Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación de un Simulador para la Evaluación de Sistemas de Recomendación Dependientes del Contexto

Implementation of a Simulator for the Evaluation of Context-Aware Recommender Systems

Autor

Alejandro Piedrafita Barrantes

Director

Sergio Ilarri Artigas

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2022

AGRADECIMIENTOS

Me gustaría agradecer en primer lugar a Sergio Ilarri, director de este proyecto, el haber confiado en mí para la realización del mismo y la ayuda proporcionada en todo momento. Su labor como profesional ha sido ejemplar y ha sido un verdadero placer poder trabajar con él.

También quiero agradecer a mi familia y a mi pareja el apoyo incondicional y la motivación que me han proporcionado en los momentos más difíciles. Ellos han sido una parte esencial durante todo este proceso y me han animado a seguir adelante en los momentos buenos y también en los menos buenos. Sin ellos no hubiera sido posible.

Agradecer también a mis amigos y a mis compañeros del grado su apoyo y sus consejos. Me han ayudado a ver las cosas con perspectiva y a recargar fuerzas para poder continuar durante todo mi periodo académico.

Finalmente, también es relevante agradecer el apoyo por parte del proyecto de investigación PID2020-113037RB-I00 / AEI / 10.13039/501100011033 (proyecto NEAT-AMBIENCE) y del Departamento de Ciencia, Universidad y Sociedad del Conocimiento del Gobierno de Aragón (Gobierno de Aragón: referencia de grupo T64_20R, grupo COSMOS).

Implementación de un Simulador para la Evaluación de Sistemas de Recomendación Dependientes del Contexto

RESUMEN

En la actualidad cada vez se dispone de más información y menos tiempo, lo que dificulta a los usuarios la toma de decisiones en cualquier ámbito. Por este motivo cada vez toman una mayor importancia los Sistemas de Recomendación (*RS*), tanto como área de investigación como en el mundo empresarial. Su propósito es ofrecer a los usuarios recomendaciones personalizadas relevantes acerca de ciertos ítems. Además, el contexto también juega un importante papel sobre la percepción de ítems por usuarios. Los Sistemas de Recomendación Dependientes del Contexto (*CARS*) operan en un espacio tridimensional (*usuario-ítem-contexto*) para ofrecer recomendaciones de mayor calidad.

En este proyecto se implementa un simulador genérico con el que poder evaluar la calidad de las recomendaciones de los *CARS* en diferentes escenarios. Se toma como punto de partida un proyecto previo desarrollado con el propósito específico de realizar simulaciones sobre las plantas 4 y 5 del museo *MoMA* de Nueva York. Al proyecto inicial se le aplican optimizaciones para obtener un simulador funcional y se incorporan nuevas funcionalidades de uso. También se generaliza su código, facilitando su desarrollo y escalabilidad y permitiendo el reconocimiento de cualquier escenario.

Se desarrolla también una herramienta complementaria al simulador, el editor de mapas, la cual permite la creación de escenarios sobre los que poder realizar simulaciones. Se trata de una herramienta gráfica que permite dibujar los elementos del nuevo escenario sobre un lienzo, similar a otras aplicaciones de dibujo ampliamente conocidas. Su propósito es que cualquier usuario no experto sea capaz de crear un mapa de manera sencilla con el que posteriormente se puedan evaluar resultados en el simulador.

Por último, para la generación de conjuntos de datos sintéticos con los que evaluar *CARS* en los nuevos escenarios creados se hace uso de la herramienta *AUTO-DataGenCARS*.

Índice

Capítulo 1

Introducción y objetivos

En este capítulo se introduce el proyecto desarrollado, sus objetivos y el alcance del mismo (Sección 1.1), la metodología empleada (Sección 1.2) y el contenido de este documento (Sección 1.3).

1.1. Objetivos y alcance del proyecto

El motivo por el cual nace este proyecto es la existencia de un simulador de entornos móviles [1] creado para evaluar **Sistemas de Recomendación Dependientes del Contexto** o **CARS (Context-Aware Recommender Systems)** sobre un escenario concreto: las plantas 4 y 5 del museo MoMA de Nueva York [2, 3]. Dado dicho proyecto, surge la idea de poder crear nuevos escenarios para poder llevar a cabo simulaciones con las que evaluar CARS sobre ellos.

Por tanto, el principal objetivo del proyecto es desarrollar un simulador más genérico a partir del código del simulador del museo para poder configurar y representar otros escenarios. Para lograrlo, se generaliza el código del simulador para que sea capaz de reconocer cualquier escenario. Además, se implementa una herramienta complementaria al simulador con la que poder crear y/o editar mapas que serán reconocibles por el simulador.

Los datos a utilizar para llevar a cabo las recomendaciones sobre los escenarios creados (referentes a usuarios, ítems, contextos, perfiles de usuarios y valoraciones) podrán ser ya existentes (en caso de disponer de ellos) o generados de forma externa. Para la generación de datos, se considerará la herramienta *DataGenCARS* [4] o bien su reciente extensión con intefraz gráfica *AUTO-DataGenCARS* [5].

1.2. Metodología

A partir de un estudio del simulador del museo ya implementado y de herramientas de creación/edición de escenarios, se determina la metodología y herramientas a utilizar

en el proyecto para lograr una solución final, desarrollada mediante prototipos. Las herramientas utilizadas son:

- Lenguaje de programación *Java* (`https://www.java.com/`) para el desarrollo del proyecto. Este puede ser compilado y ejecutado fácilmente gracias a *Apache Ant* (`https://ant.apache.org/`).

- Librerías para el desarrollo de interfaces: *Swing* (`https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html`) y *AWT* (`https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html`) para la construcción de intefaces y *mxGraph* (https://jgraph.github.io/mxgraph/) para la representación de grafos.

- *JGraphT* (`https://jgrapht.org/`), librería de algoritmos y estructuras de datos de la teoría de grafos.

- *Mahout* (`https://mahout.apache.org/`) para la creación y uso de sistemas de recomendación.

- La herramienta *AUTO-DataGenCARS* [5], la cual extiende la herramienta *DataGenCARS* [4] y le añade una interfaz gráfica de usuario.

- *SQLite* (`https://www.sqlite.org/`) como Sistema Gestor de Base de Datos (SGBD) del que consultar los datos generados mediante *AUTO-DataGenCARS* y almacenar los referentes a las valoraciones generadas por los usuarios a lo largo de las simulaciones. Para conectarse a esta BD, se utiliza el Driver JDBC (`https://github.com/xerial/sqlite-jdbc`). Sin embargo, con las nuevas implementaciones, creando un nuevo *DAO* (Data Access Object) [6] podría llegar a utilizarse cualquier base de datos deseada (explicado en la Sección 4.1.2).

- Librerías para la depuración y control de mensajes de salida: *Java Logger* [7] y *Java Application Monitor (JAMon)* [8].

- Librería *OpenCSV* [9] para la exportación de estadísticas a ficheros con formato *CSV*.

El conjunto de estas tecnologías utilizadas en las diferentes partes de la arquitectura, algunas heredadas y algunas incorporadas en este trabajo, dan como resultado el producto final que cumple los objetivos propuestos.

1.3. Contenido del documento

La información de esta memoria está estructurada de la siguiente manera. En el Capítulo 2 se introduce el proyecto heredado, los algoritmos de recomendación y el estudio de herramientas similares. En el Capítulo 3 se recogen las nuevas funcionalidades a implementar y el análisis de requisitos. Las mejoras introducidas al simulador tanto necesarias para la optimización de su rendimiento como para su generalización de cara al reconocimiento y simulación de cualquier tipo de mapa se explican en el Capítulo 4. En el Capítulo 5 se presenta la nueva herramienta implementada (complementaria al simulador) y sus partes, explicando las funcionalidades recogidas en cada sección de la interfaz. En el Capítulo 6 se explica la creación de nuevos mapas (con el caso de estudio del mapa de *GranCasa*), la generación de los datos para los escenarios, los resultados del caso de estudio y el rendimiento del nuevo proyecto comparado con el original. Por último, el Capítulo 7 recoge las conclusiones y el trabajo futuro.

Además, en este documento se incluyen varios anexos que amplían la información presentada en la memoria principal. En el Anexo A se muestran los requisitos del proyecto. En Anexo B incluye un manual de uso para ejecutar simulaciones. En el Anexo C se detallan los parámetros de configuración del simulador. En el Anexo D se detalla la arquitectura del proyecto y cómo extenderla en futuros proyectos. En el Anexo E se presenta un estudio de posibles ítems para escoger los atributos comunes. Los mapas creados y su proceso de elaboración se explican en el Anexo F. En el Anexo G se explica el proceso de generación de datos y el rendimiento del simulador y se comparan con el proyecto original. Finalmente, en el Anexo H se exponen los resultados obtenidos con el nuevo simulador en el mapa del museo.

Capítulo 2

Conceptos claves y estudio de herramientas similares

En este capítulo se describen los algoritmos de recomendación existentes con los que poder llevar a cabo simulaciones sobre los escenarios (Sección 2.1), las características del proyecto heredado (Sección 2.2) y el estudio del arte de herramientas de creación de escenarios para la ejecución de simulaciones sobre estos (Sección 2.3).

2.1. Algoritmos de recomendación

Los *CARS* pueden emplear diferentes algoritmos con los que proporcionar recomendaciones a los usuarios. El propósito del simulador es evaluar el rendimiento de los diferentes algoritmos de recomendación [1, 3]. Los algoritmos disponibles son:

- **Completamente aleatorio (Fully-Rand)**: se recomienda un ítem al azar de los existentes en el escenario, sin importar la sala o planta en la que se encuentre.
- **Visita exhaustiva (ALL)**: a partir de su posición, se visitan uno a uno todos los ítems por orden, es decir, visita todos los ítems de la sala en la que se encuentra, se dirige a la siguiente sala y repite el proceso.
- **Punto de interés más cercano (NPOI)**: visita los ítems más cercanos. A diferencia del algoritmo anterior, se guía por el orden de cercanía en lugar del orden ítems-salas (por ejemplo, en una misma sala muy grande puede haber ítems muy alejados, existiendo otros más próximos a pesar de encontrarse en una sala diferente).
- **Filtrado colaborativo basado en usuarios (UBCF)**: las recomendaciones se basan en valoraciones del pasado tanto de otros usuarios con gustos similares como de los gustos propios del usuario/a. La información de estas valoraciones es intercambiada mediante una red de comunicación entre los usuarios. El tipo de comunicación puede ser *centralizada* o *P2P* (Peer-to-Peer) [10]. Los parámetros de simulación que determinan el intercambio de información afectarán

a los resultados del experimento a realizar. Estos parámetros son: tipo y rango de comunicación, tiempo de actualización de recomendaciones, velocidad de conexión, latencia y capacidad de almacenamiento, así como *TTL* (Time to Leave) [11] si la comunicación entre usuarios es *P2P*.

- **"Sabelotodo/Oráculo" (Know-It-All)**: similar al anterior pero se dispone de un "oráculo" que permite conocer toda la información de cada votación que va siendo notificada por todos los usuarios a lo largo de la simulación. De esta forma, se dispone de más información y se eliminan las restricciones intrínsecas de una red de comunicaciones.
- **K-Ideal (K-Ideal)**: algoritmo ideal que dispone de toda la información posible existente. De esta forma, el usuario con recomendador conoce qué valoración proporcionaría cada usuario a cada ítem en cada contexto (presente, pasado y futuro), pudiendo aplicar así las mejores recomendaciones al disponer de toda la información posible, y reordenando los ítems para obtener el camino más óptimo.

Como se puede observar, los tres primeros algoritmos (simples) son independientes al resto de usuarios. Además, los dos últimos algoritmos tienen información imposible de conseguir en un escenario real, pues conocen más información de la posible. Comparando los resultados de todos ellos se puede ver el valor de las recomendaciones basadas en filtrado colaborativo, siendo algo peores que los algoritmos ideales pero aportando una mejora respecto a los algoritmos simples (estudiado en el Capítulo 6 y el Anexo H).

2.2. Simulador del museo MoMA

Como se comenta en la Sección 1.1, este proyecto parte de una arquitectura ya existente [1, 2]. Dada la importancia que han ganado los *CARS* en los últimos años, se quiere explotar la información que provee el contexto en adición a los *Sistemas de Recomendación (RS)* básicos para ofrecer mejores recomendaciones a los usuarios sobre ciertos ítems [3]. Para estudiar el impacto que estos pueden tener, se crea un escenario en el que se plantea un caso de uso real, el cual consiste en la recomendación de ítems a observar en un museo. Sobre este escenario, se utiliza una aproximación basada tanto en la trayectoria como en un filtrado colaborativo basado en las valoraciones del resto de usuarios.

Además, los datos utilizados para probar este caso de uso se generaron mediante la herramienta *DataGenCARS* [4]. Esta herramienta consiste en un generador

sintético de conjuntos de datos (o "datasets"). Está diseñada para la evaluación de *CARS* utilizando escenarios mixtos basados en datos reales y sintéticos, dotándolos de valoraciones realistas y permitiendo analizar posteriormente la calidad de los "datasets" creados mediante gráficas.

Con el mapa del museo y los datos generados, se pueden realizar múltiples experimentos para evaluar los *CARS* variando los parámetros de la configuración de las simulaciones. Estos se dividen en dos principales grupos: de simulación y de experimentación. Los parámetros para la simulación permiten definir las características que afectan al escenario y todos sus elementos (usuarios, ítems, contextos, red de comunicación, etc.) durante la ejecución. Los parámetros de para la experimentación sirven para configurar el propio experimento que se va a realizar (usuarios de cada tipo, trayectorias de usuarios sin recomendador, algoritmo de recomendación, parámetros del algoritmo y red de comunicación empleada por los usuarios). La explicación en detalle de cada uno se recoge en el Anexo C.

2.3. Estudio de otros creadores/editores de escenarios

Como aspecto fundamental para la implementación de un simulador para la evaluación de *CARS*, es necesaria una herramienta que permita la creación de escenarios sobre los que se puedan llevar a cabo simulaciones para evaluar el desempeño de estos Sistemas de Recomendación en escenarios diferentes con características diferentes (extensión, distribución de ítems, tipos y características de dichos ítems, etc.).

Para ello, se realizó un estudio de las propuestas existentes relacionadas con la creación de escenarios simulables, para determinar si alguna de las mismas podría ser de utilidad. No se encontraron muchos ejemplos de creación de escenarios para simulación, por lo que se buscaron también referencias sobre creación de escenarios en otros campos, a pesar de que los objetivos fuesen completamente diferentes. Se encontraron algunas referencias en el área de los videojuegos. Las herramientas que se encontraron y consideraron fueron:

- *SimCity* [12]: videojuego destinado a la construcción y desarrollo de ciudades, teniendo en cuenta el control del tráfico. A raíz de *Simcity* se encuentra *Micropolis* [13], la versión antigua de este juego con versión *Java* (subdirectorio "*micropolis-java*" del repositorio) y bajo licencia GLP. Sin embargo, únicamente

permite modificar mapas ya creados e insertar elementos fijos predefinidos para el videojuego.

- *SimTower* [14]: similar a *SimCity* pero esta vez construcción de interiores con diferentes plantas. Se trata de un videojuego comercial, sin licencia de software libre.

- *citygen3d* [15]: permite crear escenarios 3D a partir de escenarios reales. Es una herramienta privada y de pago, además de estar desarrollada para *Unity* (https://unity.com/).

- *Tiled* [16]: editor de niveles 2D. Dispone de una versión Java, denominada *Tiled-Java* (`https://github.com/mapeditor/tiled-java`), y es una herramienta libre y de código abierto. Esta herramienta permite crear mapas (2D) y almacenarlos como archivos *TMX* [17], formato basado en *XML*, además de incluir la opción de exportar en formato *JSON*.

- *MobEnvSimulator* [18]: sistema para probar aplicaciones de computación móvil en un entorno distribuido. Permite probar servicios de datos inalámbricos reales en un entorno simulado de proxies y objetos móviles. Implementado en *Java* sin usar librerías ni herramientas auxiliares. Permite crear entornos simulables aunque su propósito es muy diferente al de este proyecto.

Tras analizar todas las herramientas y proyectos comentados, se llegó a la conclusión de que ninguno podía llegar a encajar, pues o bien no eran de licencia libre o bien no se ajustaban a las necesidades de este proyecto ya que estaban diseñadas para el propósito específico para el que fueron creadas y no podían extrapolarse. Aunque en algunos casos el formato de salida de los mapas podría llegar a reutilizarse y ser leído por el simulador, esto obligaría a hacer uso de dichas herramientas para la creación de mapas al no disponer de "APIs" de apoyo con las que integrar únicamente las funcionalidades deseadas en el prototipo de este trabajo, implicando un alto acoplamiento a dichas herramientas.

Cabe destacar que, a pesar de no poder apoyarse en ninguna librería o herramienta externa para la implementación del creador de mapas, estas referencias sí sirvieron como inspiración para la creación de la GUI (Interfaz Gráfica de Usuario) de la nueva herramienta integrada en el proyecto (haciendo especial mención a las dos últimas: *Tiled-Java* y *MobEnvSimulator*).

Capítulo 3

Análisis del simulador del museo y definición de requisitos

En este capítulo se presenta el análisis realizado, tanto del estado del proyecto existente (Sección 3.1) como de las nuevas funcionalidades a incorporar (Sección 3.2), para finalmente determinar el análisis de requisitos a satisfacer en el proyecto (Sección 3.3).

3.1. Análisis de funcionalidades a partir del estado del simulador del museo

El proyecto heredado tenía dos notables problemas: la lentitud y la falta de generalidad. Las simulaciones costaban horas e incluso días (abortadas manualmente tras más de 24 horas ejecutándose), independientemente de los parámetros de configuración establecidos. Las pruebas, igual que todo el proyecto, se han realizado en un ordenador portátil ASUS X555LJ, con sistema operativo Windows 10 Home 64 bits, procesador Intel(R) Core(TM) i3 (4 núcleos, 1.7GHz) y 8GB de RAM.

Dada esta lentitud, antes de añadir nuevas funcionalidades, lo principal era reducir los altos retardos del simulador para poder llevar a cabo ejecuciones. Para ello se debía depurar el código, comenzando por las instrucciones del algoritmo principal de simulación hasta las encontradas en funciones específicas, midiendo tiempos y analizando el rendimiento de cada una a lo largo de la ejecución para determinar los puntos de notable consumo de tiempo. De esta forma, se identificarían las funciones o instrucciones problemáticas y se podría visualizar su coste en tiempo y su desempeño a lo largo de la simulación para posteriormente optimizarse.

También se observó que, al tratar de usar la aplicación en un monitor cuyas dimensiones eran menores que el tamaño por defecto de las ventanas de la GUI, se ocultaba parte de las mismas, siendo imposible por ejemplo acceder al botón de guardado de la configuración y por tanto siendo imposible confirmar los parámetros

para comenzar una simulación. Por tanto, se determinó que tanto el simulador como el futuro editor de mapas debían ser redimensionables y ajustables a cualquier tipo de pantalla. Además, se debía modificar la configuración para ofrecer mayor claridad al usuario, impidiéndole seleccionar parámetros que no apliquen, además de brindar nuevas posibilidades bien en parámetros existentes o nuevos.

Debido al crecimiento de código que iba a tener el proyecto y que podría ser aún mayor en futuras extensiones del mismo, se concretó que era preciso desacoplar las distintas partes del código y encapsularlas de forma que pudiesen ser sustituidas con facilidad. Un ejemplo de este hecho es el caso de la posible utilización de nuevas bases de datos (con sus correspondientes SGBD).

Por último, se debía generalizar el código del simulador para ser capaz de interpretar cualquier tipo de mapa (generado con el editor de mapas) y, por tanto, poder realizar experimentos para evaluar el rendimiento de los CARS sobre estos. Estas evaluaciones se realizarían utilizando las métricas existentes (Anexo H) además de incorporar nuevas métricas que ofrezcan más información de la evolución de las simulaciones.

3.2. Análisis de nuevas funcionalidades

En cuanto a las nuevas funcionalidades, podemos encontrar dos tipos principales de enfoques: creación y edición de mapas o escenarios que posteriormente puedan ser utilizados para realizar simulaciones y la generalización y ampliación del simulador para poder reconocer dichos escenarios y realizar simulaciones en ellos.

Para el primer objetivo se ha de crear una herramienta integrada en el proyecto y complementaria al simulador que permita crear, editar, cargar y guardar mapas. Los mapas generados deberán ser legibles por el simulador pero los mapas existentes (museo MoMA) también deberán ser legibles por el creador de escenarios. Por tanto, los mapas a crear podrán ser mucho más grandes, complejos, de mayor número de plantas e incluir nuevos tipos de objetos visitables con las características concretas de cada instancia de los mismos, pero todo esto sin dejar de respetar las condiciones requeridas en el mundo real (como salas inaccesibles o paredes que se intersequen).

El segundo gran tipo de funcionalidades implementadas se basa en la legibilidad de escenarios generados con el creador de mapas y de datos de usuarios, ítems, contextos y valoraciones del escenario generados mediante herramientas externas destinadas a

ello. Además, se incorporarán mejoras de usabilidad a la interfaz, nuevas opciones de simulación y nuevas métricas recopiladas durante las simulaciones.

Finalmente, se proveerá de dos escenarios de prueba que puedan ser tanto simulados como editados en caso de desearse, como ejemplo del resultado de este proyecto. Además, se proveerá de escenarios adicionales que demuestren la versatilidad del editor de escenarios, sin datos generados para ser simulados.

3.3. Análisis de requisitos y diseño de la arquitectura

El resultado de la combinación de las dos secciones anteriores (Sección 3.1 y Sección 3.2) es la definición del documento del Análisis de Requisitos. Este documento recoge y explica todos los aspectos claves incluidos o mejorados en este proyecto. El listado completo de requisitos funcionales y no funcionales puede encontrarse en el Anexo A. La explicación del desarrollo y la localización de estos requisitos se recoge en los siguientes bloques: ampliación y generalización del Simulador (Sección 4), creación del Editor de Mapas legibles por el Simulador (Sección 5) y resultados del proyecto (Sección 6).

A partir de este análisis se diseña la arquitectura del proyecto, extendiendo y modificando la arquitectura del simulador original. La Figura 3.1 ilustra la arquitectura final del proyecto, mostrando y explicando la disposición de los diferentes paquetes que lo componen, las relaciones entre sí y las tecnologías utilizadas en cada uno de ellos. En el Anexo D se profundiza más sobre esta información.

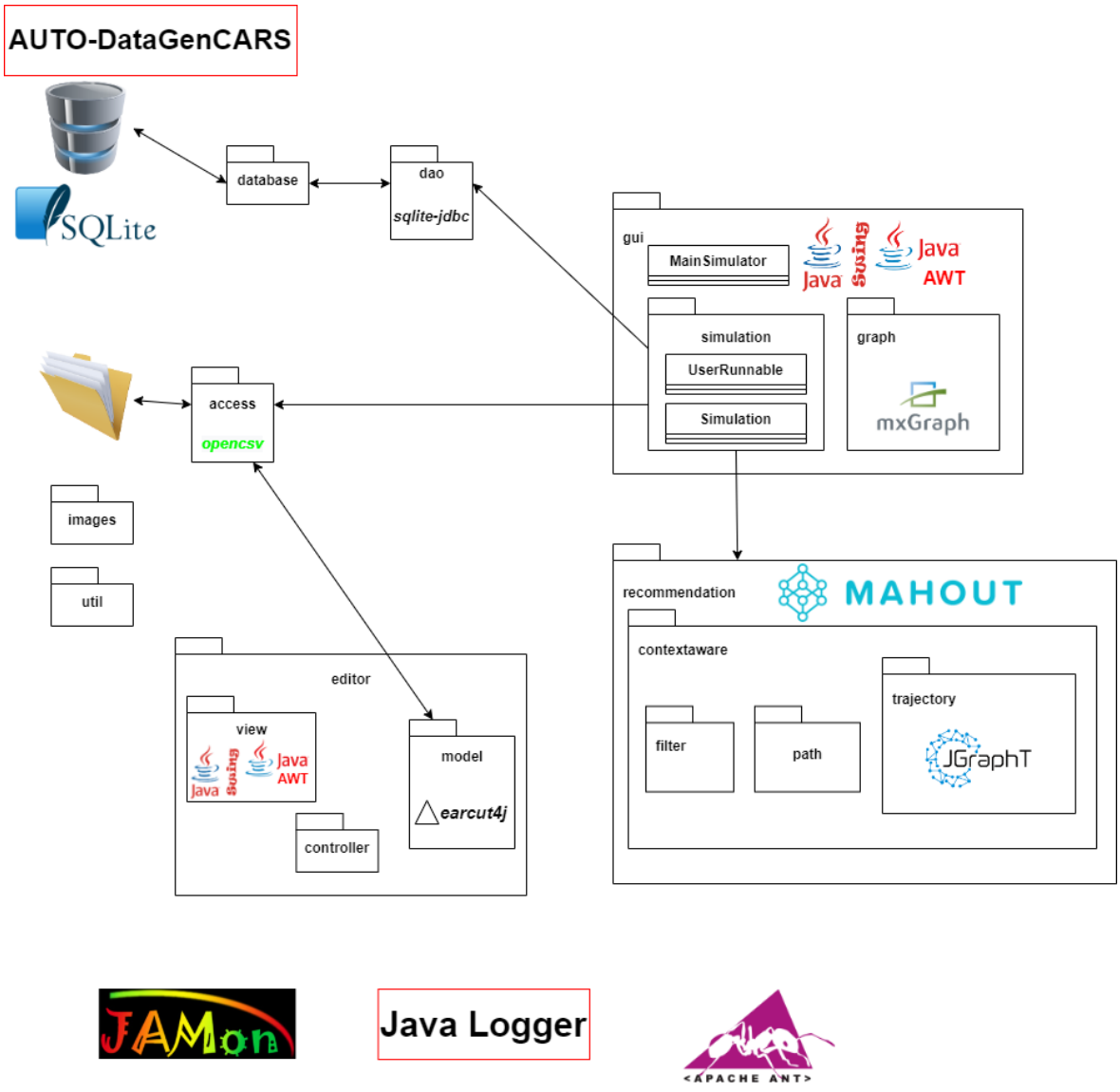


Figura 3.1: Arquitectura del proyecto y tecnologías utilizadas

Capítulo 4

Optimización y generalización del simulador

En este capítulo se explican las optimizaciones realizadas al simulador existente para su correcto funcionamiento (Sección 4.1) y las modificaciones aplicadas para su generalización y ampliación (Sección 4.2).

4.1. Optimización del funcionamiento del simulador

Como ya se ha analizado previamente (Sección 3.1), se sabe que la lentitud del simulador heredado es muy grande, volviendo imposible la finalización de las simulaciones (sin importar la configuración especificada). Por este motivo, su depuración e identificación de puntos de fallo y sobre todo de alto consumo en tiempo es vital para obtener un simulador funcional e identificar cuellos de botella para poder realizar simulaciones en este proyecto.

Todas las funciones tanto del algoritmo principal como de cada ciclo de simulación se encapsulan, midiendo sus tiempos de inicio y fin, para calcular así el tiempo consumido de cada una de ellas. Tras conocer las funciones más costosas, es decir, cuyo coste en tiempo no es despreciable, se repite este proceso profundizando en ellas hasta dar finalmente con todas las instrucciones/operaciones concretas de las distintas funciones que ralentizan la simulación.

Cuando se profundiza en la depuración, se observa que muchas de estas funciones incorporan bucles, se invocan numerosas veces y desde diversas partes del código, varían su comportamiento en función de sus llamadas y, en definitiva, añaden una complejidad extra que hace difícil la medición del coste en tiempo de las mismas. Para un mejor control, una mayor trazabilidad y un análisis de rendimiento más claro y eficaz, se hace uso de *JAMon* (*Java Application Monitor*) [8] una API que permite la monitorización de aplicaciones Java de forma simple y segura. Además, esta API es rápida y prácticamente no consume memoria, aspecto vital para la depuración del

proyecto para no degradar aún más las prestaciones tratando de mejorarlas.

4.1.1. Control del buffer

El número de mensajes mostrados tanto por la consola de la aplicación como por la consola *Java*, además de los nuevos mensajes de depuración de tiempos, era insostenible. La impresión de mensajes era constante, mostrando información de cada acción específica de cada usuario en cada ciclo de simulación. Se modifica el simulador para permitir la ejecución con cualquier número de usuarios (sin sobrepasar el máximo), haciendo posible la ejecución del escenario más sencillo posible: dos usuarios, uno que usa el recomendador y uno que no. Esta modificación facilita la labor de depuración, pudiendo identificar los tiempos más notables, aunque el número de mensajes impresos sigue siendo demasiado elevado.

Se descubre que, entre otras, las operaciones de impresión de mensajes (ya existentes en el proyecto original) consumen un tiempo no despreciable y creciente a lo largo de la simulación. Estos tiempos disminuyen en las simulaciones con el menor número de usuarios, pero siguen creciendo durante las ejecuciones. Se concluye que las operaciones I/O (de entrada y salida) causan una degradación de las prestaciones (saturación del buffer), motivo por el cual el tiempo de estas operaciones es no despreciable y ascendente en el tiempo.

Para el control de mensajes en el proyecto, se implementan dos "*Java Loggers*" [7], uno para los mensajes de información (ya existentes) y otro para los mensajes de rendimiento (incorporados para medir tiempos). A estos "loggers" se les atribuyen una serie de características que permiten una actuación diferente en cada caso.

Se encapsulan todos los mensajes en los "loggers" y se les asigna a cada uno un nivel en función de su importancia, el cual se comprueba con el nivel asignado a su "logger" correspondiente para determinar si han de imprimirse. Para solventar el problema de degradación de prestaciones del buffer, se establece el nivel más elevado a ambos "loggers", resultando en un modo "no verboso" en el cual se imprimen únicamente excepciones o fallos, así como una retroalimentación básica de la correcta evolución de la simulación o tras las acciones realizadas por el usuario.

En el caso del "logger" de depuración, además del control por niveles, se implementa un filtro adicional que permite imprimir únicamente mensajes que representen tiempos

y cuyos valores superen un umbral mínimo indicado. Este filtro permite eliminar mensajes irrelevantes o con tiempos despreciables y por tanto acotar las funciones y/o instrucciones costosas. Además, los "loggers" están dotados de un manejador ("Handler") que reenvía los mensajes a sus salidas asignadas (consola GUI de la aplicación o consola *Java*) y de un formateador ("Formatter") que especifica el formato de salida en función del tipo de mensaje y su nivel.

De esta forma se soluciona el problema de tiempos referentes al "buffer". Con esta medida, siguen sin finalizar las simulaciones con todos los usuarios (se tienen que abortan como en el estado inicial) pero se consigue finalizar aquellas que representan el escenario más simple (dos usuarios) en un par de horas (tiempo muy similar entre todas las ejecuciones, sin importar la velocidad de simulación especificada en la configuración de cada una). Además, siempre se puede incrementar la "verbosidad" de las simulaciones modificando únicamente el nivel de los "loggers" (asumiendo eso sí un decremento sustancial del rendimiento).

4.1.2. Mejora del acceso a datos

Con la eliminación de la degradación de prestaciones provocada por los mensajes y el filtrado de mensajes de consumos de tiempo, se reducen las operaciones costosas. Se reconoce en todas las restantes un patrón común: son operaciones de manejo de datos. Estas operaciones acceden a dos tipos de fuentes de datos: información del mapa, almacenada en ficheros de texto; e información de las características del mapa (usuarios, ítems, contextos y valoraciones de cada posible combinación usuario-ítem-contexto que se pueden generar durante las simulaciones), almacenada en BD (base de datos).

El primer tipo de acceso a datos (en ficheros de texto) provoca un problema de degradación de prestaciones similar al comentado en el apartado anterior (Sección 4.1.1), con operaciones de entrada en este caso. Su solución consiste en la reducción de accesos, realizando la carga de datos de los ficheros una única vez al comienzo de la simulación y consultándolos posteriormente en memoria.

La solución de los accesos a BD, en cambio, es más compleja. Para un acceso controlado y una mayor escalabilidad, se aplican diversos patrones de diseño. En primer lugar, se quiere encapsular el acceso a las fuentes de datos para tener un mayor control, definir el modo de acceso y separar estas operaciones de la lógica del

proyecto. Para ello, se aplica el patrón *DAO* (Data Access Object) [6], definiendo las interfaces con las operaciones necesarias para el acceso a cada fuente de datos. En este caso las interfaces creadas se aplican para el SGBD (Sistema Gestor de Base de Datos) *SQLite*, utilizado en el proyecto, pero dados los patrones de diseño aplicados podrían implementarse de manera análoga para cualquier SGBD deseado (utilizando el lenguaje propio de la misma).

Existen varias fuentes que se pueden consultar durante la simulación por los usuarios. Si el tipo de red de comunicación seleccionada es centralizada, los usuarios accederán a una única BD que contendrá toda la información generada a lo largo de la simulación por todos ellos. Si la comunicación es *P2P*, cada usuario mantendrá su propia BD de valoraciones, así como una "cola" (nombrada así dada su estructura de datos) con la información intercambiada en la comunicación con otros usuarios de la red durante la simulación. La información permanecerá en la red por un tiempo limitado (TTL). Además, dado que los usuarios son simulados, no pueden otorgar valoraciones a los ítems según sus gustos como podrían hacer las personas, por lo que en todos los casos se accede a la BD que contiene toda la posible información del mapa generada previamente para simular cada caso posible (valoraciones de cada usuario a cada ítem bajo cada contexto).

Para que la posibilidad de uso de cualquier fuente de datos sea real, se han de definir los *DAOs* a utilizar para cada tipo de BD accedida en la simulación en el momento de su instanciación. Para mantener la independencia de la creación, composición y representación de estas clases, se puede aplicar el patrón de diseño *Factory* [19]. Esto, sin embargo, requiere de la fábrica correspondiente encargada de crear las clases *DAO* de cada SGBD. Por tanto, se aplica el patrón *Abstract Factory* [20], pudiendo así crear la fábrica que a su vez cree las clases correspondientes para el SGBD utilizado de forma completamente transparente para el usuario. El diseño de la combinación de patrones comentados se encuentra en el Anexo D.2.

Con la implementación de los patrones anteriores, los tiempos de acceso a base de datos no mejoran, por lo que se crea una "Base de Datos en Memoria", la cual contiene y gestiona toda la información que debía almacenarse durante toda la simulación y la almacena toda a la vez al finalizar. Su uso sí decrementaba los tiempos, determinando que el problema residía en la gestión del acceso a datos. Esta solución permitió depurar inicialmente los problemas de tiempos, pero incrementa considerablemente el uso de memoria por parte de la aplicación, por lo que finalmente se descarta, manteniéndose

sus clases únicamente como prueba de la versatilidad de los patrones implementados.

Para la optimización del acceso a datos, se modifican las consultas, haciendo uso de "*Prepared Statements*" [21]. Esta característica permite reutilizar el plan de ejecución de aquellas consultas cuya estructura no varía a lo largo de la ejecución, sino únicamente sus parámetros (o nada si se trata de consultas neutras). También, al existir un único proceso que accede a base de datos, se desactiva el uso del "*autocommit*", modo de operación que provoca un "commit" por cada transacción. La modificación de este parámetro de configuración provocaba excepciones al existir múltiples conexiones a base de datos (en los accesos desde distintos puntos del código), generándose problemas de concurrencia al existir varios accesos a la misma fuente de datos sin confirmar las transacciones. Para la unificación y reutilización de conexiones, se aplica el patrón de creación *Singleton* [22], creando así una única instancia y proporcionando el acceso a la misma cuando se necesita.

Sin embargo, se seguían produciendo excepciones. Se descubrió que el uso de la librería "MOONRISE.jar", librería desarrollada en proyectos previos para el uso de sistemas de recomendación, también creaba nuevas conexiones, obviando las medidas anteriores. Para la solución de este problema se tuvo que recuperar los fuentes de esta librería, integrar los requeridos en el proyecto y aplicarles las medidas comentadas anteriormente para conseguir un acceso a datos y una reutilización de conexión reales en todo el proyecto.

Con esta integración y adaptación de los fuentes se solventan finalmente los problemas de retardos, consiguiendo realizar simulaciones que se adaptan perfectamente a los tiempos esperados dados los parámetros de configuración (Anexo C). Por este motivo, se descarta la solución previa de "Base de Datos en Memoria", pues ahora no incorpora ninguna mejoría en tiempo y, como ya se ha comentado, consume una gran cantidad de memoria hasta el almacenamiento final de la información en la base de datos, lo que puede generar errores de pila (o "heap").

4.2. Generalización y ampliación del simulador

Una vez optimizado para así poder llevar a cabo simulaciones y extraer resultados, se modifica el simulador para introducir nuevas funcionalidades y se generaliza para el reconocimiento de cualquier escenario. Para comprobar el correcto funcionamiento del simulador con las modificaciones incorporadas, se llevan a cabo los experimentos

recogidos en [1] y se comprueba que los resultados son equivalentes (Anexo H). En cada simulación existen ligeras variaciones en los resultados obtenidos debido a la aleatoriedad inherente a estas (que determina por ejemplo el lugar de comienzo de los usuarios o la probabilidad de desobediencia frente a las recomendaciones).

Cabe destacar que los datos referentes a la predicción de errores (utilizados para obtener las Figuras 6.3 y H.3) no se encontraron en ninguna parte del código en la versión del proyecto tomado como punto de partida, no sólo su exportación a fichero sino incluso su cálculo y recopilación. Por tanto, se reimplementó esta recopilación de estadísticas durante la simulación y su exportación a fichero *CSV* utilizando la librería *OpenCSV* [9]. De esta forma es posible replicar completamente los experimentos ya comentados. Además, se implementan nuevas métricas que pueden ser de interés y aportar un valor extra a los resultados. Las métricas incorporadas son:

- Número de usuarios que miran el mismo ítem que un usuario con recomendador durante el tiempo que este se detiene a mirar un ítem.
- Número de veces que un usuario se encuentra a una distancia menor que una distancia umbral de otro usuario durante un tiempo como mínimo igual a un tiempo umbral.

Otro aspecto a destacar es la modificación de interfaz. Se modifica su diseño para poder adaptarse a cualquier tamaño de pantalla. La posibilidad de redimensión y desplazamiento ("scroll") de las vistas es importante para facilitar la visualización pero, sobre todo, para permitir la ejecución de simulaciones. Este factor es importante ya que en un inicio, las dimensiones de las ventanas eran fijas y superaban las de la pantalla del dispositivo utilizado para el desarrollo de este proyecto.

Este hecho implicaba, entre otras cosas, la imposibilidad de acceso a ciertos campos en la ventana de configuración, así como su botón de confirmación, siendo necesaria una pantalla auxiliar de mayor tamaño para ello y así poder realizar simulaciones (como se explica en el manual de usuario que se encuentra en el Anexo B).

También se mejora la usabilidad en la vista de configuración, mejorando la gestión de parámetros de experimentación impidiendo combinaciones innecesarias (selección del tipo de red de comunicación en algoritmos de recomendación que no la utilizan) y permitiendo simulaciones para cualquier número de usuarios siempre que no se supere el máximo establecido (entre usuarios con y sin recomendador). Se generaliza

la gestión de los algoritmos, definiendo los disponibles y sus características propias (uso de red y/o tipo de comunicación) en un fichero de configuración (en la carpeta de recursos del proyecto). De esta forma, en caso de implementar nuevos algoritmos, es suficiente con añadir su nombre y características al fichero para su correcta inclusión y gestión en la configuración y poder así utilizarlos en las simulaciones (incorporando además el caso con su tratamiento en la función de aplicación de recomendaciones).

Se incorporan además otras mejoras y nuevas funcionalidades al simulador. Una mejora que afecta directamente a la simulación es incorporación de herramientas de pausa y continuación de la simulación, lo que permite detener la simulación para observar el estado en diferentes instantes de tiempo. Se añade también una funcionalidad a la ventana de configuración que consiste en la especificación de semillas para la generación de números aleatorios. Esta funcionalidad hace posible la ejecución de simulaciones repetibles introduciendo un mismo valor de semilla. Las simulaciones que utilizan la misma semilla proporcionan los mismos resultados, haciendo posible la replicación de experimentos.

Por otra parte, en muchas ocasiones puede interesar más la recopilación de resultados que la visualización del transcurso de la simulación en la GUI, por lo que se incorpora la posibilidad de desactivar su visualización. Además se incorpora una nueva funcionalidad: la ejecución mediante lotes para evaluaciones desatendidas. Dicha funcionalidad permite llevar a cabo ejecuciones de tantos experimentos como se desee mediante lotes (especificando la configuración de cada uno en un fichero), sin interfaz gráfica, pudiendo así recopilar múltiples resultados experimentales de diferentes pruebas (con algoritmos o parámetros de configuración diversos) de forma automática, equivalente a una ejecución en segundo plano de todas las pruebas consecutivas.

4.3. Creación del grafo y trayectorias de los usuarios

Además de las mejoras ya expuestas incorporadas al simulador, el objetivo principal es la posibilidad de reconocer, representar y realizar simulaciones sobre cualquier escenario. La creación de estos escenarios es posible gracias a la nueva herramienta desarrollada en este proyecto, explicada en la Sección 5. El módulo de acceso a datos almacenados en ficheros (mapas) se trata de forma que su almacenamiento y lectura sea equivalente, facilitando el reconocimiento de los escenarios al simulador para su representación y creación del grafo.

Tras completar el proyecto y realizar pruebas en él utilizando nuevos escenarios creados con el editor de escenarios, se observó que el comportamiento de los usuarios durante la simulación no era el apropiado. Las trayectorias que estos tomaban violaba las condiciones del mundo real, atravesando paredes en sus movimientos. Por este motivo, se tuvo que reestructurar todo el proyecto para que el grafo creado en la simulación respetase las restricciones propias del mundo real.

El movimiento de los usuarios es el más óptimo posible (la línea recta) y este movimiento se realiza a través de las aristas del grafo. El grafo se construye de manera que, dentro de cada sala, se relacionan todos los objetos existentes (ítems, puertas y/o escaleras) siendo posible desplazarse directamente desde un objeto a otro dentro de ella. Además, los objetos conectables (puertas y escaleras) se conectan entre sí, permitiendo a los usuarios desplazarse por las diferentes salas y/o plantas del mapa. Los valores que adoptan las aristas corresponden al tiempo que toma recorrer la distancia (en metros) que separa a los distintos objetos (que varía en función de la velocidad de movimiento de los usuarios) y, cuando se trata de movimiento entre plantas, el tiempo necesario para subir o bajar escaleras, establecido en la configuración de la simulación.

El problema residía en que el grafo se construye conectando los diferentes elementos de las salas (ítems, puertas y escaleras) sin tener en cuenta las paredes, como ilustra la Figura 4.1. En el mapa del museo, todas las habitaciones son rectangulares, por lo que ninguna recta entre dos elementos contenidos en ellas puede atravesar una pared (arista del rectángulo). La creación de escenarios más complejos implica salas no rectangulares que pueden contener uno o más ángulos convexos (mayores a 180^{o}). Este inconveniente se da cuando existen en ellas nodos del grafo a ambos lados de un ángulo convexo, pues la arista del grafo que los une sí interfiere con las paredes.

Estas aristas conflictivas del grafo no se pueden eliminar, pues todos los elementos de una sala han de estar conectados entre sí o de lo contrario sería necesario visitar ciertos elementos del mapa para poder acceder al deseado. Se obtiene por tanto un problema de *Planificación de Caminos Mediante Grafos de Visibilidad* (Capítulo 3 de la tesis disponible en [23]). Sin embargo, dadas las condiciones del proyecto, se termina adoptando una solución alternativa a la idea explicada en la tesis citada.

Como se ha comentado, este problema no existía en el escenario original. Por este

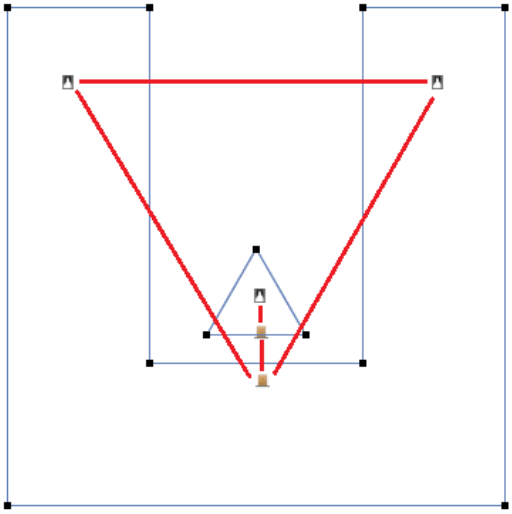


Figura 4.1: Representación del grafo inicial en el simulador, donde las aristas atraviesan paredes (escenario de prueba)

motivo, en la mayoría de casos se puede seguir operando de la misma manera y basta con tratar las situaciones donde existan ángulos convexos. La solución adoptada es la división de salas complejas en "subsalas". Estas división es tratada a efectos del editor de escenarios mediante la introducción de un nuevo tipo de objeto, el divisor de salas (explicado en la Sección 5). En el simulador, las divisiones se tratan de forma transparente para el usuario, es decir, se adapta el código del simulador para que en su lectura los escenarios se ilustren igual gráficamente pero se construya el grafo de elementos a nivel de "subsalas".

Los separadores actúan como "puertas invisibles" que conectan las dos "subsalas". En simulación, estas "puertas invisibles" se sitúan en el punto medio del separador (segmento). De este modo, si los usuarios quieren desplazarse de un objeto a otro que, a efectos del grafo, se sitúa en una "subsala" diferente, deberán hacerlo de igual modo que si se moviesen entre salas. Por tanto, los usuarios que se mueven entre "subsalas" lo hacen pasando por la "puerta invisible" situada en el punto medio de la línea que las separa (el separador de salas).

La representación del nuevo tratamiento del grafo de elementos y el comportamiento de los usuarios sobre los escenarios de prueba tras adoptar la solución de las "subsalas" se puede ver en la Figura 4.2.

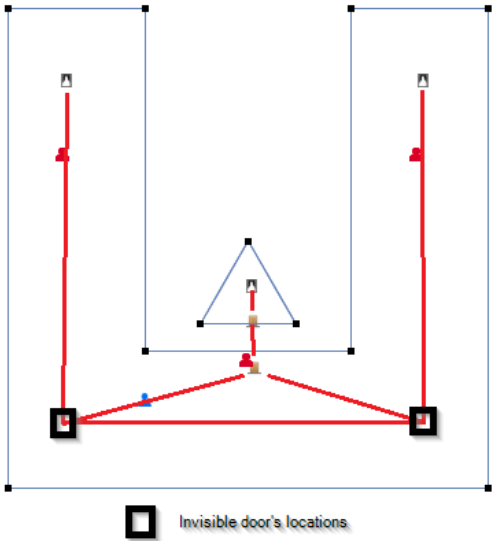


Figura 4.2: Representación del grafo correcto en el simulador (escenario de prueba)

Capítulo 5

Diseño e implementación del creador/editor de mapas

En este capítulo se describen las principales características generales de la nueva herramienta incorporada al simulador que permite la creación y/o edición de mapas legibles por el simulador (Sección 5.1), así como cada uno de los elementos que la componen con las funcionalidades que incorporan, siendo estos el menú (Sección 5.2), la barra de estado (Sección 5.3), las herramientas disponibles (Sección 5.4) y el panel del mapa que se está creando o editando (Sección 5.5).

5.1. Descripción general de la herramienta

Un aspecto fundamental de este proyecto es la creación de una herramienta que permite la **creación y/o edición de mapas o escenarios simulables**. Al ser este proyecto una continuación y ampliación de uno existente (simulador del museo MoMA), esta nueva herramienta se integra en el propio proyecto del simulador para disponer de todas las funcionalidades en un mismo programa. Por tanto, a pesar de poder considerarse como una herramienta independiente (aunque complementaria) al simulador, no se visualiza inmediatamente al ejecutar el proyecto, sino que se puede acceder a ella mediante: "*File*" → "*Create or Edit Map*".

El nuevo Editor de Mapas supone la incorporación de una nueva interfaz en la cual se encuentran los distintos elementos, tanto el mapa que está siendo modificado por el usuario como todas las herramientas de visualización y edición disponibles para hacerlo, bien sea directa o indirectamente (mediante dibujo o mediante vistas emergentes donde introducir los parámetros de la modificación). Esta interfaz se diseña buscando asimilarse a aplicaciones de dibujo utilizadas por una gran cantidad de usuarios como *Microsoft Paint* (`https://es.wikipedia.org/wiki/Microsoft_Paint`) o *CadStd* (`https://www.cadstd.com/`), siendo esta última una herramienta de dibujo técnico enfocada a la elaboración de planos. De este modo, su uso puede resultar más intuitivo para los usuarios.

Además, la interfaz es dinámica y por tanto da lugar a la generación de acciones a partir de ciertos elementos de la interfaz, las cuales pueden tener unas consecuencias tanto en los datos del escenario como en los elementos gráficos de la *GUI*. Estas acciones generan o transforman los datos del mapa (características generales, objetos y sus relaciones y tipos de ítems visitables creados por el usuario). Todos estos datos, almacenados con el formato adecuado, representan el escenario que posteriormente será leído e interpretado por el simulador.

Por tanto, dadas las características expuestas anteriormente, la construcción de los componentes del Editor de Mapas siguen la estructura del patrón arquitectural *Modelo-Vista-Controlador* o *MVC* [24]. Este patrón permite un desarrollo modular y escalable, por lo que es una gran ayuda tanto para el desarrollo de este trabajo como para posibles futuras extensiones del mismo. El desacoplamiento de los distintas módulos reduce responsabilidades a cada parte del código por lo que la incorporación de elementos y la depuración se hacen mucho más sencillas, además de proporcionar una arquitectura más limpia y clara.

Otro aspecto fundamental a tener en cuenta para la creación del editor es la definición de los diferentes objetos que puede contener un mapa. Se pueden distinguir dos grandes grupos de objetos: los básicos de construcción del escenario y los visitables por los usuarios del simulador. Los objetos de creación del escenario son: esquinas, las cuales forman las distintas salas; puertas, las cuales conectan unas salas con otras; y escaleras, las cuales conectan unas plantas con otras. Los objetos visitables o ítems, en cambio, pueden ser de cualquier tipo, estar representados por diferentes iconos y tener unas características concretas que pueden ser modificadas en cada instancia del ítem sobre el escenario para representar objetos concretos de cada tipo (analizadas en el Anexo E). El modelo de datos que representa los diferentes tipos de elementos que compondrán un mapa y las relaciones entre ellos se puede encontrar analizado en el Anexo D.3.

Una vez conocidas todas las características del editor, se implementan las diferentes funcionalidades respetando siempre tanto la arquitectura como las características que han de respetar los mapas. En la nueva interfaz se pueden observar las distintas herramientas disponibles y, por tanto, las posibilidades que ofrece. La GUI es adaptable a distintos tamaños de pantalla, al estar sus componentes dispuestos en un "layout" que define sus posiciones y dimensiones relativas, y compuesta por:

– Un menú, situado en la parte superior de la ventana.

– Una barra de estado, situada en la parte inferior.

– Un panel de herramientas, situado en la parte izquierda. Contiene las herramientas de dibujado y configuración del escenario y los objetos instanciables.

– Un panel en el que se encuentra el mapa creado y/o editado.

La interfaz de la nueva herramienta implementada con la correspondiente distribución de todos sus elementos principales se presenta en la Figura (5.1). En las siguientes secciones se explican en profundidad cada una de las partes, especificando los requisitos que comprenden e implementan.

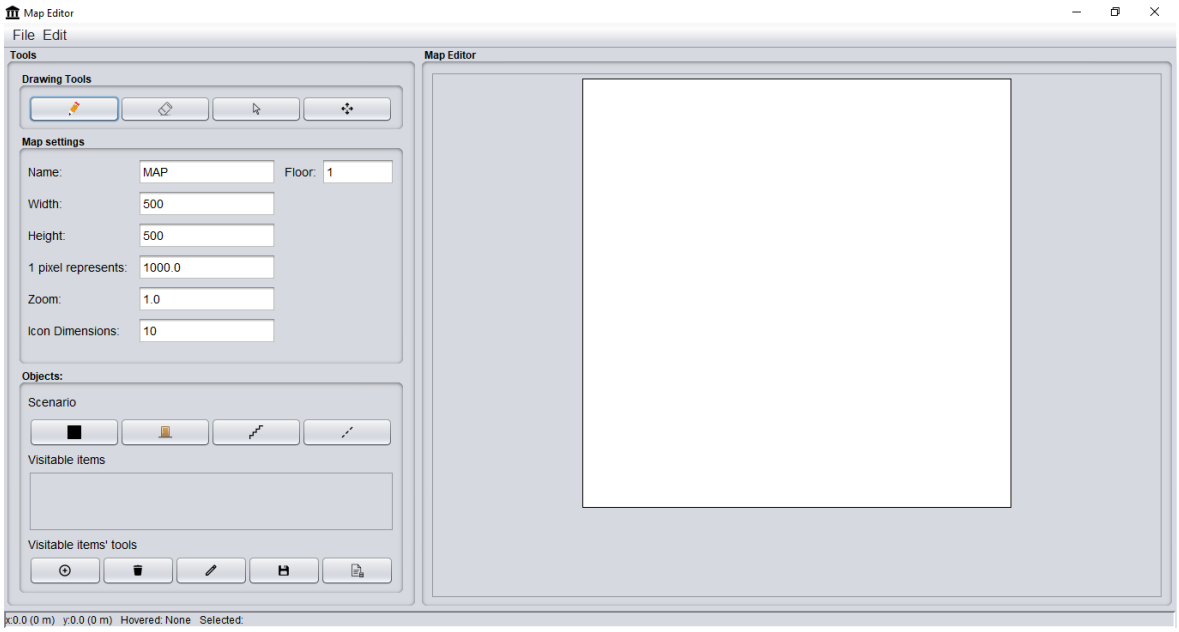


Figura 5.1: GUI del Editor de Mapas

5.2. Menú

En el menú se pueden encontrar dos tipos de acciones situados en los apartados "*File*" y "*Edit*". Las acciones recogidas en el apartado de archivo o "*File*" son las que permiten crear, guardar y cargar mapas. Mediante "*Edit*" se pueden ejecutar operaciones que afectan al mapa y sus componentes.

Al ingresar a la herramienta se genera un mapa con unos parámetros por defecto a modo de ejemplo para el usuario. También se puede crear un nuevo mapa desde la operación de creación de nuevo escenario ("File" → "New") la cual genera una ventana

que solicita al usuario las dimensiones y escala a aplicar en el nuevo mapa. Tras ello se genera un nuevo lienzo en blanco en el panel "*Map Editor*" con las características indicadas. Sobre este lienzo se colocarán los objetos que compondrán el escenario.

Las operaciones de guardado y carga de mapas ("Open" y "Save") despliegan un selector de archivos, el cual está configurado para seleccionar directorios. Esto se debe a que un escenario no está representado por un único archivo, sino que la información se separa en tres ficheros de texto que almacenan diferentes aspectos de este. El formato de estos archivos se basa en la clase "*Properties*" de *Java*. Los archivos que forman un mapa son:

- Archivo de ítems: contiene todos los ítems del escenario, cada uno con el valor de todas sus características correspondientes.

- Archivo de salas: almacena los datos referentes a la configuración del mapa y cada objeto básico de creación del escenario (no ítems) con sus características físicas (coordenadas y sala a la que pertenecen).

- Archivo de grafo: incluye todos los objetos del mapa con sus características no tangibles, utilizadas para la construcción del grafo. Estas características permiten establecer relaciones entre los objetos y son el identificador (*ID*) de cada uno, el número de objetos de cada tipo por cada sala y el número y composición de las conexiones de cada tipo (entre puertas, entre escaleras y entre ambas).

Se reutiliza y extiende la clase de acceso a datos de ficheros existente así como sus clases hijas (una por archivo), consiguiendo de esta forma operar de igual manera que en el simulador. Esto simplifica la lectura, escritura y traducción de mapas entre el simulador y editor. Las constantes también son comunes a ambas herramientas, por lo que el nombrado de archivos guardados con el editor es el esperado por el simulador, haciendo que la simulación de nuevos escenarios pueda ser inmediata (dependiendo de la disponibilidad de las bases de datos).

Las acciones de edición del mapa (apartado "*Edit*") incluyen modificaciones sobre el escenario que no se engloban en uno de los bloques del panel de herramientas, visible directamente en la interfaz. Esto se debe a que estas acciones realizan operaciones de mayor escala, que afectan a todo un escenario ya creado (o a grandes partes del mismo, como por ejemplo a una planta completa).

En primer lugar, se ofrece la opción de añadir un número determinado de ítems en posiciones aleatorias a lo largo de las salas del mapa. Esta funcionalidad es interesante sobre todo en escenarios en los que se dispone del mapa (distribución y dimensiones de las salas) y los tipos de ítems que puede contener pero no sus localizaciones exactas, como ocurrió en la construcción del proyecto previo (museo *MoMA*). Al seleccionar "*Edit*" → "*Add items in random positions*", se muestra una ventana emergente que pide al usuario el número de ítems a insertar y su tipo.

La distribución uniforme de ítems a lo largo de las salas, las cuales pueden ser polígonos con cualquier tipo de lados y vértices, resulta un problema complejo. Para ello, se implementa un algoritmo que permite obtener un punto aleatorio en el interior de un polígono irregular (Algoritmo 1), inspirado en [25]: se selecciona una sala entre las existentes, se triangula (descompone en triángulos) [26] el polígono (sala) para obtener la lista de triángulos que lo forman y simplificar así la obtención de un punto aleatorio dentro de un polígono, se obtienen las áreas de cada triángulo, se selecciona un triángulo aleatorio de los existentes dotando a cada triángulo de una probabilidad de selección igual al porcentaje de su área respecto al área total del polígono para asegurar distribuciones más equitativas a lo largo de este (áreas más grandes tendrán mayor probabilidad de contener más ítems) y se obtiene un punto dentro de dicho triángulo que equivaldrá a la localización del ítem.

$numItems \leftarrow numItemsTextArea$;
$i \leftarrow 0$;
while $i < numItems$ **do**
  $room \leftarrow randomRoom()$;
  $triangles \leftarrow triangulatePolygon(room)$;
  $weightedAreas \leftarrow weightAreas(triangles)$;
  $selectedTriangle \leftarrow$
   $getTriangleUsingWeighedtAreas(triangles, weightedAreas)$;
  $location \leftarrow getRandomPointInTriangle(selectedTriangle)$;
  $i \leftarrow i + 1$;
end

Algoritmo 1: Ítems en posiciones aleatorias

Existen varios algoritmos de triangulación de polígonos [26]. Para esta función se emplea el método *Ear clipping*, haciendo uso de la librería externa *earcut4j* (`https://github.com/earcut4j/earcut4j`).

Otra funcionalidad implementada es la combinación de plantas. Así se pueden

tener escenarios con diversas plantas, como ocurría en el museo, permitiendo simular escenarios más complejos. Se accede mediante "*Edit*" → "*Combine Floors*", lo cual muestra una ventana que permite especificar la ruta de los dos archivos a combinar (con una o varias plantas previamente combinadas) y en qué disposición hacerlo (debajo o al lado). Tras la combinación, estas se deben unir incluyendo escaleras para que los usuarios puedan cambiar de planta. Las escaleras deben estar conectadas a puertas para especificar el punto de acceso a ellas.

Para mayor versatilidad, se implementa la función "*Reverse coordinates*", la cual permite girar todas las coordenadas de todos los elementos del mapa. De este modo, si el diseño de dos plantas se ha realizado en disposiciones diferentes, es posible rotar una de ellas para poder combinarlas haciendo uso de la funcionalidad anterior.

Por último, para facilitar la generación de datos de la simulación, se permite la exportación de los datos de los ítems de los escenarios creados a formato *CSV*. Accediendo a "*Edit*" → "*Export items to CSV*", se selecciona la ruta del fichero al que exportar esta información. Se facilita la información en este formato ya que es el utilizado por *DataGenCARS*, principal herramienta considerada en la generación de datos.

5.3. Barra de estado

Esta barra provee información al usuario que está creando y/o editando el mapa. Su información se actualiza cuando el usuario se encuentra sobre el escenario (con el cursor) e informa tanto de su movimiento como de sus acciones. Dado que los mapas a crear pueden ser escenarios reales, su contenido también ha de ser traducible a la realidad. Por esto, se informará de las coordenadas (x e y) sobre las que se encuentra el cursor en píxeles, pero también en una unidad de medida real, en este caso la utilizada por el Sistema Internacional de Unidades: los metros. La relación entre ambas unidades la determina para cada escenario el campo de texto de la escala, encontrado en los ajustes del mapa ("Map Settings").

Al mismo tiempo, mostrará el objeto sobre el que se encuentra el ratón en dichas coordenadas (en caso de localizarse sobre alguno de los objetos del escenario) tras el campo "Hovered". Además, en caso de haber seleccionado alguno de los objetos con la herramienta de selección, se mostrará el último objeto seleccionado tras el campo "Selected".

5.4. Panel de Herramientas

Como se puede observar en la interfaz, se pueden distinguir tres tipos de bloques de herramientas dentro del Panel de Herramientas:

- Herramientas de dibujo ("Drawing Tools"): herramientas principales de manipulación del escenario.
- Ajustes del mapa ("Map Settings"): permiten modificar los parámetros generales del mapa.
- Objetos ("Objects"): compuesto por todos los tipos de objetos que se pueden incluir sobre el mapa.

La combinación de estos tres bloques proporciona al usuario todas las herramientas necesarias para definir y dibujar mapas sobre el lienzo.

5.4.1. Herramientas de construcción y manipulación del escenario

En el panel "Drawing Tools" se sitúan las herramientas principales que permiten modificar el mapa: herramienta de dibujo o lápiz, para "pintar" (colocar instancias de objetos) sobre el mapa; herramienta de borrado o goma, para eliminar objetos del mapa; herramienta de selección o cursor, la cual permite seleccionar objetos para ver sus atributos y modificarlos si se desea; y herramienta de movimiento, la cual permite desplazar los objetos a lo largo del escenario.

La herramienta de dibujo permite colocar instancias de objetos sobre el escenario. Para hacerlo, se pulsa el botón del lápiz y el del objeto que se desea colocar, y se dibuja mediante "click" izquierdo del ratón sobre el panel del mapa, creando un nuevo objeto en la posición sobre la que se ha pulsado. Los objetos a colocar deberán respetar las características propias del mundo real, motivo por el cual no cualquier acción podrá llevarse a cabo. Las restricciones de cada objeto se exponen a continuación.

En caso de seleccionar la esquina como objeto a pintar, se deben pintar consecutivamente al menos tres de ellas, pues este tipo de objetos no tiene sentido por sí solo, sino que su propósito es formar salas que abarquen un área en la que colocar objetos y por la cual podrán moverse los usuarios en la simulación. A partir de la segunda esquina se muestran sobre el mapa las "paredes" (líneas) que las unen. Además, las salas no podrán intersecar ni con las paredes de otra sala ni con las suyas

propias, siendo este un suceso imposible en la vida real. Por tanto, si se trata de pintar incumpliendo dicha restricción, se avisa por pantalla y se descarta esta nueva esquina. Para completar su creación, mediante "click" derecho se crea una pared entre la última y la primera esquina siempre que no se incumplan las condiciones anteriores (de lo contrario, no se completa la creación y se pueden seguir encadenando esquinas).

Los objetos visitables deben ser accesibles por los usuarios. Por tanto, su pintado incorpora una restricción que impide su colocación fuera de una sala. Las puertas y escaleras sí podrán posicionarse fuera, aunque debe especificarse en tal caso a cuál pertenecen al no poder asignarse automáticamente (este suceso se permite ya que este factor ocurre en el escenario heredado del museo). Además, los objetos conectables deben conectarse al menos a otro de ellos para que el escenario sea correcto. Las escaleras se unen por pares exclusivos, mientras que las puertas pueden hacerlo a diversos elementos conectables (puertas y escaleras). Aquellos que incumplen estas condiciones muestran un borde rojo a su alrededor en señal de error.

La herramienta de borrado elimina objetos del mapa. Para ello, se selecciona el botón de la goma y se pulsa sobre un objeto del lienzo. El borrado de salas se realiza cuando se elimina una esquina de la misma, procediendo así a suprimir todos los objetos contenidos en ella. El borrado de un elemento conectable conlleva su desconexión de todos los elementos a los que se encontraba conectado.

El cursor permite seleccionar objetos y acceder a todos sus atributos para verlos y/o modificarlos (mediante doble "click"). Esta acción despliega la ventana de edición de atributos, la cual extenderá su contenido en función del tipo de objeto seleccionado. Esta ventana siempre muestra los atributos básicos de todo elemento, siendo estos el tipo de objeto, la habitación a la que pertenece, su identificador y su posición (tanto en píxeles como en metros). Si se trata de un objeto conectable, se añade además un panel que muestra tanto las conexiones activas como las posibles, permitiendo la conexión y desconexión con otros objetos conectables. Además, si el objeto es una puerta, se permite la modificación de su sala dada su capacidad de posicionamiento fuera de la misma (no asignada en su creación al no encontrarse dentro de una sala). Si se accede a los atributos de un objeto visitable, se añade un panel con todos los atributos que estos contienen, pudiendo así dotarles de valor o modificar los ya asignados. El estudio de los posibles atributos y la elección de aquellos considerados comunes a todos los posibles objetos visitables creados por usuarios se explica en el Anexo E.

Dado el objetivo de la reutilización de código y generalización de la herramienta, todos los paneles son independientes y por tanto reutilizables en otras vistas con propósitos distintos. Un ejemplo de esto es el panel de atributos de los objetos visitables, utilizado tanto en la modificación de atributos como en la creación de nuevos tipos de objetos visitables (botón con el que poder instanciarlos, explicado en la Sección 5.4.3). De esta manera, si cambian los requisitos, su modificación se realizará en su panel concreto pero se refleja en toda la herramienta.

Por último, la herramienta de movimiento permite modificar la posición de los objetos dibujados sobre el lienzo. Esta acción es similar a cualquier herramienta de dibujo, es decir, mediante "drag and drop" (arrastrar y soltar). No todos los movimientos a realizar por el usuario serán válidos, por lo también asegura el cumplimiento de las restricciones que hacen respetar las características del mundo real (equivalente a la herramienta de dibujo). Lo mismo ocurre con el movimiento directo mediante la modificación de los parámetros de posición hallados en la ventana de edición de atributos del objeto.

5.4.2. Ajustes del mapa

Los ajustes del mapa o "Map Settings" son los valores que permiten modificar las características generales del mapa. Estos valores influyen tanto en la edición del mapa como en su simulación, pues se utilizan para su construcción y representación.

Los atributos del nombre y la planta (o plantas) tienen un carácter meramente informativo. La altura y la anchura del escenario determinan las dimensiones del mapa y su modificación afecta directamente a este (tanto el panel sobre el que colocar objetos en el editor como posteriormente el mapa en simulación). Estos campos están altamente ligados a la escala, la cual determina la distancia y dimensiones reales de los diferentes elementos. Este parámetro se define en el campo "1 pixel represents _ m" y el valor introducido se aplica a todas las coordenadas para establecer la equivalencia en metros que representa cada píxel del mapa. Por tanto, estos tres valores han de tenerse en cuenta entre sí a la hora de establecerse, pues las mismas dimensiones del mapa representarán diferentes dimensiones en la vida real según la escala y viceversa.

Por último, se puede modificar el "zoom" con el que se visualiza el mapa para una mayor versatilidad en función del mapa, sus dimensiones y su escala. También se puede

modificar la dimensión de representación de los objetos para mejorar la visualización de los mismos según desee el usuario dadas las características de su escenario.

5.4.3. Objetos

En este panel se encuentran todos los objetos disponibles y que podrán colocarse sobre el mapa para construir el escenario real deseado. Se pueden distinguir dos tipos: objetos básicos de creación del escenario (localizados en el apartado "Scenario") y objetos visitables por los usuarios en simulación ("Visitable items").

Los objetos básicos que permiten crear un mapa son: esquinas, puertas y escaleras. Estos son comunes a todos los escenarios creados con esta herramienta ya que son necesarios para el reconocimiento y representación del mapa por el simulador. Las esquinas forman salas (del modo explicado en la Sección 5.4.1), siendo el área de cada una el espacio donde se pueden situar los objetos visitables (y otros objetos básicos) y en la que por tanto se generará la parte del grafo que contiene y relaciona todos los elementos de la misma sala. Las puertas permiten el movimiento entre salas al conectarse a otras puertas o escaleras (en diferentes plantas en este caso). Las escaleras se sitúan en plantas diferentes y se conectan entre sí, permitiendo a los usuarios moverse entre ellas.

Los objetos visitables instanciados en el mapa representan los ítems que pueden ser recomendados por los CARS durante la simulación. Estos objetos pueden variar y tener tipos y/o características diferentes en función del escenario a crear. Por este motivo, se ofrecen herramientas que permiten: crear tipos de objetos visitables, eliminarlos, modificar los atributos generales a su tipo (que se mantendrán constantes para todas sus instancias) y guardar y cargar conjuntos enteros de tipos de ítems en ficheros, pudiendo así reutilizarse en la creación y/o edición de distintos mapas.

Las operaciones propias de los ítems visitables crearán, modificarán o eliminarán los botones encontrados en el apartado "Visitable items" de la interfaz, representando estos los tipos de objetos visitables disponibles a dibujar sobre el escenario. Su uso es idéntico al de los botones encontrados en "Scenario", es decir, tanto las acciones como las restricciones propias de los objetos visitables se aplican en función de la herramienta que se está utilizando (Sección 5.4.1).

Debido al propósito general tanto del editor como del simulador, se modifican también los atributos que pueden contener, manteniendo únicamente aquellos comunes

y representativos para cualquier tipo de ítem. Esto elimina atributos específicos de cada tipo de ítem (como los existentes para las pinturas y esculturas del escenario del museo) pero a cambio permite representar cualquier tipo de objeto sin proporcionarle cualidades irrelevantes o incorrectas dada su naturaleza. En el Anexo E se puede encontrar un estudio de los atributos de posibles tipos de objetos para determinar finalmente los atributos comunes a todos ellos. Además, la representación de objetos más específicos es posible mediante una nueva clase que extienda la clase "Item" (Anexo D.3).

5.4.4. Tratamiento de trayectorias en el editor

Tras el problema encontrado al finalizar la primera versión de este proyecto, introducido en la Sección 4.3, se ha de gestionar el problema de las trayectorias de los usuarios tanto en el editor como en el simulador. La solución abordada consiste en la creación de un nuevo tipo de objeto, el "divisor de salas", que permite descomponer aquellas habitaciones complejas en las que los usuarios puedan atravesar paredes en "subsalas" con formas poligonales simples como las del escenario original (museo) para que no dispongan de ángulos convexos. Se incorpora por tanto este nuevo tipo de objeto básico de construcción del escenario en el panel "Scenario".

El pintado de este tipo de objeto requiere la existencia de una sala y se realiza mediante la selección de dos esquinas no consecutivas de la misma y cuyo segmento de división esté completamente contenido en el interior de la sala, es decir, siempre que la línea divisoria de la sala no se salga de la misma. Al seleccionar la segunda esquina que cumpla estos requisitos, se completa la creación del divisor de salas, resaltando en la interfaz las subsalas en las que ha sido dividida y su correspondiente divisor (línea discontinua entre las esquinas). La Figura 5.2 muestra un ejemplo de representación de un escenario sencillo afectado por la división en subsalas.

La operación de borrado se realiza al pulsar cualquier punto contenido en el segmento divisor, dotado de un pequeño grosor para mejorar la visibilidad del mismo así como facilitar esta tarea. Las operaciones de selección y movimiento no aplican a este tipo de objeto, por lo que estos casos no se tratan en el controlador.

La operación de guardado de este nuevo tipo de objeto no supone un problema para su carga de nuevo en el editor, leyendo las esquinas que involucran y aplicando las divisiones correspondientes. En cuanto a la lectura por parte del simulador, su

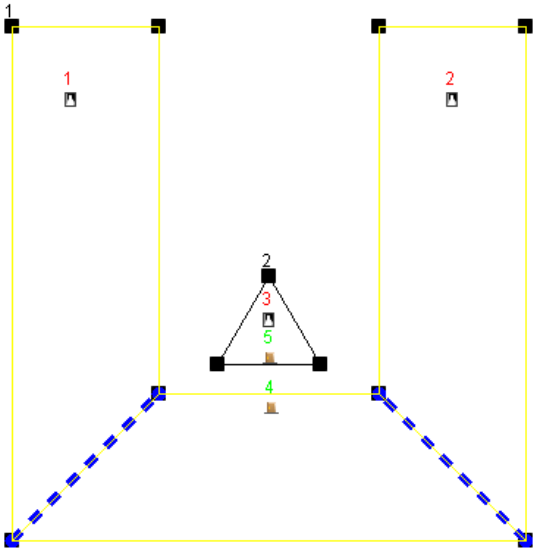


Figura 5.2: Representación de la división en subsalas - Editor

adaptación fue más compleja. Inicialmente se trataron directamente las subsalas como nuevas salas, generando problemas en la construcción del grafo visual y las conexiones de puertas y escaleras. Finalmente se almacenan tanto salas como subsalas con sus correspondientes elementos, duplicando parte de la información pero conviviendo para poder construir correctamente tanto el grafo visual como el utilizado para el movimiento de los usuarios durante la simulación.

5.5. Panel del mapa editado

Este panel ocupa la mayor parte de la interfaz, pues contiene en su interior el lienzo sobre el que aplicar las herramientas con los diferentes objetos cuando un mapa es creado o cargado. Se representa de esta forma por dos motivos: la interpretación visual del mapa y su modificación en tiempo real permite al usuario saber en todo momento qué aspecto tendrá el escenario en simulación, pues su representación y traducción son idénticas; y la facilitación al usuario de la creación de mapas, pudiendo estos ser creados por cualquier persona sea o no experta en informática y/o CARS, al asemejarse esta herramienta a otras aplicaciones de dibujo ampliamente utilizadas (al ser una herramienta básica, muchos ordenadores incorporan programas de dibujo, como por ejemplo *Microsoft Paint*, incluido en todos los ordenadores con Sistema Operativo *Microsoft Windows*).

Capítulo 6

Experimentos y resultados

En este apartado se explican los experimentos diseñados para demostrar toda la funcionalidad de la generalización del simulador, así como de la nueva herramienta: el *Editor de Mapas*.

Los resultados, por tanto, son de varios tipos: la creación de mapas sobre los cuales se podrán llevar a cabo simulaciones (Sección 6.1), la generación de datos referentes a los escenarios creados para poder aplicar recomendaciones (Sección 6.2), las estadísticas obtenidas a partir de las simulaciones en los distintos experimentos (Sección (6.3) y las estadísticas de rendimiento del simulador comparándolo con la versión original (Sección 6.4).

6.1. Creación de nuevos mapas. Experimento: Mapa de GranCasa

Una de las finalidades principales de este proyecto es la posibilidad de realizar simulaciones para evaluar CARS en cualquier tipo de mapa. Para ello, como se explica en el Capítulo 5, se implementa una nueva herramienta "user-friendly" que permite crear mapas reconocibles por el simulador mediante una interfaz gráfica, de forma que pueda ser utilizada por cualquier tipo de usuario (no experto). De esta forma se pueden generar los ficheros correspondientes al mapa creado sin importar su tamaño y/o complejidad sin tener que editar manualmente los mismos, lo cual supondría una tarea exhaustiva e insostenible para los usuarios.

El mapa seleccionado para llevar a cabo un experimento que demuestre este hecho es el del centro comercial *GranCasa* (Zaragoza, España). Este consta de una extensión y un número de salas mucho mayor que el escenario para el que fue diseñado el simulador original, las salas (tiendas y restaurantes) en conjunto contienen un número mayor de ítems visitables, las salas tienen diferentes formas poligonales y no únicamente rectangulares y el establecimiento consta de tres plantas conectadas entre sí por diversas escaleras en diferentes puntos de cada sala. En definitiva, este

escenario presenta nuevos retos no contemplados en el escenario original, permitiendo así demostrar la generalidad y versatilidad tanto del editor como del simulador de escenarios.

Una vez seleccionado el mapa, se ha de representar en el editor. Para una representación lo más precisa y fiel a la realidad posible, se ha de disponer de un de mapa que contenga las posiciones y medidas de cada elemento (salas, puertas, escaleras e ítems visitables). En el caso de *GranCasa* se dispone de la representación de las plantas (`https://www.coaaragon.es/Default.aspx?Cod=202` y `https://www.grancasa.es/mapa/#/`) pero no de las medidas concretas (únicamente las dimensiones globales, de 400x90 metros), lo que supuso un trabajo manual previo mucho más costoso en tiempo que su propia creación en la herramienta. En futuros escenarios en los que se disponga de las medidas y escala correspondientes, se simplificará notablemente su creación.

Respecto a los ítems visitables (objetos sobre los que ofrecer recomendaciones) es importante definir cuántos se colocarán, de qué tipo serán y las características de cada uno de ellos. En el caso del centro comercial *GranCasa*, estos ítems hacen referencia a tiendas o secciones de una tienda (o restaurante), extraídas del sitio web oficial. De esta manera, en cada ítem visitable colocado en el mapa se ofrece un tipo concreto de producto. Los detalles de construcción del escenario así como los tipos y selección de sus ítems se amplía en el Anexo F.1.

6.2. Generación de datos

El último paso para poder probar *CARS* una vez generalizado el simulador y creados los mapas con el editor es obtener los datos referentes al nuevo escenario creado. Para ello se ha de generar la base de datos "oráculo" que contiene todos los datos que pueden ser utilizados en simulación (usuarios, ítems, contextos y valoraciones para cada combinación de las tres anteriores). De ella se extraen los valores que los usuarios otorgan en sus valoraciones en función de su perfil de usuario predefinido.

Para la generación de los datos necesarios se utiliza *AUTO-DataGenCARS* [5], generador de datos sintéticos cuyo propósito es la evaluación de *CARS*. Su uso, sin embargo, no es trivial. Se encontraron problemas tratando de generar datos a partir de otros existentes (ítems del mapa). Además de esto, las valoraciones obtenidas no eran

las esperadas para el caso de uso necesario en este proyecto, pues no se generaba una única valoración por usuario-ítem-contexto (unas tuplas repetidas y otras ausentes). Se intentó solventar este problema mediante un "script" que permitía generar todas las combinaciones necesarias pero los resultados en simulación con estos datos eran ilógicos.

Finalmente, con la ayuda de expertos en la herramienta, se consiguió generar los datos necesarios para el mapa de *Grancasa* y unos resultados coherentes a partir de estos. Por ello, a pesar de las dificultades encontradas, se considera *DataGenCARS* como la herramienta recomendada para este propósito. Este proceso se amplía en el Anexo G.

6.3. Resultados de simulación y recopilación de estadísticas

Para analizar los resultados de la simulación, se toma como referencia la evaluación experimental del artículo [1]. De esta forma, se corroboran los resultados tras las modificaciones aplicadas al simulador obteniendo unos resultados equivalentes (analizados en el Anexo H). Comprobado el correcto funcionamiento de los algoritmos de recomendación así como el funcionamiento general del simulador con el escenario del museo, se puede proceder a realizar experimentos con otros escenarios, en este caso el centro comercial *GranCasa*. La configuración utilizada para ellos se muestra en la Tabla 6.1.

Parámetro	**Valor**
Número de visitantes & tiempo de visita	176 (inspirado por [1]) & 1 hora
Número de ítems & velocidad media de los usuarios & tiempo de observación de los ítems	283 & 5 Km/h & 30 segundos
Trayectorias seguidas por usuarios sin recomendador	NPOI (Punto de interés más cercano)
Tiempo en cambiar de planta	60 segundos
Número de ítems recomendados guardados en la lista de resultados (K)	10 ítems
Umbral de similitud para el algoritmo UBCF	0.5 (Correlación de Pearson)
Umbral de recomendación (1-5)	2.5
Umbral de incremento de la base de conocimiento	40 nuevos votos
Intervalo mínimo de tiempo de actualización de recomendaciones	30 segundos

TTL (Time To Leave) de los datos propagados	3 minutos (experimento con TTL no infinito)
Latencia en la comunicación	1 segundo
Banda ancha de comunicación	54 Mbps (IEEE 802.11g)
Rango de comunicación	250 metros
Tamaño máximo de la base de conocimiento	1 Mb
Tiempo para cambiar el estado de ánimo del usuario	1800 segundos (30 minutos)

Tabla 6.1: Configuración de los experimentos con el mapa de *GranCasa*

Se realizan experimentos en el nuevo escenario con cada uno de los algoritmos de recomendación, de forma que se pueden extraer las gráficas equivalentes al artículo original para el nuevo escenario. De esta forma se pueden establecer comparaciones que permitan evaluar la calidad de los algoritmos y el desempeño del simulador en nuevos escenarios.

Como se observa en la Figura 6.1, las valoraciones obtenidas a nivel general son inferiores a los resultados obtenidos de los experimentos del museo ([1, 3]). Es importante destacar la gran influencia de las dimensiones del escenario, la cual provoca que se reduzcan ligeramente las valoraciones, pues a pesar de recomendar los ítems más apropiados a los usuarios, entra en juego el contexto, en concreto la distancia y por tanto tiempo que se tarda en alcanzar el ítem. Además de esto, las distancias también se incrementan dado que existen más salas y estas contienen en general un número menor de ítems (algunas tiendas contienen una única sección), por lo que en muchas ocasiones la trayectoria será más compleja que una única línea recta dentro de la misma sala. Estos factores conllevan unas valoraciones más bajas, incluso en los algoritmos considerados como ideales.

La Figura 6.2 representa la satisfacción general de los usuarios medida en "likes" (valoraciones mayores al umbral establecido) y "dislikes" (inferiores al umbral). El umbral establecido tiene valor 3,5. Se observan muchos más "dislikes" respecto al mapa del museo ya que muchas de las valoraciones toman valor 3,0. A pesar de esto, se observa el notable impacto de la aplicación de algoritmos de recomendación no simples, pues la diferencia de valores comienza a ser positiva a partir del uso de estos y se incrementa notablemente conforme se aplican algoritmos mejores. Se deben analizar los resultados de las gráficas de forma conjunta, pues de lo contrario se pueden

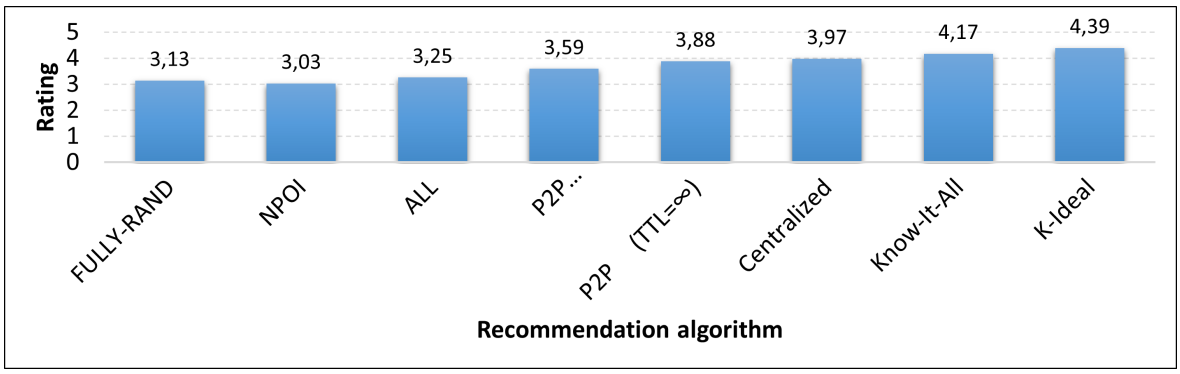


Figura 6.1: Valoración media obtenida para los ítems observados

sacar conclusiones erróneas. Por ejemplo, se puede interpretar que dada esta gráfica el algoritmo *K-Ideal* ofrece peores resultados que otros, pero esto se debe a que el número de ítems recomendados es menor y sin embargo la mayoría de estos obtienen la valoración máxima y por tanto una valoración media mucho mejor.

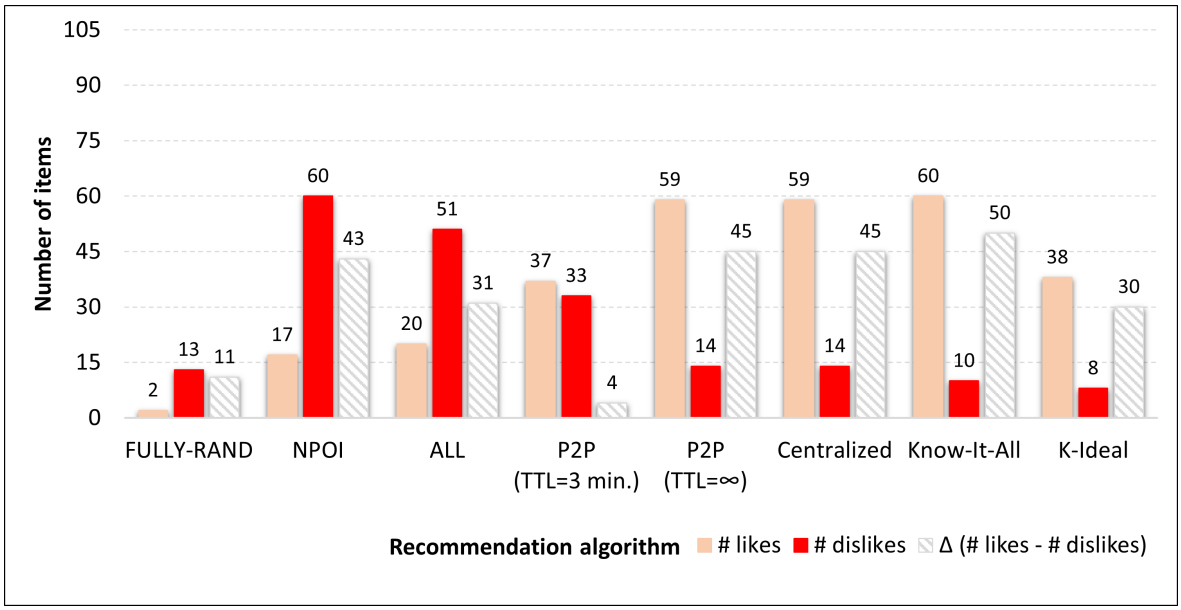


Figura 6.2: Me gusta y no me gusta obtenidos (con umbral 3,5)

Otro aspecto a tener en cuenta dada la mayor dimensión del escenario y la distancia de muchos de los ítems que afecta a los algoritmos de recomendación basados en los usuarios es el rango de comunicación. Al mantener el mismo número de usuarios que en el escenario del museo en un escenario de superficie mucho mayor, entran menos usuarios en dicho rango, pues estos pueden estar más distanciados. Además, cuando se aplican estos algoritmos, es importante que el tiempo real por iteración no sea muy elevado, pues los intercambios en las comunicaciones se ven altamente afectados al reducirse el número de ciclos de ejecución. La combinación de estos dos factores puede traducirse en la obtención de unos resultados peores de los reales al aplicar estos

algoritmos, por lo que es importante tenerlos en cuenta al plantear los experimentos.

El desempeño del algoritmo P2P, último trabajo sobre el proyecto antes del presente, se amplía en la Figura 6.3. Se observa una baja variación de la predicción de error media a lo largo del tiempo (*MAE*) y con valor menor que el obtenido previamente en el escenario del *MoMA*. Esta gráfica presenta irregularidades ya que no todos los ítems visitados son a causa de este algoritmo, sino que ha de pasar un tiempo hasta recoger valoraciones del resto de usuarios (problema de arranque) y, en algunos puntos de la simulación no se dispone de información sobre nuevos ítems en la cola del usuario para aplicar recomendaciones, visitando entonces los más cercanos.

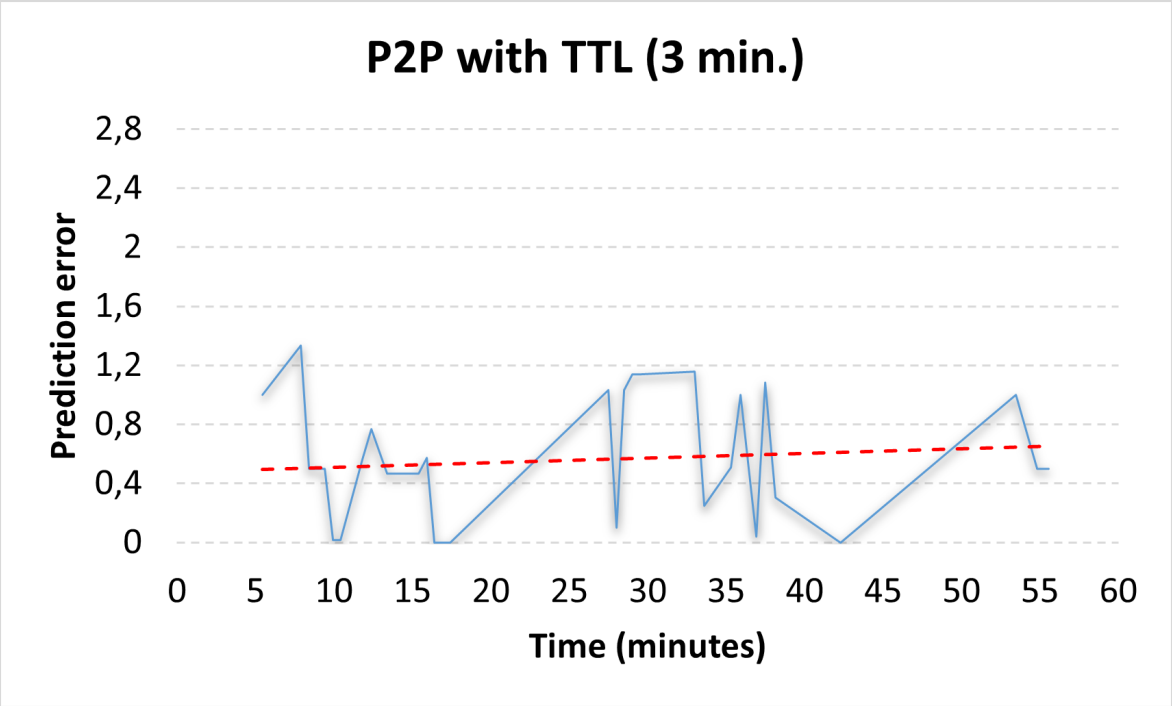


Figura 6.3: Predicción de error a lo largo del tiempo

Tampoco se puede pasar por alto la gran variabilidad de resultados de los algoritmos simples, sin recomendador (*Random*, *ALL* y *NPOI*). Estos algoritmos presentan una variabilidad en función de la semilla (valor para generar la pseudoaleatoriedad), pues esta definirá el punto de partida de los usuarios y, en el caso del algoritmo aleatorio, también su camino, siendo este el algoritmo que más variabilidad presenta (en casos excepcionales, las valoraciones se pueden aproximar al 4.0 o no llegar al 3.0). Estos algoritmos sirven como referencia para comprobar que la aplicación de recomendaciones efectivamente es positiva y ofrece a los usuarios mejores resultados que si no dispusieran de ningún tipo de recomendación, pues sus resultados se agrupan

en la mayoría de experimentos en torno al 3.0, pero es importante tener en cuenta el papel del azar en las variables aleatorias de cada simulación.

Además, se añaden dos nuevas métricas (almacenadas en ficheros *CSV*) que aportan información extra sobre la simulación. Se almacena el número de usuario que están observando un ítem a la vez que los usuarios con recomendador, con su instante de tiempo y la valoración de estos últimos. También se recogen todas las ocurrencias en las que un usuario se encuentra a una distancia menor que una distancia umbral de un usuario con recomendador y el intevalo de tiempo, el cual debe ser mayor o igual a un tiempo umbral especificado.

Por último, es importante destacar las características y disposición de los datos generados para el nuevo escenario (*GranCasa*) dadas las dificultades comentadas en el apartado anterior (Sección 6.2). La distribución de los nuevos datos es mucho más acentuada, obteniendo la mayoría de registros en la base de datos "oráculo" (con valoraciones para todos los usuarios, ítems y contextos posibles) valoraciones con valor de 3.0 o 4.0, y encontrando en muchos casos usuarios con muchos ítems en los que no pueden obtener valoración de 5.0 en ningún caso (dado ningún contexto). Esto permite comprobar que, para usuarios más críticos, las recomendaciones continúan ofreciendo resultados mucho más positivos que sin su uso, pero también provoca que los resultados globales (gráficas anteriores) disminuyan su valor.

En la Tabla 6.2 se amplía esta información, mostrando los aspectos más destacables sobre la disposición y morfología de los datos y su comparativa con los datos del museo, los cuales pueden tener un impacto en los resultados.

Dato(s) a comparar	**BD MoMA**	**BD GranCasa**
Total de valoraciones en la base de datos (número de entradas)	380.160	448.272
Número de entradas con valoración: 3.0 (y %)	132.529 (34.86 %)	180.349 (40.23 %)
Número de entradas con valoración: 3.0 o 4.0 (y %)	217.740 (57.28 %)	364.358 (81.28 %)
Total de valoraciones del usuario con recomendador (176)	2160	2547
Total de entradas del usuario con recomendador que toman valor 5.0 (y %)	336 (15.56 %)	120 (4.71 %)

Tabla 6.2: Comparativa de la disposición de datos de *GranCasa* respecto a la del *MoMA*

6.4. Evaluación de rendimiento y comparativa respecto al proyecto original

A partir de los problemas originales de rendimiento del proyecto y las optimizaciones llevadas a cabo, es relevante establecer una comparativa de tiempos para evaluar la mejoría. En el nuevo simulador se consigue respetar la relación entre tiempo de simulación y real especificada en la configuración (Anexo C). Sin embargo, en escenarios más grandes y complejos (*Grancasa*) existe un pequeño tiempo de arranque previo al comienzo de la simulación (conexión a fuentes de datos y lectura y representación del escenario). Además, si se emplea la red *P2P*, este tiempo se incrementa dado que el número de conexiones a gestionar es mucho mayor (dos bases de datos por usuario).

Se implementan también las evaluaciones desatendidas por lotes. Estas permiten encadenar ejecuciones (sin interfaz) para probar múltiples escenarios y configuraciones en conjunto. A pesar de las mejoras de control de operaciones I/O (entrada y salida), se notan descensos de prestaciones cuantas más simulaciones encadenadas (paso del tiempo). Conocidos los tiempos de cada parte, en la Tabla 6.3 se comparan distintos casos de ejecución para evaluar los costes y los detalles se amplían en el Anexo C.

Simulación(es)	**Mapa MoMA**	**Mapa GranCasa**
Simulador original, para cualquier configuración	+24 horas	-
Simulación única sin usar *P2P* a velocidad x30	2 minutos	3.5 minutos
Simulación única usando *P2P* a velocidad x30	4 minutos	6 minutos
8 simulaciones desatendidas (todos los algoritmos) a velocidad x30	40 minutos	1 hora
8 simulaciones desatendidas, algoritmos *UBCF* (centralizado y *P2P* y *Know-It-All* a velocidad 1	4:30 horas	5:30 horas

Tabla 6.3: Comparativa de tiempos de *GranCasa* respecto al museo *MoMA* (diferencias de tiempo de arranque)

Capítulo 7

Conclusiones y trabajo futuro

Este capítulo presenta el trabajo realizado y las conclusiones personales (Sección 7.1), las conclusiones del proyecto (Sección 7.2) y el trabajo futuro (Sección 7.3).

7.1. Trabajo realizado y conclusión personal

Como conclusión personal, creo este ha sido un proyecto complejo, que se ha querido llevar a cabo con mucho trabajo y detalle, pero esto también ha supuesto una gran cantidad de esfuerzo y horas invertidas (Tabla 7.1). Se ha realizado una adaptación constante tanto a las características y restricciones del proyecto inicial como a los nuevos problemas que se han ido encontrando a lo largo del desarrollo.

Además, la curva de aprendizaje fue elevada desde un comienzo ya que el rendimiento del simulador inicial era muy bajo, no sólo haciendo imposible obtener resultados sino también dificultando notablemente la depuración. Finalmente, cuando se pensaba que se había completado el proyecto y estaba casi listo para depósito, se descubrió el problema con las trayectorias (explicado en las Secciones 4 y 5), teniendo que reestructurar de nuevo tanto editor como simulador para eliminar los ángulos convexos en el editor y su correcta lectura por el simulador en la construcción de un nuevo grafo del escenario más complejo.

A pesar de estos aspectos, considero muy positiva la labor realizada a lo largo de todo este tiempo, la cual ha conllevado un amplio aprendizaje tanto profesional y educativo como a nivel personal. Entre otros, he adquirido y/o ampliado conocimientos en diversas tecnlogías, RS y CARS, algoritmos de recomendación, manejo y gestión de grafos, patrones arquitecturales y de diseño, gestión de un proyecto de ingeniería, búsqueda de información y toma de decisiones para la resolución de problemas. La realización de este TFG en el contexto del proyecto de investigación NEAT-AMBIENCE ha resultado ser una experiencia satisfactoria y enriquecedora, que me ha permitido una primera toma de contacto con el ámbito de la investigación en un entorno académico.

Por último, considero que el prototipo desarrollado puede ser útil para la investigación en sistemas de recomendación, por parte de los investigadores con los que he colaborado, en particular en el contexto del proyecto NEAT-AMBIENCE (PID2020-113037RB-I00 / AEI / 10.13039/501100011033) en el que se enmarca este trabajo, e incluso en el futuro por parte de otros grupos de investigación. En concreto, puede utilizarse directamente el simulador desarrollado para evaluar técnicas en los escenarios predefinidos y/o en nuevos escenarios que se creen, así como adaptar, mejorar y extender el código para mejorar el simulador e incorporar nuevas funcionalidades.

Trabajo	**Horas invertidas**
Estudio previo	10 h
Estudio del simulador y pruebas del prototipo existente	35 h
Depuración del proyecto: control operaciones I/O + modificaciones configuración y GUI	50 h
Depuración del proyecto: Gestión de las Bases de Datos (conexiones, operaciones e integración librerías externas)	75 h
Experimentos checkpoint mapa del museo	30 h
Estudio de opciones de creación de escenarios y de posibles ítems y atributos	30 h
Análisis y diseño del nuevo simulador y el creador de escenarios	40 h
Implementación del creador de escenarios: GUI	80 h
Implementación del creador de escenarios: Funcionalidades e integración con el simulador	120 h
Creación del escenario de prueba: *GranCasa*	30 h
Generación de datos (AUTO DataGenCARS)	25 h
Pruebas	20 h
Trayectorias de usuarios en grafo: búsqueda de soluciones	20 h
Trayectorias de usuarios en grafo: implementación de la solución (creador de mapas y simulador)	100 h
Construcción de escenario al aire libre (Campus Río Ebro)	2,5 h
Pruebas v2	25 h
Memoria y documentación	80 h
Total	772,5 h

Tabla 7.1: Horas invertidas en el proyecto (aproximadamente)

7.2. Conclusiones del proyecto

Este trabajo tenía como objetivo implementar un simulador genérico para CARS, pudiendo simularse cualquier tipo de escenario y, por tanto, permitiendo a cualquier

usuario crear estos mapas mediante una herramienta gráfica. Se destacan los principales hitos logrados:

- Se ha conseguido un simulador operativo, el cual permite obtener resultados adaptándose a la configuración deseada por el usuario. Esta configuración incluye, entre otros valores, la proporción entre tiempo real y tiempo de simulación, permitiendo este parámetro completar simulaciones rápidas.

- El simulador implementado es un simulador genérico, el cual puede reconocer cualquier mapa definido, así como cualquier tipo de ítem que contenga. Este objetivo se comprueba con escenarios simples (movimiento correcto de usuarios) y con el caso de estudio: *GranCasa*.

- Se ha implementado una nueva herramienta gráfica, complementaria al simulador, la cual permite crear escenarios de cualquier tipo. Esto implica: salas con cualquier forma poligonal, creación de cualquier tipo de ítem (visitable), escenarios con múltiples niveles/plantas y asignación aleatoria uniforme de ítems para el estudio del desempeño de escenarios aún no creados en la realidad entre otras características.

- Dada la arquitectura y los patrones de diseño utilizados para la implementación de las dos herramientas complementarias de este proyecto, se obtienen productos generales y desacoplados, no dependientes de tecnologías externas y fácilmente extensibles o escalables en caso de desearse para futuros proyectos.

A nivel de proyecto, creo que se han cumplido los objetivos definidos y puede resultar un excelente punto de partida para otros proyectos de investigación sobre *CARS*.

7.3. Trabajo Futuro

A continuación se presentan algunas ideas que podrían añadirse a extensiones futuras de este proyecto:

- Inclusión de nuevos algoritmos de recomendación para la evaluación del desempeño de estos en diversos escenarios, además de nuevos protocolos de comunicación para el intercambio de información entre los usuarios, buscando el equilibrio entre mejores resultados y los recursos utilizados en los dispositivos móviles de los usuarios.

- Mejora de la interfaz gráfica, probablemente utilizando otras tecnologías creadas para este propósito que no sobrecarguen el buffer, con el fin de mejorar la visión de las simulaciones.

- Herramienta "*replay*" que permita depurar visualmente cada instante de simulación, almacenando previamente una imagen de cada ciclo durante la ejecución con la configuración deseada.

- Inclusión de nuevos ítems visitables, creando nuevas clases que extiendan e implementen nuevas características a los objetos, por ejemplo, trayectorias de movimiento (como los usuarios de la simulación).

El código y las características del simulador y el editor pueden seguir mejorándose para obtener un producto cada vez más robusto de alto interés para proyectos de investigación sobre CARS y la simulación sobre escenarios para su evaluación por parte de otros investigadores, tanto aquellos con los que he colaborado como pertenecientes a otros grupos de investigación. En concreto, confío en que el trabajo desarrollado en este TFG sirva de base para investigaciones y extensiones en el contexto del proyecto NEAT-AMBIENCE.

Bibliografía

[1] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Raquel Trillo, and Ramón Hermoso. Context-Aware Recommendations Using Mobile P2P. In *Proceedings of the 15th International Conference on Advances in Mobile Computing & Multimedia*, MoMM2017, page 82–91, New York, NY, USA, 2017. Association for Computing Machinery.

[2] María del Carmen Rodríguez-Hernández and Sergio Ilarri. Pull-based recommendations in mobile environments. *Computer Standards & Interfaces*, 44:185–204, 2016.

[3] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Ramón Hermoso, and Raquel Trillo-Lado. Towards Trajectory-Based Recommendations in Museums: Evaluation of Strategies Using Mixed Synthetic and Real Data. *Procedia Computer Science*, 113:234–239, 2017. The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops.

[4] María del Carmen Rodríguez-Hernández, Sergio Ilarri, Ramón Hermoso, and Raquel Trillo-Lado. DataGenCARS: A generator of synthetic data for the evaluation of context-aware recommendation systems. *Pervasive and Mobile Computing*, 38:516–541, 2017. Special Issue IEEE International Conference on Pervasive Computing and Communications (PerCom) 2016.

[5] AUTO-DataGenCARS: Advanced User orienTed tOol DataGenCARS, 2021. [Online; accessed 20-June-2022].

[6] Wikipedia contributors. Data access object. `https://en.wikipedia.org/wiki/Data_access_object`, 2022. [Online; accessed 20-June-2022].

[7] *Oracle*. Class logger. `https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html`, 2020. [Online; accessed 20-June-2022].

[8] Steve Souza. JAMon (Java Application Monitor) A Java Monitoring API. `http://jamonapi.sourceforge.net/`, 2022. [Online; accessed 20-June-2022].

[9] opencsv - Opencsv Users Guide. http://opencsv.sourceforge.net/. [Online; accessed 20-June-2022].

[10] Wikipedia contributors. Peer-to-peer. https://en.wikipedia.org/wiki/Peer-to-peer, 2022. [Online; accessed 20-June-2022].

[11] Andew Zola and John Burke. time-to-live (TTL). https://www.techtarget.com/searchnetworking/definition/time-to-live, 2021. [Online; accessed 20-June-2022].

[12] Wikipedia. SimCity (videojuego) — Wikipedia, La enciclopedia libre, 2021. [Internet; downloaded 20-January-2022].

[13] P.W.D. Charles. open source micropolis, based on the original simcity classic from maxis, by will wright.

[14] Wikipedia contributors. SimTower — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=SimTower&oldid=1056044120, 2021. [Online; accessed 20-June-2022].

[15] CityGen3D. https://www.citygen3d.com/. [Online; accessed 20-June-2022].

[16] Tiled. https://www.mapeditor.org/. [Online; accessed 20-June-2022].

[17] TMX Map Format. https://doc.mapeditor.org/en/stable/reference/tmx-map-format/. [Online; accessed 20-June-2022].

[18] System to Test Mobile Computing Applications. http://webdiis.unizar.es/~silarri/prot/MobEnvSimulator/, 2018. [Online; accessed 20-June-2022].

[19] Refactoring Guru. Factory Method. https://refactoring.guru/es/design-patterns/factory-method, 2022. [Online; accessed 20-June-2022].

[20] Refactoring Guru. Abstract Factory. https://refactoring.guru/es/design-patterns/abstract-factory, 2022. [Online; accessed 20-June-2022].

[21] Oracle. PreparedStatement. https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html, 2020. [Online; accessed 20-June-2022].

[22] Refactoring Guru. Singleton. https://refactoring.guru/es/design-patterns/singleton, 2022. [Online; accessed 20-June-2022].

[23] Dr. D. Víctor Fernando Muñoz Martínez. *Planificación de Trayectorias para Robots Móviles*. PhD thesis, Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga, July 1995.

[24] Wikipedia contributors. Model–view–controller. `https://en.wikipedia.org/wiki/Model-view-controller`, 2022. [Online; accessed 20-June-2022].

[25] How to get a random point on the interior of an irregular polygon? `http://stackoverflow.com/a/19482012`, December 2013. [Online; accessed 20-June-2022].

[26] Wikipedia contributors. Polygon triangulation — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=Polygon_triangulation&oldid=1045540257`, 2021. [Online; accessed 20-June-2022].

[27] The Museum of Modern Art (MoMA) Collection. `https://github.com/MuseumofModernArt/collection`. [Online; accessed 20-June-2022].

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Análisis de requisitos

Se procede a presentar los requisitos funcionales y no funcionales del proyecto desarrollado.

A.1. Requisitos funcionales

RF	DESCRIPCIÓN
RF-1	El sistema permitirá la simulación de escenarios sobre los que realizar experimentos para evaluar sistemas de recomendación
RF-2	El sistema permitirá al usuario la definición de parámetros de configuración que afectarán al devenir de la simulación
RF-3	El sistema permitirá pausar y continuar la simulación cuando se desee, además de comenzarla e interrumpirla
RF-4	Se permitirá la especificación de una semilla como parámetro de configuración, la cual permitirá llevar a cabo simulaciones repetibles
RF-5	El sistema permitirá la recopilación de estadísticas adicionales a las valoraciones generadas por los usuarios a lo largo de la simulación
RF-6	El sistema permitirá la ejecución mediante lotes, sin interfaz gráfica, para llevar a cabo evaluaciones desatendidas
RF-7	El sistema dispondrá de un editor de escenarios
RF-8	El editor de escenarios permitirá la creación, guardado y lectura/edición de plantas, las cuales formarán escenarios simulables
RF-9	Los escenarios simulables estarán formados por un nombre, una o más plantas y una o más salas
RF-10	Las plantas de un mismo escenario tendrán una de las dos dimensiones comunes (altura o anchura), en función del sentido de su unión (derecha a izquierda o arriba a abajo)
RF-11	Las salas tendrán forma poligonal, por lo que contendrán un mínimo de tres esquinas
RF-12	Las salas deberán contener al menos una puerta (para poder ser accesibles) y podrán contener una o más escaleras y uno o más objetos

Tabla A.1: Requisitos funcionales (1/3)

RF	DESCRIPCIÓN
RF-13	El editor de escenarios proveerá al usuario de herramientas de manipulación del escenario, ajustes del escenario y creación de instancias de objetos sobre el escenario
RF-13.1	Las herramientas de manipulación serán: pintado, borrado, selección y movimiento de objetos
RF-13.1.1	La herramienta de manipulación de pintado permitirá colocar instancias de los diferentes tipos de objetos sobre el escenario (esquinas, puertas, escaleras y objetos visitables)
RF-13.1.1.1	No se permitirá la definición de escenarios que no sean válidos por violar alguna condición requerida en el mundo real
RF-13.1.1.2	Si el objeto a pintar es una esquina, el editor unirá todas las esquinas pintadas consecutivamente hasta que se indique lo contrario, para formar una sala formada por dichas esquinas
RF-13.1.1.3	Si el objeto a pintar es una instancia de un objeto, deberá encontrarse dentro de los límites de alguna de las salas existentes
RF-13.1.1.4	Si el objeto a pintar es un divisor de salas, los vértices del segmento divisor de la sala deberán pintarse sobre dos esquinas no consecutivas de una misma sala
RF-13.1.2	La herramienta de manipulación de borrado permitirá eliminar instancias de objetos del escenario
RF-13.1.2.1	Si el objeto a borrar es una esquina, se borrará toda la sala que contiene a dicha esquina
RF-13.1.3	La herramienta de manipulación de selección permitirá seleccionar un objeto, lo cual permitirá acceder a una pantalla para conocer y/o modificar los atributos de dicho objeto
RF-13.1.4	La herramienta de manipulación de movimiento permitirá mover un objeto a lo largo del escenario siempre y cuando este movimiento no viole las restricciones propias de cada tipo de objeto (no podrá haber dos paredes que se intersecan o ítems fuera de alguna sala)
RF-13.2	Los ajustes del escenario permitirán modificar las dimensiones del escenario, la proporción píxel/metros y el zoom con el que se visualiza el escenario que se está editando
RF-13.3	Los objetos dibujables sobre el escenario estarán compuestos por objetos básicos de creación del escenario y objetos visitables (puntos de interés con valoraciones asociadas)
RF-13.3.1	Los objetos básicos de creación del escenario serán: esquinas (para formar salas), puertas (para moverse entre salas), escaleras (para moverse entre plantas) y divisores de sala (para el correcto movimiento de los usuarios dentro de salas con formas poligonales complejas)
RF-13.3.2	Los botones de objetos visitables permitirán colocar instancias de distintos tipos de objetos visitables y evaluables por los usuarios del simulador

Tabla A.2: Requisitos funcionales (2/3)

RF	DESCRIPCIÓN
RF-14	El editor de escenarios proveerá al usuario de una barra de status, situada en la parte inferior del escenario, la cual mostrará al usuario en qué posición se encuentra, así como si se encuentra sobre algún objeto dibujado (con el puntero) y el tipo de objeto seleccionado (en caso de haberse seleccionado alguno)
RF-15	El sistema permitirá la creación, el borrado y la edición de botones de objetos visitables
RF-16	El sistema permitirá el guardado y la carga de conjuntos de objetos visitables, los cuales poseerán las características concretas especificadas por el usuario que poseerán las instancias creadas de cada tipo de objeto visitable
RF-17	Todos los objetos, básicos o visitables, contendrán al menos un identificador, una sala, un tipo, una posición en el escenario y un path con la ruta de su icono correspondiente
RF-18	Las puertas y las escaleras son elementos conectables, es decir, que podrán estar conectados a otros elementos conectables
RF-18.1	Las puertas estarán conectadas a uno o más elementos conectables
RF-18.2	Las escaleras estarán conectadas a otra escalera de distinta sala
RF-19	El objeto divisor de salas permitirá descomponer salas en subsalas, formando así nuevas salas con formas poligonales sin ángulos convexos a ojos del simulador
RF-20	Los objetos básicos disponen de iconos fijos, mientras que los objetos visitables podrán contener imágenes diferentes según su tipo
RF-21	El sistema permitirá la combinación de plantas de un escenario, creadas de forma independiente, formando un único mapa que las contenga a todas
RF-22	El sistema permitirá la colocación de un número determinado de instancias de un ítem visitable a lo largo del escenario de manera automática
RF-23	Los escenarios creados mediante el editor deberán poder ser leídos por el simulador
RF-24	El movimiento de los usuarios en el simulador no violará condiciones requeridas en el mundo real (los usuarios no podrán moverse atravesando las paredes ni fuera del interior de alguna sala)
RF-25	Los datos referentes a las valoraciones de cada ítem por cada perfil de usuario serán generados mediante la herramienta *DataGenCARS*/*Auto-DataGenCARS* o bien importados a partir de ficheros externos o conjuntos de datos existentes
RF-26	El simulador permitirá activar o desactivar la GUI cuando el usuario lo considere preciso durante una simulación
RF-27	El sistema permitirá exportar los datos referentes a los ítems de los escenarios creados para facilitar la generación de datos de la simulación
RF-28	Se proveerá de dos escenarios de prueba, siendo estos las plantas 4 y 5 del museo "MoMA" de Nueva York y el centro comercial "GranCasa", los cuales podrán ser leídos y modificados por el editor y simulados con sus datos correspondientes por el simulador

Tabla A.3: Requisitos funcionales (3/3)

A.2. Requisitos no funcionales

RNF	DESCRIPCIÓN
RNF-1	Las ventanas de la GUI de la aplicación serán redimensionables, pudiendo así utilizar la aplicación en cualquier tipo de pantalla
RNF-2	El simulador permitirá realizar simulaciones cuyo tiempo se adapte a la proporción entre tiempo de simulación y tiempo de ejecución especificados en la configuración, permitiendo así simulaciones más rápidas de tiempos reales elevados
RNF-3	Los mensajes de salida, tanto por consola Java como por GUI, serán controlados por un "Logger"
RNF-4	El "Logger" determinará qué mensajes mostrar, en función del nivel asociado al mensaje y el filtro asociado al "Logger", y dónde hacerlo, en función de su "Handler" asignado
RNF-5	Los datos correspondientes a las estadísticas recopiladas en simulación y a los ítems de un escenario serán exportados a formato *CSV*
RNF-6	Se aplicará el patrón arquitectural *DAO* para la gestión de la base de datos del simulador, utilizando *SQLite* y pudiendo utilizar cualquier base de datos extendiendo e implementando únicamente su acceso a datos correspondiente

Tabla A.4: Requisitos no funcionales

Anexos B

Manual de uso para la ejecución de simulaciones

En este anexo se procede a describir la forma de proceder para poder ejecutar simulaciones sobre un escenario. Se va a explicar la forma de compilar y ejecutar el proyecto, los archivos necesarios para poder ejecutar y los dos tipos de ejecuciones posibles: evaluaciones individuales y evaluaciones desatendidas.

B.1. Compilación y ejecución del proyecto

Como ya se ha comentado en el listado de tecnologías en la memoria principal (Sección 1.2) este proyecto está desarrollado en el lenguaje de programación *Java*. Por tanto, el primer requisito para poder ejecutarlo es tener instalado el *JDK*, el cual puede descargarse en el sitio oficial de *Oracle*: `https://www.oracle.com/java/technologies/downloads/`. La versión utilizada en este proyecto es: *jdk1.8.0_201*.

Para la compilación y ejecución del proyecto, se utiliza *Apache Ant*. Se ha de tener instalada esta herramienta y se recomienda incluir en la variable de entorno "*PATH*" la ruta a donde se ha instalado para poder ejecutarse desde cualquier ruta del sistema.

Por último, se ha de clonar el repositorio del proyecto de *GitHub* (`https://github.com/silarri/simulCARS`). Dentro del proyecto, se ha de acceder a: "*simulCARS/Simulator2021/MainSimulator/*". En esta ruta se encuentra el archivo "*build.xml*", script "Ant" creado para compilar y ejecutar el proyecto. Los comandos que se han de ejecutar desde la consola (Windows o Linux) son:

Limpiar y complilar:

```
ant clean
ant build
```

Y para ejecutar, se invoca a la clase principal:

```
ant MainSimulator
```

El uso de este comando en lugar de "*ant run*" se debe a que se pueden definir otros "targets" para probar pantallas concretas de la GUI (de este proyecto o de futuras implementaciones). Para ello, bastaría con incluir otra etiqueta "target" equivalente a la nombrada *MainSimulator* y modificar su nombre de etiqueta, nombre de clase y *classpath*.

B.2. Archivos necesarios para poder ejecutar

Para poder ejecutar la aplicación, no es necesario ningún archivo específico de inicio, es decir, se puede acceder a la aplicación y entrar al editor de mapas "*File* → *Create or Edit Map*" para crear y/o editar un mapa que será simulado en un futuro. Sin embargo, para llevar a cabo la simulación de un mapa para probar sistemas de recomendación sobre el mismo, hacen falta algunos ficheros:

1. **Ficheros con el contenido del mapa a simular:** Estos ficheros son tres y contienen la información correspondiente al mapa sobre el que simular. Estos archivos son el fichero de ítems, el de salas y el del grafo.

   El archivo de ítems, contiene el número total así como todos ellos con sus correspondientes características (incluídas las coordenadas de su posición en el mapa); el archivo de salas contiene la información general del mapa (nombre, anchura, altura, número de salas, etc.) y todas las salas con sus correspondientes elementos básicos para la generación de mapas (esquinas, puertas y escaleras) dotados de valor con sus coordenadas; y el archivo del grafo, similar al de salas pero conteniendo este la correspondencia de "IDs" de los elementos (para la creación del grafo) así como las conexiones entre elementos conectables (puertas y escaleras).

   La localización de estos archivos será, siguiendo la estructura del proyecto: "*Simulator2022/resources/maps*". En dicha carpeta se podrán encontrar varias subcarpetas con mapas creados. Se pueden encontrar los siguentes mapas: museo *MoMA*, actualizado para el nuevo simulador generalizado; escenario simple y escenario con salas poligonales complejas (con múltiples lados y ángulos convexos), para probar el correcto funcionamiento de las trayectorias de los

usuarios durante la simulación en distintas de salas; y centro comercial *GranCasa*, con todos los ítems y que se ha de probar con los nuevos datos generados.

2. **Bases de datos:** Se necesitan tanto la base de datos con toda la posible información del mapa creado como las bases de datos "modelo" que se crearán para los usuarios en cada simulación y en las que irán almacenando información. En función del tipo de comunicación, utilizarán o bien una única BD centralizada o cada usuario almacenará la información en su propia BD de valoraciones así como una con la información intercambiada (algoritmo *P2P*).

   El archivo correspondiente a la base de datos con toda la información actuará como un "oráculo", es decir, contendrá todas las valoraciones que cada usuario daría a cada ítem bajo cada contexto posible que pueden originarse a lo largo de la simulación, es decir, todas las posibles combinaciones de usuarios, ítems y contextos: "nºregistros = nºusuarios*nºítems*nºcontextos". Esta información se encontrará en la tabla "*user_item_context*", tabla principal a rellenar con datos para la simulación al crear una nueva fuente de datos para un nuevo escenario (con *AUTO-DataGenCARS* por ejemplo), además de las tablas "*user*", "*item*" y "*context*" en caso de requerir nuevos datos encontrados en dichas tablas (nuevos o diferentes usuarios, ítems y/o contextos en la nueva fuente de datos). El archivo referente a la BD de usuarios es similar al anterior pero tendrá la tabla de valoraciones vacía y será copiada y renombrada para almacenar las valoraciones de los usuarios (en función del tipo de red seleccionada para la comunicación de los usuarios, una centralizada o una para cada usuario). El archivo referente a la cola de información de las comunicaciones es análogo al anterior pero servirá de modelo para almacenar la información intercambiada en las comunicaciones cuando se utilice la red *P2P*.

   Estos archivos deben estár situados en la carpeta "*Simulator2022/resources/db*" del proyecto.

3. **Algoritmos de simulación:** Dada la generalización del Simulador, se ha de disponer del fichero que contiene los algoritmos disponibles y las características de estos (uso de red y redes que utiliza). Este archivo se ha de encontrar en la ruta "*Simulator2022/resources/recommenders*". En caso de añadir un nuevo algoritmo, se debe implementar su comportamiento en la función de recomendación.

La distribución esperada de todos los diferentes ficheros necesarios para la

simulación en los diversos directorios puede observarse en la Figura B.1.

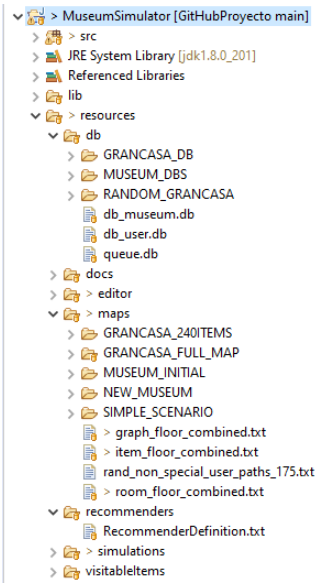


Figura B.1: Estructura del proyecto y archivos necesarios para la simulación

B.3. Ejecuciones individuales mediante GUI

Una vez se disponga de los archivos necesarios para la simulación, se puede proceder a simular. Para ello, lo primero es definir la configuración de la simulación a realizar. Esto se puede hacer en la ventana a la que se accede mediante "*Simulation* → *Configuration*".

En esta ventana, se podrán definir los parámetros deseados para la simulación. Se mostrarán por defecto los datos de la simulación del museo. La adaptación al nuevo escenario (GranCasa) así como cualquier nuevo mapa que pueda ser creado es sencilla, pues únicamente hay que modificar la escala ("*1 km represents in pixels [pixel]*" = "4000 (pixels)" en caso del mapa de GranCasa). El resto de parámetros se pueden modificar al gusto independientemente del escenario, como aumentar la velocidad de movimiento de los usuarios o el tiempo real por segundo de iteración. Este parámetro permite realizar simulaciones más rápidas que el tiempo real que representan, pero se ha de mantener bajo o igual al tiempo real simulado cuando se apliquen algoritmos de filtrado colaborativo basado en usuarios (como se explica en el Anexo C).

Opcionalmente, se pueden modificar las opciones de visualización (escalar el mapa para una visualización diferente o bien activar/desactivar la GUI en caso de delays con las operaciones de refresco de pantalla) en la opción del menú "View".

Tras haber definido la configuración, se puede proceder a simular: "*Simulation* → *Start*". Esta acción debería haberse habilitado tras definir la configuración.

B.4. Ejecuciones/evaluaciones desatendidas

Una nueva funcionalidad incorporada a este simulador son las evaluaciones desatendidas. Dicha funcionalidad permite llevar a cabo ejecuciones por lotes, sin interfaz gráfica, para recopilar múltiples resultados (probando diferentes algoritmos o diferentes parámetros de configuración en cada una) de forma automática.

Para llevar a cabo las evaluaciones desatendidas, debemos acceder a "*Simulation* → *Neglected Evaluations*". Se abrirá una ventana emergente que pedirá seleccionar el archivo (.txt) donde se encuentran las configuraciones a ejecutar y la carpeta donde se desean almacenar los resultados de cada simulación. El archivo de configuraciones para cada simulación se deberá crear siguiendo el paradigma **clave-valor**, de manera análoga al uso de la clase *java.util.Properties*, es decir, especificando cada parámetro de configuración seguido de su número de simulación (por ejemplo: "*timeAvailableUser_1*") e igualado al valor que se le desee proveer (por ejemplo: 1).

Se pueden encontrar ejemplos de archivos de configuraciones en la carpeta "*Simulator2022/resources/simulations*".

B.5. Ejemplo guiado de simulación de un escenario - GranCasa

Tras lo comentado en los apartados anteriores (Anexos B.3 y B.4) se procede a presentar un ejemplo práctico de una simulación. Para este ejemplo se utilizará el mapa de GranCasa, creado con el Editor de Mapas.

En primer lugar, es necesario colocar los archivos necesarios para la simulación en las rutas correctas. Se necesitan los archivos referentes al mapa, generados con el editor; los archivos referentes a la base de datos, generados con herramientas externas (recomendado *DataGenCARS*); y el archivo con la información de los recomendadores (no modificar a no ser que se añadan por código nuevos recomendadores). Estos archivos se han de ubicar en las ubicaciones tal y como muestra la Figura B.1.

Con los archivos necesarios, se puede proceder a ejecutar el proyecto. Al hacerlo se abre la ventana principal de la aplicación sobre la cual se plasmará la información de la simulación una vez establecida la configuración. Para establecer la configuración, se accede a la ventana *Configuration* "*Simulation* → *Configuration*" (Figura B.2).

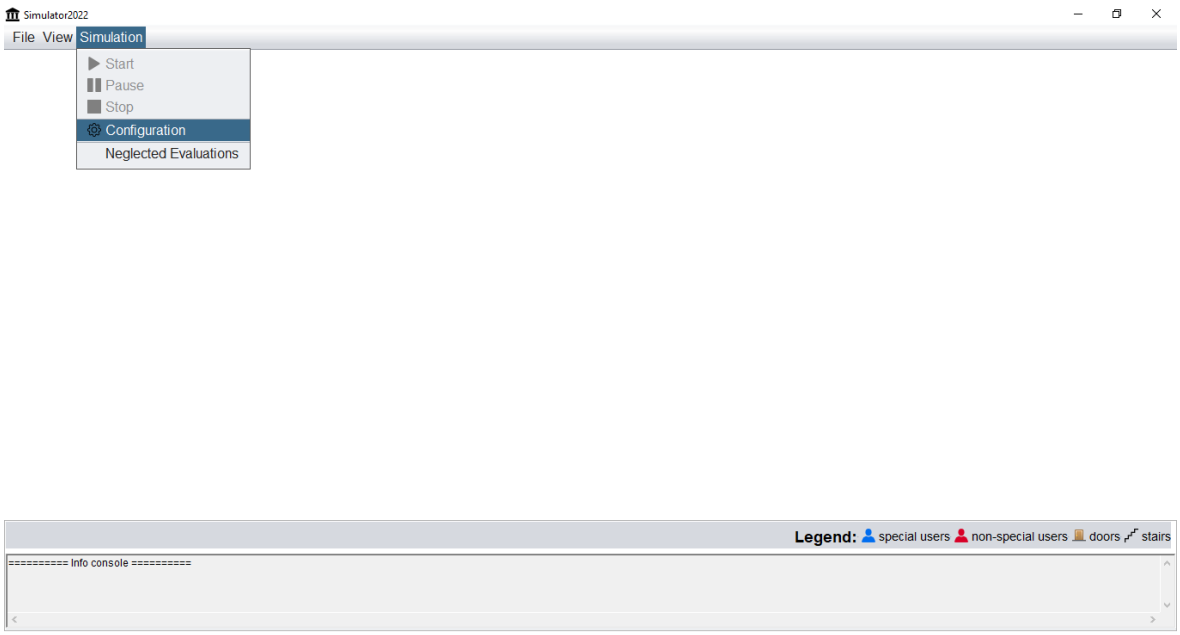


Figura B.2: Acceso a la ventana *Configuration*

Con esta acción, se muestra una ventana que permite definir todos los parámetros de la configuración de la simulación a realizar. Cabe destacar que la escala del mapa de GranCasa es: 4 píxeles equivalen a 1 metro, o lo que es lo mismo, "*1 km represents in pixels = 4000*". El resto de parámetros pueden establecerse tal y como se desee llevar a cabo la simulación. En el caso de la Figura B.3, se simulará a velocidad x30 ("*Real time per iteration = 30*"), se aplicará la estrategia *NPOI* para establecer los caminos de los usuarios sin recomendador, y el algoritmo de recomendación será *K-Ideal*. Cuando se dispone de la configuración deseada, se pulsa el botón "*Save and load combined floors*".

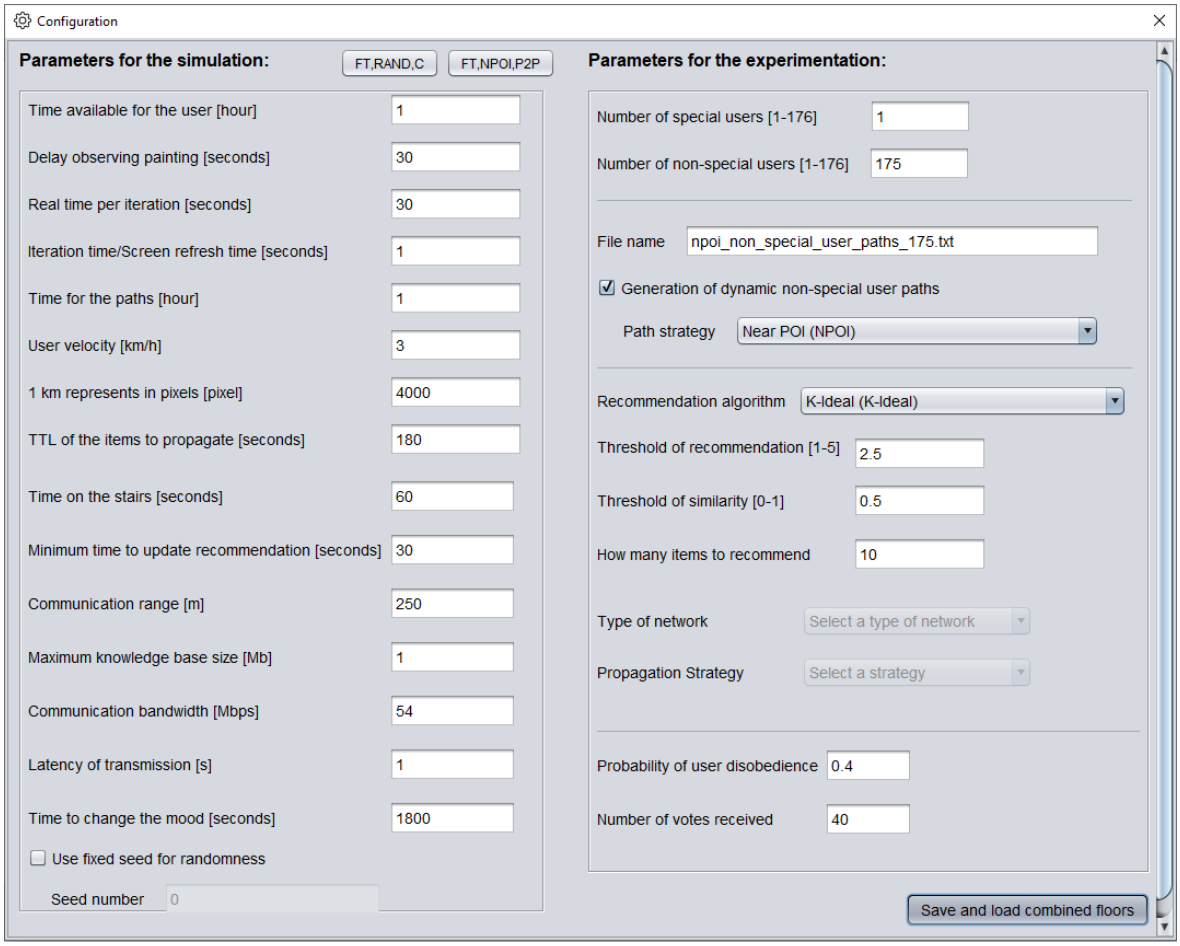
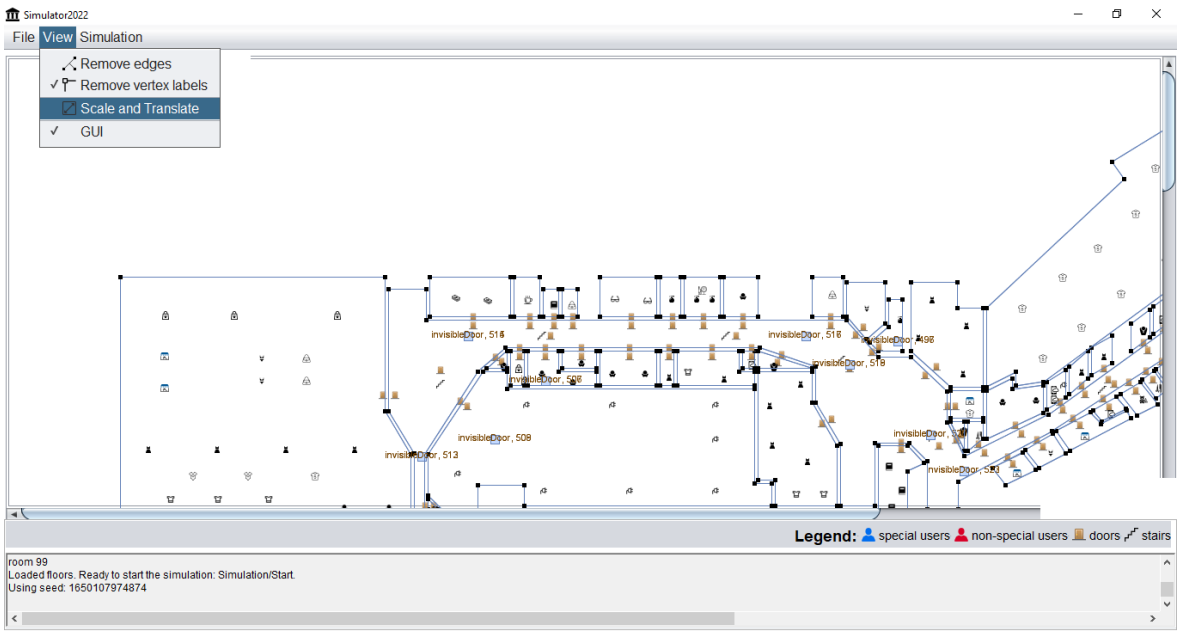


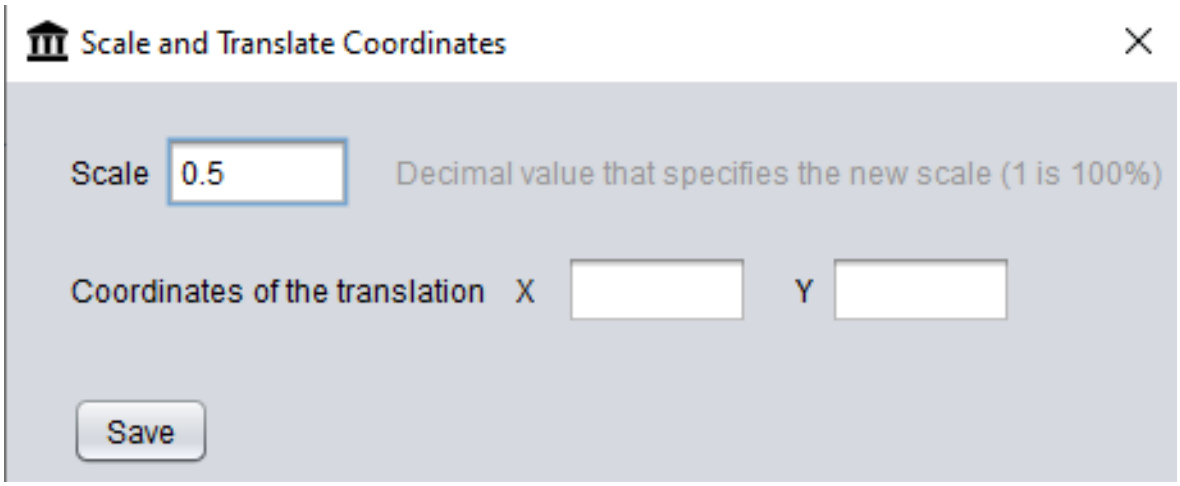
Figura B.3: Ventana *Configuration* - Ejemplo de configuración para la simulación

Sin embargo, como se puede observar en la Figura B.3, este mapa es más grande que el utilizado originalmente, es decir, el de las plantas 4 y 5 del mueseo MoMA, por lo que para visualizarlo mejor antes de comenzar con la simulación se puede escalar el mapa (Figura B.4a).

En este caso, se aplicará una escala que lo reduzca a la mitad y, como no se le quiere aplicar ningún movimiento, se dejan los campos referentes a las coordenadas de traslación vacíos (Figura B.4b).



(a) Opción *Scale and Translate*



(b) Ventana *Scale and Translate*

Figura B.4: Proceso *Scale and Translate*

Tras confirmar la configuración (y escalado si así se desea), se procede a cargar en el simulador la información de los ficheros (referente al mapa). Cuando toda la

información se ha cargado correctamente, se puede visualizar el mapa (si la GUI está activada) y se muestra en la consola localizada en la parte inferior de la ventana un mensaje que indica que se puede proceder a simular. Para ello, como indica el mensaje, hay que acceder a "*Simulation* → *Start*" (Figura B.5). Cuando se lleva a cabo dicha acción, se inicializan las trayectorias a seguir por los usuarios sin recomendador y las conexiones con las bases de datos y comienza a ejecutarse el algoritmo de simulación. También se puede visualizar en dicha imagen el cambio tras escalar, pudiendo observarse ahora dos de las tres plantas del centro comercial.

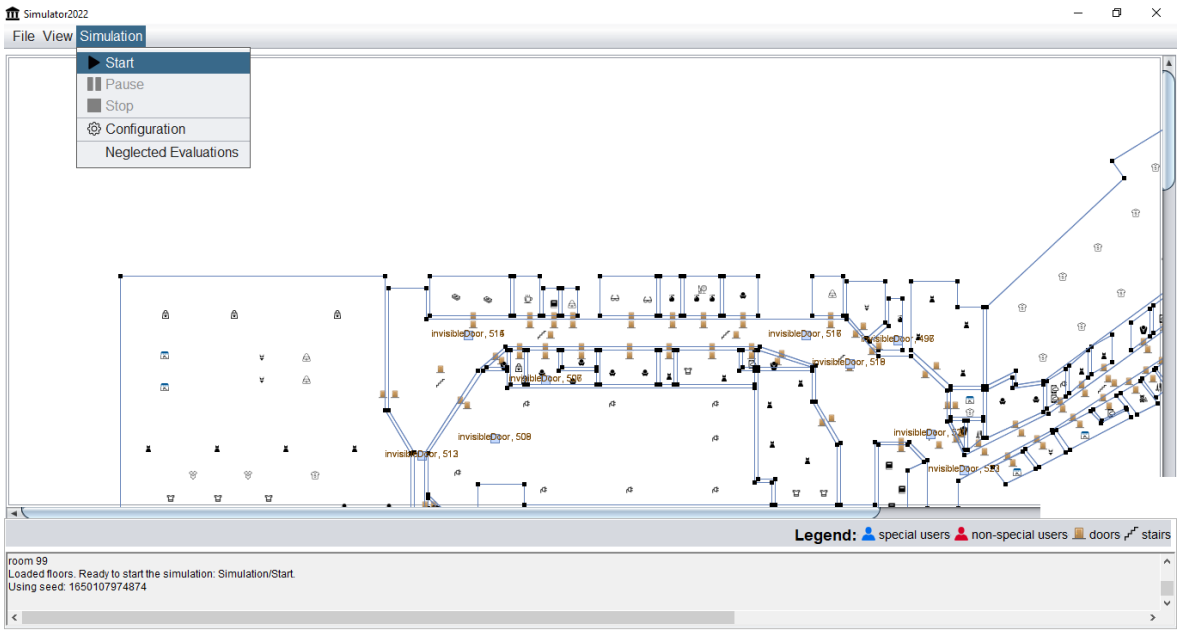


Figura B.5: Comenzar la simulación

Una vez comenzada la simulación, se muestran por la consola de la ventana de visualización mensajes que indican la evolución de la simulación. Además, si la GUI está activada, se verá la evolución de la simulación en el panel principal de la pantalla (escenario, movimiento de los usuarios en cada instante de la simulación e información de los elementos del escenario si se coloca el cursor sobre alguno de ellos). Esto se puede ver en la Figura B.6.

Una vez terminada la simulación, se muestra por consola un mensaje que indica el fin de la misma, por lo que ya se puede cerrar la aplicación o proceder con otra acción (otra simulación o acceder al editor).

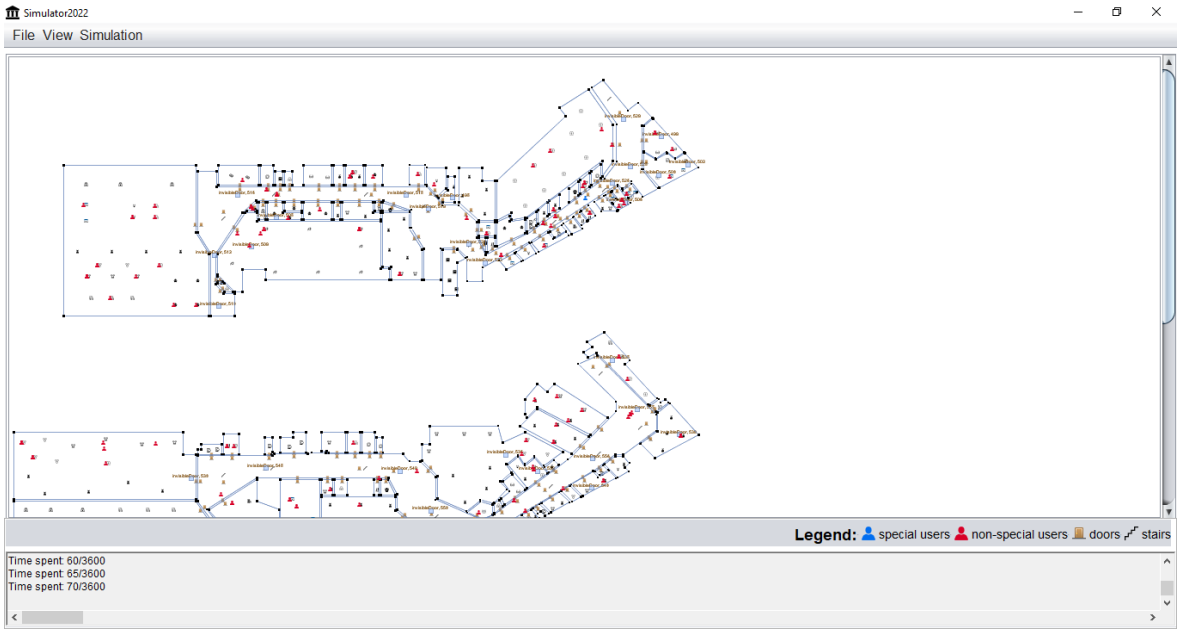


Figura B.6: Simulación en marcha

B.6. Ejemplo guiado de simulación de un escenario - museo MoMA u otros escenarios

La ejecución para cualquier tipo de mapa es equivalente a la mostrada en el apartado anterior (Sección B.5), pues los procesos y acciones a seguir son los mismos. La única diferencia que permite simular otros mapas (el del museo *MoMA* por ejemplo) se encuentra en los archivos que se ubiquen en las rutas especificadas en el Apartado B.2. Estos ficheros contienen la información del mapa sobre el cual simular y la información de los usuarios, ítems, contextos y valoraciones (de cada usuario a cada ítem en cada contexto) para el escenario que se desea simular.

Para simular otro escenario, por ejemplo el escenario del museo *MoMA*, se deben sustituir los siguientes ficheros:

– Dentro del propio proyecto, en la ruta en la que se deben situar los ficheros referentes al mapa (`MuseumSimulator/resources/maps`), se encuentran además diversas carpetas con los mapas disponibles. En dichas carpetas se ubican los archivos referentes a los ítems, salas y grafo del escenario correspondiente. Para utilizar el mapa del museo en las simulaciones, se copia el contenido (los tres ficheros) de la carpeta `MuseumSimulator/resources/maps/MoMA_Museum` en la ruta del mapa que utiliza el simulador (`MuseumSimulator/resources/maps`), sustituyendo los archivos que se estuvieran utilizando.

– De forma análoga a la gestión de ficheros del escenario, para utilizar la base de datos de otro escenario (museo) se ha de acceder a la ruta `MuseumSimulator/resources/db` y sustituir los archivos de la carpeta del mapa que se quiera simular (en este caso *MUSEUM_DBS*) por los que haya en la carpeta raíz de las bases de datos (*db*).

Los ficheros referentes a los mapas de los diferentes escenarios que se encuentran disponibles (diferentes carpetas en `MuseumSimulator/resources/maps`) han sido generados mediante la nueva herramienta implementada en este proyecto (creador de mapas). La única excepción es el caso del mapa del museo, el cual ya se encontraba disponible en el proyecto original, el cual no se encontraba adaptado a las nuevas características de este proyecto (sobre todo en lo referente a la gestión de "subsalas"). Esta adaptación, sin embargo, es prácticamente automática, pues basta con cargarlo en el creador de mapas y volver a guardarlo, quedando así aplicadas las modificaciones pertinentes para poder ser reconocido por el simulador.

Respecto a las bases de datos (y la información contenida en ellas) de los escenarios, tanto en el caso del proyecto original (museo) como en el caso del nuevo escenario (*GranCasa*), han sido generados con la ya mencionada herramienta *AUTO-DataGenCARS*.

Anexos C

Parámetros de configuración del simulador

En este Anexo se explican los parámetros de configuración del simulador, los cuales definen e impactan directamente en la simulación a realizar. La Figura C.1 muestra la ventana de configuración, en la cual se muestran todos ellos. Además, se crean dos botones para iniciar simulaciones rápidas (FT, Fast Test) usando distintas trayectorias (RAND y NPOI) y algoritmos de recomendación y uso de red (UBCF con red Centralizada o P2P). Son útiles para pruebas rápidas o depuración.

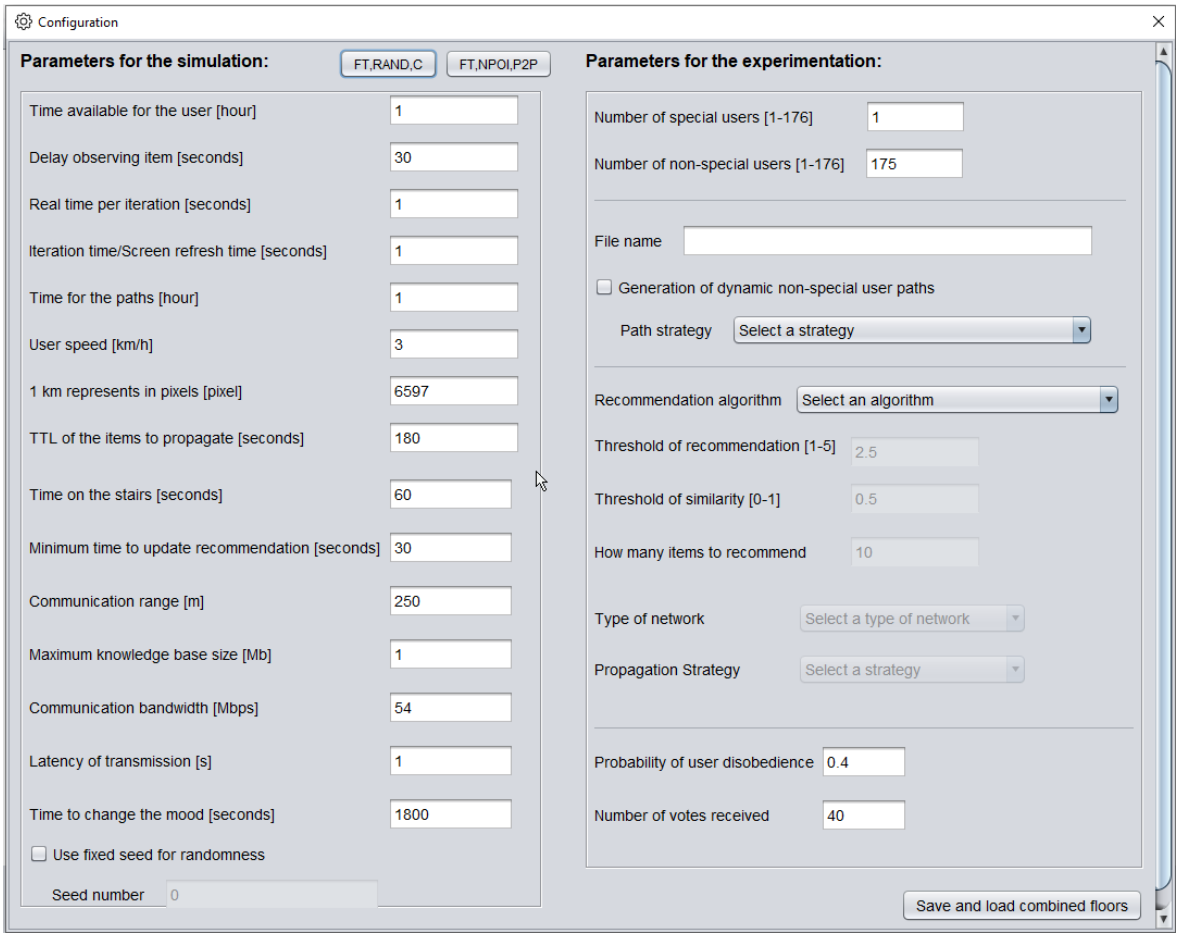


Figura C.1: Ventana de configuración del simulador

Los parámetros disponibles que permiten definir la configuración de la simulación se clasifican en dos grupos:

- *Parámetros de la simulación*: sirven para definir las características que afectarán a la simulación que se desea realizar. Se podrán definir parámetros como: el tiempo de simulación, velocidad de movimiento de los usuarios, tiempo real que representa cada iteración de la simulación, etc.

- *Parámetros para la experimentación*: determinan el tipo de experimento que se quiere realizar. Se podrá definir el algoritmo que adoptarán los usuarios sin recomendador en su trayectoria, el algoritmo de recomendación que probarán los usuarios con recomendador, los parámetros a considerar por el algoritmo de recomendación (umbrales de valoración dada, similitud y número de ítems a recomendar) y la red de comunicaciones que utilizarán los usuarios (en caso de que el algoritmo de recomendación permita diferentes tipos de comunicación).

Cada uno de ellos tiene un propósito específico que permite al usuario plantear diferentes experimentos con la variación de sus valores. Los parámetros configurables son:

- Time available for the user: tiempo (en horas) del que disponen los usuarios durante la simulación.

- Delay observing item: tiempo (en segundos) que estará cada usuario viendo cada ítem.

- Real time per iteration: tiempo real (en segundos) que representa cada iteración. Este parámetro permite ejecutar simulaciones mucho más rápidas que el tiempo real que están simulando. El valor introducido equivale a la velocidad de simulación (si el valor introducido es 5, la simulación se ejecutará a velocidad x5).

- Iteration time/Screen refresh time: cada cuántos segundos se ha de refrescar la pantalla, mostrando la evolución de la simulación de forma visual (para simulaciones con GUI).

- Time for the paths: tiempo límite (en horas) hasta el cual se puede generar la trayectoria de los usuarios.

- User speed: velocidad media (km/h) a la que se mueven los usuarios.

– 1 km represents in pixels: parámetro de la escala. Su valor equivale al número de píxeles que representan un kilómetro (relación entre kilómetros reales y píxeles en la interfaz).

– TTL of the items to propagate: cuántos segundos permanece la información de las votaciones de los usuarios en la red de comunicación *P2P*.

– Time on stairs: cuánto tiempo le cuesta a los usuarios subir/bajar escaleras para moverse entre plantas.

– Minimum time to update recommendation: si el número de votos que recibe el usuario con recomendador es mayor que umbral máximo de votos (parámetro de experimentación), tiempo que le cuesta al algoritmo de recomendación actualizar la trayectoria del usuario.

– Communication range: rango máximo de comunicación (en metros) cuando se utiliza la red *P2P*. Su representación se observa en la Figura 1 de [2].

– Maximum knowledge base size: tamaño máximo (en Mb) de almacenamiento disponible en la cola de información cuando se utiliza *P2P*.

– Communication bandwidth: ancho de banda (en Mbps) disponible en la comunicación entre usuarios de la red *P2P*.

– Latency of transmission: segundos de retardo en la transmisión de comunicaciones (en *P2P*).

– Time to change the mood: tiempo mínimo para modificar el estado de ánimo de los usuarios.

– Seed number: en caso de activarse, número de semilla establecido para la simulación (pudiendo así realizar simulaciones repetibles con el mismo valor de semilla).

– Number of special/non-special users: número de usuarios con y sin recomendador durante la simulación. La suma de estos no puede superar el máximo establecido, pero sí permitir cualquier combinación hasta dicho número.

– Path strategy: estrategia de trayectoria que seguirán los usuarios sin recomendador durante la simulación. Los valores posibles equivalen a los algoritmos simples (RAND, ALL y NPOI).

- Recommendation algorithm: algoritmo que utilizarán los usuarios con recomendador (special users). Los algoritmos disponibles se explican en la Sección 2.1.

- Threshold of recommendation: umbral mínimo de valoración a utilizar en la aplicación de recomendaciones. Su valor puede oscilar entre 1 y 5 (valores posibles de valoraciones). Utilizado en los algoritmos no simples.

- Threshold of similarity: valor del umbral mínimo de similitud entre usuarios (perfil de usuario), aplicando únicamente recomendaciones basadas en aquellas proporcionadas por otros usuarios similares. El valor introducido debe estar entre 0 y 1. Utilizado en los algoritmos no simples.

- How many items to recommend: número máximo de ítems a recomendar entre los disponibles, ordenados por orden de puntuación (valoración).

- Type of network: tipo de red de comunicación utilizada. Puede ser centralizada o *P2P*. Únicamente aplica si el algoritmo de recomendación es UBCF. En cualquier caso se emplea una única base de datos para almacenar toda la información generada por los usuarios durante la simulación (como la red centralizada) pero no se utiliza para la comunicación entre los usuarios.

- Propagation Strategy: estrategia de propagación de la información para la red *P2P*.

- Probability of user disobedience: probabilidad con la que los usuarios con recomendador desobedecerán las recomendaciones que se les han brindado y visitarán otro ítem.

- Number of votes received: si el usuario recibe un número de votos mayor al definido en este parámetro, se actualiza su trayectoria a partir de toda la información disponible.

El valor especificado en estos parámetros impacta directamente sobre las simulaciones, modificando sus características, su desempeño y/o sus resultados. Se pueden realizar múltiples pruebas con combinaciones diferentes de estos para ver su impacto sobre las simulaciones y las valoraciones de los usuarios.

Anexos D

Arquitectura del proyecto y posibles extensiones

En este anexo se explica la arquitectura del proyecto, con los diferentes paquetes y tecnologías que lo componen (Anexo D.1), los aspectos de diseño más relevantes del simulador (Anexo D.2) y los aspectos de diseño más relevantes del creador de escenarios (Anexo D.3)

D.1. Arquitectura del proyecto

El proyecto está compuesto por diferentes paquetes, los cuales componen una parte fundamental de código que cumple un propósito específico. Según su objetivo, en cada uno de ellos se utilizan diferentes tecnologías que permiten desempeñar sus respectivas tareas. La distribución y relación de los diferentes paquetes que componen el proyecto, así como las tecnologías utilizadas en cada uno de ellos, se muestra previamente en la Figura 3.1.

Como se puede observar, existen paquetes destinados específicamente al editor (*es.unizar.editor.**), otros al simulador (GUI, base de datos y acceso a datos y sistemas de recomendación) y otros tienen un propósito general y sus clases pueden reutilizarse en ambas partes del proyecto (acceso a datos de fichero, constantes y variables de configuración e imágenes). De esta manera se pueden identificar mejor las diferentes partes del proyecto y modificar únicamente las partes deseadas. Se pueden encontrar los siguientes paquetes:

– Paquete "*access*": contiene todas las clases de acceso a datos contenidos en ficheros. Estas clases permiten la lectura y/o escritura de los ficheros que contienen la información necesaria por el simulador y el creador de escenarios, referente a los escenarios y algoritmos de simulación disponibles (Sección B.2). Ofrecen una traducción transparente de los ficheros a las clases *Java*, basándose en la clase estándar *Properties* para un acceso óptimo a la información. La clase *DataAccess* contiene las operaciones básicas de lectura y escritura de propiedades

y el resto de clases extienden dicha clase e implementan los "getters" y "setters" propios de cada fichero.

– Paquete "*database*" y "*dao*": parte perteneciente al modelo del simulador, controlan la conexión y el acceso a las fuentes de datos respectivamente. Como se comenta en el Anexo B.2, hay diferentes bases de datos necesarias para la simulación:

  - La base de datos "oráculo", de la cual se consulta durante la simulación ya que contiene toda la información referente a usuarios, ítems y contextos del escenario, además de las valoraciones de cada posible combinación de los tres elementos anteriores. Al ser usuarios simulados y no reales, se han de generar previamente las posibles valoraciones que otorgarían a cada ítem en cada contexto en función de su perfil de usuario en un escenario real, siendo esta la información consultada cuando los usuarios valoran ítems durante la simulación.
  - La base de datos de los usuarios, equivalente a la anterior pero con la información de valoraciones vacía, siendo esta completada por los usuarios a lo largo de la simulación a medida que van valorando ítems según contextos y consultada por los usuarios con recomendador para la aplicación de recomendaciones. Puede ser única o existir tantas de ellas como usuarios, en función de si la red de comunicación es centralizada o P2P.
  - Las colas de información de los usuarios, adoptando este nombre dada su estructura de datos, almacenan las valoraciones que otros usuarios han dado a ciertos ítems. Esta información permanece por un tiempo determinado (TTL) en la red de comunicación (especificado en la configuración de la simulación). Estas bases de datos únicamente se crean y utilizan si la red de comunicación utilizada es P2P.

  Este paquete se ha diseñado para ser lo suficientemente general y desacoplado que se puedan incorporar fácilmente otras fuentes de datos a utilizar sin afectar a otros paquetes. Esta información se amplía en el Anexo D.2.

– Paquete "*editor*": paquetes pertenecientes al Editor de Mapas. Siguen el patrón *MVC* [24], para una mejor gestión de los diferentes módulos y para mayor escalabilidad y desacoplamiento en caso de querer ampliarse en futuros proyectos. Se explica más a fondo en el Anexo D.3.

– Paquete "*gui*": contiene las clases que componen la interfaz del simulador. Entre ellas, se encuentra la clase principal (que contiene el método principal o *main*) llamada *MainSimulator*, aquellas que contienen los algoritmos de ejecución y simulación (*UserRunnable* y *Simulation*) y las clases encargadas de representar visualmente el grafo (los elementos de la simulación) y actualizarlo en cada ciclo de simulación (dentro del paquete *graph*).

– Paquete "*recommendation*": incluye todas las clases empleadas en la aplicación de recomendaciones y trayectorias de los usuarios. Se encuentran aquí las clases encargadas de filtrar los objetos aplicando recomendaciones dependientes del contexto (paquete *filter*), las que permiten generar caminos mediante algoritmos simples (paquete *path*) y las que gestionan y modifican las trayectorias de los usuarios con la aplicación de algoritmos de recomendación (paquete *trajectory*).

– Paquete "*images*": paquete creado para almacenar imágenes. Algunas de ellas son imprescindibles para el correcto funcionamiento del proyecto, siendo estas las referentes a los elementos básicos de la simulación (usuarios, esquinas, puertas, escaleras, lápiz, goma de borrar, etc.). Además puede utilizarse para almacenar nuevas imágenes de los ítems que se creen en nuevos mapas si así se desea.

– Paquete "*util*": en él se encuentran las constantes y clases con operaciones básicas comunes, utilizables por otros paquetes. Entre las clases de propósito general se incluyen el filtro y formato de los "loggers", la gestión de semillas de números aleatorios, distancias e intersección de elementos de los mapas o las estructuras de datos de las estadísticas a recopilar durante la simulación.

Esta arquitectura ha sido extendida y redistribuida respecto al proyecto original, extrayendo y adaptando únicamente las clases necesarias de la librería *MOONRISE.jar* (la cual suponía un alto acoplamiento a la misma e impedía la optimización del acceso a datos), creando la nueva estructura del editor de escenarios y redistribuyendo las clases y paquetes con la ampliación y generalización del simulador.

D.2. Diseño del Simulador

En esta sección se presentan las partes de diseño más relevantes del simulador. Se incide principalmente en los cambios más notables realizados durante este proyecto. Los patrones de diseño empleados para la correcta gestión de datos en el simulador se

ilustran en la Figura D.1.

Como ya se ha explicado anteriormente en la Sección 4.1.2, una de las modificaciones más notables realizada al simulador se encuentra en la gestión y acceso a bases de datos. Inicialmente no se controlaban ni las conexiones creadas ni las consultas realizadas, dando como resultado una notable degradación de prestaciones, además de hacer muy complicada la gestión de este aspecto del código. Se aplican diferentes patrones de diseño con el fin de desacoplar y poder escalar estos módulos.

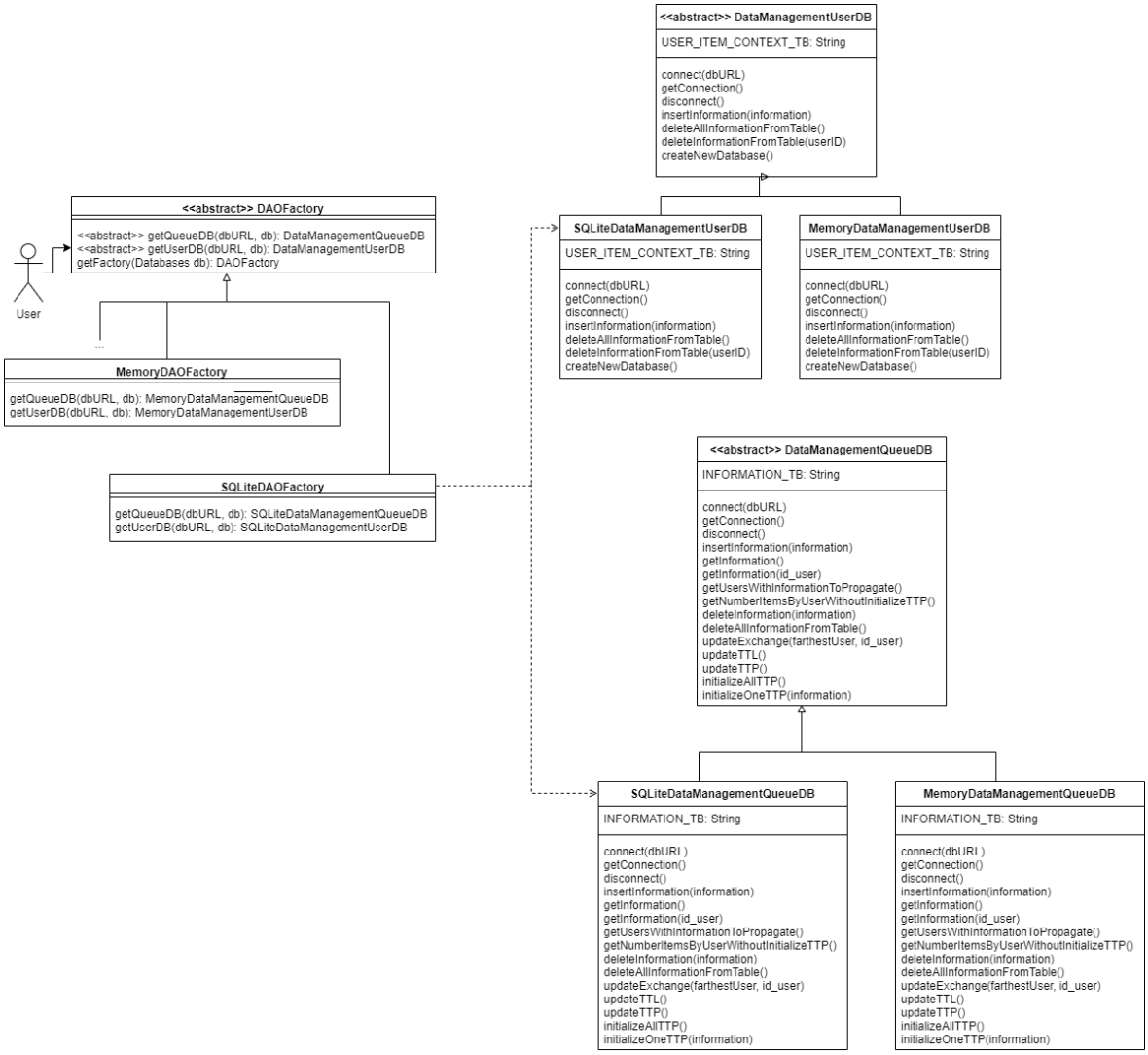


Figura D.1: Diagrama patrón DAO + patrón Factory para el uso de diversas fuentes de datos

Para optimizar y separar la lógica de acceso a datos del resto de la lógica del proyecto se aplica el patrón de diseño *DAO*, que permite unificar todas las operaciones de acceso a datos. De esta forma, se encapsula la lógica de acceso a datos en el paquete

dao del proyecto, facilitando la incorporación o modificación de operaciones de acceso a datos en función de necesidades de información de este o de futuros proyectos.

Además, al desacoplar la lógica de acceso a datos se obtiene un módulo independiente fácilmente sustituible. Por tanto, se pueden implementar nuevos módulo de acceso a datos que implementen las operaciones requeridas adaptadas a una nueva fuente de datos. De esta forma se puede incorporar cualquier fuente de datos deseada, creando un nuevo módulo de acceso a datos (que extienda las clases con las operaciones de acceso a datos requeridas adaptadas a la misma) y un módulo de conexión a la nueva fuente de datos a incorporar.

Para poder sustituir automáticamente estos módulos en ejecución se aplica el patrón de diseño *Factory*. Este patrón permite crear en ejecución una instancia del producto (en este caso módulo de acceso a datos) deseado de forma transparente para el usuario. Para ello, se definen las interfaces comunes de las clases de acceso a datos requeridas en este proyecto y se crean productos de las mismas, implementando la lógica de su fuente de datos correspondiente. De esta forma se puede definir en ejecución el acceso a datos (y conexión) de la fuente de datos deseada modificando únicamente el valor de una constante (que indica la fuente de datos a utilizar).

D.3. Diseño del creador/editor de mapas

El Editor de Mapas tiene como objetivo crear y/o editar los mapas a simular. Estos mapas pueden contener los diversos objetos disponibles e instanciables sobre el mapa en dicha herramienta, como se explica en la Sección 5. La estructura de los diversos objetos, así como la relación entre los mismos, viene explicada dado el diagrama de clases de la Figura D.2.

Los objetos se estructuran de forma que se crean las salas (mediante una lista de esquinas concatenadas) y en ellas se pueden dibujar una o varias puertas (para ser accesibles) y tantos objetos como se quiera. Además, las puertas y escaleras implementan la interfaz *Connectable*, pudiendo conectarse entre sí.

De esta manera, si quisiera ampliarse esta herramienta, podría hacerse de forma sencilla mediante la creación de nuevas clases que extiendan el comportamiento de alguna de las clases existentes.

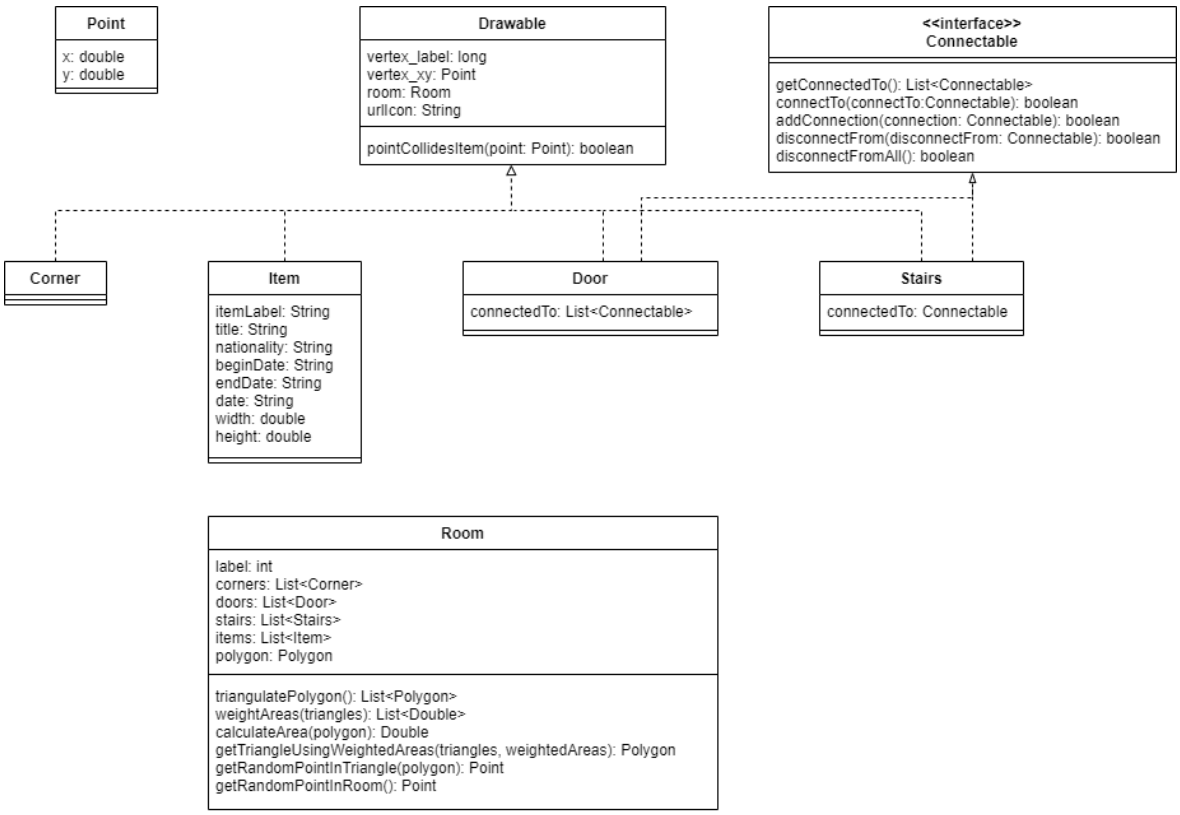


Figura D.2: Diagrama de Clases del Contenido del Modelo del Editor

Si quisieran crearse nuevos ítems (visitables por los usuarios durante la simulación) que posean unas características o comportamientos más concretos que los distingan del resto, bastaría con crear una nueva clase para este tipo de ítem que extienda la clase *Item*. Esto permite que los nuevos ítems puedan dibujarse, tratarse y almacenarse de forma idéntica a los demás, teniendo que gestionar únicamente las nuevas características según proceda.

El tratamiento de los nuevos tipos de ítems implicaría crear una nueva clase en la GUI (vista) que extienda **el panel de atributos del ítem**, incluido por ejemplo en la ventana de creación del tipo de ítem o en la ventana de atributos del objeto (mostrada con doble "click" sobre el objeto utilizando la herramienta de selección del editor), incorporando al panel los campos necesarios que representen el nuevo comportamiento del ítem. También se debería gestionar el correspondiente tratamiento (controlador) de la modificación de los campos añadidos en la interfaz para reflejarlos en el modelo. Por último, se deberían añadir las nuevas operaciones de lectura y escritura de los atributos añadidos en la clase de acceso a datos del fichero de ítems para preservar y recuperar esta nueva información de los ficheros del mapa e incorporarse el tratamiento

de este comportamiento añadido en el simulador.

Del mismo modo, si se quisiese implementarse algún tipo de objeto nuevo, la forma de proceder sería extender la clase genérica de objetos dibujables (*Drawable*) e implementar el nuevo comportamiento de este tipo de objeto. De manera similar al caso anterior, fuera del modelo únicamente sería necesario añadir un nuevo botón en la sección de objetos que corresponda en la interfaz del editor de mapas (GUI) y un nuevo panel de características del objeto (si procede); y tratar cada una de las operaciones disponibles (dibujado, borrado, selección y movimiento) para el nuevo tipo de objeto según sus restricciones (controlador). Por último, habría que añadir una nueva estructura de datos (lista) para almacenar el nuevo tipo de objetos, o bien dentro de las salas o bien de forma independiente a ellas.

Esto se comprobó dado el problema surgido con las trayectorias de los usuarios en simulación. La solución que se adoptó pasaba por un nuevo tipo de objeto que permitiera dividir las salas en "subsalas". Su incorporación a la herramienta fue menos costosa dado el diseño adoptado, permitiendo incorporar en poco tiempo el nuevo tipo de objeto y teniendo que enfocar los esfuerzos principalmente en el desarrollo del comportamiento de este (en editor y simulador) y en su persistencia en los ficheros de los mapas (creados en el editor y leídos por el simulador).

Anexos E

Estudio de posibles ítems y sus correspondientes atributos

Uno de los aspectos fundamentales de este proyecto son los ítems visitables. Estos son los objetos que pueden ser visitados por los usuarios a lo largo del escenario y sobre los que explotar los CARS, recomendando aquellos ítems que puedan resultar de su interés en función de sus gustos personales (además del contexto). La aplicación de recomendaciones sobre ítems de interés para los usuarios se basa en las características propias de estos.

Sin embargo, el propósito de este proyecto es la elaboración de un simulador genérico, que permita simular cualquier tipo de escenario con cualquier tipo de ítem. Para ello, los objetos visitables que pueden ser creados e instanciados en el editor de mapas han de poseer unas características lo suficientemente genéricas como para representar cualquier tipo de ítem. No todas las características de todos los ítems han de tener valor (esto ya ocurría en el escenario del museo), pero a pesar de esto, no se ha de permitir crear ítems con características que no tengan sentido dada su naturaleza. Por ejemplo, el atributo artista tiene sentido para las pinturas o esculturas de un museo pero no para las tiendas o secciones de un centro comercial.

Por este motivo, se estudian diferentes posibles escenarios sobre los cuales puede tener interés realizar simulaciones y sus posibles elementos visitables para determinar así las características que todos ellos puedan tener en común.

Los posibles escenarios estudiados así como sus posibles elementos visitables son:

- Museo, que puede contener pinturas y esculturas
- Centro comercial, que puede contener tiendas y restaurantes. Además, las tiendas se pueden desglosar en secciones de tiendas en el caso de tiendas muy amplias
- Mercado o supermercado, que puede contener tiendas. Similar al caso anterior, según el nivel de detalle deseado, los ítems pueden descomponerse en secciones

dentro de una tienda. Se piensa este escenario teniendo como referencia el Mercado Central de Zaragoza

- Librerías o videoclubs, que contienen productos (libros o películas)
- Cines, en los que se pueden visitar sus salas y/o películas visualizables
- Eventos sociales, en los cuales buscar amigos y/o gente con gustos similares

A partir de estos escenarios variados, se estudian los posibles atributos que pueden representar a cada tipo de objeto, para determinar los atributos comunes. Este estudio se puede visualizar en las Tablas E.1 y E.2. Las celdas con "X" implican que el ítem tiene dicho atributo y "*" implica que podría contenerlo pero no es obligatorio.

	Tipo de ítem	Nombre/Título	Imagen	Nacionalidad	Fecha	Fecha de inicio	Fecha de fin	Altura	Anchura	Artista	Constitución	Medio	Catalogado
Pinturas/esculturas	X	X	X	X	X	X	X	X	X	X	X	X	X
Tiendas/secciones	X	X	X	X	X	X	X	X	X				
Restaurantes	X	X	X	X	X	X	X	X	X				
Puestos	X	X	X	X	X	X	X	X	X				
Salas/películas cine	X	X	X	X	X	X	X	X	X	X			
Libros/películas	X	X	X	X	X	X	X	X	X	X			*
Productos	X	X	X	X	X	X	X	X	X				*
Personas/amigos	X	X	X	X	X	X	X	*					

Tabla E.1: Estudio de posibles ítems y sus correspondientes atributos (1 de 3)

Finalmente, los atributos suficientemente generales y por tanto utilizados en el editor para la creación y representación de ítems visitables son: tipo de ítem, nombre/título, imagen, nacionalidad, fecha, fecha comienzo, fecha fin, altura y anchura. La categoría se considera equivalente al tipo de ítem, motivo por el cual no se incluye en los atributos generales seleccionados. Además, la descripción se considera opcional en muchos casos y es muy concreta (única para cada objeto), por lo que se excluye ya que no aporta valor en la generación de datos por dicho motivo.

Los campos de fecha pueden ser utilizados en función del tipo de ítem para representar distinta información. Por ejemplo, el atributo "fecha" puede representar la fecha de apertura de una tienda o restaurante, la fecha de creación de una obra de arte o la fecha de nacimiento de una persona; y los atributos "fecha comienzo" y "fecha fin"

	Línea de crédito	Número de acceso	Departamento	Fecha de adquisición	Profundidad	Diámetro	Peso	Productos	Descripción	URL Página web	WhatsApp pedidos	Correo electrónico	Categoría
Pinturas/esculturas	X	X	X	X	X	X	X						
Tiendas/secciones			X					X	X	X	*	X	X
Restaurantes								X	*	X	*	*	X
Puestos								X	X	*	X		X
Salas/películas									*				X
Libros/películas									X				X
Productos			*	*	*	*	*		X				X
Personas/amigos									*			*	

Tabla E.2: Estudio de posibles ítems y sus correspondientes atributos (2 de 3)

	Pedido mínimo	Sólo a recoger	Número de teléfono	Género	Cartelera	Rango de edad recomendado	Asientos disponibles	Valoración de los expertos	Velocidad de movimiento	Trayectoria	Hobbies
Pinturas/esculturas											
Tiendas/secciones			*								
Restaurantes		*	*				*				
Puestos	X	X	X								
Salas/películas				X	X	X	X	X			
Libros/películas				X		X		X			
Productos	*			X		*		*			
Personas/amigos				X					X	X	X

Tabla E.3: Estudio de posibles ítems y sus correspondientes atributos (3 de 3)

pueden representar el horario de apertura de un establecimiento, el periodo histórico en el que se creó una obra de arte o la duración de una película.

Si se considera que alguno de los campos, como las fechas o la nacionalidad, no se ajustan o no se quieren definir para algún tipo de ítem concreto, sus valores pueden mantenerse nulos (salvo el tipo de ítem, definido como obligatorio). Esto puede aplicarse también en el caso concreto de los usuarios, en el que pueden obviarse la altura y la anchura

En caso de así desearse para escenarios concretos, la creación de nuevos tipos de ítems con mayor número de atributos es posible extendiendo la clase *Item* en el modelo del editor. De esta forma se pueden interpretar de igual manera por el editor y simulador estos nuevos ítems y añadirles además las nuevas características o comportamientos deseados. En caso de añadir comportamientos, como trayectorias de usuarios en el caso del escenario al aire libre, se ha de adaptar el simulador para reconocer dicho comportamiento (en este caso tratándose de igual manera que las trayectorias de los usuarios de la simulación).

Anexos F

Mapas creados con el editor

En este Anexo se presentan los escenarios creados con el editor, tanto el escenario del centro comercial *GranCasa* (Anexo F.1) como el del Campus Río Ebro (Anexo F.2).

F.1. Mapa *GranCasa*

El escenario principal utilizado como caso de estudio en este proyecto es el mapa de *Grancasa*. Este es el motivo por el cual su elaboración se pone un especial detalle.

Para la construcción del escenario en el editor, se toman como referencia dos fuentes, la página del *Colegio Oficial de Arquitectos de Aragón* (`https://www.coaaragon.es/Default.aspx?Cod=202`) y el sitio web del propio Centro Comercial *Grancasa* (`https://www.grancasa.es/mapa/#/`). Es necesario combinar la información encontrada en ambas para poder representar dicho mapa, además de un trabajo adicional que se comenta a continuación.

En la primera referencia únicamente se encuentran los planos de la planta baja y, aunque dispone de puntos de referencia, no se encuentran las acotaciones (medidas) de cada uno de los elementos (tiendas, pasillos, etc.). En cambio, en este enlace sí se comentan las dimensiones totales del centro, 400x90 metros, lo que se toma como referencia para la creación de las diferentes salas. De la web oficial del centro se pueden extraer los mapas de todas las plantas con sus respectivas tiendas/restaurantes, aunque de nuevo sin ningún tipo de medidas.

Por este motivo, la mayor parte del trabajo realizado en la elaboración de este escenario se realiza fuera del editor de mapas, imprimiendo las tres plantas del centro comercial de las páginas comentadas, midiendo cada elemento (altura, anchura y posición) y aplicando la equivalencia de dichas medidas frente a las dimensiones totales del mapa (transformadas a centímetros sobre el papel). Esta tarea suma un tiempo mucho mayor del necesario para crear un mapa directamente con el editor si se

dispone de las medidas pertinentes (el 70 % aproximadamente), pero se realiza ya que se quiere una representación del escenario de prueba lo más próxima y fiel a la realidad.

Con las medidas obtenidas, basta con "dibujar" las salas como se podría hacer con cualquier otra herramienta de dibujo común. Con el lápiz y la barra de estado se colocan los elementos sobre el mapa y, si las medidas no han sido completamente precisas en una primera instancia, se mueven los objetos con la herramienta de movimiento o bien desde la ventana de propiedades del elemento, proporcionando con esta última la posición exacta deseada (en píxeles o en metros). Finalmente se establecen las conexiones entre puertas y escaleras.

Una vez obtenidas las medidas y creadas las plantas en el editor, se han de crear los tipos de ítems que componen el mapa y que pueden visitar los usuarios durante la simulación. En este caso, los ítems a visitar son secciones dentro de una tienda. Se elige este como el tipo de ítems ya que las tiendas completas son demasiado generales, con mucha diferencia de tamaño entre ellas, y sobre todo porque muchas tiendas venden diferentes tipos de productos y engloban varias categorías diferentes.

Las secciones así como sus tipos o categorías se extraen del apartado de tiendas del sitio oficial del centro (`https://www.grancasa.es/tiendas/?`). El escenario final está formado por 26 tipos de ítems diferentes a partir de las categorías y subcategorías encontradas en este enlace. La creación de estos tipos de ítems conlleva la búsqueda y descarga de sus correspondientes iconos representativos en formato *PNG*. De nuevo esta tarea implica un tiempo mucho mayor del necesitado para la creación de los tipos de ítems en el editor.

Además, se buscan una a una las tiendas y las categorías en las que se encuentran para crear instancias de los ítems a lo largo del mapa. En aquellas salas que aparecen en más de una categoría en la web, se crea al menos un tipo de ítem por cada categoría. Además, en las tiendas con una extensión mayor, se crean múltiples secciones a lo largo de su extensión (por ejemplo, en la tienda *Decathlon* se incluyen las secciones de fútbol, baloncesto, montaña, atletismo, etc.). Las tiendas con unas dimensiones pequeñas y que comprenden una única categoría contienen una única sección que engloba la propia tienda. El escenario final contiene un total de 283 secciones de tiendas diferentes.

En las Figuras F.2, F.4 y F.6 se muestran las plantas creadas con el editor, las cuales se pueden comparar con las Figuras F.1, F.3 y F.5, extraídas del sitio oficial y

tomadas como referencia en la construcción del mapa con el editor. Cabe destacar que en el diseño final, no todas las tiendas con ángulos convexos se dividen en subsalas (o en tantas subsalas como ángulos convexos contienen) si todos sus ítems son accesibles mediante una líneas rectas sin interferir con las paredes.

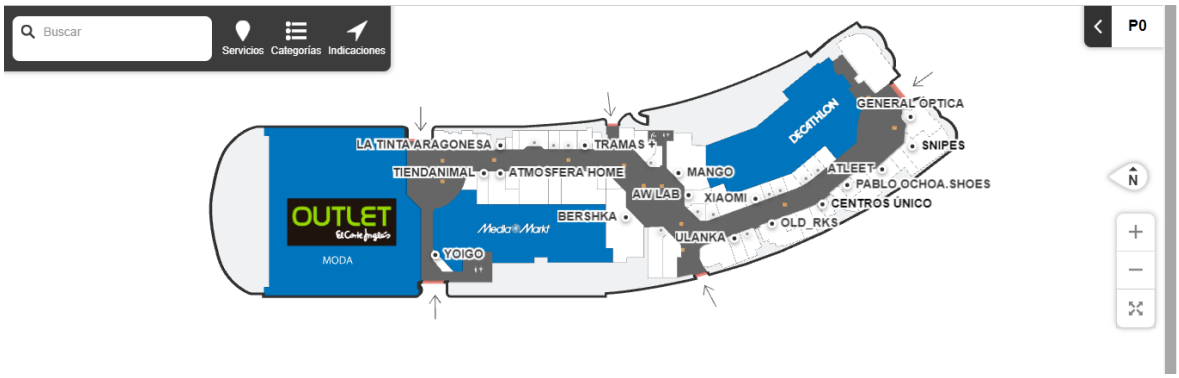


Figura F.1: Mapa del sitio oficial de *Grancasa* - Planta 0

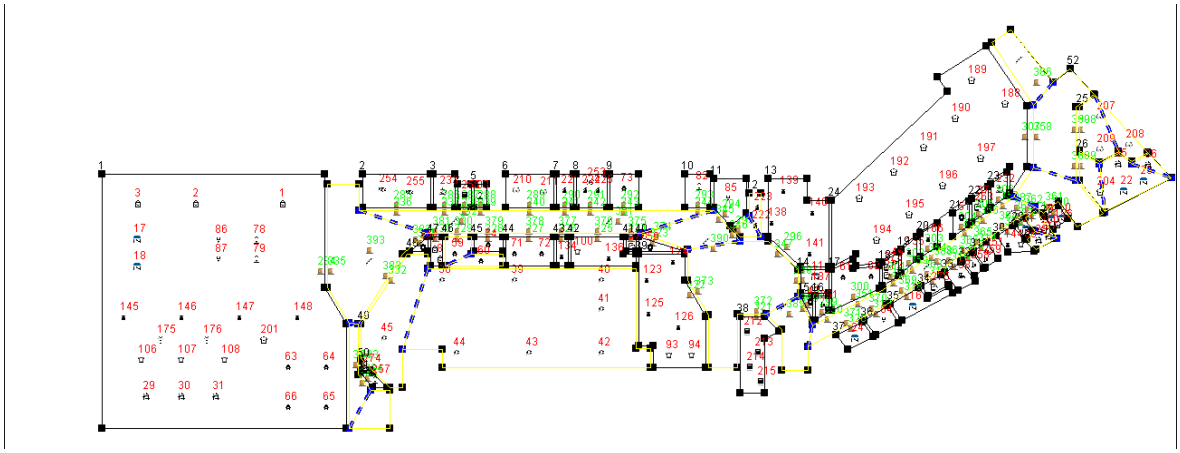


Figura F.2: Mapa de *Grancasa* - Planta 0

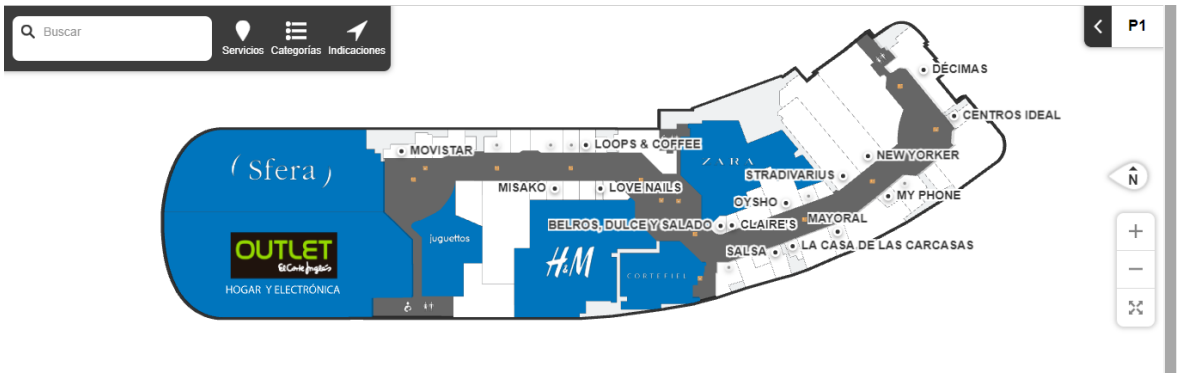


Figura F.3: Mapa del sitio oficial de *Grancasa* - Planta 1

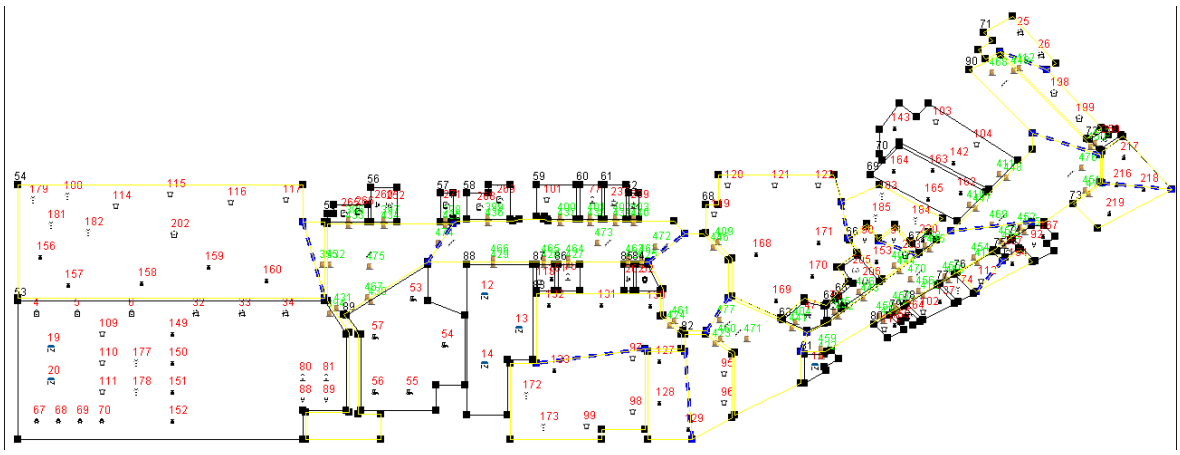


Figura F.4: Mapa de *Grancasa* - Planta 1

Figura F.5: Mapa del sitio oficial de *Grancasa* - Planta 2

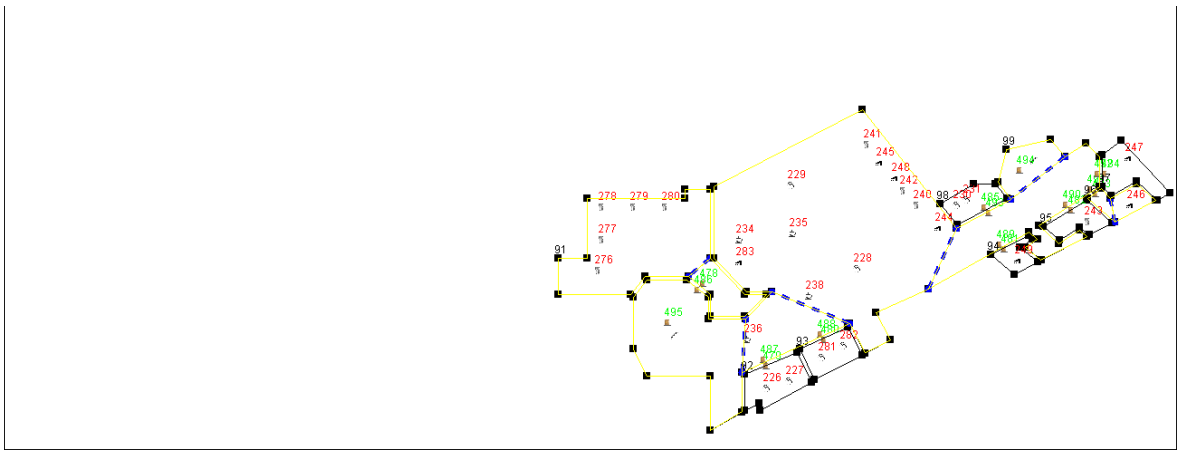


Figura F.6: Mapa de *Grancasa* - Planta 2

F.2. Mapa Campus Río Ebro

Para comprobar el alcance de esta herramienta se crea un escenario al aire libre. La idea es simular un evento social, desarrollado al aire libre pero conteniendo también edificios, y en el que los ítems visitables sean personas que se moverán durante la simulación (ítems móviles).

El escenario creado se basa en el Campus Río Ebro, de la Universidad de Zaragoza. El caso de estudio a simular es una jornada de intercambio, en la cual varios expertos de diferentes lugares se reparten por el campus y se van desplazando compartiendo sus conocimientos con aquellos alumnos que lo deseen. En este caso, los expertos se interpretan como ítems visitables y los alumnos como los usuarios de la simulación.

Para la construcción de este escenario se han consultado dos fuentes, la imagen del campus encontrada en la página de la Escuela Universitaria de Ingeniería Técnica Industrial de Zaragoza (`http://euitiz.unizar.es/es/betancourt.php?seccion=situacion`) y la herramienta *Google Maps* (`https://www.google.es/maps/`). Dado que no se encuentran medidas en la primera fuente, para la elaboración de este escenario se han tomado unas medidas aproximadas extraídas de *Google Maps* como muestra la Figura F.7.

La Figura F.8 muestra el mapa completo y las Figuras F.9, F.10 y F.11 muestran en mayor detalle cada una de las plantas. Estas figuras han sido extraídas del editor de mapas, sin utilizar el objeto divisor en subsalas para mayor claridad de las imágenes. Sin embargo, en caso de utilizar este escenario, habría que hacer uso de dicho objeto para el correcto movimiento de los usuarios y expertos durante la simulación.

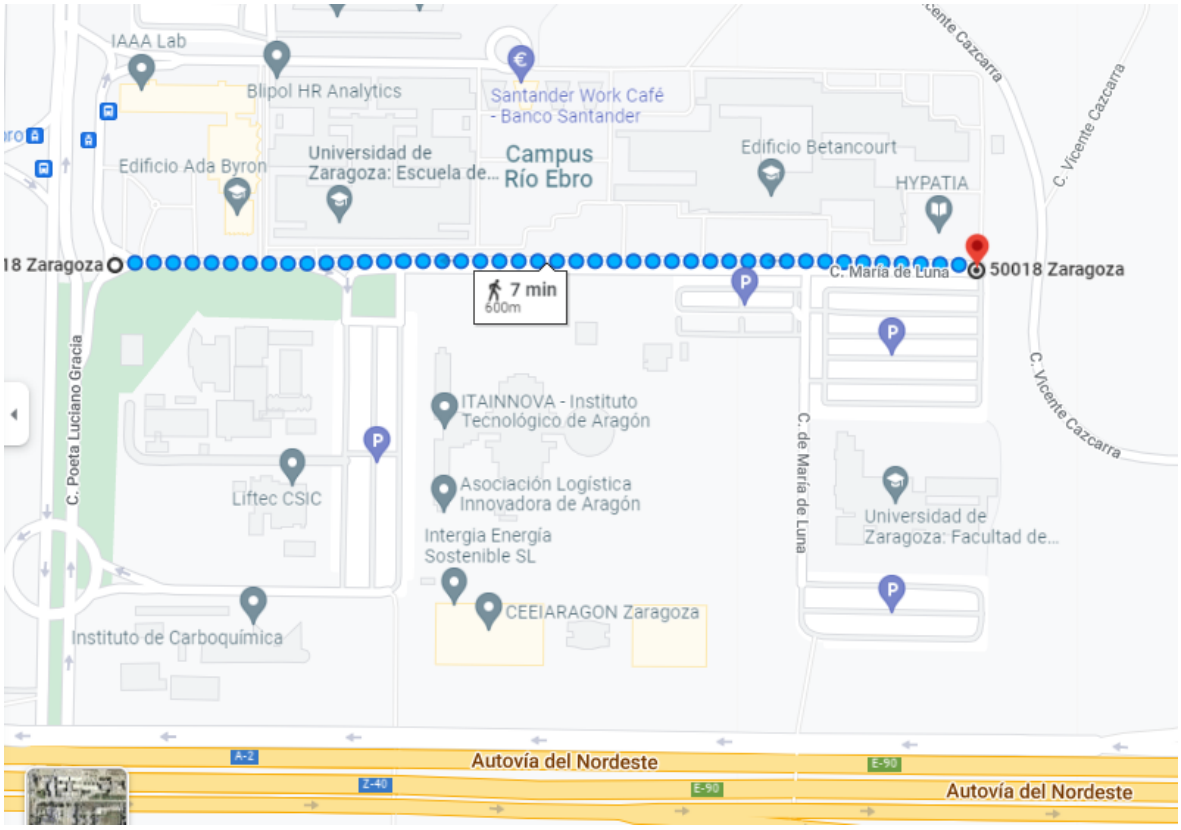


Figura F.7: Mapa Campus Río Ebro - Referencia *Google Maps*

En este escenario se observa el alcance y las limitaciones de este proyecto. Por un lado, sí es posible representar el mapa, como se puede observar en las figuras. Sin embargo, existen otro tipo de limitaciones de visualización o actuación.

Cuando se representa un escenario con elementos con dimensiones tan dispares (escenario al aire libre con edificios representados en menor dimensión), se encuentra un problema con la escala. Si el mapa se representa como en las figuras de este apartado, es complicado representar las salas del interior de los edificios, pues su tamaño sería demasiado pequeño y los ítems contenidos en ellas se solaparían. Por otro lado, si se amplía el mapa para poder representar el interior de los edificios, el mapa creado tendría un tamaño muy grande, dificultando la visión en el simulador.

Por otro lado, se encuentran ítems con comportamientos propios más amplios que los atributos asignados a los tipos de ítems que se pueden crear con el editor (Anexo (E). Por tanto, sería necesario extender la clase ítem y añadir el comportamiento requerido (trayectorias) y su posibilidad de creación en el editor. Por otro lado, habría que reconocer y tratar también este comportamiento en el simulador como el movimiento los usuarios de la simulación (moviendo también los ítems con trayectoria).

Figura F.8: Mapa Campus Río Ebro - Escenario Completo

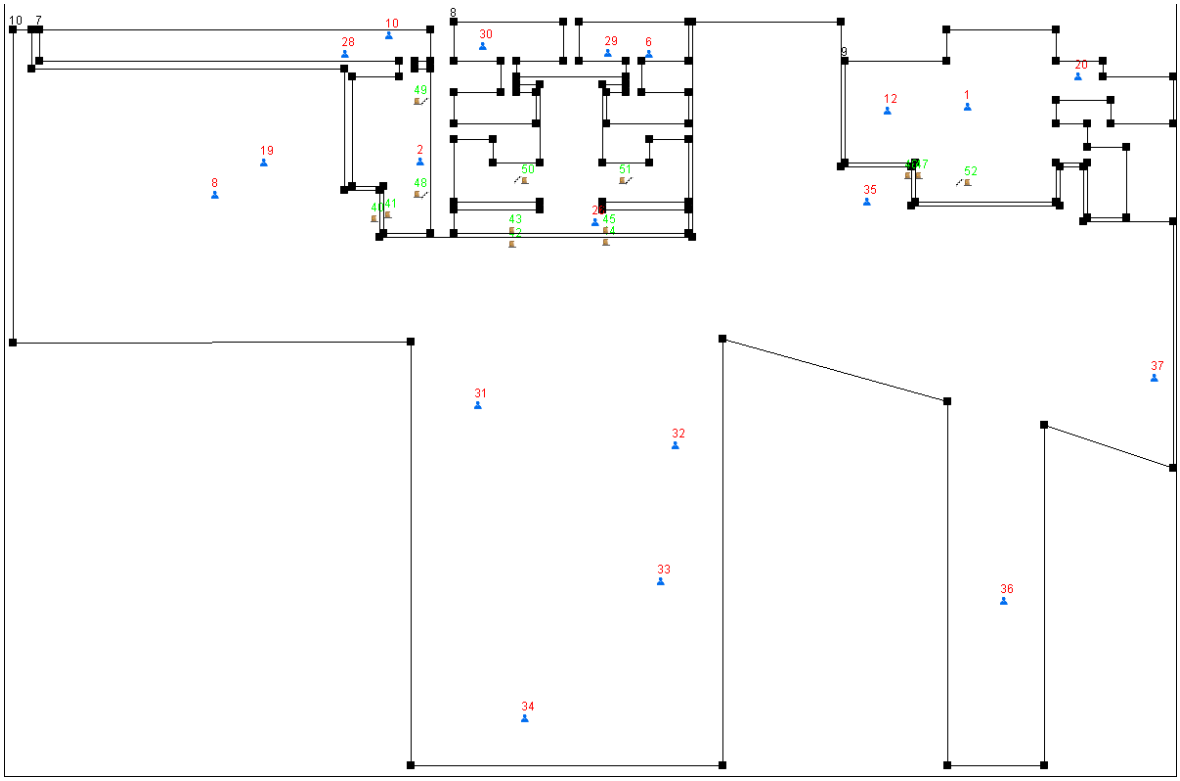


Figura F.9: Mapa Campus Río Ebro - Planta calle

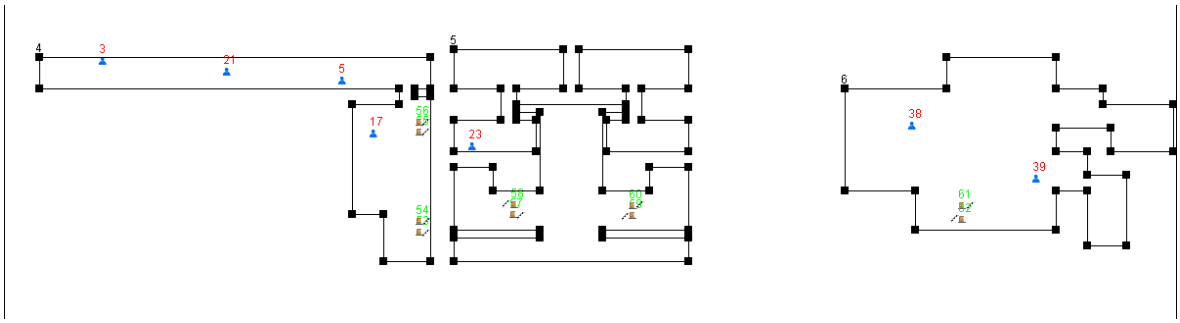


Figura F.10: Mapa Campus Río Ebro - Primera planta (edificios)

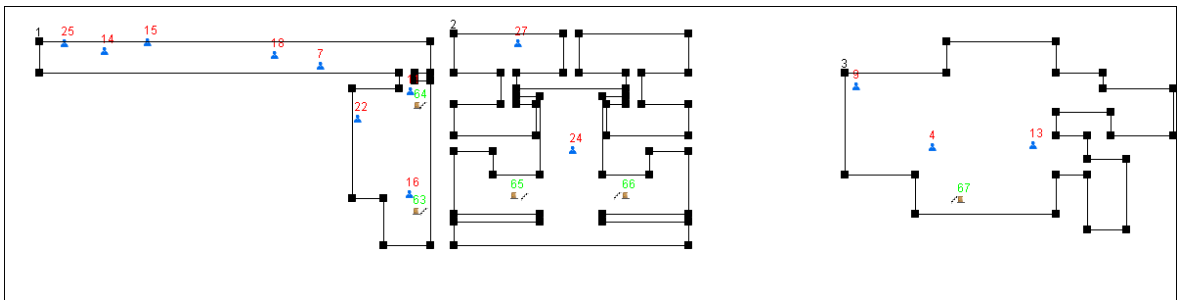


Figura F.11: Mapa Campus Río Ebro - Segunda planta (edificios)

Anexos G

Comparativa de la generación de datos y el rendimiento del escenario respecto al proyecto original

En este Anexo se amplía y detalla el proceso de generación de datos para la realización de pruebas sobre el escenario de *GranCasa* en el nuevo simulador, tras su modificación y generalización (Anexo G.1) y se extiende la comparativa de rendimiento respecto al proyecto original y entre los escenarios disponibles (Anexo G.2).

G.1. Proceso de generación de datos para el escenario de *GranCasa*

El último paso para poder probar CARS una vez generalizado el simulador y creados los mapas con el editor es obtener los datos referentes al nuevo escenario creado. La base de datos utilizada en la simulación de un escenario posee la información de todos los ítems que contiene, de todos los posibles contextos que se basan en diversos factores (condiciones de la sala, estado de ánimo de cada usuario, etc.), los posibles usuarios (tanto su información personal como su perfil de usuario, es decir, sus tipo de gusto) y de todas las posibles valoraciones, una por cada combinación de usuario, ítem y contexto posibles (producto de las tres cantidades). Los usuarios con un mismo perfil proporcionarán valoraciones similares en los mismos contextos a los mismos ítems durante la simulación, siendo muy influyentes sus valoraciones.

La obtención los datos comentados en el párrafo anterior (requeridos para poder llevar a cabo la simulación) puede realizarse de diferentes formas, pues siempre que se generen todos los registros de todas las tablas mencionadas anteriormente con todas las posibles combinaciones en el caso de las valoraciones, el simulador puede utilizar la información sin importar su origen. Se consideran las diferentes posibles formas de obtención de datos:

– Generación de datos mediante *AUTO-DataGenCARS*, herramienta para la

generación de datos sintéticos para la evaluación de CARS y opción recomendada para nuevos escenarios.

- Obtención mediante programas/scripts externos.

- Disposición previa de los datos, como en el caso de simulación en las plantas 4 y 5 del museo MoMA (aunque estos fueron generados con *DataGenCARS*).

Sin embargo, la calidad de los resultados y la actuación de los sistemas de recomendación dependen de la calidad de los datos, por lo que a pesar de ser posible su generación por otros medios, se recomienda el uso de DataGenCARS (en caso de no disponer de datos reales previos). El uso de esta herramienta o bien de *AUTO-DataGenCARS* (extensión con interfaz gráfica), sin embargo, no es trivial.

En primer lugar, para la generación de datos de valoraciones utilizando datos existentes (en este caso los ítems del nuevo escenario exportados a formato *CSV*), se requieren todos los tipos de ficheros de entrada (usuarios, ítems, contextos, perfiles de usuarios y valoraciones) incluido por tanto el propio fichero de valoraciones (el cual se desea generar); así como los esquemas pertenecientes a los usuarios, ítems y contextos, los cuales se deben de crear de forma independiente aunque se disponga de los datos.

Además, la generación de datos no se aplica de la forma deseada, pues el número de valoraciones especificadas (cuyo valor introducido es el producto de los usuarios, ítems y contextos) no coincide con el número de registros únicos (tuplas usuario-ítem-contexto-valoración) generados por la herramienta, repitiéndose algunos y dejando sin completar otros.

Como consecuencia, los resultados obtenidos no eran suficientes para llevar a cabo la simulación del escenario, además de obtener valoraciones pésimas (para valoraciones cuyo valor puede ser de uno a cinco, los valores obtenidos oscilaban entre uno y tres, teniendo la mayoría de valoraciones el valor más bajo). Se precisó la ayuda de expertos en la herramienta para la mejor comprensión sobre esta y para la correcta generación del conjunto de datos utilizado en el nuevo escenario de prueba.

Durante el tiempo que los datos generados con esta herramienta eran incorrectos, se trató de elaborar un "script" que generase las valoraciones necesarias para la simulación. Este programa generaba todas las tuplas necesarias (no logrado en un comienzo con *DataGenCARS*) y aplicaba una distribución de probabilidades en la

asignación de valoraciones equivalente a las encontradas en la base de datos del museo. Los datos obtenidos mediante este método eran pésimos, provocando que no hubiera diferencia entre los algoritmos básicos e ideales e incluso siendo mejores los primeros en algunos casos.

Por ello, a pesar de los inconvenientes encontrados, se sigue considerando *DataGenCARS* como herramienta principal para la generación de datos. Además, se cree que la reducción de atributos de los ítems también puede dificultar la generación de datos de mayor calidad por parte de esta herramienta al disponer de menos campos (en los ítems) en los que basarse en el proceso de generación de valoraciones.

No se dispone de demasiada información sobre el proceso de generación de datos para el simulador original (del museo), pero sí se mantiene en común la herramienta utilizada (*DataGenCARS*). Es por ello que se demuestra que, a pesar de su complejidad inicial, esta herramienta es la más idónea para la generación de datos con los que evaluar *CARS* y se recomienda su uso para este propósito tanto en este como en futuros proyectos que puedan extenderlo y profundizar más en este campo de investigación.

G.2. Evaluación de rendimiento tras las optimizaciones y costes en tiempo respecto al proyecto original

Como ya se ha comentado previamente, el proyecto original tomaba tiempos insostenibles para llevar a cabo simulaciones y por tanto poder obtener resultados con el equipo utilizado. Este equipo es un ASUS X555LJ, con sistema operativo Windows 10 Home 64 bits, procesador Intel(R) Core(TM) i3 (4 núcleos, 1.7GHz) y 8GB de RAM.). Debido a este aspecto, se considera fundamental llevar a cabo una comparativa de rendimiento respecto al proyecto original.

En un comienzo, todas las pruebas con distintos algoritmos y configuraciones tuvieron que ser abortadas manualmente (las más longevas tras más de 24 horas). Tras las optimizaciones realizadas al simulador (Sección 4.1), se consiguió que finalmente que la proporción "tiempo de simulación/tiempo real" se respetase. Esto permitió que ejecución cuyo tiempo simulado es de una hora terminasen en una hora real (o en escasos minutos en función del valor del campo comentado). Por ejemplo, al poner el valor 30 en este parámetro de la configuración, las simulaciones se ejecutan a velocidad "x30", consiguiendo finalizar simulaciones de una hora en dos minutos.

Existe además un pequeño tiempo de arranque previo al comienzo de la simulación, el cual se necesita para leer e interpretar el escenario, crear las conexiones con las bases de datos necesarias para el experimento (en función de su configuración) y crear los caminos de los usuarios que no disponen de recomendador. Cuanto mayor es el escenario (dimensión y número de objetos que contiene), mayor es este tiempo (más propiedades a leer y elementos a representar en la GUI). Además, si se usa la red *P2P*, este tiempo se incrementa, pues se han de crear todas las bases de datos tanto de valoraciones como de la cola P2P para todos los usuarios.

Se implementó además la posibilidad de realizar evaluaciones desatendidas. Estas permiten realizar experimentos por lotes y sin GUI, encadenando varias simulaciones y extrayendo sus resultados. Sin embargo, a pesar de haber reducido drásticamente el número de operaciones I/O (mensajes impresos principalmente), sí que se observa un pequeño descenso del rendimiento con el paso del tiempo y número de simulaciones encadenadas. A pesar de ser evaluaciones desatendidas, se informa al usuario por la consola de la interfaz del tiempo transcurrido, para que cuando regrese a ver el estado de los experimentos pueda tener un feedback de la evolución y comprobar que no se ha entrado en estado de inanición.

Finalmente, explicados todos los costes de tiempos de cada parte, es posible ejecutar **los 8 algoritmos recogidos en las gráficas** (realizando 2 experimentos para P2P con distinto TTL) en una hora aproximadamente aplicando velocidad de simulación x30. Sin embargo, para obtener mejores resultados de los algoritmos de recomendación que tienen en cuenta las valoraciones del resto de usuarios (*UBCF* y *Know-It-All*), es preferible reducir la velocidad de simulación para el mayor intercambio de información entre usuarios (más ciclos de ejecución). En la Tabla 6.3 se estudian los tiempos obtenidos para cada prueba, realizando pruebas de ejecuciones únicas y pruebas de todos los algoritmos (8 simulaciones, 7 algoritmos pero dos experimentos para *UBCF - P2P* con distinto "TTL").

Anexos H

Resultados obtenidos con el nuevo simulador para el mapa del museo MoMA

En este Anexo se presentan los resultados del desempeño de los CARS con el nuevo simulador para el escenario inicial, las plantas 4 y 5 del museo *MoMA*. Estos experimentos se realizaron tras la optimización inicial del simulador, explicada en la Sección 4.1, a modo de punto de control para corroborar que las mejoras incorporadas no afectaban al rendimiento del simulador (comparando los resultados con los obtenidos en la Sección 3.3 de [1]). Finalmente, se repiten los experimentos tras la finalización del proyecto y se comprueba que todos los resultados son equivalentes, por lo que la generalización del simulador no disminuye su rendimiento.

Los parámetros de configuración utilizados para la recopilación de resultados en este escenario se mantiene idéntica a la utilizada en el artículo utilizado como referencia, para así poder establecer comparaciones y determinar si los resultados son coherentes.

Las Figura H.1 muestra las valoraciones medias de los votos que los usuarios con recomendador han proporcionado a los ítems que han visitado a lo largo de la simulación para cada algoritmo de recomendación utilizado. El valor que pueden tomar los votos de los usuarios es de 1 a 5. Los resultados obtenidos son prácticamente idénticos, salvo una ligera disminución en el algoritmo ideal. Continúa pareciendo que la diferencia entre la estrategia *ALL* y *P2P* no es muy significativa, pero además de sus mejores resultados, el valor añadido de la estrategia *P2P* se sigue observando en la comparación de los ítems de interés para los usuarios (Figura H.2).

La Figura H.2 muestra la diferencia entre "likes" y "dislikes" de los usuarios frente a los ítems visitados. Se definen como "likes" las valoraciones cuyo valor supera el umbral (de 3,5 de valoración), considerándose que los ítems con una valoración superior han

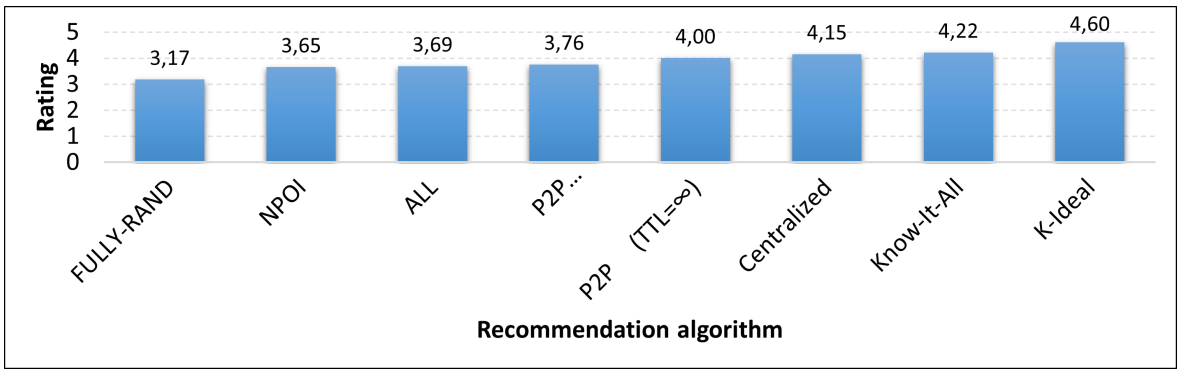


Figura H.1: Valoración media obtenida para los ítems observados - Museo MoMA

resultado de interés para los usuarios, y "dislikes" aquellas por debajo y que por tanto no han despertado interés en los usuarios.

Es en dicha figura donde se puede notar una mejora en la estrategia *P2P* frente a las estrategias simples, recomendando al usuario más ítems de su gusto. La diferencia "likes"-"dislikes" de esta estrategia la sitúa más próxima a los algoritmos ideales y óptimos que a los simples.

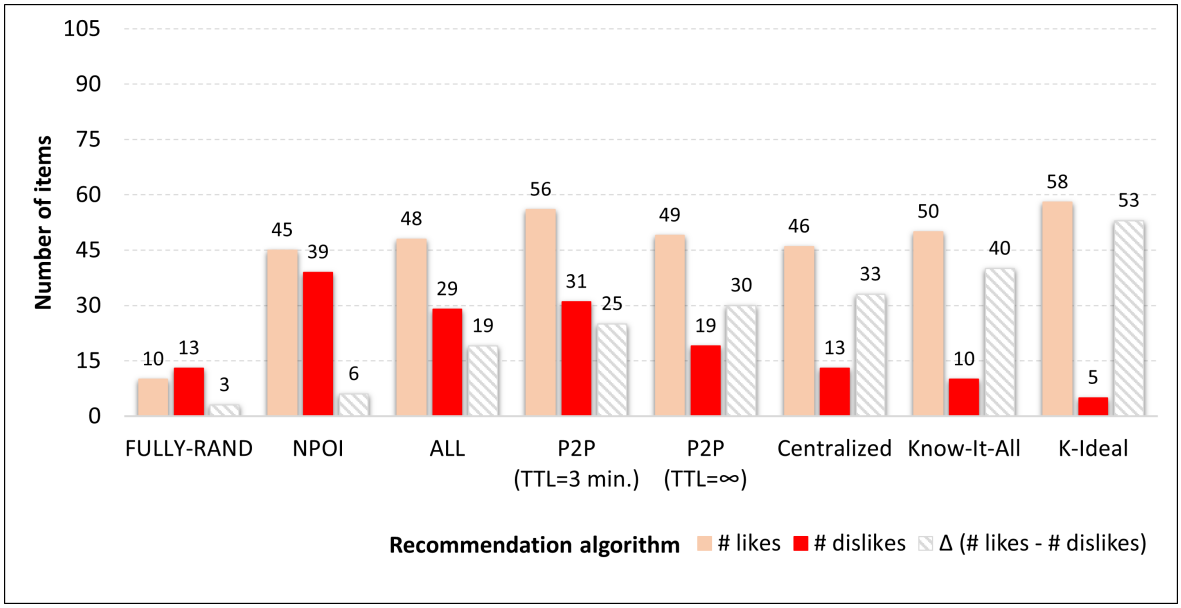


Figura H.2: Me gusta y no me gusta obtenidos (con umbral 3,5) - Museo MoMA

Finalmente, se analiza más en profundidad el algoritmo colaborativo que emplea la comunicación *P2P*. La Figura H.3 muestra la predicción de error media a lo largo del tiempo empleando un valor de TTL de 180 segundos. Se observa una disminución de la predicción de error a lo largo del tiempo, pero la línea de tendencia toma unos valores superiores a los del artículo previo. Además, el número de datos de error predicho recuperados para la elaboración de esta gráfica es notablemente menor al de la gráfica

original, pues se aplica el algoritmo *NPOI* cuando no se dispone de información para recomendaciones durante la simulación. Este algoritmo simple no hace uso de *Apache Mahout* y por tanto no produce estos valores predichos, encontrando así mayor separación entre algunos valores en la gráfica.

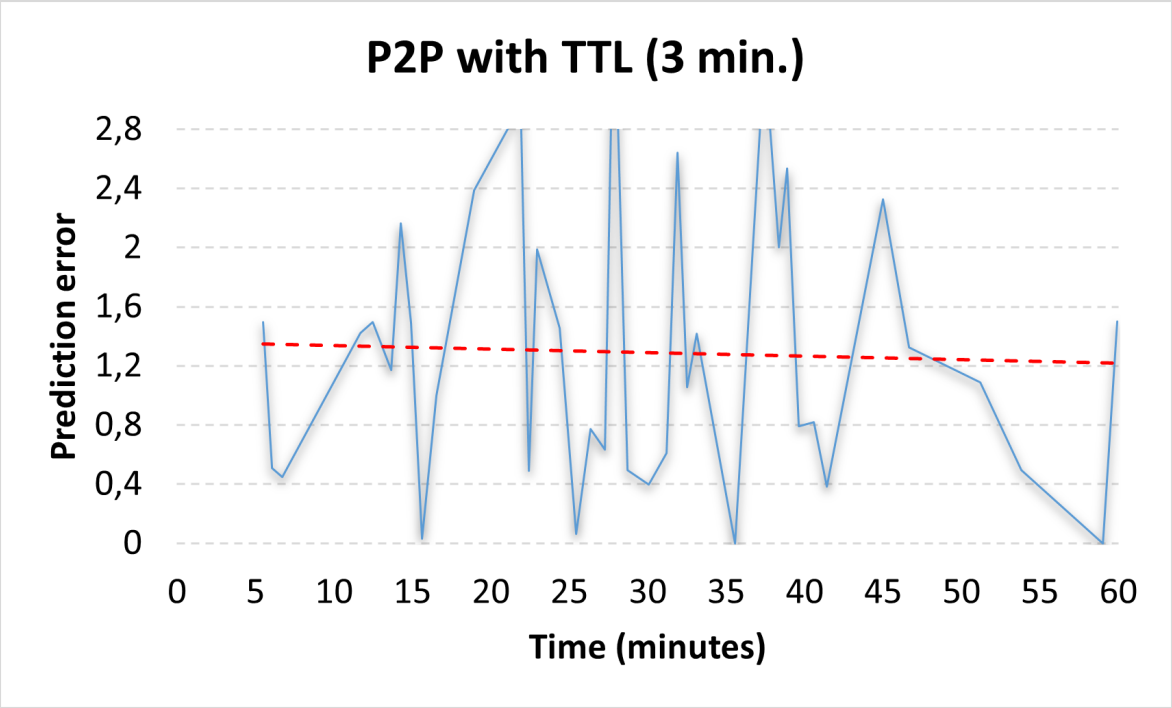


Figura H.3: Predicción de error a lo largo del tiempo - Museo MoMA

Cabe destacar que las gráficas mostradas en este Anexo son representativas de los experimentos realizados. Es decir, los datos mostrados en las gráficas representan los resultados medios de simulaciones realizadas para los diferentes algoritmos. Esto se debe a que, dada la aleatoriedad inherente a las simulaciones, los resultados pueden variar unas pocas décimas (del orden de 0,1/0,2 para cada algoritmo).