UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Extensión de un simulador del hogar inteligente para incorporar mecanismos de interacción humano-sistema

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Scherle Carboneres, Fabian

Tutor/a: Albert Albiol, Manuela

Cotutor/a externo: GIL PASCUAL, MIRIAM

CURSO ACADÉMICO: 2021/2022

Resumen

Gracias a las investigaciones y al desarrollo en los campos de la inteligencia artificial y del aprendizaje automático, existen servicios automatizados que nos ayudan en las actividades de la vida cotidiana mediante el análisis de conjuntos de datos. Tal es su auge que hasta en nuestros hogares existen dispositivos que incorporan estos mecanismos, sin embargo estos sistemas requieren de un entrenamiento con conjuntos de datos de grandes volúmenes. Para su generación destacan dos posibilidades, la recopilación mediante sensores en un entorno real y la obtención de datos con herramientas de simulación.

Tomando como base la segunda aproximación para la generación de conjuntos de datos, nace el propósito del presente trabajo, analizar y ampliar un simulador de casas inteligentes, específicamente *OpenSHS*. La extensión se basa en la incorporación de mecanismos humano-sistema, de los cuales se incluyen la creación de una interfaz gráfica de usuario para la selección de las diferentes herramientas del simulador, la adición de un entorno simulado, la integración de un asistente virtual, una pizarra para mostrar mensajes de otros dispositivos y un mapa interactivo que representa la vista frontal de la casa.

Por último, para que el resultado del trabajo tenga un mayor impacto se genera una versión distribuible para *windows* y *linux*, y se procede a solicitar la adición de los cambios al repositorio original de *OpenSHS* con el propósito de que los colaboradores puedan mejorar el simulador e integrar los avances.

Palabras clave: openshs, simulador, casa, inteligente, python, blender.

Abstract

Thanks to the research and development in the fields of artificial intelligence and machine learning, there are automated services that help us in the activities of daily living by analyzing data sets. Such is its popularity that even in our homes there are devices that incorporate these mechanisms, however these systems require training with large volume data sets. For its generation, two possibilities stand out, the collection of data by sensors in a real environment and the collection of data with simulation tools.

The purpose of this work is based on the second approach: the generation of data sets, analyzing and improving a smart home simulator, specifically *OpenSHS*. The improvement is based on the incorporation of human-system mechanisms, which include the creation of a graphical user interface for the selection of different simulator tools, the addition of a simulated environment, the integration of a virtual assistant, a whiteboard to display messages from other devices and an interactive map that represents the front view of the house.

Finally, in order to make the results of the work have a greater impact, a distributable version is generated for *windows* and *linux*, and the addition of the changes to the original *OpenSHS* repository is requested so that collaborators can improve it and integrate the advances.

Keywords : openshs, simulator, home, smart, python, blender.

Índice de contenido

Índice de figuras

1. Introducción

El tema de este trabajo surge como derivación de un proyecto nacional de investigación, desarrollo e innovación (*I+D+i*), entre cuyos objetivos se encuentra ayudar a los diseñadores de sistemas autónomos con participación del humano a construir soluciones que se adecúen a las necesidades de los usuarios. Por ende, busca conseguir una integración humano-sistema personalizada y adaptativa en tiempo de ejecución.

Entre los dominios de aplicación de la investigación realizada en el proyecto mencionado se encuentra el *Hogar Digital*. En este dominio se buscan simuladores del hogar que permitan validar las soluciones construidas. A su vez, uno de los simuladores que puede servir para ese propósito, es *OpenSHS*. Sin embargo, para que este simulador pueda ser utilizado es necesario extenderlo para poder incluir mecanismos de interacción adicionales.

De esta forma, en el presente trabajo se propone analizar y extender un simulador con la capacidad de obtener datos útiles en el dominio expuesto. Específicamente, en esta sección se introduce la idea y la necesidad del trabajo, se describen los objetivos y se presenta la metodología utilizada para el desarrollo del mismo.

1.1. Motivación

En los últimos años se han visto significativos avances en los campos de la inteligencia artificial, domótica e internet de las cosas, un ejemplo muy conocido son los asistentes de voz o altavoces inteligentes como *Google Home* o *Amazon Echo*. Sus aplicaciones en el dominio del hogar han dado como resultado lo que hoy en día se conoce como casas inteligentes, es decir, hogares que disponen de servicios automatizados que utilizan sistemas capaces de aprender mediante el análisis de grandes conjuntos de datos obtenidos por mecanismos o dispositivos de interacción humano-sistema.

La implementación de algoritmos de aprendizaje automático requiere de un gran volumen de conjuntos de datos y que a su vez sean representativos en los escenarios capturados de un hogar inteligente (Alshammari et al., 2017). Esta premisa es indispensable para poder entrenar los algoritmos y a partir de ese punto obtener modelos que ofrezcan resultados con un porcentaje de error significativamente pequeño según los requisitos del sistema.

Una solución para generar conjuntos de datos de forma rápida y eficiente sin tener que montar un sistema real, consiste en el uso de simuladores, en este caso de casas inteligentes. Entre estos simuladores se encuentra *OpenSHS* (Alshammari et al., 2017), el cual es de uso gratuito, de código abierto y utiliza dos de las tecnologías más populares en el mundo de la programación y el diseño *3D* que son *Python* y *Blender*.

A título personal este trabajo es considerado un desafío para adquirir conocimientos y experiencia en el lenguaje de programación de *Python* utilizado en varias asignaturas de la rama de Computación, integrada en la carrera de Ingeniería Informática.

1.2. Objetivos

El presente trabajo tiene como objetivo extender el simulador de generación de datos para casas inteligentes *OpenSHS*. Al margen de este objetivo principal, como objetivo personal se pretende mejorar conocimientos en el lenguaje de programación *Python* y en la herramienta de diseño y modelado *3D* de *Blender*.

Para el objetivo planteado es necesario realizar una serie de pasos:

- Indagar y sentar las bases para el uso de las tecnologías y entornos en las que fue construido el simulador.
- Realizar un análisis del simulador.
- Plantear las especificaciones y requisitos que serán usados para la extensión que se quiere realizar sobre al simulador.
- Implementar los requisitos planteados.
- Validar las extensiones realizadas.
- Solicitar un *push request* al repositorio original de *OpenSHS*.

1.3. Metodología

Se evaluó la idea de utilizar una metodología ágil como *SCRUM*, sin embargo finalmente se decidió no utilizar una metodología en específico, pero sí tener en cuenta algunas prácticas con un enfoque similar siendo más flexible por las siguientes razones:

- Las metodologías ágiles son ideales para equipos de desarrollo y el presente trabajo es individual.
- El simulador carece de *frameworks* e implementaciones para la ejecución de pruebas.

Cada requisito se dividió en tareas anotadas en una pizarra en la plataforma de *Trello* tal y como se puede ver en la Figura 1. Por ende, una vez listadas las tareas a implementar, estas se ordenaron acorde a la secuencia de la lista de requisitos.

Una vez ordenadas las tareas, se establecieron las dependencias entre ellas y cuando fue posible se desarrollaron en paralelo aquellas que no se relacionaban directamente. De esta forma resultaría más eficiente el desarrollo y permitiría que se lleven a cabo un mayor número de validaciones en menos tiempo.

Por otra parte, la pizarra contiene cuatro fases en el que se especifica el ciclo de las actividades:

- **Pendiente**: tareas que se debían llevar a cabo para finalizar el proyecto, ordenadas por prioridad.
- **Implementando**: acciones que se estaban llevando a cabo.
- **Bloqueado**: tareas que están en proceso de implementación pero exigen más tiempo dedicado que otras tareas con mayor prioridad o que para continuar su desarrollo dependen de la culminación o avance de otras actividades.
- **Finalizado**: Tareas que han sido completadas, si se requiere algún cambio se crea una nueva tarea para dicha mejora.

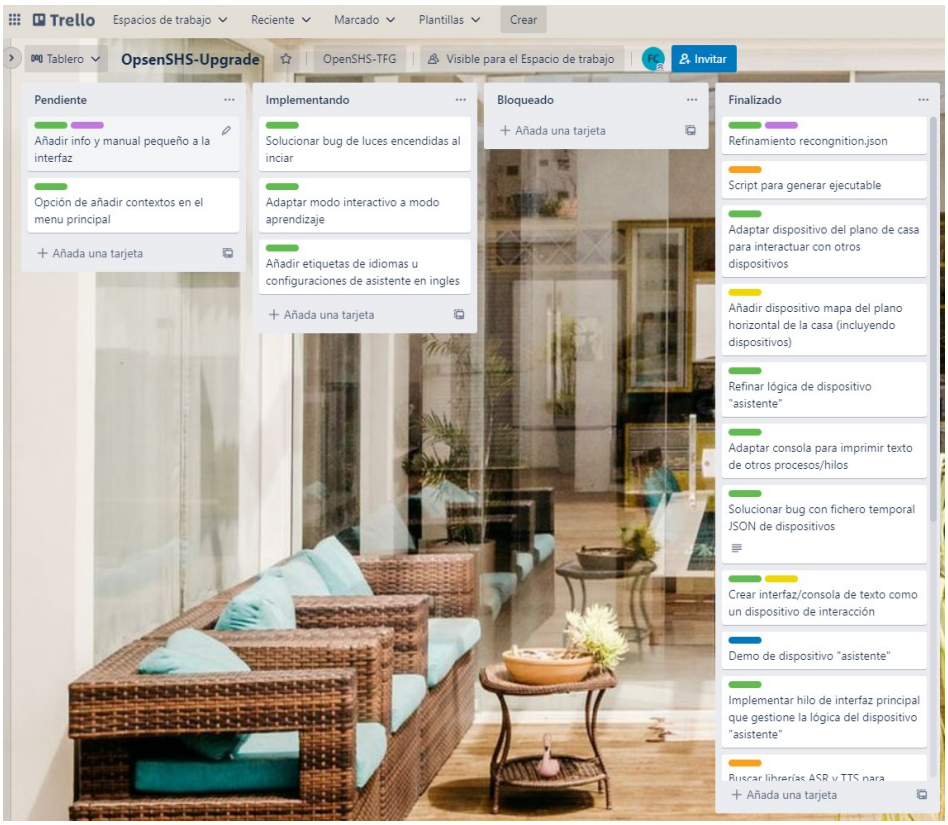

Figura 1: Pizarra de tareas *Trello*

Por otra parte, los avances del proyecto se han mantenido mediante el controlador de versiones *git* y un repositorio gratuito y público en la nube de *Github*. Cabe destacar que este repositorio es un fork del original de *OpenSHS*, el cual es público y se encuentra bajo la licencia de *GNU General Public License v2.0*, la cual da libertad a cualquier individuo a compartir el contenido del repositorio y modificarlo (*GNU Library General Public License, version 2.0*, 2022).

2. Estado del arte

Existen investigaciones que se relacionan con la generación de conjuntos de datos para aplicaciones domésticas inteligentes. Alshammari et al (Alshammari et al., 2017) presenta un análisis de estos trabajos y los clasifica en dos ramas: generación de conjuntos de datos en hogares inteligentes reales o generación de datos con herramientas de simulación. A continuación, se presentan las propuestas que se han analizado en este trabajo.

Primero, en la categoría de obtención de datos mediante un entorno real existen los siguientes proyectos:

- Proyecto realizado por el Centro de Estudios Avanzados en Sistemas Adaptativos (*CASAS*) (Cook et al., 2013), donde crearon un conjunto de herramientas, que se instala fácilmente en un hogar para proporcionar servicios inteligentes. Los conjuntos de datos están disponibles públicamente en la web del centro de estudios.

- *SmartLab* (Nugent et al., 2009) se define como un laboratorio inteligente diseñado para experimentos en el campo de las casas inteligentes. Presenta muchos tipos de sensores, como sensores de presión, infrarrojos pasivos y sensores de contacto. Las interacciones de los participantes con *SmartLab* se capturan en un esquema basado en *XML* llamado *homeML*.

- El hogar ubicuo (Yamazaki, 2007), consiste en un hogar inteligente construido para estudiar servicios conscientes del contexto al proporcionar cámaras, micrófonos, sensores de presión, acelerómetros y otros sensores. Para proporcionar servicios individuales para cada residente, se reconoce a cada uno proporcionando etiquetas de identificación por radiofrecuencia y utilizando las cámaras instaladas.

- *PlaceLab* (Intille et al., 2006) es un apartamento inteligente de 1000 pies cuadrados que tiene varias habitaciones. Consta de muchos sensores distribuidos por cada habitación, en los que se incluyen sensores de corriente eléctrica, sensores de humedad, sensores de luz y sensores de flujo de agua. El participante puede vivir en el apartamento para generar un conjunto de datos a partir de su interacción y comportamiento.

- *HomeLab* (BER et al., 2005) es un hogar inteligente equipado con 34 cámaras distribuidas en varias habitaciones. Cuenta con una sala de observación que permite al investigador analizar y monitorear los experimentos realizados.

Por otra parte, en la rama de la utilización de herramientas de simulación surgen dos nuevas categorías (Synnott et al., 2015), las basadas en el modelo y las interactivas.

La aproximación de las herramientas basadas en el modelo (Alshammari et al., 2017) consiste en la utilización de modelos predefinidos de actividades para generar datos sintéticos. Se especifica el orden de los eventos, la probabilidad de que ocurran y la duración de cada actividad. Dentro de esta categoría se encuentran:

- *Persim 3D* (Lee et al., 2015), herramienta para simular y modelar las actividades de los usuarios en espacios inteligentes. Proporciona una interfaz gráfica de usuario (*GUI*) para visualizar las actividades en *3D*. El investigador puede definir contextos y establecer rangos de valores aceptables para los sensores en el hogar inteligente. Es importante aclarar que la herramienta no está disponible para el dominio público.

- *SIMACT* (Bouchard et al., 2010) es un simulador de hogar inteligente en 3D diseñado para el reconocimiento de actividades. Tiene muchos escenarios grabados que pueden ser utilizados para generar conjuntos de datos para el reconocimiento de *ADL*. Es de código abierto y multiplataforma, desarrollado con Java y utiliza *Java Monkey Engine* (*JME*) como su motor *3D*.

- *DiaSim* (Bruneau et al., 2009) es un simulador desarrollado en Java para sistemas informáticos que pueden interactuar con dispositivos domésticos inteligentes heterogéneos. Tiene un editor de escenarios que permite al investigador construir el entorno virtual.

- El Sistema de Simulación Consciente del Contexto (*CASS*) (Park et al., 2007), consiste en otra herramienta cuyo objetivo es generar información del contexto de simulación. Permite al investigador establecer reglas para diferentes contextos, donde cada regla puede ser una acción que se ejecuta al cumplirse ciertas condiciones en el entorno. Puede detectar conflictos entre las reglas de los escenarios contextuales y determinar el mejor posicionamiento de los sensores. Proporciona una interfaz gráfica de visualización *2D* para el hogar inteligente virtual.

El enfoque de las herramientas interactivas se basa en tener un avatar controlado por un participante o el propio investigador, dicho avatar se podrá mover e interactuar en un entorno simulado que estará en un monitoreo constante mediante sensores. Su principal inconveniente es el tiempo invertido para obtener conjuntos de datos grandes. Algunos ejemplos son:

- *Intelligent Environment Simulation* (*IE Sim*) (Synnott et al., 2014), herramienta utilizada para generar conjuntos de datos simulados que capturan actividades cotidianas (*ADL*) normales y anormales de los habitantes. Permite el diseño de casas inteligentes al proporcionar una vista *2D* de la parte superior del plano. Se pueden añadir diferentes tipos de sensores como sensores de temperatura y sensores de presión. El formato del conjunto de datos generado es *homeML*. No está disponible en el dominio público.

- *UbiREAL* (Nishikawa et al., 2006), herramienta de simulación implementada en Java que permite el desarrollo de aplicaciones ubicuas en un espacio inteligente virtual *3D*. Permite simular las operaciones y comunicaciones de los dispositivos inteligentes a nivel de red.

- *V-PlaceSims* (Lertlakkhanakul et al., 2008), permite diseñar una casa inteligente a partir de un plano. También, permite que múltiples usuarios interactúen con este entorno a través de una interfaz web. El enfoque de esta herramienta es la mejora de los diseños y la gestión del hogar inteligente.

De forma resumida, el enfoque basado en modelos tiene como ventaja la capacidad de generar grandes conjuntos de datos en menor tiempo de simulación, sacrificando el nivel de

captura de los detalles que se obtienen con interacciones realistas. Por su parte, el enfoque interactivo captura estas interacciones, sacrificando el tiempo de simulación.

Adicionalmente, se puede señalar como una desventaja de las herramientas de simulación para la comunidad de investigación, la carencia en algunas de ellas de un código de software que esté accesible y disponible. Otro elemento desfavorable a destacar sobre estos simuladores es su incompatibilidad con múltiples sistemas operativos.

3. Análisis

En esta sección se analizan los componentes, la arquitectura y las tecnologías que conforman el simulador. Además, se exponen los requisitos a tener en cuenta para la implementación de las extensiones.

3.1. Análisis del simulador

Tomando como base la documentación oficial (Alshammari et al., 2017), *OpenSHS* se define como una herramienta de código abierto multiplataforma presentado en 2007 para la simulación *3D* de casas inteligentes. Se enfoca en la generación de conjuntos de datos y para ello sigue una aproximación que une las características del enfoque interactivo (obtención de datos en tiempo real de las actividades de un individuo durante la simulación) y del enfoque basado en el modelo (obtención de datos mediante el uso de modelos estadísticos). Además, posee la característica de avance rápido, permitiéndole al investigador avanzar en el tiempo en dependencia de la duración de una actividad.

El proceso de obtención de conjuntos de datos del simulador se divide en tres fases: diseño, simulación y agregación. Primero, la fase de diseño (Figura 2) consiste en la construcción de un entorno virtual utilizando la herramienta de *Blender*. Este entorno debe contener tanto los objetos de la casa inteligente propuesta como la lógica de la misma, incluyendo dispositivos inteligentes, etiquetas de las actividades, diseño del plano de la casa y controladores de dispositivos.

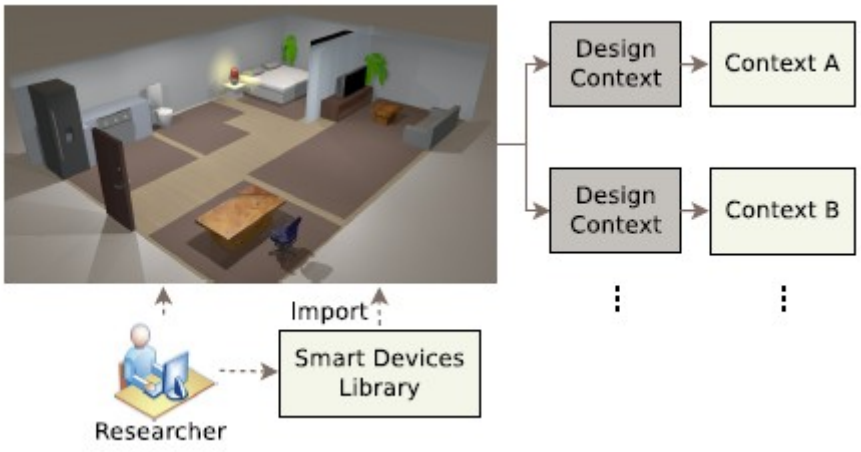

Figura 2: Fase de diseño (Alshammari et al., 2017)

Actualmente dicha fase es opcional debido a que el simulador proporciona una lista de dispositivos y sensores, en los que se incluyen sensores de presión, sensores de puertas, dispositivos de cerrojo, interruptores de electrodomésticos y controladores de luces. A su vez, incorpora cinco actividades (dormir, comer, personal, trabajo y otros) y dos contextos

(“mañana“ y “noche”). El significado de contexto hace referencia a un modelo de casa con una serie de características fijas relacionadas al entorno que pretende simular, por ejemplo, el contexto de “noche” contiene una casa con un diseño similar al de la Figura 2, no presenta iluminación ambiental, el jugador está posicionado en la entrada de la casa y los dispositivos por defecto están configurados con un estado específico.

Por otra parte, en la fase de simulación (Figura 3) el investigador o usuario al iniciar la herramienta debe especificar el contexto. Cada contexto tiene una fecha y hora de inicio que puede ser alterada. Además, se tiene en cuenta un estado por defecto para todos los sensores y dispositivos.

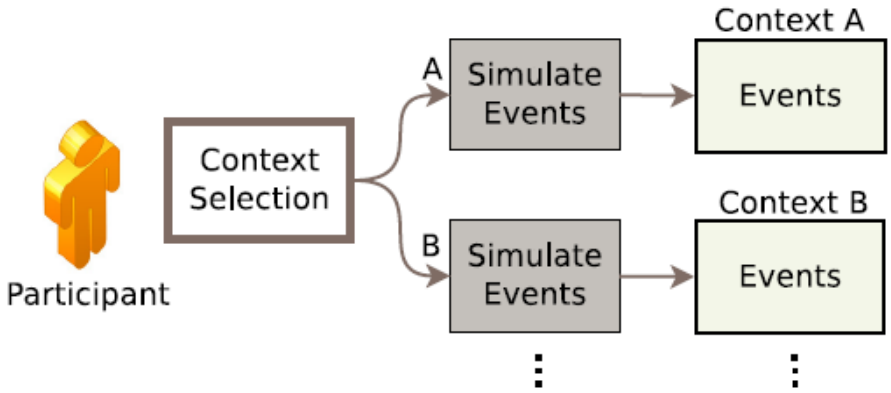

Figura 3: Fase de simulación (Alshammari et al., 2017)

Las salidas de los sensores y el estado de los diferentes dispositivos se capturan y almacenan en un fichero con extensión *csv* que contendrá el conjunto de datos temporal. La frecuencia de muestreo es de un segundo por defecto pero se puede re-configurar según sea necesario.

En relación al mecanismo de avance rápido, el usuario puede controlar la duración de tiempo de una determinada actividad, por ejemplo, si se desea ver la televisión durante un período de tiempo y no desea realizar toda la actividad en tiempo real, se puede abrir un menú para especificar cuánto dura dicha actividad tal y como se muestra en la Figura 4.

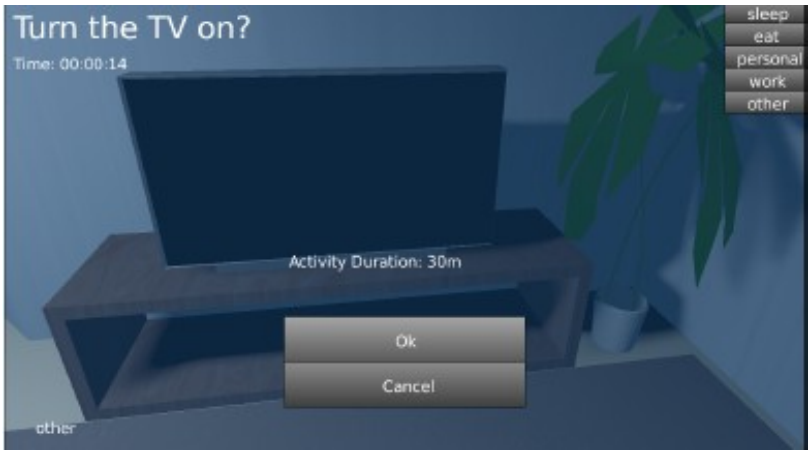

Figura 4: Selección de actividad y duración (Alshammari et al., 2017)

Por último en la fase de agregación (Figura 5), se utiliza como base las actividades de muestra generadas por los usuarios o participantes para construir un conjunto de datos final

mediante un algoritmo ya implementado que replica la salida de la fase de simulación dibujando muestras apropiadas para cada contexto. Dicho algoritmo es explicado y detallado en profundidad en el documento oficial de *OpenSHS* (Alshammari et al., 2017), como no se relaciona directamente con el enfoque del presente trabajo no se indagará en su diseño.

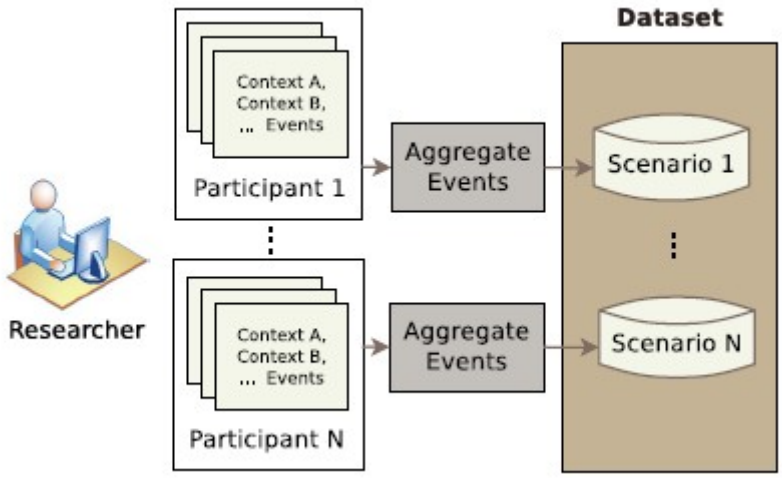

Figura 5: Fase de agregación (Alshammari et al., 2017)

Cabe destacar que estas fases no se realizan automáticamente, es responsabilidad del desarrollador o investigador realizarlas. Tanto la primera como tercera fase son opcionales y alterables.

En cuanto a la interfaz de usuario, utiliza como base una interfaz por comandos de consola implementada en *Python*. Dicha interfaz es capaz de ejecutar el algoritmo de replicación de datos y lanzar los contextos diseñados en *Blender* versión 2.7 o inferior debido a que requiere del motor de juegos (*BGE*), el cual ha sido eliminado en versiones más modernas y reemplazado por *EEVEE* ("Blender (software)", 2022).

3.2. Tecnología empleada

El simulador está desarrollado casi en su totalidad con el lenguaje de programación *Python* y los contextos que incluyen el diseño de la casa, escenas y dispositivos se crearon con la herramienta de *Blender*. Resulta relevante profundizar en estas tecnologías para la implementación de las extensiones.

3.2.1. *Python*

Python ("Python", 2022) es un lenguaje interpretado de alto nivel de programación, consta de una filosofía enfocada a la legibilidad de su código. Se utiliza para el desarrollo de aplicaciones de todo tipo, entre las más populares: *Instagram*, *Netflix* y *Spotify*. Una de sus principales características es ser multiparadigma, ya que es orientado a objetos y a su vez soporta la programación imperativa y la programación funcional.

Adicionalmente, es un lenguaje multiplataforma y de código abierto. Esto es importante puesto que se adapta al enfoque perseguido por el simulador.

3.2.2. *Blender*

Blender ("Blender (software)", 2022) es un conjunto de herramientas de software de gráficos por computadora *3D* gratuito y de código abierto. Es utilizado para la creación de películas animadas, efectos visuales, arte, modelos impresos en *3D*, gráficos en movimiento, aplicaciones interactivas en *3D*, realidad virtual y videojuegos.

Sus características incluyen modelado *3D*, mapeo *UV* (proceso que consiste en proyectar una imagen *2D* a la superficie de un modelo *3D* para el mapeo de texturas), texturizado, dibujo digital, edición de gráficos ráster, simulación de fluidos y humo, simulación de partículas, simulación de cuerpo blando, escultura, animación, movimiento de coincidencias, renderizado, gráficos en movimiento, edición de video y composición.

En versiones inferiores a la 2.7 incorpora un motor para el desarrollo de videojuegos (*BGE*), el cual fue escrito en el lenguaje de *C++* como un módulo independiente e incluye soporte para *python scripting* y sonido *OpenAL* (*Open Audio Library*).

3.3. Requisitos

Como consecuencia de la investigación previa, se plantearon una serie requisitos para la ampliación del simulador y la adición de mecanismos de interacción. Estos requisitos van enfocados a mejorar la interacción humano-sistema de un simulador de casas inteligentes, siendo uno de los objetivos pertenecientes al proyecto del que parte el presente trabajo.

Los requisitos se dividen en tres secciones, la primera relacionada con la interfaz por consola creada con el lenguaje de *Python* (requisito 1), la segunda vinculada con la herramienta de *Blender*, la simulación en sí (requisito 2 al 4), y por último se incluyen aquellos que convergen con ambas tecnologías o requisitos extras (requisito 5 al 7).

- **Requisito 1:** el simulador debe disponer de una interfaz gráfica de usuario para utilizar todos los modos y herramientas que ofrece.
- **Requisito 2:** el simulador debe disponer de un dispositivo capaz de reconocer el habla, realizar una acción, generar una respuesta y transformar la respuesta de texto a voz.
- **Requisito 3:** el simulador debe disponer de un dispositivo estilo consola o pantalla que muestre cierta información, deberá ser capaz de mostrar información proveniente de otros dispositivos.
- **Requisito 4:** el simulador debe permitir que los usuarios consulten un mapa que represente la vista superior de la casa. En dicho mapa se deben mostrar las luces u otros dispositivos.

- **Requisito 5:** el simulador debe permitir a los usuarios cambiar fácilmente el estado de algunos dispositivos mediante el uso de los dispositivos creados en los requisitos previos.
- **Requisito 6:** el simulador debe ofrecer una configuración para cambiar el idioma tanto de la interfaz gráfica como de los contextos simulados.
- **Requisito 7:** el simulador debe ofrecer un contexto que simule situaciones que se dan de forma frecuente durante la tarde en un hogar.

3.4. Detección de problemas

La versión de *OpenSHS* utilizada como base para este trabajo es la perteneciente al *commit* 42 subido el 30 de noviembre del 2021 a la rama principal del repositorio en *Github*. Dicha versión contiene una serie de errores y *bugs* que se han solucionado como parte de este trabajo. A continuación, se comentan cuáles han sido los problemas que se han solucionado:

- Formato incorrecto de la fecha por defecto, al no introducir una fecha para iniciar la simulación de un contexto el programa lanza una excepción debido a que utiliza una fecha preestablecida que sigue un formato incorrecto.
- Error al ejecutar el comando *aggregate*, al ejecutarlo se lanzaba una excepción durante el análisis de los datos almacenados en los ficheros con extensión *csv*.
- Solapamiento de conjuntos de datos obtenidos en simulaciones previas con las misma fecha y hora. Ejecutar un contexto dos o más veces en un mismo día y hora producía un error, debido a que se almacena como fichero temporal durante la simulación con el nombre de *output.csv* y al finalizar se modifica, por lo que si existe un fichero con el mismo nombre falla.

Por último, se detectó que si un investigador desea añadir un nuevo contexto debe alterar la constante *CONTEXTS* del código implementado en *Python*. Sin embargo, el investigador no tiene porqué estar familiarizado con dicho código fuente. En un principio, se planteó la idea de que el programa lea de forma automática los contextos ubicados en el directorio *blender*, pero se descartó, debido a que en dicho directorio se pueden ubicar otros ficheros que no representan contextos, un ejemplo es el fichero *apartment.blend*. Finalmente se decidió añadir un fichero *JSON* de configuración que incluirá los contextos a utilizar por el simulador además de otros posibles ajustes.

4. Desarrollo

En esta sección se detallan las soluciones y decisiones tomadas durante el proceso de implementación de los requisitos. Con este fin, se enumeran las herramientas utilizadas, los ficheros creados y los dispositivos añadidos.

4.1. Herramientas

Como se explica en la sección Análisis, el simulador está desarrollado con el lenguaje de programación *Python*, por lo que se debe seleccionar un editor código para su correcta programación y un controlador de versiones.

4.1.1. *Visual Studio Code*

El *VS Code* fue lanzado por *Microsoft* en el 2015 y actualmente se utilizan en sistemas operativos *Windows*, *Linux* y *MacOS* ("Visual Studio Code", 2022). Entre sus características más importantes se pueden citar: soporte para la depuración (análisis de traza de ejecución y detección de errores), control integrado de *Git*, resaltado de sintaxis, gestor de extensiones públicas creadas por la comunidad y refactorización de código.

También hay que destacar que es gratuito y dispone de una gran cantidad de extensiones para *Python*, que facilitan notablemente el proceso de codificación.

La elección como editor de código fuente (*IDE*) para el desarrollo propuesto estuvo determinada por la experiencia adquirida en su uso previo en otros proyectos, así como por las características anteriormente mencionadas.

4.1.2. *Git y Github*

Se empleó *Git* ("Git", 2022) como software de control de versiones, alojando las extensiones del simulador en un repositorio *Github*, como se ha hecho en el repositorio de *OpenSHS*. El propósito de este sistema es llevar el registro de los cambios en los archivos, de esta forma se logra coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código. Por otra parte, *GitHub* ("GitHub", 2022) se define como una plataforma de desarrollo colaborativo, donde se alojan proyectos utilizando *Git*.

Cabe destacar que el repositorio al ser un *fork* hereda la licencia del repositorio original y todos los cambios que se hagan se realizarán en la rama principal.

4.2. Solución de errores

Antes de llevar a cabo la extensión del simulador se debían solucionar los fallos detectados. Para el caso de la fecha por defecto fue necesario modificar el formato de lectura de la cadena de caracteres que representa la propia fecha, tal y como se puede apreciar en la Figura 6, el código original en rojo y la modificación en color verde.

```
dt = datetime.strptime(value, "%Y-%m-%d %H-%M-%S")
# The string value of the variable "value" contains or may contain
# ':' characters that must be replaced by '-'
dt = datetime.strptime(str(value).replace(':','-'),
    "%Y-%m-%d %H-%M-%S")
```

Figura 6: Modificación formato fecha

En relación al comando aggregate se han solucionado los errores de ejecución, sin embargo no se ha comprobado el correcto funcionamiento del algoritmo debido a las limitaciones del trabajo propuesto. Por otra parte, el solapamiento de los conjuntos de datos se solucionó agregando un número al final del nombre del fichero en caso de que ya exista uno con el mismo nombre, un posible ejemplo se puede ver en la Figura 7.

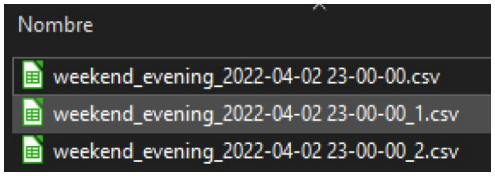

Figura 7: Ficheros de datos no solapados

4.3. Implementación de requisitos

Con el propósito de satisfacer los requisitos se tuvo muy en cuenta el uso de ficheros de tipo *JSON* para la escritura de los datos de configuración que debían ser persistentes y para el intercambio de información entre procesos. Este tipo de fichero es de gran utilidad para la transmisión de datos y posee una notación que es considerada como subconjunto de la notación literal de objetos de *JavaScript* (“JSON”, 2022). El entorno de desarrollo *Python* por defecto incorpora una librería para el tratamiento de ficheros con dicho formato.

4.3.1. Interfaz gráfica de usuario

Para el uso del simulador resulta muy útil tener disponible una interfaz gráfica de usuario, ya que permite una mejor navegación entre las opciones y un uso más fluido de las diferentes herramientas que ofrece.

Previo a la implementación de la interfaz gráfica se optó por utilizar el mismo lenguaje de programación, *Python*, con el objetivo de que los colaboradores del proyecto original pudiesen alterar, mejorar y entender el código. Posteriormente se valoraron varias opciones para la librería de interfaces gráficas, entre ellas:

- ***Tkinter*:** es la interfaz estándar de *Python* para el conjunto de herramientas *GUI Tcl/Tk* (*tkinter — Python interface to Tcl/Tk*, 2022). Está disponibles en la mayoría de las plataformas *Unix*, incluido *MacOS*, así como en los sistemas *Windows*.

- ***PyQT5***: es un conjunto completo de enlaces de *Python* para *Qt v5*. Se implementa como más de 35 módulos de extensión y permite que *Python* se use como un lenguaje de desarrollo de aplicaciones alternativo a *C++* en todas las plataformas compatibles, incluidas *iOS* y *Android* (Thompson, 2022).

- ***PySide2:*** es una biblioteca para crear aplicaciones *GUI* utilizando el conjunto de herramientas *Qt*. Es el enlace oficial para *Qt* en *Python* y ahora lo desarrolla *The Qt Company* (Fitzpatrick, 2022).

- ***Kivy:*** es un *framework* gratuito para *Python* y de código abierto para el desarrollo aplicaciones móviles y otro software de aplicaciones multitáctiles con una interfaz de usuario natural (*NUI*) ("Kivy (framework)", 2022). Se distribuye bajo los términos de la licencia *MIT* y puede ejecutarse en *Android*, *iOS*, *Linux*, *macOS* y *Windows*.

Finalmente, la elegida fue *Tkinter*, puesto que de todas resultó ser la más adecuada para el trabajo, gracias a su capacidad de ser multiplataforma, estar enfocada para el desarrollo de aplicaciones de escritorio, ir incorporada en el conjunto de paquetes de *Python* y ser muy conocida por la comunidad.

Al tener planeado el lenguaje y la librería a utilizar solo quedaba realizar un *mockup* o maqueta de diseño, este se hizo gracias a la herramienta *online* *Moqups App*. Una primera versión de cómo podría lucir la interfaz oficial se muestra en la Figura 8.

Figura 8: Maqueta de diseño *GUI*

Durante el proceso de implementación se realizaron ligeros cambios, incluyendo una opción para cambiar de idioma, obteniendo como resultado la interfaz que se puede ver en la Figura 9.

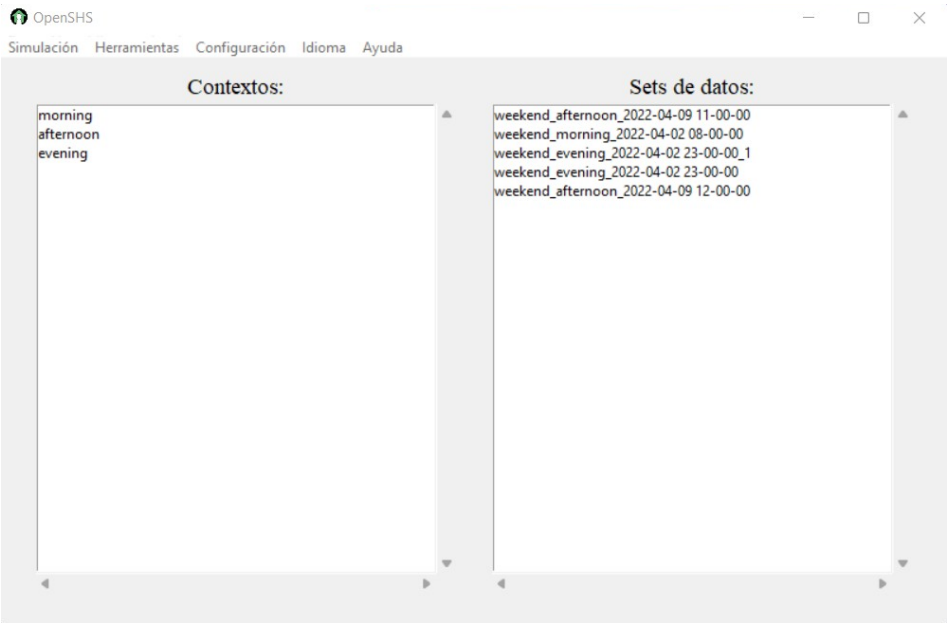

Figura 9: Diseño final *GUI*

Esta interfaz incorpora en la parte superior una barra de navegación con las opciones de "Simulación", "Herramientas", "Configuración", "Idioma" y "Ayuda". En la opción de "Simulación" se puede acceder al modo de "Generación de datos", el cual se carga un contexto seleccionado y se genera a partir de la simulación el conjunto de datos. También se puede acceder al modo interactivo que consiste en lo mismo que la "Generación de datos" eliminando la creación de datos. En la opción de "Herramientas" actualmente solo está disponible la utilización del algoritmo de replicación de datos. La sección de "Configuración" solo permite habilitar o deshabilitar la *verbosidad*, que consiste en mostrar información detallada sobre el estado de los dispositivos en una de las esquinas de la vista del usuario durante de la simulación. Finalmente, la opción de "Idioma" permite cambiar el texto de los componentes de la interfaz entre español e inglés y la opción de "Ayuda" da una breve descripción del simulador y su uso.

Por otra parte, en la sección central e inferior se ubican dos listas de texto, la que se encuentra en la zona izquierda muestra todos los contextos a seleccionar que estén escritos en el fichero de configuración de tipo *JSON* y la lista de la derecha contiene los ficheros con formato *csv* almacenados en el directorio *data*.

Por último, se destaca la adición los distintos diálogos o ventanas modales para presentar mensajes de error o solicitar información adicional al usuario. Las ventanas emergentes por avisos de error pueden surgir por diferentes circunstancias, entre las que se incluyen la falta de selección de un contexto para la simulación, la ausencia del contexto seleccionado a simular, y para el caso del modo interactivo, la selección de un contexto que no lo soporte.

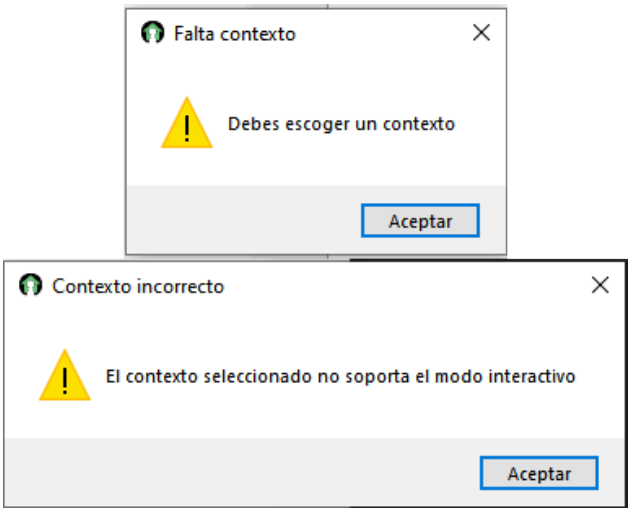

Figura 10: Ventanas emergentes de error

En cambio las ventanas para solicitar información solo aparecen para el modo de "Generación de datos" (Figura 11) y la herramienta de "Replicación de datos" (Figura 12), donde se solicitará la información necesaria para realizar las respectivas acciones.

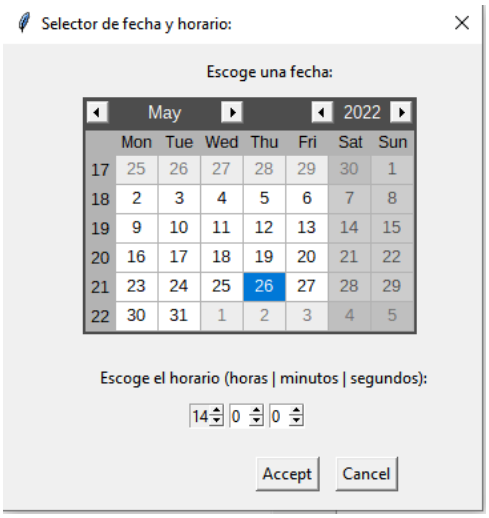

Figura 11: Ventana modal para fecha y horario

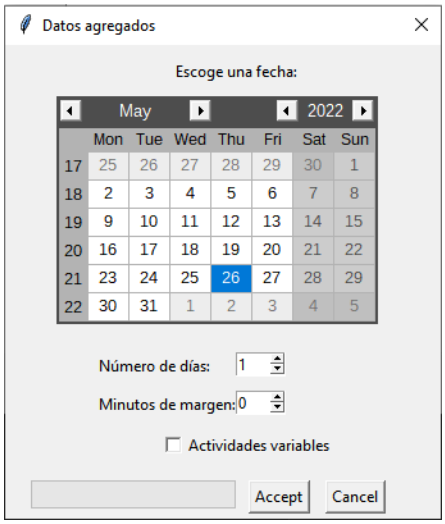

Figura 12: Ventana modal de datos a replicar

4.3.2. Asistente virtual

La idea de un asistente surge de la popularidad de los dispositivos *Echo Dot* como sistemas de interacción entre los usuarios y el hogar, permitiéndoles controlar mediante la voz otros dispositivos conectados a la red y asistir en las actividades cotidianas.

En relación al modelo se optó por diseñar en *Blender* un dispositivo con forma de parlante, tal y como se puede apreciar en la Figura 13, que resulta muy similar al producto *Echo Dot* de

tercera generación patentado por *Amazon*. Además del diseño del modelo se le añadió el respectivo controlador para responder ante eventos del ratón.

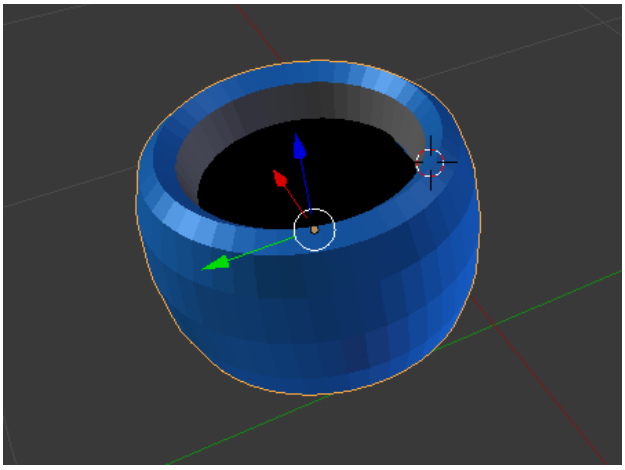

Figura 13: Modelo *3D* asistente virtual

Blender incorpora en su motor de videojuegos un intérprete de *Python* embebido, por ello no se pueden importar paquetes externos en los *scripts* de los controladores de eventos de los dispositivos. Esto es de vital importancia, debido a que, por conveniencia, se necesita el paquete *SpeechRecognition* (Zhang, 2017) para poder cargar el sonido proveniente del micrófono e interpretarlo. Para solucionarlo, se optó por crear una arquitectura de procesos e hilos.

Para entender dicha arquitectura se debe entender primero qué son los procesos e hilos y sus diferencias:

- **Proceso:** se puede definir de forma resumida como un programa en ejecución pero formalmente se le considera *"una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados"* (Stallings, 2005). Además, los procesos son gestionados por el propio sistema operativo y están formados por: las instrucciones a ser ejecutadas, un estado de ejecución que variará en el tiempo, los valores de los registros que se encuentran en la *unidad central de procesamiento* (*CPU*), la memoria reservada para su ejecución y su contenido, y la información extra útil para que el sistema operativo puede realizar su planificación. Hay que destacar que dos o más procesos pueden ser ejecutados en un mismo instante tiempo siempre y cuando el sistema lo permita.

- **Hilo:** se define como una tarea que parte de un mismo proceso y comparte sus recursos, por ejemplo: si un hilo modifica un dato en la memoria, los otros hilos pueden acceder a su valor; en cambio, los procesos por sí mismos no pueden, debido a que cada uno tiene un mapa de memoria distinto. Generalmente se suelen utilizar para la realización de tareas específicas, para aumentar la eficiencia del uso de la *CPU* y por ende, disminuir el tiempo de ejecución del programa. Esto se debe a que si el sistema lo permite, dos o más hilos se pueden ejecutar en paralelo o en concurrencia. Cabe destacar que cuando el proceso que crea los hilos finaliza, sus hilos también lo hacen.

Partiendo de las definiciones, se realizó la implementación de tal forma que al iniciar el simulador se crease un proceso dedicado a la interfaz gráfica de usuario. Por tanto, al acceder al

modo interactivo o de "Generación de datos", se crea un hilo dedicado a la lectura del micrófono solo en caso de que el contexto seleccionado soporte dicha característica y se arranca el proceso *Blender*.

Por simplicidad, para crear el hilo en *Python*, se decidió importar el paquete *threading*. Este paquete incluye la clase *Thread*, el cual a partir de una función a ejecutar y un nombre, crea una instancia de un hilo que solo será ejecutado al invocar su método *start*. Para adaptarlo al asistente se optó por crear una clase *Assistant* que hereda los métodos y atributos de *Thread*, añadiéndole la funcionalidad requerida.

Por otra parte, para lanzar el proceso *Blender* se utiliza el paquete *subprocess* ya importado en la versión original del simulador. Sobre esta librería se invoca la función *call* pasando como parámetro el comando a ejecutar; en este caso será "blender" y el nombre del fichero del contexto como argumento.

```
subprocess.call(["blender", bl_file], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL,
    shell=False)
```

Figura 14: Creación del proceso *Blender*

El hilo del asistente se encargará de leer cada 30 *ticks* lógicos (*frames*) de un fichero *temp.json* comprobando el valor del parámetro *listen*. Si se encuentra con un valor *booleano* de verdadero, procede a leer la entrada del micrófono para interpretarla y realizar otras acciones. Dicho parámetro *listen* será modificado por el proceso que llevará a cabo la simulación del contexto, por ende si el controlador del modelo del asistente detecta un evento del ratón haciendo *click*, este modificará el valor de *listen* a verdadero y a la inversa si se vuelve a producir evento del ratón estando el valor a verdadero se cambia a falso.

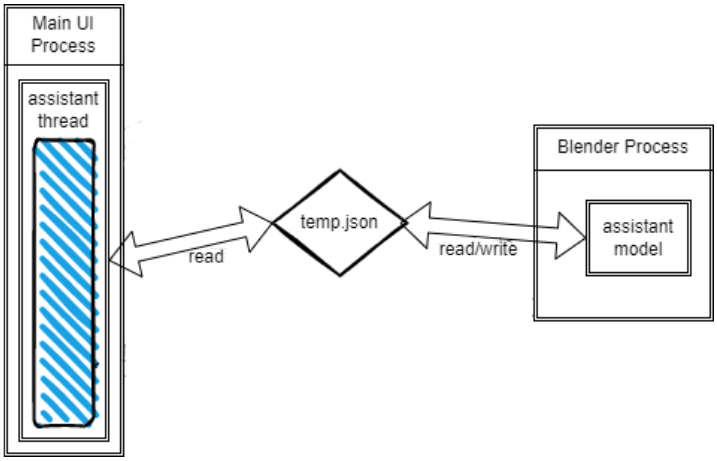

Figura 15: Lectura y escritura *temp.json* por procesos del asistente

Una vez habilitada la lectura de la entrada del micrófono, el paquete *SpeechRecognition* se configura para que lleve a cabo el reconocimiento de la voz con un *timeout* de un segundo si no detecta sonido, y de cuatro segundos si lo detecta. De esta forma dado un idioma determinado, la voz del usuario se transformará en una cadena de caracteres que tendrá una respuesta asociada según la programación del fichero *rec_thread.py* en su método *analyze* o alternativamente en el fichero de configuración del hilo del asistente denominado *recognition.json*.

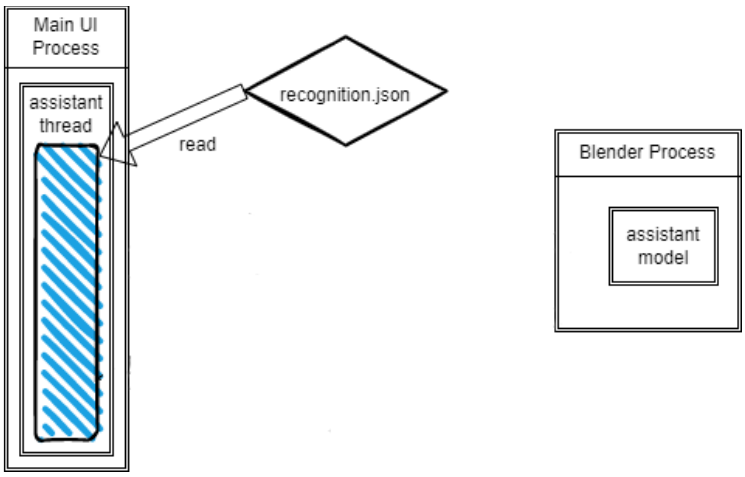

Figura 16: Lectura *recognition.json* por procesos del asistente

Paralelamente, la estructura que compone al servicio de *Amazon Lex* para la creación de *bots* con capacidades de reconocimiento de voz, análisis y selección de respuestas, sirvió de inspiración para el esquema utilizado en el fichero *recognition.json*.

Con el fin de comprender la base de dicho servicio y asociarla con el esquema utilizado, se deben tener en cuenta algunos conceptos ("Amazon Lex: cómo funciona", 2022):

- **Intenciones:** representa una acción que el usuario desea realizar. Cuando un usuario interactúa con el *bot*, la consulta coincide con la mejor intención disponible, es por ello que cada intención tiene un conjunto de enunciados de muestra.
- **Enunciados de muestra:** es la colección de posibles frases que un usuario podría decir, que significan lo mismo que la intención asociada.
- **Ranuras:** son parámetros que se definen como parte de la configuración de la intención, contienen datos estructurados que se pueden usar fácilmente para realizar alguna lógica o generar respuestas. El valor de una ranura se extrae en tiempo de ejecución de la consulta del usuario.

Como complemento al esquema utilizado en el fichero de tipo *JSON*, es conveniente precisar en qué consiste un diccionario en el campo de la programación. Un diccionario es una estructura de datos, que contiene una lista de asociaciones de pares de objetos. Al primer objeto del par se le denomina clave, y su característica es que no pueden existir dos o más claves iguales en todo el diccionario. En lo que respecta al segundo objeto del par, se conoce como valor y debe estar asociado al menos a una clave.

El fichero de configuración *recognition.json* proporcionado consta de un diccionario con tantas claves como idiomas soportados por el simulador. A su vez, cada idioma tiene como valor un diccionario que consta de una clave *name* que representa el nombre del asistente en el respectivo idioma y otra clave *requests* cuya funcionalidad es similar al de las intenciones en el servicio *Lex*. Cada *request* tendrá asociada asociada una lista de samples que se asimilan a los enunciados de muestra detallados anteriormente, una lista *tags* cuyo propósito es detectar variaciones en las frases y un parámetro *type* que servirá al hilo del asistente para clasificar la respuesta. En caso de que el parámetro *type* sea tipo *command* se procede a ejecutar una acción programada cuya etiqueta sea igual al contenido de *cmd* y en el resto de casos se limita a

convertir a voz el contenido del atributo response. La conversión de texto a voz se lleva a cabo con una librería gratuita de *Google* denominada *gTTS* (*gTTS 2.2.4*, 2022), obteniendo el audio en un fichero con formato mp3, el cual se reproduce con el paquete *PyGame*.

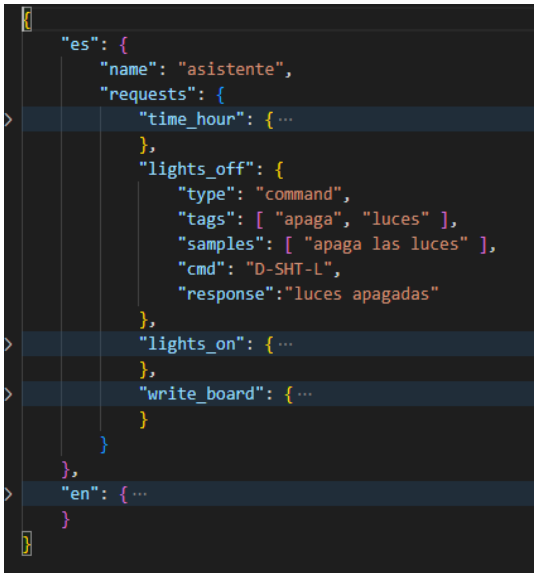

Figura 17: Ejemplo *recognition.json*

Cabe destacar que se ha excluido el uso de ranuras en la implementación del asistente. Sin embargo, tanto la adición de dicha característica como de nuevos tipos de *request* se puede llevar a cabo mediante la modificación del fichero *rec_thread.py*.

4.3.3. Consola de mensajes

Se optó por brindar un diseño similar al de una pizarra de gran dimensión al modelo de la consola de mensaje, como se puede apreciar en la Figura 18. Su principal característica es mostrar un mensaje de texto que pueda variar, ya sea porque el propio usuario lo ha impuesto manualmente o debido a que otro proceso o dispositivo lo ha modificado durante la simulación.

El modo del dispositivo consta de dos componentes: la propia consola y el texto a mostrar. Puesto que el último componente puede variar su comportamiento en ciertas condiciones, se agregó un *script Python* como controlador. Dicho *script* lee cada 30 *ticks* lógicos (*frames*) el contenido del atributo *console_dev* del fichero *temp.json* y asigna el contenido a la propiedad *text* de la componente de tipo texto. Cabe destacar que si el texto leído contiene más de 119 caracteres se perderá información y que cada 19 caracteres se incluirá un salto de línea.

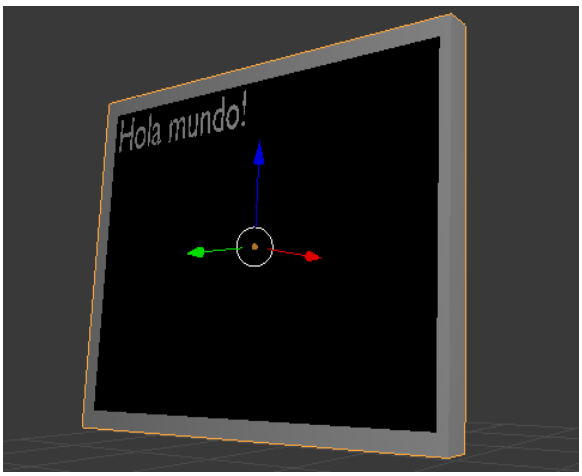

Figura 18: Modelo consola de mensajes

4.3.4. Mapa de dispositivos

En relación al diseño del mapa de dispositivos se parte del modelo de las paredes de la casa, se aplica un ángulo de 90° y se añade un volumen con forma de ortoedro. El resultado se puede ver en la Figura 19.

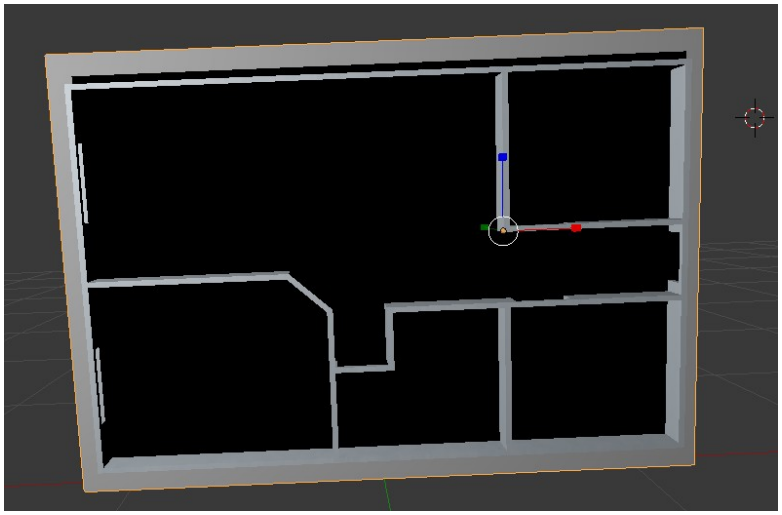

Figura 19: Base mapa de dispositivos

Para representar los dispositivos se deben añadir tantos cilindros como se deseen. Cada cilindro estará ubicado por ende en el punto análogo a la zona de la casa donde se encuentra el dispositivo que representa y debe ir asociado a su controlador activándolo por pulsación del *click* izquierdo del ratón. La Figura 20 es un ejemplo de mapa con las luces añadidas.

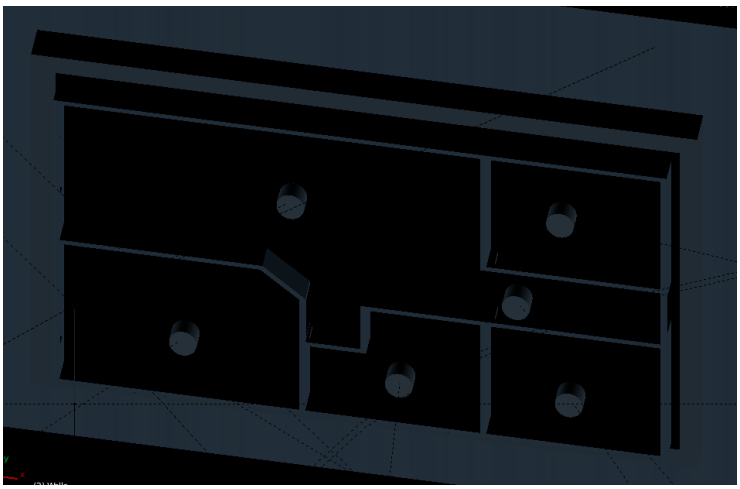

Figura 20: Mapa de dispositivos con las luces

Cabe destacar que para casas con más de una planta se recomienda crear un mapa de dispositivos por cada planta.

4.3.5. Traducciones

Con el fin de permitir el uso del simulador a múltiples usuarios, independientemente de su país de origen o idiomas que comprendan, se planteó añadir traducciones a todos aquellos componentes de la interfaz gráfica y contextos de simulación *Blender* que contengan texto. Para ello se creó un fichero *translations.json* cuyo formato se basa en un diccionario donde las claves son etiquetas (cadenas de caracteres únicas) para cada texto a traducir. El contenido del valor asociado a cada etiqueta es otro diccionario. Este diccionario dispone de tantas claves como idiomas son soportados por el simulador, y a su vez, lleva asociado a sus claves, la traducción del texto correspondiente a la etiqueta.

```
{
    "es": {
        "es": "español",
        "en": "spanish"
    },
    "en": {
        "es": "ingles",
        "en": "english"
    },
    "Simulation": {
        "es": "Simulación",
        "en": "Simulation"
    },
    "Tools": {
        "es": "Herramientas",
        "en": "Tools"
    },
    "Config": {···
    },
    "Lang": {···
    },
```

Figura 21: Ejemplo de *translations.json*

De esta forma cuando se quiera añadir un componente que contenga una cadena de caracteres a la interfaz o dispositivo a un contexto se debe incorporar una nueva entrada en el fichero *translations.json* siguiendo el esquema explicado previamente (Figura 21). Adicionalmente, en el código programado se debe leer del fichero y cargar la respectiva traducción dependiendo del idioma establecido en fichero *config.json* en el atributo *language*.

4.3.6. Verbosidad

Para depurar los contextos implementados en el simulador resulta muy útil mostrar información sobre el estado de los dispositivos presentes en la escena. Por ello se añadió una configuración en la que se puede habilitar o deshabilitar desde la interfaz gráfica la aparición de información extra en la esquina superior izquierda de la vista principal del usuario durante la simulación (Figura 22).

La información que se muestra puede variar dependiendo de las necesidades del investigador o usuario. Si así se desea, se puede acceder al controlador *hud_verbose.py* en los contextos proporcionados. Dicho controlador va asociado al objeto de tipo texto con la etiqueta "verbose" en la escena *hud* y se ejecuta cada 30 *ticks* lógicos (*frames*).

Figura 22: Ejemplo de verbosidad

4.3.7. Nuevo contexto de simulación

Para la creación de un nuevo contexto que represente la tarde, se añadió un objeto de tipo lámpara cuya luz es de tipo luz solar. La luz solar tiene una intensidad constante y una dirección que puede ser configurada. Dicha dirección representa los rayos del sol, y para este caso, fue establecida apuntando verticalmente hacia el suelo de la escena, a una distancia aproximada de 25 centímetros del centro de coordenadas en la *dimensión Z*, con respecto a un punto cercano al centro de la casa.

Dicha lámpara está configurada para crear reflejos especulares ("Resaltado especular", 2022) (puntos brillantes de luz que aparecen en objetos brillantes cuando se encuentran iluminados) y sombreados de difusión ("Sombreado de Phong", 2022) (se aplica una técnica para obtener la intensidad de los colores de cada pixel de las superficies en objetos *3D*). Igualmente, tanto la cámara como la posición del jugador se encuentran situados en el salón, como se aprecia en la Figura 23.

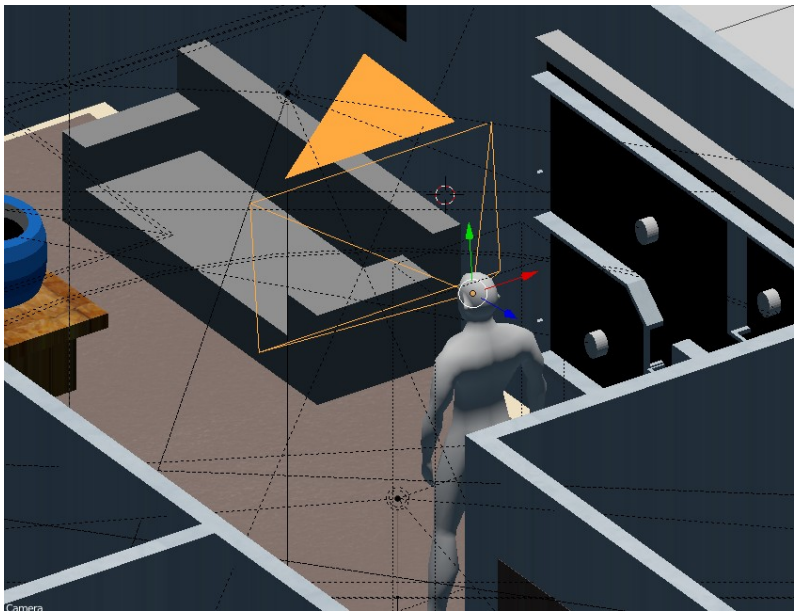

Figura 23: Jugador y cámara en contexto de "tarde"

Hay que aclarar que la luz utilizada viene de una dirección constante, tiene una intensidad uniforme y no arroja sombras. En la versión de *Blender* utilizada, ninguna luz, excepto las del tipo *spot*, arrojan sombras (*Tema: Manual de Blender parte v iluminación*, 2008).

4.4. Ficheros *JSON*

A modo de recopilación, en esta sección se pretende explicar el propósito, contenido y esquema de cada uno de los ficheros *JSON* utilizados y mencionados durante el presente trabajo.

- **Config**: está conformado por un diccionario que contiene como entradas los posibles ajustes que pueden afectar la visualización de la interfaz gráfica y el arranque de contextos. Actualmente, se encuentran los siguientes atributos (Figura 24):

  → *language*: indica el idioma a cargar para las traducciones de los componentes de la interfaz gráfica y de los mensajes mostrados por los controladores de los dispositivos en la escena *hud* de los contextos simulados, y el lenguaje que utilizará el asistente virtual para el reconocimiento de la voz y de forma inversa llevar el texto a sonido.

  → *start_time*: es una medida de tiempo cuyo formato se basa en el *"Tiempo Unix"* (corresponde a la cantidad de segundos transcurridos desde la medianoche *UTC* del 1 de enero de 1970). Se utiliza para indicar al proceso de *Blender* el tiempo en el que transcurre la simulación.

  → *verbose*: es de tipo booleano e indica si se usa la característica de verbosidad explicada en la sección anterior.

  → *contexts*: lista de contextos disponibles, cada uno hace referencia a un fichero con extensión ".blend" ubicado en el directorio "blender".

  → *interactive*: también representa una lista de contextos, en este caso hace referencia a los contextos disponibles que soportan el modo interactivo (no se crean ficheros con los datos de los estados de los dispositivos).

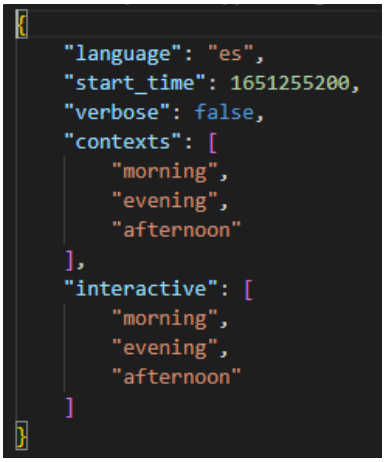

Figura 24: Ejemplo *config.json*

- **Devices:** este fichero es de gran utilidad para representar en tiempo real el estado de los dispositivos durante la simulación. Actualmente, solo el hilo de la interfaz gráfica encargado del reconocimiento y habla del asistente virtual es capaz de modificar el contenido de *devices.json* para forzar el cambio del estado de las luces de la casa. En los contextos incluidos, se encuentra un ejemplo de controlador denominado *check_lights.py* que cada 30 *ticks* lógicos lee y actualiza el estado de las luces en la propia escena o en el fichero *devices.json*.

Figura 25: Ejemplo *devices.json*

- **Recognition:** el fichero *recognition.json* surge de la necesidad de establecer un esquema que determine el comportamiento del asistente virtual recreado por el hilo *rec_thread.py* del asistente virtual. El esquema planteado se explica a detalle en el apartado del "Asistente virtual".

- **Temp:** contiene aquellos datos cuya información solo resulta útil durante ciertos períodos de ejecución del simulador y no requieren persistencia. Tal y como se puede apreciar en la Figura 26, actualmente se encuentra el parámetro "listen" que indica al hilo del asistente cuando debe escuchar por el micrófono (solo si está con un valor de verdadero) y el parámetro "console_dev" que será leído por el controlador de la consola de mensajes detallado en el apartado de "Consola de mensajes".

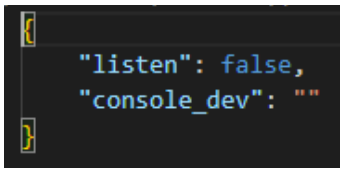

Figura 26: Ejemplo *temp.json*

- **Translations:** este fichero es tratado en el apartado de "Traducciones".

En relación a la lectura y escritura de los ficheros mencionados, se han usado trozos de código que se repiten con pocas variaciones en la gran mayoría de controladores o *scripts* de la interfaz gráfica, un ejemplo de lectura se puede apreciar en la Figura 27 y para escritura en la Figura 28.

```
# Reads from a json file the status of the devices
# according to the assistant
with open(CONFIG_PATH, "r") as json_file:
    devices = json.load(json_file)
```

Figura 27: Ejemplo lectura fichero de configuración

```
# Status of the devices are written
json_object = json.dumps(devices, indent = 4)
with open(CONFIG_PATH, "w") as json_file:
    json_file.write(json_object)
```

Figura 28: Ejemplo escritura fichero de configuración

Un posible fallo en este tipo de diseño es la condición de carrera. En el período en el que un controlador, proceso o hilo lee de un fichero de los mencionados al sobrescribirlo puede que elimine los cambios realizados por algún otro proceso sobre el mismo recurso. Esta situación podría llegar a ocurrir en los ficheros *devices.json* y *temp.json*, y no se considera como crítica.

4.5. Estructura

Como resultado de las extensiones realizadas en el simulador se ha obtenido una estructura de directorios. Para la obtención del resultado mostrado en la Figura 29 se han tenido en cuenta los mejores patrones de modularización de clases.

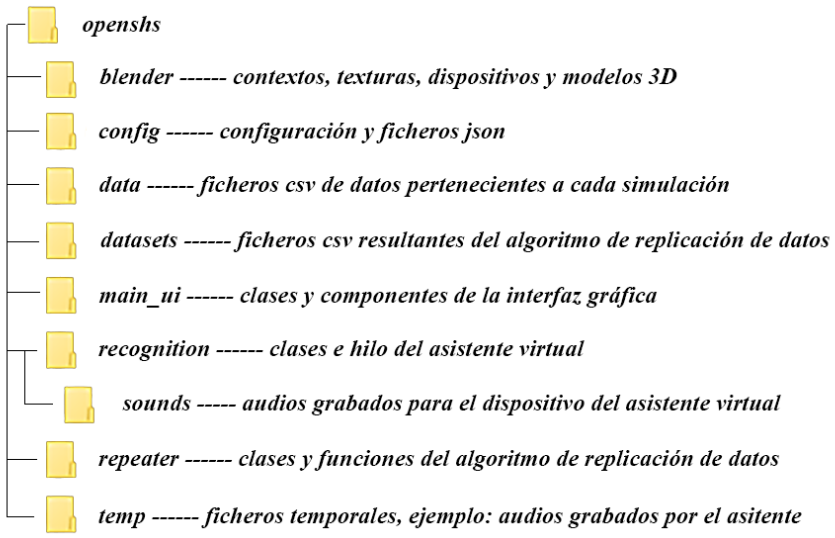

Figura 29: Estructura de directorios

5. Implantación

Una parte importante de cualquier trabajo o proyecto software es la distribución e implantación del mismo. En esta sección se explica las formas en las que el simulador se puede distribuir siguiendo el enfoque de la compatibilidad en múltiples plataformas.

El repositorio principal de *OpenSHS* no dispone de ninguna *Realese*, por ende, para poder usarlo se debe descargar el repositorio completo, luego instalar *Python 3* y *Blender 2.7* para ejecutar el fichero *openshs* desde la consola. Desde el punto de vista de un desarrollador realizar estos pasos no debe ser un problema, sin embargo para los investigadores que solo buscan usar el simulador y/o probar nuevos contextos puede ser un camino engorroso. Por ello se planteó en un inicio generar un fichero ejecutable con los directorios necesarios del simulador para almacenar los datos, ficheros con extensión ".blend" y configuraciones en formato ".json", esto sería posible mediante la herramienta de *PyInstaller*.

PyInstaller (*PyInstaller Manual*, 2008) empaqueta una aplicación implementada en *Python* junto con sus dependencias en un solo paquete. De esta forma el usuario puede ejecutar la aplicación empaquetada sin la necesidad de instalar el intérprete ni ningún módulo. Es compatible con versiones superiores a *Python 3.7* y agrupa correctamente muchos de los principales paquetes como *numpy*, *matplotlib*, *PyQt*, *wxPython*.

Acorde con la documentación oficial, *PyInstaller* soporta los sistemas operativos de *Windows*, *MacOS X* y *Linux*. Sin embargo, presenta una desventaja, no es un compilador cruzado; para hacer una aplicación en *Windows*, se debe empaquetar en Windows, y para hacer una aplicación de Linux, se empaqueta en Linux y así respectivamente. No obstante, se probó la generación de un ejecutable en *Windows 10* con los últimos cambios del trabajo y algunas vistas no mostraron los componentes pertenecientes al módulo *tkinter*. Por esas razonas se descartó su uso para empaquetar la parte del simulador implementada en *Python*.

Por consecuente, se optó por generar un fichero que fuese capaz de iniciar la interfaz gráfica del simulador como un proceso independiente usando el intérprete de *Python.* Para el caso de *Windows* se creó un fichero *openshs.py*, con el objetivo de empaquetarlo usando la herramienta mencionada anteriormente y generar un fichero *openshs.exe.* Para el caso de *Linux* se implementó un *script* que se ejecuta mediante el binario */bin/sh* para el arranque de la interfaz gráfica. En ambos casos basta con tener instalado *Python 3* y *Blender 2.7* en el sistema operativo, y ejecutar el fichero que lleva el nombre del simulador.

Por último, se comprimieron las versiones para *Windows* y *Linux* en formato *zip* para ser subido como *Release* en el repositorio de *Github* del presente trabajo (Figura 30).

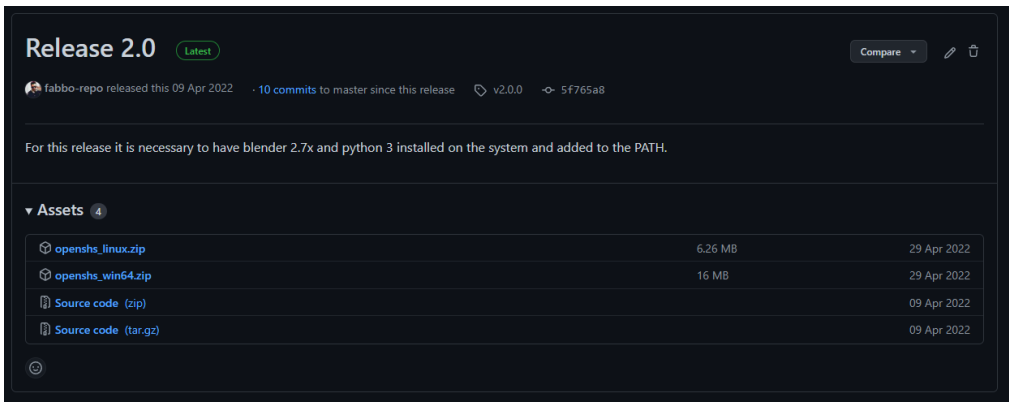

Figura 30: *Release 2.0* de *OpenSHS* ampliado

Hay que destacar que para otros sistemas operativos se puede utilizar la versión de *Linux* y ejecutar con el interprete de *Python* el fichero *ui.py*.

5.1. *Push request*

Al acabar los objetivos del trabajo y tener por ende el código listo, se envían los cambios al repositorio remoto de *Github*, que parte como base del repositorio principal de *OpenSHS*. Posteriormente se realizó una solicitud para añadir los cambios al repositorio remoto del simulador. De esta forma, los colaboradores de dicho repositorio podrán evaluar las extensiones, y si procede, incorporarlas al simulador original.

6. Manual de uso

En la presente sección se mostrará el manual de uso del simulador ampliado. Las imágenes y descripciones corresponden a la ejecución de la *release* del programa para el sistema operativo de *Windows* (*Windows 10* específicamente).

6.1. Variable de entorno

Una variable de entorno en el ámbito de los sistemas operativos corresponde a una variable dinámica que puede afectar al comportamiento de los procesos en ejecución en un ordenador ("Variable de entorno", 2022). Por ello para el correcto funcionamiento del simulador, y previo a su primer uso, se deben añadir un par de rutas a la variable de entorno *PATH*. Dicha variable de entorno contiene los directorios en los que el intérprete de comandos buscará archivos ejecutables que no se invocan con una ruta absoluta. Específicamente, se deben añadir las rutas de *Python* y *Blender*.

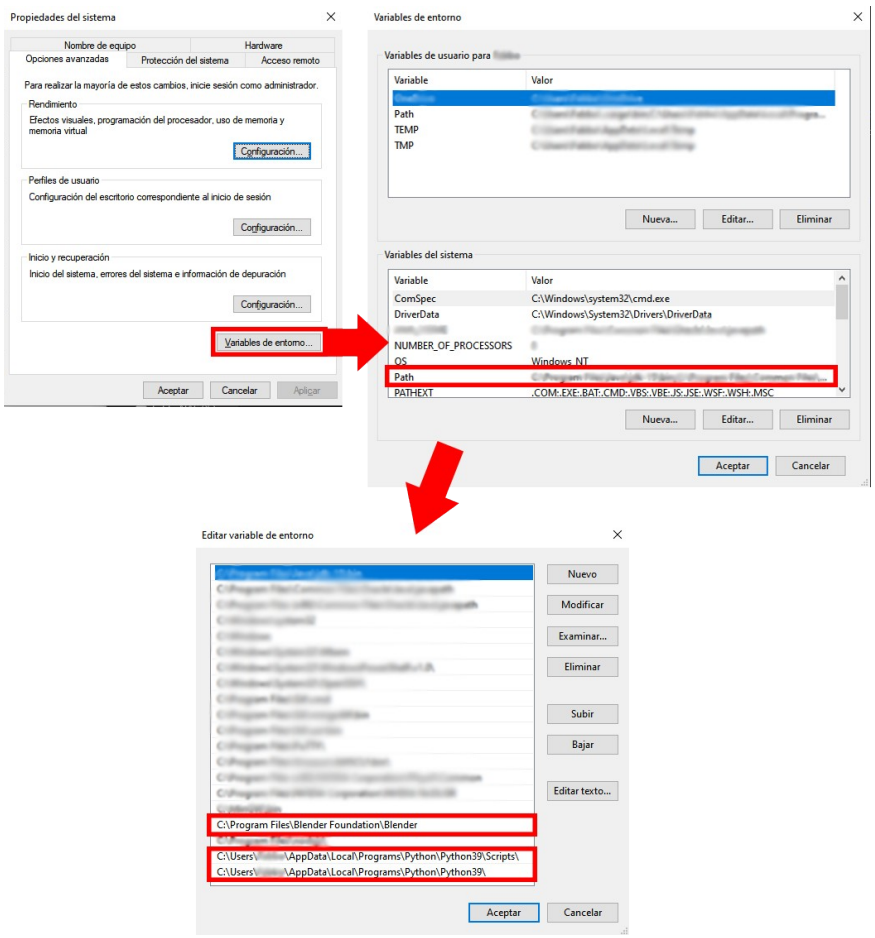

Figura 31: Proceso añadir rutas a la variable *PATH*

Al añadir las rutas siguiendo el proceso que se muestran en la Figura 31 se deben clickear los botones con el texto “Aceptar” de las ventanas abiertas al cerrarlas para que se apliquen los cambios.

6.2. Iniciar simulación

Al ejecutar el programa se abre la interfaz gráfica de usuario (Figura 9) y desde la barra de navegación en la opción de “Simulación” se puede acceder al modo de “Generación de datos” y al modo interactivo. Para lanzar cualquiera de estos modos, explicados en el apartado de “Interfaz gráfica de usuario”, se debe seleccionar uno de los contextos que se muestran en la interfaz. Si se lanza el modo interactivo se abrirá directamente el programa de *Blender* con el contexto seleccionado. Por otra parte, en el modo de “Generación de datos” se abrirá una ventana emergente que solicitará datos importantes para almacenar los datos generados durante la simulación. En la Figura 32 se muestra el proceso para lanzar la simulación del contexto de “tarde” seleccionando el primer modo.

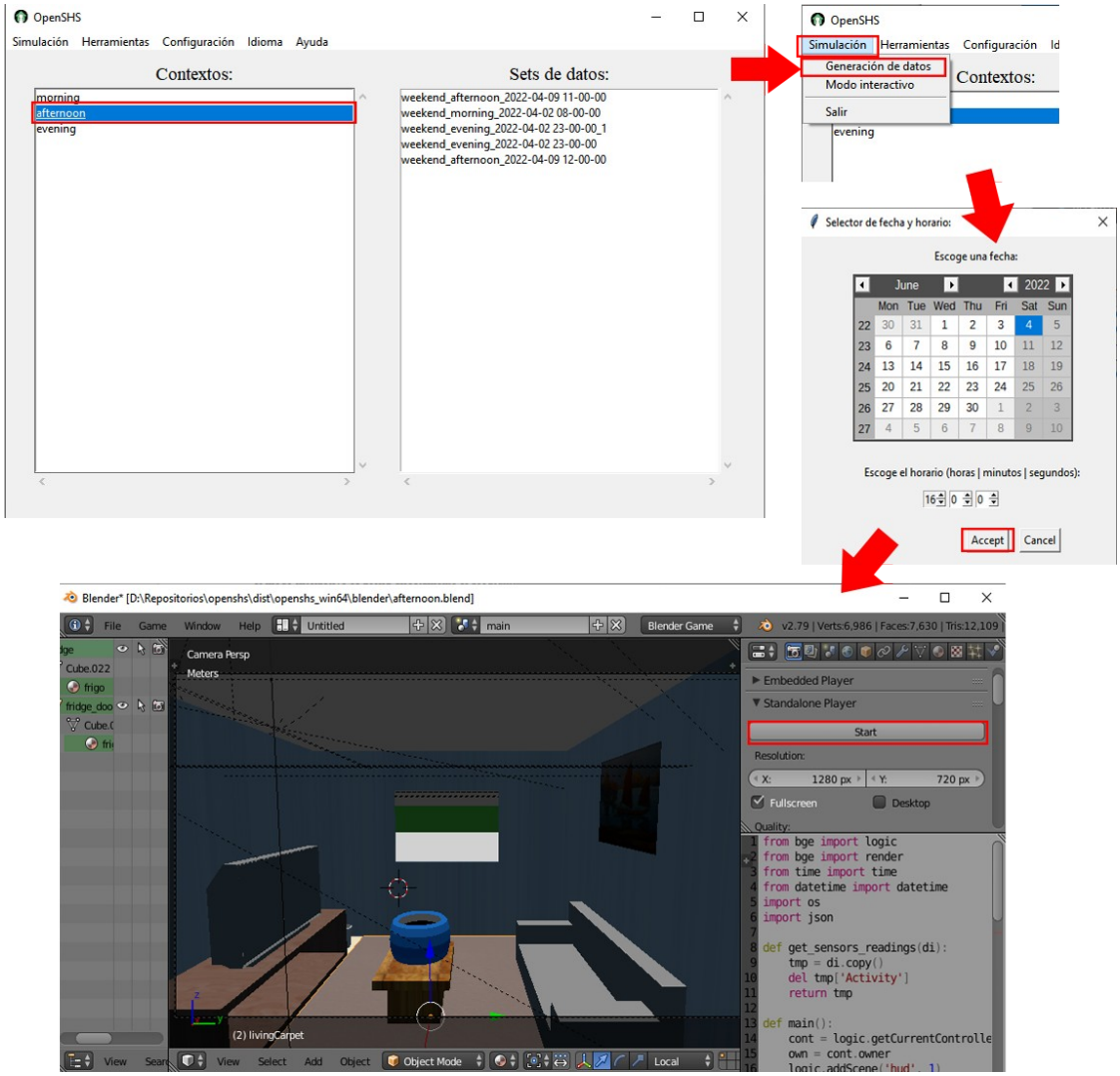

Figura 32: Inicio simulación en modo "Generación de datos"

Una vez iniciado el proceso de Blender se debe acceder al panel de propiedades. En la opción de renderizar, usando el ícono con forma de cámara, se debe pulsar el botón de "Start" dentro de la sección "Standalone Player" para iniciar la simulación. Es importante resaltar, que para mostrar la simulación en pantalla completa, la casilla de "Fullscreen" debe estar marcada, y si se requiere, se puede incrementar o reducir la resolución del renderizado (proceso de generar los *frames* a partir del modelo *3D*) que por defecto está configurada en 1280 x 720 pixeles.

Para acabar la simulación se debe presionar la tecla *ESC* del teclado. A continuación, se mostrará un mensaje con el nombre del fichero almacenado correspondiente a los datos generados. El nombre del fichero también se podrá encontrar en el panel principal de la interfaz gráfica.

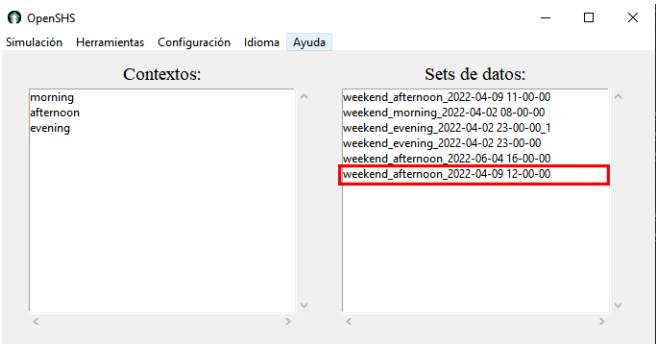

Figura 33: Datos almacenados

6.3. Dispositivos añadidos

Al iniciar la simulación en cualquiera de los contextos ("mañana", "tarde" y "noche") se pueden interactuar con los dispositivos incorporados en el presente trabajo:

- **Asistente virtual:** Sobre la mesa del salón se encuentra un dispositivo de color azul con forma de parlante. Al pasar el ratón por encima aparece un mensaje preguntando si se debe encender el asistente, al clickearlo se escucha un sonido de encendido y se activa el micrófono. A partir de este punto se le puede hablar pero solo responderá a aquellas frases que estén configuradas en el fichero *recognition.json*. Por ejemplo, si se dice "asistente, ¿qué hora es?" se escuchará una respuesta con la hora. Para evitar que el asistente siga activado, se debe volver a clickear sobre el dispositivo, escuchando en el proceso un sonido de apagado.

- **Consola de mensajes:** En la pared del salón cercana al pasillo se encuentra un dispositivo con forma de pizarra. Al insertar una frase en la variable *console_dev* del fichero *temp.json* se mostrará dicha frase en la consola de mensajes. Otra forma de mostrar los mensajes es mediante el asistente virtual, por ejemplo: al decir "asistente, escribe en la consola hola" se mostrará el mensaje "hola" en la consola.

- **Mapa de dispositivos:** En otra pared cercana a la consola de mensajes, se ubica el mapa de dispositivos. En dicho mapa se pueden apreciar unas figuras con forma de cilindro, al pasar el ratón por encima se mostrará un aviso preguntando si se desea

apagar o encender la luz que representa dicha figura. Al clickearla se apagará o encenderá la luz respectiva.

6.4. Replicar datos

Desde la barra de navegación en la opción de “Herramientas” se encuentra el algoritmo de replicación de datos. Al pulsar en la herramienta aparecerá una ventana solicitando los parámetros de entrada que requiere el algoritmo para su funcionamiento, uno de ellos es la fecha a la que pertenecen los conjuntos de datos almacenados en el directorio *data*. Una vez haya acbado de ejecutarse el algoritmo, el resultado se ubicará en el directorio *datasets*.

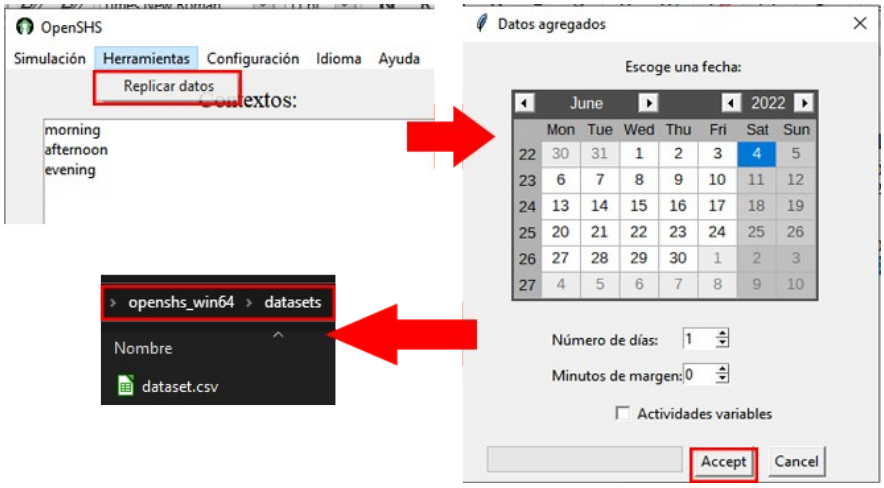

Figura 34: Ejemplo de replicación de datos

7. Pruebas

Dado que la versión del repositorio oficial del simulador *OpenSHS* no proporciona ningún *framework* para pruebas ni tests unitarios, se realizaron pruebas de *GUI* o también denominadas pruebas *end-to-end*. Al finalizar la implementación de una de las actividades en estado "Implementado" en la pizarra de *Trello* (Figura 1) se precedía a comprobar su correcto funcionamiento desde la interfaz gráfica de usuario solo si se trataba de una tarea relacionada con *Python*, para el caso de *Blender* bastaba con lanzar el respectivo contexto y añadir secciones *try-except* para identificar los errores en los controladores. A continuación se exponen más detalles sobre las pruebas manuales agrupadas por requisitos:

- **Requisito 1:** se puso a prueba la interfaz gráfica de usuario, comprobando que los componentes (botones, barra de navegación, elementos del menú, selector de fecha, entre otros) se mostrasen correctamente y cumplieran su función. Se probó que todas las ventanas de error o de solicitud de información aparecieran en la circunstancia correcta. Por último se comprobó que los contextos se cargasen correctamente y que la lista de conjuntos de datos se actualizase al finalizar un contexto.

- **Requisito 2 al 4:** Para el caso de los dispositivos añadidos se realizaron varias pruebas periódicas con diferentes ajustes en los ficheros de configuración correspondientes. Por ejemplo, para el asistente virtual se alteró el fichero *recognition.json* para identificar fallos en el reconocimiento. Con respecto a la consola de mensajes, se modificó en tiempo real el parámetro *console_dev* del fichero *temp.json* para comprobar que se actualizaba la pizarra en la simulación.

  En los casos que el controlador de uno de los dispositivos fallase o mostrase un comportamiento erróneo se añadió secciones de código con el objetivo de depurar escribiendo en un fichero *err.log* las excepciones durante la ejecución de la simulación.

```
try:
    ...
except:
    with open(os.path.join('temp','err.log'), 'w', encoding='utf-8') as log:
        log.write(traceback.format_exc())
```

Figura 35: Sección *try-except* para indentificación de excepciones

  Específicamente para el asistente virtual, se añadieron acciones a ejecutar que modificaron los ficheros *devices.json* y *temp.json*. Como resultado, se lograron alteraciones en los estados de algunos dispositivos, tanto en las luces como en el texto mostrado en la consola de mensajes. Como prueba del funcionamiento esperado en el mapa de dispositivos, bastó con pulsar los cilindros que representan las luces para comprobar que éstas se apagaban o encendían.

- **Requisito 5:** Al añadir las traducciones en la interfaz bastaba con ejecutar la interfaz y comprobar que los textos de los componentes se mostraban en el idioma indicado en la configuración. Este ajuste también afecta los contextos, por lo que se ejecutó cada uno y

se verificó que los dispositivos mostraran los mensajes en el idioma correcto al mantener el ratón encima de ellos.

- **Requisito 6:** Dado que este contexto parte de los ya proporcionados y validados en los requisitos anteriores, solo se comprobó el estado de los dispositivos durante la simulación y la existencia de la luz que representa el sol.

Adicionalmente, se comprobó que tanto el modo “Generación de datos” como la herramienta de replicación de datos generaban los ficheros en el directorio adecuado. Pero, para el caso del algoritmo no se comprobó si los datos almacenados eran los correctos, debido a que suponía extender el trabajo fuera de sus requisitos.

8. Conclusiones

Tomando en consideración los objetivos y requisitos propuestos en el inicio del presente trabajo, se puede afirmar que se pudo alcanzar la meta prevista. En especial, se debe destacar la consolidación lograda desde el punto de vista personal en las tecnologías y entornos presentes en el simulador, lo que permitió adquirir en el proceso, experiencia y conocimientos tanto en el lenguaje de programación *Python* como en la herramienta de diseño *3D* de *Blender*.

Gracias a la experiencia previa adquirida en el manejo de servicios relacionados con el reconocimiento automático de voz (*ASR*) y texto a voz (*TTS*) en la plataforma de *AWS*, específicamente con los servicios de *Amazon Lex* y *Polly*, se desarrollaron ampliaciones como la del asistente virtual, permitiendo comprender el trasfondo del diseño de muchos de los sistemas que integran este tipo de servicios.

Como resultado del proceso de la investigación realizada, se asimiló la trascendencia de la utilización de alternativas para la creación de conjuntos de datos sin la necesidad de montar sensores en un entorno realista. Por lo tanto, partiendo de que la generación de datos constituye un pilar para el desarrollo de servicios que ofrecen sistemas automatizados, también denominados *SMART*, el presente trabajo puede contribuir a los esfuerzos en el desarrollo de mejores tecnologías en el campo de la inteligencia artificial.

8.1. Relación del trabajo desarrollador con los estudios cursados

Para la realización del presente trabajo, han resultado imprescindibles los conocimientos adquiridos en el Grado de Ingeniería Informática, sobre todo aquellos propios de las asignaturas de la rama de Computación.

A continuación se realizará una recopilación de la influencia de estos estudios en el trabajo:

- Para la elección de la metodología a utilizar y la forma de llevar a cabo el desarrollo del trabajo, han servido de gran ayuda la documentación de la asignatura de Ingeniería del Software y las prácticas realizadas en la asignatura de Gestión de Proyectos.
- Gracias al proyecto realizado en las asignaturas de Sistemas de Almacenamiento y Recuperación de Información y Algorítmica, se han podido llevar a cabo las implementaciones en código *Python* sin contratiempos.
- En relación al diseño de la interfaz gráfica de usuario se han puesto en evidencia conocimientos adquiridos en la asignatura de Interfaces Persona Computador.
- Como base para el aprendizaje de la herramienta *Blender* se han referenciado algunos conceptos vistos en la asignatura de Sistemas Gráficos Interactivos.

Paralelamente, han sido necesarias una serie de competencias transversales estudiadas y vistas durante el grado. Aquellas con mayor relevancia y presentes en este trabajo son las siguientes:

- Análisis y resolución de problemas: una de las más recurrentes y presentes durante la realización de este trabajo. Gracias a ésta, se ha analizado la problemática a resolver, creando las soluciones a implementar y verificando el funcionamiento de las ampliaciones.

- Instrumental específica: Gracias a esta competencia se han analizado los entornos y tecnologías utilizados presentes en el simulador. Eligiendo la librería de *tkinter* para la implementación de la interfaz gráfica de usuario y el uso de la herramienta *Blender* para la creación de un nuevo entorno simulado.

- Aplicación y pensamiento práctico: Gracias a esta competencia se podido establecer una metodología y forma de trabajo para llevar cabo los objetivos. Ha permitido la toma decisiones para abordar los problemas y obstáculos durante el desarrollo.

9. Trabajos futuros

La realización de este trabajo se ha centrado en la implementación de los requisitos abordados en secciones previas. Sin embargo, durante el proceso de desarrollo se detectaron una serie de puntos o mejoras relevantes a incluir para futuras ampliaciones del propio simulador.

Como se ha mencionado en la sección de análisis, el simulador monta tanto la interacción como simulación sobre el motor de videojuegos de *Blender* (*BGE*), y desde la versión 2.8 este motor ya no es soportado. Por ende, resulta imprescindible cambiar el motor para futuras actualizaciones del simulador. Aunque actualmente el motor de *EEVEE* es el remplazo para *BGE*, se recomienda utilizar motores de videojuegos más populares y potentes como *Unity* o *Unreal Engine*.

En caso de que se pretenda mantener el motor de videojuegos de *Blender*, se propone la adición de nuevos contextos con múltiples plantas o pisos de vivienda. De esta manera, se enriquece el simulador, ampliando la variedad de entornos en los que se puede aplicar.

Por otra parte, en la implementación del asistente virtual se importa el paquete gratuito denominado *SpeechRecognition*. Este paquete no da buenos resultados si se compara con servicios de reconocimiento por voz, como el de *Amazon Lex*. Incluso, al usarlo con el idioma inglés, le cuesta reconocer varias palabras y frases. Por lo tanto, se recomienda diseñar un sistema que ofrezca mejores resultados para el reconocimiento de la voz para transformarla en texto. Adicionalmente, en relación con el asistente, se podrían añadir el soporte a conceptos de ranuras para utilizar variables durante el reconocimiento y un mejor sistema para la ejecución de acciones personalizadas.

El algoritmo de replicación de datos debería ser validado para futuras *realeases* dado que en el presente trabajo no se verificó su correcto funcionamiento. Cabe destacar que sí se corrigieron algunos errores de compilación.

Por último, se sugiere llevar a cabo una mejor implantación del simulador construyendo un solo binario para los sistemas operativos más populares. Dicho binario debería empaquetar todo el código *Python* ajeno a los contextos: la interfaz gráfica, el hilo del asistente virtual y el algoritmo de replicación. También se podría crear un instalador para ofrecer una mejor experiencia al usuario.

10. Bibliografía

Alshammari, N., Alshammari, T., Sedky, M., Champion, J., & Bauer, C. (2017). OpenSHS: Open Smart Home Simulator. *Sensors*, *17*(5). https://doi.org/10.3390/s17051003

Amazon Lex: Cómo funciona. (2022). En *Docs.aws.amazon.com*. https://docs.aws.amazon.com/es_es/lex/latest/dg/how-it-works.html

BER, R., Aarts, EHL., & Markopoulos, P. (2005). Ambient Intelligence Research in HomeLab: Engineering the User Experience. *Electronics Letters - ELECTRON LETT*. https://doi.org/10.1007/3-540-27139-2_4

Blender (software). (2022). En *Wikipedia*. https://en.wikipedia.org/wiki/Blender_(software)

Bouchard, K., Ajroud, A., Bouchard, B., & Bouzouane, A. (2010). *SIMACT: A 3D Open Source Smart Home Simulator for Activity Recognition*. 524-533. https://doi.org/10.1007/978-3-642-13577-4_47

Bruneau, J., Jouve, W., & Consel, Ch. (2009). *DiaSim: A parameterized simulator for pervasive computing applications*. 1-2. https://doi.org/10.4108/ICST.MOBIQUITOUS2009.6907

Cook, D. J., Crandall, A. S., Thomas, B. L., & Krishnan, N. C. (2013). CASAS: A Smart Home in a Box. *Computer*, *46*(7), 62-69. https://doi.org/10.1109/MC.2012.328

Fitzpatrick, M. (2022, febrero 4). *The complete PySide2 tutorial—Create GUI applications with Python*. PySide2 Tutorial 2022, Create Python GUIs with Qt. https://www.pythonguis.com/pyside2-tutorial/

Git. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Git

GitHub. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/GitHub

GNU Library General Public License, version 2.0. (2022, junio 18). GNU Library General Public License v2.0 - GNU Project - Free Software Foundation. https://www.gnu.org/licenses/old-licenses/lgpl-2.0.html

GTTS 2.2.4. (2022, marzo 15). gTTS · PyPI. https://pypi.org/project/gTTS/

Intille, S. S., Larso, K., Tapia, E. M., Beaudin, J. S., Kaushik, P., Nawyn, J., & Rockinson, R. (2006). *Using a Live-In Laboratory for Ubiquitous Computing Research*. 349-365. https://doi.org/10.1007/11748625_22

JSON. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/JSON

Kivy (framework). (2022). En *Wikipedia*. https://en.wikipedia.org/wiki/Kivy_(framework)

Lee, J. W., Cho, S., Liu, S., Cho, K., & Helal, S. (2015). Persim 3D: Context-Driven Simulation and Modeling of Human Activities in Smart Spaces. *IEEE Transactions on Automation Science and Engineering*, *12*(4), 1243-1256. https://doi.org/10.1109/TASE.2015.2467353

Lertlakkhanakul, J., Choi, J. W., & Kim, M. Y. (2008). Building data model and simulation platform for spatial interaction management in smart home. *Automation in Construction*, *17*(8), 948-957.

Nishikawa, H., Yamamoto, S., Tamai, M., Nishigaki, K., Kitani, T., Shibata, N., Yasumoto, K., & Ito, M. (2006). *UbiREAL: Realistic Smartspace Simulator for Systematic Testing*. 459-476. https://doi.org/10.1007/11853565_27

Nugent, C. D., Mulvenna, X. H., & S., D. (2009). *Experiences in the development of a Smart Lab*. *2*(4), 62-69.

Park, J., Moon, M., Hwang, M., & Yeom, K. (2007). *CASS: A Context-Aware Simulation System for Smart Home*. 461-467. https://doi.org/10.1109/SERA.2007.60

PyInstaller Manual. (2008, marzo 12). Pagina web: PyInstaller Manual -- PyInstaller 5.1 documentation. https://pyinstaller.org/en/stable/

Python. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Python

Resaltado especular. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Resaltado_especular

Sombreado de Phong. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Sombreado_de_Phong

Stallings, W. (2005). *Sistemas operativos: Aspectos internos y principios de diseño* (5.ª ed.). Pearson Prentice Hall.

Synnott, J., Chen, L., Nugent, C. D., & Moore, G. (2014). *The creation of simulated activity datasets using a graphical intelligent environment simulation tool*. 4143-4146.

https://doi.org/10.1109/EMBC.2014.6944536

Synnott, J., Nuent, Ch., & Jeffers, P. (2015). Simulation of Smart Home Activity Datasets. *Sensors (Basel, Switzerland)*, *15*, 14162-14179. https://doi.org/10.3390/s150614162

Tema: Manual de Blender parte v iluminación. (2008, marzo 12). [Blender] Manual de Blender parte v iluminación. https://www.foro3d.com/f111/manual-de-blender-parte-v-iluminacion-58830.html

Thompson, P. (2022, junio 18). *PyQt5 5.15.7*. PyQt5 · PyPI. https://pypi.org/project/PyQt5/

tkinter—Python interface to Tcl/Tk. (2022, junio 19). tkinter — Python interface to Tcl/Tk — Python 3.10.5 documentation. https://docs.python.org/3/library/tkinter.html

Variable de entorno. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Variable_de_entorno

Visual Studio Code. (2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Visual_Studio_Code

Yamazaki, T. (2007). The Ubiquitous Home. *International Journal of Smart Home*, *1*. https://www.researchgate.net/publication/228771570_The_Ubiquitous_Home

Zhang, A. (2017, diciembre 5). *SpeechRecognition 3.8.1*. SpeechRecognition · PyPI. https://pypi.org/project/SpeechRecognition/

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.		X		
ODS 12. **Producción y consumo responsables.**		X		
ODS 13. **Acción por el clima.**		X		
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

De los objetivos de desarrollo sostenibles anteriormente mencionados, el trabajo desarrollado está relacionado con los siguientes:

- **Salud y bienestar:** el trabajo se enfoca en la extensión de un simulador capaz de generar datos útiles para entrenar sistemas de casas inteligentes. Por lo que de forma indirecta promueve el bienestar para los individuos que viven en casas que utilicen este tipo de sistemas. Este bienestar incluye no solamente el confort que añade a la vida cotidiana de los individuos que la utilizan, incluido un aumento de la seguridad y fomento de la accesibilidad del entorno a personas con discapacidad y adultos mayores, sino también la prevención de accidentes domésticos (sobrecarga de la red, problemas de combustión de dispositivos del hogar, entro otras). Las ventajas señaladas influyen positivamente en la calidad de vida de los inquilinos de la casa y por la tanto en su salud.
- **Educación de Calidad:** considero que este trabajo contribuye a fortalecer el desarrollo y uso de software de código abierto al potenciar nuevos elementos en los proyectos de generación de conjuntos de datos para el campo de la inteligencia artificial. Por lo tanto incide directamente en una educación de calidad al alcance de más personas.
- **Industria, innovación e infraestructura:** el tema desarrollado en este trabajo está estrechamente relacionado con el avance en infraestructuras en especial de casas inteligentes. A su vez, se vincula con la domótica, tecnología también aplicada en la industria, por lo que de manera indirecta puede contribuir al desarrollo de proyectos industriales. De manera general, los proyectos vinculados con la domótica forman parte de su carácter innovador, al constituir una novedad en la forma de satisfacer las necesidades de las sociedades actuales.
- **Ciudades y comunidades sostenibles:** de manera indirecta, con este trabajo se busca la optimización de dispositivos y servicios del hogar que presuponen el uso de recursos algunos de los cuales son naturales. En este sentido, existe un relación directa con el objetivo de la sostenibilidad de comunidades y ciudades, enfocado en el uso responsable de los recursos naturales.
- **Producción y consumo responsables:** el fundamento del trabajo reside en la utilización de simuladores y ello evita el exceso de producción de equipos y dispositivos con sensores para la captura y generación de datos en un entorno realista. En este sentido, se actúa de manera consciente para lograr condiciones óptimas de producción y de consumo.

- **Acción por el clima:** los sistemas automatizados que aprenden de los datos generados por el simulador, pueden ser configurados para ser eficientes en cuanto a consumo energético. Al enfocarse en los sistemas para los hogares, este trabajo incide indirectamente en la optimización y por lo tanto disminución del consumo energético del hogar, que a su vez repercute en la reducción de contaminantes generados en la generación de energía, siendo como consecuencia favorable para la protección del medio ambiente y la preservación del clima del planeta.

- **Paz, Justica e Instituciones sólidas:** el uso de software de código abierto ha estado siempre unido al proceso creativo libre de condicionamientos económicos, y por lo tanto, más próximo a ideales de justicia y paz social. En este sentido, como premisa de este trabajo se parte de un simulador de código abierto.
- **Alianzas para lograr objetivos:** el trabajo desarrolla un tema de particular interés para los investigadores de proyectos relacionados con la inteligencia artificial, que puede relacionarse de manera indirecta con la promoción de alianzas de actores del campo de la investigación, del mundo empresarial y de la sociedad civil. Eso está dado porque los expertos desarrolladores de software con objetivos esencialmente profesionales coinciden, y en ocasiones se favorecen, del interés especial de las empresas en la innovación para buscar ventajas competitivas. Paralelamente, los consumidores se sienten atraídos cada vez más por servicios inteligentes que les permite fomentar su ahorro. La confluencia de los intereses de estos actores posibilita el desarrollo de alianzas a favor de la promoción de sistemas de hogares inteligentes.