo más probable es que tengas un montón de referencias a incidencias que debes comprobar y tratar adecuadamente.

Por suerte el IDE empleado para el proecyo incorpora por defecto la opción de configuración de Pylint. En la ventana de configuración debemos elegir donde se encuentra Pylint en nuestro ordenador y asignar qué errores queremos que ignore porque no deseamos tenerlos en cuenta. A continuación se presenta un pantallazo de la ventana de configuración de Pylint en Eclipse:



Una vez configurado, Eclipse ejecuta Pylint cada vez que guardamos un fichero, de forma que nos avisa como si errores de sintaxis se tratase las incidencias que Pylint detecta. A partir de aquí es nuestra decisión corregir la incidencia o ignorarla.

6.4 Pruebas Unitarias del Software

La integración contínua (Continuous integration) es un conjunto de metodologías de ingeniería de software que aplicadas sirven para acelerar el desarrollo de software basándose en la aplicación de cambios de una forma ágil y controlada. La integración contínua fue propuesta inicialmente por Martin Fowler y Kent Beck a partir de las metodologías que forman la programación extrema.

La realización de pruebas unitarias (unit testing) juega un papel fundamental en este conjunto de metodologías de integración continua, ya que es responsable de la comprobación automática de la funcionalidad individual de cada unidad de código, habitualmente cada clase, utilizando el paradigma de programación orientada a objetos. Dicho de otra forma, una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

El concepto de las pruebas unitarias, es sencillo. Si tenemos que hacer un programa, e ir comprobando según lo vamos haciendo si la funcionalidad del programa se corresponde con lo que en principio queríamos hacer, ¿porqué no programar una pequeña aplicación que haga las comprobaciones por nosotros? El comportamiento habitual de la mayoría de programadores es o bien ir escribiendo texto por la salida estandar como “llego hasta aqúi”, “el valor de esta variable es…”, etc. código que hay que escribir, comprobar, y luego eliminar. Otra posibilidad sería escribir todo el código confiando en nuestra concentración y luego usar algún algún tipo de debugger si el código no hace lo esperado. En el primer caso, perderíamos mucho tiempo. En el segundo quizá la pérdida de tiempo sería menor pero probablemente estemos pasando por alto casos de prueba que podrían ser propensos a error.

Las pruebas unitarias son una herramienta muy importante para el personal de pruebas pero, sobre todo, para los desarrolladores. Son pruebas dirigidas a probar clases aisladamente y están relacionadas con el código y la responsabilidad de cada clase y sus fragmentos de código más criticos.

El principal factor que se debe considerar al inicio de las pruebas es el tamaño del módulo a probar, se debe considerar si el tamaño del módulo permitirá probar adecuadamente toda su funcionalidad de manera sencilla y rápida. También es importante separar los módulos de acuerdo a su funcionalidad, si los módulos son muy grandes y contienen muchas funcionalidades, estos se volverán más complejos de probar y al encontrar algún error será más difícil ubicar la funcionalidad defectuosa y corregirla. Al hacer esta labor el analista de pruebas podrá recomendar que un modulo muy complejo sea separado en 2 o 3 módulos más sencillos.

¿Porque realizar pruebas unitarias?

• Asegura calidad del código entregado. Es la mejor forma de detectar errores tempranamente en el desarrollo. No obstante, esto no asegura detectar todos los errores, por tanto prueba de integración y aceptación siguen siendo necesarias.

• Ayuda a definir los requerimientos y responsabilidades de cada método en cada clase probada.

• Constituye una buena forma de ejecutar pruebas de concepto. Cuando es necesario hacer pruebas de conceptos sin integrar usar pruebas unitarias se convierte en un método efectivo.

• Permite hacer refactoring tempranamente en el código. No es necesario todo un ciclo de integración para hacer refactoring en la aplicación, basta con ver como se comporta un caso de prueba para hacer refactoring unitario sobre la clase que estamos probando en cuestión.

• Permite incluso hacer pruebas de estress tempranamente en el código. Por ejemplo un método que realize una consulta SQL que exceda los tiempos de aceptación es posible optimizarla antes de integrar con la aplicacíón.

• Permite encontrar errores o bugs tempranamente en el desarrollo. Y está demostrado que mientras más temprano se corrigan los errores, menos costará corregirlos.

Por esto, las pruebas unitarias son tan importantes en la integración contínua. Sin ellas, los cambios realizados a intervalos pequeños y regulares introducirían probablemente errores difíciles de detectar.

Para la realización de pruebas unitarias, existen herramientas y entornos de desarrollo (frameworks) que facilitan su creación en multitud de lenguajes de programación. Para Pyhon tenemos el módulo PyUnit, que integra una forma sencilla de programar pruebas unitarias para nuestro código.

6.5 La herramienta PyUnit

PyUnit es la herramienta o framework oficial para hacer pruebas unitarias en Python [7]. Se incluye en la librería estándar (en el módulo unittest) y ha sido creado por los desarrolladores de JUnit para facilitar la creación y gestión de tests de prueba en módulos Python.

Este framework es soportado perfectamente por el IDE que estoy usando en el desarrollo del proyecto, que se trata del Eclipse con la extensión PyDev. Permite separar el código de pruebas del código del propio proyecto para mejorar la comprensión, refactorización y la

facilidad para hacer cambios en el programa. Por otra parte, se pueden realizar los tests por línea de comandos.

Las pruebas que es capaz de realizar son (comprobar que no se producen situaciones imposibles) y pruebas de mayúsculas (relacionadas con las cadenas de caracteres).

Debido a todo esto, y a que se trata de la herramienta estándar de Python, tiene mucha difusión y existe mucha documentación sobre esta librería en la red. Es la que he elegido para realizar las pruebas unitarias en mi proyecto.

Los bloques básicos para construir una prueba de unidad se denominan "casos de prueba" (escenarios únicos que deben ser configurados para probar que son correctos). para poder hacer un caso de prueba propio en PyUnit es necesario hacer una subclase de "TestCase", que es un objeto que puede correr un único método de prueba.

Para realizar una prueba de algo en Python se usa el comando "assert", si este comando falla cuando se ejecuta el caso de prueba PyUnit dará el caso por fallido.

Al escribir los casos de prueba es mejor hacerlo en un modulo separado, y no incluirlos entre el código fuente ya que asi los casos de prueba se pueden correr por separado, se evita la mala idea de cambiar el codigo fuente para que se ajuste a los casos de prueba, el codigo probado puede ser cambiado mas fácilmente y el código de prueba es modificado en una medida menor que el código que es probado.

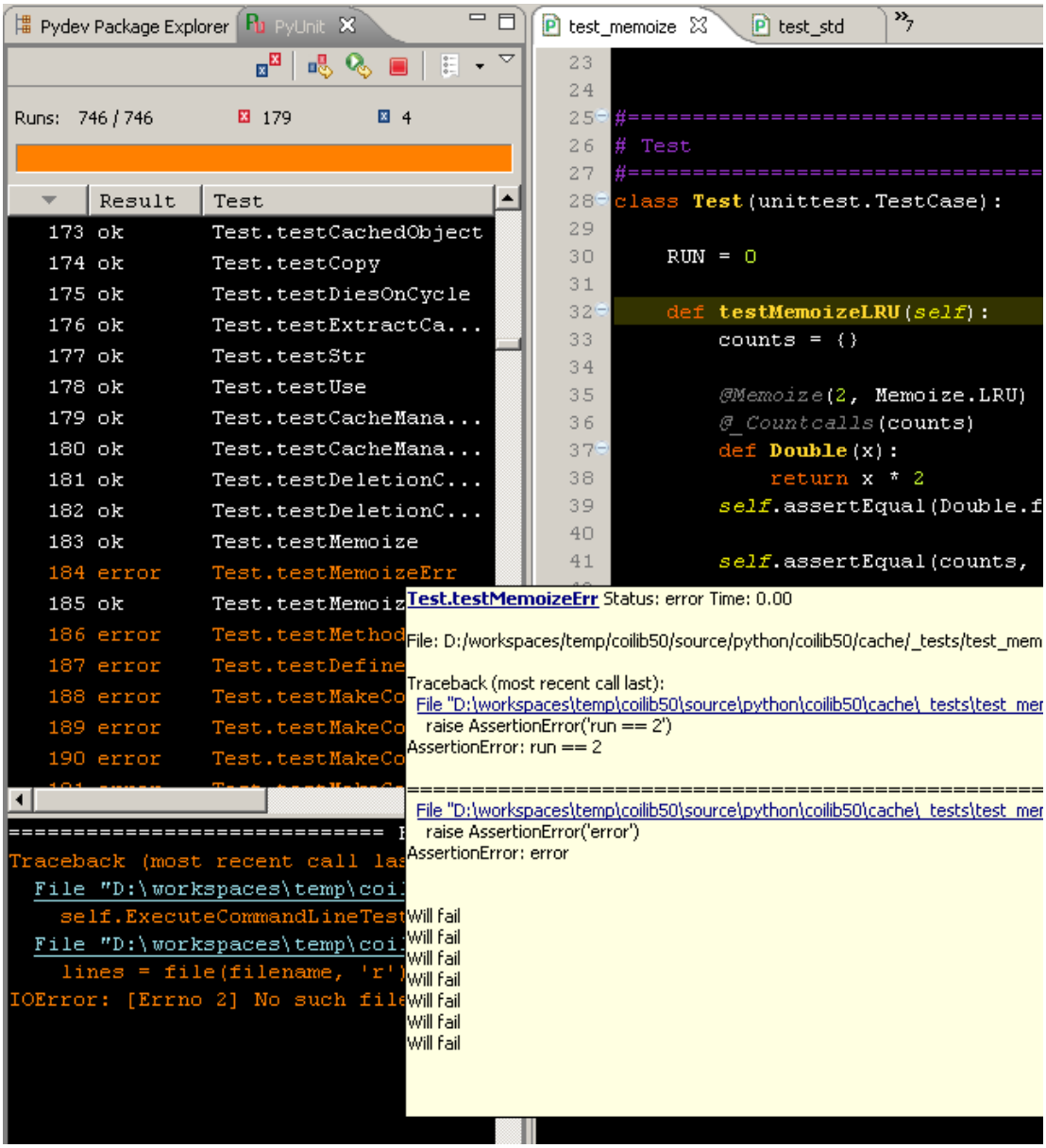
Para ejecutar un caso de prueba se usan las clases "TestRunner" que dan un entorno en el cual las pruebas se pueden desarrollar, la clase mas común para ejecutar las pruebas es "TextTestRunner", pues permite ver un reporte de los resultados de la prueba de forma textual.

6.6 Integración de PyUnit en el IDE Eclipse

En el plugin “Pydev and Pydev Extensions” para el IDE Eclipse, encontramos un interfaz para configurar nuestras pruebas unitarias. Si nos dirigimos a la ventana de configuración de Eclipse, encontramos dentro de la sección “Pydev” un apartado dedicado al módulo “PyUnit”. Aquí simplemente tenemos que asignar dónde se encuentra el módulo “PyUnit” en nuestro ordenador y ya estaría lista nuestra configuración. A partir de aquí es trabajo nuestro el realizar las pruebas unitarias para cada módulo que deseeamos probar su comportamiento.

Una vez desarrolladas las pruebas unitarias, nuestro IDE nos proporciona una nueva vista para ejecutarlas, que por supuesto permite que la configuremos a nuestro antojo como el resto de vistas de la plataforma. A la hora e ejecutar nuestra script de prueba unitaria, le decimos a Eclipse que queremos ejecutar el script como prueba unitaria que es, y automáticamente entramos en la vista para pruebas unitarias. Podemos destacar de dicha

vista, que va ejecutando todos los test que le digamos y nos guarda un registro del códigoque ha sido porbado satisfactoriamente y en cual se han producido errores. En caso de error, nos lo notifica y nos facilita mucho encontrar los posibles errores. A continuación se presenta un pantallazo de la vista de Eclipse para PyUnit:



Dicho todo esto, solo queda evidenciar lo que facilita el trabajo a la hora de desarrollar un proyecto software la elección de un buen entorno de desarrollo, y en nuestro caso la plataforma Eclipse junto con Pydev integra todo lo necesario para un cómodo desarrollo.

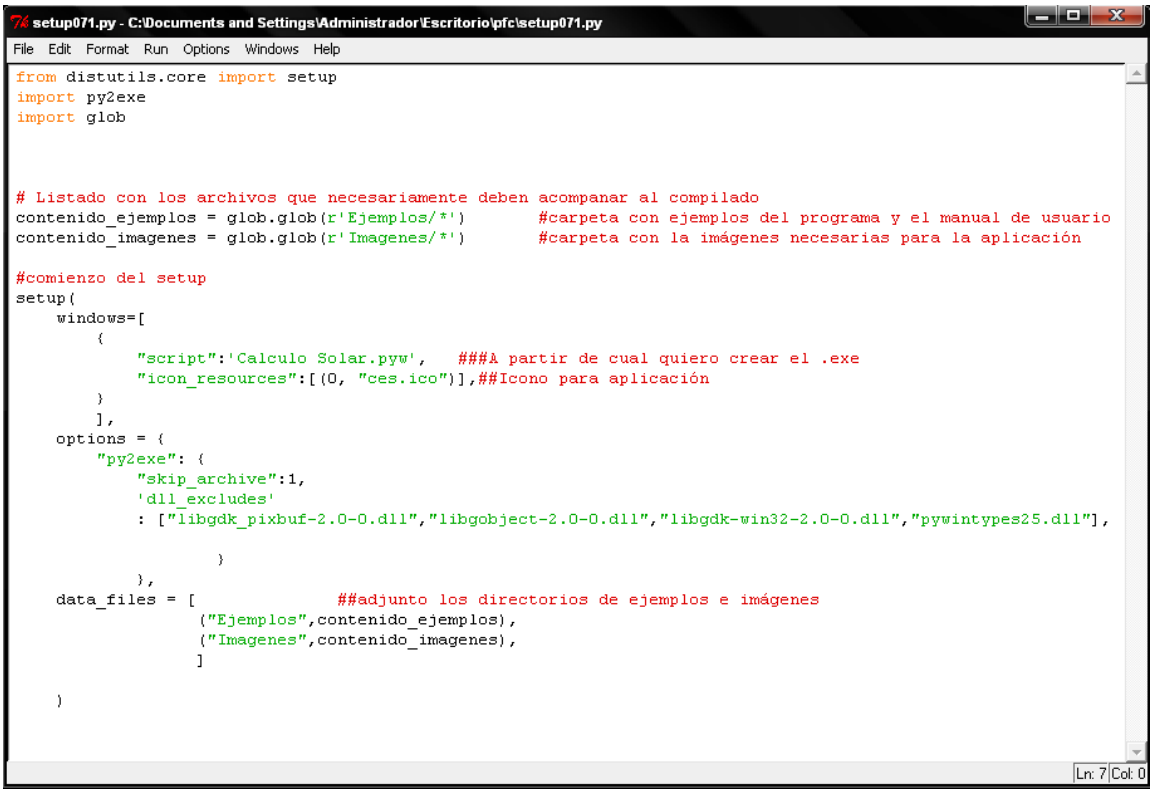
Capítulo 7: Empaquetado y distribución

Tras la implementación y período de pruebas de la aplicación, el último paso es empaquetar todos los ficheros binarios del programa y crear un instalador para que el usuario final no tenga que preocuparse más que de seguir un cómodo asistente y empezar a utilizar el programa. A lo largo de este capítulo describimos qué herramientas hemos utilizado para realizar la compilación del programa (Py2exe) y generar su instalador (Inno Setup).

7.1 Compilación con Py2exe

Dado que el lenguaje de programación utilizado para el proyecto es python, utilizaremos el módulo "py2exe" [8] para crear la distribución del programa. Éste módulo se encarga de juntar en un directorio todos los ficheros binarios y resto de recursos (imágenes, bases de datos…) necesarios para el correcto funcionamiento de la aplicación sin que el código fuente tenga que ser distribuido ni el usuario final tenga que instalarse las librerías de python necesarias para ejecutar la aplicación.

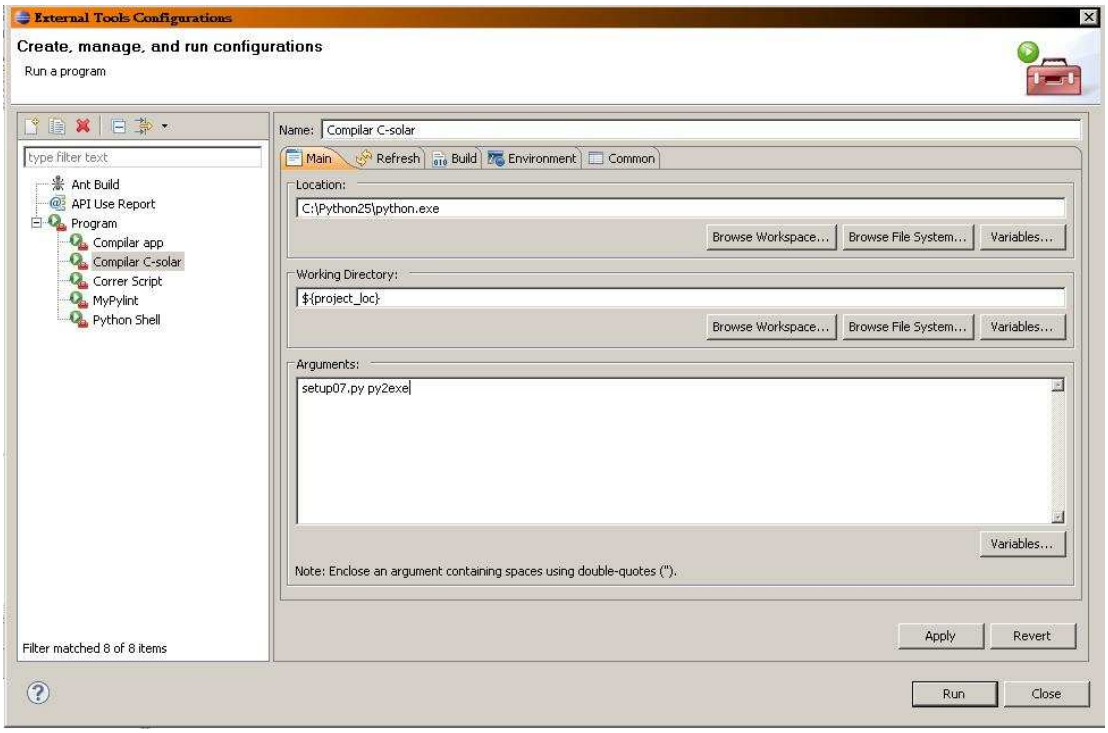
El funcionamiento del módulo "py2exe" es bastante sencillo, simplemente hay que descargar e instalar el paquete, crear un script de configuración y ejecutarlo. A continuación se presenta una captura con comentarios del script de configuración utilizado (setup07.py):



Una vez creado el script lo guardamos en el mismo directorio que tengamos el proyecto, abrimos el "Símbolo del Sistema" y nos dirigimos al directorio del proyecto. Con la orden

"python setup07.py py2exe", se genera el directorio "dist", donde encontraremos la versión compilada de nuestra aplicación y el ejecutable "Calculo Solar.exe".

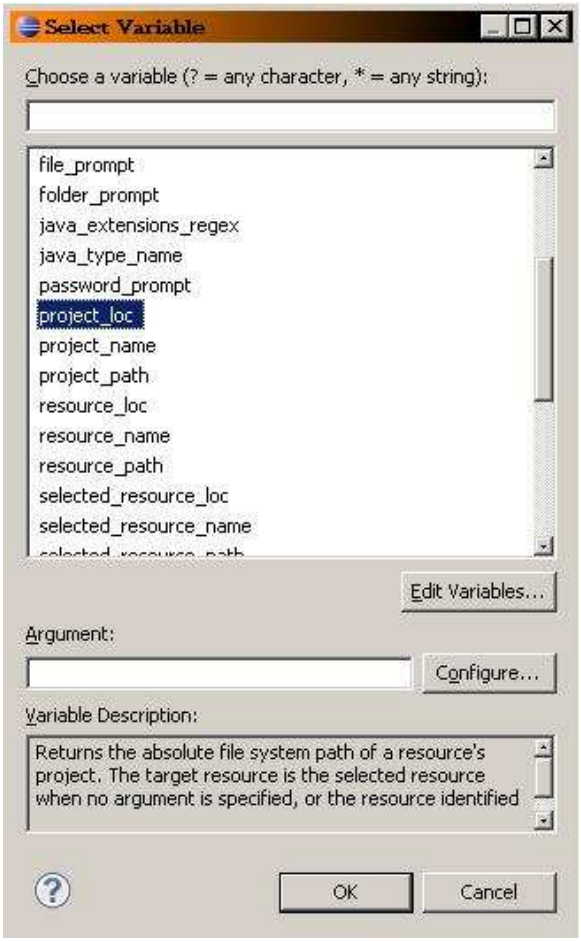
Llegados a este punto del proyecto, de nuevo el IDE eclipse nos vuelve a echar una mano con una de sus utilidades, se trata de la opción de incorporar aplicaciones externas. Echemos un vistazo a la ventana de configuración de las "External tools":



Como vemos en la pestaña de configuración debemos definir el nombre de la herramienta que queremos definir, la localización del programa de queremos correr, el directorio de trabajo sobre el que queremos aplicar nuestra herramienta externa y los argumentos.

Como nombre de nuestra herramienta externa hemos elegido "Compilar C-Colar", como es de suponer el nombre que elijamos es indiferente para el funcionamiento, simplemente ponemos uno que nos parezca apropiado. En los siguientes campos si que debemos tener más cuidado. Recordamos que la orden que debemos poner en el símbolo del sistema de Windows es "python setup07.py py2exe", por lo que el resto de campos deben componer esa orden. Por tanto en el campo localización debemos poner la ruta del intérprete de Python, en nuestro caso "C:\Python25\python.exe" y en los argumentos "setup07.py py2exe".

Por último, para el campo del directorio de trabajo recurrimos al cuadro de diálogo que se abre al pinchar en el botón "Variables" disponible para todos los campos:



Como podemos ver se trata de una serie de variables globales que Eclipse pone a nuestra disposición para hacer referencia a diferentes recursos de forma dinámica. Además nos ofrece una pequeña descripción en el mismo cuadro de diálogo para que sepamos a que se refiere cada una.

Ya que el fichero para compilar se encuentra en el mismo directorio que nuestro proyecto, elegimos la variable "${proyect_loc}" que como observamos en la descripción que nos ofrece Eclipse hace referencia a dicho directorio.

Y con esto ya tenemos configurada nuestra herramienta externa en la plataforma Elipse para ser utilizada en cualquier momento. Al igual que hemos configurado esta herramienta podemos agregar todas las que queramos para que nos ayuden a desarrollar nuestro proyecto.

7.2 Python y el problema de los ficheros compilados .pyc

El lenguaje Python es un lenguaje interpretado. Esto significa que el código fuente contenido en los ficheros .py es interpretado en el momento de ser ejecutado. Aunque, en realidad, el intérprete no ejecuta directamente el código fuente. El código fuente es "traducido" primeramente a un lenguaje intermedio o "bytecodes" que es una representación binaria del código fuente. Estos bytecodes residen en ficheros .pyc (código compilado). Una vez, compilado, el intérprete de Python lo que realmente ejecuta es este código intermedio.

A partir de los bytecodes contenidos en los ficheros .pyc es posible obtener el código fuente original (con mayor o menor precisión) a través de un proceso que se denomina decompilación. Una vez obtenido el código fuente (fichero .py), éste puede ser modificado e incluido posteriormente como parte de la aplicación original, sin necesidad de hacer ningún paso adicional.

Si distribuimos nuestra aplicación en formato compilado (ficheros .pyc), pueden ser decompilados fácilmente con herramientas públicamente disponibles y obtener así el código fuente original de una forma más o menos precisa. Dependiendo de la potencia del procedimiento de decompilación utilizado, el código fuente obtenido puede ser más o menos representativo del código fuente original. En cualquier caso, el código fuente obtenido puede servir para conocer la estructura interna del código. De esta forma se puede saber cómo funciona la aplicación, que es el primer paso para poder modificarla.

Hay que partir de la base de que cualquier software puede ser crackeado. La diferencia está en el esfuerzo que suponga crackear un software y el beneficio que se pueda obtener haciéndolo. Cuanto mayor sea el esfuerzo y menor el beneficio obtenido, menos sentido tiene el intentar crackear un software. Además hay que tener en cuenta que ninguna medida es completamente infalible.

Para solucionar este problema de seguridad, existen varias opciones, pero la más segura en la práctica es traducir las librerías sensibles a código C, y recompilar estas librerías de forma que el intérprete de Python las entienda.

Desde luego a estas alturas del proyecto resulta impensable tener que reprogramar las librerías que queramos proteger en código C, por lo que hemos optado por otra solución, que aunque perdiendo algo de eficiencia es perfectamente válida. Se trata de generar código C automáticamente desde el código Python ya implementado. Para conseguir nuestro propósito hemos utilizado la herramienta "Cython", que partiendo de ficheros con código fuente en lenguaje Python genera código C ofuscado con la misma funcionalidad que el original. Una vez obtenidos los ficheros de código C, compilamos las librerías con la herramienta "gcc" y las incorporamos a la aplicación.

7.3 Creando el instalador con Inno Setup

Partiendo de la versión compilada que acabamos de crear y una vez encriptadas las librerías que nos interasa, vamos a generar el instalador de la aplicación que distribuiremos a los usuarios finales. Hoy día hay muchas formas de crear instaladores pero nos hemos decantado por utilizar la aplicación "Inno Setup". "Inno Setup" es un potente generador de instaladores para Windows(c) gratuito, disponible en distintos idiomas, que emplea una interface grafica en la que el principal elemento es el Script, donde podemos definir multitud de opciones para nuestro instalador. Además dispone de un asistente con el que podemos generar de una forma sencilla el script que se adapte a nuestras necesidades. Además de lo ya dicho, esta herramienta tiene una extensa documentación de configuración de scripts, en la que podemos encontrar todo lo necesario para crear nuestro instalador. A continuación se presenta el script de configuración de "Inno Setup" utilizado para nuestra aplicación:

```
[Setup]

AppId={{39874249-F83D-4DDE-8651-9895ED5B14A6}

AppName=CSolar

AppVerName=CSolar v1.0

AppPublisher=FJT

DefaultDirName=C:\CSolar

DefaultGroupName=CSolar

LicenseFile=C:\Documents and
Settings\Administrador\Escritorio\pfc\licencia.txt

OutputBaseFilename=setupCSolar

Compression=lzma

SolidCompression=yes

ChangesASSociations=yes


[Languages]

Name: "english"; MessagesFile: "compiler:Default.isl"
```

```
Name: "spanish"; MessagesFile: "compiler:Languages\Spanish.isl"


[Tasks]

Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}";
GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked


[Files]

Source: "C:\Documents and
Settings\Administrador\Escritorio\pfc\dist\*"; DestDir:
"C:\CSolar"; Flags: ignoreversion recursesubdirs createallsubdirs

; NOTE: Don't use "Flags: ignoreversion" on any shared system
files


[Icons]

Name: "{group}\CSolar "; Filename: "C:\CSolar\CSolar.exe";
WorkingDir: "C:\CSolar"

Name: "{commondesktop}\CSolar "; Filename: "C:\CSolar\CSolar.exe";
Tasks: desktopicon  ;WorkingDir: "C:\CSolar"


[Registry]

Root: HKCR; Subkey: ".fjt"; ValueType: string; ValueName: "";
ValueData: "ArchivoCSolar"; Flags: uninsdeletekey

Root: HKCR; Subkey: "ArchivoCSolar"; ValueType: string; ValueName:
""; ValueData: "Archivo CSolar"; Flags: uninsdeletekey

Root: HKCR; Subkey: "ArchivoCSolar\DefaultIcon"; ValueType:
string; ValueName: ""; ValueData: "C:\CSolar\CSolar.exe,0"; Flags:
uninsdeletekey

Root: HKCR; Subkey: "ArchivoCSolar\shell\open\command"; ValueType:
string; ValueName: ""; ValueData: """C:\CSolar\CSolar.exe""
""%1"""; Flags: uninsdeletekey
```

```
[Run]

Filename: "C:\CSolar\Calculo Solar.exe"; Description:
"{cm:LaunchProgram,Calculo Solar }"; Flags: nowait postinstall
skipifsilent
```

Como vemos, el script se compone de diferentes partes separadas unas de otras, a continuación se presenta una breve descripción de cada una

Sección [Setup]:

Definimos las variables globales que "Inno setup" necesita para generar el instalador. Simplemente indicamos una "ID" para la aplicación así como su nombre, versión, directorio de trabajo por defecto, empresa que la crea…

Destacamos aquí la variable "LicenseFile", que debe estar asignada a la ruta de un fichero de texto. El contenido de dicho fichero será el que se muestre en el instalador como licencia del programa.

Sección [Languages]:

Definimos en qué idiomas queremos que esté disponible la instalación de la aplicación. En nuestro caso hemos elegido inglés y castellano.

Sección [Tasks]:

Aquí podemos definir diferentes tareas que debe realizar el instalador, en nuestro caso sólo le pedimos que deje al usuario la opción de decidir si quiere que se cree un icono del programa en el escritorio.

Sección[Files]:

Aquí debemos definir los ficheros a incluir en el instalador, en nuestro caso incluimos todo lo que hay en la carpeta "dist" generada con el módulo "py2exe".

Sección [Icons]:

En esta sección podemos definir los diferentes iconos que queremos que se asocien a nuestra aplicación. En nuestro caso sólo necesitamos uno.

Sección [Registry]:

Esta sección nos permite crear entradas del registro de Windows, algo muy útil y que en nuestro caso lo hemos utilizado para dos cosas:

1. Para que nos asocie la extensión de fichero (*.fjt) que utiliza la aplicación, al icono de la misma. De esta forma veremos con el icono de la aplicación todos los ficheros con esta extensión
2. Para que al hacer doble clic sobre un fichero con extensión "*.fjt" , el sistema operativo entienda que debe ejecutar nuestra aplicación pasándole como único argumento el fichero que ha sido clicado.

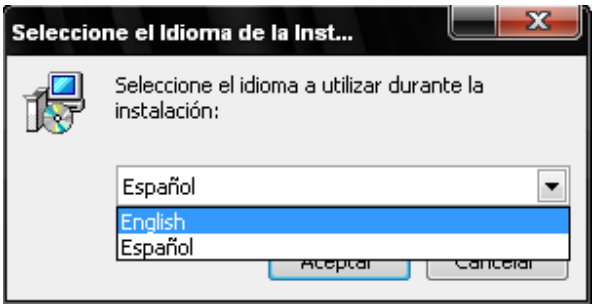
Sección [Run]:

Aquí podemos elegir si queremos que se ejecute algún programa al terminar la instalación. En nuestro caso dejamos que el usuario decida si desea o no correr la aplicación tras ser instalada.

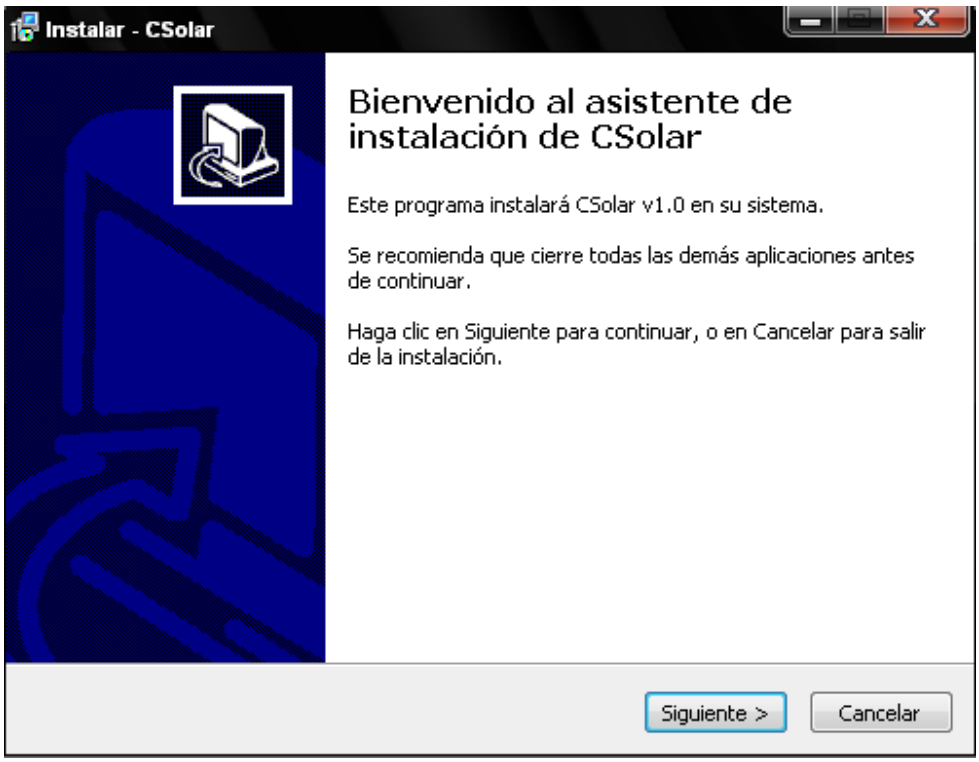
De todas las secciones utilizadas, solamente [Setup] y [Files] son obligatorias, es decir que "Inno Setup" nos crearía nuestro instalador con tan solo definir esas dos secciones. El resto son opcionales permitiendo de esta forma generar un instalador personalizado y ajustado a nuestras necesidades.

Una vez creado el script, lo ejecutamos directamente desde la aplicación "Inno Setup", elegimos el directorio de destino y en unos segundos se crea el instalador de nuestra aplicación. A continuación se presentan las capturas de las diferentes pantallas que tiene el instalador creado:

1. Selección de idioma

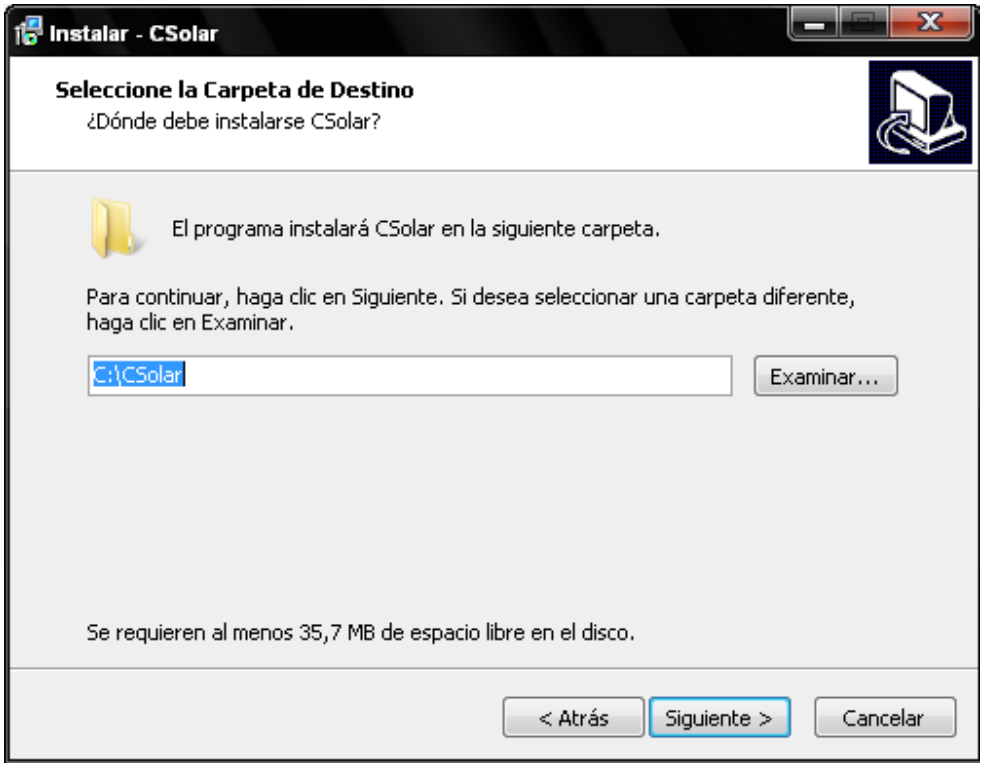


2. Pantalla bienvenida

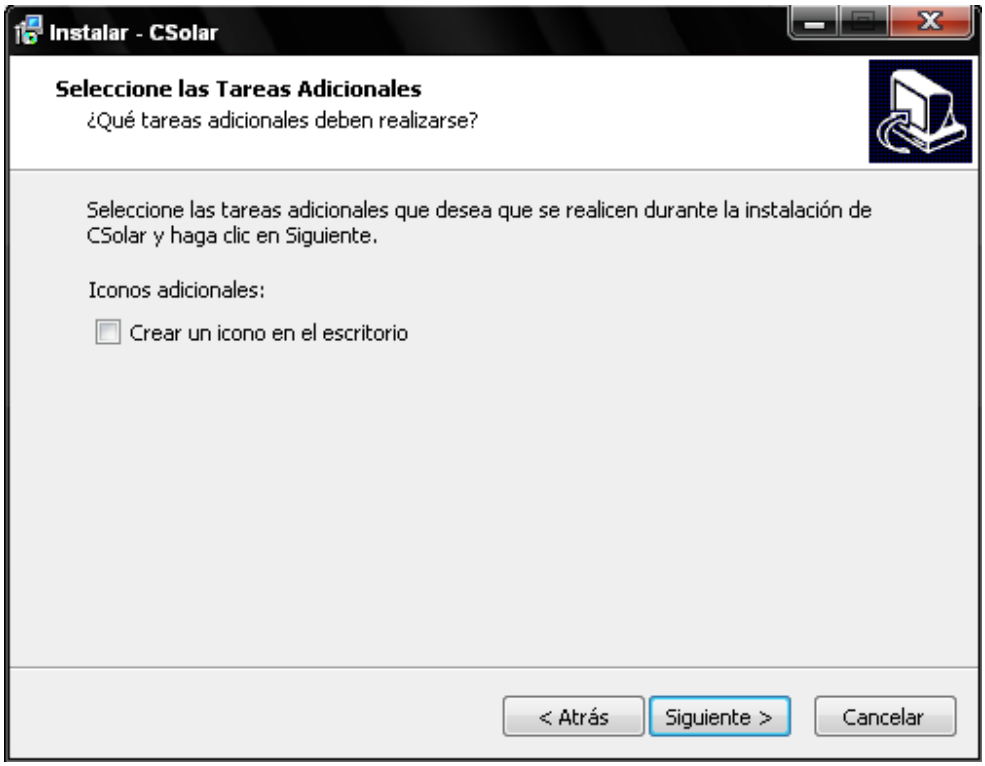


3. Acuerdo de licencia

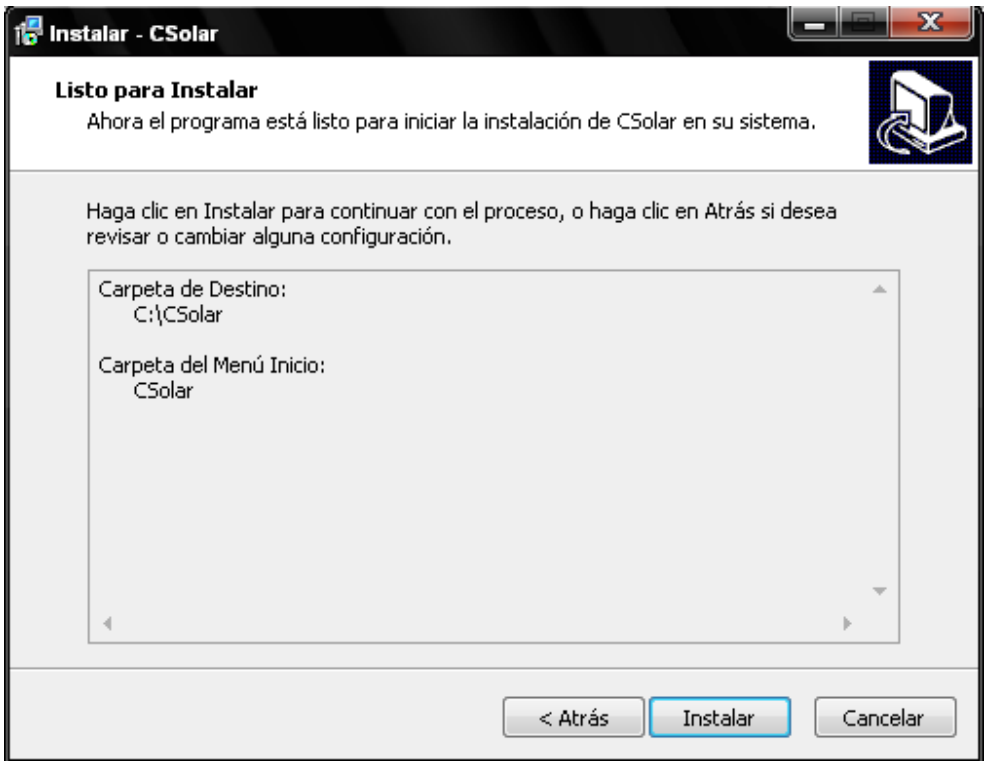
4. Selección de la carpeta de instalación



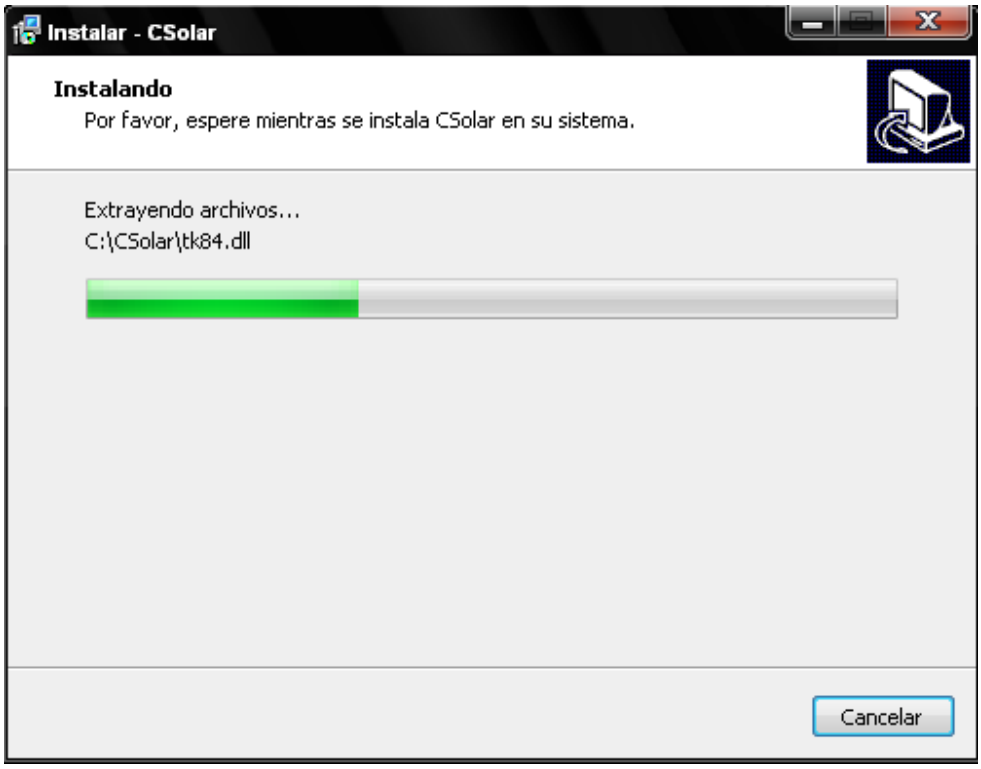
5. Tareas Adicionales



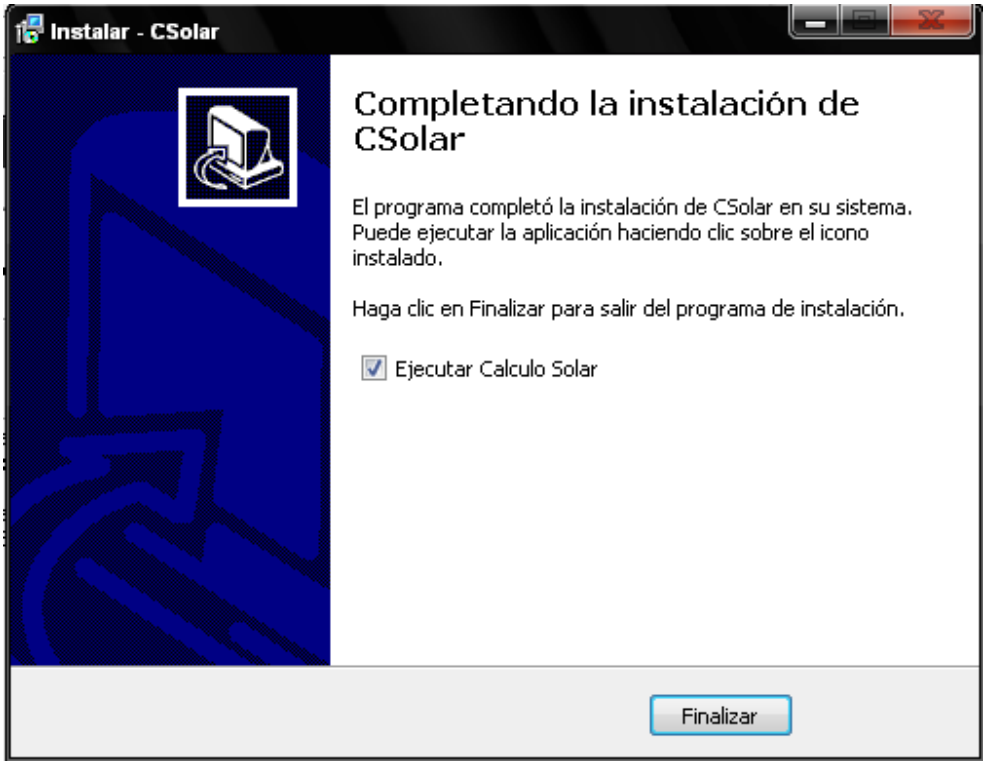
6. Confirmando la instalación



7. Instalando…



8. Fin de la instalación



Capítulo 8: Conclusiones y líneas futuras

En este último capítulo de proyecto analizaremos la conclusiones finales que tenemos una vez el proyecto se ha terminado y las posibles líneas futuras que quedan abiertas para seguir trabajando con el proyecto.

8.1 Conclusiones

En primer lugar me gustaría decir que se ha cumplido el objetivo principal del proyecto, que era realizar una aplicación sencilla para calcular las instalaciones solares térmicas que necesita un edificio de nueva construcción en España para cumplir las exigencias del Código Técnico de la Edificación de 2007.

Al comenzar el proyecto, teníamos ante nosotros el documento HE-4 del citado Código Técnico de la Edificación, que recoge toda la normativa que regula las instalaciones solares térmicas que todo edificio nuevo debe tener para cumplir con la exigencia de producción mínima de agua caliente sanitaria que el edificio en cuestión consume al cabo del año. Como se indica en dicho documento, no basta con hacer un cálculo aproximado, si no que además hay que tener en cuenta las posibles pérdidas por sombras de obstáculos remotos.

Con el desarrollo del cálculo e interfaces gráficas hemos obtenido una serie de módulos que implementan el problema y ayudan al usuario final a realizar una tarea que a mano resulta bastante costosa. Además gracias a la metodología orientada a objetos desarrollada los módulos obtenidos son reutilizables para líneas futuras de trabajo con el presente proyecto.

Personalmente, el desarrollo de este proyecto me ha servido para aprender y crecer un poco como futuro profesional de la informática. He aprendido un nuevo lenguaje de programación y he perfeccionado en desarrollo de pequeñas aplicaciones software. Además he tenido que ingeniarmelas para resolver problemas complejos de cálculo que no he visto en la carrera. Por último debo destacar que sin la ayuda de algunos compañeros habría sido imposible terminar este proyecto, por lo que debo agradecer su paciencia para explicarme lo que no entendía. Supongo que gracias a esto último también he aprendido a trabajar un poco mejor en equipo.

8.2 Líneas futuras

La verdad que una vez terminado el proyecto quedan unas cuantas líneas futuras abiertas para continuar trabajando. La primera de ellas se menciona en esta misma memoria, y es adaptar la aplicación desarrollada para que pueda utilizarse en otros sistemas operativos como Linux o Mac OS X. Y la verdad que este primer paso es bastante sencillo, ya que todas las librerías de programación que se han utilizado son multiplataforma, con lo que resolviendo el problema del cambio de codificación (cp1252 a utf-8) quedaría resuelto.

Por otro lado, como se comenta en la presentación del presente proyecto, ha sido desarrollado en la empresa Natural Climate Systems S.A (MIYABI), que fue adjudicataria de dos concursos del Ministerio de Industria para desarrollar los "Procesos simplificados de calificación energética de edificios nuevos" y los "Procesos simplificados de calificación energética de edificio existentes."

La aplicación "CES", es el proceso simplificado de calificación energética de edificios nuevos con la que concursa MIYABI para que se haga método oficial. Por lo que incluir la funcionalidad de este proyecto en la aplicación es otra tarea de obligado cumplimiento, ya que como sabemos desde 2007 todos los edificios disponen de instalación solar térmica y evidentemente esto afecta a su calificación final.

La aplicación "CEX", es el proceso simplificado de calificación energética de edificios existentes con la que concursa MIYABI para que se haga método oficial. Evidentemente los edificios ya existentes no disponen de instalación solar térmica, pero podemos reaprobechar el módulo de cálculo de pérdidas por sombras. En función de la radiación solar que tiene un edificio, su calificación cambia. Resulta obvio que un piso orientado a sur cnsume menos calefacción que otro de las mismas características pero orientado a norte. De la misma manera, dos pisos iguales orientados a sur tendrán consumos de calefacción diferentes si a uno le pega el sol todo el día y al otro le hace sombra un edificio que tiene en frente. Por lo tanto, podemos reutilizar el interfaz gráfico y el cálculo de porcentajes cubiertos en cada casilla de la carta solar para ver el perfil de sombras que afecta al edificio que se va a calificar.

Bibliografía

[1] Python – http://www.python.org

[2] wxPython - http://www.wxpython.org

[3] Python Polygon Library - http://www.pypolygon.com

[4] Python Image Library - http://www.pythonware.com/PIL

[5] Numpy - http://numpy.scipy.org

[6] PyLint - http://www.logilab.org

[7] PyUnit - http://docs.python.org/library/unittest.html

[8] Py2exe - http://www.py2exe.org

[9] Wikipedia - http://www.wikipedia.org

[10] Martelli, Alex (2007) *Python. Guía de Referencia*, Anaya

[11] Edwards, Brian (2005) *Guía básica de la sostenibiliidad*, G.Gil

[12] Yañez, Guillermo (1982) *Energía solar, edificación y clima*, Ministerio de obras públicas y urbanismo

[13] *Código Técnico de la Edificación* (2007), Ministerio de obras públicas y urbanismo

[14] Eclipse-Pydev- http://www.pydev.org

Anexos

Documentos básicos para la comprensión y realización del proyecto:

1. Documento HE-4 del Código Técnico de la Edificación (2007)
2. PEP 8 – Style Guide for Python Code

C-SOLAR

Aplicación para el cálculo de instalaciones solares térmicas en edificios residenciales

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

CONTEXTO

- Natural Climate Systems S.A (MIYABI)
- Arquitectura bioclimática
- Asesoría energética

CONTEXTO

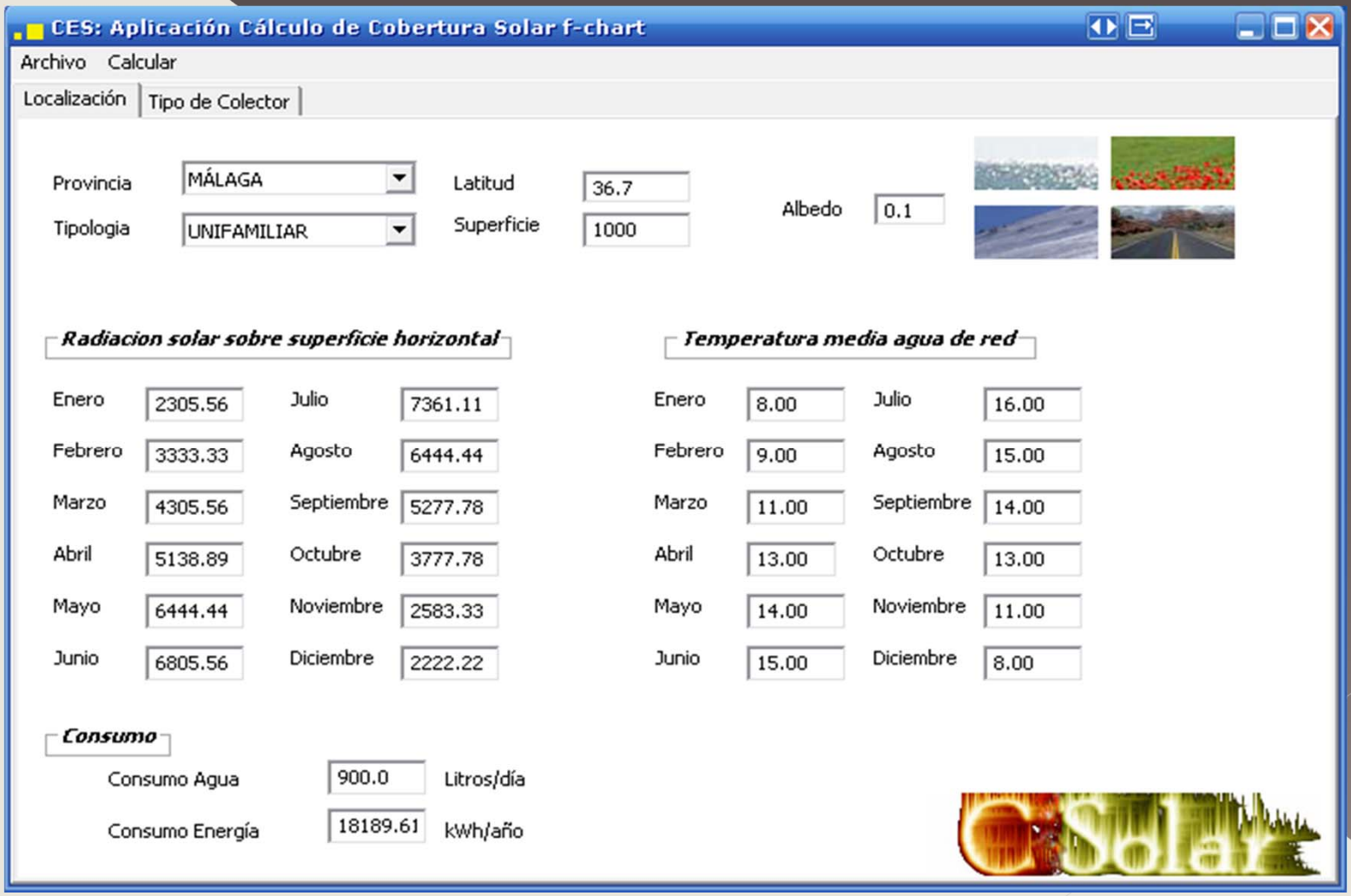
- **Código Técnico de Edificación (CTE) de 2007:** obliga a calificar energéticamente los todos los edificios nuevos que se construyan-

- **Documento HE-4:** exige que parte del agua caliente sanitaria que consumen los edificios nuevos, se produzca con energías renovables.

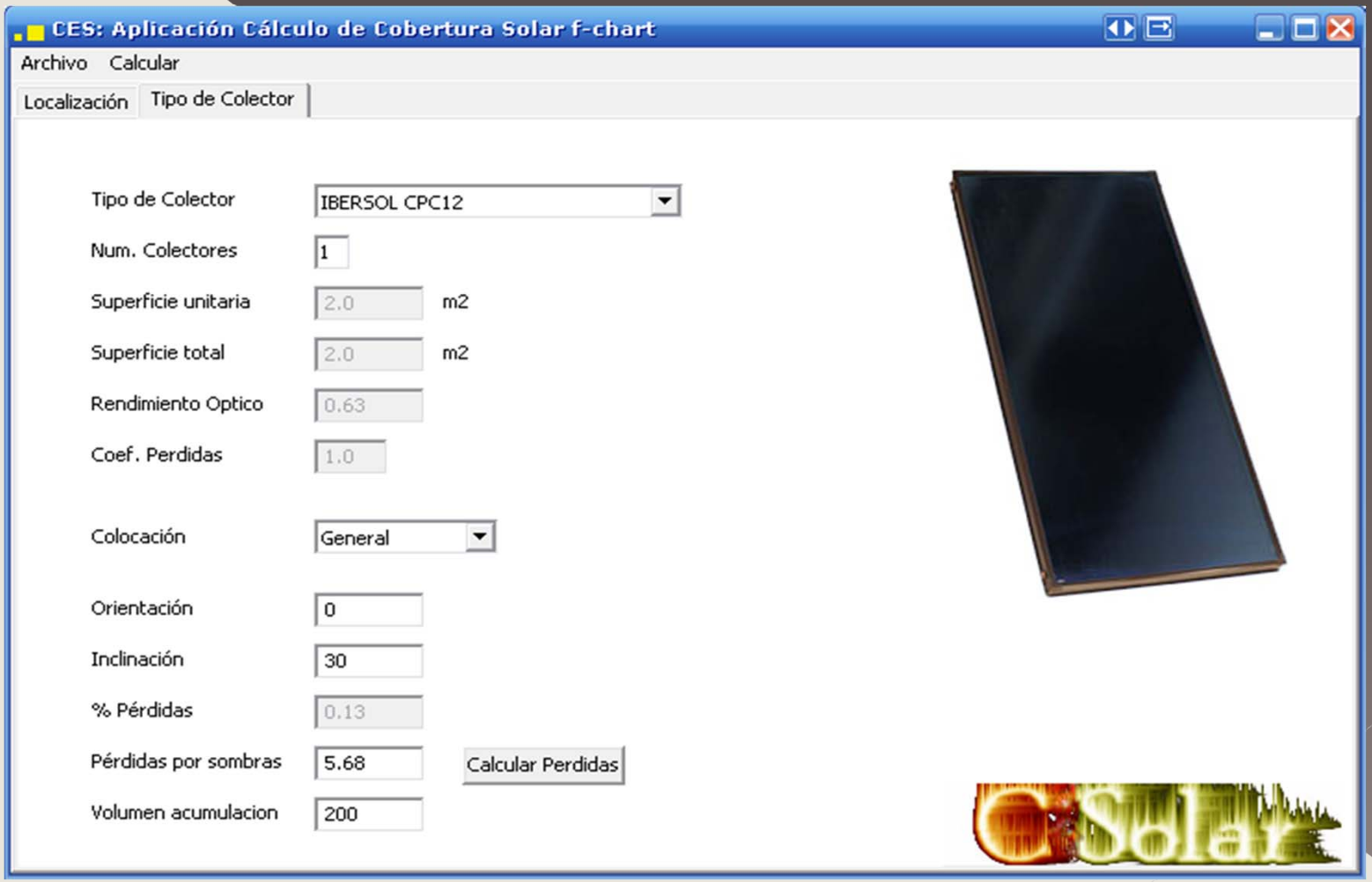
REQUISITOS generales

- Aplicación de escritorio aislada
- Fácil de utilizar y que simplifique el trabajo
- Claridad y simplicidad en la introducción de datos
- Facilidad de interpretación de los resultados

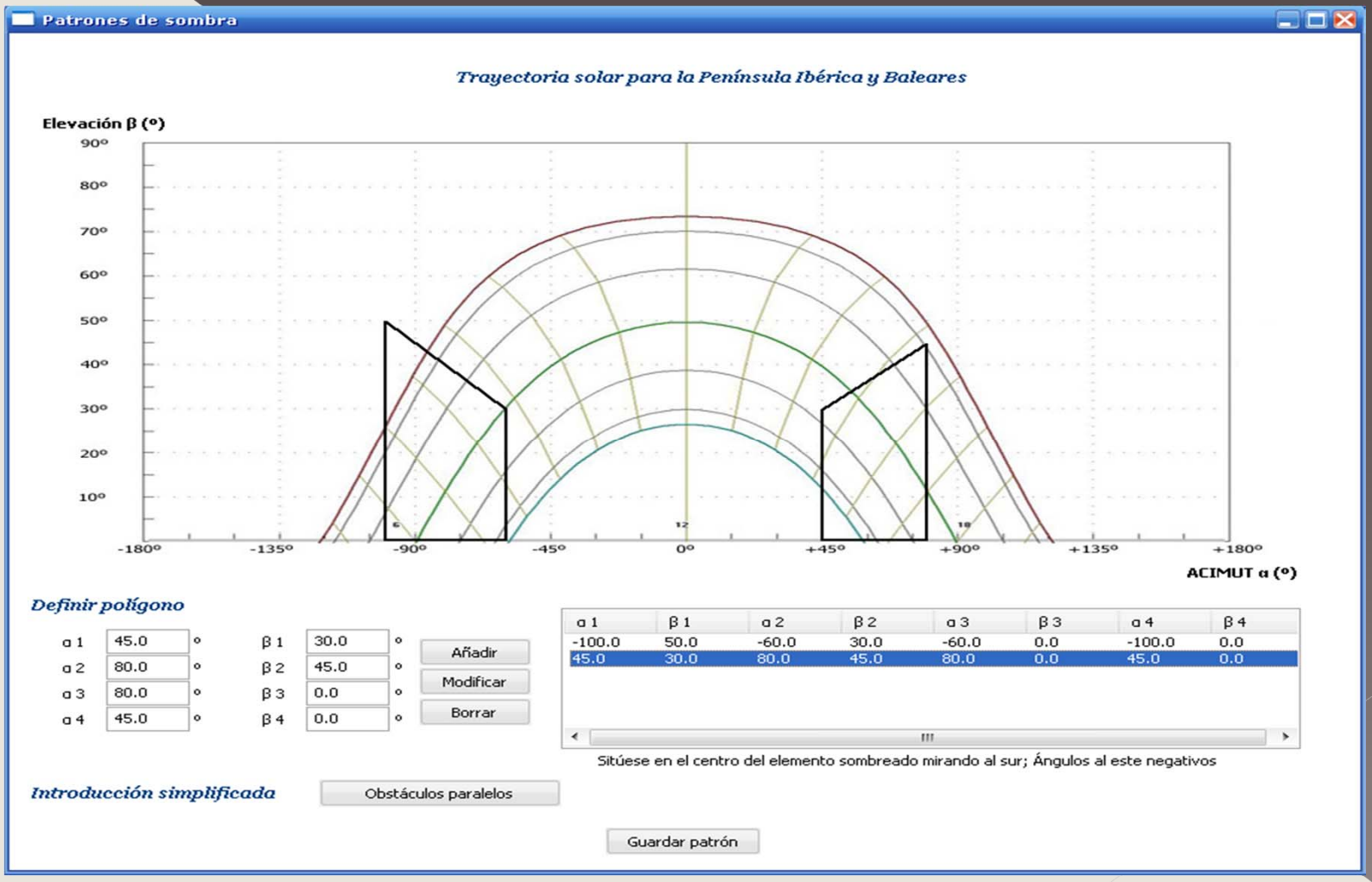
REQUISITOS funcionales



REQUISITOS funcionales



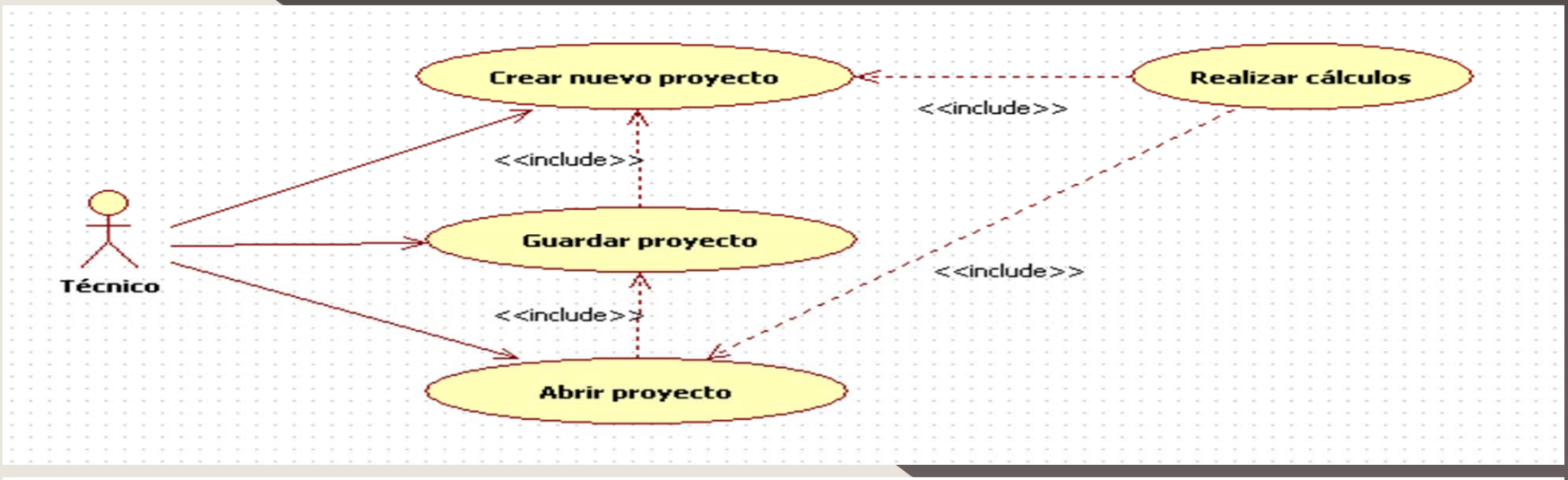
REQUISITOS funcionales

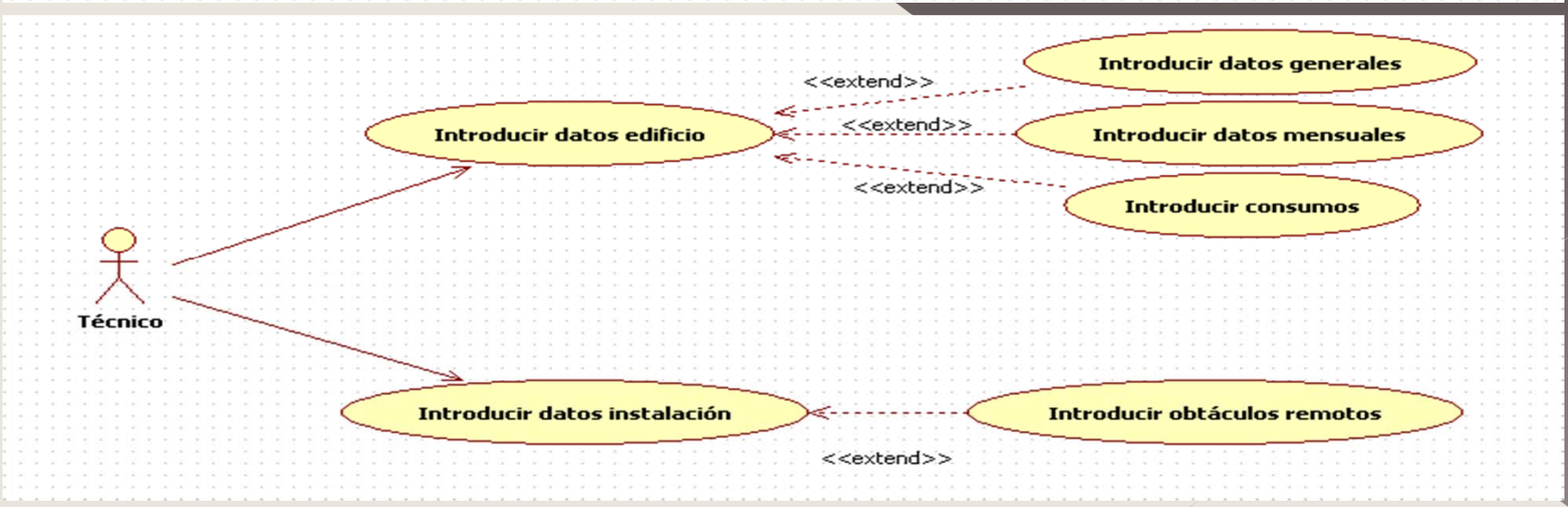


REQUISITOS técnicos

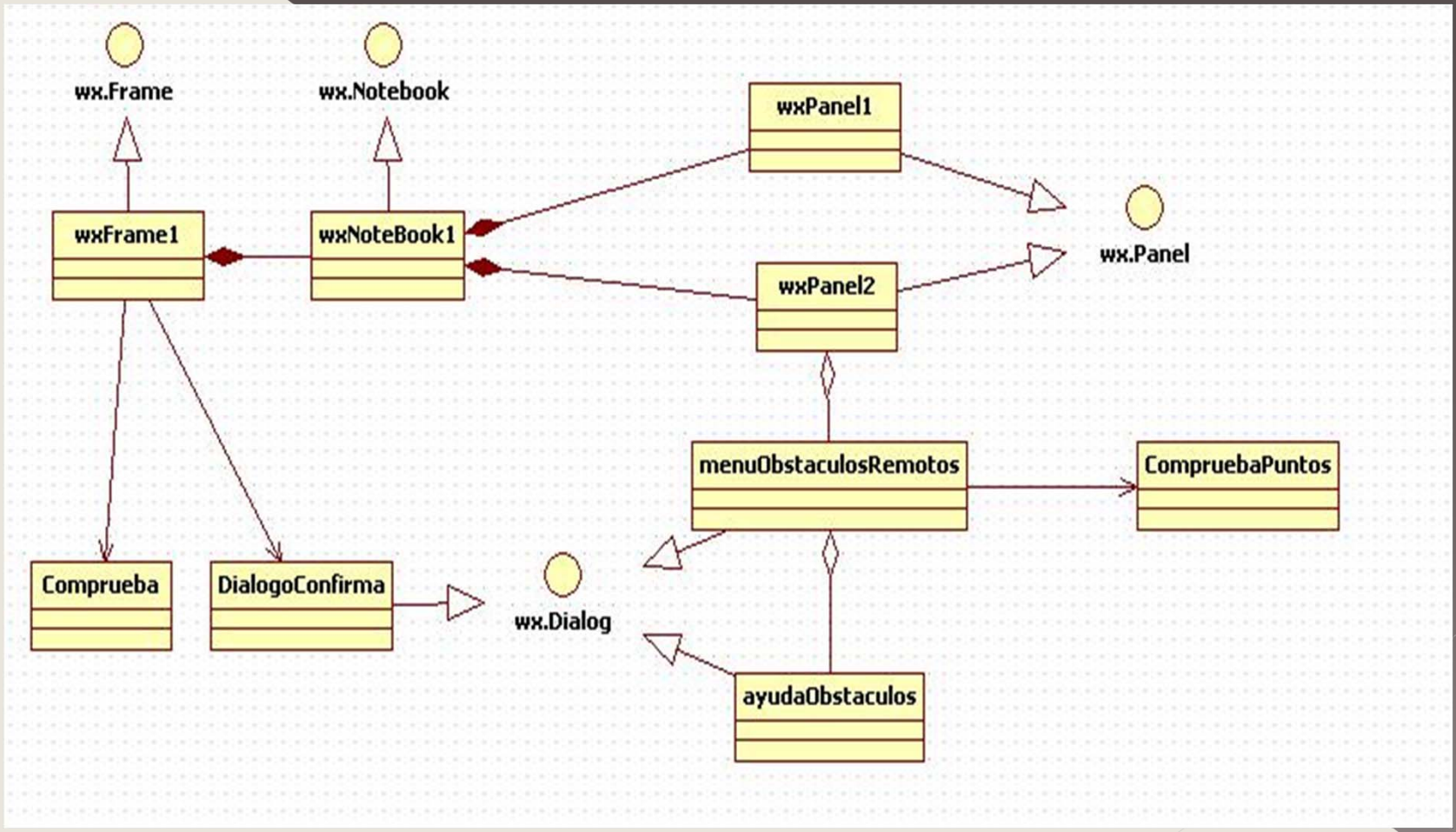
- Capacidad de análisis y comprensión
- Desarrollo de cálculos
- Desarrollo de métodos de representación

ANÁLISIS





DISEÑO



IMPLEMENTACIÓN

- Python 2.5
  - - lenguaje interpretado, multiparadigma, tipado dinámico
  - - librerías multiplataforma
  - - filosofía "pythonica"
- wxPython
  - - heredero de la biblioteca gráfica "wxWidgets" de C++
  - - multiplataforma
- IDE y GUI BUILDER
  - - Boa Constructos
  - - Eclipse + Pydev

IMPLEMENTACIÓN

- Cálculo F-chart
  - - cálculo de la cobertura de un sistema solar
  - - cálculo suficientemente exacto para largas estimaciones
  - - en todo tipo de edificios, mediante captadores solares planos

$$f = 1{,}029\,D1 - 0{,}065\,D2 - 0{,}245\,D1^2 + 0{,}0018\,D2^2 + 0{,}0215\,D1^3$$

IMPLEMENTACIÓN

Contribución solar mínima

Tabla 2.1. Contribución solar mínima en %. Caso general

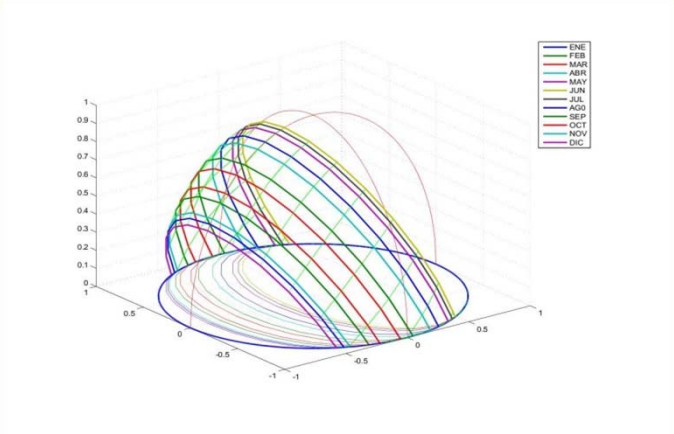
Demanda total de ACS del edificio (l/d)	Zona climática I	II	III	IV	V
50-5.000	30	30	50	60	70
5.000-6.000	30	30	55	65	70
6.000-7.000	30	35	61	70	70
7.000-8.000	30	45	63	70	70
8.000-9.000	30	52	65	70	70
9.000-10.000	30	55	70	70	70
10.000-12.500	30	65	70	70	70
12.500-15.000	30	70	70	70	70
15.000-17.500	35	70	70	70	70
17.500-20.000	45	70	70	70	70
> 20.000	52	70	70	70	70

Límites de pérdidas

Tabla 2.4 Pérdidas límite

Caso	Orientación e inclinación	Sombras	Total
General	10 %	10 %	15 %
Superposición	20 %	15 %	30 %
Integración arquitectónica	40 %	20 %	50 %

IMPLEMENTACIÓN

- **Cálculo de pérdidas de radiación solar por sombras**
  - - Proyección cilíndrica
  - - Difícil interpretación
  - -Conceptos : acimut y elevación
- 

IMPLEMENTACIÓN

- Carta solar

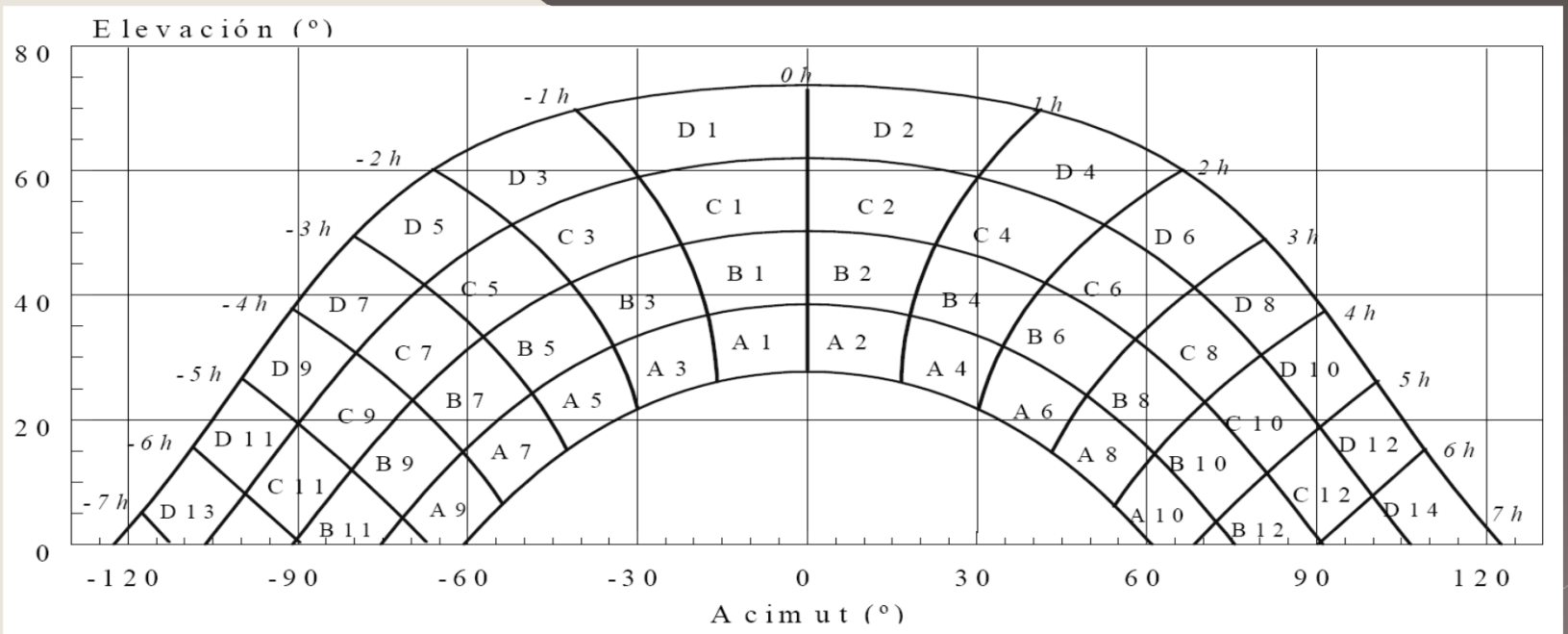


Figura 3.4 Diagrama de trayectorias del sol

Pérdidas por sombras = $\sum Pc \cdot Cc$

IMPLEMENTACIÓN

- Tratamiento fotográfico
  - - Imagen a 256 colores (1 color por casilla)
  - - Introducción de polígonos
  - - Histograma
  - - Máscara de una imagen

Análisis de Código

- Herramientas de análisis estático
- Pylint
  - - Revisiones básicas
  - - Revisión mediante inferencia de tipos
  - - Revisión de variables
  - - Revisión de clases
  - - Revisión de diseño
  - - Revisión de imports
  - - Revisión de formato
  - - …

PRUEBAS

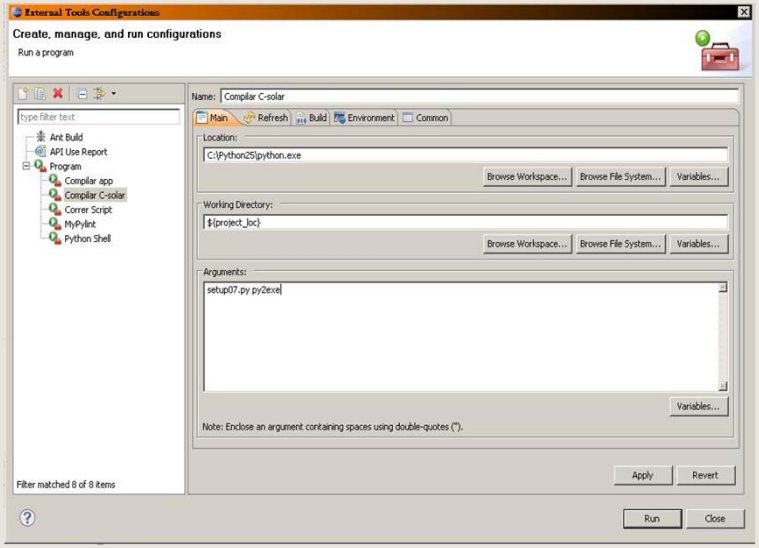
- Pruebas del software
  - - pruebas unitarias
  - - pruebas de integración
  - - …
- PyUnit
  - - framework oficial para hacer pruebas unitarias en Python
  - - en la librería estándar
  - - pruebas de igualdad, de excepciones, de fallo, de cordura…
  - - caso de prueba → herencia de "TestCase "

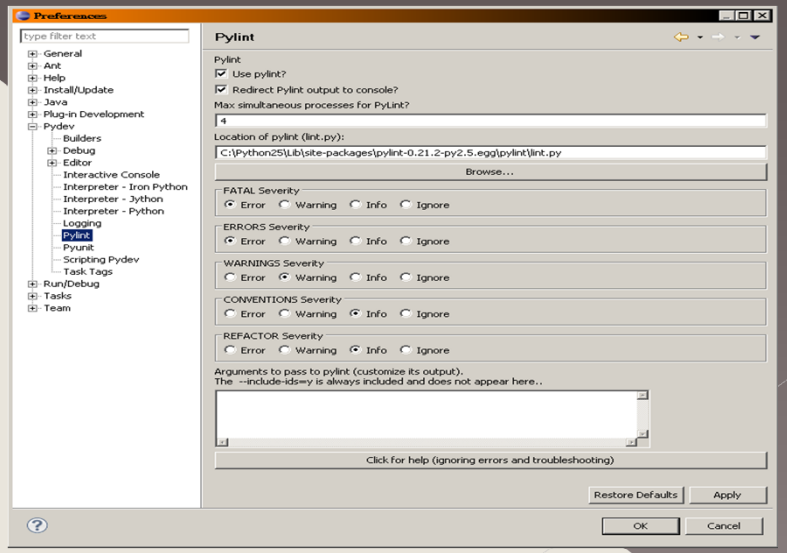
DISTRIBUCIÓN

- Py2exe
  - - Recopila recursos, los agrupa y genera un .exe
  - - Problemas de seguridad con los ficheros compilados
  - - Librerías en código C: "Cython"
- Inno Setup
  - - Crea instalador de la aplicación
  - - Script de configuración
  - - Configuración con Pascal
  - - Resultado: instalador amigable para el usuario

ENTORNO DE TRABAJO

- Eclipse
  - - Permite integrar todas las herramientas explicadas
  - - Pydev + Pydev Extensions + External Tools





CONCLUSIONES

- Hemos cumplido el objetivo inicial
- Cálculo bastante exacto que ahorra trabajo al usuario final
- Crecimiento personal
  - - Nuevo lenguaje de programación
  - - Nuevas herramientas
  - - Más capacidad creativa

LÍNEAS FUTURAS

- Integración del sistema en CES
- Reutilización del módulo de cálculo de pérdidas por sombras
- Integración en CEX
- Adaptar para otros sistemas operativos