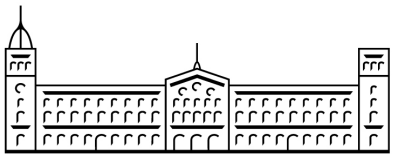


Trabajo final de grado

GRADO DE INFORMÁTICA

Facultat de Matemàtiques
Universitat de Barcelona

Simulación y Visualización de Smart Grids

Autor: Martin Azpillaga Aldalur

Director:	Dr. Jesús Cerquides
Realitzat a:	Institut de Investigació de Intel.ligència Artificial
Barcelona,	17 de enero de 2016

Agradecimientos

Me gustaría agradecer a los siguientes personas por proporcionar material indispensable para la realización del proyecto: - iconos: - usuarios de stackoverflow:

Abstract

Example abstract.

Resúmen

Resúmen

Índice

1. Introducción y antecedentes **1**

1.1. El entorno: La humanidad necesita energía eléctrica 1

1.2. La idea de los Smart Grids 1

1.3. El problema de la liquidación de mercados distribuidos de energía: CEAP 1

1.4. Algoritmos de clearing para mercados distribuidos de energía: ILP, Radpro 1

1.5. RadPro 1

1.6. A continuación 1

2. Objetivos **2**

2.1. En este capítulo 2

2.2. El problema 2

2.3. Partes del problema: Creación de grafo, CEAP, Simulación, Visualización, Reports 2

3. Metodología y herramientas **2**

3.1. En este capítulo 2

3.2. Git 2

3.3. LaTeX 3

3.4. Blender 4

3.5. Unity 5

3.6. Java 5

3.7. JSON 6

3.8. GnuPlot 7

3.9. A continuación 8

4. Creación de la ciudad **9**

4.1. En este capítulo 9

4.2. Definiendo la ciudad 9

4.3. Leyendo el fichero 9

4.4. Creando la red 10

4.5. Información completa: Steiner Trees 10

4.6. Información incompleta: Nearest neighbour problem 10

4.7. Ampliar a Open Street Map, CityEngine. 11
4.8. A continuación 11

5. Simulación **12**
5.1. En este capítulo 12
5.2. Filosofía de diseño de java. POO, encapsulación, interfaces. 12
5.3. Input/Output. 12
5.4. Diagramas. 12
5.5. Modelado de una casa 12
5.6. Battery, ... 12
5.7. Generator, ... 12
5.8. Appliance, ... 12
5.9. Bid. 12
5.10. A continuación 12

6. Visualización **13**
6.1. En este capítulo 13
6.2. Filosofía de diseño de Unity 13
6.3. Diseñando los paneles 14
6.4. Control de la cámara 15
6.5. Eligiendo la ciudad 15
6.6. Seleccionando el perfil de las casas 15
6.7. Seleccionando la franja horaria 16
6.8. Ejecutando la animación 16
6.9. Mostrando la información relevante 17
6.10. Programando las animaciones 17
6.11. Controlando el flow 18
6.12. Parseando el JSON 18
6.13. A continuación 18

7. Futuro trabajo **19**
7.1. Futuro trabajo 19

1. Introducción y antecedentes

1.1. El entorno: La humanidad necesita energía eléctrica

La energía eléctrica es básica para la producción de bienes y servicios. Cada vez necesitamos más energia. REE En España la distribución de la energía se hace a base de distribuidoras que proveen energía a la red que conecta todos los consumidores, bien sean hogares o empresas. Bajo la amenaza de que la materia de energías fósiles, como el petróleo o el carbón, se van agotando, muchos goviernos muestran interés en invertir en investigar maneras más eficientes de tratar con la energía.

Por un lado ha aumentado el índice de plantas solares instaladas:

Pero lo que realmente puede llegar a alterar el proceso de energía es el comienzo de la expansió del sector de plantas renovables personales. Cada vez son más económicos y eficientes, de manera que requieren menor inversión inicial.

Auge de energías renovables personales. Mencionar artículos donde se tratan smart grids.

1.2. La idea de los Smart Grids

Aquí es donde entran las Smart Grids. Las smart grids proponen un sistema de distribución distribuido en vez de centralizado. Permiten la idea que los propios consumidores puedan vender energía que produzcan convirtiendose así en prosumidores - productores y consumidores - Una Smarti grid Auge de energías renovables personales. Mencionar artículos donde se tratan smart grids.

1.3. El problema de la liquidación de mercados distribuidos de energía: CEAP

Quien controla todo el flujo, cuando se hacen los intercambios, como se hacen los intercambios, como pueden interferir los usuarios en estos intercambios. Mencionar reglamentos de otros países y estado actual de España.

1.4. Algoritmos de clearing para mercados distribuidos de energía: ILP, Radpro

1.5. RadPro

1.6. A continuación

Definiremos en que manera queremos ampliar este trabajo.

2. Objetivos

2.1. En este capítulo

2.2. El problema

Explicar qué se intenta resolver

2.3. Partes del problema: Creación de grafo, CEAP, Simulación, Visualización, Reports

Explicar por que existe cada parte y por que está diferenciado del resto.

3. Metodología y herramientas

3.1. En este capítulo

Explicar programas adeuados para cada parte así como posibles distintas maneras de implementar cada aspecto a grandes rasgos

3.2. Git

Bitbucket Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de revisiones Mercurial y Git. Bitbucket ofrece planes comerciales y gratuitos. Se ofrece cuentas gratuitas con un número ilimitado de repositorios privados (que puede tener hasta cinco usuarios en el caso de cuentas gratuitas) desde septiembre de 2010,1 los repositorios privados no se muestran en las páginas de perfil - si un usuario sólo tiene depósitos privados, el sitio web dará el mensaje .Este usuario no tiene repositorios". El servicio está escrito en Python.2

Es similar a GitHub, que utiliza Git. En una entrada de blog del 2008,3 Bruce Eckel hace una comparación favorablemente de Bitbucket frente a Launchpad, que utiliza Bazaar.

Git Git (pronunciado "guit"2) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Al principio, Git se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. 3 Sin embargo, Git se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. 4 Hay algunos proyectos de mucha relevancia que ya usan Git, en particular, el grupo de programación del núcleo Linux.

El mantenimiento del software Git está actualmente (2009) supervisado por Junio Hamano, quien recibe contribuciones al código de alrededor de 280 programadores.

Índice [ocultar] 1 Características 2 Órdenes básicas 3 Buenas prácticas 4 Véase también 5 Referencias 6 Enlaces externos Características[editar] El diseño de Git se basó en BitKeeper y en Monotone. 5 6

El diseño de Git resulta de la experiencia del diseñador de Linux, Linus Torvalds, manteniendo una enorme cantidad de código distribuida y gestionada por mucha gente, que incide en numerosos detalles de rendimiento, y de la necesidad de rapidez en una primera implementación.

Entre las características más relevantes se encuentran:

Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. Git incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Una presunción fundamental en Git es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan. Gestión distribuida. Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, Git le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local. Los almacenes de información pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH. Git también puede emular servidores CVS, lo que habilita el uso de clientes CVS pre-existentes y módulos IDE para CVS pre-existentes en el acceso de repositorios Git. Los repositorios Subversion y svk se pueden usar directamente con git-svn. Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución. Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial). Esto existía en Monotone. Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja con base en cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos. Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles, y posiblemente desastrosas, coincidencias de ficheros diferentes en un único nombre. Realmacenamiento periódico en paquetes (ficheros). Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (con base en diferencias) no ocurre cada cierto tiempo.

3.3. LaTeX

(escrito LaTeX en texto plano) es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica.

Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas.

LaTeX está formado por un gran conjunto de macros de TeX, escrito por Leslie Lamport en 1984, con la intención de facilitar el uso del lenguaje de composición tipográfica, creado por Donald Knuth. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea.

LaTeX es software libre bajo licencia LPPL.

MikTeX MiKTeX es una distribución TeX/LaTeX para Microsoft Windows que fue desarrollada por Christian Schenk.

Las características más apreciables de MiKTeX son su habilidad de actualizarse por sí mismo descargando nuevas versiones de componentes y paquetes instalados previamente, y su fácil proceso de instalación.

La versión actual de MiKTeX es 2.9 y está disponible en su página oficial. Además, tiene características que incluyen MetaPost y pdfTeX y compatibilidad con Windows 7. A partir de la versión 2.7 se incluyó soporte integrado para XeTeX. Desde la versión 2.9 ofrece soporte para ConTeXt Mark IV. Licencia FSF/Debian

Sublime Text es un editor de texto y editor de código fuente está escrito en C++ y Python para los plugins.1 Desarrollado originalmente como una extensión de Vim, con el tiempo fue creando una identidad propia, por esto aún conserva un modo de edición tipo vi llamado Vintage mode.2

Se distribuye de forma gratuita, sin embargo no es software libre o de código abierto,3 se puede obtener una licencia para su uso ilimitado, pero el no disponer de ésta no genera ninguna limitación más allá de una alerta cada cierto tiempo.

3.4. Blender

Blender es un programa informático multi plataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. También de composición digital utilizando la técnica procesal de nodos, edición de vídeo, escultura (incluye topología dinámica) y pintura digital. En Blender, además, se puede desarrollar vídeo juegos ya que posee un motor de juegos interno.

El programa fue inicialmente distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, GNU/Linux, Solaris, FreeBSD e IRIX.

3.5. Unity

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux, y permite crear juegos para Windows, OS X, Linux, Xbox 360, PlayStation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. Gracias al plugin web de Unity, también se pueden desarrollar videojuegos de navegador para Windows y Mac. Desde el sitio web oficial se pueden descargar dos versiones: Unity y Unity Pro.

C# Los scripts de Unity pueden ser programados en tres posibles lenguajes de programación: C# Javascript, Boo.

3.6. Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.1 2

El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

La compañía Sun desarrolló la implementación de referencia original para los compiladores de Java, máquinas virtuales, y librerías de clases en 1991 y las publicó por primera vez en 1995. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun volvió a licenciar la mayoría de sus tecnologías de Java bajo la Licencia Pública General de GNU. Otros también han desarrollado implementaciones alternas a estas tecnologías de Sun, tales como el Compilador de Java de GNU y el GNU Classpath.

Usamos java porque radpro está en java y por la capacidad de exportar un jar y por ser multiplataforma Netbeans NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE2 es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo

el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos (Actualmente Sun Microsystems es administrado por Oracle Corporation).

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

NetBeans IDE 6.5.2, la cual fue publicada el 19 de noviembre de 2008, extiende las características existentes del Java EE (incluyendo Soporte a Persistencia, EJB 3 y JAX-WS). Adicionalmente, el NetBeans Enterprise Pack soporta el desarrollo de Aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies (for BPEL), y modelado UML. El NetBeans C/C++ Pack soporta proyectos de C/C++, mientras el PHP Pack, soporta PHP 5.

Modularidad. Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente.

Sun Studio, Sun Java Studio Enterprise, y Sun Java Studio Creator de Sun Microsystems han sido todos basados en el IDE NetBeans.

Desde julio de 2006, NetBeans IDE es licenciado bajo la Common Development and Distribution License (CDDL), una licencia basada en la Mozilla Public License (MPL). En octubre de 2007, Sun anunció que NetBeans desde entonces se ofrecerá bajo licenciamiento dual de Licencia CDDL y la GPL versión 2.

3.7. JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido

fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

En la práctica, los argumentos a favor de la facilidad de desarrollo de analizadores o del rendimiento de los mismos son poco relevantes, debido a las cuestiones de seguridad que plantea el uso de eval() y el auge del procesamiento nativo de XML incorporado en los navegadores modernos. Por esa razón, JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, etc, que atienden a millones de usuarios) cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente.

Si bien es frecuente ver JSON posicionado contra XML, también es frecuente el uso de JSON y XML en la misma aplicación. Por ejemplo, una aplicación de cliente que integra datos de Google Maps con datos meteorológicos en SOAP hacen necesario soportar ambos formatos.

Cada vez hay más soporte de JSON mediante el uso de paquetes escritos por terceras partes. La lista de lenguajes soportados incluye ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua y Visual FoxPro.

En diciembre de 2005 Yahoo! comenzó a dar soporte opcional de JSON en algunos de sus servicios web.1

El término JSON está altamente difundido en los medios de programación, sin embargo, es un término mal descrito ya que en realidad es solo una parte de la definición del estándar ECMA-262 en que está basado Javascript. De ahí que ni Yahoo, ni Google emplean JSON, sino LJS[cita requerida]. Una de las cualidades intrínsecas de Javascript denominada LJS (Literal Javascript) facilita el flujo de datos e incluso de funciones, para la cual no requiere la función eval() si son datos los que se transfieren como en el caso de XML. Todo lo referente a transferencia de datos en todos sus tipos, incluyendo arrays, booleans, integers, etc. no requieren de la función eval(), y es precisamente en eso en donde supera por mucho JavaScript al XML, si se utiliza el LJS y no la incorrecta definición de JSON.

Cada lenguaje tiene librerias propias de parseo/creación de json. Usaremos Gson de google para Java y JsonObject de ¡ref¿para C#.

3.8. GnuPlot

gnuplot es un programa muy flexible para generar gráficas de funciones y datos.

Este programa es compatible con los sistemas operativos más populares (Linux, UNIX, Windows, Mac OS X...). El origen de gnuplot data de 1986.

gnuplot puede producir sus resultados directamente en pantalla, así como en multitud de formatos de imagen, como PNG, EPS, SVG, JPEG, etc. Se puede usar interactivamente o en modo por lotes (batch), usando scripts. Este programa tiene

gran base de usuarios y está convenientemente mantenido por sus desarrolladores. Existe una ingente cantidad de ayuda en Internet, aunque gran parte está en inglés.

gnuplot es la herramienta de dibujo de gráficas del programa GNU Octave. Existen además interfaces para su empleo a través de diversos lenguajes de programación como Perl (via CPAN), Python (via Gnuplot-py o SAGE), Java (via jgnuplot), Ruby (via Ruby Gnuplot), Ch (via Ch Gnuplot), o Smalltalk (Squeak y GNU Smalltalk). Gnuplot también puede ser empleado a través de tuberías.1

Licencia[editar] A pesar de su nombre, este programa no tiene ninguna relación con el Proyecto GNU. Originalmente, se eligió el nombre "gnuplot"para evitar conflictos con otro programa de dibujo de gráficas llamado "newplot", que en inglés se pronuncia de igual modo. También se tuvo en cuenta el parecido con otros dos de los nombres propuestos, "llamaplotz "nplot"(Gnu, en inglés, significa ñu).[1]

Este programa se distribuye bajo una licencia de software libre que permite la copia y modificación de su código fuente. Sin embargo, las modificaciones sólo se pueden distribuir en forma de parches, por lo que no es compatible con la licencia GPL. [2]

3.9. A continuación

Iremos explicando cada parte desde el núcleo (CEAP) hasta el exterior (Visualización)

4. Creación de la ciudad

4.1. En este capítulo

Empezaremos por crear una ciudad sobre la cual simular una red eléctrica inteligente. Para ello, primero deberemos definir que entendemos por ciudad dentro de nuestro programa, en que formato deberíamos guardar esta información, como acceder a esta información desde dentro del visualizador, como se interpreta y reproduce esta información dentro del visulizador, algoritmos que permiten crear una red que conecte todas las parcelas de nuestra ciudad y por último como usar herramientas como OpenStreetMap y CityEngine para crear ciudades realistas.

4.2. Definiendo la ciudad

Para visualizar una simulación, lo primero que necesitmos es una ciudad sobre la que hacer los cálculos. Aprovechando que la visualización será en 3D podemos usar como ciudad un modelo 3D. Dado que queremos permitir que la ciudad pueda ser elegida por el usuario en tiempo de ejecución, tendremos que crear un mecanismo que permita importar un modelo 3D desde un fichero.

Los grandes programas de modelado como Blender o 3DS Max permiten crear un objeto tridimensional para después exportarla a un fichero en una variada de formatos: max, 3ds, obj... Usaremos el formato Wavefront OBJ por su simplicitud y por ser de licencia abierta. Además, OBJ es un formato muy extendido, por lo que la gran mayoría de programas de modelado 3D como blender o 3ds max lo soportan.

Wavefront OBJ guarda la información en líneas, donde cada línea esta precedida por una cadena de caracteres que indican el tipo de la información que sigue. Por ejemplo v 1 1 1 indica un vértice en el punto (1,1,1) mientras que f 1 2 3 indica una cara triangular que une los tres primeros vérices. Una ventaja de Wavefront OBJ es la simplicidad en el que se pueden diferenciar multiples objetos dentro del mismo modelo. Nuestra ciudad será un único modelo que tenga por submodelo cada una de las casas/parcelas que requieran ser conectadas a la red.

4.3. Leyendo el fichero

Quiero agradecer a ¡Referencia¿por donar a la comunidad un importador de OBJ especialmente diseñado para Unity. El script coge como entrada un string a un fichero y devuelve una componente de tipo Mesh de Unity. El problema es que este script considera que todos los vertices pertenecen al mismo objeto. El código se ha ampliado de manera que detecte las líneas de formato ö ¡nombre¿ÿ así crear una Mesh para cada parcela. El resultado es un array de Meshes. Crearemos un GameObject por cada uno que tenga como Mesh filter este mesh y por nombre su nombre y un GameObject empty llamado city Por último el modelo de ciudad se situa en la escena y se escala de manera que quepa en el terreno. Ya estamos listos para proveer una red eléctrica a esta ciudad.

4.4. Creando la red

Hay muchos algoritmos que permiten unir todos los puntos dados en un espacio creando así un grafo. Recordamos que por limitaciones del RadPro nuesra red será un árbol, de manera que no podrá contener ningún ciclo. Dentro de los árboles podemos considerar dos posiciones: Creamos la red conociendo todos los puntos a unir o los puntos van añadiendose a lo largo del tiempo y la red va ampliandose secuencialmente de manera que preserve la estructura de árbol.

4.5. Información completa: Steiner Trees

En el primer caso encontramos el conocido problema de Minimum Spanning Tree Problem, que trata de construir un camino que permita a un viajero pasar por todos los puntos recorriendo la mínima distancia posible. Sin embargo, en e entorno de nuestro problema, minimizar el tiempo o la distancia que ha de recorrer la electricidad para llegar de un extremo a otro no es relevante (ocurre en milésimas de segundo), sino que podemos plantearnos el siguiente problema:

Dado un conjunto de puntos P dentro de un espacio E, cual es la configuración que minimiza la distancia total de la red permitiendo crear nodos . Esto se traduciría en un menor coste de construcción, ya que se ahorraría en material y en tiempo requerido para montar la red. Este problema es conocido bajo el nombre de Steiner Tree Problem y ha sido muy estudiado por sus enormes aplicaciones en creación de redes. Los puntos añadidos se llaman Steiner points y cumplen ciertas propiedades curiosas como: Como maáximo existen tantos steiner points como nodos iniciales menos dos Cada steiner points es un nodo en el que se cruzan exactamente 3 caminos Los tres caminos que intersectan en el punto se cortan en 120 grados entre sí.

Resulta que el problema es de complejidad NP-Hard y por lo que a partir de un número humilde de nodos (50 en un ordenador standard) la solución óptima es dificilmente alcanzable. Existen también varias maneras de aproximar steiner points, pero no he podido encontrar ninguna librería que los implemente más allá de artículos. El producto final contiene un jar llamado FindSteinerTree que recoge un listado de puntos en un archivo .txt y genera un .txt donde se guardan los steiner points y las conexiones entre los puntos.

Formato del txt: ¡imagen¿

4.6. Información incompleta: Nearest neighbour problem

en el caso anterior considerábamos que conocíamos la posición de todos los puntos que formarían el grafo. Sin embargo, si nos fijamos en la realidad, esta idea no sería práctica, ya que a lo largo del tiempo se van añadiendo y eliminando parcelas de manera que alteran los puntos del problema. Para crear un árbol en este caso usaremos el algoritmo Nearest Neighbour. Este algoritmo recorre cada uno de los puntos secuencialemnte y une el punto con el punto más cercano a menos que ese punto ya esté unido a él. Como tal tiene una complejidad cuadrática respecto a la

cantidad de nodos y es resoluble hasta para un grafo enorme sin problemas.

Este algoritmo puede adaptarse correctamente a un entorno totalmente secuencial en el que los puntos van añadiendose de forma continuada y a cada paso la red es un árbol completo.

Implementación en pseudocódigo: ¡código¿

4.7. Ampliar a Open Street Map, CityEngine.

Por último se ha considerado como construir los modelos 3D de las ciudades. Y qué mejor manera que utilizando ejemplos existentes del mundo real. El mundo está repleto de ciudades y gracias a mapas online como OpenStreetMap es fácil conseguir un fichero que contenga toda la información relevante de una zona del mundo, como por ejemplo el corazón de Manhattan. En OpenStreetMap podemos exportar un trozo de mapa en formator .osm muy parecido a xml que contine toda la información sobre calles, parcelas parques etc. que se encuentran en ella. Ahora solo queda usar un programa de modelado procedural como CityEngine para crear una ciudad en 3D a partir del mapa! CityEngine soporta nativamente el tipo de archivos .osm y es capaz de crear modelos 3D a partir de mapa extruyendo las parcelas de edificaciones y añadiendoles materiales adecuados.

OpenStreetMap, como el nombre lo indica, es una iniciativa de software libre en el que son los mismos usuarios lso que van completando el mapa y aportando cada vez información más detallada como donde se encuentran fuentes de agua o ciertas señales de tráfico. Sin embargo, CityEngine es un programa con licencia de pago, por lo que se ha dejado fuera del proyecto final. De todas maneras, podremos visualizar algunos resultados obtenidos usando la aplicación de prueba de 30 días que ofrece la empresa, que soporta importar archivos .osm y exportar el modelo en formato OBJ.

4.8. A continuación

Ahora somos capaces de crear modelos 3D, bien desde mapas reales o usando un software de modelado, guardarlos en un fichero obj, abrirlos desde una aplicación y crearles una red eléctrica a medida. Ya tenemos todo listo para poder simular como sería el trade de la energía en una ciudad así en el que cada parcela sea capaz de comprar y vender energía a toda la red. Veremos como se define el perfil de una casa y como afecta cada uno de sus parámetros a la simulación final, así como los detalles más importantes sobre el diseñado e implementación del simulador.

5. Simulación

5.1. En este capítulo

5.2. Filosofía de diseño de java. POO, encapsulación, interfaces.

5.3. Input/Output.

5.4. Diagramas.

5.5. Modelado de una casa

5.6. Battery,

5.7. Generator,

5.8. Appliance,

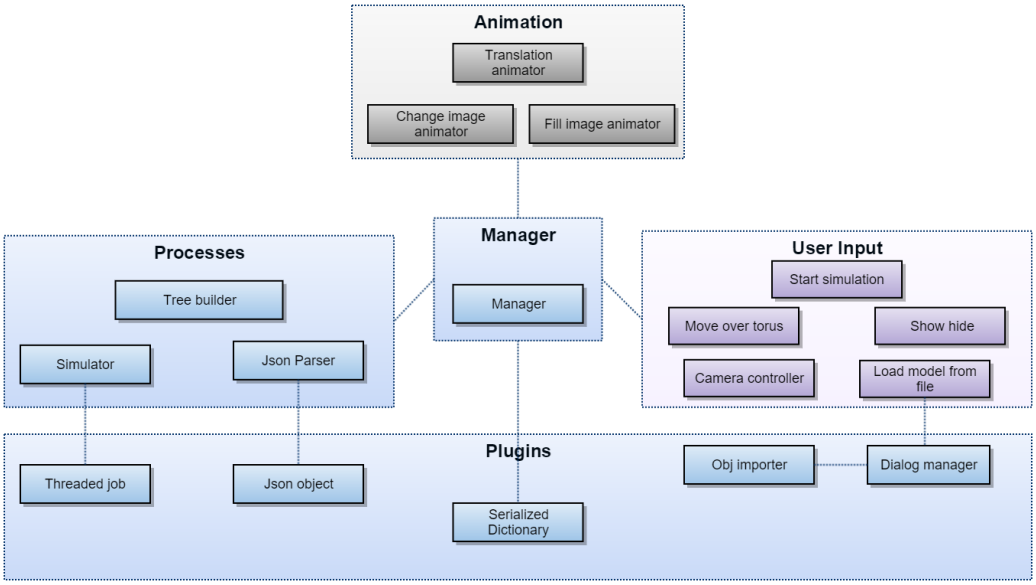
5.9. Bid.

5.10. A continuación

6. Visualización

6.1. En este capítulo

Dada la naturaleza tridimensional del problema, se optó por usar un programa que permitiera visualizar los datos en 3D e interaccionar con ellos. Unity, como editor de juegos, permite esta interacción y tiene la gran ventaja de poder crear un ejecutable para las plataformas más reconocidas como Windows, Mac, Linux, pero también web o android. Dicho ejecutable será el producto final del proyecto, ya que incluirá el simulador, mientras que este a su vez incluye el resolvedor RadPro. Veremos primero la filosofía de diseño de Unity que cambia bastante en referente a paradigmas de POO comunes como Java o C++. Explicaremos el esquema básico de las componentes que forman el visualizador, y nos adentraremos en los detalles más interesantes que esconden. Por último veremos como crear el ejecutable para poder distribuir nustro programa fácilmente.



6.2. Filosofía de diseño de Unity

La programación en Unity puede efectuarse en los lenguajes C++, JavaScript y Boo, aunque este último no se utiliza habitualmente. Todos ellos son lenguajes dentro del paradigma Programación Orientada a Objetos, pero en Unity cambia la manera en el que se organizan los datos. El cambio más importante es que lo que se programa son comportamientos, no entes. Los objetos centrales de Unity son los prefabs: Packs de componentes que definen como se comporta ese objeto dentro de la escena. Por ejemplo, todos los objetos tienen una componente Transform que indica la posición, rotación y escala del objeto. Unity ofrece varias componentes comunes predefinidas listas para usar en tus prefabs tal como colliders para detectar

colisiones entre objetos o mesh renderers para definir la geometría (los vértices) que forman el objeto. Por ello la implementación de funcionalidades se centra en encontrar acciones/comportamientos de los objetos de la escena y definir cdad uno de ellos en un fichero aparte. Por ejemplo: De manera que un mismo objeto que camine y hable tendrá, en vez de un fichero persona en el que se especifican ambos comportamientos como métodos, dos comportamientos definidos en ficheros distintos Caminar y Hablar. La idea detrás de esto es que si a posteriori se crea un objeto que solo camine, baste con añadirle la componente de caminar y ajustar sus parámetros desde dentro del editor. Las componentes son clases especiales que heredan de la clase MonoBehaviour de la librería de Unity. Esto permite que los comportamientos sean añadidos a modelos creando lo que se conoce como gameObjects o prefabs, packs de modelo + comportamientos. Comportamientos comunes como colisiones o animaciones ya vienen implementadas en Unity y basta con añadirle la componente correspondiente a

6.3. Diseñando los paneles

Una vez ejecutada la simulación, contamos con un json que describe toda la información relevante que ha ocurrido en cada frame. La información sobre el momento del día lo mostraremos en la GUI principal de manera que sea apreciable desde cualquier punto de la escena. Aprovechando la naturaleza 3D de nuestro visualizador intentaremos mostrar to da la información posible en 3D. La información referente a una casa la mostraremos en un canvas (panel 2D) pero situado dentro de la escena 3D sobre la casa en cuestión. Por último visualizaremos los cables como líneas que van desde la casa origen hasta la casa de destino, su tamaño representará la capacidad de ese cable, cisualizaremos el flujo de electricidad moviendo unas partículas en la dirección del flujo sobre el cable y el flujo en si en un panel 3D sobre las partículas.

Diseño del panel de una casa:¡imagen¿

- En la barra izquierda se muestran los generadores que contiene la casa. Un icono representa el tipo de generador que se trata, bien solar o eólica y sobre la imagen se sobrepone un relleno que representa la eficiencia que está teniendo ese generador en ese momento en referencia al máximo rendimiento que podría tener.

- En el panel central se muestra la gráfica de la apuesta realizada por esta casa.

- En el panel derecho se añaden los dispositivos. De manera similar a los generadores, están representados por un icono referente al tipo del dispositivo y sobre ellos se sitúa un relleno que muestra su progreso.

- En el panel inferior se encuentra la batería, que también contiene una imagen que se rellena indicando el porcentaje de batería actual respecto al total.

Tanto los generadores como los dispositivos son variables y se deben añadir en tiempo de ejecución, una vez se sepas cuantos y de que tipo son. Por ello, crearemos un prefab para cada uno de ellos.

Los cables los modelamos usando la componente LineRenderer de unity, que

dados dos puntos genera una línea personalizable entre que las une. Sobre ella moveremos un objeto con la componente TrailRenderer, que deja una estela detrás del objeto mientras esta se mueva. Por último, sobre esta partícula situaremos un texto que indica la energía que se transfiere en ese instante.

6.4. Control de la cámara

En cualquier momento del programa, el usuario puede controlar la posición y rotación de la cámara. Los controles imitan el funcionamiento que ofrece el editor de unity por defecto: Con las teclas WASD se controla la posición de la cámara mientras que el movimiento del ratón mientras se tenga el click derecho presionado controla la rotación.

MonoBehaviour contiene un método Update() que se llama a cada frame de la simulación. Cabe notar que entre llamadas al método update pasa un tiempo variable. Por ello hay que ir con cuidado a la hora de hacer animaciones etc. hay que considerar la variable Time.deltaTime de unity en dondde se guarda el valor de tiempo que ha transcurrido entre el frame actual y el anterior. Usando los métodos estáticos de la clase Input de UnityEngine OnMouseDown y GetKeyDown() se detecta si alguna de las teclas del control de cámara ha sido pulsada. A partir de aquí, se implementa la funcionalidad deseada (Translate para las teclas de control y Rotate para la mouse).

6.5. Eligiendo la ciudad

Añadir un modelo a una escena previamente a su ejecución es relativamente sencillo, basta con crear un prefab que tenga las componentes que te interesan. Sin embargo, importar un modelo en tiempo de ejecución en Unity es más complicado de lo que puede parecer. El primer problema es abrir un cuadro de diálogo que permita al usuario elegir un archivo. Unity usa Mono que es un subconjunto de .NET FrameWork 2.0 especializado para juegos, que por defecto no incorpora la opción de crear ventanas propias del SO. La solución ha sido añadir el dll Windows.Forms.dll de .NET FrameWork 2.0 en una carpeta llamada Plugins dentro del proyecto de unity de manera que el compilador de unity comprenda que queremos usar tal extensión y permita crear las ventanas. El soporte de .NET FrameWrk 2.0 no está garantizado por Unity por lo que pueden saltar errores de seguridad al intentar abrir la ventana, pero ignorando estos defectos podemos permitir que el usuario abra un fichero. Quiero agradecer a ¡Referencia¿por la solución prestada.

6.6. Seleccionando el perfil de las casas

Una vez la ciudad se encuentre en la escena el usuario podrá ver y modificar el perfil de la gente que vive en esa casa en tres niveles: Alto consumo, consumo regular o alto ahorro.

Cuando el usuario pasa el ratón por encima de una casa, está casa se coloreará de un color dependiendo del perfil que tenga asignado. Por defecto, todas las casas tienen asignado el perfil de consumo regular que viene representado por el color naranja. Una vez situado el ratón sobre la casa el usuario podrá elegir el perfil deseado para esa casa simplemente pulsando las teclas 1, 2 o 3 siendo 1 el de menor consumo y 3 el mayor.

Por último si se pulsa la tecla P, las casas se colorearán con el color correspondiente a su perfil hasta que se vuelva a pulsar P. Esto permite visualizar la distribución de perfiles a lo largo de una ciudad.

6.7. Seleccionando la franja horaria

El segundo paso es seleccionar de que hora a que hora se ejecutará la simulación. Para ello contamos con un torus fino con dos bolas sobre ella. Estas bolas se pueden seleccionar y arrastrar sobre el torus y muestran un lienzo al moverse que indica la hora que señalan. La idea bajo esta implementación es: - Detectar si el mouse está en modo drag con el método OnMouseDrag de Monobehaviour

- Coger el punto que representa el ratón en la ventana

- Hacer un cambio de base para que deje el pixel 0,0 esté situado en el centro de la screen

- normalizar el vector que va desde el centro hasta el punto del ratón.

- escalarlo por el radio mayor del torus para situarlo encima del mismo

Por defecto estos valores empiezan de 06:00h a 18:00h. Cabe considerar que para que el funcionamiento del programa sea el esperado, la hora inicial debería ser inferior a la hora final

6.8. Ejecutando la animación

Como tercer paso, el usuario hace click en la estrella de acuerdo con que está contento con los parámetros elegidos hasta ahora. Esto crea un nuevo thread que se encarga de correr la simulación. La creación del thread se consigue gracias a la clase ThreadedJob que es aportación de ¡referencia¿En particular este thread tiene un método que se ejecutará al ser creado. Además, cada vez que el thread principal llame al metodo update de ThreadedJob, este actualizará su estado y en caso de que haaya acabado generará un evento que será tratado por el thread principal. Por último el thread se destruirá porque ya ha completado su trabajo. El thread al mismo tiempo crea un Process para abrir una ventana de terminal y ejecuta el simulador con los parámetros proveidos: java -jar simulator.jar. Cabe decir que si se utilizan rutas absolutas para decir a la terminal donde se encuentra el jar, nuestra aplicación no será distribuible, porque no funcionará en otros ordenadores, por ello debemos poder acceder al simulador usando rutas relativas. Unity provee con la variable Application.dataPath que referencia en tiempo de ejecución a la carpeta de datos que está utilizando Unity en ese momento. Mientras estamos en el editor de

Unity, esta carpeta es la carpeta Assets del proyecto mientras que una vez se haya hecho la build, es la carpeta que acompaña al ejcutable ,exe. Por eso situaremos el jar dentro de esta carpeta y la aplicación será distribuible.

6.9. Mostrando la información relevante

Durante toda la ejecución del visualizador, se minimiza la nformación mostrada de manera que sea más intuitivo y disfrutable para el usuario final. En particular diversos objetos comparten el comportamiento de revelar/ocultar cierta información al ser pulsados. Por ejemplo, al pulsar sobre las casas, veremos el informe de qué está ocurriendo en esa casa mientras se ejecuta la simulación, mientras que si pulsamos sobre los focos de energía que pasan sobre los cables, veremos o ocultaremos un texto con cuanta energía transportan en cada momento.

Por otro lado, mediante las teclas H y G podemos controlar qué partes de la ciudad se ven en cada momento. Con H (de house) podemos revelar/ocultar las casas mientras que con G (de grid) podemos revelar/ocultar los cables. Por último, con la tecla Esc podremos salir de la aplicación. El objeto manager es el que contendrá estos comportamientos.

6.10. Programando las animaciones

La simulación nos da frames discretos en los que sabemos como es el estado del sistema. Nuestra intención será interpolar linealmente estos valores para que parezca que el movimiento es continuo.

Unity cuenta con 2 modos por defecto que permiten hacer animaciones: -La componente Animation permite ejecutar animation clips guardados en una lista, es la manera original de hacer las animaciones. -La componente Animator también conocida como Mecanim, que genera una máuina de estados que tiene en cada estado el clip de la animación y contiene transiciones entre estados que se encargan de controlar como cambian las animaciones.

Sin embargo, ninguna de ellas permite cambiar los clips de animación en tiempo de ejecución. Los clips han de estar guardados en ficheros .anim y ser referenciados en estas componentes, no se permite añadir clips en tiempo de ejecución. Es una limitación del sistema actual que han anunciado que añadirán en una próxima versión. Por ello, la solución es crearnos nosotros un algoritmo sencillo y simple que se encargue de simular animaciones básicas.

Hay dos modos de animación: Repetitivo y continuo. En el repetitivo, el mismo frame se anima una y otra vez para analizar correcamente que ha pasado concretamente entre esos instantes de tiempo. En el continuo, la animación va recorriendo todos los frames y se repite una vez haya llegado al final. Todos los animadores tienen como entrada un array de datos. En el caso de la batería estos datos representan el porcentaje de llenitud, mientras que en el caso de los dispositivos representan su progreso. La idea es llamar a un método un número concreto de veces cada segundo

donde se recalcule el valor actual y se actualize la información necesaria correspondiente. Al finalizar el segundo, se llama a otro método en el que o bien se pasa al siguiente estado o bien se reinicia el estado actual dependiendo de si la animación está en modo repetitivo o no. Una vez que hayamos calculado el valor actual basta con actualizar Image.fillAmount con el valor correcto.

6.11. Controlando el flow

Incluso en un entorno dirigido por componentes como unity, necesitamos un objeto que controle el estado del programa y decida que opciones están disponibles al usuario en cada momento. En nuestro proyecto, la componente Manager se encarga de esta función. Además sirve como puente sobre las diferentes componentes. Si un objeto quiere comunicar algo a otro pero no tiene guardada una referencia a él, puede pasar la información al manager y este será el que haga llegar la información al destinatario. Funciona similarmente al método main de los programas Java.

6.12. Parseando el JSON

Así como crear el json era muy sencillo, parsearlo lleva más tiempo. Primero debemos encontrar una librería adecuada que sirva para Unity. Probablemente la herramienta más popular utilizada para controlar archivos JSON en C# sea Json.NET creado por James Newton-King. De todas formas, dadas las limitaciones de compatibilidad de Unity con .NET, se necesita adaptar el código a Unity. Existe esta adptación a la venta en la Asset Store de Unity, pero para mantener este proyecto gratuito, se ha evitado usarlo.

De todos modos existen parseadores de JSON creados por la comunidad especialmente para Unity. Nosotro usaremos JSONObject de ¡referencia¿.

El primer paso es detectar con cuantos generadores y dispositivos cuenta cada casa e instanciar un prefab por cada uno en el panel de la casa correspondiente. Una vez todos los objetos de la escena existan popularemos sus componentes de animación con los datos guardados en el JSON. Para cada tipo de dato deberemos identificar cual es su animador, o bine FillImageAnimator en el caso de la batería, los generadores o los dispositivos, o bien ChangeImageAnimator en el caso de las apuestas, o bien TranslationAnimator en el caso

6.13. A continuación

Mostraremos las conclusiones obtenidas y posibles rutas de avance para este proyecto.

7. Futuro trabajo

7.1. Futuro trabajo

Referencias

[1] Batut, C.; Belabas, K.; Bernardi, D.; Cohen, H.; Olivier, M.: User's guide to *PARI-GP*,
`pari.math.u-bordeaux.fr/pub/pari/manuals/2.3.3/users.pdf`, 2000.