Universidad Zaragoza
1542

Trabajo Fin de Grado

Gestión de escenarios para la evaluación de sistemas de recomendación
Management of scenarios for the evaluation of recommendation systems

Autor/es

Álvaro Andrada Gimeno

Director/es

Sergio Ilarri Artigas

Escuela de Ingeniería y Arquitectura

2024

AGRADECIMIENTOS

Para empezar, me gustaría agradecer al director del proyecto, Sergio Ilarri, por su orientación y ayuda proporcionada durante la elaboración del mismo.

También quiero agradecer a mi familia por su apoyo durante este largo proceso, han sido una auténtica motivación para superar los momentos más complicados, y sin ellos no habría podido. Por supuesto, también a mis amigos por su gran apoyo.

Por último, hay que agradecer al proyecto de investigación NEAT_AMBIENCE (PID2020-113037RB-I00 / AEI / 10.13039/501100011033) financiado por la Agencia Estatal de Investigación, además de al apoyo del Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón (Gobierno de Aragón: referencia de grupo T64_23R, grupo COSMOS).

Gestión de escenarios para la evaluación de sistemas de recomendación

Resumen

En el mundo moderno la información es cada vez más útil en muchos aspectos de la sociedad. En particular, datos asociados con personas, por lo que cosas como los sistemas de recomendación cobran cada vez más importancia. Un sistema de recomendación es una herramienta que realiza predicciones de recomendaciones de elementos que pueden ser de interés para los usuarios.

Los Sistemas de Recomendación Dependientes del Contexto o CARS (Context-Aware Recommender Systems), como su propio nombre indica, trabajan con contextos además de con los datos de usuarios y los elementos potencialmente de interés (ítems). Ese contexto puede ser un lugar, fechas, otras personas,…

En este proyecto se trabaja con un simulador cuyo objetivo es evaluar distintos CARS. El simulador, RecMobiSim, trabaja con escenarios como contexto. Lo que se ha realizado en el proyecto ha sido añadir funcionalidades que permiten gestionar mejor los escenarios o mapas, y se ha mejorado la usabilidad de la herramienta.

Se ha implementado compatibilidad con ficheros SVG que representen escenarios, tanto para el editor de escenarios de RecMobiSim como para hacer simulaciones sobre ellos.

Otro elemento que se ha añadido es la posibilidad de elegir en qué escenario se va a realizar una simulación.

Se ha añadido la posibilidad de hacer gestiones con una base de datos espacial en la que se almacenan datos de los propios escenarios y información de traza de simulaciones realizadas. También se pueden hacer consultas de forma sencilla.

También se pueden gestionar grafos de conocimiento creados en Neo4j con datos de mapas, siendo especialmente útil para realizar consultas sobre ítems. Además, se puede visualizar el aspecto del grafo desde la poca aplicación.

Por último, se han añadido tablas de información sobre los usuarios simulados que se pueden consultar durante una simulación para ver esa información en tiempo real.

Se han realizado pruebas de las nuevas funciones sobre un escenario de un lugar real para ver el nivel de mejora en la experiencia de uso.

Índice

1. Introducción

Este proyecto está basado en RecMobiSim [1], un simulador que sirve para evaluar CARS (Sistemas de Recomendación Dependientes del Contexto) sobre escenarios móviles. El simulador incluye un editor de mapas para crear o modificar escenarios propios. El programa se creó en el contexto del proyecto NEAT-AMBIENCE [2] de la Universidad de Zaragoza, el cual está orientado a abordar problemas de gestión de datos de las personas en su vida diaria, respetando los principios de cuidado del medio ambiente y limitación del consumo de recursos.

1.1 Objetivos

El proyecto tiene el objetivo de iterar sobre el trabajo hecho en RecMobiSim [1]. Lo que se quiere conseguir es desarrollar la herramienta para mejorar su usabilidad al usar sus funcionalidades. Para empezar, se va a modificar el editor de escenarios que incluye el programa para que pueda importar mapas en ficheros SVG, ya que previamente solo se podían abrir mapas creados por el propio editor y almacenados en ficheros txt con un formato específico. Por otro lado, también se podrán exportar mapas creados o modificados en el editor en formato SVG.

Otro nuevo añadido importante es la posibilidad de establecer una conexión con una base de datos espacial (PostgreSQL + PostGIS) a través de la cual realizar distintas funciones (almacenar datos de mapas, consultas, registro de información de trazas de simulaciones…).

De forma similar, también se permitirá la gestión de grafos de conocimiento con información de los escenarios a través de una conexión con una base de datos Neo4J, incluyendo la posibilidad de visualizar esos grafos en la propia aplicación.

Además, se han añadido otras mejoras para una mejor experiencia de usuario, así como algunas correcciones de bugs ya presentes en RecMobSim.

1.2 Metodología

RecMobiSim está escrito con el lenguaje de programación Java.

Además, durante el proyecto se han utilizado distintas herramientas:

- Swing [3]: Para crear las interfaces gráficas de la aplicación para las funciones nuevas y para modificar las existentes.

- Batik [4]: Para crear archivos SVG se ha utilizado la biblioteca de manipulación de gráficos SVG Batik, la cual permite generar un archivo SVG y añadirle elementos con sus etiquetas y atributos de forma sencilla.

- PostgreSQL con extensión PostGIS [5]: Para almacenar datos de escenarios, incluyendo datos geométricos (como los polígonos que forman las paredes de las habitaciones, o las localizaciones de puertas y otros elementos), así como datos de simulaciones realizadas (y realizar consultas sobre esos datos. Para la conexión con Java se ha utilizado su JDBC.

- Neo4j [6]: Para almacenar datos de escenarios en forma de grafos. Para la conexión con Java se ha utilizado su JDBC.

- GraphStream [7]: Biblioteca de Java de gestión de grafos usada para visualizar los grafos de escenarios.

- Apache Ant [8]: Herramienta usada para compilar el programa en un jar ejecutable.

- Figma [9]: Editor de gráficos vectoriales usado para dibujar el mapa del ejemplo de uso del proyecto.

1.3 Contenido de la memoria

El documento está estructurado de la siguiente manera. El apartado 2 está dedicado a la nueva compatibilidad del RecMobiSim con ficheros SVG, tanto al exportar como al importar del editor. El apartado 3 está dedicado a la conexión con PostgreSQL con PostGIS y todas las funciones que aprovechan esa conexión para usar una base de datos. El apartado 4 se centra en la gestión de grafos con Neo4j, abordando su creación, diseño, y visualización, así como su utilidad para obtener cierta información de los escenarios. El apartado 5 se dedica a mejoras relacionadas con la realización de simulaciones. El apartado 6 se dedica a otras funciones que no encajaban en los otros apartados. Después está el apartado 7, que es la conclusión del proyecto, y el 8 presenta el posible trabajo futuro en relación a este proyecto.

También se incluyen anexos, que contienen fragmentos de código aludidos en la memoria principal, instrucciones para realizar simulaciones, y pruebas realizadas en un escenario real.

1.4 Explicaciones sobre escenarios en RecMobiSim

Un escenario/mapa de RecMobiSim está compuesto por:

- Esquinas: juntas componen las paredes del escenario.

- Habitaciones: formadas por esquinas conectadas que forman paredes contiguas que delinean una zona cerrada en el mapa

- Puertas: conectan habitaciones en la misma planta.

- Escaleras: conectan habitaciones en distintas plantas.

- Separadores: separan una habitación en subhabitaciones. Son líneas recta cuyos extremos son dos de las esquinas de la habitación.

- Subhabitaciones: secciones de una habitación separadas por un separador. Las subhabitaciones están conectadas por 2 puertas invisibles colocadas a ambos lados del separador.

- Ítems: representan elementos visitables en el mapa por los usuarios simulados. Pueden ser establecimientos como tiendas o restaurantes, cuadros o estatuas, o cualquier otra cosa de interés. Contienen varios atributos: título,item label (esto es una cadena que sirve como categoría, como por ejemplo "Restaurante", "Cuadro", "Tienda de electrónica",…) ,fecha de apertura (para visitas), fecha de cierre, nacionalidad, altura, anchura y fecha histórica en la que se creó.

En el código de RecMobiSim, en el editor de escenarios existe un "modelo" que contiene todos los datos (los elementos mencionados) del mapa que está cargado en el editor en cada momento, y se lee o modifica para realizar cualquier función con el editor. Además, el mapa que se visualiza en el editor es el formado por todos los elementos que contiene el modelo.

2. SVG

El editor de mapas de RecMobiSim permite crear mapas en los cuales poder realizar simulaciones para evaluar sistemas de recomendación.

Esos mapas originalmente se podían exportar como ficheros de texto. Concretamente, en estos 3 ficheros:

- room_floor_combined.txt: datos de configuración de mapa y localización de esquinas, puertas y escaleras.
- graph_floor_combined.txt: etiquetas de elementos (números de identificación de habitaciones, objetos, puertas y escaleras), relaciones entre elementos (objetos, puertas y escaleras dentro de habitaciones, conexiones de puertas y escaleras entre sí).
- item_floor_combined.txt: objetos con sus atributos.

Estos 3 ficheros (los correspondientes a un solo escenario) tienen que estar en el mismo directorio y no se les puede cambiar el nombre para ser útiles, ya que para volver a abrir un mapa con el editor y modificarlo el programa busca los 3 ficheros con esos 3 nombres en el directorio indicado. Para la simulación también se buscan esos 3 ficheros en un directorio concreto.

Los ficheros son ficheros de propiedades, lo que quieren decir están compuestos por filas de parejas clave-valor.

```
map_width=1700
name=Painting and Sculpture I and II
room_number_door_19=2
room_number_door_18=3
room_number_door_17=2
room_number_door_16=2
room_number_door_15=4
room_number_door_14=1
```

Este modelo más complicado de lo que debería, ya que hay que estar pendientes de 3 ficheros, y esos ficheros solo se pueden generar con RecMobiSim, por que que si se quiere realizar una simulación en un lugar concreto se tiene que crear de cero en el editor. Por esta razón, se ha decidido implementar la compatibilidad del programa con ficheros SVG, tanto al exportar como al importar (tanto ficheros creados con el editor como otros ficheros SVG que cumplan con ciertos requisitos) en el editor, y también al realizar las simulaciones.

Un SVG, además de poder contener de forma organizada la información en un solo fichero, tiene una utilidad más allá de contener información, ya que es un formato de gráficos vectoriales que se pueden visualizar, por lo que además es más fácil que, al necesitar el plano de algún lugar/edificio concreto, exista ya algún archivo SVG o fácilmente transformable en SVG de ese plano, y se pueda adaptar rápidamente a los requisitos necesarios para abrirlo con RecMobiSim.

Al visualizarse un archivo SVG exportado, será idéntico a ese mismo mapa visualizado en el editor, pero también contendrá otros datos no visibles relevantes (usados durante las simulaciones) incluidos en las etiquetas de los elementos del archivo (los datos que se incluyen en los 3 ficheros txt de propiedades).

Para realizar la importación y exportación, se ha añadido al menú File de la barra de menú del editor 2 menús desplegables, “Open” y “Save as”, los cuales tienen dos opciones cada uno, importar/exportar con ficheros de texto o con fichero SVG. Al importar un SVG, se selecciona un fichero .svg, a diferencia de al importar los ficheros de texto, donde se tiene que seleccionar un directorio que contenga los 3 ficheros de texto con los nombres mencionados anteriormente.

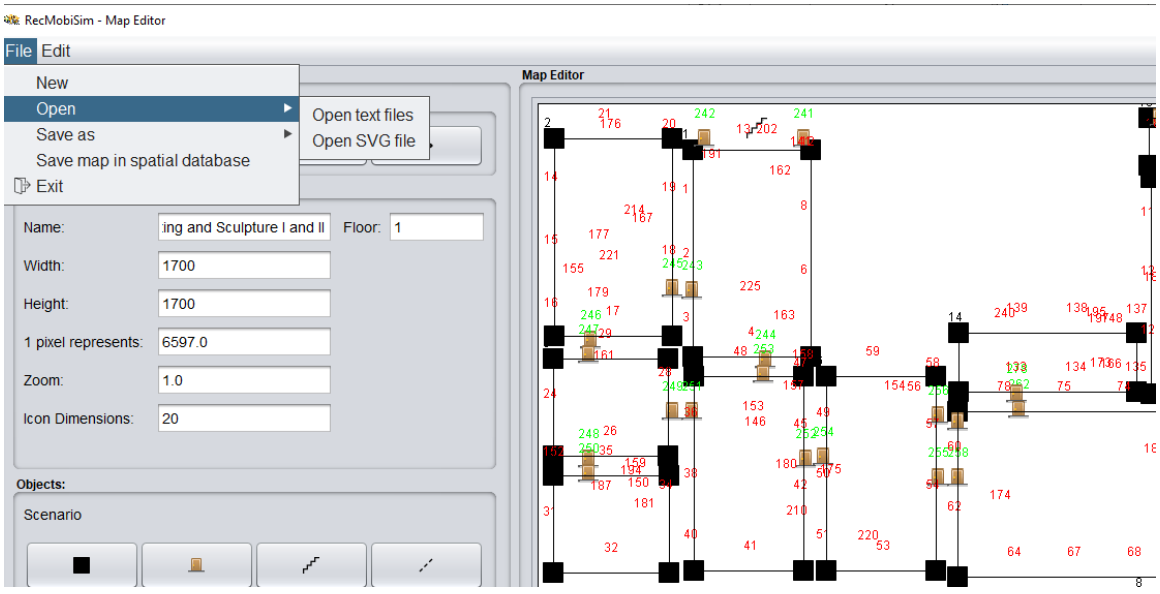


Ilustración 1 - Menú "Open" (importar)

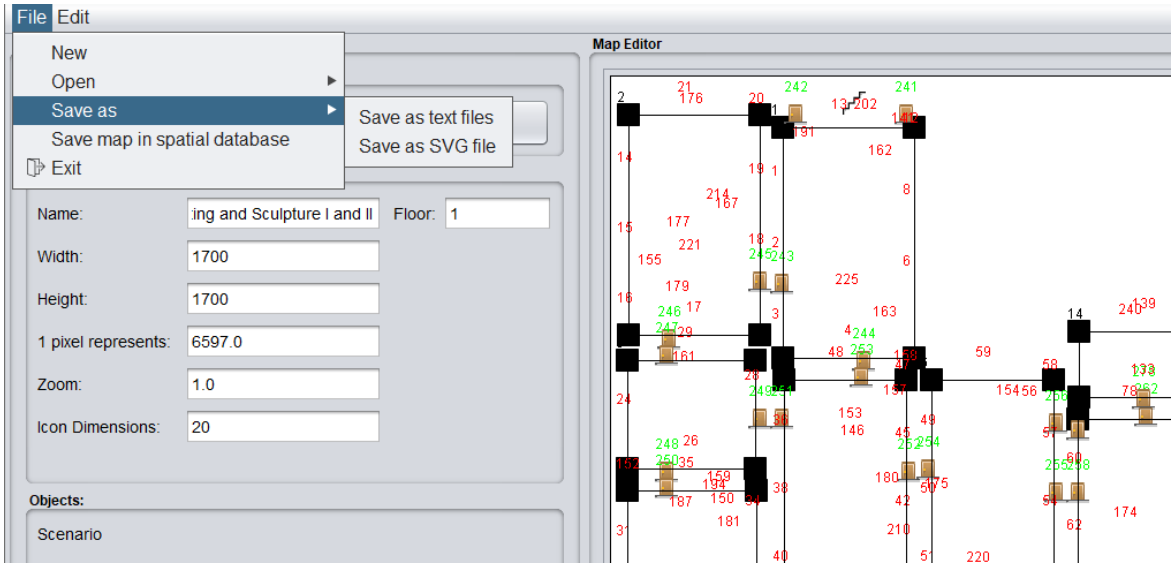


Ilustración 2 - Menú "Save as" (exportar)

2.1 Exportación en SVG

El código que crea los ficheros SVG se encuentra en una nueva clase SVGCreator, la cual sólo tiene un método llamado *crear* (al cual se llama al hacer click en el botón para guardar SVG). De la misma forma, el código que lee los archivos SVG está en la clase SVGParser, en su único método *parse* (al cual se llama al hacer click en el botón para abrir SVG).

Para crear los SVG, se ha utilizado la biblioteca de manipulación de gráficos SVG Batik, la cual permite generar un archivo SVG y añadirle elementos con sus etiquetas y atributos de forma sencilla.

```
impl = SVGDOMImplementation.getDOMImplementation();
svgNS = SVGDOMImplementation.SVG_NAMESPACE_URI;
doc = impl.createDocument(svgNS, "svg", null);
```

Ilustración 3 - Creación de documento SVG

Al usar el botón “Save as SVG file”, se crea un SVG vacío y se itera sobre cada elemento importante del escenario, creando un elemento XML para cada uno y colocando los datos relevantes en sus atributos.

```
svgRoot = doc.getDocumentElement();
Element im = doc.createElementNS(svgNS, "image");
im.setAttributeNS(null, "href", d.getUrlIcon());
im.setAttributeNS(null, "x", Integer.toString((int) (d.getVertex_xy().getX() * model.getZOOM())));
im.setAttributeNS(null, "y", Integer.toString((int) (d.getVertex_xy().getY() * model.getZOOM())));
im.setAttributeNS(null, "width", Integer.toString((int) (model.getDRAWING_ICON_DIMENSION() * model.getZOOM())));
im.setAttributeNS(null, "height", Integer.toString((int) (model.getDRAWING_ICON_DIMENSION() * model.getZOOM())));
im.setAttributeNS(null, "style", "background-color:red");
im.setAttributeNS(null, "label", Integer.toString((int) d.getVertex_label()));
svgRoot.appendChild(im);
```

Ilustración 4 - Introducción al documento de un elemento "image" con varios atributos (para representar una puerta)

De esta forma, al visualizar el fichero de forma externa a RecMobiSim, el mapa que se ve es idéntico al que se ve en el editor de escenarios, y también incluirá todos los datos para las simulaciones que no afectan al aspecto de los gráficos.

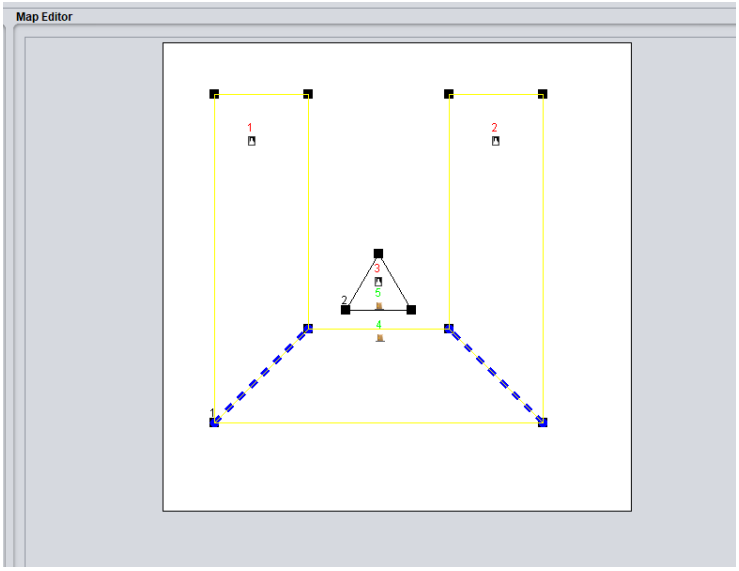


Ilustración 5 - Mapa en editor

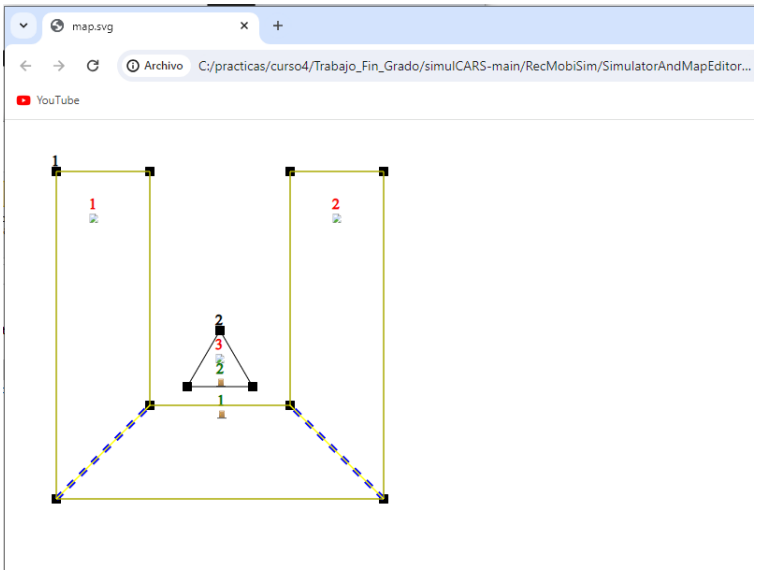


Ilustración 6 - Fichero SVG abierto en Chrome

Un ejemplo de cómo está estructurada la información del modelo en el SVG se puede ver en la siguiente imagen:

```
><line roomLabel="63" labelCorner1="1" labelCorner2="2" x1="1073.5" x2="1103.0" y1="1064.0" y2="1038.0" style="stroke:rgb(0,0,0);stroke-width:1"
/><line roomLabel="63" labelCorner1="2" labelCorner2="3" x1="1103.0" x2="1131.0" y1="1038.0" y2="1045.0" style="stroke:rgb(0,0,0);stroke-width:1"
/><line roomLabel="63" labelCorner1="3" labelCorner2="4" x1="1131.0" x2="1131.0" y1="1045.0" y2="1069.0" style="stroke:rgb(0,0,0);stroke-width:1"
/><line roomLabel="63" labelCorner1="4" labelCorner2="5" x1="1131.0" x2="1107.5" y1="1069.0" y2="1078.5" style="stroke:rgb(0,0,0);stroke-width:1"
/><line roomLabel="63" labelCorner1="5" labelCorner2="1" x1="1107.5" x2="1073.5" y1="1078.5" y2="1064.0" style="stroke:rgb(0,0,0);stroke-width:1"
/><text style="stroke:black" x="1070" y="1060"
>63</text
```

Ilustración 7 - Ejemplo de contenido de SVG

En la imagen, se ven varias líneas (elementos “line”) contiguas que forman una habitación cerrada. Aparte de las coordenadas y el estilo, cada uno de esos elementos tiene otros atributos “personalizados” que no afectan a la visualización del SVG pero son necesarios para el correcto funcionamiento del editor (al abrir el archivo SVG) y el simulador (al elegir un escenario de un archivo SVG sobre el que hacer una simulación). Otros elementos, como los elementos “image” que representan ítems, puertas y escaleras, o líneas discontinuas que representan separadores, también tienen sus propios datos invisibles.

2.2 Importación en SVG

Para importar un documento SVG, no se utiliza ninguna biblioteca externa de Java. Lo que se ha hecho en este case es parsear el documento con DocumentBuilderFactory y normalizarlo para crear el árbol de nodos/elementos, a los cuales se acceden después para leer la etiqueta y atributos de cada elemento del documento.

```
builder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document doc = builder.parse(new File(filePath));
doc.getDocumentElement().normalize();
```

Ilustración 8 - Parsear documento SVG

A continuación se obtiene el primer elemento del documento (el de etiqueta “svg”, que contiene el resto de elementos), y se van recorriendo sus hijos, los cuales se clasifican según su etiqueta (los elementos “line” de línea continua se consideran paredes, los de línea discontinua se consideran separadores de subsalas, y las imágenes se consideran simplemente imágenes).

```
Node first = doc.getElementsByTagName("svg").item(0);
nodeList = first.getChildNodes();
int n = nodeList.getLength();
while(i < n) {
    Node node = nodeList.item(i);
    if(node.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element)node;
        //if(e.getNodeName() == "line" && e.getAttribute("roomLabel").
        if(e.getNodeName() == "line") {
            if(e.hasAttribute("stroke-dasharray")) separators.add(e);
            else walls.add(e);
        }
        else if(e.getNodeName() == "image") images.add(e);
    }
    i++;
}
```

Ilustración 9 - Clasificación de elementos

Luego, esos elementos recolectados se convertirán poco a poco en componentes del modelo. Lo primero que se hace para conseguir eso es añadir las paredes del mapa al editor. Como se ha dicho, las líneas continuas se consideran paredes de las habitaciones del mapa. De cada una de esas líneas, se obtiene la posición de sus dos extremos, las esquinas, que se añaden al modelo del mapa del editor.

Para cada pared (el elemento line de la pared es la variable e):

```
double c1_x = (Double.parseDouble(e.getAttribute("x1"))*2-model.getDRAWING_ICON_DIMENSION())/2;
double c1_y = (Double.parseDouble(e.getAttribute("y1"))*2-model.getDRAWING_ICON_DIMENSION())/2;
double c2_x = (Double.parseDouble(e.getAttribute("x2"))*2-model.getDRAWING_ICON_DIMENSION())/2;
double c2_y = (Double.parseDouble(e.getAttribute("y2"))*2-model.getDRAWING_ICON_DIMENSION())/2;

Point p1 = new Point(c1_x,c1_y);
Point p2 = new Point(c2_x,c2_y);
Corner corner1, corner2;
corner1 = new Corner(null, e.getAttribute("labelCorner1").equals("") ? (0) : Long.parseLong(e.getAttribute("labelCorner1")), p1);
corner2 = new Corner(null, e.getAttribute("labelCorner2").equals("") ? (0) : Long.parseLong(e.getAttribute("labelCorner2")), p2);
if(!wallCollidesCorners(corner1,corner2,wallsPoints)) {
    wallsPoints.add(new PointPair(p1,p2));
    if (findCorner(corners,c1_x,c1_y) == null && model.addCorner(corner1)) {
        corners.add(corner1);
    }else {
        errorList.add("COULDN'T ADD CORNER TO MAP");
    }
    if (findCorner(corners,c2_x,c2_y) == null && model.addCorner(corner2)) {
        corners.add(corner2);
    }else {
        errorList.add("COULDN'T ADD CORNER TO MAP");
    }
}else {
    errorList.add("Walls colliding");
}
```

Ilustración 10 - Añadido de esquinas de paredes del SVG al modelo

Después, se buscan habitaciones completas (esquinas que formen habitaciones cerradas) entre las paredes que se han leído, y se añaden las habitaciones al modelo del mapa del editor. Más concretamente, lo que se hace es, una vez que se tienen todas las esquinas del mapa guardados en una lista, se vuelven a recorrer las paredes una a una, y con un algoritmo se busca la lista de paredes contiguas que forman una habitación.

```
List<Element> lookForRoom(List<Element> SVGElementList, Element startLine) {
    int n = SVGElementList.size();
    Element line = SVGElementList.get(0);
    List<Element> room = new ArrayList<Element>();
    boolean foundAdjWall = false;
    boolean finished = false;
    Element currentLine = startLine;
    Element firstLine = startLine;
    Element lastLine = startLine;
    room.add(startLine);
    while(!finished){
        int i = 0;
        foundAdjWall = false;
        while(!foundAdjWall && i < n) {
            line = SVGElementList.get(i);
            if(line.getNodeName() == "line") foundAdjWall = areAdjacentWalls(currentLine,line,lastLine);
            i++;
        }
        if(foundAdjWall){
            if(line == firstLine) {
                finished = true;
            }else {
                room.add(line);
                lastLine = currentLine;
                currentLine = line;
            }
        }else{
            finished = true;
        }
    }
    if(foundAdjWall){
        Element e = room.remove(0);
        room.add(e);
        return room;
    }else{
        return null;
    }
}
```

Ilustración 11 - Algoritmo que busca habitación de la que forma parte una pared (startLine es la pared, SVGElementList es la lista de todas las paredes)

Con la lista de paredes contiguas de una habitación, se buscan de entre las esquinas guardadas las esquinas de esa habitación, y con esas esquinas de crea la habitación, que se guarda en el modelo del editor.

A continuación, con los separadores de habitaciones en subhabitaciones se buscan las esquinas (de entre las previamente creadas que ahora están en el modelo) que se encuentran en los dos extremos de la línea discontinua, y la habitación que separa (de entre las previamente creadas), y con esos elementos se crea y añade cada separador al modelo.

Luego, con las imágenes se hace algo distinto. Si la imagen es del icono de una puerta o escalera, se crea y añade la puerta/escalera al modelo. Si no lo es, y tampoco de una esquina, quiere decir que es de un objeto o “ítem”, que representa un lugar de interés que los usuarios simulados pueden visitar en el mapa. Por tanto, se crea y añade el ítem usando todos los atributos que contiene el elemento en el documento como atributos de ese ítem.

Por último, se recorre cada conectable (puertas y escaleras) y se conecta con cada conectable al que está conectado según el documento.

3. Base de datos espacial

Otra de las aportaciones a RecMobiSim que se ha realizado en este TFG es posibilitar la conexión con un gestor de base de datos espaciales (PostgreSQL con la extensión PostGIS) y usarla para implementar acciones que posibiliten un mayor control sobre los mapas y sus datos.

Para empezar, se añadió un botón “Database connection” a la barra de menú de la pantalla principal del simulador, el cual al ser pulsado abre una ventana con un formulario que correctamente rellenado permitirá establecer una conexión con una base de datos PostgreSQL.

Ilustración 12 - Botón "Database connection"

El formulario pide el puerto donde esté escuchando PostgreSQL, un usuario de PostgreSQL, la contraseña de ese usuario y el nombre de una base de datos a la que tenga acceso ese usuario. Al pulsar el botón “Connect”, si los datos son correctos, se realizará una conexión con esa base de datos.

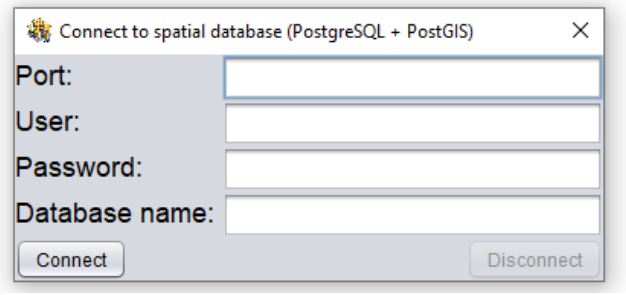


Ilustración 13 - Formulario de conexión con DB

Una vez establecida la conexión, se permitirá el acceso a las mencionadas nuevas funciones de la aplicación. Además, se crearán las tablas necesarias para el funcionamiento de RecMobiSim al hacer una conexión por primera vez a una base de datos PostgreSQL (o sea, si las tablas no existen).

El esquema de las tablas es:

MAP
ID INT PRIMARY KEY
NAME VARCHAR(200) NOT NULL UNIQUE
WIDTH INT NOT NULL
HEIGHT INT NOT NULL
PIXEL_REPRESENTS_IN_METERS DOUBLE PRECISION NOT NULL -> los metros que representa en la vida real cada píxel del mapa
DRAW_ICON_DIMENSION INT NOT NULL -> tamaño de iconos en mapa (altura y anchura)

ROOM
LABEL INT
MAP INT REFERENCES MAP(ID) ON DELETE CASCADE
GEOM GEOMETRY(POLYGON) NOT NULL -> geometría de habitación
PRIMARY KEY(LABEL,MAP)

SUBROOM_SEPARATOR
LABEL DECIMAL
ROOM_LABEL INT
MAP INT
GEOM GEOMETRY(LINESTRING) NOT NULL -> geometría de separador
FOREIGN KEY(ROOM_LABEL,MAP) REFERENCES ROOM(LABEL,MAP) ON DELETE CASCADE
PRIMARY KEY(LABEL,ROOM_LABEL,MAP)

ITEM
ID DECIMAL
MAP INT
ROOM_LABEL INT
LOCATION GEOMETRY(POINT) NOT NULL
URL_IMAGE TEXT NOT NULL -> URL o directorio de la imagen del icono en el mapa
TITLE TEXT NOT NULL
WIDTH DOUBLE PRECISION
HEIGHT DOUBLE PRECISION
NATIONALITY TEXT
BEGIN_DATE TIME
END_DATE TIME
DATE TEXT -> período histórico
ITEM_LABEL TEXT -> categoría
FOREIGN KEY(ROOM_LABEL,MAP) REFERENCES ROOM(LABEL,MAP) ON DELETE CASCADE
PRIMARY KEY(ID,MAP)

CONNECTABLE
ID DECIMAL
MAP INT
ROOM_LABEL INT
LOCATION GEOMETRY(POINT) NOT NULL -> localización en el mapa
TYPE CONN_TYPE NOT NULL -> 'DOOR' o 'STAIRS'
URL_IMAGE TEXT NOT NULL -> URL o directorio de la imagen del icono en el mapa
FOREIGN KEY(ROOM_LABEL,MAP) REFERENCES ROOM(LABEL,MAP) ON DELETE CASCADE
PRIMARY KEY(ID,MAP,TYPE)

CONNECTION
ID SERIAL PRIMARY KEY
ID_CONN1 DECIMAL
MAP_CONN1 INT
TYPE_CONN1 CONN_TYPE
ID_CONN2 DECIMAL
MAP_CONN2 INT
TYPE_CONN2 CONN_TYPE
FOREIGN KEY(ID_CONN1,MAP_CONN1,TYPE_CONN1) REFERENCES CONNECTABLE(ID,MAP,TYPE) ON DELETE CASCADE
FOREIGN KEY(ID_CONN2,MAP_CONN2,TYPE_CONN2) REFERENCES CONNECTABLE(ID,MAP,TYPE) ON DELETE CASCADE
CHECK(MAP_CONN1 = MAP_CONN2)

USER_SIM
ID INT
IS_SPECIAL BOOLEAN
SIMULATION INT REFERENCES SIMULATION(ID) ON DELETE CASCADE
PATH GEOMETRY(LINESTRING) -> recorrido del usuario por el mapa durante la simulación
PRIMARY KEY(ID,SIMULATION)

SIMULATION
ID SERIAL PRIMARY KEY
MAP_NAME TEXT
MAP_ID INT REFERENCES MAP(ID) ON DELETE CASCADE
TIME_AVAILABLE_USER INT -> tiempo en segundos que tiene cada usuario para hacer su recorrido
DELAY_OBSERVING_ITEM INT -> segundos que les cuesta a los usuarios observar un ítem visitable
USER_SPEED DECIMAL -> velocidad simulada de los usuarios en km/h
KM_TO_PIXEL DECIMAL -> píxeles que representan un kilómetro en la simulación
TIME_ON_STAIRS INT -> tiempo en segundos que le cuesta a un usuario usar una escalera
BEGIN_TIME TIMESTAMP -> hora y fecha en la que se ha iniciado la simulación

VISIT
ID INT
SIMULATION INT REFERENCES SIMULATION(ID) ON DELETE CASCADE
ROOM_LABEL INT
MAP INT
USER_ID INT
DURATION DECIMAL -> duración en segundos simulada de la estancia del usuario en la habitación
PATH GEOMETRY(LINESTRING)
PRIMARY KEY(ID,USER_ID,SIMULATION)
FOREIGN KEY(USER_ID,SIMULATION) REFERENCES USER_SIM(ID,SIMULATION) ON DELETE CASCADE
FOREIGN KEY(ROOM_LABEL,MAP) REFERENCES ROOM(LABEL,MAP) ON DELETE CASCADE)

ITEM_OBSERVATION
ID INT
VISIT INT
USER_ID INT
SIMULATION INT
ITEM_ID INT
MAP INT
PRIMARY KEY(ID,VISIT,SIMULATION)
FOREIGN KEY(ITEM_ID,MAP) REFERENCES ITEM(ID,MAP) ON DELETE CASCADE
FOREIGN KEY(VISIT,USER_ID,SIMULATION) REFERENCES VISIT(ID,USER_ID,SIMULATION) ON DELETE CASCADE)

Para empezar, se ha añadido en el editor de mapas la opción de almacenar los datos correspondientes al mapa en la base de datos. Para ello, hay que pulsar “Save map in spatial database” en el menú File del editor.

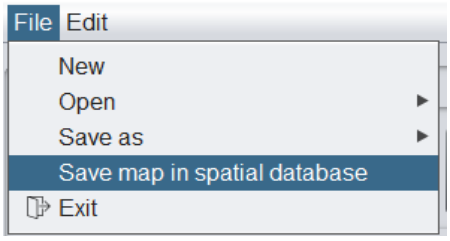


Ilustración 14 - Guardar mapa en DB

Al guardar un mapa en la base de datos PostgreSQL, algunos de los datos que se almacenen son datos geométricos compatibles con la extensión PostGIS:

- Columna geom en Room (tipo Polygon):
  Al introducir una fila en Room, se crea un dato Linestring (serie de puntos conectados por líneas) con el formato: LINESTRING(x1 y1,x2 y2,x3 y3,…). Esa cadena se tiene que transformar en Linestring con la función ST_GeomFromText (esto también se usa con todos los datos geométricos de las otras tablas). Sin embargo, lo que se quiere es un Polygon (para poder realizar en consultas más funciones geométricas de PostGIS). Para eso, se utiliza la función ST_MakePolygon para transformar el Linestring en Polygon. Por tanto, lo que se introduce en la base de datos es ST_MakePolygon(ST_GeomFromText(LINESTRING(x1 y1,x2 y2,x3 y3,…))).

```
String walls = "LINESTRING(";
for(Corner c : r.getCorners()) {
    walls += c.getVertex_xy().getX()+" "+c.getVertex_xy().getY()+",";
}
walls += r.getCorners().get(0).getVertex_xy().getX()+" "+r.getCorners().get(0).getVertex_xy().getY()+")";
pst = conn.prepareStatement("INSERT INTO ROOM(LABEL, MAP, GEOM) VALUES (?,?,ST_MakePolygon(ST_GeomFromText(?)));");
pst.setInt(1, r.getLabel());
pst.setInt(2, mapID);
pst.setString(3, walls);
pst.executeUpdate();
```

Ilustración 15 - Inserción de una fila en tabla Room

- Columna geom en Subroom_separator:
  Se introduce un simple Linestring de dos extremos para representar la línea del separador en el mapa.

- Columnas location en Connectable e Item:

  Se introduce un punto (Point) para representar la localización de las puertas, escaleras e ítems en el mapa. Para crear un dato Point de PostGIS se utiliza ST_GeomFromText(POINT(x y)).

3.1. Visualizador de mapas en base de datos

Otra de las funciones añadidas es un visualizador de mapas en la base de datos junto con una fácil visualización de algunos datos importantes (todo realizado a partir de consultas a la base de datos). Para acceder al visualizador, hay que pulsar “View maps in database” en el menú File del simulador.

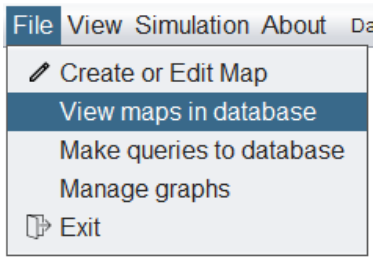


Ilustración 16 - Abrir visualizador de DB

La pantalla del visualizador está dividida en 2 partes, el panel de control (mitad izquierda de la pantalla) y el panel de visualización (mitad derecha).

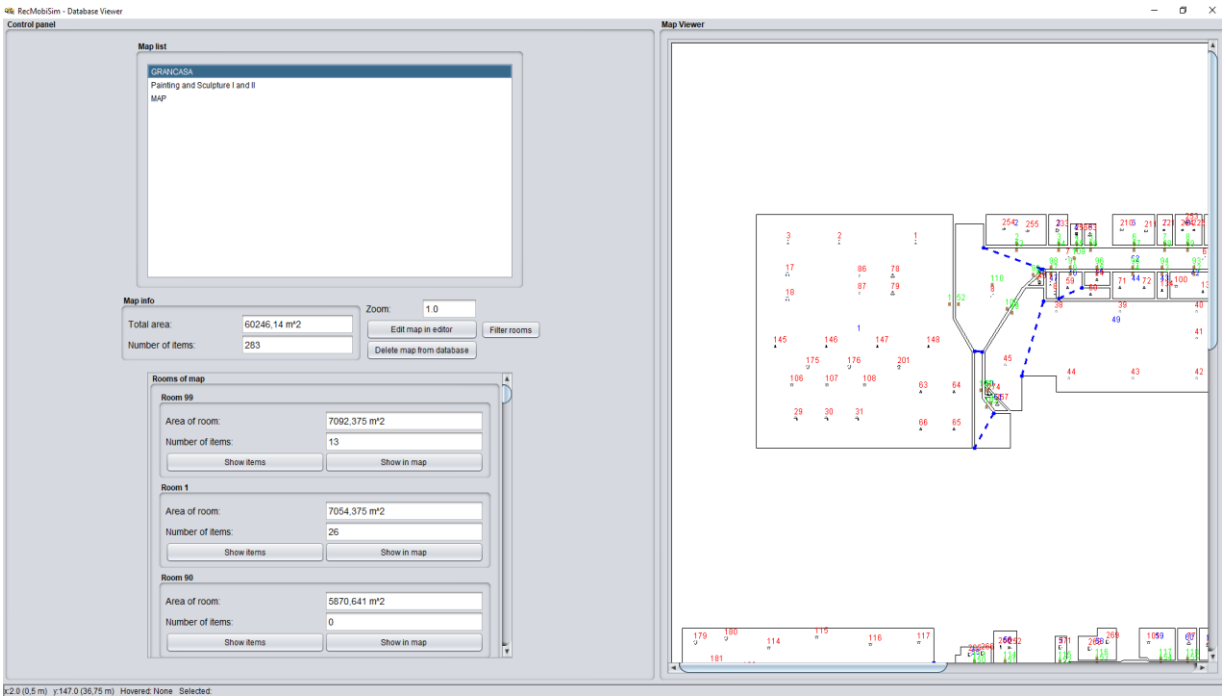


Ilustración 17 - Pantalla del visualizador

En el panel de control hay un seleccionador de mapas en la parte superior (se listan todos los mapas que hay en la base de datos por su nombre). Al hacer click en uno de los mapas, se cargará ese mapa en el visualizador. Eso quiere decir que aparecerá el

mapa en el panel de visualización (como en el editor de mapas), y el panel de control se llenará con datos de ese mapa. En la zona de “Map info” se visualizará el área total del mapa seleccionado en metros cuadrados y el número total de ítems o puntos de interés en el mapa.

A la derecha de esa zona, hay un modificador de zoom para el visualizador, y tres botones, “Edit map in editor”, “Delete map from database” y “Filter rooms”. Al pulsar el botón “Edit map in editor”, el mapa de la base de datos se abre en el editor de mapas, donde se podrá modificar y guardar en archivos o actualizarlo en la base de datos. El botón “Delete map from database”, como su nombre indica, borrará todos los datos del mapa en PostgreSQL.

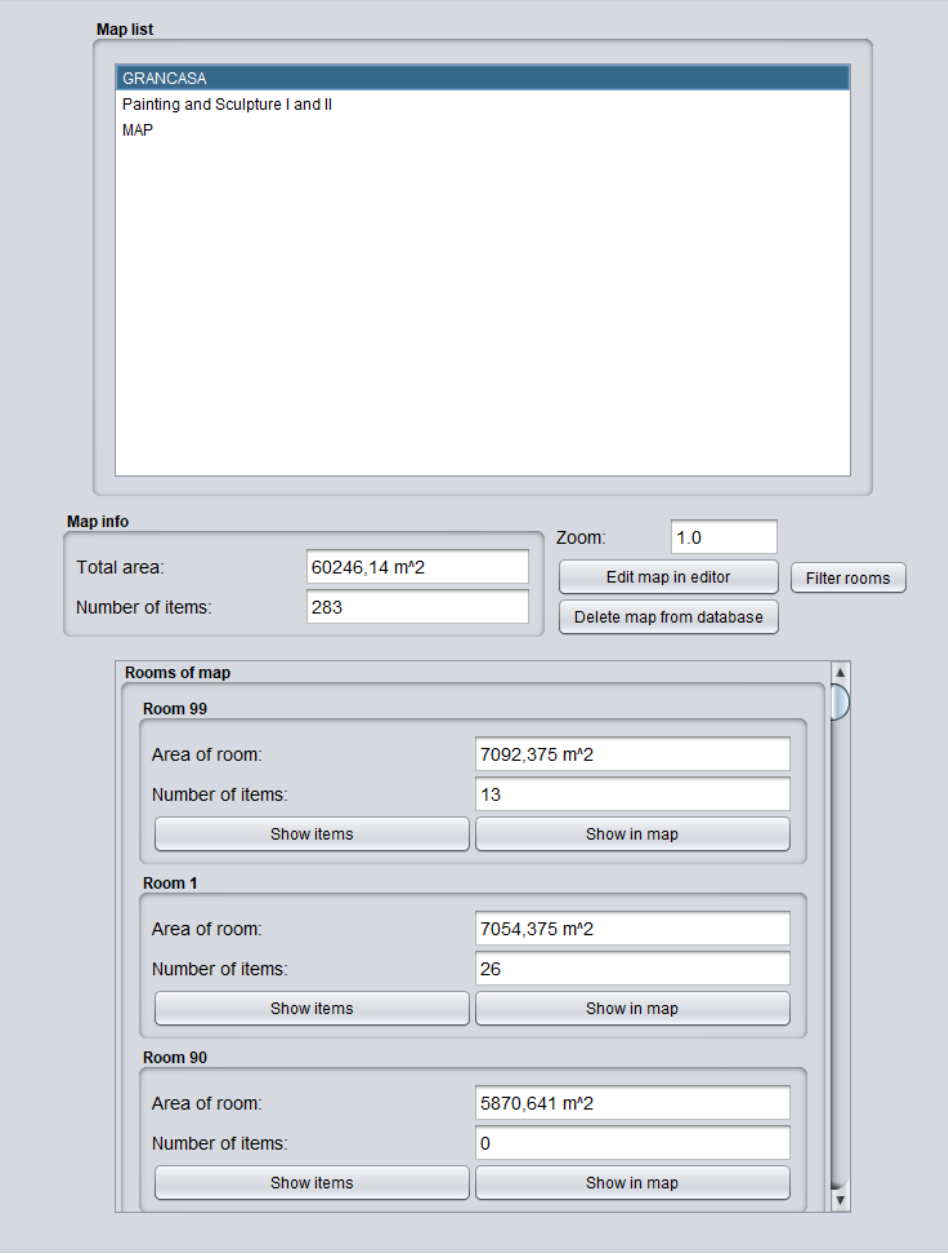


Ilustración 18 - Panel de control de visualizador

En la parte inferior del panel de control está el panel “Rooms of map”, el cual muestra una lista de las habitaciones que forman el mapa. De cada habitación se muestra su número identificador, el área y número de ítems de esa habitación en concreto y, si se pulsa “Show items” se abre una ventana donde se puede ver la lista de cada ítem con todos sus atributos.

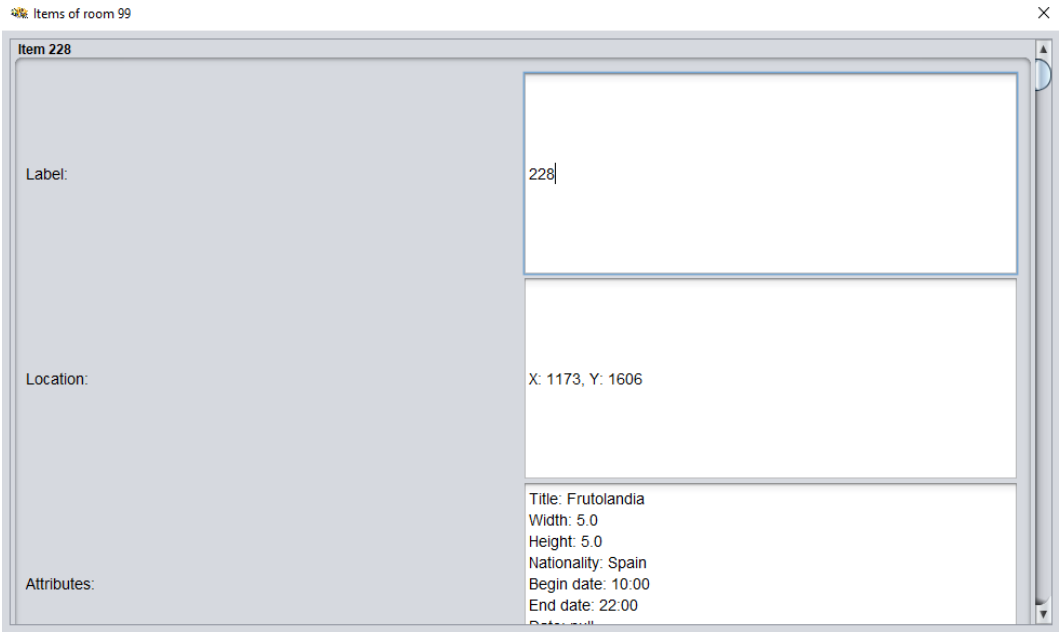


Ilustración 19 - Ítems de habitación

Además, si se hace click en “Filter rooms”, se abre una lista de habitaciones con una casilla de selección cada una. Si se desmarca una casilla, esa habitación desaparecerá de la lista (sólo aparecen las habitaciones marcadas, por defecto aparecen todas marcadas).

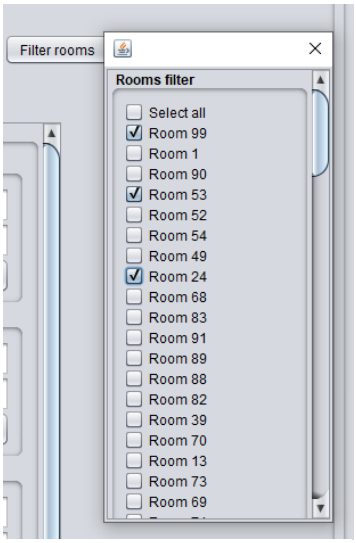


Ilustración 20 - Menú de filtrado de habitaciones (habitaciones 99, 53 y 24 marcadas)

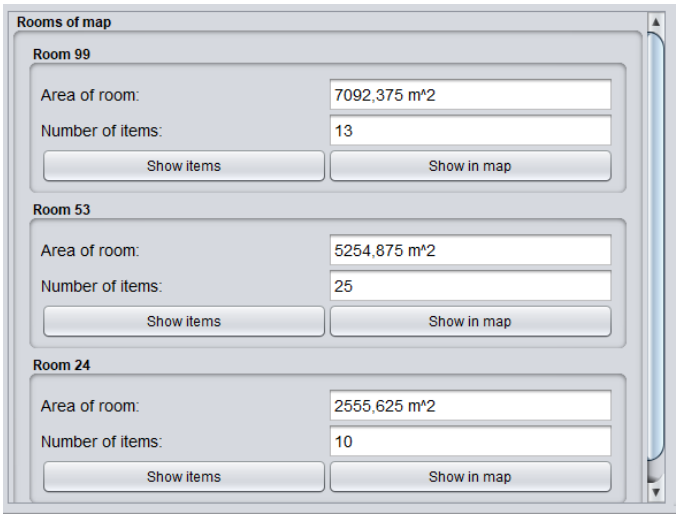


Ilustración 21 - Lista de habitaciones con todas las habitaciones filtradas excepto 99, 53 y 24

Como se ha dicho anteriormente, en el lado derecho de la pantalla del visualizador está la zona de visualización en sí, donde se puede ver el mapa. En esta visualización se ha añadido una función extra que no tiene el visualizador de mapas del editor.

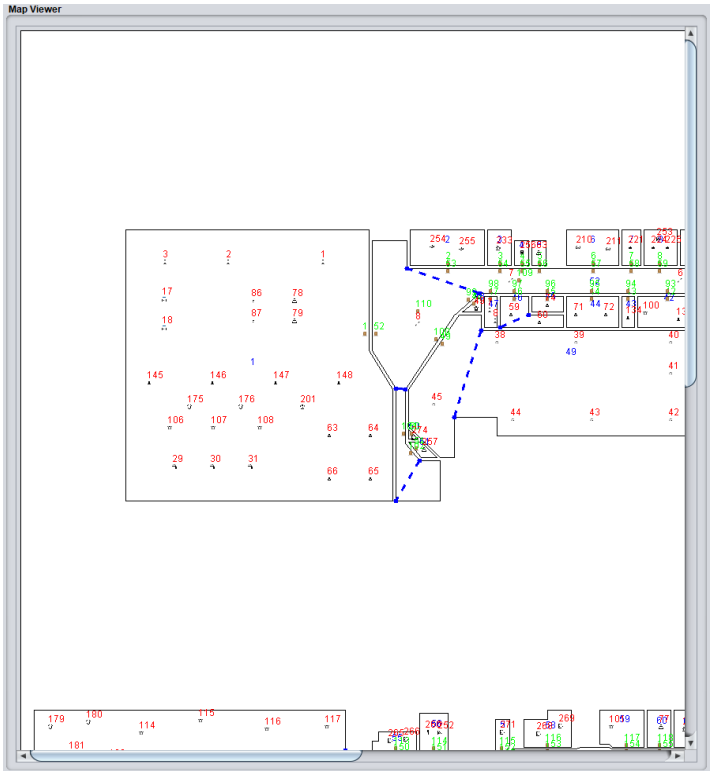


Ilustración 22 - Ejemplo de mapa visualizado

Esta función es que, al hacer doble click en uno de los números azules en el mapa que indican la etiqueta numérica de cada habitación, en la lista de habitaciones del panel de control se muestra esa habitación marcada en rojo para poder echar un vistazo a su información rápidamente.

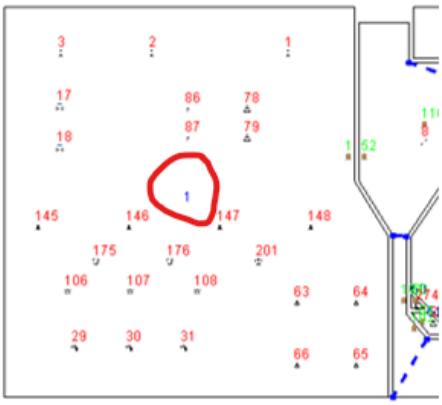


Ilustración 23 - Número de habitación en azul

Por ejemplo, al pulsar el número 1 azul de la habitación 1 en el mapa, la lista de habitaciones se queda como en la siguiente imagen:

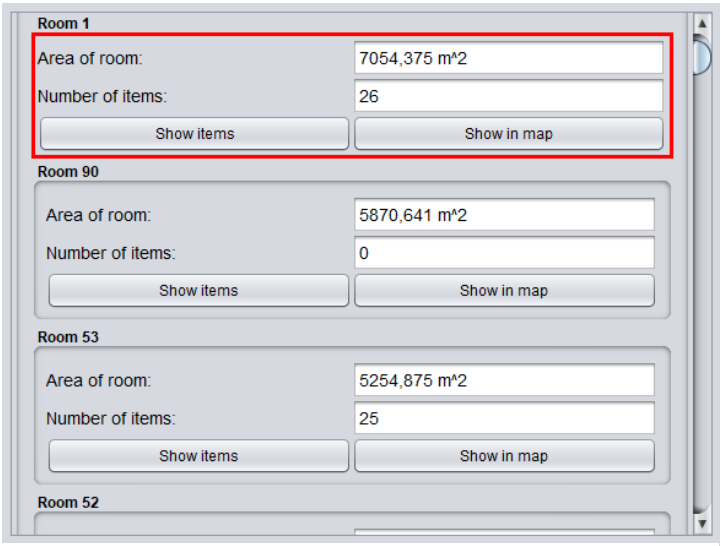


Ilustración 24 - Habitación seleccionada en lista

Por otro lado, al pulsar “Show in map” en cualquiera de las habitaciones, se muestra en el mapa enmarcada en rojo para localizarla fácilmente. Por ejemplo, al pulsar “Show in map” de la habitación 1 en la lista, el mapa de la habitación se muestra así:

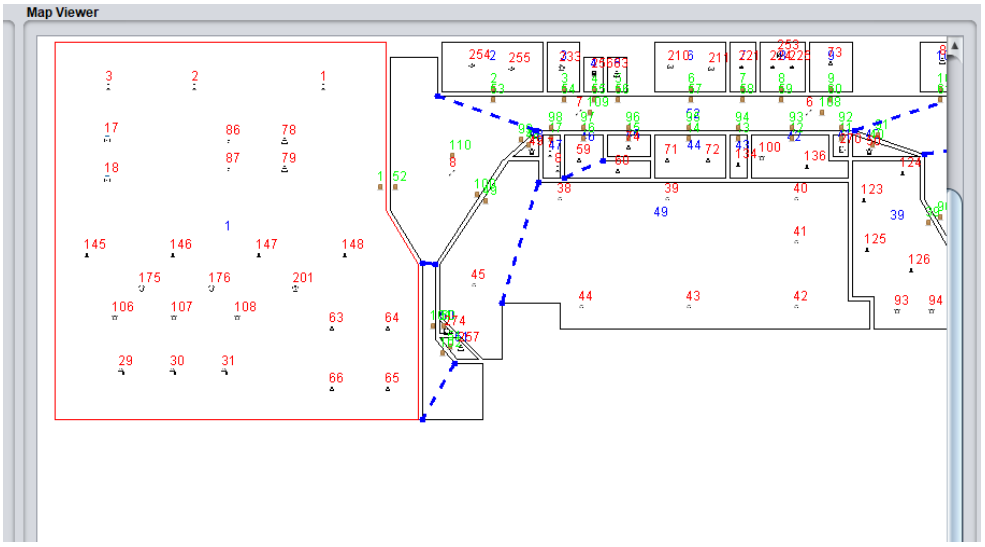


Ilustración 25 - Habitación seleccionada en mapa

3.2. Pantalla de consultas a base de datos

Además de lo anterior, también se ha creado otra pantalla nueva en la aplicación que también interacciona con la base de datos PostgreSQL + PostGIS, la cual es una pantalla que permite hacer consultas SQL a la base de datos a la que se esté conectada.

Para acceder a ella, hay que pulsar “Make queries to database” en el menú File de la pantalla principal del simulador (si hay una conexión activa a una base de datos).

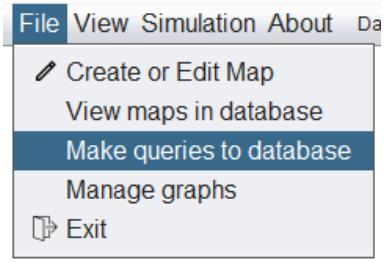


Ilustración 26 - Acceder a consultas

En la página de consultas hay un panel de control con varios botones en la parte izquierda de la pantalla, mientras que en la derecha hay una zona para escribir consultas y ver los resultados devueltos.

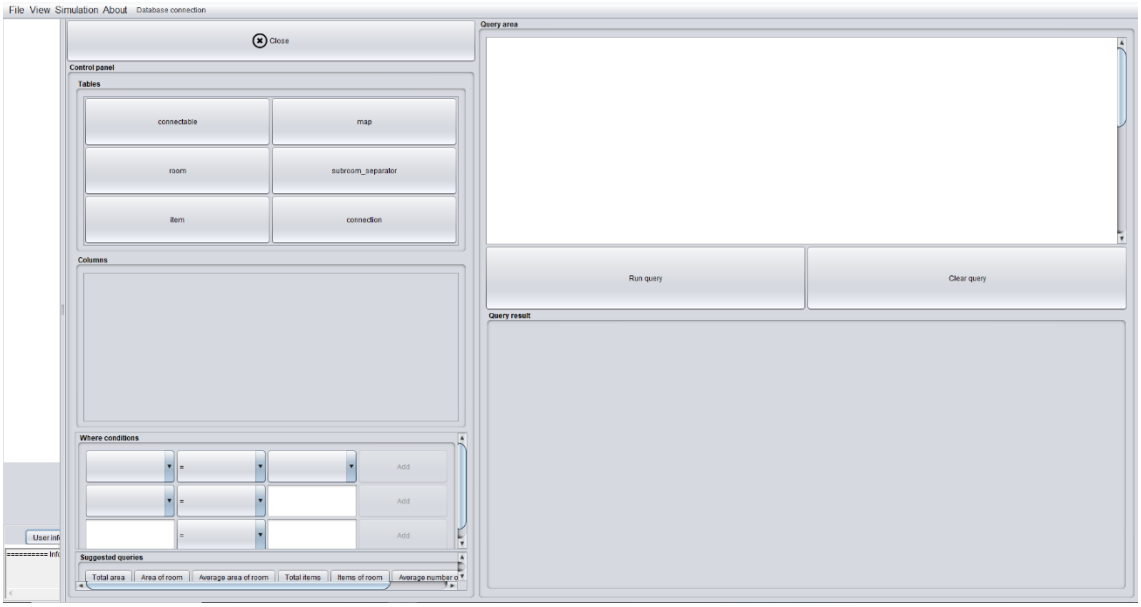


Ilustración 27 - Pantalla de consultas a PostgreSQL con PostGIS

En el panel de control hay una sección “Tables” con botones con los nombres de las 6 tablas de la base de datos relacionadas con los mapas en sí (map,room,subroom_separator,item,conectable,connection). El resto de tablas están dedicadas a información de traza de simulaciones y tienen su propia pantalla de consultas (se verá en otro apartado). Al hacer click en uno de ellos cuando no hay nada escrito en la zona de consultas, se genera una consulta básica que devuelve todas las filas de esa tabla. Tras haber generado esa consulta, si se pulsan los demás botones de tablas, se añadirán esas tablas a la sección “from” de la consulta.

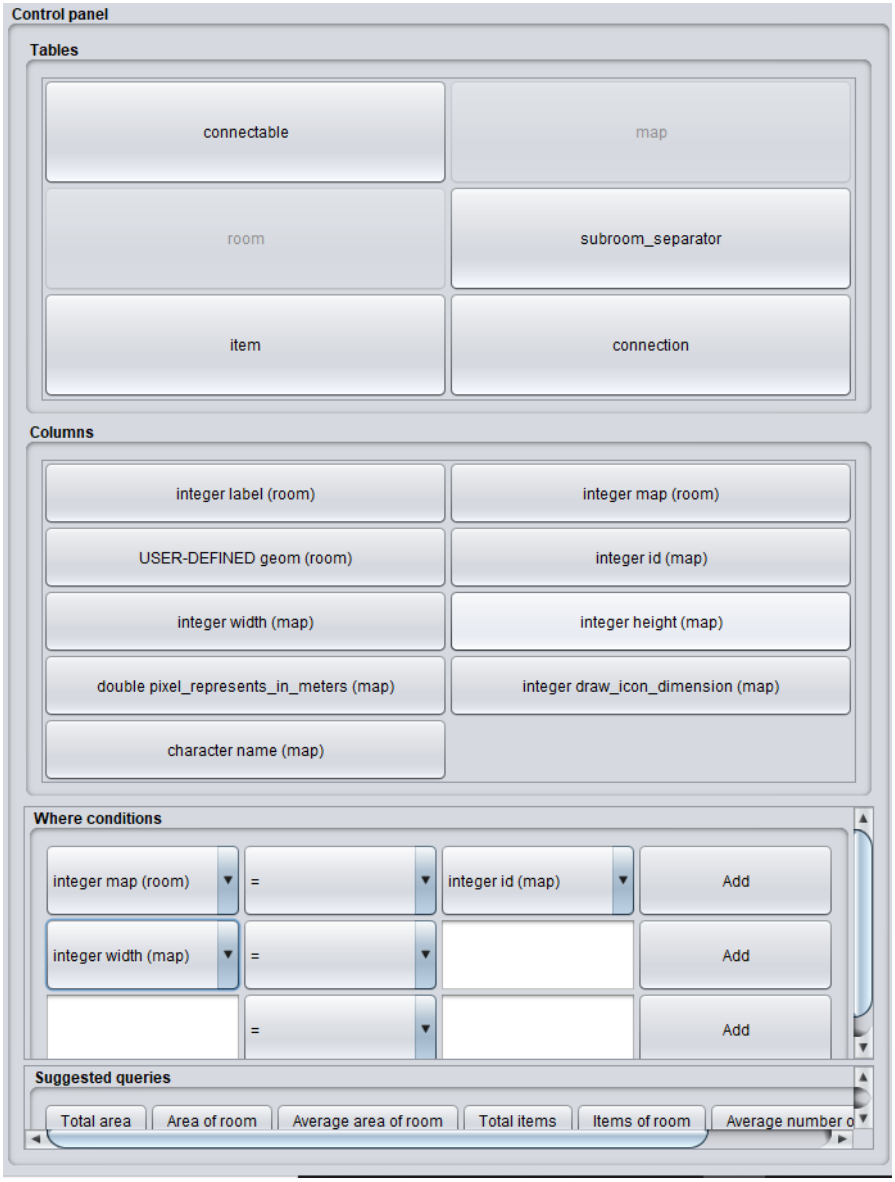


Ilustración 28 - Panel de control de consultas PostgreSQL

Al pulsar cada botón de tabla, se puede ver que en la sección inferior “Columns” aparecen botones de cada una de las columnas de esa tabla. Si se hace click en ese botón, se selecciona esa columna en la consulta (se añade esa columna a la sección “select” de la consulta).

Debajo de “Columns” se encuentra la sección “Where conditions”, que permite añadir condiciones a la sección “where” de la consulta. Hay 3 filas: se pueden comparar 2 columnas (de entre las mismas que hay en la zona “Columns”) en la primera fila, 1 columna con lo que se escriba en el campo de texto en la segunda, o lo que se escriba en 2 campos de texto en la tercera.

En la parte inferior del panel de control hay una lista de “Suggested queries”, pulsar en una de ellas permite generar una consulta que devuelve una información específica que se considera relevante (se eligen mapa y habitación de una lista).

Esas consultas de “Suggested queries” son:

- Total area (área total del mapa):

```
select sum(ST_Area(geom))*pow(pixel_represents_in_meters,2)
from room,map
where map.id = room.map and map.name = __ ;
```

- Area of room (área de una habitación):

```
select ST_Area(geom)*pow(pixel_represents_in_meters,2)
from room,map
where map.id = room.map and map.name = __ and room.label = __ ;
```

- Average area of room (área media de las habitaciones):

```
select avg(ST_Area(geom))*pow(pixel_represents_in_meters,2)
from room,map
where map.id = room.map and map.name = __ ;
```

- Total items (lista de todos los ítems):

```
select
item.id,item.map,item.room_label,item.location,item.url_image,item.titl
e,item.width,item.height,item.nationality,item.begin_date,item.end_dat
e,item.date,item.item_label
from item,map
where map.id = item.map and map.name = __ ;
```

- Items of room (ítems de una sola habitación):

```
select
item.id,item.map,item.room_label,item.location,item.url_image,item.title,item.
width,item.height,item.nationality,item.begin_date,item.end_date,item.date,it
em.item_label
from item,map,room
where map.id = item.map and map.id = room.map and map.name = __ and
room.label = __;
```

- Average number of items per room (cantidad media de ítems por habitación):

```
select avg(num) from (
    select count(*) as num
    from item,map
    where item.map = map.id and map.name = __
    group by item.room_label
);
```

Los elementos (map.name y room.label) que están sin especificar en las consultas se seleccionan de un menú desplegable que aparece al usar el botón de la consulta correspondiente:

Ilustración 30 - Selección de mapa para la "Total area"

```
select sum(ST_Area(geom))*pow(pixel_represents_in_meters,2)
from room,map
where map.id = room.map and map.name = 'GRANCASA'
```

Ilustración 29 - Consulta generada por "Total area"

El botón Close de la parte superior de la pantalla cierra la pantalla de consultas.

En la parte izquierda de la pantalla se encuentra la zona de consultas, donde se pueden escribir consultas en la parte superior (o generarlas con el panel de control), y obtener resultados pulsando “Run query”, los cuales aparecen en la parte inferior. El botón “Clear query” borra el área de texto de la consulta.

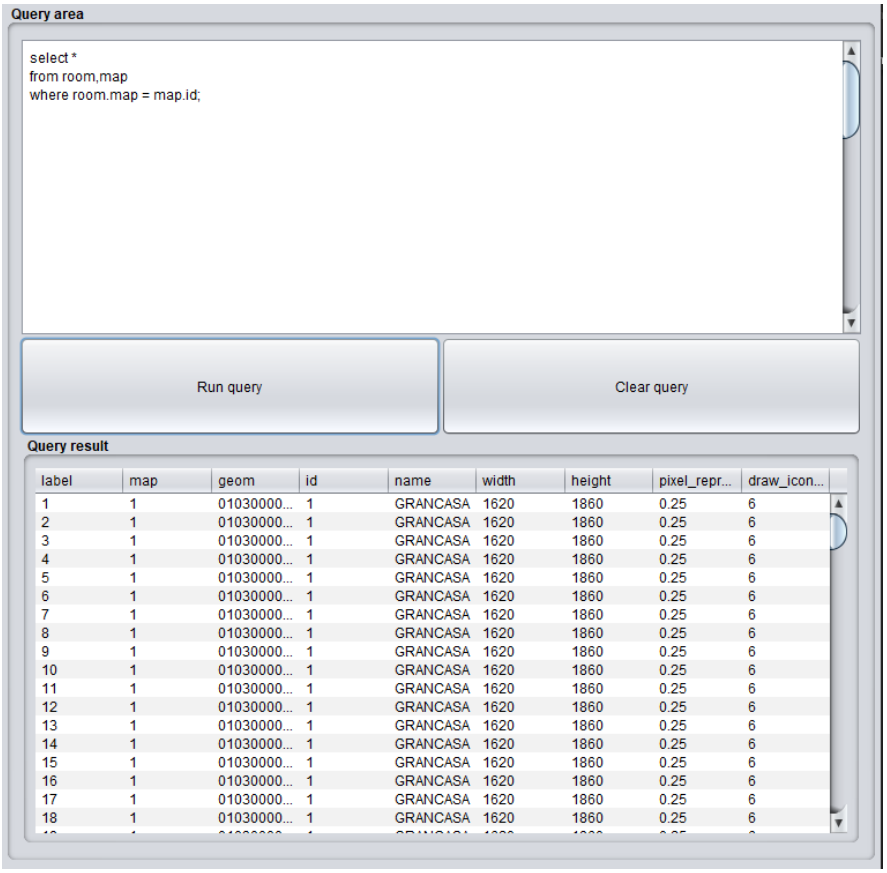


Ilustración 31 - Zona de consultas PostgreSQL

4. Grafos

Otra función que se ha añadido es la posibilidad de conectar con una base de datos Neo4j y crear grafos con los datos relacionados con los mapas. Para ello, se ha creado una pantalla a la que se accede desde el menú File en la pantalla de simulación (la opción “Manage graphs”):

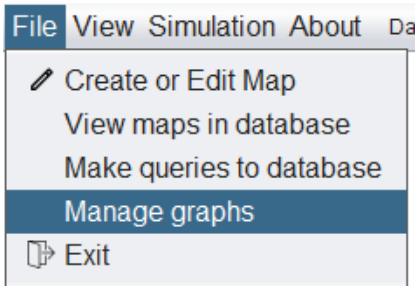


Ilustración 32 - Abrir pantalla gestión de grafos

Las clases de Java que se encargan del funcionamiento de la pantalla de grafos se encuentran en el paquete es.unizar.graph del proyecto. La clase que genera la pantalla de gestión de grafos es GraphManager.

Esta pantalla de gestión de grafos está dividida en 2 partes. La parte izquierda de la pantalla (“Graph manager”) se encarga de gestionar la conexión con Neo4j, creación de grafos y visualización de los mismos, mientras que la parte derecha está dedicada a la ejecución de consultas Cypher (el lenguaje de consultas de Neo4j).

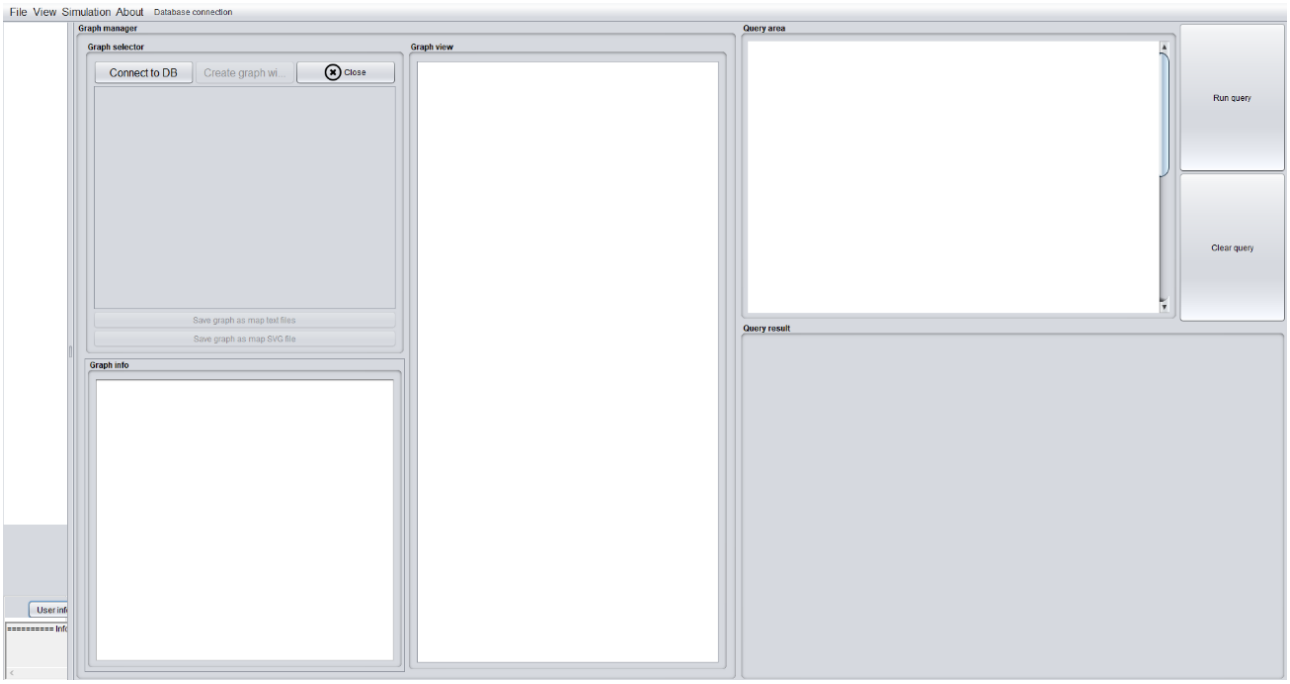


Ilustración 33 - Pantalla de consultas Cypher

4.1 Creación y visualización de grafo

Al pulsar “Connect to DB” se abrirá un diálogo en el que introducir los datos de conexión a la base de datos Neo4j a través de su JDBC. Si se establece conexión, aparecerá la lista de grafos ya creados en la base de datos en la parte inferior.

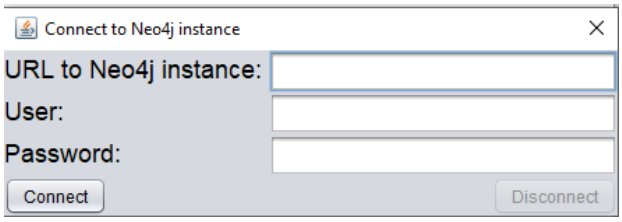


Ilustración 34 - Diálogo de conexión con Neo4j

Para crear un grafo nuevo, hay que hacer click en “Create graph”, el cual abrirá un selector de archivos, en el que se puede seleccionar un directorio con los 3 ficheros de texto o un fichero SVG de un mapa creado/editado con RecMobiSim. De esta manera, se creará un grafo con información relevante del mapa en la base de datos (el proceso de creación puede tardar varios minutos). Una vez creado, el nombre del mapa aparecerá en la lista de grafos.

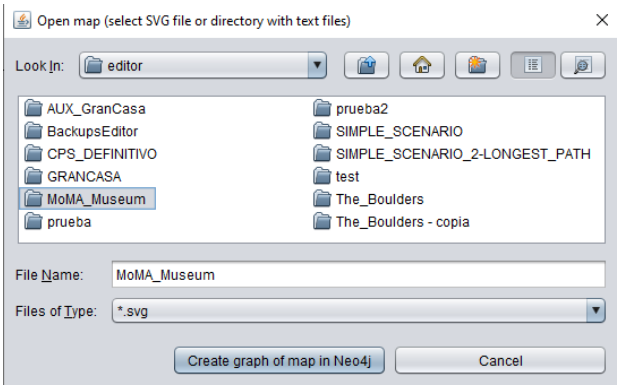


Ilustración 35 - Crear un grafo a partir de un escenario

Al hacer doble click en un grafo de la lista, éste quedará seleccionado y en el panel que está a la derecha de la lista “Graph view” aparecerá una visualización del grafo. Esa visualización es generada con la biblioteca GraphStream. Al hacer click en un nodo del grafo, se podrán ver sus atributos en el panel “Graph info”. Cada tipo de nodo es de un color distinto.

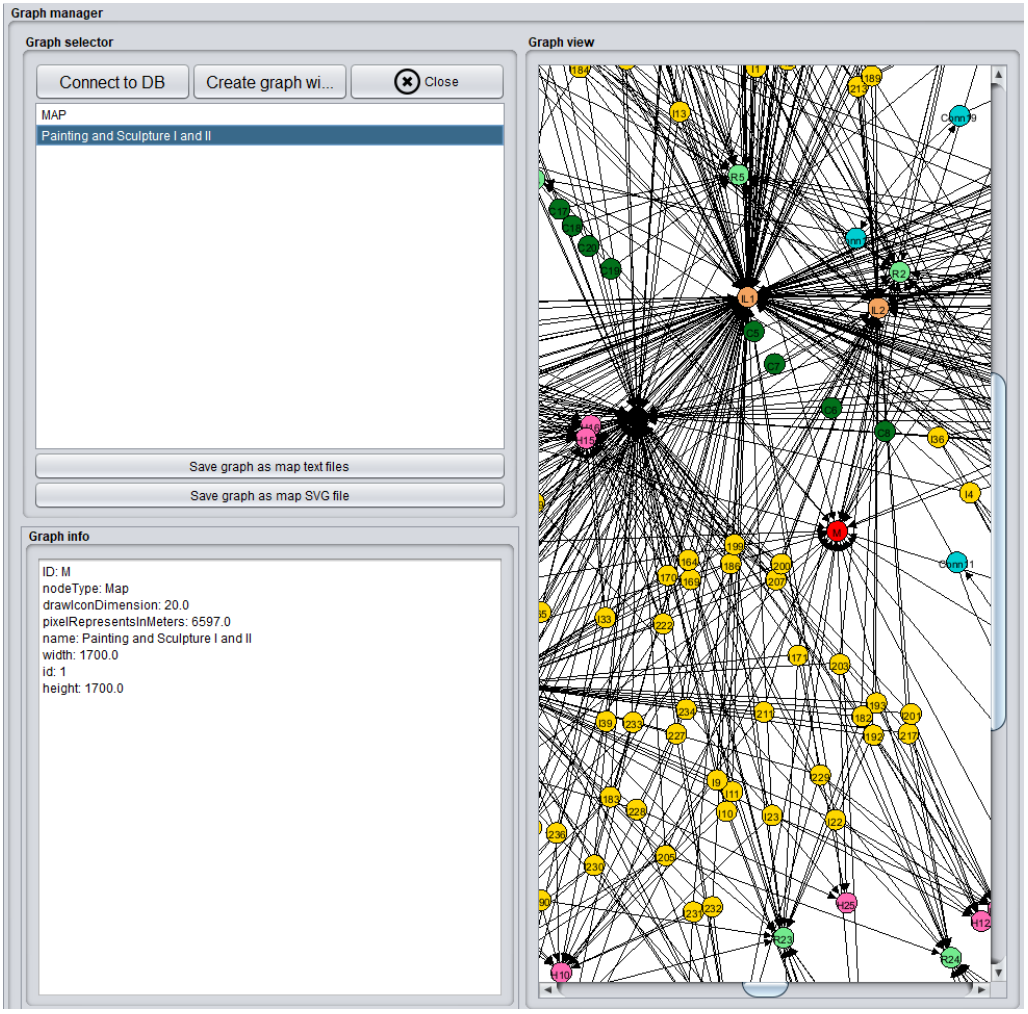


Ilustración 36 - Grafo de escenario "Painting and Sculpture I and II" visualizado

Para que el programa detecte los clicks a los nodos del grafo, se ha tenido que crear una clase, GraphController, el cual se encarga de detectar si hay clicks nuevos (se crea un hilo que está constantemente enviando a GraphController si se ha hecho algún click) y de buscar el nodo al que se ha hecho click y colocar sus atributos en el panel correspondiente en la pantalla de grafos.

4.2 Diseño del grafo

Cada grafo de cada mapa tiene un nodo “Map”, del cual cuelgan el resto de nodos de ese mapa. Un nodo Map contiene además los atributos: id, name, width, height, pixelRepresentsInMeters y drawIconDimension.

Otros tipos de nodos:

- Room: Los nodos “Room” representan las habitaciones del mapa. Están conectadas a un nodo Map con la relación IS_PART_OF (Room -> Map).

- Corner: Los nodos que representan las esquinas de las habitaciones. Sus atributos son: coordX, coordY (coordenadas en el mapa) y vertexLabel (número identificador). Cada uno está conectado a un nodo “Room” con la relación CORNER_OF. También pueden estar conectados a un nodo “RoomSeparator” con una relación también llamada CORNER_OF.

- RoomSeparator: Los nodos que representan los separadores de habitaciones en subhabitaciones. Tienen un atributo vertexLabel. Cada uno está conectado a un nodo “Room” con la relación SEPARATOR_OF, y dos nodos “Corner” están conectados con cada “RoomSeparator” (los extremos del separador de habitaciones).

- Connectable: Representan puertas y escaleras, que conectan las habitaciones. Sus atributos son vertexLabel, coordX, coordY y type (es “DOOR” o “STAIRS” según el tipo que sea). Cada uno puede estar conectado a un nodo “Room” si se encuentran en una habitación o a un “Map” si no se encuentran en ninguna (las dos conexiones se realizan con la relación CONNECTABLE_LOCATED_IN). Cada uno está también conectado a uno o varios “Connectable” con la relación CONNECTED_TO para representar las conexiones físicas en el mundo real, ya que los usuarios se mueven de un conectable a otro (por ejemplo, dos puertas conectadas en el mapa son los dos lados de una puerta en el mundo real).

- Item: Son los ítems visitables del lugar que representa el mapa. Sus atributos son: vertexLabel, title, width, height, iconURL, coordX y coordY. Cada uno está conectado a una habitación con la relación ITEM_LOCATED_IN.

- Los demás tipos de nodos (ItemLabel, Hour, Nationality, Date) están conectados a nodos Item y se usan para representar el resto de características de los ítems. “ItemLabel” contiene solo la etiqueta del ítem (la categoría). “Hour” almacena una hora, esto se usa para indicar la hora a la que un establecimiento abre o cierra (cada nodo “Item” puede estar conectado a dos nodos “Hour”, a uno con la relación OPEN_AT y a otro con CLOSES_AT). “Nationality” indica nacionalidad y “Date” indica período histórico (de un cuadro o escultura por ejemplo) si los

ítems los necesitan. Varios nodos de ítems pueden estar conectados a uno solo de estos nodos (si tienen esa misma característica). Esto ayuda a la hora de hacer consultas Cypher relacionadas con ítems filtrando según una de estas características (los demás atributos de “Item” son únicos o menos dados a repetirse entre distintos ítems).

Solamente en la visualización del grafo en RecMobiSim, todos los tipos de nodo tienen un atributo nodeType que indica el tipo de nodo.

4.3 Consultas

Como se ha dicho anteriormente, la parte derecha de la pantalla está dedicada a la ejecución de consultas con el lenguaje Cypher, el cual está basado en patrones.

Para ejecutar una consulta hay que escribirla en la zona superior y hacer click en “Run query”. Los resultados aparecen en una tabla en la zona inferior.

Una consulta básica de Cypher está formada por la palabra clave MATCH seguida de un patrón de nodos y relaciones. Un nodo se representa con (nombreVariable+”:”+tipoNodo). Para representar una relación entre nodos se conecta un nodo con un guion, la relación de esta forma [nombreVariable+”:”+tipoRelación], otro guión, un símbolo “>” pare representar la dirección de la relación y luego otro nodo. De esta forma la relación se quedaría así: (nombreVariable+”:”+tipoNodo)-[nombreVariable+”:”+tipoRelación]->(nombreVariable+”:”+tipoNodo). Los nombres de variables son opcionales. Eso sirve para encontrar relaciones en los grafos que sigan ese patrón junto con sus nodos.

Sin embargo, para que la consulta devuelva resultado hay que utilizar a continuación RETURN seguido de las variables que contengan los datos que se quieren obtener.

Por ejemplo, se ejecuta una consulta para encontrar todas las esquinas (y sus habitaciones) que forman todas las habitaciones de un mapa de la base de datos que se llama “Painting and Sculpture I and II”:

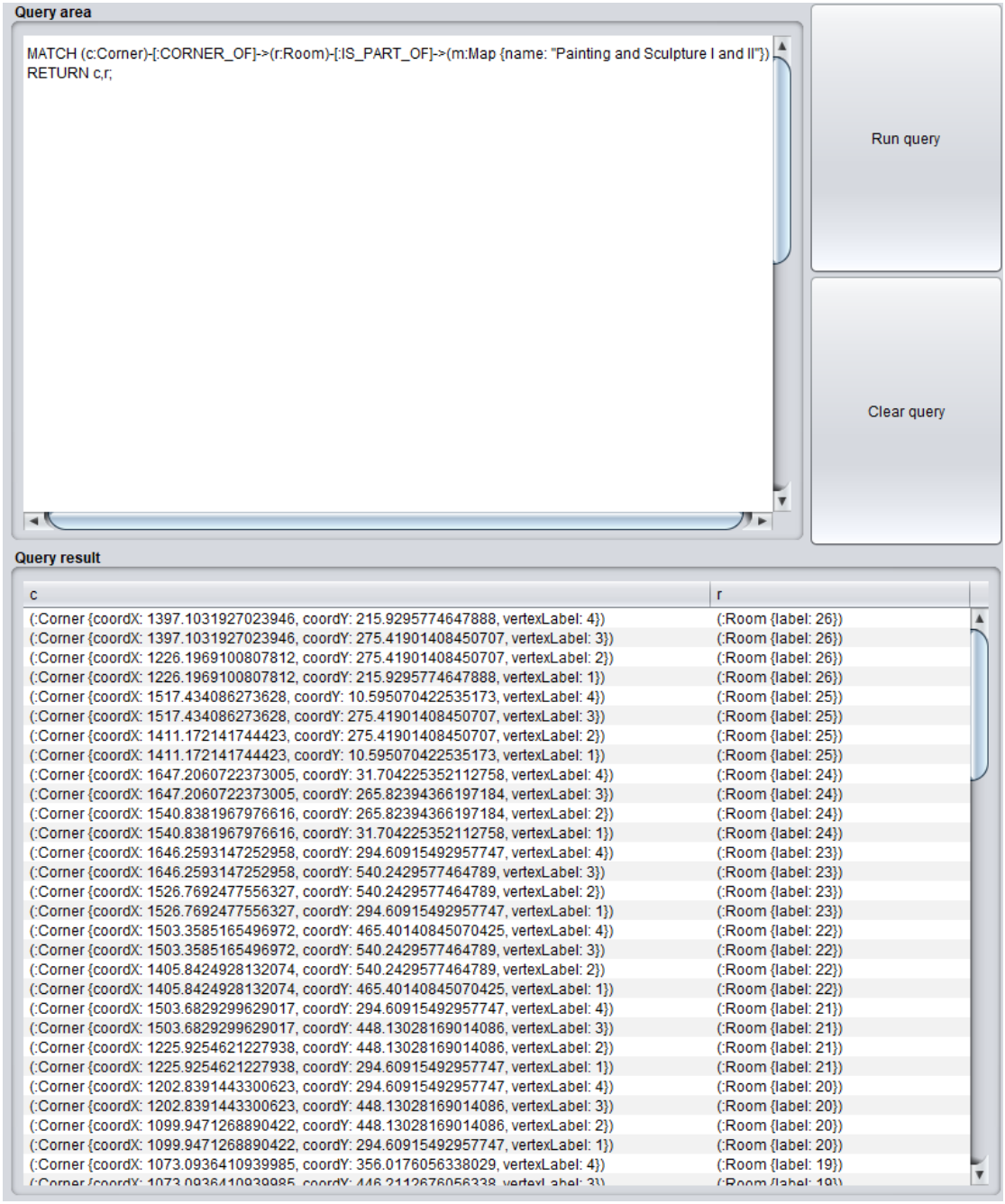


Ilustración 37 - Consulta de Cypher y su resultado

5. Mejoras en simulación

Se han hecho varias mejoras relacionadas con las simulaciones en sí realizadas en RecMobiSim. Para empezar, se ha añadido un selector de mapas en los que hacer simulaciones (previamente sólo se podían realizar simulaciones con el mapa de un directorio concreto, por lo que para cambiar de mapa se tenían que modificar los ficheros de mapa de ese directorio). Ahora, para elegir el mapa deseado desde la aplicación hay que usar el botón “Choose map” en el menú de configuración de simulación.

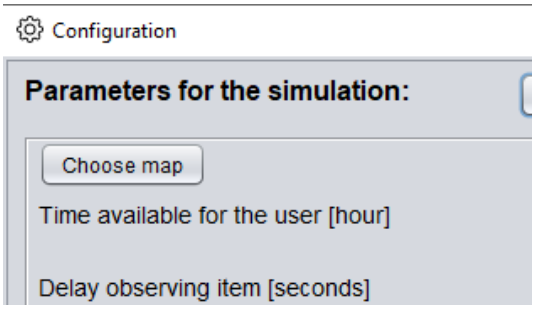


Ilustración 38 - Botón "Choose map" para elegir escenario en simulación

Se puede elegir tanto un directorio con los 3 archivos txt o un archivo SVG con los correspondientes requerimientos de compatibilidad con RecMobisim: tienen que haber sido generados por el editor de mapas de RecMobiSim o tener la misma estructura de los datos.

Una vez elegido el mapa, cualquier simulación que se realice hasta que se cambie el mapa otra vez (o se cierre la aplicación) será en ese mapa.

En el caso de utilizar un SVG como mapa, se han tenido que hacer trabajo extra, ya que originalmente durante una simulación el programa accedía a los datos de los 3 ficheros txt del mapa a través de las clases DataAccessGraphFile, DataAccessRoomFile y DataAccessItemFile. Por tanto, para hacer que RecMobiSim pueda acceder a un SVG lo que se ha decidido hacer es crear ficheros txt temporales con los datos del SVG cuando el código lo necesite.

Para ello, se ha tenido que modificar la superclase de DataAccessGraphFile, DataAccessRoomFile y DataAccessItemFile, DataAcess, para que cree el fichero temporal si el escenario seleccionado es de un SVG. Todos los ficheros temporales tienen de nombre “file”+número+”.txt” (los números son secuenciales).

```
public DataAccess(File file) {
    if(file != null && file.exists() && file.getName().endsWith(".svg")) {
        int i = 1;
        this.file = new File("resources" + File.separator + "tmp" + File.separator + "file"+ i + ".txt");
        try {
            while(!this.file.createNewFile()) {
                i++;
                this.file = new File("resources" + File.separator + "tmp" + File.separator + "file"+ i + ".txt");
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }else {
        this.file = file;
    }
      this.file = file;
    this.properties = new Properties();
    this.dataAccessMon = null;
    if (this.file != null && this.file.exists() && !file.getName().endsWith(".svg")) {
        loadProperties();
    }
}
```

Ilustración 39 - Ilustración 39 - Crear fichero temporal para propiedades de escenario

Al cargar un nuevo mapa para la simulación, se borran todos los ficheros temporales ya existentes.

Después de que se cree el fichero temporal al crearse un DataAccessGraphFile, DataAccessRoomFile o DataAccessItemFile, se usa la nueva clase clase SVGParserSimulation, la cual lee el SVG y carga sus propiedades en el fichero temporal txt correspondiente.

```
public DataAccessRoomFile(File file) {
    super(file);
    if(file != null && file.getName().endsWith(".svg")) {
        new SVGParserSimulation(this,file);
        loadProperties();
    }
}
```

Ilustración 40 - Cargar propiedades de escenario en fichero temporal

Otros elementos que se han añadido son dos botones de la parte inferior de la pantalla de simulación, “User info” y “Simulation queries”.

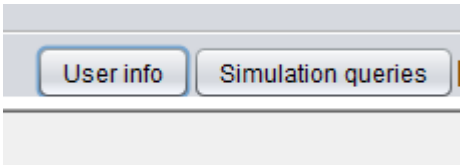
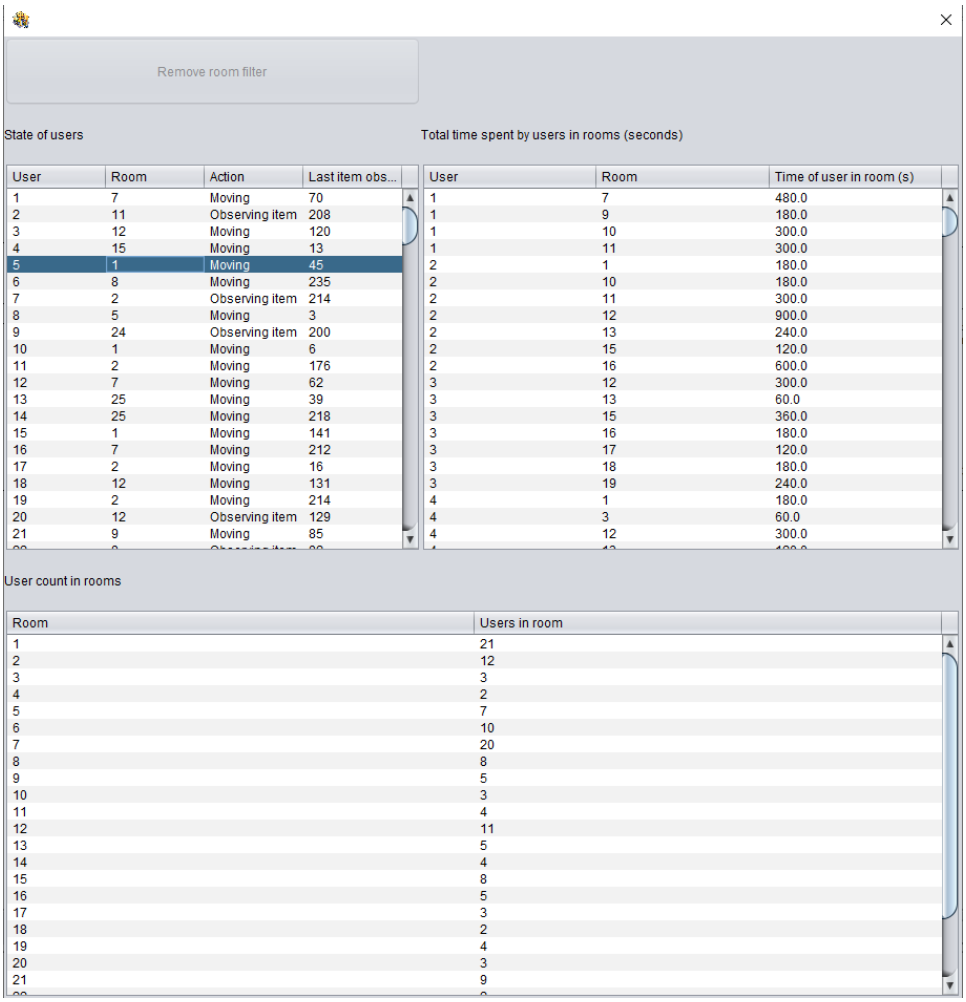


Ilustración 41 - Botones "User info" y "Simulation queries"

Al pulsar “User info” durante una simulación, se abrirá una ventana con 3 tablas.



User	Room	Action	Last item obs...
1	7	Moving	70
2	11	Observing item	208
3	12	Moving	120
4	15	Moving	13
5	1	Moving	45
6	8	Moving	235
7	2	Observing item	214
8	5	Moving	3
9	24	Observing item	200
10	1	Moving	6
11	2	Moving	176
12	7	Moving	62
13	25	Moving	39
14	25	Moving	218
15	1	Moving	141
16	7	Moving	212
17	2	Moving	16
18	12	Moving	131
19	2	Moving	214
20	12	Observing item	129
21	9	Moving	85

User	Room	Time of user in room (s)
1	7	480.0
1	9	180.0
1	10	300.0
1	11	300.0
2	1	180.0
2	10	180.0
2	11	300.0
2	12	900.0
2	13	240.0
2	15	120.0
2	16	600.0
3	12	300.0
3	13	60.0
3	15	360.0
3	16	180.0
3	17	120.0
3	18	180.0
3	19	240.0
4	1	180.0
4	3	60.0
4	12	300.0

Room	Users in room
1	21
2	12
3	3
4	2
5	7
6	10
7	20
8	8
9	5
10	3
11	4
12	11
13	5
14	4
15	8
16	5
17	3
18	2
19	4
20	3
21	9

Ilustración 42 - Tablas de información de usuarios simulados actualmente

La tabla “State of users” indica en qué habitación se encuentra cada usuario actualmente en la simulación, qué está haciendo (moviéndose,observando un ítem o ha terminado de moverse) y el vertexLabel del último ítem que ha visitado. La tabla “Total time spent by users in romos (seconds)” indica cuántos segundos simulados ha pasado en total cada usuario en cada habitación de la simulación (si una combinación de usuario y habitación no aparece significa que ese usuario no ha estado en la habitación). La última tabla, “User count in rooms”, muestra cuántos usuarios se encuentran en cada habitación en ese momento.

Si se hace doble click en una fila de cualquiera de las 3 tablas, la habitación que se indica en la fila se marcará en rojo en el mapa de la simulación (puede tardar unos segundos en cambiar).

User	Room	Action	Last item obs...
1	7	Moving	70
2	11	Observing item	208
3	12	Moving	120
4	15	Moving	13
5	1	Moving	45
6	8	Moving	235
7	2	Observing item	214
8	5	Moving	3
9	24	Observing item	200
10	1	Moving	6
11	2	Moving	176
12	7	Moving	62
13	25	Moving	39
14	25	Moving	218
15	1	Moving	141
16	7	Moving	212
17	2	Moving	16
18	12	Moving	131
19	2	Moving	214
20	12	Observing item	129
21	9	Moving	85

Ilustración 44 - Doble click en fila de tabla

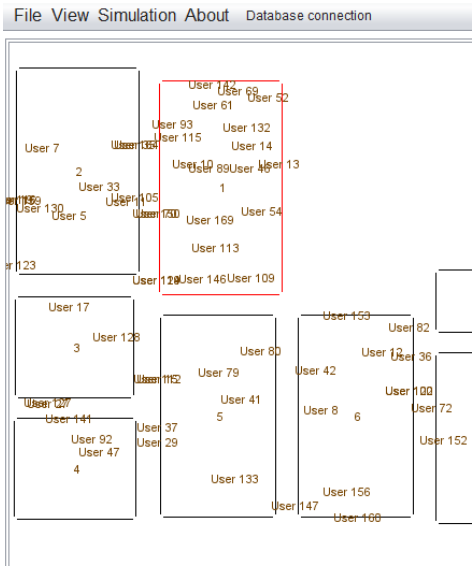


Ilustración 43 - Habitación marcada en el mapa

Por otro lado, el otro botón “Simulation queries” abre una pantalla de escritura de consultas PostgreSQL, similar a la pantalla mencionada en el apartado 3.2 de este documento. Las diferencias recaen en las tablas presentes en el menú de tablas del panel de control (por lo demás, el funcionamiento es igual). Estas otras tablas presentes en la base de datos PostgreSQL + PostGIS representan información de traza de simulaciones anteriormente realizadas con RecMobiSim. Las tablas son: simulation (cada fila representa una simulación), user_sim (cada fila representa un usuario simulado de una simulación), visit (cada fila representa un periodo de tiempo en el que un usuario ha estado en la misma sala durante su recorrido en una simulación) e item_observation (cada fila representa un ítem observado por un usuario durante una visita). También están presentes las tablas map, room e item, ya que están relacionadas con las tablas anteriores.

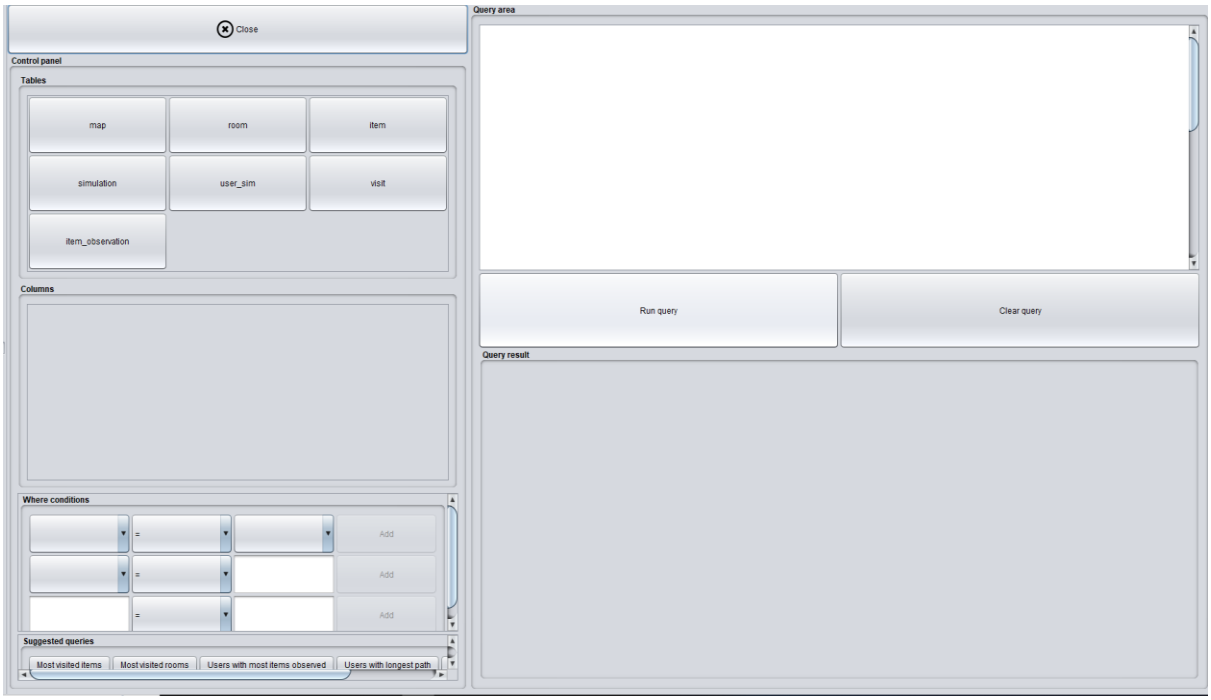


Ilustración 45 - Pantalla de consultas de trazas de simulaciones

Como las tablas son distintas, las “Suggested queries” también lo son. Esta vez, las consultas sirven para obtener: ítems más observados, habitaciones más visitadas, usuarios con más ítems observados, usuarios que han hecho un recorrido más largo, y estancias de usuarios más largas en una habitación. Todas esas consultas son por simulación, la cual se puede elegir de una lista.

Si al iniciar una simulación hay una conexión establecida con una base de datos PortgreSQL, el programa comprobará si existe un mapa (en la tabla map) con el mismo nombre que el mapa de la simulación. Si existe, RecMobiSim preguntará si se quiere que se almacene información de traza de la simulación el la base de datos. Si se contesta que sí, eso es lo que hará.

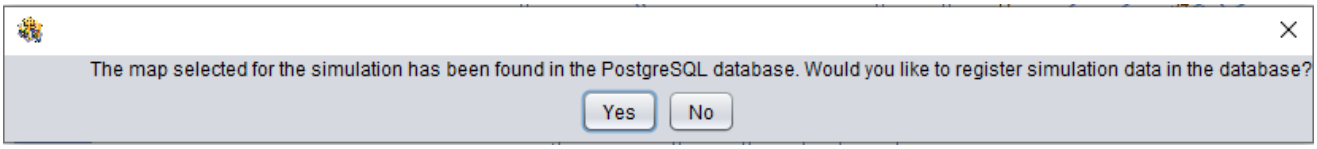
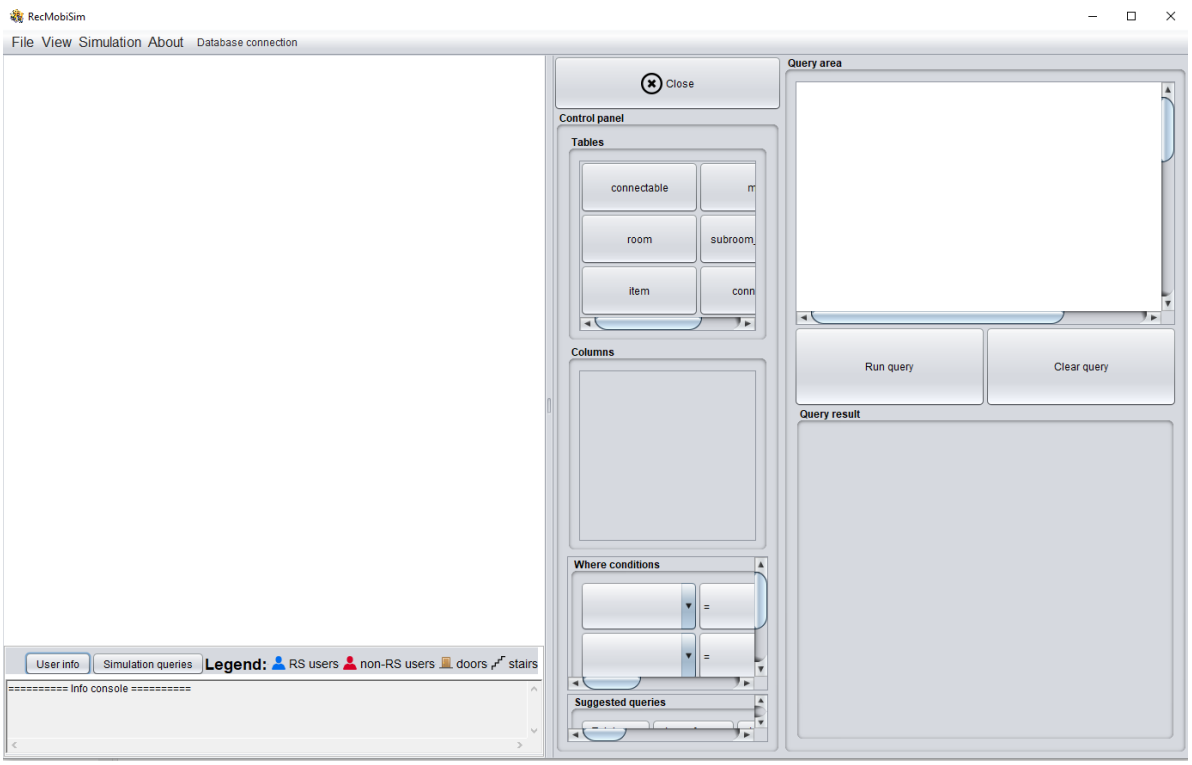


Ilustración 46 - Pantalla de confirmación de guardado de información de traza de simulación

6. Otros añadidos

Todas las nuevas pantallas creadas para este TFG se abren con en “Split view”, lo que quiere decir que al abrir una pantalla, la pantalla de simulación y la otra pantalla ocupan el mismo espacio de pantalla con una barra entre las dos que se puede mover para cambiar cuánto espacio de la pantalla ocupa cada una.



Bugs corregidos:

- En Simulation.java, hay una función updateUsers() que se ejecuta durante una simulación. En esa función se ejecuta en dos ocasiones la función checkDoorsConnectedByStairs de un objeto Path (pathStrategyUsed).

```
1040                        currentTimeOfUsers[userPosition] += getTimeOnStairs();
1041                    }*/
1042                    if (this.pathStrategyUsed.checkDoorsConnectedByStairs(v1, v2)) {
1043                        currentTimeOfUsers[userPosition] += getTimeOnStairs();
1044                    } else {
1045                        currentTimeOfUsers[userPosition] += getCurrentTime(location_v1, location_v2);
1046                    }
1047                }
1048
1049                finalTimeMovement = System.currentTimeMillis();
1050                log.log(Level.INFO, "       - TIME MOVING ARRIVED DESTINATION ITEM: " + (finalTimeMoveme
1051            } else {
1052                initialTimeMovement = System.currentTimeMillis();
1053                // If it is a door of stairs, then the next movement of the user will be directly to th
1054                boolean connectedStairs = false;
1055                try {
1056                    connectedStairs = this.pathStrategyUsed.checkDoorsConnectedByStairs(v1, v2);
1057                }
1058                catch (Exception e) {
1059                    e.printStackTrace();
1060                }
1061
1062                if (connectedStairs) {
```

Ilustración 47 - Función con el error

El bug ocurre cuando la casilla “Generation of Dynamic non-RS user paths” de la configuración de simulación está marcada.

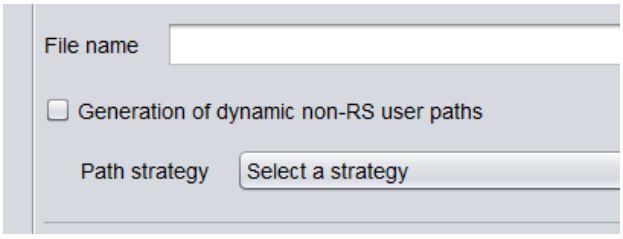


Ilustración 48 - Origen del error

Por tanto, al ejecutar la simulación se producirá una excepción en ese punto y la ejecución parará. Sin embargo, se ha comprobado que el funcionamiento de ese función no depende de nada que contenga el objeto Path (excepto otra función de Path, getConnectedStairs). Por tanto, se ha adaptado checkDoorsConnectedByStairs (y getConnectedStairs) y se ha colocado en Simulation. De esta forma, se puede usar esta función si el objeto Path es null y ya no se producirá la excepción.

```
            || ((v1 == 279 || v1 == 280 || v1 == 308 || v1 == 309) && (v2 == 241 || v2 == 242 || v2 == 275 || v2 == 276))) {
            currentTimeOfUsers[userPosition] += getTimeOnStairs();
        }*/
        if ((this.pathStrategyUsed != null && this.pathStrategyUsed.checkDoorsConnectedByStairs(v1, v2)) || checkDoorsConnectedByStairs(v1, v2)) {
            currentTimeOfUsers[userPosition] += getTimeOnStairs();
        } else {
            currentTimeOfUsers[userPosition] += getCurrentTime(location_v1, location_v2);
        }
    }

    finalTimeMovement = System.currentTimeMillis();
    log.log(Level.INFO, "      - TIME MOVING ARRIVED DESTINATION ITEM: " + (finalTimeMovement - initialTimeMovement));
    moveTime = finalTimeMovement - initialTimeMovement;
} else {
    initialTimeMovement = System.currentTimeMillis();
    // If it is a door of stairs, then the next movement of the user will be directly to the door input of stairs.
    boolean connectedStairs = false;
    try {
        connectedStairs = (this.pathStrategyUsed == null) ? checkDoorsConnectedByStairs(v1, v2) : this.pathStrategyUsed.checkDoorsConnectedByStairs(v1, v2);
    }
    catch (Exception e) {
        e.printStackTrace();
```

Ilustración 49 - Código de la función corregido

7. Conclusiones

Este proyecto ha resultado ser una experiencia muy positiva, ya que se han trabajado con tecnologías de las que no se tenía experiencia previa, por ejemplo, interfaces gráficas en Java con la biblioteca Swing, gestores de bases de datos espaciales como la extensión de PostgreSQL PostGIS o gestores de bases de datos de grafos como Neo4j. Otro aspecto interesante ha sido trabajar sobre un proyecto ya existente creado y modificado por otras personas, teniendo que investigar la estructura y funcionamiento del código y programar teniendo en cuenta eso como base. También se ha aprendido a ser constante y estructurar proyectos, además de pensar en el usuario.

Los objetivos pensados al iniciar el proyecto han sido cumplidos, aunque durante el mismo han surgido nuevos elementos que se podían añadir y estaban en línea con la meta de mejorar la usabilidad del programa, así como formas de ampliar los objetivos iniciales.

Durante los meses de trabajo han surgido dificultades, pero afrontarlas ha servido para aprender a buscar alternativas y ser creativo al programar, además de utilizar más en profundidad herramientas de depuración.

8. Trabajo futuro

Algunas posibles formas de expandir el proyecto serían:

- Compatibilidad del editor y el simulador con más formatos de archivo.

- Más funciones en el editor que ayuden en el proceso de creación de escenarios, como rejillas para ser más preciso al dibujar, fácil alineación de objetos en el espacio, posibilidad de usar una imagen de fondo como plantilla, o una opción que activada permite ver líneas que unen las puertas y escaleras según cómo están conectadas.

- Visualizador de grafos con más funciones para acercarlo más al visualizador de la aplicación de Neo4j.

Bibliografía

[1] Project NEAT-AMBIENCE (2023). Disponible en: RecMobiSim. https://webdiis.unizar.es/~silarri/prot/RecMobiSim/ [Consultado 22-01-2024]

[2] Instituto Universitario de Investigación en Ingeniería de Aragón – Universidad de Zaragoza (2024). Project NEAT-AMBIENCE. https://i3a.unizar.es/en/projects/neat-ambience [Consultado 22-01-2024]

[3] Wikipedia collaborators. Java Swing. https://en.wikipedia.org/wiki/Swing_(Java) [Consultado 23-01-2024]

[4] Apache Software Foundation (2023). Apache Batik. https://xmlgraphics.apache.org/batik/ [Consultado 22-01-2024]

[5] PostGIS PSC & OSGeo (2023). PostGIS. https://postgis.net/ [Consultado 22-01-2024]

[6] Neo4j, Inc. (2024). Neo4j. https://neo4j.com/ [Consultado 22-01-2024]

[7] GraphStream Team (2020). GraphStream. https://graphstream-project.org/ [Consultado 23-01-2024]

[8] Apache Software Foundation (2023). Apache Ant. https://ant.apache.org/ [Consultado 23-01-2024]

[9] Figma, Inc. (2023). Figma. https://www.figma.com/ [Consultado 22-01-2024]

Anexos

Fragmentos de código importantes

- areAdjacentWalls

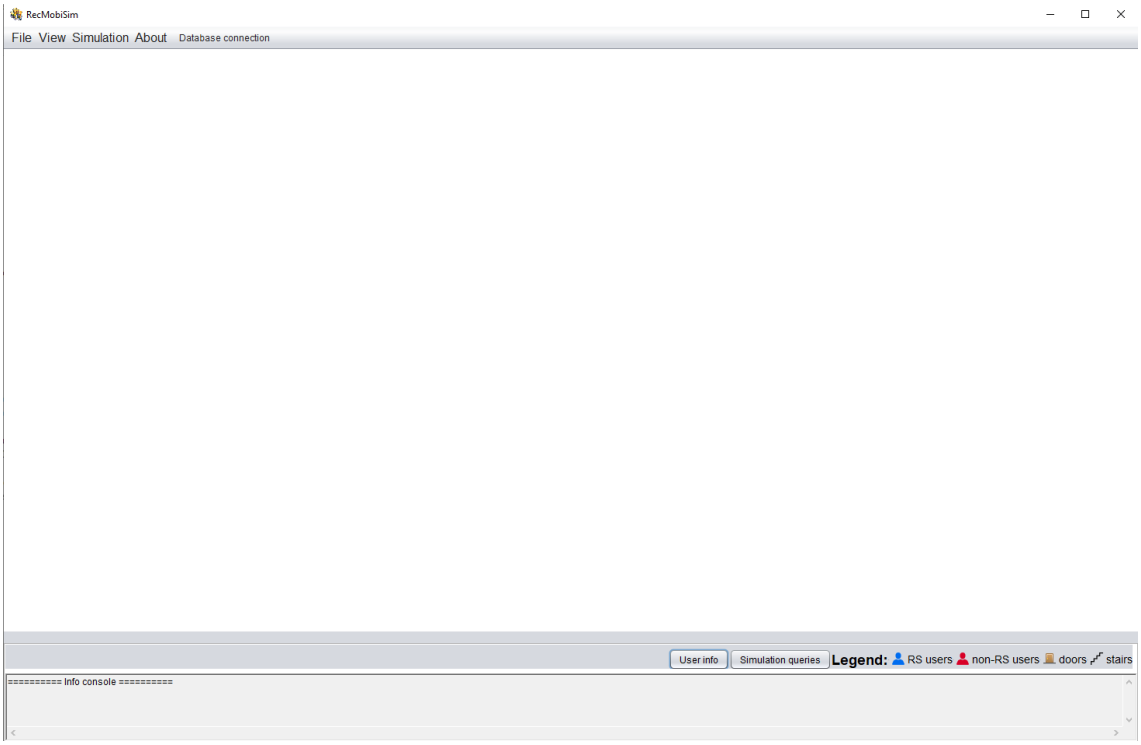
```
boolean areAdjacentWalls(Element line1, Element line2, Element lastLine) {
    if(line1 != line2 && line2 != lastLine) {
        if((Double.parseDouble(line1.getAttribute("x1")) == Double.parseDouble(line2.getAttribute("x1"))
        &&
        Double.parseDouble(line1.getAttribute("y1")) == Double.parseDouble(line2.getAttribute("y1")))
        ||
        (Double.parseDouble(line1.getAttribute("x2")) == Double.parseDouble(line2.getAttribute("x2"))
        &&
        Double.parseDouble(line1.getAttribute("y2")) == Double.parseDouble(line2.getAttribute("y2")))
        ||
        (Double.parseDouble(line1.getAttribute("x1")) == Double.parseDouble(line2.getAttribute("x2"))
        &&
        Double.parseDouble(line1.getAttribute("y1")) == Double.parseDouble(line2.getAttribute("y2")))
        ||
        (Double.parseDouble(line1.getAttribute("x2")) == Double.parseDouble(line2.getAttribute("x1"))
        &&
        Double.parseDouble(line1.getAttribute("y2")) == Double.parseDouble(line2.getAttribute("y1")))
        ) {
            return true;
        }else {
            return false;
        }
    }else {return false;}
}
```

- Inserción de elementos del mapa en el editor en el orden explicado al importar un SVG (paredes, habitaciones, separadores, imágenes, conexiones entre conectables)

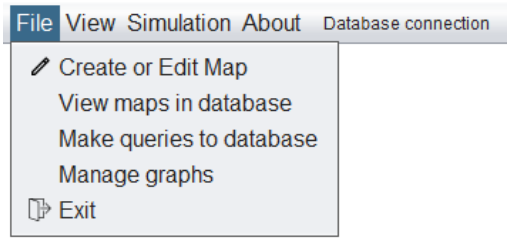
```
Element current;
for(Element w : walls) {
    saveElement(w);
}

while(walls.size() > 0) {
    Element e = walls.get(0);
    List<Element> roomWalls = lookForRoom(walls,e);
    if(roomWalls != null) {
        List<Corner> roomCorners = new ArrayList<Corner>();
        sortCorners(roomWalls);
        long lastCorner = 0;
        for(i = 0; i < roomWalls.size(); i++) {
            Element el = roomWalls.get(i);

            double c1_x = (Double.parseDouble(el.getAttribute("x1"))*2-model.getDRAWING_ICON_DIMENSION())/2;
            double c1_y = (Double.parseDouble(el.getAttribute("y1"))*2-model.getDRAWING_ICON_DIMENSION())/2;
            double c2_x = (Double.parseDouble(el.getAttribute("x2"))*2-model.getDRAWING_ICON_DIMENSION())/2;
            double c2_y = (Double.parseDouble(el.getAttribute("y2"))*2-model.getDRAWING_ICON_DIMENSION())/2;

            if(findCorner(roomCorners,c1_x,c1_y) == null) {
                Corner c1 = findCorner(corners,c1_x,c1_y);
                if(c1 != null) {
                    c1.setVertex_label((c1.getVertex_label() <= lastCorner) ? (lastCorner+1) : c1.getVertex_label());
                    roomCorners.add(c1);
                    lastCorner = c1.getVertex_label();
                }
            }
            if(findCorner(roomCorners,c2_x,c2_y) == null) {
                Corner c2 = findCorner(corners,c2_x,c2_y);
                if(c2 != null) {
                    c2.setVertex_label((c2.getVertex_label() <= lastCorner) ? (lastCorner+1) : c2.getVertex_label());
                    roomCorners.add(c2);
                    lastCorner = c2.getVertex_label();
                }
            }
        }
        if(roomCorners.size() == roomWalls.size()) {
            Room room;
            if(roomWalls.get(0).hasAttribute("roomLabel")) {
                room = new Room(Integer.parseInt(roomWalls.get(0).getAttribute("roomLabel")), roomCorners);
            }else {
                room = new Room(model.getNumRooms()+1, roomCorners);
            }
            if(rooms.stream().filter(r -> sameRoom(room,r)).findAny().orElse(null) == null) {
                //System.out.println(room.getCorners());
                if(!model.addRoom(room)) {
                    errorList.add("Couldn't add room to map");
                }else {
                    rooms.add(room);
                }
            }
        }
        for(Element w : roomWalls) {
            walls.remove(w);
        }
    }else {
        walls.remove(e);
    }


}

for (Element sep : separators) {
    saveElement(sep);
}

for (i=0; i<images.size(); i++) {
    current = images.get(i);
    //System.out.println(i);
    saveElement(current);
}

for (i=0; i<connectables.size(); i++) {
    current = connectables.get(i);
    //System.out.println(i);
    connectConnectable(current);
}
```

Cómo iniciar una simulación

Esta es la pantalla principal de RecMobiSim y la que se muestra al iniciar el programa. Es donde tienen lugar las simulaciones.

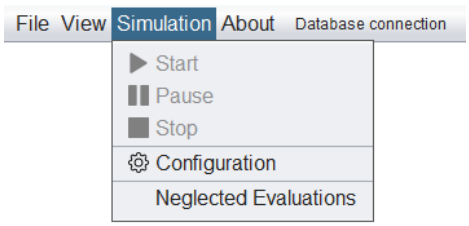


En la parte superior tiene una barra de menú con la que acceder a todas las funcionalidades. El primer menú desplegable es “File”, que contiene 5 botones.

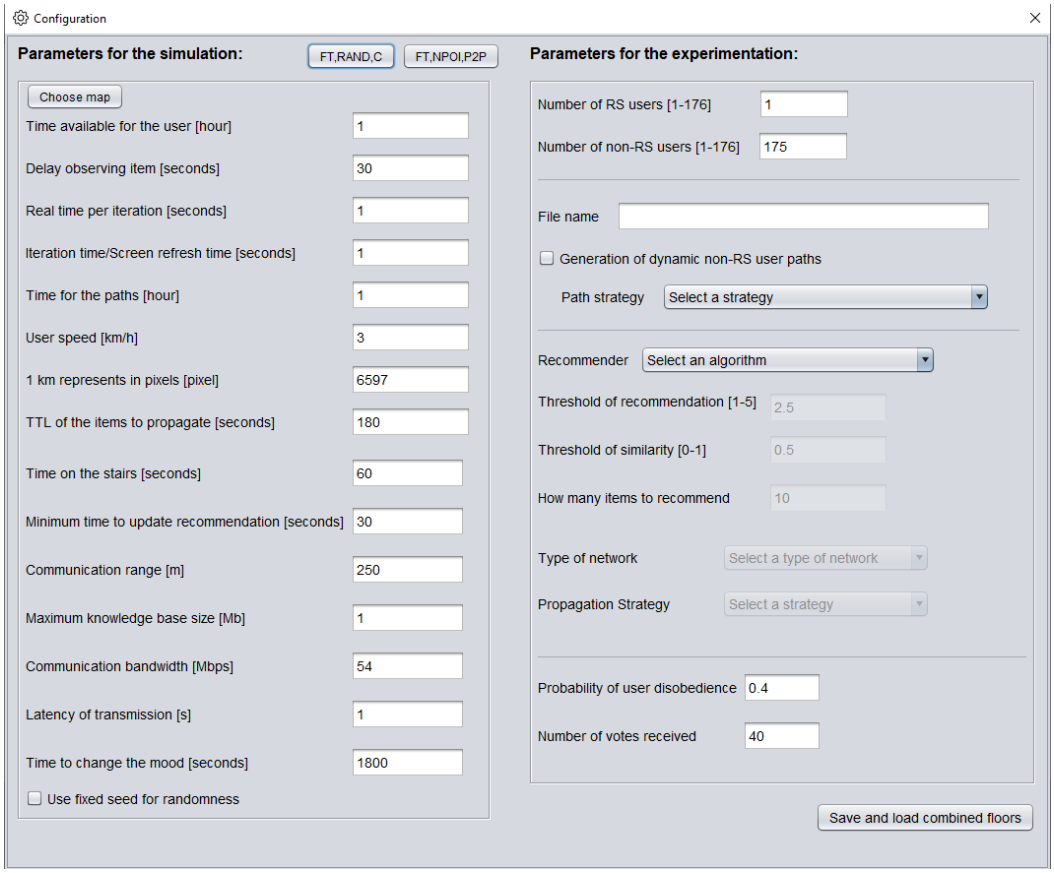


De arriba abajo, esos botones sirven para: acceder al editor de mapas, acceder al visualizador de mapas en PostgreSQL, acceder a la pantalla de consultas PostgreSQL, acceder a la pantalla de gestión de grafos y cerrar RecMobiSim. En el editor de mapas se pueden crear mapas sobre los que realizar simulaciones y guardarlos en archivos.

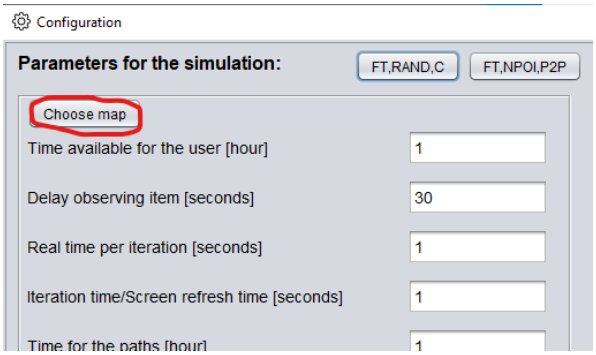
El menú “Simulation” se encarga de controlar las simulaciones. Los 3 primeros botones se encargar de iniciar, pausar/reanudar y terminar simulaciones. El botón Start está deshabilitado hasta que se cargue un mapa en el simulador. Pause y Stop están deshabilitados hasta que se inicie una simulación.

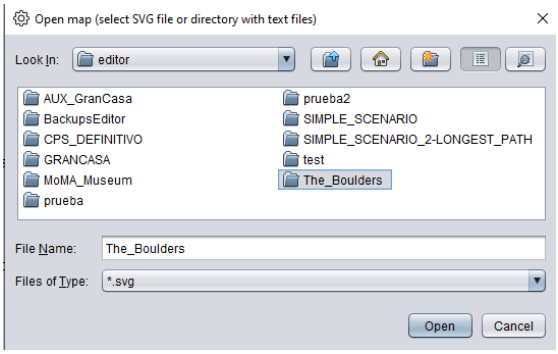


El botón “Configuration” abre el menú de configuración de simulaciones.

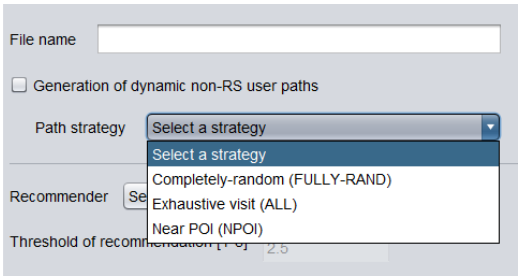


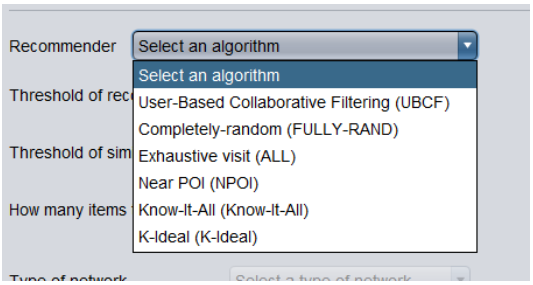
Con el botón “Choose map” se elige el mapa el que se quiere simular.



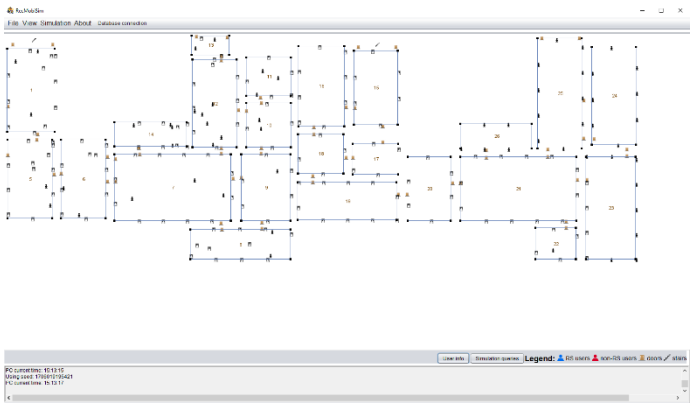


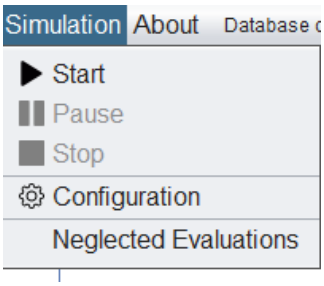
Se debe elegir una estrategia de generación de la ruta de los usuarios simulados sin sistema de recomendación, y el algoritmo de recomendación para los usuarios que sí lo tienen.





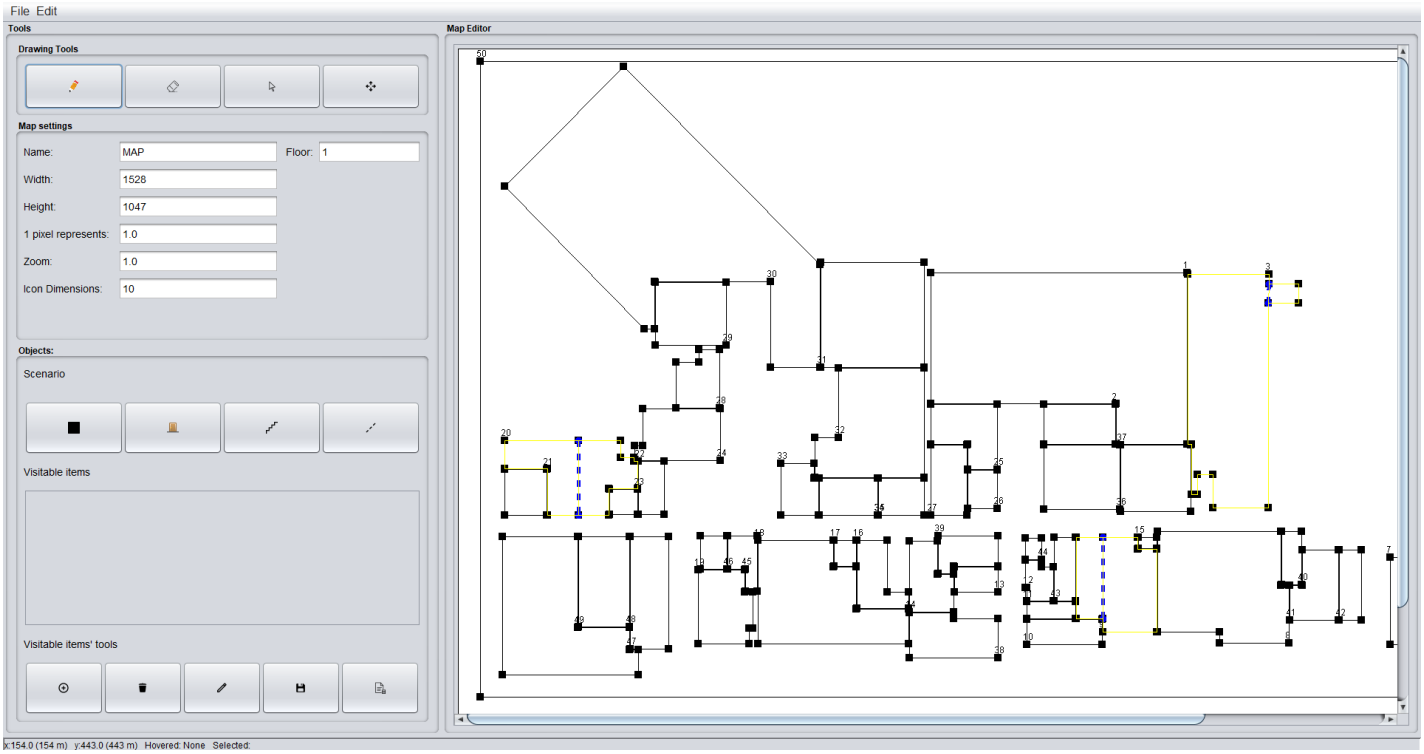
El botón “Save and load combined floors” carga un mapa para la simulación (si todos los parámetros están seleccionados).



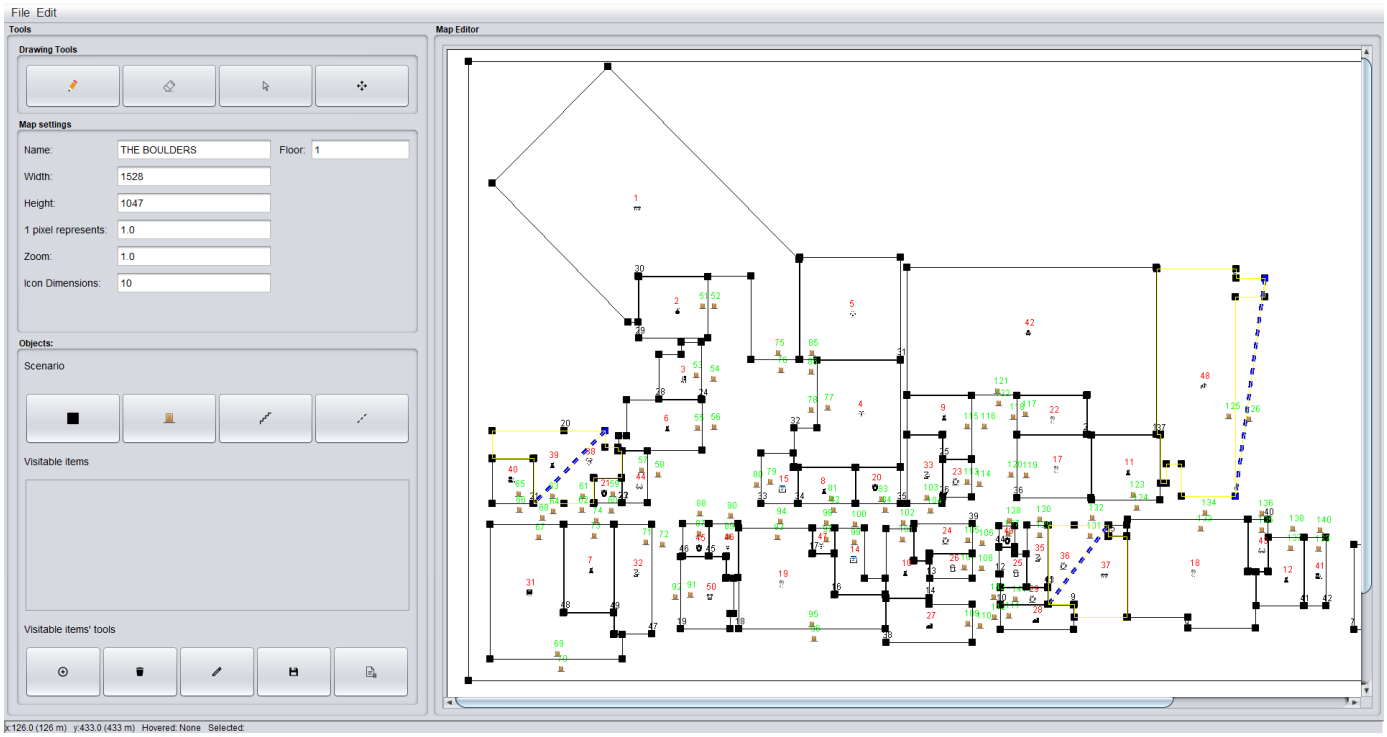


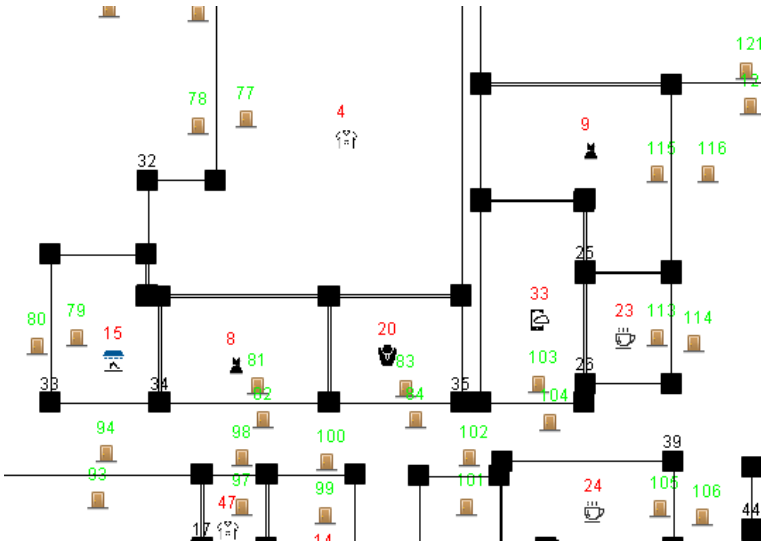
Ejemplo de uso de RecMobiSim mejorado para simulación de escenario

Juan quiere montar una tienda en un centro comercial. Él tiene un archivo del plano del centro comercial en SVG (para el ejemplo se está usando el plano del centro comercial The Boulders) que cumple los requisitos necesarios para el editor de mapas de RecMobiSim (las paredes que forman las habitaciones son elementos “line” de línea continua), por lo que puede abrirlo en el editor. En el ejemplo el SVG se ha dibujado y exportado con Figma, usando una imagen de un plano del centro comercial como modelo.

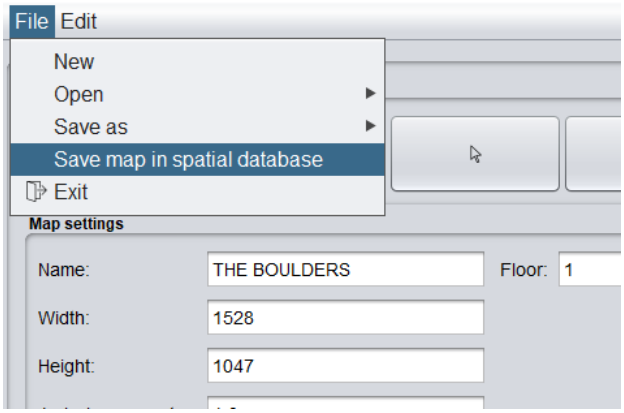


Con el editor de mapas se puede añadir elementos al mapa en el caso de que falten en el archivo original del mapa, como es el caso (solamente están representadas las paredes de las salas del centro comercial). Se insertan puertas y establecimientos. Cada habitación contiene un solo ítem o establecimiento, y una sola puerta que conecta con otra puerta contigua en el exterior (el exterior es representado con la habitación de label 50). Los establecimientos no son los reales.

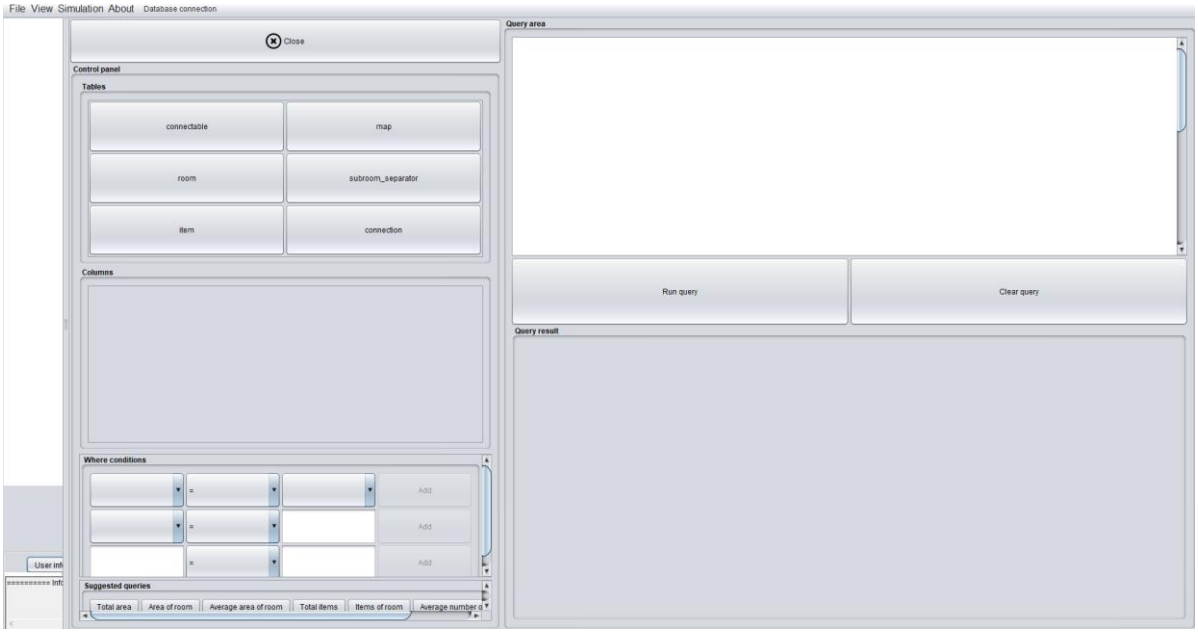

File Edit
Tools
Drawing Tools
Map settings
Name: THE BOULDERS
Floor: 1
Width: 1528
Height: 1047
1 pixel represents: 1.0
Zoom: 1.0
Icon Dimensions: 10
Objects:
Scenario
Visitable items
Visitable items' tools
Map Editor



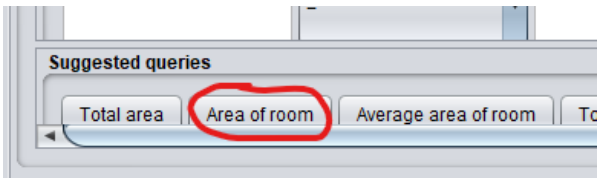
Si se almacena el mapa en la base de datos PostgreSQL + PostGIS, se podrán realizar algunas cosas interesantes.

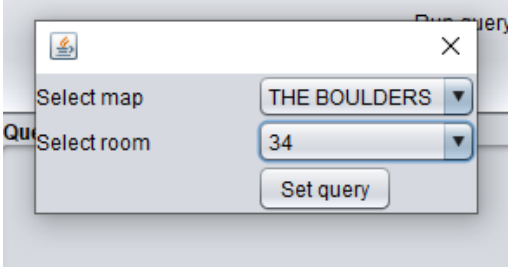


Por ejemplo, para saber el área de una sala. Primero, se abre la pantalla de consultas a PostgreSQL.

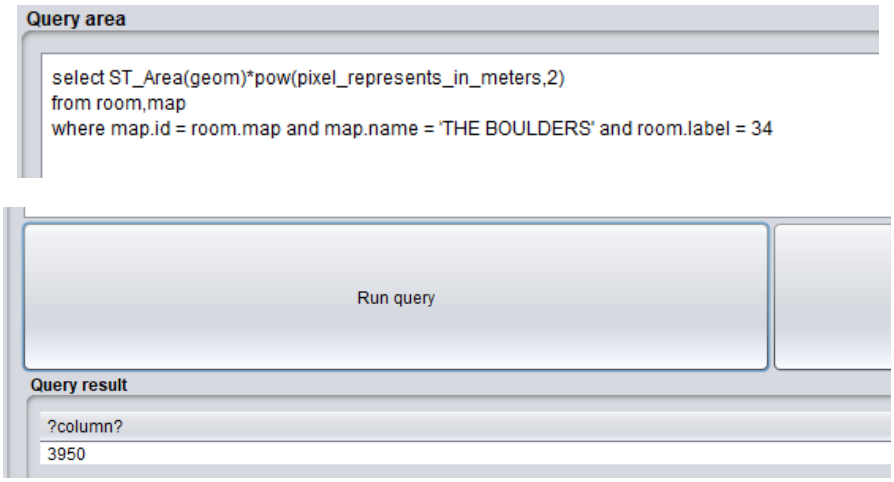


En el menú “Suggested queries” de la parte inferior del mapa se selecciona “Area of room” y se elige el nombre del mapa y el número (label) de la habitación que interese. Para el ejemplo se va a considerar la habitación 34 como posible lugar para que Juan coloque su tienda, por lo que se va a seleccionar esa.

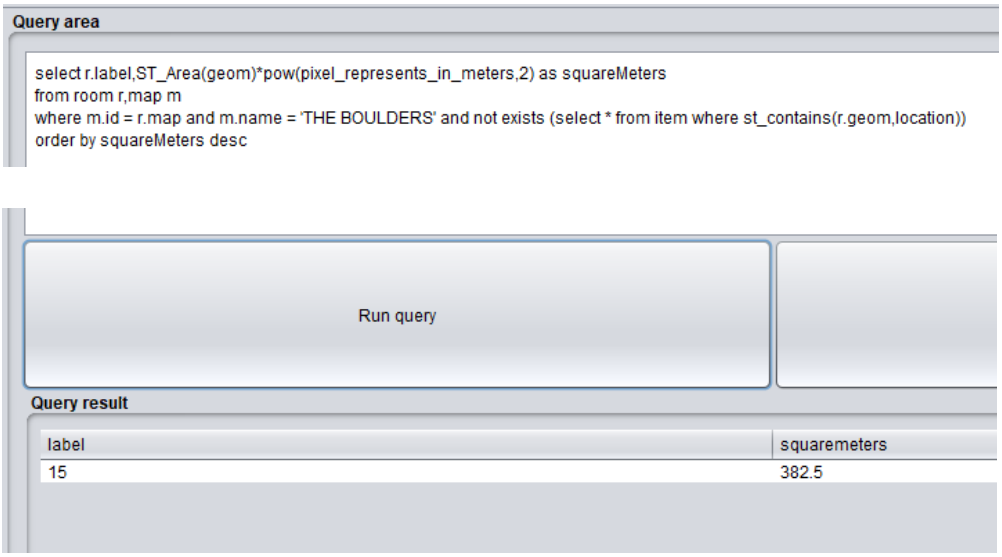




En el área de escritura de consultas se generará una consulta que devuelve el área en metros cuadrados de esa habitación.



También se puede usar una consulta alternativa (se puede usar el botón “Area of room” para crear la consulta anterior y luego modificarla un poco para conseguir esta) para listar todas las habitaciones vacías (sin item dentro) ordenadas de mayor a menor área, por si el resultado de la consulta anterior no es el deseado y se quiere una sala con un área mayor o menor.

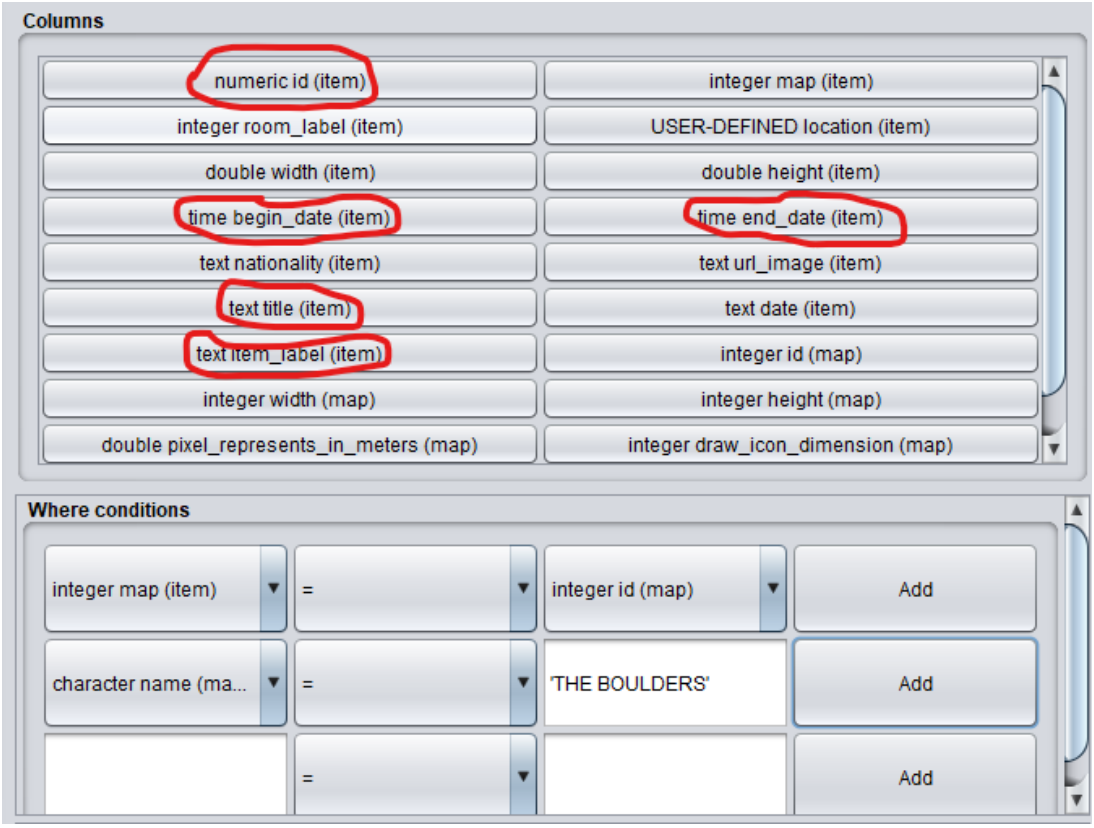


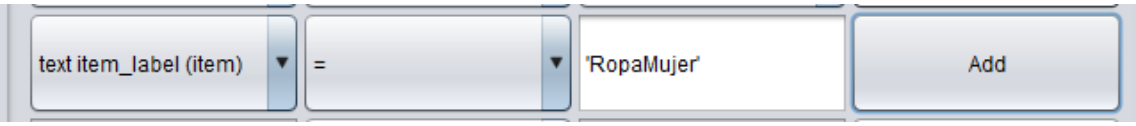
La pantalla de consultas se puede utilizar para muchas otras cosas, como mirar qué establecimientos que pueden hacerle competencia están abiertos en la mismas horas que el suyo.

Esto se puede hacer con el menú de personalización de consultas implementado:

Primero, se eligen las tablas “item” y “map”.

Luego, se seleccionan las columnas que interesan obtenerse durante la consulta (en la sección “Columns”) y las condiciones de where de la consulta (para hacer join de las dos tablas y obtener los datos del mapa del ejemplo y solamente de establecimientos que puedan hacer competencia, por lo que el item_label de la tabla item, o sea, la categoría, tiene que ser el mismo que el de la tienda de Juan).

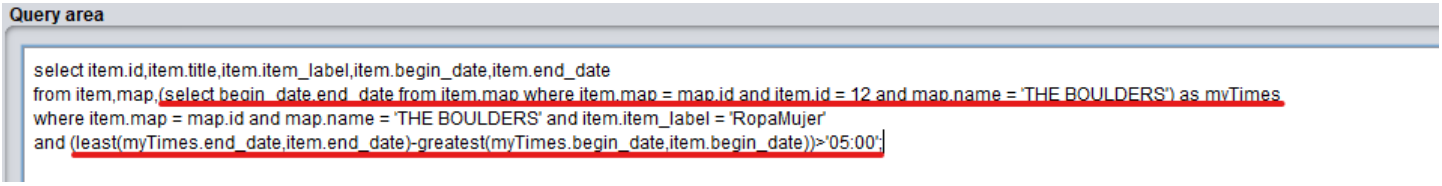




Con los pasos anteriores tenemos la base de la consulta, pero luego hay que modificarla a mano.

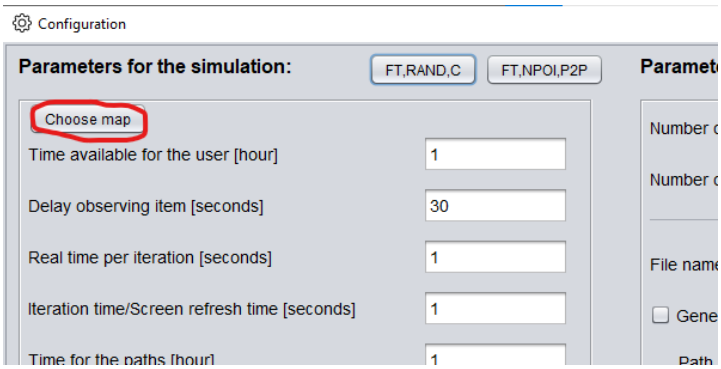
```
select item.id,item.title,item.item_label,item.begin_date,item.end_date
from item,map
where item.map = map.id and map.name = 'THE BOULDERS' and item.item_label = 'RopaMujer';
```

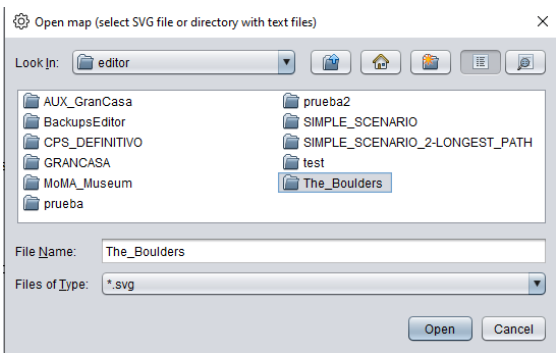
Se tiene que modificar:



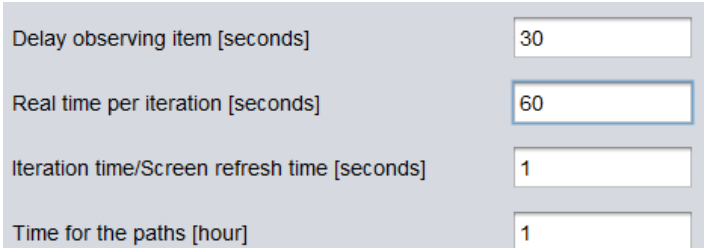
id	title	item_label	begin_date	end_date
6	Mango	RopaMujer	10:00:00	21:00:00
7	Zara	RopaMujer	10:00:00	22:00:00
8	H&M	RopaMujer	10:00:00	22:00:00
9	Primark	RopaMujer	10:00:00	22:00:00
10	Bershka	RopaMujer	10:00:00	21:00:00
11	AdolfoDominguez	RopaMujer	10:00:00	21:00:00
12	GranRopaje	RopaMujer	10:00:00	22:00:00
13	TejidosX	RopaMujer	10:00:00	22:00:00
39	Mulaya	RopaMujer	10:00:00	22:00:00

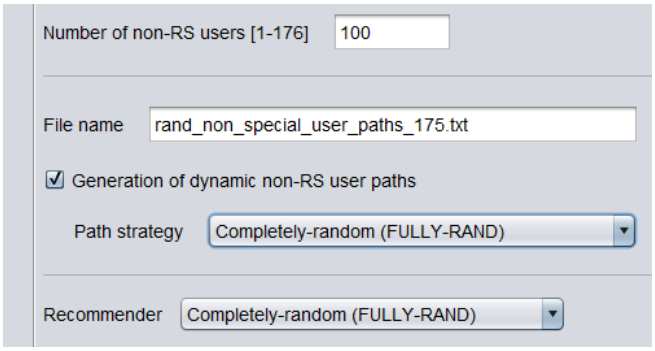
Una vez modificado el mapa con las puertas y establecimientos, se puede guardar en otro archivo SVG o en archivos de texto (en una carpeta nueva). Una vez hecho eso, se puede seleccionar en el menú de configuración de la simulación.

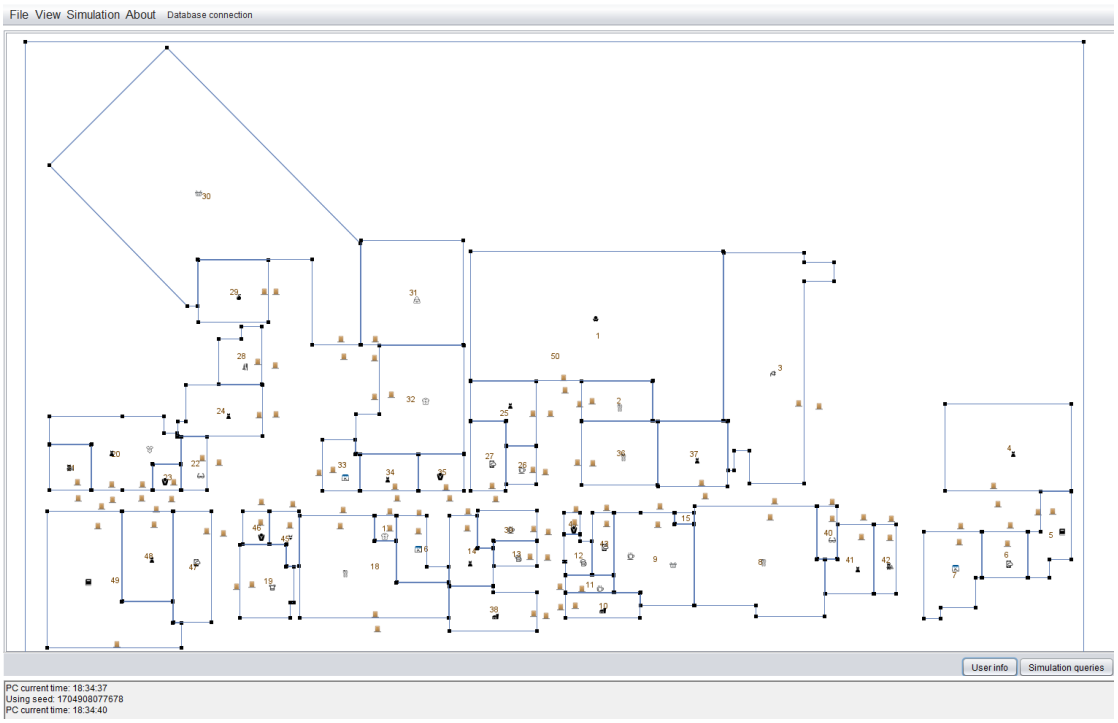




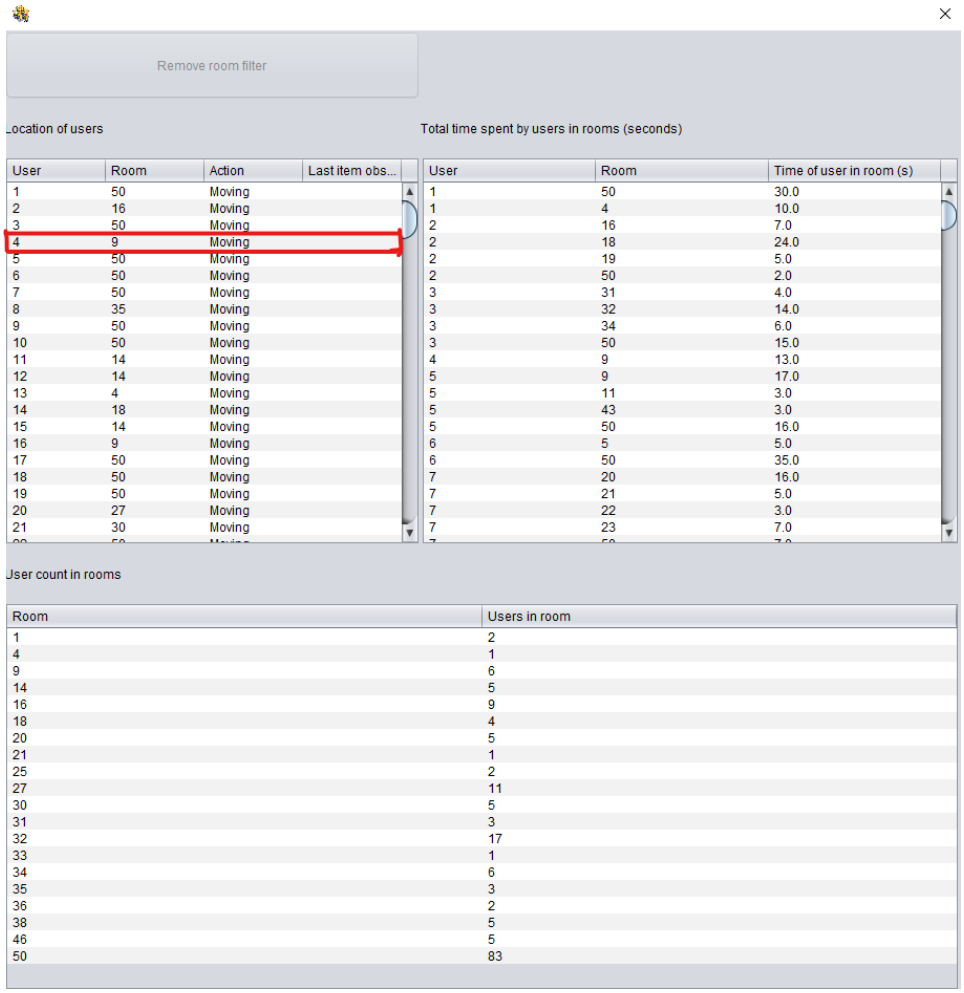
Se va a probar una simulación de ejemplo. Se va a escribir como tiempo por iteración 60 segundos, como número de usuarios no especiales 100, y se va a elegir como Path strategy y como Recommender el algoritmo Fully-rand, y se va a seleccionar la generación de rutas para usuarios sin RS (sistema de recomendación). Lo demás se deja como está.







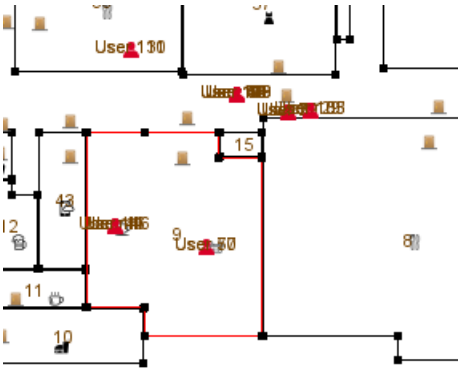
Tras iniciar la simulación, se puede seleccionar “User info” en la parte inferior de la pantalla de simulación para ver información, y al hacer doble click en las tablas se pueden localizar las habitaciones de las filas (para localizar una entre las más pequeñas es útil).



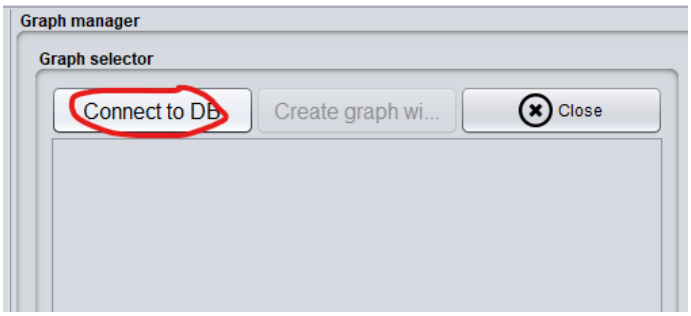
User	Room	Action	Last item obs...
1	50	Moving	
2	16	Moving	
3	50	Moving	
4	9	Moving	
5	50	Moving	
6	50	Moving	
7	50	Moving	
8	35	Moving	
9	50	Moving	
10	50	Moving	
11	14	Moving	
12	14	Moving	
13	4	Moving	
14	18	Moving	
15	14	Moving	
16	9	Moving	
17	50	Moving	
18	50	Moving	
19	50	Moving	
20	27	Moving	
21	30	Moving	

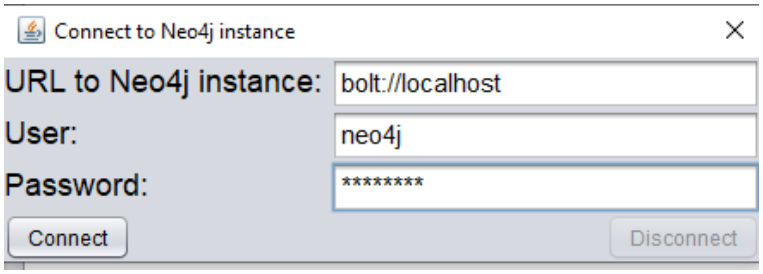
User	Room	Time of user in room (s)
1	50	30.0
1	4	10.0
2	16	7.0
2	18	24.0
2	19	5.0
2	50	2.0
3	31	4.0
3	32	14.0
3	34	6.0
3	50	15.0
4	9	13.0
5	9	17.0
5	11	3.0
5	43	3.0
5	50	16.0
6	5	5.0
6	50	35.0
7	20	16.0
7	21	5.0
7	22	3.0
7	23	7.0

Room	Users in room
1	2
4	1
9	6
14	5
16	9
18	4
20	5
21	1
25	2
27	11
30	5
31	3
32	17
33	1
34	6
35	3
36	2
38	5
46	5
50	83

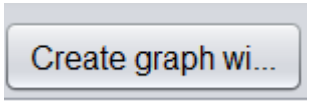


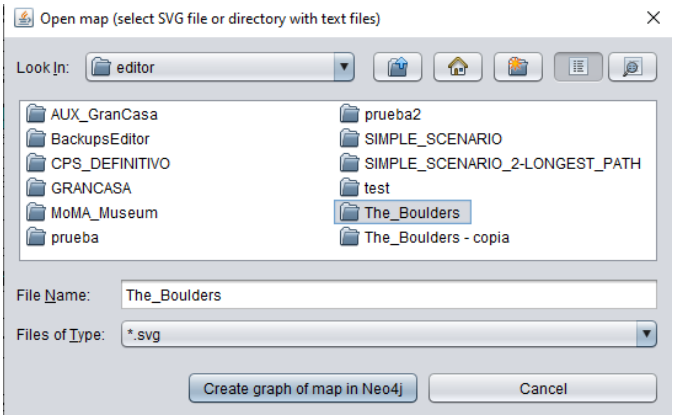
También se puede ir a la pantalla de grafos y conectarse a una base de datos Neo4j.



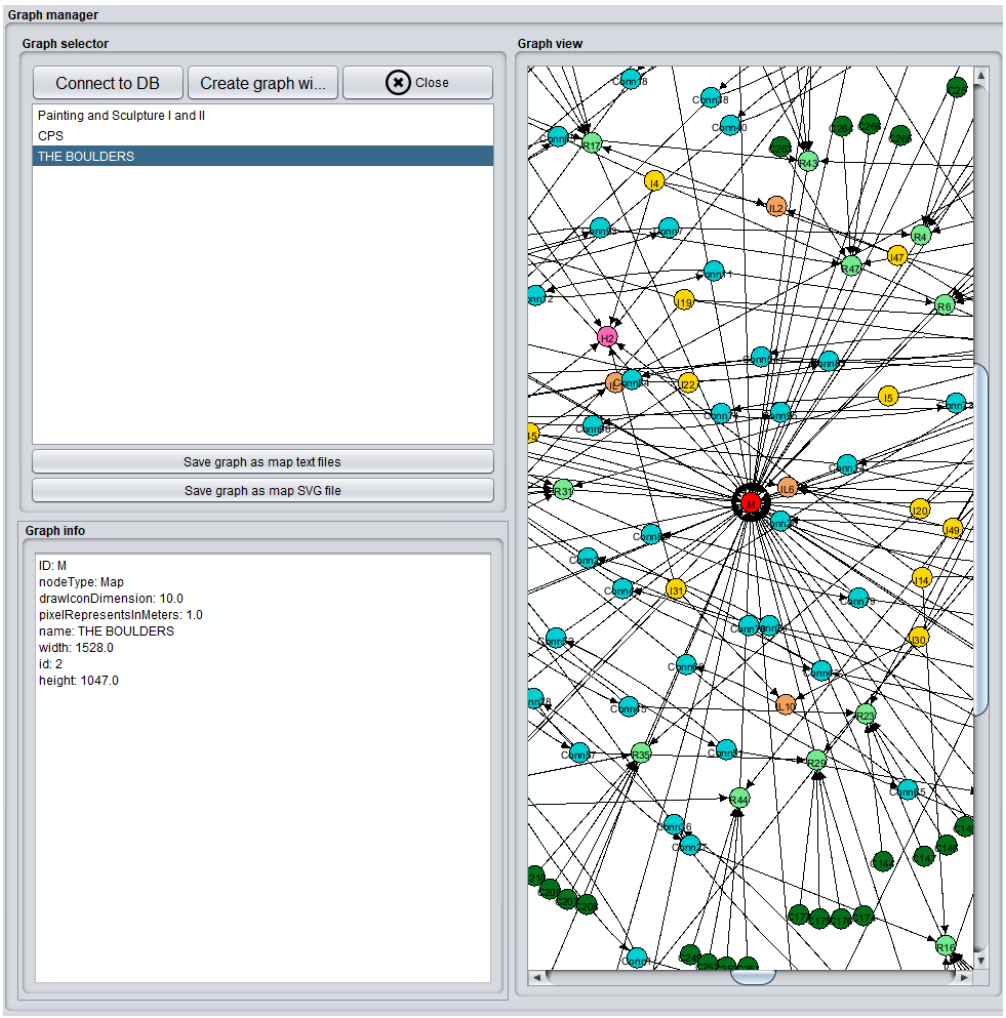


Una vez establecida la conexión se puede crear un grafo en esa base de datos seleccionando los ficheros (svg o directorio con txt) del mapa.





Al crearse el grafo y aparecer en el selector, si se hace click se generará una visualización del grafo, pudiendo ver la información de un nodo al hacer click en él.



También se puede usar el área de consultas para obtener más información del mapa. Por ejemplo, esta consulta para ver todos los establecimientos que cierran a una hora más tarde que las 23:00.