Trabajo Fin de Grado

Grado en ingeniería Informática

Creación de mundos virtuales en Minecraft a partir de datos geográficos y planos de CAD

Creating Minecraft virtual worlds based on geographic data and CAD blueprints

Autor

Jorge Sanz Alcaine

Director

Rubén Béjar Hernández

Universidad de Zaragoza

Escuela de Ingeniería y Arquitectura

2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Jorge Sanz Alcaine,

con nº de DNI 73020230Z en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Creación de mundos virtuales de Minecraft a partir de datos geográficos y planos de CAD,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 27 de septiembre de 2017

Fdo: Jorge Sanz Alcaine

Resumen

El objetivo de este proyecto es la creación de una aplicación que permita la creación de mundos Minecraft a partir de datos geográficos y planos de CAD.

La aplicación ofrece una serie de heurísticas que intentan maximizar el parecido de una zona con la realidad. Los desarrolladores deberán elegir aquellas transformaciones que mejor se adecuen a su problema para luego generar el mundo en formato Minecraft

En la aplicación podemos distinguir dos partes, EINA-TO-VOXELS y EINA-TO-NBT. EINA-TO-VOXELS se encarga de la lectura de datos geográficos y planos en CAD, y de aplicar heurísticas sobre el mundo virtual. EINA-TO-NBT se encarga de la conversión de los mundos a formato Minecraft.

Durante el procesamiento en EINA-TO-VOXELS, el mundo virtual se representa mediante una matriz tridimensional dispersa. Sobre esa matriz se aplican una serie de heuristicas y posteriormente se genera un fichero intermedio con un formato predeterminado que representa el mundo virtual. Ese fichero es interpretado por EINA-TO-NBT para su conversión a formato Minecraft.

Antes de que los planos de CAD puedan ser usados, es necesario un proceso manual para su transformación a estructuras de voxels. Este procesamiento se lleva a cabo con herramientas externas como QCAD, FREECAD o binvox.

La aplicación utiliza información de OpenStreetMap para la clasificación de algunas zonas del mundo virtual.

El desarrollo del proyecto se realizado principalmente entre los meses de Junio y Agosto, y se ha dividido en tres fases. Una primera fase para la elaboración de los requisitos y el estudio de las tecnologías. Una segunda fase para la elaboración de un primer prototipo de la aplicación. Y una fase final para el refinamiento del prototipo y para la elaboración de la documentación.

A pesar del tiempo dedicado aún queda trabajo que podría realizarse, siendo el principal la optimización de la aplicación mediante la división en zonas menores o incluso su paralelización.

Índice

1 Introducción

En este documento se describe el proceso de realización y los resultados obtenidos en el trabajo de fin de grado titulado "Creación de mundos virtuales en Minecraft a partir de datos geográficos y planos de CAD"

1.1 Minecraft

Minecraft [17] es un videojuego creado en 2009 por Markus Alexej Persson y más adelante por el equipo de Mojang. Para febrero de 2017, Minecraft ya había vendido más de 121 millones de copias convirtiéndolo en el videojuego más vendido de la historia en PC. Además de para PC, Minecraft tiene versiones para PS3, XBOX 360, Android e IOS.

Es un juego de mundo abierto, por lo que no posee un objetivo específico, permitiéndole al jugador una gran libertad en cuanto a la elección de su forma de jugar. El juego se centra en la colocación y destrucción de bloques, siendo que este se compone de objetos tridimensionales cúbicos. Los jugadores son libres de desplazarse por su entorno y modificarlo mediante la creación, recolección y transporte de los bloques que componen al juego. Al inicio del juego, el jugador se encuentra en un mundo generado mediante un algoritmo, lo que permite que el mundo sea infinito y nunca se generen dos iguales.

1.2 Objetivo

El objetivo del proyecto es el desarrollo de una aplicación que permita la creación de mundos Minecraft a partir de datos geográficos y planos en CAD [6].

Aunque la aplicación puede utilizarse para cualquier zona que disponga de datos geográficos suficientes, la mayoría de las pruebas se han realizado sobre el área en la que se encuentra la universidad de Zaragoza, ya que es la única zona que disponemos de planos de CAD.

La aplicación está destinada principalmente a usuarios con conocimientos informáticos. Ya que las operaciones necesarias para la transformación a formato Minecraft pueden ser diferentes dependiendo de la zona en la que se realicen. Los desarrolladores deberán elegir aquellas transformaciones que mejor se adecuen a su problema para luego generar el mundo en formato Minecraft. Se establecerán opciones por defecto para facilitar la creación de mundos a aquellos usuarios que no tengan estos conocimientos.

Con la llegada de Minecraft, se ha creado una nueva forma de arte que consiste en la construcción de edificios o esculturas con bloques. Esta aplicación permite la generación de zonas mucho más amplias de las que podrían construirse a mano. Muchos jugadores de Minecraft desearían poder empezar una partida en su ciudad natal o en diversos lugares conocidos del mundo.

Otro posible uso de la aplicación es para proyectos de urbanismo, dado que es posible introducir construcciones y ver como quedarían en una zona específica sin la necesidad de construir una maqueta.

1.3 Proyectos similares

La creación de mundos virtuales parecidos al nuestro no es una idea reciente. Existen varias aplicaciones que, con algunas diferencias, han tratado de realizar lo mismo que en este proyecto. A continuación, se describirán algunas de estas aplicaciones poniendo especial interés en sus puntos fuertes y débiles.

Geoboxers [28]

Es un servicio para la creación de mundos virtuales en Minecraft a partir de datos geográficos y geoespaciales. Al igual que en este proyecto, Geoboxers utiliza datos LIDAR y OpenStreetMap para la generación de mundos virtuales. Para los colores utiliza datos vectoriales principalmente.

- Puntos fuertes: Es una empresa con varios años de recorrido y han hecho recreaciones de algunas ciudades importantes del mundo. Permiten la utilización de gran variedad de datos geográficos.
- Puntos débiles: No permite el uso de planos de CAD, por lo que todos los edificios son huecos y están pensados para verse desde fuera. Los datos vectoriales suelen omitir detallen que si se aprecian en otros tipos de datos como las imágenes

Figura 1: Ciudad de Viborg versión Minecraft

Baidu Maps [7]

Baidu es el buscador más utilizado de China, y al igual que Google también tienen un servicio de mapas. Algunas de los datos que utiliza el servicio de mapas de Baidu utiliza proceden de OpenStreetMap, MapKing o Here. Para su visualización en 3D, Baidu utiliza un estilo similar a "Los Sims".

- Puntos fuertes: Baidu es una empresa importante con una gran cantidad de recursos y de personal. No se utilizan bloques, por lo que las formas y los colores resultan más realistas.

- Puntos débiles: Tan solo los edificios están representados de forma tridimensional, el suelo sigue siendo plano. Además, la vista es fija y tan solo se puede aumentar o reducir el zoom, por lo que no es posible observar la ciudad a nivel del suelo.

Figura 2: Área comercial de la ciudad de Shanghái

Binvox [29]

Binvox es una aplicación para la lectura de modelos en tres dimensiones y su rasterización a estructura de voxels en un fichero. Esta herramienta se va a utilizar en el proceso de inserción de edificios en el mundo virtual.

- Puntos fuertes: Permite la rasterización de objetos Wavefronts [33] o incluso de fichero DXF de AutoCAD siempre que solo contengan polígonos. Software libre.
- Puntos débiles: No está disponible en Python, por lo que no puede integrarse completamente en la aplicación. No tiene un soporte tan grande como las dos anteriores.

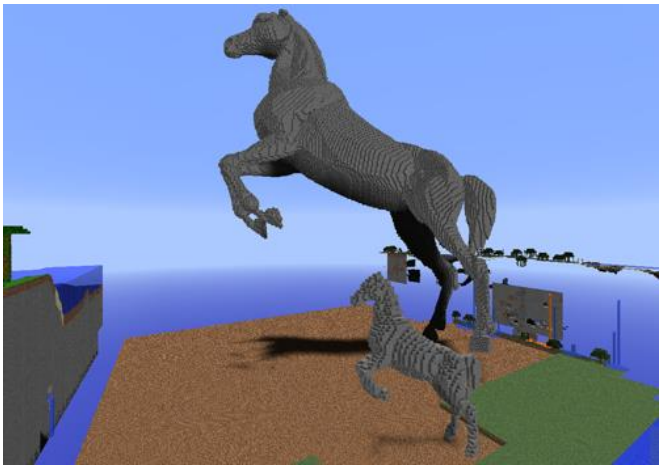


Figura 3: Modelos tridimensionales insertados en Minecraft

1.4 Estructura del documento

Este documento se ha dividido en 6 apartados diferentes y los anexos.

- Introducción: En este apartado se explica en qué consiste el proyecto, cuál es su objetivo y describe Minecraft para aquellos que no lo conozcan.
- Análisis: En este apartado se detalla cuáles son los requisitos del proyecto, a que público va dirigido y cuáles son los datos disponibles.
- Diseño de la solución: En este apartado se describen algunas vistas que reflejan el diseño de la aplicación con un alto nivel de abstracción.
- Implementación de la solución: Este apartado describe la aplicación a un bajo nivel de abstracción. Algunos de los temas que trata son las herramientas utilizadas los problemas encontrados y sus soluciones.
- Pruebas: En este apartado se especifican cuáles han sido las pruebas realizadas para validar los resultados y se realiza una medición de las prestaciones de la aplicación
- Gestión del proyecto: En este apartado se muestra la planificación inicial, el control de esfuerzos, un análisis de riesgos y la gestión de configuraciones de la aplicación.
- Conclusiones: en este apartado se analizan los resultados de los apartados anteriores para decidir si el trabajo ha sido satisfactorio y se realiza una valoración personal sobre el proyecto.

2 Análisis

En este apartado se describen los datos disponibles, el procesamiento de estos datos para que puedan ser utilizados y el formato del resultado de la aplicación. También se especifican los requisitos funcionales y los no funcionales.

2.1 Datos de partida

Los datos de partida tienen un papel fundamental en el resultado final. Es mucho más sencillo obtener resultados realistas si se dispone de datos completos y variados.

Como se ha dicho anteriormente, la mayoría de las pruebas se realizarán sobre la zona del EINA y, por tanto, es necesario comprobar los datos disponibles para esa zona.

El CNIG [11] (centro nacional de información geográfica) ofrece sus datos de forma libre y gratuita, siempre y cuando se reconozca su origen. Es de esta fuente de donde se ha obtenido prácticamente toda la información geográfica necesaria para las pruebas. Entre los datos del CNIG se encuentran diferentes tipos de mapas, algunos de ellos no son de utilidad para este proyecto, mientras que otros sí que lo son. Dado que el proyecto tiene una duración determinada, no hay suficiente tiempo como para incluir todos los tipos de datos en el proyecto, por lo que se ha decidido utilizar únicamente datos LIDAR [14] y Ortofotos [20].

Un LIDAR es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie. En este caso se ha utilizado para obtener una nube de puntos desde el aire. Cada uno de los puntos es una colisión del láser con la superficie. Estos datos se encuentran en formato .laz.

Un Ortofoto es una fotografía aérea de una zona en la que todos los elementos presentan la misma escala.

A pesar de que el CNIG dispone de datos vectoriales, estos no son muy completos y no están demasiado actualizados, por lo que se ha decidido utilizar OpenStreetMap [19] para los datos vectoriales. OpenStreetMap es un proyecto colaborativo para crear mapas libres y editables.

Por último, se van a utilizar los planos del EINA que han sido facilitados desde la UTCE (Unidad Técnica de Construcciones y Energía). Estos datos se encuentran en formato .dwg, uno de los formatos utilizados en AutoCAD.

Los datos del CNIG utilizan un sistema de coordenadas UTM [31]. UTM divide la tierra en 60 husos de 6º de longitud. Cada huso tiene asignado un meridiano central, que es donde se sitúa el origen de coordenadas, junto con el ecuador. Para facilitar la introducción de coordenadas se utiliza solo el huso 30, que es donde se encuentra España. Si el alcance de la aplicación aumentase, podría modificarse para que se pueda utilizar en el resto del mundo.

2.2 Procesamiento de datos CAD

Para que la aplicación pueda utilizar los planos CAD, es necesario procesarlos hasta generar estructuras de voxels [24]. Un vóxel constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del pixel en un objeto 2D.

Este procesamiento podría haberse integrado con la aplicación, sin embargo, esto hubiera supuesto dedicar menos tiempo a otros aspectos importantes. Como ya existen herramientas que realizan este tipo de transformaciones se decidió utilizarlas.

Los elementos de un plano en formato .dwg se dividen en capas. Cada capa tiene un conjunto de polígonos que representan las paredes, los suelos, etc. Algunas de estas capas resultaban innecesarias o incompletas, por lo que fue necesario procesar los datos antes de poder usarlos. Este procesamiento se llevó a cabo de forma manual con las herramientas QCAD y FREECAD.

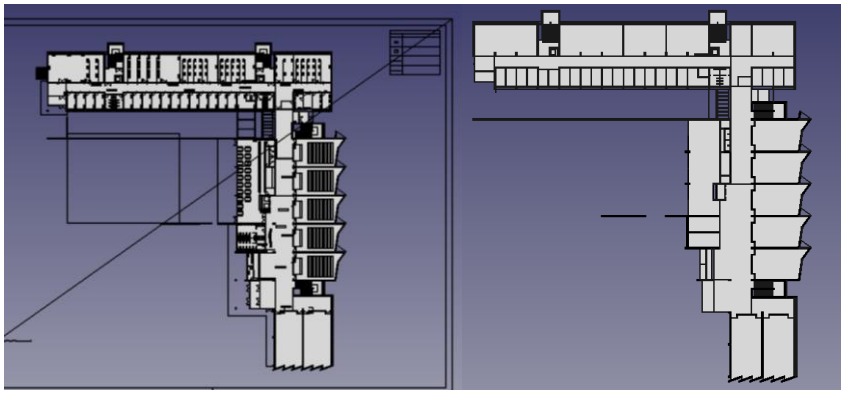


Figura 4: Comparativa limpieza inicial.

Tras una primera limpieza se realizó la transformación a 3D. FREECAD permite extruir polígonos de dos dimensiones para darles volumen. Se agruparon las capas que contenían las paredes para extruirlas y así conseguir muros de 3 metros de altura. Al extruir una capa, la altura seleccionada es la misma para todos los elementos de la capa, obtener las escaleras de esta forma hubiera sido bastante costoso. Por esta razón, se decidió eliminar las escaleras para crearlas de nuevo con el servicio de creación de escaleras de FREECAD. En los planos originales, las puertas se representan como una línea y un arco. Si se extruyesen las puertas, las habitaciones quedarían cerradas, por lo que se decidió eliminarlas para dejar así huecos en las entradas.

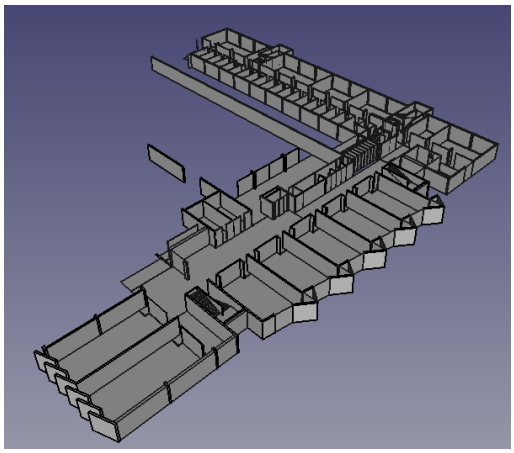


Figura 5: Modelo tras extrusión de las paredes

Una vez se han realizado las transformaciones a 3D en todos los pisos del edificio hay que juntarlos en un solo edificio. Todos los planos se encuentran al nivel del suelo, por lo que hay que cambiar la altura de cada una de las plantas. El suelo de cada planta es el techo de la anterior, salvo en el último piso donde es necesario duplicar el suelo para subirlo a la altura del techo.

Para la transformación del modelo en 3D a una estructura de voxels, se ha utilizado la herramienta binvox. Binvox transforma modelos tridimensionales en formato Wavefront a estructuras de voxels en formato binvox. Wavefront es uno de los formatos a los que FREECAD permite exportar un proyecto.

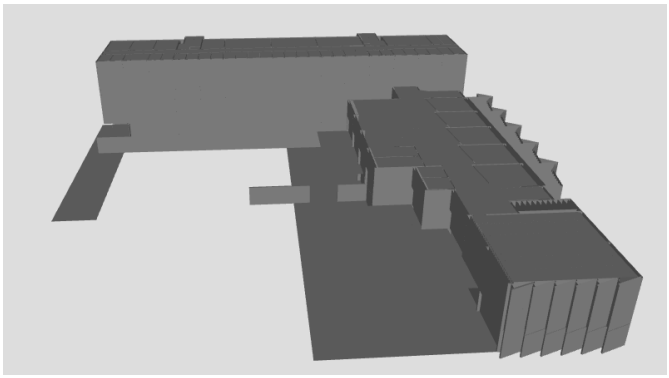


Figura 6: Modelo del edificio completo en formato Wavefront

Dado que la herramienta binvox está preparada para ejecutarse con Windows y las pruebas se han realizado desde Linux, se ha utilizado el emulador Wine [36]. A continuación, se muestra el comando utilizado con Wine para la transformación del edificio Ada Byron.

```
Wine binvox.exe -e -d 220 -down adaByron.obj
```

El parámetro -e indica que cualquier voxel que contenga parte del edificio, aunque esa parte se pequeña, se marca como edificio. El parámetro -d especifica el tamaño del edificio en voxels. El parametro -down transforma el resultado en otra estructura de la mitad del tamaño original.

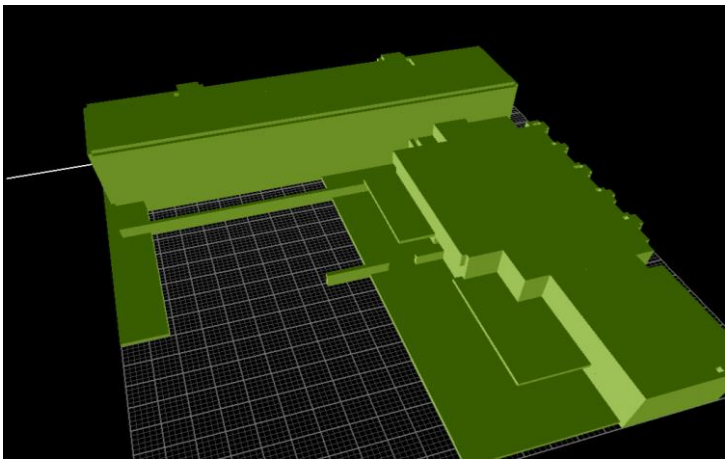


Figura 8: Edificio en formato binvox

2.3 Formato de salida

El resultado de la aplicación es un mundo al que se puede jugar directamente en Minecraft.

Minecraft guarda sus mundos como carpetas dentro del directorio “Worlds”. Para que Minecraft pueda iniciar uno de estos mundos debe contener como mínimo un fichero llamado level.dat, que guarda los datos del jugador. Los mapas se crean de forma procedural, por lo que, si no hay ningún mapa, se generarán automáticamente. Si se incluye un mapa, Minecraft lo utilizará y seguirá creando zonas cuando el jugador salga de sus dimensiones.

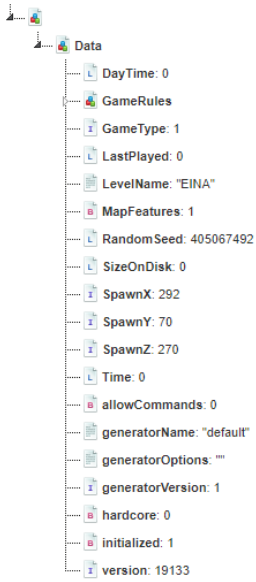
NBT es el formato más usado para almacenar ficheros en Minecraft. En este formato, un dato se almacena como una estructura de árbol compuesta por varios TAGS.

Un TAG contiene siempre su tipo, el nombre que identifica a el dato y dato en sí. Existen diferentes tipos de datos:

1. TAG_End
2. TAG_Byte
3. TAG_Short
4. TAG_Int
5. TAG_Long
6. TAG_Float
7. TAG_Double
8. TAG_Byte_Array
9. TAG_String
10. TAG_List
11. TAG_Compound
12. TAG_Int_Array

Un TAG empieza siempre por su tipo, seguido de dos bytes para indicar el tamaño del nombre, luego el nombre como un String en UTF8 y por último el payload que es el dato en sí y que depende del tipo elegido. Los TAG de tipo Compund no tienen un tamaño específico para su payload, sino que son listas de TAGS que terminan con un TAG_End.

Un ejemplo de este formato es el fichero level.dat.



La imagen representa los datos que contiene un fichero level.dat, que sigue el formato NBT. Este fichero es necesario para que Minecraft pueda iniciar el juego y debe contener como mínimo el tipo de juego (GameType), el nombre (LevelName) y la semilla utilizada para generar el mundo (RandomSeed).

Figura 9: level.dat formato NBT

Para los mapas se utiliza un formato diferente, MCA [34].

En este formato el mapa de un mundo se divide en regiones dentro del directorio region. Las regiones tienen un nombre con formato “r.x.z.mca”, donde “x” y “z” son las coordenadas de la región. Una región está compuesta por 1024 chunks [35]. Un chunk es un área de tamaño 16x256x16 bloques. Los chunks se guardan en formato NBT.

2.4 Requisitos

A continuación, se muestran los requisitos funcionales y no funcionales de la aplicación

RF1	El sistema permitirá al usuario la creación de mundos virtuales en Minecraft a partir de nubes de puntos.
RF2	El sistema permitirá al usuario insertar edificios en el mundo virtual dadas sus coordenadas UTM (x,y,z)
RF3	El sistema permitirá al usuario seleccionar una serie de heurísticas a utilizar en el mundo virtual.
RF4	El sistema permitirá al usuario ejecutar las heurísticas seleccionadas previamente.
RF5	El sistema permitirá al usuario integrar datos de OpenStreetMap en el mundo virtual.
RF6	El sistema permitirá al usuario introducir una imagen para modificar los colores del mundo virtual.
RF7	El sistema permitirá al usuario exportar el mundo virtual a formato Minecraft y definir la localización.

RNF1	Se admitirá como mínimo nubes de puntos en formato. LAS
RNF2	Las imágenes introducidas deberán estar en formato jpg o png.

3 Diseño de la solución

En este apartado se muestran las vistas de la aplicación para comprender su arquitectura interna. En concreto, se ha incluido una vista de módulos, otra de componente y conector, otra de distribución y un diagrama de secuencia.

3.1 Vista de módulos

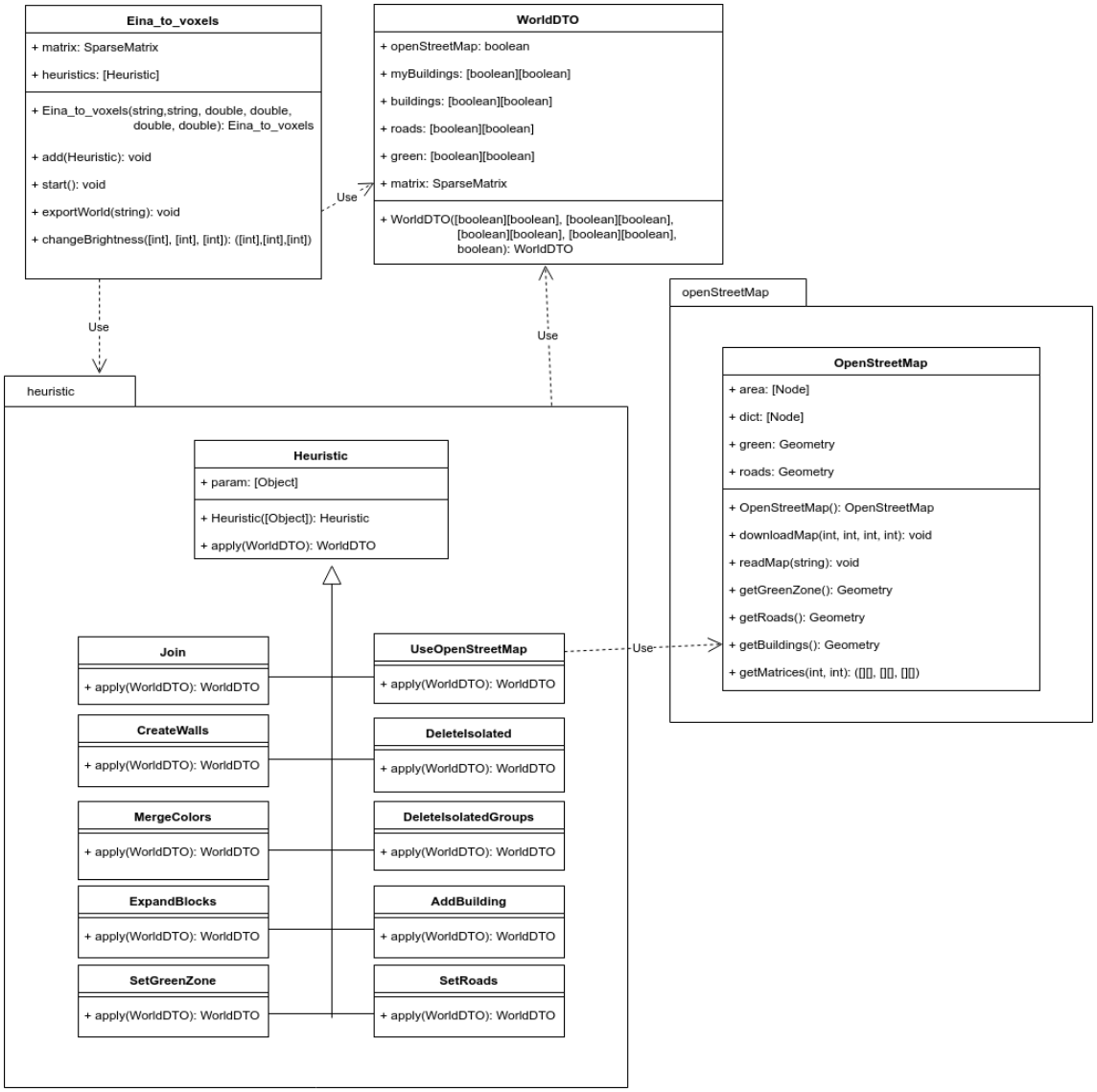


Figura 10: Diagrama de clases y paquetes

Como puede verse en el diagrama superior, la aplicación está formada principalmente por 4 módulos, la clase Eina_to_voxels, la clase WorldDTO, la clase OpenStreetMap y las heurísticas.

La clase Eina_to_voxels es la encargada de la creación y exportación del mundo virtual y de la elección de las heurísticas que se van a ejecutar. Eina_to_voxels sigue un patrón de Filtro-Y-Tubería [22] para la ejecución de las heurísticas. Almacena en una lista las heurísticas que se van a utilizar y cuando el usuario lo ordena se ejecutan una después de otra.

La clase WorldDTO representa toda la información relativa a un mundo virtual. Esta información incluye una matriz tridimensional y metadatos. Esta clase se utiliza para las comunicaciones entre la clase Eina_to_voxels y sus heurísticas. No contiene nada de la lógica del negocio, solo permite la lectura o modificación de los datos que contiene.

Una heurística es una clase con una operación que recibe como parámetro un objeto del tipo WorldDTO y lo devuelve tras realizar algunas modificaciones. Todas las heurísticas descienden de la clase heuristic y reciben en su constructor una lista con los parámetros para su operación. Cada una de las heurísticas implementa el método apply de forma diferente. La mayoría de las heurísticas modifican sólo la matriz tridimensional, sin embargo, UseOpenStreetMap utiliza la clase OpenStreetMap para modificar los metadatos a partir de datos de OpenStreetMap.

La clase OpenStreetMap es la encargada de obtener información de OpenStreetMap y procesarla para adaptarla al formato de Minecraft.

3.2 Vista de componentes y conectores

Suele ocurrir que aspectos de la implementación determinen el diseño de la aplicación y eso es lo que ocurre en este caso. El método exportWorld de la clase Eina_to_voxels debería crear directamente un mundo en formato Minecraft, pero la librería para trabajar con datos en formato NBT está en un lenguaje diferente al resto de la aplicación.

Por esa razón, se ha decidido dividir el sistema en dos aplicaciones, EINA-TO-VOXELS y EINA-TO-NBT.

EINA-TO-VOXELS se encarga de la lectura de datos geográficos y planos en CAD y de aplicar heurísticas sobre el mundo virtual para aumentar el parecido con la realidad.

EINA-TO-NBT se encarga de generar mundos Minecraft en formato NBT a partir de estructuras de voxels.

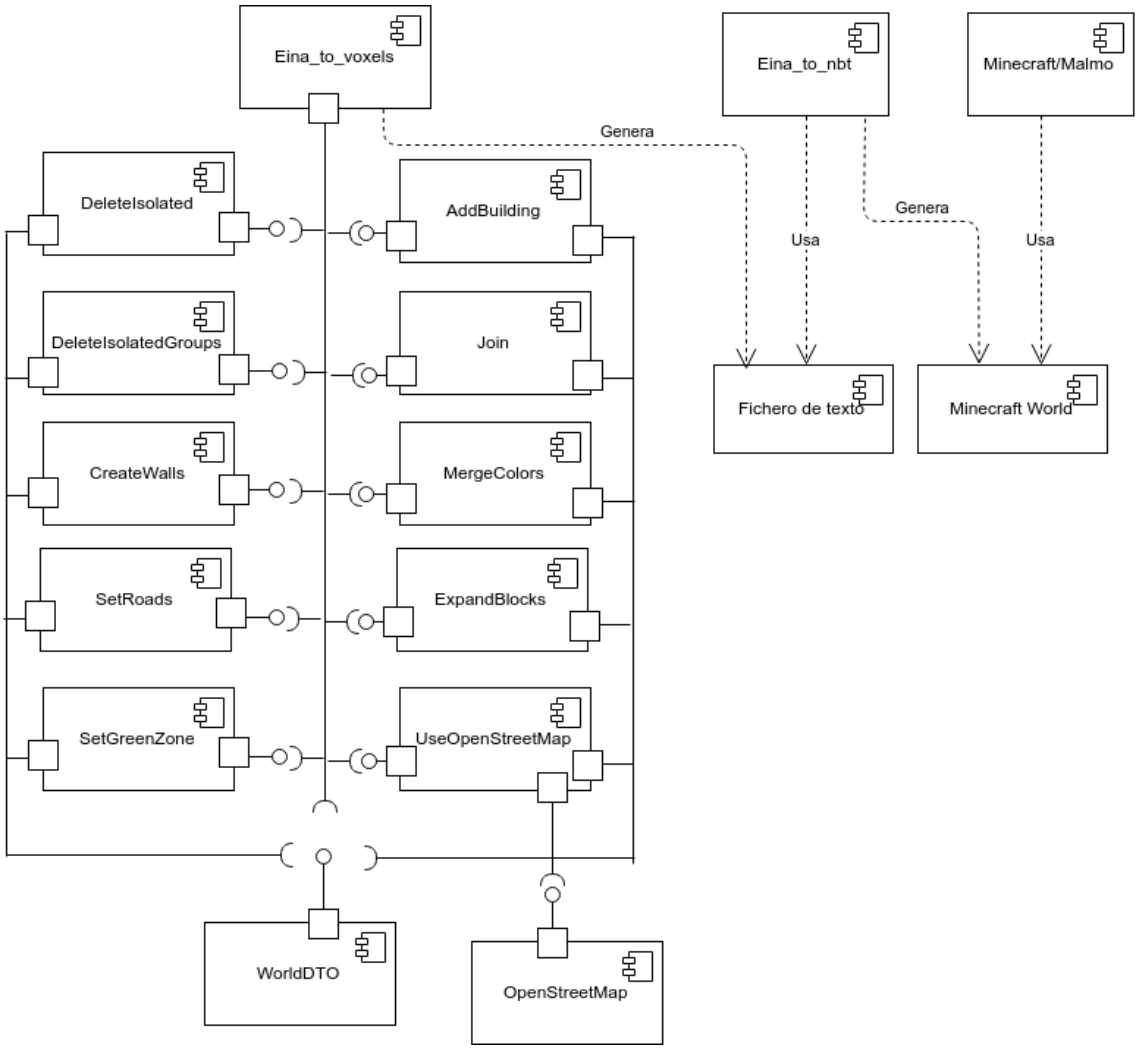


Figura 11: Diagrama de componentes y conectores

El componente Eina_to_voxels está conectado con cada una de las heuristicas porque es el encargado de su ejecución. La heurística UseOpenStreetMap se conecta al componente OpenStreetMap para obtener metadatos a partir de OpenStreetMap. Todas las heuristicas junto con el componente Eina_to_voxels, comparten una misma instancia de WorldDTO.

El componente Eina_to_voxels puede generar un fichero que representa el mundo virtual en un formato intermedio. El componente Eina_to_nbt lee el fichero para generar un mundo en formato Minecraft.

Se ha creado el componente Minecraft para representar a Malmo, que es el cliente Minecraft utilizado en las pruebas. El mundo Minecraft es leído por el componente Minecraft/Malmo en la carpeta “Minecraft/run/saves/” para iniciar el juego.

3.3 Diagrama de secuencia

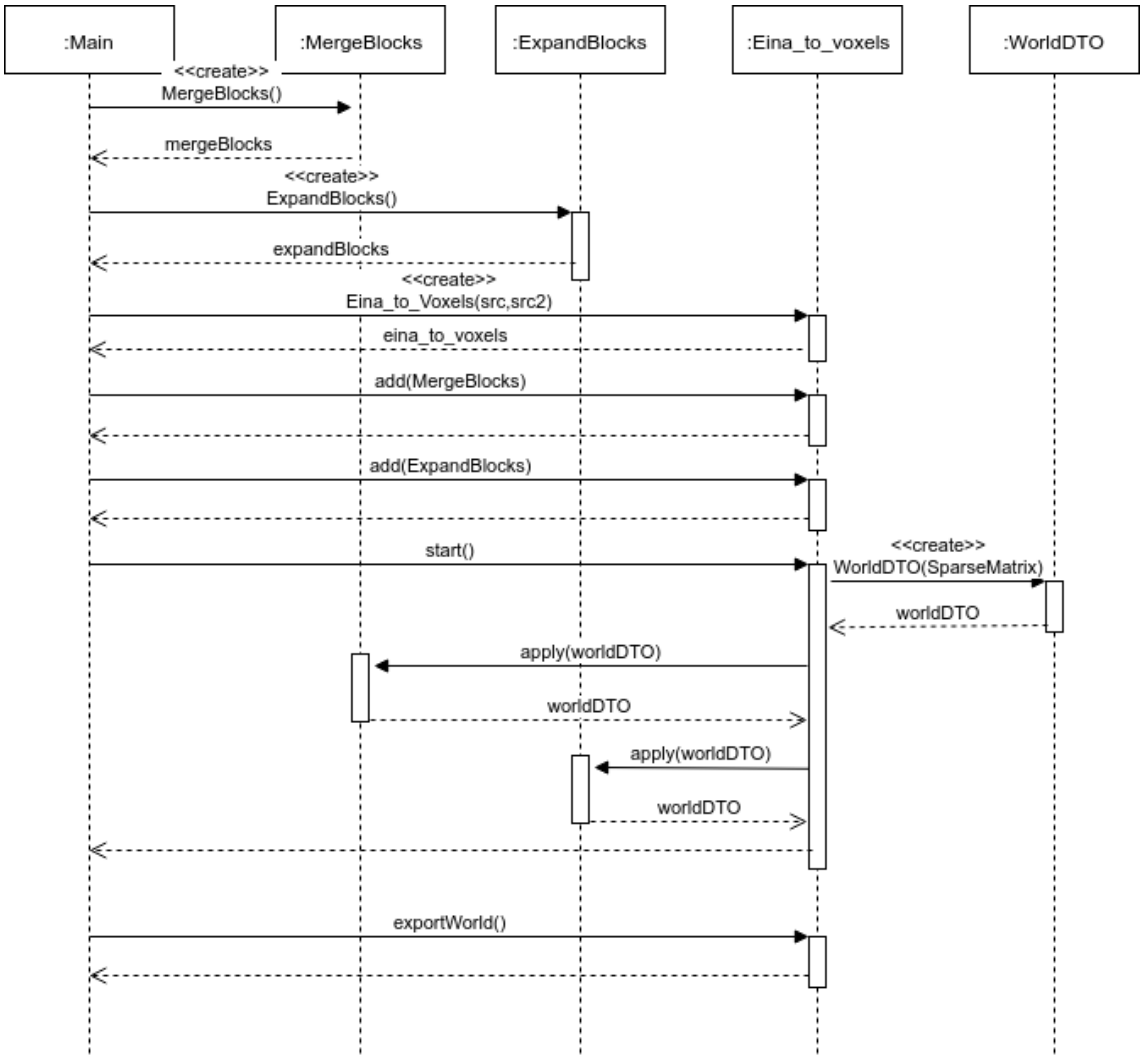


Figura 11: Diagrama de secuencia

El diagrama superior muestra un ejemplo de la ejecución de la aplicación. Main es el nombre del módulo que usa la librería EINA-TO-VOXELS. Este modulo es independiente de la aplicación y puede ser desarrollado por el usuario para incluir las heurísticas más apropiadas para su problema. En el ejemplo, el usuario inicializa la clase Eina_to_voxels pasándole la dirección de la nube de puntos y de una imagen, y añade a la lista de heurísticas de la clase Eina_to_voxels aquellas que ha inicializado previamente, en este caso MergeBlocks y ExpandBlocks. Una vez que ha añadido las heuristicas utiliza el método start. El método start de la clase Eina_to_voxels inicializa un WorldDTO con la información del mundo y ejecuta cada una de las heuristicas pasándoles como parámetro el objeto WorldDTO. Cada una de las heuristicas realiza transformaciones sobre worldDTO y luego lo devuelve para que lo ejecute la siguiente heurística siguiendo el patrón de filtro-y-tubería. Cuando ya se han ejecutado todas las heuristicas, el usuario ejecuta el método exportWorld para generar un fichero de texto que contenga el mundo virtual.

3.4 Heurísticas utilizadas

Para cada una de las heurísticas se va a explicar su funcionamiento y se van a comparar imágenes del mundo virtual con y sin ella.

3.4.1 UseOpenStreetMap

Esta heurística no modifica la matriz tridimensional, sino los metadatos. Utiliza el módulo OpenStreetMap para obtener matrices bidimensionales que indican si en una coordenada (x,y) hay una zona verde, un edificio o una carretera.

3.4.2 MergeColors

Divide el espacio en clústeres tridimensionales de tamaño reducido. Para cada vóxel se recalcula su color dando un mayor peso a aquellos que se encuentren dentro de su mismo clúster.

De esta forma se consigue disminuir las irregularidades en el color de los voxels.

Esta heurística puede ser utilizada más de una vez para que todos los clústeres tengan un solo color.

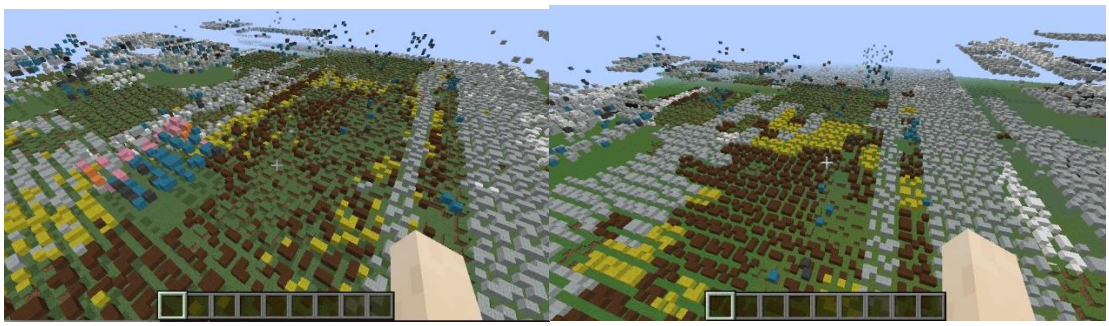


Figura 11: Comparativa heurística MergeBlocks

El color es más uniforme en la imagen de la derecha y algunos colores aislados como el rosa de la izquierda han desaparecido.

3.4.3 DeleteIsolated

Elimina aquellos voxels que no tengan ningún otro vóxel a su alrededor.

Figura 12: Comparativa heurística DeleteIsolated

El numero de bloques aislados disminuye considerablemente.

3.4.4 DeleteIsolatedGroups

Divide el espacio en clústeres planos que contienen conjuntos pequeños de voxels.

Para cada clúster se suma el número de vecinos de cada uno de sus voxels. Una vez se ha calculado se divide entre el número de voxels de dicho clúster. Si el valor es bajo, se eliminan todos los voxels de dicho clúster.

El objetivo de esta heurística es eliminar zonas con baja densidad de voxels. Al utilizar clústeres planos se consigue que el suelo quede plano y sin irregularidades. Las estructuras que tengan bastante inclinación serán eliminadas. Esta heurística se ha creado principalmente para eliminar los árboles.

Para evitar que esta heurística elimine los tejados inclinados de los edificios u otras edificaciones, se pueden utilizar metadatos de OpenStreetMap. Si se usan, la heurística solo actúa en zonas que no estén marcadas como edificio.

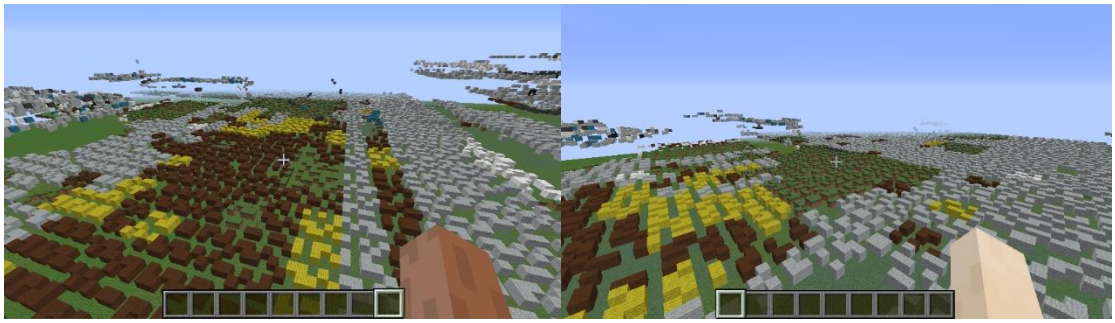


Figura 13: Comparativa heurística DeleteIsolatedGroups

Al utilizar la heuristica desaparecen los bloques aislados que no habían sido eliminados con la heurística deleteIsolated.

3.4.5 ExpandBlocks

Para cada vóxel se comprueba si existe algún vóxel a su derecha, en frente suyo o en los dos casos. Si no existe un vóxel en alguna de las celdas, se crea una copia del vóxel en dicha celda.

Esta heurística se utiliza porque la densidad de la nube de puntos puede no ser suficiente para rellenar celdas contiguas. Elimina gran parte de los huecos del mundo virtual.

Figura 14: Comparativa heurística ExpandBlocks

El suelo se convierte en una superficie lisa.

3.4.6 Join

Para cada voxels se comprueba si hay algún vóxel a su derecha o frente a él. Si en alguno de los casos no lo hay, se comprueba si dicho hueco puede rellenarse. Para cada hueco se calcula el número de voxels con su misma altura que hay en sus alrededores. Si el número de voxels es alto, se crea un vóxel con el color más frecuente entre sus alrededores.

El objetivo de esta heurística es rellenar pequeños huecos en el suelo o en los tejados.

Como se dijo anteriormente, un LIDAR utiliza rayos laser para calcular la distancia a objetos, por lo que los objetos que se encuentran tapados por otros objetos no aparecen en la nube de puntos, como es el caso de las paredes o el suelo de debajo de los árboles. Cuando esta heurística rellena esos huecos, comprueba si se encuentra en una zona verde, si es así, es probable que ese sea el hueco que ha dejado un árbol. Si se encuentra en una zona adecuada y tiene espacio suficiente se crea un árbol en ese hueco.

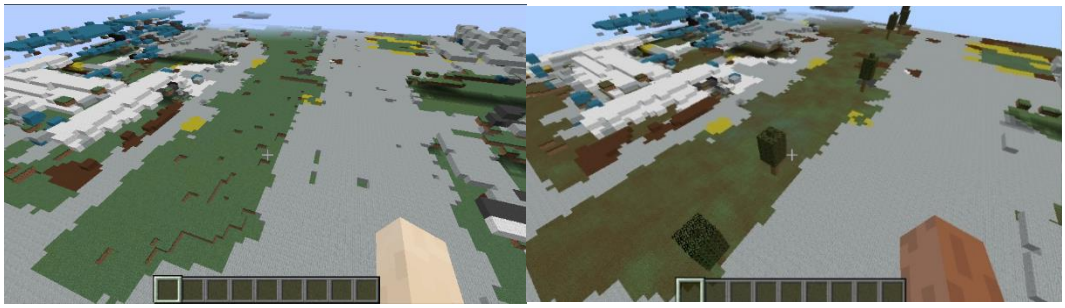


Figura 15: Comparativa heurística Join

Los huecos que aun quedaban se rellenan y en las zonas verdes aparecen arboles .

3.4.7 SetGreenZone

Cambia el color a verde en todos los voxels que se encuentren en una zona verde. Esta heurística no tendrá efecto si no se ha ejecutado UseOpenStreetMap previamente.

Figura 16: Comparativa heurística MergeBlocks

Las zonas verdes se pintan de color verde, por lo que la separación con el resto del mundo resulta más evidente.

3.4.8 SetRoads

Cambia el color a gris oscuro en todos los voxels que se encuentren en una carretera. Esta heurística no tendrá efecto si no se ha ejecutado UseOpenStreetMap previamente.

Figura 17: Comparativa heurística SetRoads

Las carreteras se pintan de color gris oscuro para resaltar la diferencia con la acera.

3.4.9 AddBuilding

Esta heurística inserta una estructura en formato binvox dentro del matriz tridimensional. Recibe como parámetros las coordenadas UTM en las que situar la estructura.

Si es necesario se modificará el tamaño de la matriz tridimensional para que la estructura no sobrepase sus límites.

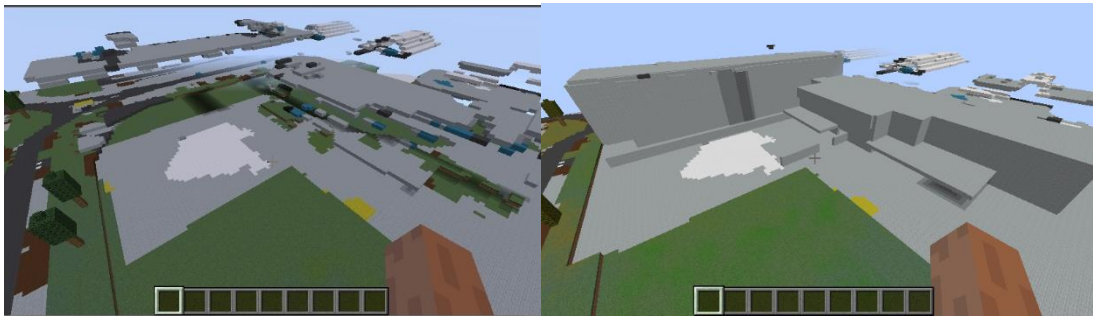


Figura 18: Comparativa heurística AddBuilding

Los bloque donde estaba el edificio desaparecen y en su lugar aparece el edificio insertado.

3.4.10 CreateWalls

Para cada vóxel se comprueba si existe algún otro voxels en sus mismas coordenadas a una altura menor, si es así se crea una columna de voxels entre los dos puntos.

Para evitar que se rellenen los interiores de los edificios, solo se crean columnas en aquellos voxels que no esté rodeado de voxels, es decir, solo se crean columnas en los laterales de los edificios. Al crearse columnas contiguas en los laterales de los edificios, da la sensación de tratarse de una pared.

Existen otras estructuras en las que podrían generarse columnas de forma no deseada. No crearán columnas en los edificios insertados con la heurística AddBuilding ni tampoco en los voxels que componen los árboles.

Si una columna tiene más de 6 bloques se generan ventanas mediante bloques de vidrio. La separación entre ventanas en una columna debe ser mayor a 3 y el número de bloques entre una ventana y el suelo o el tejado debe ser mayor a 2

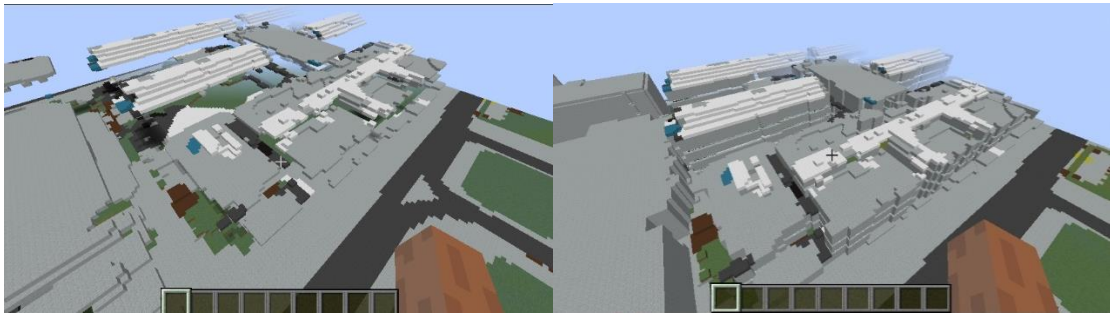


Figura 19: Comparativa heurística CreateWalls

Se crean las paredes con sus ventanas. Las ventanas se encuentran a la altura adecuada y la separación entre ellas es suficiente.

3.5 Orden de las heurísticas

El orden al aplicar las heurísticas del apartado anterior influye mucho en el resultado obtenido. No es necesario utilizar todas las heurísticas, pero si se utilizan es recomendable seguir las siguientes recomendaciones.

- Join y ExpandBlocks deberían utilizarse antes que CreateWalls, ya que para que se creen las columnas debe haber un suelo en el que terminen. AddBuilding también debería utilizarse antes para evitar que se creen muros en un edificio que ya los tienen.
- Si no se usa UseOpenStreetMap antes que SetGreenZone y SetRoads, no tendrán ningún efecto.
- Join y ExpandBlocks deberían utilizarse antes que AddBuildings para evitar que se cierren huecos en el interior del edificio.
- DeleteIsolatedGroups y DeleteIsolated debería utilizarse antes que ExpandBlocks, ya que se la densidad de puntos crece, su efecto disminuye.
- MergeColors debería utilizarse antes que SetGreenZone y SetRoads para no modificar sus colores.

4 Implementación de la solución

En este apartado se describen aspectos concretos de la implementación, como tecnologías utilizadas o problemas encontrados durante el desarrollo.

4.1 EINA-TO-VOXELS

4.1.1 OpenStreetMap

Para obtener información de OpenStreetMap, se ha utilizado la librería osmapi. Esta librería permite la lectura y descarga de zonas en OpenStreetMap a partir de dos puntos expresados mediante latitud y longitud.

Para explicar las responsabilidades del módulo OpenStreetMap es necesario conocer el modelo de datos usado en los mapas de OpenStreetMap.

Un mapa en OpenStreetMap tiene 3 tipos de datos diferentes, también llamados elementos:

- Nodos: Definen puntos en el espacio.
- Caminos: Conjuntos de nodos para formar líneas o áreas.
- Relaciones: Conjuntos de nodos y caminos con información sobre cómo se relacionan.

Cada elemento lleva asociado diversos tags que lo describen.

El objetivo del módulo OpenStreetMap es obtener las zonas verdes, edificios y carreteras de OpenStreetMap como una matriz de booleanos.

Para distinguir unos de otros se han utilizado los siguientes criterios.

- Las zonas verdes y los edificios se representan con caminos o con relaciones mientras que las carreteras únicamente con caminos.
- Las zonas verdes llevan asociado alguno de los siguientes tags (leisure=grass, leisure=park, natural=wood).
- Los edificios siempre tienen un tag building distinto de nulo.
- Las carreteras llevan asociado alguno de los siguientes tags (highway=residential, higway=secondary, highway=service)

Para facilitar la transformación de los elementos a una matriz de booleanos, se ha utilizado otra librería llamada ogr que permite crear geometrías a partir de un conjunto de puntos. Para cada camino o relación se forma una geometría a partir de las coordenadas de sus nodos. Estas geometrías se añaden a un conjunto de geometrías. En total se calculan 3 conjuntos de geometrías, las zonas verdes, los edificios y las carreteras. Comprobando la distancia de cada coordenada de las matrices a los conjuntos es posible determinar si dicha coordenada forma o no forma parte del conjunto.

4.1.2 Exportar mundo

Como ya se dijo en la sección de diseño, se ha decidido que EINA-TO-VOXELS genere un fichero que represente el mundo virtual con un formato específico para que EINA-TO-NBT lo pueda leer. Dicho fichero tiene el siguiente formato.

Cada línea del fichero representa un vóxel. Se utilizan espacios en blanco para separar los valores del vóxel.

Los tres primeros valores son las coordenadas (x,y,z) del vóxel.

El cuarto valor es el tipo de bloque del vóxel dentro del intervalo 0-19, en donde:

- 0-15: Algodón [27] en cada uno de sus posibles colores.
- 16: Tierra
- 17: Madera
- 18: Hojas
- 19: Vidrio

Para agilizar la importación del mundo en EINA-TO-NBT. Se han agrupado los voxels en chunks. Los voxels que pertenecen a un mismo chunk aparecen en líneas consecutivas. De esta forma se acelera el proceso y se reduce el uso de memoria en EINA-TO-NBT.

4.1.3 Problemas y soluciones

Durante el transcurso del proyecto han aparecido algunos problemas. Para todos ellos se ha hecho lo posible por mitigar sus efectos. A continuación, se describirán los problemas más importantes y sus soluciones.

Las nubes de puntos del CNIG estaban en formato .laz, pero la librería para la lectura de nubes de puntos solo aceptaba el formato .las. Fue necesario utilizar una herramienta para convertir las nubes de puntos a formato .las.

Tanto los datos LIDAR como las ortofotos se obtuvieron durante el día y por tanto contienen sombras. Una sombra disminuye la luminosidad de un área volviéndola oscura. Es posible aumentar la luminosidad de un color en RGB convirtiéndolo a formato HSV [30], aumentando su luminosidad y convirtiendo de nuevo a RGB. Si se aumenta la luminosidad en un área, la sombra desaparece mostrando el color original, el problema está en que resulta difícil determinar automáticamente que zonas son sombras. Si se aumenta la luminosidad del conjunto, las zonas que no son sombras quedan demasiado claras. Al final se decidió iluminar en proporción de la luminosidad según la fórmula de abajo. Al utilizar la inversa de la luminosidad, se iluminan principalmente las zonas más oscuras. Otra opción hubiese sido utilizar varias imágenes a distinta hora del día para detectar el movimiento de las sombras, pero no se disponían de los datos suficientes.

$$newLum = Lum + 0.2 * (1 - Lum)^2$$

Otro de los problemas fue la detección de árboles. En los datos LIDAR, los árboles se representan como un conjunto de puntos bastante disperso sobre un hueco en el suelo. La apariencia de estos árboles resulta extraña, por lo que fue necesario identificarlos para rehacerlos de una forma más realista y para ello se barajaron varias opciones. La primera opción fue identificar voxels de color verde que se encuentren cerca de grupos de voxels dispersos de color verde para bajarlos a nivel del suelo y crear árboles, sin embargo, no todos los árboles son de color verde y resulta difícil calcular el nivel del suelo en una zona, por lo que esta opción tuvo que ser descartada. Al final, se decidió eliminar los voxels dispersos, utilizando datos de OpenStreetMap para no eliminar otros elementos, y rellenar los huecos del suelo creando árboles en las zonas verdes.

Para representar los colores en Minecraft se decidió utilizar la paleta de 16 colores del algodón. Para transformar los colores en formato RGB a sólo 16 colores se realizaron comparaciones para determinar cuál era el color más próximo. A pesar de ello, había colores que no se ajustaban bien a ninguno de los 16 y como resultado aparecían colores poco usuales como el morado o el rosa. Para mejorar la apariencia final se asignaron pesos a cada uno de los colores en función de su frecuencia en el mundo real.

El método getMatrices de la clase OpenStreetMap tenía algunos problemas de rendimiento. Calcular la distancia de un punto a una geometría no es una operación demasiado costosa, pero si esta operación se utiliza para cada coordenada de la matriz bidimensional, entonces si lo es. Por esta razón, fue necesaria una optimización para disminuir el coste. Si la distancia de un punto a un conjunto es mayor a 0, quiere decir que todas las coordenadas que se encuentren entre dicho punto y esa distancia, no forman parte del conjunto y por tanto no es necesaria su comprobación.

Las zonas descargadas desde el CNIG eran demasiado grandes. Para transformar una de esas zonas, la aplicación tardaba bastantes minutos en terminar, por lo que las pruebas eran demasiado lentas. Se decidió reducir la zona para que las pruebas fueran más rápidas. La herramienta utilizada para ello es la misma que se utilizó para convertir nubes de puntos .laz a .las.

4.1.4 Herramientas utilizadas

Para esta aplicación se decidió utilizar python como lenguaje de programación, a pesar de no tener mucha experiencia con este lenguaje, ya que muchas de las librerías necesarias lo utilizaban. En concreto, se ha usado la versión 2.7 de python.

Las librerías utilizadas son:

- Pointcloud_proc: para la creación de matrices dispersas tridimensionales y para la transformación de coordinadas UTM a coordinadas dentro de la matriz.
- Binvox_rw [9]: para la lectura de ficheros binvox en la heurística AddBuilding.
- Utm [1]: para la transformación entre coordinadas UTM y latitud, longitud y viceversa.
- Osmapi [21]: para la descarga y lectura de mapas de OpenStreetMap.
- OGR [25]: para la creación de formas geométricas a partir de puntos.
- Laspy [12]: para la lectura de nubes de puntos en formato .las.

Para la instalación de las librerías se ha utilizado el sistema de gestión de paquetes PIP [2], permite la instalación de las dependencias de la aplicación mediante un solo comando.

Para la conversión entre formatos .laz y .las y para la reducción del tamaño de las nubes de puntos se utilizó la herramienta LasTools

4.2 EINA-TO-NBT

4.2.1 Importar mundo

Para generar un mundo en formato NBT se ha utilizado la librería Substrate [23]. Esta librería resulta bastante completa y permite abstraerse de la mayoría de los conceptos de los ficheros NBT. No ha sido necesario realizar ninguna modificación sobre la librería, por lo que lo único que se ha implementado en EINA-TO-NBT es la parte de importación del mundo virtual.

La aplicación recibe un fichero como parámetro, que es el que se ha generado con EINA-TO-VOXELS. El programa añada cada línea del fichero, que representa un bloque, a un objeto mundo de la librería Substrate. Al terminar de leer el fichero se guarda el mundo en la localización que recibe como segundo parámetro.

La gestión de los bloques se realiza a nivel de chunk, por lo que al haberse agrupado las líneas del fichero por chunks, no es necesario volver a buscarlo.

Unos bloques por debajo del mapa se ha insertado una capa de tierra, para que, si por algún motivo hay un hueco en el mapa, no se vea una caída sin final.

4.2.2 Problemas y soluciones

Cuando el fichero a importar es de gran tamaño, el sistema utilizaba gran cantidad de memoria para su lectura y dependiendo del tamaño del fichero, el programa podía colgarse. Para evitar este problema se modificó para que se guardase el mundo cada 500.000 líneas, liberando así la memoria.

4.2.3 Herramientas utilizadas

Al igual que en el caso anterior, fueron las librerías la que determinaron el lenguaje de la aplicación. EINA-TO-NBT está escrito en C# que se ejecuta dentro de la plataforma .NET.

La única librería utilizada ha sido la de Substrate para la abstracción del formato NBT.

5 Pruebas

En este apartado se describen las pruebas utilizadas durante el desarrollo y los resultados de dichas pruebas.

5.1 Pruebas unitarias

Para verificar el correcto funcionamiento del proyecto a lo largo de su desarrollo, se han creado una serie de pruebas unitarias para cada uno de sus módulos. Tan solo se han desarrollado pruebas para la aplicación EINA-TO-VOXELS, dado que EINA-TO-NBT es una aplicación pequeña y se ha considerado suficiente con la ejecución de pruebas manuales.

Para la ejecución de las pruebas en EINA-TO-VOXELS se ha utilizado la librería unittest [5], mientras que la cobertura de código se ha calculado con la herramienta coverage [4].

Coverage calcula la cobertura de todos los ficheros del directorio de trabajo. Algunos de estos ficheros, son librerías que tienen su propio banco de pruebas y no es necesario comprobarlas. Creando un fichero con nombre ".coveragerc" es posible omitir calcular la cobertura en algunos ficheros. El fichero ".coveragerc" contiene lo siguiente:

```
[run]
omit =
        ../eina_to_voxels/binvox_rw*
        ../einat_to_voxels/pointcloud_proc*
        *__init__.py
```

Al calcular la cobertura del código para los tests realizados se obtiene un 89%.

Module ↓	statements	missing	excluded	coverage
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/eina_to_voxels.py	69	12	0	83%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/addBuilding.py	35	26	0	26%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/createWalls.py	34	1	0	97%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/deleteIsolated.py	11	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/deleteIsolatedGroups.py	38	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/expandBlocks.py	22	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/heuristic.py	7	2	0	71%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/join.py	79	2	0	97%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/mergeColors.py	44	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/setGreenZone.py	16	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/setRoads.py	15	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/heuristic/useOpenStreetMap.py	23	0	0	100%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/openStreetMap/openStreetMap.py	137	17	0	88%
/home/jorge/GIT/EINA-ON-MINECRAFT/EINA-TO-VOXELS/eina_to_voxels/worldDTO.py	9	0	0	100%
Total	**539**	**60**	**0**	**89%**

Figura 20: Cobertura de código

5.2 Comparativa Minecraft vs realidad

Además de las pruebas unitarias, se han realizado pruebas manuales. Estas pruebas consisten en la comparación del mundo Minecraft obtenido, con las fuentes utilizadas y con imágenes de la zona.

Para la ejecución de estas pruebas se ha utilizado Malmo [15]. Malmo es un proyecto software libre, que cuenta con un cliente Minecraft para máquinas Linux.

Estas pruebas han sido realizadas principalmente por el propio desarrollador. Sin embargo, en ocasiones se ha solicitado la ayuda de personas externas al proyecto para que realicen las pruebas desde un punto de vista más imparcial.

A continuación, se muestra la comparativa de la aplicación en su estado actual.

Figura 21: Resultados en la zona del EINA

Las imágenes superiores muestran diversas zonas del EINA tras su transformación a formato Minecraft.

Figura 22: Comparativa aérea de la zona del EINA

La imágenes superiores muestran una vista aera de la zona elegida. La de arriba a la izquierda muestra la zona en formato minecraft, la de la derecha es una ortofoto del CNIG, mientras que la de abajo es la nube de puntos utilizada para la generación del mundo.

Durante el desarrollo del primer prototipo, debido a que la aplicación no estaba lo suficientemente avanzada como para realizar las pruebas desde un cliente Minecraft, se realizó otro tipo de pruebas manuales. Estas pruebas consistían en exportar la matriz tridimensional a javascript y visualizarlo mediante ThreeJS [32]. La biblioteca utilizada para la creación de la matriz dispersa permitía también su exportación a este formato, por lo que no fue necesario desarrollar nada.

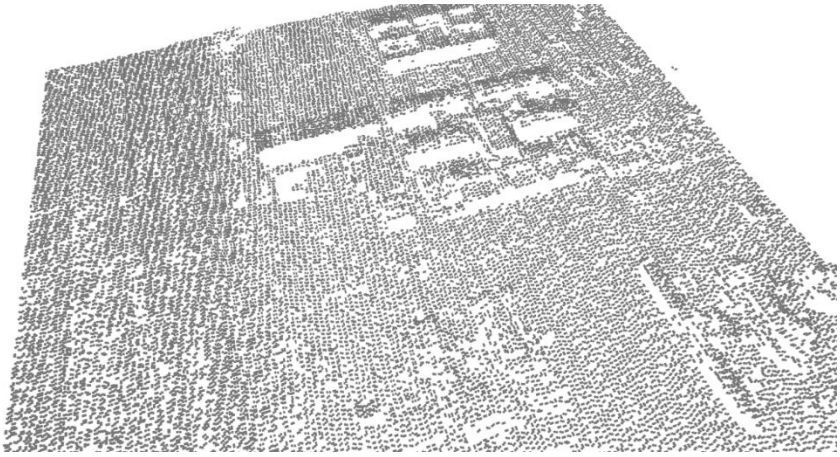


Figura 23: Zona del EINA visualizada con ThreeJS

5.3 Medición de prestaciones

Se han realizado algunas pruebas para comprobar las prestaciones de la aplicación.

Dado que no todas las heurísticas tienen el mismo coste en tiempo, se ha preparado una ejecución que utiliza todas las heurísticas una única vez. De esta forma puede calcularse el tiempo que tarda cada una de ellas. A su vez, se ha medido el tiempo de ejecución para EINA-TO-VOXELS, EINA-TO-NBT y para las dos aplicaciones juntas. Para disminuir el margen de error se ha realizado cada una de las mediciones 5 veces y se ha calculado la media en cada una de ellas. A continuación, se muestra una tabla con los resultados obtenidos.

Heurística/tamaño	**250 m^2**	**500 m^2**	**1000 m^2**	**2000 m^2**
UseOpenStreetMap	3,506	19,666	118,054	265,188
MergeColors	0,304	1,71	8,812	38,764
DeleteIsolated	0,552	2,222	10,288	38,222
DeleteIsolatedGroups	0,474	3,214	16,792	102,888
ExpandBlocks	0,134	0,662	3,304	14,05
Join	0,564	2,49	11,096	42,988
SetGreenZone	0,04	0,37	1,06	3,59
SetRoads	0,026	0,338	1,602	4,538
CreateWalls	0,51	2,79	13,974	90,162
EINA-TO-VOXELS	11,998	48,694	231,474	959,566
EINA-TO-NBT	3	10	40,4	180,6
Total	14,998	58,694	271,874	1140,166

Al poner en común los tiempos totales en segundos para cada uno de los tamaños en m ² obtenemos la siguiente gráfica.

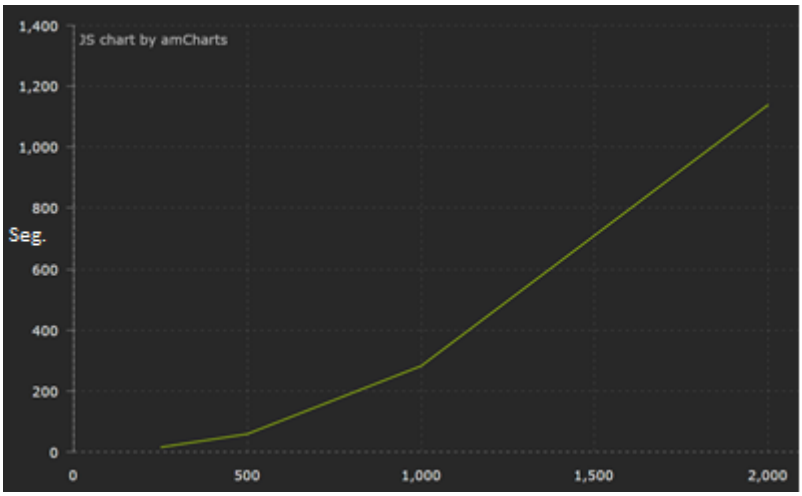


Figura 24: Función de coste temporal total

El coste temporal aumenta exponencialmente en función del tamaño de la zona.

De la misma forma que para el coste temporal, se han realizado medidas del uso de memoria en función del tamaño de la zona.

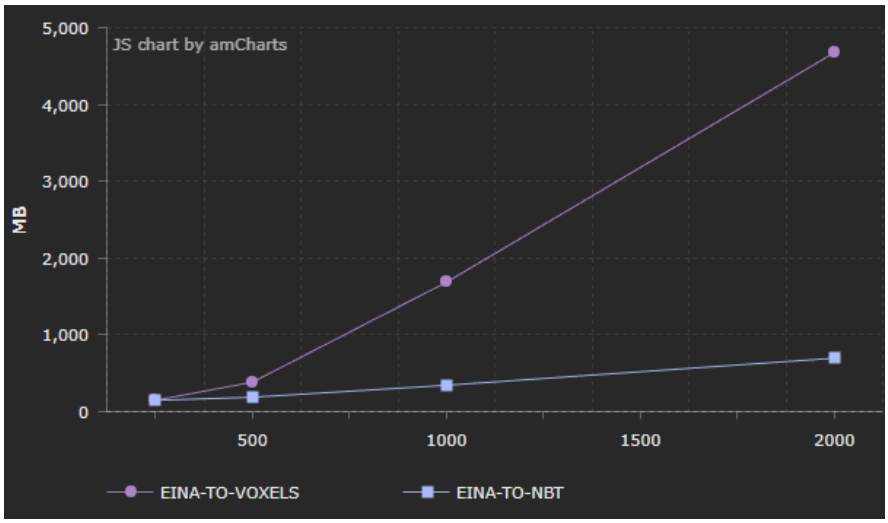


Figura 25: Función de coste temporal en memoria

La aplicación EINA-TO-VOXELS consume considerablemente más memoria que EINA-TO-NBT. Para ejecutar la aplicación en una zona no es necesario disponer de tanta memoria, pero si no es así, el rendimiento de la aplicación descenderá considerablemente.

De las gráficas superiores puede sacarse una conclusión. Si el tamaño de la zona es elevado, su división en zonas de tamaños menores aceleraría el proceso de transformación y reduciría el uso de memoria.

6 Gestión del proyecto

En este apartado se describen aspectos sobre la gestión del proyecto durante su desarrollo, como la planificación, el control de esfuerzos, el análisis de riesgos o la gestión de configuraciones.

6.1 Planificación y esfuerzos

Tal como se dijo en la propuesta de proyecto, el proyecto se ha dividido en tres fases:

1. Análisis: Esta primera fase consiste en establecer el objetivo del proyecto, elaborar los requisitos, buscar librerías que puedan ser de utilidad para la aplicación y realizar estudios sobre los formatos de ficheros y las tecnologías a utilizar.
2. Primer prototipo: El objetivo de esta fase es elaborar un primer prototipo de la aplicación, en el que la zona del EINA sea distinguible y que utilice planos CAD para recrear el interior de los edificios.
3. Versión final y documentación: El objetivo de esta tercera fase es escribir la memoria del proyecto y realizar iteraciones sobre el prototipo inicial para mejorarlo y solucionar errores hasta obtener la versión final.

Aunque en general se han respetado todas las fases, en ocasiones fue necesario continuar con el análisis para estudiar tecnologías que no se habían tenido en cuenta o para incluir o modificar algún requisito.

Para controlar los esfuerzos, se decidió utilizar un documento en Google Docs que dividía el trabajo en 4 apartados, Análisis y diseño, Desarrollo del proyecto, Documentación y Pruebas. El documento también divide los esfuerzos según el día en el que se realizaron. En total, el proyecto ha tenido una duración de 280 horas.

Puede consultarse el documento en detalle de Google Docs desde el siguiente enlace:

https://docs.google.com/spreadsheets/d/e/2PACX-1vQrO-0DHHw49yT2etx9XaNEpLJtoBWkWdIhE3hYkBGlUVJNZLfPfSz_tQbjByJ01XWMgY1WoJlxBC8L/pubhtml

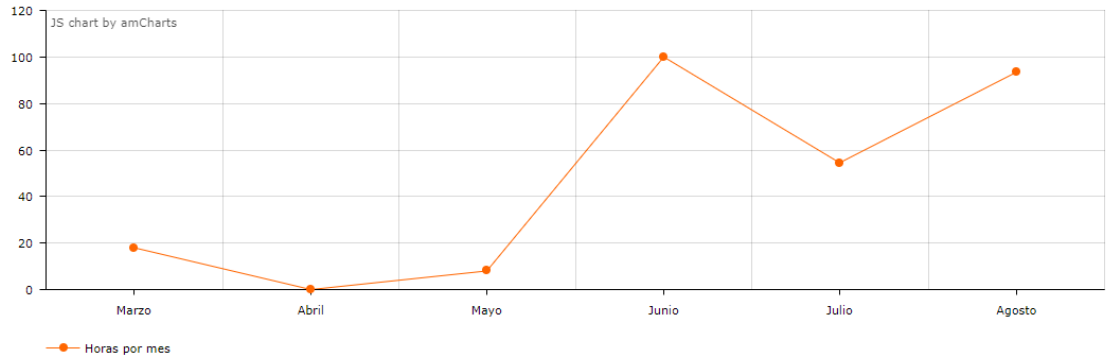


Figura 26: Esfuerzos por més

Debido a la carga de trabajo en otras asignaturas, apenas se pudo dedicar esfuerzos durante los meses de Marzo, Abril y Mayo. Al terminar el curso fue posible dedicar muchas más horas, es por eso por lo que la mayoría de los esfuerzos se agrupan en los meses de Junio Julio y Agosto.

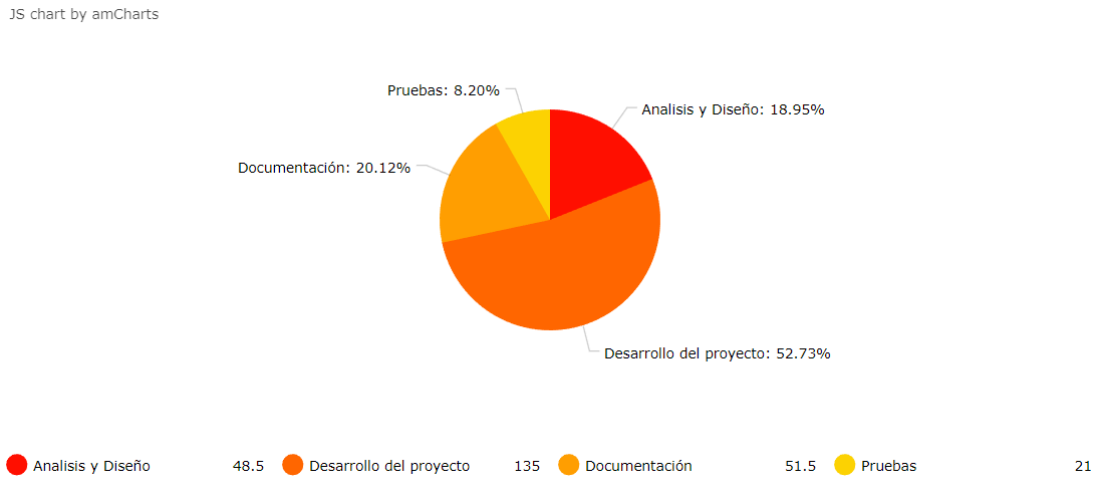


Figura 27: Esfuerzos por tipo

Más de la mitad del esfuerzo del proyecto se ha dedicado al desarrollo de la aplicación. A pesar de ello, también se ha dedicado una considerable cantidad de horas a el resto de los apartados.

6.2 Análisis de riesgos

Analizar daño y probabilidad de fallo		Probabilidad de fallo		
		Alto	Medio	Bajo
Daño en caso de fallo	Alto	A	A	B
	Medio	B	B	C
	Bajo	B	C	C

Para evitar problemas graves, se ha decidido dar una mayor clase de riesgo a los riesgos que en caso de materializarse produzcan un daño alto a pesar de la probabilidad de realizarse no sea muy alta.

Riesgo	Prob.	Daño	Clase de Riesgo	Justificación	Estrategia
Retraso en el desarrollo del proyecto	Medio	Medio	B	Si se produce un retraso en el desarrollo del proyecto es posible que alguna de las funcionalidades no pueda realizarse y el proyecto no estaría completo	Realizar una buena planificación para que en caso de retrasarse con el desarrollo tener tiempo suficiente como para compensarlo. Si a pesar de todo se produce un retraso, cambiar la fecha de entrega del trabajo.
Perdida de alguno de los datos del proyecto	Medio	Medio	B	Si se pierde algún dato del proyecto se perderá tiempo en recuperarlo, ya sea descargándolo de nuevo o rehaciéndolo.	Mantener tanto el código como la documentación en un sistema de control de versiones para que siempre sea posible recuperar una versión anterior del proyecto.
Actualiza-ción de alguna de las funciones utilizadas	Medio	Alto	A	Si alguna de las librerías que utiliza la aplicación se actualiza o cierra, alguna de las funciones o la aplicación entera podría dejar de funcionar.	Realizar un estudio sobre las librerías a utilizar para ver si siguen en activo y descargar una versión de las librerías para conservarlas en caso de cierre y para evitar que estas se actualicen.
Desconoci-miento de alguna las tecnologías empleadas	Alto	Bajo	B	La falta de experiencia con alguna de las tecnologías utilizadas podría suponer un retraso en el desarrollo del proyecto.	Dedicar una parte del tiempo del proyecto al estudio de las tecnologías desconocidas.
Actualiza-ción de Minecraft	Medio	Alto	A	Una actualización importante de Minecraft podría cambiar la forma en la que interpreta los mundos o el formato de estos y la aplicación podrían dejar de funcionar.	Utilizar solo aquellos aspectos de Minecraft que parezcan más estables. Informar de la versión de Minecraft en la que se han realizado las pruebas.

6.3 Gestión de configuraciones

A lo largo del proyecto se han utilizado sistemas de control de versiones para evitar la pérdida de datos.

El código del proyecto se encuentra en GitHub. Se han creado dos repositorios diferentes, uno para EINA-TO-VOXELS y otro para EINA-TO-NBT. Estos repositorios contienen únicamente el código de la aplicación y pequeños datos utilizados en las pruebas. En ambos casos, los cambios se suben directamente sobre la rama master.

Tanto los diagramas como la memoria del proyecto se encuentran en una carpeta de Dropbox porque es más apropiado para este tipo de documentos. Aunque los datos geográficos y los planos en CAD se encuentran en una carpeta fuera del sistema de control de versiones, hay una copia de dichos datos dentro de la carpeta de Dropbox.

6.4 Licencias

En este proyecto solo se han utilizado herramientas software libre. De la misma forma, los datos utilizados también tienen una licencia de uso libre.

Las dos aplicaciones desarrolladas, se han subido en GitHub junto con una licencia de uso GNU. Esta licencia garantiza a los usuarios finales la libertad de usar, estudiar, compartir y modificar el software. Si otros usuarios distribuyen, modifican o amplían el software, deberán mantener estas libertades.

7 Conclusiones

En este último apartado se realiza una retrospectiva del proyecto y se analiza cómo podría continuarse el proyecto.

7.1 Trabajo para un futuro

El trabajo realizado es el correspondiente a la duración del proyecto, es decir, unas 300 horas aproximadamente. Sin embargo, aún hay algunas tareas que podrían haberse realizado en caso de disponer de más tiempo.

Mejorar el coste temporal y en memoria para zonas grandes. Como ya se ha dicho en la medición de prestaciones, hubiera sido posible disminuir el tiempo de ejecución dividiendo las zonas en zonas más pequeñas. Si a eso añadimos la posibilidad de guardar en disco cada una de las zonas, disminuiría también el coste en memoria, ya que solo se utilizaría para la zona que se está procesando en ese momento. Además, el proceso de transformación de una zona podría paralelizarse fácilmente, lo que reduciría bastante el coste temporal.

Explotar más datos de OpenStreetMap: Actualmente, solo se ha utilizado OpenStreetMap para detectar zonas verdes, edificios o carreteras, pero existen muchos otros elementos que podrían detectarse con OpenStreetMap, como ríos, mares o edificios emblemáticos. También podrían utilizarse más datos del CNIG. Algunos datos como los mapas de elevación del terreno o los mapas vectoriales podrían incluirse en la aplicación.

Generar mapas de un territorio mayor, como Aragón o España a partir de los datos del CNIG. Para procesar áreas tan grandes sería necesario utilizar servicios de computación en la nube [8] como Amazon Web Services. Una vez procesado el territorio podrían ofrecerse mapas en formato Minecraft a través de un servicio web.

Crear y mejorar heurísticas para seguir mejorando el parecido con la realidad.

Aumentar el número de bloques utilizados. En Minecraft hay un total de 255 bloques para la construcción. Muchos de estos bloques no están siendo utilizados a pesar de que podrían contribuir al aspecto general, por ejemplo, los bloques de agua, flores o escaleras.

Mejorar la automatización de los planos CAD. Actualmente es necesario un proceso manual antes de poder insertar planos en el mundo virtual. Podría crearse un módulo que se encargue de leer los planos CAD y transformarlos directamente a una estructura de voxels.

Crear otro módulo para la exportación a formato Minecraft. Si se crease un módulo en python con este cometido, ya no sería necesario utilizar EINA-TO-NBT, bastaría con una única aplicación.

7.2 Valoración personal

Este proyecto ha sido diferente a todos los que se han realizado en el grado, un trabajo de gran tamaño y sobre todo individual.

Al trabajar en todos los aspectos del proyecto, uno acaba adquiriendo una visión global que te permite avanzar más rápido que en un proyecto en equipo. Además, se ahorra mucho tiempo en reuniones y toma de decisiones. Por otro lado, al trabajar de forma individual es normal descuidar algunos aspectos como la documentación o la actualización de los repositorios de GitHub. Estos aspectos siguen siendo necesarios en el proyecto y no deben ser descuidados.

A diferencia de otros trabajos, este proyecto se ha aplicado a un dominio específico, la geografía. Como resultado se han adquirido conocimientos sobre este dominio. También se ha aprendido al trabajar con tecnologías nuevas como python o C#.

En general considero que la realización del trabajo ha sido satisfactoria. Se han conseguido los objetivos propuestos al principio del proyecto y se han arreglado los problemas que han ido surgiendo durante el desarrollo.

Referencias

1. (n.d.). Retrieved August 21, 2017, from https://pypi.python.org/pypi/utm
2. (n.d.). Retrieved August 21, 2017, from https://pypi.python.org/pypi/pip
3. (n.d.). Retrieved August 21, 2017, from http://www.patrickmin.com/binvox/
4. (n.d.). Retrieved August 21, 2017, from https://pypi.python.org/pypi/coverage
5. 25.3. unittest - Unit testing framework¶. (n.d.). Retrieved August 21, 2017, from https://docs.python.org/2/library/unittest.html
6. AutoCAD. (2017, August 18). Retrieved August 21, 2017, from https://es.wikipedia.org/wiki/AutoCAD
7. Baidu Maps. (2017, August 08). Retrieved August 21, 2017, from https://en.wikipedia.org/wiki/Baidu_Maps
8. Computación en la nube. (2017, August 17). Retrieved August 21, 2017, from https://es.wikipedia.org/wiki/Computaci%C3%B3n_en_la_nube
9. D. (2016, April 12). Dimatura/binvox-rw-py. Retrieved August 21, 2017, from https://github.com/dimatura/binvox-rw-py
10. F. (2017, August 21). FreeCAD/FreeCAD. Retrieved August 21, 2017, from https://github.com/FreeCAD/FreeCAD
11. Instituto Geográfico Nacional - Ministerio de Fomento. (n.d.). Retrieved August 21, 2017, from http://centrodedescargas.cnig.es/
12. L. (2017, August 09). Laspy/laspy. Retrieved August 21, 2017, from https://github.com/laspy/laspy
13. LAStools. (2017, July 06). Retrieved August 21, 2017, from https://rapidlasso.com/lastools/
14. LIDAR. (2017, August 17). Retrieved August 21, 2017, from https://es.wikipedia.org/wiki/LIDAR
15. M. (2017, August 08). Microsoft/malmo. Retrieved August 21, 2017, from https://github.com/Microsoft/malmo
16. Minecraft. (2017, August 20). Retrieved August 21, 2017, from https://en.wikipedia.org/wiki/Minecraft
17. Minecraft como forma de arte. (n.d.). Retrieved August 21, 2017, from http://www.microsiervos.com/archivo/arte-y-diseno/minecraft-como-forma-de-arte.html
18. Mustun, W. B. (n.d.). QCAD - 2D CAD for Windows, Linux and Mac. Retrieved August 21, 2017, from https://qcad.org/

19. OpenStreetMap. (n.d.). Retrieved August 21, 2017, from https://www.openstreetmap.org/

20. Ortofotografía. (2017, March 25). Retrieved August 21, 2017, from https://es.wikipedia.org/wiki/Ortofotograf%C3%ADa

21. Osmapi. (n.d.). Retrieved August 21, 2017, from http://wiki.openstreetmap.org/wiki/Osmapi

22. Pipeline (software). (2017, July 18). Retrieved August 21, 2017, from https://en.wikipedia.org/wiki/Pipeline_(software)

23. R. (2012, January 11). R-plus/substrate. Retrieved August 21, 2017, from https://github.com/r-plus/substrate

24. Vóxel. (2017, June 01). Retrieved August 21, 2017, from https://es.wikipedia.org/wiki/V%C3%B3xel

25. Welcome to the Python GDAL/OGR Cookbook!¶. (n.d.). Retrieved August 21, 2017, from https://pcjericks.github.io/py-gdalogr-cookbook/

26. Wiki, M. (2017, June 23). NBT format. Retrieved August 21, 2017, from https://minecraft.gamepedia.com/NBT_format

27. Wiki, M. (2017, August 20). Wool. Retrieved August 21, 2017, from https://minecraft.gamepedia.com/Wool

28. GeoBoxers. (n.d.). Retrieved August 21, 2017, from http://www.geoboxers.com/

29. (n.d.). Retrieved August 27, 2017, from http://www.patrickmin.com/binvox/

30. Modelo de color HSV. (2017, August 05). Retrieved August 27, 2017, from https://es.wikipedia.org/wiki/Modelo_de_color_HSV

31. Sistema de coordenadas universal transversal de Mercator. (2017, August 26). Retrieved August 27, 2017, from https://es.wikipedia.org/wiki/Sistema_de_coordenadas_universal_transversal_de_Mercator

32. Three.jsr87. (n.d.). Retrieved August 27, 2017, from https://threejs.org/

33. Wavefront. (2017, August 21). Retrieved August 27, 2017, from https://en.wikipedia.org/wiki/Wavefront

34. Wiki, M. (2016, April 07). Region file format. Retrieved August 27, 2017, from https://minecraft.gamepedia.com/Region_file_format

35. Wiki, M. (2017, July 21). Chunk format. Retrieved August 27, 2017, from https://minecraft.gamepedia.com/Chunk_format

36. Wine. (2017, August 26). Retrieved August 27, 2017, from https://es.wikipedia.org/wiki/Wine

Anexos

A. Manual de instalación

En esta sección aparecen las dependencias de cada una las aplicaciones y se muestran los comandos para la instalación de las dependencias y de la aplicación en sí.

A.1 Instalación de EINA-TO-VOXELS

La aplicación EINA-TO-VOXELS se encuentra en el repositorio de GitHub:
https://github.com/sanz1995/EINA-TO-VOXELS

Para la ejecución de EINA-TO-VOXELS es necesario utilizar la versión 2.7 de python o superior. Esta aplicación cuenta con algunas dependencias que pueden instalarse manualmente o con el sistema de gestión de paquetes pip.

Estas dependencias son:

- utm 0.4.2
- osmapi 0.8.1
- laspy 1.5.0

Si estas dependencias ya están instaladas, no es necesario utilizar pip. El fichero requirements.txt es un listado con todas las dependencias de la aplicación. Para instalar las dependencias con pip basta con ejecutar el siguiente comando.

```
pip install -r requirements.txt
```

Una vez se han instalado las dependencias, la aplicación puede instalarse copiando la carpeta “eina_to_voxels” en el directorio de trabajo o ejecutando los siguientes comandos.

```
python setup.py build
```

```
python setup.py install
```

Para instalar el cliente Minecraft de Malmo hay que clonar o descargar su repositorio en GitHub:

```
git clone https://github.com/Microsoft/malmo.git
```

Una vez se ha descargado, puede ejecutarse con:

```
./Minecraft/launchClient.sh
```

A.2 Instalación de EINA-TO-NBT

La aplicación EINA-TO-NBT se encuentra en el repositorio de GitHub:

https://github.com/sanz1995/EINA-TO-VOXELS

Para poder ejecutar la aplicación en una máquina Linux, es necesario tener instalado Mono en su versión 4.6.2 , ya que contiene un compilador para C#, que es el lenguaje en el que está escrita la aplicación.

Para instalar la aplicación debe instalarse primero la librería de Substrate mediante:

```
xbuild Substrate.csproj
```

Posteriormente hay que hacer lo mismo con la aplicación en sí.

```
xbuild EINA_TO_NBT.csproj
```

El comando anterior da como resultado un ejecutable en la carpeta bin.

B. Manual del desarrollador

En esta sección se muestra como descargar los datos que se utilizaran para su transformación a Minecraft. También se muestran los comandos necesarios para la ejecución de la aplicación y un ejemplo de su uso.

B.1 Obtener datos de una zona

Para descargar los datos del CNIG, hay que acceder a su página de descargas en http://centrodedescargas.cnig.es

Si se pulsa en buscar, aparecerá un mapa de España. Hay que pulsar en “Buscar por punto”, acercarse a la zona deseada y pulsar en ella. Al pulsar sobre un punto aparecerá una lista con todos los datos que incluyen ese punto. Los datos que se pueden usar en la aplicación son las ortofotos de máxima actualizad y los LIDAR en formato .laz. Para cada uno de ellos se pulsa sobre añadir a la lista de descargar y luego sobre el botón cesta de descargas. Se pulsa sobre aceptar descarga y aceptar. Aparecerá una encuesta, pero no es necesario rellenarla. Finalmente se pulsa sobre descargar en cada uno de los ficheros de la lista.

Tras descargar los datos LIDAR en formato .laz, es necesario convertirlos a .las para que la aplicación pueda leerlos. LasTools es una herramienta que ofrece conversión de laz a las.

https://rapidlasso.com/lastools/

Para ejecutarlo hay que ir a la carpeta bin y ejecutar laszip.exe. Si se quiere ejecutarlo desde una máquina Linux, puede utilizarse el emulador Wine con el siguiente comando.

```
Wine laszip.exe
```

Al ejecutar el comando se iniciará la interfaz gráfica del programa. Para seleccionar ella nube de puntos a convertir, hay que seleccionar la pestaña browse del lado izquierdo, introducir el nombre del fichero y hacer doble click en él.

El CNIG divide el espacio en áreas de 2 X 2 Km, por lo que dependiendo de la máquina en la que se ejecuta puede ser demasiado. Esta misma herramienta ofrece también la posibilidad de filtrar los puntos para obtener zonas más pequeñas. En la parte inferior izquierda aparecen las coordenadas máximas y mínimas de la nube de puntos. Si se pulsa sobre la pestaña filter pueden añadirse filtros por coordenadas. En el ejemplo se han utilizado los filtros para obtener la zona del EINA, es decir, descartar ((x < 675500) | (y < 4616500) | (x > 676000) | (y > 4617000))

Una vez se ha leído el fichero y se han creado los filtros hay que pulsar sobre el botón DECOMPRESS del lado derecho, asegurándose de que está marcada la opción LAS. Aparecerá una nueva ventana con el comando a ejecutar, si se pulsa en start se lleva a cabo la conversión.

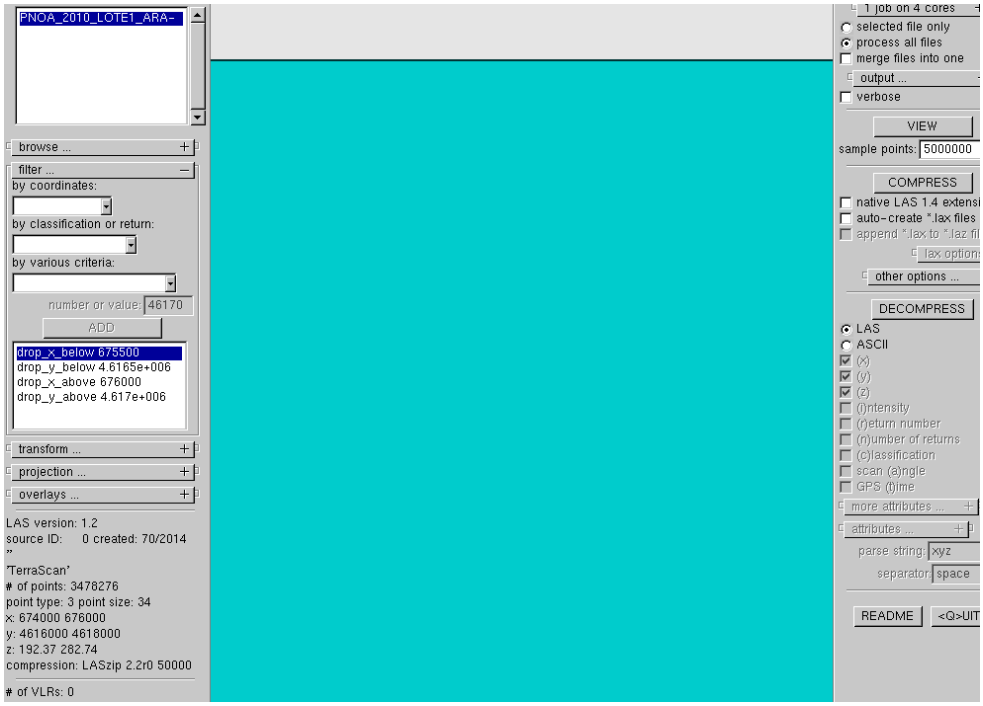


Figura 28: Interfaz gráfica de LasZip

B.2 Ejecución de EINA-TO-VOXELS

Para utilizar la aplicación el desarrollador debe importar el módulo “eina_to_voxels” desde el fichero en el que se quiera utilizar. Una vez importado ya es posible utilizar las operaciones que se describen en su interfaz.

A continuación, se muestra un ejemplo de fichero que utiliza “eina_to_voxels” con algunas heurísticas.

```
import sys
import eina_to_voxels

if __name__ == '__main__':

   world = eina_to_voxels.World(sys.argv[1])
   world.add(eina_to_voxels.UseOpenStreetMap([]))

   for i in range(0,3):
      world.add(eina_to_voxels.MergeColors([]))

   world.add(eina_to_voxels.DeleteIsolated([]))

   for i in range(0,3):
      world.add(eina_to_voxels.DeleteIsolatedGroups([]))

   world.add(eina_to_voxels.ExpandBlocks([]))

   for i in range(0,15):
      world.add(eina_to_voxels.Join([7]))

   for i in range(0,5):
      world.add(eina_to_voxels.Join([]))

   world.add(eina_to_voxels.SetGreenZone([]))
   world.add(eina_to_voxels.SetRoads([]))
   world.add(eina_to_voxels.AddBuilding([sys.argv[3],
         float(sys.argv[4]),float(sys.argv[5]),float(sys.argv[6])]))

   world.add(eina_to_voxels.CreateWalls([]))
   world.start()
   world.exportWorld(sys.argv[2])
```

El ejemplo anterior corresponde con el fichero main.py del repositorio. Es posible su utilización con el siguiente comando:

```
python main.py area.las out.txt
```

El primer parámetro del script es la dirección en la que se encuentra el fichero .las con el que se generará el mundo y el segundo es la dirección del fichero de salida de la aplicación. Este tipo de ejecución utiliza las heurísticas por defecto y no incluye planos de edificios.

Si se quiere ejecutar las pruebas, hay que acceder al directorio tests y ejecutar.

```
python eina_to_voxelsTests.py
```

Para calcular la cobertura de código con coverage hay que ejecutar.

```
coverage run  eina_to_voxelsTests.py
```

Seguido del comando inferior para generar un html con los resultados.

```
coverage html
```

Al igual que con el resto de la aplicación, es necesario que las dependencias estén instaladas.

B.3 Ejecución de EINA-TO-NBT

Para utilizar la aplicación es necesario ejecutar con mono el fichero generado en la fase de instalación.

```
mono bin/EINA_TO_NBT.exe src dest
```

La aplicación recibe dos parámetros. src es la ruta del fichero que describe el mundo a generar con el formato de EINA-TO-VOXELS. dest es la ruta en la que se guardará el mundo en formato Minecraft.

C. API

En esta sección se describen los métodos más importantes de cada uno de los módulos de la aplicación EINA-TO-VOXELS

C.1 Eina_to_voxels

- **Eina_to_voxels(pointCloud, image, minX, minY, maxX, maxY)**

  Es el constructor de la clase Eina_to_voxels. Lee los ficheros pointCloud y image, y a partir de ellos, genera una matriz dispersa llamada que representa el mundo virtual. También inicializa la una lista vacia donde se guardan las heurísticas a ejecutar.

  Parámetros:

  - pointCloud (string): Localización de una nube de puntos en formato .las.
  - image (string): Localización de una imagen en formato jpg o png. Si se recibe None no se lee ninguna imagen.
  - minX: Coordenada x más pequeña de la imagen
  - minY: Coordenada y más pequeña de la imagen
  - maxX: Coordenada x más grande de la imagen
  - maxY: Coordenada y más grande de la imagen

  Devuelve:

  - Objeto Eina_to_voxels inicializado

- **add (heuristic)**

  Añade una heurística a la lista de heurísticas a realizar.

  Parámetros:

  - heuristic (Heuristic): objeto que deriva de la clase Heuristic.

  Devuelve:

  - Nada

- **start ()**

  Ejecuta todas las heurísticas de la lista de heurísticas y actualiza la matriz dispersa que representa el mundo virtual.

  Parámetros:

  - Nada

  Devuelve:

  - Nada

- **changeBrightness(red, green, blue)**

  Aumenta el brillo de los colores recibidos en función de su luminosidad.

  Parámetros:

  - red ([int]): Valor del rojo para cada color en formato RGB
  - green ([int]): Valor del verde para cada color en formato RGB
  - blue ([int]): Valor del azul para cada color en formato RGB

  Devuelve:

  - Una tupla de tamaño 3 con los valores del rojo, verde y azul tras aumentar la luminosidad en cada uno de los colores.

- **exportWorld (dest)**

  Exporta la matriz tridimensional a un fichero en un formato intermedio y lo guarda en dest. El fichero intermedio puede ser leído por la aplicación EINA-TO-NBT.

  Parámetros:

  - dest (string): Localización en la que guardar el fichero intermedio.

  Devuelve:

  - Nada

C.2 OpenStreetMap

- **OpenStreetMap ()**

  Constructor de la clase OpenStreetMap. Inicializa el objeto OpenStreetMap.

  Parámetros

  - Nada

  Devuelve:

  - Objeto OpenSteetMap inicializado.

- **readMap (map)**

  Lee map e inicializa un diccionario con cada uno de los elementos de dicho mapa.

  Parámetros

  - map (string): Localización de un mapa en formato.osm.

  Devuelve:

  - Nada

- **downloadMap (minX, minY, maxX, maxY)**

  Lee un mapa de los servidores de OpenStreetMap e inicializa un diccionario con los elementos de dicho mapa. El mapa obtenido de OpenStreetMap está delimitado por minX, minY, maxX y maxY.

  Parámetros

  - minX: Coordenada x más pequeña del mapa a obtener.
  - minY: Coordenada y más pequeña del mapa a obtener.
  - maxX: Coordenada x más grande del mapa a obtener.
  - maxY: Coordenada y más grande del mapa a obtener.

  Devuelve:

  - Nada

- **getGreenZone ()**

  Comprueba el diccionario para obtener todos los elementos que componen las zonas verdes y transformarlos a un conjunto de polígonos. Un elemento es una zona verde si *"(tag.get("landuse") == "grass") | (tag.get("leisure") == "park") | (tag.get("natural") == "wood")".*

  Parámetros

  - Nada

  Devuelve:

  - Conjunto de polígonos con las zonas verdes.

- **getRoads ()**

  Comprueba el diccionario para obtener todos los elementos que componen las carreteras y transformarlos a un conjunto de líneas. Un elemento es una carretera sí *"(tag.get("landuse") == "grass") | (tag.get("leisure") == "park") | (tag.get("natural") == "wood")".*

  Parámetros

  - Nada

  Devuelve:

  - Conjunto de polígonos con las zonas verdes.

- **getGreenZone ()**

  Comprueba el diccionario para obtener todos los elementos que componen las carreteras y transformarlos a un conjunto de lineas. Un elemento es una zona verde si "(tag.get("highway")=="residential")|(tag.get("highway")=="secondary")|(tag.get("highway")=="service")"

  Parámetros

  - Nada

  Devuelve:

  - Conjunto de líneas con las carreteras.

- **getBuildings ()**

  Comprueba el diccionario para obtener todos los elementos que componen los edificios y transformarlos a un conjunto de polígonos. Un elemento es un edificio sí "(tag.get("building") != "") & (tag.get("building") != None)"

  Parámetros

  - Nada

  Devuelve:

  - Conjunto de polígonos con los edificios.

- **getMatrices (xSize, ySize)**

  Transforma los conjuntos de polígonos de las zonas verdes, edificios y carreteras a matrices booleanas de dos dimensiones. Si el valor de una coordenada de esa matriz es cierto, en ese punto hay una zona verde, un edificio o una carretera, dependiendo de que matriz se trate.

  Parámetros

  - xSize (int): Número de filas de las matrices booleanas a crear.
  - ySize (int): Número de columnas de las matrices booleanas a crear.

  Devuelve:

  - Una tupla de tamaño 3 con las matrices booleanas para las zonas verdes, las carreteras y los edificios.

C.3 UseOpenStreetMap

- **UseOpenStreetMap (params)**

  Constructor de la clase UseOpenStreetMap. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Utiliza el módulo OpenStreetMap para obtener matrices bidimensionales que indican si en una coordenada (x,y) hay una zona verde, un edificio o una carretera. Añade las matrices a world.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.4 MergeColors

- **MergeColors ()**

  Constructor de la clase MergeColors. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Divide el espacio de world en clústeres tridimensionales de tamaño reducido. Para cada vóxel se recalcula su color dando un mayor peso a aquellos que se encuentren dentro de su mismo clúster

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.5 DeleteIsolated

- **DeleteIsolated ()**

  Constructor de la clase DeleteIsolated. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Elimina aquellos voxels de world que no tengan ningún otro vóxel a su alrededor.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.6 DeleteIsolatedGroups

- **DeleteIsolatedGroups ()**

  Constructor de la clase DeleteIsolatedGroups. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Divide el espacio de world en clústeres planos que contienen conjuntos pequeños de voxels. Para cada clúster se suma el número de vecinos de cada uno de sus voxels. Una vez se ha calculado se divide entre el número de voxels de dicho clúster. Si el valor es bajo, se eliminan todos los voxels de dicho clúster.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.7 ExpandBlocks

- **ExpandBlocks ()**

  Constructor de la clase ExpandBlocks. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Para cada vóxel de world se comprueba si existe algún vóxel a su derecha, en frente suyo o en los dos casos. Si no existe un vóxel en alguna de las celdas, se crea una copia del vóxel en dicha celda.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.8 Join

- **Join ()**

  Constructor de la clase Join. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. Si existe, el primer parámetro de la lista indica la altura máxima sobre la que se realiza la heurística.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Para cada vóxel de world se comprueba si hay algún vóxel a su derecha o frente a él. Si en alguno de los casos no lo hay, se comprueba si dicho hueco puede rellenarse. Para cada hueco se calcula el número de voxels con su misma altura que hay en sus alrededores. Si el número de voxels es alto, se crea un vóxel con el color más frecuente entre sus alrededores.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.9 SetGreenZone

- **SetGreenZone ()**

  Constructor de la clase SetGreenZone. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Cambia el color a verde en todos los voxels de world que se encuentren en una zona verde. Esta heurística no tendrá efecto si no se ha ejecutado UseOpenStreetMap previamente.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.10 SetRoads

- **SetRoads ()**

  Constructor de la clase SetRoads. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Cambia el color a gris oscuro en todos los voxels de world que se encuentren en una carretera. Esta heurística no tendrá efecto si no se ha ejecutado UseOpenStreetMap previamente.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.11 AddBuilding

- **AddBuilding ()**

  Constructor de la clase AddBuilding. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. El primer parámetro es la localización del fichero .binvox. El segundo y tercer parámetro son las coordenada UTM en las que insertar el edificio.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Inserta una estructura en formato binvox dentro de la matriz de world.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.

C.12 CreateWalls

- **CreateWalls ()**

  Constructor de la clase CreateWalls. Inicializa la heurística.

  Parámetros

  - Params ([Object]): Lista con los parámetros para esta heurística. No se usa ninguno.

  Devuelve:

  - Heurística inicializada.

- **apply (world)**

  Para cada vóxel de world se comprueba si existe algún otro voxels en sus mismas coordenadas a una altura menor, si es así se crea una columna de voxels entre los dos puntos.

  Parámetros

  - world (WorldDTO): Objeto que representa el mundo virtual.

  Devuelve:

  - Objeto WorldDTO tras las modificaciones realizadas.